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

Settembre 2023

Progetto e implementazione di un sistema embedded per il riconoscimento di attività sportiva

Design and development of an embedded system for physical activity

Candidato: Lorenzo Giampietro

Relatore: Prof. Riccardo Berta

Correlatori: Dr. Matteo Fresta, Dr. Ali Dabbous

SOMMARIO

Il presente elaborato è stato redatto con l'obiettivo di misurare e riconoscere i colpi svolti da un atleta nel campo delle arti marziali, più precisamente nella boxe. In questo caso specifico è richiesto il riconoscimento di tre tipi di colpi (cross, left hook e right hook). Attraverso i sensori applicati sulla scheda Arduino Nano 33 BLE Sense Rev2 [1] sono stati misurati valori dell'accelerometro e del giroscopio durante i movimenti; successivamente i dati sono stati trasmessi tramite la connessione BLE (Bluetooth Low Energy) presente sulla scheda ad un'applicazione Flutter installata su un dispositivo mobile per poi essere inviati a un API (application programming interface) Cloud chiamata Measurify.

Una volta terminata la raccolta del dataset è stato utilizzato un algoritmo di machine learning per generare un modello di rete neurale in grado di classificare in tempo reale quale delle tre azioni scelte è stata effettuata dall'atleta.

1 INTRODUZIONE

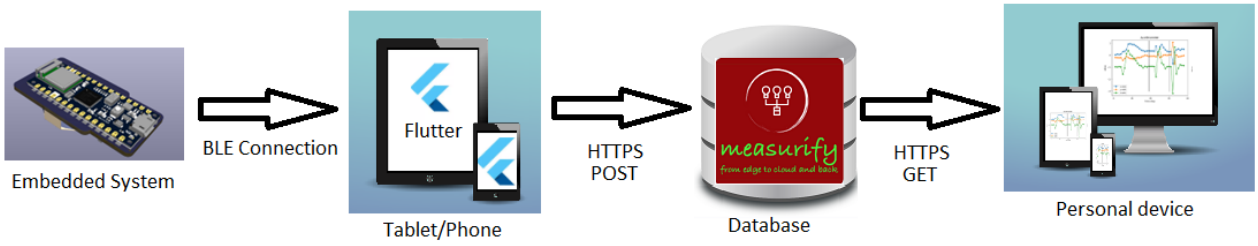
Negli ultimi anni l'"Internet of Things" ha guadagnato sempre più rilevanza sia dal punto di vista scientifico che commerciale fino a diventare parte integrante delle nostre vite, ad esempio, nel campo della domotica (smart home), nei sistemi di intrattenimento nelle automobili e attraverso lo sviluppo di dispositivi utili a monitorare la salute degli utenti (e-health). Un utilizzo molto comune di questa tecnologia consiste nel monitorare l'attività sportiva dell'utente tramite dispositivi indossabili con sensori utili a captare movimenti.

Uno strumento molto importante presente in questa tesi è il machine learning. Recentemente l'intelligenza artificiale è un argomento molto discusso sia pubblicamente sia all'interno della comunità scientifica; la capacità di generare modelli automatici per rilevare movimenti è molto utile e già presente, ad esempio, negli smartwatch attualmente in commercio (nel riconoscimento dell'attività fisica svolta dall'utente come la corsa, il nuoto o la bicicletta) e in questa tesi questa è utilizzata nel riconoscimento di quale dei movimenti conosciuti è stato effettuato dall'utente.

Più nello specifico questo progetto applica i concetti di cui abbiamo parlato al mondo della boxe, creando un sistema in grado di riconoscere quale movimento è stato svolto dal pugile e potenzialmente captarne le caratteristiche come la velocità, la potenza e la precisione della tecnica. Sarebbe inoltre possibile visualizzare i dati in modo semplificato così da permettere all'utente di monitorare facilmente e in modo molto efficace il proprio allenamento e la propria performance nel tempo e, per esempio, capire in quali parti del proprio sport debba concentrarsi di più per diventare un atleta completo.

Nella nostra architettura vengono rilevati i valori dell'accelerometro e giroscopio dai sensori presenti sul dispositivo e questi vengono inviati tramite Bluetooth Low Energy al dispositivo mobile che si occuperà, tramite una interfaccia utente dedicata, oltre a interfacciarsi con l'Arduino e memorizzare i dati sulla memoria del dispositivo anche ad avviare e fermare le misurazioni. Successivamente il dispositivo mobile tramite connessione a internet invierà i valori tramite HTTPS POST al database in cui verranno salvati. È possibile richiedere questi dati tramite HTTPS GET e visualizzarli su qualsiasi dispositivo in qualsiasi momento.

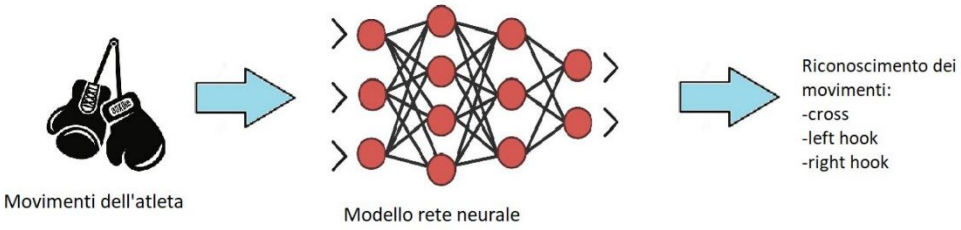
Il progetto si basa quindi su questi quattro fattori:



Schema raccolta del dataset

Inizialmente la scheda viene posizionata sull'atleta all'altezza della cintura e, durante lo svolgimento del movimento, i dati vengono inviati tramite BLE al dispositivo smartphone, il quale a sua volta li invierà al framework IoT Measurify [2] tramite l'interfaccia API RESTful. Questo framework si occuperà di salvare i dati sul database non relazionale MongoDB. Successivamente i colpi potranno essere classificati tramite l'utilizzo dell'apposito modello.

Successivamente i dati sono recuperati dal database in cui sono stati salvati e usati per generare il modello in grado di classificare i movimenti in tempo reale.



Schema del funzionamento del modello

La tesi è divisa in tre paragrafi principali:

- Metodi e strumenti utilizzati, dove si discute come è stato realizzato il progetto e quali strumenti sono stati necessari
- Sperimentazioni e risultati, con la valutazione dei risultati ottenuti fino a verificare l'accuratezza dei modelli generati.
- Contributo personale e considerazioni conclusive con una esposizione delle conclusioni finali.

2 METODI E STRUMENTI UTILIZZATI

2.1 HARDWARE

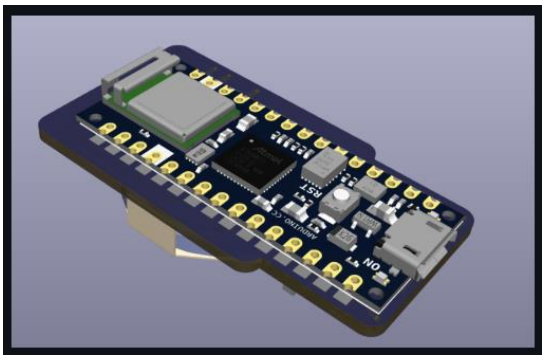
SISTEMA EMBEDDED

Il sistema è formato da un microcontrollore che acquisisce i dati dai sensori montati sulla scheda e questi dati vengono poi raggruppati in tre caratteristiche:

- IMU (unità di misura inerziale)
- Environment
- Orientation

Attualmente la parte di invio dei dati riguarda solo i dati della caratteristica IMU ma il sistema rende disponibile anche l'invio e la ricezione delle altre caratteristiche se necessarie ad altre applicazioni.

La parte di edge del progetto è composta da una scheda Arduino nano 33 BLE sense con una batteria di supporto.

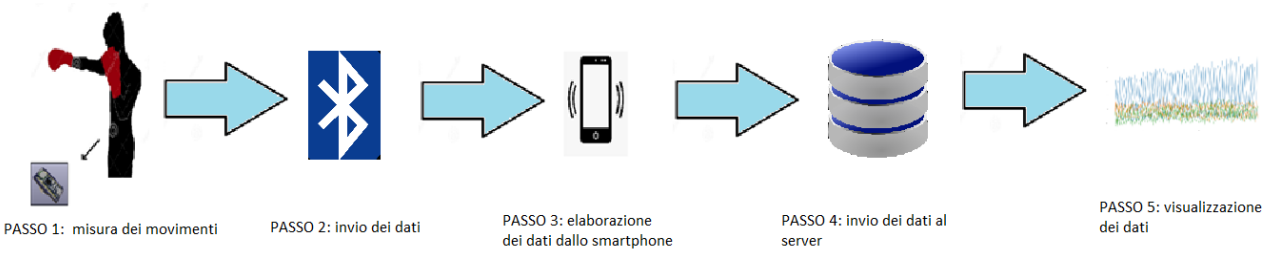


Arduino nano 33 BLE Sense Rev2

Il microcontrollore riceve i valori misurati dai sensori e poi li espone tramite la connettività BLE.

Le caratteristiche fornite sono quelle di IMU (accelerometro, giroscopio, e magnetometro), di Ambiente (prossimità, temperatura, umidità, pressione e luce) e di orientamento (rollio, beccheggio, ed heading) e il tempo di campionamento viene deciso dal client che si connette (250ms è il valore di default). Nel nostro caso viene utilizzata la caratteristica di IMU e un tempo di campionamento di 100ms.

Una limitazione della connessione Bluetooth Low Energy è la possibilità di inviare un massimo di 20 bytes per ogni messaggio. Questo ha creato un problema per l'invio di dati di IMU ed Environment perché hanno dimensioni che superano i 20 bytes (IMU 9 float ed Environment 3 float e 5 int per un totale di 32 bytes). Per ovviare a questo problema questi valori sono stati convertiti da 32 bit a 16 bit signed integer (int16t) tramite delle opportune moltiplicazioni, senza che però avvenga alcuna perdita di informazione in quanto l'accuratezza dei sensori dell'Arduino è a 16 bit. Quando questi dati vengono ricevuti, nell'applicazione flutter vengono riconvertiti al valore originale prima di essere inviati a Measurify.



Schema complessivo del progetto

Inertial Measurement Unit (IMU)

La caratteristica IMU [3] è la più importante e la più utilizzata in questa tesi. I dati che vengono misurati dal sensore sono quelli di accelerazione, velocità angolare e campo magnetico che agiscono sull'Arduino tramite i sensori accelerometro, giroscopio e magnetometro. Ognuno dei sensori elencati ha 3 gradi di libertà (vengono quindi misurati i valori per ognuno dei 3 assi dello spazio) arrivando quindi ad avere 9 gradi di libertà in totale.

Dato che il limite di invio di dati imposto dalla connessione è di 20 bytes e i 9 float di IMU superano questo limite, sarà necessario svolgere una conversione del formato di quest'ultimi in interi prima dell'invio tramite bluetooth.

La conversione viene svolta nel codice Arduino e avviene tramite una moltiplicazione di ogni dato per un fattore moltiplicativo precedentemente dichiarato:

- Accelerazione [G]*8192
- Velocità Angolare [dps]*16.384
- Campo Magnetico [uT]*81.92

```
void manageIMU() {
  int16_t imu[9] = { (int16_t)round(acceleration[0]*ACC_MULTIPLIER), (int16_t)round(acceleration[1]*ACC_MULTIPLIER), (int16_t)round(acceleration[2]*ACC_MULTIPLIER),
                     (int16_t)round(angular_speed[0]*GYR_MULTIPLIER), (int16_t)round(angular_speed[1]*GYR_MULTIPLIER), (int16_t)round(angular_speed[2]*GYR_MULTIPLIER),
                     (int16_t)round(magnetic_field[0]*MAG_MULTIPLIER), (int16_t)round(magnetic_field[1]*MAG_MULTIPLIER), (int16_t)round(magnetic_field[2]*MAG_MULTIPLIER)
                   };
  imuCharacteristic.writeValue(imu, sizeof(imu));
}
```

Funzione nel codice Arduino [4] che si occupa della conversione in interi

Nel codice dell'applicazione questi dati vengono poi riconvertiti da int a float, dividendo per gli stessi valori.

```dart
//convert the IMU data 9 int16 array back to float and create the json object ready to be sended
Map<String, dynamic> parseIMUData(Uint8List value, int ArrayLength) {
  final byteData = ByteData.view(value.buffer);
  final imuData = List<int>.filled(ArrayLength, 0);
  List<double> floatValues = List<double>.filled(9, 0.0);
  for (var i = 0; i < imuData.length; i++) {
    if (i < value.lengthInBytes ~/ 2) {
      int intValue = byteData.getInt16(i * 2, Endian.little);
      imuData[i] = intValue;
      print("intValue:" + intValue.toString());
    }
  }
  for (int i = 0; i < imuData.length; i++) {
    //conversion IMU from int to float
    if (i < 3) {
      floatValues[i] = imuData[i] / 8192;
    } else if (i < 6) {
      floatValues[i] = imuData[i] / 16.384;
    } else {
      floatValues[i] = imuData[i] / 81.92;
    }
  }
  Map<String, dynamic> jsonObj = {
    "timestamp": DateTime.now().millisecondsSinceEpoch,
    "values": floatValues,
  };
  //To visualize jsonObj
  //String jsonDataString = jsonEncode(jsonObj);
  //print(jsonDataString);
  //print(floatValues);
  return jsonObj;
}
```

Funzione nell'applicazione Flutter che riconverte i dati in float

In questo modo l'invio e la ricezione dei dati avviene senza problemi e senza perdita di informazione

ENVIRONMENT

La caratteristica di Environment viene gestita in un modo molto simile a quello usato per i dati provenienti dal sensore IMU. Il formato originale dei dati è 3 float e 5 int di dimensione superiore a 20 bytes; quindi, nel codice, i dati per essere inviati vengono moltiplicati per il loro fattore moltiplicativo (nel nostro caso 100). Questo passaggio viene svolto ovviamente solo per i 3 float (gli altri dati essendo già int non hanno bisogno della conversione per essere inviati) e questi float rappresentano i dati provenienti dai sensori di:

- Temperatura[C]*100
- Umidità [percentuale]*100
- Pressione [kPA]*100

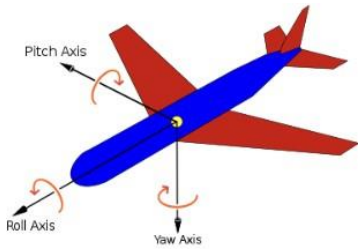
I dati che invece non sono stati convertiti sono:

- Prossimità
- Luce di ambiente
- Colore (RGB, quindi salvato in 3 int)

I dati verranno poi riconvertiti in float.

ORIENTATION

La caratteristica di Orientation misura la posizione nello spazio della scheda Arduino nello stesso modo che viene utilizzato anche per definire la posizione nello spazio degli aerei in volo come rappresentato in figura



Esempio di Orientation

- Heading[degree]
- Pitch[dpdegrees]
- Roll[degree]

In questo caso non viene svolta nessuna conversione perché i dati sono salvati e inviati in un formato di 3 float rispettando quindi il vincolo di 20 bytes come dimensione massima inviabile.

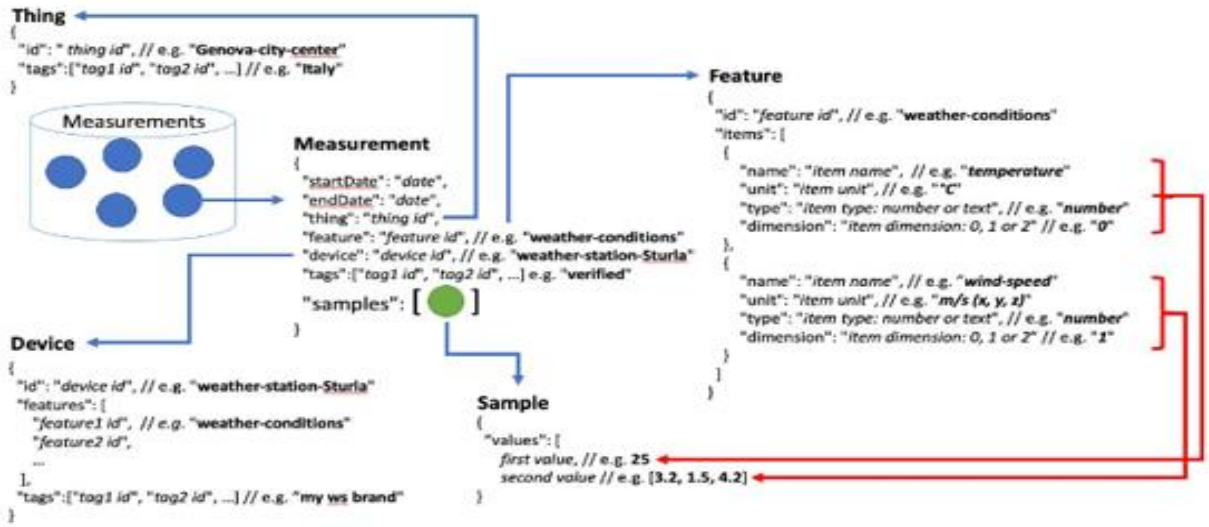
2.2 MEASURIFY

Per il funzionamento del progetto è essenziale trovare un modo per memorizzare i dati in modo che possano essere successivamente recuperati e usati per l'addestramento del modello di machine learning. Per fare questo sarà essenziale l'utilizzo del cloud e per farlo ci affidiamo al framework Measurify, sviluppato dall'Elios Lab dell'università di Genova per l'ambito IoT.

Measurify è un API cloud di tipo RESTful utilizzato per la raccolta di dati provenienti da sensori in ambito IoT ed è quindi essenziale per la gestione delle "smart things" e dei dati da loro prodotte.

Il sistema IoT viene modellato come risorsa formata dai seguenti elementi

- il soggetto della misurazione, può essere una Persona, un Ambiente o un'entità come un Viaggio (Ad esempio la stanza in cui eseguiamo le misurazioni).
- Device: è il dispositivo hardware o software che si occupa della misurazione (Ad esempio un termometro).
- Feature: è la definizione del tipo dei dati raccolti dal Device e rappresenta lo schema che deve essere rispettato da ogni misura appartenente a quella Feature. La lista degli elementi che compone la Feature è chiamata items (Temperatura ad esempio).
- Measurement: la misura effettuata dal Device riferita a una determinata Thing di un determinato tipo (Feature)
- Tag: è una etichetta che può essere aggiunta a una misurazione, utile per distinguere un gruppo di misurazioni da un altro.



Schema del funzionamento di Measurify

Nel nostro caso l'atleta è la THING, il dispositivo Arduino è il DEVICE. La FEATURE sarà misurata dal sensore IMU e quindi per ogni MEASUREMENT ci sarà una FEATURE contenente i valori di: accelerazione, velocità angolare e campo magnetico.

Quando effettuiamo un'operazione che richiede di salvare dei dati sul cloud utilizziamo delle RESTful API e le solite richieste HTTPS (GET, POST, PUT, DELETE) con il body presente al loro interno in formato JSON (javascript object notation), un formato utilizzato dalle applicazioni per scambiarsi dati, ma comunque in un formato facilmente comprensibile per un essere umano.

Per comunicare con il cloud inizialmente va svolta l'autenticazione tramite TOKEN. Il TOKEN viene acquisito tramite una chiamata POST con al suo interno (sempre in formato JSON) username e password dell'utente. Il TOKEN così acquisito avrà una validità temporanea in modo da evitare problemi di sicurezza eccessivi in caso di breccia nei dati.

POST https://students.measurify.org/v1/login

{

"username" : "utente-username",

"password" : "utente-password",

"tenant" : "utente-tenant"

}

Questo era un esempio di una POST di autenticazione, e in risposta otteniamo un messaggio JSON contenente il TOKEN che andrà poi messo nell'header della prossima richiesta (ad esempio il voler salvare una misura sul cloud). Per comodità è stata aggiunta una seconda modalità di login con un TOKEN senza limiti di scadenza ma con limitate funzioni possibili.

Un esempio di body contenente la richiesta di salvare sul cloud una misura

POST https://students.measurify.org/v1/login

```
{
"thing" : "player1",
"feature" : "IMU",
"device" : "edge-meter",
"samples" : [ {"values" : 80,10}]
}
```

Nei samples vanno messi i valori della misura e sono in generale più di uno e la misura è la collezione di molti valori in diversi istanti di tempo e ogni sample può avere un campo "delta" che ne specifica l'istante di tempo del campione a partire dalla startDate della misura. Di seguito un esempio di risposta:


```
{
   "visibility": "public",
   "tags": [“Martial_Arts”, “Right_Hook”],
   "_id": "lorenzo_right_hook_test_updated",
   "thing": "player1",
   "feature": "IMU",
   "device": "edge-meter",
   "samples": [
      {
         "values": [
            25.2
         ]
      }
   ],
   "owner": "5eecbb119770994272457c75",
```

```
    "startDate": "2020-06-19T13:25:43.647Z",
    "endDate": "2020-06-19T13:25:43.647Z",
    "timestamp": "2020-06-19T13:25:43.647Z",
    "lastmod": "2020-06-19T13:25:43.647Z"
}
```


Se invece si vuole interrogare le API per avere le misure memorizzate con certi parametri di ricerca:

```
GET {{url}}/v1/measurements?filter={"tags":”Cross"}
```

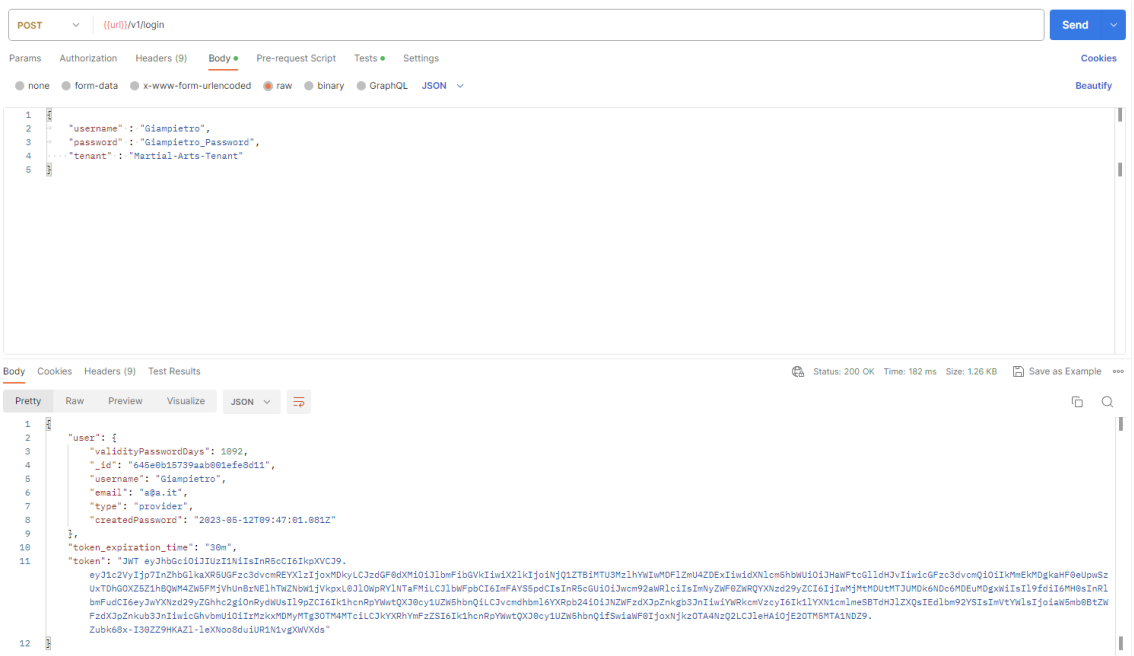
In questo modo otteniamo come risposta le misure che hanno come tag “Cross”

2.3 POSTMAN

Postman[5] è una piattaforma API per sviluppatori e consente di inviare richieste ad una API in modo veloce ed efficiente. L'applicazione supporta richieste HTTP alle quali è anche possibile aggiungere degli header. L’interfaccia intuitiva e leggera permette di agevolare l’invio di chiamate POST, GET, PUT e DELETE.

Nella seguente immagine vediamo l'interfaccia del programma al momento della POST e possiamo vedere come la risposta che riceviamo sia molto più dettagliata di quella di un semplice browser.

Un'altra feature che Postman ci fornisce è la possibilità di scegliere il formato in cui stiamo visualizzando il body della risposta (XLM, HTML, JSON, ecc.…).



Risposta ad una login su Measurify

Nella POST è necessaria la presenza di un body: nel nostro caso la POST di richiesta di effettuare login avrà al suo interno le informazioni di username, password e tenant in formato JSON. Il tenant è come un contenitore nel database dedicato al nostro progetto e quindi va specificato.

Una volta premuto SEND di questa richiesta riceviamo come risposta con al suo interno un TOKEN assegnato a noi valido per un tempo limitato e sarà richiesto per poter salvare o richiedere dati dal sensore con le chiamate POST e GET.

L'applicazione software si occuperà di fare una POST dei dati ricevuti dall'Arduino tramite bluetooth e di mandarli al cloud. Questo strumento Postman è un modo che abbiamo per verificare che la misura sia stata inviata correttamente, più precisamente effettuando una richiesta di tipo GET nella rotta measurement.

2.4 APPLICAZIONE SOFTWARE FLUTTER

In questa sezione prendiamo in considerazione l'applicazione software Flutter [6] che è stata sviluppata per rendere questo progetto possibile.

L'applicazione è in grado di raccogliere i dati che vengono inviati dal microcontrollore, apportare loro delle modifiche se e dove necessario e inviarli al cloud tramite POST https utilizzando API (Application Programming Interface) rese disponibili da Measurify. Una volta raccolti i dati sono stati utilizzati per il processo di training di un algoritmo di machine learning (che non viene trattato in questa tesi) il quale, una volta completato, ha permesso al sistema di riconoscere il movimento svolto dall'atleta.

Ricordiamo brevemente quali sono le necessità a cui questa applicazione deve ovviare: abbiamo bisogno di un software che renda facile per l'utente connettersi alla scheda Arduino (sempre tramite Bluetooth Low Energy), ricevere i dati che quest'ultima espone, ed inviarli sul cloud tramite le API di cui abbiamo parlato prima (utilizzando quindi i concetti chiave di HTTPS con le chiamate POST, PUT, GET, DELETE). La comunicazione con le API potrebbe essere svolta tramite Postman, come è stato spiegato nella sezione precedente, ma sarebbe troppo complicato per l'utente e quindi le chiamate HTTPS verranno gestite nel codice dell'applicazione.

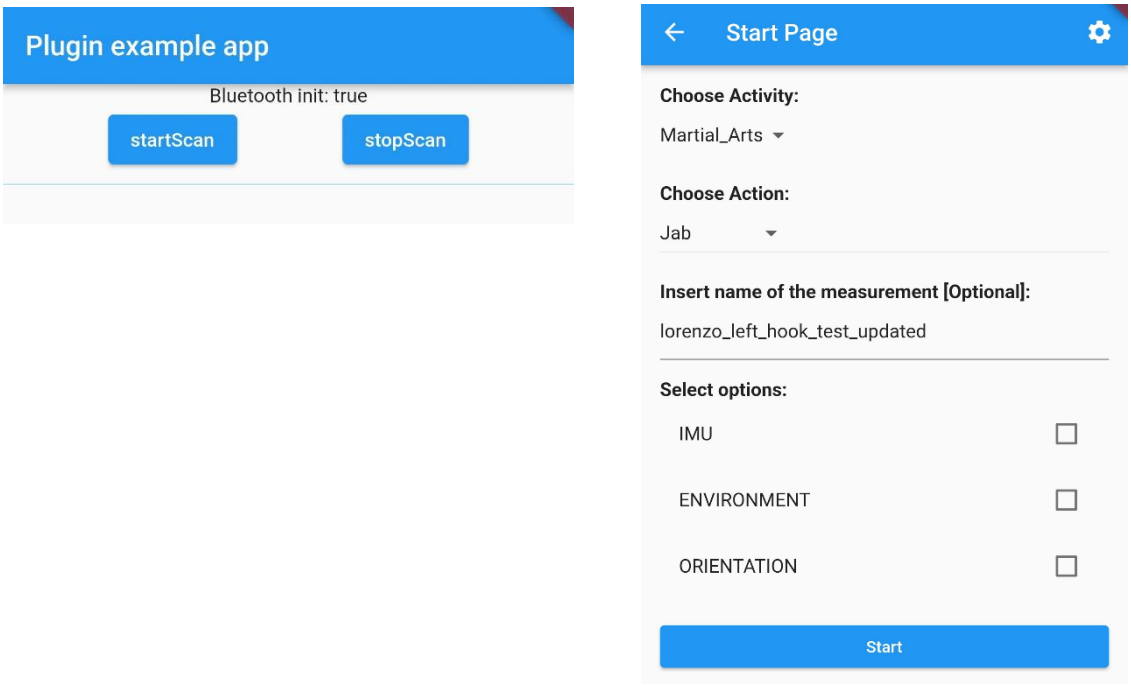
Il framework utilizzato per lo sviluppo è Flutter, più precisamente il suo plugin quickblue di cui parleremo più avanti.

Flutter è un framework open source di Google utilizzato per sviluppare applicazioni multipiattaforma da una sola codebase. Il linguaggio utilizzato è Dart, sviluppato da Google per sostituire Javascript come linguaggio per lo sviluppo di pagine web. Inizialmente il framework era limitato alla creazione di interfacce native iOS, Android, Linux, Windows e MacOS ma successivamente è stato introdotto anche il supporto allo sviluppo di applicazioni web per siti statici e i suoi componenti sono i seguenti:

- Dart Platform: il compilatore del linguaggio Dart utilizza due diverse piattaforme per lo sviluppo delle due diverse interfacce. Una piattaforma Dart Native usata per lo sviluppo su dispositivi (come smartphone, desktop, server, ecc....) e comprende la Dart VM con una compilazione JIT e una AOT. L'altra piattaforma invece è la piattaforma Dart Web per lo sviluppo di interfacce web con una compilazione per lo sviluppo e una per la produzione
- Flutter Engine: scritto principalmente in C++ e si interfaccia con SDK della piattaforma specifica utilizzata
- Foundation Library: classi e funzioni base utilizzate nelle applicazioni
- Widget: una descrizione immutabile dell'interfaccia utente
- L'applicazione utilizza un plugin di Flutter di nome quickblue in grado di:
- Ricercare dispositivi BLE nei dintorni
- Connettersi con i dispositivi rilevati
- Scoprire i servizi che vengono offerti dai dispositivi
- Trasferire i dati fra dispositivo e server

- Smart Collector è il nome che è stato dato dall'applicazione vera e propria. Il Bluetooth viene gestito attraverso diverse funzioni:
- handleValueChange: usata per gestire i cambiamenti dei valori
- parseData: un insieme di funzioni (parseIMUData, parseEnvironmentData, parseOrientationData) che convertono i dati ricevuti da Arduino in float (quando necessario) e ritornano un oggetto JSON contenente i dati pronti ad essere inviati
- toggleCollecting: per iniziare o fermare la raccolta di dati
- sendData: manda i dati al server

La user interface è composta da una serie di pagine, fra cui l'utente deve navigare per inviare i dati al cloud. La pagina iniziale è quella che si occupa di scannerizzare i dintorni per rilevare dispositivi disponibili e di connettersi ad essi tramite dei pulsanti. Nel momento in cui siamo connessi ad Arduino veniamo portati alla "uploading page" con le seguenti sezioni:



User Interface

- Choose Activity: è la sezione dove possiamo scegliere lo sport di cui stiamo facendo misure. In questo caso "Martial Arts"
- Choose Action: in questa sezione selezioniamo l'azione di cui stiamo facendo la misura (per esempio Jab, Cross, ecc...). Questo diventerà il TAG della misura
- Insert name of the measurement: sezione dove viene inserito il nome della misura, essenziale per essere salvata correttamente nel database e successivamente recuperarla.
- Select options: sezione dove selezioniamo il sensore da cui provengono i dati (IMU, Environment o Orientation). Nel nostro caso verrà selezionato solo IMU.

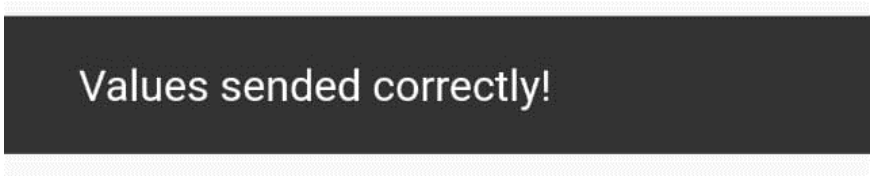
2.5 Creazione del Dataset

Una volta completato lo sviluppo della parte software e hardware il codice viene caricato sul microcontrollore e Arduino viene poi posizionato sull'atleta all'altezza della cintura come mostrato in figura:

Posizione della scheda sull'atleta

Una volta caricato il codice e posizionata la scheda si passa alla parte di registrazione del dataset dove l'atleta dovrà compiere ogni movimento per almeno 50 ripetizioni utilizzando l'applicazione smartphone per specificare (nel momento dell'invio) quale movimento è stato fatto, il nome con cui verrà salvato su Measurify e quale delle caratteristiche viene utilizzata (nel nostro caso IMU).

Nel momento in cui comincia la misura dei dati vediamo in fondo alla pagina il valore di "values" aumentare e, completato il movimento, è necessario aspettare per pochi istanti il feedback che l'invio sia avvenuto correttamente per evitare che l'invio di più misure contemporaneamente causi una sovra scrittura di una sull'altra.



Messaggio di feedback ricevuto dall'applicazione

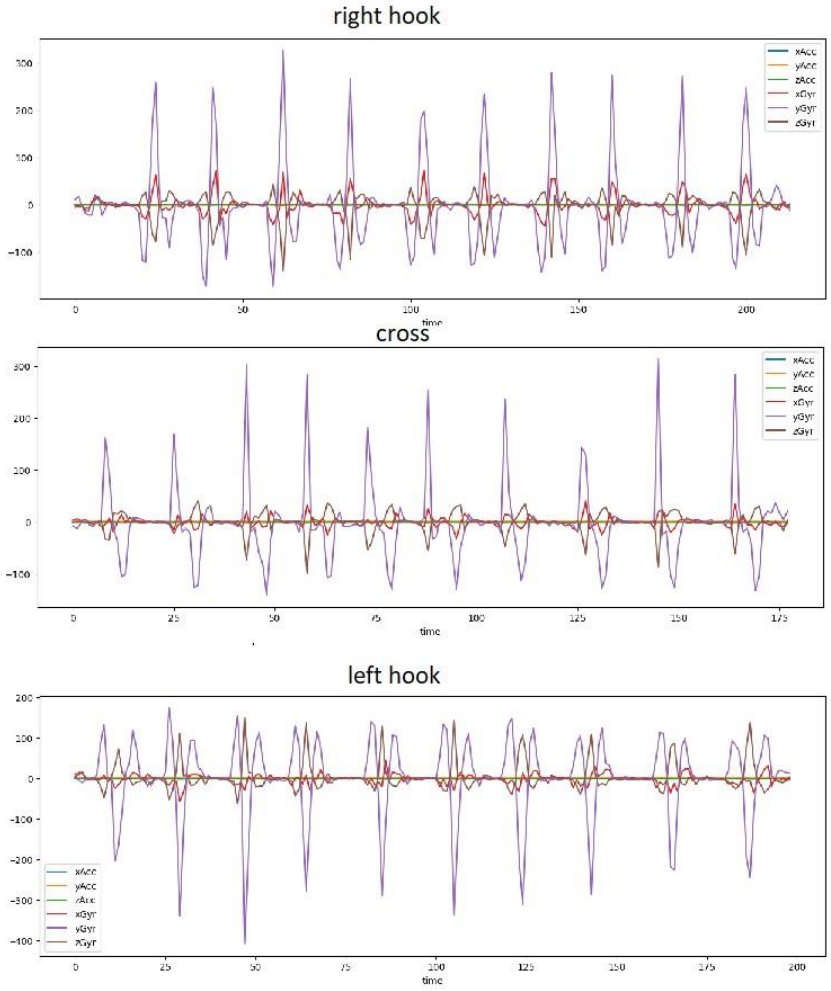
Una volta ricevuto il messaggio rappresentato in figura si può iniziare una nuova misurazione senza rischiare che ci siano errori nell'invio di dati.

Per verificare che l'invio a Measurify sia avvenuto correttamente si può utilizzare Postman per inviare una chiamata GET al server e controllare le misurazioni salvate al momento:

Risposta del server alla chiamata GET

Dalla figura vediamo come per ogni misurazione sono salvati certi parametri come i tags (abbiamo il tenant "Martial_Arts" e il movimento "Left_Hook"), id, feature (nel nostro caso viene utilizzata solo la feature IMU) il device e startDate ed endDate. Queste caratteristiche possono essere usate su Postman per fare delle chiamate con dei "filtri" in cui ricevo, per esempio, tutte le misurazioni di un certo movimento.

Nella prossima immagine possiamo vedere i grafici che si ottengono dai valori dell'accelerazione e della velocità angolare che sono stati rilevati dalla scheda Arduino nel caso di ognuno dei tre movimenti:



Grafici corrispondenti ai movimenti

Infine, Elios Lab si è occupato dello sviluppo e del training di un modello di rete neurale CNN [7] (convolutional neural network) e, una volta completato, è stato modificato in modo tale da poter essere caricato sulla scheda Arduino, cosa non banale dato che come ben sappiamo quest'ultima possiede risorse e potenza di calcolo molto limitate. Una volta che questi passaggi sono stati completati il sistema embedded era pronto per essere testato.

3 SPERIMENTAZIONE E RISULTATI

Il modello addestrato è stato poi compresso in una sua versione più leggera per essere caricato sulla scheda.

Per verificare il corretto funzionamento del modello addestrato è stata verificata la classificazione dei tre movimenti in tempo reale tramite uno script che è stato caricato sull'Arduino. Sul monitor seriale inizialmente vediamo un messaggio che avverte che il sensore è pronto per rilevare il movimento:

```
Starting inferencing in 2 seconds...
Sampling...
```

Messaggio iniziale serial monitor

Tramite connessione seriale viene stampato sul monitor seriale dell'Arduino IDE l'avviso che è iniziato il sampling del movimento stampando i valori di accelerometro e giroscopio rilevati. Una volta rilevati una ventina di campioni sul monitor seriale vediamo le predizioni del modello.

```
Predictions (DSP: 0 ms., Classification: 6 ms., Anomaly: 0 ms.):
    left: 0.00000
    right: 0.00000
    cross: 1.00000
```

Risposta del modello a un cross

Questo significa che il modello neurale riceve i dati del movimento che abbiamo svolto e in base a come è stato addestrato cercherà di capire quale dei tre è stato svolto.

Per quanto riguarda la parte di testing della rete neurale è stato utilizzato come metodo di visualizzazione dei dati la "confusion matrix"[8], una tabella impostata nel seguente modo:

		Predicted condition	
	Total population = P + N	**Positive (PP)**	**Negative (PN)**
Actual condition	**Positive (P)**	**True positive (TP)**	**False negative (FN)**
	Negative (N)	**False positive (FP)**	**True negative (TN)**

Prediction table generica

Ogni movimento è stato svolto per 10 volte con la scheda collegata al computer tramite cavo USB per rendere la visualizzazione dei dati possibile sul monitor seriale e sulla tabella di cui sopra sono stati segnati i movimenti realmente svolti e le predizioni fatte dal modello. I risultati ottenuti sono:

		PREDICTION		
		left hook	right hook	cross
REAL	left hook	10		
	right hook		9	1
	cross			10

Prediction table risultante

Come si può notare il risultato ci suggerisce che il modello sia molto affidabile, nonostante i movimenti svolti coinvolgessero principalmente le braccia e il sensore misurasse invece il movimento del bacino.

4 Contributo personale e considerazioni conclusive

Il progetto è nato con l'idea di assistere l'allenamento dell'atleta monitorando i suoi movimenti nella boxe e quindi dandogli la possibilità di migliorarsi consultando le informazioni raccolte dal sistema dopo che l'allenamento si è concluso. Questo progetto è considerabile un punto di partenza per applicazioni future nelle quali si potrebbe aggiungere una funzionalità che permette all'utente di confrontare i movimenti svolti da lui con i movimenti insegnati dal coach per individuare in quali aree devono lavorare di più. Altre possibili applicazioni possono essere la misura della velocità e della potenza dei colpi oppure quanti colpi vengono svolti e con quale frequenza in una sessione di sparring o addirittura in un incontro. Le possibili applicazioni sono innumerevoli dato che il progetto è stato mantenuto molto generale e quindi applicabile anche a diverse attività sportive.

Un passo successivo potrebbe essere sviluppare una applicazione, per rendere l'utilizzo del sistema più semplice all'utente, e lo sviluppo di un'applicazione software in grado di esporre in modo chiaro e comprensibile i dati raccolti.

Gli strumenti utilizzati sono stati oltre alla scheda, il linguaggio di programmazione C++ nell'IDE Arduino e il Dart per lo sviluppo dell'applicazione software.

I dati raccolti sono stati poi usati per il training del modello di rete neurale convoluzionale (CNN) che, dato che non fa parte delle competenze di una laurea triennale, è stato fornito da Elios Lab.

Il mio contributo personale è stato partecipare nello sviluppo dell'applicazione Flutter (più precisamente nella conversione e invio delle caratteristiche di Environment e Orientation), nella raccolta del dataset e nella verifica della correttezza del modello.

Infine, un modo per rendere il sistema più preciso nel riconoscimento e capace anche di generalizzare meglio nella classificazione dei movimenti sarebbe quello di registrare un dataset più corposo e composto da movimenti svolti da diverse persone dato che al momento è composto da un dataset piuttosto limitato e riferito esclusivamente ai movimenti svolti da me.

5 RIFERIMENTI BIBLIOGRAFICI

[1] Arduino Nano 33 BLE Sense Rev2: https://docs.arduino.cc/hardware/nano-33-ble-sense-rev2

[2] Measurify: https://measurify.org/

[3] Sensore IMU LSM9DS1: https://www.arduino.cc/reference/en/libraries/arduino_bmi270_bmm150/

[4] Arduino IDE: https://www.arduino.cc/en/software

[5] Postman: https://www.postman.com/

[6] Flutter: https://it.wikipedia.org/wiki/Flutter_(software)

[7]CNN:https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/

[8] Confusion Matrix: https://en.wikipedia.org/wiki/Confusion_matrix

Un ringraziamento al professore Riccardo Berta, ai dottori Matteo Fresta e Ali Dabbous per avermi aiutato nello svolgimento di questa tesi. Grazie alla mia famiglia e ai miei amici per essermi stati vicini e avermi sostenuto per il mio percorso universitario.