



**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 2024

**Progetto e implementazione di un sistema embedded multi-sensore per il
riconoscimento di attività umana.**

**Design and development of a multi-sensor embedded system for human
activity recognition**

Candidato: Andreea Elisabeta Vint

Relatore: Prof. Riccardo Berta

Correlatore: Dr. Matteo Fresta

SOMMARIO

Questo elaborato descrive l'implementazione di un sistema embedded multi-sensore per il riconoscimento dell'attività umana. Il focus è stato posto sullo sviluppo di un software per dispositivi mobili, capace di gestire la connessione a molteplici sensori e di raccogliere dati simultaneamente, al fine di migliorare l'accuratezza nella rappresentazione del gesto da analizzare. In questo specifico contesto, l'attività presa in considerazione è la pallavolo, con particolare attenzione alle tecniche del palleggio, del servizio e della schiacciata.

Per raggiungere questo obiettivo, è stato utilizzato il framework Flutter. I sensori sono integrati nelle schede Arduino Nano 33 BLE Sense, che permettono di raccogliere dati relativi all'accelerazione, al giroscopio e al magnetometro. I dati acquisiti vengono inviati all'applicazione Flutter tramite Bluetooth Low Energy (BLE), la quale li trasmette successivamente al framework Measurify, dove possono essere visualizzati ed elaborati ulteriormente.

Lo scopo di questa tesi è, quindi, quello di sviluppare un sistema capace di gestire la connessione e la raccolta dati da molteplici sensori operanti in modo simultaneo, al fine di garantire una rappresentazione più precisa e accurata del gesto che si vuole analizzare.

INDICE

1	Introduzione.....	4
2	Strumenti utilizzati.....	5
2.1	Sistema embedded.....	5
2.2	Applicazione Flutter.....	6
2.3	Measurify.....	12
3	Sperimentazione e risultati.....	14
4	Contributo personale.....	17
5	Riferimenti bibliografici.....	18
6	Ringraziamenti.....	19

1 INTRODUZIONE

Con il termine Internet of Things [1] ci si riferisce all'estensione delle tecnologie internet al mondo degli oggetti, permettendo ad essi di poter comunicare con altri oggetti connessi ed offrire determinati servizi agli utenti, oggetti che acquisiscono di fatto una vera e propria identità digitale. Questi si rendono riconoscibili e acquisiscono intelligenza comunicando dati su sé stessi e avendo accesso alle informazioni su altre things.

Le possibili applicazioni dell'IoT sono molto vaste e vanno dalla domotica, con sistemi di automazione domestica che controllano illuminazione, riscaldamento, sicurezza e altri aspetti legati alla casa, alla sanità, mediante dispositivi medici indossabili, fino all'agricoltura, mediante l'implementazione di sistemi di irrigazione intelligenti e monitoraggio delle colture, passando per le città intelligenti con veicoli sempre connessi, gestione del traffico e ottimizzazione delle rotte.

Nello specifico questo progetto mira ad applicare il concetto dell'IoT al campo sportivo, in modo particolare alla pallavolo, creando un sistema embedded multi-sensore di riconoscimento della tecnica di base eseguita, rappresentando anche un modo per monitorare lo sviluppo nel tempo della propria attività pallavolistica.

I valori rilevati dalle IMU vengono spediti al dispositivo mobile, il quale mediante un'applicazione appositamente sviluppata, si interfaccia con i sistemi Arduino connessi.

In seguito, lo smartphone android collegato ad internet invierà i valori registrati tramite l'utilizzo di HTTPS, al cloud, dove verranno conservati, come si può osservare in figura 1.

Il progetto consiste quindi nello sviluppo dell'applicazione mobile tramite Framework Flutter, rendendola funzionale con molteplici sensori che raccolgono i dati, e li spedisce al database Measurify in cui verranno conservati. Questi dati infine verranno utilizzati per generare un modello capace di classificare i movimenti in tempo reale.

Come già accennato, il focus di questa tesi verte principalmente sullo sviluppo software di un'app per smartphone che permette di implementare un sistema di rilevamento del movimento multi-sensore. Questo sistema è particolarmente utile in quanto consente una rappresentazione più accurata del movimento dell'atleta, creando un modello preciso e affidabile per monitorare lo sviluppo della tecnica nel tempo e ridurre il rischio di infortuni causati da un'esecuzione scorretta del gesto tecnico.

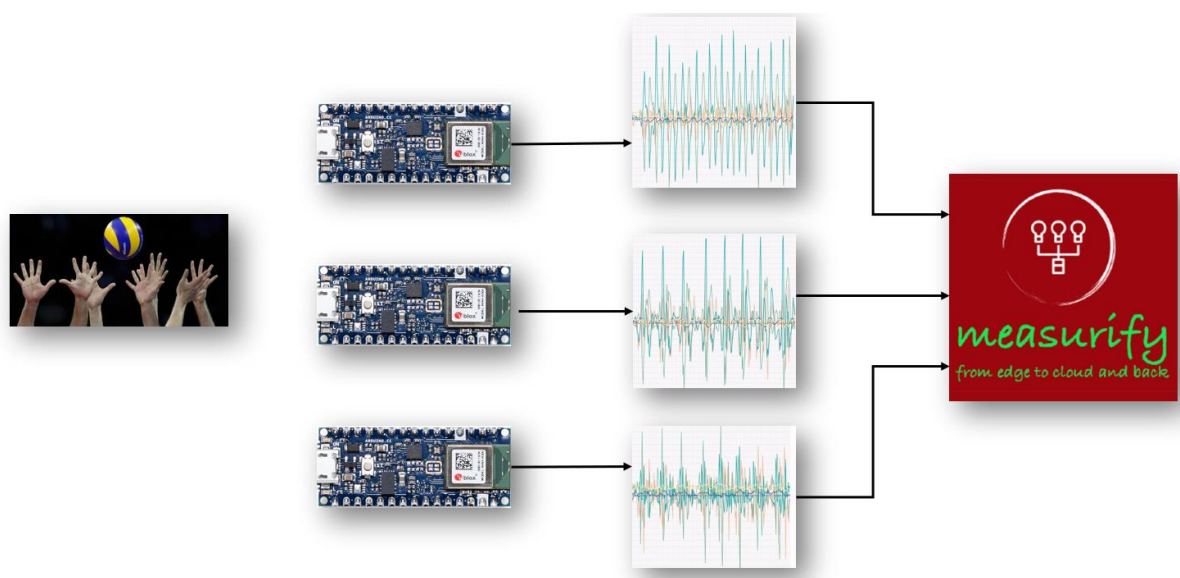


Figura 1. Flusso del lavoro

2 STRUMENTI UTILIZZATI

2.1 Sistema embedded

I sistemi embedded utilizzati per questa tesi sono gli Arduino Nano 33 BLE Sense [2].

Questi sono dei microcontrollori in grado di rilevare dati dall'ambiente esterno, attraverso i moduli IMU (Inertial Measurement Unit) [3]. L'IMU è dotata tipicamente di un accelerometro, un giroscopio e un magnetometro, permettendo di misurare l'accelerazione, la rotazione e l'orientamento nello spazio. Questi sensori sono fondamentali per applicazioni quali appunto il rilevamento dell'attività umana.



Figura 2. Arduino Nano 33 BLE Sense

Per funzionare secondo le necessità di questo progetto, gli Arduino sono stati dotati di un circuito extra, sviluppato dal laboratorio Elios, per permettere l'utilizzo di batterie ricaricabili, in modo da ovviare al problema di dover sempre essere collegati tramite cavo ad una fonte di alimentazione, che altrimenti avrebbe limitato l'utilizzo di questi dispositivi per il mio scopo.

Tramite l'IDE [4] Arduino è stato caricato il codice sui microcontrollori per permettere la connessione BLE e l'invio dei dati. Come specificato precedentemente i dati inviati provengono unicamente dall'unità IMU, che fornisce informazioni sull'accelerazione, sulla velocità angolare e sul campo magnetico, permettendo in tal modo di fornire una rappresentazione tridimensionale completa.

La comunicazione Bluetooth low energy presenta notevoli limiti in termini di quantità di dati che possono essere trasmessi per pacchetto, limite imposto a 20 bytes; perciò, si è ritenuto opportuno convertire i valori da float32 a int16 mediante l'aggiunta di costanti moltiplicative, osservate in figura 3, per ciascuno dei nove valori. Una volta che questi dati vengono inviati all'applicazione flutter, vi è un'apposita funzione che converte nuovamente questi numeri nel loro formato originale (figura 5).

Il periodo di campionamento è stato impostato a 50 ms.

```
#define ACC_MULTIPLIER 8192
#define GYR_MULTIPLIER 16.384
#define MAG_MULTIPLIER 81.92
#define TEMP_MULTIPLIER 100
#define HUM_MULTIPLIER 100
#define PRES_MULTIPLIER 100
```

Figura 3. Costanti moltiplicative presenti nel codice per Arduino

```

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));
}

```

Figura 4. Funzione che si occupa di convertire in interi

```

//convert the IMU data 9 int16 array back to float and create the json object ready to be send
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;
}

```

Figura 5. Funzione in Dart che riconverte gli interi in float

2.2 Applicazione Flutter

Sono partita da una applicazione già pienamente funzionale, ovvero in grado di connettersi ad un Arduino e trasmettere i dati raccolti da quest'ultimo al cloud Measurify. Lo scopo della mia tesi è stato quello di modificare l'applicazione mobile in modo da potersi collegare anche a due o più Arduino e raccogliere dati simultaneamente ed indipendentemente da ciascuno, ed infine fare tante richieste di POST HTTPS su Measurify quanti sono i dispositivi.

Il framework utilizzato per lo sviluppo è Flutter [5], un framework opensource di Google che permette lo sviluppo di applicazioni multiplatforma, il cui linguaggio nativo è Dart. Questo framework si basa sull'utilizzo di widget per sviluppare tutte le funzionalità della User Interface, widget che possono essere stateless o stateful, a seconda che si voglia o meno modificare lo stato del sistema.

L'applicazione, chiamata Smart Collector [6], utilizza una libreria chiamata quick_blue in grado di trovare e connettersi a dispositivi dotati di connessione Bluetooth Low Energy, scoprire i servizi che questi mettono a disposizione e trasferire dati fra dispositivi e server.

Come già sottolineato in precedenza Smart Collector è un'applicazione già pienamente funzionale, composta da diverse pagine, tra cui le più importanti per questa tesi sono il Main, ovvero la schermata principale, la StartPage e PeripheralDetailPage, su cui mi sono appunto maggiormente concentrata in quanto hanno dovuto subire il maggior numero di modifiche affinché l'applicazione funzioni con molteplici sensori.

La pagina iniziale Main è adibita alla scannerizzazione per la rilevazione di dispositivi Bluetooth.

Quest'ultima è stata modificata in modo da dare la possibilità di poter selezionare più dispositivi, come si vede dall'immagine in figura 6. Questo è stato possibile mediante l'utilizzo di un apposito widget che è presente in Flutter e cioè CheckboxListTile (figura 7), che permette di selezionare più dispositivi mediante dei riquadri presenti di fianco ad essi, aggiungendoli ad un array che di fatto contiene gli Arduino a cui ci vogliamo collegare.

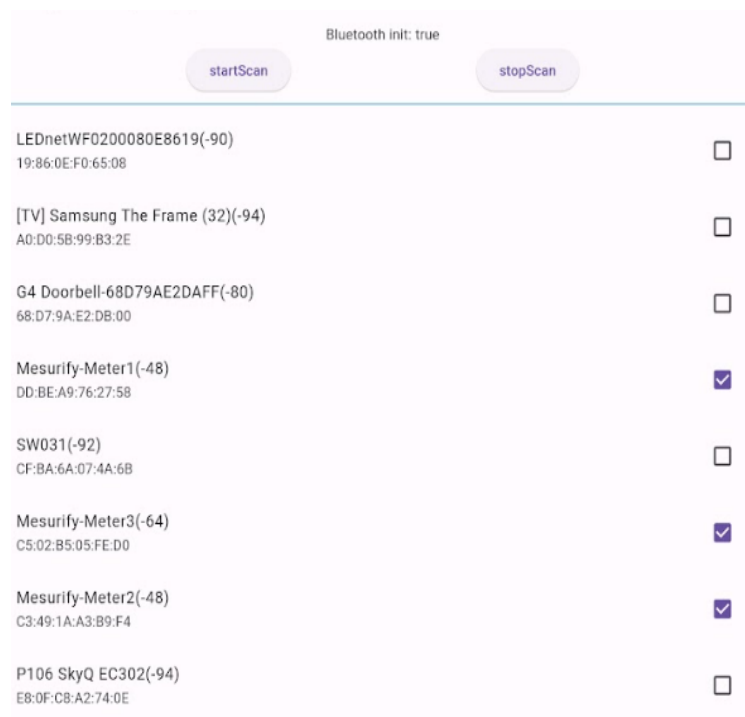


Figura 6. Schermata che permette di selezionare più dispositivi

```

Widget _buildListView() {
  return Expanded(
    child: ListView.builder(
      itemCount: _scanResults.length,
      itemBuilder: (BuildContext context, index) {
        return CheckboxListTile(
          title: Text(
            '${_scanResults[index].name}(${_scanResults[index].rssi})',
          ),
          subtitle: Text(_scanResults[index].deviceId),
          //value: selectedStates[index],
          value: selectedItems.contains(_scanResults[index].deviceId),
          onChanged: (bool? value) {
            setState(() {
              //if (value == null) {
              //  print('There are no devices selected');
              if (value == true) {
                selectedItems.add(_scanResults[index].deviceId);
              } else {
                selectedItems.remove(_scanResults[index].deviceId);
              }
            });
          },
        );
      },
    ),
  );
}

```

Figura 7. Funzione in Flutter per aggiungere più dispositivi Bluetooth ad un array

Questo array a cui abbiamo aggiunto i dispositivi a cui vogliamo collegarci viene mandato ad una seconda pagina, PeripheralDetailPage, dove avviene la connessione vera e propria. Questa pagina è stata predisposta affinché possa prendere come argomento la lista con gli Arduino interessati, come si vede in figura 8. Nell'immagine sottostante si può osservare come il codice è stato modificato, per ottenere il risultato desiderato.

```

class PeripheralDetailPage extends StatefulWidget {
  // final String deviceId;
  final List<String> selectedItems;

  //PeripheralDetailPage(this.SdeviceId);
  PeripheralDetailPage({required this.selectedItems});
}

```

Figura 8. Modifiche apportate agli argomenti di PeripheralDetailPage

La connessione simultanea può essere ottenuta iterando la funzione `Quickblue.connect`, presente nella libreria, su ciascun elemento dell'array dei dispositivi selezionati, filtrando in base al rispettivo `deviceId`. In questo modo l'array contenente gli Arduino precedentemente selezionati diventa una lista con gli apparecchi effettivamente connessi. Lo stesso procedimento è stato eseguito per disconnettere i sensori, come si può osservare nella figura 9.

<pre> ElevatedButton(child: Text('connect'), onPressed: connecting ? null : () { setState(() { connecting = true; }); for (var deviceId in widget.selectedItems) { QuickBlue.connect(deviceId); } },) </pre>	<pre> ElevatedButton(child: Text('disconnect'), onPressed: isConnected ? () { setState(() { isConnected = false; connecting = false; }); for (var deviceId in widget.selectedItems) { QuickBlue.disconnect(deviceId); } } : null,) </pre>
--	---

Figura 9. Codice per connettere e disconnettere tutti gli Arduino presenti nella lista selectedItems

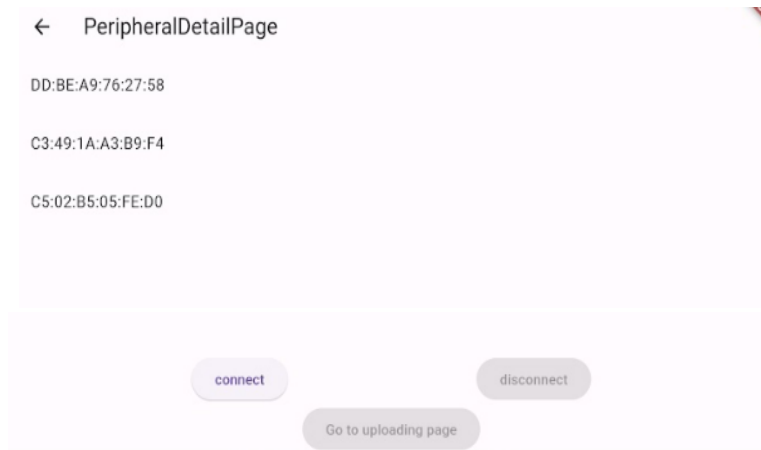


Figura 10. La pagina dell'app Flutter in cui si ha la funzione Connect

Per una corretta gestione e differenziazione dei dati raccolti dalle varie IMU, è stato modificato in StartPage la funzione `handleValueChange`; questa è responsabile della raccolta dei dati, precedentemente convertiti in formato JSON, in una lista, permettendo di fatto di raccogliere i dati in un array di arrays, discriminando la provenienza dei dati in base all'indice con cui i dispositivi connessi sono posizionati nell'array. Infatti, in base proprio a questo indice possiamo poi recuperare i dati dalla lista per essere opportunamente processati, come si evince dalla figura 11.

```
List<List<Map<String, dynamic>>> receivedIMUJsonValues = [];
void _handleValueChange(
  String deviceId,
  String characteristicId,
  Uint8List value,
) {
  int index = widget.globals.selectedItems.indexOf(deviceId);

  if (characteristicId == "8e7c2dae-0002-4b0d-b516-f525649c49ca") {
    // IMU characteristic
    Map<String, dynamic> jsonObj = parseIMUData(value, 9);
    setState(() {
      if (widget.globals.receivedIMUJsonValues.length <= index) {
        // If the index does not exist, expand the list
        widget.globals.receivedIMUJsonValues.add([]);
      }
      widget.globals.receivedIMUJsonValues[index].add(jsonObj);
    });
  }
}
```

Figura 11. Codice Flutter per trattare i dati provenienti da dispositivi diversi, che vengono inseriti nell'array di arrays `receivedIMUJsonValues`, in base all'index del dispositivo che trasmette i dati

Anche la funzione `sendData` (il cui scopo come si evince dal nome è proprio quello di inviare i dati raccolti, in figura 12) è stata modificata di conseguenza, permettendo sempre di discriminare la provenienza dei dati in base all'indice in cui sono posizionati nell'array, correlato quindi con l'indice del dispositivo. Come si vede dall'immagine sottostante, oltre a distinguere i dati in base all'Arduino di provenienza, questa funzione esegue una richiesta HTTPS di POST in modo da trasmettere i dati in base alla fonte da cui vengono inviati, effettuando tanti POST quanti sono i dispositivi che effettivamente trasmettono dati.

```

Future<void> sendData(String deviceId) async {
  int index = widget.globals.selectedItems.indexOf(deviceId);

  print("Sending data to " + widget.globals.measureName.toString());
  String jsonDataString =
    | | jsonEncode(widget.globals.receivedIMUJsonValues[index]);
  print(jsonDataString);
  var headers = {
    | 'Content-Type': 'application/json',
    | 'Authorization': widget.globals.deviceToken
  };
  var request = null;

  request = http.Request(
    | 'POST',
    | Uri.parse(widget.globals.url +
    | | "measurements/" +
    | | widget.globals.measureName +
    | | '_device' +
    | | index.toString() +
    | | '/timeserie'));

  request.body = jsonDataString;
  request.headers.addAll(headers);

  http.StreamedResponse response = await request.send();
}

```

Figura 12. Funzione sendData che recupera i dati dalla lista che li contiene, in base all'index dell'Arduino e fa la richiesta di trasmissione dati

Come penultimo step sono state opportunamente cambiate anche le funzioni checkMeasureExists e postMeasure, la prima che si occupa di verificare che effettivamente siano presenti delle misurazioni con il corrispettivo nome, mentre PostMeasure si occupa di fare le varie richieste di POST HTTPS su Measurify, con il corpo della richiesta contenente i dati che si vogliono inviare. Le modifiche apportate rispecchiano quelle già viste nelle precedenti funzioni, ovvero di fare in modo che le richieste di POST per le misurazioni siano tante quanto sono gli Arduino che trasmettono, e che i nomi con cui vengono pubblicate su Measurify contengano l'index in cui sono posizionati nell'array dei dispositivi connessi. Le due funzioni sono illustrate nelle figure 13 e 14.

```

//Check if exist a measure with that name //
Future<void> checkMeasureExist(String deviceId) async {
  int index = widget.globals.selectedItems.indexOf(deviceId);
  if (widget.globals.measureName == '') {
    | await postMeasure(deviceId);
  } else {
    | var headers = {
    | | 'Content-Type': 'application/json',
    | | 'Authorization': widget.globals.deviceToken
    | };
    | //var request = null;

    | var request = http.Request(
    | | 'GET',
    | | Uri.parse(widget.globals.url +
    | | | "measurements/" +
    | | | widget.globals.measureName +
    | | | "_device" +
    | | | index.toString()));

    | request.headers.addAll(headers);

    | http.StreamedResponse response = await request.send();
  }
}

```

Figura 13. Funzione per verificare l'esistenza di una misurazione

```

//Create measurement to save data
Future<void> postMeasure(String deviceId) async {
  var headers = {
    'Content-Type': 'application/json',
    'Authorization': widget.globals.deviceToken
  };
  var request =
    http.Request('POST', Uri.parse(widget.globals.url + "measurements"));
  // Prepare the data to be sent in the request body as a Map
  Map<String, dynamic> requestBody = {
    "thing": widget.globals.thingName,
    "feature": "IMU",
    "device": widget.globals.deviceName,
    "tags": [_selectedOption1, _selectedOption2],
    "visibility": "public"
  };
  if (widget.globals.measureName != '' &&
    widget.globals.measureName.isNotEmpty) {
    // If the measureName is available, include it in the request body
    requestBody["_id"] = widget.globals.measureName +
      "_device" +
      widget.globals.selectedItems.indexOf(deviceId).toString();
  }
  // JSON encode the data in the request body
  String requestBodyJson = json.encode(requestBody);
  request.body = requestBodyJson;
  request.headers.addAll(headers);

  http.StreamedResponse response = await request.send();
}

```

Figura 14. Funzione adibita ad effettuare POST di HTTPS contenente nel body della richiesta i dati che si vogliono inviare

Come ultimo passo, si è iterato sulla lista di dispositivi connessi (figura 15), mediante cicli for, in modo da poter fare la richiesta di trasmissione di dati e di conseguenza l'invio di questi, per ciascun dispositivo; inoltre, il codice è stato predisposto in modo da notificare tramite l'apposita funzione setNotifiable di quick_blue, ciascun deviceId nella nostra lista.

<pre> //create the measurement and start notification for (int i = 0; i < widget.globals.selectedItems.length; i++) { await checkMeasureExist(widget.globals.selectedItems[i]); } for (int i = 0; i < widget.globals.selectedItems.length; i++) { QuickBlue.setNotifiable(widget.globals.selectedItems[i], widget.globals.bleServiceId, widget.globals.imuCharacteristicId, BleInputProperty.notification, </pre>	<pre> //stop notification and send data for (int i = 0; i < widget.globals.selectedItems.length; i++) { QuickBlue.setNotifiable(widget.globals.selectedItems[i], widget.globals.bleServiceId, widget.globals.imuCharacteristicId, BleInputProperty.disabled,); } for (int i = 0; i < widget.globals.selectedItems.length; i++) { sendData(widget.globals.selectedItems[i]); } </pre>
---	--

Figura 15. Codice Flutter per iterare le funzioni su tutti gli Arduino

Nell'ultima pagina dell'applicazione mobile, nella StartPage, osservabile in figura 16, si potrà scegliere l'attività umana da monitorare, in questo caso trattasi della pallavolo. Sono state predisposte dal laboratorio Elios alcune tecniche fondamentali che sono il servizio, il palleggio e la schiacciata, di cui si vogliono appunto rilevare i movimenti eseguiti, mediante la voce Choose Action.

Mediante la voce Select options si seleziona infine la caratteristica a cui siamo interessati, ovvero l'IMU.

Per dare inizio alla registrazione dei dati si seleziona Start.

Figura 16. Pagina per la selezione dell'attività e l'invio dei dati

2.3 Measurify

Al fine di memorizzare i dati raccolti, è stato usato Measurify [7], un framework di tipo RESTful [8], opensource, necessario per la gestione dei dispositivi intelligenti nel sistema IoT.

Measurify permette di gestire le misurazioni raccolte, rendendoli disponibili ad ulteriori elaborazioni o visualizzazione, come per esempio nella applicazione mobile Flutter.

Per capire meglio il suo funzionamento prendiamo in considerazione i concetti fondamentali su cui è modellato:

-THING: si riferisce appunto ad una “cosa” che può essere una persona, un ambiente oppure un oggetto sul quale si vogliono fare delle misurazioni.

-FEATURE: si tratta dell'oggetto della misurazione.

-DEVICE: il dispositivo in grado di misurare una certa dimensione di una determinata “cosa”.

-MEASUREMENT: è la misura effettuata dal dispositivo su una determinata thing.

TAG: è un'etichetta che si può aggiungere ad una misurazione, utile per discriminare tra diverse measurements.

Per lo scopo di questa tesi, i dispositivi Arduino sono i device, l'atleta è la thing, e la feature è misurata dai sensori presenti sui device.

Per salvare i dati si utilizzano i verbi di HTTPS (GET, POST, PUT, DELETE), contente ciascuno un body in formato JSON, utilizzato come standard all'interno del web per lo scambio di informazioni.

HTTPS [9] è un protocollo application-level, generico e stateless, usato per diversi scopi in base ai verbi utilizzati, agli headers ed ai codici d'errore. Rappresenta un metodo standard di comunicazione tra terminali diversi. È caratterizzato da alcune proprietà come l'essere connectionless, ovvero il client ed il server sono a conoscenza l'uno dell'altro solo durante la comunicazione dopo la quale si “dimenticano”; è media-independent cioè qualsiasi

tipo di dato può essere trasmesso a patto che sia il client che il server sappiano maneggiare il rispettivo dato, necessitando di fatto di specificare il dato mediante il suo tipo MIME appropriato; ed è stateless ovvero server e client sono a conoscenza l'uno dell'altro solo durante la connessione e non memorizzano informazioni sul tipo di richiesta effettuato. I metodi che HTTPS usa sono GET, POST, PUT, DELETE e le richieste contengono un header in cui sono specificate informazioni aggiuntive per la richiesta, ed il body, in cui è presente il dato che è richiesto. La risorsa, ovvero il target di una richiesta viene identificata tramite il suo URL.

Per poter creare dei POST verso le API bisogna innanzitutto disporre di un token con cui fare l'autenticazione, che si ottiene mediante una chiamata POST con all'interno username e password dello studente.

Welcome back Vint

Role: Provider

Tenant volleyball-tenant

Logout

Resources

Tags

Things

Features

Devices

Measurements



















thing	feature	device	startDate	tags	_id	Management
user1	IMU	edge-meter	2024-07-31T21:13:48.712Z	[Volleyball , Palleggio]	palleggio20_device2	  
user1	IMU	edge-meter	2024-07-31T21:13:48.412Z	[Volleyball , Palleggio]	palleggio20_device1	  
user1	IMU	edge-meter	2024-07-31T21:13:48.162Z	[Volleyball , Palleggio]	palleggio20_device0	  
user1	IMU	edge-meter	2024-07-31T20:50:52.607Z	[Volleyball , Servizio]	servizio20_device2	  
user1	IMU	edge-meter	2024-07-31T20:50:51.981Z	[Volleyball , Servizio]	servizio20_device1	  
user1	IMU	edge-meter	2024-07-31T20:50:51.313Z	[Volleyball , Servizio]	servizio20_device0	  

Figura 17. Dashboard di Measurify

Una volta che le misurazioni sono state registrate sarà possibile tramite Visualize Timeseries (figura 18) illustrare i dati ottenuti su di un grafico che coinvolge tutte le 9 variabili presenti, potendo inoltre isolare una o più di esse, e si potrà inoltre selezionare anche il numero di campioni che si vuole esaminare, rispetto alla totale lunghezza della misurazione.

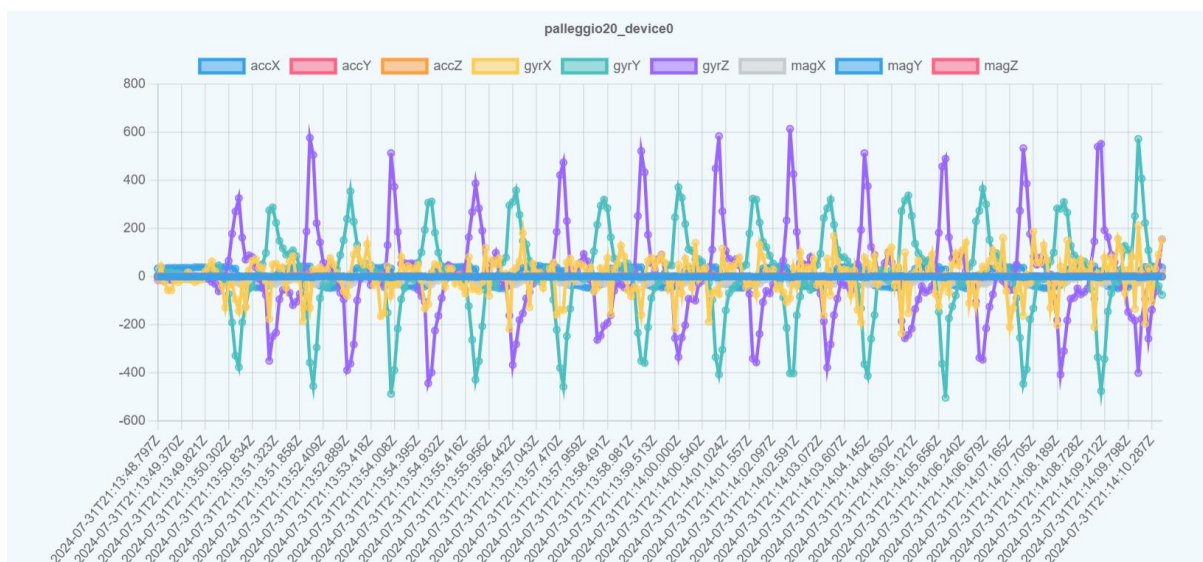


Figura 18. Esempio di una serie temporale visualizzata su Measurify

3 SPERIMENTAZIONE E RISULTATI

Dopo aver completato lo sviluppo dell'applicazione Flutter, sono passata alla fase di testing del progetto: la rilevazione dei movimenti attraverso l'utilizzo dei microcontrollori Arduino e la trasmissione dei dati raccolti su Measurify. Questo passaggio ha richiesto una particolare attenzione sia alla configurazione dei dispositivi sia al contesto operativo in cui sarebbero stati utilizzati.

I microcontrollori Arduino sono stati programmati utilizzando l'IDE ufficiale, con codici specifici per garantire il corretto funzionamento e la rilevazione accurata dei dati. Successivamente, questi dispositivi sono stati posizionati strategicamente su diverse parti del corpo: tre unità, distribuite con cura su un braccio, il busto e una gamba, come illustrato nella figura 19. Il posizionamento dei sensori è stato un aspetto cruciale del progetto. Non solo è stato necessario fissarli saldamente per assicurare letture precise durante i movimenti, ma è stato anche fondamentale evitare interferenze che potessero causare il reset dei dispositivi, interrompendo la trasmissione dei dati.

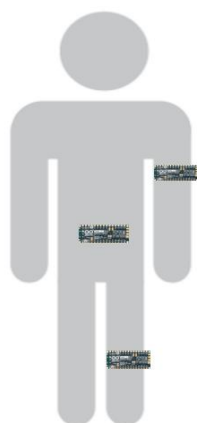


Figura 19. Posizionamento degli Arduino

Una volta posizionati correttamente, i dispositivi Arduino sono stati connessi all'applicazione Flutter tramite Bluetooth Low Energy (BLE). Questa connessione ha permesso di selezionare all'interno dell'app l'attività e la tecnica specifica che si desiderava monitorare. Le misurazioni effettuate hanno compreso parametri come l'accelerazione, la velocità angolare e la gravità. Le ripetizioni registrate per ciascuna tecnica sono state venti.

Al termine della rilevazione, i dati raccolti sono stati inviati automaticamente su Measurify. Questo processo ha richiesto qualche secondo per garantire che tutti i dati fossero correttamente trasferiti e archiviati. Durante questo intervallo, è stato importante attendere il messaggio di conferma dell'avvenuta trasmissione per evitare di compromettere i dati salvati. Solo dopo la comparsa del messaggio illustrato di seguito è stato possibile procedere con una nuova sessione di registrazione, assicurandosi che i dati precedenti fossero stati gestiti correttamente.

Values sended correctly!

Le tecniche della pallavolo che si è voluto studiare sono tre: il palleggio, il servizio e l'attacco.

- Il palleggio (figura 20) è una delle tecniche fondamentali della pallavolo, utilizzata principalmente per eseguire passaggi precisi tra i giocatori. È un gesto tecnico che richiede coordinazione, precisione e un buon controllo del pallone. Il palleggio viene eseguito impiegando entrambe le mani, con le dita ben aperte, formando una specie di "coppa" che avvolge il pallone. Le mani devono essere posizionate sopra la fronte, con i gomiti leggermente piegati e aperti. Quando la palla arriva, l'atleta deve ammortizzare il suo impatto piegando i polsi e le dita, spingendola poi verso l'alto con un movimento congiunto delle braccia e delle gambe. In particolare, io mi sono focalizzata sul palleggio in salto, una tecnica che consente di eseguire il palleggio mentre si è in aria. L'atleta infatti esegue un salto esplosivo e solo quando è al culmine del salto le mani entrano in contatto con la palla. I vantaggi di questa tecnica sono

una maggior velocità di gioco, una maggior elevazione offrendo più opzioni tattiche, nonché maggior imprevedibilità per l'avversario.



Figura 20. Il palleggio [10]

- Il servizio nel volley può essere eseguito mediante varie tecniche, tra cui Jump Float e Jump Spin. Il Jump Spin ha molto in comune con la tecnica della schiacciata che vedremo in seguito, per cui ho preferito prendere in esame la Jump Float (figura 21). Questa, è una tecnica che combina una rincorsa con un salto finale in cui la palla viene impattata in modo “flottante”, ovvero non viene impresso nessun tipo di rotazione. Alla partenza il pallone viene tenuto di solito con la mano non dominante, e mentre si esegue la rincorsa, che è molto simile a quella che si può osservare per la schiacciata, seppur più corta e meno esplosiva, viene lanciato il pallone. Quando l'atleta arriva alla massima altezza colpisce il pallone con la mano tesa in modo da imprimere l'effetto “float”, da cui il nome.



Figura 21. Il servizio [10]

- La schiacciata (figura 22) è la principale tecnica offensiva nella pallavolo che richiede una combinazione di tecnica, strategia e potenza. È un colpo molto potente e molto impattante che può segnare punti diretti oppure mettere in difficoltà gli avversari, costretti ad una ricostruzione del gioco in condizioni non ottimali. Come anche per il servizio, il colpo sulla palla viene eseguito in seguito ad una rincorsa; l'atleta si posiziona esternamente al campo da gioco, ad una distanza di circa tre metri dalla rete, ed inizia con un passo corto e controllato, continuando con due passi ben sincronizzati, utili a dare l'esplosività al salto e quindi la spinta verso l'altezza massima raggiungibile. Al momento del salto, le gambe devono dare la spinta finale verso l'alto con le braccia che vengono portate in avanti, e la mano dominante pronta a colpire la palla con forza oltre che con precisione; infatti, sarà proprio la mano a indirizzare la palla nella porzione di campo che si desidera colpire.



Figura 22. La schiacciata [10]

Nell'immagine riportata in Figura 23, è presentato un esempio di rilevazione della tecnica del palleggio, ottenuto attraverso la dashboard di Measurify. I dati sono stati registrati utilizzando simultaneamente tre diversi sensori Arduino, e nell'immagine sono chiaramente visibili dei pattern ripetitivi che rappresentano i movimenti del palleggio.



Figura 23. Esempio delle misurazioni effettuate per il palleggio

4 CONTRIBUTO PERSONALE

L'idea centrale della mia tesi nasce dall'esigenza di sviluppare un sistema embedded multi-sensore avanzato, dedicato al rilevamento e all'analisi delle attività umane in tempo reale. Questo sistema si basa su una rete di sensori integrati con dispositivi Arduino, che permette di monitorare in modo continuo e preciso i movimenti e i comportamenti degli utenti. La mia specifica responsabilità all'interno di questo progetto è stata lo sviluppo di un'applicazione mobile utilizzando Flutter. Quest'app è stata progettata per gestire in modo efficiente la comunicazione con molteplici Arduino, raccogliendo e visualizzando i dati provenienti dai sensori in una piattaforma unica e accessibile. I dati raccolti vengono poi inviati a un server cloud chiamato Measurify, dove vengono memorizzati in modo sicuro e da cui possono essere utilizzati per ulteriori analisi. Questo approccio consente, tra le altre cose, di costruire modelli avanzati di riconoscimento delle attività, migliorando così la capacità del sistema di adattarsi a diverse esigenze e applicazioni.

Tra le sfide affrontate, ho dovuto apprendere un nuovo linguaggio di programmazione, Dart, che è alla base del framework Flutter, oltre a comprendere il funzionamento del sistema nel suo complesso. Questo ha richiesto un notevole sforzo nell'integrare le mie conoscenze esistenti con nuove competenze, sia nello sviluppo mobile che nella gestione dei dispositivi embedded, assicurando comunque un risultato finale in linea con gli obiettivi prefissati per la tesi.

In futuro, si potranno effettuare misurazioni da un numero maggiore di persone, ampliando così il dataset e rendendolo più completo. Questo permetterà di addestrare il modello di riconoscimento con una maggiore varietà di dati, migliorandone l'accuratezza e la capacità di generalizzare a diversi contesti e utenti. L'obiettivo è creare un sistema sempre più robusto e preciso, capace di riconoscere le attività umane con un alto grado di affidabilità, grazie a un training basato su dati eterogenei e rappresentativi di un'ampia gamma di situazioni reali.

5 RIFERIMENTI BIBLIOGRAFICI

- [1] Internet of things: https://it.wikipedia.org/wiki/Internet_delle_cose
- [2] Arduino nano 33: <https://docs.arduino.cc/hardware/nano-33-ble/>
- [3] IMU: <https://docs.arduino.cc/tutorials/nano-rp2040-connect/rp2040-imu-advanced/>
- [4] IDE: <https://www.arduino.cc/en/software>
- [5] Flutter: <https://docs.flutter.dev/>
- [6] Smart Collector: <https://github.com/measurify/smart-collector>
- [7] Measurify: <https://github.com/measurify>
- [8] REST: <https://www.ibm.com/it-it/topics/rest-apis>
- [9] HTTPS: <https://it.wikipedia.org/wiki/HTTPS>
- [10] <https://www.istockphoto.com/it/search/2/image?mediatype=illustration&phrase=battuta%20pallavolo>

6 RINGRAZIAMENTI

Desidero esprimere la mia profonda gratitudine al Prof. Riccardo Berta per avermi concesso questa preziosa opportunità, permettendomi di partecipare a questo progetto che mi ha consentito di consolidare le conoscenze acquisite durante il percorso di laurea e di apprenderne di nuove. Un ringraziamento speciale va anche al mio corelatore Matteo Fresta per la pazienza dimostrata e per il grande sostegno offertomi.

Vorrei inoltre ringraziare i miei genitori per avermi sempre incoraggiata a migliorare e per aver sostenuto costantemente le mie idee.