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

Dicembre 2022

Progetto e realizzazione di un sistema embedded per la gestione di corse automobilistiche

(Design and development of an embedded system for motor racing management)

Candidato: Alessandro Cammalleri

Relatore: Prof. Riccardo Berta

Correlatore: Luca Lazzaroni

Sommario

Il presente elaborato è stato redatto con l'obiettivo di simulare la gestione di corse automobilistiche, in particolare delle sessioni di qualifica in circuito.
Il progetto è composto da due sistemi indipendenti; il primo si occupa di individuare la presenza di un veicolo in pista e, conseguentemente, rilevare ed elaborare dati utili per il riconoscimento della vettura e il tempo sul giro della macchina stessa; il secondo si occupa di acquisire informazioni necessarie per la gestione della telemetria dell'auto. Successivamente tutte le informazioni acquisite dai software vengono inviate a un API (*Application Protocol Interface*) Cloud chiamata Measurify [1]. Infine, per consentire una rappresentazione dei dati *user-friendly*, è stata creata un'applicazione *cross-platform* che consente di visualizzare i dati rilevati tramite grafici per la telemetria delle macchine, e tabelle per i tempi di percorrenza delle vetture.

1. Introduzione

La presente tesi ha come oggetto il progetto e la realizzazione di un sistema in grado di gestire una sessione di qualifica in un circuito automobilistico utilizzando strumenti economicamente accessibili, per fornire una valida alternativa ad un sistema regolato da standard molto sofisticati e costituito da sensori e software complessi.

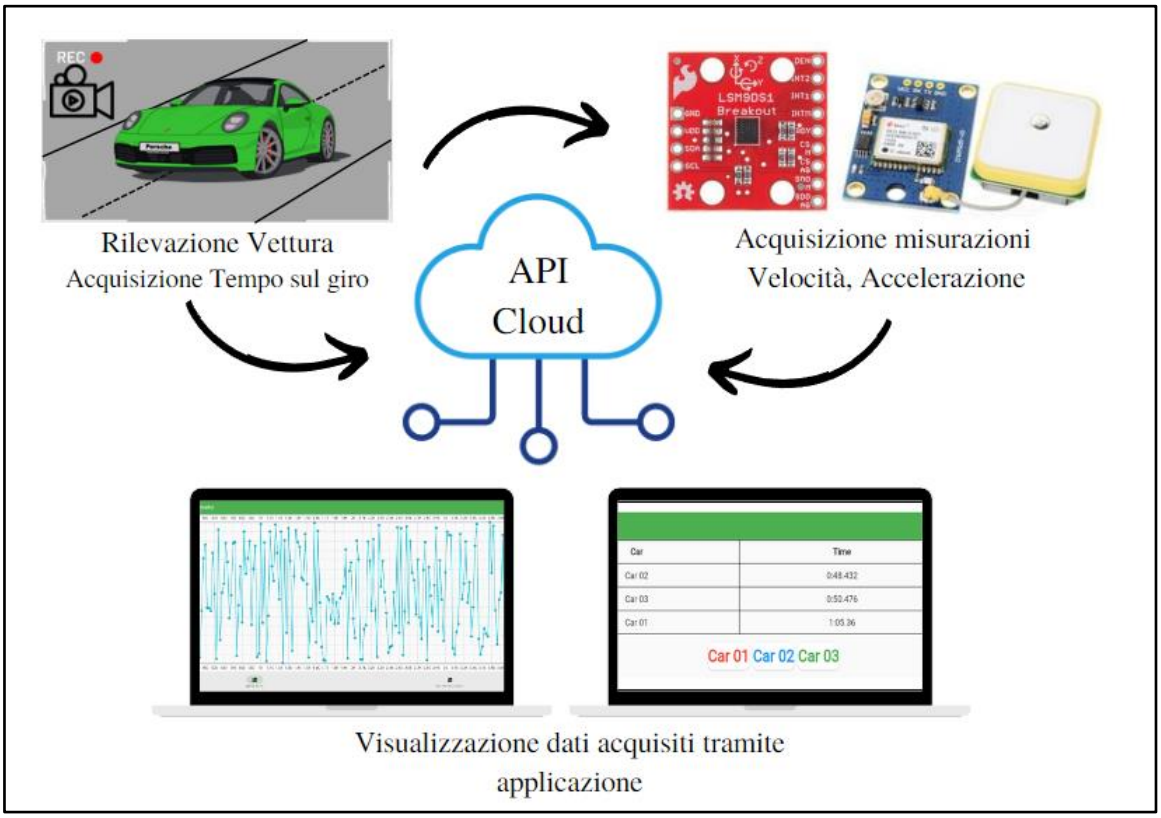

Fig. 1.1 -flusso di acquisizione dati-

L'immagine in alto mostra il corretto processo di acquisizione dei dati dai sistemi.
Come mostrato nella parte superiore, i due sistemi indipendenti si occupano di ottenere misurazioni differenti. Il primo, grazie ad una telecamera integrata, è in grado di rilevare il passaggio di ogni vettura in un punto specifico del circuito e, attraverso un'elaborazione dell'immagine, riconoscere quale auto è passata; una volta ottenute queste informazioni, il sistema associa alla vettura il tempo di percorrenza da essa compiuto.
Il secondo sistema è posizionato all'interno di ogni vettura e si occupa di analizzare i dati relativi alla velocità e accelerazione rispettivamente grazie ad un sensore IMU e un sensore GPS collegati ad un microcontrollore ESP32.
Una volta che la sessione di qualifica è terminata, tramite l'applicazione è possibile consultare i dati raccolti, come la classifica finale dettata dal tempo sul giro, oppure grafici telemetrici di ogni giro di ciascun veicolo che ha partecipato alla sessione.

Le componenti principali dei due sistemi sono brevemente presentate nella seguente figura e verranno analizzate più specificatamente nei paragrafi successivi.

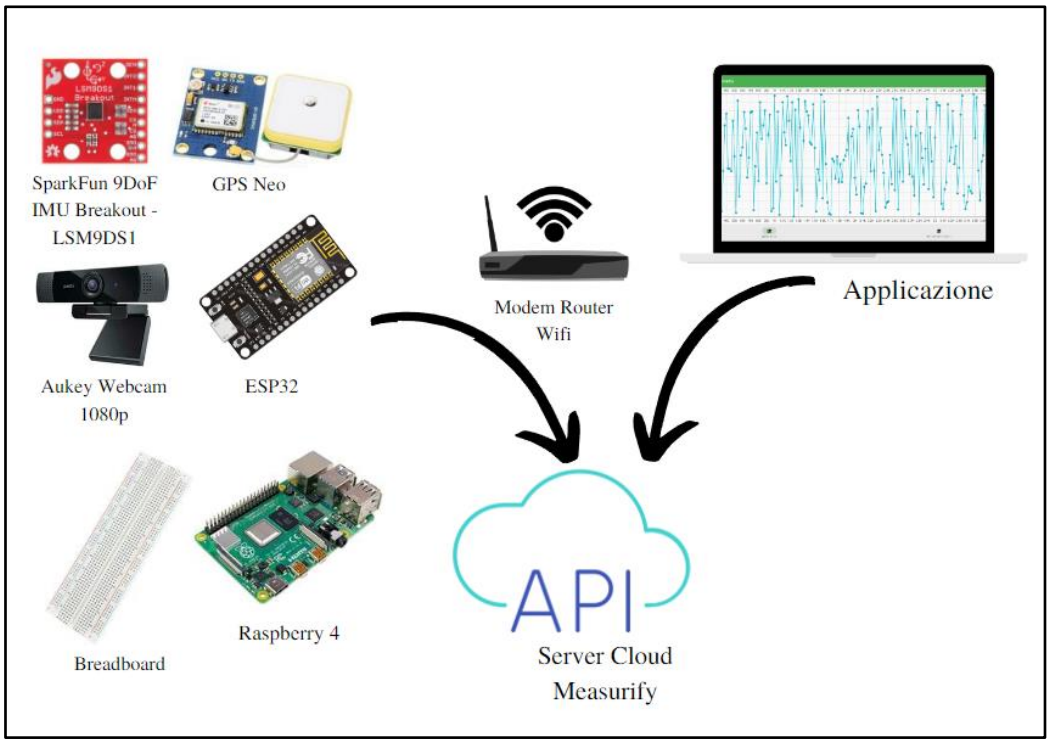

Fig. 1.2 -componenti utilizzati-

La parte hardware del sistema 1 è formata da un Raspberry 4 a cui è collegata la Webcam che si occupa delle rilevazioni di vetture in *real-time*. Il sistema embedded all'interno di ogni vettura è composto da tre moduli operanti in aggiunta ad una breadboard e fili di collegamento. I moduli sono un sensore IMU (SparkFun 9DoF IMU Breakout - LSM9DS1), un sensore GPS-Neo 6MV2 e un microcontrollore ESP32.
È imprescindibile che entrambi i sistemi abbiano la connessione Internet effettuabile tramite Modem Router Wi-Fi per il corretto funzionamento del progetto.
Il server cloud utilizzato è Measurify [1] con il quale i due sistemi interagiscono condividendo i dati raccolti.
Infine, è stata creata un'applicazione *cross-platform* connessa a Measurify da cui è possibile visualizzare le informazioni raccolte in modo semplice e *user-friendly*.

La tesi è articolata in tre paragrafi:

- nel primo vengono elencati e discussi i mezzi per la realizzazione del progetto
- nel secondo si passa alla valutazione dei risultati ottenuti dopo aver testato il sistema
- nel terzo ed ultimo sono esposte alcune considerazioni finali e commenti

2. Metodi e strumenti utilizzati

Le componenti principali del progetto sono:

1. una videocamera (Aukey Webcam 1080p)
2. un RaspBerry Pi 4
3. un microcontrollore (ESP32)
4. un sensore IMU (SparkFun 9DoF IMU Breakout - LSM9DS1)
5. un sensore GPS-Neo 6MV2

Vediamo in dettaglio le singole parti, partendo dalle componenti che costituiscono il sistema 1.

2.1 Sistema di tracciamento vetture e acquisizione tempo sul giro

Per poter realizzare un software in grado di riconoscere e tracciare vetture in *real-time* la scelta sul tipo di elaboratore è ricaduta su un Raspberry 4 che ben si adatta alle esigenze; infatti ha una potenza di calcolo notevole e include un modulo Wi-Fi che permette una più veloce programmazione di applicazioni basate sull'IoT.
Per l'acquisizione del video da analizzare è stata utilizzata una videocamera Aukey con una risoluzione di 1080p, connessa al Raspberry tramite porta USB.

Per il corretto funzionamento del sistema si devono impiegare alcune librerie: OpenCV [2], Request [3], Math [4] e Time [5].
La prima è una libreria *open-source* che comprende algoritmi ottimizzati di ogni genere, tra cui anche algoritmi di deep learning per il riconoscimento di vetture; la seconda consente di gestire agevolmente l'interazione con Measurify [1]; le ultime due servono rispettivamente per effettuare operazioni all'interno del software e per tenere traccia del tempo trascorso dall'avvio del sistema.

All'accensione del sistema, oltre ad inizializzare contatori, array e oggetti utili per l'elaborazione delle immagini e per il tracciamento delle vetture, grazie alla libreria OpenCV viene istanziata una variabile adibita all'analisi del video in entrata proveniente dalla telecamera esterna. Viene inoltre importato il dataset delle macchine da un file .xml che sarà utile per il riconoscimento delle vetture. L'ultima routine della fase di setup è infine *login*: la funzione si occupa di connettersi al server Measurify tramite processo di autenticazione e ottenere un *token*. Quest'ultimo dovrà sempre essere inserito all'interno di un *header* per ogni chiamata alle API.
In generale la sicurezza è un aspetto critico all'interno dell'IoT: l'autenticazione basata su *token* consente di identificare ed eventualmente accettare ciascuna richiesta mandata dal client. Ogni API dovrebbe generare questo "gettone" in maniera pseudo-casuale e forte a livello criptografico. L'impiego di questo tipo di autorizzazione all'interno del seguente elaborato soddisfa i requisiti minimi di sicurezza.

```
login_url = 'https://students.measurify.org/v1/login'
measurement_url = 'https://students.measurify.org/v1/measurements'


def login():

    response = requests.post(login_url, data= {
        'username' : "car-manager-user-username",
        'password' : "car-manager-user-password",
        'tenant' : "car-manager-tenant"
    })
    response = response.json()
    return response["token"]
```

Fig. 2.1 -funzione *login*-

In seguito alla fase di setup del software, il sistema entra in un ciclo *while* infinito che può essere interrotto solo dalla pressione di un determinato carattere della tastiera, nel nostro caso il tasto *Esc*. All'interno della routine viene analizzato il video in entrata *frame-by-frame* per ciascuno dei quali il sistema cerca di individuare le vetture presenti tramite la funzione *detectMultiScale*; la funzione prende in ingresso quattro parametri: l'immagine da analizzare, nel nostro caso un frame del video, un fattore di scala che serve all'algoritmo per sapere quanto grandi siano le vetture nel frame rispetto a quelle del dataset, un numero che influenza la qualità degli oggetti rilevati, e infine una misura minima sotto alla quale gli oggetti vengono ignorati. Per ogni vettura rilevata viene disegnato un rettangolo intorno ad essa e aggiunto il punto centrale della box in un array.

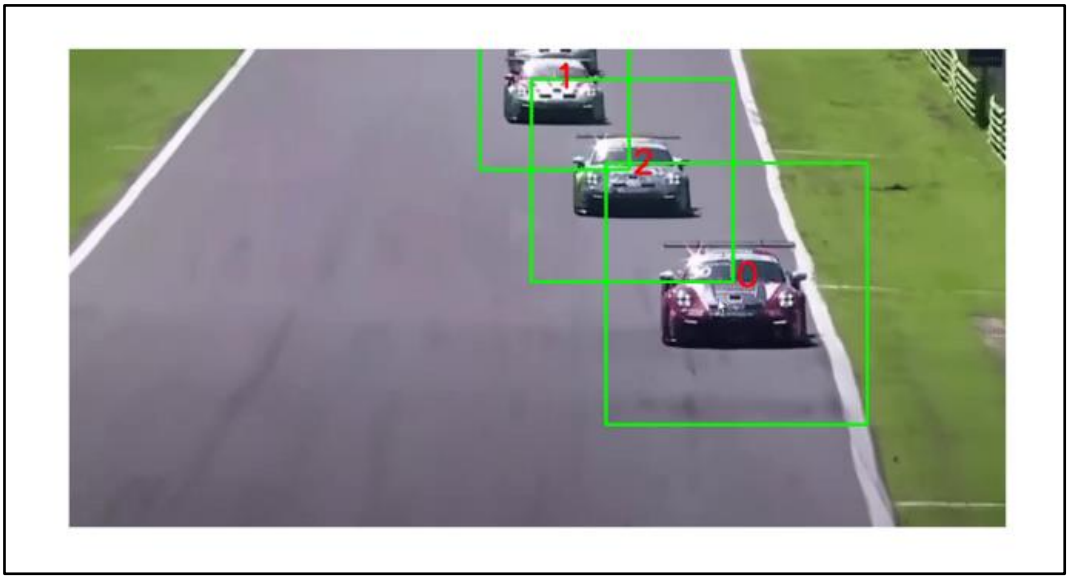

Fig. 2.2 -esempio di rilevazione delle vetture-

Dal momento che viene analizzato un frame alla volta l'algoritmo per il rilevamento delle macchine non è in grado di tenere traccia della storia delle analisi dei frame precedenti, per cui è stato aggiunto un controllo tra frame successivi basato sullo spostamento di ogni box: in questo modo, quando una vettura viene riconosciuta all'interno di un frame specifico, in quello successivo si sarà spostata di poco. Pertanto anziché essere valutata come nuova macchina rilevata il sistema gli attribuisce lo stesso identificativo del frame precedente.

Dopo aver analizzato il frame e aver rilevato le macchine presenti il software è in grado di eseguire la parte adibita all'acquisizione dei tempi di percorrenza delle vetture. Per ognuna di esse viene controllato se ha superato la linea del traguardo e, in caso affermativo, si procede con le funzioni *getTime*, *detect* e *send_measurify*.

La funzione *getTime* si occupa di restituire il numero di millisecondi trascorsi dall'accensione del programma. Nei sistemi il tempo viene rappresentato in secondi rispetto alla mezzanotte (UTC) del 1° gennaio 1970, pertanto per ottenere la misurazione necessaria si calcola la differenza tra la misura ottenuta quando la vettura sorpassa la linea del traguardo e quella registrata all'accensione, convertita poi in millisecondi.

La funzione *detect* ha il compito di comprendere quale vettura è passata sul traguardo: per facilità il progetto prevede automobili che si distinguano l'una dall'altra attraverso il colore della livrea. In questo modo, per riconoscere la vettura, la funzione analizza un insieme di pixel intorno al punto centrale della box corrispondente e individua il colore prevalente.

I dati ora possono essere inviati all'API per notificare il passaggio di una vettura tramite la funzione *send_measurify*: la routine prende in ingresso il valore in millisecondi ottenuto in precedenza e una stringa contenente l'identificativo della macchina passata rilevato con la funzione *detect* ed invia una richiesta POST alla rotta indicata. In aggiunta alla rotta di Measurify, vengono specificati alcuni *header*, tra cui l'autorizzazione tramite *token*, e il corpo della richiesta contenente la misurazione da registrare.

L’analisi del frame corrente è terminata, pertanto il software procede all’analisi del frame successivo, a meno che non sia stato premuto il carattere di terminazione che determina la conclusione del programma.

Riassumendo

INIT:

- Inizializzazione variabili.
- Preparazione e setting dei parametri della videocamera.
- Richiesta e acquisizione del token dal server API Measurify.

LOOP:

- Analisi del frame corrente e rilevamento vetture all’interno dell’immagine.
- Confronto delle vetture rilevate con il frame precedente per il tracciamento continuo.
- Verifica del passaggio sul traguardo del veicolo:
  1. **Esito positivo:**
     - **i.** Funzioni *getTime*, *detect* e *send_measurify* per l’acquisizione e l’invio della misurazione a Measurify.
     - **ii.** Verifico risposta positiva dal server.
     - **iii.** Ritorno al loop iniziale.
  2. **Esito negativo:**
     - **i.** Nulla di rilevante da eseguire, ritorno al loop iniziale.
- Verifica del carattere di terminazione:
  1. **Esito positivo:**
     Interruzione del loop e conseguente uscita dal programma
  2. **Esito negativo:**
     Iterazione del loop

2.2 Sistema acquisizione velocità e accelerazione

Analizziamo le singole parti del secondo sistema partendo dal microcontrollore, l’ESP32, che si adatta bene alle nostre esigenze computazionali. La scheda infatti, oltre ad avere le caratteristiche tipiche di un Arduino, include un modulo Wi-Fi che permette l’invio delle misurazioni senza l’utilizzo di moduli esterni.
Per programmare la scheda Arduino fornisce un ambiente di sviluppo libero comune a tutte le board; essendo dunque il software comune a tutte le schede è necessario installare il pacchetto specifico per il nostro Arduino.

Il sensore responsabile alla misurazione dell’accelerazione, l’IMU LSM9DS1, ospita un accelerometro a 3 assi, un giroscopio a 3 assi e un magnetometro a 3 assi per un totale di 9 gradi di libertà su un unico circuito integrato; le loro unità di misura sono espresse rispettivamente in m/s2, DPS (degrees per second, °/s) e gauss (Gs).
All’interno di questo progetto il sensore viene impiegato per percepire solo l’accelerazione e lungo un solo asse, avendo l’accorgimento di posizionare il sensore nella maniera corretta all’interno della vettura. L’LSM9DS1 supporta sia SPI sia I2C: nel progetto si è optato per l’utilizzo dell’I2C, che richiede un numero minore di cavi per l’interconnessione rispetto a SPI.

Il sensore deputato alla misurazione della velocità è il NEO-6MV2 GPS Module. Il sensore fa parte della famiglia NEO-6M che vanta un motore u-blox6 a 50 canali che è in grado di effettuare massicce ricerche spaziali. Il design e la tecnologia innovativi eliminano le fonti di disturbo e mitigano gli effetti *multipath*, dando ai ricevitori GPS NEO-6 eccellenti prestazioni di navigazione anche negli ambienti più difficili. All’interno del progetto il sensore viene impiegato per estrarre la velocità del GPS e, di conseguenza, della vettura.
Il modulo supporta solo la connessione di tipo seriale tramite l’utilizzo di due fili: Rx e Tx. È importante ricordare che i due fili vanno connessi ai rispettivi pin opposti del microcontrollore.

Sia il microcontrollore ESP32 che i due sensori lavorano alla tensione di 3.3V: questo dettaglio è molto rilevante perché permette la comunicazione senza l'utilizzo di *level-shifter* che complicherebbero la realizzazione circuitale del sistema.

Di seguito è illustrato un esempio di collegamento del sistema in esame seguendo i consigli dei produttori dei chip Sparkfun [6] e Ublox [7].

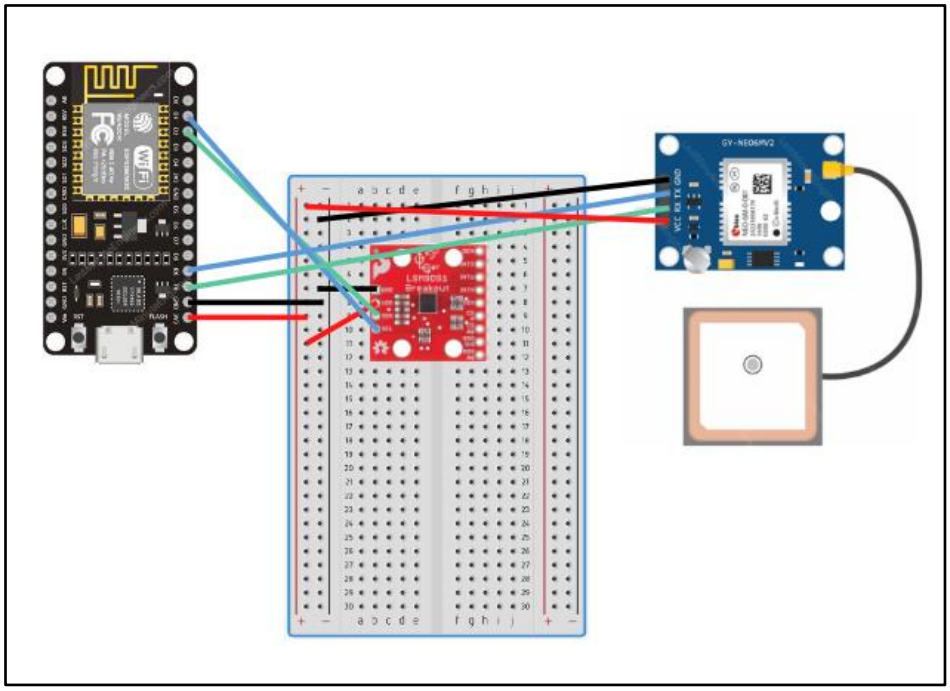

Fig. 2.3 -esempio di connessione del sistema-

Dopo aver approfondito la configurazione hardware, passiamo a quella software. Per interfacciarsi con i due sensori è necessario scaricare le librerie Arduino fornite dai produttori dei chip; oltre ad esse si deve impiegare la libreria EdgeEngine che permette l'interazione con le API di Measurify.

All'accensione del programma il microcontrollore tenta di connettersi ad una rete Wi-Fi, i cui parametri SSID e password sono definiti nel file *configuration.h*. Se la connessione non va a buon fine verrà notificato sul monitor seriale e il dispositivo ripeterà la routine.

Successivamente vengono settati tutti i parametri utili per inizializzare la connessione con EdgeEngine: in questo modo, con la funzione *Edge->init*, il software si collega alle API di Measurify per avere la propria descrizione, che include in particolare l'elenco degli script da eseguire e i valori dei parametri per configurarsi. Invocando questa funzione la libreria si occupa inoltre di effettuare il login al server ed ottenere il *token*.

Le ultime operazioni presenti nel setup sono le funzioni adibite all'inizializzazione dei sensori del sistema: per l'IMU abbiamo *setupSensor* che permette di variare il range dell'accelerometro; per il modulo GPS la funzione *begin* permette di stabilire una prima connessione seriale con il sensore.

Come è noto per qualsiasi Arduino, dopo il setup segue la funzione loop, facilmente intuibile dal nome della funzione stessa e permette di reiterare il codice scritto al suo interno. Nel nostro caso, per ogni ciclo della routine il sistema rileva un valore per la velocità e uno per l'accelerazione interagendo con i sensori, prepara un array contenente i valori registrati che la libreria EdgeEngine utilizzerà per inviare al server Measurify le misurazioni ottenute.

L’array costruito dalla libreria EdgeEngine è composto da due valori: il primo è il valore della velocità, mentre il secondo dell’accelerazione misurata sull’asse x. Quando viene invocata la funzione *evaluate*, Edge si occupa di predisporre la richiesta HTTP nella maniera corretta e inviare i dati a Measurify. Una volta eseguita questa operazione, vengono cancellati i dati pronti per ospitare le misurazioni del ciclo successivo.

Riassumendo

INIT:

- Inclusione librerie necessarie.
- Inizializzazione dei sensori.
- Connessione alla rete Internet.
- Inizializzazione EdgeEngine, tra cui la connessione a Measurify e l’acquisizione del token.

LOOP:

- Acquisizione dei valori di velocità e di accelerazione.
- Creazione della misurazione da inviare a Measurify grazie alla libreria EdgeEngine.
- Invio dei dati.

2.3 Applicazione

Prendiamo ora in considerazione l’applicazione Flutter dedicata, scritta in linguaggio Dart [8].

Dart è un linguaggio di programmazione sviluppato da Google, orientato agli oggetti per il web e completamente *open source*. Lo scopo ultimo di Dart è quello di favorire il lavoro dello sviluppatore nella costruzione di moderne applicazioni *cross-platform* con Flutter. Per fare ciò, Dart offre un linguaggio moderno e generico, una Virtual Machine performante adibita alla compilazione del codice, una base di librerie e repository di gestione dei pacchetti per aggiungere le funzionalità. Viene utilizzato specialmente per sviluppare applicazioni *client side*.

All’avvio dell’applicazione, la prima routine, eseguita in *background*, è *login*: anche in questo caso è importante eseguire una chiamata al server Measurify per ottenere il *token*, in modo da inserirlo per le future richieste effettuate dal client. Tutte le richieste HTTP all’interno dell’applicazione sono svolte in maniera asincrona, in modo da non bloccare l’esecuzione del programma.

Il client può scegliere se visualizzare il resoconto della sessione di qualifica oppure consultare la telemetria di ogni vettura che ha partecipato alla sessione: l’applicazione prevede la navigazione tra la HomePage in cui è presente un bottone per i risultati, e la pagina dedicata alle auto e i relativi dati, accessibili attraverso la pressione del bottone corrispondente.

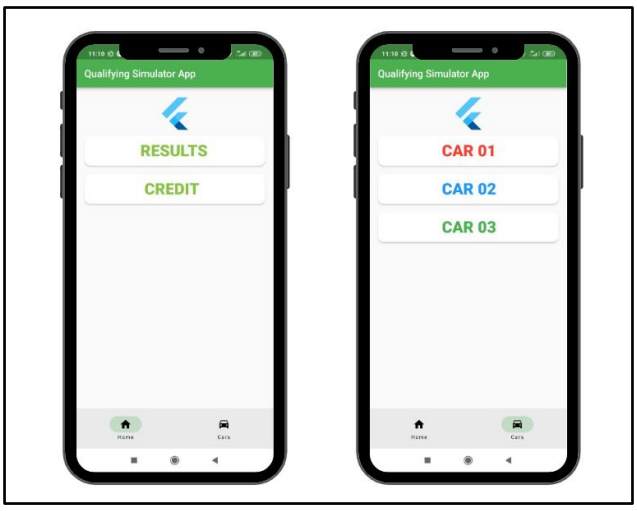

Fig. 2.4 -schermata principale dell’applicazione-

Quando il client desidera visualizzare i risultati, alla pressione del bottone viene esposta una pagina di caricamento, in cui avviene la richiesta di ottenere le informazioni utili dall'API di Measurify e, una volta ottenute, l'elaborazione dei dati per ottenere la classifica finale dettata dai tempi sul giro. Le misurazioni vengono ottenute tramite una GET alla seguente rotta:

'https:\\students.measurify.org/v1/measurements?filter={"feature": "check-point",
"thing":"Car 01"}'
(viene effettuata una richiesta per ogni "thing")

Dal momento che le misurazioni indicano il tempo trascorso tra un passaggio sul punto di interesse e l'avvio del sistema, per ottenere i tempi sul giro occorre calcolare la differenza tra due misurazioni contigue. Questa elaborazione viene eseguita dalla funzione *manageData*, che crea oggetti di tipo *Lap* con tutte le informazioni necessarie e, per ogni vettura, un oggetto di tipo *CarSession*, costituita dall'identificativo della macchina e una lista di *Lap* compiuti, ordinati dal più veloce al più lento. L'ultima funzione, *classify*, si occupa semplicemente di ordinare le *CarSession* precedentemente ottenute in base al giro più veloce tra tutte le macchine.

Una volta finita l'elaborazione dei dati, la pagina di caricamento lascia spazio alla pagina adibita alla visualizzazione dei risultati, in cui è presente una tabella esplicativa: le intestazioni della tabella sono rispettivamente *Pos* per la posizione, *Car* per l'identificativo della vettura, *Time* per il tempo sul giro, *Lap* per il giro in cui è stato conseguito quel tempo ed infine *Gap* per indicare il distacco cronometrico dal giro più veloce. Il tempo sul giro, prima di essere visualizzato, viene convertito nel formato *minuti:secondi.millisecondi*.

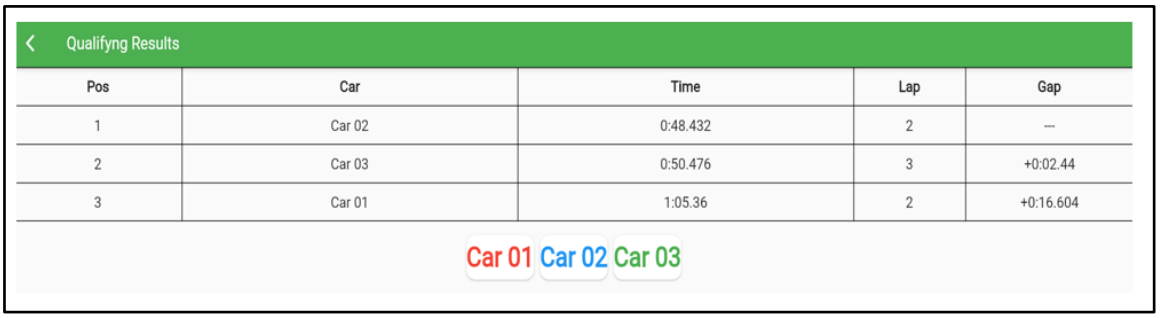

Fig. 2.5 -tabella dei tempi-

È possibile da questa schermata analizzare i tempi sul giro di una singola vettura attraverso la pressione del rispettivo bottone. La tabella risultante è molto simile alla precedente, ma vengono indicati solo il tempo, il numero del giro corrispondente e il divario da quello migliore.

Se il client desidera invece consultare la telemetria di una vettura, alla pressione del bottone viene anche questa volta esposta una pagina di caricamento, in cui cambia il target della richiesta all'API e l'elaborazione dei dati ottenuti. Le misurazioni per visualizzare i dati relativi a velocità e accelerazione si ottengono con una GET alla seguente rotta:

'https:\\students.measurify.org/v1/measurements?filter={"feature": "state",
"thing":"Car 0' +**target.toString()** + "'}'
(**target.toString()** serve per non duplicare il codice, cambia in base al bottone premuto)

Una volta ottenuti i dati, la funzione *manageData* si occupa di creare per ogni misurazione un oggetto di tipo *SpeedSample* e uno di tipo *AccSample*, costituiti da una variabile *milliseconds* e, rispettivamente, una variabile *speed* e una *acceleration*.

Finita l'elaborazione, viene caricata una pagina in cui sono disponibili i grafici della velocità e dell'accelerazione della vettura selezionata in precedenza. Grazie alla libreria di Widget Fl_Chart, ogni campione viene mappato sul grafico con il valore di *milliseconds* in ascissa e, il valore di *speed* o *acceleration* (a seconda del grafico selezionato) in ordinata.

In alto a destra è possibile selezionare il giro di cui si vuole visualizzare il grafico tramite un menù a tendina.

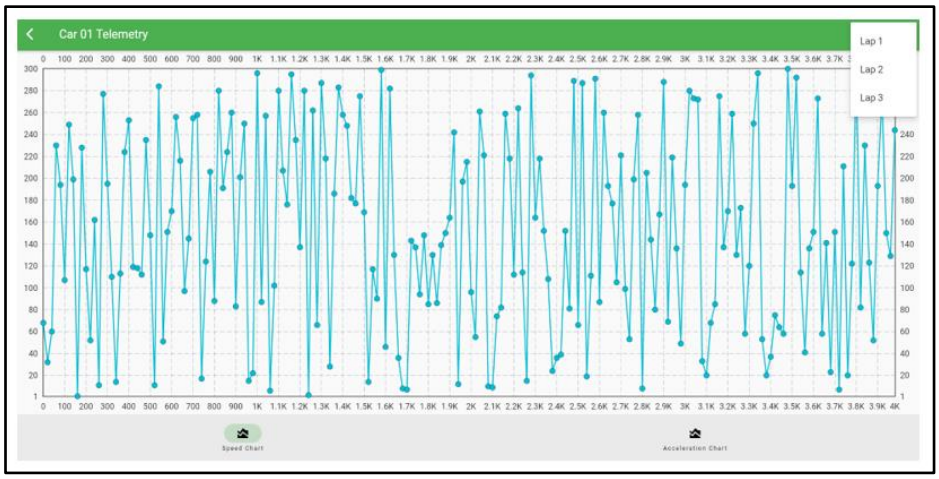

Fig. 2.6 -grafico simulato velocità-

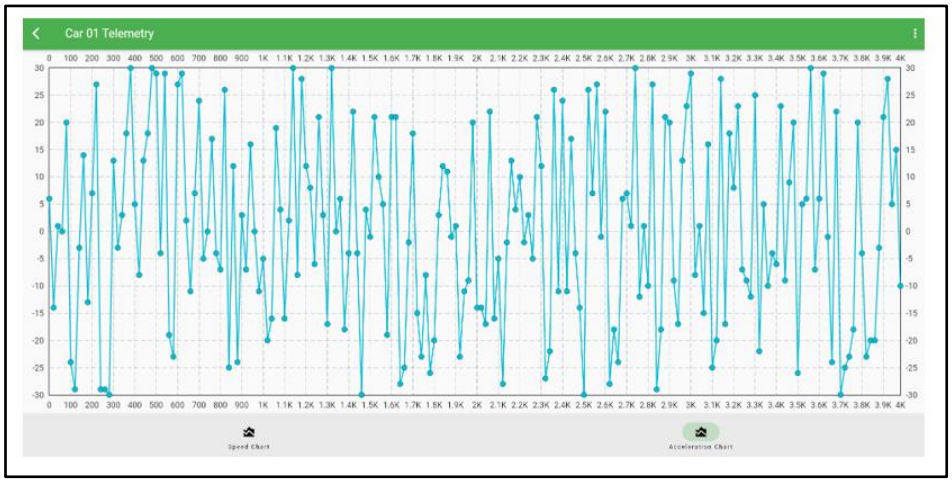

Fig. 2.7 -grafico simulato accelerazione-

3. Sperimentazione e risultati

Al termine dell'analisi hardware e software del progetto, i sistemi sono stati testati.

Per testare il sistema di tracciamento dei veicoli, per prima cosa si è trascurata la sezione dedicata all'invio delle misurazioni a Measurify per concentrarsi sull'effettivo funzionamento dell'algoritmo di rilevazione delle vetture. La libreria OpenCV offre la possibilità di analizzare non solo il video in uscita di una telecamera, ma anche video formato mp4 già registrati; pertanto il primo test è stato fatto analizzando un video di un frammento di gara automobilistica scaricato dalla rete.
Nell'immagine che segue si può notare il corretto funzionamento del sistema tra due frame successivi del video: nel primo vengono riconosciute le vetture più prossime al punto di ripresa; nel secondo frame si nota come il software sia in grado di mantenere il tracciamento delle vetture rilevate in precedenza (lo si comprende tramite l'identificativo posto all'interno della box che non cambia).

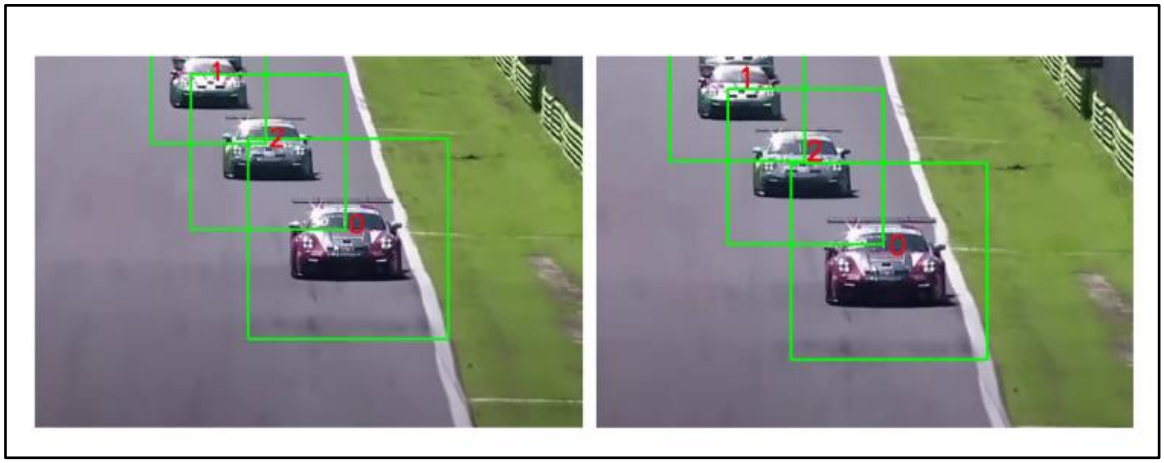

Fig. 3.1 -analisi video tra frame successivi-

Un'altra routine da testare è sicuramente la funzione *detect* che riconosce il colore della vettura in esame. Quest'ultima prende in ingresso tre parametri: naturalmente il frame da analizzare, il punto centrale della box corrispondente alla vettura ed infine un valore (*dist*) che determina quanti pixel analizzare. La routine consiste nel creare una matrice quadrata bidimensionale di dimensione *dist* contenente i pixel intorno al punto centrale della vettura e analizzarne il colore punto per punto. È importante sottolineare che, in primo luogo, viene convertito lo standard di rappresentazione dei colori nell'immagine da RGB (*Red Green Blue*) a HSV (*Hue Saturation Brightness*) per facilitare l'analisi del pixel e l'estrazione del colore.
Un primo test è stato eseguito con l'immagine precedente analizzando la vettura con l'identificativo 0: a tutti gli effetti intorno al punto centrale, in prossimità del cofano, l'auto è scura e l'algoritmo riconosce su 64 pixel (una matrice 8x8) 48 di colore blu e 16 di colore rosso (dovuti probabilmente al numero 0 rappresentato nella figura).

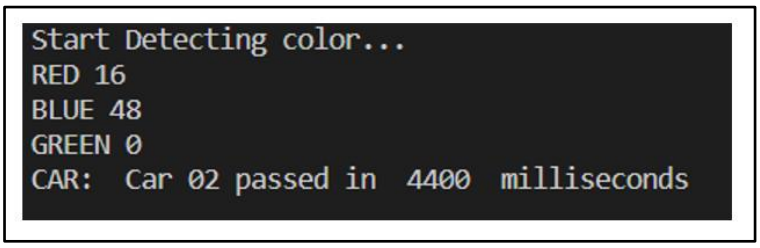

Fig. 3.2 -risultati funzione *detect*-

Dopo aver verificato il corretto funzionamento del sistema, la sperimentazione è continuata includendo le funzioni legate all'invio dei dati a Measurify; prima di fare ciò è stato ritenuto necessario testare le rotte per interagire con le API.
Come spiegato in precedenza, per poter inviare misurazioni a Measurify è necessario essere in possesso del token, ottenibile eseguendo una POST alla rotta *https:\\students.measurify.org/v1/login* e specificando all'interno del body le credenziali per l'accesso.

Senza utilizzare il sistema costruito, esiste un programma che consente di inviare richieste ad una API in modo veloce ed efficiente: Postman [9].
Nell'immagine che segue è visibile l'interfaccia del programma al momento della POST: il token ricevuto in risposta è coerente con quando annunciato nel capitolo precedente, ovvero è un codice generato in maniera pseudo-casuale e con un tempo di utilizzo limitato. Postman fornisce inoltre una risposta molto dettagliata rispetto a quella di un semplice browser, per esempio lo *status code* o il tempo impiegato dal momento della richiesta.

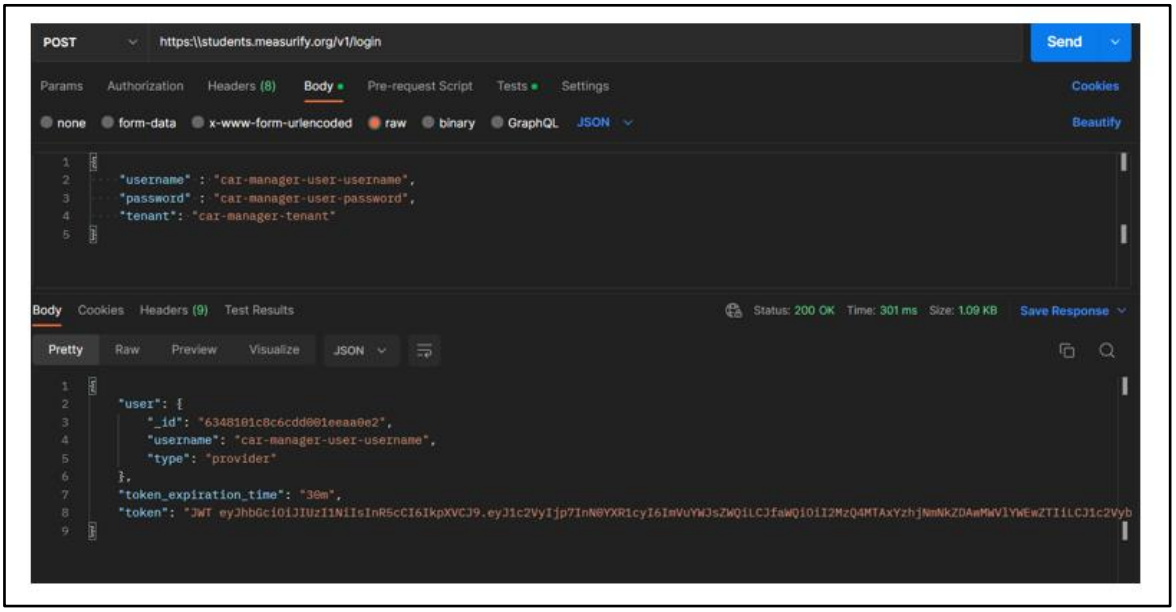

Fig. 3.3 -richiesta e risposta per l'ottenimento del *token*-

Tornando al nostro sistema, invocando la funzione *login* siamo sicuri che fornirà in uscita un codice simile al precedente; per averne la certezza visualizziamo sul monitor seriale, grazie alle *print* inserite nel codice, lo status code della richiesta e la stringa contenente il *token* (immagine successiva).

Fig. 3.4 -visualizzazione dello *status-code* e del *token* sul monitor seriale-

L'ultima funzione del sistema da testare è *send_measurify*: deve essere in grado di eseguire una POST alla rotta *https:\\students.measurify.org/v1/measurements*, con all'interno del body la misurazione scritta in JSON e l'utilizzo del *token* negli *headers* della richiesta. Si può notare dall'immagine successiva come la chiamata vada a buon fine, creando la nuova misurazione per la vettura indicata.

Fig. 3.5 -visualizzazione della risposta ad una chiamata HTTP per inviare una misurazione-

Per quanto riguarda il sistema embedded per l'acquisizione di velocità e accelerazione, il software è stato redatto con il pattern *#define* e *#ifdef* per ogni componente, in modo da rendere il codice scalabile e soprattutto più facile da testare. Il file *configuration.h* inizia con la definizione di tutti i moduli (tramite la parola chiave *#define*) preceduti da un *#undef*. Semplicemente invertendo le due righe di codice, tutto il modulo corrispondente viene ignorato dal compilatore quando, all'interno del programma, incontrerà un'istruzione di *#ifdef* (seguita dal nome del modulo). Con questo pattern si può testare un sensore o una funzionalità alla volta. Come si nota nella figura sottostante, sono stati definiti il modulo LSM e EdgeEngine, ma non il modulo GPS. Nel programma principale, infatti, le righe di codice all'interno dell'*#ifdef* non verranno eseguite.

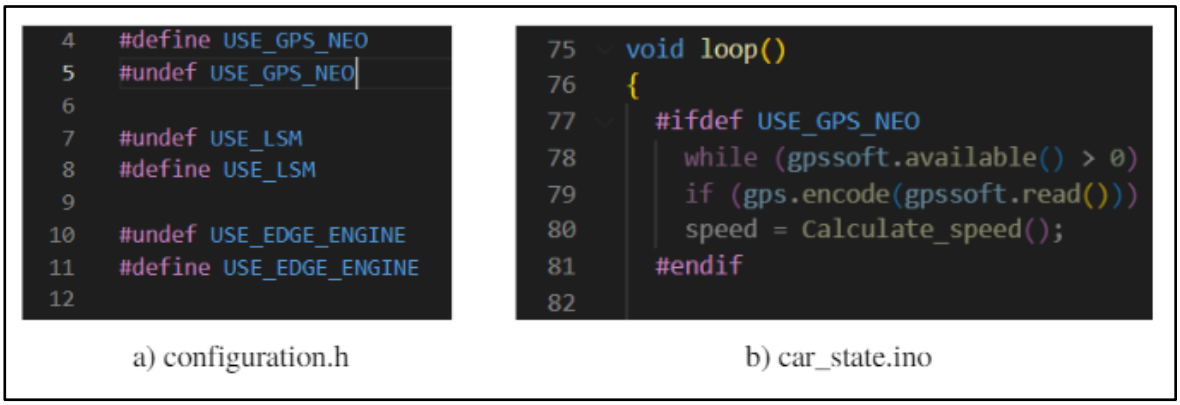

Fig. 3.6 -esempio pattern *#define #ifdef*-

Per prima cosa è stato provato il corretto funzionamento del sensore IMU: dopo aver collegato i fili nella maniera opportuna, con l'aiuto di diverse *print* sul monitor seriale si è potuto vedere quale dei tre assi venisse modificato da una spinta laterale, in modo da trasmettere una sola componente alle API.
Verificato che si trattasse proprio della componente sull'asse x come indicato sul sensore, lo step successivo è stato quello di testare il secondo sensore, il GPS 6M Module.

Per prima cosa, dal momento che il sensore viene venduto senza i connettori, è stato necessario saldare i componenti per permettere la connessione con il microcontrollore. Essendo un sensore GPS, l'obiettivo primario consiste nel cercare una connessione satellitare: per fare ciò è presente un led che notifica il successo.
Se quest'ultimo lampeggia ogni secondo vuol dire che è collegato ad un numero sufficiente di satelliti per permetterne il corretto funzionamento; se invece il led non lampeggia il sensore sta ancora cercando una connessione.
Al termine della prima routine il sensore viene collegato al microcontrollore e viene testato il codice corrispondente: dal momento che la comunicazione avviene in modo seriale è importante controllare che il canale dedicato funzioni e, tramite istruzioni *print*, verificare che il sensore notifichi le varie misure della velocità elaborate.

Una volta testati i sensori l'ultima parte da verificare è quella relativa alla connessione ad Internet e l'invio dei dati a Measurify; terminato questo test il sistema embedded è pronto e può essere inserito all'interno di una vettura.

Il processo per connettere il microcontrollore ad Internet richiede solo di specificare l'SSD e la password della rete desiderata (definiti nel file *configuration.h*) e, grazie all'utilizzo della libreria EdgeEngine, l'interazione con le API di Measurify per l'invio dei dati è piuttosto semplice.
Come si può osservare dalla seguente immagine, la libreria notifica il corretto funzionamento scrivendo sul monitor seriale il corpo della risposta ottenuta. Inoltre, facendo una GET per ottenere tutte le misurazioni tramite Postman, è possibile avere un'ulteriore prova.

Fig 3.7 -visualizzazione della misurazione e il conseguente invio a Mesurify-

Fig. 3.8 -visualizzazione delle misurazioni su Postman-

Sebbene Postman sia un software molto utile per la visualizzazione dei dati raccolti non è sicuramente di facile interpretazione al momento della lettura dei risultati ottenuti: per questo motivo è stata sviluppata un'applicazione in grado di garantire una facile comprensione all'utente finale.

Per testare il corretto funzionamento dell'applicazione è stato creato uno *script* di test che si occupa di creare tutte le misurazioni con valori casuali per i campioni di velocità e accelerazione e valori per il tempo sul giro coerenti con quelli che elaborerebbe il sistema di tracciamento.
Sono stati creati per ogni vettura 3 giri di percorrenza, ognuno dei quali possiede 200 coppie di campioni velocità-accelerazione. Una volta inviate tutte le misurazioni a Measurify, tramite l'applicazione si è potuto vedere il corretto modellamento dei dati.

L'aspetto che ha richiesto la maggiore attenzione è stata la suddivisione dei campioni di velocità e accelerazione tra giri di percorrenza diversi: per fare ciò gli oggetti di tipo *Lap* prevedono una data di inizio e una di fine estratti dai campi *startDate* e *endDate* predisposti da Measurify, mentre gli oggetti di tipo *SpeedSample* o *AccSample* hanno al loro interno la data in cui sono stati misurati. In questo modo, quando l'utente finale vuole visualizzare la telemetria di un determinato giro, la lista di campioni viene filtrata selezionando solo quelli che hanno una data compresa tra la data iniziale e finale del giro selezionato.

Avendo verificato il corretto funzionamento dei sistemi e dell'applicazione, seppur con dati simulati, l'ultimo test completo è stato effettuato utilizzando una macchina radio-comandata equipaggiata del sistema embedded per l'acquisizione della telemetria e posizionando il sistema di tracciamento in un punto strategico. Al termine della sessione automobilistica in cui si è fatta girare la vettura si è potuto apprezzare l'efficacia dell'intero sistema sfruttando l'applicazione dedicata.

Fig. 3.9 -sistema di acquisizione della telemetria-

4. Contributo personale e considerazioni conclusive

Dopo aver esposto dettagliatamente quali componenti sono stati utilizzati, come sono stati impiegati e che risultati abbiamo ottenuto, è importante descrivere come è stato affrontato il problema e analizzare le scelte fatte durante il progetto.

Come già esposto in precedenza, un sistema per la gestione di qualunque sessione di gare automobilistiche utilizza componenti con specifiche tecniche rigorose ed è regolato da algoritmi complessi; pertanto il presente elaborato ha come scopo trovare una soluzione più accessibile e con tecniche realizzative diverse.

Il sistema di rilevazione delle automobili contiene un algoritmo molto prestazionale della libreria OpenCV, a cui è stata aggiunta un'elaborazione del video tra frame successivi ideata da PySource [10]. La combinazione delle due routine è risultata la migliore soluzione per svolgere il compito desiderato, scartando altre idee come per esempio l'utilizzo di una fotocellula per rilevare il movimento e di una fotocamera per scattare immagini subito dopo il passaggio di una vettura, risultata non appropriata per via dei tempi di latenza ed altri parametri descritti nei datasheet della fotocellula.

Un altro problema sorto durante la realizzazione del sistema è stato il metodo di riconoscimento delle vetture: il modo più intuitivo è stato quello di utilizzare un'ulteriore elaborazione dell'immagine per estrarre il colore del veicolo; anche in questo caso si è scelto di utilizzare una procedura di PySource, modificata per utilizzarla su una matrice di pixel invece che su uno solo.

L'intero modulo è stato pensato per essere duplicato ed avere lo stesso sistema in più punti del circuito: in questo modo ci si avvicina di più alla situazione reale di una sessione di qualifica, in cui il tracciato è diviso in più settori e il tempo sul giro è la somma dei tempi di ogni settore.

Fig. 4.1 -grafica della F1 in cui il giro è diviso in 3 settori-

Un aspetto molto importante che non è stato approfondito all'interno del presente elaborato è la gestione dell'energia del sistema embedded: come enunciato in precedenza tutti i componenti del modulo da inserire all'interno della vettura sono alimentati a 3.3V; durante la fase di test è stato collegato ad un powerbank che eroga energia sufficiente per il corretto funzionamento, ma questo non esclude altre possibili idee, come per esempio una pila o batteria anche a voltaggio più alto, dal momento che integrato al microcontrollore ESP32 è presente un regolatore di tensione che permette la stabilità di erogazione a 3.3V.

Per il sistema da posizionare all'interno del circuito abbiamo un Raspberry Pi 4 con una sola periferica connessa, la videocamera: un modo sufficiente e anche ecosostenibile potrebbe essere l'utilizzo di un pannello solare per alimentare il sistema.

Questo progetto può rappresentare il primo passo verso qualcosa di più elaborato e completo aggiungendo funzionalità: per esempio, il sistema embedded all'interno dell'auto da corsa potrebbe raccogliere più informazioni con l'aggiunta di nuovi sensori per avere un quadro più completo della telemetria della vettura stessa. Le monoposto nel campionato mondiale di Formula 1 possiedono numerosissimi sensori in grado di raccogliere in un weekend di gara circa 100 Gigabyte di informazioni per avere una visione ampia sulla telemetria e sulla sicurezza.

Lo stesso sensore GPS usato per il progetto permette di avere informazioni aggiuntive molto rilevanti, per esempio la posizione esatta della vettura: si potrebbe utilizzare questa informazione per avere una mappa del circuito con le posizioni esatte delle auto in pista.

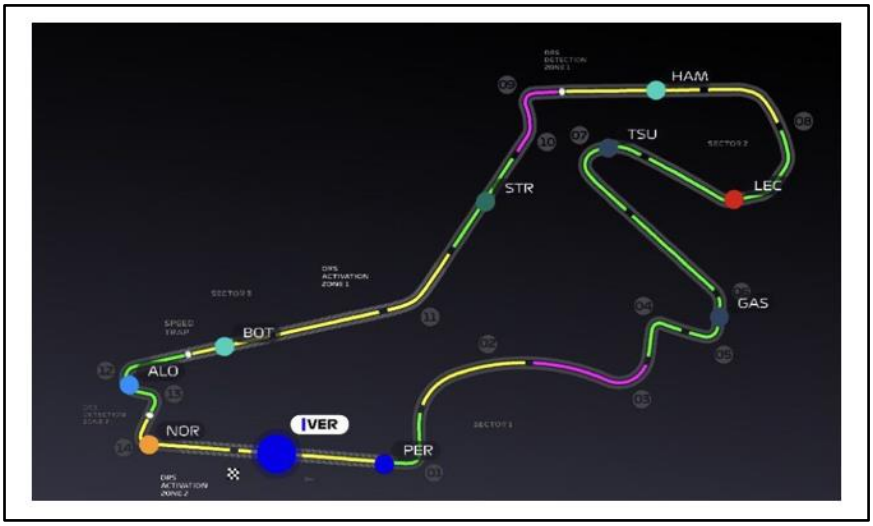

Fig. 4.2 -circuito F1 con la posizione dei piloti in pista-

Per concludere, questo elaborato mi ha permesso per la prima volta di confrontarmi con la realizzazione di un progetto partendo da zero: definita l'idea di base, si è pensato all'hardware necessario per la progettazione. Ottenute le informazioni fondamentali, è cominciato il lavoro di ricerca su Internet di progetti da cui prendere spunto e aggiungere le funzionalità per completare la realizzazione.

A conclusione di questo elaborato, desidero menzionare tutte le persone, senza le quali questo lavoro di tesi non esisterebbe.
Grazie al mio relatore Prof. Berta, sempre presente e disponibile. Grazie al percorso intrapreso insieme ho sviluppato maggiormente la mia capacità di analisi e di problem solving.
Un sentito grazie al Prof. Zunino, molto gentile e disponibile; grazie per avermi aiutato con la sua esperienza.
Ringrazio infinitamente i miei genitori e mio fratello che mi hanno sempre sostenuto, appoggiando ogni mia decisione, fin dalla scelta del mio percorso di studi.
Ringrazio il mio compagno di corso Cristiano, sia per l'aiuto svolto per il presente elaborato, sia per avermi accompagnato durante l'intero percorso universitario.
Grazie ai miei amici Luigi, Alessandro, Gabriele, Luca, Leonardo, Stefano e Alice per essere stati sempre presenti anche durante questa ultima fase del mio percorso di studi; grazie per aver ascoltato i miei sfoghi, grazie per tutti i momenti di spensieratezza.

5. Riferimenti bibliografici

https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
http://esp32.net/

[1] https://measurify.org/

[2] https://opencv.org/ https://github.com/opencv/opencv

[3] https://realpython.com/python-requests/

[4] https://docs.python.org/3/library/math.html

[5] https://docs.python.org/3/library/time.html

[6] https://learn.sparkfun.com/tutorials/lsm9ds1-breakout-hookup-guide?_ga=2.217884755.452313816.1563890620-213251003.1554896041

[7] https://components101.com/modules/neo-6mv2-gps-module

[8] https://dart.dev/

[9] https://www.postman.com/

[10] https://pysource.com/object-detection-opencv-deep-learning-video-course-full-program/