



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

Febbraio 2021

**Progettazione e sviluppo di un sistema embedded per il
monitoraggio di una smart-bike**

Candidato: Marco Rossi

Relatore: Prof. Riccardo Berta

Sommario

Lo scopo di questa tesi è quello di sviluppare un sistema embedded per il monitoraggio di una smart-bike. Attraverso svariati sensori collocati lungo la bicicletta, vengono rilevate velocità, altitudine, numero di compressioni, angolo di piegata ed altri parametri con lo scopo di fornire all'utente impegnato lungo una discesa di enduro la possibilità di valutare le sue prestazioni sportive. Le informazioni ottenute dai sensori vengono elaborate da un microcontrollore Arduino Nano 33 IoT dotato di modulo WiFi, anch'esso collocato sul mezzo, e successivamente inviate su un cloud. La comunicazione con quest'ultimo viene effettuata attraverso delle API rilasciate da un framework chiamato Measurify. Esso ha il ruolo di interfaccia tra il sistema di rilevamento e l'applicazione per smartphone. Quest'ultima, sviluppata con Flutter, permette all'utente di analizzare le informazioni ottenute lungo la discesa attraverso una rappresentazione grafica dei dati.

1. Introduzione

La necessità di semplificare la vita delle persone ha sempre stimolato l'ingegno umano verso un progresso tecnologico difficilmente un tempo immaginabile. Dalla clava al computer, dalla ruota ai satelliti, dalla scrittura all'Internet of Things, l'uomo si è sempre sforzato di agevolare la propria esistenza attraverso nuove tecniche e tecnologie che gli permettessero di semplificare ciò che era complesso e di facilitare ciò che era scomodo.

Uno dei più grandi balzi in avanti è stato fatto per mezzo di Internet, grazie al quale miliardi di dispositivi oggigiorno possono comunicare e interagire tra di loro, permettendoci di sfruttare servizi fondamentali di uso quotidiano (ospedali, banche, negozi, ecc...) con estrema facilità, rendendoli sempre più fruibili attraverso una tecnologia sempre più accessibile.

Ed è nell'uso quotidiano della tecnologia che ha preso sempre più spazio l'Internet of Things (IoT). L'approccio è quello di rendere "intelligenti" oggetti d'uso comune e non solo, permettendogli di comunicare con l'utente (attraverso Internet e applicazioni) informazioni utili, prestando così un servizio.

L'IoT si sviluppa in diversi settori, tra i più conosciuti la domotica, il monitoraggio di ambienti, la sanità, il wearable, il settore automobilistico. Attualmente nel mondo vediamo connessi circa 31 miliardi di dispositivi IoT, numero destinato a raddoppiare entro il 2024 secondo le stime del *Statista Research Department*.

Inserire negli oggetti elettronica quale sensori, microcontrollori e moduli per la connessione internet è diventato sempre più facile grazie alle dimensioni estremamente ridotte e al costo di produzione sempre più contenuto. Ciò che invece risulta complesso è l'elaborazione e la gestione dei dati che miliardi e miliardi di dispositivi riversano in rete e rendere questi in grado di comunicare tra loro per sfruttare al meglio i servizi che ognuno è in grado di offrire.

Ovviamente anche nel contesto sportivo l'IoT ha messo piede, permettendo di monitorare e confrontare le prestazioni degli atleti attraverso dispositivi direttamente indossati. Questo rende possibile ad esempio conoscere la forza impressa al pallone da un calciatore durante un tiro attraverso un sensore di accelerazione nel tacco della scarpa, misurare i parametri vitali di un maratoneta per analizzare la sua prestazione sportiva, studiare il gesto atletico di un saltatore con l'asta attraverso sensori inerziali collocati lungo il corpo.

Talvolta i mezzi attraverso i quali viene svolta l'attività motoria possono essere oggetto di interesse, come ad esempio le forze che agiscono sul casco di uno sciatore in caso di caduta, per le quali può automaticamente scattare la chiamata ai soccorsi, oppure basti pensare al mondo del ciclismo dove le prestazioni dell'atleta possono essere studiate attraverso un insieme di sensori collocati direttamente sul mezzo anziché sulla persona.

L'interesse di questo elaborato si sofferma proprio sull'ultimo esempio citato. Sfruttare la tecnologia IoT per monitorare le performance di un ciclista impegnato lungo una discesa montuosa accidentata per mezzo di una mountain bike. Lo scopo è quello di attingere dati quali:

- andamento della velocità nel tempo
- profilo altimetrico del tracciato
- tempo di percorrenza
- angolo massimo di piegata
- pendenza massima del tracciato
- numero di compressioni della forcella anteriore
- velocità massima raggiunta

e renderli fruibili, una volta conclusa la discesa, attraverso un'applicazione per smartphone. I valori verranno misurati da sensori collocati lungo la bicicletta e connessi ad un microcontrollore dotato di modulo wifi, il quale si occuperà di una prima elaborazione per poi inviarli ad un cloud dove verranno memorizzati. L'applicazione per smartphone preleverà i dati dal cloud e si occuperà di elaborarli e di visualizzarli in maniera schematica e grafica, così da renderli analizzabili dall'atleta.

Di seguito viene schematizzato l'intero sistema, composto da tre parti:

- **Bicicletta e hardware:** una bicicletta resa “intelligente” grazie alla presenza di sensori che permettono di effettuare misurazione di interesse e un microcontrollore dotato di modulo wifi per l’acquisizione, l’elaborazione e l’invio delle informazioni attinte
- **Measurify:** un API cloud in grado di immagazzinare, elaborare e fornire dati interfacciandosi direttamente con la bicicletta e l’applicazione per smartphone.
- **SmartBike:** un’applicazione Android per smartphone in grado di attingere dati grazie alle API di Measurify e renderli visualizzabili all’utente per mezzo di grafici e icone.

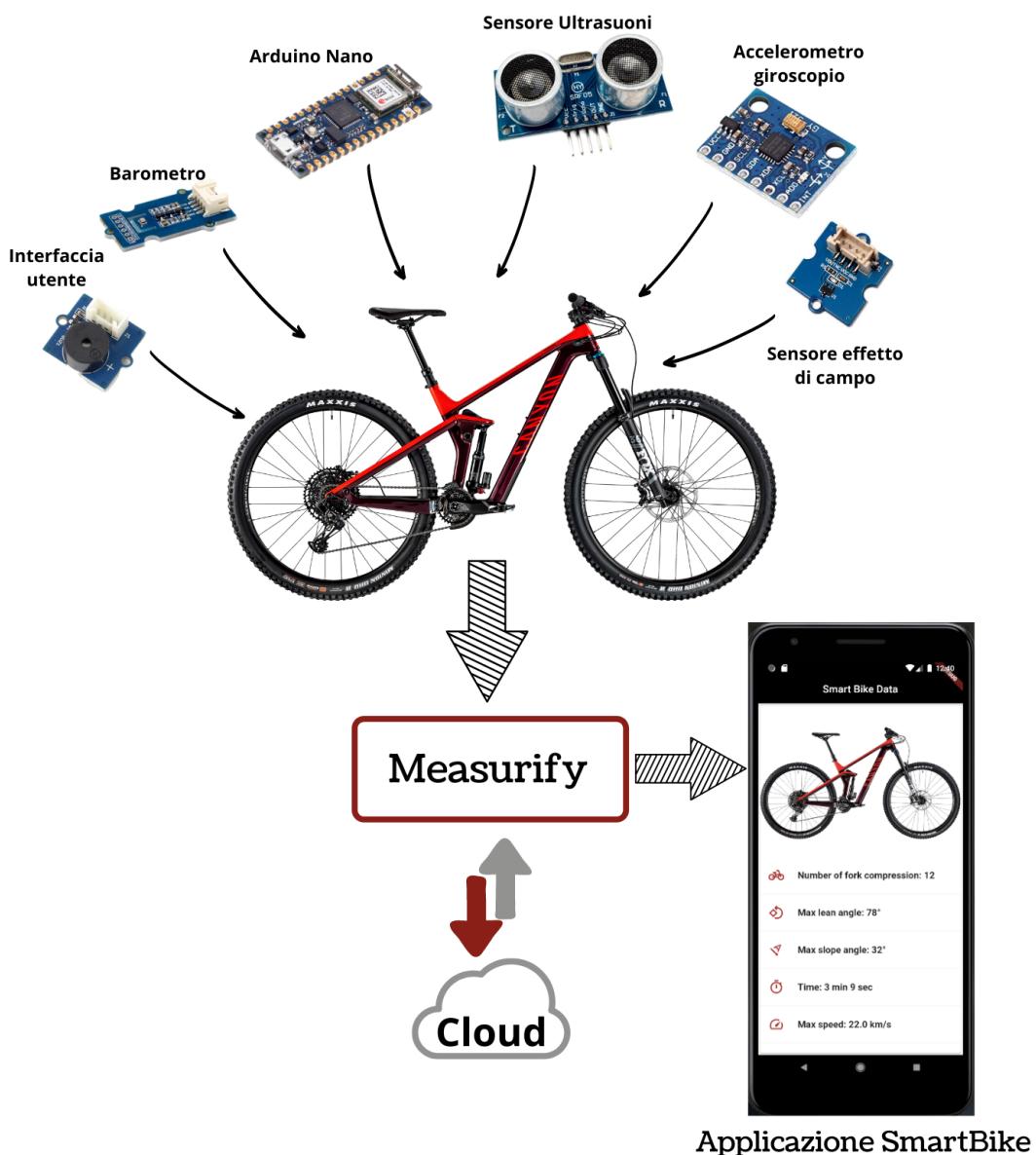


Figura 1: schema complessivo del sistema

2. Metodi e strumenti utilizzati

2.1 Sensore ad effetto di campo

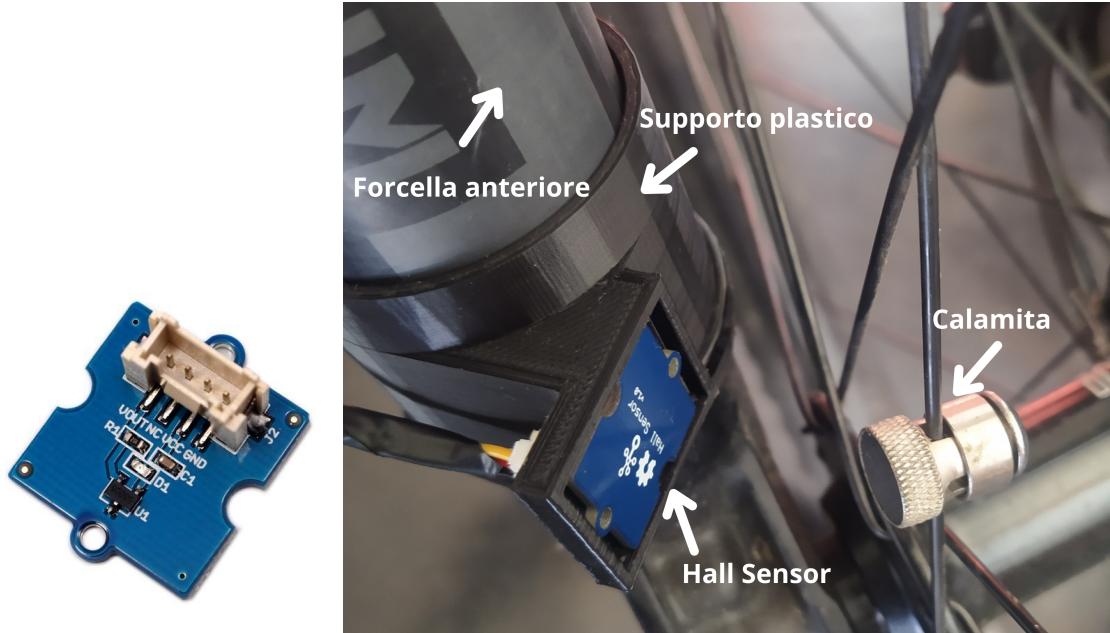


Figura 2: Grove - Hall sensor e fissaggio alla bicicletta

Il Grove - Hall Sensor (figura 2 sinistra) opera ad una tensione compresa tra 3.8V e 24V, fornisce in uscita un segnale digitale e basa il proprio funzionamento sull'effetto di campo, ossia la variazione di tensione attraverso un conduttore elettrico, causata dalla presenza di una corrente nel conduttore e di un campo magnetico posto perpendicolarmente ad essa. Il circuito di condizionamento del segnale presente sul sensore permette di ottenere una tensione di output pari a zero Volt quando un campo magnetico, la cui intensità supera un valore di soglia, viene rilevato. Quando non è presente nessun campo magnetico il dispositivo fornisce in uscita una tensione pari a quella di alimentazione. Il tempo minimo al di sotto del quale non è possibile rilevare una variazione del campo è pari a 400 nS.

Il sensore è stato agganciato al tubo della forcella anteriore e una calamita è stata collocata tra i raggi della ruota (Figura 2 destra). Ad ogni rotazione completa della ruota il sensore rileva dunque il passaggio della calamita (segnalato attraverso l'accensione di un led rosso). Questa informazione permette di risalire allo spazio percorso, semplicemente moltiplicando il numero di rotazioni della ruota per la circonferenza della stessa. Conoscendo lo spazio percorso e misurando il tempo che intercorre tra un passaggio del magnete e il precedente, è possibile invece risalire alla velocità, dividendo lo spazio per il tempo. Il microcontrollore gestisce il segnale di uscita del Grove - Hall Sensor come un segnale digitale di interrupt (figura 3).

```
//hall sensor setup
pinMode(HALLSENSOR, INPUT_PULLUP);
pinMode(LED_BUILTIN, OUTPUT);
attachInterrupt(digitalPinToInterrupt(HALLSENSOR), ISR, FALLING);
```

Figura 3: Hall sensor and interrupt setup

L'interruzione del flusso di esecuzione del codice si verifica quando in ingresso al pin 15 la tensione passa da un valore logico alto a basso, in tal caso viene eseguita la funzione di interrupt routine *ISR()* all'interno della quale vengono calcolati spazio percorso e velocità (figura 4).

```
////////// Interrupt Service Routine hall effect sensor //////////

void ISR0 {
    //***** wheel cycle detection *****/
    !state;
    digitalWrite(LED_BUILTIN, state); //blinking led
    distance += wheelCirc; //adding circonference value to total space travelled
    dt = millis() - t1; //calculate time between wheel rotations
    actualSpeed = (wheelCirc / dt)*3600; //calculate speed Km/h
    t1 = millis();
}
```

Figura 4: Interrupt Service Routine

2.2 Sensore ad ultrasuoni



Figura 5: SRF05

Per poter misurare quante volte la forcetta anteriore è stata compressa viene utilizzato un sensore ad ultrasuoni SRF05 (Figura 5). Il dispositivo richiede una tensione di alimentazione pari a 5V, fornisce una tensione di uscita digitale ed è connesso al microcontrollore per mezzo dei due pin Trigger ed Echo sui quali vengono rispettivamente generati e misurati gli impulsi. Il principio di funzionamento si basa sulla trasmissione e ricezione di onde ultrasoniche e il calcolo del tempo che intercorre tra le due.

Inizialmente viene emesso dal microcontrollore un impulso della durata di 10 microsecondi sul pin Trigger, il sensore genera dunque un burst, ossia una sequenza ravvicinata di 8 impulsi di onde ultrasoniche a frequenza 40KHz. Le onde emesse dal trasduttore cilindrico viaggiano attraverso l'aria e se è presente uno ostacolo rimbalzano e tornano indietro verso la fonte. Un segnale a valore logico alto viene generato sul pin Echo fintantoché le onde riflesse dall'ostacolo non vengono captate dal canale di ricezione (figura 6).

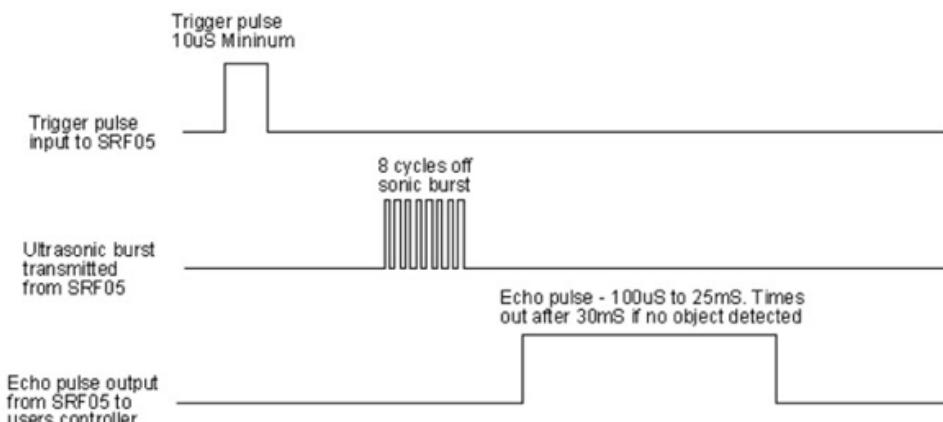


Figura 6: processo di misura della distanza

Misurando la durata del segnale Echo ottengo il tempo impiegato dall'onda per viaggiare fino all'ostacolo e tornare indietro, è possibile dunque risalire alla distanza dell'oggetto che ha causato la riflessione attraverso il seguente calcolo:

$$distanza = \frac{durata\ Impulso\ Echo}{2} \cdot velocita' Suono$$

dove la velocità del suono è pari a 331,45 m/s a cui si somma il valore della temperatura ambientale moltiplicato per la costante 0,62. La durata dell'impulso viene divisa per due siccome nel tempo misurato è considerato sia il viaggio di andata che di ritorno dell'onda.

L'SFR05 è stato agganciato alla parte superiore della forcella, mentre nella parte inferiore è stata posizionata una superficie tale da consentire alle onde ultrasoniche di essere riflesse e tornare alla fonte (figura 7). In questo modo quando la forcella è a riposo il sensore misurerà un valore costante, ossia la distanza tra il sensore stesso e la superficie, pari a circa la lunghezza della sospensione (160mm). Quando invece l'atleta lungo la discesa incorre in un ostacolo, un dosso o atterra da un salto, la forcella viene compressa e la distanza misurata si accorcia, rendendo così possibile il conteggio delle compressioni avvenute.

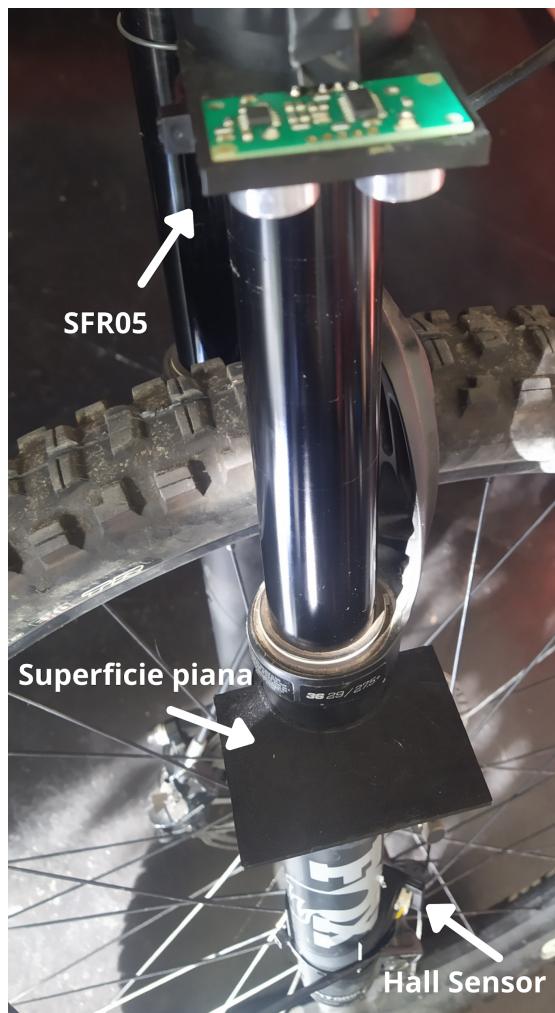


Figura 7: aggancio alla forcella anteriore del sensore ad ultrasuoni

2.3 Barometro

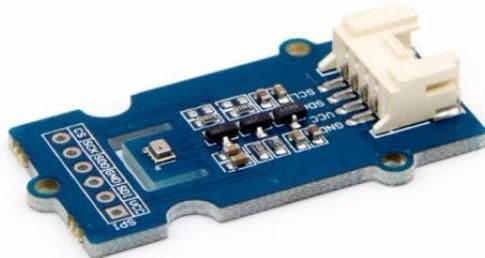


Figura 8: Grove - Barometer BMP280

Per poter misurare l'altitudine e dunque tracciare un profilo altimetrico del percorso, è necessario utilizzare un barometro, ossia uno strumento in grado di misurare la pressione atmosferica.

Il sensore utilizzato è il Grove - Barometer BMP280 (figura 8), in grado di rilevare la pressione atmosferica nel range 300 - 1100 hPa con una accuratezza di ± 1.0 hPa, la tensione di alimentazione deve essere compresa tra 3.3V e 5V e comunica col microcontrollore attraverso il protocollo di comunicazione sincrono I2C.

Una volta rilevata la pressione questa viene convertita in altitudine attraverso la seguente formula:

$$\text{altitudine} = \frac{1 - \left(\frac{\text{pressione}}{101325} \right)^{0.190263}}{0.0000225577}$$

2.4 Giroscopio e accelerometro

All'interno del microcontrollore Arduino Nano 33 IoT è inserito un LMS6DS3, ossia un IMU (Inertial Measurement Unit) composto da un accelerometro ed uno giroscopio. Il dispositivo comunica attraverso protocollo I2C e permette di misurare l'accelerazione e la velocità angolare rispetto una terna di assi XYZ (in figura 9 la disposizione della terna XYZ del sensore MPU6050, analoga a quella del LSM6DS3).

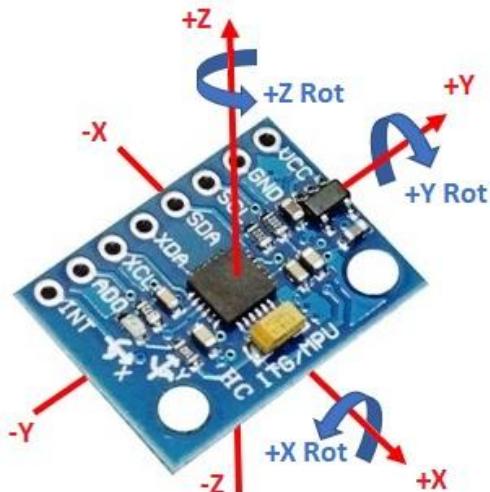


Figura 9: composizione terna XYZ

Gli angoli di pitch, roll e yaw vengono calcolati sia a partire dall'accelerazione lungo gli assi sia attraverso la velocità angolare di rotazione attorno all'asse. Infine i due valori ottenuti vengono uniti

attraverso un filtro complementare, ossia una somma pesata dei valori. Questo passaggio è fondamentale, infatti l'angolo calcolato dal solo accelerometro non è preciso essendo estremamente sensibile anche alla più piccola vibrazione, producendo così una misurazione rumorosa; il giroscopio invece è soggetto al cosiddetto drift, ossia una deriva progressiva della misurazione rendendola così inaffidabile. Unendo i due valori attraverso un filtro di Kalman o un filtro complementare si uniscono i pregi dei due sensori (sensibilità dell'accelerometro e stabilità della misura per il giroscopio) attenuando i difetti che entrambi introducono, ottenendo dunque con precisione i tre angoli. In figura 10 viene riportato il codice che implementa la lettura delle misurazioni del sensore e il filtro complementare, dove `angleX`, `angleY` e `angleZ` rappresentano rispettivamente gli angoli pitch, roll e yaw.

```
***** IMU angles *****
if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()) {
    IMU.readAcceleration(gx, gy, gz);
    IMU.readGyroscope(ax, ay, az);
    t = pow(IMU.gyroscopeSampleRate(), -1);
}

X = atan(gy / sqrt(pow(gx, 2) + pow(gz, 2))) * 57.2957795130823;
Y = atan(gx / sqrt(pow(gy, 2) + pow(gz, 2))) * 57.2957795130823;
Z = atan(gz / sqrt(pow(gy, 2) + pow(gx, 2))) * 57.2957795130823;

//complementary filter
angleX = (0.98 * (angleX + t * ax) + 0.02 * X) - angleXoffset;
angleY = (0.98 * (angleY + t * ay) + 0.02 * Y) - angleYoffset;
angleZ = (0.98 * (angleZ + t * az) + 0.02 * Z) - angleZoffset;
```

Figura x: implementazione lettura valori e filtro complementare

All'interno dell'intero sistema questo sensore permette di calcolare l'angolo di piegata della bicicletta e la pendenza del tracciato.

2.5 Led, interruttori e buzzer

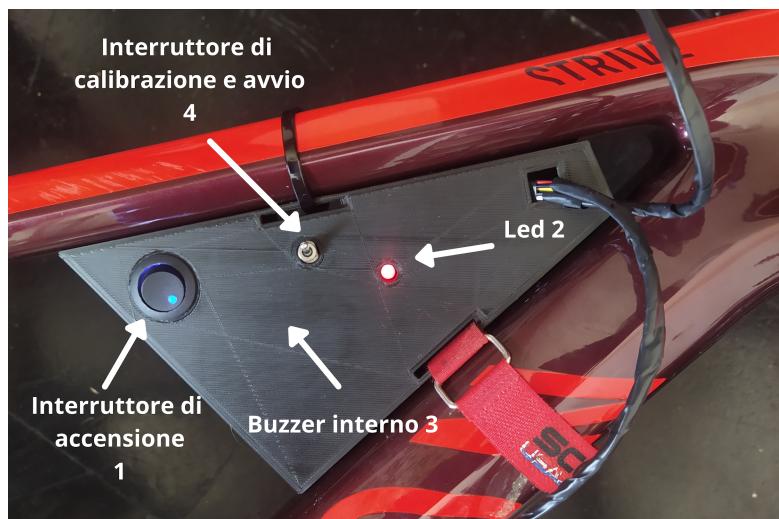


Figura 11: interfaccia utente

Per permettere all'utente di interfacciarsi con l'hardware predisposto sulla bicicletta sono presenti led, interruttori e buzzer (figura 11). L'interruttore (vedi 1) che accende l'intero sistema è dotato di un led blu che consente all'utente di verificare l'avvenuta accensione. Un led rosso (vedi 2) rimane acceso fisso nella fase di setup e lampeggiante durante l'estrazione dei dati dai sensori. Il corretto funzionamento del sistema è segnalato anche da un buzzer (vedi 3) che emette un segnale acustico. Per la fase di

calibrazione del sensore inerziale e l'avvio della fase di estrazione dati, l'utente dispone di un secondo interruttore a leva (vedi 4).

2.6 Microcontrollore e logic level shifter

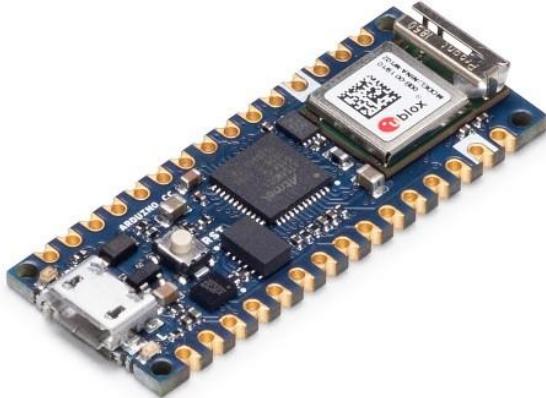


Figura 12: Arduino Nano 33 IoT

Per la gestione dei sensori e l'elaborazione e l'invio dei dati raccolti viene impiegato un Arduino Nano 33 IoT (figura 12). Proprio come dice il nome stesso è un microcontrollore dalle dimensioni estremamente ridotte (45mm per 18mm), utile per applicazioni dove lo spazio è contenuto. La sigla IoT (Internet of Things) è dovuta alla presenza del modulo WiFi NINA - W10 che permette la connessione ad una rete WiFi, opera ad una frequenza di 2.4GHz e garantisce la sicurezza della connessione grazie al chip di crittografia Microchip ECC608. Il processore su cui si basa è un Arm Cortex M0 32 - bit SAMD21 ed è anche dotato di un modulo Bluetooth e BLE per la connessione wireless con dispositivi dotati di medesima tecnologia. La tensione a cui opera è 3.3V a differenza dei 5V tipici e può essere alimentato tramite micro USB oppure alimentazione esterna compresa tra i 5V e 21V. Fornisce 14 pin digitali, 8 pin analogici con possibilità di selezionare la risoluzione dell'ADC, implementa i protocolli di comunicazione I2C e SPI ed è dotato di un sensore IMU LSM6DS3.

Tutti i sensori citati nei paragrafi precedenti e i componenti per l'interfaccia utente sono collegati ai pin digitali di I/O messi a disposizione dalla board (figura 13) avendo cura che la tensione in ingresso sia compresa tra gli 0V e 3.3V e che la corrente e tensione erogate dalla scheda siano sufficienti per soddisfare il consumo dei vari dispositivi connessi.

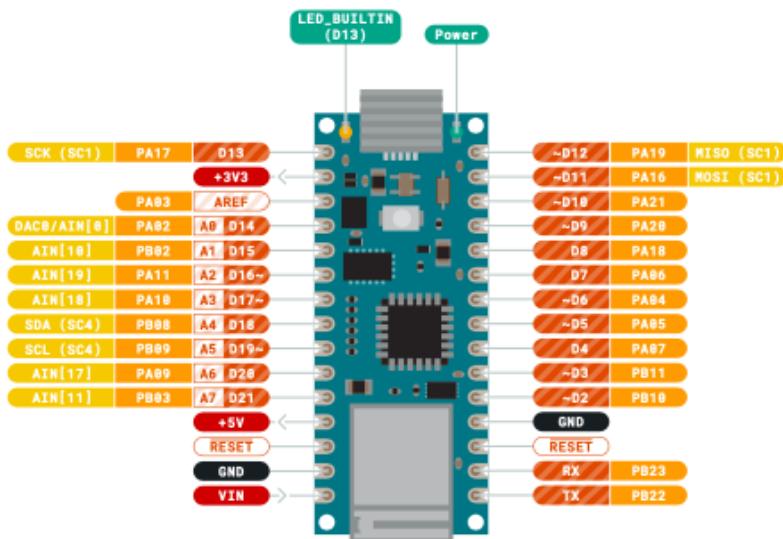


Figura 13: Arduino Nano 33 IoT pinout

Vista la presenza di sensori operanti a una tensione superiore a 3.3 V (hall sensor e sensore ad ultrasuoni) occorre anteporre al microcontrollore un logic level shifter che permette di abbassare i livelli di tensione da 0-5 V a 0-3.3 V e viceversa. Questo consente all'Arduino sia di ricevere i segnali

di output dei dispositivi connessi senza danneggiamenti, sia di inviare segnali di output senza che i rispettivi valori logici vengano male interpretati. Il logic level shifter utilizzato è quello della Pololu a 4 canali bidirezionali (figura 14).

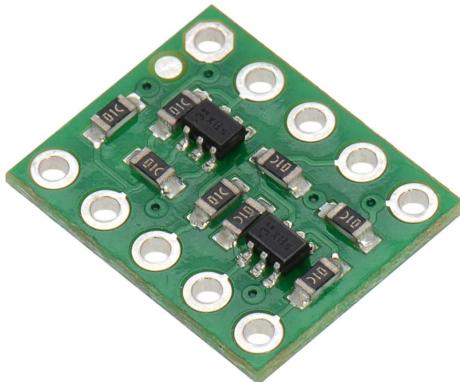


Figura 14: Pololu Logic Level Shifter

2.7 Arduino IDE e librerie

La scheda Arduino Nano 33 IoT è programmabile attraverso l'ambiente di sviluppo Arduino IDE, il linguaggio di programmazione è il C++ con l'aggiunta di una serie di funzione appositamente create per la gestione del microcontrollore e della scheda di interfacciamento, come ad esempio la gestione degli I/O.

Le librerie incluse per la gestione dei diversi dispositivi e la connessione internet sono:

- **<Seeed_BMP280.h>**: per la gestione del barometro, in grado di estrarre i valori della pressione e calcolare l'altitudine
- **<Arduino_LSM6DS3.h>**: per l'estrazione dei valori di accelerazione e velocità angolare dei tre assi dell'accelerometro e giroscopio
- **<Wire.h>**: che consente ad entrambi i sensori sopracitati di comunicare con microcontrollore attraverso il protocollo I2C
- **<WiFiNINA.h>**: per l'utilizzo del modulo WiFi NINA - W10, creata appositamente per la creazione di una connessione internet alla rete WiFi
- **<ArduinoHttpClient.h>**: necessaria per poter effettuare delle GET, POST e PUT attraverso il protocollo HTTP e connessione Internet
- **<ArduinoJson.h>**: per il parsing del JSON di risposta proveniente dal server Measurify

2.8 Measurify

Un aspetto fondamentale del progetto è la gestione dei dati ottenuti attraverso l'hardware predisposto sulla bicicletta. E' necessario che l'utente possa leggere agevolmente le informazioni ottenute lunga la discesa, in qualunque momento e in modo tale da non dover avere con sé la bicicletta, ma un dispositivo più comunemente utilizzato durante la giornata, come ad esempio uno smartphone.

Ponendosi l'obiettivo di creare un'applicazione che consenta la visualizzazione dei dati sopra citata, è indispensabile l'utilizzo di un cloud col quale comunicare agevolmente e dover poter custodire le misurazioni effettuate.

Measurify è un API cloud di tipo RESTful per la raccolta e l'elaborazione di dati provenienti da varie sorgenti dati, compresi sensori sul campo in ambito IoT, sviluppato nell'Università di Genova da Elios Lab.

Measurify si basa su alcuni concetti chiave:

- **THING**: si tratta di una "cosa" (una persona, un ambiente, un oggetto, ecc.) per la quale si sta facendo una misurazione. Ad esempio, la stanza di una casa.
- **FEATURE**: si tratta di che cosa si sta misurando (la temperatura, l'umidità, il peso, ecc.). Ad esempio, potrebbe trattarsi della temperatura che si ha in una stanza.
- **DEVICE**: si tratta di un dispositivo (hardware/software) che può misurare una certa dimensione (feature) su una determinata cosa (thing). Ad esempio, il termometro che misura la temperatura della stanza.
- **MEASUREMENT**: si tratta di una misura effettuata da un dispositivo (device) per una certa dimensione (feature) su una determinata cosa (thing). Ad esempio: il fatto che nella stanza (things), vi sia una certa temperatura (feature) in un certo istante e che lo dica il dispositivo termometro (device).

In questo progetto la bicicletta è la THING, sulla quale sono agganciati diversi sensori, ossia i DEVICEs. Ogni device misura una FEATURE effettuando una MEASUREMENT, in questo caso saranno velocità, altitudine, spazio percorso, pendenza, angolo di piegata e compressione della forcella.

Tali concetti diventano in Measurify degli oggetti, modellati poi come risorse; queste contengono al loro interno oggetti veri e propri, ognuno con i rispettivi modelli e funzionalità accessibili attraverso un set di RESTful API. Tramite le API è possibile accedere grazie alla connessione internet agli oggetti di ogni risorsa. Gli oggetti sono modellati in formato JSON e sono collegati insieme tra loro (figura 15).

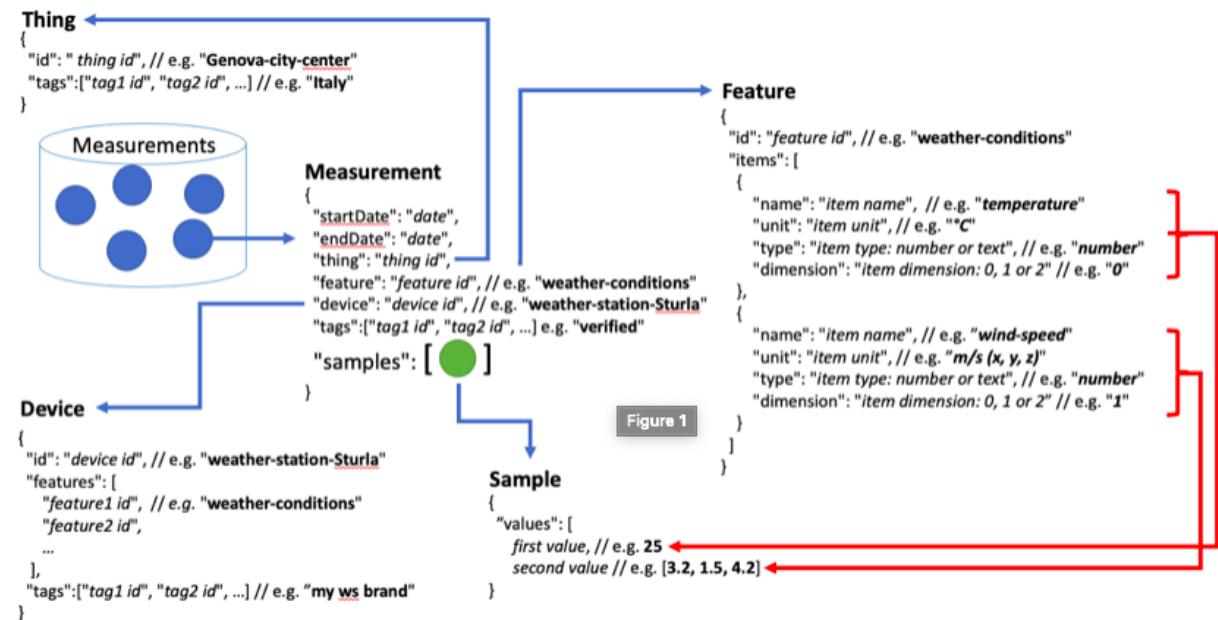


Figura 15: modellazione e interconnessione degli oggetti Measurify

La gestione delle risorse da parte delle Cloud API è schematizzabile come in figura 16. Come si può vedere il server Measurify è dotato di un modulo Secure Proxy che garantisce la sicurezza delle risorse obbligando l'utente ad autenticarsi al server e fornendo un token necessario per effettuare qualunque operazione su di esse (ad esempio POST, GET e PUT).

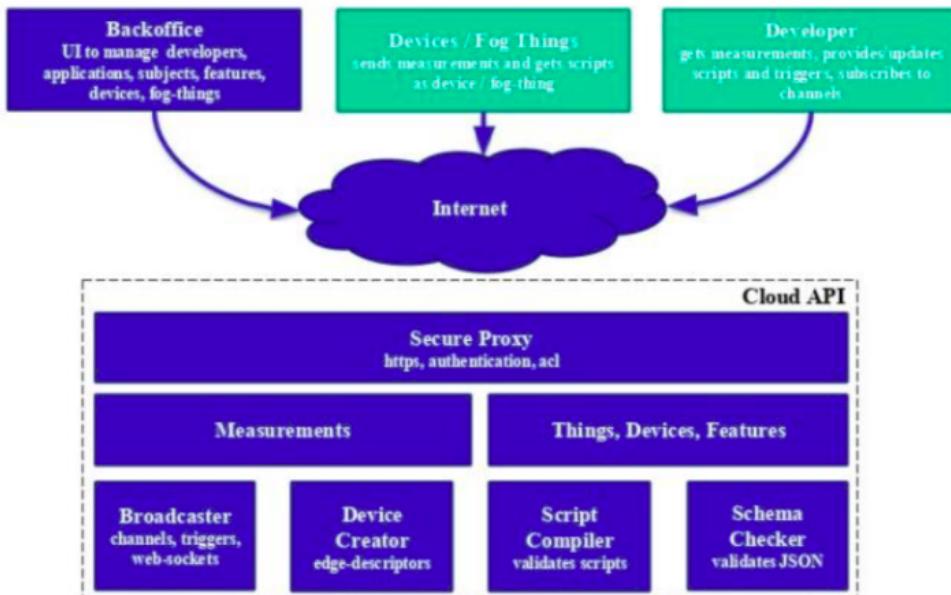


Figura 16: schema della gestione delle risorse

2.9 Flutter

Flutter è un framework open-source creato da Google, con lo scopo di sviluppare applicazioni native per IOS e Android e anche applicazione Web. Come editor è stato utilizzato Visual Studio Code, realizzato da Microsoft, il quale implementa e si interfaccia con l'SDK.

Flutter si basa su questi elementi principali:

Dart platform: E' un linguaggio di programmazione sviluppato da Google con lo scopo di sostituire JavaScript per lo sviluppo Web. Il compilatore di Dart permette di scrivere programmi sia per web che per desktop, smartphone e server, tramite due diverse piattaforme. La prima è Dart Native pensata per smartphone, server, desktop e altro. La seconda è Dart Web pensata appositamente per il web.

Flutter Engine: Scritto principalmente in C++, è ciò che rende possibile il rendering di basso livello utilizzando la libreria grafica di Google, chiamata Skia Graphics. Una delle funzionalità più apprezzate di questo motore, è nominata "hot-reload" e permette, in fase di sviluppo, di mostrare le modifiche apportate senza il bisogno di riavviare l'applicazione.

Foundation Library: Si tratta di una libreria scritta in Dart, la quale fornisce tutte le classi e i metodi di base per sviluppare applicazioni tramite Flutter.

Widgets: Il codice sviluppato con Flutter si basa sui widgets, i quali sono descrizioni immutabili dell'interfaccia utente, alcuni esempi possono essere testi, buttoni, forme, grafici e animazioni. I widgets sono una caratteristica di Flutter che rende semplice e intuitivo lo sviluppo di applicazioni, grazie ad essi e alla loro combinazione è possibile creare applicazioni complesse a partire da elementi semplici.

3. Sperimentazione e risultati

Come accennato nei paragrafi precedenti, il sistema embedded atto all'estrazione dei dati dai sensori, alla loro elaborazione ed infine al loro invio è composto dal microcontrollore Arduino Nano 33 IoT, il quale gestisce l'accelerometro - giroscopio LSM6DS3 e il barometro BMP280 attraverso comunicazione I2C, il sensore ad effetto di campo il cui output funge da interrupt, il sensore ad ultrasuoni SFR05 ed infine led, interruttori e buzzer per l'interfaccia utente.

I collegamenti dei diversi componenti al microcontrollore è avvenuta mediante saldatura a stagno su basetta "millefori". L'intero sistema è alimentato da una batteria da 6V. Di seguito uno schema rappresentante in maniera semplificata i circuiti di collegamento:

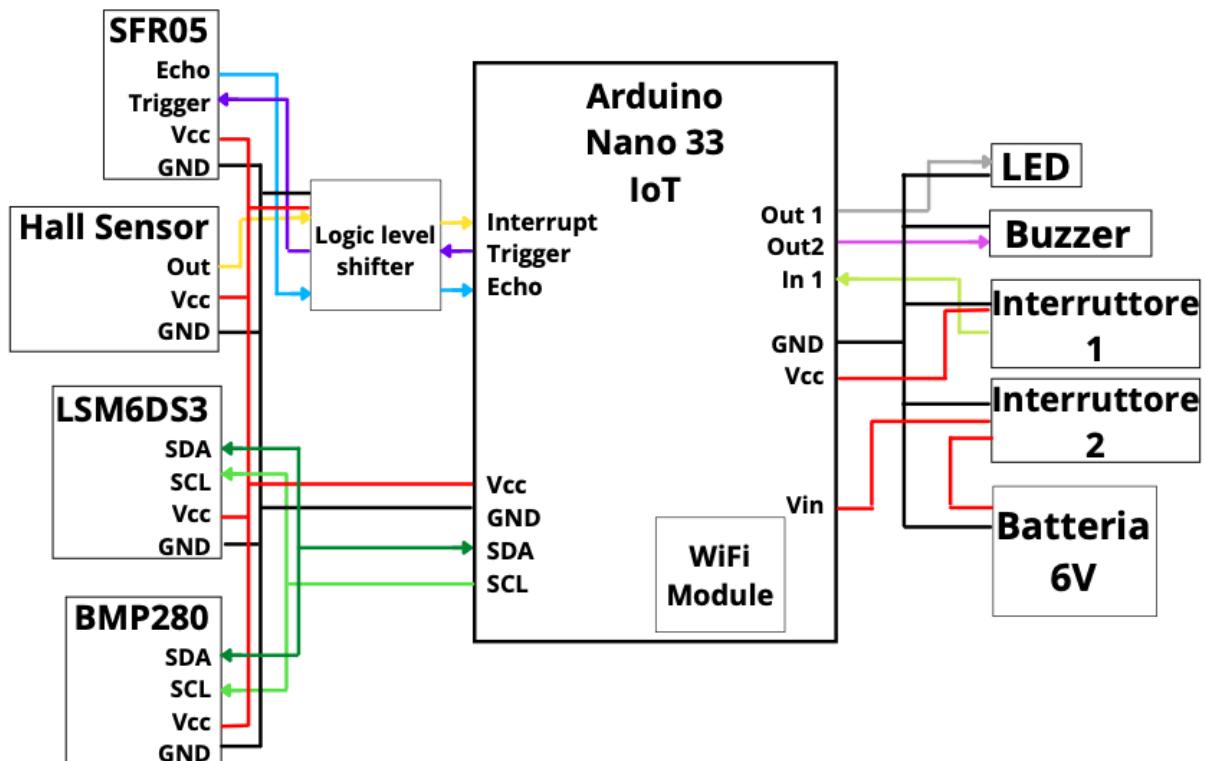


Figura 17: collegamenti hardware

Nell'immagine sottostante sono raffigurati tutti i componenti hardware contenuti all'interno del case, gli unici dispositivi esterni sono il sensore ad effetto di campo e il sensore ad ultrasuoni, dei quali si vedono i cavi uscenti (collegamenti gialli).

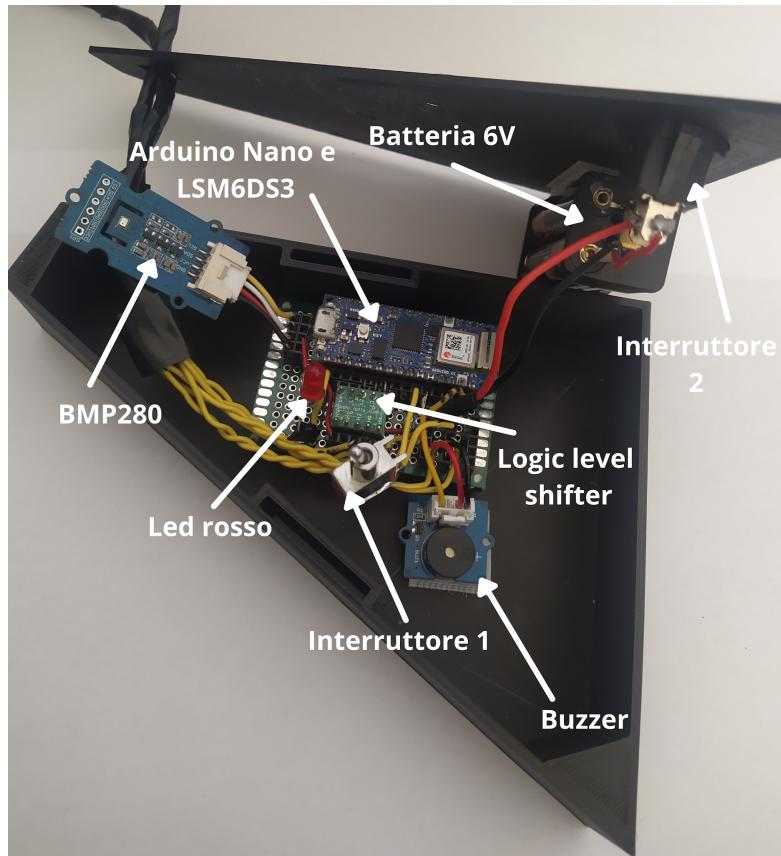


Figura 18: fotografia dell'hardware contenuto nel case

La programmazione dell'hardware è avvenuta attraverso l'ambiente di sviluppo Arduino IDE, in linguaggio C++. La logica che segue l'algoritmo si compone delle seguenti fasi:

- Set Up
- Calibrazione
- Estrazione dati
- Invio dati

3.1 Set Up

Al momento dell'accensione del sistema, vengono inizializzate tutte le variabili necessarie per la gestione dei sensori e attuatori, della connessione internet, delle post HTTP e definite le modalità dei pin (input/output e analogico/digitale). Viene inoltre verificato il corretto collegamento e funzionamento dei sensori e degli attuatori.

3.2 Calibrazione

Per poter calcolare la pendenza del tracciato e l'angolo di piegata raggiunto durante la discesa, occorre impostare i cosiddetti valori di offset, ossia gli angoli registrati dal sensore quando la bicicletta è in piano. Viene avviata dall'utente mediante un interruttore a leva e segnalata attraverso un segnale acustico.

La calibrazione è gestita da una funzione apposita chiamata *IMUcalibration()*, all'interno della quale vengono effettuate 30 misurazioni e poi calcolata la media.

3.3 Estrazione dati

Quando l'utente cambia lo stato dell'interruttore inizia la fase di estrazione dei dati dai sensori e rispettiva elaborazione, entrando in loop destinato a concludersi quando l'interruttore viene nuovamente premuto o quando la memoria dell'Arduino raggiunge la capienza massima. All'interno del suddetto loop vengono eseguite le seguenti istruzioni.

Barometro e accelerometro e giroscopio attraverso il protocollo di comunicazione I2C scambiano dati con l'Arduino sul canale SDA a ritmo di clock del canale SCL. Le misurazioni di accelerazione e velocità angolare avvengono ogni 20 mS, trattandosi di un'informazione soggetta a repentini cambiamenti, combinate poi assieme attraverso il filtro complementare. Per la pressione atmosferica invece, dalla quale si deriva l'altitudine, è sufficiente una misurazione ogni secondo, così come per l'aggiornamento della velocità. Ciclicamente viene calcolata anche la distanza misurata dal sensore ad ultrasuoni, e conservando le distanze precedenti è possibile determinare se la forcella è stata compressa o meno, aggiornando così la variabile del conteggio. Durante il regolare flusso di svolgimento delle istruzioni appena descritte, avvengono delle interruzioni dovute dal segnale emesso dal sensore ad effetto di campo. Qualora la calamita passa vicino al sensore il segnale di interrupt impone al microcontrollore l'esecuzione di una funzione predisposta all'aggiornamento dello spazio percorso e calcolo della velocità.

Altitudine e velocità vanno a riempire due array inserendo un elemento ciascuno ogni secondo. La grandezza degli array è fissa e pertanto limitata. Qualora la discesa supera la durata ci circa 25 min gli array raggiungendo il massimo della capienza, obbligando il sistema a concludere l'estrazione dei dati e passare alla fase di invio. Di seguito viene presentato lo schema logico del loop una volta avviata la fase di estrazione dati:

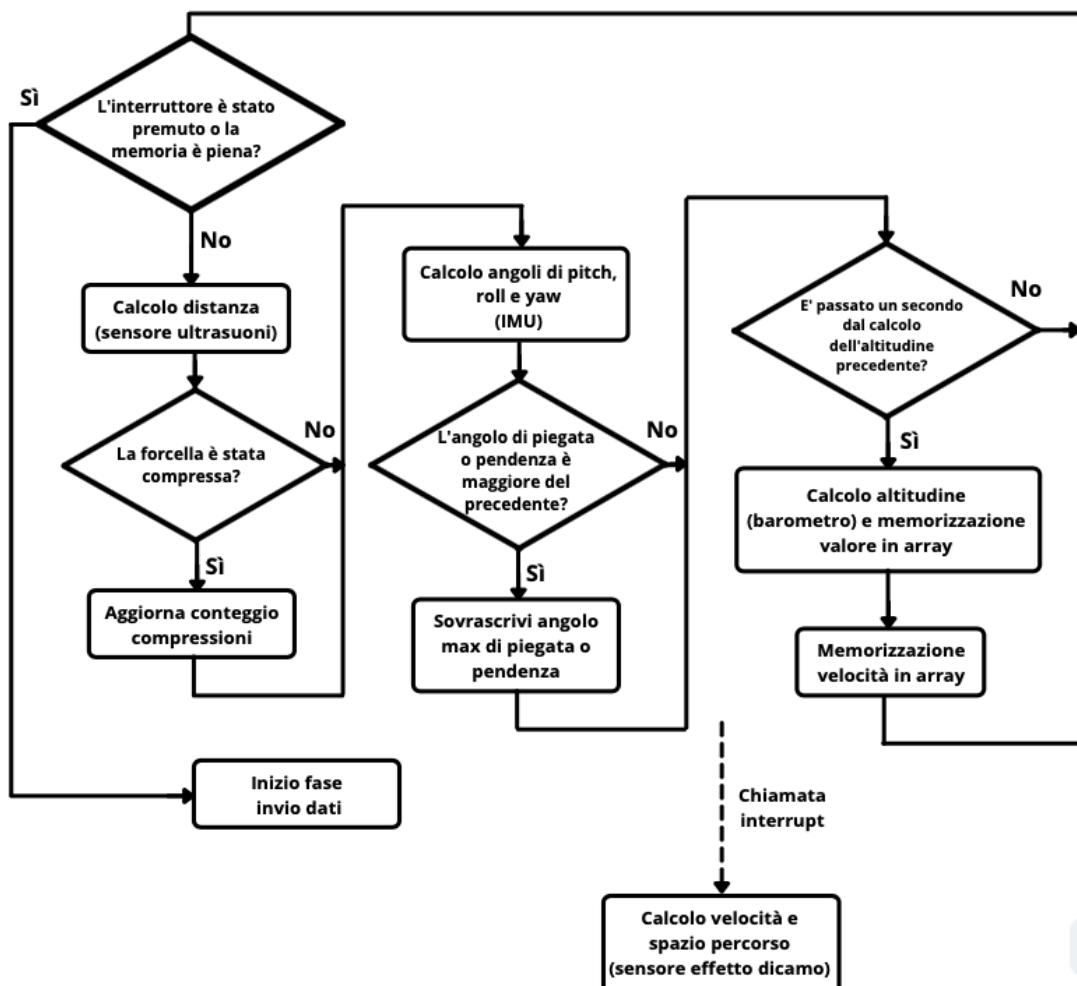


Figura 19: schema logico loop estrazione dati

3.4 Invio dei dati

Cambiando nuovamente lo stato dell'interruttore si esce dalla fase di registrazione e si passa a quella di invio. Inizialmente viene stabilita una connessione internet, è necessario dunque attivare l'hotspot del telefono o essere nei pressi di una rete WiFi locale. Fintantoché la connessione non è avvenuta l'Arduino attende e riprova.

Dopodichè si passa all'invio dei dati: Arduino effettua una POST di login al server di Measurify inviando le credenziali di accesso in formato JSON.

```
{  
  "username" : "smartbike-monitor-user-username",  
  "password" : "smartbike-monitor-user-password"  
}
```

Figura 20: credenziali di accesso in formato JSON

La risposta è un token da inserire nella POST successiva la quale invia tutte le misurazioni effettuate (sempre in formato JSON).

```
{  
  "token": "JWT eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJ1c2VyIjp7InN0YXR1cyI6ImVuYWJsZWQiLCJfaWQiOiI1ZWNmYTxk3NzA5OTQyNzI0NTQyYmQiLCJ1c2VybmtZSI6InNt  
YXJ0YmlrZS1tb25pdG9yLXVzZXItdXNlcmb5hbWUiLCJwYXNzd29yZC16InNtYXJ0YmlrZS1tb25pdG9yLXVzZXItcGFzc3dvcmQi  
LCJ0eXBBIjoicHJvdmlkZXIiLCJFX3Y10jb9LCJ0ZW5hbnQiOnsicGFzc3dvcmRoYXNoIjpmYWxzzSwiX2lkIjoiYXRtb3NwaGVy  
ZS1wcm9kIn0sImlhdCI6MTYxMDcyMzI4NSwiZXhwIjoxNjEwNzI1MDg1fQ.  
ZoBV5A8GFTRZUgbibQrnTZmv0Xl-s8h2QjzypP_hZIM"  
}
```

Figura 21: esempio di token di risposta

La post viene eseguita attraverso il protocollo HTTP, esplicitando alcuni headers, come mostrato in immagine.

```
String postData = "{\"thing\": \"mybike\", \"feature\": \"bike_run\", \"device\":\n    \"bike-monitor\", \"samples\": [ { \"values\": " + samples + " } ]}";\nSerial.println(postData);\nclient.beginRequest();\nclient.post("/v1/measurements");\nclient.sendHeader("Content-Type", "application/json");\nclient.sendHeader("Content-Length", postData.length());\nclient.sendHeader("Host", "students.atmosphere.tools");\nclient.sendHeader("Connection: keep-alive");\nclient.sendHeader("Authorization", token);\nclient.beginBody();\nclient.print(postData);\nclient.endRequest();
```

Figura 22: headers post HTTP

Il body contiene nel primo caso le credenziali di accesso a Measurify e nel secondo caso la feature da inviare. La feature, anch'essa in formato JSON, è composta da svariati campi, il più importante è *samples*, contenente un “array” così strutturato:

- 1° elemento: è a sua volta un array contenente i valori della velocità registrati ciclicamente.
- 2° elemento: array contenente i valori di altitudine registrati
- 3° elemento: durata totale della discesa

- 4° elemento:** distanza percorsa
- 5° elemento:** angolo massimo di piegata
- 6° elemento:** pendenza massima del tracciato
- 7° elemento:** numero di compressioni della forcella

Tutte le informazioni rilevate durante la discesa vengono dunque inviate in un'unica post, sfruttando la possibilità di comporre una lista di array e valori singoli del campo samples.

3.5 Applicazione smartphone

Il modo più pratico per far sì che l'atleta possa consultare agevolmente i risultati della discesa è sicuramente quello di realizzare un'app dedicata. Attraverso lo smartphone è possibile leggere i dati rilevati comodamente e in qualunque momento, senza la necessità di essere fisicamente nei pressi della bicicletta a cui è collegato l'intero sistema.

L'applicazione è stata sviluppata attraverso il framework open source Flutter, realizzato da Google. Mediante la conoscenza del linguaggio Dart è stato possibile gestire le post e get al server di Measurify e visualizzare i dati attraverso grafici ed icone.

L'applicazione chiamata SmartBike è composta essenzialmente da due schermate, la prima funge da schermata di caricamento, nella seconda invece si passa direttamente alla visualizzazione dei dati.

3.5.1 Fase di caricamento

L'applicazione effettua come prima fase l'autenticazione a Measurify. Attraverso una post al server contenente come body le credenziali di accesso, l'app ottiene in risposta il token, fondamentale per poter effettuare qualunque operazione di scambio, inserimento o modifica dei dati. La risposta ricevuta deve essere gestita come un evento asincrono della quale non è possibile sapere né quando arriverà né se effettivamente ne otterrò una. Il programma aspetta dunque la risposta una volta inviata la post e verifica se lo status code di ritorno è pari a 200 (operazione andata a buon fine).

Il token ottenuto viene dunque inserito nella get successiva, nella quale viene fatta richiesta di prelevare l'ultima feature inserita all'interno del server Measurify dal sistema smart-bike. La risposta sarà dunque un file JSON contenente la feature che, opportunamente "parsata", mi fornisce in unica Map tutti i valori inviati dalla bicicletta, mantenendo la struttura dati con la quale sono state inviate (velocità e altitudine rimangono sempre due array, mentre le altre misurazioni valori puntuali).

Le operazioni di accesso a Measurify e parsing del JSON di risposta alla get, vengono effettuati durante la schermata di caricamento. Fintantoché non vengono eseguite compare il simbolo circolare rappresentante la fase di loading. Se una tra post e get non forniscono uno status code di risposta pari a 200, compare un messaggio di errore ed occorre effettuare nuovamente l'operazione.

3.5.2 Visualizzazione dei dati

Ogni elemento della Map è composto dalla chiave di ricerca, ossia il nome della misurazione (ad esempio "Time" oppure "Max slope angle"), e dal valore associato, in tal modo è possibile ricercare un dato semplicemente chiamando la chiave di ricerca corrispondente. La funzione che si occupa di prelevare i dati e visualizzarli nel modo opportuno effettua una ricerca sfruttando la struttura sopra citata.

Gli array della velocità e dell'altitudine vengono utilizzati per creare dei grafici aventi sull'asse delle ordinate il tempo in secondi e su quello delle ascisse la velocità in chilometri orari o l'altitudine in metri.

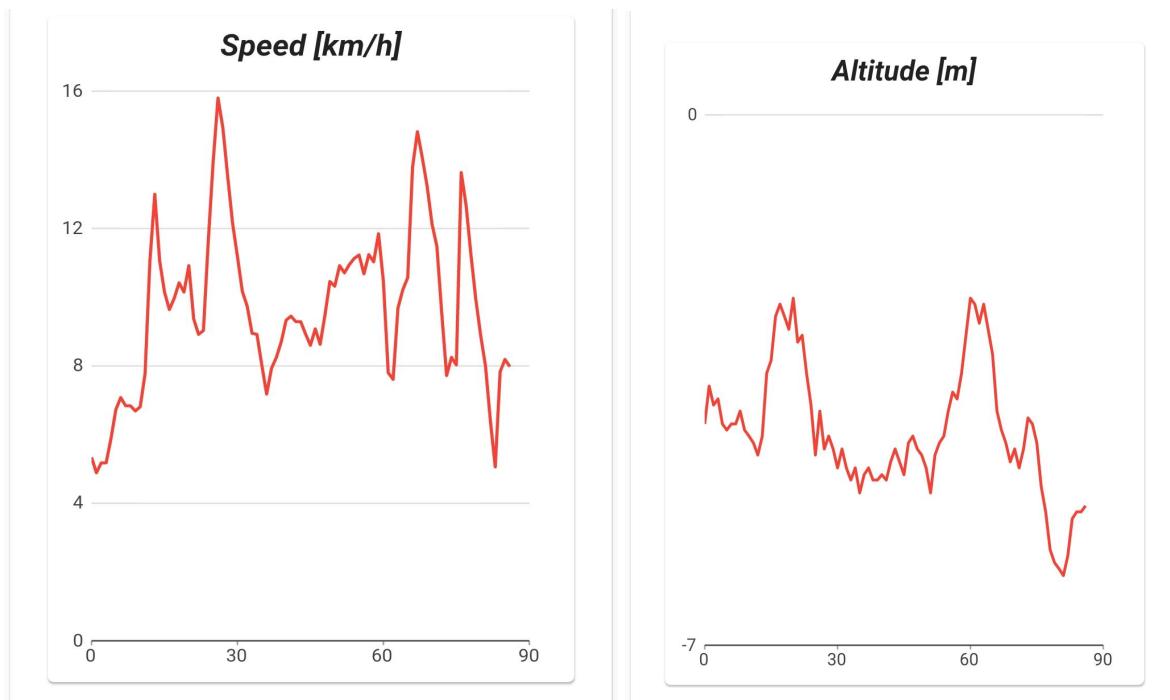


Figura 23: grafici della velocità e altitudine

Le misurazioni puntuali invece vengono affiancate da una icona e da un'intestazione che descrive il significato del dato ed esplicita l'unità di misura.

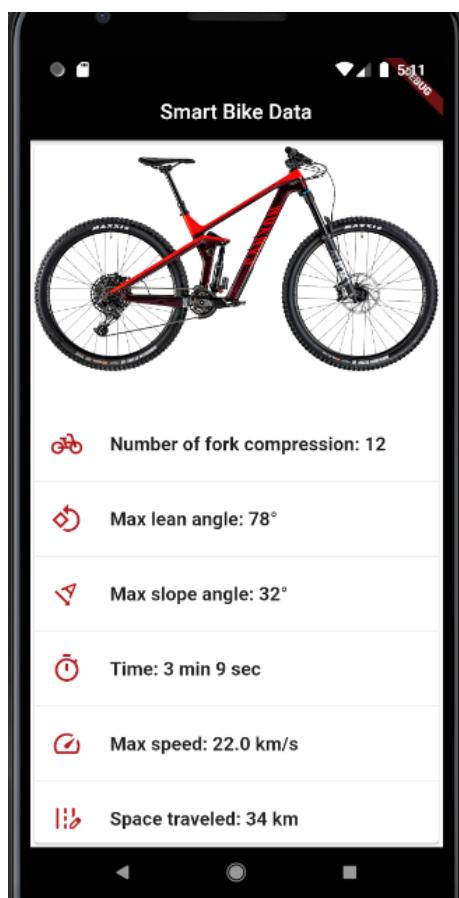


Figura 24: elenco misurazioni puntuali

3.6 Sperimentazione su sentiero

Una volta verificato il corretto funzionamento dei sensori, dell'elaborazione dei dati, dell'invio delle informazioni e della loro visualizzazione tramite applicazione, si passa alla fase di sperimentazione successiva. Pertanto, dopo aver agganciato l'hardware alla bicicletta attraverso gli appositi supporti plastici, interamente modellati con AutoCad 2021 e stampati con una Ender 3 Pro, è necessario mettere alla prova l'intero sistema nell'ambiente per il quale è stato progettato, ossia su sentiero.

Dopo svariate discese l'hardware risulta saldamente agganciato al mezzo, senza aver subito danneggiamenti dovuti agli scossoni. I supporti plastici si sono rivelati robusti e funzionali, permettendo ai sensori di svolgere il loro compito senza differenze rispetto ai primi test a bici ferma. L'unica eccezione si presenta per il sensore ad ultrasuoni, le sue misurazioni infatti, a causa delle condizioni ambientali (polvere, vento, schizzi d'acqua, vibrazioni), non risultano precise, pertanto il numero di compressioni contate si discosta leggermente da quello reale. L'interfaccia usata per far comunicare l'utente col sistema, composta da interrutori, buzzer e led, benché minimale, consente in maniera semplice ed efficace di avviare la fase di calibrazione, misurazione ed invio dati.

L'applicazione esegue correttamente la rappresentazione dei dati ogni volta che viene avviata, senza presentare differenze rispetto alla simulazione effettuata con gli appositi tool di Flutter.

Nell'immagine sottostante si può vedere il sistema Smart Bike collegato alla bicicletta, durante uno dei test di collaudo.

Figura 25: sistema embedded su bicicletta

4. Contributo personale e considerazioni conclusive

Lo scopo del progetto SmartBike è quello di monitorare le prestazioni di un atleta che intraprende una discesa in bicicletta lungo un percorso montuoso. Per raggiungere tale obiettivo si è scelto di utilizzare un approccio IoT, rendendo dunque non solo la bicicletta intelligente attraverso sensori e microcontrollore, ma anche in grado riversare informazioni in rete attraverso protocollo http. Le conoscenze nella programmazione di dispositivi elettronici e applicazioni si sono rivelate fondamentali per la riuscita del progetto.

Per quanto concerne l'hardware utilizzato, la scelta è ricaduta principalmente su dispositivi e componenti della Grove per la sensoristica e della Arduino e Pololu per il microcontrollore e l'elettronica utilizzata nell'interfaccia utente. Tutti i componenti si sono dimostrati all'altezza delle aspettative, considerando anche il budget limitato.

Uno degli aspetti più complessi da gestire è stata la connessione dell'intero sistema alla bicicletta. Per creare dei supporti di aggancio che si adattassero alla forma dei componenti elettronici e alla complessa geometria del telaio della bicicletta, è stato usato AutoCad 2021, creando così dei modelli 3D stampati successivamente mediante una stampante 3D Ender 3 Pro. Tali supporti sono creati ad hoc per il modello di mountain bike utilizzata, questo aspetto fa pensare ad un possibile miglioramento del progetto, ossia rendere il sistema adattabile a qualunque mezzo.

La questione dell'alimentazione del sistema non è stata affrontata nel dettaglio. Le batterie utilizzate infatti permettono un utilizzo limitato e occupano molto spazio, inoltre non essendo ricaricabili costringono l'utente a doverle sostituire periodicamente. L'utente non ha nemmeno la possibilità di sapere il livello di carica al momento dell'accensione. Indubbiamente sono aspetti che andrebbero approfonditi per migliorare la qualità del servizio che questo progetto si pone di offrire.

L'ambiente nel quale la SmartBike viene utilizzata è decisamente impervio, basti pensare al fango, la polvere, l'umidità, le temperature elevate o viceversa, eventuali schizzi d'acqua, il terreno accidentato. Sono tutti elementi che possono compromettere l'hardware e l'elaborazione delle informazioni attinte. Inoltre un'eventuale caduta potrebbe causare la rottura dei supporti. Occorre dunque rendere i case nei quali si trovano i diversi dispositivi a prova di polvere, acqua, urti, ecc...

Lo sviluppo dell'applicazione avvenuto attraverso Flutter ha reso semplice e veloce la creazione di un'app che soddisfi i requisiti di estrazione dei dati da Measurify e la loro rappresentazione grafica. L'aspetto grafico della SmartBike App è molto semplice, ma dedicando ulteriore tempo al suo sviluppo sarebbe possibile renderla di maggiore impatto sia aggiungendo una rappresentazione grafica dei dati animata, sia inserendo delle funzionalità che permettono di customizzare l'app da parte dell'utente. Un possibile upgrade consiste nell'inserimento della traccia GPS del tracciato e la possibilità di comunicare all'utente i risultati delle discese nelle diverse aree geografiche in cui si sono svolte. Addirittura è possibile creare una community dove ognuno mostra i propri risultati nei tracciati percorsi, dando la possibilità così di confrontarsi e stabilire una classifica delle prestazioni.

5. Riferimenti bibliografici

- [1] Arduino Nano 33 IoT, <https://store.arduino.cc/arduino-nano-33-iot>
- [2] Grove - Hall Sensor, https://wiki.seeedstudio.com/Grove-Hall_Sensor/
- [3] Grove - BMP280, https://wiki.seeedstudio.com/Grove-Barometer_Sensor-BMP280/
- [4] Grove - Buzzer, <https://wiki.seeedstudio.com/Grove-Buzzer/>
- [5] SFR05, <https://www.robot-electronics.co.uk/htm/srf05tech.htm>
- [6] LMS6DS3, <https://www.arduino.cc/reference/en/libraries/accelerometer-and-gyroscope-lsm6ds3/>
- [7] Pololu Logic Level Shifter, <https://www.pololu.com/product/2595>
- [8] BMP280 Library, https://github.com/Seeed-Studio/Grove_BMP280
- [9] LMS6DS3 Library, <https://www.arduino.cc/en/Reference/ArduinoLSM6DS3>
- [10] WiFiNINA Library Arduino, <https://www.arduino.cc/en/Reference/WiFiNINA>
- [11] ArduinoJson Library, <https://arduinojson.org/>
- [12] Measurify, <https://measurify.org/>