



UNIVERSITÀ DEGLI STUDI DI GENOVA

DIPARTIMENTO DI INGEGNERIA NAVALE, ELETTRICA,
ELETTRONICA E DELLE TELECOMUNICAZIONI

CORSO DI STUDIO IN INGEGNERIA ELETTRONICA E TECNOLOGIE
DELL'INFORMAZIONE

Tesi di laurea triennale

Luglio 2025

**Progetto e implementazione di un sistema embedded per
la rilevazione degli ostacoli su auto radiocomandate**

**Design and development of an embedded system for
obstacle detection on radio-controlled cars**

Candidato: Pietro Fallanca

Relatore: Prof. Riccardo Berta

Correlatore: Dr. Matteo Fresta

Sommario

Il presente elaborato si propone di descrivere la progettazione e l'implementazione di un sistema embedded per la rilevazione degli ostacoli su auto radiocomandate. Il sistema è composto da un microcontrollore Arduino Due, incaricato di raccogliere le misurazioni di distanza provenienti da 8 sensori di prossimità posti sulla macchinina. I dati acquisiti vengono trasmessi tramite comunicazione seriale a un Arduino Nano 33 BLE, che si occupa di elaborarli, filtrare eventuali valori anomali e inviarli via Bluetooth Low Energy a un'applicazione sviluppata con Flutter. Quest'app consente all'utente di visualizzare in modo semplice e intuitivo la presenza di ostacoli attorno alla macchinina. Successivamente, i dati vengono inviati tramite connessione dati o Wi-Fi al server Measurify, dove vengono archiviati in un database. Infine, un sito web dedicato consente di visualizzare le misure raccolte, rappresentate graficamente per mostrare l'andamento delle distanze rilevate nel tempo.

Indice

1 Introduzione.....	4
2 Strumenti utilizzati.....	5
2.1 Schede Arduino	5
2.2 Sensori di prossimità	7
2.3 Codice Arduino	8
2.4 Applicazione Flutter	13
2.5 Measurify	17
3 Sperimentazione e risultati.....	19
4 Contributo personale e considerazioni conclusive.....	21
5 Riferimenti bibliografici.....	22
6 Ringraziamenti.....	23

1 Introduzione

L'Internet of Things (IoT) [1] è una rete interconnessa di oggetti e dispositivi (things) dotati di sensori e altre tecnologie in grado di trasmettere e ricevere dati, da e verso altre cose e sistemi. L'IoT comprende qualsiasi oggetto in grado di essere collegato tramite connessione wireless a una rete Internet. Negli anni le possibili applicazioni sono aumentate a dismisura: spazia dalla robotica, domotica, videosorveglianza fino all'agricoltura e sanità. I dispositivi IoT sono ormai diventati i nostri occhi e le nostre orecchie, si occupano dell'acquisizione dei dati nell'ambiente in cui sono installati attraverso i sensori, successivamente possono rimanere archiviati localmente, inviati a un sistema cloud in rete o a un altro dispositivo. Questi dati vengono quindi elaborati da un software e possono essere utilizzati per compiere una semplice azione tipo aprire una porta, oppure essere analizzati per la creazione di grafici e tabelle per essere visualizzate dall'utente finale.

Lo scopo di questo progetto è quello di applicare il concetto di IoT a una macchinina radiocomandata. Nel nostro caso le "things" sono rappresentate dagli 8 sensori di prossimità e dai microcontrollori Arduino utilizzati. Il primo Arduino Due ha la funzione di raccogliere le distanze dagli 8 sensori, trasformarle in un formato adatto e trasmetterli al secondo Arduino Nano 33 BLE tramite comunicazione seriale. Dopodiché i dati vengono adattati a un formato compatibile con la trasmissione Bluetooth, che viene effettuata tramite il secondo Arduino e una applicazione Flutter programmata in Dart. L'app è costituita da una schermata principale che permette all'utente che la utilizza di visualizzare tutte le periferiche bluetooth in prossimità del proprio cellulare, con una barra di segnale dinamica che ne indica la distanza, da qui è possibile selezionare il secondo Arduino. Fare ciò apre una seconda schermata adibita alla connessione con il dispositivo precedentemente selezionato, qui sarà possibile connettersi e disconnettersi. Una volta connessi si potrà premere un pulsante per aprire una terza schermata che consentirà all'utente di visualizzare la macchinina a schermo con una grafica user friendly che mostra la distanza degli otto sensori tramite una scala di colori che vanno dal rosso al verde. L'app oltre a ciò ha anche il compito di analizzare una certa quantità di dati ricevuti dai sensori e restituire in output una loro media per aumentare il più possibile la precisione ed evitare edge case contenenti valori sballati. Dopo aver raccolto una certa quantità di dati l'app li invierà tramite API REST al server di Measurify mediante il protocollo HTTPS e richieste POST, a questo punto sarà possibile visualizzare tutti i dati raccolti con grafici nel dominio del tempo tramite l'apposito sito di Measurify.

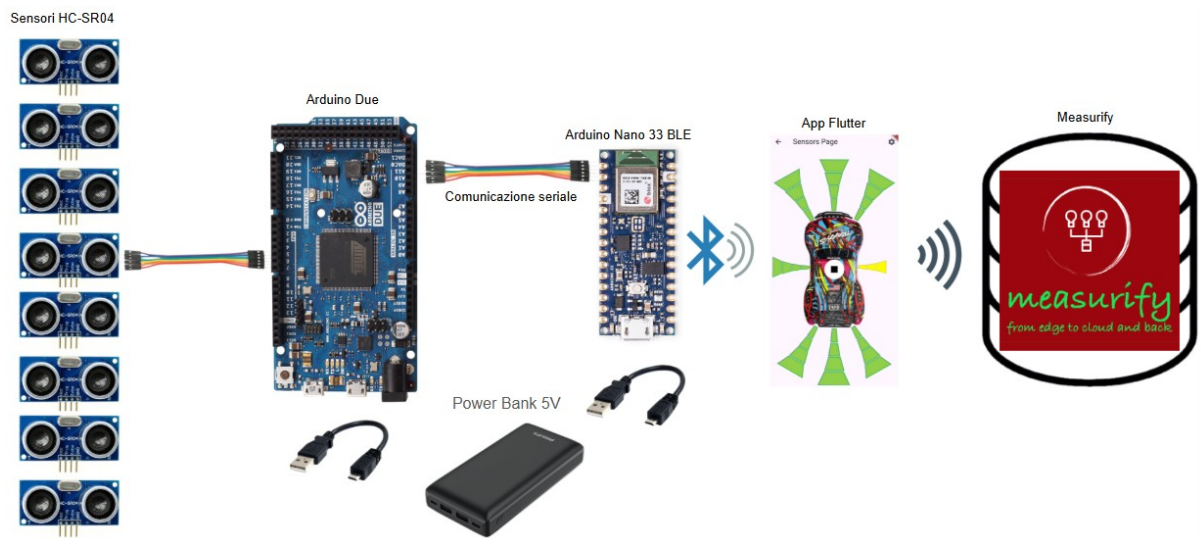


Figura 1: Schema di progetto

2 Strumenti utilizzati

2.1 Schede Arduino

I sistemi embedded utilizzati per lo sviluppo di questo progetto sono Arduino Due [3] e Arduino Nano 33 BLE (Bluetooth Low Energy) [4]. Il primo è dotato di un processore Atmel SAM3X8E ARM da 32-bit, più potente e ampio del suo predecessore Arduino Uno. Fornisce ben 54 pin digitali e 12 pin analogici, la sua grande quantità di pin ci permette di collegare un numero molto elevato di sensori, a noi ne bastano 8, ma nel caso ce ne fosse il bisogno è possibile aumentarne il numero ed eventualmente aggiungerne di ulteriori. Ci interessano soprattutto i pin PWM, questa scheda ne fornisce fino a 12 e servono per l'acquisizione dei segnali di ritorno inviati dai sensori di prossimità. È possibile fornirgli alimentazione a 5V tramite cavo micro-USB oppure mediante un alimentatore esterno dai 7V ai 12V.

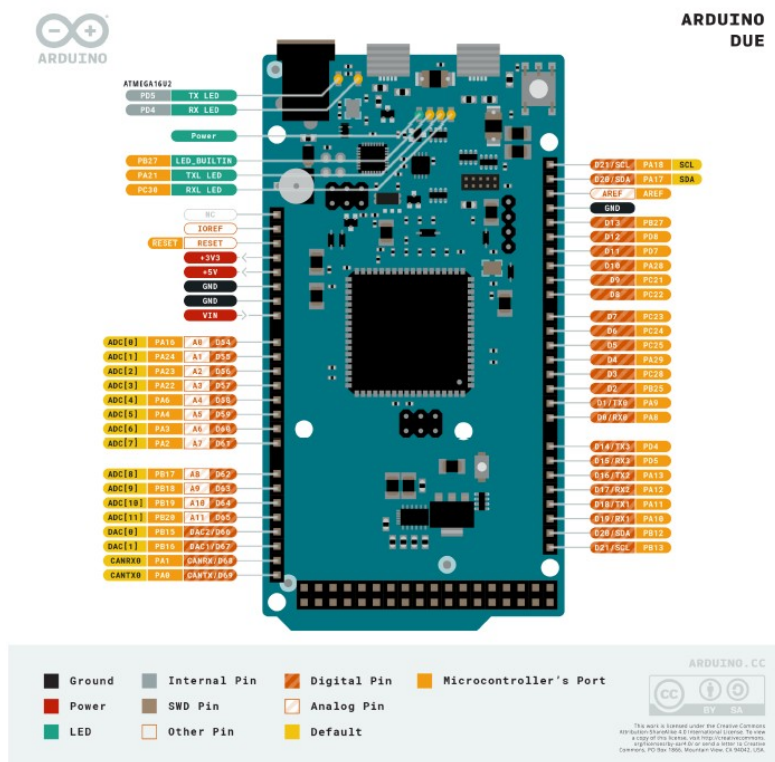


Figura 2: Pinout scheda Arduino Due

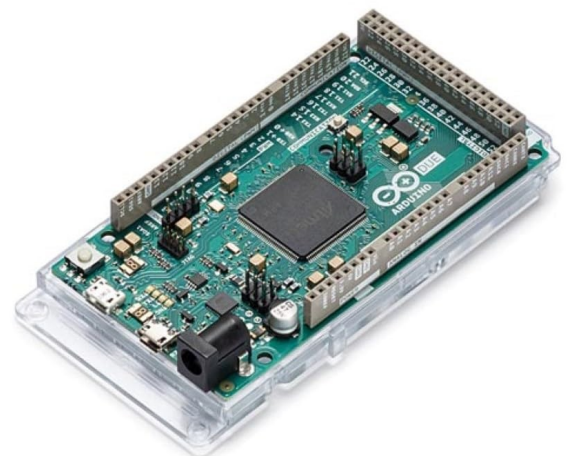


Figura 3: Arduino Due

L'Arduino Nano 33 BLE è dotato del microcontrollore nRF52840, di un modulo Bluetooth Low Energy 5 da 2,4 GHz che ci consentirà di connettersi all'app Flutter inviare i dati, di un IMU a 9 assi che non utilizzeremo. È inoltre dotato di 14 pin digitali, tra cui 5 di essi utilizzabili anche come PWM e 8 pin analogici. Può essere alimentato tramite cavo micro-USB a 5V oppure tramite il pin dedicato 5V. È importante tenere in considerazione che la scheda lavora con una tensione in input di 3,3V, non è quindi possibile utilizzare 5V negli altri pin.

In laboratorio è stata effettuata un'operazione di saldatura, come nella figura sottostante, in modo tale da poter utilizzare l'Arduino Nano così da avere la certezza che i collegamenti non possano generare falsi contatti.

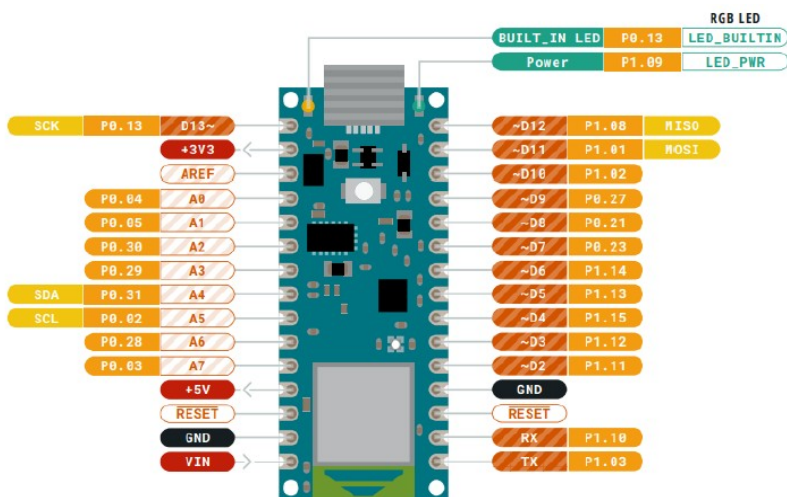


Figura 4: Pinout Arduino Nano 33 BLE

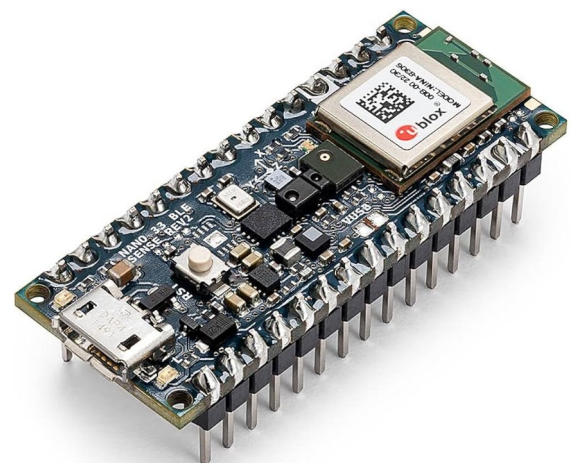


Figura 5: Arduino Nano 33 BLE

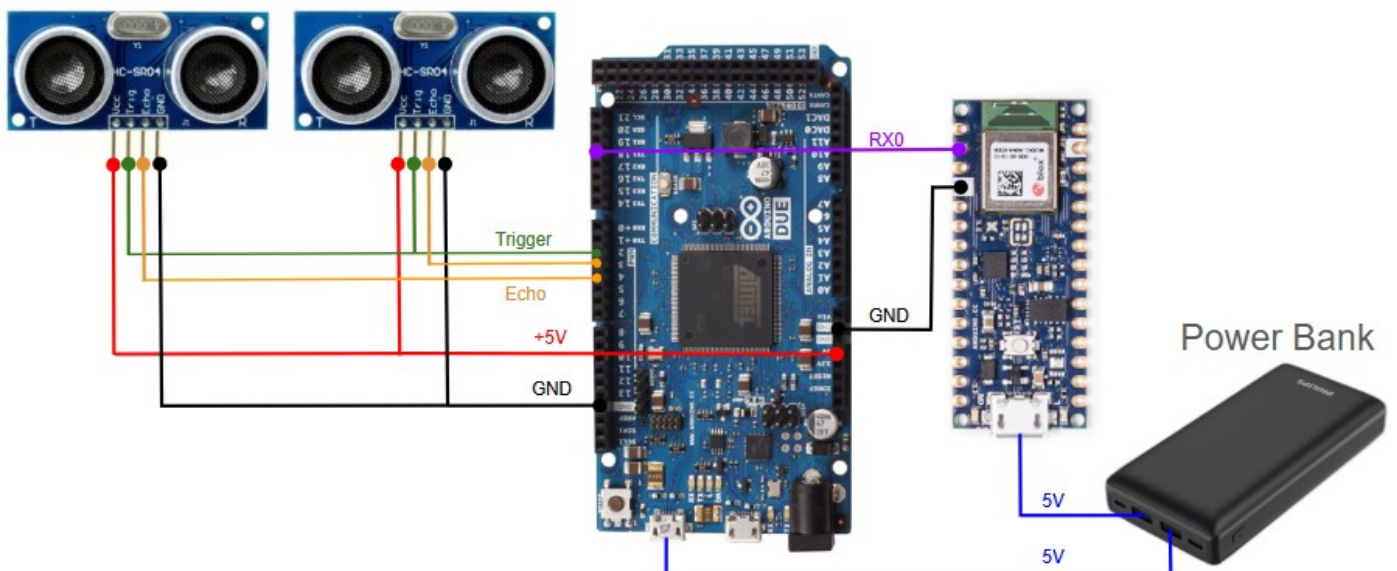


Figura 6: Collegamenti hardware

2.2 Sensori di prossimità

Per poter misurare la distanza degli ostacoli da più punti possibili della macchinina radiocomandata ho utilizzato 8 sensori a ultrasuoni HC-SR04 [5], sono dotati di 4 pin, in ordine: Vcc, Trigger (input), Echo (output) e GND. Il pin Vcc deve essere alimentato da 3 a 5,5V e il sensore può misurare le distanze da 2cm fino a 400cm con un'accuratezza di 3mm e un angolo di misurazione che può arrivare fino a 15°.

Il processo di misurazione è abbastanza immediato, bisogna inviare al pin Trigger un segnale alto per 10µs in modo tale da inizializzarlo e fargli inviare un'onda a ultrasuoni, una volta che si è scontrata con l'ostacolo ed è tornata a destinazione il pin Echo manderà in output uno stato alto pari all'ammontare di tempo che il segnale ha impiegato per tornare indietro al sensore. Si possono utilizzare due formule per calcolare la distanza finale in cm, quella approssimata: **Tempo Echo pin alto / 58** oppure **Tempo Echo pin alto * velocità del suono (340 m/s) / 2**. Nella seconda formula bisogna dividere per due perché viene anche considerato il tempo che ci mette il segnale per arrivare all'ostacolo.



Figura 7: Sensore a ultrasuoni HC-SR04



Figura 8: Esempio pratico di rilevazione di un ostacolo

2.3 Codice Arduino

Per la programmazione delle schede ho utilizzato l'IDE (Integrated Development Environment) di Arduino [6], offre un'interfaccia grafica molto user friendly e mi ha consentito di effettuare il debug del codice dei due Arduino in contemporanea in modo semplice e immediato.

Partiamo dal codice scritto per il funzionamento dell'Arduino Due: nella parte iniziale della porzione sottostante ho inizializzato le costanti che si occupano dei pin I/O in modo tale da raggiungere un grado di configurabilità maggiore, anche la velocità del suono è facoltativamente modificabile nel caso in cui si dovesse operare in un ambiente molto caldo o molto freddo. TriggerDelay indica in microsecondi l'intervallo di tempo che deve passare tra una rilevazione di un sensore e quello successivo. Dopo aver effettuato vari test in laboratorio abbiamo stabilito che per poter ottenere una maggiore precisione delle misurazioni bisognava impostare il valore minimo rilevato dal sensore a 200 cm, ovvero quando non rileva alcun oggetto perché troppo distante, stessa cosa riguardo al valore massimo. DelayMicroseconds fa aspettare l'Arduino un certo quantitativo di tempo in millisecondi dopo aver rilevato le distanze da tutti i sensori.

```
#define triggerPin 2
#define echoPinStart 3
#define echoPinCount 8
#define triggerDelay 100000

// In the air, at 20° C, in m/s
#define soundSpeed 343.1

// Duration is in ms, distance is in centimeters
long durations[echoPinCount], distances[echoPinCount];

// Sound speed in cm/μs. It's divided by 2 because we only need the return distance
const double soundSpeedCmPerUs = (soundSpeed / 10000) / 2;

// In milliseconds
const int delayBetweenMeasures = 100, zeroDistanceValue = 200, maxDistanceValue = 200;
```

Figura 9: Definizione delle costanti

Il setup si occupa invece di inizializzare la comunicazione seriale tramite USB per permettere alla scheda di comunicare con l'IDE di Arduino, oltre a quella per la comunicazione con l'Arduino Nano 33 BLE. Dopodiché imposta i pin indicando quali sono adibiti per l'input e quali per l'output, mettendo pure il led built-in a uno stato basso in modo tale da spegnerlo nel caso fosse rimasto acceso.


```

void setup()
{
    // Initialize with baud rate of 9600
    SerialUSB.begin(9600);
    Serial1.begin(57600);

    // Set the built in LED pin as output
    pinMode(LED_BUILTIN, OUTPUT);
    pinMode(triggerPin, OUTPUT);

    // Switch off LED pin
    digitalWrite(LED_BUILTIN, LOW);

    // Sets all echopins
    for (int i = echoPinStart; i < echoPinStart + echoPinCount; i++)
    {
        pinMode(i, INPUT);
    }
}

```

Figura 10: Setup iniziale

Il loop ha invece il compito di inviare il segnale di trigger mediante l'apposita funzione SendTrigger e leggere il segnale di risposta (echo) di ogni singolo sensore, in sequenza, calcolare quindi per quanto tempo il segnale di echo ha assunto un livello alto e ottenere la distanza rilevata tramite la moltiplicazione con la velocità del suono in cm/μs. Ogni misura viene separata da una virgola e accodata a una stringa, per fare in modo che una volta completato il ciclo, possa essere mandata tramite comunicazione seriale all'Arduino Nano 33 BLE.

```

void sendTrigger(int duration, int pinToWrite)
{
    digitalWrite(pinToWrite, LOW);
    delayMicroseconds(2);

    digitalWrite(pinToWrite, HIGH);
    delayMicroseconds(duration);
    digitalWrite(pinToWrite, LOW);
}

```

Figura 11: Funzione di trigger

```

void loop()
{
    String sensorDistances = "";

    for (int i = echoPinStart; i < echoPinStart + echoPinCount; i++)
    {
        int index = i - echoPinStart;

        sendTrigger(10, triggerPin);

        durations[index] = pulseIn(i, HIGH, 100000);

        int distance = min(soundSpeedCmPerUs * durations[index], maxDistanceValue);

        distances[index] = distance == 0 ? zeroDistanceValue : distance;

        sensorDistances += (String)distances[index];

        if (i < echoPinStart + echoPinCount - 1)
        {
            sensorDistances += ",";
        }

        delayMicroseconds(triggerDelay);
    }

    Serial1.println(sensorDistances);
    SerialUSB.println("Sent data: " + sensorDistances);

    delay(delayBetweenMeasures);
}

```

Figura 12: Loop del programma

Anche nel codice dell'Arduino Nano 33 BLE ho inizializzato una serie di costanti, dedite alla definizione del numero di sensori, se la scheda possa o meno trasmettere, i vari UUID (Universally Unique Identifier) associati ai servizi forniti dalla connessione Bluetooth Low Energy e alcune costanti utili per effettuare il debug senza bisogno della comunicazione seriale con Arduino Due.

```

#include <ArduinoBLE.h>
#include <time.h>
#include <stdlib.h>

int sensorsCount = 8;
int canTransmitData = 0;
int maxRandomValue = 100;

const char* localName = "Arduino 33 IoT";
const char* serviceUUID = "8e7c2dae-0000-4b0d-b516-f525649c49ca";
const char* switchModeUUID = "8e7c2dae-0001-4b0d-b516-f525649c49ca";
const char* sensorUUID = "8e7c2dae-0002-4b0d-b516-f525649c49ca";

bool isDebugEnabled = false;
bool isFirstString = true;

BLEByteCharacteristic switchModeCharacteristic(switchModeUUID, BLEWrite | BLERead );
BLECharacteristic sensorsCharacteristic(sensorUUID, BLENotify, sensorsCount * 2);

BLEService sensorsService(serviceUUID);

```

Figura 13: Inizializzazione costanti Arduino Nano

Il setup anche in questo caso si occupa dell'inizializzazione seriale, del pin built-in che servirà durante la trasmissione dati e della configurazione dei servizi Bluetooth per l'attivazione e l'esecuzione dell'invio delle distanze.

```

// Set advertised local name and service UUID:
BLE.setLocalName(localName);
BLE.setAdvertisedService(sensorsService);

// Add the characteristics to the service
sensorsService.addCharacteristic(switchModeCharacteristic);
sensorsService.addCharacteristic(sensorsCharacteristic);

// Add the service
BLE.addService(sensorsService);

// Set the initial value for the characteristics
switchModeCharacteristic.writeValue(0);
sensorsCharacteristic.writeValue((uint8_t)0, false);

// Start advertising
BLE.advertise();

```

Figura 14: Inizializzazione servizi Bluetooth

Il loop gestisce invece la connessione tramite Bluetooth ascoltando ciclicamente se delle periferiche si vogliono connettere. Una volta stabilito il collegamento con l'app Flutter la scheda attende un'ulteriore conferma per iniziare a inviare i dati dei sensori. Una volta ricevuta questa ulteriore validazione, il dispositivo comincia ad ascoltare la linea seriale per raccogliere i dati che gli arrivano dall'Arduino Due, qui è stato inserito un sistema che permette lo scarto automatico di dati corrotti. Le distanze ricevute vengono convertite da una stringa a un array di Int16, dopodiché questa array viene convertita in un'ulteriore array di Byte e inviata tramite Bluetooth all'app Flutter.

```

// Serial communication
while (Serial1.available() >= 0 && canTransmitData == 1)
{
    // Read the data in the RX pin
    String receivedString = isDebugEnabled ? getRandomSensorsValueAsString(maxRandomValue, sensorsCount) : Serial1.readString();

    // Discard the first received string since it might be corrupted or invalid
    if (isFirstString)
    {
        isFirstString = false;
        break;
    }

    // Check if there's valid data available
    if (receivedString == "")
        break;

    Serial.println("Received data: " + receivedString);

    int16_t* values = splitStringToInt16Array(receivedString, ',', sensorsCount);
    byte* valuesInBytes = convertFromInt16ArrayToByteArray(values, sensorsCount);

    // Write the array of bytes of the sensor values
    sensorsCharacteristic.writeValue(valuesInBytes, sensorsCount * 2);

    // Free the allocated memory
    delete[] values, valuesInBytes;

    if (isDebugEnabled)
        delay(500);
}

```

Figura 15: Ricezione seriale, conversione e invio dati tramite Bluetooth

2.4 Applicazione Flutter

L'applicazione è stata sviluppata attraverso Flutter [7]: un framework open-source creato da Google adibito alla creazione di interfacce native multiplatforma quali iOS, Android, Linux, MacOS e Windows, si può inoltre utilizzare per lo sviluppo di web apps. Il linguaggio di programmazione utilizzato è Dart [8], anch'esso open-source e sviluppato da Google con lo scopo iniziale di sostituire il JavaScript per lo sviluppo web, il compilatore permette di scrivere programmi web e desktop.

Per la realizzazione dell'app ho utilizzato un editor di codice sorgente open-source molto leggero ma al contempo molto potente, chiamato Visual Studio Code [9], sviluppato da Microsoft. Il mio compito è stato modificare e ampliare le funzionalità di un'applicazione base già funzionante, in grado di connettersi tramite Bluetooth ad Arduino e al server di Measurify. Tra le feature inserite ho leggermente migliorato la pagina iniziale aggiungendo una barra del segnale dinamica che permette di capire quanto sia distante la periferica a cui ci si vuole connettere, oltre ad avere aggiunto un controllo per fare in modo che il GPS debba essere attivato assieme al Bluetooth, altrimenti nelle nuove versioni di Android non è possibile rilevare alcun device.

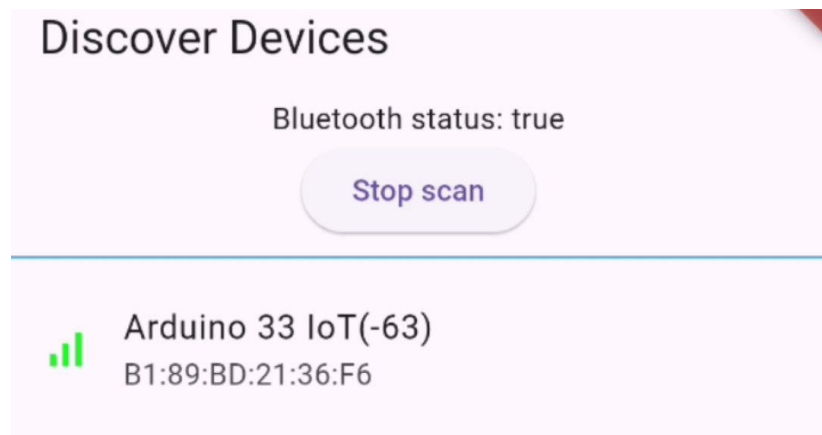


Figura 16: Schermata iniziale

Dopodiché ho migliorato la seconda schermata facendo gestire la connessione a uno solo dei due pulsanti che erano presenti (Connect e Disconnect) in modo tale da snellire l'interfaccia grafica.

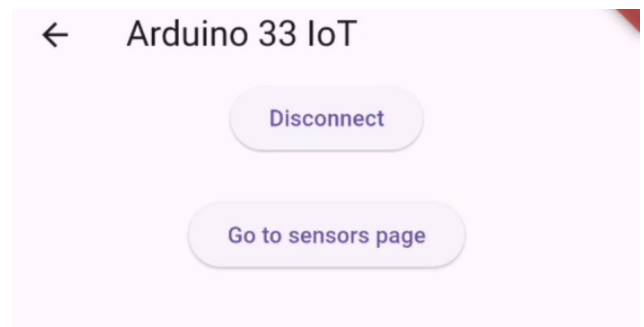
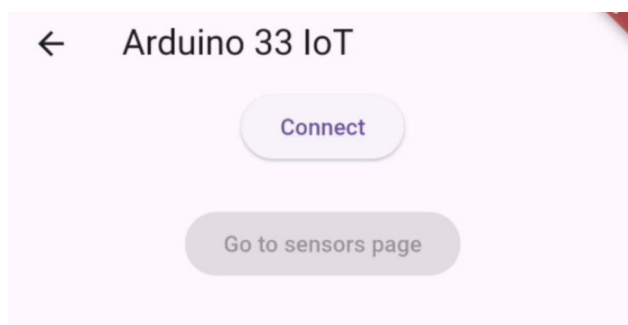


Figure 17 e 18: Connessione e disconnessione dal device

La terza schermata è quella che ho creato da zero, presenta la visualizzazione grafica degli 8 sensori di prossimità, che cambiano dinamicamente in base alla distanza rilevata: fino a 10 cm la barra sarà rossa, da 10 a 30 cm arancione, da 30 a 60 cm gialla e infine dopo i 60 cm verrà mostrata come verde.

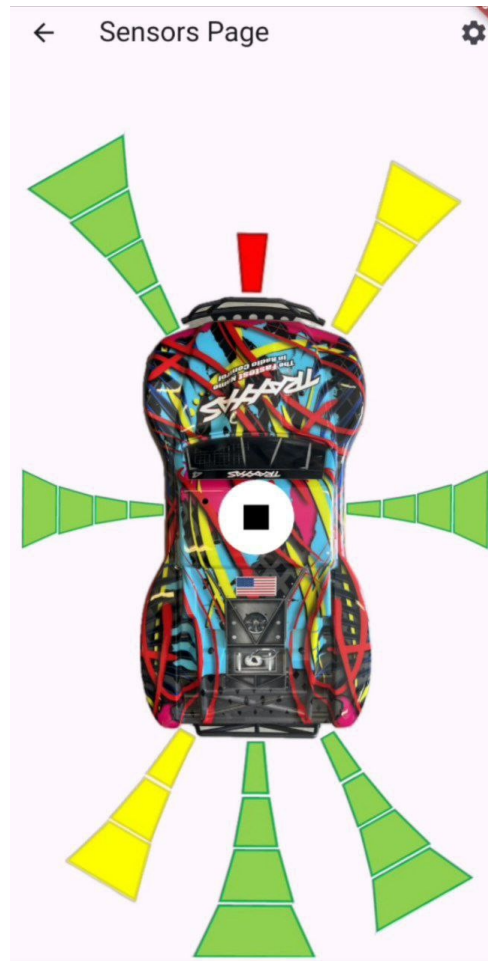


Figura 19: Visualizzazione grafica delle distanze

La parte di codice che si occupa della ricezione delle distanze le memorizza in una mappa costituita da un numero intero come chiavi (il timestamp della ricezione) e una lista di classi `SensorData` come valori. La classe `SensorData` è costituita da due campi: `distance` e `duration`, il secondo non ci serve ma è stato inserito per una maggiore estensibilità dell'app.


```

// When a characteristic change he read the value and decode it and save into different variables
void _handleValueChange(
  String deviceId, String characteristicId, Uint8List value) {
  // Check if the characteristicId is the correct one and if the data count equals to twice the number of sensors
  if (characteristicId == widget.globals.sensorsCharacteristicId &&
    value.length == sensorsCount * 2) {
    final byteData = ByteData.view(value.buffer);
    int timestamp = DateTime.now().millisecondsSinceEpoch;

    for (int i = 0; i < sensorsCount; i++) {
      if (i < value.lengthInBytes ~/ 2) {
        int distance = byteData.getInt16(i * 2, Endian.little);
        SensorData sensorData = SensorData(distance, 0);

        sensorsDataMap.update(
          timestamp, (sensors) => [...sensors, sensorData],
          ifAbsent: () => [sensorData]);
      }
    }
  }
}

```

Figura 20: Codice Dart ricezione dati

Successivamente ho implementato un algoritmo in grado di raccogliere in input un quantitativo configurabile di distanze e restituire in output 8 valori dei sensori che risultano più attendibili dei singoli, dato che ogni tanto potrebbe essere presente qualche misura errata causata dall'imprecisione dei singoli sensori. Una volta raccolta una certa quantità di dati, anch'essa configurabile, l'app li invierà automaticamente al server cloud di Measurify mediante una richiesta POST, è necessario inserire un token di autenticazione per fare in modo che le informazioni inviate vengano registrate nel database associato al proprio account. La quarta e ultima schermata consente di modificare le impostazioni dell'app senza doverla ricompilare e riapirla, molte di esse erano già presenti, il mio compito è stato quello di aggiungere tutti i settings che ho trovato rilevanti per la raccolta e visualizzazione dei sensori a ultrasuoni.

← Config Page

URL Measurify
https://tracker.elioslab.net/v1/ [Reset Settings](#)

ID Tenant
distance-sensors

Token Device

ID BLE service
8e7c2dae-0000-4b0d-b516-f525649c49ca

Collecting Count
10

Average Measures Count
5

Thing Name
car1

Feature Name
distance

Figura 21: Pagina dei settings

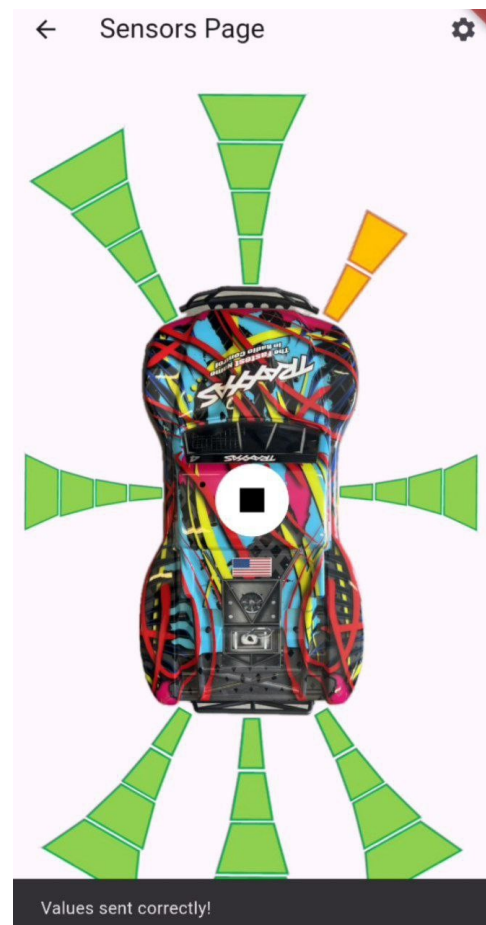


Figura 22: Invio dei dati riuscito

2.5 Measurify

Per la memorizzazione dei dati raccolti ho utilizzato l'API (Application Programming Interface) framework Measurify [10], open-source e RESTful [11], usata per la gestione dei dispositivi IoT. È stata sviluppata dall'Elios Lab dell'Università di Genova, basata sul cloud e dedicata alla raccolta e l'elaborazione di dati.

Measurify si basa su alcuni elementi chiave:

- **Thing:** una persona, oggetto o cosa per la quale si sta effettuando una misurazione;
- **Feature:** indica quello che si sta misurando;
- **Device:** un dispositivo utilizzato per misurare una certa feature su una specifica thing;
- **Tags:** etichette da associare a una determinata misurazione, per poterle filtrare meglio;
- **Measurement:** è una misura effettuata da un device, per una certa feature su una determinata thing, come ad esempio la distanza di un sensore misurata in uno specifico istante di tempo.

Per memorizzare i dati raccolti dai sensori sul server di Measurify bisogna prima creare una measurement effettuando una richiesta POST all'URL <https://tracker.elioslab.net/v1/measurements>, nell'header è necessario inserire un campo "Authorization" contenente il proprio token, in modo tale da non dover effettuare il login tramite username e password e un campo.

Il body dovrà essere in formato JSON ed essere strutturato in questo modo:

```
{
  "_id": "seriesensori",
  "thing": "car1",
  "feature": "distance",
  "device": "sensors",
  "startDate": "2025-05-01T00:00:00.000Z",
  "endDate": "2025-05-01T00:00:00.000Z"
}
```

Una volta creata la measurement, bisognerà memorizzare i dati effettivi raccolti, per farlo sarà sufficiente effettuare una richiesta POST al seguente URL <https://tracker.elioslab.net/v1/measurements/seriesensori/timeserie> sempre con l'header contenente l'Authentication e un body nel seguente formato:

```
{
  "timestamp": "1684833177652.00",
  "values": [0.1, 0.2, 0.5, 20.123, 60.987, 100.456, 1, 2]
}
```

Dove timestamp indica la data da memorizzare in formato unix, mentre i valori le 8 misurazioni.










thing	feature	device	startDate	tags	_id	Management
car1	distance	sensors	2025-05-01T11:20:47.364Z	[]	seriesensoritest	  
car1	distance	sensors	2025-05-01T10:22:28.540Z	[]	661fac1a9cf5eb1cef7f9732755d1ef7460ad558e60154fce7201b22a38d54a0	  
car1	distance	sensors	2025-03-10T14:50:26.708Z	[]	seriesensori	  

Figura 23: Interfaccia grafica gestione IoT su Measurify

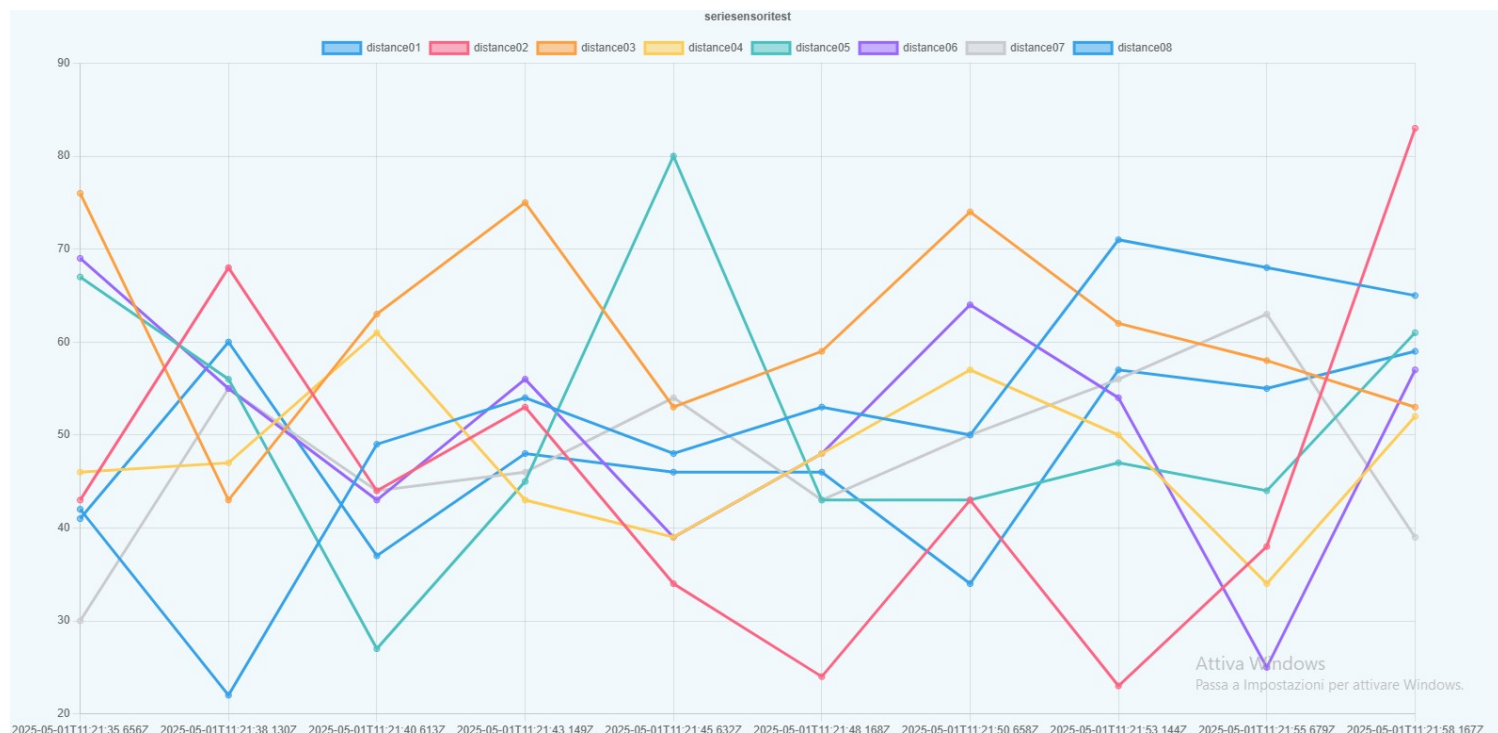


Figura 24: Grafico sul dominio del tempo delle distanze rilevate

3 Sperimentazione e risultati

Ho iniziato la sperimentazione testando tre sensori a ultrasuoni su una breadboard e mano a mano adattando il codice e il distanziamento tra loro per fare in modo che potessero rilevare le misurazioni più affidabili possibili. Successivamente, nel laboratorio Elaios Lab, ho avuto la possibilità di farmi saldare dei connettori pin maschio sull'Arduino Nano 33 BLE in modo tale da poterlo utilizzare senza dovermi preoccupare di eventuali collegamenti fallaci, ho quindi testato tutti e 8 i sensori in contemporanea sempre montati su breadboard e distanziati come se fossero stati piazzati sulla macchinina radiocomandata. Dopo svariati test abbiamo notato che i sensori che non rilevavano alcuna distanza, perché fuori range, rallentavano il resto delle misurazioni, ciò è stato risolto riducendo il tempo di lettura sui pin di echo a un valore accettabile

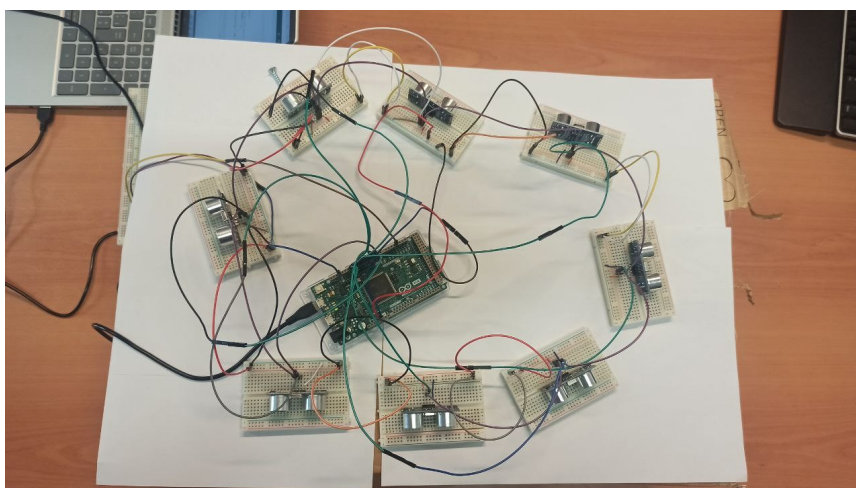


Figura 25: Visualizzazione dall'alto del circuito montato

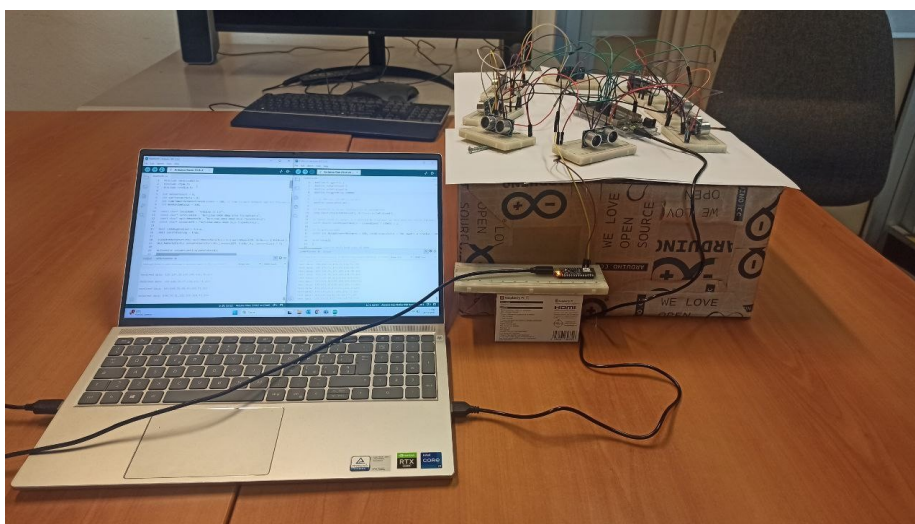


Figura 26: Visuale laterale del progetto completo

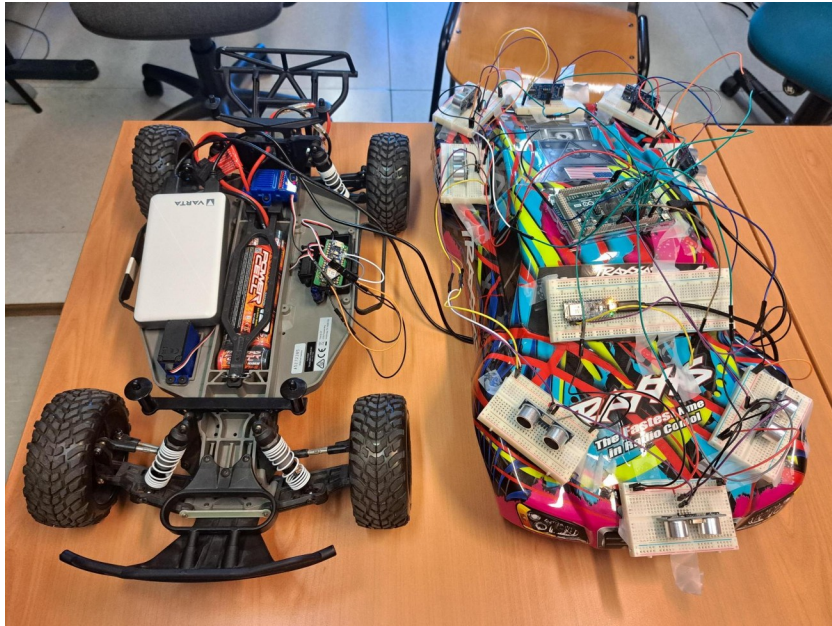


Figura 27: Visuale dall'alto del montaggio su macchinina

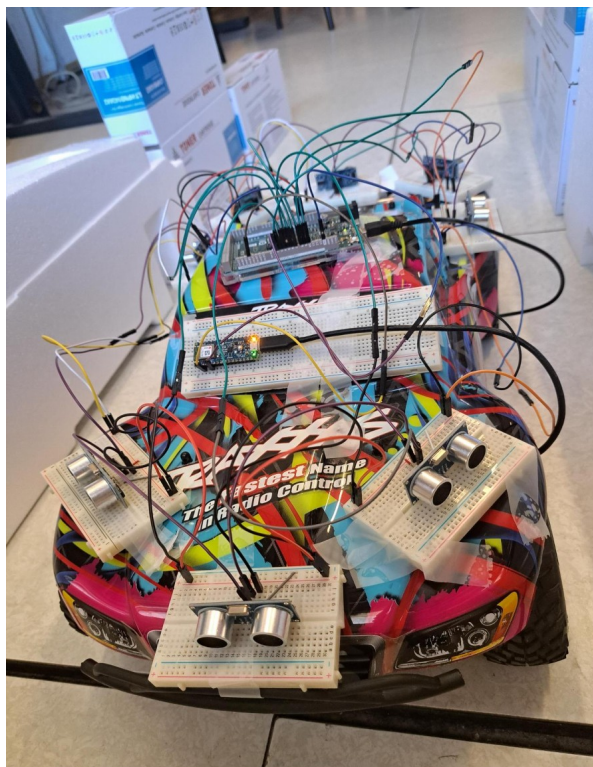


Figura 28: Macchinina all'interno del percorso a ostacoli

4 Contributo personale e considerazioni conclusive

L'idea del progetto nasce dal voler creare un sistema di guida assistita tramite sensori di prossimità come quello presente nelle auto moderne, in questo caso applicato a una macchinina radiocomandata, l'utente potrà quindi guidarla con più sicurezza tramite l'app grazie alla visualizzazione grafica fornita. Ho creato da zero i programmi dei due Arduino, dopo essermi documentato tramite i datasheet dei sensori, creato un circuito non troppo complesso tramite la scelta ottimale dei pin di Arduino e utilizzando una libreria specifica per la connessione Bluetooth chiamata ArduinoBLE. La sfida più grande è stata imparare un nuovo linguaggio di programmazione: Dart, con esso tramite il framework Flutter ho ampliato le funzionalità dell'applicazione aggiungendo delle migliori e una nuova schermata dedicata a questo progetto, il numero dei sensori come molti altri parametri è configurabile. Dovendo utilizzare una libreria un po' datata per la connessione Bluetooth ho dovuto scaricare delle versioni specifiche di altre librerie che ho utilizzato per fare in modo che non andassero in conflitto.

In futuro sarà possibile ampliare le funzionalità dell'applicazione o aumentare il numero dei sensori all'interno della macchinina radiocomandata per avere ancora più controllo sulla guida, sarà ad esempio possibile implementare un sistema di guida tramite intelligenza artificiale che sfrutti i sensori a ultrasuoni e si adatti in base alle distanze ricevute.

5. Riferimenti Bibliografici

- [1] <https://www.sap.com/italy/products/technology-platform/what-is-iot.html>
- [2] <https://docs.flutter.dev/>
- [3] <https://docs.arduino.cc/hardware/due/>
- [4] <https://docs.arduino.cc/hardware/nano-33-ble/>
- [5] <https://components101.com/sensors/ultrasonic-sensor-working-pinout-datasheet>
- [6] https://it.wikipedia.org/wiki/Arduino_IDE
- [7] [https://it.wikipedia.org/wiki/Flutter_\(software\)](https://it.wikipedia.org/wiki/Flutter_(software))
- [8] [https://it.wikipedia.org/wiki/Dart_\(linguaggio_di_programmazione\)](https://it.wikipedia.org/wiki/Dart_(linguaggio_di_programmazione))
- [9] https://it.wikipedia.org/wiki/Visual_Studio_Code
- [10] <https://measurify.info/>
- [11] https://it.wikipedia.org/wiki/Representational_state_transfer

6. Ringraziamenti

Ringrazio il professor Riccardo Berta e il dottore Matteo Fresta che mi hanno guidato durante tutta la progettazione, implementazione in laboratorio e stesura della tesi.

Ringrazio la mia famiglia per avermi supportato durante tutto il percorso di studio.

Ringrazio tutti i miei amici, specialmente Arian e Sofia per avermi particolarmente motivato durante i giorni della stesura della tesi tenendomi alto il morale.