UNIVERSITÀ DEGLI STUDI
DI GENOVA

DIPARTIMENTO DI INGEGNERIA NAVALE, ELETTRICA, ELETTRONICA E DELLE TELECOMUNICAZIONI

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA E TECNOLOGIE DELL'INFORMAZIONE

Tesi di laurea triennale

Progetto e implementazione di un sistema embedded per il monitoraggio dell'orientazione spaziale

Design and development of an Embedded System for Spatial Orientation Monitoring

Relatore: Prof. Riccardo Berta

Correlatore: Dott. Matteo Fresta

Candidato:
Leonardo Giacobbe

Dicembre 2023

Sommario

Il presente elaborato è stato redatto con l'obiettivo di realizzare un dispositivo embedded che misuri e rappresenti in tempo reale l'orientazione spaziale di un modellino di fuoristrada in scala, mediante un'applicazione che realizza un inclinometro digitale e una vista tridimensionale del mezzo nello spazio.
I dati di orientazione spaziale che vengono forniti all'applicazione sono raccolti da una scheda Arduino Nano 33 BLE Sense incorporato ad una batteria che lo rende un dispositivo standalone. I dati sono trasmessi mediante Bluetooth Low Energy e vengono ricevuti da uno smartphone sul quale è installata l'applicazione scritta in Dart con l'ausilio del framework open-source Flutter. L'applicazione effettua la connessione ad Arduino e permette di scegliere tra due widget: un inclinometro digitale, che mostra il pitch e il roll agenti sulla macchinina mediante l'animazione delle viste frontale e laterale del veicolo, e una vista tridimensionale che mostra un fuoristrada che ruota nello spazio in base ai valori di yaw, pitch e roll ricevuti.
Le prospettive future del progetto sono di creare una mappa del tracciato percorso da una macchina e di fornire uno storico dei valori di inclinazione e velocità raggiunti dal modello radiocomandato.

Indice

1. Introduzione

La presente tesi ha come oggetto lo studio e la realizzazione di una applicazione per dispositivi mobili in grado di replicare virtualmente un inclinometro analogico e di fornire una vista tridimensionale animata di un modello in scala radiocomandato di un fuoristrada.

Nel mondo del modellismo radiocomandato vi sono varie tipologie di modelli in diverse scale e categorie, tra cui auto, barche, aerei, eccetera.
Il modello utilizzato per questo progetto è un fuoristrada in scala 1:10, appartenente alla categoria che viene definita “scaler”. Gli scaler sono modelli di auto da fuoristrada specificatamente progettati per affrontare salite ripide e terreni impervi e, a differenza dei crawler, presentano schemi sospensivi e telaistici che ricalcano realisticamente le soluzioni presenti sulle auto da fuoristrada reali.

La categoria degli scaler è molto richiesta sul mercato e, di conseguenza, essi vengono prodotti da innumerevoli marche, ognuna con una sua line-up di modelli e un suo target di clienti.
Considerando gli scaler 1:10 i prezzi variano da 250 euro a oltre 1000 euro nelle marche più famose e ricercate, offrendo componenti estremamente realistici realizzati con materiali più pregiati e leggeri, come ad esempio il carbonio. Alcune delle marche di più alto livello, come ad esempio Traxxas, offrono la possibilità di scaricare una applicazione per smartphone che permette di visualizzare diverse informazioni relative al proprio modellino come ad esempio: telemetria del giro, temperatura della centralina, voltaggio della batteria, numero di giri del motore, velocità istantanea e persino un inclinometro virtuale.
Da qui nasce l’idea di realizzare uno strumento simile senza dovere acquistare un modello notevolmente più costoso mantenendo le stesse funzionalità fornite da esso.

Per questo progetto è stato utilizzato un Vrx Varanus in scala 1:10, una scheda Arduino 33 BLE Sense per l’acquisizione dei dati di orientazione spaziale e uno smartphone per eseguire l’applicazione che elabora i dati e li visualizza attraverso un inclinometro bidimensionale.

Fig. 1.1: sintesi delle funzioni e relativa realizzazione del progetto

Come mostrato in figura 1.1, la struttura del progetto si basa sull'acquisizione, trasmissione, elaborazione e visualizzazione dei dati raccolti sul dispositivo mobile.

Il primo passo consiste nel montare l'Arduino sulla scocca della macchina, il quale si occuperà di acquisire i dati riguardanti l'orientazione e, dopo averli elaborati, li invierà tramite Bluetooth Low Energy allo smartphone. L'applicazione installata sullo smartphone gestisce la connessione Bluetooth Low Energy con l'Arduino e offre la possibilità di visualizzare due pagine differenti: la prima consiste nella virtualizzazione di un classico inclinometro analogico per fuoristrada nel quale sono rappresentate l'inclinazione frontale (pitch) e l'inclinazione laterale (roll), mentre la seconda fornisce una vista tridimensionale del mezzo nello spazio, animato in base ai dati di orientazione yaw, pitch e roll calcolati da Arduino.
La parte hardware del progetto è costituita esclusivamente dalla scheda Arduino Nano 33 BLE Sense, mentre la parte software si suddivide in codice Arduino, per la calibrazione dei sensori e l'invio dei dati dalla scheda, e codice Dart con l'utilizzo del framework Flutter, per ricevere i dati e animare l'inclinometro e il modello tridimensionale.

La tesi è articolata in tre paragrafi:

- Metodi e strumenti utilizzati: qui verranno analizzati tutti gli strumenti, hardware e software, mediante i quali è stata possibile la creazione del progetto
- Sperimentazione e risultati: in questa parte verranno studiati sul campo gli esiti derivati dall'utilizzo del sistema progettato
- Contributo personale e considerazioni conclusive: qui, oltre a presentare le considerazioni finali e possibili ulteriori sviluppi futuri, verranno messe in luce le scelte fatte durante la fase progettuale e la loro motivazione.

2. Metodi e strumenti utilizzati

In questo capitolo verranno analizzati tutti gli strumenti utilizzati per portare a compimento la realizzazione del progetto. Questi ultimi, come già accennato nella sezione introduttiva, si dividono in due grandi categorie:

- Hardware
- Software

Iniziamo analizzando la scheda Arduino utilizzata.

2.1 Arduino Nano 33 Ble Sense

Per la raccolta dei dati di orientazione spaziale necessari alla realizzazione del progetto è stato utilizzato un Arduino Nano 33 Ble Sense: si tratta di una scheda molto compatta che misura solamente 4,5 cm di lunghezza e 2,5 cm di larghezza e che viene fornita equipaggiata con il supporto Bluetooth Low Energy e il sensore Inertial Measurement Unit (IMU) LSM9DS1.

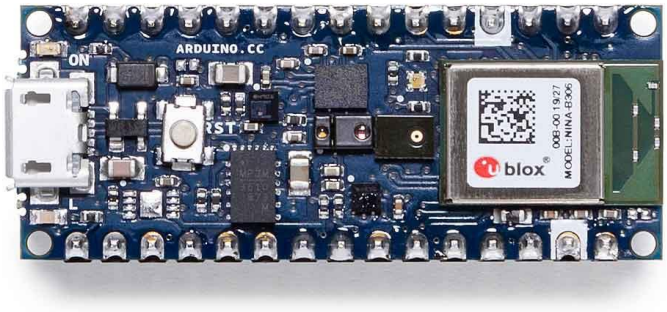

Fig. 2.1: Scheda Arduino Nano 33 BLE Sense

Questo sensore è un modulo inerziale chiamato IMU che permette di acquisire i seguenti nove valori:

- Accelerazione lineare misurata su tre assi
- Velocità angolare misurata su tre assi
- Campo magnetico misurato su tre assi

Sostanzialmente è come se le misurazioni venissero effettuate da tre sensori differenti, ossia accelerometro, giroscopio e magnetometro.
L'accelerometro misura l'accelerazione totale che agisce sul sensore, inclusa l'accelerazione statica dovuta alla gravità che viene impressa sul sensore quando è fermo.
Il giroscopio è un dispositivo che misura l'orientazione angolare di un oggetto e, a partire da questo, riesce a fornire la velocità angolare alla quale è sottoposto l'oggetto in questione.
Il magnetometro, invece, misura il campo magnetico circostante, basandosi sul campo magnetico terrestre.

Il solo accelerometro, però, consente di avere una misurazione senza drift ma molto rumorosa dato che misura tutte le forze agenti sull'oggetto, anche le più piccole. Il solo giroscopio, invece, fornisce una misura precisa ma che, con il passare del tempo, tende ad avere un drift notevole.

La soluzione, dunque, è quella di integrare tutti i dati proveniente dai tre diversi sensori in questione (accelerometro, giroscopio e magnetometro) mediante il filtro di Kalman.
Il filtro di Kalman è un algoritmo ricorsivo utilizzato per stimare lo stato di un sistema dinamico in presenza di misure affette da rumore.
Il suo funzionamento si articola in una procedura iterativa che comprende la predizione dello stato futuro basata

sul modello dinamico del sistema, il confronto con le misurazioni effettive e l'aggiornamento della stima dello stato. Questo processo tiene conto sia delle informazioni fornite dal modello del sistema sia di quelle misurate, assegnando opportuni pesi in base alla precisione stimata.
L'obiettivo finale è ottenere una stima accurata dello stato del sistema considerando le incertezze e il rumore presenti nelle misure.

2.2 Calibrazione di Arduino

Per ottenere una misurazione il più precisa possibile da questi sensori è stata necessaria una calibrazione accurata di ognuno di essi.

Sono stati necessari tre codici differenti, uno per sensore, ognuno dei quali richiedeva di posizionare la scheda lungo i diversi assi spaziali e di mantenerla ferma per un certo periodo di tempo in modo tale da fornire un array di sei valori di offset del rispettivo sensore. Questi parametri di offset sono risultati essenziali per poter compensare correttamente i piccoli errori di misura dovuti alla fabbricazione del singolo sensore e sono stati inseriti all'interno del codice finale utilizzato sulla scheda per l'acquisizione e la trasmissione dei dati di orientazione spaziale.

```
//Accelerometer setup
IMU.setAccelFS(2); //full scale setting
IMU.setAccelODR(5); //ODR: output data rate measured in Hz
IMU.setAccelOffset(-0.013457, -0.007203, -0.012840);
IMU.setAccelSlope (1.001740, 0.996517, 0.999404);
//Gyrsoscope setup
IMU.gyroUnit = DEGREEPERSECOND;
IMU.setGyroFS(3);
IMU.setGyroODR(5);
IMU.setGyroOffset (-1.489929, -0.464600, 0.043884);
IMU.setGyroSlope (1.243725, 1.144730, 1.074688);
//Mangetometer setup
IMU.setMagnetFS(0);
IMU.setMagnetODR(5);
IMU.setMagnetOffset(7.622681, 32.914429, 156.889038);
IMU.setMagnetSlope (2.323434, 1.434650, 1.358518);
```

Fig. 2.2: Inizializzazione sensori mediante dati ottenuti dalla calibrazione

Nella figura 2.2 viene mostrato l'inserimento dei dati di offset ottenuti dalla calibrazione dei sensori. Questi valori vengono utilizzati per ottenere misurazioni più stabili e precise dell'IMU da fornire poi in ingresso alle funzioni della libreria che andrà a calcolare il filtro di Kalman.

2.3 Programmazione di Arduino

Il codice caricato all'interno di Arduino è molto vasto e si suddivide principalmente in due parti:

- Prima parte relativa alla gestione dell'inizializzazione del sensore IMU e della connessione mediante Bluetooth Low Energy
- Seconda parte relativa all'acquisizione dati grezzi dai sensori e al calcolo dell'orientazione spaziale mediante filtro di Kalman

2.3.1: Bluetooth Low Energy

Il Bluetooth Low Energy (BLE), noto anche come Bluetooth Smart, è una tecnologia wireless progettata per consentire la comunicazione a corto raggio tra due dispositivi mantenendo un basso consumo energetico.

Il Bluetooth Low Energy è una versione ottimizzata nell'efficienza del Bluetooth classico ed è stato pensato per tutti i dispositivi con requisiti energetici ridotti come, ad esempio, dispositivi IoT, indossabili, sensori, dispositivi medici. Dunque, l'obiettivo principale del BLE è quello di consentire ai dispositivi alimentati a batteria di rimanere connessi e operare per lunghi periodi senza la necessità di essere ricaricati.

Il limite di invio di dati imposto dalla connessione è di 20 bytes ed è stato sufficiente ai fini della realizzazione di questo progetto in quanto i dati di orientazione spaziale vengono salvati e inviati in un formato di 3 valori float e dunque rispettano il vincolo della dimensione massima inviabile.

2.3.2: Acquisizione ed invio dati

Il microcontrollore fornisce le caratteristiche chiamate IMU (per accelerometro, giroscopio e magnetometro), di ambiente ENV (per prossimità, temperatura, umidità, pressione e luce) e di orientazione ORI (per yaw, pitch, roll).
Una caratteristica Bluetooth Low Energy rappresenta un'unità di dati identificata da un UUID (Universally Unique Identifier) che definisce il tipo di dati e il suo significato.

In particolare, in questo progetto verranno utilizzati i dati del sensore IMU (inertial measurement unit) per calcolare, mediante le capacità computazionali del microcontrollore, i valori di ORI, ossia i valori di orientazione spaziale.

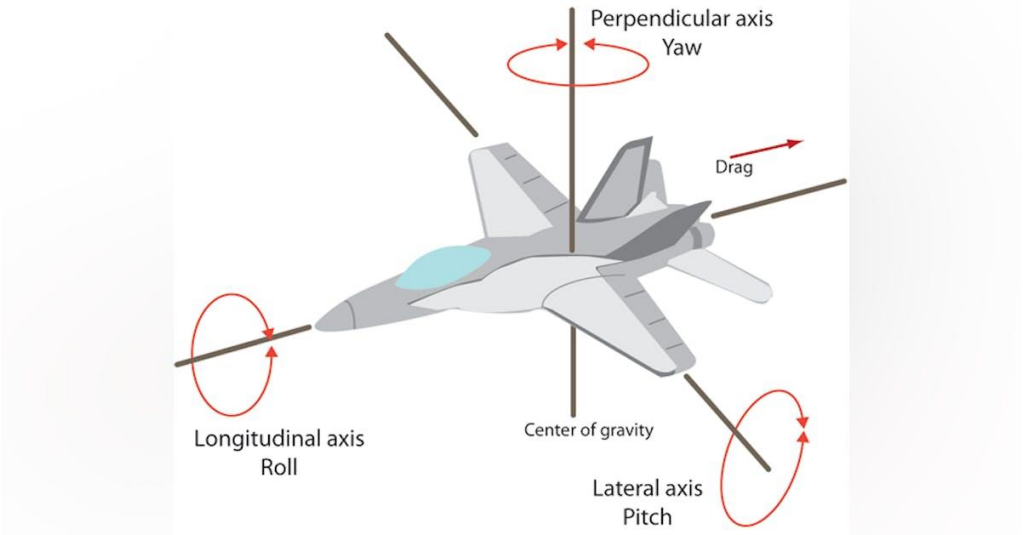

Fig. 2.3: visualizzazione grafica di yaw, pitch e roll applicati ad un caso fisico

Nella figura 2.3 si possono osservare le conseguenze delle variazioni degli angoli di yaw, pitch e roll in un caso reale. In dettaglio:

- Yaw: è la rotazione intorno all'asse verticale di un oggetto, ossia l'imbardata
- Pitch: è la rotazione lungo l'asse laterale di un oggetto, ossia il beccheggio
- Roll: è la rotazione lungo l'asse longitudinale di un oggetto, ossia il rollio.

Per ottenere questi valori di orientazione dal codice Arduino è stata utilizzata la libreria MadgwickAHRS.h. Quest'ultima implementa via software il filtro di Kalman, necessario per fondere i valori di accelerometro, giroscopio e magnetometro e ricavare i valori ORI desiderati.

```
void manageOrientation(){
  // read all 9 DOF of the IMU:
    IMU.readAcceleration(acceleration[0], acceleration[1], acceleration[2]);
    IMU.readGyro(angular_speed[0], angular_speed[1], angular_speed[2]);
    IMU.readMagneticField(magnetic_field[0], magnetic_field[1], magnetic_field[2]);
  // update the filter, which computes orientation:
  // note X and Y are swapped, X is inverted
  filter.update(angular_speed[1], angular_speed[0], angular_speed[2],
                acceleration[1], acceleration[0], acceleration[2],
                magnetic_field[1], -magnetic_field[0], magnetic_field[2]);

  heading = filter.getYawRadians();
  pitch = filter.getPitchRadians();
  roll = filter.getRollRadians();
```

Fig. 2.4: acquisizione yaw, pitch e roll

Nella figura 2.4 si può apprezzare quanto descritto prima: i dati IMU vengono acquisiti mediante le funzioni IMU.readAcceleration, IMU.readGyro e IMU.readMagneticField e, successivamente, vengono passati come parametri alla funzione filter.update che si occupa di realizzare il filtro di Kalman. I valori di yaw, pitch e roll sono dunque ora ricavabili chiamando le funzioni filter.getYawRadians, filter.getPitchRadians e filter.getRollRadians che sono sempre funzioni della libreria MadgwickAHRS. Come è facilmente intuibile dal nome della funzioni, i valori ORI vengono forniti in radianti.

Il segno meno prima del secondo parametro relativo al campo magnetico e i valori invertiti per i parametri x e y di velocità angolare e accelerazione sono dovuti all'orientazione dei sensori che, di fabbrica, non coincide con gli assi previsti all'interno della libreria MadgwickAHRS.

Successivamente, se vi è un dispositivo connesso e li richiede, i valori ORI vengono inviati mediante Bluetooth Low Energy al destinatario.

2.4: Flutter

Per realizzare l'applicativo su smartphone per ricevere i valori ORI e riprodurre digitalmente un inclinometro analogico si è scelto di utilizzare Dart in combinazione con il framework Flutter.

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 realizzate attraverso l'ausilio del framework Flutter.
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 funzionalità.
Utilizzandolo in combinazione con Flutter si possono sviluppare applicazioni client side, in particolare per dispositivi mobili, web e server.
Il grande vantaggio di Flutter è la possibilità di creare app native per dispositivi mobili (iOS e Android) e per il web, il tutto utilizzando un singolo codice sorgente.

2.5: Programmazione applicazione mobile

Si è deciso dunque di creare l'applicazione mobile mediante Dart e Flutter.

L'applicazione si suddivide in due parti principali:

- Acquisizione dati mediante connessione Bluetooth da Arduino
- Visualizzazione dati acquisiti

Per quanto riguarda la visualizzazione dei dati si compone di due pagine differenti:

- Inclinometro bidimensionale: esso riproduce fedelmente un inclinometro analogico reale che mostra l'inclinazione frontale (pitch) e laterale (roll) della vettura
- Vista tridimensionale: un modello tridimensionale animato che viene ruotato nello spazio in base ai valori di orientazione ricevuti.

2.5.1: Acquisizione dati

Per la gestione della connessione ai dispositivi bluetooth e, di conseguenza, per la ricezione dei dati è stata utilizzata la libreria quick_blue. Si tratta di un plugin bluetooth multipiattaforma e può dunque funzionare, come d'altronde il framework Flutter, con dispositivi Android, iOS, macOS e Windows.
La libreria fornisce alcune funzioni di base, come ad esempio, la scansione di dispositivi, la connessione con loro e l'avviso di dati disponibili.

Le funzioni della libreria utilizzate all'interno del progetto sono le seguenti:

- startScan/stopScan: questa funzione serve a iniziare e fermare la scansione di dispositivi Bluetooth ai quali connettersi
- connect/disconnect: serve per connettersi o disconnettersi da un dispositivo
- setNotifiable: serve a notificare la presenza di nuovi dati resi disponibili da un dispositivo
- readValue: questa funzione ha lo scopo di leggere i valori resi disponibili da un dato dispositivo.

Inoltre, sono state scritte anche alcune altre funzioni sempre relative alla gestione dei dati provenienti dal dispositivo, che, in questo caso, è Arduino nano 33 BLE sense. Di queste, le funzioni utilizzate all'interno del mio progetto sono:

- parseORIData: si occupa di ricevere i dati ORI provenienti da Arduino e riempie un array di tre valori. Nella prima posizione dell'array viene inserito lo yaw, nella seconda il pitch e nella terza il roll.
- _handleValueChange: quando una caratteristica dell'Arduino cambia, ossia quando vengono resi disponibili nuovi valori ORI, IMU o ENV, essa si occupa di leggere i nuovi valori e decodificarli per poi salvarli all'interno di specifiche variabili.

```
//When a characteristic change he read the value and decode it and save into different variables
void _handleValueChange(
    String deviceId, String characteristicId, Uint8List value) {
  if (this.mounted) {
    if (characteristicId == "8e7c2dae-0004-4b0d-b516-f525649c49ca") {
      //ORI
      Map<String, dynamic> jsonObj = parseORIData(value, 3);
      double pitch = jsonObj["values"][1];
      double roll = jsonObj["values"][2];
      setState(() {
        _rz1 = roll;
        _rz2 = -pitch;
      });
    }
```

Fig. 2.5: funzione handleValueChange per la gestione del cambiamento dei valori provenienti da Arduino

In figura 2.5 si può osservare l'implementazione della funzione _handleValueChange descritta precedentemente.

In particolare, questa è la sua implementazione all'interno del widget dell'inclinometro bidimensionale.
Si può infatti osservare come all'interno di questa funzione vengano associati il pitch e roll a due variabili di tipo double e come queste ultime vengano poi assegnate all'interno del setState ai valori di rotazione da conferire alle due viste bidimensionali della macchina.

```dart
Map<String, dynamic> parseORIData(Uint8List value, int ArrayLength) {
  final byteData = ByteData.view(value.buffer);
  List<double> oriValues = List<double>.filled(3, 0.0);
  oriValues[0] = byteData.getFloat32(0, Endian.little);
  oriValues[1] = byteData.getFloat32(4, Endian.little);
  oriValues[2] = byteData.getFloat32(8, Endian.little);
  print(oriValues);

  Map<String, dynamic> jsonObj = {
    "timestamp": DateTime.now().millisecondsSinceEpoch,
    "values": oriValues,
  };
  return jsonObj;
}
```

Fig. 2.6: implementazione della funzione parseORIData per l'acquisizione dei valori di yaw, pitch e roll rispettivamente

Nella figura 2.6 si può osservare l'implementazione della funzione parseORIData: viene creata una lista di array di double nei quali vengono poi inseriti rispettivamente i valori di yaw, pitch e roll.
Inoltre, viene anche associato un timestamp ai valori, ma questa funzionalità non viene utilizzata all'interno del progetto.

2.5.2: Realizzazione widget inclinometro bidimensionale

Per la realizzazione dell'inclinometro bidimensionale sono stati definiti quadrati bidimensionali orientati nello spazio tridimensionale.
Questi ultimi sono stati realizzati mediante la definizione di due container quadrati di dimensioni 200 pixel per 200 pixel.
In uno di questi container è stata inserita come sfondo l'immagine della vista frontale della macchina, opportunamente ritagliata mediante l'utilizzo di Adobe Photoshop, mentre nel secondo container è stata inserita l'immagine della vista laterale della macchina.

La vista frontale dell'auto verrà animata mediante una rotazione corrispondente al valore di roll ricevuto, mentre la vista laterale dell'auto subirà una rotazione corrispondente al valore di pitch ricevuto.

Tutta l'animazione dell'inclinometro viene gestita da un bottone di start che, se premuto, permette di avviare la rotazione di entrambe le viste.
Il bottone permette di avviare o fermare la rotazione delle immagini.

```dart
@override
Widget build(BuildContext context) {
  return Stack(
    children: [
      // FRONT
      Transform(
        transform: Matrix4.identity()..translate(0.0, 0.0, -100.0),
        child: Container(
          color: Colors.white,
          child: Image.asset(
            'assets/front_square_200px.png',
            width: 200,
            height: 200,
          ), // Image.asset
        ), // Container
      ), // Transform
```

Fig. 2.7: realizzazione dell'indicatore dell'inclinometro raffigurante la vista frontale del veicolo

Nella figura 2.7 si può osservare quanto descritto precedentemente circa l'implementazione dell'indicatore della vista frontale dell'inclinometro. Viene creato innanzitutto un container al quale viene sovrapposta l'immagine della vista frontale del modellino di uguali dimensioni al container, ossia 200 pixel per 200 pixel.

Fig. 2.8: widget dell'inclinometro bidimensionale

In figura 2.8 si può osservare il widget descritto in precedenza.
Esso consiste nelle due viste frontale e laterale della macchina animate rispettivamente dai valori di roll e pitch ricevuti. L'animazione inizia nel momento in cui viene premuto il bottone con su scritto start e, una volta premuto, esso cambia il testo in stop. Come è facile intuire, se il bottone di stop viene premuto l'animazione si ferma e le immagini tornano all'inclinazione di zero gradi iniziale.

2.5.3: Realizzazione widget vista tridimensionale

Il secondo obiettivo del progetto era quello di realizzare un widget che mostrasse un modello tridimensionale animato da tutti i valori ORI ricevuti, ossia yaw, pitch e roll.

Per realizzare quanto descritto è stato necessario utilizzare una libreria chiamata BabylonJS viewer che sostanzialmente è un visualizzatore di oggetti tridimensionali in formato glb, in quanto Flutter di per sé non prevede la possibilità di renderizzare modelli tridimensionali.
Il concetto di ricezione dei valori di orientazione spaziale è uguale al caso dell'inclinometro bidimensionale ma con l'aggiunta anche dell'acquisizione dello yaw, non necessaria nel caso precedente.

È stato scaricato un modello di un veicolo tridimensionale e poi è stato caricato mediante BabylonJS viewer.
In seguito, sono state scritte diverse funzioni in Javascript, visto che la libreria è basata su questo linguaggio, per la gestione della rotazione del modello tridimensionale.

Inizialmente sono state scritte delle funzioni in grado di ruotare il modello con valori di rotazione fissati e lungo assi differenti, in modo da simulare variazioni di yaw, pitch e roll. Successivamente, le rotazioni sono state connesse all'aggiornamento dei dati di orientazione spaziale forniti dall'Arduino mediante la funzione rotateImage.

Inoltre, dato che di default il visualizzatore tridimensionale fa ruotare automaticamente la vista intorno all'oggetto tridimensionale, sono state anche definite una funzione toggleAutoRotate la quale impedisce alla visualizzazione tridimensionale di modificare la vista esterna e di far ruotare la visuale intorno all'oggetto e una funzione movePosition che si occupa di aggiornare la posizione spaziale delle mesh.

```
function toggleAutoRotate(texture) {
                    let viewer = BabylonViewer.viewerManager.getViewerById('viewer-id');
                    viewer.sceneManager.camera.useAutoRotationBehavior = !viewer.sceneManager.camera.useAutoRotationBehavior
                      }
function movePosition(texture) {
                    let viewer = BabylonViewer.viewerManager.getViewerById('viewer-id');

                    let scene = viewer.sceneManager.scene; // Ottieni l'oggetto della scena

                    let meshesInScene = scene.meshes; // Ottieni una lista di tutte le mesh nella scena
                    let mesh;
                    for (let i = 0; i < meshesInScene.length; i++) {
                      if(meshesInScene[i].name == 'pivotMesh'){
                        console.log("Found mesh")
                        mesh = meshesInScene[i];
                        meshesInScene=null;
                        break;
                      }
                    }
                    let x = mesh.position.x;
                    let y = mesh.position.y;
                    let z = mesh.position.z;
                }
```

Fig. 2.9: implementazione delle funzioni Javascript toggleAutoRotate e movePosition

Nella figura 2.9 si possono osservare nel dettaglio l'implementazione delle funzioni Javascript toggleAutoRotate e movePosition.

Inizialmente sceglie l'istanza del visualizzatore tridimensionale e poi inverte lo stato della rotazione automatica della visuale la quale, se era attiva, viene bloccata. Se quest'ultima, invece, era già disattivata, premendo nuovamente il bottone che chiama la funzione toggleAutoRotate, sblocca la visuale che ricomincia a ruotare automaticamente.

In figura 2.10, invece, si può osservare nel dettaglio l'implementazione della funzione rotateImage. Inizialmente viene ottenuto l'oggetto della scena, si esegue un'iterazione su tutte le mesh e poi vengono assegnati i valori di yaw, pitch e roll per consentire la rotazione del modello tridimensionale nello spazio.
I segni negativi antecedenti allo yaw e al roll sono dovuti all'orientamento predefinito degli assi del visualizzatore tridimensionale che risulta quindi opposto all'orientamento degli assi dei sensori di Arduino. Dunque, questo problema viene risolto mediante un'inversione di segno ove necessario.

```
function rotateImage(yaw, pitch, roll, texture) {
  let viewer = BabylonViewer.viewerManager.getViewerById('viewer-id');

  let scene = viewer.sceneManager.scene; // Ottieni l'oggetto della scena

  let meshesInScene = scene.meshes; // Ottieni una lista di tutte le mesh nella scena
  let mesh;
  for (let i = 0; i < meshesInScene.length; i++) {
    if(meshesInScene[i].name == 'pivotMesh'){
      console.log("Found mesh")
      mesh = meshesInScene[i];
      meshesInScene=null;
      break;
    }
  }
  mesh.rotation.x = pitch;
  mesh.rotation.y = -yaw;
  mesh.rotation.z = -roll;
}
```

Fig. 2.10: implementazione della funzione Javascript rotateImage

È stato necessario modificare il pivot del modello tridimensionale della macchinina in quanto originariamente era situato in corrispondenza dell'assale posteriore. Per modificare la posizione del pivot è stato utilizzato Blender, un software di modellazione e texturing di immagini tridimensionali.

Nella figura 2.11 si può osservare il modello tridimensionale con il pivot posizionato correttamente al centro della vettura.

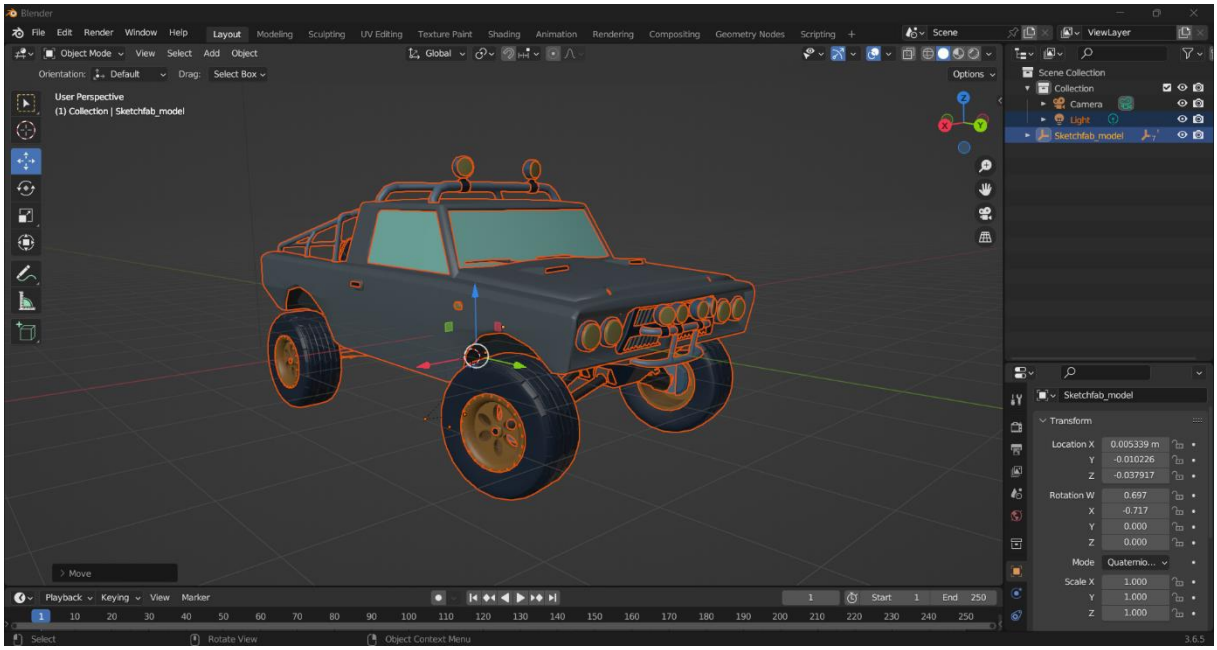

Fig. 2.11: centraggio del pivot mediante Blender

Nella figura 2.12, è possibile osservare uno screenshot del widget che realizza la vista tridimensionale.

Fig. 2.12: widget per la vista di un modello tridimensionale animato

Il bottone start serve a fare iniziare l'animazione del modello basata sui valori di orientazione spaziale calcolati da Arduino. Una volta premuto esso diventa un bottone di stop che permette di fermare la rotazione.

3. Sperimentazione e Risultati

Al termine dell'analisi hardware e software il sistema è stato testato.

Inizialmente, il sistema è stato testato senza essere effettivamente montato sul modellino. Da questi test sono emersi dei risultati poco coerenti e con molto drift: mantenendo Arduino fermo su una superficie piana i valori di yaw, pitch e roll non rimanevano costanti a zero, bensì aumentavano in maniera incontrollata.
Si è dunque compreso che era necessario effettuare una calibrazione di accelerometro, magnetometro e giroscopio in quanto questi sensori, essendo particolarmente economici, non forniscono una precisione molto accurata senza una calibrazione meticolosa.
Avendo diverse schede Arduino Nano a disposizione sono state effettuate delle misurazioni su ognuno di essi e presentavano tutti delle differenze misurate.

È stata quindi effettuata una calibrazione completa di accelerometro, magnetometro e giroscopio grazie a degli sketch specifici e sono stati ricavati i valori di offset da passare alla funzione che li interpolava per ottenere i valori di orientazione spaziale.
Così facendo i valori misurati risultano ora coerenti e presentano un leggero drift accettabile.
Questa piccola quantità di imprecisione e talvolta di lentezza nell'aggiornamento nei valori di orientazione spaziale è dovuta principalmente a due fattori differenti:

- Hardware economico: infatti, come evidenziato prima, i sensori che equipaggiano Arduino Nano hanno un costo ridotto e relativamente meno precisi
- Software: è importante ricordare che la libreria utilizzata è una libreria open source e non completamente ottimizzata. Probabilmente scrivendo interamente la funzione che realizza il filtro di Kalman interpolando i valori dei sensori si potrebbero ottenere risultati migliori.

Dopo aver verificato che i valori di orientazione spaziale misurati dalla scheda erano corretti si è proseguito con l'installazione della stessa a bordo del modellino in scala del fuoristrada.

Dato che internamente alla scocca lo spazio era ridotto e non pianeggiante, Arduino è stato fissato al cassone posteriore del pick-up che risulta perfettamente parallelo al suolo.

Fig. 3.1: fissaggio di Arduino sulla scocca del modellino in scala

Il sistema è stato successivamente testato sul campo e si è dimostrato preciso e responsivo ai cambiamenti di inclinazione del mezzo mentre affrontava tracciati rocciosi, in salita e in discesa.

4. Contributo personale e considerazioni conclusive

Ora che sono stati esposti tutti i dettagli tecnici dell'implementazione si possono fare alcune considerazione sulle scelte progettuali.

Inizialmente l'obiettivo della tesi era quello di realizzare esclusivamente un inclinometro bidimensionale, ma poi lavorandoci su è stato deciso di proseguire e aggiungere anche la funzionalità di una vista tridimensionale di un modello animato.
È importante chiarire che una vista tridimensionale non è particolarmente utile come un inclinometro bidimensionale in ambito di macchinine fuoristrada, però risulta comunque essere una feature interessante e piacevole da avere e visualizzare per l'utente finale.
Il progetto, inoltre, potrebbe essere ampliabile includendo funzionalità come la visualizzazione della telemetria in tempo reale, velocità istantanea, statistiche su massimi angoli di inclinazione raggiunti o velocità massima raggiunta, oppure anche una mappa bidimensionale che mostra il tracciato che è stato seguito.
Il lavoro svolto potrebbe anche essere di potenziale interesse per diverse aziende produttrici di modellini in scala, in quanto permetterebbe ai loro utenti di ottenere le stesse funzionalità offerte dalle case più costose, come ad esempio la Traxxas, ma il tutto a prezzi molto più contenuti e convenienti.
In particolare, si potrebbero includere tutte le funzionalità di Arduino Nano all'interno dell'ESC (electronic speed controller) del modello, all'interno della quale si potrebbe inserire un microcontrollore connesso ad un unità Bluetooth Low Energy e ad una sensoristica comprendente accelerometro, giroscopio e magnetometro.

In conclusione, questo progetto di tesi mi ha permesso di ampliare le mie conoscenze, affrontando nuove sfide e dandomi quindi la possibilità di mettermi in gioco rispetto ad argomenti mai incontrati finora nel mio percorso universitario. Per tali motivazioni è stata un'esperienza molto formativa e che ha indubbiamente arricchito il mio bagaglio culturale, fondamentale nella mia carriera professionale futura.

5. Ringraziamenti

In primo luogo, desidero ringraziare il mio relatore Riccardo Berta per aver accolto con interesse la mia proposta di tesi e avermi permesso di realizzare il progetto.
Un ringraziamento particolare va al mio correlatore Matteo Fresta, che mi ha seguito durante la realizzazione e la stesura del progetto, con numerosi e preziosi consigli, trasmettendomi nuove conoscenze e permettendomi di migliorare le mie capacità di problem solving.

Infine, vorrei ringraziare tutte le persone che mi sono state vicino e mi hanno supportato durante questo percorso universitario. Grazie di cuore a ciascuno di voi per aver reso possibile questo importante capitolo della mia vita.

6. Bibliografia

1. https://github.com/Activity-Tracker-Framework/edge-meter
2. https://forum.arduino.cc/t/nano-33-heading-keeps-on-dropping-while-resting/635234/55
3. https://docs.arduino.cc/hardware/nano-33-ble-sense-rev2
4. https://www.arduino.cc/reference/en/libraries/arduino_bmi270_bmm150/
5. https://dart.dev/
6. https://pub.dev/packages/babylonjs_viewer
7. https://it.wikipedia.org/wiki/Flutter_(software)
8. https://docs.arduino.cc/hardware/nano-33-ble-sense
9. https://github.com/h65wang/slide-puzzle-challenge
10. https://github.com/NF1198/nano33ble-tilt-compensated-compass
11. https://www.youtube.com/watch?v=BLvYFXoP33o