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

Progetto e implementazione di un sistema per la trasmissione video da dispositivo embedded

Design and development of an embedded system for video streaming

Relatore: Prof. Riccardo Berta

Correlatore: Dr. Matteo Fresta

Candidato: Alessandro Demarchi

Settembre 2024

Sommario

Il presente progetto di tesi propone la realizzazione e progettazione di un sistema embedded IoT per una macchinina radiocomandata chiamata Traxxas, controllata a distanza tramite un'applicazione mobile. Il lavoro si è concentrato sull'aggiunta di una nuova funzionalità: l'acquisizione, trasmissione e ricezione su un dispositivo mobile di un flusso video che mostra il punto di vista anteriore del veicolo. Per ottenere questo risultato, è stata utilizzata una telecamera su una scheda ESP-EYE-, un microcontrollore basato su un chip ESP32, posizionata sul parafango anteriore, in modo da fornire una visione in tempo reale del percorso davanti alla macchina.

Le sequenze di immagini che compongono il video vengono trasmesse dall'ESP-EYE tramite WebSocket su una rete WiFi LAN, per essere poi ricevute dal dispositivo mobile su cui è in esecuzione l'applicazione. Quest'ultima proietta il flusso video sullo schermo, mantenendo i comandi per il controllo della macchina Traxxas al di sopra del video. In questo modo, l'utente piò manovrare il veicolo a distanza sfruttando la visione in tempo reale del contesto di guida.

Indice

1 Introduzione

I sistemi embedded IoT rappresentano una fusione tra dispositivi integrati e la connettività Internet, creando una rete di oggetti intelligenti capaci di raccogliere, elaborare e trasmettere dati. Questi sistemi sono progettati per svolgere compiti specifici in tempo reale, utilizzando risorse limitate come memoria e potenza di calcolo. Grazie alla loro connessione alla rete, possono interagire con altri dispositivi o sistemi, offrendo soluzioni efficienti e innovative per migliorare la qualità della vita e ottimizzare i processi produttivi.

L'Internet of Things (IoT) ha rivoluzionato il modo in cui interagiamo con i dispositivi, consentendo la creazione di sistemi intelligenti capaci di comunicare e scambiare dati in tempo reale. All'interno di questo contesto tecnologico, il presente progetto di tesi si propone di approfondire la progettazione e la realizzazione di un sistema embedded, analizzando le scelte tecniche adottate e i risultati ottenuti, con l'obiettivo di fornire un contributo innovativo nell'ambito dei sistemi embedded e dell'IoT e delle applicazioni mobile.

Nello specifico l'obiettivo di questa tesi è l'integrazione della funzionalità video sull'applicazione, ossia, fare in modo che l'utente possa visualizzare il flusso video a schermo insieme ai comandi, con il risultato di una guida dinamica a distanza.

Prima di spiegare il funzionamento del sistema è opportuno eseguire una presentazione dei principali componenti di esso, con una breve spiegazione introduttiva della loro funzione:

- La macchina Traxxas; una macchina radiocomandata la cui ESC (Electronic Speed Control, cioè la centralina di comando originale) è stata sostituita con un Arduino nano 33 BLE Sense. La sua funzione è quella di assumere nel contesto globale, insieme all'Arduino, il ruolo di dispositivo radiocomandato programmabile, affinché il sistema possa essere controllato nella maniera desiderata.
- L'Arduino nano 33 BLE; piccolo microcontrollore scelto come "cervello" della macchina Traxxas in quanto possedente potenza di calcolo più che sufficiente per il suo scopo in questo sistema: controllo dei motori all'interno del veicolo per l'accelerazione e lo sterzo.
- L'ESP-EYE; una piccola scheda basata su un chip ESP32 la cui peculiarità per cui è stata scelta è la telecamera montata su di essa. Questo elemento è cruciale per la funzionalità che questo progetto si propone di aggiungere: il flusso video captato da questo microcontrollore sarà riprodotto sullo schermo del dispositivo che eseguirà l'applicazione.
- Il dispositivo mobile; si tratta dell'interfaccia che l'utente possiede con il sistema, cioè il metodo con cui egli interagisce attivamente e visualmente attraverso l'applicazione sviluppata in questo progetto.

Il sistema finale può essere schematizzato in questo modo basandosi sui dati scambiati tra i vari componenti del sistema, ossia il video, il dato più pesante, e i dati relativi al movimento della macchina:

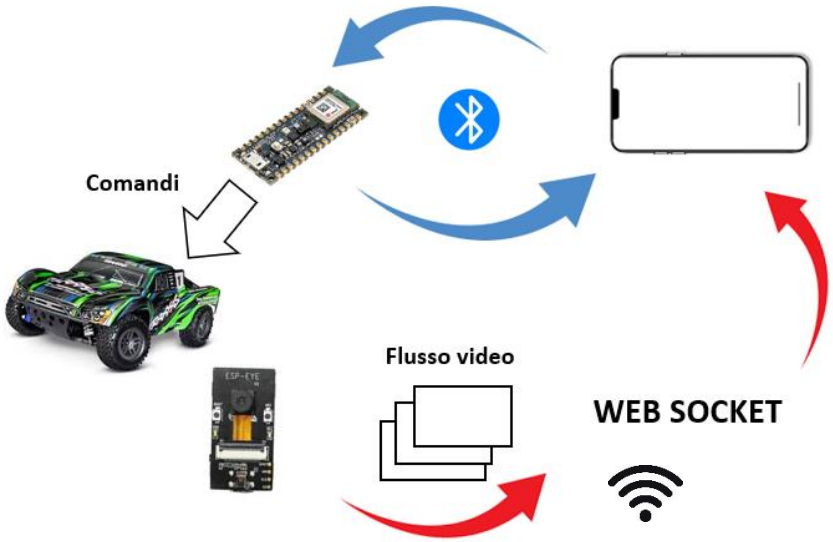

Figura 1: Schema funzionale del sistema

Come si evince dalla figura 1 possiamo distinguere due canali di comunicazione, uno per il flusso video, che viene registrato dall'ESP-EYE per poi essere trasmesso su Web Socket, e infine recepito dal dispositivo mobile. Mentre l'altro dato trasmesso è quello dei comandi che sono inviati dal dispositivo mobile all'Arduino tramite Bluetooth, nello specifico BLE (Bluetooth Low Energy), una forma di connessione Bluetooth a basso consumo di energia.

L'Arduino e l'ESP-EYE sono posizionati entrambi in maniera funzionale sul veicolo. L'Arduino è al di sotto della scocca della macchina, mentre l'ESP-EYE è posizionato sul parafango anteriore.

Per motivi progettuali l'ESP-EYE e il dispositivo mobile devono essere connessi alla stessa rete LAN WiFi, in quanto il WebSocket è inizializzato all'interno della rete e non vi può essere fatto accesso da reti esterne. Questa caratteristica impone sicuramente dei limiti, che saranno trattati successivamente, ma assicura anche più sicurezza all'accesso del video.

2 Metodi e strumenti utilizzati

Sono illustrati in questo capitolo tutti gli strumenti, hardware e software, che sono stati necessari per arrivare alla progettazione e realizzazione pratica del sistema.

2.1 L'ESP-EYE

Per questo compito è stata scelta questa scheda di sviluppo dotata di:

- un chip ESP 32, un microcontrollore dual-core a 32-bit con frequenza operativa fino a 240 MHz. Il chip è dotato di connettività Wi-Fi e Bluetooth, insieme a un gran numero di GPIO e periferiche, che lo rendono estremamente flessibile per molte applicazioni IoT
- 8 MB di PSRAM (un tipo di memoria a metà strada tra DRAM e SRAM, con un utilizzo tipico nei dispositivi IoT)
- 4 MB di memoria flash
- una fotocamera OV2640 con risoluzione di 2 Megapixel, sebbene non abbia una risoluzione elevatissima, è più che sufficiente per molte applicazioni embedded
- la scheda dispone anche di un microfono integrato, che permette di sviluppare applicazioni basate su riconoscimento vocale o audio processing.

La sua struttura si presta molto a questo tipo di applicazione, infatti, non possiede dei pin liberi configurabili (come la ESP32-CAM, scheda simile) ma si interfaccia all'esterno tramite una porta USB, che permette un debugging e alimentazione agevole.

Un vantaggio di questa scheda, che si è rivelato fondamentale nella fase di analisi e strutturazione preliminare del progetto, è sicuramente quello di essere molto utilizzata e di avere le stesse applicazioni e utilità dell'ESP32 CAM. I codici sviluppati per la ESP32 CAM sono compatibili e utilizzabili con l'ESP-EYE, cambiando eventuali inizializzazioni e configurazioni dei pin.

Figura 2: ESP-EYE e ESP32 CAM

L'ESP-EYE nasce anche per applicazioni di image recognition che richiedono poco calcolo computazionale; infatti, il firmware originale possiede un algoritmo di face recognition, in grado riconoscere i volti delle persone memorizzate. Questa caratteristica permetterebbe sicuramente una futura evoluzione del progetto, nel quale è possibile inserire un algoritmo di object detection utile al riconoscimento di alcuni oggetti basilari, questa possibilità è trattata nella sezione “sviluppi futuri” in uno dei capitoli successivi.

2.2 L’Arduino Nano 33 BLE Sense

Si tratta di una scheda di sviluppo progettata per applicazioni IoT e di intelligenza artificiale, che integra potenti funzionalità di elaborazione e una vasta gamma di sensori. Basata sul microcontrollore nRF52840 della Nordic Semiconductor, dotato di un core ARM Cortex-M4 a 32 bit con supporto FPU (Floating Point Unit), questa scheda offre capacità computazionali avanzate e supporto nativo per la connettività Bluetooth Low Energy (BLE) e NFC (Near Field Communication).

Uno degli aspetti distintivi dell'Arduino Nano 33 BLE Sense è l'integrazione di diversi sensori ambientali e di movimento direttamente sulla scheda. Tra questi troviamo un sensore di temperatura, umidità e pressione barometrica (Bosch BME280), un sensore di qualità dell'aria e composti organici volatili (Sensirion SGP40), un accelerometro a 9 assi (LSM9DS1), un microfono MEMS (MP34DT05), e un sensore di luce, prossimità e colore (APDS-9960). Questa combinazione di sensori rende la scheda particolarmente adatta per applicazioni di monitoraggio ambientale, progetti di machine learning on-device e interfacce uomo-macchina avanzate.

In termini di connettività, l'implementazione del protocollo Bluetooth 5.0 consente un'ampia gamma di comunicazione wireless con bassa latenza e basso consumo energetico, cruciale per dispositivi alimentati a batteria. La scheda è compatibile con l'ecosistema Arduino e supporta il framework Arduino IDE.

Le prestazioni di questa scheda sono sicuramente sovradimensionate per il compito fornitogli in questo contesto: mantenere la connettività BLE con il dispositivo mobile e trasmettere i comandi ai motori del veicolo. Ciò è anche a beneficio di una futura espansione o modifica del codice.

L’alimentazione dell’Arduino è fornita attraverso una batteria a bottone sviluppata ad hoc per questa scheda, con l’obiettivo di proteggerla da sovratensioni date dalla batteria della macchina.

Figura 3: Arduino Nano 33 BLE Sense

2.3 La macchina Traxxas

Si tratta di un veicolo radiocomandato (RC), dotato di:

- un motore in tensione continua per generare l’accelerazione
- un motore servo per gestire la sterzata
- originariamente, di una centralina ricevente per ricevere i comandi dall’esterno
- di un ESC (Electronic Speed Control), cioè il componente necessario per gestire la potenza fornita al motore DC sulla base di segnali PWM

La centralina ricevente è stata sostituita con un Arduino Nano 33 BLE Sense, così da poter programmare il controllo della macchina in maniera personalizzata, cioè attraverso il dispositivo mobile in cui è eseguita l’applicazione.

Figura 4: Macchinina Traxxas

2.4 Flutter

Per questa sezione del progetto è stato necessario lavorare in un ambiente di sviluppo di applicazioni adeguato, cioè Flutter; si tratta di un framework UI open-source creato da Google specifico nello sviluppo di applicazioni mobile, web, desktop ed embedded, che ha permesso, attraverso il linguaggio Dart, la programmazione della parte preesistente del progetto, cioè il collegamento Bluetooth e trasmissione dei comandi, e di quella dedicata all’inserimento di un’interfaccia video.

2.5 Linguaggio Dart

Dart è un linguaggio di programmazione open-source sviluppato da Google, progettato per essere versatile, veloce e facile da usare, con un focus particolare sullo sviluppo di applicazioni client moderne per piattaforme multiple, come il web, mobile, e desktop.

Dart è diventato il linguaggio principale per lo sviluppo di applicazioni con Flutter, ma è utilizzato anche per applicazioni lato server e di scripting, cioè applicazioni che automatizzano attività quotidiane o amministrative.

È stato introdotto per la prima volta da Google nel 2011, con l'obiettivo di sostituire JavaScript nel contesto dello sviluppo web, offrendo una sintassi più moderna e prestazioni migliori. Anche se Dart non ha mai sostituito JavaScript nel web come originariamente previsto, ha trovato successo come il linguaggio principale dietro Flutter, grazie alla sua capacità di gestire interfacce utente complesse e animazioni con alte performance.

2.6 Visual Studio Code

Visual Studio Code (VS Code) è uno degli editor di codice sorgente più popolari per la programmazione in Flutter, si tratta di un software libero e gratuito sviluppato da Microsoft che può essere usato con vari linguaggi di programmazione C, C++, C#, Python, HTML e ovviamente anche Dart.

È stato uno strumento software cruciale per lo sviluppo dell'applicazione su Flutter, tramite le varie funzioni di debugging e correzione e revisione del codice sia runtime che non.

2.7 Arduino IDE 2.3.3

Per la programmazione di questo sistema è stato necessario uno strumento che potesse descrivere i codici dei due microcontrollori, l'Arduino Nano 33 BLE Sense e l'ESP-EYE, nel linguaggio C++, e per questo è stato utilizzato l'Arduino IDE 2.3.3.

Si tratta di una versione aggiornata dell'ambiente di sviluppo integrato (IDE) per programmare sia schede Arduino sia altri tipi di schede con microcontrollori a bordo. Rispetto alle versioni precedenti, presenta un'interfaccia utente più moderna e funzionalità avanzate, come l'autocompletamento del codice, la navigazione del codice, il debug integrato (per schede supportate) e un editor reattivo.

Per l'ESP-EYE, l'IDE offre supporto nativo tramite l'estensione del core ESP32, facilitando l'integrazione delle librerie per la gestione della telecamera e connettività Wi-Fi.

2.8 Specifiche progettuali

Affinché venisse visualizzato il punto di vista davanti alla macchina, è stata scelta come soluzione progettuale quella dell'ESP-EYE, che esegue la funzione di webcam, e che trasmette il video captato su un server Web Socket all'interno di una rete locale WiFi.

A partire dal Web Socket, l'applicazione creata col framework Flutter sulla piattaforma di sviluppo Visual Studio Code acquisirà il video e lo imposterà nella stessa interfaccia dove sono collocati i comandi per il veicolo, in maniera da rendere la guida dinamica e diretta, anche a distanza.

Essendo la scheda ESP-EYE abilitata alla connessione bluetooth, sarebbe anche stato possibile l'invio dei dati del video direttamente da essa fino all'Arduino, che avrebbe potuto poi inviarlo al dispositivo mobile dell'utente (sfruttando la già presente connessione dedicata all'invio dei comandi all'Arduino), ma questa possibilità si sarebbe rivelata sicuramente più complessa, in quanto l'invio di un video streaming tramite Bluetooth è un'attività molto più complessa e pesante computazionalmente dell'invio di dati semplici come comandi.

Il flusso video avrebbe dovuto rispettare un giusto equilibrio tra qualità e framerate affinché sia possibile, in futuro, l'aggiunta di un algoritmo di object detection. Per rispettare questa specifica è stato necessario intervenire all'interno del codice creato appositamente per il microcontrollore ESP-EYE, nel quale è possibile regolare un parametro che gestisce framerate e risoluzione del video.

Inoltre, questa aggiunta permetterebbe un'altra ipotetica evoluzione del sistema, che comprende l'aggiunta di un algoritmo di guida autonoma che si basa sul riconoscimento di oggetti sopra citato, questa possibilità è trattata in un prossimo capitolo dedicato agli sviluppi futuri.

3 Sperimentazione e risultati

Sono riportate in questo capitolo le due fasi di sperimentazione e attuazione del progetto, la prima legata alla acquisizione e trasmissione del video su Web Socket, e la seconda, dove il focus è stato analizzare e comprendere la struttura dell'applicazione, per poi inserire la funzionalità video ricevendolo da Web Socket e implementandolo nell'interfaccia comandi.

3.1 Fase 1: Acquisizione e trasmissione video

Per l'acquisizione del flusso video e streaming su Web Socket è stata necessaria la scrittura di un apposito codice C++ dedicato all'ESP-EYE. Attraverso l'Arduino IDE si può verificare che il codice occupa il 32 % della memoria dedicata ai programmi nella scheda, usando 1008597 byte su 3145728, lasciando così spazio a future implementazioni di image processing all'interno dell'ESP-EYE.

3.1.1 Inclusione delle librerie

I compiti che il microcontrollore doveva svolgere sono due:

- acquisizione video
- streaming del video sul server

Per il primo, è stata necessaria l'inclusione della libreria esp_camera.h, mentre per il secondo WebSocketsServer.h. Inoltre, è servita anche l'aggiunta di WiFi.h per la connessione della scheda al web, affinché accedesse al Web Socket.

3.1.2 Configurazioni

Per configurare e inizializzare correttamente la scheda sono state necessarie alcune inizializzazioni e assegnamenti di valori standard dell'ESP-EYE, che sono stati ricavati da progetti analoghi trovati sul web.

Tra questi vi sono:

- configurazione dei pin
- configurazione della risoluzione e del frame rate
- configurazione di altri parametri dell'immagine (saturazione e luminosità)

Per quanto riguarda la seconda, è stata necessaria, in una fase successiva, la scelta del giusto rapporto dei due parametri. Infatti, considerando il movimento della telecamera stessa, in quanto posta sulla macchina, il video avrebbe dovuto avere il giusto compromesso tra risoluzione e frame per secondo rispettando così le specifiche; questa scelta è stata effettuata nello specifico durante la fase di test dell'applicazione flutter.

Mentre per le altre configurazioni è bastato adattarsi allo standard degli altri progetti, basandosi sugli stessi valori di inizializzazione dei pin (attraverso l'utilizzo di un file esterno chiamato camera_pins.h) e dell'immagine.

3.1.3 Connessione WiFi

Per la scrittura del codice per stabilire una connessione alla rete WiFi sono stati utilizzati i seguenti metodi forniti dalla libreria WiFi.h:

- WiFi.begin(ssd,password) è stata utilizzata per instaurare la connessione con la rete specificata nei campi (ssd, password) della funzione.

- WiFi.setSleep(false) per disabilitare la modalità di sospensione della connessione Wi-Fi, che la connessione sempre attiva e pronta per trasmettere dati (la modalità sleep può aiutare a risparmiare energia, ma può introdurre ritardi o interruzioni nella trasmissione).
- Attesa della connessione alla rete Wi-Fi, questo ciclo while controlla continuamente lo stato della connessione Wi-Fi utilizzando WiFi.status(). Se il dispositivo non è ancora connesso (WL_CONNECTED), il ciclo continua a stampare un punto ogni 500 millisecondi (Serial.print(".")), segnalando che sta ancora tentando di connettersi. Quando il dispositivo si connette correttamente alla rete, esce dal ciclo e stampa "WiFi connected" sulla console seriale dell'IDE.
- Comunicazione del collegamento WiFi instaurato, e stampa sul monitor seriale dell'indirizzo IP a cui connettersi, attraverso il metodo WiFi.localIP(). L'IP stampato corrisponde all'indirizzo locale che il router ha assegnato all'ESP-EYE all'interno della rete locale (LAN, Local Area Network) tramite il DHCP (Dynamic Host Configuration Protocol), e viene utilizzato per accedere al server Web Socket dall'interno della rete, non dall'esterno. In particolare, il Web Socket viene istanziato sulla porta 81 su cui trasmettervi le immagini del video.

```
81  WiFi.begin(ssid, password);
82  WiFi.setSleep(false);
83
84  while (WiFi.status() != WL_CONNECTED) {
85    delay(500);
86    Serial.print(".");
87  }
88  Serial.println("");
89  Serial.println("WiFi connected");
90
91  webSocket.begin();
92  webSocket.onEvent(onWebSocketEvent);
93
94  Serial.print("Camera Ready! Use 'ws://");
95  Serial.print(WiFi.localIP());
96  Serial.println(":81' to connect");
97
98  startCameraServer();
```

Figura 5: Sketch collegamento WiFi dell'ESP-EYE

3.1.4 Inizializzazione e utilizzo del Web Socket

Il Web Socket viene inizializzato attraverso la funzione webSocket.begin(), che instaura la comunicazione tra il client e il dispositivo ESP-EYE; si tratta di una comunicazione bidirezionale, ma in realtà in questo caso è utilizzata monodirezionalmente, cioè, nonostante sia tecnicamente bidirezionale, l'unico flusso di dati è da server (ESP-EYE) a client, cioè l'utente che acquisisce il video.

In caso di disconnessione e connessione di un client o di altri messaggi, vengono stampati dei messaggi attraverso il metodo webSocket.onEvent(onWebSocketEvent), per esempio in caso di connessione di un client vengono stampati sul monitor seriale il suo indirizzo IP e il suo ID.

Il flusso video non è altro che una serie di immagini, ed esse sono mandate sul websocket chiamando la funzione startCameraServer(), essa avvia un task FreeRTOS che si occupa di catturare e inviare continuamente immagini dalla fotocamera ai client connessi, sfruttando la funzione send_image_task.

Usa xTaskCreate() per creare un nuovo task che esegue un ciclo infinito (while (true)) all'interno del quale chiama send_image_task() per catturare e inviare un'immagine ai client. Il ciclo si ripete ogni 100 millisecondi grazie alla chiamata a delay(100), il che significa che le immagini vengono inviate periodicamente ai client connessi. Questo task è eseguito in parallelo rispetto al loop principale e gestisce in modo continuo lo streaming delle immagini.

3.1.5 send_image_task

Cattura un'immagine dalla fotocamera e la invia a tutti i client connessi tramite WebSocket. Usa esp camera fb get() per catturare un'immagine (frame) dalla fotocamera, il risultato viene memorizzato in un buffer (fb) che contiene i dati dell'immagine. Se la cattura dell'immagine fallisce (!fb), viene stampato un messaggio di errore e la funzione termina; se invece ci sono client connessi (webSocket.connectedClients() 0), l'immagine viene trasmessa a tutti i client tramite il metodo webSocket.broadcastBIN(fb-buf, fb-len), che invia i dati in formato binario. Infine, rilascia il buffer dell'immagine chiamando esp camera fb return(fb) per evitare perdite di memoria.

Dopo lo sviluppo codice per la webcam è stato possibile iniziare a modificare la applicazione affinchè usufruisse del flusso video presente sul Web Socket.

3.2 Fase 2: Ricezione e implementazione del video sull'applicazione

In questa sezione di progetto il focus è stato quello di analizzare l'applicazione per il controllo della macchina sia visualmente sia dal punto di vista della programmazione del codice, implementandovi la funzionalità video.

3.2.1 Main

Il file main.dart contiene dapprima l'inizializzazione dell'applicazione tramite la solita definizione della classe principale (in questo caso MyApp) con ad essa associato lo stato della classe (MyAppState).

In seguito, si occupa di descrivere, tramite la funzionalità dei widget, la schermata iniziale e principale dell'applicazione; in questa schermata l'utente visualizza in alto il titolo della schermata "Discover Devices", e subito sotto è presente una scritta che serve come feedback per la connettività bluetooth del dispositivo mobile: "bluetooth init:" seguito da "true" se il Bluetooth è disponibile, mentre "false" in caso contrario. In basso è presente un bottone, "request permissions", che se premuto richiede all'utente i permessi per l'accesso dell'app alla connettività bluetooth del dispositivo mobile; mentre sopra visualizza due bottoni: "startscan", che se premuto inizia a scannerizzare l'ambiente alla ricerca di dispositivi bluetooth disponibili, e "stopscan" che ferma la ricerca.

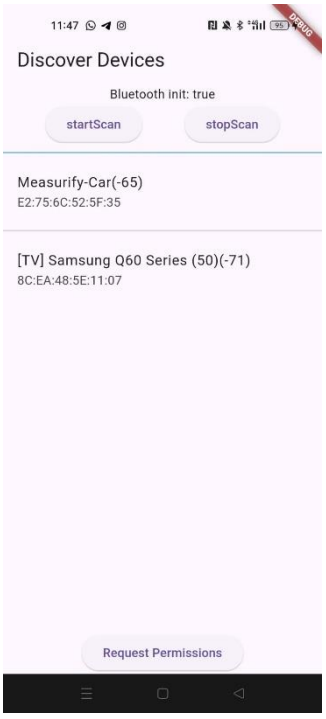

Figura 6: Schermata principale dell'applicazione

Sempre nella stessa schermata vengono elencati i dispositivi disponibili attraverso un elenco di righe in cui sono scritti i rispettivi nomi. Se uno di loro viene premuto, si passa alla pagina successiva in cui viene gestita la connettività con esso: "PeripheralDetailPage.dart". Insieme al nome vengono presentati altri due parametri:

- L'RSSI (Received Signal Strength Indicator), un parametro che misura la potenza del segnale ricevuto da un dispositivo Bluetooth, rappresenta la forza con cui un dispositivo riceve un segnale Bluetooth proveniente da un altro dispositivo; esso viene solitamente espresso in decibel milliwatt (dBm). Generalmente ha un valore negativo e più è prossimo allo zero più il segnale è forte
- l'id del dispositivo, cioè una stringa esadecimale di 48 bit (un indirizzo MAC)

Il passaggio di schermata è gestito tramite il metodo Navigator.push, all'interno del quale viene passato l'id del device selezionato. In generale tutte le funzioni e gestioni delle attività bluetooth sono adibite al package "quickblue" presente nella directory del progetto, che non è altro che una libreria di Flutter specifica per implementare funzionalità BLE.

3.2.2 PeripheralDetailPage

Figura 7: schermata PeripheralDetailPage dedicata alla connessione bluetooth

Questo file gestisce la seconda schermata, dedicata alla connettività bluetooth con il dispositivo selezionato nella pagina precedente. In alto presenta il nome della schermata, "PeripheralDetailPage", e sotto tre bottoni:

- "Connect": per connettere definitivamente il dispositivo con l'arduino
- "Disconnect": per disconnetterlo
- "Go to command page": per passare alla pagina dei comandi della macchina traxxas

Oltre a ciò, questa sezione di codice gestisce anche tre funzioni molto importanti per la connettività bluetooth BLE:

- “QuickBlue.setValueHandler”, necessario per la ricezione dei dati %potrebbe servire anche per la trasmissione non sono sicuro
- “QuickBlue.setServiceHandler”, per il riconoscimento di altri dispositivi bluetooth
- “QuickBlue.setConnectionHandler”, per impostare definitivamente la connessione

3.2.3 Command

Infine, questa sezione di codice rappresenta e descrive l'ultima pagina, che implementa i controlli del veicolo senza mostrare il flusso di immagini del video.

In questo caso (in seguito ovviamente all'inclusione dei package necessari) all'inizio del codice vi è una parte dedicata all'inizializzazione di costanti dedicate al movimento, e di un parametro (timing) che identifica i millisecondi che passano tra ogni pacchetto BLE inviato. Nella stessa sezione possiamo trovare la definizione della classe principale e del suo stato:"TraxxasJoystickExample".

In seguito, il codice prosegue con la definizione di alcune funzioni:

- void initState(), che definisce l'inizializzazione del widget "TraxxasJoystickExample" attraverso 4 sezioni:
    - super.initState(), necessaria per assicurarsi che venga eseguita la logica di inizializzazione della classe base State.
    - loadConfigVariables(), metodo che carica variabili di configurazione importanti per la comunicazione BLE, come gli ID di servizio.
    - Impostazione di un timer che invia ogni "timing" millisecondi i dati di _sendData(), descritta in seguito, per assicurare che i dati vengano trasmessi a intervalli regolari in maniera da essere sempre aggiornati con gli ultimi input.
    - Blocco dell'orientamento dello schermo in modalità orizzontale attraverso "SystemChrome.setPreferredOrientations([...])".

- void _sendData(), Che si occupa dell'invio bluetooth dei dati di acceleratore (throttle) e sterzo (steering).
- Future<void> loadConfigVariables(), volto ad assegnare direttamente i valori predefiniti alle variabili globali bleServiceId e movingCharacteristicId, utilizzando i valori presenti in defaults.dart, un file che possiede solo una classe contenente i valori sopracitati e il valor di default di "deviceId", inizializzato come una stringa vuota.

E infine, il codice della pagina termina con la scrittura del widget dei controlli. Esso si presenta con il titolo del progetto dell'applicazione: "Measurify Car" scritto in alto su una barra bianca. Sotto sono definiti i comandi, che sono stati descritti in maniera riassuntiva nell'introduzione, sono riportate in questa sezione le rispettive descrizioni più accurate.

- L'acceleratore è realizzato, come già citato, come un joystick, cioè un tipo di comando che consente all'utente di interagire con l'interfaccia tramite un movimento verticale (mode: JoystickMode.vertical). ll joystick è posizionato all'interno di un contesto di allineamento tramite il widget Align, che ne regola la posizione orizzontale e verticale sullo schermo utilizzando il parametro Alignment. La posizione orizzontale è determinata da una variabile (stateCommand) moltiplicata per un fattore di -0.8, mentre l'allineamento verticale è fisso a 0.5, centrando il comando nella parte bassa dello schermo.
  Il comportamento del joystick viene gestito attraverso un listener, che rileva i movimenti dell'utente. Quando l'utente interagisce con il joystick, la posizione verticale (details.y) viene rilevata e utilizzata per calcolare il valore del parametro throttle, che controlla l'intensità del movimento del motore.
  Questo valore viene calcolato sottraendo la posizione verticale rilevata, opportunamente scalata in base ai parametri rangeValueThrottle e powerSliderValue, da un valore iniziale (idleValueThrottle).\newline
  Questo meccanismo permette di tradurre il movimento dell'utente in un controllo fine di una variabile legata alla potenza espressa dai motori.

- Lo sterzo, analogamente all'acceleratore, è un joystick posizionato nella parte opposta rispetto a quest'ultimo, che si muove nella direzione orizzontale (mode: joystickMode.horizontal). La posizione orizzontale viene determinata sempre dalla variabile stateCommand, moltiplicata per un fattore positivo di 0.8, mentre la posizione verticale è fissata a 0.5, centrando così il joystick a metà altezza dello schermo. Qui il listener rileva la posizione orizzontale (details.x), utilizzata per calcolare il valore del parametro steering, che controlla l'angolo di sterzata.
  Questo valore viene calcolato sommando la posizione orizzontale rilevata, scalata tramite il parametro rangeValueSteering, a un valore iniziale definito come idleValueSteering. Questo approccio consente di tradurre il movimento del joystick in comandi direzionali precisi, che possono essere utilizzati per controllare la sterzata di un veicolo.

- Per invertire i comandi dello sterzo e dell'acceleratore, è presente un bottone "Reverse command" posto in alto a destra dello schermo, precisamente è posizionato tramite il widget "Positioned" a 10 pixel dal lato destro dello schermo e 10 pixel dalla cima (top:10, right:10).
  L'inversione è eseguita tramite il cambio di segno di una variabile "stateCommand"; la posizione di default dei due è acceleratore a sinistra e sterzo a destra.
  La gestione della potenza è controllata da uno slider, cioè una piccola barra spostabile dall'utente da 0 a 100%. Questo widget è posizionato a 20 pixel dalla cima, 300 da sinistra e 300 da destra creando così un'area orizzontale per il contenuto. Il contenuto è racchiuso in una "Column", che allinea verticalmente una serie di widget figli; in cima troviamo una casella di testo: l'etichetta del comando "power percentage". Al centro visualizziamo il valore numerico (da 0 a 100\%) della potenza dei motori, selezionato dall'utente tramite lo slider. E infine sotto è presente il comando dello slider vero e proprio, che nel codice possiede un valore minimo di 0 e massimo di 1. Questo comando è utile a rendere la guida più versatile e adattiva all'ambiente circostante alla macchina Traxxas, in quanto in uno spazio più stretto sarà più adatta una guida più sensibile e meno potente, mentre in uno spazio aperto si è liberi di usare più potenza. Il valore dello slider di default è 10, per evitare che la macchina parta con troppa potenza ai primi comandi dell'utente.

- L'offset dello sterzo è controllato analogamente da un altro slider che possiede le medesime caratteristiche del gestore della potenza. È posizionato al di sotto di quest'ultimo e anch'esso possiede una struttura a colonna con etichetta "Steering Offset", nuovamente una rappresentazione numerica dello slider da -10 a 10 e uno slider identico. I valori negativi rappresentano un offset dello sterzo verso sinistra mentre positivi verso destra. Regolando lo slider, l'utente può compensare eventuali disallineamenti dello sterzo, centrando le ruote del veicolo in modo che il comando di sterzata sia preciso. I due slider sono messi in colonna tramite un widget "column", "Power percentage" sopra e "Steering Offset" sotto.

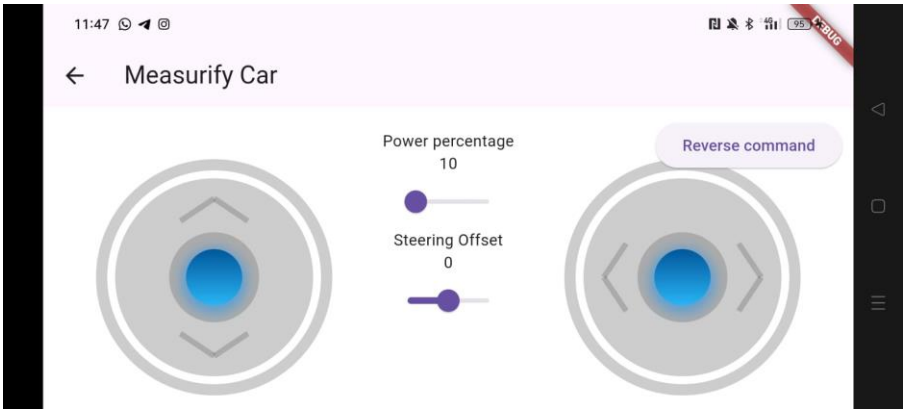

Figura 8: Schermata di controllo della macchina Traxxas priva di video

3.2.4 Sviluppo della prima pagina relativa al video

Una volta analizzata la struttura del codice relativo all'applicazione di controllo della macchina Traxxas, è stato trovato opportuno cominciare lo sviluppo della funzionalità video in un file a parte, procedendo a step, in maniera tale da implementarlo più facilmente per poi fondere il file con quello dei comandi (così che la pagina visualizzata dei comandi e del video sia la stessa).

È stato quindi creato il file webcam.dart in cui è stato scritto il primo codice per acquisire il flusso video dal Web Socket, che è un tipo di connessione client-server che si presta particolarmente per lo streaming, in quanto non richiede un continuo polling al sito siccome, una volta stabilita la connessione WebSocket, non è necessario continuare a fare richieste HTTP per ricevere nuovi dati. Questo riduce la latenza poiché il server può inviare i dati direttamente al client non appena sono disponibili, senza dover attendere una richiesta.

Per lavorare con i Web Socket è stata fondamentale l'aggiunta di una libreria, o package, di flutter cioè: "web_socket_channel", che ha semplificato enormemente la gestione della connessione WebSocket e del flusso di dati in Flutter, offrendo un'interfaccia standardizzata per l'invio e la ricezione di messaggi.

Come di consueto, all'inizio del file, subito dopo l'inclusione delle librerie, vi è la definizione della classe ("Webcam"), e del suo stato "_WebcamState". All'interno della definizione dello stato è definito a sua volta il canale sulla quale si connette il web socket, "channel".

Per connettersi al Web Socket si usa il metodo WebSocketChannel.connect(Uri.parse("indirizzo IP"),) dove l'indirizzo IP è quello restituito dal codice dell'ESP-EYE sul serial monitor, ossia l'indirizzo IP del server Web Socket. Esso si trova, come già precedentemente citato, sulla porta 81 di una LAN; quindi, non è accessibile dall'esterno in maniera normale. Nel paragrafo dedicato agli sviluppi futuri ci sarà una sezione che tratta come accedere al Web Socket dall'esterno della rete LAN, attraverso Internet.

Subito dopo la definizione dello stato troviamo il dispose, metodo di ciclo di vita nelle classi State di Flutter, che viene chiamato quando lo stato associato al widget viene rimosso (cioè, quando il widget viene eliminato). La funzione del metodo dispose() è di pulire le risorse utilizzate dallo stato del widget prima che venga distrutto. In particolare, questo è importante quando si utilizzano risorse che devono essere esplicitamente liberate, come i timer, i controller, i listener o altre risorse asincrone.

In questo caso specifico viene eseguito l'override (@override) della funzione dispose classica, cioè viene ridefinito in maniera personalizzata: viene cancellato il _timer (_timer?.cancel()) e poi viene chiamato il dispose relativo alla classe State, che gestisce ulteriori attività di pulizia necessarie da parte di Flutter (super.dispose()).

E infine vi è la solita descrizione del widget scaffold, cioè quello che comprende tutto lo schermo; la schermata presenta in alto un titolo in formato text "Webcam Stream" e subito sotto il widget del video.

Di base quest'ultimo si presenta come totalmente nero, ma attraverso un costrutto if else, differenzia la presenza di un dato effettivo da visualizzare sullo schermo oppure no. In caso negativo (if (!snapshot.hasData)), cioè in mancanza di segnale del video sul server, il codice genera un "CircularProgressIndicator" di colore bianco, cioè un simbolo di caricamento che si presenta fino a che non si presenta un effettivo segnale video da presentare a schermo.

Mentre in caso positivo, nel caso il segnale captato sia una lista di 8 interi senza segno (Uint8List), il codice restituisce il dato ricevuto dal Web Socket come immagine, altrimenti stampa a schermo "Invalid data format" per identificare un formato di dato non consono alla riproduzione su schermo.

Per riprodurre l'immagine a schermo il metodo è stato quello di far ritornare al widget streambuilder Image.memory, cioè il componente responsabile del rendering dell'immagine a partire dai dati binari presenti in memoria. Il parametro gaplessPlayback: true viene utilizzato per evitare che l'immagine lampeggi o venga rimossa momentaneamente quando cambia il contenuto dello stream, garantendo così una transizione fluida tra i frame successivi.

Per verificare che il video streaming venisse effettivamente riprodotto a schermo, sono stati fatti dei test modificando il file "main.dart" affinchè mandasse in esecuzione il file "webcam.dart" anzichè "command.dart".

Dopo qualche test si è verificato che il progetto possedeva la funzionalità desiderata: il video captato dall'ESP-EYE sullo schermo del dispositivo che eseguiva l'applicazione.

3.2.5 Implementazione di un pulsante generico

Per fondere la pagina dei comandi con quella della webcam, i comandi avrebbero dovuto essere posti come dei widget al di sopra del video, in maniera tale da essere visualizzati. Perciò, prima di provare ad aggiungere al file webcam tutti i comandi, si è pensato di aggiungere in maniera provvisoria un pulsante che non fa nulla, a scopo didattico, cioè per iniziare a capire come impostare un pulsante al di sopra di un altro widget.

Attraverso il widget "Positioned" il bottone è posizionato verticalmente a 20 pixel a partire dal fondo (bottom: 20.0) e orizzontalmente a metà dello schermo con questo sistema: vengono sottratti 40 pixel dalla metà della larghezza dello schermo (left: MediaQuery.of(context).size.width / 2 - 40), il bottone ne è lungo 80 quindi in questo modo è sempre centrato.

Al di sopra di esso è posta la scritta "Press Me" e attraverso il parametro onPressed stampa a schermo "Button Pressed" come funzione di debugging. Il bottone è posto al di sopra del video attraverso la modalità "Stack" del body del widget scaffold contenente anche il video, che pone un widget sopra ad un altro nel caso sia stato descritto dopo.

Dopo qualche breve test si è constatato che il bottone effettivamente funzionava e produceva le scritte "button pressed" sul terminale di debug.

```
156             // Button on top of the video
157             Positioned(
158               bottom: 20.0, // Adjust the distance from the bottom as needed
159               left: MediaQuery.of(context).size.width / 2 -
160                   40, // Center horizontally
161               child: ElevatedButton(
162                 onPressed: () {
163                   // Add your button action here
164                   print('Button Pressed');
165                 },
166                 child: Text('Press Me'),
167                 style: ElevatedButton.styleFrom(
168                   padding: EdgeInsets.symmetric(horizontal: 20.0, vertical: 10.0),
169                 ),
170               ), // ElevatedButton
171             ), // Positioned
```

Figura 9:Sketch sezione di codice che implementa un semplice bottone

3.2.6 Unione effettiva dei codici

In questa sezione è descritta l'implementazione finale della funzionalità video con il resto della applicazione. Per unire i codici si è pensato di trasferire i comandi descritti in "command.dart" all'interno del file "webcam.dart".

All'inizio vengono quindi riportate le inclusioni dei package presenti in entrambi i file, e subito dopo le costanti che erano definite all'inizio di command.

All'interno della definizione della classe principale vengono ridefinite le variabili globali contenute nel file "globals.dart" (final Globals globals), e il costruttore Webcam in questo caso prende come input le variabili globali globals (Webcam({required this.globals})), esattamente come in command.dart.

Nello stato della classe, "_WebcamState", troviamo la definizione di altre costanti dedicate al comando del veicolo, già presenti in command. La funzione initState si presenta identicamente a quella descritta in "command.dart".

Ovviamente è presente la definizione del canale channel attraverso il quale l'applicazione si connette al Web Socket. E anche le funzioni dispose(), _sendData() e loadConfigVariables() non variano rispetto ai file precedenti, e sono presenti per mantenere indipendentemente le vecchie funzioni.

Il widget sfrutta ciò che ha dimostrato l'esperienza dell'aggiunta del bottone "Press Me": l'utilizzo del body: Stack nello Scaffold permette di inserire comandi sovrastanti al video, che rimane sullo sfondo. Quindi ogni comando è inserito in parallelo al container del video attraverso due align per i joystick e due positioned, uno per il "Reverse Command" e uno per i due slider.

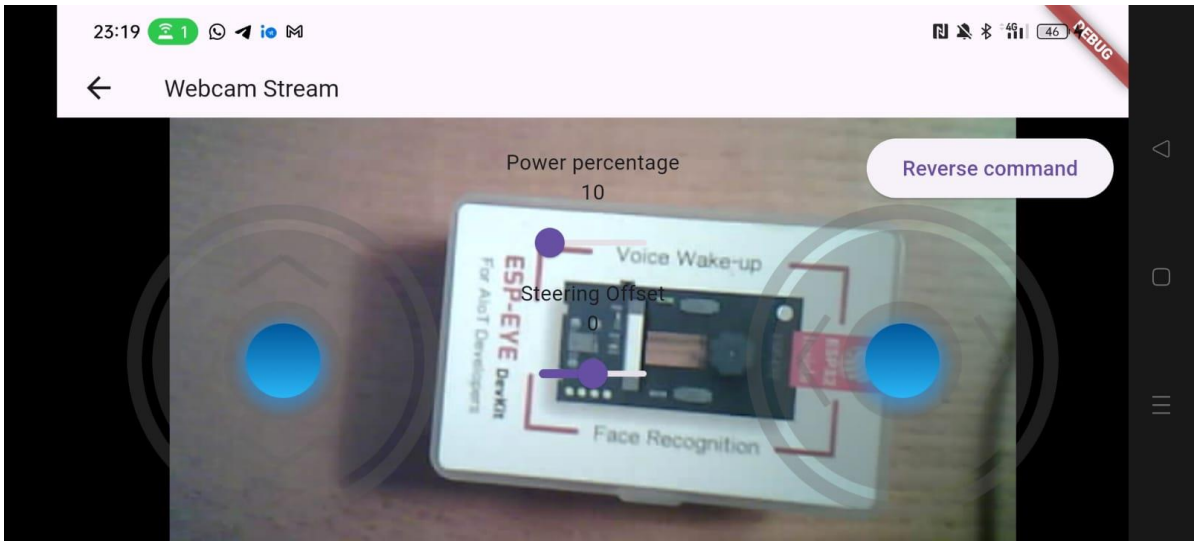

Figura 10: Schermata finale con tutte le funzionalità

3.2.7 Test funzionale del sistema e le ultime correzioni

Al netto dell'ottenimento dell'applicazione finale con tutte le funzionalità e adattamenti, il sistema è stato ovviamente sottoposto a test funzionale, in maniera tale da verificare il rispetto o meno delle specifiche citate nel capitolo introduttivo. L'ESP-EYE è stato posizionato sul parafango anteriore della macchina Traxxas in maniera che l'utente possa visualizzare il punto di vista davanti al veicolo, e il microcontrollore è stato alimentato provvisoriamente attraverso un powerbank, posto sotto la scocca in plastica, e un cavo USB per collegarli.

Figura 11: Scheda ESP-EYE posizionata sul parafango della macchinina Traxxas

Durante questa fase ci si è concentrati, oltre che sull'adattamento fisico, soprattutto sulla qualità generale del video rappresentato a schermo; infatti, sono state necessarie alcune correzioni sia al codice dell'applicazione sia al codice dell'ESP-EYE, al fine di migliorarne la visibilità tenendo conto del fatto che il sistema è in movimento durante la guida.

Le prime correzioni si sono basate sul correggere dei problemi dal punto di vista del layout del video, cioè hanno portato ad una revisione del codice di webcam.dart affinché venissero migliorati questi due punti:

- Ai lati dello schermo si presentavano due grosse bande nere, che limitavano la visibilità del video.
- La scritta “Webcam Stream” al di sopra del video si presentava all’interno di una “appbar” (barra bianca contenente un testo) troppo grande che portava ad una ulteriore riduzione della porzione di video visibile.

Attraverso poche righe di codice sono stati migliorati questi punti, inserendo dei metodi di definizione della altezza e larghezza del widget video, che di default si presentava molto piccolo, e riducendo le dimensioni della “appBar” modificando due parametri all’interno della sua definizione.

Ma l’intervento più importante è stato sicuramente nella scelta della qualità dal lato della risoluzione e del frame rate; è stato svolto un lavoro di verifica di quale fosse il compromesso tra questi due elementi. Infatti, una scelta di una maggiore risoluzione comporta sicuramente un calo di prestazioni dal punto di vista del numero di immagini al secondo captate, elaborate e trasmesse dall’ESP-EYE. La risoluzione iniziale che risultava sull’app (QVGA, 320 x 240) era certamente bassa, ma assicurava un buon numero di frame per secondo al video.

Un altro parametro fondamentale per la gestione della qualità dell’immagine ricevuta è sicuramente la JPEG quality, che influenza quanto viene compresso il file dell’immagine prima di essere inviato su Web Socket. Esso varia da 0 a 63, in questo caso è stato fissato a 10.

Per modificare la risoluzione è stato necessario intervenire sul codice dell’ESP-EYE, utilizzando il metodo "s->set_framesize(s, "risoluzione")" dove "risoluzione" è un parametro tra questi disponibili in ordine crescente di qualità visiva:

- FRAMESIZE_QVGA (320 x 240) 25 – 30 fps
- FRAMESIZE_CIF (352 x 288) 20-25 fps
- FRAMESIZE_VGA (640 x 480) 10-15 fps
- FRAMESIZE_SVGA (800 x 600) 8-12 fps
- FRAMESIZE_XGA (1024 x 768) 5- 8 fps
- FRAMESIZE_SXGA (1280 x 1024)3-6 fps
- FRAMESIZE_UXGA (1600 x 1200)1-3 fps

Sono stati eseguiti numerosi test per giudicare quale sarebbe stato il miglior compromesso tra qualità dell'immagine e frame rate del video. E il risultato dipende sicuramente dal tipo di guida che si vuole adottare: una guida più veloce in spazi larghi richiederà sicuramente più frame rate e meno risoluzione, la risoluzione CIF o QVGA potrebbe essere le più indicate, mentre in una guida più lenta e articolata in spazi piccoli sarà possibile sacrificare maggiormente la quantità di immagini al secondo visualizzate per una qualità maggiore visiva, nell’ottica di una futura aggiunta di un algoritmo di object detection; in quest’ultimo caso risoluzioni più adatte potrebbero essere VGA o SVGA. Risoluzioni maggiori inciderebbero troppo sulla quantità di frame al secondo per una guida sicura, ma potrebbero essere utili in altri ambiti dell’utilizzo IoT e AI dell’ESP-EYE.

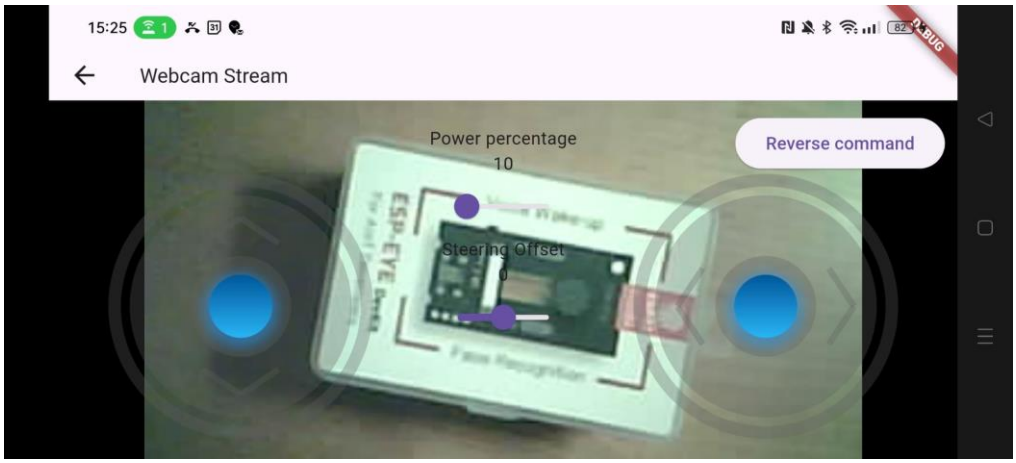

Figura 52: Interfaccia video con qualità QVGA

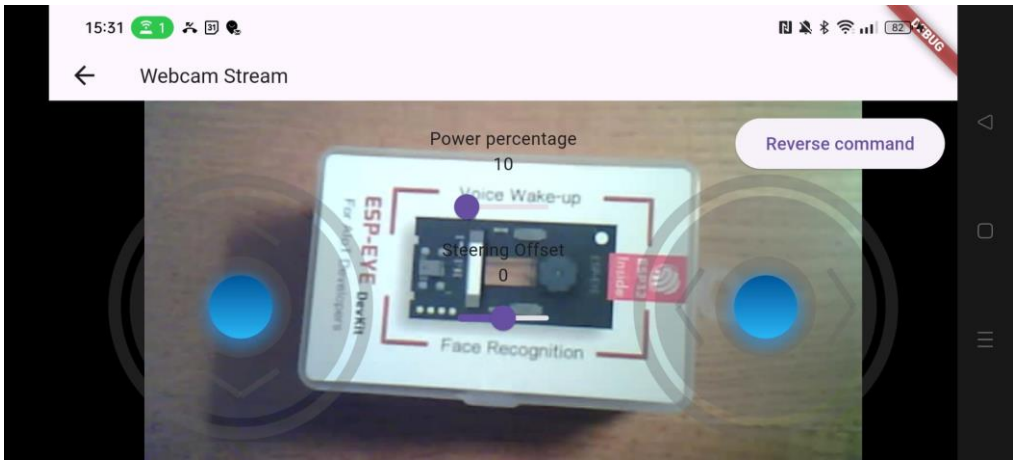

Figura 4:Interfaccia video con qualità VGA

I comandi presenti all'interno della applicazione, non risultano invadenti alla visualizzazione del video, in quanto i due joystick sono per lo più trasparenti mentre gli altri pulsanti non interferiscono in quanto sono non trasparenti, ma piccoli a sufficienza da non risultare fastidiosi o sgradevoli alla vista e alla guida. Per questo motivo non è stato necessario alcun tipo di adattamento della forma dei comandi a schermo.

4 Contributo personale e considerazioni conclusive

4.1 Possibili miglioramenti

Nonostante i risultati ottenuti, sicuramente il sistema presenta dei limiti che sono dettati da questioni progettuali logistiche e di scelta dell'hardware. Infatti, possiamo schematizzare le limitazioni in queste due categorie sopracitate:

- Hardware, il sistema sfrutta l'utilizzo di schede a basso consumo come l'ESP-EYE e l'arduino 33 BLE, che di contro possiedono poca potenza di calcolo, che limita la quantità di dati elaborabili e scambiabili. Infatti, se l'ESP-EYE avesse avuto più capacità di invio di dati, sarebbe stato possibile innalzare ulteriormente la qualità dell'immagine rappresentata sullo schermo; inoltre, ciò limita anche la possibilità di inserire algoritmi computazionalmente complessi, per esempio di object detection, questo argomento sarà approfondito nel paragrafo successivo.
- Logistica, lo scambio di dati da ESP-EYE a WebSocket e da WebSocket a dispositivo dell'utente avviene all'interno di una LAN, il che significa che l'ESP-EYE e il dispositivo devono essere connessi alla stessa rete

WiFi affinché il video possa essere riprodotto a schermo; quindi, la limitazione di distanza fisica non è solo data dalla connessione bluetooth.

Per quanto riguarda l'hardware sarebbe sicuramente possibile trovare delle alternative valide all'ESP-EYE con una potenza di calcolo più elevata, per esempio una NVIDIA Jetson Nano (con collegata una telecamera compatibile attraverso un connettore CSI), oppure un Raspberry Pi 4 con modulo camera. Ovviamente questa soluzione aumenterebbe il consumo energetico totale del sistema, ma permetterebbe l'implementazione di algoritmi ben più complessi di quelli che può supportare l'ESP-EYE.

Invece, per risolvere il problema di logistica, esistono svariati metodi; è possibile eseguire il port forwarding, cioè l'instradamento del traffico in entrata del router da una porta specifica dell'IP pubblico (quello del router) verso un dispositivo all'interno della rete locale (in questo caso, l'ESP-EYE). Ma potrebbe risultare più comodo e sicuro, per esempio, la configurazione di un server VPN sul router, che consente di connettersi alla rete locale dall'esterno, e quindi al WebSocket.

4.2 Sviluppi futuri

Le possibili evoluzioni che il progetto in questione offre sono molteplici, in questo capitolo ne analizziamo le più probabili e utili.

Sicuramente, uno sviluppo futuro potrebbe essere l'inserimento di un sistema di object detection in grado di riconoscere e mostrare a schermo attraverso una bounding box (rettangolo circostante all'oggetto riconosciuto), che si può basare o sulle capacità fornite dall'ESP-EYE (molto limitate) o su quelle del dispositivo su cui gira l'applicazione flutter. Per quanto riguarda il primo caso, come già citato nel capitolo dedicato, l'ESP-EYE possiede una predisposizione progettuale a piccoli algoritmi di object detection e recognition, ma si tratta di algoritmi con grossi limiti: piccoli dataset, accuracy e variabilità degli oggetti riconoscibili. Affinché il microcontrollore possa riconoscere svariati oggetti bisognerebbe sacrificare ancora più accuracy, senza contare che deve anche eseguire il compito assegnatogli in questo progetto di tesi: lo streaming video, che occupa già il 32 % della memoria dello sketch.

Mentre nel caso in cui l'algoritmo fosse definito all'interno della applicazione, potrebbe senza dubbio avere meno limitazioni di quel genere, avendo a disposizione la potenza di calcolo del dispositivo mobile su cui gira.

Nel caso si riuscisse, con uno dei due metodi ad avere un algoritmo di object detection si potrebbe anche pensare di arrivare all'obiettivo della guida autonoma o semi-autonoma, che cambierebbe radicalmente il progetto. In quel caso, però, se l'algoritmo dovesse funzionare anche in mancanza di connettività con l'applicazione dovrebbe girare sull'Arduino Nano 33 BLE Sense che, in questa fase del progetto, ha il solo compito di ricevere i comandi e trasmetterli ai motori; esso non possiede le risorse computazionali necessarie per fare ciò. Quindi sarebbe necessario un upgrade del "cervello" della macchina, utilizzando per esempio un Raspberry Pi per questo compito.

Un'altra prospettiva di sviluppo potrebbe essere l'inserimento a schermo di un widget mostrante la batteria della macchina, che può essere misurata tramite un pin di input dell'Arduino, che però, potendo assumere in ingresso un valore massimo di tensione (di 3.3 V o 5 V a seconda del pin) necessita di un sistema di adattamento della tensione da batteria a Arduino. Esso può poi sfruttare la già presente connessione BLE per inviare il dato alla applicazione che lo aggiornerà periodicamente.

4.3 Commento finale

Questo progetto di tesi mi ha permesso di acquisire diverse competenze di programmazione e pratiche, attraversando diversi ambiti della progettazione:

- Programmazione e gestione di dispositivi embedded IoT, infatti è stato necessario lavorare su codice per microcontrollore rispettando i limiti tecnologici che questo mondo impone. Nello specifico i limiti imposti dal microcontrollore ESP-EYE, cioè pochi MB di PSRAM e di flash, oltre a una ridotta potenza di

calcolo. Questa competenza migliorata ha accompagnato, nello stesso contesto, quella del lavoro su ambiente di lavoro specifico: l'Arduino IDE 2.3.3, utilizzando il linguaggio di programmazione C++.

- Analisi delle caratteristiche degli elementi utilizzati per il sistema e delle interazioni fra loro, quali l'ESP-EYE, l'Arduino 33 BLE e il veicolo: la macchina Traxxas. È stata sicuramente importante la fase di valutazione di come trasmettere il flusso video.

- Acquisizione, trasmissione e ricezione, attraverso Web Socket, di immagini utilizzando i metodi forniti dalle librerie orientate ai server Web Socket sia su framework Flutter sia su ambiente di lavoro come l'Arduino IDE.

- Implementazione di nuove funzionalità su applicazione mobile all'interno del framework Flutter nell'ambiente di sviluppo Visual Studio Code. Questa parte di lavoro è sicuramente stata didattica per quanto riguarda la programmazione dart in Flutter, che presenta importanti differenze da quella che è stata introdotta nei corsi di programmazione C, C++ e assembly presentati nel corso della triennale, un esempio lampante ne sono la definizione dei widget e degli stati delle classi. Inoltre, ha rappresentato un'occasione di dimostrazione dell'approccio modulare per quanto riguarda la programmazione; infatti, ogni funzionalità è stata prima creata a parte e poi inserita in un secondo momento, dalla acquisizione del video da WebSocket a App, alla creazione di uno stadio intermedio di progettazione di un semplice bottone per poi evolverlo alla soluzione finale.

- Utilizzo di un sistema di versioning: Source Tree; si tratta di un programma dedicato al tracciamento delle versioni di codice, che tiene conto di ogni modifica fatta ad ogni file del progetto tenuto sotto tracciamento. Il versioning rimane all'interno del dispositivo nella quale si lavora se si esegue un "commit", ma, se si collega il lavoro ad un progetto GitHub, è possibile anche salvare su internet il lavoro svolto, attraverso un "push". Questo sistema è molto utilizzato in ambito professionale.

Nel complesso il progetto ha sicuramente assunto uno scopo sia di ricerca sia didattico, in quanto ha permesso un'evoluzione all'applicazione dal punto di vista pratico e funzionale, ma soprattutto ha permesso di capire in che cosa consista lavorare sull'aggiunta di funzionalità per un sistema di questo tipo.

I contributi personali sono stati:

- Inserimento nel sistema di un metodo di acquisizione e trasmissione del video che è consistito nell'aggiunta fisica dell'ESP-EYE e nella progettazione del codice per esso, tenendo conto delle specifiche.
- Implementazione della nuova funzionalità all'interno della applicazione mobile, ossia inserimento dell'interfaccia video mantenendo quella dei comandi al di sopra di essa.

5 Riferimenti bibliografici

1. Libreria arduinoWebSockets: https://github.com/Links2004/arduinoWebSockets/tree/master

2. Github da cui son stati ricavati i valori dei pin: https://github.com/espressif/arduino esp32/blob/master/libraries/ESP32/examples/Camera/CameraWebServer/camera pins.h

3. Scelta della risoluzione e altri parametri dell'immagine: https://randomnerdtutorials.com/esp32cam-ov2640-camera-settings/

4. Libreria Wifi arduino: https://www.arduino.cc/reference/en/libraries/wifi/

5. Linguaggio Dart: https://it.wikipedia.org/wiki/Dart_(linguaggio_di_programmazione)

6. Libreria quickblue: https://pub.dev/packages/quick blue

7. Web socket channel package flutter:https://docs.flutter.dev/cookbook/networking/websockets

8. Documentazione Arduino Nano 33 BLE Sense: https://docs.arduino.cc/hardware/nano-33-ble-sense/

6 Ringraziamenti

Ci tengo a ringraziare il professore Riccardo Berta e i dottori Matteo Fresta e Luca Lazzaroni che mi hanno assistito nello svolgimento di questa tesi. Un ringraziamento anche alla mia famiglia e ai miei amici per essermi stati vicini nel periodo universitario.