Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Ingegneria Informatica, Comunicazioni ed Elettronica

Elaborato finale

Visualizzazione e analisi dei dati per sistemi di localizzazione UWB

Supervisore

Gian Pietro Picco

Co-supervisore

Davide Vecchia

Laureando

Nicolò Fadigà

Anno accademico 2023/2024

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Ingegneria Informatica, Comunicazioni ed Elettronica

Elaborato finale

Visualizzazione e analisi dei dati per sistemi di localizzazione UWB

Supervisore

Gian Pietro Picco

Co-supervisore

Davide Vecchia

Laureando

Nicolò Fadigà

Anno accademico 2023/2024

Ringraziamenti

Voglio ringraziare di cuore i miei genitori per avermi sempre sostenuto e per non aver mai fatto pesare gli insuccessi che si sono presentati lungo questo percorso. Un grazie speciale alla mia ragazza Matilde, che mi ha supportato nei momenti di difficoltà. E infine, un grande ringraziamento a tutti i miei amici: senza di voi, tutto questo percorso sarebbe stato molto più difficile e decisamente più solitario.

Indice

Sommario

I sistemi di localizzazione indoor sono fondamentali in numerosi settori, ma la complessità degli ambienti chiusi rende difficile ottenere dati precisi. Tecniche come il Time Difference of Arrival (TDoA) e l'uso della tecnologia Ultra Wideband (UWB) hanno migliorato la precisione, ma manca uno strumento intuitivo per visualizzare e analizzare questi dati. Questa tesi presenta lo sviluppo di TALLA Visualizer, un'applicazione desktop concepita per facilitare la visualizzazione e l'analisi dei dati TDoA raccolti attraverso il sistema TALLA, realizzato dal gruppo di ricerca che ha supervisionato questo lavoro. L'obiettivo è fornire uno strumento user-friendly che faciliti l'identificazione di anomalie e supporti gli operatori nel monitoraggio dei sistemi di localizzazione. Per realizzare TALLA Visualizer, sono state intraprese diverse attività chiave. In primo luogo, è stata condotta un'analisi dei requisiti approfondita, identificando le esigenze specifiche degli utenti e definendo le funzionalità fondamentali dell'applicazione. Ciò ha permesso di stabilire una solida base su cui costruire il progetto, assicurando che lo strumento soddisfacesse le reali necessità degli operatori nel settore. Successivamente, si è passati alla progettazione e implementazione dell'applicazione. È stata sviluppata un'architettura software modulare, utilizzando tecnologie come Electron e React per l'interfaccia utente, garantendo un'esperienza fluida e interattiva. Per l'elaborazione dei dati, sono stati impiegati Python e librerie come Pandas, permettendo una gestione efficiente di grandi dataset. Particolare attenzione è stata dedicata all'ottimizzazione delle prestazioni, implementando strategie come il lazy loading e la segmentazione dei dati, al fine di migliorare la reattività dell'applicazione e ridurre l'utilizzo delle risorse di sistema. Il processo di sviluppo ha incluso una fase rigorosa di testing e debugging. Sono stati effettuati test su diversi sistemi operativi, tra cui Windows e Linux, per garantire la compatibilità e l'affidabilità dell'applicazione in vari ambienti. Inoltre, sono state raccolte osservazioni dagli utenti finali, apportando miglioramenti all'usabilità e correggendo eventuali anomalie riscontrate. Questo approccio ha permesso di affinare il software, assicurando che rispondesse efficacemente alle esigenze degli utilizzatori.
I risultati ottenuti dimostrano che TALLA Visualizer è uno strumento efficace per la visualizzazione e l'analisi dei dati TDoA, migliorando l'efficienza nell'interpretazione dei dati e nell'identificazione di problemi. Il contributo personale si evidenzia in tutte le fasi del progetto, dalla concezione iniziale alla realizzazione finale, comprendendo la definizione dei requisiti, la progettazione dell'architettura, l'implementazione delle funzionalità e le attività di testing e debugging. Questo lavoro apre la strada a futuri sviluppi, come l'integrazione di funzionalità per la visualizzazione in tempo reale e l'implementazione di algoritmi di rilevamento automatico degli errori, contribuendo così al miglioramento delle pratiche nel campo della localizzazione indoor.

1 Introduzione

I sistemi di localizzazione rivestono un ruolo fondamentale in numerosi ambiti applicativi, dalla navigazione e tracciamento alla gestione delle risorse sia in ambienti esterni che interni. Mentre i Sistemi Globali di Navigazione Satellitare (GNSS), come il GPS, offrono soluzioni affidabili per la localizzazione all'aperto, l'ambiente indoor presenta sfide significative a causa dell'attenuazione del segnale e degli effetti di multipath. In questo contesto,

le tecniche basate sul Time Difference of Arrival (TDoA) sono emerse come soluzioni efficaci per la localizzazione indoor, sfruttando tecnologie come l'Ultra Wideband (UWB) per raggiungere elevati livelli di precisione.
Nonostante i progressi nelle tecnologie di localizzazione, esiste una carenza di strumenti intuitivi e user-friendly per la visualizzazione e l'analisi dei dati TDoA. I software esistenti spesso presentano una curva di apprendimento ripida e una complessità che può ostacolare l'interpretazione efficiente dei dati e il troubleshooting.
Questa tesi affronta la necessità di uno strumento intuitivo per la visualizzazione dei dati TDoA attraverso lo sviluppo di TALLA Visualizer, un'applicazione desktop progettata per semplificare l'analisi dei dati raccolti dal sistema TALLA. L'applicazione mira a fornire un'interfaccia utente intuitiva e funzionalità avanzate per facilitare l'identificazione di anomalie e migliorare l'esperienza complessiva nell'analisi dei dati.
Gli obiettivi principali di questa tesi sono:

- Analizzare i requisiti per un efficace strumento di visualizzazione dei dati TDoA.
- Progettare e implementare TALLA Visualizerutilizzando tecnologie moderne come Electron e React per l'interfaccia utente e Python per l'elaborazione dei dati.
- Valutare l'applicazione attraverso attività di testing e debugging per garantirne l'affidabilità e l'usabilità.

La tesi è strutturata come segue:

- **Capitolo 2:** Presenta il background teorico, fornendo informazioni sui sistemi di localizzazione, le tecniche TDoA, la tecnologia UWB e il sistema TALLA.
- **Capitolo 3:** Discute i lavori correlati e confronta TALLA Visualizercon strumenti esistenti come MATLAB, Navigine Tracking e QGIS.
- **Capitolo 4:** Dettaglia la progettazione di TALLA Visualizer, includendo gli obiettivi del progetto, i requisiti funzionali e non funzionali, le funzionalità proposte e l'architettura del sistema.
- **Capitolo 5:** Descrive l'implementazione di TALLA Visualizer, illustrando la struttura generale del codice, i dettagli del Main Process (Back-end) e del Renderer Process (Front-end), l'interazione tra front-end e back-end, la visualizzazione e la gestione dei dati. Inoltre, presenta i processi di testing e debugging impiegati per assicurare le prestazioni e l'affidabilità dell'applicazione.
- **Capitolo 6:** Esplora possibili estensioni e nuove funzionalità per sviluppi futuri, come la visualizzazione in tempo reale dei dati, lo sviluppo di algoritmi integrati di rilevamento degli errori e l'implementazione di visualizzazioni aggiuntive.
- **Capitolo 7:** Conclude la tesi, riassumendo i risultati ottenuti e delineando possibili direzioni per lavori futuri.

Questa struttura riflette il percorso intrapreso durante la realizzazione del progetto, partendo dalle basi teoriche fino ad arrivare all'implementazione pratica e alle prospettive di sviluppo. Ogni capitolo è concepito per fornire una comprensione approfondita degli aspetti chiave di TALLA Visualizer, mettendo in luce le scelte progettuali, le sfide affrontate e le soluzioni adottate.

2 Background

Un sistema di localizzazione è un insieme di tecnologie e metodologie utilizzate per determinare la posizione di un soggetto, che può essere un oggetto o una persona. Tali sistemi

possono operare sia in ambienti esterni, attraverso l'uso di satelliti e altre infrastrutture, sia in ambienti interni, utilizzando segnali radio, ultrasuoni, luce visibile e altre tecnologie. La precisione della localizzazione dipende dalla tecnologia impiegata e dalle condizioni ambientali.

Esistono diversi tipi di sistemi di localizzazione:

- **GNSS (Global Navigation Satellite Systems)**: sono sistemi che basano la localizzazione sull'utilizzo di dati satellitari, come il GPS (Global Positioning System), GLONASS (Russia), Galileo (Europa) e BeiDou (Cina). Questi sistemi sono ampiamente utilizzati per fornire una copertura globale e vengono impiegati nell'ambito della navigazione in ambienti esterni.

- **RTLS (Real Time Localization Systems)**: identificano un insieme di tecnologie che hanno come obiettivo comune la localizzazione in tempo reale, specialmente in aree circoscritte o in ambienti interni. Alcune di queste tecnologie includono Wi-Fi, Bluetooth, RFID e UWB.

- **Sistemi ibridi**: utilizzano contemporaneamente o alternativamente tecnologie diverse per ottenere prestazioni migliori indipendentemente dall'ambiente in cui vengono utilizzate. Ad esempio, vengono combinate tecnologie GNSS e RTLS per ottenere buone prestazioni sia all'interno di edifici sia all'esterno.

2.1 TDoA

Il TDoA (Time Difference of Arrival) è una tecnica di localizzazione che "sfrutta le differenze di tempo nella propagazione dei segnali, misurate alla ricezione" [12]. Affinché ciò sia possibile, è necessario che i nodi fissi (ancore) siano sincronizzati temporalmente tra loro. Esistono vari modi di effettuare la sincronizzazione tra le ancore, ad esempio:

- **Sincronizzazione tramite segnali di riferimento:** In questo approccio, un'ancora viene scelta come riferimento e invia periodicamente segnali di sincronizzazione agli altri nodi. Questi segnali aiutano a calcolare gli offset e le derive degli orologi, permettendo di predire l'alterazione delle misurazioni nel tempo e determinare la frequenza necessaria del processo di sincronizzazione affinché l'errore rimanga accettabile [10].

- **Sincronizzazione mediante clock di riferimento esterno:** Un'alternativa è fornire a tutte le ancore un clock di riferimento esterno, come un segnale GPS o un clock distribuito attraverso una rete cablata. Questo metodo consente alle ancore di mantenere una sincronizzazione temporale precisa senza la necessità di scambiare segnali di sincronizzazione tra di loro [11][6].

Una volta che le ancore sono sincronizzate, quando un nodo mobile invia un beacon, le ancore registrano il proprio timestamp di ricezione e lo inviano al server risolutore TDoA, "il quale si occupa di traslare i timestamp nel dominio temporale di riferimento e successivamente di convertirli in distanze fisiche" [10]. Le equazioni relative a queste distanze corrispondono a equazioni di iperboli.
Calcolando almeno tre di queste iperboli, si ottiene un sistema di equazioni non lineari, la cui soluzione, ottenuta attraverso metodi numerici come il metodo dei minimi quadrati, fornisce le coordinate del nodo mobile.
La precisione della sincronizzazione è fondamentale per la localizzazione TDoA. Un errore di 1 ns nel timestamp si traduce in un errore di circa 30 cm nella stima della distanza [10]. Per questo motivo, per semplificare l'intercettazione di questi errori, si rende necessario l'uso di uno strumento di visualizzazione dei dati in grado di fornire mezzi per l'identificazione di anomalie, riducendo così i tempi di individuazione delle stesse.

2.2 UWB

L'UWB (Ultra Wideband) è una tecnologia di comunicazione wireless per la trasmissione di dati, che utilizza un'ampia gamma di frequenze per trasmettere segnali su brevi distanze. Grazie all'impiego di uno spettro elettromagnetico molto ampio, l'UWB è in grado di trasportare grandi quantità di dati fino a 70 metri di distanza, consumando quantità minime di energia elettrica. Inoltre, è capace di trasmettere segnali che oltrepassano pareti e muri, rendendola ideale per spazi chiusi [4].

Questa tecnologia funziona attraverso lo scambio di impulsi digitali tra sorgente e ricevitori a intervalli di tempo estremamente precisi, richiedendo una sincronizzazione accurata a livello di trilionesimi di secondo. Questa precisione ha due effetti importanti: la trasmissione quasi istantanea di grandi quantità di dati e la possibilità di misurare con grande precisione il tempo di transito dei segnali, determinando con un margine di pochi centimetri la distanza tra trasmettitore e ricevitor [4].

L'UWB sta trovando sempre maggiore applicazione nell'ambito della localizzazione indoor grazie alle sue caratteristiche distintive. In particolare, il suo spettro radio non subisce interferenze da altri tipi di segnali come Wi-Fi o Bluetooth. Inoltre, la brevissima durata degli impulsi UWB minimizza l'effetto multipath, poiché riduce la probabilità che il segnale venga riflesso da ostacoli multipli. Questo permette di identificare con maggiore precisione il percorso diretto del segnale e di fornire una stima accurata del Time of Flight (ToF), migliorando l'accuratezza nella determinazione della posizione [12].

2.3 TALLA

TALLA è la tecnologia con cui sono stati raccolti i dati da visualizzare tramite il software oggetto di questa tesi. Essa si colloca nell'ambito della localizzazione indoor e rappresentando un miglioramento significativo rispetto alla tecnica TdoA precedentemente descritta. Infatti, nei sistemi TDoA inizialmente la sincronizzazione tra i nodi avveniva tramite costosi collegamenti cablati, nel corso del tempo sono emerse alcune soluzioni wireless, però limitate per quanto riguarda l'area coperta. TALLA (Time Difference of Arrival Localization for Large-scale Areas) [10] prende il concetto di sincronizzazione wireless e la estende a reti e spazi arbitrariamente grandi, senza dover scendere a compromessi sulla precisione delle posizioni. Questo è reso possibile dall'implementazione della metodologia TDMA (Time Division Multiple Access), che suddivide il tempo di accesso al canale di trasmissione in slot, distribuendoli equamente tra i nodi e consentendo a quest'ultimi di comunicare evitando collisioni, migliorandone così l'affidabilità e frequenza di aggiornamento.
La sincronizzazione riveste un ruolo fondamentale in TALLA ed è ottenuta attraverso la trasmissione periodica di beacon tra le ancore. Ogni ancora trasmette un beacon nel proprio slot temporale, permettendo alle altre di sincronizzare il proprio orologio rispetto ad essa. Inoltre, i messaggi di sincronizzazione consentono di raggiungere una sincronizzazione temporale del server con una precisione fino a frazioni di nanosecondo [10]. I beacon trasmessi dalle ancore contengono timestamp precisi e informazioni di identificazione, come l'ID del nodo. Queste informazioni vengono archiviate dal server in una struttura dati che permette di scegliere qualsiasi ancora come riferimento temporale per il calcolo della posizione.
Considerando il tipico drift degli orologi, la frequenza di sincronizzazione necessaria per ottenere una misurazione TDoA accurata deve essere superiore a $1Hz$; pertanto, la finestra di trasmissione complessiva deve essere sufficientemente breve da ridurre l'errore di posizione [10]. L'ancora scelta come riferimento è quella con il maggior numero di nodi sincronizzati; in caso di parità, viene selezionata quella sincronizzata più recentemente.
Una volta sincronizzate, alla trasmissione di un beacon di localizzazione da parte di un tag, le ancore registrano il timestamp di ricezione e lo inviano al server il quale si occuperà di calcolare la posizione del nodo e validarla: se la posizione risulta essere fuori dal range

di comunicazione, questa viene considerata invalida e l'operazione di localizzazione viene ripetuta con un sottoinsieme di ancore finché il risultato non verrà considerato valido [10].

3 Lavori correlati

Esistono numerosi software per la simulazione e la visualizzazione di traiettorie, molti dei quali risultano complessi e poco intuitivi, rendendo difficile l'approccio da parte degli utenti. TALLA Visualizer è un'applicazione desktop che mira a semplificare la visualizzazione dei dati TDoA, fornendo un'interfaccia utente intuitiva e piacevole, facilitandone così l'utilizzo. Di seguito viene presentata una comparazione con gli strumenti maggiormente correlati al programma oggetto di studio.

3.1 Navigine tracking

Navigine Tracking è un programma per il miglioramento della produttività, sviluppato dall'azienda omonima. Pensato per le aziende, questo prodotto permette di monitorare in tempo reale risorse e dipendenti, sia in ambienti interni che esterni, ottimizzando i flussi di lavoro attraverso una piattaforma web intuitiva. Le sue principali funzionalità includono la localizzazione in tempo reale, la possibilità di riprodurre la cronologia di tracciamento, l'analisi dei dati storici e il monitoraggio degli oggetti sia indoor che outdoor [7]. Questo sistema, tuttavia, necessita di appoggiarsi su un'infrastruttura RTLS affinché la piattaforma funzioni. Sebbene alcune funzionalità siano simili a quelle del progetto qui trattato, Navigine pone maggiore attenzione all'aspetto del miglioramento del flusso di lavoro aziendale, con molte caratteristiche specializzate in quell'ambito. TALLA Visualizer, invece, è concepito con l'obiettivo di migliorare l'esperienza di risoluzione degli errori e facilitare la lettura dei dati, grazie a un ambiente particolarmente semplice e facile da utilizzare.

3.2 QGIS

QGIS è un software open-source per la gestione delle informazioni geografiche (GIS). È utilizzato per visualizzare, creare, modificare e analizzare dati geospaziali. Le sue funzionalità di spicco includono la gestione di formati diversi, permettendo di importare ed esportare dati facilmente, la visualizzazione cartografica, l'analisi spaziale e l'estensibilità tramite plugin creati dalla comunità, che consentono di integrare funzionalità specifiche. QGIS è spesso utilizzato per visualizzare e analizzare dati raccolti da molteplici fonti, inclusa la localizzazione indoor e outdoor.

Idealmente, TALLA Visualizer avrebbe potuto essere implementato come plugin di QGIS anziché come software standalone, ma ciò avrebbe comportato alcune limitazioni imposte dalla piattaforma:

- I plugin sono integrati nell'interfaccia esistente del software e sono quindi soggetti alle sue linee guida in termini di design, limitando la possibilità di offrire un'esperienza utente personalizzata.
- Il plugin avrebbe richiesto che il software principale fosse installato, il che potrebbe rappresentare una barriera per alcuni utenti non familiari con i sistemi GIS o il cui interesse riguarda esclusivamente la visualizzazione di dati TDoA.
- La curva di apprendimento per un utilizzo efficace del programma potrebbe costituire un ostacolo per alcuni utenti.
- Offrire un'esperienza utente moderna e intuitiva sarebbe stato limitato dalle componenti disponibili in PyQt, l'interfaccia grafica utilizzata da QGIS.

Per questi motivi, sebbene QGIS potesse rappresentare una valida opzione, si è preferito sviluppare un software autonomo che potesse essere realizzato in base alle specifiche necessità degli utenti finali.

4 Design

Nel capitolo precedente abbiamo analizzato le limitazioni dei software esistenti, le quali hanno portato alla luce la necessità di un software dedicato alla visualizzazione dei dati di localizzazione TDoA. In questo capitolo, invece, verrà trattato in dettaglio il design dello strumento di visualizzazione TALLA, progettato specificamente per soddisfare esigenze di questo tipo di sistemi, questa volta però in ambito universitario.

4.1 Obiettivi del progetto

L'obiettivo principale è stato quello di creare uno strumento intuitivo ed interattivo che permetta agli utenti di visualizzare ed analizzare le traiettorie di diversi dispositivi all'interno di un ambiente chiuso, permettendo di semplificare il processo di debug.

4.2 Requisiti

Il software è stato sviluppato ascoltando bisogni e necessità degli utenti finali, i quali hanno stilato una lista di requisiti che sono emersi sia durante lo sviluppo sia durante le loro esperienze passate.

4.2.1 Requisiti funzionali

- **Input dei dati:** l'applicazione deve essere in grado di importare dati in formato CSV, predisponendo il sistema ad importare altri dati in formato JSON che permetterebbero di importare impostazioni di visualizzazione predefinite dell'utente, in modo da poter avere viste personalizzate per diverse situazioni;

- **Riproduzione:** la riproduzione dei dati all'interno del grafico deve avvenire in una modalità simile a quella di un video player, ovvero con la possibilità di gestire le opzioni di riproduzione (play, pausa, aumento/diminuzione velocità riproduzione, ecc).

- **Grafico interattivo:** il grafico visualizzato deve essere interattivo, permettendo all'utente di effettuare uno zoom su specifiche aree del grafico.

- **Visuale fullscreen** possibilità di ingrandire il grafico fino a coprire l'intera finestra.

- **Ispezione dei dettagli:** durante la riproduzione, possibilità di visualizzare dati aggiuntivi relativi ad un'ancora o ad un tag selezionato.

- **Visualizzazione delle tracce:** rappresentazione di tracce multiple, ognuna associata ad un tag id. In ogni traccia devono essere riportati sulla timeline i momenti in cui il tag appare nel grafico.

- **Visualizzazione di sottoinsiemi di dati:** possibilità di poter selezionare un sottoinsieme di tag da voler visualizzare nel grafico

- **Registrazione e Taglio:** Modalità di registrazione per salvare la registrazione come clip video, rispettando le impostazione attuali (area di zoom, ID selezionati, velocità di riproduzione). Possibilità di scegliere un tempo di inizio e fine per registrare solo una parte specifica.

4.2.2 Requisiti non funzionali

- **Usabilità:** interfaccia utente fluida e reattiva in grado di trasmettere un'esperienza d'uso piacevole sia per utenti esperti sia per neofiti.
- **Portabilità:** Compatibilità con i principali sistemi operativi (Windows, macOS, Linux).
- **Prestazioni:** fornire una gestione efficiente di grandi quantità di dati senza incorrere in degradazioni significative delle prestazioni.
- **Manutenibilità:** strutturare il codice in maniera modulare e ben documentato per facilitare future estensioni o modifiche. Utilizzare tecnologie e librerie all'avanguardia per evitare di incorrere in errori dovuti alla deprecazione.
- **Scalabilità:** Capacità di gestire un numero crescente di tag e ancore senza impatti negativi sulle performance.

4.3 Funzionalità proposte

Alla luce dei requisiti funzionali e non funzionali delineati, sono state progettate una serie di funzionalità chiave per l'applicazione. Queste funzionalità mirano a soddisfare le esigenze degli utenti nel contesto della visualizzazione di dati TDoA per la localizzazione indoor, garantendo nello stesso momento prestazioni elevate ed un esperienza utente ottimale. Di seguito, vengono descritte in dettaglio le funzionalità proposte, evidenziando le motivazioni alla base di ciascuna e come esse contribuiscono al raggiungimento dell'obiettivo del progetto.

4.3.1 Gestione ed Elaborazione dei Dati

L'applicazione offre una robusta funzionalità di gestione ed elaborazione dei dati, consentendo agli utenti di importare facilmente dataset provenienti da sistemi TDoA. Supporta formati standard come CSV e JSON, assicurando flessibilità nell'integrazione con vari sistemi TDoA. I dati vengono elaborati per offrire all'utente diverse opzioni di visualizzazione, permettendogli di visualizzare i dati con differenti frequenze di fotogrammi (fps). Ciò si traduce concretamente nella compressione o nell'espansione dei dati all'interno dei frame, producendo una visualizzazione più o meno dettagliata a seconda delle necessità.

4.3.2 Pagine di selezione dell'input

La pagina di selezione degli input è il primo elemento che verrà visualizzato all'avvio del software. Essa consiste in un modulo (form) che permette di selezionare il file desiderato tra quelli presenti all'interno della cartella di lavoro, utilizzando un selettore ad albero (tree selector).Questo componente dell'interfaccia utente consente di navigare attraverso una struttura gerarchica di cartelle e file, facilitando la scelta del file appropriato.
Una volta selezionato il file, nel gruppo di radio buttons si evidenzieranno di azzurro le opzioni corrispondenti ai frame rate (FPS) che sono già state processate e sono quindi pronte all'uso. Le opzioni non ancora processate rimangono con il testo in nero e richiedono un'elaborazione preliminare, come si può notare in figura 4.1. Una volta selezionato il file e il frame rate desiderato e aver premuto il pulsante *Submit*, l'utente verrà indirizzato ad una nuova pagina. In questa sezione è possibile scegliere quali tag visualizzare, nel caso l'utente volesse seguire solamente uno specifico dispositivo, ad esempio per un debug più mirato. Questa funzionalità consente di migliorare le prestazioni del sistema, poiché si evita di elaborare e visualizzare una quantità eccessiva di dati non necessari.

4.3.3 Visualizzazione dei dati con grafico interattivo

Per offrire agli utenti un'interfaccia intuitiva e centralizzata per l'accesso alle varie funzionalità dell'applicazione, è stata sviluppata una dashboard principale. Come illustrato in Figura (4.2), essa consente di gestire la visualizzazione dei dati, selezionare diverse modalità di analisi e personalizzare le impostazioni in base alle esigenze. In particolare, il grafico

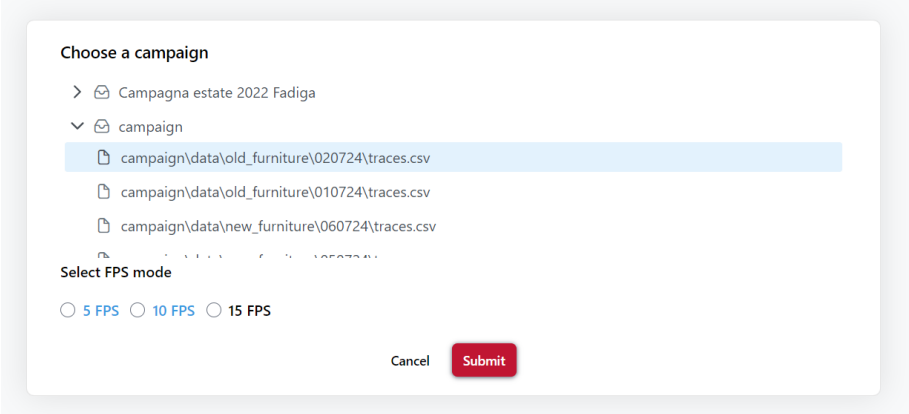

Figura 4.1: Pagina di selezione dell'input.

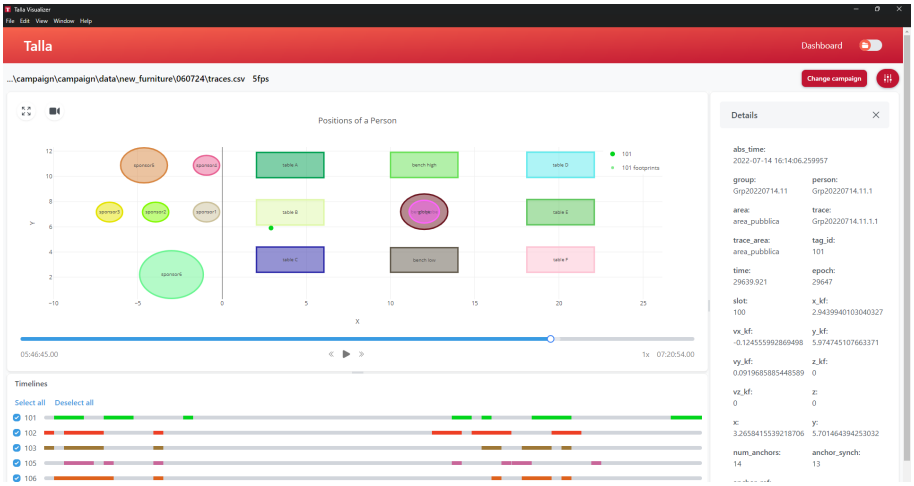

Figura 4.2: Dashboard di analisi dei dati, schermata presa dal software TALLA Visualizer.

è controllabile attraverso un player che consente di navigare nel tempo tramite la barra di avanzamento. È possibile avviare, fermare, avanzare o retrocedere la riproduzione di 30s utilizzando i comandi posti sotto il grafico. Nella stessa area è disponibile un pulsante per selezionare la velocità di riproduzione, variabile tra 0.5x e 4x, offrendo la possibilità di adattare la riproduzione in base al livello di dettaglio desiderato.
Il grafico è interattivo, consentendo all'utente di navigare al suo interno utilizzando il mouse. Nel menu che appare in seguito al passaggio del mouse sul grafico, sono disponibili alcuni strumenti come lo zoom, l'autoscale, il tasto per resettare gli assi, lo strumento lazzo, e la selezione rettangolare. Inoltre questi strumenti possono essere abilitati e disabilitati a piacimento all'interno del menu di personalizzazione della visualizzazione. All'interno dello stesso menu è presente un pulsante, rappresentato da un'icona di una fotocamera, che permette di catturare screenshot del grafico.
Nell'angolo superiore sinistro sono presenti due pulsanti: il primo consente di attivare alla modalità di visualizzazione a schermo intero, mentre il secondo avvia la registrazione dello schermo. Quando la registrazione è in corso, il pulsante mostra un quadrato rosso lampeggiante; per terminarla, è sufficiente cliccare nuovamente sullo stesso pulsante.

4.3.4 Visualizzazione dettagli tag

Cliccando su un tag presente nel grafico, comparirà a destra una sezione in cui sono presenti tutti i dati, relativi a quel tag. Quest'ultimi sono molto utili per svolgere processi di debug in quanto forniscono informazioni riguardanti:

- **Sincronizzazione temporale:** Come introdotto in 2.1 la sincronizzazione precisa tra le ancore è fondamentale. Dati come *epoch*, *slot*, *synch_epoch* e *synch_slot* aiutano

a verificare che la sincronizzazione sia mantenuta. Identificano possibili problemi che possono causare errori nella stima della posizione.

- **Qualità del segnale:** Informazioni sulle potenze dei segnali (*ref_rxpwr*, *ref_fppwr*, *receivers_rxpwr*, *receivers_fppwr*) permettono di valutare la qualità della comunicazione tra ancore e tag. Questo è essenziale per identificare interferenze, attenuazioni o problemi hardware che possono degradare le prestazioni del sistema.
- **Precisione della localizzazione:** Confrontando le coordinate non filtrate (x, y, z) con quelle filtrate (*x_kf*, *y_kf*, *z_kf*), è possibile valutare l'efficacia degli algoritmi di filtraggio e ottimizzazione. Residui elevanti o costi alti indicati in *residuals* e *cost* possono suggerire problemi nei dati o nell'algoritmo.
- **Gestione di errori o anomalie:** Campi come *status* e *timeout* forniscono segnali immediati su errori o condizioni anomale, facilitando l'individuazione di problematiche che richiedono attenzione.
- **Diagnostica delle ancore:** Sapere quali ancore hanno ricevuto il segnale (*anchor_rec*) e quali sono state utilizzate per la sincronizzazione e come riferimento (*anchor_synch*, *anchor_ref*) aiuta a identificare ancore malfunzionanti o mal posizionate che possono influenzare negativamente la localizzazione.

4.3.5 Timeline dei tag

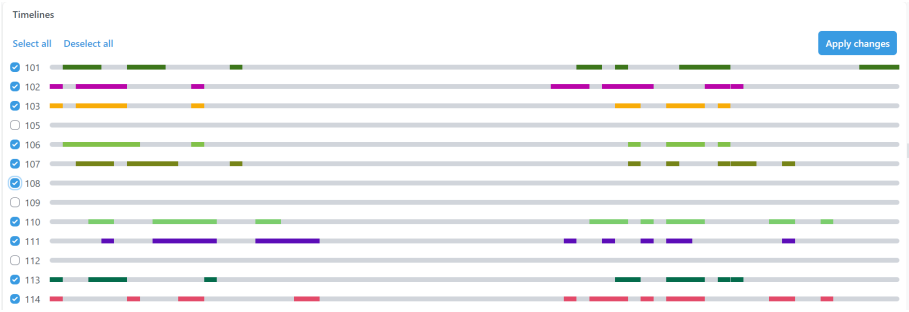

Figura 4.3: Primo piano del pannello delle timeline dei tag.

Nella sezione sottostante il grafico troviamo il pannello di gestione dei tag, nel quale sono elencati tutti i dispositivi registrati nella traccia caricata. Attraverso i pulsanti posti in alto, è possibile selezionare o deselezionare con un solo click tutti i tag; in alternativa, utilizzando le caselle di controllo (checkbox) presenti a sinistra di ciascuna timeline è possibile selezionarli individualmente. Una volta scelti i dispositivi desiderati, se la selezione differirà dalla precedente, apparirà un pulsante in alto a destra (*Apply changes*) che consente di applicare le modifiche. Dopo aver confermato le modifiche, la traccia verrà riprocessata e ricaricata. Come illustrato in figura 4.3, a destra di ogni etichetta indicante il tag è presente una timeline che al suo interno, tramite i segmenti colorati, indica i periodi in cui i dati relativi al dispositivo saranno visualizzati nel grafico. Cliccando su questi segmenti, la barra di progressione si allineerà al segmento selezionato, consentendo di navigare la traccia in modo più mirato ed efficiente.

4.3.6 Menu di personalizzazione

All'interno del grafico, è possibile aggiungere elementi come tavoli, panche e altri oggetti che riproducono la planimetria dell'ambiente in cui i dati sono stati raccolti. Questo avviene selezionando i file desiderati all'interno del menu di personalizzazione. Tale menu è accessibile tramite il pulsante raffigurante l'icona delle impostazioni, posto in alto a destra. Come si può osservare nell'immagine 4.4, il menu consente la gestione completa delle impostazioni del grafico e degli oggetti visualizzati. In particolare, nella sezione superiore, l'utente può selezionare il file relativo alle ancore e agli elementi ambientali che desidera

caricare, permettendo una personalizzazione completa della visualizzazione. Ciò consente di adattare i dati all'ambiente fisico in cui sono stati raccolti.
Inoltre, il menu fornisce opzioni per configurare diversi paramenti del grafico, come la disposizione della vista (ad esempio, impostando i limiti degli assi x e y) e l'attivazione o disattivazione di strumenti utili alla navigazione, quali lo zoom, il pan, il lasso e l'autoscale. È possibile anche abilitare o disabilitare l'auto-adattamento degli assi tramite l'opzione *Autorange* per garantire che i dati visualizzati siano sempre scalati correttamente. Nella sezione Shapes del menu, visibile nella parte inferiore dell'immagine, è possibile personalizzare ulteriormente gli oggetti ambientali e i tag, modificandone il colore di riempimento e il colore del bordo. Ogni riga della tabella corrisponde ad un oggetto specifico, come suggeriscono le etichette e include caselle di selezione che permettono di rendere visibile o nascondere gli oggetti selezionati. Questa flessibilità consente agli utenti di gestire quali elementi visualizzare nel grafico, migliorando cosi la chiarezza della visualizzazione.

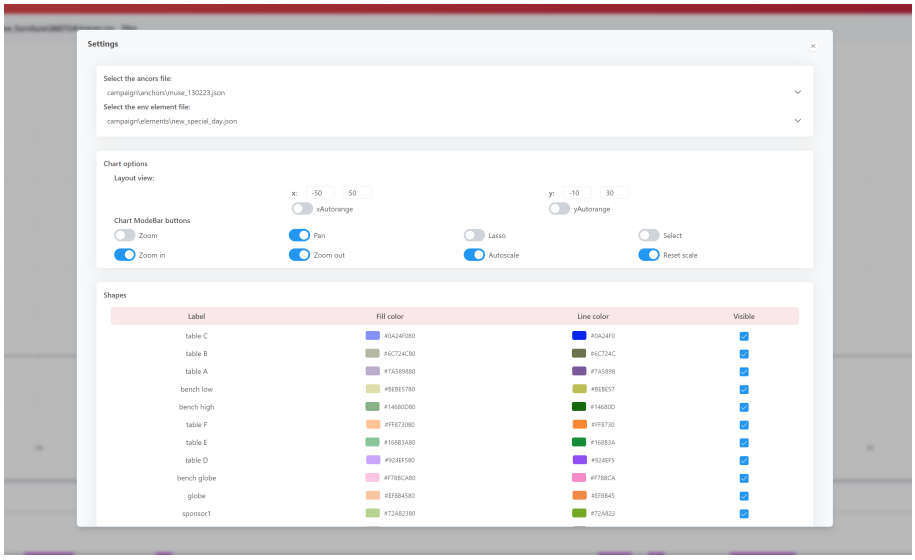

Figura 4.4: Menu di personalizzazione degli oggetti ambientali e delle impostazioni del grafico. L'utente può selezionare i file, configurare la vista e personalizzare colore e visibilità degli oggetti.

4.3.7 Ulteriori viste

Dopo aver selezionato un file per reperire ulteriori informazioni sulle ancore, all'interno della sezione dei dei dettagli dei tag (4.3.4), compariranno altre due opzioni di visualizzazione dei dati: la vista della qualità dei collegamenti e la vista delle iperboli.

La **vista delle qualità dei collegamenti** ha come scopo principale di controllare la qualità della comunicazione tra ancore e tag, cosi da poter porre rimedio a problemi di errato posizionamento delle ancore o a problemi di comunicazione che potrebbero causare errori all'interno del processo di calcolo della posizione. La qualità dei collegamenti viene rappresentata come segmenti che collegano il tag alle varie ancore (come illustrato in figura 4.5). che possono assumere un colore compreso all'interno del gradiente che va dal verde al rosso in base alla loro forza, mentre possono assumere una linea intera o tratteggiata secondo la seguente formula.

$$|fppwr| - |rxpwr| > 8$$

Il parametro *fppwr* (First Path Power) corrisponde alla potenza del segnale che arriva per primo al ricevitore, ovvero quello che percorre il percorso in Line of Sight (LOS), ed è ottimale se il valore si aggira attorno ai $-80dBm$ [9]. Il parametro *rxpwr* (Received Power) rappresenta la potenza totale del segnale ricevuto, inclusa sia la componente del primo percorso (*fppwr*), sia le componenti dovute a riflessioni e multipath. Se la differenza tra $|fppwr|$ e $|rxpwr|$ è superiore a $8dBm$, significa che gran parte della potenza

ricevuta è dovuta al multipath, il che è un'indicazione di una condizione di Non Line of Sight (NLOS). La presenza di NLOS tende a causare errori nella stima del timestamp, compromettendo la precisione del sistema di localizzazione. Al contrario, se $|fppwr|$ e $|rxpwr|$ sono simili, significa che la componente del primo percorso ($fppwr$) include la maggior parte della potenza ricevuta, indicando tendenzialmente una condizione di Line of Sight (LOS), favorevole per una localizzazione accurata.
I colori dei segmenti assumeranno un colore differente a seconda del valore di $fppwr$. Con $fppwr > -80dBm$ il segmento sarà verde, mentre con $fppwr < -110dBm$ il collegamento sarà rappresentato con il colore rosso acceso. Per i valori che saranno compresi in questo intervallo assumeranno dei colori intermedi indicativi della tendenza positiva o negativa, il colore cambierà ogni $5dBm$.

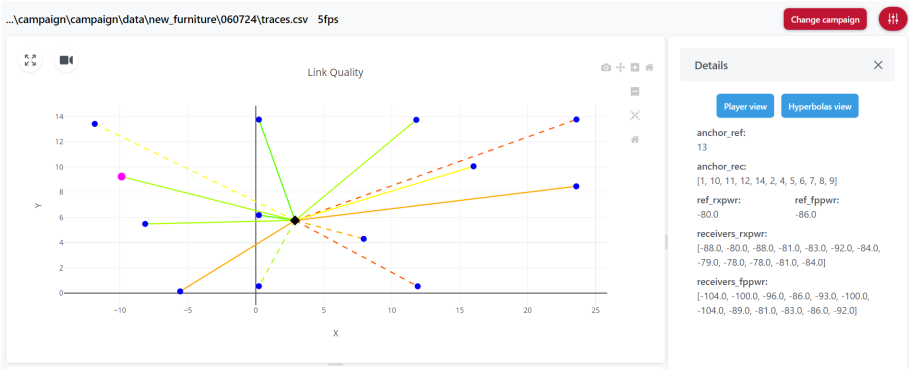

Figura 4.5: Vista della qualità dei collegamenti.

La **vista delle iperboli** serve a rappresentare graficamente le curve iperboliche calcolate dal sistema TDoA per la stima della posizione di un tag. Ogni iperbole descrive l'insieme di punti in cui il tag potrebbe trovarsi, calcolati sulla base della differenza di tempo di ricezione dei segnali tra diverse ancore. L'importanza di questa visualizzazione sta nel fatto che la posizione del tag può anche venire determinata dall'intersezione di due o più iperboli generate da coppie di ancore, consentendo di ottenere una stima precisa della sua localizzazione [2]. Le iperboli forniscono un mezzo visivo chiaro per identificare anomalie o errori nel processo di localizzazione. Se le iperboli non convergono correttamente in unico punto, o se alcune risultano significativamente lontane dall'intersezione con le altre, ciò potrebbe indicare un problema con una delle ancore, un errore nei dati di sincronizzazione o un malfunzionamento nell'hardware. Queste problematiche sono particolarmente importanti da identificare, in quanto influiscono direttamente sulla precisione del sistema di localizzazione. All'interno dell'immagine (4.6) il rombo di colore nero indica la posizione del tag, mentre il triangolo indica l'ancora che funge da riferimento per la sincronizzazione.

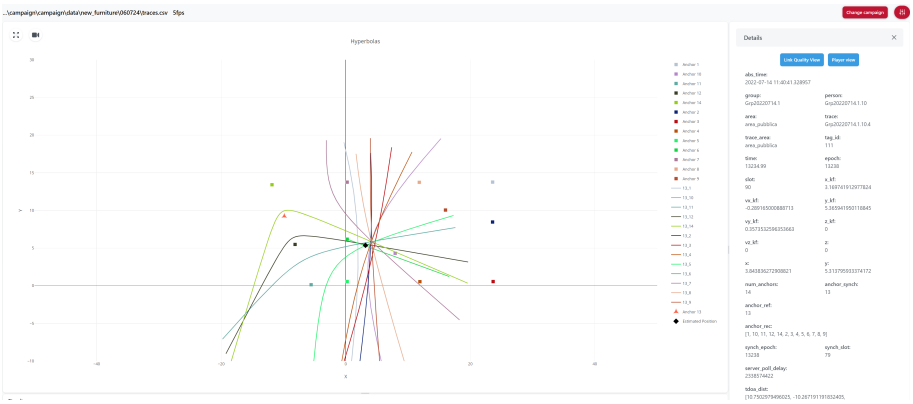

Figura 4.6: Vista della iperboli.

4.4 Architettura del sistema

L'architettura del sistema sviluppato si basa su una struttura modulare che combina diverse tecnologie back-end e front-end per garantire un'esperienza fluida ed un'elaborazione efficiente dei dati. Il software utilizza **Electron**, un framework che consente di creare applicazioni desktop multi-piattaforma utilizzando tecnologie web avanzate, come **React** e **Tailwind CSS**, per il rendering dell'interfaccia utente. Il back-end sfrutta **Node.js** per gestire i processi principali e l'interazione con i file di dati e gli **script Python** per l'elaborazione dei dati.
Come mostrato nel diagramma 4.7, l'architettura è composta da diversi processi che comunicano tra di loro attraverso un sistema Inter Process Comunication (IPC), garantendo una chiara separazione tra il Main Process e i vari Renderer Process. Questa separazione permette di gestire le operazioni computazionalmente pesanti all'interno del processo principale, mentre le interfacce utente sono gestite nei processi di rendering, garantendo così una migliore reattività all'applicazione.
Il Main Process, realizzato in Node.js, gestisce le funzionalità principali dell'applicazione, come la comunicazione con il sistema operativo e l'esecuzione degli script Pyhton per l'elaborazione dei dati TDoA. I Renderer Process, basati sul motore Chromium, consentono di interpretare il linguaggio web, permettendo cosi l'uso di tecnologie come React ed altri framework per l'interfaccia utente. Questo approccio favorisce un'esperienza utente ottimizzata [3]. La scelta di utilizzare **Electron** permette all'applicazione di sfruttare sia le potenzialità del front-end sia le funzionalità native offerte da Node.js e dal sistema operativo. Grazie alla popolarità di queste tecnologie e alla loro vasta community, è stato possibile accedere ad una vasta gamma di librerie, agevolando notevolmente il processo di sviluppo. L'adozione di tecnologie web per lo sviluppo dell'interfaccia utente non solo semplifica il processo di sviluppo e migliora il risultato finale, ma consente anche di predisporre il software per una futura evoluzione come webapp. Questo approccio, che prevede il riutilizzo del codice UI (user inteface), è adottato da molte applicazioni moderne come Visual Studio Code, Discord e Figma, le quali condividono una base tecnologica simile.

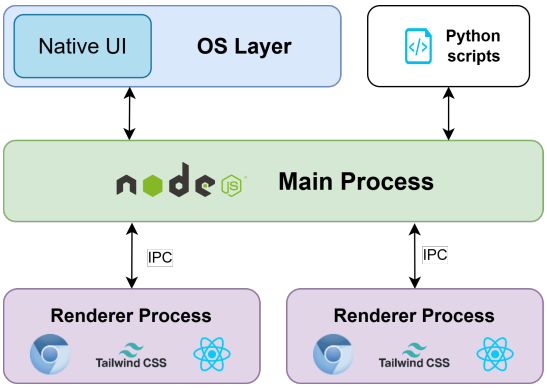

Figura 4.7: Diagramma ad alto livello del software TALLA Visualizer.

5 Implementazione

Questo capitolo descrive in dettaglio l'implementazione di TALLA Visualizer, un software sviluppato per la visualizzazione dei dati di localizzazione basati su TDoA, nell'ambito dell'architettura TALLA. L'obiettivo del progetto è stato creare un applicazione altamente efficiente e reattiva, in grado di elaborare e rappresentare visivamente i dati in un modo comprensibile e facilmente interpretabile, con l'intento di semplificare il processo di identificazione dei bug. Per raggiungere questo scopo, sono state utilizzate tecnologie moderne

e ampiamente diffuse, come Electron, React, Node.js e Python, ciascuna selezionata per il suo contributo specifico al progetto.
La scelta di Electron ha permesso di sviluppare un applicazione multipiattaforma che sfrutta le capacità del web per l'interfaccia utente, pur mantenendo accesso alle risorse native del sistema operativo. Node.js è impiegato nel back-end per la gestione delle operazioni principali, mentre React e Tailwind CSS sono stati utilizzati per la costruzione di un'interfaccia utente intuitiva e dinamica. Python, invece, è stato scelto per gestire l'elaborazione dei dati TDoA, grazie alla sua versatilità e alle sue librerie specializzate nel calcolo numerico.
Questo capitolo sarà suddiviso in sezioni che documentano l'implementazione di ciascuna componente principale, a partire dal Main Process fino al Renderer Process. Verranno inoltre descritte le tecniche di comunicazione tra i processi principali, attraverso l'Inter-Process Comunication (IPC), utilizzate per mantenere il sistema sincronizzato. Infine, saranno discusse le ottimizzazioni adottate per garantire un alto livello di prestazioni ed usabilità.

5.1 Struttura generale del codice

La struttura del codice di TALLA Visualizer è stata progettata per mantenere modularità e manutenibilità, con ciascuna directory che contiene moduli specifici e componenti organizzati in base alla loro funzione. La directory principale *src* include i file di configurazione chiave (*main.js*, *renderer.js*, *index.html*) e sottocartelle che separano i componenti dell'interfaccia utente, le pagine principali, gli script di elaborazione e gli stati condivisi dell'applicazione. Le sottodirectory principali includono:

- **components**, che contiene i vari componenti React utilizzati nell'interfaccia utente. Tra questi, *Chart.jsx* che si occupa di gestire la visualizzazione dei dati TDoA, mentre LinkQualityChart.jsx gestisce la visualizzazione della vista della qualità dei collegamenti.

- **scripts**, che raccoglie gli script utili all'elaborazione dei dati di localizzazione, come *handleHyperbolas.py* per il calcolo delle parabole.

- **store**, al suo interno si possono trovare i contesti per la gestione dello stato globale dell'applicazione, come *FileHandlerContext.jsx* per la gestione dei file selezionati, e *GraphContext.jsx* per la gestione degli stati riguardanti il grafico.

- **pages**, che contiene i componenti di livello superiore che rappresentano le principali pagine dell'applicazione, navigabili tramite il sistema di routing integrato di React. Questi componenti gestiscono la logica delle diverse viste e organizzano i sottocomponenti necessari per ogni sezione.

Questa struttura permette di mantenere separate le logiche di presentazione, elaborazione, gestione dello stato e di routing, rendendo il codice più semplice da capire e da espandere.

Listing 5.1: Struttura della directory di TALLA Visualizer.

```
TallaVisualizer/src
|   ├── context-menu.js
|   ├── costants.js
|   ├── index.css
|   ├── index.html
|   ├── main.js
|   ├── main.jsx
|   ├── preload.js
|   └── renderer.js
|
├────components
|       ├── AnalyticsPage.jsx
|       ├── CampaignForm.jsx
|       ├── CampaignGroupForm.jsx
|       ├── Card.jsx
|       ├── Chart.jsx
|       ├── EnvObjForm.jsx
|       ├── HyperbolasChart.jsx
|       ├── InputFile.jsx
|       ├── LinkQualityChart.jsx
|       ├── ModeSwitch.jsx
|       ├── PopoverColorPicker.jsx
|       ├── PositionDetails.jsx
|       ├── ProgressBar.jsx
|       ├── RecordingButton.jsx
|       ├── SelectCampaignForm.jsx
|       ├── SpeedDropDown.jsx
|       ├── Spinner.jsx
|       ├── Switch.jsx
|       ├── TagsForm.jsx
|       ├── TagShowBar.jsx
|       ├── TallaNavbar.jsx
|       ├── TreeCampaign.jsx
|       └── TreeCampaignSelect.jsx
|
├────icon
|       ├── file.ico
|       └── file.png
|
├────pages
|       ├── Dashboard.jsx
|       ├── Home.jsx
|       └── Upload.jsx
|
├────scripts
|      ├── handleHyperbolas.py
|      ├── hyperplotter.py
|      ├── mergeCsvAndSplit.py
|      └── processCsvThroughFps.py
|
└────store
        ├── FileHandlerContext.jsx
        ├── GraphContext.jsx
        ├── ModeContext.jsx
        ├── uploadedFileContext.jsx
        └── viewSettingsContext.jsx
```

5.2 Main process (Back-end)

Il Main Process in Electron ha un ruolo centrale nella gestione dell'applicazione, occupandosi di creare e monitorare le finestre, gestire il ciclo di vita dell'applicazione e coordinare i processi secondari [3]. Questa separazione è ispirata ai browser moderni: così come un browser isola le schede in processi separati per migliorare la stabilità, anche Electron usa i Renderer Process per le interfacce utente, mentre il processo principale esegue compiti più complessi e ad alto privilegio, come l'accesso a risorse di sistema o l'apertura di finestre di dialogo native [3].
Il processo primario, inoltre, supporta la comunicazione inter-processo, attraverso cui può inviare e ricevere messaggi dai Renderer Process, garantendo che le operazioni che richiedono risorse native (come l'accesso al file system) siano gestite in modo centralizzato e sicuro. Ad esempio il processo principale può rispondere agli eventi del sistema operativo (come la chiusura della finestra o la gestione del ciclo di vita dell'app) senza compromettere le prestazioni dell'interfaccia utente, che è gestita dai Renderer process isolati. In questo modo eventuali problemi non interferiscono con la gestione delle risorse più delicate come i file o le notifiche di sistema.
Nello specifico il Main Process nel progetto è rappresentato dal file \textit{main.js}, dove sono definiti i comportamenti in risposta ai messaggi provenienti dal frontend. In figura 5.1, è mostrato un esempio di comunicazione tra il processo principale e il Renderer Process, necessaria per ottenere i dati relativi alle ancore presenti in un file specifico all'interno dello spazio di lavoro. Il flusso di comunicazione è organizzato come segue:

1. Inizialmente, tramite il canale "tree:anchors:getFileAndFolders", il Main Process recupera i nomi dei file e delle cartelle contenuti nella directory designata. Questi nomi vengono poi inviati al processo che ha effettuato la richiesta.

2. Una volta che l'utente ha selezionato il file desiderato, il nome del file viene inviato al canale "graph:getAnchorsData". Il processo principale verifica l'esistenza del file e, in caso affermativo, legge i dati del file e li restituisce al richiedente.

```
90  ipcMain.handle('tree:anchors:getFilesAndFolders', () => {
91    return searchWorkspaceDirectory(workspaceDir, "**/anchors/*.json");
92  });
93  ipcMain.handle('graph:getAnchorsData', async (event, filePath) => {
94    const dirPath = path.dirname(filePath);
95    const filename= path.basename(filePath);
96    const dirs = glob.sync(`${filename}`, { cwd: dirPath, root: dirPath });
97    let data=[];
98    if(dirs.length !== 0){
99      data=JSON.parse(fs.readFileSync(filePath, 'utf8'));
100     //console.log(data);
101   }
102   return data;
103 });
```

Figura 5.1: Comunicazioni IPC per la lettura dei dati relativi alle ancore.

La figura 5.2 mostra il codice utilizzato per la configurazione e la gestione della finestra principale dell'applicazione, nonché per il controllo del ciclo di vita dell'app in Electron. Nello specifico, la funzione createWindow inizializza una nuova finestra mediante il modulo BrowserWindow, definendo le dimensioni della finestra, l'icona e una serie di preferenze di sicurezza come contextIsolation, nodeIntegration e il percorso di un file preload.js. Queste impostazioni sono importanti per mantenere l'applicazione sicura, poiché isolano l'ambiente del frontend da potenziali vulnerabilità e controllano l'accesso alle risorse del sistema. Inoltre, viene applicata una CSP (Content Security Policy) alla finestra che

permette di definire restrizioni alle risorse che possono essere caricate, riducendo rischi di attacchi come Cross-Site Scripting (XSS).

```
const createWindow = () => {
  mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    icon: app.isPackaged?path.join(process.resourcesPath,"icon", 'file.ico') : path.join(app.getAppPath(),"src","ico
n", 'file.png'),
    webPreferences: {
      preload: path.join(__dirname, 'preload.js'),
      nodeIntegration: false,
      contextIsolation: true,
      enableRemoteModule: false,
    },
    autoHideMenuBar: false,
  });
  mainWindow.setTitle('Talla');
  console.log(path.join(__dirname, 'preload.js'))
  mainWindow.loadURL(MAIN_WINDOW_WEBPACK_ENTRY);
  //create menu on right click
  createContextMenu(mainWindow);

        mainWindow.webContents.session.webRequest.onHeadersReceived((details, callback) => {
      callback({
        responseHeaders: {
          ...details.responseHeaders,
          'Content-Security-Policy': [
            "default-src 'self' 'unsafe-inline' 'unsafe-eval' data:; style-src 'self' 'unsafe-inline' https://fonts.go
ogleapis.com; font-src 'self' https://fonts.gstatic.com; img-src 'self' data: blob:; script-src 'self' 'unsafe-eva
l';",
          ],
        },
      });
        });
  // mainWindow.webContents.openDevTools();
};
app.on('activate', () => {
  if (BrowserWindow.getAllWindows().length === 0) {
    createWindow();
  }
});
app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit();
  }
});
```

Figura 5.2: Codice rappresentativo della gestione delle finestre e del ciclo di vita dell'app.

5.3 Renderer Process (Front-end)

Il Renderer Process in Electron è responsabile della gestione dell'interfaccia utente dell'applicazione, utilizzando tecnologie web come HTML, CSS e JavaScript. In TALLA Visualizer viene utilizzato React, una libreria JavaScript per la costruzione di componenti riutilizzabili, al fine di fornire un interfaccia reattiva e efficiente. Grazie a React, il Renderer Process può gestire efficacemente le interazioni dell'utente e gli stati interni dell'applicazione, facilitando anche la comunicazione delle informazioni necessarie al Main Process. Per quanto riguarda lo styling dei componenti, è stato impiegato Tailwind CSS, un framework CSS utility-first che offre classi di stili predefinite, agevolando l'ottenimento di un design coerente e accelerando il processo di prototipazione. Inoltre, l'integrazione di React Router consente una navigazione intuitiva tra le diverse pagine dell'applicazione.

5.4 Interazione tra front-end e back-end

Le interazioni tra il back-end e il front-end avvengono esclusivamente tramite Inter-Process Communication (IPC), ovvero attraverso la creazione di canali dedicati in cui il Main Process e il Renderer Process si scambiano messaggi. Questi canali sono arbitrari e bidirezionali, ed esistono varie metodologie per implementare tale tecnica. Alla base di questo concetto vi è un'altra proprietà che Electron mette a disposizione dei programmatori per aumentare la sicurezza e l'affidabilità del software: l'isolamento dei contesti (context iso-

lation). Questa proprietà assicura che il processo padre e il processo di rendering non possano esporre dati potenzialmente sensibili tramite il file *preload.js*. Pertanto, per consentire la comunicazione tra le due parti è necessario un intermediario, che in questo caso è l'oggetto *ContextBridge*, il quale espone determinate funzioni in modo sicuro (tipicamente le funzioni per creare e comunicare attraverso i canali IPC) [3]. Un esempio di comunicazione è illustrato nella figura 5.1 dove è mostrata solo la parte relativa al Main Process. Tuttavia,la chiamata dal Renderer process non differisce significativamente, in quanto il nome del canale rimane lo stesso, ma viene utilizzato il metodo *invoke*.

5.5 Visualizzazione dei dati

Secondo i requisiti, TALLA Visualizer doveva offrire un'esperienza di visualizzazione dei dati fluida ed interattiva, capace di fornire funzionalità utili al debugging del sistema TALLA. La principale sfida nel soddisfare tali requisiti è stato trovare uno strumento che li soddisfacesse in larga misura, senza dover sviluppare una soluzione personalizzata. In questo contesto è stato impiegato *Plotly.js*, "un libreria di grafici dichiarativi ad alto livello" [8], che consente l'implementazione predefinita di grafici interattivi; pertanto, si è rivelato lo strumento ideale per la visualizzazione dei i dati TDoA. Una delle sfide più significative è stata riuscire a coordinare il grafico con il player (Figura 4.2), poiché non è stato possibile utilizzare la soluzione nativa della libreria, in quanto ciò avrebbe comportato il trasferimento al Renderer Process di un'intera traccia di dati in una sola volta, causando seri problemi di usabilità e prestazioni dell'applicazione. Per ovviare a questo problema, è stato implementato il *lazy loading*,una tecnica che consente di ottimizzare le prestazioni delle applicazioni web posticipando il caricamento delle risorse non essenziali finché non diventano effettivamente necessarie. Invece di caricare l'intera pagina in un'unica soluzione, il lazy loading si focalizza sul caricamento del contenuto immediatamente visibile all'utente, mentre il resto viene caricato solo quando richiesto [1].

5.6 Gestione dei dati

Per rendere i dati idonei alla visualizzazione, è necessario sottoporli a un processo di elaborazione che li renda più gestibili e adeguati a una rappresentazione grafica efficiente. Affinché una traccia possa essere visualizzata, è indispensabile che essa sia presente all'interno della cartella *data* di ogni gruppo di campagne dati. La struttura delle cartelle all'interno di ciascun gruppo di campagne dati è organizzata come segue: *anchor*, *aspect*, *elements* e *data*. Una volta verificata la presenza del file CSV contenente la traccia dei dati, si procede con una prima fase di elaborazione. Il processo di elaborazione dei dati è realizzato attraverso l'utilizzo di due script Python che, grazie a librerie come Pandas, consentono di elaborare grandi moli di dati in tempi ragionevoli. L'impiego di queste librerie permette di gestire efficacemente i dataset, eseguendo operazioni di manipolazione e trasformazione necessarie per preparare i dati alla visualizzazione.

5.6.1 Elaborazione Iniziale: Segmentazione dei Dati per Tag

In questa fase iniziale, si determina quale colonna utilizzare come riferimento temporale; le possibili opzioni sono *time* e *abs_time*. Stabilito il riferimento temporale, l'intervallo di tempo viene suddiviso in frame sulla base delle impostazioni fornite dall'utente. A ogni record del file viene assegnato il corrispondente numero di frame.
In base alle impostazioni, può accadere che più dati appartenenti allo stesso tag siano associati allo stesso numero di frame, poiché sono stati campionati con una differenza temporale molto ridotta. In tali casi, a questi dati viene assegnata un'ulteriore etichetta che indica il ruolo del dato all'interno del frame. L'etichetta può assumere i valori "main" o "footprint", che graficamente corrispondono rispettivamente a un colore del tag con opacità al 100% e al 50%. L'assegnazione dell'etichetta si basa sul tempo di campionamento: al dato più recente campionato all'interno del frame viene attribuito il valore "main", mentre agli altri viene assegnato il valore "footprint". Questo processo viene applicato

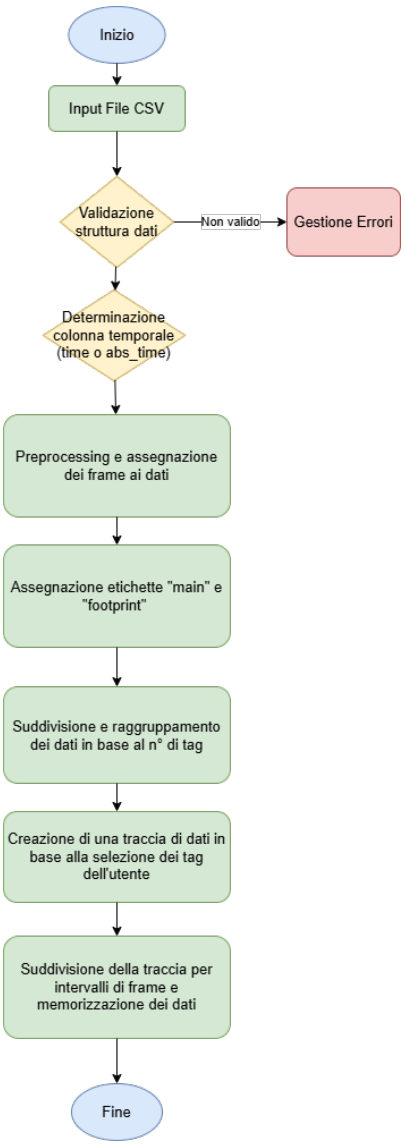

Figura 5.3: Diagramma di flusso del processo di elaborazione dei dati.

individualmente a ciascun tag.
Successivamente i dati elaborati vengono salvati nella cartella *data/processed_data*; al suo interno vengono creati tanti file quanti sono i tag presenti nella traccia, ognuno contenente i dati relativi a un singolo tag.

5.6.2 Elaborazione Secondaria: Unione dei Dati Selezionati e Lazy Loading

Dopo questa prima elaborazione, tramite l'interfaccia grafica, viene richiesto all'utente di selezionare i dispositivi di interesse. Una volta ottenuta la selezione, si avvia una seconda fase di elaborazione: i file corrispondenti ai tag scelti vengono uniti in un'unica traccia che comprende tutti i dati selezionati. Questa traccia viene nuovamente suddivisa in base agli intervalli di frame, generando nuovi file. Questo procedimento è attuato in funzione della tecnica di *lazy loading* descritta nella sezione *Visualizzazione dei dati* 5.5, la quale ottimizza le prestazioni dell'applicazione caricando le risorse non essenziali solo quando diventano necessarie.

In conclusione, questo processo di elaborazione realizzato tramite script Python e l'utilizzo di librerie come Pandas, migliora significativamente l'efficienza nella gestione e visualizzazione dei dati. Organizzando i dati per tag e permettendo la selezione personalizzata dei dispositivi di interesse, l'applicazione evita il caricamento di dati superflui e riduce l'u-

tilizzo delle risorse di sistema. Inoltre, grazie all'implementazione del lazy loading, i dati vengono caricati progressivamente durante la visualizzazione, migliorando la reattività e offrendo un'esperienza utente fluida e responsiva. Questo approccio assicura la gestione efficace di grandi quantità di dati, mantenendo elevate prestazioni dell'applicazione.

5.7 Testing e Debugging

Nel corso dello sviluppo di TALLA Visualizer, è stata posta particolare enfasi sulle attività di testing e debugging per garantire la qualità, l'affidabilità e la compatibilità dell'applicazione. Non essendo stati implementati test automatizzati, il processo di verifica si è concentrato su test manuali, test eseguiti direttamente dagli utenti finali e test ambientali. Queste attività hanno permesso di identificare e risolvere tempestivamente eventuali anomalie, assicurando che l'applicazione soddisfacesse pienamente i requisiti funzionali.

5.7.1 Testing Manuale

Il testing manuale ha rappresentato una componente fondamentale del processo di sviluppo. Ogni funzionalità implementata è stata sottoposta a rigorose verifiche per accertarne il corretto funzionamento e l'aderenza alle specifiche progettuali. Questo approccio ha coinvolto una serie di attività, tra cui:

- **Verifica delle funzionalità dell'interfaccia utente**: sono stati testati tutti gli elementi interattivi, assicurandosi che rispondessero correttamente agli input dell'utente e che le interazioni fossero intuitive e coerenti.
- **Controllo della gestione dei dati**: si è verificato che i dati di input fossero correttamente elaborati e che i risultati visualizzati fossero accurati e aggiornati.
- **Validazione dei flussi operativi**: sono stati simulati diversi scenari d'uso per garantire che l'applicazione gestisse correttamente sia le operazioni standard sia eventuali situazioni eccezionali.
- **Monitoraggio delle prestazioni**: si è prestata attenzione ai tempi di risposta dell'applicazione, specialmente durante l'elaborazione di grandi quantità di dati, per assicurare un'esperienza utente fluida.

Queste attività hanno permesso di individuare e correggere bug, migliorare l'usabilità e ottimizzare le prestazioni dell'applicazione.

5.7.2 Testing Eseguito dagli Utenti Finali

Una fase cruciale del processo di testing ha coinvolto gli utenti finali, che hanno utilizzato TALLA Visualizer in contesti operativi reali. Il loro feedback è stato essenziale per:

- **Identificare problemi non emersi durante il testing manuale**: gli utenti possono incontrare situazioni non previste dagli sviluppatori, portando alla luce bug o comportamenti anomali.
- **Valutare l'usabilità dell'applicazione**: le osservazioni degli utenti hanno contribuito a migliorare l'interfaccia utente e a rendere le funzionalità più intuitive.
- **Confermare l'efficacia delle funzionalità implementate**: l'utilizzo sul campo ha permesso di verificare che l'applicazione rispondesse adeguatamente alle esigenze per cui è stata progettata.

Il coinvolgimento degli utenti finali ha dunque fornito un prezioso contributo al processo di miglioramento continuo dell'applicazione.

5.7.3 Testing Ambientale

Dato che TALLA Visualizer è un'applicazione multipiattaforma sviluppata con Electron, è stato fondamentale assicurare il corretto funzionamento su diversi sistemi operativi. Il testing ambientale ha compreso la verifica su vari sistemi operativi, il controllo su hardware differente e l'analisi dell'interazione con componenti esterni. La verifica è stata condotta su Windows e Linux per garantire la compatibilità dell'applicazione e individuare eventuali differenze di comportamento; tuttavia, non è stato possibile testare l'applicazione su macOS a causa della mancanza di un dispositivo con tale sistema operativo. Inoltre, sono state eseguite prove su macchine con diverse configurazioni hardware per assicurare prestazioni adeguate in vari contesti. Si è verificato anche il corretto funzionamento dell'applicazione in presenza di periferiche o software che potessero influenzarne l'operatività. Queste attività hanno permesso di identificare e risolvere problemi specifici legati all'ambiente di esecuzione, migliorando la robustezza e l'affidabilità dell'applicazione.

5.7.4 Debugging durante lo Sviluppo

Il processo di debugging è stato integrato nell'intero ciclo di sviluppo, permettendo di affrontare e risolvere tempestivamente le problematiche emerse. Gli strumenti principali utilizzati sono stati:

Chrome DevTools Essendo il Renderer Process basato su tecnologie web, Chrome DevTools si è rivelato uno strumento indispensabile. Ha consentito di:

- **Ispezionare il Document Object Model (DOM)**: per verificare la struttura degli elementi e modificare dinamicamente il layout durante il debugging.
- **Analizzare le prestazioni**: attraverso il monitoraggio del consumo di risorse e l'identificazione di colli di bottiglia nel rendering o nell'esecuzione di script.
- **Debuggare il codice JavaScript**: utilizzando breakpoints, watch expressions e il call stack per tracciare l'esecuzione del codice e individuare errori logici.

React Developer Tools L'estensione React Developer Tools ha fornito funzionalità specifiche per il debugging dei componenti React, permettendo di:

- **Esplorare la gerarchia dei componenti**: visualizzando la struttura dell'applicazione e le relazioni tra i vari componenti.
- **Verificare props e state**: controllando i valori passati ai componenti e lo stato interno, facilitando l'individuazione di incongruenze.
- **Analizzare le performance di rendering**: identificando componenti che si renderizzavano inutilmente e ottimizzando il ciclo di aggiornamento dell'interfaccia.

Debugging del Main Process Per quanto riguarda il Main Process, il debugging ha coinvolto l'utilizzo della console di Node.js e l'implementazione di un sistema di logging dettagliato. Questo approccio ha permesso di tracciare il flusso di esecuzione, monitorando le chiamate alle funzioni e gli eventi gestiti dal processo principale. Inoltre, ha facilitato la diagnosi di problemi di comunicazione IPC, verificando l'invio e la ricezione dei messaggi tra il Main Process e il Renderer Process e assicurandosi che i canali fossero configurati correttamente. Infine, ha reso possibile gestire efficacemente eccezioni ed errori runtime, identificando situazioni anomale e implementando adeguate procedure di gestione degli

errori. Questo insieme di strategie ha contribuito a migliorare la robustezza e l'affidabilità dell'applicazione.

6 Possibili estensioni e nuove funzioni

Nel corso dello sviluppo di TALLA Visualizer sono emerse diverse idee per potenziali estensioni e nuove funzionalità che potrebbero arricchire ulteriormente l'applicazione, migliorandone l'utilità e l'efficacia. In questa sezione vengono presentate alcune delle possibili implementazioni future, con l'obiettivo di offrire spunti per sviluppi successivi e di delineare le direzioni in cui il progetto potrebbe evolvere.

6.1 Visualizzazione in Tempo Reale dei Dati

Un'estensione significativa dell'applicazione potrebbe consistere nell'abilitare la visualizzazione in tempo reale dei dati. Attualmente, TALLA Visualizer è progettato per elaborare e visualizzare dati pre-registrati; tuttavia, l'integrazione di funzionalità real-time richiederebbe modifiche sia all'infrastruttura di raccolta dei dati sia all'architettura del software. Questo permetterebbe agli utenti di monitorare in diretta la posizione dei tag, facilitando il rilevamento immediato di anomalie o problemi nella rete di localizzazione. L'implementazione di questa funzionalità comporterebbe la gestione efficiente dei flussi di dati in ingresso e l'ottimizzazione delle prestazioni per garantire una risposta tempestiva dell'interfaccia utente.

6.2 Sviluppo di un Algoritmo Integrato di Rilevamento degli Errori

Attualmente, l'identificazione di errori nei dati di localizzazione è affidata all'utente, che può avvalersi degli strumenti di visualizzazione forniti dall'applicazione. Un possibile miglioramento sarebbe l'integrazione di un algoritmo automatico di rilevamento degli errori, in grado di analizzare i dati e segnalare all'utente eventuali anomalie o incongruenze. Questo algoritmo potrebbe basarsi su metriche quali i residui e i costi del solver TDoA, identificando automaticamente i casi in cui gli errori superano determinate soglie. In questo modo si ridurrebbe il carico cognitivo sull'utente e si migliorerebbe l'efficienza nel processo di diagnosi dei problemi.

6.3 Visualizzazione degli Intervalli di Arresto dei Tag

La capacità di identificare e visualizzare gli intervalli di tempo in cui un tag rimane stazionario, ottenuta analizzando i dati di posizione raccolti da TALLA, offre insights significativi, sia in tempo reale sia nell'analisi di dati storici. Ad esempio, in un contesto museale, comprendere quando un visitatore si ferma davanti a un'opera d'arte potrebbe fornire indicazioni sul suo livello di interesse [5]. Diversi algoritmi, come SPD, DBSCAN, SeqScan e KBV, possono essere utilizzati per analizzare i dati di TALLA e identificare accuratamente le fermate. La visualizzazione di questi intervalli di arresto, ad esempio tramite grafici temporali o mappe, può offrire una comprensione più approfondita del comportamento dei visitatori e guidare decisioni strategiche.

6.4 Implementazione di Visualizzazioni Aggiuntive

Per arricchire l'esperienza utente e fornire informazioni più dettagliate, si potrebbero sviluppare ulteriori modalità di visualizzazione:

6.4.1 Visualizzazione Riassuntiva della Qualità delle Ancore

Una rappresentazione sintetica della qualità di una ancora in un dato momento potrebbe essere molto utile. Integrando indicatori come il residuo e altri parametri che segnalano condizioni di Non Line of Sight (NLOS), l'applicazione potrebbe offrire una panoramica immediata delle prestazioni delle ancore. Questo permetterebbe di identificare rapidamente eventuali problemi nella rete di ancore e di intervenire per migliorarne l'affidabilità.

6.4.2 Visualizzazione dello Stato dei Filtri di Kalman

L'inclusione di un toggle che permetta di visualizzare lo stato dei filtri di Kalman, come ad esempio il vettore velocità dei tag, potrebbe fornire informazioni aggiuntive agli utenti avanzati. Questa funzionalità consentirebbe di analizzare più in dettaglio il comportamento dinamico dei tag, offrendo strumenti utili per applicazioni che richiedono un monitoraggio accurato dei movimenti.

6.4.3 Visualizzazione focalizzata sul tag

Implementare una modalità "follow-tag" in cui, selezionato un tag, l'applicazione mantiene automaticamente il tag al centro della visualizzazione, spostando il resto dell'ambiente di conseguenza. Questa caratteristica sarebbe particolarmente utile durante le demo o in situazioni in cui è importante monitorare un tag specifico senza dover regolare manualmente la vista sugli assi.

Considerazioni Finali Le estensioni e le nuove funzionalità descritte rappresentano potenziali direzioni per lo sviluppo futuro di TALLA Visualizer. Implementare queste caratteristiche richiederebbe un'analisi approfondita delle esigenze degli utenti e delle implicazioni tecniche. Tuttavia, l'integrazione di tali miglioramenti potrebbe aumentare significativamente il valore dell'applicazione, rendendola uno strumento ancora più potente e versatile per la visualizzazione e l'analisi dei dati di localizzazione.

7 Conclusioni

Il presente lavoro ha avuto come obiettivo lo sviluppo di TALLA Visualizer, un'applicazione desktop progettata per facilitare la visualizzazione e l'analisi dei dati TDoA (Time Difference of Arrival) nel contesto della localizzazione indoor. La necessità di uno strumento dedicato nasce dalla complessità intrinseca dei sistemi di localizzazione e dalla mancanza di soluzioni esistenti che offrano un'interfaccia utente intuitiva e funzionalità specifiche per l'identificazione e la risoluzione di anomalie nei dati.
Nel corso della tesi, sono state affrontate diverse sfide legate alla gestione di grandi quantità di dati, all'ottimizzazione delle prestazioni e alla progettazione di un'interfaccia utente che fosse al contempo intuitiva e completa. L'applicazione è stata sviluppata utilizzando tecnologie moderne come Electron e React per la realizzazione dell'interfaccia grafica e Python per l'elaborazione dei dati, sfruttando librerie come Pandas per migliorare l'efficienza del processo di manipolazione dei dataset.
Uno dei contributi principali di questo lavoro consiste nell'implementazione di un processo di elaborazione dei dati che permette di gestire efficacemente grandi volumi di informazioni, mantenendo elevate prestazioni dell'applicazione. La segmentazione dei dati per tag e l'utilizzo del *lazy loading* hanno consentito di ottimizzare l'uso delle risorse di sistema e di offrire un'esperienza utente fluida e reattiva. L'applicazione offre diverse viste per

la visualizzazione dei dati, tra cui la rappresentazione delle traiettorie dei tag, la qualità dei collegamenti tra tag e ancore e la possibilità di visualizzare informazioni dettagliate relative ai singoli dispositivi. Queste funzionalità permettono agli utenti di analizzare in profondità i dati raccolti, facilitando l'identificazione di anomalie e la comprensione del comportamento del sistema di localizzazione.
Durante lo sviluppo, sono state adottate pratiche di testing e debugging che hanno coinvolto test manuali, test eseguiti dagli utenti finali e test ambientali su diversi sistemi operativi. Queste attività hanno garantito la robustezza e l'affidabilità dell'applicazione, assicurando il corretto funzionamento in vari contesti d'uso.
Le possibili estensioni delineate nel capitolo precedente aprono la strada a futuri sviluppi che potrebbero ampliare le capacità dell'applicazione, come l'integrazione della visualizzazione in tempo reale dei dati, l'implementazione di algoritmi di rilevamento automatico degli errori e l'aggiunta di nuove modalità di visualizzazione.
In conclusione, TALLA Visualizer si propone come un valido strumento per la visualizzazione e l'analisi dei dati TDoA, offrendo una soluzione che combina facilità d'uso, efficienza e funzionalità avanzate. Il lavoro svolto fornisce una solida base per ulteriori miglioramenti e adattamenti, offrendo un supporto concreto agli operatori che necessitano di strumenti efficaci per il monitoraggio e l'ottimizzazione dei sistemi di localizzazione indoor.

Bibliografia

[1] Răzvan-Mihail BÂRA, Costin Anton BOIANGIU, and Cătălin TUDOSE. Analysing the performance impacts of lazy loading in web applications. *Journal of Information Systems & Operations Management*, 18.1:1–15, 2024.

[2] S.R. Drake and K. Dogancay. Geolocation by time difference of arrival using hyperbolic asymptotes. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages ii–361, 2004.

[3] Electron. Electron doc. `https://www.electronjs.org/docs`. ultimo accesso: 25/10/2024.

[4] Fastweb. Le applicazioni dell'ultra wide band. `https://www.fastweb.it/fastweb-plus/digital-magazine/le-applicazioni-dell-ultra-wide-band`. ultimo accesso: 06/08/2024.

[5] Fatima Hachem, Davide Vecchia, Maria Luisa Damiani, and Gian Pietro Picco. Fine-grained stop-move detection in uwb-based trajectories. In *2022 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 111–118, 2022.

[6] Zan Li, Desislava C. Dimitrova, Torsten Braun, and Denis Rosário. Highly accurate evaluation of gps synchronization for tdoa localization. In *2013 IFIP Wireless Days (WD)*, pages 1–3, 2013.

[7] Navigine. Navigine tracking platform. `https://navigine.com/platform/tracking/`. ultimo accesso: 09/08/2024.

[8] Plotly. Plotly.js documentation. `https://plotly.com/javascript/`. ultimo accesso: 28/10/2024.

[9] Sewio. Sewio documentation: Fist path. `https://docs.sewio.net/docs/first-path-25593240.html`. ultimo accesso: 22/10/2024.

[10] Davide Vecchia, Pablo Corbalán, Timofei Istomin, and Gian Pietro Picco. Talla: Large-scale tdoa localization with ultra-wideband radios. In *2019 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–8, 2019.

[11] Jun yong Yoon, Jae-Wan Kim, Woo-Yong Lee, and Doo-Seop Eom. A tdoa-based localization using precise time-synchronization. In *2012 14th International Conference on Advanced Communication Technology (ICACT)*, pages 1266–1271, 2012.

[12] Faheem Zafari, Athanasios Gkelias, and Kin K. Leung. A survey of indoor localization systems and technologies. *IEEE Communications Surveys & Tutorials*, 21(3):2568–2599, 2019.