POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Progettazione e realizzazione di un sistema integrato per la navigazione indoor per persone con disabilità visive

Relatori
Prof. Federico Manuri
Prof. Andrea Sanna

Candidato
Salvatore Giugliano

marzo 2026

«C'è sempre una via d'uscita
per chi è abbastanza ingegnoso da trovarla.»

Rick Riordan

A chi, pur non potendo vedere la strada, ha il coraggio di cercarla.
Ai miei genitori e alla mia famiglia, per essere stati la mia guida costante.
E alla tecnologia, lo strumento più bello per trasformare
un limite invalicabile in una nuova via d'uscita.

Sommario

In questa tesi viene sviluppato un sistema integrato di ausilio alla navigazione indoor per persone con disabilità visive, basato sulla combinazione di tecniche di visione artificiale, stima della profondità e localizzazione mediante SLAM. L'obiettivo principale è offrire un supporto in tempo reale che permetta all'utente di muoversi in ambienti sconosciuti evitando ostacoli, mantenendo una corretta percezione dello spazio circostante e seguendo percorsi sicuri verso una destinazione selezionata. L'intero sistema è progettato per essere non invasivo, adattabile a diversi contesti e utilizzabile senza modifiche dell'infrastruttura dell'edificio. La piattaforma sensoriale principale è costituita dagli occhiali di Project Aria, dotati di telecamere RGB e monocromatiche e di un'unità inerziale ad alta frequenza. I dati raccolti vengono utilizzati per la localizzazione tramite gli algoritmi ORB-SLAM3 (sfruttando una mappa registrata in precedenza), mentre per il riconoscimento degli oggetti e la stima della loro distanza vengono usati i modelli YOLO e Zoe-Depth. Questa combinazione consente di identificare ostacoli fissi e dinamici, stimarne la distanza e integrare tali informazioni all'interno della rappresentazione dell'ambiente. L'elaborazione è affidata a un server che gestisce in parallelo i flussi sensoriali, sincronizzando inferenze AI e dati di posa e inviandoli in tempo reale a un'applicazione mobile sviluppata in Unity. L'app integra moduli di navigazione basati su NavMesh, funzioni di Text-to-Speech e Speech-to-Text e algoritmi di aggiornamento dinamico del percorso, generando istruzioni vocali contestuali e inserendo nella scena gli ostacoli rilevati per garantire una guida fluida e sicura. Il risultato è una soluzione modulare, estensibile e potenzialmente applicabile in scenari reali, capace di migliorare l'autonomia e la sicurezza degli utenti con deficit visivi. La tesi analizza le scelte progettuali, le tecnologie impiegate, l'architettura software e le principali limitazioni identificate, proponendo possibili sviluppi futuri orientati all'elaborazione totalmente on-device e all'ottimizzazione dei modelli AI per applicazioni real-time. L'applicazione sviluppata è stata infine testata in contesti controllati all'interno del padiglione delle aule R del Politecnico di Torino, coinvolgendo studenti in scenari di navigazione indoor realistici.

Indice

Glossario

AADT	*Auto-Adaptive Double Thresholding*
ADF	*Area Description File*
AP	*Average Precision*
APBB	*Average Precision on Bounding Box*
APBEV	*Average Precision Bird-Eye View*
API	*Application Programming Interface*
APr	*Average Precision on rare*
APc	*Average Precision on common*
APf	*Average Precision on frequent*
AR	*Augmented Reality*
ASE	*Aria Synthetic Environments*
BB	*Bounding Box*
BEV	*Bird's Eye View*
BGR	*Blue-Green-Red*
BLE	*Bluetooth Low Energy*
CAD	*Computer-Aided Design*
CLIP	*Contrastive Language-Image Pretraining*
CNN	*Convolutional Neural Network*
CSP	*CrossStage Partial*
DFOV	*Diagonal Field Of View*
DoF	*Degrees of Freedom*
ETA	*Electronic Travel Assistant*
FPCIM	*Fusion of Point Cloud and Image*
FPS	*Frame per Second*
FV	*Front View*
GELAN	*Generalized Efficient Layer Aggregation Network*
GLIP	*Grounded Language-Image Pretraining*

GNSS	*Global Navigation Satellite System*
GPS	*Global Positioning System*
GPU	*Graphics Processing Unit*
HFOV	*Horizontal Field Of View*
IA	Intelligenza Artificiale
IMU	*Inertial Measurement Unit*
IoU	*Intersection over Union*
IPC	*Inter-Process Communication*
JSON	*JavaScript Object Notation*
LAN	*Local Area Network*
LiDAR	*Light Detection and Ranging*
O&MO	*Orientation & Mobility Operator*
ORB	*Oriented FAST and Rotated BRIEF*
mAP	*mean Average Precision*
MAP	*Maximum-a-Posteriori*
MCIM	*Monocular Image*
MPS	*Machine Perception Services*
MVIM	*Multi-view Image*
NaN	*Not a Number*
OBB	*Oriented Bounding Box*
PCM	*Point Cloud*
PGI	*Programmable Gradient Information*
PoI	*Point of Interest*
PoI-graph	*Point of Interest Graph*
POV	*Point Of View*
px	*pixel*
RepVL-PAN	*Re-parameterizable Vision-Language Path Aggregation Network*
RFID	*Radio Frequency Identification*
RGB	*Red-Green-Blue*
RGB-D	*Red-Green-Blue-Depth*
RSSI	*Received Signal Strength Indicator*
SLAM	*Simultaneous Localization and Mapping*
STT	*Speech-to-Text*
SUS	*System Usability Scale*

SVD	*Singular Value Decomposition*
TSM-KF	*Time-Stamped Map Kalman Filter*
TTS	*Text-to-Speech*
VAL	*Vibration Array Language*
VIO	*Visual-Inertial Odometry*
VPS	*Visual Positioning Service*
VR	*Virtual Reality*
VRS	*Virtual Reality Sensor*
Wi-Fi	*Wireless Fidelity*
YAML	*YAML Ain't Markup Language*
YOLO	*You Only Look Once*

Capitolo 1

Introduzione

La mobilità autonoma è un diritto fondamentale e un elemento cruciale per l'indipendenza e la qualità della vita di ogni individuo. Per le persone con disabilità visive, muoversi in modo sicuro ed efficiente, specialmente in ambienti interni complessi e sconosciuti come università, uffici o ospedali, rappresenta una sfida quotidiana di notevole entità. Le dinamiche della mobilità non visiva richiedono un elevato carico cognitivo: l'utente deve costantemente mappare mentalmente lo spazio circostante, stimare le distanze e riconoscere punti di riferimento, prestando contemporaneamente attenzione a ostacoli statici e dinamici (come pedoni, porte aperte o barriere architettoniche impreviste). In assenza di informazioni visive dirette, la percezione anticipatoria si riduce drasticamente, diminuendo i tempi di reazione e aumentando il rischio di collisioni, con ricadute significative sia sull'incolumità fisica sia sulla sicurezza psicologica.

Nel corso degli anni, il mercato e la ricerca hanno proposto diversi approcci per mitigare queste difficoltà, ma con risultati alterni. Gli ausili tradizionali, come il bastone bianco o il cane guida, pur essendo essenziali, offrono una percezione limitata all'area immediatamente vicina all'utente o richiedono lunghi percorsi di addestramento. In ambito commerciale, si è assistito alla diffusione di *Electronic Travel Assistant* (ETA) e di applicazioni per smartphone (come Seeing AI [32] o Be My Eyes [4]) che sfruttano la fotocamera del dispositivo per descrivere l'ambiente. Tuttavia, queste soluzioni presentano limiti pratici evidenti: l'uso dello smartphone impegna le mani, risultando scomodo e potenzialmente pericoloso durante la deambulazione, e l'analisi dell'immagine è spesso bidimensionale, priva di una reale comprensione spaziale del contesto. Altri dispositivi, come i bastoni intelligenti dotati di sensori a ultrasuoni e *Global Positioning System* (GPS), si scontrano con l'impossibilità di rilevare ostacoli ad altezza testa e con la totale inefficacia del segnale satellitare all'interno degli edifici.

Per superare queste barriere, la frontiera tecnologica si sta orientando verso un paradigma profondamente diverso: la cosiddetta *egocentric vision* (visione egocentrica). Questo approccio si basa sull'acquisizione e l'analisi dei dati visivi dal punto di vista in prima persona dell'utente (*Front View* (FV)), tipicamente attraverso dispositivi indossabili come gli *smart glasses*. L'*egocentric vision* rivoluziona le dinamiche di assistenza: il sistema "vede" esattamente ciò che l'utente ha di fronte, allineando naturalmente il campo visivo dei sensori con i movimenti della testa e la traiettoria di cammino. Questo

permette una comprensione contestuale molto più precisa e anticipatoria, liberando le mani dell'utente e riducendo il carico cognitivo.

Il presente lavoro di tesi si inserisce esattamente all'interno di questa prospettiva e propone la progettazione e realizzazione di un sistema integrato di supporto alla navigazione *indoor* per persone con disabilità visive, basato sull'elaborazione in tempo reale dei dati acquisiti dagli *smart glasses* di Project Aria [39], sviluppati da Meta Reality Labs. Gli occhiali, dotati di telecamere *Red-Green-Blue* (RGB) e monocromatiche e di un'unità inerziale (*Inertial Measurement Unit* (IMU)) ad alta frequenza, consentono di raccogliere flussi sensoriali eterogenei (perfettamente in linea con i principi dell'*egocentric vision*) che vengono elaborati da un sistema server dedicato.

L'architettura proposta elabora i flussi sensoriali articolandosi in tre macro-componenti fondamentali per una navigazione sicura ed efficiente:

- localizzazione e mappatura tramite l'algoritmo ORB-SLAM3 [8], utilizzando una mappa precedentemente registrata dell'edificio;
- riconoscimento di ostacoli tramite il modello *You Only Look Once* (YOLO) [50], utile per individuare ostacoli fissi e dinamici;
- stima della profondità mediante il modello ZoeDepth [5], necessario per determinare la distanza degli ostacoli rilevati.

Le informazioni elaborate vengono quindi trasmesse a un'applicazione mobile sviluppata in Unity, che integra un sistema di navigazione basato su NavMesh, funzionalità di *Text-to-Speech* (TTS) e *Speech-to-Text* (STT), e tecniche di aggiornamento dinamico del percorso. Il risultato è un sistema in grado di fornire istruzioni vocali contestuali, evitare ostacoli rilevati in tempo reale e guidare l'utente lungo un tragitto sicuro verso una destinazione selezionata.

L'obiettivo principale di questa tesi è dimostrare la fattibilità di un sistema modulare, non invasivo e che non richieda modifiche fisiche all'edificio, capace di combinare visione artificiale, intelligenza artificiale e tecniche di navigazione in un'unica piattaforma a supporto delle persone con disabilità visive. Il lavoro include la progettazione dell'architettura software, l'integrazione e il coordinamento dei vari moduli, nonché la realizzazione di una fase sperimentale condotta all'interno del Politecnico di Torino, volta a valutare prestazioni, limiti e usabilità della soluzione.

In particolare, la fase di validazione del sistema é stata strutturata su due livelli: test progettuali e test di usabilitá. I test progettuali, condotti durante la fase di sviluppo, hanno confermato la soliditá dell'approccio proposto: il sistema ha dimostrato un'elevata precisione spaziale (scostamento medio di 20 cm) e un'accuratezza ampiamente compatibile con le esigenze di sicurezza della navigazione *indoor*. Sotto il profilo computazionale, l'elaborazione locale ha garantito prestazioni stabili in tempo reale, evidenziando l'efficienza dei modelli di Intelligenza Artificiale (IA) (circa 85 ms per frame) e dell'algoritmo ORB-SLAM3 (meno di 1 ms per frame). Al contempo, l'analisi tecnica ha permesso di mappare criticamente i limiti fisici dell'infrastruttura, identificando nelle latenze di rete e nelle interferenze *Wireless Fidelity* (Wi-Fi) il principale collo di bottiglia per la trasmissione dei dati, pur dimostrando una buona resilienza dell'architettura asincrona

nel garantire la guida verso l'obiettivo. Per quanto riguarda i test di usabilitá, la tesi definisce un rigoroso protocollo sperimentale, ispirato dalla letteraura, e le metriche di valutazione specifiche per simulare casi d'uso realistici con utenti bendati, gettando cosí le basi formali per le future prove di navigazione e la validazione sul campo dell'esperienza utente.

La tesi è articolata come segue:

- Il capitolo 2 presenta un'analisi dello stato dell'arte, esaminando le principali tecnologie per la navigazione indoor e i sistemi più rilevanti sviluppati negli ultimi anni. Nonché una panoramica si principali tipi di modelli di reti neurali usati per l'identificazione di oggetti nell'ambiente 3D.
- Il capitolo 3 descrive in dettaglio gli strumenti hardware e software utilizzati, con particolare attenzione ai modelli di IA e alla piattaforma Project Aria.
- Il capitolo 4 illustra l'architettura del sistema progettato, descrivendone i componenti, i flussi di comunicazione e le soluzioni adottate.
- Il capitolo 5 riporta i test effettuati e un'analisi quantitativa e qualitativa dei risultati ottenuti.
- Il capitolo 6 presenta le conclusioni e discute i possibili sviluppi futuri, suggerendo come il sistema possa evolvere verso una piattaforma totalmente *on-device* e più robusta in scenari reali.

Capitolo 2

Stato dell'arte

A seguito dell'introduzione degli obiettivi della ricerca, il presente capitolo propone una disamina dei sistemi allo stato dell'arte per la navigazione *indoor* dedicati agli utenti con disabilità visive. La trattazione prosegue con un approfondimento sulle architetture neurali di maggiore utilizzo per il rilevamento e l'identificazione degli ostacoli in contesti reali.

2.1 Sistemi di navigazione 3D indoor

La letteratura scientifica comprende diversi lavori sui sistemi di ausilio alla navigazione per non vedenti; in particolare, lavori come "A comprehensive review of navigation systems for visually impaired individuals" [1] e "Analysis of Navigation Assistants for Blind and Visually Impaired People: A Systematic Review" [21] forniscono un'analisi dettagliata dello stato dell'arte. Questo capitolo illustra le soluzioni più rilevanti, classificandole in base alle tecnologie impiegate, alle modalità di interazione e alle prestazioni ottenute.

2.1.1 Sistemi basati sulla visione artificiale

2.1.1.1 Tyflos

Tyflos [15] è un sistema progettato per assistere ciechi e ipovedenti nella navigazione autonoma indoor, integrandosi come ausilio secondario al classico bastone bianco. L'approccio si basa su una strategia multimodale che combina la visione stereoscopica per la percezione ambientale e la trasmissione delle informazioni tramite un'interfaccia tattile, riducendo così il sovraccarico uditivo e lasciando libere le mani e le orecchie dell'utente. La struttura dell'informazione segue un formalismo chiamato *Vibration Array Language* (VAL). L'architettura di Tyflos è mostrata nella figura 2.1.

Tecnologie impiegate

- Sistema di visione stereo: due microcamere fissate su occhiali tradizionali catturano immagini in tempo reale per creare mappe di profondità tramite tecniche di visione stereoscopica.

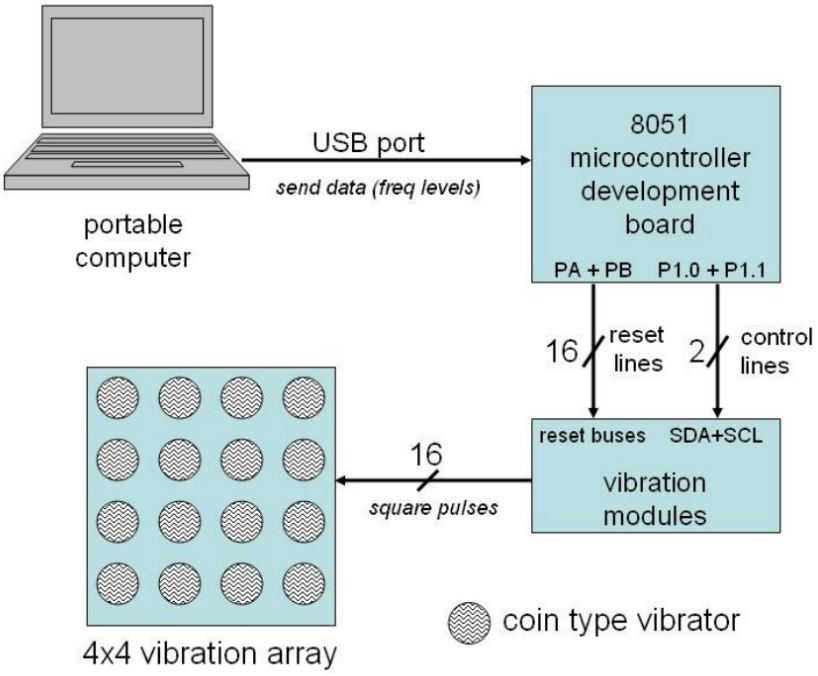


Figura 2.1: Architettura Tyflos (fonte: [15])

- Unità di calcolo portatile: elabora i dati acquisiti dalle telecamere, estraendo ostacoli, percorsi sicuri e oggetti di interesse, con attenzione a persone e oggetti in movimento.
- Interfaccia tattile: consiste in un gilet equipaggiato con una matrice 4×4 di sensori vibro-motori che fornisce all'utente informazioni sulla direzione e la distanza degli ostacoli tramite pattern di vibrazione di diversa intensità e posizione.
- Comunicazione vocale opzionale: microfono e auricolare per comandi vocali e feedback (per funzioni avanzate).
- Formalismo VAL: un linguaggio creato per rappresentare e codificare le informazioni ambientali sulle vibrazioni della matrice tattile, ottimizzando la quantità di dati trasmessi all'utente e la loro interpretabilità.

Metodologia di lavoro

- Architettura del sistema: composto da due moduli principali: (a) Navigator, un ETA per il rilevamento ostacoli e la navigazione sicura; (b) Reader, per il riconoscimento di testi.
- Elaborazione visiva e mappatura 3D:
  - calibrazione e rettifica: allineamento delle immagini delle due telecamere;

  - corrispondeza stereo: creazione di una mappa di disparità per calcolare la distanza degli oggetti nella scena.
  - algoritmo *High-to-Low Resolution*: riduzione intelligente della risoluzione dell'immagine per adattare la complessità visiva alla bassa risoluzione della matrice tattile, preservando però informazioni critiche come i percorsi liberi.

- Analisi del movimento e riconoscimento: implementazione di algoritmi per la stabilizzazione dell'immagine (per compensare il passo dell'utente) e il rilevamento di oggetti in movimento o volti, fornendo avvisi specifici all'utente.
- Sviluppo del linguaggio tattile VAL: un vocabolario formale di pattern vibratori che l'utente deve apprendere per decodificare rapidamente le informazioni ambientali (es. distinguere un ostacolo a terra da uno sospeso).
- Testing e validazione esperimenti condotti sia su soggetti vedenti (bendati) che su non vedenti per valutare:
  - accuratezza nel riconoscimento dei livelli di vibrazione (distanza) e dei simboli tattili (forma ostacolo);
  - prove di navigazione in ambienti interni reali (corridoi con ostacoli, porte, persone in movimento).

Vantaggi e svantaggi

I vantaggi e gli svantaggi sono riassunti nella tabella 2.1.

Tabella 2.1: Analisi dei vantaggi e svantaggi delle componenti del sistema Tyflos

Aspetto	**Vantaggi**	**Svantaggi**
VAL	Approccio *Free-ears*: lascia libero l'udito per i suoni ambientali. Fornisce una mappa 2D spaziale intuitiva sull'addome.	Risoluzione limitata (4×4 array vibro-tattili) che richiede un algoritmo di riduzione aggressivo. Difficoltà per alcuni utenti nel distinguere frequenze di vibrazione vicine.
Visione stereo	Rileva ostacoli a qualsiasi altezza (terra/testa) e calcola la distanza reale senza emettere segnali attivi (come i sonar).	Prestazioni dipendenti dalle condizioni di luce. Richiede calibrazione precisa delle telecamere.
Algoritmo *High-to-Low*	Filtra l'eccesso di informazioni visive, presentando all'utente solo i dati essenziali per la navigazione (ostacoli vs. percorsi liberi).	Possibile perdita di dettagli fini dell'ambiente a causa della drastica riduzione della risoluzione.
Indipendenza dall'infrastruttura	Sistema completamente autonomo e indossabile, non richiede sensori esterni o modifiche all'ambiente (es. beacon).	Hardware del prototipo ancora ingombrante (cavi, gilet, computer portatile) rispetto a soluzioni commerciali più semplici.
Riconoscimento volti/oggetti	Funzionalità avanzata che aggiunge un livello sociale e informativo oltre al semplice evitamento ostacoli.	Aumenta il carico computazionale e la complessità del sistema.

2.1.1.2 ISANA

ISANA [24] è un sistema di navigazione assistita per persone cieche e ipovedenti.È progettato per ambienti indoor ed è basato su una combinazione di tecnologie di visione artificiale (es. *Simultaneous Localization and Mapping* (SLAM)), mappa semantica e interfaccia multimodale. L'architettura di ISANA è mostrata in figura 2.2.

Figura 2.2: Architettura ISANA (fonte: [24])

Tecnologie utilizzate

- Dispositivo *Google Tango*: tablet con fotocamera *Red-Green-Blue-Depth* (RGB-D) (per la percezione della profondità), fotocamera grandangolare (per il tracciamento del movimento) e IMU a nove assi.
- Mappe semantiche indoor: ricavate tramite editor da modelli *Computer-Aided Design* (CAD) 2D per estrarre informazioni geometriche e contestuali (stanze, corridoi, porte, ecc.).
- *Visual Positioning Service* (VPS) e *Area Description File* (ADF): consentono il tracciamento dell'utente all'interno della mappa.

- Algoritmo di allineamento tramite *Singular Value Decomposition* (SVD): per sovrapporre le coordinate reali e quelle della mappa semantica.
- Rilevamento ed evitamento di ostacoli: algoritmo *Time-Stamped Map Kalman Filter* (TSM-KF) per il rilevamento e la stima del movimento di ostacoli in tempo reale tramite i dati RGB-D.
- *SmartCane* [10]: bastone elettronico dotato di pulsanti, vibrazione, IMU e comunicazione *Bluetooth Low Energy* (BLE) per feedback tattile e input dell'utente.
- Interfaccia uomo-macchina multimodale: include input vocale (STT personalizzato) e output audio/tattile.
- Pianificazione percorso globale: con ricalcolo dinamico del percorso per evitare ostacoli (usando l'algoritmo A*).

Metodologia di lavoro

- Preparazione: si importa la planimetria dell'edificio in formato CAD nell'editor che estrae le informazioni semanticamente rilevanti.
- Configurazione: una persona vedente allinea la mappa semantica con la posizione reale tramite punti di controllo riconosciuti sia fisicamente che sulla mappa.
- Navigazione: l'utente cieco interagisce con ISANA tramite comandi vocali o la *SmartCane*. Il sistema guida l'utente con feedback audio e/o vibrazioni verso la destinazione, rilevando e prevedendo la presenza di ostacoli e ricalcolando il percorso in tempo reale.
- *Feedback*: gli allarmi e le informazioni sono gestiti da un sistema di priorità per evitare sovraccarico cognitivo.
- *SmartCane*: fornisce indicazioni di direzione attraverso vibrazioni e può essere utilizzato come input alternativo nei casi in cui l'interazione vocale sia inaffidabile (es. ambienti rumorosi).

Vantaggi e svantaggi

I vantaggi e svantaggi sono riassunti nella tabella 2.2.

2.1.1.3 INVys

Il sistema INVys [52] è stato sviluppato per offrire una soluzione efficace di navigazione indoor destinata a persone cieche o ipovedenti. L'obiettivo principale è permettere una mobilità autonoma e sicura tramite il riconoscimento di ostacoli in tempo reale (micro-navigazione) e il riconoscimento di marcatori ottici per la localizzazione e l'orientamento verso destinazioni interne (macro-navigazione). Il focus è su una soluzione indipendente dall'infrastruttura e utilizzabile anche in ambienti sconosciuti, con un approccio *hands-free* e riscontro immediato. L'architettura è mostrata in figura 2.3:

Tabella 2.2: Analisi dei vantaggi e svantaggi delle componenti del sistema ISANA

Aspetto	Vantaggi	Svantaggi
Piattaforma Google Tango	Fornisce hardware integrato potente per SLAM e stima di profondità senza accessori ingombranti.	La tecnologia di Google Tango è stata cancellata (sostituita da ARCore/ARKit), rendendo l'hardware specifico obsoleto.
***SmartCane* (interfaccia aptica)**	Feedback discreto che non interferisce con l'udito; mantiene l'uso familiare del bastone bianco.	Prestazioni dipendenti dalle condizioni di luce. Richiede calibrazione precisa delle telecamere.
Mappe semantiche	Ricchezza di informazioni contestuali (es. "Sei davanti alla stanza 101") oltre alla semplice geometria.	Possibile perdita di dettagli fini dell'ambiente a causa della drastica riduzione della risoluzione.
Algoritmo TSM-KF	Efficace gestione degli ostacoli dinamici (persone), spesso problematici per sistemi di sola navigazione statica.	Richiede capacità di calcolo elevate per l'elaborazione in tempo reale su dispositivo mobile.
Approccio ibrido (audio + aptico)	Riduce il carico cognitivo: audio per info complesse, vibrazione per guida immediata.	L'utente deve apprendere il significato dei pattern di vibrazione.

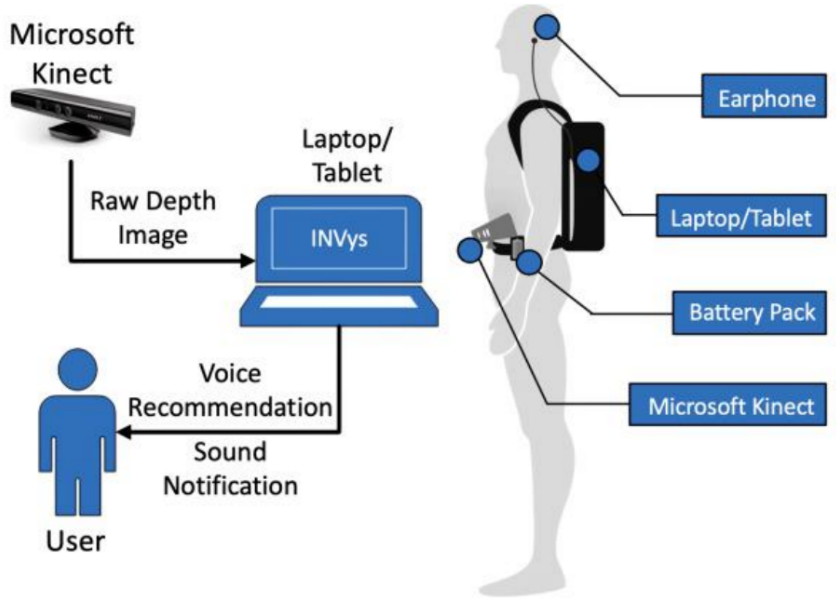


Figura 2.3: Architettura INVys (fonte: [52])

Tecnologie utilizzate

- Fotocamera RGB-D (es. Microsoft Kinect 360): acquisisce contemporaneamente un'immagine a colori e una mappa di profondità, fornendo informazioni su distanza e layout tridimensionale dell'ambiente.
- *Auto-Adaptive Double Thresholding* (AADT): algoritmo proprietario per la rilevazione e classificazione degli ostacoli nella mappa di profondità, ottimizzato per

precisione e robustezza.

- Glifi ottici: marcatori stampati in bianco e nero posizionati nei punti di interesse per il riconoscimento della destinazione tramite visione artificiale, processati con un algoritmo di binarizzazione automatica avanzato.
- Cuffie: usate per feedback e istruzioni trasmesse tramite auricolare all'utente.
- Computer portatile/tablet: usati per l'elaborazione dati in tempo reale.

Metodologia di lavoro

- Progettazione dell'architettura: divisione del sistema in tre sottosistemi principali: rilevamento ostacoli, riconoscimento marcatori ottici e feedback audio all'utente.
- Sviluppo algoritmi di rilevamento ostacoli:
  - implementazione della suddivisione dell'area di osservazione in tre zone (sinistra, centro, destra) per semplificare le indicazioni direzionali;
  - valutazione comparativa tra diverse versioni di sogliatura su criteri di accuratezza, robustezza, precisione, e tempo di esecuzione.
- Implementazione del riconoscimento dei marcatori:
  - sviluppo di un algoritmo di binarizzazione automatica per il riconoscimento affidabile dei glifi su diverse dimensioni, distanze e angolature;
  - test su differenti dimensioni e condizioni di orientamento per misurare accuratezza e performance.
- Test e validazione sperimentale:
  - architettura composta da: Kinect a livello anca, computer portatile e auricolare;
  - test di rilevamento ostacoli con oggetti posizionati a diverse distanze e in diverse zone del campo visivo;
  - prove di riconoscimento marcatori con variazione di dimensione, distanza e inclinazione dei glifi.
- Analisi dati: valutazione quantitativa su accuratezza, deviazione standard, robustezza all'aumentare della distanza, tempi d'esecuzione, e confronto tra condizioni normali e inclinate.

Vantaggi e svantaggi

I vantaggi e svantaggi sono riassunti nella tabella 2.3.

Tabella 2.3: Analisi dei vantaggi e svantaggi delle componenti del sistema INVys

Aspetto	Vantaggi	Svantaggi
Algoritmo AADT per ostacoli	Alta precisione (errore medio 50,2 mm), robustezza a diverse distanze, ottima consistenza	Tempo d'esecuzione leggermente superiore ($\sim$ 9,5 ms) rispetto alle alternative
Fotocamera RGB-D	Acquisisce simultaneamente info su colore e profondità con un solo sensore	Ingombro dispositivo, necessita elaborazione hardware esterna
Glifi ottici	Facile implementazione, riconoscimento fino a 1,5 m (con glifi di 11,5 cm), efficace marcatori	Sensibile a erronea inclinazione/rotazione, accuratezza cala con glyph piccoli
Indipendente da infrastruttura	Non richiede RFID o beacon fissi, quindi facile da implementare in vari contesti	Necessario posizionare marcatori fisici e predisporre l'ambiente.
Feedback audio diretto	Istruzioni chiare, immediate, *hands-free*	Dipende dalla comprensibilità del feedback audio in ambienti rumorosi

2.1.1.4 BLE e Google Tango

L'approccio proposto da [36] mira a sviluppare un sistema di localizzazione e navigazione indoor sicuro e accessibile per persone cieche o ipovedenti. L'obiettivo è combinare tecnologie diverse per superare i limiti delle soluzioni tradizionali, migliorando accuratezza e robustezza nell'assistenza alla mobilità in ambienti sconosciuti.

Tecnologie utilizzate

- BLE: sensori a basso costo che trasmettono segnali radio, utilizzati per la localizzazione approssimativa attraverso il rilevamento dell'intensità del segnale *Received Signal Strength Indicator* (RSSI) e una tecnica di fingerprinting radio.
- Google Tango: dispositivo mobile dotato di fotocamera RGB-D per localizzazione *Visual-Inertial Odometry* (VIO) e creazione di modelli 3D degli ambienti. Consente una localizzazione fine grazie alle mappe di *feature* (ADF).
- Dispositivi vibro-tattili: sensori a infrarossi indossabili che forniscono un riscontro vibrante in relazione alla vicinanza di ostacoli, migliorando la percezione dell'ambiente immediato.
- Applicazione per la navigazione: sviluppata per offrire navigazione turn-by-turn tramite guida audio, integrando BLE e Tango per la localizzazione e l'utilizzo di algoritmi di percorso minimo.

Metodologia di lavoro

- Integrazione sistema ibrido: BLE viene utilizzato per una localizzazione grossolana e per determinare la selezione di mappe/ADF. Tango fornisce una localizzazione dettagliata e in tempo reale sul piano selezionato, tramite allineamento delle mappe 3D a planimetrie 2D con trasformazione affine.

- Modulo di annotazione mappe: consente la definizione di waypoint sulla mappa digitale, collegati a posizioni reali nell'ambiente.
- Analisi statistica: sono stati condotti test per confrontare le diverse soluzioni in termini di durata del percorso, sicurezza, e necessità d'assistenza.

Vantaggi e svantaggi

I vantaggi e svantaggi sono riassunti nella tabella 2.4

Tabella 2.4: Analisi dei vantaggi e svantaggi delle componenti del BLE & Tango

Aspetto	Vantaggi	Svantaggi
BLE	Economico, versatile, utilizzabile con la maggior parte degli smartphone.	Rumoroso, bassa accuratezza, sensibile ai materiali edilizi, localizzazione solo approssimativa.
Google Tango	Localizzazione molto precisa, modelli 3D dettagliati, aggiornamenti in tempo reale.	Richiede dispositivi speciali, errori di deriva su grandi distanze, ADF limitato a un piano.
Ibrido BLE-Tango	Unisce accuratezza e copertura, consente robustezza e affidabilità superiori.	Maggior complessità iniziale, dipendenza da Tango per la localizzazione finale.
Indipendenza da infrastruttura	Non richiede RFID o beacon fissi, quindi facile da implementare in vari contesti.	Necessario posizionare marcatori fisici e predisporre l'ambiente.
Vibro-tattili	Favoriscono la percezione degli ostacoli vicini, potenzialità di sostituire/supportare il bastone.	Necessitano apprendimento, non sostituiscono completamente i supporti tradizionali.

2.1.2 Sistemi basati su feedback aptico e sonoro

2.1.2.1 Sound of Vision

Il progetto Sound of Vision [9] è un sistema indossabile avanzato destinato a supportare la mobilità autonoma di persone cieche o ipovedenti. Il suo obiettivo principale è fornire una rappresentazione tridimensionale e multisensoriale (audio e tattile) dell'ambiente circostante, sia in ambienti interni che esterni, indipendentemente dalle condizioni di illuminazione e senza necessità di sensori o marcatori fissi. Il sistema mira a migliorare la percezione spaziale, la sicurezza e l'integrazione sociale degli utenti con disabilità visiva. L'architettura è illustrata in figura 2.4.

Tecnologie utilizzate

- Caschetto con telecamere 3D e IMU: struttura indossabile dotata di telecamere RGB stereo e sensori di profondità insieme ad una IMU per il tracciamento dell'orientamento della testa.
- Cintura/tuta aptica: dispositivi aptici con 60 attuatori vibranti posti sull'addome per fornire riscontro tattile direzionale e dettagliato.

Figura 2.4: Architettura Sound of Vision (fonte: [9])

- Processore *Graphics Processing Unit* (GPU) dedicato: computer portatile indossabile (in uno zaino) con potenza di calcolo sufficiente per il processamento in tempo reale di dati complessi 3D, audio e aptici.
- Cuffie: per il feedback audio spazializzato, mantenendo la percezione sonora ambientale naturale.
- Software di elaborazione 3D: pipeline che include acquisizione, ricostruzione 3D, segmentazione automatica di oggetti (muri, ostacoli, scale, porte, testi, segnaletica) e modellazione audio/aptica dell'ambiente.
- Telecomando wireless: per il controllo remoto e la personalizzazione in tempo reale delle modalità di feedback.
- Ambienti virtuali per l'allenamento: simulazioni immersive per addestrare gli utenti all'uso del sistema in sicurezza e con progressione controllata.

Metodologia di lavoro

- Analisi dei requisiti utente: coinvolgimento di utenti ciechi/ipovedenti e operatori *Orientation & Mobility Operator* (O&MO) per definire bisogni, barriere e funzionalità richieste.
- Progettazione modulare: sviluppo hardware integrato su caschetto e cintura, con agevole montaggio e comfort; selezione di sensori complementari per garantire funzionamento sia all'interno che all'esterno e in ogni condizione luminosa.

- Sviluppo e integrazione software:
  - acquisizione dati con sincronizzazione tra fotocamera stereo, sensore di profondità e IMU;
  - elaborazione 3D in tempo reale mediante GPU, con segmentazione e riconoscimento di elementi rilevanti;
  - modellazione e rendering audio-aptica personalizzabile in tempo reale.
- Test preliminari e ottimizzazione: valutazione tecnica di accuratezza, latenza e robustezza in scenari interni ed esterni con utenti ciechi e vedenti.
- Addestramento e valutazione d'uso: sessioni di allenamento sia in ambienti virtuali sia reali, con raccolta di dati quantitativi (collisioni, tempi, precisione) e qualitativi (questionari, feedback).

Vantaggi e svantaggi

I vantaggi e svantaggi sono riassunti nella tabella 2.5.

Tabella 2.5: Analisi dei vantaggi e svantaggi delle componenti di Sound of Vision

Aspetto	Vantaggi	Svantaggi
Tecnologie 3D e sensori	Acquisizione ambientale ricca e dettagliata. Funziona sia all'interno che all'esterno. Non richiede marcatori fissi.	Ingombro e peso iniziali del dispositivo. Complessità del sistema indossabile.
Feedback audio-aptico	Rappresentazione naturale e personalizzabile. Supporto multisensoriale che migliora la percezione spaziale	Possibile sovraccarico informativo per l'utente. Necessità di adattamento e training.
Indipendenza ambientale	Utilizzabile in vari ambienti e condizioni di luce.	Costi più elevati rispetto a soluzioni basate su beacon o tecnologie più semplici.
Addestramento e usabilità	Efficace training virtuale e reale. Coinvolgimento diretto degli utenti nel miglioramento continuo.	Periodo di apprendimento richiesto per uso efficace.

2.2 Tipologia di calcolo percorsi

Per il calcolo del percorso, gli approcci si dividono in due macro categorie, quelli usati per la macro-navigazione e quelli usati per la micro-navigazione.

2.2.1 Macro-navigazione

Per Macro-Navigazione si intende il calcolo del percorso da un punto A iniziale ad un punto B finale. I sistemi di navigazione per non vedenti descritti precedentemente usano diversi algoritmi e strategie per calcolare il percorso ottimale:

- *Point of Interest Graph* (PoI-graph) e Algoritmi di *Shortest Path*: molti sistemi, come quelli basati su SLAM, costruiscono una mappa dei punti di interesse PoI-graph durante una fase di esplorazione iniziale. Gli ambienti sono rappresentati come grafi, dove i nodi sono punti chiave (stanze, intersezioni, ingressi) e gli archi rappresentano i percorsi percorribili.
- Algoritmi utilizzati:
  - Dijkstra o A* per trovare il percorso globalmente più breve dall'origine alla destinazione;
  - in alcuni casi, vengono anche considerati pesi sugli archi che riflettono la difficoltà o la distanza tra due nodi.
- Funzionamento:
  - quando l'utente specifica la destinazione, il sistema calcola il percorso sul grafo;
  - in tempo reale, la posizione dell'utente è stimata tramite tecniche di localizzazione per aggiornarlo con istruzioni di svolta o ricalcolare il percorso in caso di deviazioni.
- Sotto-obiettivi dinamici: per la navigazione dettagliata, alcuni dispositivi selezionano automaticamente sotto-obiettivi come intersezioni o cambi di direzione, per facilitare la guida dell'utente e gestire deviazioni e ostacoli.

2.2.2 Micro-navigazione

Per evitare collisioni, i sistemi utilizzano sensori e tecniche di computer vision per riconoscere oggetti e mappare l'ambiente circostante.

- Sensori Utilizzati:
  - telecamere RGB-D;
  - telecamere stereo;
  - sensori a ultrasuoni (per compensare limiti delle telecamere, in caso di materiali trasparenti o superfici riflettenti);
  - IMU per stimare la posizione e orientamento della testa o del dispositivo.
- Raccolta e segmentazione dell'ambiente:
  - Le immagini/mappe di prodonfità acquisite vengono elaborate per costruire una rappresentazione 3D o una mappa semantica dell'ambiente.
  - Vengono rilevate superfici (pavimento, pareti) attraverso tecniche di segmentazione delle nuvole di punti o delle superfici piane.
  - Gli oggetti e gli ostacoli vengono separati dal suolo tramite analisi delle normali e delle distanze nei dati 3D o dalla profondità.
- Algoritmi di rilevamento e mappatura:

  - SLAM: utilizzato per localizzare il dispositivo nell'ambiente e aggiornare costantemente la mappa delle superfici e degli oggetti;
  - Segmentazione e classificazione degli oggetti: le superfici segmentate vengono analizzate per identificare:
    * ostacoli generici;
    * oggetti particolari (porte, scale, segnali);
    * spazi liberi per la navigazione (es. corridoi).
  - Rilevamento di ostacoli dinamici: alcuni sistemi applicano tecniche di identificazione del movimento per monitorare oggetti che si spostano (come persone).

- Integrazione dei dati e feedback all'utente: i dati degli ostacoli (posizione, dimensioni, distanza) vengono usati per:
  - Segnalare in tempo reale la presenza di ostacoli tramite output audio, feedback vibro-tattile o combinati.
  - Aggiornare il percorso suggerito se viene rilevato un ostacolo improvviso.
  - Calcolare le direzioni di cammino più sicure.

2.3 Reti neurali per la segmentazione 3D

L'articolo "A survey of 3D object detection" [27], offre un'analisi dettagliata dei principali metodi di object detection 3D, classificandoli in quattro categorie principali sulla base del tipo di dato di input:

- *Point Cloud* (PCM): utilizzano nuvole di punti 3D (ottenute tramite *Light Detection and Ranging* (LiDAR), ricostruzione 3D con immagini o modelli CAD) per rappresentare la geometria degli oggetti
  - vantaggi: informazioni spaziali ricche e alta precisione;
  - svantaggi: alto costo di acquisizione dati e computazionalmente onerosi, manca l'informazione della texture.

- *Monocular Image* (MCIM): usano una singola immagine RGB come input, sfruttando reti profonde per ricavare informazioni 3D
  - vantaggi: costo dati basso, sfruttano tecniche 2D mature;
  - svantaggi: mancanza di informazioni di profondità precise, minor accuratezza rispetto ai metodi PCM.

- *Multi-view Image* (MVIM): combinano immagini da più angolazioni (camere stereo) per stimare la profondità e generare la struttura 3D
  - vantaggi: migliore precisione rispetto al monoculare, grazie all'uso della profondità stimata;

  - svantaggi: richiede sintonia delle camere, la precisione dipende fortemente dalla qualità della stima della profondità.

- *Fusion of Point Cloud and Image* (FPCIM): integrano dati LiDAR e immagini RGB per sfruttare sia informazioni geometriche sia di texture. La fusione può essere effettuata in tre modi:

  - *Early-fusion*: combina i vari tipi di dati prima dell'estrazione delle *feature*;
  - *Late-fusion*: combina le *feature* estratte dell'analisi dei vari tipi di dati;
  - *Deep-fusion*: la fusione avviene in più punti all'interno della rete, tipicamente a diversi livelli di astrazione delle *feature*. Si utilizzano meccanismi iterativi (es. calcolo multiplo della media elemento per elemento) per fondere le *feature* di diversi livelli, combinando vantaggi di early e late fusion.

  - vantaggi: massima accuratezza sfruttando punti di forza di entrambe le modalità;
  - svantaggi: computazionalmente più costosi e complessi da implementare.

2.3.1 Metodi PCM

2.3.1.1 SceneScript

SceneScript [3] è un metodo per la ricostruzione automatica di scene 3D interne che rappresenta l'intera scena come una sequenza di comandi in un linguaggio strutturato di alto livello, generata auto-regressivamente da dati visivi. Ha lo scopo di:

- Superare le limitazioni delle rappresentazioni tradizionali (mesh, voxel, point cloud, radiance fields) offrendo un formato compatto, interpretabile, modificabile e facilmente estendibile.

- Fornire una rappresentazione metrica, semantica e geometrica capace di descrivere in modo preciso e scalabile layout architettonici (muri, porte, finestre) e oggetti 3D (*Bounding Box* (BB) orientate, forme primitive).

Architettura

Architettura *Transformer encoder-decoder* (mostrata in figura 2.5):

- *Encoder* multimodale: riceve come input sequenze video egocentrici, trasformate in rappresentazioni latenti tramite:

  - nuvola di punti (convoluzione sparse 3D su nuvole di punti ricostruite da SLAM);
  - immagini (*Convolutional Neural Network* (CNN) + transformer *RayTran 2D-3D bidirezionale*);
  - combinato (immagini + punti).

- *Decoder* auto regressivo: genera token che rappresentano i comandi del linguaggio strutturato SceneScript (es. `make_wall`, `make_door`, `make_bbox`).

Linguaggio strutturato progettato manualmente con comandi parametrizzati per elementi architettonici e oggetti, tokenizzati in sequenze di interi per la predizione autonoma. Modello addestrato con funzione di perdita a entropia incrociata standard (*standard cross-entropy*) per la predizione del *next-token*.

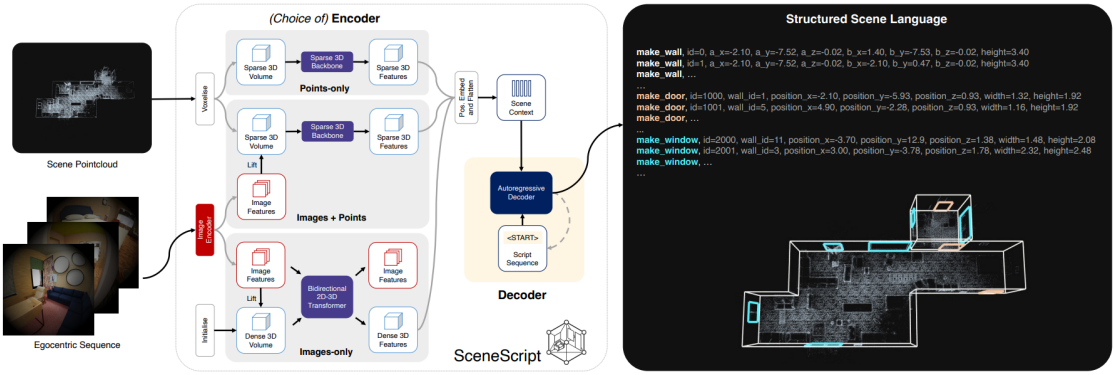


Figura 2.5: Architettura SceneScript (fonte: [3])

Dataset

Aria Synthetic Environments (ASE) [40] è un nuovo dataset sintetico creato appositamente con 100000 scene di interni complete, ciascuna composta da:

- video di percorsi egocentrici renderizzati con traiettorie realistiche;
- nuvole di punti generate da SLAM;
- annotazioni dettagliate in linguaggio SceneScript (comandi per muri, porte, finestre e oggetti);
- *ground truth* fotorealistico.

Include anche un insieme per il test con 1000 scene.

ScanNet [14]: utilizzato per validare la generalizzazione su dati reali.

Input e output

- Input:
  - Video egocentrici di percorsi all'inerno.
  - Opzionale: nuvole di punti stimate da SLAM semi-denso a 5cm di risoluzione, immagini RGB con pose.
  - I punti possono essere arricchiti con *feature* basati sulle immagini proiettate (*lifted features*).

- Output: Sequenza tokenizzata di comandi (linguaggio SceneScript), che rappresenta:
  - elementi architettonici: muri (`make_wall`), porte (`make_door`), finestre (`make_window`);
  - oggetti 3D con *Oriented Bounding Box* (OBB) (`make_bbox`);
  - estensioni facoltative: primitive volumetriche (`make_prim`) e stati di oggetti (es. apertura porte).

Statistiche e risultati

Metriche di valutazione

- *F1 Score* sull'entità predetta, calcolato su diversi elementi (pareti, porte, finestre) con soglie di accuratezza per la distanza degli angoli (da 1cm a 100cm).
- *F1 Score* per l'individuazione di oggetti 3D (*Intersection over Union* (IoU) 0,25 e 0,5), preferito rispetto a *mean Average Precision* (mAP) per questioni di classificazione dei comandi predetti.

Conclusioni e vantaggi

- Compatto ed interpretabile: la scena è codificata in pochi KB di testo, facilmente editabile ed estendibile (es. per curve, stati degli oggetti, primitive parametriche).
- Performance: netta superiorità sulla stima della planimetria e competitività sull'individuazione di oggetti 3D rispetto ai metodi tradizionali (mostrato in tabella 2.6), anche senza ottimizzazioni specifiche su ciascun tipo di input.
- Flessibilità: la pipeline può essere aggiornata con nuovi obiettivi semplicemente aggiungendo nuovi comandi nel linguaggio strutturato, senza modifiche architetturali alla rete.
- Generalizzazione: allenato con solo su dati sintetici, mostra ottimi risultati anche su dataset reali (ScanNet [14]).

Tabella 2.6: Confronto tra SceneScript ed altri modelli[a]

Metodo	**Input**	**F1 @ IoU 0,25**	**F1 @ IoU 0,5**
3DETR [33]	Point Cloud	0,201	0,078
Cube R-CNN [6]	RGB	0,394	0,228
ImVoxelNet [44]	RGB	0,584	0,516
SceneScript	Point Cloud	0,620	0,577

[a] fonte: [3].

2.3.2 Metodi MCIM

2.3.2.1 YOLO-World

YOLO-World [13] è una rete neurale che ha lo scopo di:

- Realizzare un metodo di identificazione di oggetti a "vocabolario aperto" (capace cioè di riconoscere categorie non viste durante l'addestramento) con elevata efficienza e inferenza real-time, adattabile a scenari reali.
- Superare i limiti dei metodi tradizionali YOLO a vocabolario chiuso (con categorie predefinite e fisse) estendendo la capacità di riconoscere oggetti di centinaia o migliaia di categorie mai viste.
- Integrare le capacità di modelli linguaggio visivo (*vision-language*) in un framework di identificazione leggero e veloce.

Architettura

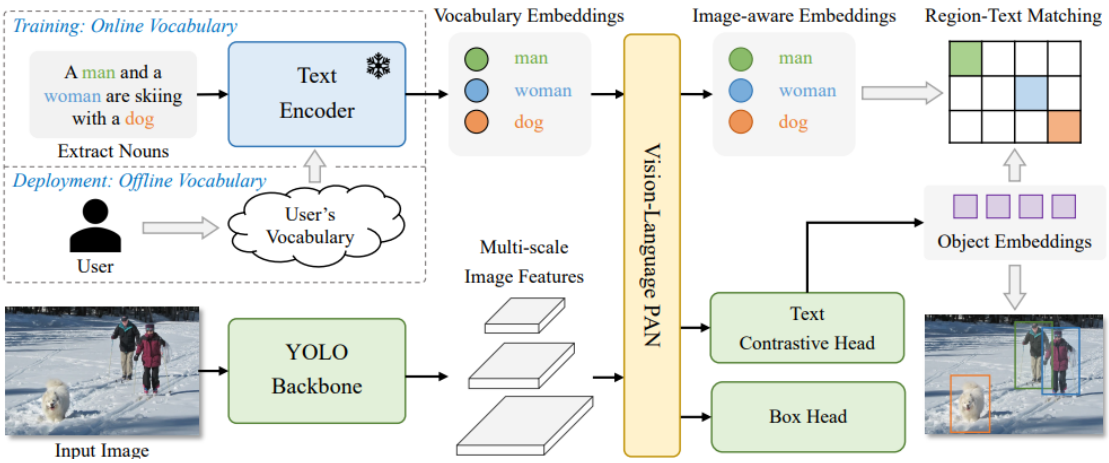


Figura 2.6: Architettura YOLO-World (fonte: [13])

Architettura (figura 2.6) basata su YOLOv8 [20], un modulo multi-scala che aggrega e fonde testo e immagini (chiamato *Re-parameterizable Vision-Language Path Aggregation Network* (RepVL-PAN)) tramite:

- *Layer CrossStage Partial (CSP) text-guided* che sono estensioni dei layer CSP normali per iniettare informazioni linguistiche.
- Meccanismo di attenzione con aggregazione spaziale delle immagini (*Image-Pooling Attention*) per arricchire le rappresentazioni vettoriali testuali a partire dalle caratteristiche visive.

Utilizza l'encoder di testo *Contrastive Language-Image Pretraining* (CLIP) per trasformare input testuali in *embedding*. Il testo *(categoria, noun phrase)* e l'immagine viaggiano paralleli, con forte interazione dentro RepVL-PAN. Impiega un meccanismo di *region-text*

contrastive learning per affinare la corrispondenza tra box e categorie testuali. Durante l'inferenza, il modello può re-parametrizzare il testo in pesi convoluzionali, eliminando la necessità di elaborarlo continuamente e migliorando l'efficienza.

Dataset

Allenamento su un dataset composto da:

- *Objects365* [45]: comprende 365 categorie, ∼609k immagini con ∼9,6M di annotazioni bounding box.
- *GQA* [2]: è un dataset composto da domande e risposte visive, ∼621k immagini, ∼3,6M regioni.
- *Flickr30k Entities* [37]: composto da ∼149k immagini con regioni e frasi associate.
- *Conceptual Captions (CC3M)* [46]: ∼246k immagini con pseudo-etichettatura per generare coppie regione-testo (mediante *Grounded Language-Image Pretraining* (GLIP) e CLIP).

Dataset di valutazione:

- *LVIS* [18]: oltre 1200 categorie, destinato a valutare l'identificazione a "vocabolario aperto" con metriche *Average Precision* (AP) e velocità di inferenza.
- *MS COCO* [28]: dataset standard per l'identificazione a vocabolario chiuso, per test di *fine-tuning* e comparazione.

Input e output

- Input: immagine RGB e testo descrittivo sotto forma di lista di categorie, nomi di oggetti o *noun phrases* (per esempio *man*, *dog*, *chair*).
- Output: BB predetti con punteggi ed *embedding* associati per ogni oggetto rilevato, con classificazione aperta alle categorie fornite a testo (compresa possibilità di vocabolario offline costruito e riusato).

Statistiche e risultati

Metriche di valutazione

- AP: valore di precisione della detection;
- *Average Precision on rare* (APr), *Average Precision on common* (APc), *Average Precision on frequent* (APf): precisione su categorie rare, comuni e frequenti;
- *Frame per Second* (FPS): indica la velocità di inferenza su GPU moderna (NVIDIA V100).

Confronto su LVIS

YOLO-World-L (la versione più completa di YOLO-World) supera lo stato dell'arte in AP su *LVIS* (mostrati in tabella 2.7) mantenendo una velocità delle 40 alle 200 volte superiore rispetto ai metodi precedenti.

Tabella 2.7: Confronto tra YOLO-World ed altri modelli su dataset LVIS [a]

Metodo	Backbone	FPS	AP	APr	APc	APf
GLIP-T [26]	Swin-T [29]	0,12	26,00	20,80	21,40	31,00
DetCLIP-T [54]	Swin-T [29]	2,30	34,40	26,90	33,90	38,00
YOLO-World-L	YOLOv8-L [20]	52,00	35,40	27,60	34,10	38,00

fonte: [13].

2.3.3 Metodi MVIM

2.3.3.1 Pseudo-LiDAR

L'approccio seguito da [51] ha come scopo quello di migliorare drasticamente la precisione della rilevazione 3D di oggetti in scenari di guida autonoma usando solo immagini (monoculari o stereo), superando il divario tradizionalmente attribuito alla scarsa qualità della stima della profondità rispetto a sistemi basati su LiDAR .

Architettura

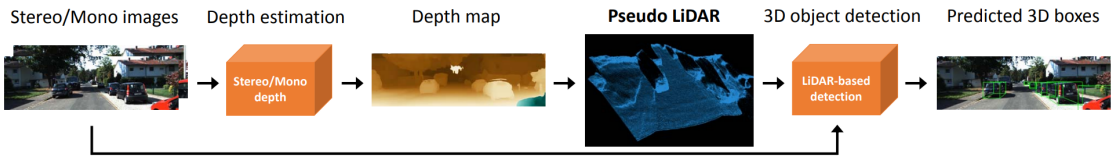


Figura 2.7: Pipeline di lavoro di Pseudo-LiDAR (fonte: [51])

L'architettura (mostrata in figura: 2.7) è divisa in due stadi:

- Stima della profondità a partire da immagini monoculari o stereo, ottenuta tramite reti profonde di stima della disparità (es. PSMNet, DispNet, DORN).
- Conversione della mappa di profondità in una rappresentazione pseudo-LiDAR (nuvola di punti tridimensionale), identica a quella del vero sensore LiDAR. Questa rappresentazione viene poi fornita come input a standard pipeline di identificazione 3D originariamente progettate per dati LiDAR (es. AVOD, PointNet), senza modificarne l'architettura.

Dataset

KITTI Vision Benchmark Suite [17]:

- Composto da 7,481 immagini per il training e 7,518 per il test.
- Per ogni immagine sono forniti: dati stereo (sinistra/destra), annotazioni di oggetti 3D e file di calibrazione della fotocamera.
- Le performance sono valutate secondo le metriche ufficiali KITTI per 3D e *Bird's Eye View* (BEV) per l'identificazione degli oggetti.

Input e output

- **Input**:
  - coppia di immagini stereo o singola immagine monoculare;
  - parametri di calibrazione della fotocamera (matrice intrinseca, baseline stereo).
- **Output**:
  - nuvola di punti generata dalla mappa di profondità;
  - liste di BB 3D per ogni oggetto rilevato (coordinate centro, dimensioni, orientamento).

Statistiche e risultati

Metriche di valutazione

- *Average Precision on Bounding Box* (APBB): precisione BB 3D;
- *Average Precision Bird-Eye View* (APBEV): precisione proiezione sul piano stradale;
- risultati separati per i livelli *Easy*, *Moderate* e *Hard*; e con soglie di IoU 0,5 e 0,7.

Confronto su KITTI

La pipeline pseudo-LiDAR con AVOD [22] raggiunge valori di precisione tripli rispetto ai precedenti approcci basati solo su immagini. I risultati sono in tabella 2.8.

Tabella 2.8: Confronto tra Pseudo-LiDAR ed altri modelli su dataset KITTI[a]

Detection algorithm	**Input signal**	**IoU = 0,5**			**IoU = 0,7**		
		Easy	Moderate	Hard	Easy	Moderate	Hard
MONO3D [11]	Mono	30,5 / 25,2	22,4 / 18,2	19,2 / 15,5	5,2 / 2,5	5,2 / 2,3	4,1 / 2,3
MLF-MONO [53]	Mono	55,0 / 47,9	36,7 / 29,5	31,3 / 26,4	22,0 / 10,5	13,6 / 5,7	11,6 / 5,4
AVOD	Mono	61,2 / 57,0	45,4 / 42,8	38,3 / 36,3	33,7 / 19,5	24,6 / 17,2	20,1 / 16,2
3DOP [12]	Stereo	55,0 / 46,0	41,3 / 34,6	34,6 / 30,1	12,6 / 6,6	9,5 / 5,1	7,6 / 4,1
MLF-STEREO [53]	Stereo	-	53,7 / 47,4	-	-	19,5 / 9,8	-
AVOD	Stereo	89,0 / 88,5	77,5 / 76,4	68,7 / 61,2	74,9 / 61,9	56,8 / 45,3	49,0 / 39,0

[a] fonte: [51].

Capitolo 3

Strumenti e Tecnologie

A valle della disamina sullo stato dell'arte affrontata nel capitolo precedente, il presente capitolo illustra nel dettaglio gli strumenti, sia hardware che software, selezionati per lo sviluppo del progetto di tesi. Nello specifico, la trattazione introdurrà i dispositivi indossabili impiegati per l'acquisizione dei dati ambientali, per poi approfondire le librerie di tracciamento spaziale e i modelli di intelligenza artificiale scelti per la percezione visiva. Infine, verrà descritto il motore grafico utilizzato per orchestrare i vari moduli e gestire l'interfaccia di navigazione.

3.1 Project Aria

Project Aria [39] è un programma di ricerca del Meta Reality Lab [31] finalizzato a supportare i ricercatori nella comprensione e nella risoluzione delle criticità legate allo sviluppo di dispositivi per l'IA e l'*Augmented Reality* (AR). Gli occhiali prodotti da Project Aria forniscono ai ricercatori un dispositivo in grado di catturare il contesto dell'utente grazie a un sistema di sensori avanzati. Project Aria utilizza questi occhiali per produrre dataset e strumenti *open-source* per migliorare i sistemi IA per la *Scene Understanding* e la *Contextual IA*, come il dataset ASE [40], o di reti che si occupano di elaborare dati da immagini o nuvole di punti per segmentare o ricostruire un ambiente.

3.1.1 Occhiali Aria Gen1

Per questa tesi sono stati utilizzati gli occhiali Aria Gen1 [16], prodotti nel 2020 e mostrati in figura 3.1. Tali dispositivi sono progettati per un utilizzo prolungato (*all-day wearable)*, ovvero non intrusivi e socialmente accettabili. A differenza di altri prodotti commerciali, questi occhiali non hanno un display, ma fungono esclusivamente da dispositivo di registrazione e streaming di dati.

3.1.1.1 Sensori

Gli occhiali, mostrati nella figura 3.2, sono composti dai seguenti sensori:

- Sensori Visivi:

Figura 3.1: Occhiali Aria Gen1 (Fonte: [16])

  - Due fotocamere monocromatiche con un *Horizontal Field Of View* (HFOV) di 150°, lenti tipo Fisheye 624 e risoluzione 640 × 480 *pixel* (px). Sono orientate verso l'esterno per massimizzare la visione periferica, mantenendo una parte in sovrapposizione. Usate per la localizzazione e la mappaturaSLAM.
  - Una fotocamera *Point Of View* (POV) RGB con un HFOV di 110° con lenti di tipo Fisheye 624 e risoluzione di 2880 × 2880 px. È posta sul lato sinistro degli occhiali con 4° di bias verso il pavimento. Utilizzate per catturare la prospettiva dell'utente.
  - Due fotocamere monocromatiche per il tracciamento dell'occhio con un *Diagonal Field Of View* (DFOV) di 80°, con lenti di tipo Kannala Brandt K3 ed una risoluzione 320 × 240 px.

- Audio ed Inerziali:
  - Un array di 7 microfoni, 5 sul fronte e gli altri due piazzati sul lato sinistro e destro, per catturare l'audio spaziale a 24 bit ed una frequenza fino a 48 kHz.
  - Due unità IMU indipendenti. Una posta a sinistra con una frequenza di 800 Hz e una saturazione limite di 4g (accelerometro) e 500°/s (giroscopio). Mentre quella di destra ha una frequenza di 1000 Hz con una saturazione limite di 8g e 1000°/s.

- Posizionamento e Ambiente:
  - Magnetometro: posizionato sul bordo degli occhiali per ridurre l'interferenza elettromagnetica. Misura il campo magnetico dell'ambiente con una risoluzione di 0.1μT ed una frequenza di 10 Hz.
  - Barometro e Termometro: catturano la pressione e la temperatura dell'ambiente con una risoluzione, rispettivamente, di 0.66 Pa e 0.005°C ed una frequenza di 50 Hz.
  - Un ricevitore *Global Navigation Satellite System* (GNSS): supporta sia il sistema GPS che il sistema Galileo, che fornisce misure di *pseudo-range* e soluzioni di latitudine/longitudine/altezza con una risoluzione di 1 Hz.
  - Wi-Fi e Bluetooth: questi sensori analizzano e registrano il RSSI dei punti di accesso Wi-Fi (sia da 2.4GHz che 5GHz) e Bluetooth con una frequenza di 0.1 Hz

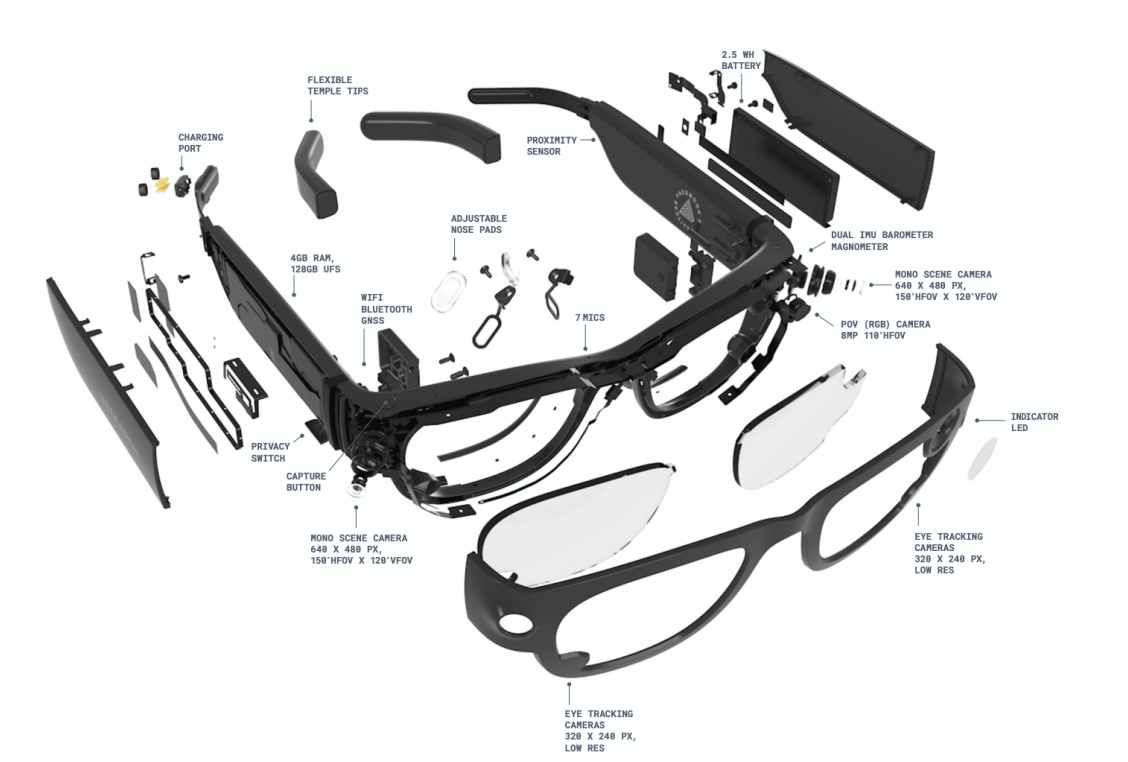


Figura 3.2: Sensori Occhiali Aria Gen1 (Fonte: [16])

I sensori sono montati su un telaio rigido per garantire un allineamento preciso e un peso massimo di 75 grammi. Tutti i flussi dei sensori sono calibrati temporalmente rispetto a una sorgente locale.

3.1.1.2 Registrazione

Attraverso un'applicazione, o tramite l'*Application Programming Interface* (API) ufficiale, è possibile registrare o eseguire uno streaming in tempo reale dei vari sensori. In caso di registrazione, il dispositivo salva in formato *Virtual Reality Sensor* (VRS) [43], che è un contenitore di dati implementato per registrare e riprodurre dati di sensori AR e supportare file di dimensioni elevate.

3.1.1.3 MPS

Il client *Machine Perception Services* (MPS) è un insieme di algoritmi che elaborano i dati grezzi registrati dagli occhiali su un server remoto. L'output fornisce:

- Traiettoria: calcolo preciso della posizione e dell'orientamento del dispositivo nello spazio, utilizzando algoritmi di VIO e SLAM all'avanguardia, fornendo sia odometria ad alta frequenza sia traiettorie *closed-loop* ottimizzate globalmente.

- Calibrazione: stima continua dei parametri di calibrazione per compensare le deformazioni fisiche del dispositivo durante l'uso o dovute al deterioramento.
- Nuvola di Punti: nuvole di punti semi-dense per la ricostruzione geometrica dell'ambiente statico. Ottenuta generando nuovi punti in regioni dell'immagine con un alto gradiente e tracciandoli nel tempo attraverso le due fotocamere monocromatiche, e poi posizionandoli nell'ambiente globale di riferimento definito dalla traiettoria.
- Tracciamento dello sguardo: grazie alle due fotocamere per il tracciamento oculare, il sistema traccia lo sguardo dell'utente stimando, ad ogni frame, un raggio 3D ancorato alla cornice centrale della pupilla [1].

3.1.1.4 Limitazioni

Gli occhiali non consentono l'elaborazione sul device, ma hanno bisogno di un sistema di appoggio (es. MPS o un server dedicato). Il client MPS richiede il caricamento della registrazione sul server remoto, ponendo sia un problema di privacy (discusso nel capitolo 5 di [16]) sia un impedimento per l'elaborazione in tempo reale. In più il tempo di elaborazione da parte del client MPS può essere molto oneroso, come mostrato in tabella 3.1. Per maggiore dettaglio nel Grafico 3.3 viene mostrato che la dimensione del file non è il solo fattore che influisce sulla durata dell'elaborazione, in quanto, anche con una dimensione del file simile, i tempi di esecuzione aumentano.

3.2 ORB-SLAM3

ORB-SLAM3 [8] è una libreria *open-source* in grado di calcolare lo SLAM visuale ed eseguire una mappatura di uno o più ambienti. Il sistema estende le capacità dei suoi predecessori ORB-SLAM [34] e ORB-SLAM2 [35], supportando diverse configurazioni di sensori:

- monoculare;
- monoculare inerziale;
- stereo;
- stereo inerziale;
- RGB-D;
- RGB-D inerziale;

Il sistema è in grado di operare in tempo reale sia in ambienti esterni che interni, associando i dati a breve e a lungo termine, con un accuratezza fino a 10 volte maggiore delle versioni precedenti.

[1] L'origine della cornice è calcolata come il punto medio tra la pupilla di destra e quella di sinistra.

Tabella 3.1: Stima della durata delle elaborazioni da parte del MPS[a]

Durata rec. (sec)	**Dim. (MB)**	**RGB Cam (px[b] - FPS)**	**Eye Cam (FPS)**	**Mono Cam (FPS)**	**Sensori Attivi**						**Durata MPS (mm:ss)**
					IMU	**Mag**	**Bar**	**GPS**	**BT**	**Wi-Fi**	
26	424,08	1408 - 30	30	30	✓	✓	✓	✓	✓	✓	18:53,603
3	39,13	1408 - 30	30	30	✓	✓	✓	✓	✓	✓	12:24,405
17	275,03	1408 - 30	30	30	✓	×	×	×	×	×	13:36,484
27	282,23	1408 - 20	20	20	✓	✓	✓	✓	✓	✓	18:13,129
11	111,49	1408 - 20	20	20	✓	✓	✓	✓	✓	✓	18:04,258
17	171,79	1408 - 20	20	20	✓	×	×	×	×	×	07:53,852
22	1219,56	2880 - 20	20	20	✓	✓	✓	✓	✓	✓	22:31,410
16	902,79	2880 - 20	20	20	✓	×	×	×	×	×	15:58,053
20	571,16	2880 - 10	10	10	✓	✓	✓	✓	✓	✓	14:55,071
21	111,67	1408 - 10	10	10	✓	×	×	×	×	×	18:38,251
22	305,20	2880 - 5	5	5	✓	✓	✓	✓	✓	✓	17:10,204
25	71,69	1408 - 5	5	5	✓	✓	✓	✓	✓	✓	12:57,022
24	261,86	1408 - 20	20	-[c]	✓	×	×	×	×	×	28:28,579

[a] Questi dati sono stati ottenuti registrando su una stanza di circa 13 m^2.
[b] Indica la risoluzione usata per la registrazione.
[c] Indica che il sensore è stato spento durante la registrazione.

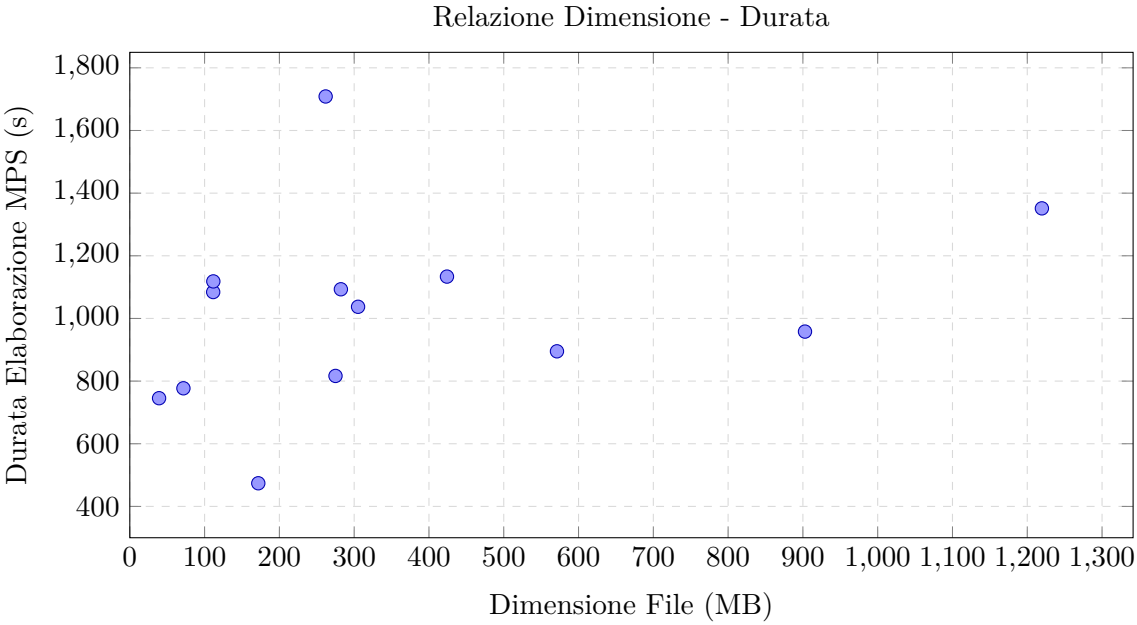


Figura 3.3: Scatter plot della durata di elaborazione in funzione della dimensione del file dal client MPS

3.2.1 Principali Componenti

I principali componenti del sistema, mostrati in figura 3.4, sono:

- Atlas e sistema multi-Mappa: ORB-SLAM3 genera un atlante (Atlas) che il sistema costruisce in tempo reale, composto da più mappe, anche disconnesse. Esiste una mappa "attiva" in cui il sistema si posiziona, mentre le altre sono "non attive". Quando il sistema si "perde" crea una nuova mappa che viene poi fusa a quelle precedenti quando si ritorna in luoghi noti.

- tracciamento: elabora le informazioni dei sensori in tempo reale per calcolare la posa del frame corrente rispetto alla mappa attiva. In modalità visuale-inerziale, stima anche la velocità del corpo e i bias dell'IMU.

- *Local Mapping*: aggiunge nuovi keyframe e punti alla mappa attiva, rimuovendo quelli ridondanti e raffinando la struttura. Include inoltre una tecnica di inizializzazione IMU rapida basata sulla stima *Maximum-a-Posteriori* (MAP), che permette di calibrare i parametri inerziali in soli 2 secondi.

- *Loop and Map Merging*: rileva regioni comuni tra la mappa attiva e l'intero Atlas tramite un algoritmo di riconoscimento dei luoghi (*place recognition*) a richiamo elevato. Se l'area appartiene alla mappa attiva, esegue una correzione del loop; se appartiene a una mappa diversa, fonde le mappe in un'unica entità attiva.

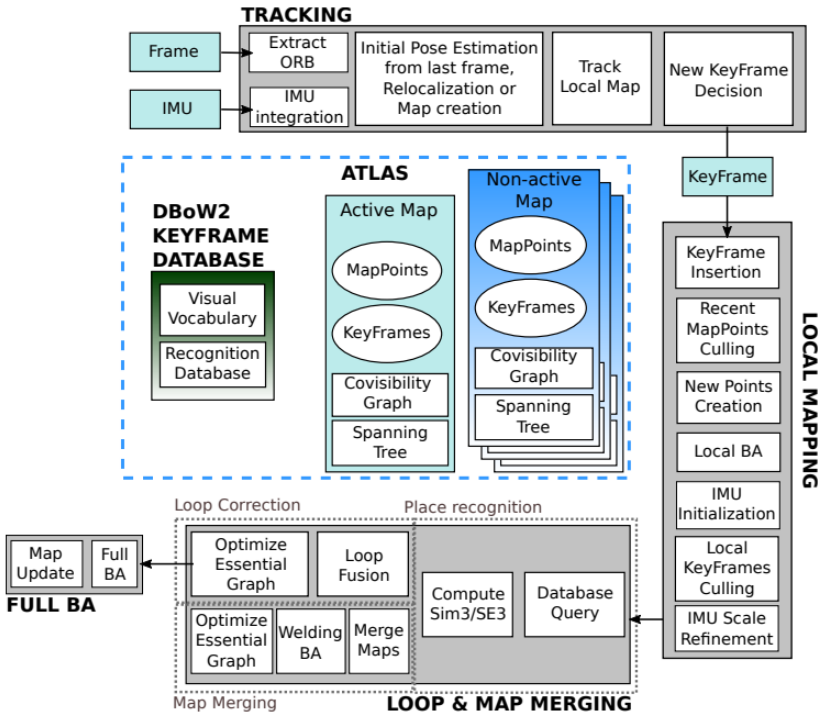


Figura 3.4: Principali Componenti del Sistema ORB-SLAM3 (Fonte: [8])

- Supporto per diversi tipi di sensori: il sistema fornisce un documento che spiega come creare il file di configurazione (in formato *YAML Ain't Markup Language* (YAML)) che consente al sistema di supportare qualsiasi tipo di fotocamera o sensore IMU.

3.2.2 Limitazioni

Nonostante l'elevata accuratezza, ORB-SLAM3 presenta alcune limitazioni critiche:

- Ambienti a bassa tessitura: il sistema si basa sull'estrazione e il matching di feature descrittive (*Oriented FAST and Rotated BRIEF* (ORB)), il che lo rende meno robusto in ambienti con poche texture (come corridoi omogenei) rispetto ai metodi diretti, che utilizzano direttamente le intensità dei pixel.

- Robustezza del tracking: sebbene il confronto dei descrittori sia efficace per l'associazione di dati a lungo termine, risulta meno robusto per il tracciamento continuo rispetto a metodi basati su Lucas-Kanade [30], che sfruttano meglio l'informazione fotometrica.

- Inizializzazione IMU: in situazioni di movimento lento o assenza di rotazioni su rollio e beccheggio (come un'auto in un'area pianeggiante), l'IMU può essere difficile da inizializzare a causa della scarsa osservabilità dei parametri inerziali.

- Punti distanti in esterni: in ambienti aperti, punti molto distanti (come le nuvole) possono introdurre deriva (*drift*) nella posa della fotocamera, rendendo necessaria l'implementazione di filtri per scartare punti oltre una certa distanza.

3.3 YOLO

La rete YOLO [42] propone un approccio radicalmente nuovo al problema del rilevamento degli oggetti. Fino a quel momento, i sistemi di rilevamento più avanzati riadattavano classificatori preesistenti per eseguire il rilevamento. Sebbene efficaci, queste pipeline complesse risultavano lente e difficili da ottimizzare, poiché ogni singola componente doveva essere addestrata separatamente. YOLO ha rivoluzionato questo processo riformulando l'identificazione degli oggetti come un unico problema di regressione, passando direttamente dai pixel dell'immagine alle coordinate dei BB e alle probabilità di classe. Come suggerisce il nome, il sistema "guarda" l'immagine una sola volta (*you only look once*) per predire quali oggetti sono presenti e dove si trovano.

3.3.1 Yolov9

YOLOv9 [50] è un'architettura per il rilevamento di oggetti in tempo reale che affronta un problema fondamentale nel *deep learning*: la perdita di informazioni man mano che i dati attraversano reti molto profonde (fenomeno noto come *Information Bottleneck*). L'obiettivo principale non è stato solo progettare una rete più profonda o larga, ma garantire che le informazioni necessarie per mappare l'input (immagine) al target (oggetti) non vadano perse durante il processo di *feedforward*.

3.3.1.1 Principali Miglioramenti e Componenti

YOLOv9 introduce due innovazioni principali per risolvere i problemi di perdita di informazioni e inefficienza delle architetture esistenti:

- *Programmable Gradient Information* (PGI): Nelle reti profonde, l'errore calcolato alla fine (funzione di perdita) spesso non riesce a generare gradienti affidabili per aggiornare i pesi nei livelli iniziali, a causa della perdita di informazione. Il PGI risolve questo problema introducendo un ramo ausiliario durante l'addestramento:
  - funzionamento: genera gradienti affidabili attraverso un ramo reversibile ausiliario che preserva le informazioni complete dell'input;
  - vantaggio: a differenza della classica *Deep Supervision*, il PGI funziona anche su modelli leggeri senza causare accumulo di errori o costi aggiuntivi durante l'inferenza.
- *Generalized Efficient Layer Aggregation Network* (GELAN): è la nuova architettura della rete neurale, progettata per essere leggera, veloce e accurata:
  - generalizza il concetto di ELAN (usato in YOLOv7) permettendo l'uso di qualsiasi blocco computazionale.

– dimostra un utilizzo dei parametri superiore rispetto alle architetture basate su *Depth-wise Convolution* (considerate lo stato dell'arte per leggerezza), mantenendo un'accuratezza elevata.

3.3.1.2 Risultati

Le prestazioni di YOLOv9 sono state verificate sul dataset MS COCO [28], dimostrando superiorità rispetto alle versioni precedenti, tabella 3.2. In sintesi, YOLOv9 offre un miglior compromesso tra velocità, leggerezza e accuratezza, rendendolo ideale per applicazioni in tempo reale su dispositivi con risorse limitate

Tabella 3.2: Confronto tra YOLOv9 e le versioni precedenti[a]

Model	**#Param.** **(M)**	$\mathbf{AP}^{val}_{50:95}$ (%)	$\mathbf{AP}^{val}_{50}$ (%)	$\mathbf{AP}^{val}_{75}$ (%)	$\mathbf{AP}^{val}_{S}$ (%)	$\mathbf{AP}^{val}_{M}$ (%)	$\mathbf{AP}^{val}_{L}$ (%)
YOLOv5-X r7.0 [19]	86,7	50,7	68,9	–	–	–	–
YOLOv6-L v3.0 [25]	59,6	51,8	69,2	–	–	–	–
YOLOv7-X [49]	71,3	52,9	71,1	51,4	36,9	57,7	68,6
YOLOv8-X [20]	68,2	53,9	71,0	58,7	35,7	59,3	70,7
YOLOv9-E	57,3	55,6	72,8	60,6	40,2	61,0	71,4

[a] I dati sono stati reperiti da [50].

3.4 Zoe Depth

ZoeDepth [5] è un framework innovativo per la stima della profondità metrica da una singola immagine (MCIM). La principale novità è la capacità di colmare il divario tra due approcci tradizionalmente distinti:

- la stima della profondità relativa, che vanta eccellenti capacità di generalizzazione, ma manca di scala fisica;
- la stima della profondità metrica, che fornisce misure precise, ma tende a non generalizzare bene oltre il dominio di addestramento.

ZoeDepth integra questi due paradigmi, proponendo un approccio che unisce le capacità di generalizzazione dei modelli relativi con l'accuratezza metrica.

3.4.1 Architettura

L'architettura di ZoeDepth (mostrata in figura: 3.5) si costruisce su un modello di stima della profondità relativa pre-esistente (il *backbone*, ad esempio MiDaS [41]), trasformandolo in un previsore di profondità metrica attraverso l'aggiunta di una "testa" leggera chiamata *ZoeDepth head*. I componenti chiave includono:

- *Metric Bins Module*: è il cuore del sistema. Invece di predire direttamente valori continui, il modulo stima i centri dei contenitori (*bin*) metrici. Questi centri non

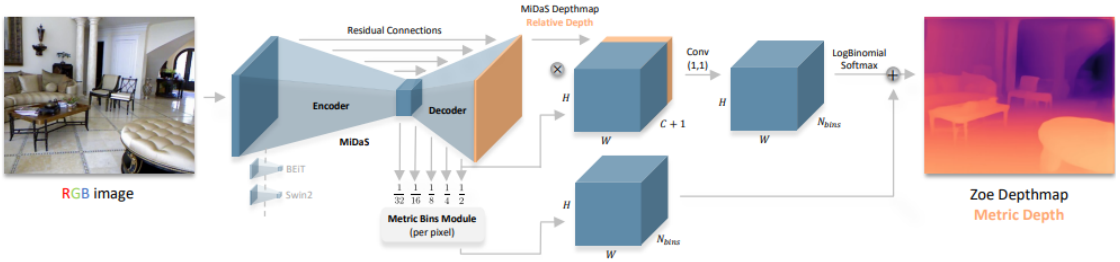


Figura 3.5: Architettura di ZoeDepth (Fonte: [5])

sono fissi, ma vengono adattati per ogni pixel attraverso livelli chiamati *attractor layers*, permettendo una quantizzazione della profondità fine e precisa.

- *Router (Latent Classifier)*: usato per gestire la stima su domini multipli (es. interni ed esterni) con un singolo modello. Questo router analizza le feature dell'immagine per determinare automaticamente se la scena è un interno o un esterno e instrada l'input verso la configurazione di contenitori metrici appropriati, senza supervisione esplicita durante l'inferenza.

3.4.2 Limitazioni

L'approccio presenta alcune limitazioni:

- Dipendenza dal *backbone*: essendo costruito su un modello di stima relativa, ZoeDepth ne eredita le debolezze. Se il modello relativo fallisce o produce artefatti (ad esempio bordi sfocati o errati), questi errori si propagano alla stima metrica finale.
- Errori del router: l'accuratezza in un modello multi-dominio dipende dal router. Se il classificatore latente sbaglia a identificare il dominio (es. classifica un esterno come interno), la scala metrica predetta sarà errata.
- Ambiguità di dominio: il sistema assume che ogni immagine appartenga nettamente a uno dei domini addestrati (interno o esterno). Ambienti ambigui, o che non rientrano chiaramente in specifiche categorie, potrebbero portare a stime di scala imprecise.

3.5 Unity

Per lo sviluppo e la validazione dell'ambiente virtuale e delle logiche di interazione, è stato scelto Unity [48]. Sebbene sia nato come motore grafico per videogiochi (*Game Engine*), Unity si è affermato come standard industriale per la creazione di *Digital Twins*, simulazioni robotiche e applicazioni AR e *Virtual Reality* (VR). Nel contesto di questo lavoro, Unity funge da *hub* di integrazione, gestendo la fisica della simulazione, la renderizzazione dell'ambiente indoor e l'orchestrazione dei moduli di intelligenza artificiale e interfaccia utente. Lo scripting delle logiche è affidato al linguaggio C#, che garantisce prestazioni elevate e un vasto ecosistema di librerie.

3.5.1 Sistema di Navigazione

La gestione del movimento dell'agente virtuale e il calcolo dei percorsi ottimali sono affidati al sistema *Unity Navigation* [47]. Questo modulo permette di trasformare la geometria statica della scena in una struttura dati navigabile, essenziale per guidare l'utente verso un *Point of Interest* (PoI). I componenti principali utilizzati sono:

- *NavMesh*: una rappresentazione astratta delle superfici calpestabili dell'ambiente. Unity pre-calcola questa mesh analizzando la geometria 3D, escludendo ostacoli e definendo le aree accessibili in base al raggio e all'altezza dell'agente.
- *NavMesh Agent*: un componente assegnato all'agente mobile (l'utente) che gli permette di muoversi sulla NavMesh. Utilizza l'algoritmo A* per il calcolo del percorso, trovando il più breve che eviti ostacoli statici e dinamici.
- *NavMesh Obstacle*: entità che modifica dinamicamente la NavMesh (es. porte che si chiudono o ostacoli imprevisti), costringendo l'agente a ricalcolare il percorso in tempo reale.

3.5.2 Text to Speech e Speech to Text

Per l'implementazione dell'interazione vocale all'interno di Unity, si è fatto ricorso a due plugin specifici che sfruttano le funzionalità native dei dispositivi mobili. Per quanto riguarda il TTS, si utilizza il plugin "Android native Text To Speech" [38], che consente l'esecuzione delle funzioni native di sintesi vocale dei sistemi Android. Parallelamente, per lo STT viene usato il plugin "Unity Speech To Text" [23], che permette l'esecuzione delle funzioni native di riconoscimento vocale sia per sistemi Android che iOS. In quest'ultimo caso, le lingue supportate corrispondono a quelle già installate sul dispositivo utilizzato.

Capitolo 4

Design di Sistema

Dopo aver definito, nel precedente capitolo, il contesto tecnologico di riferimento, il presente capitolo illustra nel dettaglio la progettazione e l'implementazione dell'intera architettura di sistema. L'obiettivo è fornire una panoramica esauriente sulle modalità con cui i vari moduli interagiscono tra loro per garantire un'esperienza di navigazione fluida e affidabile. Il testo approfondirà, in prima istanza, le dinamiche di calcolo lato server, focalizzandosi sui flussi di comunicazione a bassissima latenza tra i processi di intelligenza artificiale e il tracciamento spaziale. In seconda istanza, verrà descritta l'applicazione mobile, delineando le scelte di design relative al feedback multisensoriale e al calcolo dei percorsi ottimali.

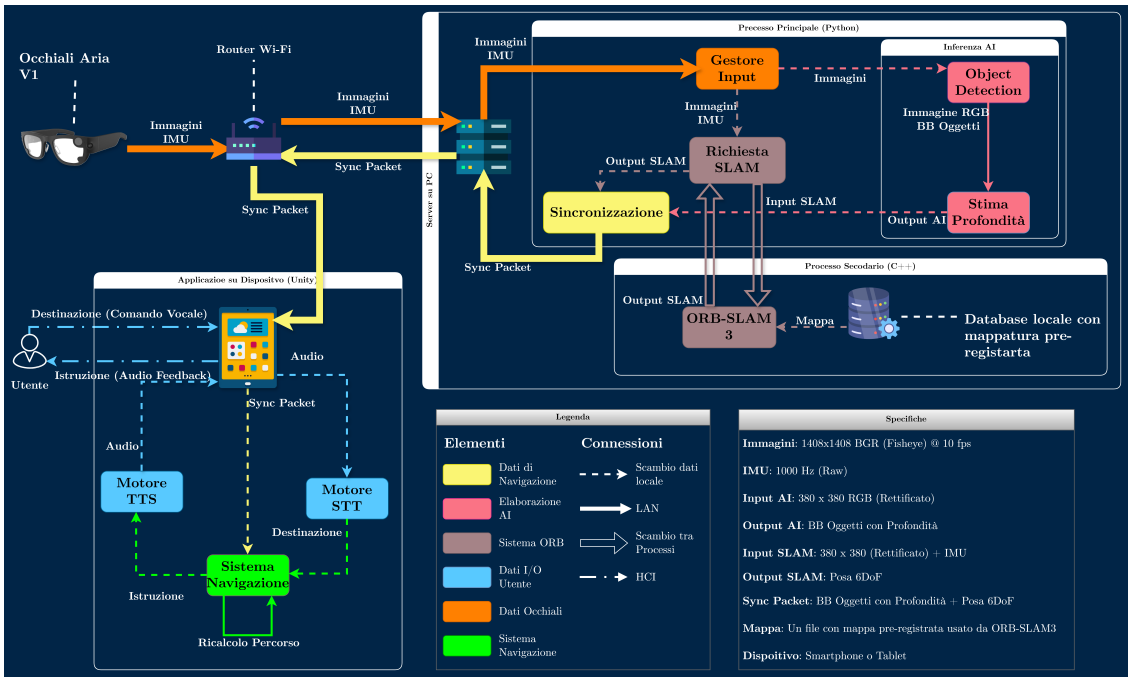


Figura 4.1: Architettura del progetto

L'architettura del progetto, mostrata in figura 4.1, è divisa in due parti principali:

- Server: elabora i dati ricevuti dagli occhiali (tramite *Local Area Network* (LAN)) e invia i risultati al dispositivo (smartphone o tablet) dell'utente.
- Applicazione: viene installata sul dispositivo dell'utente. L'applicazione ha il compito di controllare l'interazione con l'utente e di elaborare il percorso da effettuare.

Per fornire un'idea chiara del setup fisico in uno scenario di utilizzo reale, la figura 4.2 mostra il sistema completo indossato da un utente, comprensivo degli occhiali Project Aria Gen1 per l'acquisizione dei dati ambientali e del dispositivo mobile utilizzato per l'esecuzione dell'applicativo client e la riproduzione dei feedback audio.

Figura 4.2: Setup hardware completo indossato dall'utente: *Project Aria Gen1* per l'acquisizione visiva e spaziale, e *Tablet* per il feedback e l'elaborazione lato client

Di seguito si analizza nel dettaglio il funzionamento di ciascun componente.

4.1 Server

Il server, installato su un PC, è diviso in due processi concorrenti:

- Un processo Python, composto da quattro thread:
  1. Il primo, chiamato Gestore Input, riceve i dati dagli occhiali.
  2. Il secondo, di Inferenza IA, esegue le operazioni di riconoscimento degli oggetti e di stima della profondità.
  3. Il terzo, chiamato Richiesta SLAM, serve a dialogare con il processo secondario ORB-SLAM3.
  4. Il quarto, chiamato Sync, attende i dati elaborati dal secondo e terzo thread; quando i dati del thread Richiesta SLAM sono pronti, li invia, tramite LAN, al dispositivo dell'utente.

- L'altro è un processo C++, che esegue gli algoritmi di ORB-SLAM3, prendendo i dati ricevuti dal processo Python e restituendo i dati di posizionamento.

4.1.1 Gestore Input

Il *Gestore Input* è un thread del processo Python che si occupa di ricevere i dati, tramite connessione su rete locale, dagli occhiali. I dati ricevuti consistono nelle immagini riprese dalla fotocamera RGB (10 FPS e con una risoluzione di 1408×1408 px) e i dati dell'IMU di destra degli occhiali (frequenza 1000 Hz). Considerando che i dati dell'IMU arrivano molto più velocemente, vengono accumulati in una lista fino all'arrivo di un'immagine. I dati IMU ricevuti sono grezzi, perciò vengono rettificati con le API di Project Aria. Le immagini, invece, nel momento in cui arrivano, vengono inserite nelle code dei thread Richiesta SLAM (insieme ai dati IMU collezionati) e Inferenza IA.

4.1.2 Richiesta SLAM

Il thread di Richiesta SLAM prende i dati inseriti sulla coda dal Gestore Input e li invia al processo in C++ sfruttando una memoria condivisa (gestita in Python tramite il modulo `shared_memory`). Un riassunto grafico può essere visto in figura 4.3. Per garantire un trasferimento a bassissima latenza ed evitare colli di bottiglia, la *Inter-Process Communication* (IPC) implementa un *ring buffer* da 20 slot (circa 1 MB ciascuno, per ospitare *header*, dati IMU e immagine). La sincronizzazione tra i processi in fase di lettura e scrittura è garantita da semafori POSIX (interfacciati tramite la libreria `posix_ipc`). Il processo Python usa il semaforo `sem_full` per indicare se la coda è piena (in tal caso blocca l'invio), mentre sfrutta il semaforo `sem_empty` per segnalare la presenza di elementi all'interno del buffer.

Prima di inviare i dati, il thread esegue le seguenti elaborazioni:

- Rettificazione: si utilizzano le API di Project Aria per eliminare le distorsioni dovute alle lenti sferiche. Per questo motivo, le immagini sono ridotte a una dimensione di 380×380 px.

- Conversione colori: le immagini ricevute sono in *Blue-Green-Red* (BGR); quindi, vengono convertite in scala di grigi per ridurre il carico computazionale.

I dati elaborati vengono serializzati in array di byte (tramite il modulo `struct`) e scritti nella posizione corrente del buffer. Una volta inviati, il thread aspetta il risultato su una seconda area di memoria condivisa dedicata all'output, mettendosi in attesa del semaforo di completamento (`sem_out`) dal C++. Il risultato ottenuto, ovvero una struttura contenente i tempi di elaborazione e la posa a 6 *Degrees of Freedom* (DoF) (posizione e rotazione), viene poi inserito in un buffer di sincronizzazione globale.

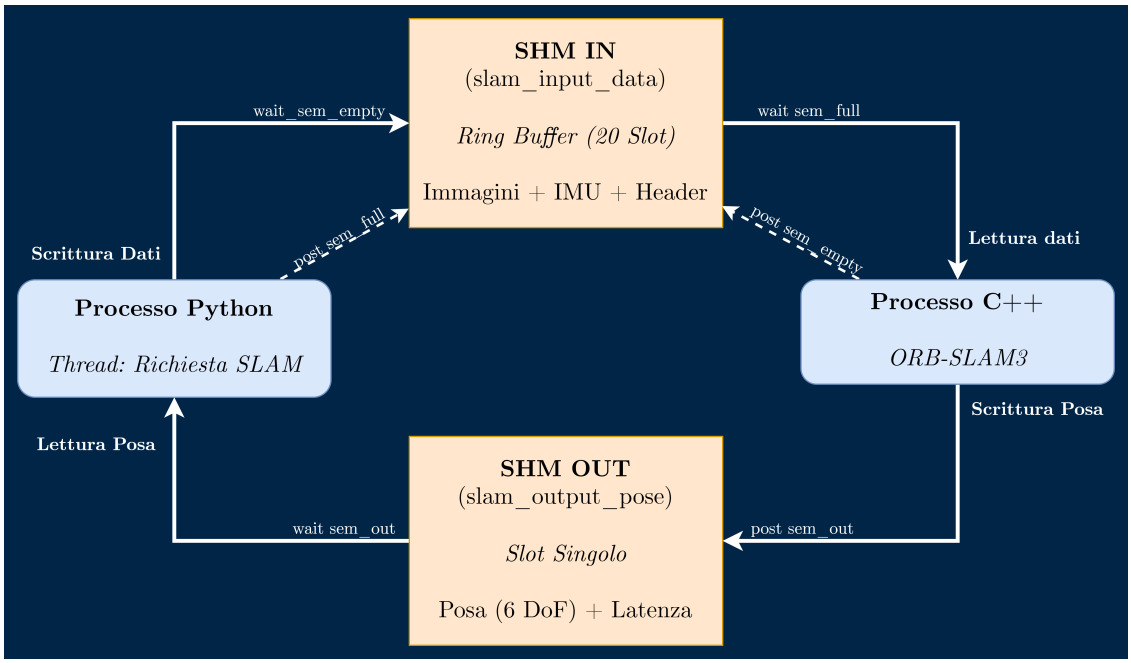


Figura 4.3: Schema architetturale della IPC. I processi Python e C++ operano in modo circolare sulle aree di memoria condivisa, sincronizzando le operazioni tramite semafori POSIX per garantire bassissima latenza ed evitare corruzioni di memoria

4.1.3 ORB-SLAM3

Il processo in C++ esegue gli algoritmi ORB-SLAM3 (si veda la sezione 3.2). Esso si interfaccia con le aree di memoria condivisa create dal processo Python, mappandole direttamente nel proprio spazio di indirizzamento tramite le chiamate di sistema `shm_open` e `mmap`.

Sincronizzandosi tramite i semafori POSIX (`sem_wait`), il processo estrae dallo slot corrente del *ring buffer* il timestamp, le misurazioni dell'IMU e i pixel dell'immagine, ricostruendo le strutture dati in memoria tramite il *casting* diretto dei puntatori, per azzerare i tempi di copia. Include anche dei controlli di sicurezza per scartare frame con misurazioni corrotte (valori *Not a Number* (NaN) o infiniti) o ritardi eccessivi, preservando la stabilità del sistema.

Una volta calcolata la posa grazie a una mappa registrata in precedenza, i risultati (insieme alle metriche di latenza) vengono scritti nella memoria condivisa di output e il processo Python viene notificato sbloccando il relativo semaforo (`sem_post`).

Creazione della Mappa

Per la creazione della mappa, è stata effettuata una registrazione di tutte le aule R usando la seguente configurazione per i parametri degli occhiali:

- Fotocamera: uso della fotocamera RGB con risoluzione 1408 × 1408 px e 20 FPS.

- IMU: uso dell'IMU destro, che ha una frequenza di 1000 Hz.
- Altri: gli altri sensori sono stati spenti.

Un esempio di mappatura è mostrato nella figura 4.4.

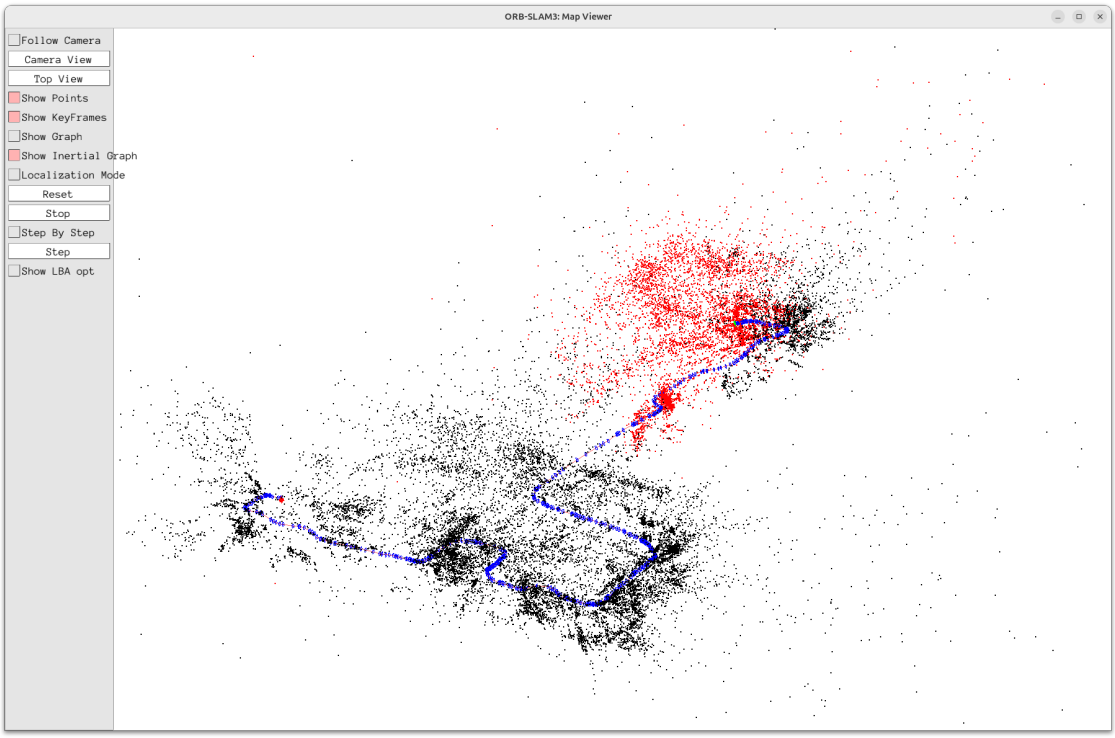


Figura 4.4: Schermata del processo di mappatura ottenuta da finestra generata dal *Viewer* di ORB-SLAM3

4.1.4 Inferenza IA

Il thread di Inferenza IA prende le immagini inserite nella coda dal *Gestore Input* e le elabora eseguendo queste operazioni:

- Rettificazione: utilizzando le API di Project Aria per eliminare le distorsioni dovute alle lenti sferiche. Per questo motivo, le immagini sono ridotte a una dimensione di 380×380 px.
- Conversione colori: le immagini ricevute sono in BGR; quindi, vengono convertite in RGB.

Il thread è costituito da due blocchi che agiscono in sequenza:

1. Identificazione ostacoli: prende l'immagine in RGB ed esegue l'inferenza tramite il modello YOLOv9, nella sua versione di tracciamento degli oggetti (`model.track()`), e ottiene gli oggetti presenti nella scena.

2. Stima della profondità: utilizza il modello ZoeDepth per creare una mappa di profondità dall'immagine ricevuta.

Infine, i dati sui BB e l'ID di tracciamento vengono usati per estrarre la profondità media dell'oggetto. Vengono ignorati gli oggetti che si trovano a una distanza media superiore a 5 m ed inferiore a 0,8 m. Gli oggetti validi vengono inseriti in una variabile globale.

4.1.5 Sync

Il thread di *Sync* funge da punto di incontro tra i flussi di dati asincroni generati dallo SLAM e dall'inferenza IA, occupandosi della loro pacchettizzazione e trasmissione verso l'applicazione Unity installata sul dispositivo dell'utente.

Per la comunicazione su rete locale (LAN), mostrata in figura 4.5, il sistema implementa un'architettura Server-Client basata sul protocollo TCP/IP, dove il processo Python agisce da server in ascolto e l'applicazione Unity da client. Per garantire la minima latenza possibile in uno scenario di *real-time streaming*, sono state adottate specifiche ottimizzazioni a livello di rete e di gestione della memoria:

- Sincronizzazione asincrona: poiché i calcoli dello SLAM e dell'IA operano a frequenze diverse, il thread estrae sempre il frame SLAM completato più recente dal buffer, unendolo agli ultimi oggetti validi rilevati dalla rete neurale (letti tramite variabili globali protette da *mutex*). I frame antecedenti a quello selezionato vengono scartati per evitare di trasmettere dati obsoleti al client.

- Controllo di congestione (*Frame Dropping*): lato Python, prima di ogni invio viene verificata la disponibilità del socket di rete (sfruttando la chiamata di sistema `select`). Se la rete Wi-Fi risulta temporaneamente congestionata, il frame viene saltato per non bloccare il ciclo di esecuzione del server, prevenendo un effetto a cascata sui tempi dell'IA.

- Disabilitazione dell'algoritmo di Nagle: Lato Unity, il client disabilita esplicitamente l'algoritmo di Nagle (`NoDelay = true`) sulla connessione TCP. Questo forza l'invio e la ricezione immediata dei pacchetti, barattando l'efficienza della banda con una latenza drasticamente inferiore.

- Framing dei dati (*Length-Prefix*): Per risolvere il problema della frammentazione tipico delle trasmissioni TCP, i dati vengono serializzati in formato *JavaScript Object Notation* (JSON) e preceduti da un *header* di 4 byte (codificato in *little endian*) che ne indica la dimensione esatta. Il client Unity legge rigorosamente prima l'*header* per determinare la dimensione del *payload*, e solo successivamente legge il corpo del messaggio, garantendo un *parsing* sicuro.

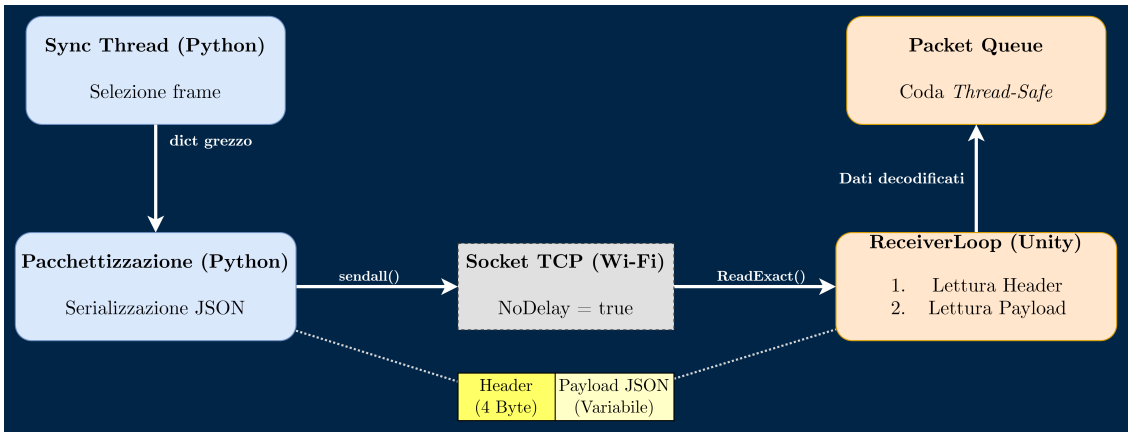


Figura 4.5: Flusso della pipeline di rete tra il server Python e il client Unity. L'uso del prefisso di lunghezza nel pacchetto TCP impedisce la frammentazione del flusso JSON, mentre il meccanismo di controllo di congestione nel buffer sorgente previene l'accumulo di latenza

Una volta decodificati correttamente lato client, i dati (contenenti la posa, l'elenco degli oggetti rilevati e le metriche di latenza) vengono inseriti in una coda *thread-safe*, da cui verranno consumati dal thread principale di Unity.

4.2 Applicazione

L'applicazione, sviluppata in C# per Unity, è strutturata in due scene principali. La prima è dedicata alla configurazione iniziale, ovvero all'inizializzazione dei sistemi TTS e STT, alla connessione con il server remoto e alla selezione della destinazione. La seconda scena, invece, racchiude l'intera logica del sistema di navigazione.

Nella prima scena, il sistema fornisce all'utente un feedback vocale (tramite TTS) riguardo allo stato della connessione al server. A connessione stabilita, l'applicazione invita l'utente a pronunciare la propria destinazione; qualora quest'ultimo non conosca le opzioni valide, può pronunciare il comando *aiuto* per ascoltare la lista delle destinazioni disponibili. Una volta riconosciuto e validato l'input vocale tramite STT, il sistema procede al caricamento della scena successiva. In questa scena il comando STT viene attivato cliccando sullo schermo e si disattiva automaticamente quando non rileva più input audio.

Nella schermata iniziale, l'interfaccia grafica è stata mantenuta volutamente essenziale e minimalista, limitandosi alla presenza di un unico elemento testuale (*label*). Questa specifica scelta progettuale è mirata a semplificare l'esperienza d'uso e a consentire all'utente, in caso di difficoltà o malfunzionamenti, di mostrare facilmente lo schermo a terze persone per ricevere immediata assistenza. La seconda scena, mostrata nella figura 4.6, ospita l'ambiente virtuale contenente la mappa delle aule. Inizialmente (figura 4.6a), l'applicativo rimane in attesa che il server calcoli e stabilisca la posizione dell'utente all'interno della mappa pre-caricata. Conclusa la localizzazione, viene generato il

percorso ottimale per raggiungere la meta (figura 4.6b). Durante il tragitto, l'applicazione fornisce istruzioni direzionali (es. "gira a destra", "continua dritto") e aggiornamenti sullo stato dell'ambiente circostante (es. livello di affollamento). Se l'utente si avvicina eccessivamente a un ostacolo, il sistema gli intima di fermarsi e attende il ricalcolo della rotta.

Per gestire questa dinamica in modo efficiente, in Unity gli ostacoli mobili (figura 4.6c) sono dotati del componente *NavMesh Obstacle* che si attiva esclusivamente se il BB dell'utente interseca quello dell'oggetto, innescando così il ricalcolo del pathfinding. Al contrario, per gli oggetti statici (come divani o tavoli), tale componente è mantenuto costantemente attivo. Questa logica è stata implementata per prevenire un sovraccarico cognitivo per l'utente: in situazioni di forte affollamento, infatti, il continuo movimento delle persone forzerebbe il sistema a ricalcolare il percorso costantemente, generando un flusso di istruzioni vocali troppo frequente e confusionario. Oltre al feedback audio, l'interfaccia grafica include un'etichetta testuale (*label*) che riporta l'ultima istruzione generata, mentre a livello visivo il percorso calcolato è evidenziato in rosso e quello già effettuato in verde.

All'interno di questa scena sono stati inoltre implementati dei controlli gestuali per l'interazione manuale con l'interfaccia a schermo. L'utente, o un eventuale assistente, può traslare l'inquadratura trascinando due dita sullo schermo (*pan*), ruotarla utilizzandone tre, o regolarne lo zoom avvicinando o allontanando due dita (*pinch-to-zoom*). Qualora il sistema non riceva alcun input tattile per più di tre secondi, la telecamera virtuale viene automaticamente ripristinata alla sua posizione di default. Quest'ultima è collocata perpendicolarmente al di sopra dell'utente, offrendo così una visuale dall'alto (o zenitale) che facilita la comprensione globale della mappa e del percorso da seguire.

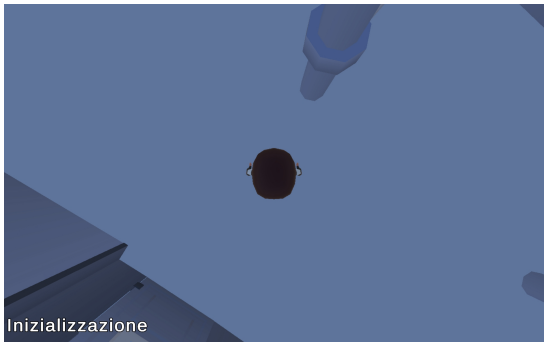
Infine, per prevenire conflitti tra l'utilizzo dei controlli della telecamera e l'attivazione del riconoscimento vocale, in questa specifica scena l'abilitazione del modulo STT non è immediata al tocco, ma richiede un'interazione intenzionale tramite una pressione prolungata sullo schermo (*long press*) della durata di circa tre secondi. Tale modalità operativa viene comunicata all'utente durante l'inizializzazione della scena tramite TTS.

4.2.1 Interazione vocale: TTS e STT

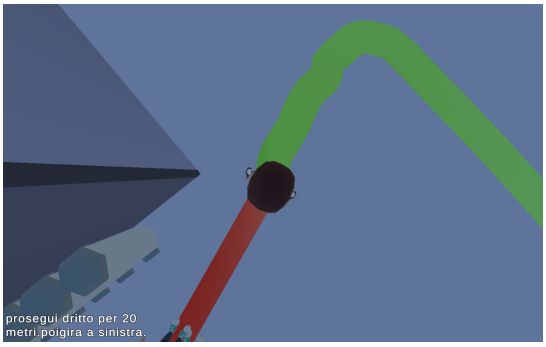
I moduli di sintesi e riconoscimento vocale, le cui basi tecnologiche sono state esposte nella sezione 3.5.2, gestiscono l'intera comunicazione audio tra l'utente e il sistema.

Il motore di TTS ha il compito di fornire un riscontro uditivo continuo, vocalizzando sia le istruzioni direzionali per la navigazione, sia gli aggiornamenti relativi allo stato dell'applicazione (come, ad esempio, l'avvenuta connessione al server).

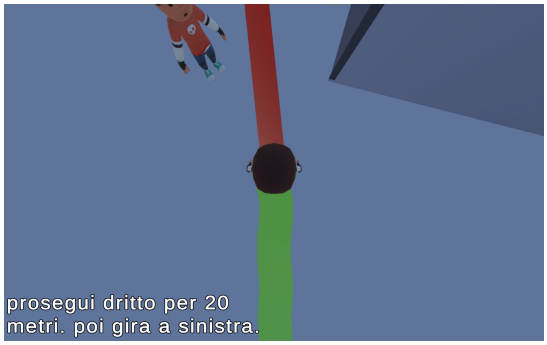
Sul fronte dell'input, il sistema di STT si attiva mediante un tocco sullo schermo e interrompe automaticamente la registrazione non appena l'utente smette di parlare. Questa funzionalità è cruciale fin dal primo avvio dell'applicazione: nella scena iniziale, infatti, all'utente viene richiesto di pronunciare la destinazione desiderata, che il sistema riconoscerà e imposterà come *target* per il calcolo del percorso.



(a) Fase di inizializzazione e attesa del posizionamento



(b) Generazione del percorso a posizionamento avvenuto



(c) Immagine raffigurante l'aggiunta di persone rilevate durante la navigazione

Figura 4.6: Interfaccia utente della scena di navigazione: (a) stato iniziale, (b) visualizzazione del percorso calcolato verso la destinazione e (c) visualizzazione di persone rilevate

4.2.2 Sistema di navigazione

La scena dedicata al sistema di navigazione si fonda su due elementi architetturali principali all'interno del motore grafico:

- La mappa dell'edificio: alle cui superfici calpestabili (pavimenti) è stata applicata una *NavMesh* per abilitare il calcolo dei percorsi.
- Il *player* (rappresentante l'utente): configurato in Unity come *NavMesh Agent*.

Ogni qualvolta l'applicativo riceve nuovi dati di posizionamento tramite rete LAN, esso aggiorna le coordinate spaziali del *player* e ricalcola dinamicamente il tragitto. Successivamente, il sistema elabora l'istruzione direzionale pertinente (es. "vai dritto", "gira a destra") e valuta se riprodurla vocalmente tramite il modulo TTS.

Per evitare di fornire informazioni ridondanti, la sintesi vocale viene innescata ogni 10 secondi riproducendo l'istruzione. Nei seguenti casi, invece, l'istruzione viene comunicata immediatamente se:

- È diversa dall'ultima comunicata (cambio di direzione).
- Si tratta di un'istruzione di stato prioritaria e urgente (es. "Arrivato a destinazione", "Posizionamento in corso", "Prendere l'ascensore", "Inizializzazione").
- Deve essere eseguita nei prossimi 2 metri.

Questa specifica logica di filtraggio dei messaggi è stata implementata per minimizzare il carico cognitivo dell'utente, fornendogli esclusivamente le indicazioni strettamente necessarie al momento opportuno ed evitando di disorientarlo con un flusso audio ininterrotto.

Per quanto riguarda gli oggetti identificati ricevuti dal server, vengono inseriti nella scena come *NavMesh Obstacle* affinché l'algoritmo li consideri ostacoli da evitare. Se l'oggetto ricevuto è una persona, allora controlla se esiste già (sfruttando l'ID di tracciamento): se esiste, allora lo sposta nella nuova posizione, altrimenti lo crea.

Capitolo 5

Test e risultati

A valle della rassegna tecnologica affrontata nel capitolo 3 e della descrizione architetturale proposta nel capitolo 4, il presente capitolo illustra e analizza i risultati sperimentali ottenuti. I dati presentati sono il frutto sia delle misurazioni tecniche eseguite in fase di progettazione, sia delle sessioni di validazione dell'usabilità condotte con gli utenti finali.

5.1 Valutazione prestazionale del sistema

In questa sezione vengono discussi i risultati sulle prestazioni ed affidabilità del sistema misurate in fase di progettazione.

5.1.1 Metriche tecniche e prestazionali

Al fine di fornire una valutazione quantitativa delle prestazioni del sistema, sono state adottate le seguenti metriche:

- Accuratezza: indica quanto il sistema consenta di avvicinarsi all'obiettivo desiderato. Nello specifico, misura la distanza finale tra l'utente e il target al termine del percorso. È espressa in metri.
- Precisione: valuta lo scostamento del sistema dal dato reale, calcolando la distanza tra la posizione stimata del target e l'effettiva collocazione spaziale dell'oggetto. È espressa in metri.
- Tempo di processamento: misura l'intervallo temporale che intercorre tra la ricezione di un frame da parte del server e il suo arrivo all'applicativo Unity. È espresso in secondi.

5.1.2 Setup sperimentale e scenari di test

Per la conduzione degli esperimenti è stato impiegato un tablet Samsung Galaxy Tab A9+ con processore Snapdragon695 e memoria RAM da 8 GB, sul quale è stata installata l'applicazione client sviluppata in Unity. Il server, invece, è stato installato su un PC

portatile Lenovo dotato di GPU NVIDIA GeForce RTX 4080, processore Intel Core i9-13900HX, memoria RAM da 32 GB e sistema operativo Ubuntu 24.04.4 LTS.

Al fine di valutare la robustezza e le prestazioni del sistema al variare delle condizioni di connessione, l'infrastruttura di rete è stata testata impiegando diversi dispositivi di router:

- **TP-Link Archer AX3000**: il supporto di questo dispositivo allo standard Wi-Fi6 si è rivelato un requisito fondamentale per garantire una comunicazione wireless rapida è in grado di gestire l'elevata mole di dati generata dagli *smart glasses* Project Aria Gen1 senza colli di bottiglia.

- **TIM HUB+**: impiegato per valutare l'affidabilità del sistema su un apparato di fascia *consumer* standard, tipicamente fornito in dotazione dagli operatori telefonici nelle reti domestiche. Tale router è dotato di Wi-Fi6

- **TP-Link AX1500**: questo dispositivo, che supporta il Wi-Fi6, è un router portatile il cui scopo è quello di superare la limitazione negli spostamenti in un'area più larga (es. tra piano terra e primo piano).

Nella tabella 5.1 vengono illustrati i risultati dei diversi test eseguiti nel corso dello sviluppo. Ogni prova è composta da due o tre obiettivi intermedi, ovvero punti di interesse (*waypoint*) spaziali raggiunti durante il test.

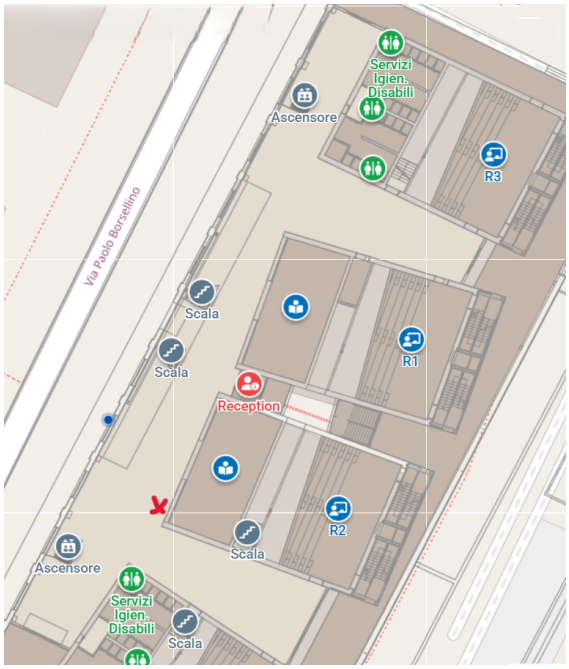


Figura 5.1: Punto iniziale (indicato con un X rossa) per i test

Tabella 5.1: Caratteristiche dei test eseguiti

Numero	Durata (# frame)	Router Usato / Connessione Server	Caratteristiche[a]
1	2596	TP-Link Archer AX3000 / Ethernet	Questo test consiste nell'andare dal punto iniziale[c] all'**uscita**, poi alla **Portineria** ed infine alla **Sala Studio 2**
2	2778	TP-Link Archer AX3000 / Ethernet	Questo test prevede di andare dal punto iniziale all'**aula R2**, poi alla **Portineria** ed infine al **Bagno**
3	1873	TIM HUB+ / Ethernet	Questo test prevede di andare dal punto iniziale all'**aula R2**, poi alla **Bagno** ed infine all'**Aula Studio 1**
4	1869	TP-Link Archer AX3000 / Ethernet	Questo test prevede di andare dal punto iniziale all'**aula R2**, poi al **Bagno** ed infine alla **Portineria**
5	1739	TP-Link AX1500 / Ethernet	Questo test prevede di andare dal punto iniziale al **Bagno**, poi all'**aula R2** ed infine all'**Aula Studio 2**
6	3254	TP-Link AX1500 / Wi-Fi[b]	Questo test prevede di andare dal punto iniziale all'**aula R2**, poi alla **Portineria** ed infine al **Bagno**
7	1693	TP-Link AX1500 / Ethernet	Questo test prevede di andare dal punto iniziale all'**aula R2**, poi al **Bagno** ed infine all'**Aula Studio 2**

[a] In grassetto vengono evidenziati gli obiettivi.
[b] In questa configurazione, il router viene portato dall'utente.
[c] Il punto iniziale è mostrato nella figura 5.1

5.1.3 Analisi spaziale: accuratezza e precisione

In questa sezione vengono presentati e discussi i dati su precisione ed accuratezza misurati durante i test. I dati raccolti durante le misurazioni sono riportati nella tabella 5.2.

Tabella 5.2: Risultati di accuratezza e precisione

Test	Accuratezza (m)			Precisione (m)		
	Obiettivo 1	Obiettivo 2	Obiettivo 3	Obiettivo 1	Obiettivo 2	Obiettivo 3
1	1,0	0,5	0,3	0,2	0,2	0,2
2	0,9	0,9	0,5	0,2	0,2	0,2
3	0,9	0,5	1,2	0,2	0,2	0,2
4	0,8	0,4	1,0	0,2	0,2	0,2
5	0,6	0,4	0,5	0,2	0,2	0,2
6	0,5	0,7	0,4	0,2	0,2	0,2
7	0,5	0,3	0,6	0,2	0,2	0,2

Come si evince dalla tabella, la precisione del sistema risulta molto elevata, garantendo un comportamento ripetibile che permette di guidare l'utente a destinazione in modo affidabile. Per quanto riguarda l'accuratezza, sebbene gli errori spaziali siano di entità

maggiore rispetto a quelli misurati per la precisione, questi risultano comunque adeguati dal punto di vista dell'usabilità del sistema. Nel contesto specifico di navigazione, infatti, un errore nell'ordine di un metro in prossimità dell'obiettivo finale (come, ad esempio, una porta) è considerato accettabile per la corretta conclusione del percorso.

La variabilità riscontrata nei valori di accuratezza è imputabile principalmente a due cause:

- Conformazione architettonica delle aule R: Spesso le porte d'ingresso sono collocate in posizione rientrata rispetto al filo della parete. Al fine di garantire la sicurezza dell'utente ed evitare potenziali collisioni con i muri durante le manovre di svolta, si è scelto deliberatamente di rimuovere tale distanza dalla mappatura. Questa scelta progettuale introduce un offset sistematico che influisce sui valori di accuratezza assoluta.
- Limiti di tracking in ambienti poveri di feature: Le prestazioni del sistema ORB-SLAM3 tendono a degradare quando ci si avvicina ad elementi architettonici privi di texture o dettagli visivi (come un muro completamente piatto). La mancanza di un numero sufficiente di punti chiave (*keypoints*) rende più complessa la stima esatta della posa, aumentando il margine di errore nella stima della posizione finale.

5.1.4 Analisi temporale: latenza ed elaborazione

In questa sezione vengono presentati e discussi i tempi di latenza e di processamento richiesti dall'architettura di sistema per l'elaborazione dei singoli frame e per il corretto funzionamento in tempo reale. I dati raccolti durante le misurazioni sono riportati nella tabella 5.3.

Tabella 5.3: Tempo medio di latenza e processamento necessario per l'elaborazione di un frame

Test	**Tempo Trasferimento Python a C++ (ms)**	**Tempo processamento ORB (ms)**	**Tempo Trasferimento C++ a Python (ms)**	**Tempo processamento IA (ms)**	**Tempo Trasferimento Python a Unity (ms)**[a]
1	8,064	0,019	0,202	84,742	3290,463
2	4,505	0,026	0,250	83,854	4689,814
3	4,759	0,029	0,169	85,392	5018,425
4	189,850	0,033	0,199	86,323	5807,417
5	1,785	0,023	0,351	85,746	543,177
6	4,296	0,025	1,043	87,892	97,711
7	1,176	0,024	0,350	81,807	1953,573

[a] Questo dato è misurato sul dispostivo usato, considerando, quindi, i soli frame che lo raggiungono.

I dati raccolti nella tabella evidenziano in modo chiaro le prestazioni dei singoli moduli dell'architettura, permettendo di distinguere tra i tempi di elaborazione locale e i tempi di trasmissione verso il dispositivo utente. L'analisi rivela una netta dicotomia tra l'efficienza dei processi computazionali e le latenze di rete.

Nello specifico, è possibile trarre le seguenti osservazioni:

- Stabilità ed efficienza dell'elaborazione locale: I moduli dedicati al processamento puro mostrano tempi altamente performanti e costanti. L'elaborazione dell'IA si attesta su una media stabile di circa 85 ms per frame, un valore compatibile con i requisiti di un sistema operante in tempo reale. Analogamente, il processamento ORB-SLAM3 e il trasferimento dati di ritorno da C++ a Python risultano estremamente rapidi, nell'ordine delle frazioni di millisecondo (o al massimo di circa 1 ms, come nel Test 6).

- Variabilità nel trasferimento verso C++: Il tempo di trasferimento da Python a C++ si mantiene generalmente basso (intorno o sotto gli 8 ms), ad eccezione di un picco isolato registrato durante il Test 4 (circa 190 ms). Tale anomalia transitoria è verosimilmente imputabile a fluttuazioni nelle risorse di sistema (ad esempio, scheduling del sistema operativo o processi di garbage collection).

- Latenza di trasmissione verso l'utente finale: Il trasferimento dei dati da Python a Unity rappresenta il collo di bottiglia dell'intera architettura e mostra una fortissima variabilità. Come evidenziato dai dati, i tempi spaziano da minimi eccellenti al di sotto del decimo di secondo (circa 98 ms nel Test 6 e 543 ms nel Test 5), fino a superare i 5.8 secondi nel Test 4. Questo ritardo altalenante è dovuto alla latenza di rete e alle capacità di ricezione del dispositivo target. Inoltre, in alcune zone delle aule si è notato un degrado del segnale, probabilmente dovuto a problemi di interferenza tra le connessioni Wi-Fi dei vari dispositivi presenti. Ciò infatti avviene principalmente nelle aree limitrofe alle aule durante le ore di lezione, spiegando così gli sbalzi drastici registrati tra un test e l'altro. Per ovviare a questo problema, dopo il Test 4 sono state modificate le impostazioni del router utilizzato: è stata disabilitata completamente la banda a 2,4 GHz ed è stato impostato un canale di trasmissione fisso per la rete a 5 GHz. Questo intervento giustifica il netto e decisivo miglioramento delle prestazioni registrato nei test successivi.

I grafici 5.2 e 5.3 espandono l'analisi del Test 7, permettendo di isolare il contributo di ciascun sottomodulo alla latenza complessiva e di valutare le prestazioni della trasmissione di rete.

Dall'analisi dei dati emergono le seguenti evidenze sperimentali:

- Carico computazionale di base: Il grafico dei "Tempi di Processamento IA" (in alto a destra, figura 5.2) conferma che l'inferenza delle reti neurali costituisce il carico di lavoro fisiologico del sistema. Ad eccezione di un prevedibile picco iniziale di circa 300 ms (dovuto al *warm-up* dei modelli in fase di avvio), i tempi si assestano stabilmente tra gli 80 e i 100 ms. Disaggregando questo dato, si nota come la stima della profondità (modello ZoeDepth) richieda la quasi totalità delle risorse (circa 65-75 ms), mentre il rilevamento degli oggetti (YOLO) e la pre-elaborazione incidono in modo marginale (complessivamente 10-20 ms).

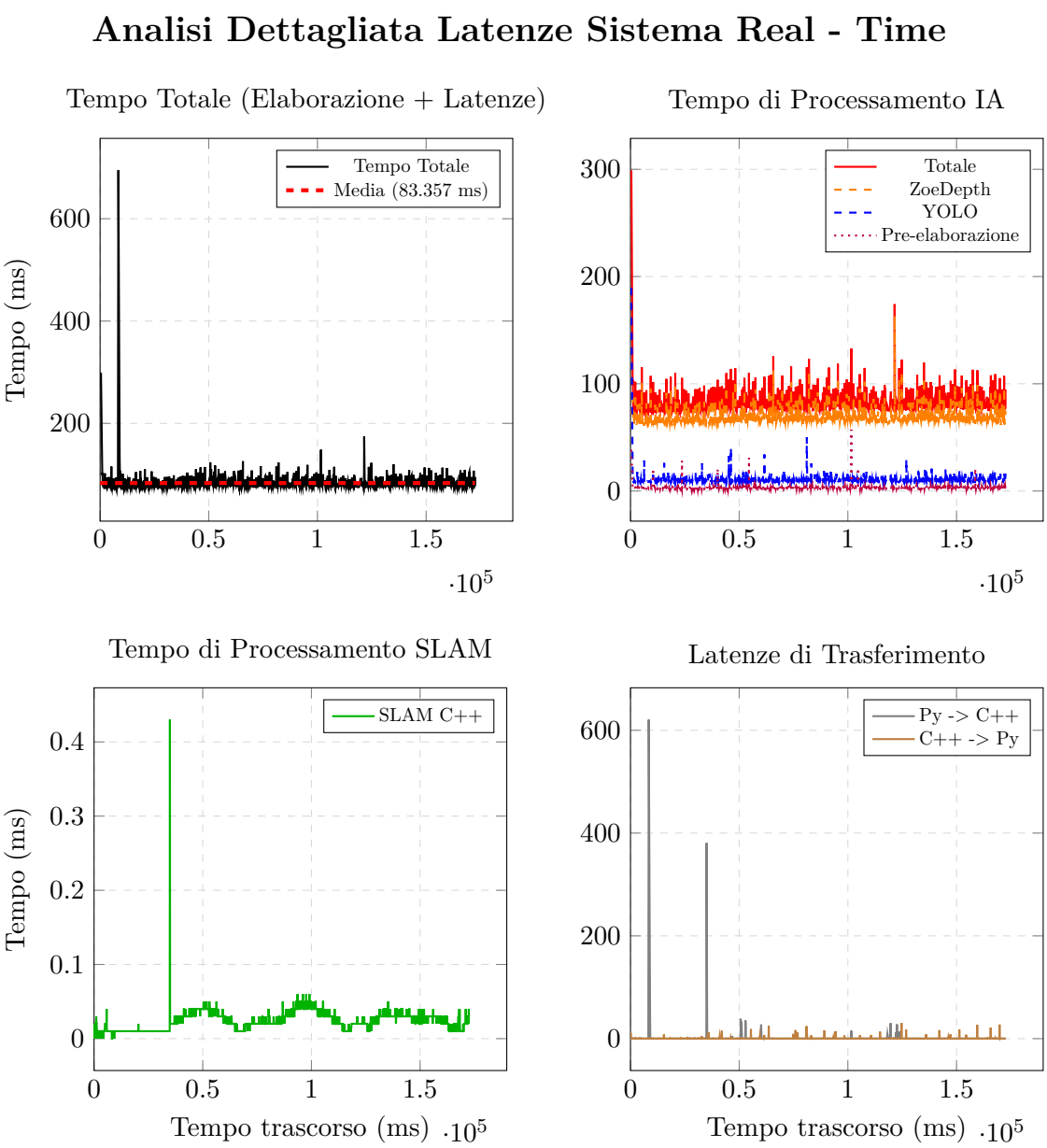


Figura 5.2: Analisi dettagliata delle latenze del sistema real-time per il Test 7, suddivise per componenti (Tempo Totale, Processamento IA, SLAM e Trasferimento in Shared Memory).

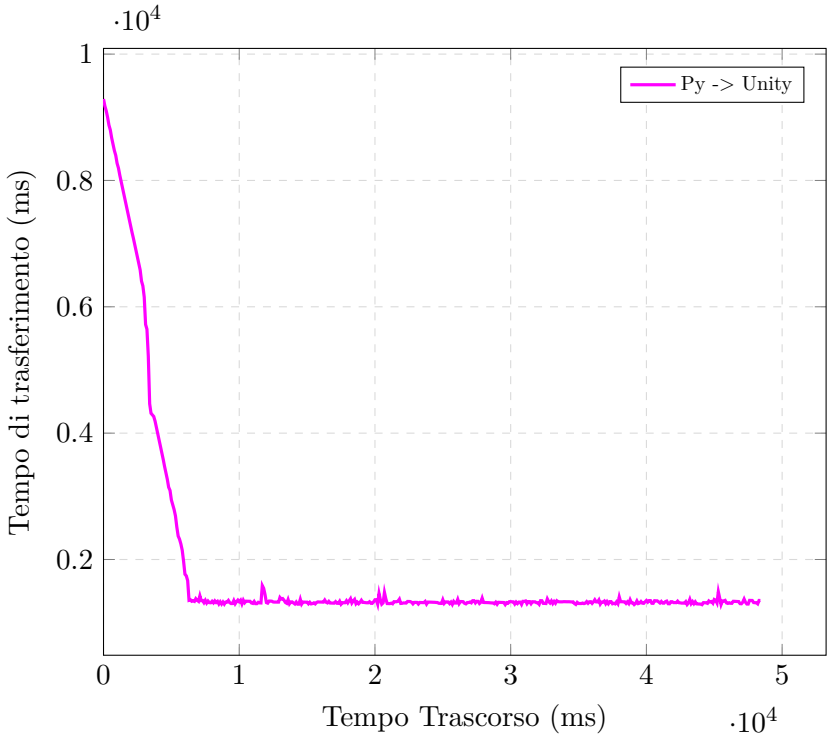


Figura 5.3: Grafico del tempo di trasferimento tra il Server in Python ed il Client Unity installato sul Device

- Efficienza del modulo SLAM: Il grafico in basso a sinistra (figura 5.2) evidenzia le altissime prestazioni dell'implementazione C++ per lo SLAM. I tempi di elaborazione puri si mantengono costantemente al di sotto di 0,05 ms. Anche in presenza di rarissime fluttuazioni (come il picco isolato a 0,43 ms registrato intorno al secondo 35), i tempi restano ampiamente al di sotto del millisecondo, confermando l'assoluta idoneità di ORB-SLAM3 per applicazioni real-time senza pesare sul bilancio temporale totale.

- Anomalie e colli di bottiglia IPC: L'aspetto più critico dell'elaborazione locale emerge dal confronto tra la "Latenza Totale" e le "Latenze di Trasferimento" (figura 5.2). Si osservano alcuni rari ma significativi picchi anomali di latenza totale, il più alto dei quali sfiora i 700 ms. Questi picchi non sono causati dagli algoritmi di visione artificiale, bensì dal trasferimento dei dati tramite Shared Memory (Python a C++). Questi rallentamenti improvvisi sono tipici di fenomeni di blocco a livello di sistema operativo (es. *lock contention* sui mutex o interventi del *garbage collector* di Python).

- Latenza di rete e stabilizzazione del buffer: Il grafico 5.3 mostra il tempo di trasferimento reale dal Server (Python) al dispositivo utente (Unity). Emerge un chiaro transitorio iniziale: nei primi secondi di esecuzione, la latenza di trasmissione è

estremamente elevata (oltre 9000 ms), per poi decadere linearmente fino a stabilizzarsi, dopo circa 6 secondi, su un valore di regime compreso tra i 1200 ms e i 1400 ms. Questo andamento è verosimilmente imputabile al tempo necessario per stabilire la connessione e smaltire i pacchetti accumulati nei buffer di rete (*queueing delay*) durante la fase di avvio del sistema.

5.1.5 Analisi della stabilità e della robustezza

Oltre alle metriche quantitative sopra indicate, in questa sezione vengono discussi ulteriori dati raccolti durante i test.

5.1.5.1 Tasso di omissione dell'elaborazione IA

Poiché il server elabora i dati di posizionamento tramite il sistema ORB-SLAM3 molto più rapidamente rispetto all'esecuzione delle reti neurali, in alcuni casi invia al client frame con informazioni sugli ostacoli obsolete. Nella tabella 5.4 sono riportati i dati ottenuti durante le misurazioni.

Tabella 5.4: Numero e percentuale di frame inviati con dati sugli ostacoli rilevati dall'IA obsoleti

Test	**Percentuale Frame con IA obsoleta**	**Numero Frame con IA obsoleta**	**Numero Frame totali**
1	10,32%	268	2596
2	6,34%	176	2778
3	5,39%	101	1873
4	5,99%	112	1869
5	4,83%	84	1739
6	11,56%	376	3254
7	2,60%	44	1693

I dati riportati nella tabella confermano la natura asincrona dell'architettura di sistema e mettono in luce la gestione delle differenze prestazionali tra i vari sottomoduli. Poiché la frequenza di aggiornamento del sistema di tracciamento spaziale è nettamente superiore ai tempi di inferenza delle reti neurali, il server è progettato per non vincolare la trasmissione dei dati di posizionamento al completamento dell'analisi visiva.

Dall'analisi dei risultati è possibile evidenziare i seguenti aspetti:

- Tasso di omissione contenuto e stabilizzazione: nel complesso, la percentuale di frame con dati obsoleti risulta contenuta. Ciò comporta che il sistema risulti stabile nel complesso.

- Scelta architetturale e usabilità: la presenza di frame con rilevamenti ambientali obsoleti non costituisce un limite, bensì una garanzia di fluidità. L'invio di frame incompleti previene la creazione di colli di bottiglia nel modulo di posizionamento

puro (ORB-SLAM3). Dal punto di vista dell'usabilità finale, l'omissione saltuaria dei dati IA risulta impercettibile per l'utente: la dinamica degli ostacoli nell'ambiente reale è sufficientemente lenta da permettere all'applicativo di compensare la temporanea mancanza di aggiornamenti mantenendo in memoria l'ultima rilevazione valida, senza compromettere la sicurezza della navigazione.

5.1.5.2 Stabilità della trasmissione video

Un ulteriore dato rilevante riguarda le tempistiche di ricezione dei frame inviati dagli occhiali al server. Durante i test, gli occhiali sono stati configurati per trasmettere a una frequenza fissa di 10 FPS (cioè un intervallo teorico di 100 ms tra un frame e l'altro). Tuttavia, non potendo registrare il timestamp esatto in cui il frame viene effettivamente inviato dall'hardware degli occhiali, non è possibile calcolare la latenza di rete assoluta del singolo pacchetto. Pertanto, per valutare la stabilità del flusso di dati, è stato misurato l'intervallo temporale trascorso tra la ricezione di due frame consecutivi lato server. I risultati di questa misurazione sono riportati nella tabella 5.5.

Tabella 5.5: Intervallo temporale trascorso tra la ricezione di due frame consecutivi inviati dagli occhiali al server

Test	**Tempo medio (ms)**	**Tempo minimo (ms)**	**Tempo massimo (ms)**
1	107,287	11,232	4180,574
2	108,380	19,108	2521,801
3	105,997	6,849	2297,453
4	100,095	5,781	560,325
5	100,027	6,837	282,207
6	102,258	4,827	843,187
7	100,388	4,783	430,603

I dati presentati nella tabella offrono una panoramica chiara sul comportamento della trasmissione wireless in uno scenario reale, evidenziando le fisiologiche fluttuazioni tipiche delle reti Wi-Fi.

Dall'analisi dei tempi di ricezione emergono le seguenti considerazioni:

- Sostenibilità del framerate medio: Il tempo medio misurato in tutti i test si attesta tra i 100 ms e i 108 ms. Questo dato conferma che, sul lungo periodo, l'hardware degli occhiali e l'infrastruttura di acquisizione riescono a sostenere efficacemente il framerate di 10 FPS.

- Fenomeni di *jitter* e *packet bunching*: l'ampia forbice tra i tempi minimi e massimi indica una notevole variabilità nel ritardo di consegna dei pacchetti (jitter di rete). I tempi massimi, che nel Test 1 superano i 4 secondi, rappresentano dei momentanei stalli della comunicazione, verosimilmente causati da interferenze ambientali o congestione del canale Wi-Fi. Di contro, i tempi minimi estremamente ridotti (fino a circa 5 ms) riflettono il fenomeno del *packet bunching*, ovvero i frame ritardati si

accumulano nei buffer di rete per poi essere recapitati al server "a raffica", quasi simultaneamente ai frame successivi.

5.1.5.3 Prestazioni di inizializzazione e rilocalizzazione SLAM

Infine, è stato analizzato il numero di volte in cui il sistema entra nello stato di *posizionamento*, ovvero quando perde la propria collocazione all'interno della mappa e necessita di una rilocalizzazione. È stato inoltre misurato il numero di frame necessari per portare a termine tale operazione, sia in fase di inizializzazione che in fase di recupero. I dati sono esposti nella tabella 5.6.

Tabella 5.6: Statistiche dei frame necessari all'aggancio e alla rilocalizzazione SLAM

Test	**Numero di posizionamenti** [a]	**Inizializzazione (# frame)**	**Rilocalizzazione (# frame)**		
			Media	**Minimo**	**Massimo**
1	3	270	299,00	205	451
2	4	168	208,00	2	477
3	1	224	293,00	293	293
4	2	1485	130,00	1	258
5	1	348	5,00	5	5
6	6	219	166,33	1	492
7	2	347	8,50	4	11

[a] Escluso il posizionamento iniziale.

I dati presentati nella tabella offrono una panoramica fondamentale sulla robustezza del sistema di tracciamento visivo, misurando la sua capacità di reagire alla perdita di riferimenti spaziali. In un sistema basato su SLAM visivo, la perdita del tracciamento è una condizione fisiologica, specialmente in ambienti con pareti uniformi come quelli descritti in precedenza. Dall'analisi dei dati si possono ricavare tre osservazioni principali:

- Frequenza della perdita di tracciamento: il sistema ha dimostrato di perdere la propria posizione dalle 1 alle 4 volte per ogni sessione di test. Questo dato conferma che, pur essendo generalmente stabile, il sistema ORB-SLAM3 e sensibile a movimenti bruschi dell'utente, a cali di framerate dovuti alla rete o all'attraversamento di zone povere di dettagli visivi, costringendo il sistema a entrare nello stato di recupero.

- Fase di inizializzazione e l'anomalia del Test 4: nella maggior parte dei casi, l'inizializzazione della mappa richiede tra i 168 e i 350 frame. Considerando un flusso teorico di 10 frame al secondo, l'utente deve eseguire la procedura di posizionamento di circa 15-30 secondi prima che il sistema inizi a guidarlo. Il Test 4, tuttavia, presenta un valore anomalo molto elevato, pari a 1485 frame. Questo ritardo estremo nello stabilire la posizione iniziale è con ogni probabilità imputabile a condizioni di partenza sfavorevoli, come la presenza di una grande numero di persone, scarsa illuminazione o una forte latenza di rete nei primi istanti di connessione, che hanno impedito all'algoritmo di estrarre keypoint sufficienti per avviare il tracciamento.

- Dinamiche e tempi di rilocalizzazione: i valori di rilocalizzazione mostrano un'elevata varianza, evidenziando il funzionamento dell'algoritmo di riconoscimento dei luoghi. I tempi minimi registrati nei Test 2 e 4 sono eccellenti (rispettivamente 2 e 1 frame): questo significa che, se l'utente volge lo sguardo verso una zona già mappata e ricca di caratteristiche visive, il sistema e in grado di recuperare la posizione in modo quasi istantaneo (frazioni di secondo). Al contrario, i tempi massimi (che sfiorano i 470 frame) indicano che se la perdita di tracciamento avviene in un'area inesplorata o geometricamente ambigua, il sistema fatica a ritrovare le coordinate, costringendo l'utente a muoversi per diversi secondi prima di ristabilire la navigazione.

5.2 Valutazione dell'usabilità e test con gli utenti

Al fine di fornire una valutazione qualitativa del sistema, sono stati effettuati dei test utenti per misurare l'usabilità del sistema.

5.2.1 Metriche di valutazione e questionari

- Accuratezza: indica quanto il sistema consenta di avvicinarsi all'obiettivo desiderato. Nello specifico, misura la distanza finale tra l'utente e il target al termine del percorso. È espressa in metri.

- Tempo di esecuzione: quantifica il tempo impiegato dall'utente per portare a termine i task previsti dal test. È espresso in secondi.

Agli utenti è stato chiesto, dopo il test, di compilare due questionari[1]:

- *System Usability Scale* (SUS): un questionario standardizzato composto da 10 domande, utilizzato per misurare l'usabilità complessiva del sistema [7].

- Questionario ad hoc: un modulo personalizzato volto a raccogliere feedback sulle specifiche funzionalità del sistema proposto.

5.2.2 Protocollo sperimentale

Il protocollo di test adottato si ispira alla metodologia proposta da [9] ed è strutturato per valutare progressivamente l'efficacia del sistema.

In una fase preliminare, vengono illustrati all'utente lo scopo del progetto e il contesto di utilizzo. Successivamente, l'utente viene equipaggiato con il dispositivo mobile e gli occhiali. Prima di iniziare le prove ufficiali, gli viene richiesto di compiere un breve tragitto verso un punto limitrofo per familiarizzare con l'apparato e comprenderne il funzionamento.

[1] Le domande sono consultabili al seguente link: `https://docs.google.com/spreadsheets/d/1-tzJO1CJWSb2HJwE7jPCDX7SQeypkxbxTJoSBIjo1EA/edit?usp=sharing`

Conclusa questa fase di addestramento, ha inizio il test vero e proprio, suddiviso in due task:

1. Fase a vista libera: all'utente è richiesto di spostarsi dal punto di partenza (vicino alla Sala Studio R2, mostrato in figura 5.1) fino all'aula R2. Questo primo task permette all'utente di consolidare la confidenza con il sistema e di apprenderne le caratteristiche tecniche in un contesto sicuro.

2. Fase in assenza di vista: l'utente viene bendato e deve raggiungere l'aula R4b partendo dall'aula R2. Questa seconda fase mira a valutare il sistema in uno scenario realistico, simulando il caso d'uso finale con una persona non vedente.

Capitolo 6

Conclusioni e Sviluppi Futuri

Il presente lavoro di tesi si è posto l'obiettivo di progettare, sviluppare e testare un sistema di navigazione indoor per persone con disabilità visive, basato sull'integrazione di smart glasses Project Aria, visione artificiale e modelli di intelligenza artificiale. Il sistema sviluppato, nella configurazione architetturale proposta nei capitoli precedenti, si è dimostrato complessivamente stabile, consentendo all'utente di eseguire con successo i task fondamentali per la navigazione e il raggiungimento di una destinazione all'interno di un ambiente precedentemente mappato.

6.1 Sintesi dei risultati sperimentali

L'analisi dei dati raccolti durante la fase di testing (descritta nel capitolo 5) ha permesso di validare l'efficacia delle scelte progettuali, confermando la fattibilità di un approccio non invasivo e senza glifi (*markerless*).

Dal punto di vista dell'analisi spaziale, il sistema ha dimostrato un'eccellente precisione, con scostamenti medi compresi tra 0,1 e 0,2 metri, garantendo un comportamento ripetibile e affidabile nel tempo. L'accuratezza assoluta ha registrato errori fisiologici nell'ordine del metro, dovuti principalmente a scelte di sicurezza (offset dalle pareti) e alla conformazione architettonica, risultando comunque ampiamente compatibile con le esigenze di navigazione e sufficiente per guidare l'utente in prossimità dell'obiettivo finale.

Per quanto concerne l'analisi temporale e computazionale, i test hanno evidenziato una netta dicotomia tra le altissime prestazioni di elaborazione locale e le latenze di trasmissione. I moduli di calcolo si sono rivelati estremamente efficienti: l'algoritmo ORB-SLAM3 ha operato in frazioni di millisecondo, mentre l'inferenza combinata delle reti neurali (YOLO e ZoeDepth) si è mantenuta stabile attorno agli 85 ms per frame. Tuttavia, il trasferimento dei dati via Wi-Fi verso l'applicativo Unity ha rappresentato il principale collo di bottiglia, con latenze di ricezione che, in condizioni di congestione di rete, hanno superato i diversi secondi.

Nonostante queste sfide infrastrutturali, l'architettura asincrona ha garantito la robustezza e l'usabilità del sistema. La scelta di disaccoppiare la frequenza di aggiornamento

dello SLAM da quella dell'IA ha comportato l'omissione fisiologica dei dati sugli ostacoli in una piccola percentuale di frame (5% - 10%), assicurando però una fluidità di tracciamento che ha permesso all'utente di non percepire interruzioni.

Sebbene gli smart glasses Project Aria Gen1 abbiano permesso un'elaborazione adeguata, i risultati dei test evidenziano alcuni margini di miglioramento lato hardware. Come suggerito dalla letteratura di riferimento per ORB-SLAM3 [8], l'adozione di dispositivi indossabili dotati di configurazioni ottiche stereo o stereo-inerziali, compatibili con il sistema, garantirebbe una maggiore stabilità e robustezza nel tracking spaziale rispetto all'attuale approccio. Inoltre, sul piano architetturale, si ipotizza che l'integrazione di sensori LiDAR risulterebbe un'evoluzione particolarmente vantaggiosa: tale soluzione permetterebbe infatti di bypassare del tutto l'uso di reti neurali per la stima della profondità (come ZoeDepth), fornendo misurazioni tridimensionali dirette e abilitando l'impiego di reti specializzate nell'elaborazione di nuvole di punti (PCM).

6.2 Limitazioni

Nonostante i risultati positivi, lo studio ha evidenziato alcune limitazioni tecniche:

- Perdita di tracking per movimenti bruschi: il sistema tende a perdere il posizionamento SLAM in caso di rotazioni rapide della testa. Questo fenomeno è strettamente legato al limite di trasmissione dati impostato a 10 FPS sugli occhiali. I tentativi di aumentare il numero di frame trasmessi hanno comportato un degrado delle prestazioni complessive, evidenziando un collo di bottiglia nella capacità di ricezione ed elaborazione real-time da parte del server.

- Dipendenza da hardware ad alte prestazioni e rete: attualmente, l'architettura si affida a un server remoto ad alte prestazioni (dotato di GPU RTX 4080) e a una connessione Wi-Fi stabile. Ciò limita la reale portabilità del sistema in scenari outdoor o in assenza di copertura di rete adeguata.

- Asincronia dell'IA: Come emerso dall'analisi delle latenze, la discrepanza tra i tempi di calcolo dello SLAM e quelli delle reti neurali costringe il sistema, in alcuni casi (più del 10% dei frame), a fornire all'utente dati di posizionamento privi delle informazioni aggiornate sugli ostacoli.

6.3 Sviluppi futuri

Partendo dalle limitazioni riscontrate, si propongono le seguenti direzioni per i futuri sviluppi del progetto:

- Navigazione a grana fine: attualmente il sistema è ottimizzato per lo spostamento tra aule diverse. Un'evoluzione naturale consisterebbe nell'affinamento dei moduli di localizzazione per guidare l'utente con elevata precisione anche all'interno di spazi ristretti, come aule o sale studio, al fine di fargli raggiungere specifici posti a sedere.

- Aggiornamento hardware (Aria Gen2): per risolvere le criticità legate al framerate e ai sensori, sarà fondamentale testare il sistema con i nuovi occhiali Project Aria Gen2 (la cui distribuzione è iniziata nei primi mesi del 2026), valutando come l'hardware aggiornato impatti sulla latenza e sulla stabilità dello SLAM.
- Identificazione avanzata degli ostacoli: si prevede l'addestramento e l'integrazione di una rete neurale in grado non solo di classificare gli oggetti e stimarne la distanza, ma anche di calcolarne l'orientamento spaziale. L'utilizzo di reti basate su OBB, opportunamente adattate per la visuale egocentrica in prima persona, migliorerebbe significativamente la precisione nell'elusione degli ostacoli.

Bibliografia

[1] Mustufa Haider Abidi, Arshad Noor Siddiquee, Hisham Alkhalefah, and Vishwaraj Srivastava. A comprehensive review of navigation systems for visually impaired individuals. *Heliyon 10*, 2024.

[2] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023.

[3] Armen Avetisyan, Christopher Xie, Henry Howard-Jenkins, Tsun-Yi Yang, Samir Aroudj, Suvam Patra, Fuyang Zhang, Duncan Frost, Luke Holland, Campbell Orme, Jakob Engel, Edward Miller, Richard Newcombe, and Vasileios Balntas. Scenescript: Reconstructing scenes with an autoregressive structured language model, 2024.

[4] Be My Eyes. Be my eyes, 2024.

[5] Shariq Farooq Bhat, Reiner Birkl, Diana Wofk, Peter Wonka, and Matthias MÃ¼ller. Zoedepth: Zero-shot transfer by combining relative and metric depth, 2023.

[6] Garrick Brazil, Abhinav Kumar, Julian Straub, Nikhila Ravi, Justin Johnson, and Georgia Gkioxari. Omni3d: A large benchmark and model for 3d object detection in the wild, 2023.

[7] John Brooke. Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189, 11 1995.

[8] Carlos Campos, Richard Elvira, Juan J. Gomez Rodriguez, Jose M. M. Montiel, and Juan D. Tardòs. Orb-slam3: An accurate open-source library for visual, visual-inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, December 2021.

[9] Simona Caraiman, Anca Morar, Mateusz Owczarek, Adrian Burlacu, Dariusz Rzeszotarski, Nicolae Botezatu, Paul Herghelegiu, Florica Moldoveanu, Pawel Strumillo, and Alin Moldoveanu. Computer vision for the visually impaired: the sound of vision system. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 1480–1489, 2017.

[10] Qingtian Chen, Muhammad Khan, Christina Tsangouri, Christopher Yang, Bing Li, Jizhong Xiao, and Zhigang Zhu. Ccny smart cane. In *2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 1246–1251, 2017.

[11] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. In *Proceedings of*

the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[12] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals using stereo imagery for accurate object class detection, 2017.

[13] Tianheng Cheng, Lin Song, Yixiao Ge, Wenyu Liu, Xinggang Wang, and Ying Shan. Yolo-world: Real-time open-vocabulary object detection, 2024.

[14] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[15] Dimitrios Dakopoulos. Tyflos: A wearable navigation prototype for blind and visually impaired; design, modelling and experimental results. Master's thesis, Wright State University, 2009.

[16] Jakob Engel, Kiran Somasundaram, Michael Goesele, Albert Sun, Alexander Gamino, Andrew Turner, Arjang Talattof, Arnie Yuan, Bilal Souti, Brighid Meredith, Cheng Peng, Chris Sweeney, Cole Wilson, Dan Barnes, Daniel DeTone, David Caruso, Derek Valleroy, Dinesh Ginjupalli, Duncan Frost, Edward Miller, Elias Mueggler, Evgeniy Oleinik, Fan Zhang, Guruprasad Somasundaram, Gustavo Solaira, Harry Lanaras, Henry Howard-Jenkins, Huixuan Tang, Hyo Jin Kim, Jaime Rivera, Ji Luo, Jing Dong, Julian Straub, Kevin Bailey, Kevin Eckenhoff, Lingni Ma, Luis Pesqueira, Mark Schwesinger, Maurizio Monge, Nan Yang, Nick Charron, Nikhil Raina, Omkar Parkhi, Peter Borschowa, Pierre Moulon, Prince Gupta, Raul Mur-Artal, Robbie Pennington, Sachin Kulkarni, Sagar Miglani, Santosh Gondi, Saransh Solanki, Sean Diener, Shangyi Cheng, Simon Green, Steve Saarinen, Suvam Patra, Tassos Mourikis, Thomas Whelan, Tripti Singh, Vasileios Balntas, Vijay Baiyya, Wilson Dreewes, Xiaqing Pan, Yang Lou, Yipu Zhao, Yusuf Mansour, Yuyang Zou, Zhaoyang Lv, Zijian Wang, Mingfei Yan, Carl Ren, Renzo De Nardi, and Richard Newcombe. Project aria: A new tool for egocentric multi-modal ai research, 2023.

[17] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[18] Agrim Gupta, Piotr Dollàr, and Ross Girshick. Lvis: A dataset for large vocabulary instance segmentation, 2019.

[19] Glenn Jocher. Ultralytics yolov5, 2020.

[20] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLOv8. `https://github.com/ultralytics/ultralytics`, 2023. Versione 8.0.0, Licenza AGPL-3.0.

[21] Sulaiman Khan, Shah Nazir, and Habib Ullah Khan. Analysis of navigation assistants for blind and visually impaired people: A systematic review. *IEEE Access*, 9:26712–26734, 2021.

[22] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Waslander. Joint 3d proposal generation and object detection from view aggregation, 2018.

[23] Süleyman Yasir Kula. Unity-speech-to-text: Speech-to-text library for unity. `https://github.com/yasirkula/UnitySpeechToText`, 2023.

[24] Bing Li, Juan Pablo Muñoz, Xuejian Rong, Qingtian Chen, Jizhong Xiao, Yingli Tian, Aries Arditi, and Mohammed Yousuf. Vision-based mobile indoor assistive navigation aid for blind people. *IEEE Transactions on Mobile Computing*, 18(3):702–714, 2019.

[25] Chuyi Li, Lulu Li, Yifei Geng, Hongliang Jiang, Meng Cheng, Bo Zhang, Zaidan Ke, Xiaoming Xu, and Xiangxiang Chu. Yolov6 v3.0: A full-scale reloading, 2023.

[26] Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, Kai-Wei Chang, and Jianfeng Gao. Grounded language-image pre-training, 2022.

[27] Wei Liang, Pengfei Xu, Ling Guo, Heng Bai, Yang Zhou, and Feng Chen. A survey of 3d object detection. *Multimedia Tools and Applications*, 80(19):29617–29641, 2021.

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr DollÃ¡r. Microsoft coco: Common objects in context, 2015.

[29] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021.

[30] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[31] Meta. Reality labs. `https://tech.facebook.com/reality-labs/`. Access: 2026/02/03.

[32] Microsoft Garage. Seeing ai, 2024.

[33] Ishan Misra, Rohit Girdhar, and Armand Joulin. An end-to-end transformer model for 3d object detection, 2021.

[34] Raùl Mur-Artal, J. M. M. Montiel, and Juan D. Tardòs. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[35] Raùl Mur-Artal and Juan D. Tardòs. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[36] Vishnu Nair, Christina Tsangouri, Bowen Xiao, Greg Olmschenk, Zhigang Zhu, and William H. Seiple. A hybrid indoor positioning system for the blind and visually impaired using bluetooth and google tango. *Journal on technology and persons with disabilities*, 6:62–82, 2018.

[37] Bryan A. Plummer, Liwei Wang, Chris M. Cervantes, Juan C. Caicedo, Julia Hockenmaier, and Svetlana Lazebnik. Flickr30k entities: Collecting region-to-phrase correspondences for richer image-to-sentence models, 2016.

[38] Hosein Porazar. Android-native-tts-plugin-for-unity-3d. `https://github.com/HoseinPorazar/Android-Native-TTS-plugin-for-Unity-3d`, 2017.

[39] projectAria. About project aria. `https://www.projectaria.com/our-work/`. Access: 2026/02/03.

[40] ProjectAria. Aria synthetic environments dataset. `https://www.projectaria.com/datasets/ase/`. Access: 2025/07/22.

[41] Renè Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer, 2020.

[42] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[43] Facebook Research. Vrs documentation. `https://facebookresearch.github.io/vrs/docs/Overview/`. Access: 2026/02/03.

[44] Danila Rukhovich, Anna Vorontsova, and Anton Konushin. Imvoxelnet: Image to voxels projection for monocular and multi-view general-purpose 3d object detection, 2021.

[45] Shuai Shao, Zeming Li, Tianyuan Zhang, Chao Peng, Gang Yu, Xiangyu Zhang, Jing Li, and Jian Sun. Objects365: A large-scale, high-quality dataset for object detection. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8429–8438, 2019.

[46] Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In *Annual Meeting of the Association for Computational Linguistics*, 2018.

[47] Unity Technologies. Unity manual: Navigation and pathfinding. `https://docs.unity3d.com/Packages/com.unity.ai.navigation@2.0/manual/index.html`.

[48] Unity Technologies. Unity real-time development platform. `https://unity.com`.

[49] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2022.

[50] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. Yolov9: Learning what you want to learn using programmable gradient information, 2024.

[51] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q. Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving, 2020.

[52] Widyawan Widyawan, Muhamad Saputra, and Paulus Santosa. Invys: Indoor navigation system for persons with visual impairment using rgb-d camera. *Jurnal Nasional Teknik Elektro dan Teknologi Informasi*, 12:293–302, 11 2023.

[53] Bin Xu and Zhenzhong Chen. Multi-level fusion based 3d object detection from monocular images. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2345–2353, 2018.

[54] Lewei Yao, Jianhua Han, Youpeng Wen, Xiaodan Liang, Dan Xu, Wei Zhang, Zhenguo Li, Chunjing Xu, and Hang Xu. Detclip: Dictionary-enriched visual-concept paralleled pre-training for open-world detection, 2022.