DOCUMENTAZIONE TECNICA

IA2 e ROBOTICA

Sommario

[DOCUMENTAZIONE IA2 e ROBOTICA 1](#_Toc220604361)

[Capitolo 1: Introduzione 3](#_Toc220604362)

[1.1 Obiettivo 3](#_Toc220604363)

[1.2 Architettura 3](#_Toc220604364)

[1.3 Ambiente (Hotel) 3](#_Toc220604365)

[1.3.1 Luoghi importanti e Topologia 3](#_Toc220604366)

[1.3.2 Ostacoli 4](#_Toc220604367)

[1.4 Robot (Pippor) 5](#_Toc220604368)

[1.4.1 Struttura gerarchica 5](#_Toc220604369)

[1.4.2 Sensori 5](#_Toc220604370)

[1.4.3 Ruote e guida differenziale 5](#_Toc220604371)

[1.4.4 La sfida della massa 6](#_Toc220604372)

[1.4.5 Watchdog e sicurezza 6](#_Toc220604373)

[Capitolo 2: Robotica 7](#_Toc220604374)

[2.1 Sistema di arbitraggio 7](#_Toc220604375)

[2.2 Sistema di guida differenziale 7](#_Toc220604376)

[2.3 Pianificazione 8](#_Toc220604377)

[2.4 Sistema di navigazione 8](#_Toc220604378)

[2.5 TF 9](#_Toc220604379)

[2.6 Localizzazione 10](#_Toc220604380)

[2.7 Elaborazione dei Segnali Biometrici tramite Filtro di Kalman 11](#_Toc220604381)

[Capitolo 3: Intelligenza Artificiale 2 13](#_Toc220604382)

[3.1 Large Language Model 13](#_Toc220604383)

[3.2 Base di conoscenza e Ontologia 13](#_Toc220604384)

[3.2.1 Rappresentazione Ontologia 14](#_Toc220604385)

[3.3 Explainability 15](#_Toc220604386)

[3.4 Implementazione scenari 15](#_Toc220604387)

[3.4.1 Scenario A 15](#_Toc220604388)

[3.4.2 Scenario B 17](#_Toc220604389)

[3.4.3 Scenario C 17](#_Toc220604390)

# Capitolo 1: Introduzione

## 1.1 Obiettivo

Lo scopo del nostro progetto è stato quello di implementare in un ambiente simulato un Robot sociale autonomo, chiamato **Pippor**, progettato per operare all'interno di strutture ricettive nel contesto dell'ospitalità. Il compito di Pippor è quello di muoversi autonomamente nella struttura per assistere i clienti a 360 gradi: dall'accoglienza personalizzata alla gestione delle emergenze, individuando anomalie o richieste di aiuto.

Durante il suo servizio, Pippor sarà costantemente connesso a dei braccialetti smart forniti agli ospiti. Questi ultimi sono strumenti essenziali capaci non solo di identificare l'utente, ma di monitorarne in tempo reale i parametri vitali, allertando il robot in caso di malori o situazioni di pericolo.

Durante l'interazione, Pippor raccoglierà dati sul contesto ambientale e sanitario, sfruttando un'intelligenza ibrida per dialogare empaticamente e ragionare logicamente, con l'obiettivo finale di garantire un soggiorno sicuro e di agire tempestivamente, chiamando soccorsi o supporto tecnico solo quando strettamente necessario.

## 1.2 Architettura

AGGIUNGERE RQTGRAPH

L'architettura complessiva del progetto si basa su un paradigma distribuito che disaccoppia nettamente il livello di simulazione fisico-visiva dal livello di controllo logico-cognitivo. Per realizzare questo ambiente di test, abbiamo integrato due ecosistemi distinti: **Unity**, che funge da motore di simulazione e **ROS2** che ospita l'intero stack di intelligenza del robot (navigazione, arbitraggio, reasoner e gestione dei braccialetti).

Il collegamento tra questi due mondi è stato realizzato utilizzando **Unity Robotics Hub**. Nello specifico, la comunicazione avviene tramite protocollo TCP/IP grazie all'interazione tra due componenti middleware:

* **ROS-TCP-Connector (lato Unity):** Un componente scripts C# che serializza i dati di simulazione in messaggi standard ROS e li trasmette alla rete.
* **ROS-TCP-Endpoint (lato ROS):** Un nodo server in ascolto che riceve i pacchetti serializzati, li deserializza in topic ROS2 nativi e, viceversa, inoltra i comandi di controllo provenienti dal robot verso Unity.

## 1.3 Ambiente (Hotel)

L'ambiente di simulazione è stato progettato per rappresentare uno scenario realistico di servizio in un hotel. La scelta di questo setting non è puramente estetica, ma funzionale: offre una varietà di spazi con caratteristiche geometriche e di navigazione distinte (corridoi stretti, aree aperte, ostacoli dinamici) ideali per testare algoritmi di localizzazione e pianificazione del movimento.

### 1.3.1 Luoghi importanti e Topologia

L'architettura della scena Unity è stata suddivisa in tre zone macroscopiche, ognuna delle quali presenta sfide specifiche per il sistema di navigazione autonoma del robot Pepper. La continuità spaziale è garantita da un sistema di coordinate coerente (Unity World Space) che viene poi mappato nel sistema di riferimento di ROS2 per la navigazione.

#### Reception (Punto di Ingresso)

È un'area caratterizzata da un bancone statico che funge da barriera fisica e visiva. L'area è ampia per permettere manovre di rotazione sul posto (zero-turn) tipiche della guida differenziale. Qui è collocato il robot. È il luogo dove idealmente inizia l'interazione uomo-macchina. La geometria del bancone è stata impostata con Collider statici precisi per evitare compenetrazioni durante le fasi di docking o avvicinamento.

#### Corridoi

Rappresentano la sfida principale di “Path Planning”, collegando tutti i luoghi di interesse e costringono il robot a muoversi in uno spazio confinato. La larghezza ridotta dei corridoi riduce il margine di errore per l'algoritmo di localizzazione. Qualsiasi deviazione significativa dalla traiettoria ottimale rischia di portare il robot in collisione con le pareti. Qui sono stati posizionati dei *Waypoints* invisibili utilizzati dagli NPC per il loro pattugliamento. Questo trasforma il corridoio in un'area ad alto traffico, dove il robot deve costantemente ricalcolare la traiettoria per evitare collisioni dinamiche.

#### CoffeeRoom

È un ambiente non strutturato e denso. A differenza dei corridoi, qui gli ostacoli non sono solo perimetrali. La presenza di tavoli rotondi e sedie crea una mappa di occupazione complessa. Le gambe dei tavoli e delle sedie sono ostacoli sottili che possono essere difficili da rilevare per sensori a bassa risoluzione, richiedendo un Raycasting ad alta frequenza. Quest'area funge da punto di aggregazione naturale per gli NPC, permettendo di testare la capacità del robot di navigare in spazi ristretti e potenzialmente affollati. Tuttavia, la simulazione prevede che gli eventi critici (ad esempio un malore) siano distribuiti logicamente in tutto l'hotel in quanto sono eventi stocastici che possono verificarsi casualmente in qualsiasi punto della mappa, inclusa questa sala, obbligando il robot ad una risposta adattiva ovunque si trovi.

#### Stazione di ricarica

È il punto dove staziona il robot nei momenti di riposo. Non è solo un oggetto passivo, ma un volume logico. È stato implementato un Box Collider con proprietà IsTrigger. Il sistema è progettato affinché il robot debba fisicamente entrare in questo volume per attivare la logica di ricarica. Questo ha richiesto un tuning preciso dell'altezza del collider della stazione per garantire l'intersezione con il collider del robot (che si trova sollevato da terra a causa delle ruote), risolvendo problemi di mancata rilevazione dovuti a collider troppo bassi o piatti ("Plane").

### 1.3.2 Ostacoli

La navigazione nell'hotel si basa sul sistema *NavMesh* (Navigation Mesh) di Unity, che definisce le aree camminabili.

Distinguiamo 2 tipi di ostacoli: statici e dinamici.

#### Ostacoli statici

Tutti gli elementi architettonici (muri, colonne, bancone, tavoli) sono stati marcati come *Navigation Static*. Durante il processo di "Baking" della NavMesh, Unity calcola le superfici percorribili sottraendo il volume di questi ostacoli più un raggio di sicurezza (Agent Radius). Questo impedisce al robot di pianificare percorsi che lo porterebbero a sfiorare i muri, riducendo il rischio di incastro fisico.

#### Ostacoli dinamici

La vera complessità è introdotta dagli NPC. Ogni persona è governata da uno script NPCController personalizzato. Essi non sono semplici ostacoli mobili, ma agenti intelligenti che utilizzano la stessa NavMesh del robot. Gli NPC hanno stati logici (Pattugliamento, Attesa, Scenari di Emergenza). Quando si attiva uno scenario, l'NPC diventa un ostacolo statico improvviso in una posizione non prevista dalla mappa statica.

Gli NPC non sono entità anonime; le loro identità (Nome, Cognome, ID) e parametri vitali sono iniettati dinamicamente all'avvio prelevandoli da **Neo4j**. Questo aggiunge un livello di semantica all'ambiente: il robot non vede solo un "cilindro" da evitare, ma un'entità con cui può comunicare via ROS2 (topic /unity/health\_raw e /unity/dialogo). L'implementazione delle animazioni è servita per garantire una maggiore **realisticità** alla simulazione. I personaggi non si limitano a traslare nello spazio, ma eseguono movimenti complessi (camminata, attesa, caduta) che mimano il comportamento umano reale. Parallelamente, è stata posta grande attenzione alla configurazione dei *Collider* (i volumi di collisione fisica). Questi sono essenziali per **evitare problemi di navigazione**: impediscono ai personaggi di attraversare muri o oggetti e assicurano che il robot rilevi correttamente la presenza fisica delle persone, permettendo al sistema di guida di calcolare percorsi sicuri senza compenetrazioni.

## 1.4 Robot (Pippor)

Il protagonista della simulazione è una replica digitale del robot Pepper di SoftBank Robotics. La sua implementazione in Unity non è una semplice animazione, ma una simulazione fisica completa basata su giunti, forze e comunicazione bidirezionale con ROS2. Il robot è stato modellato come un sistema multi-corpo (Multi-Body System).

### 1.4.1 Struttura gerarchica

La gerarchia della Prefab di Unity riflette la cinematica del robot reale.

* **Chassis (Base):** Il corpo principale, che ospita il computer di bordo simulato e le batterie. Ha una massa definita di circa 28 kg, un fattore determinante per la simulazione fisica che ha richiesto un tuning avanzato dei motori.
* **Batteria (Power Management):** È stato implementato un sottosistema energetico tramite lo script RobotBattery. Questo componente non si limita a decrementare un contatore, ma simula cicli di carica/scarica basati sul tempo e sull'interazione con la stazione di ricarica. I dati vengono serializzati in un formato **JSON** ({"level": 85, "is\_charging": true}) e inviati a ROS, permettendo al sistema di prendere decisioni autonome.

### 1.4.2 Sensori

La percezione ambientale del robot è affidata a un **LiDAR 2D simulato**, che costituisce il sensore cardine per le operazioni di mapping e navigazione autonoma. All'interno dell'ambiente Unity, il funzionamento fisico del dispositivo è riprodotto fedelmente attraverso la tecnica del **Raycasting**: uno script dedicato emette ciclicamente una scansione radiale di raggi invisibili su un piano orizzontale, coprendo il campo visivo del robot. Il motore fisico calcola in tempo reale le intersezioni tra questi vettori e i Collider degli elementi scenici (siano essi ostacoli statici come muri o dinamici come gli NPC), determinando la distanza esatta di ogni punto di impatto. Questi dati geometrici vengono infine serializzati nel formato standard sensor\_msgs/LaserScan e trasmessi al nodo ROS, permettendo allo stack di navigazione di ricostruire la geometria degli ostacoli circostanti con la stessa logica di un sensore hardware reale.

### 1.4.3 Ruote e guida differenziale

Il sistema di locomozione è il cuore ingegneristico della simulazione. Pippor utilizza una configurazione a **guida differenziale** (due ruote motrici indipendenti e ruote omnidirezionali passive per l'equilibrio).

Per la simulazione fisica delle ruote, abbiamo abbandonato i classici HingeJoint in favore dei componenti **ArticulationBody** di Unity. Essi offrono una simulazione molto più stabile e precisa per le catene cinematiche robotiche, utilizzando un solver fisico dedicato (Featherstone) che riduce il "jittering" e gli errori di calcolo nelle giunzioni. Ogni ruota è controllata in velocità (TargetVelocity) attraverso il drive interno del componente.

### 1.4.4 La sfida della massa

Una delle sfide tecniche maggiori riscontrate durante lo sviluppo è stata la gestione della massa.

Inizialmente, con una massa realistica di 28 kg per lo chassis, il robot risultava "pigro", lento ad accelerare e incapace di frenare tempestivamente, scivolando come su ghiaccio o muovendosi a scatti. Questo avveniva perché i parametri di default di Unity sono tarati per oggetti leggeri (~1 kg).

La soluzione è stata quella di ricalibrare manualmente i parametri dell'azionamento (xDrive) nello script DiffDriveRobot.

* **Force Limit (Torque):** Elevato a valori molto alti (es. 1.000). Questo non aumenta la velocità massima, ma fornisce al motore la "coppia" (N·m) necessaria per vincere l'inerzia di un corpo da 28 kg.
* **Damping (Smorzamento):** Impostato a valori elevati (es. 500). Nel controllo in velocità, il Damping agisce come un guadagno proporzionale: definisce quanto "aggressivamente" il motore cerca di annullare la differenza tra velocità attuale e velocità target. Senza un damping elevato, il motore non applicava abbastanza forza frenante per arrestare la massa del robot.
* **Massa delle Ruote:** Abbiamo aumentato la massa delle singole ruote a circa 1.5 kg. Un rapporto di massa troppo elevato tra corpo (28kg) e ruote (pochi grammi) destabilizza il solver fisico. Bilanciando le masse, la simulazione è diventata fluida e reattiva.

### 1.4.5 Watchdog e sicurezza

Nello script di controllo (DiffDriveRobot.cs), è stato implementato un sistema di Safety Watchdog. Se il robot non riceve comandi di velocità dai topic ROS (/cmd\_vel o specifici per ruota) per più di un secondo (a causa di latenza di rete o crash del nodo di navigazione), il robot non continua a muoversi all'infinito con l'ultimo comando ricevuto. Invece di un blocco istantaneo (che causerebbe ribaltamenti o slittamenti irrealistici), il codice applica una decelerazione progressiva, riducendo la targetVelocity delle ruote verso zero in modo controllato, simulando l'attrito e la frenata rigenerativa dei motori reali.

# Capitolo 2: Robotica

In questo capitolo tratteremo più nello specifico gli argomenti del corso che ci sono stati utili per la realizzazione del progetto, di come sono stati implementati e di come la loro coesione permetta di costruire un sistema autonomo ed efficiente.

## 2.1 Sistema di arbitraggio

Il nodo Arbitraggio costituisce il nucleo decisionale dell'architettura di controllo, orchestrando le attività del robot attraverso un ciclo di valutazione ad alta frequenza (10 Hz) basato su un paradigma di priorità fisse. A differenza delle macchine a stati finiti tradizionali, questo sistema adotta una struttura a soppressione (Subsumption) definita da una lista ordinata di comportamenti: ad ogni iterazione, il sistema scansiona i trigger partendo dall'alto e attiva esclusivamente il primo modulo che soddisfa le condizioni logiche, inibendo l'esecuzione di tutti i livelli sottostanti.

La gerarchia è stata progettata con un preciso criterio di sicurezza: al vertice assoluto (Indice 0) è stato collocato lo **Scenario C (Gestione Emergenze)**. Questa scelta riflette un vincolo etico e operativo imprescindibile: la salvaguardia della salute dell'ospite deve prevalere su qualsiasi altra funzione, inclusa la sopravvivenza del robot stesso. L'attivazione di questo scenario, governata sia dalla pressione manuale del pulsante di soccorso (reattiva) che dal rilevamento automatico di anomalie biometriche critiche (proattiva), provoca l'immediata sospensione di ogni altra attività.

Immediatamente subordinata all'emergenza troviamo la **Gestione Energetica** (Indice 1). Il comportamento di ricarica scatta quando l'autonomia scende sotto la soglia di sicurezza, ma la sua posizione secondaria rispetto allo Scenario C implica che il robot sacrificherà la propria autonomia (rischiando lo spegnimento) pur di non abbandonare un ospite in pericolo di vita.

A livello intermedio (Indici 2 e 3) si collocano gli scenari di interazione sociale on-demand: lo **Scenario B** (Assistenza Guasti) e lo **Scenario A** (Info-Concierge). Per questi task, il codice implementa un meccanismo di esclusione reciproca per evitare comportamenti incoerenti durante il dialogo, pur mantenendo sempre la possibilità di essere interrotti da un'emergenza medica improvvisa. Infine, alla base della piramide (Indici 4 e 5), troviamo i comportamenti di routine come la **Navigazione** e il **Riposo**, che fungono da fallback operativi attivi solo in assenza di richieste prioritarie.

Immagine che contiene testo, diagramma, schermata, Piano

Il contenuto generato dall'IA potrebbe non essere corretto.

## 2.2 Sistema di guida differenziale

Il controllo basso livello della piattaforma mobile è gestito dal nodo DiffRobotController, il quale funge da interfaccia bidirezionale tra la logica di navigazione (che ragiona in termini di velocità lineari e angolari complessive) e gli attuatori fisici (che richiedono comandi specifici per singola ruota). L'architettura del nodo si fonda sul modello cinematico del **pilota differenziale** (Differential Drive), parametrizzato sulle specifiche fisiche del robot: il raggio delle ruote (R) e la distanza interasse (L).

Il funzionamento del controller si divide in due pipeline principali: la **cinematica inversa per l'attuazione** e la **cinematica diretta per la stima della posa**.

Per quanto riguarda l'attuazione (metodo command\_turn), il sistema sottoscrive i comandi di velocità (/cmd\_vel) e li traduce in velocità angolari per i motori destro e sinistro; la logica applica le equazioni del *modello unicycle*, distribuendo la velocità lineare e quella angolare sulle due ruote in funzione della carreggiata, permettendo al robot di curvare variando la velocità relativa dei due lati.

Parallelamente, il sistema esegue la stima odometrica (metodo compute\_odometry) leggendo i feedback degli encoder (/wheels\_state). L'implementazione adottata è robusta rispetto alle singolarità matematiche, infatti l'algoritmo valuta dinamicamente la differenza di velocità tra le ruote per distinguere tra moto rettilineo e curvilineo. Nel caso di moto curvilineo, viene calcolato il **Centro Istantaneo di Curvatura (ICC)** per aggiornare la posa (x, y, ) lungo un arco di circonferenza esatto, garantendo una precisione superiore rispetto alla semplice integrazione lineare, mentre nel caso di moto rettilineo si applica una integrazione standard per evitare divisioni per zero.

Immagine che contiene diagramma, cerchio, design

Il contenuto generato dall'IA potrebbe non essere corretto.

A completamento dell'infrastruttura ROS, il nodo gestisce la pubblicazione delle trasformate (TF) necessarie: trasmette dinamicamente la trasformazione tra il sistema di riferimento odometrico e la base del robot, pubblica le trasformate statiche per il Lidar e mantiene uno storico delle pose (Path) per la visualizzazione della traiettoria effettiva in RViz, occupandosi inoltre di inizializzare il sistema di localizzazione AMCL alla posa di start predefinita.

## 2.3 Pianificazione

Per il modulo di navigazione, è stata sviluppata un'implementazione personalizzata dell'algoritmo **A\***, ottimizzata per operare su mappe a griglia di occupazione (*Occupancy Grid*) con connettività a 8 vicini. A differenza di pianificatori più semplici limitati ai movimenti cardinali, il nostro sistema sfrutta una topologia "8-connected" che permette spostamenti diagonali; per gestire correttamente i costi di attraversamento, l'algoritmo assegna un peso di 1.0 ai passi ortogonali e di a quelli diagonali, utilizzando coerentemente una **euristica Octile** per la stima del costo residuo (h), che garantisce l'ammissibilità del percorso minimizzando l'errore di stima rispetto alla distanza Euclidea o Manhattan.

Un aspetto critico della sicurezza di navigazione è gestito attraverso un controllo esplicito di **"Corner Cutting Prevention"**: prima di validare un movimento diagonale, l'algoritmo verifica che le due celle cardinali adiacenti all'angolo di svolta siano libere. Questo impedisce al robot di tentare traiettorie che "taglierebbero" lo spigolo di un ostacolo, evitando collisioni fisiche in passaggi stretti.

L'implementazione offre inoltre una flessibilità operativa nella gestione dei dati di mappa: oltre a distinguere tra spazio libero (0) e occupato (100), è stato introdotto il flag allow\_unknown per decidere dinamicamente se considerare le celle inesplorate (-1) come attraversabili o meno, adattando il comportamento in base alla fase. Infine, per ottimizzare i tempi di calcolo durante le operazioni di ripianificazione locale, la funzione supporta un vincolo di **Bounding Box** (bbox), che circoscrive l'espansione dei nodi all'interno di una sottofinestra specifica dell'area operativa, riducendo drasticamente il carico computazionale rispetto a una ricerca sull'intera mappa globale.

Immagine che contiene testo, cartone animato, schermata, clipart

Il contenuto generato dall'IA potrebbe non essere corretto.

## 2.4 Sistema di navigazione

Il modulo Naviga costituisce il nucleo architetturale deputato alla gestione della mobilità autonoma, integrando la percezione ambientale, la pianificazione globale e il controllo di traiettoria in un unico loop di controllo ad alta frequenza (10 Hz). La gestione della mappa, ricevuta sul topic /map, implementa una strategia di sicurezza preventiva denominata **Map Inflation**: per garantire che il robot mantenga una distanza di sicurezza dagli ostacoli fisici, i dati della griglia di occupazione vengono elaborati tramite un algoritmo ottimizzato che "gonfia" gli ostacoli di un raggio di sicurezza (SAFETY\_RADIUS). Per massimizzare le performance in Python ed evitare colli di bottiglia computazionali, l'algoritmo calcola anticipatamente gli offset geometrici del "pennello" di espansione una sola volta e applica le modifiche esclusivamente agli indici delle celle occupate, evitando iterazioni ridondanti sull'intera matrice.

La pianificazione del percorso è demandata alla funzione pianifica, che integra l'algoritmo A\* precedentemente descritto. Una volta ottenuto il percorso grezzo, viene applicata una routine di semplificazione (simplify\_path) che rimuove i waypoint intermedi ridondanti lungo i segmenti rettilinei, generando una traiettoria vettoriale più pulita composta solo dai punti di svolta essenziali.

Il cuore operativo è il metodo control\_loop, che implementa una macchina a stati per l'inseguimento dei waypoint. Il sistema aggiorna costantemente la propria posa interrogando il buffer delle trasformate TF2 (map -> base\_link) e calcola l'errore angolare rispetto al prossimo target.

La logica di controllo adotta un approccio ibrido per gestire la cinematica differenziale:

* **Stato di rotazione (Turn-in-place):** Se l'errore angolare supera una soglia critica (TOLLERANZA\_START\_ROT), il robot si ferma e ruota sul posto utilizzando un controllore PI (Proporzionale-Integrale) per annullare l'errore di orientamento.
* **Stato di avanzamento (Drive-to-goal):** Quando l'allineamento rientra nella tolleranza, il robot avanza applicando una velocità lineare proporzionale alla distanza, correggendo simultaneamente la rotta con un termine angolare dinamico per mantenere la traiettoria fluida.

Il passaggio tra i waypoint avviene in continuità quando il robot entra nel raggio di tolleranza (dist\_threshold) del punto corrente, garantendo un movimento armonico senza stop superflui, a meno che la geometria del percorso non imponga una curva a gomito che riattiva lo stato di rotazione sul posto.

## 2.5 TF

Per garantire la coerenza spaziale tra i dati sensoriali e la navigazione, il sistema sfrutta il modulo **TF2** per mantenere e aggiornare in tempo reale l'albero delle trasformazioni coordinate (TF Tree). All'interno del nostro progetto, i sistemi di riferimento non sono entità isolate, ma sono legati gerarchicamente per permettere al robot di comprendere la propria posizione rispetto all'ambiente globale. Possiamo distinguere due tipologie di relazioni spaziali:

* **Trasformate statiche (Intra-Robot):** Sono le relazioni immutabili, definite dalla geometria fisica del robot. Un esempio è la trasformata tra il centro del robot (base\_link o base\_footprint) e il sensore laser (lidar\_link). Poiché il Lidar è imbullonato allo chassis, la sua posizione relativa non cambia mai; questa trasformazione viene pubblicata una sola volta dallo StaticTransformBroadcaster per dire al sistema che i dati del laser provengono da un punto traslato e ruotato rispetto al centro di controllo.
* **Trasformate dinamiche (World-to-Robot):** Qui risiede la complessità maggiore. Il controller differenziale pubblica costantemente la trasformata tra **odom** (il punto di partenza locale) e **base\_link** (il robot), basandosi sulle ruotazioni delle ruote; questa stima è fluida ma soggetta a drift nel tempo. Per correggere questo errore senza causare "salti" discontinui nella navigazione, il sistema di localizzazione non sposta il robot, ma pubblica una trasformata correttiva tra **map** (il mondo fisso) e **odom**. In pratica, l'origine del sistema odometrico viene "fatta scivolare" rispetto alla mappa globale per compensare l'errore accumulato, garantendo che le coordinate del robot risultino corrette rispetto all'ambiente circostante.

Immagine che contiene diagramma, testo, linea

Il contenuto generato dall'IA potrebbe non essere corretto.

## 2.6 Localizzazione

La localizzazione del robot all'interno della mappa globale è gestita dal nodo **AMCL** (Adaptive Monte Carlo Localization), che implementa un Filtro Particellare probabilistico per stimare la posa in un ambiente noto. Il sistema è stato configurato per operare in sincronia con il tempo simulato (use\_sim\_time: true), fondamentale per l'integrazione con Unity, e si aggancia all'albero delle trasformate standard ROS collegando il frame del robot base\_footprint e quello odometrico odom al sistema di riferimento globale map.

Il cuore dell'algoritmo risiede nella gestione della nuvola di particelle, configurata per adattarsi dinamicamente all'incertezza del sistema: il numero di campioni varia tra un minimo di min\_particles: 500 (quando la posizione è certa) e un massimo di max\_particles: 2000 (durante fasi di incertezza o inizializzazione globale). Per ottimizzare il carico computazionale della CPU, il filtro non viene aggiornato a ogni ciclo di clock, ma segue una logica basata sullo spostamento delta: i parametri update\_min\_d: 0.2 e update\_min\_a: 0.2 impongono che il ricampionamento avvenga solo dopo una traslazione di 20 cm o una rotazione di circa 11 gradi (0.2 rad).

Il modello di movimento (fase di predizione) è definito dal parametro odom\_model\_type: "diff", specifico per robot a guida differenziale. L'errore intrinseco dell'odometria è modellato attraverso i quattro parametri di rumore alpha1...alpha4, che definiscono la varianza dell'errore rotazionale e traslazionale generato dal movimento. Nel nostro setup, questi valori sono stati impostati molto bassi (0.01 e 0.02), indicando che il sistema ripone un'elevata fiducia nella precisione degli encoder simulati e introduce solo una minima dispersione gaussiana per prevenire la degenerazione del filtro.

Per quanto riguarda il modello sensoriale (fase di correzione), il nodo sottoscrive il topic scan\_topic: "scan", ma per ridurre il costo computazionale non utilizza tutti i raggi del Lidar, limitandosi a processarne un massimo di max\_beams: 180 distribuiti uniformemente. La probabilità di misura è calcolata tramite un modello a mistura (Mixture Model) pesato dai parametri z:

* **z\_hit: 0.7**: È il peso predominante, assegnato alla probabilità che il laser colpisca un ostacolo reale presente nella mappa (modellato con una gaussiana di deviazione standard sigma\_hit: 0.3).
* **z\_short: 0.1**: Gestisce gli ostacoli dinamici non mappati (es. NPC che passano davanti al robot), che accorciano la lettura prevista.
* **z\_max: 0.1** e **z\_rand: 0.1**: Gestiscono rispettivamente i fallimenti del sensore e il rumore casuale di fondo.

Infine, il risultato della stima viene reso operativo abilitando il tf\_broadcast: true, che pubblica la trasformata correttiva map-odom con una transform\_tolerance: 3.0 secondi, garantendo robustezza anche in presenza di leggere latenze di rete tra il simulatore e lo stack di navigazione.

Immagine che contiene testo, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

## 2.7 Elaborazione dei Segnali Biometrici tramite Filtro di Kalman

Per garantire la robustezza e l'affidabilità del monitoraggio dei parametri vitali (battito cardiaco e pressione sanguigna), non ci siamo limitati a inoltrare i dati grezzi provenienti dai braccialetti, intrinsecamente soggetti a rumore di misura e artefatti di movimento. Abbiamo invece ingegnerizzato un modulo ROS2 dedicato, il BraccialettiManager, che implementa manualmente un **Filtro di Kalman Lineare** utilizzando esclusivamente la libreria NumPy per massimizzare le performance e ridurre le dipendenze esterne.

L'architettura del filtro è stata modellata su uno spazio di stato bidimensionale composto dalla coppia **[Valore, Trend]**. Questa scelta progettuale, basata su un modello fisico a "velocità costante", permette al sistema non solo di stimare il valore attuale del parametro (es. i battiti al minuto), ma anche la sua derivata temporale (la tendenza di crescita o decrescita). La matrice di transizione di stato (F) proietta lo stato nel futuro basandosi sul tempo trascorso (dt) tra due pacchetti successivi, mentre la componente di trend viene limitata fisicamente (tramite np.clip) per evitare che il filtro reagisca eccessivamente a picchi momentanei non fisiologici. Questo approccio matematico consente di "prevedere" il valore atteso prima ancora di analizzare la misura reale.

Il cuore della logica di sicurezza risiede nella gestione delle anomalie, implementata nel metodo process\_signal. Anziché accettare ciecamente ogni dato in ingresso, il sistema confronta la misura grezza ricevuta con la predizione matematica del filtro. Se la discrepanza (errore residuo) supera una soglia di tolleranza dinamica predefinita (adattata in base al ritardo temporale del pacchetto), il dato viene classificato come **outlier** e scartato temporaneamente, privilegiando la stima del modello interno. Tuttavia, per distinguere un artefatto momentaneo (es. un movimento brusco del braccio) da un reale cambiamento fisiologico repentino (es. un inizio di tachicardia), è stato introdotto un meccanismo di persistenza: solo se l'anomalia persiste per un numero consecutivo di campioni superiore a MAX\_OUTLIERS, il filtro interpreta il dato come un effettivo cambio di stato e si "resetta" sul nuovo valore, garantendo così una reattività adeguata alle vere emergenze. Infine, il sistema gestisce autonomamente la disconnessione o la rimozione del dispositivo: se non giungono dati per un intervallo superiore al RESET\_TIMEOUT, lo stato interno dell'ospite viene invalidato, prevenendo l'elaborazione di dati obsoleti.

Immagine che contiene testo, orologio, Accessorio di moda, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

2.8 Infrastruttura di Comunicazione: Pattern RPC su Topic

La gestione della comunicazione tra i nodi del sistema (in particolare tra la logica di controllo, il database Neo4j, il Reasoner e il modulo LLM) ha richiesto l'implementazione di un meccanismo di **chiamata sincrona bloccante**. Sebbene ROS2 offra nativamente i *Services* per questo scopo, abbiamo optato per un'architettura custom basata su **Topic asincroni** (Pattern Request-Response over Topics), implementata nelle classi SincronizzaManager e SincronizzamiTutto.

Questa scelta architetturale, apparentemente controintuitiva, offre vantaggi strategici in termini di disaccoppiamento e scalabilità futura, in particolare per il **Caching**. A differenza dei Service (che sono punto-punto), i Topic operano in modalità *publish-subscribe*: ciò significa che le risposte del server (es. un output complesso dell'LLM o una query pesante al database) sono visibili sul bus di comunicazione globale. Questo permetterebbe, in sviluppi futuri, di inserire nodi di caching che memorizzano le risposte frequenti senza dover modificare né il client né il server, riducendo la latenza e i costi computazionali.

A livello implementativo, la classe generica SincronizzamiTutto astrae la complessità di trasformare uno scambio di messaggi asincrono in una chiamata procedurale bloccante. Il funzionamento si basa sull'uso di **Correlation IDs**:

1. Ogni richiesta viene marcata con un **UUID** univoco e inviata sul topic di request.
2. Immediatamente viene istanziato un oggetto concurrent.futures.Future, che blocca il thread chiamante in attesa del risultato (con gestione del timeout).
3. Quando arriva una risposta sul topic di response, la callback on\_response verifica l'ID del pacchetto JSON. Se l'ID corrisponde a una richiesta pendente, il risultato viene iniettato nel Future corrispondente, sbloccando istantaneamente il thread che era in attesa.

La classe SincronizzaManager invece incapsula le istanze di trasporto e fornisce metodi semantici di alto livello. In questo modo, la logica di business può invocare operazioni complesse come se fossero semplici funzioni locali, ignorando totalmente la complessità della comunicazione distribuita sottostante e la gestione dei messaggi JSON.

# Capitolo 3: Intelligenza Artificiale 2

Questo capitolo illustra l'architettura cognitiva del sistema, focalizzandosi sull'implementazione di un approccio ibrido che integra l'Intelligenza Artificiale Simbolica con i moderni modelli generativi. Verranno descritte le metodologie adottate per la **Rappresentazione della Conoscenza** (Knowledge Representation) tramite ontologie formali e Graph Database, e le strategie di **Ragionamento Automatico** (Reasoning) utilizzate per dedurre nuove informazioni e validare le decisioni del robot. L'obiettivo è mostrare come il sistema garantisca coerenza logica, sicurezza operativa e trasparenza (**Explainability**) in un ambiente dinamico caratterizzato dall'Assunzione del Mondo Aperto (OWA), superando i limiti di determinismo tipici dei soli modelli statistici.

## 3.1 Large Language Model

Nell'ambito del sottosistema di interazione uomo-robot, l'adozione dei Large Language Models (LLM) è stata strategica ma rigorosamente confinata al livello di presentazione, escludendo tassativamente qualsiasi coinvolgimento nei processi decisionali critici o nella pianificazione delle azioni. Nello specifico, è stata integrata l'API di Google Gemini sfruttando il modello gemini-2.5-flash. Il ruolo del modello è limitato alla *Natural Language Generation* (NLG): esso non decide il contenuto informativo, che rimane vincolato alla logica deterministica e sicura della base di conoscenza, bensì si occupa esclusivamente della forma, trasformando gli output strutturati e i dati grezzi del sistema in risposte discorsive, empatiche e grammaticalmente curate, migliorando drasticamente la user experience senza introdurre rischi di "allucinazioni" operative.

## 3.2 Base di conoscenza e Ontologia

L'architettura della Base di Conoscenza è stata ingegnerizzata disaccoppiando la fase di definizione strutturale (TBox) da quella di popolamento e persistenza delle istanze (ABox), al fine di ottimizzare sia il rigore logico che l'efficienza operativa. Per la modellazione dello schema ontologico è stato adottato **Protégé**, strumento che ha permesso di definire con precisione formale le gerarchie di classi, le proprietà e le restrizioni assiomatiche, garantendo la coerenza semantica e la validazione del modello prima del suo utilizzo. Parallelamente, per la gestione dinamica dei dati e l'iniezione delle istanze nel sistema, si è optato per l'utilizzo di **Neo4j**; questa scelta strategica consente di traslare la struttura logica dell'ontologia in un database a grafo nativo ad alte prestazioni, capace di gestire volumi elevati di nodi e relazioni con tempi di latenza ridotti. Tale approccio ibrido coniuga l'espressività inferenziale tipica del Web Semantico, garantita dalla progettazione in Protégé, con la scalabilità e la velocità di accesso ai dati offerte da Neo4j, requisiti indispensabili per supportare i processi decisionali del robot in tempo reale.

Immagine che contiene testo, diagramma, schermata, Piano

Il contenuto generato dall'IA potrebbe non essere corretto.

### 3.2.1 Rappresentazione Ontologia

|  |  |  |  |
| --- | --- | --- | --- |
| Classe | Sottoclasse | Attributi | Relazioni |
| Persona | Ospite:  OspiteInStatoDiAllerta OspiteInStatoChiamataSpecialista | nome  cognome  eta  lingua  bpm\_attuale  pressione\_min\_attuale  pressione\_max\_attuale  soglia\_bpm\_anomala  soglia\_bpm\_allerta | (Ospite)-[:EFFETTUA]->(Prenotazione)  (Ospite)-[:SEGNALA]->(Guasto)  (Ospite)-[:AVVERTE]->(Anomalia)  (Ospite)-[:SOFFRE\_DI]->(Patologia)  (Ospite)-[:HA\_SINTOMO]->(Sintomo)  (Ospite)-[:HA\_INTERESSE]->(Interesse) |
|
| Specialista | nome  cognome  numero\_telefono  specialita |  |
| Robot |  | nome\_robot | (Robot)-[:NOTIFICA]->(Specialista) |
| EventoLocale | EventoCitta | nome\_evento\_locale  data\_ora\_evento\_locale |  |
| EventoMare |
| EventoMontagna |
| Evento | EventoConsigliabile | nome\_evento  data\_ora\_evento | (Evento)-[:INVIATO\_A]->(Ospite) |
| EventoNonConsigliabile |
| PrevisioneMeteo |  | data\_ora\_meteo  condizione  gradi |  |
| Prenotazione |  | data\_inizio  data\_fine | (Prenotazione)-[:ASSOCIATA\_A]->(Stanza) |
| Luogo | Stanza | x  y  theta  nome\_stanza |  |
| Hall | x  y  theta |
| CoffeeRoom |
| Guasto |  | tipo\_oggetto\_guasto |  |
| OggettoGuastabile | Stufa | tipo\_oggetto\_guastabile  temperatura\_impostata  temperatura\_stanza  tempo\_acceso  tempo\_acceso\_soglia | (OggettoGuastabile)-[:CONTENUTO\_IN]->(Stanza) |
| Lavandino | tipo\_oggetto\_guastabile |
| Phon |
| Patologia | Cardiopatia |  |  |
| Ipertensione |
| Sintomo | SintomoMuscolare | nome\_sintomo |  |
| SintomoRespiratorio |
| Interesse | Citta | nome\_interesse |  |
| Mare |
| Montagna |

## 3.3 Explainability

La strategia di Explainability del sistema è stata concepita per risolvere le ambiguità intrinseche della **Open-World Assumption (OWA)** tipica dei reasoner semantici, dove l'assenza di un'informazione non implica la sua falsità ma solo uno stato di non-conoscenza. Per garantire decisioni sicure e trasparenti, abbiamo adottato un approccio di **modellazione esplicita dei vincoli**: le condizioni di validità (permessi) e di rischio (divieti) sono modellate nell'ontologia come classi positive distinte (es. EventoConsigliabile vs EventoNonConsigliabile). In questo modo, il motore inferenziale non deve "indovinare" la negazione, ma deduce attivamente la presenza di vincoli ostativi.

L'implementazione di questo paradigma avviene attraverso una pipeline a due stadi orchestrata dal nodo ROS2 SpiegamiTutto. La prima fase, gestita dal metodo logic\_crea\_ontologia\_istanze, si occupa dell'**istanziazione contestuale on-the-fly**: il sistema estrae dal Graph Database (Neo4j) solo il sottografo rilevante per l'interazione corrente (l'ospite, le sue patologie, gli eventi candidati) e lo transcodifica dinamicamente in un file OWL temporaneo usando la libreria rdflib. Questo passaggio trasforma i nodi e le relazioni del database in NamedIndividual e ObjectProperty semantici, colmando il divario tra la persistenza dei dati e la logica formale.

La seconda fase è il cuore dell'Explainability e viene delegata a un modulo esterno sviluppato in Java, invocato tramite subprocesso per sfruttare la robustezza delle librerie **OWLAPI** e **Openllet**. La scelta di Java è dettata dalla necessità di utilizzare la classe DefaultExplanationGenerator, uno strumento avanzato non disponibile in ambiente Python, capace di tracciare la catena causale delle inferenze. Il codice Java non si limita a verificare se un assioma è vero (isEntailed), ma esegue un'iterazione mirata sulle classi target. Per ogni istanza, il reasoner tenta di classificarla sia nelle categorie positive che in quelle negative; qualora l'inferenza abbia successo, il generatore di spiegazioni estrae l'insieme esatto di assiomi e regole SWRL che hanno portato a quella conclusione. Il risultato è un output JSON strutturato che mappa ogni entità non solo al suo stato logico, ma alla giustificazione formale di tale stato. Questo payload viene infine restituito al controller Python, che applica la policy di sicurezza: inibisce l'azione se esistono spiegazioni nella categoria dei divieti e propone l'azione se esistono evidenze nella categoria dei permessi, garantendo così che ogni comportamento del robot sia non solo corretto, ma pienamente intellegibile dall'utente.

Immagine che contiene testo, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.

## 3.4 Implementazione scenari

### 3.4.1 Scenario A

**Descrizione:**

In questo primo scenario abbiamo modellato la fase di ingaggio iniziale e la profilazione dinamica dell'utente. Il sistema gestisce il riconoscimento dell'ospite e il conseguente adattamento linguistico, avviando un dialogo mirato all'estrazione di interessi e preferenze non deducibili dai soli dati di prenotazione. Tali informazioni vengono iniettate in tempo reale nella base di conoscenza e immediatamente elaborate dal reasoner per filtrare le attività disponibili; il robot è quindi in grado di formulare suggerimenti iper-personalizzati, verificandone la fattibilità rispetto a vincoli ambientali e temporali, e gestendo proattivamente eccezioni complesse come la notifica di eventi riprogrammati a causa di allerte meteo.

**Implementazione:**

La gestione dello scenario di accoglienza è stata incapsulata all'interno della classe InteragisciScenarioA, la quale implementa una macchina a stati finiti per gestire la sequenzialità dell'interazione (dall'ingresso dell'ospite fino al congedo). La logica di controllo non si limita a seguire un copione predefinito, ma coordina dinamicamente le chiamate ai servizi cognitivi, al database a grafo e al motore inferenziale.

Per quanto concerne il modulo preliminare di identificazione della lingua, implementato nel metodo rileva\_lingua, abbiamo scartato l'adozione di librerie standard basate su modelli statistici n-grams (come langdetect). In casi d'uso con input brevi, come una frase di presentazione *"My name is Peppe Rossi"*, i classificatori statistici tendono a etichettare erroneamente l'input come "Italiano" a causa del peso preponderante delle feature testuali del nome proprio rispetto alle parole funzionali inglesi. Per garantire un'esperienza utente deterministica, abbiamo optato per un approccio basato su **Keyword Spotting con dizionari controllati**: il sistema analizza l'input confrontando i token con vocabolari precaricati (italiano e inglese) e assegna la lingua basandosi su un punteggio di frequenza, garantendo precisione assoluta in questo specifico contesto.

Una volta stabilita la lingua e recuperati i dati di prenotazione tramite query Cypher al database Neo4j, il sistema avvia la fase di profilazione. Qui adottiamo un approccio ibrido: il metodo rileva\_interesse utilizza il Large Language Model per l'estrazione semantica dell'interesse dal linguaggio naturale (es. la frase *"mi piace camminare in montagna"* viene ricondotta alla parola *"trekking"*). Successivamente, il sistema valida questo output mappandolo rigorosamente a una classe ontologica esistente tramite la funzione trova\_classe\_da\_sinonimo, che sfrutta le alt label dello schema ontologico per ricondurre il termine allo specifico concetto formale. Questo passaggio è fondamentale per evitare di inserire nel Knowledge Graph dati non strutturati o ambigui.

Il cuore decisionale dello scenario risiede nel metodo suggerisci\_evento\_locale, che concretizza la strategia di Explainability e gestione OWA. Il sistema costruisce prima un sottografo di contesto recuperando da Neo4j l'ospite, le sue patologie, i nuovi interessi acquisiti e gli eventi compatibili per data e meteo (recupera\_dati\_per\_suggerimento). A questo punto interviene il nodo Ros2 SpiegamiTutto, che funge da **ponte architetturale tra gestione dati e logica formale**: per evitare un reasoning computazionalmente insostenibile sull'intero database, i dati contestuali vengono serializzati dinamicamente in un file OWL temporaneo (trasformando i nodi Neo4j in individui ontologici tramite rdflib) e processati da un modulo Java esterno, invocato appositamente per sfruttare librerie inferenziali robuste non disponibili in ambiente Python.

Analizzando il JSON restituito da questo processo, il codice applica la policy di sicurezza: verifica la presenza di assiomi nella chiave EventoNonConsigliabile per applicare eventuali veti (es. conflitti medici o ambientali) e, solo in loro assenza, valuta la chiave EventoConsigliabile per formulare la proposta. In questo modo, il robot è in grado di verbalizzare all'utente non solo il suggerimento, ma anche le motivazioni specifiche, garantendo trasparenza totale prima di chiudere l'interazione.

**Schema:**

**Immagine che contiene testo, diagramma, Piano, Disegno tecnico

Il contenuto generato dall'IA potrebbe non essere corretto.**

### 3.4.2 Scenario B

**Descrizione:**

Questo scenario è dedicato alla gestione delle richieste di assistenza tecnica e manutenzione all'interno della struttura. Il flusso operativo viene innescato da una segnalazione dell'ospite che richiede l'intervento del robot in camera per un malfunzionamento.

Una volta arrivato in camera, il robot chiede quale è l’oggetto malfunzionante, dopo aver ricevuto la risposta la elabora e pone all’ospite delle domande per capire se è necessario l’intervento immediato dello specialista. Se il riscontro è positivo allora il robot notifica lo specialista, in caso contrario pone comunque la possibilità di chiamare lo specialista qualora l’ospite lo desideri.

**Implementazione:**

**Schema:**

Immagine che contiene testo, diagramma, Piano, Disegno tecnico

Il contenuto generato dall'IA potrebbe non essere corretto.

### 3.4.3 Scenario C

**Descrizione:**

Questo scenario rappresenta il livello più alto della gerarchia di sicurezza ed è dedicato alla gestione delle criticità sanitarie e al primo soccorso informativo. La peculiarità architetturale di questo modulo risiede nella sua doppia modalità di attivazione, progettata per coprire sia le richieste esplicite che le anomalie latenti. In modalità **reattiva**, il robot si attiva su comando diretto dell'ospite tramite la pressione del pulsante di emergenza sul braccialetto smart; in modalità **proattiva**, invece, il sistema agisce autonomamente qualora il modulo di filtraggio biometrico (Filtro di Kalman) rilevi un'anomalia persistente o un trend preoccupante nei parametri vitali, anticipando una potenziale crisi prima che questa diventi sintomatica.

Una volta raggiunto l'ospite, il robot avvia una procedura di anamnesi preliminare in linguaggio naturale, raccogliendo informazioni sui sintomi percepiti per arricchire il quadro clinico. La logica decisionale per il coinvolgimento dello specialista umano segue un protocollo di sicurezza prudenziale: l'allerta viene inoltrata automaticamente se l'analisi semantica dei sintomi suggerisce una condizione di rischio, ma il sistema garantisce sempre l'override manuale, procedendo alla chiamata anche qualora sia l'ospite stesso a richiederlo esplicitamente, indipendentemente dalla valutazione algoritmica della gravità. Infine, è previsto un livello di intervento critico (bypass dell'interazione) nel caso in cui i parametri vitali superino soglie di rischio estremo, innescando una notifica immediata ai soccorsi.

**Implementazione:**

Lo Scenario C costituisce il vertice della gerarchia di sicurezza del sistema, progettato per gestire le criticità sanitarie attraverso un'architettura a doppia attivazione che combina reattività e proattività. L'ingresso nella macchina a stati finiti può avvenire su richiesta esplicita dell'ospite, tramite pressione del pulsante di soccorso (modalità reattiva), oppure in modo autonomo qualora il nodo BraccialettiManager rilevi un'anomalia persistente nei parametri vitali (modalità proattiva). In quest'ultimo caso, il rilevamento è affidato a un Filtro di Kalman Lineare che processa in tempo reale i flussi di dati (frequenza cardiaca e pressione), filtrando il rumore di misura e identificando trend patologici; solo al superamento di soglie critiche convalidate statisticamente, il sistema innesca l'intervento del robot, il quale avvia l'interazione fornendo immediatamente una spiegazione contestuale ("Ho rilevato un'emergenza") e verificando lo stato di coscienza dell'utente con un timeout di sicurezza di 30 secondi.

La successiva fase di anamnesi adotta un approccio neuro-simbolico: i sintomi descritti dall'ospite in linguaggio naturale vengono estratti semanticamente tramite LLM, ma vengono validati e iniettati nel Knowledge Graph solo se mappabili a classi ontologiche esistenti, garantendo la coerenza della base di conoscenza. Il cuore decisionale risiede nell'interazione con il reasoner: il sistema istanzia temporaneamente il quadro clinico aggiornato e interroga l'ontologia per determinare la necessità di intervento medico. Tuttavia, per mitigare i rischi intrinseci dell'approccio logico in scenari incerti (dove il reasoner potrebbe non dedurre pericoli per mancanza di assiomi specifici), è stato implementato un meccanismo di Human-in-the-loop: qualora l'inferenza non suggerisca l'allerta ma l'ospite la richieda comunque, il sistema esegue un override della logica algoritmica, procedendo incondizionatamente alla notifica dello specialista e registrando l'evento come disaccordo tra valutazione macchina e percezione umana.

**Schema:**

Immagine che contiene testo, diagramma, Piano, Disegno tecnico

Il contenuto generato dall'IA potrebbe non essere corretto.