

# Realizzazione di un sistema per la pulizia automatica basato su Internet of Robotic Things

Andrea De Lorenzis<sup>1\*</sup>, Emanuele Lattanzi<sup>2</sup>

## Sommario

L'Internet of Robotic Things (IoRT) rappresenta l'evoluzione dell'Internet of Things (IoT), in cui i robot si integrano con una rete di sensori, dispositivi e servizi cloud, formando una sinergia che consente di superare i limiti delle singole unità robotiche, estendendo il loro potenziale grazie a un ecosistema distribuito di dati e intelligenza. In questo progetto è stato preso come caso studio un robot aspirapolvere commerciale economico, inizialmente privo di funzionalità avanzate. In una prima fase, il robot è stato potenziato con l'obiettivo di incrementarne le capacità robotiche di base, quali percezione, controllo e adattamento al contesto. Successivamente, il robot è stato integrato in un ambiente arricchito di sensori esterni, trasformandolo così in un nodo attivo in un'architettura IoRT. Il risultato è una piattaforma capace non solo di eseguire la pulizia in modo più autonomo ed efficiente, ma anche di coordinare le proprie decisioni sulla base di informazioni provenienti dall'ambiente circostante.

## Keywords

Internet of Robotic Things — Human Activity Detection — Context-Aware Robotics

<sup>1</sup> Laurea Magistrale in Informatica Applicata, Università degli Studi di Urbino Carlo Bo, Urbino, Italia

<sup>2</sup> Docente di Programmazione per l'Internet of Things, Università degli Studi di Urbino Carlo Bo, Urbino, Italia

\*Corresponding author: a.delorenzis@campus.uniurb.it

## Introduzione

Negli ultimi anni, la diffusione dell'Internet of Things (IoT) ha reso possibile la creazione di ambienti intelligenti capaci di raccogliere e condividere dati in tempo reale. Parallelamente, il campo della robotica di servizio ha visto un crescente interesse verso robot in grado di svolgere compiti quotidiani in modo autonomo, con applicazioni che spaziano dall'assistenza domestica alla logistica. L'integrazione di questi due domini ha portato alla nascita dell'Internet of Robotic Things (IoRT), un paradigma in cui i robot non solo interagiscono con l'ambiente circostante, ma cooperano con una rete distribuita di sensori e dispositivi per migliorare le proprie capacità di percezione, decisione e azione [1].

Tuttavia, molti robot commerciali destinati all'uso domestico, come i comuni robot aspirapolvere, non sono pensati per essere integrati in ambienti IoRT con una vasta gamma di sensori. Inoltre, i robot più economici presentano forti limitazioni in termini di autonomia, adattabilità e controllo, che li rendono poco efficaci anche negli ambienti più semplici. Quelli di fascia alta invece, nonostante siano dotati di maggiori capacità hardware e autonomia, sono poco accessibili, avendo spesso delle interfacce proprietarie che non permettono l'estensione delle funzionalità software senza complesse modifiche al firmware o all'hardware, e ciò rende impossibile interfacciarsi a dispositivi esterni che non siano il cloud proprietario o poche altre applicazioni. Tutti questi fattori riducono l'efficienza e limitano il loro potenziale all'interno di scenari IoRT.

L'obiettivo di questo lavoro è dimostrare come un robot di uso commerciale di fascia bassa possa essere integrato all'interno di un sistema IoRT, senza la necessità di modificare internamente il dispositivo, al fine di creare una sinergia tra robotica e IoT che possa contribuire alla realizzazione di una pulizia più intelligente di ambienti di vario tipo.

In questo progetto si propone un approccio che affronta il problema su due livelli complementari. In una prima fase, un robot aspirapolvere di tipo commerciale di fascia bassa è stato potenziato per migliorarne le capacità di percezione e controllo, così da renderlo più autonomo ed efficace nelle operazioni di pulizia. Il robot è stato reso controllabile tramite la riproduzione di segnali infrarossi generati dal telecomando originale e potenziato dal punto di vista robotico tramite un elaboratore di bordo e un sensore LiDAR. La parte software è stata sviluppata utilizzando il framework ROS (Robot Operating System) e diverse librerie al suo interno per lo svolgimento di guida autonoma tramite algoritmi di SLAM (Simultaneous localization and mapping). Successivamente, il robot è stato inserito in un ambiente arricchito da sensori esterni, in grado di stimare tra le altre cose la presenza e l'attività degli occupanti, trasformandolo così in un nodo di un ecosistema IoRT. In questo modo, le decisioni di pulizia vengono influenzate non solo dalle capacità intrinseche del robot, ma anche dalle informazioni contestuali fornite dall'ambiente. Infine, l'operatività del robot e l'interazione con i sensori è stata testata prima in ambiente di simulazione e poi in un ambiente domestico reale.

Il resto della relazione è organizzato come segue: nella Sezione 1 è descritta la progettazione del robot a livello hardware e software; nella Sezione 2 è descritta l'architettura IoRT proposta; nella Sezione 3 sono riportati gli esperimenti svolti; infine, vengono presentate le conclusioni e le possibili linee di sviluppo futuro.

## 1. Progettazione del sistema robotico

### 1.1 Progettazione hardware

La base di partenza è un robot aspirapolvere commerciale del costo di un centinaio di euro. Il sistema di guida è differenziale, con due ruote motrici posteriori e una ruota sferica anteriore. La scocca ha una forma circolare con un raggio di 15cm. È presente sia un sistema di aspirazione che un sistema lavapavimenti, entrambi con tre livelli di intensità (basso, medio, alto). I motori delle ruote sono dotati di encoder per fornire una stima dello spostamento del robot a partire dalla sua posizione di partenza all'inizio della pulizia. Il rilevamento degli ostacoli si basa su un sensore di contatto che viene attivato quando il robot colpisce un oggetto. La navigazione consiste in una successione di algoritmi di pulizia preimpostati, che non tengono conto in anticipo della conformazione della stanza. Tra le modalità disponibili ci sono: pulizia a zigzag, pulizia circolare in un punto e pulizia del bordo. Il robot è controllabile manualmente tramite un telecomando IR in dotazione o tramite l'app proprietaria, con 4 comandi direzionali. Si possono inoltre decidere il livello di intensità dell'aspirazione e della fuoriuscita dell'acqua. Infine, il robot è dotato di una base di ricarica, a cui può collegarsi da solo se abbastanza vicino ad essa.

Il problema principale di questo robot, ma in generale della maggior parte dei robot di fascia bassa, è la sua limitata capacità di percezione dell'ambiente, che lo rende adatto solo alla pulizia di piccoli spazi. In ambienti più grandi non è in grado di coprire efficacemente tutta l'area per via della mancanza di sensori affidabili che gli permettano di localizzarsi all'interno dell'ambiente. L'odometria misurata dagli encoder rotatori non è sufficiente, in quanto questa misura è soggetta a *drifting* nel corso del tempo, sfalsando nel giro di poco la stima di posizione del robot rispetto al suo punto di origine. Per poterlo integrare all'interno di un sistema IoRT, il robot deve conoscere precisamente la sua posizione nel mondo e deve poter raggiungere i luoghi che necessitano di pulizia secondo il piano deciso dal cloud.

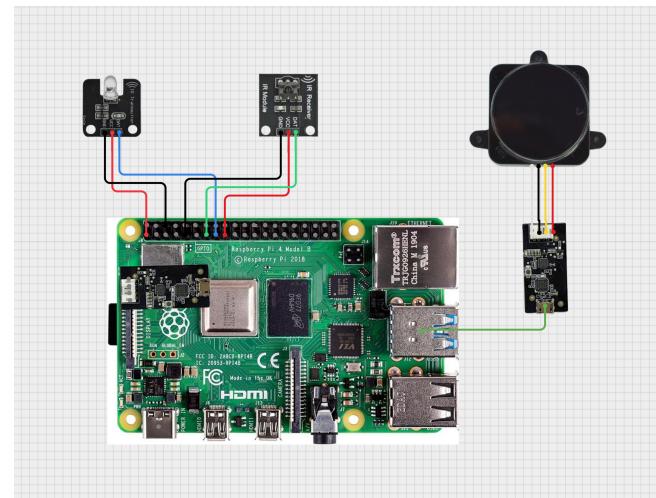
Per ovviare a questo problema sono state fatte diverse aggiunte all'hardware del robot, senza andare a toccare le sue componenti interne. Nel dettaglio, queste sono le componenti aggiuntive che sono state montate a bordo del robot:

- trasmettitore e ricevitore a infrarossi (IR), usati per comandare il robot esternamente riproducendo gli stessi comandi IR del telecomando in dotazione;
- LiDAR LD19, un sensore LiDAR 2D per creare una mappa dell'ambiente e localizzare il robot al suo interno.

È collegato ad una porta usb del Raspberry Pi tramite un adattatore USB to UART/TTL;

- ESP32-CAM, una piccola videocamera economica dotata di ESP32, usata per ottenere un POV del robot;
- Raspberry PI 4, il cervello del robot. Elabora i dati sensoriali a bordo, li invia ad un edge computer per la pianificazione e controlla l'attuazione del robot tramite il trasmettitore IR;
- powerbank con output 5V/3A per fornire alimentazione a tutti i componenti.

In figura 1 è riportato lo schema circuitale delle componenti aggiuntive montate a bordo del robot.

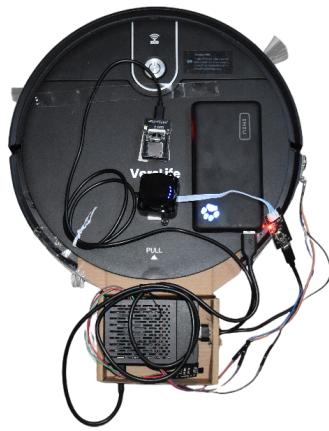


**Figura 1.** Schema circuitale delle componenti aggiuntive del robot.

In figura 2 è possibile osservare il robot visto dall'alto. Il Raspberry Pi è alloggiato in un supporto di cartone al di sotto della scocca superiore del robot, in modo da non coprire il raggio del LiDAR. Al centro è presente il LiDAR e davanti è montata la telecamera ESP32-CAM. Incollato lateralmente e collegato ai pin del Raspberry Pi tramite dei fili, c'è il trasmettitore IR. L'alimentazione viene fornita al Raspberry Pi e agli altri componenti tramite una powerbank da 15 watt.

### 1.2 Progettazione software

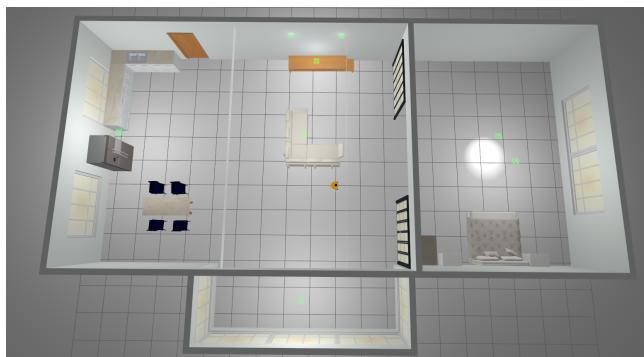
Per la parte software del robot è stato utilizzato ROS2 (Robotic Operating System 2), un framework open source per lo sviluppo di applicazioni robotiche. ROS permette di gestire la comunicazione tra sensori, attuatori e algoritmi di elaborazione in maniera modulare e scalabile, facilitando l'integrazione di componenti software diversi. Rispetto alla prima versione, ROS2 offre miglioramenti significativi in termini di gestione della comunicazione in tempo reale, supporto multi-piattaforma e distribuzione su sistemi embedded. L'architettura software proposta è stata sviluppata prendendo ispirazione da tutorial e guide pratiche, tra cui una serie di tutorial pubblicati da *Articulated Robotics* [2].



**Figura 2.** Visuale dall'alto del robot con componenti aggiuntive.

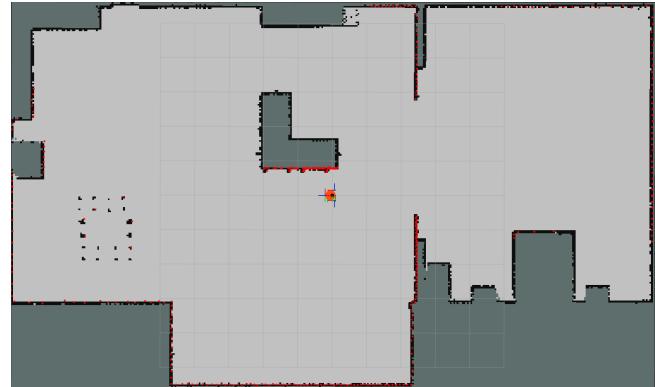
ROS2 è basato su un'architettura modulare a nodi, in cui ogni nodo rappresenta un processo indipendente che esegue una specifica funzione del sistema robotico. I nodi comunicano tra loro attraverso topic, messaggi pubblicati dai publisher e ricevuti dai subscriber, oppure tramite servizi e azioni per comunicazioni sincrone. Inoltre, ROS2 consente l'integrazione di controller che gestiscono l'attuazione dei motori e l'esecuzione di algoritmi complessi, permettendo al robot di reagire in tempo reale ai dati sensoriali e agli stimoli dell'ambiente. La separazione in nodi modulari e la gestione tramite topic semplifica l'aggiunta di nuove funzionalità, come sensori aggiuntivi, algoritmi di pianificazione o moduli di navigazione autonoma.

**Simulazione con Gazebo** La fase di test e sviluppo è stata supportata da Gazebo, simulatore 3D solitamente utilizzato in combinazione con ROS2. È stato creato un mondo virtuale in cui il robot può navigare e testare SLAM, navigazione e copertura. In figura 3 è visualizzato il mondo creato nell'ambiente di simulazione Gazebo. I sensori e gli attuatori del robot sono simulati utilizzando diversi plugin di ROS2 e sono stati configurati per essere il più vicini possibile alla realtà.



**Figura 3.** Mondo simulato tramite il software Gazebo.

In aggiunta al software di simulazione, ROS2 viene spesso combinato con Rviz, un software di visualizzazione che permette di visionare lo stato delle varie componenti del robot durante la sua operatività, come gli scan del lidar, la mappa generata o le immagini visualizzate dalla telecamera di bordo. Mentre Gazebo crea la simulazione del mondo, Rviz è utilizzato per sapere ciò che il robot pensa del mondo che lo circonda. Questo stesso strumento di visualizzazione viene usato nella realtà al di fuori della simulazione.



**Figura 4.** Visuale dall'alto in Rviz. Sono visibili il robot al centro, la mappa generata e i punti provenienti dalle scansioni del LiDAR.

**Localizzazione e SLAM** Lo SLAM (Simultaneous Localization and Mapping) è una tecnica che permette a un robot di costruire una mappa dell'ambiente mentre si localizza al suo interno, utilizzando i dati provenienti dai sensori. Solitamente, viene svolto utilizzando un sensore LiDAR (Light Detection and Ranging), una tecnologia che utilizza impulsi laser per misurare la distanza dagli oggetti circostanti, consentendo di ricostruire una rappresentazione 2D o 3D dell'ambiente.

Per la mappatura dell'ambiente e la localizzazione del robot, è stato utilizzato SLAM Toolbox, che permette di generare mappe 2D in tempo reale a partire dai dati LiDAR. La navigazione autonoma sulla mappa è gestita tramite Nav2, un'altra popolare libreria dell'ambiente ROS, la quale calcola i percorsi e genera comandi di velocità per il robot in modo dinamico.

Il LiDAR LD19 comunica con il Raspberry Pi tramite il driver/nodo fornito dal produttore, che pubblica continuamente le scansioni sul topic appropriato di ROS2. Questi dati sono utilizzati sia dal modulo SLAM, che dai sistemi di navigazione e pianificazione del percorso.

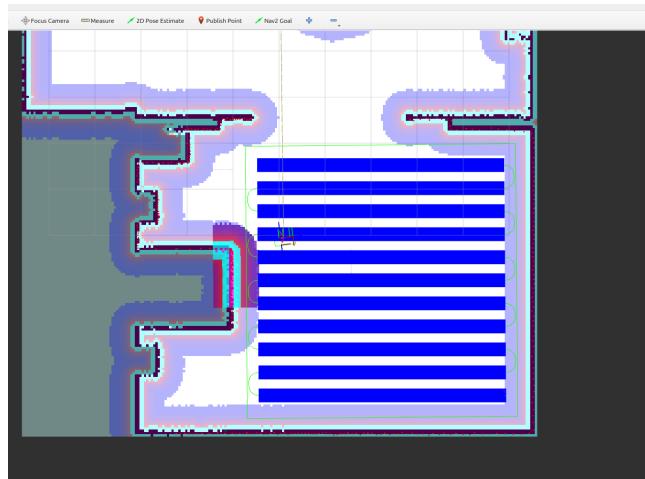
In figura 4 si può vedere una visuale dall'alto della mappa generata dagli algoritmi di SLAM. I contorni neri sono stati creati dallo SLAM, mentre i pallini rossi rappresentano ciò che il LiDAR vede in quella zona della mappa.

**Controllo direzionale discreto** Per adattare i comandi continui generati da Nav2 ai quattro comandi discreti del telecomando IR del robot, è stato creato un nodo ROS denominato step controller. Questo nodo utilizza una macchina a stati finiti

per rimappare i comandi di Nav2 in azioni discrete (avanti, indietro, sinistra, destra), garantendo un controllo coerente e stabile del robot sia in simulazione sia nel mondo reale. Per ogni stato (es. FORWARD) lo step controller invia dei segnali IR ad una frequenza prefissata che permette di ottenere un movimento fluido per il robot.

**Coverage Path Planning (CPP)** Il Coverage Path Planning (CPP) è un problema di pianificazione del percorso che mira a coprire completamente un'area specificata, utilizzando uno o più veicoli. Questo approccio è fondamentale in applicazioni come la pulizia automatica, la sorveglianza e l'agricoltura, dove è essenziale esplorare ogni parte dell'ambiente in modo sistematico ed efficiente. La copertura completa dell'area da pulire è stata implementata usando OpenNav-Coverage, una libreria dell'ecosistema ROS per generare percorsi ottimali. Questo modulo assicura che l'intera zona selezionata venga esplorata senza lasciare zone non pulite. La libreria sfrutta a sua volta un'altra libreria chiamata Fields2Cover, pensata per la copertura dei campi agricoli ma utilizzabile anche in altri contesti, come quello dei robot aspirapolvere.

In figura 5 è visibile un esempio di copertura generata dalla libreria sulla base di una zona fornita in input, all'interno dello strumento di visualizzazione Rviz.



**Figura 5.** Esempio di un percorso di copertura generato da OpenNav-Coverage.

**Integrazione su robot reale** Per l'esecuzione sul robot fisico, è stato creato un file di lancio dedicato che sostituisce lo step controller simulato con quello reale. I comandi discreti generati dal nodo step controller node vengono inviati al trasmettitore IR, che li trasmette al robot. La ricezione e l'invio dei comandi dal Raspberry Pi sono gestiti tramite le librerie LIRC (Linux Infrared Remote Control) e PIGPIO. La prima è stata utilizzata per leggere i segnali IR del telecomando salvandoli in formato raw, mentre la seconda per pilotare il trasmettitore IR e inviare i comandi grezzi salvati.

In questo progetto, il sistema ROS2 è distribuito su due dispositivi: il Raspberry Pi a bordo del robot e un edge compu-

ter collegato alla stessa rete locale. Sul Raspberry Pi vengono eseguiti solo i nodi strettamente necessari al funzionamento locale del robot: il nodo StepController, il robot state publisher e il nodo del LiDAR, responsabili rispettivamente del controllo discreto, della pubblicazione dello stato dei joint del robot e della raccolta dei dati del sensore. Tutti gli altri nodi computazionalmente più pesanti, come Nav2, SLAM Toolbox e OpenNav-Coverage, vengono eseguiti sull'edge computer. Il Raspberry Pi comunica in tempo reale con questa macchina tramite ROS2, ricevendo i comandi di movimento calcolati dall'edge computer e inviando i dati sensoriali (scan del LiDAR e stato del robot) per la mappatura, la pianificazione dei percorsi e la navigazione autonoma. Questa configurazione permette di ridurre il carico computazionale a bordo del robot, ma aggiunge una latenza tra edge computer e robot, e può risultare a volte in dei cali di connessione soprattutto nelle zone meno raggiunte dal WiFi.

## 2. Architettura del sistema IoRT

Per arricchire il comportamento del robot e permettere una pianificazione più intelligente delle attività di pulizia, è stato realizzato un sistema IoRT (Internet of Robotic Things) che integra una rete di sensori ambientali distribuiti. L'obiettivo è fornire al robot informazioni di contesto, come la presenza di persone, le condizioni di luce o altri dati ambientali, in modo da generare piani di navigazione personalizzati.

### 2.1 Sensoristica ambientale

I sensori sono stati collegati, tramite il supporto di una breadboard, ad un microcontrollore ESP32, scelto per la sua connettività Wi-Fi e la semplicità di integrazione con l'infrastruttura IoT. L'ESP32 gestisce i seguenti sensori:

- sensore IR di rilevamento ostacoli;
- sensore di luminosità;
- sensore di temperatura e umidità;
- sensore PIR.

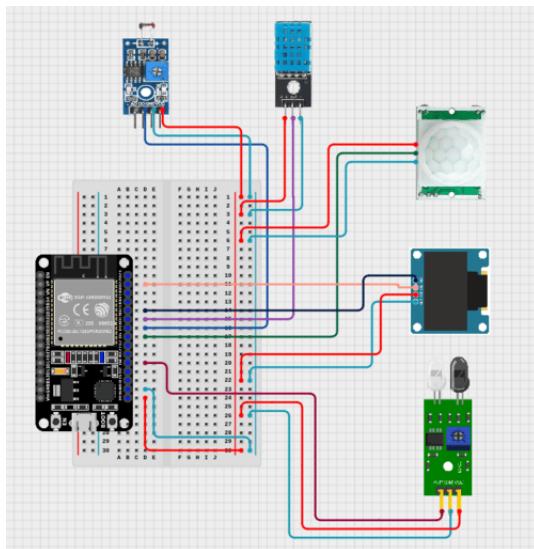
È presente inoltre un piccolo display OLED, per visualizzare alcune informazioni direttamente sulla breadboard. In figura 6 è riportato lo schema circuitale della breadboard con tutte le componenti.

### 2.2 Infrastruttura IoT

L'infrastruttura IoT è stata messa in piedi utilizzando il tool IOTstack, un tool basato su Docker che semplifica la creazione e la gestione di ambienti IoT modulari. IOTstack consente di avviare rapidamente container preconfigurati per servizi comunemente impiegati in scenari IoT [3].

Questi sono i container che sono stati configurati dall'interfaccia del tool:

- Node-RED, per la gestione logica dei flussi di dati e il backend applicativo;



**Figura 6.** Schema circuitale che mostra i collegamenti tra ESP32 e sensori ambientali.

- Mosquitto, un broker MQTT per lo scambio di messaggi tra i vari nodi IoT;
- InfluxDB, per la memorizzazione dei dati sensoriali in un database time-series;
- Grafana, per la visualizzazione tramite dashboard interattive e personalizzabili;
- MariaDB, per la persistenza delle configurazioni impostate tramite l’interfaccia web;
- Nginx, reverse proxy che agisce da punto d’ingresso per la rete privata e serve l’interfaccia web;
- Portainer CE, per la gestione e il monitoraggio dei container in esecuzione.

Tutti i container sono eseguiti all’interno di una rete privata. Il server Nginx agisce sia da web server che da reverse proxy, facendo sì che la rete privata sia accessibile dall’esterno in maniera sicura. I container sono gestiti tramite Docker Compose, uno strumento per definire ed eseguire applicazioni multi container. Le configurazioni dei container sono scritte all’interno del file `docker-compose.yaml`. L’infrastruttura descritta, assieme al nodo robotico, è mostrata nel diagramma di figura 7.

**Flussi di dati** L’ESP32 invia periodicamente i dati al broker MQTT sul topic `/sensors` in formato JSON, con una frequenza di 5 secondi. Per temperatura e umidità, i dati inviati sono gli ultimi valori misurati alla fine del periodo. Per il sensore PIR, viene pubblicato 0 se non è stato rilevato movimento nel periodo, e 1 altrimenti. Per il sensore IR, si invia il numero di rilevamenti all’interno dell’intervallo. Infine, il valore inviato per il sensore di luminosità è l’ultimo misurato alla fine del periodo e vale 1 se la luminosità supera la soglia, 0 altrimenti.

Per gestire i flussi di dati è stato utilizzato Node-RED. Questo è uno strumento low-code per la gestione di flussi, che agisce essenzialmente come un backend per il sistema. Usando questo programma sono stati creati diversi flussi. Tra questi è presente un flusso che legge i dati sensoriali pubblicati dal ESP32 sul topic `/sensor_data`. Il nodo di partenza è un nodo MQTT IN. I dati passano poi attraverso un nodo function che rinomina alcuni campi dell’oggetto JSON e arrivano al nodo InfluxDB, che li inserisce all’interno del database nel measurement chiamato `sensor_data`. In figura 8 è visualizzato questo flusso.

Per implementare la gestione delle regole e delle sessioni di pulizia, è stato creato un flusso ad hoc, raffigurato in figura 9. Questo flusso parte con un nodo `inject` che invia un trigger di attivazione ogni 5 secondi. Il flusso può anche essere fatto partire tramite richiesta HTTP sull’endpoint `/start_plan`. Il nodo successivo crea una query `SELECT` per leggere dal database di configurazioni, ovvero l’istanza MariaDB. La query fa una `select` sulla tabella `session`, per ottenere le sessioni che devono partire. Ogni record è messo in `join` con la tabella `rule`, che contiene le regole da verificare per una certa sessione e zona. Il risultato di questa query è pertanto un array di record che rappresentano le zone per la sessione di pulizia, assieme alle regole associate ad ogni zona, se presenti. Tramite un nodo `split`, l’array viene suddiviso in dei messaggi singoli. Per ciascun messaggio, viene fatta una query al nodo InfluxDB, in modo da ottenere i valori sensoriali e poter fare una comparazione con le soglie impostate nell’eventuale regola. Questo controllo viene fatto nel nodo function successivo, denominato `validate condition`, il quale inserisce nel payload del messaggio un booleano che indica ai nodi successivi se la regola per la zona interessata è soddisfatta oppure no. Successivamente, tutti i messaggi vengono di nuovo uniti assieme tramite un nodo `join`. Ciò che arriva al nodo `check cleaning` è quindi un array di messaggi con informazioni sulle zone da pulire e sul soddisfacimento delle regole associate. Il nodo `check cleaning` produce a questo punto un JSON di output con il piano di pulizia da inviare al robot. Ad esempio:

```
{
  "type": "plan",
  "zones": [
    {
      "zone_id": 2,
      "intensity": 1,
      "frequency": 1,
      "ros_points": [
        ...
      ],
      ...
    },
    ...
  ],
  "ros_base_pos": {
    "x": -1.62,
    "y": -5.11
  }
}
```

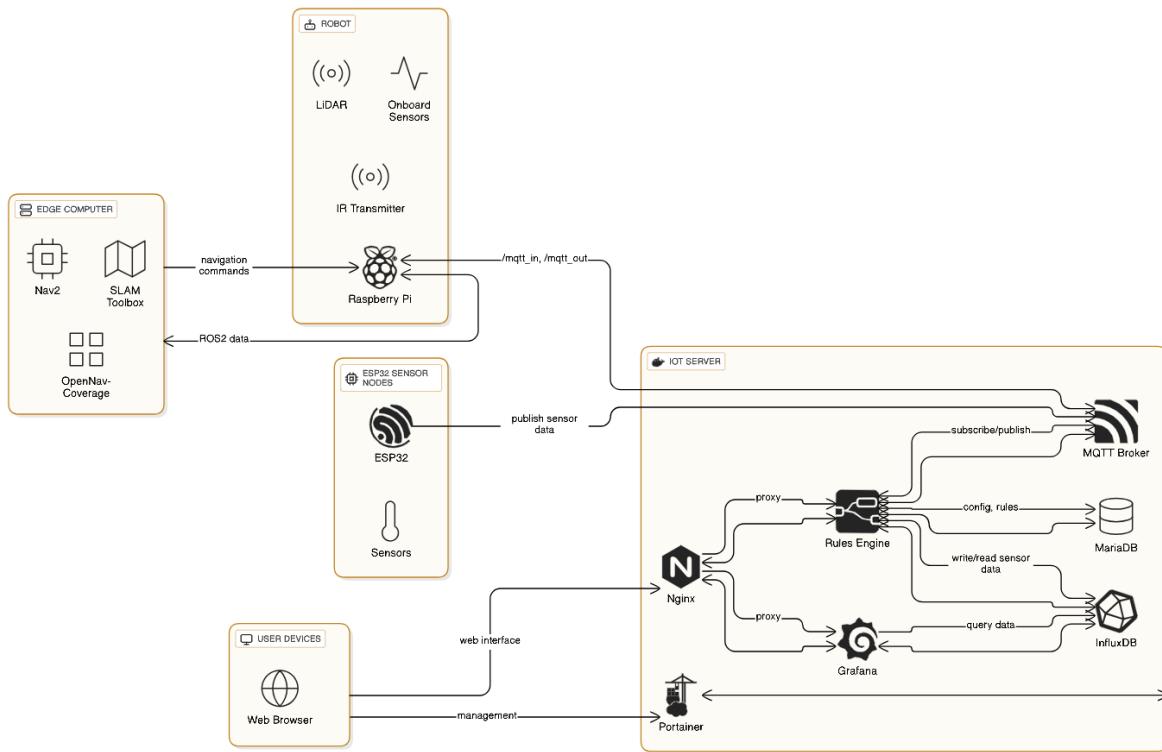


Figura 7. Diagramma dell'infrastruttura IoRT.



Figura 8. Flusso Node-RED di pubblicazione dei dati sensoriali da MQTT a influxDB.

dove *type* indica il tipo del messaggio da inviare (in questo caso "plan"); *zones* è un array di zone, dove ciascun elemento ha un array di punti che definiscono la zona in coordinate ROS, e i campi *intensity* e *frequency*, che vengono valorizzati dal nodo sulla base del controllo fatto precedentemente sulle regole. Nel JSON è presente anche un campo *ros\_base\_pos* che indica la posizione della base di ricarica del robot, da cui parte all'inizio della pulizia, e a cui ritorna alla fine del piano. Infine, tramite un nodo MQTT OUT, viene inviato un messaggio sul topic `/mqtt/in`, a cui il robot è in ascolto. Parallelamente all'invio del messaggio MQTT, viene fatta una update sulla tabella *session* nel database, per portare lo stato della sessione di pulizia da *scheduled* a *started*.

Per implementare tutte le operazioni CRUD sull'interfaccia web, sono anche stati aggiunti dei flussi che implementano semplici endpoint HTTP. Questi accettano richieste, interagiscono con il database MariaDB per selezionare o aggiornare dati, e infine ritornano il risultato al client. In figura 10 è mostrato un esempio per l'endpoint `/rules`.

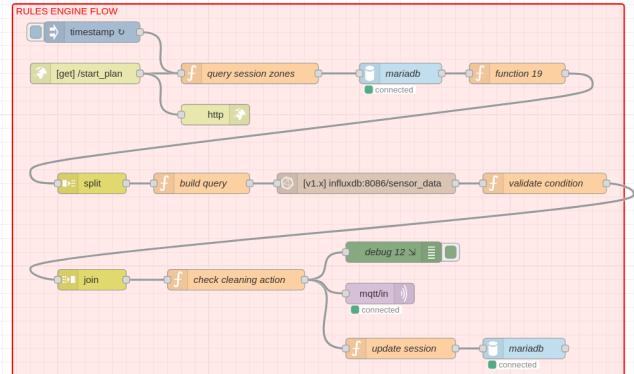
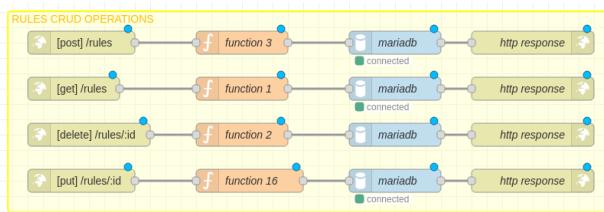


Figura 9. Flusso di verifica delle regole in Node-RED.

**Interfaccia Web** Per poter configurare agevolmente il sistema IoRT è stata sviluppata un'interfaccia web che è composta da tre sezioni principali:

- controllo mappa, consente di visualizzare la mappa creata dal robot e gestire le zone virtuali;
- controllo manuale, consente di guidare il robot manualmente e osservare il feed video ricevuto dalla telecamera;
- gestione sessioni, pagina in cui è possibile configurare i parametri di sistema, pianificare le sessioni di pulizia e configurare le regole.



**Figura 10.** Flussi che implementano le operazioni CRUD sulle regole.

In figura 11 si può osservare la schermata Controllo Mappa dell’interfaccia web con la mappa creata dal robot all’interno dell’ambiente simulato.

### 3. Risultati

Per testare il funzionamento del sistema sono stati realizzati diversi casi d’uso. Di seguito, verrà descritto come questi casi d’uso sono stati implementati sfruttando l’architettura IoRT e successivamente validati in ambiente simulato. I casi sono:

- pianificazione in base alla presenza: se il sensore PIR rileva movimento in una stanza, il robot può rimandare la pulizia in quell’area fino a quando non è libera, evitando così interruzioni durante la pulizia;
- modalità notturna: in caso di bassa luminosità rilevata dal sensore di luce, il robot può avviare un piano di pulizia silenzioso, riducendo l’intensità di pulizia per limitare il rumore;
- adattamento alle condizioni climatiche: leggendo i dati di umidità, il sistema può provare a prevedere una condizione di pioggia. In tal caso, il robot può intensificare la pulizia vicino agli ingressi, dove è più probabile che si accumulino tracce di fango;
- adattamento di pulizia in base al livello di attività: i sensori IR alle porte possono stimare il numero di persone che entrano ed escono dalle stanze. Se il traffico è elevato, il robot può aumentare la frequenza e/o l’intensità delle pulizie.

In figura 12 è visibile la pagina *Gestione Sessioni* dell’interfaccia web, in cui sono stati impostati i dati per implementare i casi d’uso elencati. Ai fini di test, per le regole sono stati impostati dei valori fintizi, tra cui degli intervalli di osservazioni passati nell’ordine di qualche secondo.

Per osservare il funzionamento del sistema sono stati aggiunti dei log degli eventi più importanti con corrispondente istante temporale.

La regola denominata *Motion Check* prevedeva che, in presenza di rilevamenti da parte del sensore PIR in una stanza, la pulizia venisse rimandata fino a quando non si fosse registrata l’assenza di movimento per almeno 5 secondi consecutivi. Per fare ciò, sono state impostate la condizione *LESS\_THAN*, l’operazione *SUM*, un valore soglia pari a 1,

un intervallo pari a 5s e l’azione *START\_CLEANING*. Questa regola dichiara essenzialmente ”se nei 5 secondi passati la somma dei rilevamenti del sensore PIR è minore di uno, allora procedi con la pulizia”. Essendo associata ad una azione di tipo *START\_CLEANING*, se non soddisfatta, questa regola ha l’effetto di bloccare la sessione di pulizia, anche se altre regole sono soddisfatte in quel momento. Durante il test, si è osservato che, fintanto che il sensore rilevava movimento, la pulizia non partiva, ritardando il tempo programmato di inizio della sessione. Non facendo movimenti davanti al sensore per il tempo necessario, la sessione invece è partita. I parametri iniziali di intensità e frequenza avevano correttamente entrambi il valore di default pari a 1. La regola è stata associata alla sessione di pulizia *Afternoon cleaning*, la quale prevedeva la pulizia di tutte le stanze, ovvero delle 3 stanze visibili in figura 11. L’intensità e la frequenza di pulizia delle singole zone è stata decisa dalla altre regole. Al termine, il robot è tornato in autonomia alla base di ricarica.

Per testare la regola *Humidity Check*, associata anch’essa alla sessione di pulizia *Afternoon cleaning*, si è fissato un valore di 50, più basso rispetto al valore ambientale registrato in quel momento, una condizione *GREATER\_THAN*, l’operazione *MEAN*, un intervallo di 5s e l’azione *INTENSITY\_3* (che vuol dire ”se il valore di umidità medio degli ultimi 5s è maggiore di 50, allora imposta l’intensità di pulizia della zona a 3”). Durante il test, si è osservato che, una volta arrivata alla zona ”Entrance”, il robot ha correttamente modificato la propria intensità al valore massimo di 3, ed è ritornato all’intensità di default una volta uscito dalla zona.

Alla stessa sessione si sono inoltre associate le regole *Door IR Check* per implementare il caso d’uso sul passaggio dalle stanze. La regola denominata *Door IR Check (Low)* rappresenta un numero intermedio di passaggi, con un valore soglia da superare di 5 e un’azione associata *INTENSITY\_2*. Le altre due regole per il sensore IR, denominate *Door IR Check (High)*, rappresentano invece un numero alto di passaggi, superiore a 10, e hanno associate le azioni *INTENSITY\_2* e *FREQUENCY\_2* (che significa ”se la regola è soddisfatta pulisci la zona con intensità pari a 2 per due volte di fila”). Sono stati simulati un certo numero di passaggi passando più volte la mano di fronte al sensore e superando il valore soglia di 10. Durante l’esecuzione, si è osservato che il robot, una volta arrivato alla zona, ha cambiato la propria intensità da 1 a 2, e ha svolto due passaggi completi della zona, prima di terminare la pulizia e tornare alla base di ricarica.

Infine, l’ultima regola testata riguarda la modalità notturna, che è stata associata ad una sessione di pulizia separata, denominata *Night cleaning*. La regola è stata impostata in modo che, se l’ultimo valore di luminosità letto fosse pari a zero, il robot sarebbe partito con la pulizia a bassa intensità, in modo da fare poco rumore. Allo scattare della sessione, il sensore misurava un valore low, pertanto la pulizia è partita con intensità bassa. Nel caso di sensore high invece, il robot ha correttamente mantenuto l’intensità di default.



**Figura 11.** Sezione Controllo Mappa dell’interfaccia web.

## 4. Conclusioni

In questo progetto è stato esplorato il paradigma dell’Internet of Robotic Things (IoRT) applicato alla pulizia automatica, dimostrando come l’integrazione di un robot aspirapolvere commerciale con una rete di sensori possa migliorare le sue capacità operative e aumentare la sua utilità.

Partendo da un robot di fascia bassa privo di funzionalità avanzate, si è proceduto al suo potenziamento attraverso l’aggiunta di componenti hardware come un LiDAR, un Raspberry Pi e un sistema di controllo a infrarossi, abilitando così funzionalità di mappatura e navigazione autonoma tramite ROS2. Questa estensione ha dimostrato come fosse possibile prendere il controllo di un robot commerciale chiuso, senza apportare modifiche all’hardware interno o al firmware, step essenziale per poterlo poi integrare come nodo nell’infrastruttura IoT.

Il sistema IoRT è stato poi arricchito da una rete di sensori ambientali (PIR, IR, luminosità, temperatura e umidità) gestiti da un ESP32 e integrati in un’infrastruttura basata su Docker, che comprende Mosquitto, InfluxDB, Grafana e Node-RED. Questa architettura ha permesso al robot di ricevere informazioni contestuali dall’ambiente, consentendo una pianificazione delle attività di pulizia più intelligente e adattativa. L’interfaccia web sviluppata ha fornito un mezzo intuitivo per configurare regole di pulizia basate su eventi ambientali.

I test condotti hanno dimostrato l’efficacia del sistema nel gestire scenari complessi, come la pianificazione della pulizia in base alla presenza umana, l’attivazione di una modalità notturna, l’adattamento alle condizioni climatiche e l’intensificazione della pulizia in aree ad alto traffico. Questi risultati confermano il potenziale dell’IoRT nel trasformare robot semplici in agenti intelligenti, capaci di interagire in modo più autonomo ed efficiente con l’ambiente circostante.

Le limitazioni riscontrate includono la latenza nella comunicazione tra il Raspberry Pi e l’edge computer per l’esecuzione dei nodi ROS2 più pesanti, e la dipendenza dalla copertura Wi-Fi, che può risultare peggiore o assente in alcune zone. Per ovviare a questo problema, il robot dovrebbe montare un computer più potente del Raspberry Pi che gli consenta di eseguire gli algoritmi di navigazione e mapping direttamente a bordo, invece di affidarsi ad un edge computer. Tuttavia, l’utilizzo di un edge computer rappresenta comunque un vantaggio significativo, in quanto consente di delegare i compiti più complessi e risparmiare pertanto le limitate risorse del robot, come batteria, memoria e capacità di calcolo.

Il codice sorgente che implementa l’architettura proposta in questo progetto è disponibile su GitHub: [https://github.com/andreadelorenzis/iort\\_project](https://github.com/andreadelorenzis/iort_project).

Per quanto riguarda gli sviluppi futuri, diverse direzioni potrebbero essere esplorate:

- machine learning per l’analisi del contesto: l’integrazione di algoritmi di machine learning potrebbe consentire al robot di apprendere pattern di comportamento dagli occupanti e dalle condizioni ambientali, ottimizzando ulteriormente i piani di pulizia. Ad esempio, il robot potrebbe prevedere i momenti di maggiore affluenza o le aree più soggette a sporcarsi;
- collaborazione multi-robot: sviluppare la cooperazione tra più robot coordinati dai sensori ambientali, per distribuire i compiti e aumentare l’efficienza;
- utilizzo di protocolli più resistenti: esplorare protocolli di comunicazione più resistenti (es. LoRa, ZigBee, 5G), per ridurre la dipendenza esclusiva dal Wi-Fi e garantire continuità di servizio.

Sessioni										<a href="#">+ Aggiungi sessione</a>
ID	Nome	Robot	Inizio programmato			Stato		Actions		
3	Afternoon cleaning	Vacuum Robot 1	9/9/2025, 5:00:00 PM			scheduled		<a href="#">Modifica</a>	<a href="#">Elimina</a>	
11	Night cleaning	Vacuum Robot 1	9/10/2025, 2:03:00 AM			scheduled		<a href="#">Modifica</a>	<a href="#">Elimina</a>	
Regole										<a href="#">+ Aggiungi regola</a>
Name	Sensor	Session	Robot	Zone	Operation	Condition	Value	Interval	Action	Actions
Humidity Check	humidity	Afternoon cleaning	Vacuum Robot 1	Entrance	MEAN	GREATER_THAN	50.00	2h	INTENSITY_3	<a href="#">Modifica</a> <a href="#">Elimina</a>
Motion Check	pirDetection	Afternoon cleaning	Vacuum Robot 1	Bedroom	SUM	LESS_THAN	1.00	5s	START_CLEANING	<a href="#">Modifica</a> <a href="#">Elimina</a>
Door IR Check (Low)	obstacleDetection	Afternoon cleaning	Vacuum Robot 1	Living room	SUM	GREATER_THAN	5.00	5s	INTENSITY_2	<a href="#">Modifica</a> <a href="#">Elimina</a>
Door IR Check (High)	obstacleDetection	Afternoon cleaning	Vacuum Robot 1	Living room	SUM	GREATER_THAN	10.00	10s	FREQUENCY_2	<a href="#">Modifica</a> <a href="#">Elimina</a>
Door IR Check (High)	obstacleDetection	Afternoon cleaning	Vacuum Robot 1	Living room	SUM	GREATER_THAN	10.00	10s	INTENSITY_2	<a href="#">Modifica</a> <a href="#">Elimina</a>
Luminosity Check	luminosity	Afternoon cleaning	Vacuum Robot 1	Bedroom	LAST	EQUAL	0.00	10s	INTENSITY_1	<a href="#">Modifica</a> <a href="#">Elimina</a>
Sensori										<a href="#">+ Aggiungi sensore</a>
ID	Nome				Actions					
11	humidity				<a href="#">Modifica</a> <a href="#">Elimina</a>					
12	luminosity				<a href="#">Modifica</a> <a href="#">Elimina</a>					
13	pirDetection				<a href="#">Modifica</a> <a href="#">Elimina</a>					
14	obstacleDetection				<a href="#">Modifica</a> <a href="#">Elimina</a>					
15	temperature				<a href="#">Modifica</a> <a href="#">Elimina</a>					
Robot										<a href="#">+ Aggiungi robot</a>
ID	Nome				Actions					
17	Vacuum Robot 1				<a href="#">Modifica</a> <a href="#">Elimina</a>					

**Figura 12.** Tabelle con sessioni e regole impostate per testare casi d'uso.

In conclusione, questo progetto ha dimostrato un approccio efficace per trasformare un semplice robot aspirapolvere commerciale in un componente attivo di un ecosistema IoT, aprendo la strada a soluzioni di pulizia automatica più intelligenti, autonome e contestualmente consapevoli.

[3] Learn Embedded Systems. Raspberry pi iot server tutorial. [https://youtu.be/\\_D02wH16JWQ?si=BnSwNpCwvcmxhXwS](https://youtu.be/_D02wH16JWQ?si=BnSwNpCwvcmxhXwS).

## Riferimenti bibliografici

- [1] David Uchechukwu, Arslan Siddique, Aizhan Maksatbek, and Ilya Afanasyev. Ros-based integration of smart space and a mobile robot as the internet of robotic things. In *2019 25th Conference of Open Innovations Association (FRUCT)*, page 339–345. IEEE, November 2019.
- [2] ArticulatedRobotics. Building a mobile robot. <https://www.youtube.com/watch?v=OWeLUSzxMsw&list=PLunhqkrRNrhYAffV8JDiFOatQXuU-NnxT>.