



UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA
Dipartimento di Informatica, Sistemistica e
Comunicazione
Corso di Laurea in Informatica

Percezione dell'Ambiente e Costruzione della Base di Conoscenza per Robot Umanoidi

Relatore: Prof. Dimitri Ognibene

Tesi di Laurea di:
Luca Brini
Matricola 879459

Anno Accademico 2023-2024

Indice

Introduzione	3
Stato dell'Arte	4
1 Mappe semantiche	5
1.1 Definizione	5
1.1.1 Nodi	6
1.1.2 Archi	8
1.1.3 Grafo stanze	9
1.1.4 Grafo Oggetti	9
1.1.5 Esempio di Mappa Semantica	9
1.2 Scena semantica	10
2 Grafo di Scena	11
2.1 Generazione del Grafo di Scena	11
2.1.1 Lettura dei frame RGB-D	11
2.1.2 Inferenza	12
2.1.3 Costruzione del grafo	13
2.2 Aggiornamento del Mappa Semantica	17
2.2.1 Trovare la stanza corrente del robot	18
2.2.2 Aggiornamento Grafo degli Oggetti	19
2.2.3 Salvataggio a DB e pubblicazione su MQTT	24
2.2.4 Esempio di funzionamento	24
2.3 Conclusioni	26
3 Riconoscimento di Stanze	27
3.1 Grid maps	27
3.2 Riconoscimento delle Stanze	28
3.2.1 Segmentazione basata su Voronoi	28
3.2.2 Preprocessing	31
3.2.3 Analisi dello spazio libero	32
3.2.4 Analisi dello spazio occupato	35
3.2.5 Ricerca delle porte	37
3.2.6 Separazione delle stanze	38
3.2.7 Generazione Grafo delle Stanze	39
3.3 Visualizzazione e Modifica delle Stanze	39
3.3.1 Layers e modelli	40
3.3.2 Schema di funzionamento	40

3.3.3	Features	41
3.4	Conclusioni	43
4	Conclusioni	44
A	Appendice	45
A.1	RoBee	45
A.1.1	Architettura Cloud Native	46
A.1.2	Dashboard and Console	46
A.1.3	Mappe, navigazione e LiDaR	47
A.1.4	Giunti e controllo	47
A.1.5	Camere e point cloud	47
A.2	Codice	47

Introduzione

Negli ultimi anni il campo della robotica ha vissuto un significativo incremento di applicazioni e innovazioni. Lo sviluppo di nuove tecnologie e la disponibilità di nuovi strumenti hanno reso possibile la creazione di robot in grado di svolgere compiti sempre più complessi. La **pianificazione automatica delle missioni** è sempre stata una delle attività di sviluppo in questo campo più affascinanti, pur essendo una delle più tediosa. Con l'avvento di ChatGPT e modelli simili, si è iniziato a pensare di integrare i **Large Language Models**, come alternativa ai classici planner, all'interno del sistema robot, con l'obiettivo di pianificare missioni autonome sulla base della descrizione in linguaggio naturale di ciò che si vuole far eseguire al robot.

La percezione dell'ambiente circostante è dunque una delle attività più importanti per un robot, soprattutto nell'ambito del **Mission Planning**. La capacità di riconoscere gli oggetti e di calcolarne la posizione è fondamentale per poterci interagire. Inoltre, è essenziale potersi localizzare nella mappa, sia in modo geometrico che topologico, in modo da poter pianificare anche eventuali movimenti verso gli oggetti desiderati che si trovano in punti non raggiungibili al momento dal robot.

In questo documento definiremo il significato di **Mappa Semantica**, le ragioni alla base della sua esistenza e la struttura. Successivamente entreremo nel dettaglio del **Grafo di Scena**, come viene generato e aggiornato per integrare i cambiamenti dell'ambiente. Infine analizzeremo il **Riconoscimento delle Stanze** a partire dalla grid map di occupazione generata attraverso i sensori LiDaR del Robot, essenziale per suddividere l'insieme degli oggetti nelle loro stanze e gestire le missioni che necessitano lo spostamento in altre stanze.

Questo sistema è stato studiato e progettato per RoBee, robot umanoide cognitivo di Oversonic Robotics (*maggiori informazioni in appendice A.1*).

Stato dell'Arte

Il mapping semantico e la ricostruzione dell'ambiente sono aspetti, fondamentali per il planning di robot autonomi, che hanno ricevuto molta attenzione per diverso tempo. Sin dai primi lavori nel campo si è cercato di ricostruire l'ambiente in cui il robot opera definendo dei link tra rappresentazione spaziale e semantica dell'ambiente.

Per esempio, [4] propone un sistema multi-gerarchico di conoscenza suddiviso in gerarchia spaziale, utilizzata per la pianificazione e esecuzione di task, e la gerarchia concettuale che modella la conoscenza semantica, in relazione tra loro tramite una serie di *anchor* che collegano nodi *Thing* delle due gerarchie. Per quanto riguarda la costruzione del modello semantico, la label associata agli oggetti è assegnata dal sistema di visione artificiale mentre il tipo di stanza è inferito grazie a quali oggetti essa contiene.

L'approccio proposto da [3] integra il feedback da parte dell'utente nei *beliefs* del sistema con l'obiettivo di ridurre l'incertezza riguardo le informazioni semantiche associate ad un ambiente. La mappa è divisa in tre layer di rappresentazione: il layer geometrico, il layer topologico e il layer semantico. Il layer geometrico consiste nella mappa probabilistica di occupazione, il layer topologico è automaticamente costruito dalla mappa geometrica suddividendo lo spazio libero in regioni utilizzando l'algoritmo di Voronoi [2] e il layer semantico estende quello topologico allegando label simboliche a gruppi di regioni.

Infine, con la crescente popolarità di modelli LLM e l'introduzione di GPT-4 Vision si è pensato di utilizzare i Large Language Models per l'identificazioni di *regions of interest* all'interno di una immagine, come proposto da [17].

Il lavoro di questa tesi mira ad approfondire queste tematiche, unendo il meglio degli approcci sopra citati e cercando di migliorarne le lacune. Per quanto riguarda la costruzione della mappa semantica, il riconoscimento degli oggetti è completamente autonomo, senza bisogno dell'intervento dell'umano. L'utilizzo di una variante dell'algoritmo di Voronoi permette di riconoscere le stanze all'interno della mappa geometrica e solo successivamente sarà l'utente a definirne il nome. Infine, il tutto viene implementato con processi di ML o algoritmi deterministi, senza l'utilizzo di LLM che rallenterebbero il tutto rendendo l'applicazione non adatta al contesto real time: infatti, così come per gli essere umani, il sistema permette integrare continuamente nuove informazioni riguardo l'ambiente, senza necessità di doversi fermare e aspettare i risultati di determinati processi.

Capitolo 1

Mappe semantiche

Gli esseri umani, talvolta senza rendersene conto, riescono a integrare continuamente nuove informazioni riguardo l'ambiente che li circonda, sia esso una casa, un edificio pubblico o un parco. Questa capacità, consci e inconscia, è essenziale per la successiva pianificazione di quei obiettivi o movimenti basati sulle informazioni appena apprese.

Così come per gli essere umani, anche i robot hanno bisogno di informazioni per poter essere considerati "cognitivi" e pianificare rispetto alla propria base di conoscenza. In particolar modo quando l'obiettivo è pianificare missioni data la descrizione in linguaggio naturale di ciò che il robot deve fare, come in questo caso.

Esempio Consideriamo una persona che entra per la prima volta in una biblioteca. Egli osserva scaffali pieni di libri, tavoli per lo studio e un'area dedicata ai computer. Queste informazioni vengono immagazzinate e utilizzate successivamente per trovare un libro specifico o un luogo tranquillo per studiare.

Allo stesso modo, immaginiamo un robot progettato per operare in una casa intelligente. Riceve l'istruzione: "Prendi il libro dal tavolo del soggiorno e portalo in cucina." Per svolgere questo compito, il robot deve comprendere la struttura della casa, identificare il tavolo corretto e navigare verso la cucina.

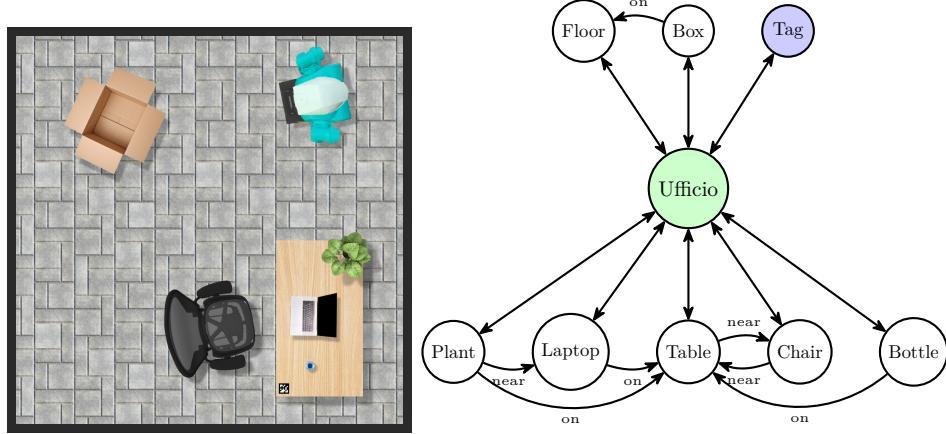
1.1 Definizione

La Mappa Semantica è un grafo orientato $G_m = (V_m, E_m)$ che rappresenta questa base di conoscenza dove:

- Un nodo può essere un:
 - Nodo stanza;
 - Nodo oggetto;
 - Nodo tag.
- Un arco può rappresentare:
 - La relazione tra due oggetti;
 - Il collegamento tra due stanze;

- L'appartenenza di un oggetto o un tag ad una ed una sola stanza.

Di conseguenza, per trovare gli oggetti o i nodi appartenenti ad una stanza s è sufficiente considerare il sottografo del nodo stanza s .



(a) Ambiente circostante il robot (*posto in alto a destra*) (b) In verde i nodi stanza, in blu i nodi tag e in bianco i nodi oggetto.

Figura 1.1: Ambiente e relativa mappa semantica.

1.1.1 Nodi

Nodi oggetto

I nodi oggetto rappresentano gli oggetti riconosciuti all'interno dell'ambiente attraverso la segmentazione panoptica dei frame video proveniente dalle camere del robot. Ogni nodo oggetto ha i seguenti attributi:

- Identificativo: utilizzato per identificare un oggetto all'interno della Mappa Semantica;
- Nome: label inferita dal modello di segmentazione panoptica;
- Posizione: terna (x, y, z) rappresentante la posizione dell'oggetto all'interno dell'ambiente rispetto alla Reference Posizione;
- Reference Posizione: origine del sistema di riferimento delle posizioni degli oggetti. Può essere l'origine del sistema Mappa o l'origine del sistema Robot;
- Tipo: rappresenta la tipologia dell'oggetto che può essere scelta tra:
 - Pickable: qualora l'oggetto possa essere preso attraverso gli end effectors del robot;
 - Non Pickable: qualora l'oggetto non possa essere preso attraverso gli end effectors del robot;
 - Asset: in tutti gli altri casi (*Per esempio un tavolo*).

I nodi oggetto vengono aggiornati con le inferenze di nuovi frame video: possono dunque essere eliminati dalla mappa semantica qualora un oggetto non si presenti più all'interno della scena oppure aggiornati, per esempio a livello di posizione, qualora l'oggetto venga spostato.

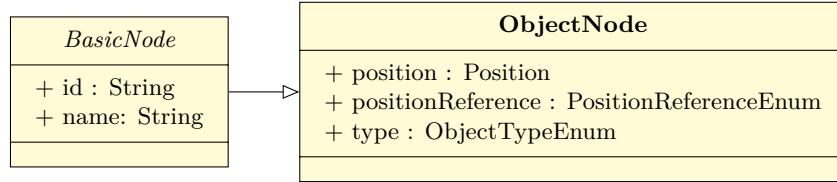


Figura 1.2: Diagramma delle classi - Nodo Oggetto

Nodi tag

I nodi tag rappresentano tutti quei riferimenti che vengono utilizzati dal robot per localizzarsi o localizzare oggetti di particolare rilevanza (come la stazione di ricarica o il tavolo di lavoro). Ogni nodo tag ha i seguenti attributi:

- Identificativo: utilizzato per identificare il tag all'interno della Mappa Semantica;
- Nome: assegnato dall'utente;
- Posizione: terza (x, y, z) rappresentante la posizione del tag all'interno dell'ambiente rispetto all'origine della mappa;
- Dimensione: dimensione del tag in millimetri;
- Di Navigazione: flag che indica se il tag è utilizzato per la navigazione del robot;
- Per Picking: flag che indica se il tag è utilizzato per identificare un oggetto di cui fare il picking con gli end effectors.

I nodi tag sono permanenti all'interno della mappa semantica perché si assume che questi non vengano mai spostati o rimossi dall'ambiente del robot

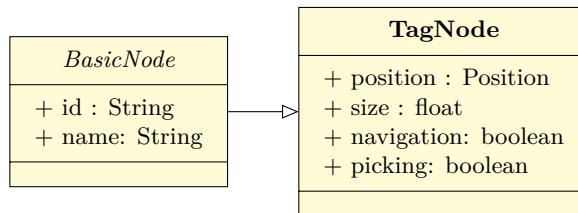


Figura 1.3: Diagramma delle classi - Nodo Tag

Nodi stanza

I nodi stanza rappresentano un'area semantica all'interno della mappa slam generata attraverso i sensori LiDaR del robot che, data in input ad un algormito, questo individua le stanze e ne genera il poligono. Ogni nodo stanza ha i seguenti attributi:

- Identificativo: utilizzato per identificare la stanza all'interno della Mappa Semantica;
- Nome: assegnato dall'utente;
- Segmenti: Lista di segmenti che delimitano il poligono della stanza. Viene usato per verificare se un oggetto appartiene ad una stanza o no;
- Oggetti: Sottografo degli oggetti appartenenti alla stanza.

La presenza di queste aree nella mappa è fondamentale per diverse ragioni:

- Permette di suddividere gli oggetti rispetto alla stanza di appartenenza, facilitando la discriminazione degli omonimi in base alla stanza di appartenenza e aggiungendo **keypoint** per la descrizione in linguaggio naturale di una missione;
- Consentirà l'utilizzo di algoritmi di ricerca su grafo per la pianificazione del percorso per raggiungere gli oggetti;
- Permetterà di creare percorsi pianificati che evitano determinate stanze.

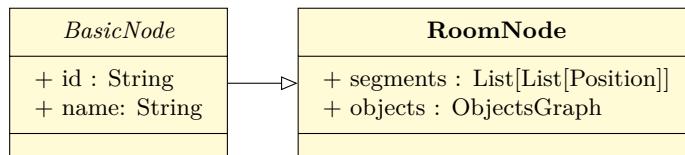


Figura 1.4: Diagramma delle classi - Nodo Stanza

1.1.2 Archi

Archi tra stanze

Gli archi tra i nodi di tipo stanza rappresentano il collegamento diretto tra due stanze.

Archi tra oggetti

Gli archi orientati tra i nodi di tipo oggetto rappresentano la relazione tra due oggetti. L'etichetta associata ad ogni arco appartiene all'insieme delle relazioni che possono essere inferite dal modello PSGTr. Queste informazioni sono importanti per poter pianificare task all'interno della missione che supportano il raggiungimento dell'obiettivo.

Per esempio, immaginiamo la missione "Prendi la bottiglia". Se è presente un ostacolo davanti alla bottiglia, rispetto alla posizione di presa del robot, questo deve prima pianificare lo spostamento dell'ostacolo. Ecco il motivo per il quale vi è la necessità di rappresentare queste relazioni tra oggetti.

1.1.3 Grafo stanze

Definiamo il grafo delle stanze come il sottografo (V_s, E_s) tale che:

- $V_s = \{v \in V_m \mid v \text{ è un nodo stanza}\}$
- $E_s = \{(v, u) \in E_m \mid v \in V_s \wedge u \in V_s\}$

dove V_m è l'insieme dei vertici del grafo della Mappa Semantica e E_m è l'insieme degli archi del grafo della Mappa Semantica.

Questo grafo, viene generato a partire dall'algoritmo di riconoscimento delle stanze, illustrato nel Capitolo 3, ed è la prima parte di Mappa Semantica creata, in concomitanza con la generazione della mappa slam. Solo successivamente sarà possibile la costruzione del grafo degli oggetti.

1.1.4 Grafo Oggetti

Definiamo il grafo degli oggetti (o grafo di scena) come il sottografo (V_o, E_o) tale che:

- $V_o = \{v \in V_m \mid v \text{ è un nodo oggetto o tag}\}$
- $E_o = \{(v, u) \in E_m \mid v \in V_o \wedge u \in V_o\}$

dove V_m è l'insieme dei vertici del grafo della Mappa Semantica e E_m è l'insieme degli archi del grafo della Mappa Semantica.

Una versione grezza del grafo degli oggetti viene generata dal modello PSG-Tr di [15]. Successivamente, come verrà mostrato nel Capitolo 2, questa verrà fusa con la Mappa Semantica rendendo dunque la base di conoscenza coerente rispetto lo stato dell'ambiente attuale.

1.1.5 Esempio di Mappa Semantica

Di seguito un esempio di mappa semantica e i vari sottografi.

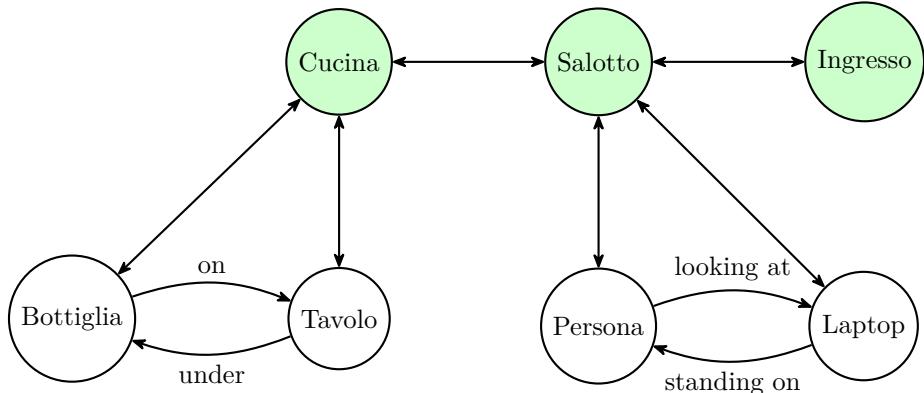
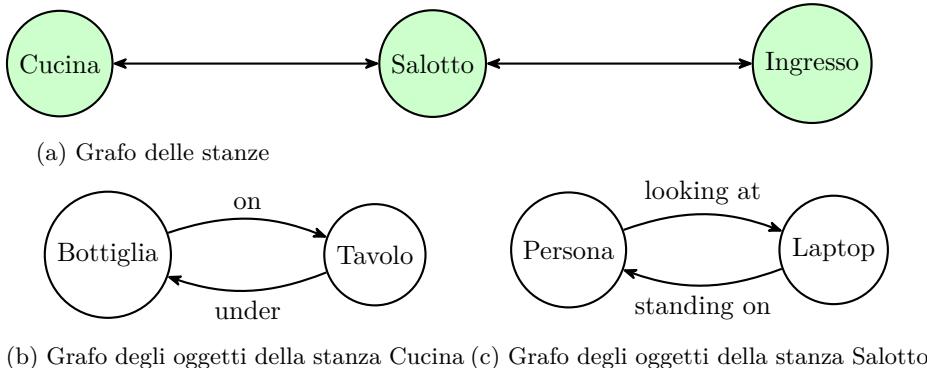


Figura 1.5: Mappa Semantica



1.2 Scena semantica

La Scena Semantica non è altro che un sottografo $G_{ss} = (V_{ss}, E_{ss})$ della mappa semantica che viene costantemente aggiornato con le ultime inferenze, senza considerare la suddivisione tra stanze e i precedenti oggetti individuati, dove:

- $V_{ss} = \{v \in V_m \mid v \text{ è un nodo oggetto}\}$
- $E_{ss} = \{(v, u) \in E_m \mid v \in V_{ss} \wedge u \in V_{ss}\}$

In poche parole è il grafo della scena direttamente inferito dal modello seppur con alcune differenze:

- I nodi oggetto presenti sono i soli rilevati entro un certo range dalla camera del robot. Questo range è impostato dalle configurazioni del servizio
- I nodi oggetto hanno la posizione rispetto alla terna del robot e non della mappa

Capitolo 2

Grafo di Scena

In questo capitolo verrà affrontata la generazione del grafo di scena dato un frame e l'aggiornamento della Mappa Semantica con queste nuove informazioni per manterla aggiornata rispetto all'ambiente.

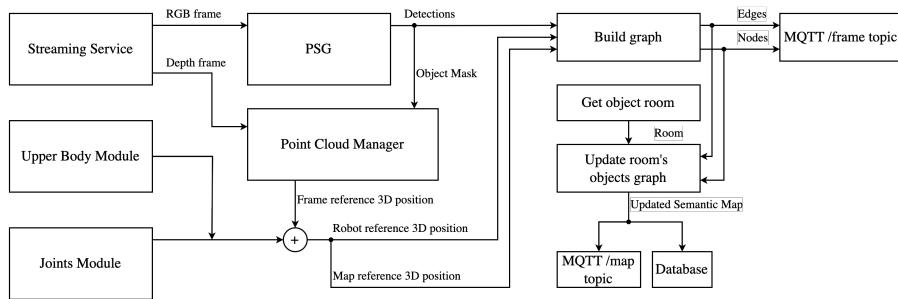


Figura 2.1: Schema dei flussi dati per la generazione del grafo di scena e aggiornamento della mappa semantica

2.1 Generazione del Grafo di Scena

La generazione del grafo di scena è un passo fondamentale per il mantenimento della coerenza tra Mappa Semantica e Ambiente reale.

Il grafo di scena è una struttura dati che rappresenta gli oggetti presenti nell'ambiente e le relazioni tra loro composta da nodi e archi. I nodi rappresentano gli oggetti, mentre gli archi rappresentano le relazioni tra gli oggetti.

In questa sezione verrà affrontata la costruzione di questa struttura dati a partire da un frame RGB-D e il modello di ML utilizzato per l'inferenza degli oggetti e delle relazioni.

2.1.1 Lettura dei frame RGB-D

All'interno dell'architettura cloud-native di Robee vi è la presenza di un pod chiamato "Streaming Module" il cui compito è streammare il feed video delle camera sul pod redis del robot, in modo che gli altri servizi o moduli possano

accedere a questi dati tramite l'utilizzo di librerie wrapper, rendendo il tutto agnostico rispetto alla tipologia e al modello di videocamera utilizzati.

2.1.2 Inferenza

Ogni frame ricevuto dal feed video viene successivamente dato in input al modello PSGTr [15] che restituisce un oggetto di tipo `Detections` il quale contiene i seguenti dati:

- `labels`: lista con lunghezza pari al numero di oggetti rilevati. Ogni valore indica la label corrispondente all' i -esimo oggetto. Per esempio, l'oggetto i -esimo ha label $labels[i]$;
- `masks`: lista contenente le maschere di ogni oggetto rilevato;
- `bboxes`: lista contenente le bounding boxes di ogni oggetto rilevato;
- `rel_pair_idxes`: lista con lunghezza pari al numero di relazioni tra oggetti rilevate. Ogni valore è a sua volta un array di dimensione due contenente gli indici dell'oggetto target e dell'oggetto sorgente della relazione;
- `rel_labels`: lista con lunghezza pari al numero di relazioni tra oggetti rilevate. Ogni valore indica la label della i -esima relazione
- `rel_dists`: lista con lunghezza pari al numero di relazioni tra oggetti rilevate. Ogni valore indica la probabilità associata alla i -esima relazione.

Questi dati vengono successivamente utilizzati per la costruzione del grafo di scena.

Panoptic Scene Graph - Transformer

Il modello PSGTr [15] è un modello di deep learning a singolo stato basato su architettura Transformer [8] il cui obiettivo è quello di generare una rappresentazione a grafo della scena data la segmentazione panottica piuttosto che le bounding box degli oggetti rilevati.

Training Il modello, per quanto riguarda gli oggetti, è stato addestrato su un dataset composto da 49mila immagini annotate basato su COCO [5] e Visual Genome [7]. Per le relazioni hanno estratto e costruito un dataset di 56 predicati a partire da dataset come VG-150 [9], VrR-VG [11] and GQA [10].

Segmentazione Panoptica La segmentazione panoptica individua gli oggetti e assegna a ogni pixel la label della classe dell'oggetto a cui appartengono. L'utilizzo di questa rispetto alle bounding da notevoli vantaggi:

- Garantisce una localizzazione più precisa degli oggetti, segmentandoli a livello di pixel e riducendo la presenza di pixel rumorosi o ambigui tipici delle bounding box, che spesso includono porzioni di altre categorie o oggetti;
- Copre l'intera scena di un'immagine, inclusi gli sfondi, offrendo una comprensione più completa del contesto rispetto alle bounding box, che tendono a trascurare importanti informazioni di sfondo;

- Riduce anche le informazioni ridondanti o irrilevanti presenti nei dataset basati su bounding box, focalizzandosi sulla segmentazione degli oggetti piuttosto che sulle loro parti.

Funzionamento di PSGTr L’architettura di PSGTr è basata su DETR [12] e HOI [13]. Il modello predice triple (*soggetto, predicato, verbo*) e la localizzazione degli oggetti simultaneamente.

Pipeline PSGTr Attraverso una backbone CNN, PSGTr estrae le features dell’immagine e i positional encodings che, insieme alle triplet queries, vengono dati in input al transformer encoder-decoder. In questo processo, l’obiettivo è che le query apprendano la rappresentazione del grafo di scena a triple in modo che per ognuna di esse, le predictions di (*soggetto, predicato, verbo*) possano successivamente essere estratte da tre Feed Forward Network. Infine, il task di segmentazione viene eseguito da due head panoptiche, una per il soggetto e una per l’oggetto della relazione.

2.1.3 Costruzione del grafo

L’obiettivo di questo step è la costruzione del grafo di scena rispetto all’ultimo frame. Per farlo, è necessario estrarre i dati dai risultati dell’inferenza di PSGTr e calcolare quei valori che dipendono dal sistema robot, come la posizione. L’algoritmo di costruzione del grafo è costituito da 2 fasi principali:

- Costruzione dei nodi per la scena semantica e per la mappa semantica:
 - Calcolo della posizione dell’oggetto
- Costruzione degli archi per la scena semantica e per la mappa semantica

Costruzione dei nodi

Estrazione dati oggetto dai risultati L’oggetto MMDetResult ritornato dalla funzione di inferenza del modello, come detto precedentemente, possiede un attributo *labels* che è una lista con lunghezza pari al numero di oggetti rilevati dove il valore *i*-esimo, indica l’indice della classe di appartenenza dell’oggetto *i*. Lo stesso meccanismo vale anche per le maschere.

Algorithm 1 Estrazione classi e maschere degli oggetti individuati

```

1: obj_classes  $\leftarrow$  []
2: obj_masks  $\leftarrow$  []
3: obj_labels_ids  $\leftarrow$  detectionResults.labels
4: for i = 0 to obj_labels_ids.length do
5:   obj_classes.append(PSG_CLASSES[obj_labels_ids[i]])
6:   obj_masks.append(detectionResults.masks[i])
7: end for
```

Calcolo posizioni 3D Per ogni oggetto, si estrae la posizione 3D nella mappa del robot in modo che questo possa successivamente localizzarlo e raggiungerlo.

Calcolo posizione 3D nel sistema pixel Le maschere generate dal modello consentono di calcolare il centroide (x_i, y_i) dell'oggetto i -esimo. Tuttavia, queste maschere forniscono solo un valore in due dimensioni. Per il calcolo del valore z_i si utilizza il Point cloud che, combinando il frame RGB con il frame Depth, permette di ottenere una rappresentazione 3D della scena. Per ogni oggetto i , si maschera il Point Cloud con la maschera i -esima e si calcola z_i come valore mediano tra le z_s di tutti i punti mascherati ottenendo così una posizione $P_{ipd} = (x_i, y_i, z_i)$.

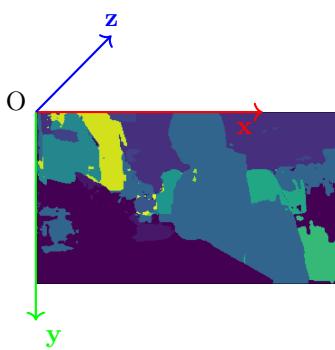


Figura 2.2: Sistema di coordinate pixel

Algorithm 2 Calcolo della posizione 3D nel sistema pixel

```

1: procedure GET_PIXEL_COORDS(depth_frame, obj_mask)
2:    $z_{ip} \leftarrow \text{median}(depth\_frame[obj\_mask])$             $\triangleright$  Point Cloud
3:    $x_{ip} \leftarrow \text{median}(obj\_mask[:, 0])$ 
4:    $y_{ip} \leftarrow \text{median}(obj\_mask[:, 1])$ 
5:   return  $x_{ip}, y_{ip}, z_{ip}$ 
6: end procedure

```

Calcolo posizione 3D nel sistema camera Per ogni oggetto i , è necessario trasformare la posizione P_{ipd} nel sistema di coordinate della camera, ovvero con la camera nell'origine. A tale scopo, si utilizza la **Matrice Intrinsic della Camera** ovvero la matrice di trasformazione affine usata per convertire le coordinate in sistema camera a coordinate in sistema pixel. Questa dipende da caratteristiche fisiche della camera come apertura focale, campo visivo e risoluzione.

Poichè è necessario eseguire il procedimento inverso, ovvero trasformare le coordinate P_{ipd} in sistema pixel a coordinate in sistema camera, viene utilizzata la Matrice Intrinsic Inversa e si esegue il prodotto matriciale tra questa e le coordinate P_{ipd} aumentate, ottenendo così la posizione P_{ic} degli oggetti in sistema camera.

$$\begin{bmatrix} x_{ic} \\ y_{ic} \\ z_{ic} \\ 1 \end{bmatrix} = M_{ic}^{-1} * \begin{bmatrix} x_{ip} \\ y_{ip} \\ z_{ip} \\ 1 \end{bmatrix}$$

Dove:

- M_{ic} è la Matrice Intrinseca della Camera
- x_{ip}, y_{ip}, z_{ip} sono le coordinate in sistema pixel dell'oggetto i
- x_{ic}, y_{ic}, z_{ic} sono le coordinate in sistema camera dell'oggetto i

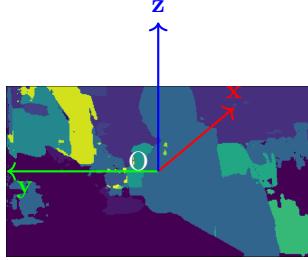


Figura 2.3: Sistema di coordinate camera

Algorithm 3 Calcolo della posizione 3D nel sistema camera

```

1: procedure GET_CAMERA_COORDS( $P_{ip}$ )
2:    $M_{ic} \leftarrow \text{GET\_CAMERA\_INTRINSICS}$                                  $\triangleright$  Funzione libreria Robee
3:    $P_{ic} \leftarrow M_{ic}^{-1} \times P_{ip}$ 
4:    $distance = \text{norm}(P_{ic} - P_{\text{camera}})$ 
5:   return  $P_{ic}, distance$ 
6: end procedure

```

Calcolo posizione 3D nel sistema mappa L'ultimo passaggio per ottenere la posizione dell'oggetto nella mappa è quello di utilizzare la Matrice Estrinseca della Camera, ovvero la matrice di trasformazione affine usata per convertire le coordinate in sistema mondo a coordinate in sistema camera. Dipende dalla posizione e dall'orientamento della camera nel mondo.

Dato che in questo contesto la camera è montata sulla testa del robot, la matrice estrinseca dipende dalla posizione e dall'orientamento di quest'ultima e a seguire di tutte le trasformate nell'albero delle TF del robot. All'interno dell'architettura di Robee, sono presenti due moduli che si occupano di calcolare le trasformate per passare da un sistema di coordinate ad un altro. Si utilizzano dunque questi moduli per calcolare la matrice estrinseca inversa rispetto alla mappa/robot che moltiplicata per le coordinate P_{ic} ottenute precedentemente, permette di ottenere la posizione dell'oggetto nel sistema mappa/robot.

$$\begin{bmatrix} x_{im} \\ y_{im} \\ z_{im} \\ 1 \end{bmatrix} = M_{ec}^{-1} * \begin{bmatrix} x_{ic} \\ y_{ic} \\ z_{ic} \\ 1 \end{bmatrix}$$

Dove:

- M_{ec} è la Matrice Estrinseca della Camera
- x_{ic}, y_{ic}, z_{ic} sono le coordinate in sistema camera dell'oggetto i
- x_{im}, y_{im}, z_{im} sono le coordinate in sistema mappa dell'oggetto i

Algorithm 4 Calcolo della posizione 3D nel sistema mappa

```

1: procedure GET_MAP_COORDS( $P_{ic}$ )
2:    $M_{ec} \leftarrow \text{GET\_CAMERA\_EXTRINSICS}$             $\triangleright$  Funzione libreria Robee
3:    $P_{im} \leftarrow M_{ec}^{-1} \times P_{ic}$ 
4:   return  $P_{im}$ 
5: end procedure

```

Istanziamento dei nodi Con tutti i dati ora a disposizione è possibile istanziare i nodi per la scena semantica e per la mappa semantica. Per la scena semantica, si istanziano i nodi con la posizione 3D nel sistema di riferimento del robot, mentre per la mappa semantica, si istanziano i nodi con la posizione 3D nel sistema di riferimento della mappa.

A causa del rumore del frame Depth nelle zone troppe vicine o troppo lontane dalla camera, è necessaria l'applicazione di un filtro per escludere gli oggetti che distano troppo dalla camera o che sono troppo vicini. Le soglie di distanza vengono impostate nei parametri di configurazione del servizio.

Algorithm 5 Instanziamento dei nodi

```

1:  $obj\_ids \leftarrow []$ 
2: for  $i = 0$  to  $obj\_labels\_ids.length$  do
3:    $obj\_pixel\_coords \leftarrow \text{GET\_PIXEL\_COORDS}(depth\_frame, obj\_masks[i])$ 
4:    $obj\_camera\_coords, distance \leftarrow \text{GET\_CAMERA\_COORDS}(obj\_pixel\_coords)$ 
5:    $obj\_coords \leftarrow \text{GET\_MAP\_COORDS}(obj\_camera\_coords)$ 
6:   if  $distance > min\_distance$  and  $distance < max\_distance$  then
7:      $node \leftarrow \text{Node}(i, obj\_classes[i], obj\_coords)$ 
8:      $obj\_ids.append(i)$ 
9:      $semantic\_scene.add\_node(node)$ 
10:     $semantic\_map.add\_node(node)$ 
11:   end if
12: end for

```

Costruzione degli archi

Estrazione dati relazione dai risultati L'oggetto MMDetResult ritornato dalla funzione di inferenza del modello, come detto precedentemente, possiede gli attributi:

- rel_labels : lista con lunghezza pari al numero di relazione rilevate dove il valore j -esimo, indica l'indice della classe di appartenenza della relazione j .

- *rel_pair_idxes*: lista con lunghezza pari al numero di relazioni tra oggetti rilevate. Ogni valore è a sua volta un array di dimensione due contenente gli indici dell'oggetto target e dell'oggetto sorgente della relazione;
- *rel_dist*: lista con lunghezza pari al numero di relazione rilevate dove il valore j -esimo, indica la probabilità associata alla relazione j .

È necessario estrarre questi dati e mettere in relazione gli oggetti tra loro per costruire gli archi del grafo.

Per ogni relazione j , si estrae l'indice dell'oggetto sorgente s e l'indice dell'oggetto target t e viene creato un arco tra i nodi corrispondenti se:

- Entrambi i nodi siano presenti nella lista di oggetti precedentemente calcolata, ovvero rispettano i vincoli di distanza.
- La probabilità associata alla relazione sia maggiore di una certa soglia, impostata nei parametri di configurazione del servizio.

Algorithm 6 Instanziamento degli archi

```

1: for  $j = 0$  to rel_labels.length do
2:   source_idx  $\leftarrow$  rel_pair_idxes[ $j$ ][0]
3:   target_idx  $\leftarrow$  rel_pair_idxes[ $j$ ][1]
4:   if source_idx in obj_ids and target_idx in obj_ids then
5:     source_node  $\leftarrow$  semantic_scene.get_node(source_idx)
6:     target_node  $\leftarrow$  semantic_scene.get_node(target_idx)
7:     if rel_dist[ $j$ ] > rel_threshold then
8:       semantic_scene.add_edge(source_node, target_node, rel_labels[ $j$ ])
9:     end if
10:   end if
11: end for
```

2.2 Aggiornamento del Mappa Semantica

In questa sezione verrà affrontato il processo di aggiornamento della Mappa Semantica con i nuovi dati ottenuti dalla Generazione del Grafo di Scena della sezione precedente.

La Mappa Semantica è una struttura dati che rappresenta l'ambiente circostante il robot e che viene aggiornata in real time per mantenere la coerenza con l'ambiente reale.

Il processo di aggiornamento della Mappa Semantica è composto da 3 fasi principali:

- Trovare la stanza corrente del robot utilizzando il Grafo delle Stanze della Mappa Semantica
- Aggiornare il Grafo degli Oggetti della stanza corrente con i nuovi nodi e archi.
- Aggiornare la Mappa Semantica su DB e pubblicare i nuovi risultati sul relativo topic MQTT.

2.2.1 Trovare la stanza corrente del robot

Trovare la stanza corrente del robot è uno step necessario: permette di aggiornare il Grafo degli Oggetti della sola la stanza in cui il robot si trova, evitando di aggiornare l'intera mappa, efficientando il processo.

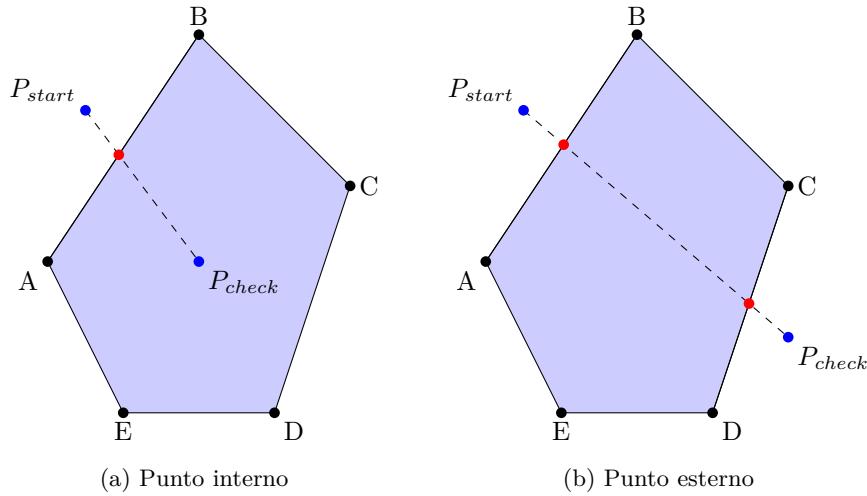
Algoritmo di Ray Casting

Ogni nodo stanza della Mappa Semantica ha un attributo *segments* che rappresenta i segmenti che delineano i confini della stanza fisica.

Per determinare la stanza in cui si trova il robot, si utilizza l'algoritmo di Ray Casting [1] che permette di determinare se un punto è all'interno di un poligono, il cui funzionamento è il seguente:

- Si sceglie come punto di inizio P_{start} del raggio un punto che sicuramente è all'esterno del poligono.
- Si emette un raggio dal punto P_{start} al punto P_{check} che si vuole verificare se è all'interno del poligono.
- Si contano quante volte il raggio interseca i segmenti del poligono:
 - Se il numero è dispari, il punto è all'interno del poligono;
 - Se il numero è pari, il punto è all'esterno del poligono.

L'intuizione alla base di questo algoritmo è che se il segmento che congiunge il punto esterno e il punto P_{check} interseca un numero dispari di segmenti del poligono significa che ho incontrato solo un "bordo" del poligono e quindi il punto è all'interno. Se il numero è pari, significa che ho incontrato due "bordi" e quindi sono o entrato e poi uscito dal poligono oppure nemmeno entrato.



Algorithm 7 Ray Casting

```
1: procedure IS_INSIDE_POLYGON( $P_{start}, P_{check}, segments$ )
2:    $intersection\_count \leftarrow 0$ 
3:   for  $i = 0$  to  $segments.length$  do
4:      $A \leftarrow segments[i]$ 
5:      $B \leftarrow segments[(i + 1) \% segments.length]$ 
6:     if DO_INTERSECT( $P_{start}, P_{robot}, A, B$ ) then       $\triangleright$  Appendice A.2
7:        $intersection\_count \leftarrow intersection\_count + 1$ 
8:     end if
9:   end for
10:  return  $intersection\_count \% 2 == 1$ 
11: end procedure
```

Determinare la stanza corrente

Data la posizione del robot P_{robot} nella mappa, per ogni stanza s della Mappa Semantica si effettua l'algoritmo di Ray Casting sopra descritto dove:

- P_{start} viene scelto come un punto medio di un lato della bounding box con padding della stanza s ;
- P_{check} è la posizione del robot nella mappa.

Se il punto P_{check} , ovvero il robot, giace all'interno del poligono della stanza s , allora quest'ultima è la stanza corrente.

È bene notare che è impossibile che le stanze si sovrappongano, poiché sia il riconoscimento delle stanze (*capitolo 3*) che la console per modificare la segmentazione effettuano un controllo per evitare questa casistica. Di conseguenza, l'algoritmo restituirà sempre una e una sola stanza.

Algorithm 8 Trovare la stanza corrente

```
1:  $P_{robot} \leftarrow \text{robot.get\_position()}$ 
2:  $current\_room \leftarrow \text{None}$ 
3: for  $s$  in  $semantic\_map.get\_rooms()$  do
4:    $bb\_x\_min, bb\_x\_max, bb\_x\_min \leftarrow \text{GET\_BOUNDING\_BOX}(s.segments)$ 
5:    $P_{start} \leftarrow [\text{MEAN}(bb\_x\_min, bb\_x\_max), bb\_y\_min - \epsilon]$ 
6:   if IS_INSIDE_POLYGON( $P_{start}, P_{robot}, s.segments$ ) then
7:      $current\_room \leftarrow s$ 
8:   end if
9: end for
```

2.2.2 Aggiornamento Grafo degli Oggetti

Una volta individuata la stanza corrente, si procede con l'aggiornamento del Grafo degli Oggetti della stanza corrente utilizzando i nuovi nodi e archi ottenuti dalla Generazione del Grafo di Scena.

Poichè il modello PSGTr individua e riconosce gli oggetti in modo non deterministico, non assegna sempre lo stesso identificativo a un oggetto nei diversi

frame. Questo comportamento rende difficile tracciare gli oggetti nel tempo. Di conseguenza, non ci si può basare su questi, tantomeno delle label a causa dei possibili omonimi, per aggiornare il grafo. La soluzione proposta è molto semplice:

- Si proietta il Camera Frustum nel sistema mappa per determinare l'area della stanza attualmente visibile dalla telecamera;
- Si eliminano tutti i nodi oggetto dal Grafo degli Oggetti della stanza che giacciono all'interno del Camera Frustum e gli archi che entrano/escano da questi;
- Si aggiungono i nuovi nodi e archi ottenuti dall'ultima generazione del grafo di scena che rappresenta lo stato attuale della porzione di stanza visibile.

Proiezione del Camera Frustum

Il Camera Frustum è la porzione di spazio visibile dalla telecamera.

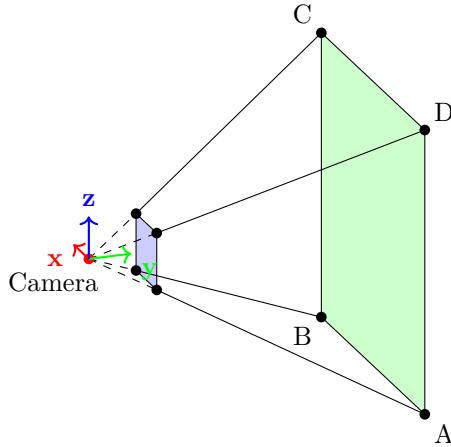


Figura 2.5: Camera Frustum. I vertici del piano verde rappresentano la proiezione degli estremi del frame nel sistema di coordinate mappa.

Il funzionamento alla base di questo processo è molto simile a quello affrontato nella sezione 2.1.3 per il calcolo della posizione 3D di un oggetto. I punti di partenza sono però i vertici del frame della camera e non le maschere degli oggetti. I vertici del frame dipendono dalla risoluzione della camera e sono:

- $A = (0, 0)$
- $B = (width, 0)$
- $C = (width, height)$
- $D = (0, height)$

Dove $width$ e $height$ sono rispettivamente la larghezza e l'altezza del frame. Per proiettare i vertici del frame nel sistema mappa, si procede come segue:

- Gli estremi del frame vengono aumentati con $z = 1$ e moltiplicati per la Matrice Intrinseca Inversa per ottenere i vettori $\vec{a}, \vec{b}, \vec{c}, \vec{d}$ nel sistema camera;
 - I vettori $\vec{a}, \vec{b}, \vec{c}, \vec{d}$ vengono moltiplicati per la Matrice Estrinseca Inversa per ottenere i vettori $\vec{a}', \vec{b}', \vec{c}', \vec{d}'$ nel sistema mappa.
 - Per ogni vettore \vec{v} del punto precedente:
 - Viene calcolato il vettore direzione \vec{d} come il vettore differenza tra \vec{v} e \vec{P}_{camera} normalizzato:
- $$\vec{d} = (\vec{v} - \vec{P}_{camera}) / \text{norm}(\vec{v} - \vec{P}_{camera}) \quad (2.1)$$
- Viene calcolato il punto finale \vec{v}' come la somma tra il vettore normalizzato \vec{d} moltiplicato per uno scalare $distance$ e \vec{P}_{camera} :
- $$\vec{v}' = \vec{d} \cdot distance + \vec{P}_{camera} \quad (2.2)$$

Questo processo consente di costruire un tetraedro con vertici A', B', C', D' e P_{camera} che approssima il Camera Frustum nel sistema mappa che verrà utilizzato per eliminare gli oggetti del grafo che giacciono all'interno di esso in modo da poter aggiornare il grafo.

Algorithm 9 Proiezione del Camera Frustum

```

1: procedure PROJECT_CAMERA_FRUSTUM( $P_{camera}, frame\_vertices$ )
2:    $map\_vertices \leftarrow []$ 
3:    $M_{ic} \leftarrow \text{GET\_CAMERA\_INTRINSICS}$ 
4:    $M_{ec} \leftarrow \text{GET\_CAMERA\_EXTRINSICS}$ 
5:   for  $i = 0$  to  $frame\_vertices.length$  do
6:      $\vec{v} \leftarrow M_{ic}^{-1} \times frame\_vertices[i]$ 
7:      $\vec{v}' \leftarrow M_{ec}^{-1} \times \vec{v}$ 
8:      $\vec{d} \leftarrow (\vec{v}' - \vec{P}_{camera})$ 
9:      $\vec{d}_n \leftarrow \vec{d} / \text{norm}(\vec{d})$ 
10:     $V \leftarrow \vec{d}_n * distance + \vec{P}_{camera}$ 
11:     $map\_vertices.append(V)$ 
12:   end for
13:   return  $map\_vertices$ 
14: end procedure

```

Controllo della posizione degli oggetti

Per stabilire se un oggetto giace nel frustum della camera e quindi è un oggetto da eliminare dal Grafo degli Oggetti della stanza corrente, si utilizza un algoritmo che coinvolge la costruzione dei piani del frustum e il controllo della posizione degli oggetti.

Costruzione dei piani Per definire un piano è necessario conoscere almeno tre punti \vec{v}_1, \vec{v}_2 e \vec{v}_3 in modo da poter calcolare la normale \vec{n} e la distanza d con segno:

$$\begin{cases} \vec{n} = (\vec{v}_1 - \vec{v}_2) \times (\vec{v}_3 - \vec{v}_2) \\ d = \vec{n} \cdot \vec{v}_1 = \vec{n} \cdot \vec{v}_2 = \vec{n} \cdot \vec{v}_3 \end{cases} \quad (2.3)$$

Il verso della normale del piano dipende dall'ordine dei vertici \vec{v}_1, \vec{v}_2 e \vec{v}_3 : se i vertici sono disposti in senso orario, la normale punta verso l'interno del tetraedro, altrimenti punta verso l'esterno.

In questo processo, la normale deve necessariamente puntare verso l'interno del tetraedro. Per esserne certi, si calcola il prodotto scalare tra la normale del piano e un vettore che sicuramente è all'interno del tetraedro, ad esempio il centroide \vec{c} del solido. Se questo meno la distanza d è minore di zero, allora la normale punta verso l'esterno del tetraedro e si negano i valori di \vec{n} e d .

$$\vec{n} = \begin{cases} \vec{n} & \text{se } \vec{n} \cdot \vec{c} - d \geq 0 \\ -\vec{n} & \text{altrimenti} \end{cases} \quad d = \begin{cases} d & \text{se } \vec{n} \cdot \vec{c} - d \geq 0 \\ -d & \text{altrimenti} \end{cases} \quad (2.4)$$

A supporto di queste operazioni, è stata definita una classe *Plane* che permette di costruire un piano a partire da tre punti e di controllare se la normale punta verso l'interno del tetraedro.

Plane
<code>+ normal : Vector</code> <code>+ d : float</code>
<code><<create>></code> <code>+ Plane(v1 : Vector, v2 : Vector, v3 : Vector)</code> <code>+ checkNormal(center : Vector) : void</code> <code>- changeSign() : void</code>

Figura 2.6: Classe Plane

I piani creati sono i seguenti:

- Piano che passa per i vertici $\vec{d}', \vec{c}', \vec{b}'$;
- Piano che passa per i vertici $\overrightarrow{P_{camera}}, \vec{a}', \vec{b}'$
- Piano che passa per i vertici $\overrightarrow{P_{camera}}, \vec{b}', \vec{c}'$
- Piano che passa per i vertici $\overrightarrow{P_{camera}}, \vec{c}', \vec{d}'$
- Piano che passa per i vertici $\overrightarrow{P_{camera}}, \vec{d}', \vec{a}'$

Appartenenza di un punto al frustum Data la posizione dell'oggetto P_i e i piani del camera frustum si controlla se per ognuno di questi l'oggetto giace nel sottospazio delimitato dal piano. Se l'oggetto giace nei sottospazi di tutti i piani allora l'oggetto è all'interno del frustum.

Sia $\vec{n} = (x_n, y_n, z_n)$ la normale del piano e d la distanza con segno di un punto

\vec{p} dal piano. Il punto $\vec{p} = (x_p, y_p, z_p)$ giace nel sottospazio delimitato dal piano se:

$$\vec{n} \cdot \vec{p} \geq d \quad (2.5)$$

Che in coordinate cartesiane diventa:

$$x_n \cdot x_p + y_n \cdot y_p + z_n \cdot z_p \geq d \quad (2.6)$$

Algorithm 10 Controllo appartenenza di punto al frustum

```

1: procedure IS_INSIDE_FRUSTUM( $P_i, planes$ )
2:    $inside \leftarrow$  True
3:   for  $i = 0$  to  $planes.length$  do
4:      $\vec{n} \leftarrow planes[i].normal$ 
5:      $d \leftarrow planes[i].d$ 
6:     if  $\vec{n} \cdot P_i < d$  then
7:        $inside \leftarrow$  False
8:     end if
9:   end for
10:  return  $inside$ 
11: end procedure

```

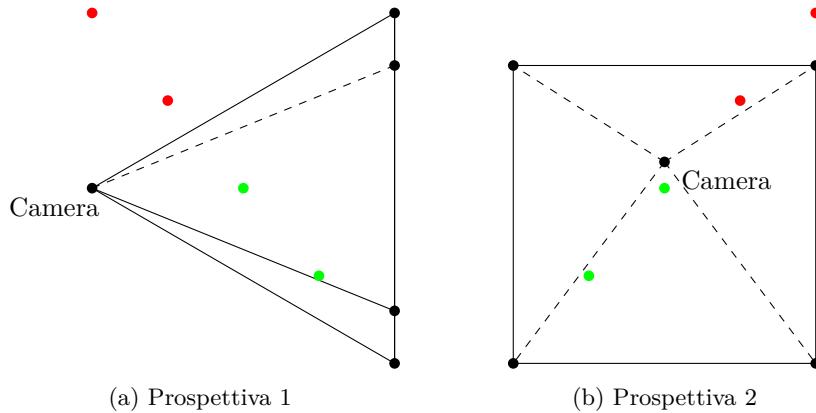


Figura 2.7: Controllo appartenenza di un punto al frustum. I punti verdi giacciono all'interno del frustum, i punti rossi all'esterno.

Aggiornamento del Grafo

Una volta proiettato il Camera Frustum e costruiti i piani, si procede con l'aggiornamento del Grafo degli Oggetti della stanza corrente.

Per ogni nodo oggetto i del Grafo degli Oggetti della stanza corrente, si controlla se giace all'interno del frustum. Se è all'interno, allora si eliminano il nodo e gli archi che entrano/escano da esso. Se è all'esterno, si mantiene il nodo e gli archi che entrano/escano da esso.

Infine, si aggiungono i nuovi nodi e archi ottenuti dalla Generazione del Grafo di Scena della sezione precedente.

Algorithm 11 Aggiornamento del Grafo degli Oggetti

```
1: map_vertices  $\leftarrow$  PROJECT_CAMERA_FRUSTUM( $P_{camera}, frame\_vertices$ )
2: frustum_planes  $\leftarrow$  CREATE_FRUSTUM_PLANES( $P_{camera}, map\_vertices$ )
3: room_nodes  $\leftarrow$  currentRoom.get_nodes()
4: room_edges  $\leftarrow$  currentRoom.get_edges()
5: nodes_to_remove  $\leftarrow$  [ ] ▷ Scelta nodi da rimuovere
6: for  $i = 0$  to room_nodes.length do
7:   node  $\leftarrow$  room_nodes[ $i$ ]
8:   if IS_INSIDE_FRUSTUM(node.position, frustum_planes) then
9:     nodes_to_remove.append(node)
10:    end if
11:   end for
12: edges_to_remove  $\leftarrow$  [ ] ▷ Scelta archi da rimuovere
13: for  $i = 0$  to room_edges.length do
14:   edge  $\leftarrow$  room_edges[ $i$ ]
15:   if nodes_to_remove[edge.source] or nodes_to_remove[edge.target] then
16:     edges_to_remove.append(edge)
17:   end if
18:   end for
19: for  $i = 0$  to nodes_to_remove.length do ▷ Rimozione nodi e archi
20:   currentRoom.remove_node(nodes_to_remove[ $i$ ])
21: end for
22: for  $i = 0$  to edges_to_remove.length do
23:   currentRoom.remove_edge(edges_to_remove[ $i$ ])
24: end for
25: current.add_nodes(last_scene_graph.nodes) ▷ Aggiunta nuovi risultati
26: current.add_edges(last_scene_graph.edges)
```

2.2.3 Salvataggio a DB e pubblicazione su MQTT

Una volta aggiornato il Grafo degli Oggetti della stanza corrente, si procede con il salvataggio della Mappa Semantica aggiornata su DB e la pubblicazione dei nuovi risultati sul relativo topic MQTT.

Il salvataggio su DB è necessario per mantenere la coerenza della mappa tra le varie esecuzioni del sistema e condividere la conoscenza tra lo swarm di Robot che opera sulla stessa stanza. La pubblicazione su MQTT permette di rendere disponibili i nuovi risultati a tutti i moduli o servizi dell'architettura cloud di Robee.

2.2.4 Esempio di funzionamento

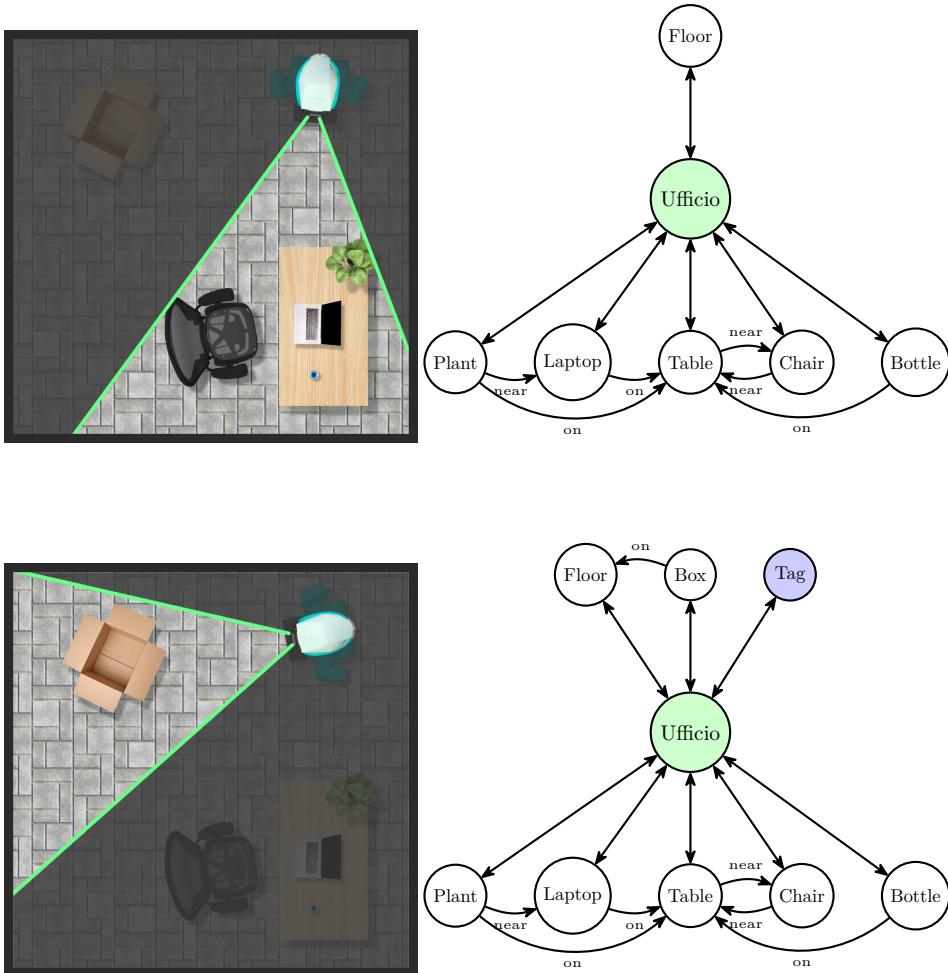
Per chiarire il funzionamento dell'algoritmo, si consideri la seguente stanza, un ufficio, con i relativi oggetti:

- Tavolo
- Laptop
- Pianta

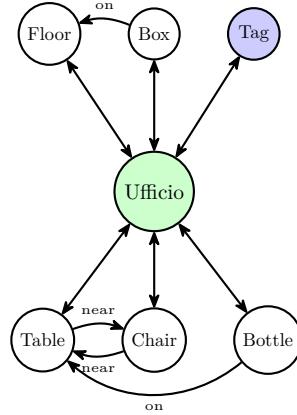
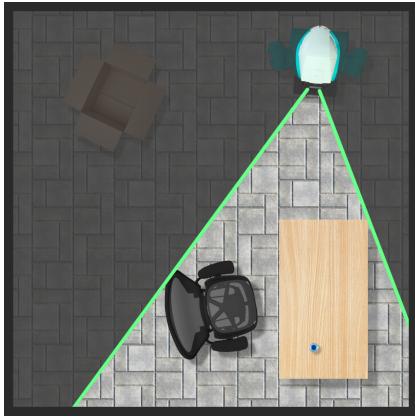
- Sedia
- Bottiglia
- Scatolone

Assumiamo che la Mappa Semantica sia già stata inizialmente generata e che quindi sia presente un nodo stanza *ufficio* con i relativi attributi e un Grafo degli Oggetti che inizialmente è vuoto. Il robot, muovendosi all'interno della stanza, rileva i vari oggetti e integra nuova conoscenza seguendo il flusso di lavoro descritto in questo capitolo.

Nelle seguenti figure, si mostrano l'ambiente e le regioni della stanza attualmente visibili dal robot sulla sinistra e l'evoluzione della Mappa Semantica sulla destra.



Nel seguente step, il robot ruota nella posizione iniziale dove la scena è cambiata. Tutti gli oggetti della Mappa Semantica che cadevano all'interno del frustum vengono rimossi e i nuovi oggetti rilevati, che in questo caso sono la sedia, il tavolo e la bottiglia, vengono aggiunti.



2.3 Conclusioni

In questo capitolo è stata presentata la pipeline di generazione della Mappa Semantica attraverso l'utilizzo del modello PSGTr e l'aggiornamento di questa con i nuovi dati di inferenza del modello utilizzando tecniche geometriche.

Il modello PSGTr è stato scelto per la sua capacità di generare il grafo di scena in un tempo di inferenza ragionevole per applicazioni real time: richiede infatti circa $400ms$ in un cluster avente accesso a NVIDIA T4 permettendo così di aggiornare la rappresentazione semantica dell'ambiente praticamente instantaneamente.

La pipeline di aggiornamento della Mappa Semantica è semplice, efficiente ed efficace: Grazie all'utilizzo di operazioni di geometria lineare, che si traducono in moltiplicazioni e somme di numeri, è possibile aggiornare la mappa in un tempo ragionevole.

In futuro si può sperimentare introducendo modelli [16] che utilizzano anche il tracking degli oggetti segmentati in modo da poter tracciare gli oggetti nel tempo e aggiornare direttamente la mappa senza dover ricorrere a tecniche geometriche.

Capitolo 3

Riconoscimento di Stanze

In questo capitolo verrà affrontato il processo di riconoscimento delle stanze a partire dalle mappa SLAM generata attraverso i sensori LIDAR del robot. In particolare, verrà presentato il processo di individuazione proposto da [14], le modifiche adottate e la console per poter visualizzare e modificare i risultati ottenuti.

I risultati ottenuti da questo processo verranno utilizzati per la generazione del grafo delle stanze nella mappa semantica.

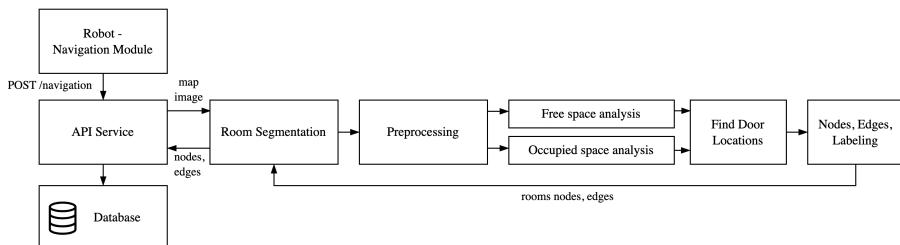


Figura 3.1: Schema dei flussi dati per il riconoscimento delle stanze e il salvataggio del grafo delle stanze

3.1 Grid maps

Le Grid Maps sono rappresentazioni spaziali dell'ambiente che utilizzano una griglia per suddividere lo spazio in celle o pixel. In questo caso, sono utilizzate per rappresentare le grid map di occupazione, ovvero mappe in cui ogni cella può assumere uno dei seguenti valori:

- Nero: se la cella è occupata. Tipicamente viene utilizzato per rappresentare le pareti o gli ostacoli presenti nell'ambiente;
- Bianco: se la cella è libera. Tipicamente viene utilizzato per rappresentare lo spazio libero, quello percorribile dal robot;
- Grigio: se la cella è sconosciuta. Tipicamente viene utilizzato per rappresentare le celle di cui non si ha informazione.

Questa tipologia di mappe è utilizzata per la navigazione dei robot autonomi, in quanto permette di localizzarsi e mappare l'ambiente circostante simultaneamente (SLAM).

Nel nostro caso, le grid map di occupazione vengono generate dal modulo di navigazione del robot e la loro immagine, in formato pgm, viene utilizzata per il riconoscimento delle stanze. In particolare, quando il modulo di navigazione effettua una richiesta http POST /navigation al servizio di API per la creazione di una nuova mappa in contemporanea viene eseguito il job di Room Segmentation.

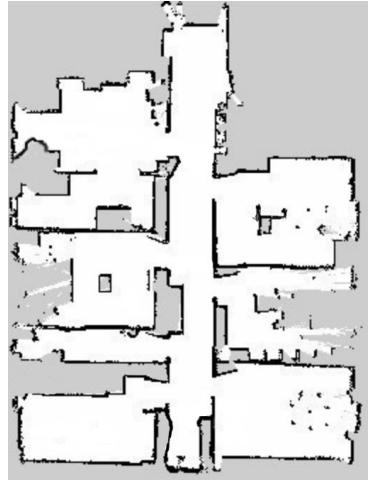


Figura 3.2: Esempio di occupancy grid map utilizzata dal modulo di navigazione del robot

3.2 Riconoscimento delle Stanze

In questa sezione verrà presentato il processo di riconoscimento delle stanze proposto da [14] e basato sull'algoritmo originale di Voronoi [2].

3.2.1 Segmentazione basata su Voronoi

Il riconoscimento delle stanze è un task affrontato molto spesso nel campo. Ci sono infatti innumerevoli approcci per affrontare questo problema [6]. Uno dei metodi più utilizzati è quello basato sull'algoritmo di Voronoi. Nel contesto di questa tesi, l'algoritmo di Voronoi [2] è molto utile poiché genera anche un grafo topologico dove ogni nodo rappresenta una regione individuata e ogni arco rappresenta la connessione tra due di queste regioni. Questo è molto utile come base di partenza per la generazione del grafo delle stanze.

L'obiettivo dell'algoritmo di Voronoi è quello di decomporre le grid maps in un piccolo insieme di regioni separate da passaggi stretti e le porte, rappresentate da linee critiche. Questo processo è composto da tre fasi principali:

- Binarizzazione: trasformazione della occupancy grid map in una mappa binaria; le celle con valori superiori della soglia sono considerate spazio libero C . Tutte le altre come spazio occupato \bar{C} .
- Costruzione del Diagramma di Voronoi: Per ogni punto $p = \langle x, y \rangle \in C$, esistono uno o più punti vicini $s \in \bar{C}$. Questi punti vengono chiamati *punti base* di $\langle x, y \rangle$ e la distanza tra $\langle x, y \rangle$ e i suoi punti base come *clearance* di $\langle x, y \rangle$

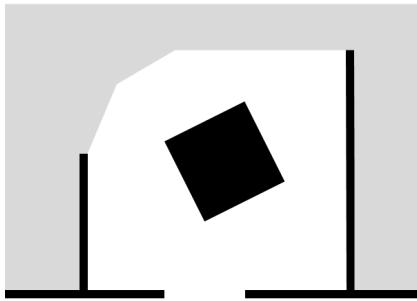
$$\text{punti base} = \{s \mid s \in \bar{C}, \text{dist}(p, s) = \min_{s \in \bar{C}} \{\text{dist}(p, s)\}\} \quad (3.1)$$

$$\text{clearance}(p) = \min_{s \in \text{punti base}} \{\text{dist}(p, s)\} \quad (3.2)$$

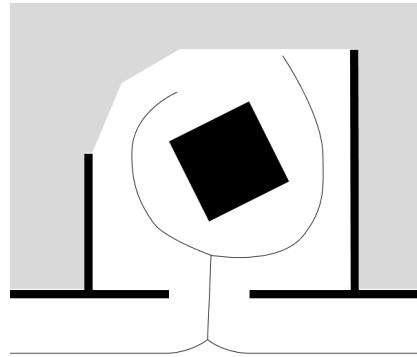
Il diagramma di Voronoi è l'insieme dei punti $p \in C$ che hanno almeno due diversi punti equidistanti base;

- Identificazione dei punti critici: punti $p \in C$ che minimizzano la (clearance) localmente. In teoria corrispondono ai passaggi stretti e alle porte.

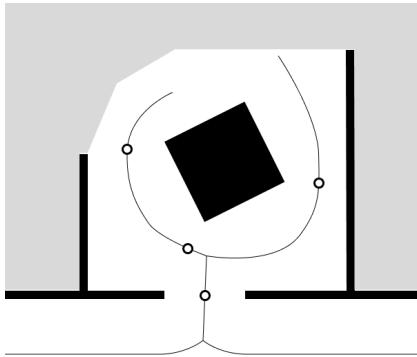
L'approccio proposto da [14], e utilizzato in questo lavoro, si basa sugli stessi concetti base dell'algoritmo di Voronoi, ma con alcune modifiche per migliorare la qualità dei risultati. In particolare, analizza sia lo spazio libero, ricercando i punti critici che chiama *intersection points*, sia lo spazio occupato, ricercando gli *endpoints* di ogni ramo del diagramma di Voronoi dello spazio occupato. Questi dati sono successivamente utilizzati per individuare le linee critiche e separare quindi le regioni.



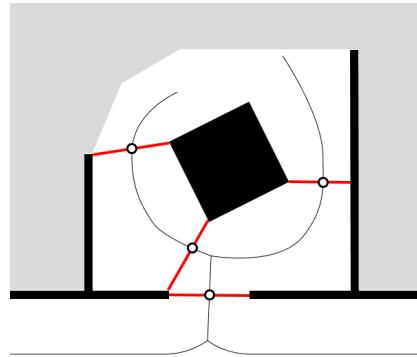
(a) Grid Map d'esempio



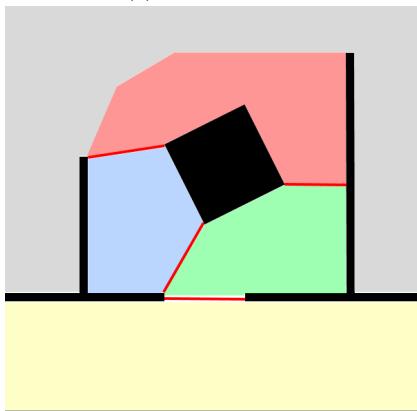
(b) Diagramma di Voronoi



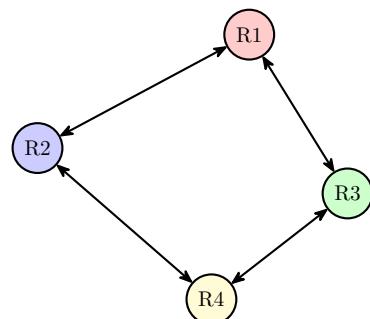
(c) Punti Critici



(d) Linee Critiche



(e) Regioni



(f) Grafo topologico

Figura 3.3: Esempio di riconoscimento delle stanze tramite l'algoritmo di Voronoi originale [2]

3.2.2 Preprocessing

In questa fase, l'immagine della mappa viene binarizzata e si cerca di limitare il rapporto Signal-to-Noise (SNR) per migliorare la qualità dei risultati.

Binarizzazione

L'immagine della mappa viene binarizzata due volte, una volta per la generazione della maschera dello spazio libero e una volta per la generazione della maschera per lo spazio occupato. Questo processo è necessario per poter analizzare separatamente i due spazi.



(a) Immagine binaria dello spazio libero



(b) Immagine binaria dello spazio occupato

Morfologia matematica

In questa fase vengono applicate delle operazioni di morfologia matematica per ridurre il rumore. In base alla tipologia di spazio analizzato, vengono applicate operazioni diverse, anche per accettuare elementi diversi.

Spazio libero Viene applicato un processo di erosione con un elemento strutturante elissoideale di dimensione 5x5. Successivamente si applica un filtro mediano 5x5 per rimuovere il rumore sale e pepe e per smussare gli angoli.

Spazio occupato Viene applicato un processo di dilatazione con un elemento strutturante quadrato di dimensione 5x5.



(a) Immagine binaria dello spazio libero dopo il preprocessing



(b) Immagine binaria dello spazio occupato dopo il preprocessing

3.2.3 Analisi dello spazio libero

L'obiettivo di questa fase è quello di individuare i punti critici e le linee critiche che giacciono in corrispondenza dei passaggi stretti.

Diagramma di Voronoi

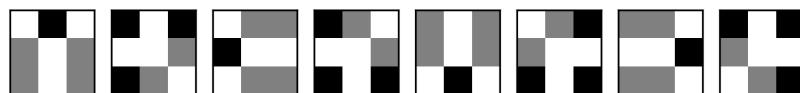
Innanzitutto si procede con il determinare il diagramma di Voronoi dello spazio libero. Nel paper originale viene utilizzato lo scheletro della mappa binaria come approssimazione di Voronoi. In questo lavoro, viene invece utilizzata la trasformata asse mediano poichè si è notato che fornisce risultati migliori.

Branching

Si individuano nel diagramma di Voronoi i punti di branching ovvero i punti dove il diagramma si biforca. Per individuarli si utilizza la trasformata HIT or MISS con i kernels 3x3 seguenti:



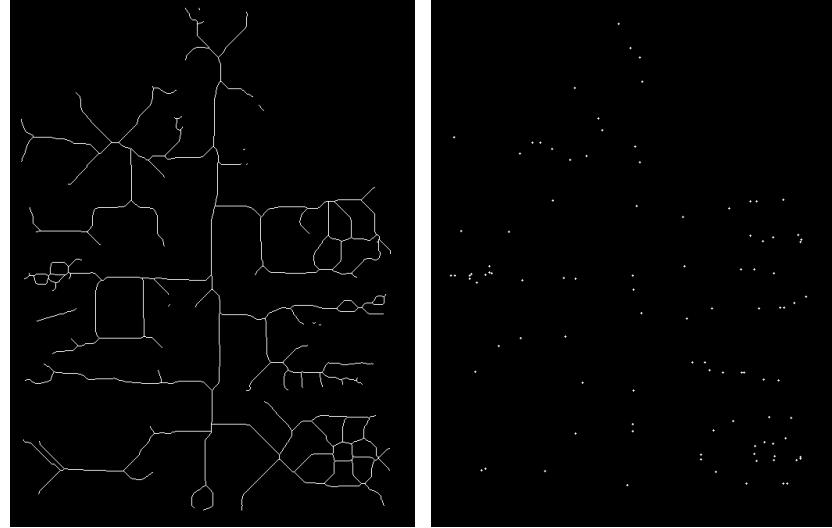
(a) Kernels per i branch a T



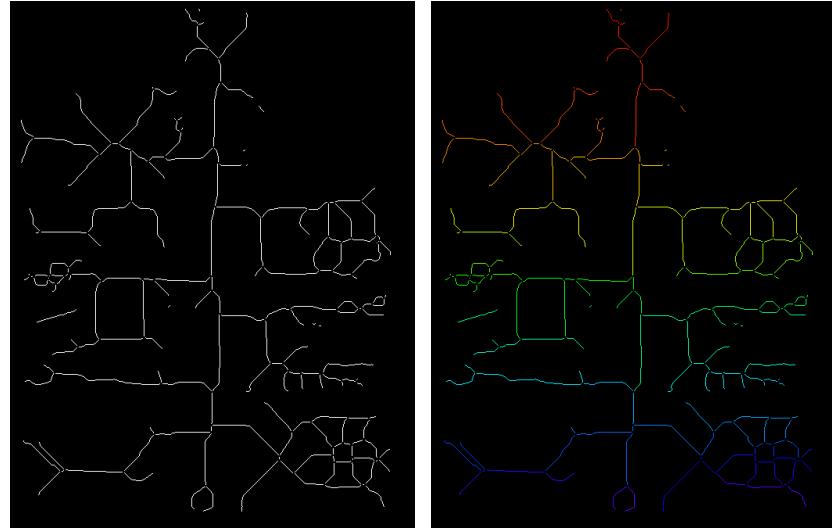
(b) Kernels per i branch a Y

Figura 3.6: In nero i valori -1, in grigio i valori 0 e in bianco i valori 1 dei kernels

Successivamente si procede con la rimozione dei punti di branch dal diagramma di Voronoi e si effettua il labeling delle componenti connesse, in modo da poter individuare ogni ramo del diagramma. I rami il cui numero di pixels è inferiore a dieci pixels vengono eliminati.



(a) Diagramma di Voronoi approssimato
(b) Punti di branch del diagramma di Voronoi



(c) Differenza tra il diagramma di Voronoi e i punti di branch
(d) Labeling delle componenti connesse dell'immagine (c)

Ricerca dei segmenti

Per ogni ramo del diagramma di Voronoi si procede con la ricerca dei segmenti. Un segmento è definito come l'insieme di punti del branch che hanno una clearance nell'intorno di due pixels rispetto alla clearance minima del branch.

$$\text{segmento} = \{p \mid |\text{clearance}(p) - \text{clearance}_{\min}| \leq 2\} \quad (3.3)$$

A supporto dell'individuazione dei segmenti, si calcola la trasformata distanza che assegna ad ogni pixel il valore della distanza minima da un pixel dello spazio occupato e si utilizza per calcolare la clearance di ogni punto del branch. Per ogni segmento si salva centroide e orientamento.

Calcolo orientamento del segmento Si trovano gli endpoints a e b (attraverso la trasformata HIT or MISS con kernels che verranno approfonditi nella sezione successiva) e si calcola l'orientamento del segmento come l'angolo tra la retta passante per gli endpoints e l'asse x.

In particolare, se la retta passante per i due punti non è verticale, si calcola $m = \frac{y_a - y_b}{x_a - x_b}$, coefficiente angolare della retta e si calcola l'orientamento come $\theta = \arctan(m)$.

$$\theta = \begin{cases} \arctan\left(\frac{y_a - y_b}{x_a - x_b}\right) & \text{se } x_a \neq x_b \\ \frac{\pi}{2} & \text{altrimenti} \end{cases} \quad (3.4)$$

È bene notare che il sistema di riferimento delle immagini è invertito rispetto a quello classico, con l'asse y che cresce verso il basso. Di conseguenza, la funzione arctan di *numpy* restituisce i valori diversi da quelli che si aspetterebbe.

Per esempio, nella figura sottostante, *numpy* restituisce l'angolo α invece di β . Per ovviare a questo problema, l'angolo finale viene calcolato come $\theta = \pi - \alpha$.

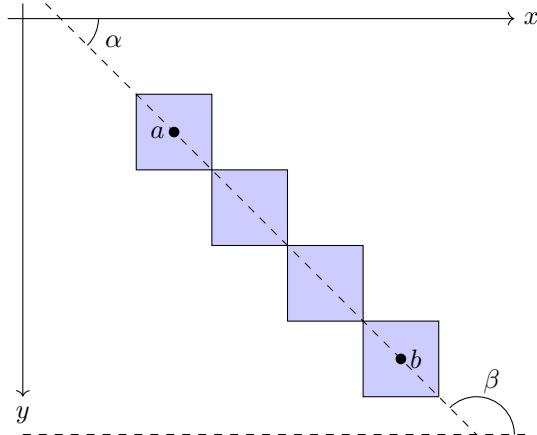


Figura 3.8: Calcolo dell'orientamento del segmento. I quadrati in blu rappresentano i pixel del segmento

Inoltre, α è positivo in verso orario e negativo in verso antiorario. In quest'ultimo caso, l'angolo finale viene calcolato come $\theta = -\alpha$.

$$\theta = \begin{cases} \pi - \alpha & \text{se } y_a < y_b \wedge x_a < x_b \\ -\alpha & \text{altrimenti} \end{cases} \quad (3.5)$$

Ricerca degli intersection points

Un intersection point è definito come il punto nello spazio occupato più vicino al centroide di un segmento lungo la direzione perpendicolare all'orientamento del segmento.

A supporto dell'individuazione degli intersection points, si utilizza la trasformata distanza per determinare di quanto scalare il vettore perpendicolare al segmento per trovare il punto più vicino nello spazio occupato. Gli intersection points vengono dunque calcolati come:

$$ip_1 = \begin{bmatrix} x_{\text{centroid}} \\ y_{\text{centroid}} \end{bmatrix} + \begin{bmatrix} \cos(\theta + \frac{\pi}{2}) \\ \sin(\theta + \frac{\pi}{2}) \end{bmatrix} \cdot \text{dist}(x_{\text{centroid}}, y_{\text{centroid}}) \quad (3.6)$$

$$ip_2 = \begin{bmatrix} x_{\text{centroid}} \\ y_{\text{centroid}} \end{bmatrix} + \begin{bmatrix} \cos(\theta - \frac{\pi}{2}) \\ \sin(\theta - \frac{\pi}{2}) \end{bmatrix} \cdot \text{dist}(x_{\text{centroid}}, y_{\text{centroid}}) \quad (3.7)$$

Per ogni segmento si determinano quindi gli intersection points i quali delimitano una possibile linea critica tra due regioni.



Figura 3.9: Segmenti e intersection points. In rosso i segmenti, in verde le linee critiche delimitate dagli intersection points, in blu il diagramma di Voronoi

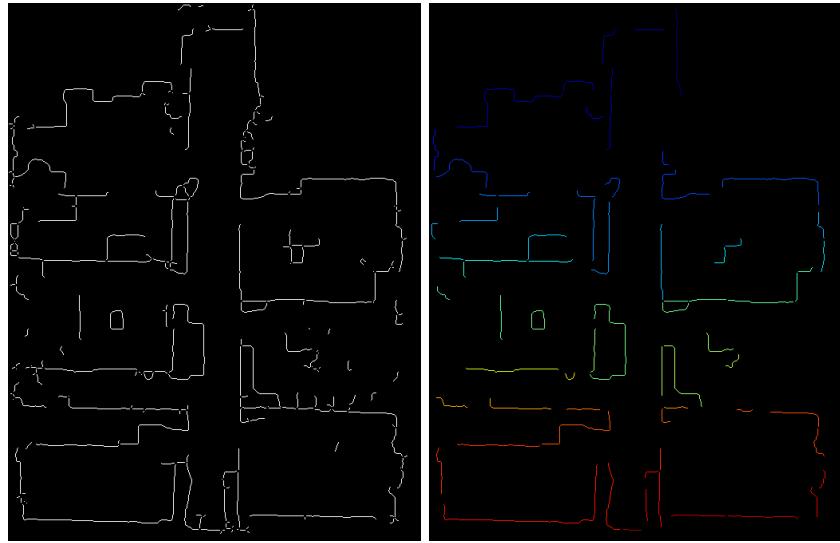
3.2.4 Analisi dello spazio occupato

L'obiettivo di questa fase è quello di individuare i cosiddetti endpoints, ovvero i punti che delimitano i rami del diagramma di Voronoi dello spazio occupato. Questi punti sono utili per scremare dalle linee critiche quelle che non sono effettivamente delle porte, come verrà illustrato nella sezione successiva.

Diagramma di Voronoi

Così come per lo spazio libero, anche per lo spazio occupato viene determinato il diagramma di Voronoi approssimato alla trasformata asse mediano. Si ottiene dunque una foresta di diagramma di Voronoi ognuno dei quali corrisponde ad una zona occupata diversa.

I diagrammi con meno di 20 pixels vengono scartati, in modo da rimuovere i



(a) Diagramma di Voronoi dello spazio occupato (b) Labeling delle componenti connesse di (a)

diagrammi relativi le piccole zone che normalmente corrispondono a errori di mappatura dovuti al rumore.

Ricerca degli endpoints

I diagrammi di voronoi estratti dallo spazio occupato sono utili per la ricerca delle protuberanze presenti vicino alle porte. Queste protuberanze sono individuate come endpoints, localizzati attraverso l'utilizzo della trasformata HIT or MISS con i kernels 3x3 seguenti:

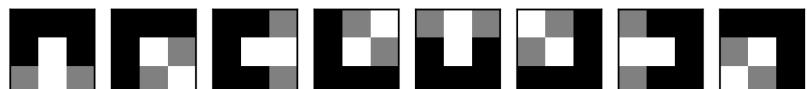


Figura 3.11: In nero i valori -1, in grigio i valori 0 e in bianco i valori 1 dei kernels

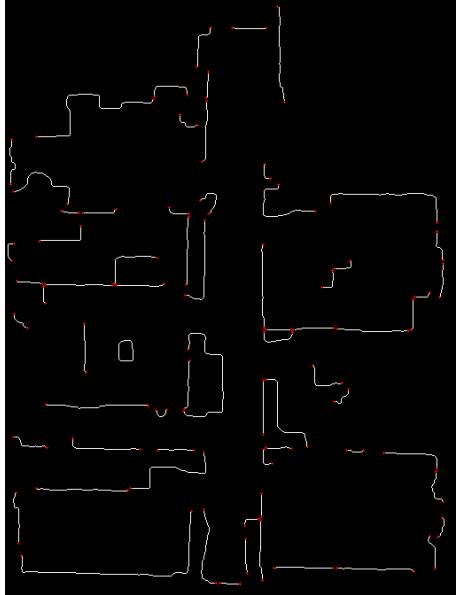


Figura 3.12: Diagramma di Voronoi dello spazio occupato con gli endpoints evidenziati in rosso

3.2.5 Ricerca delle porte

Le porte sono definite come passaggi stretti nello spazio libero dove vi è la presenza di protuberanze dello spazio occupato.

Utilizzando entrambe le tipologie di informazioni estrapolate negli step precedenti, è possibile individuare in modo abbastanza preciso le linee critiche che effettivamente dividono due stanze.

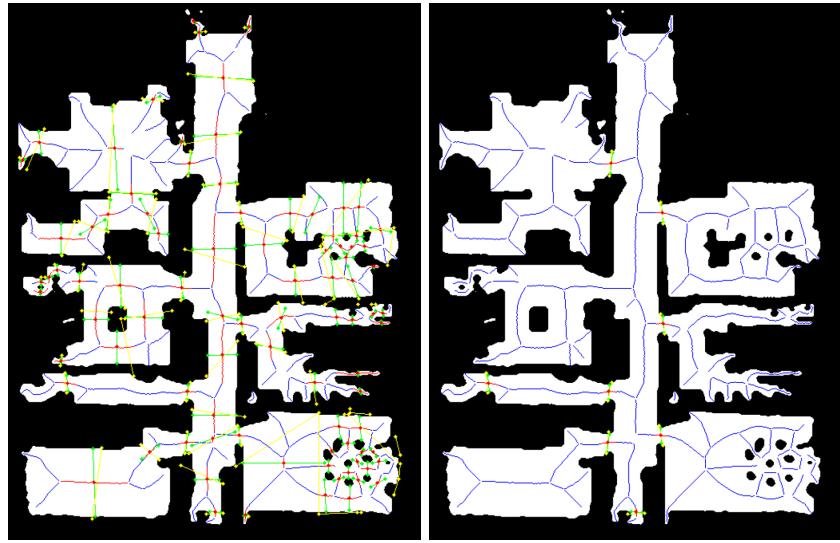
Il primo passo è quello di individuare, per ciascuno intersection point dello spazio libero, l'endpoint dello spazio occupato più vicino. Unendo gli endpoint ce_1, ce_2 più vicini rispettivamente al primo e al secondo intersection point, si ottiene un'ulteriore linea $\overline{ce_1ce_2}$.

Infine, se:

- $distance(ip_1, ce_1) \leq \frac{threshold}{2}$
- $distance(ip_2, ce_2) \leq \frac{threshold}{2}$
- $distance(ip_{centroid}, ce_{centroid}) \leq threshold$

e il segmento $\overline{ip_1ip_2}$ non interseca alcun diagramma di Voronoi dello spazio occupato, allora questo rappresenta una door location valida.

Il vantaggio di questa approccio è che la soglia $threshold$ non è fissa: rappresenta infatti ma viene calcolata per ogni mappa come la media dei valori della trasformata distanza dello spazio occupato nei punti dove è presente il suo diagramma di Voronoi, moltiplicata per due.



(a) Segmenti, intersection points e endpoints. In giallo i segmenti che congiungono gli endpointi più vicini
(b) Segmenti, intersection points e endpoints che soddisfano le condizioni e quindi sono elegibili come porte

3.2.6 Separazione delle stanze

Si procede infine con la separazione delle stanze. Per ogni segmento in prossimità di una porta valida, si crea la maschera della linea critica che viene sottratta alla maschera dello spazio libero. Successivamente si esegue il labeling delle componenti connesse, e dopo aver filtrato le stanze con area inferiore a trenta pixels, si ottengono le stanze separate.

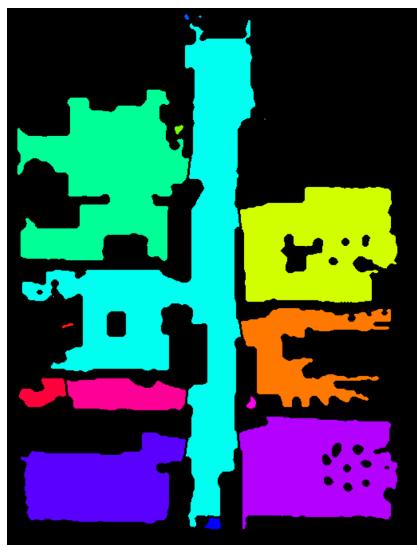


Figura 3.14: Stanze separate

Come si può notare, ci sono delle incorrettezze. Tuttavia, queste posso-

no essere corrette con l'ausilio dell'utente, come verrà illustrato nella sezione successiva.

3.2.7 Generazione Grafo delle Stanze

Il grafo delle stanze viene generato a partire dalle stanze separate e dai segmenti validi.

Viene creato un nodo per ogni stanza individuata. La lista di segmenti rappresentante il perimetro della stanza viene determinata approssimando il contorno della maschera della stanza con l'algoritmo di Ramer–Douglas–Peucker implementato in OpenCV.

Per ogni segmento valido, si controlla la label di appartenenza degli estremi del segmento che teoricamente giacciono in due stanze diverse, e si crea un arco tra i due nodi delle stanze corrispondenti.

Infine se eliminano i nodi che non sono connessi a nessun arco, rimuovendo così le stanze isolate che per definizione non dovrebbero esistere.

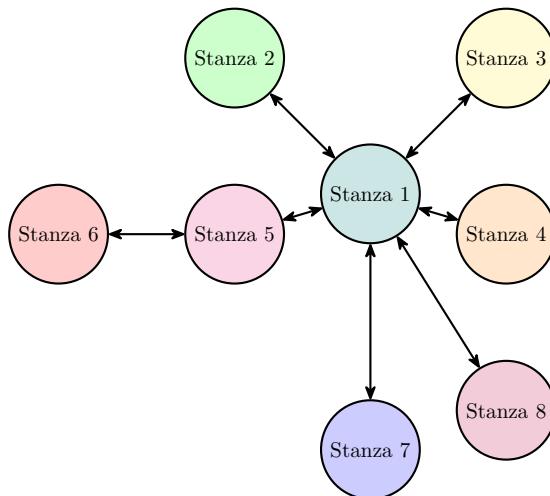


Figura 3.15: Grafo delle stanze generato a partire dalle stanze separate

3.3 Visualizzazione e Modifica delle Stanze

Affinchè l'utente possa visualizzare i risultati del riconoscimento e modificare la divisione in stanze aggiungendone di nuove, eliminandone o modificandone di esistenti, è stato sviluppato un tool grafico all'interno della webapp già esistente per la gestione di Robee, la cosiddetta *console*. Il funzionamento della console esula dagli obiettivi di questa tesi, tuttavia è importante illustrare come è stata integrata la funzionalità di visualizzazione e modifica delle stanze all'interno di essa.

3.3.1 Layers e modelli

A supporto di tale feature sono stati creati dei modelli per rappresentare un generico *layer* della mappa, anche in vista di futuri sviluppi, composto da una lista di *layer area* che rappresentano delle zone all'interno della mappa. Infine la classe *layer area outline* rappresenta il contorno di una layer area.

Layer	LayerArea	LayerAreaOutline
+ enabled : bool + filled : bool + name : string + areas : List[LayerArea]	+ id : string + label : string + outline: LayerAreaOutline + color : string	+ points : List[Position]

Figura 3.16: Classi per la gestione dei layer

Per la gestione delle stanze, ogni layer area rappresenta il poligono della stanza.

3.3.2 Schema di funzionamento

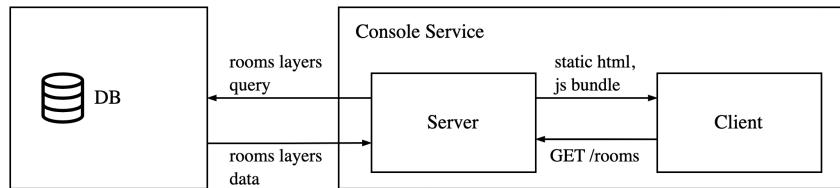


Figura 3.17: Schema di funzionamento per la visualizzazione e modifica delle stanze

Console Service è la webapp per la gestione di Robee, sviluppato con Python, Tornado, Turbo e Stimulus JS e che sfrutta il pattern MVC.

Quando un utente visita la pagina per la visualizzazione di una mappa, il server scarica l'immagine della mappa e costruisce l'html dal template iniettando i dati necessari e restituendo la pagina all'utente.

Successivamente, quando questo seleziona la visualizzazione del layer delle stanze dal menu a tendina, il client effettua una chiamata http GET /rooms al server che esegue una query al db, costruisce i modelli e li restituisce al client che renderizza i poligoni delle stanze in un canvas sovrapposto all'immagine della mappa.

3.3.3 Features

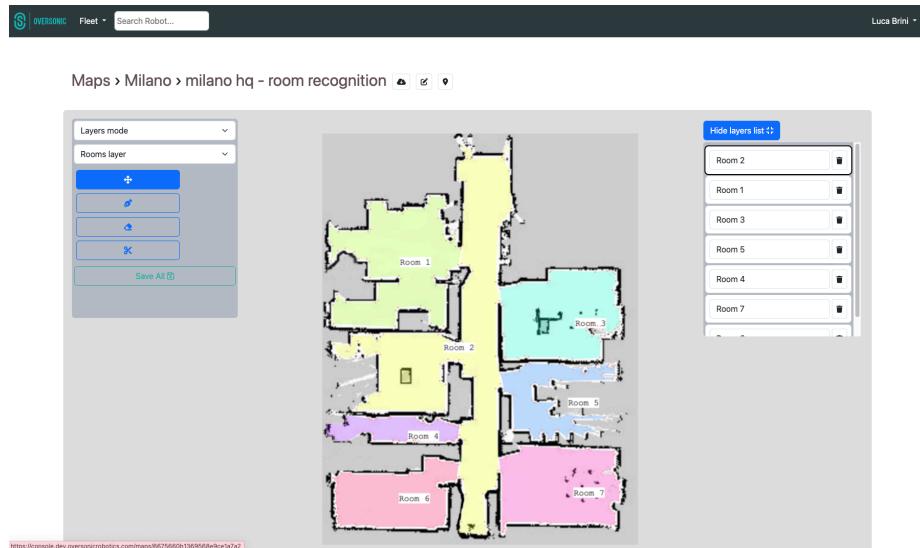
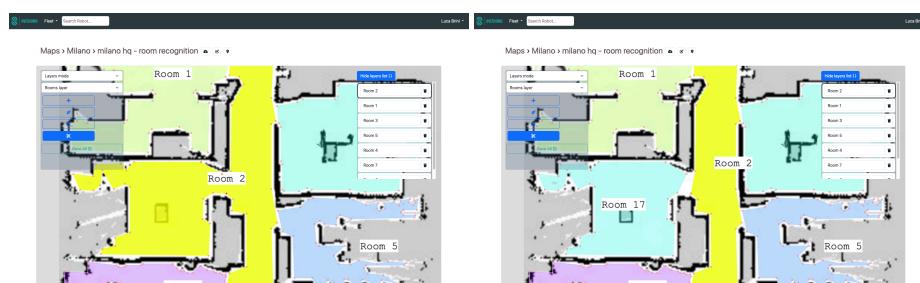


Figura 3.18: Pagina di visualizzazione delle stanze

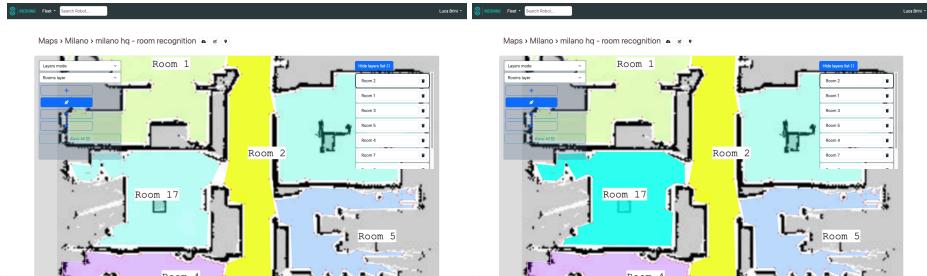
Taglio di una stanza

Selezionando la modalità *cut* e cliccando su due punti del poligono, si effettua un taglio della stanza in due nuove stanze.



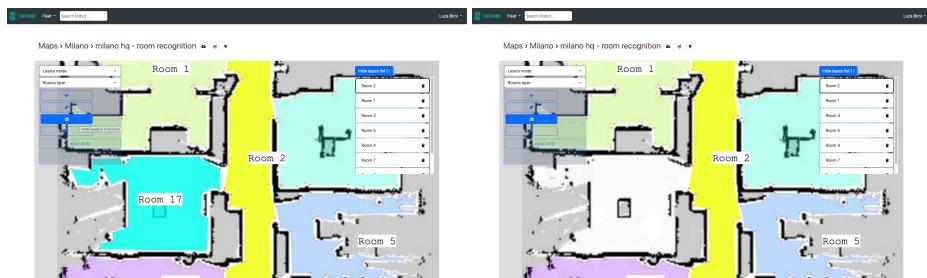
Nuovo punto nel poligono di una stanza

Selezionando la modalità *add*, dopo aver selezionato un punto del poligono con un doppio click è possibile creare un nuovo punto successivo a quello selezionato precedentemente.



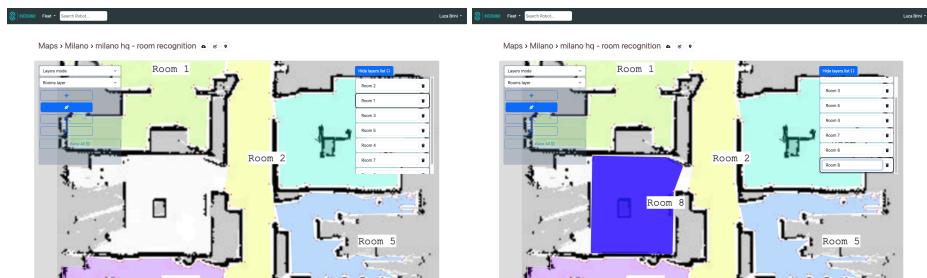
Eliminazione di una stanza

Selezionando la modalità *delete*, è possibile eliminare una stanza cliccando su una qualsiasi parte di essa



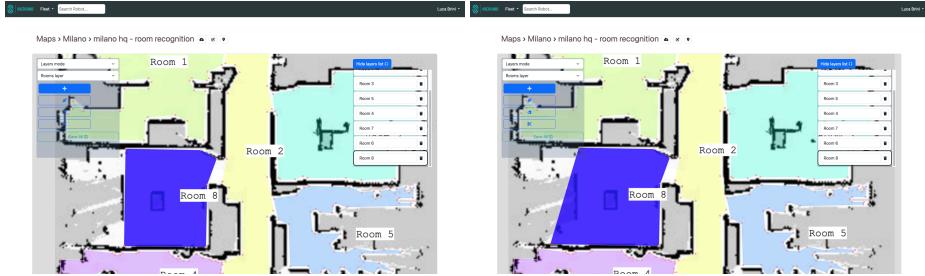
Creazione di una nuova stanza

Selezionando la modalità *add*, senza aver selezionato un qualsiasi punto di una stanza è possibile creare una nuova stanza facendo doppio click su un qualsiasi spazio vuoto della mappa. Non sono ammessi sovrapposizioni.



Spostamento di un punto del poligono di una stanza

Selezionando la modalità *move*, è possibile spostare un punto del poligono della stanza cliccando su di esso e trascinandolo.



3.4 Conclusioni

In questo capitolo è stato affrontato il processo di riconoscimento delle stanze a partire dalle mappe SLAM generate dai sensori LIDAR del robot. In particolare, è stato presentato il processo di individuazione proposto da [14] e basato su Voronoi [2] che, prima dell'implementazione sembrava molto preciso ed efficace ma che tuttavia così non si è rivelato. Per questo motivo è stato sviluppato uno strumento che oltre alla classica visualizzazione dei risultati consente anche di modificare la disposizione delle stanze, come i classici strumenti di annotazione delle immagini. Le annotazioni delle stanze corrette, in futuro, potranno essere utilizzati come dati di training di un modello ML apposito.

Capitolo 4

Conclusioni

Questo lavoro di tesi ha avuto come obiettivo principale la realizzazione di un sistema per percepire l'ambiente e costruire la conoscenza acquisita frame dopo frame. Il sistema è stato progettato per RoBee, robot umanoide cognitivo di Oversonic Robotics.

Come illustrato nei capitoli precedenti, il modello di PSGTr consente di acquisire informazioni sugli oggetti, e le relazione tra loro, presenti nell'ambiente in modo efficiente, caratteristica essenziale per un'applicazione real time di questo tipo. L'integrazione delle nuove informazioni sfrutta operazioni di geometria lineare, che si traducono in una serie di moltiplicazioni e addizioni molto veloci da eseguire per un calcolatore. Tuttavia, un miglioramento in questo senso si potrebbe ottenere utilizzando altri modelli che, sfruttando l'object tracking, consentono di ridurre il numero di operazioni necessarie per aggiornare la mappa semantica. Inoltre si potrebbe pensare di utilizzare un database a grafo, al posto dell'attuale MongoDB, per memorizzare la mappa semantica e sfruttare così le capacità di query offerte da un database di questo tipo. Un ulteriore miglioramento potrebbe essere quello di eseguire un fine tuning del modello PSGTr in base al contesto di utilizzo finale di RoBee.

Il sistema di riconoscimento delle stanze, basato sull'Algoritmo di Voronoi, consente di estrapolare a partire da una grid map di occupazione il grafo delle stanze della mappa semantica. I risultati ottenuti sono stati abbastanza soddisfacenti, per le varie mappe a disposizione di Oversonic. Tuttavia, non mancano gli errori che, grazie allo strumento di modifica delle stanze, possono essere innanzitutto corretti e in futuro si potranno utilizzare le informazioni ottenute per addestrare un modello ad-hoc per il riconoscimento delle stanze.

Appendice A

Appendice

A.1 RoBee

RoBee è un robot umanoide made in Italy, sviluppato e costruito a Besana Brianza dall'azienda Oversonic Robotics presso la quale ho svolto la mia esperienza di tirocinio.

È un robot la cui altezza varia dai 135 ai 200 cm in funzione della configurazione dei giunti delle gambe, ha 39 gradi di libertà e un ingombro in pianta di 65 cm quadrati. L'estensione del braccio può arrivare fino a 80 cm e può navigare fino ad una velocità di 1.2 m/s. È dotato di certificazioni industriali e di sicurezza.



Figura A.1: Robee in versione 18

A.1.1 Architettura Cloud Native

L’architettura di RoBee è basata su microservizi la cui orchestrazione è gestita da Kubernetes. Ci sono due tipologie di container:

- Moduli: pod che vengono eseguiti localmente su ogni Robot e gestiscono le varie funzionalità di base del robot, come la navigazione, la gestione dei giunti, dello streaming delle camera e così via. La gestione dei pod sulle macchine locali è gestita da KubeEdge
- Servizi: pod che vengono eseguiti sul cloud e gestiscono funzionalità più generali e che non sono strettamente legate al singolo robot. Un esempio di servizio è *scene perception*, il modulo sviluppato durante il tirocinio il cui funzionamento è descritto in questa tesi.

RoBee è dotato di innumerevoli moduli e servizi ma in questa sezione verrano illustrati brevemente solo quelli che sono strettamente legati al funzionamento di *scene perception* e di questa tesi.

A.1.2 Dashboard and Console

Dashboard e Console sono due servizi utilizzati per la gestione dei robot, delle mappe, delle missioni e tante altre funzionalità.

Entrambe le applicazioni web sono state sviluppate con lo stack Python, Tornado, Turbo e Stimulus JS.

Dashboard

La dashboard è lo strumento principale per gli sviluppatori e gli utentidi Oversonic per la gestione dei moduli e dei servizi. Attraverso la dashboard è possibile:

- Aggiornare i Moduli e i Servizi;
- Monitorare i log dei Servizi;
- Disabilitare e abilitare i Moduli per ogni robot;
- Disabilitare e abilitare i Servizi e scegliere per ogni robot verso quale servizio “puntare”: è infatti possibile per debug far puntare un robot ad un servizio diverso in esecuzione sui PC degli sviluppatori piuttosto che a quello di produzione;
- Modificare le configurazione dei Robot, Moduli o Servizi.

Console

La console è lo strumento principale per gli utenti finali per la gestione dei robot e di tutto ciò che lo riguarda. Attraverso la console è possibile:

- Monitorare la telemetria dei robot in tempo reale;
- Programmare missioni attraverso un tool visuale;
- Visualizzare la mappa e modificarne i layer semanticci, tra cui quello delle stanze oggetto del capitolo 3;

- Impostare goal nella mappa;
- Creare, Modificare o Eliminare le mappe;
- Muovere manualmente il robot;
- Visualizzare il feed proveniente dalle telecamere.

A.1.3 Mappe, navigazione e LiDaR

Il sistema di navigazione di RoBee è basato su ruote omnidirezionali la cui gestione avviene attraverso navigation module. Attraverso l'utilizzo dei LiDar è possibile mappare l'ambiente, localizzarsi e prevenire movimenti che potrebbero causare danni al robot o all'ambiente.

Il modulo è il responsabile della creazione della grid map di occupazione: riceve i dati, costruisce la mappa ed effettua una chiamata http ad API Service per effettuare il riconoscimento delle stanze e salvarla sul database.

A.1.4 Giunti e controllo

La gestione dei giunti è affidata a Joints Module, che si occupa di tener traccia dello stato, posizione e rotazione, di ogni giunto del robot nel tempo. Inoltre gestisce l'albero delle TF: soddisfa le richieste degli altri moduli, tramite MQTT, per ottenere la matrice di trasformazione tra due sistemi di riferimento.

A.1.5 Camere e point cloud

Robee è dotato di 4 telecamere base: *head*, *navigation*, *back* e *face recognition*. In aggiunta a queste, è possibile montare telecamera sul braccio apposite per gli end effectors.

La gestione delle camere è affidata a streaming module che si occupa di leggere i frame dalle telecamere e streammarli su Redis per renderli disponibili ai moduli o servizi che ne fanno richiesta. Per esempio, scene perception si interfaccia al pod Redis eseguito on edge su un robot per poter leggere i frame provenienti dalla camera selezionata per fare inferenza.

In base alla tipologia di modello di telecamera, è possibile ottenere anche la point cloud, ovvero una rappresentazione tridimensionale dell'ambiente circostante il robot.

A.2 Codice

È possibile trovare il codice sorgente di questa tesi e di parte del codice del lavoro su GitHub al seguente indirizzo:

<https://github.com/lucabrini/environment-perception-thesis>

Glossario

albero delle TF Struttura ad albero che descrive le relazioni spaziali tra i diversi componenti di un robot. Ogni nodo dell'albero rappresenta un sistema di coordinate, e ogni ramo rappresenta una trasformazione (rotazione e traslazione) che collega un sistema di coordinate al suo sistema padre. 15

backbone Architettura di rete neurale pre-addestrata che funge da base per ulteriori sviluppi e adattamenti specifici di una particolare applicazione. Grazie al transfer learning è possibile costruire architetture per task complessi sopra a questo modello. Utilizzata molto spesso nella visione artificiale come supporto . 13

KubeEdge Progetto open source che estende Kubernetes da un cluster cloud a un cluster edge, permettendo di eseguire i pod su dispositivi edge come robot, macchine industriali e così via. 46

matrice di trasformazione affine Matrice di trasformazione 4x4 che combina rotazione traslazione.

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Dove:

- $R_{11}, R_{12}, R_{13}, \dots, R_{33}$ sono gli elementi della matrice di rotazione 3x3.
- T_x, T_y, T_z sono le componenti del vettore di traslazione.

Attraverso una sola moltiplicazione di matrici è possibile applicare sia la rotazione che la traslazione ad un punto (x, y, z)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Il risultato (x', y', z') rappresenta le nuove coordinate del punto dopo la trasformazione . 14, 15

point cloud Raccolta di dati che rappresenta oggetti o superfici tridimensionali. Ogni punto nella nuvola ha coordinate (x, y, z) che ne definiscono la posizione nello spazio. A volte, ai punti sono associati anche altri attributi, come colore o intensità . 14

sistema camera Sistema di coordinate che prevede la camera nell'origine, solitamente al centro. Tutte le coordinate espresse in questo sistema hanno quindi come riferimento la posizione della camera. 14

sistema pixel Sistema di coordinate intere 2D con l'origine in alto a sinistra.

14

Bibliografia

- [1] E. Haines, «I.4. - Point in Polygon Strategies,» in *Graphics Gems*, P. S. Heckbert, cur., Academic Press, 1994, pp. 24–46, ISBN: 978-0-12-336156-1. DOI: <https://doi.org/10.1016/B978-0-12-336156-1.50013-6>.
- [2] S. Thrun, «Learning Metric-Topological Maps for Indoor Mobile Robot Navigation,» *Artificial Intelligence*, vol. 99, n. 1, pp. 21–71, 1998.
- [3] C. Theobalt, J. Bos, T. Chapman et al., «Talking to Godot: Dialogue with a Mobile Robot,» ago. 2002.
- [4] C. Galindo, A. Saffiotti, S. Coradeschi, P. Buschka, J. Fernandez-Madrigal e J. Gonzalez, «Multi-hierarchical semantic maps for mobile robotics,» in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 2278–2283. DOI: 10.1109/IROS.2005.1545511.
- [5] T.-Y. Lin, M. Maire, S. Belongie et al., *Microsoft COCO: Common Objects in Context*, 2014.
- [6] R. Bormann, F. Jordan, W. Li, J. Hampp e M. Hägle, «Room segmentation: Survey, implementation, and analysis,» in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1019–1026. DOI: 10.1109/ICRA.2016.7487234.
- [7] R. Krishna, Y. Zhu, O. Groth et al., «Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations,» *International Journal of Computer Vision*, vol. 123, mag. 2017. DOI: 10.1007/s11263-016-0981-7.
- [8] A. Vaswani, N. Shazeer, N. Parmar et al., «Attention Is All You Need,» *CoRR*, 2017.
- [9] D. Xu, Y. Zhu, C. B. Choy e L. Fei-Fei, «Scene Graph Generation by Iterative Message Passing,» in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [10] D. A. Hudson e C. D. Manning, «GQA: a new dataset for compositional question answering over real-world images,» *CoRR*, 2019.
- [11] Y. Liang, Y. Bai, W. Zhang, X. Qian, L. Zhu e T. Mei, «Rethinking Visual Relationships for High-level Image Understanding,» *CoRR*, 2019.
- [12] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov e S. Zagoruyko, «End-to-End Object Detection with Transformers,» *CoRR*, 2020.
- [13] C. Zou, B. Wang, Y. Hu et al., «End-to-End Human Object Interaction Detection with HOI Transformer,» *CoRR*, 2021.

- [14] A. Mora, A. Prados e R. Barber, «Segmenting Maps by Analyzing Free and Occupied Regions with Voronoi Diagrams,» giu. 2022.
- [15] J. Yang, Y. Z. Ang, Z. Guo, K. Zhou, W. Zhang e Z. Liu, «Panoptic Scene Graph Generation,» in *ECCV*, 2022.
- [16] J. Yang, W. Peng, X. Li et al., «Panoptic Video Scene Graph Generation,» in *CVPR*, 2023.
- [17] Y. Z. Runjia Tan Shanhe Lou e C. Lv, «Multi-modal LLM-enabled Long-horizon Skill Learning for Robotic Manipulation,» in *CIS-RAM*, 2024.