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

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, la struttura e come viene utilizzata per pianificare le missioni del robot. Successivamente entreremo nel dettaglio del **Grafo di Scena**, come viene generato e tenuto aggiornato con i cambiamenti dell'ambiente. Infine analizzeremo il **Riconoscimento delle Stanze** a partire dalla mappa SLAM 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.

Capitolo 1

Stato dell'Arte

La capacità di percepire l'ambiente e riconoscere oggetti è cruciale per i robot autonomi. Questi processi permettono ai robot di eseguire compiti complessi, come la navigazione e la manipolazione degli oggetti.

Capitolo 2

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, conscia 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 in 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.

2.1 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

2.1.1 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.

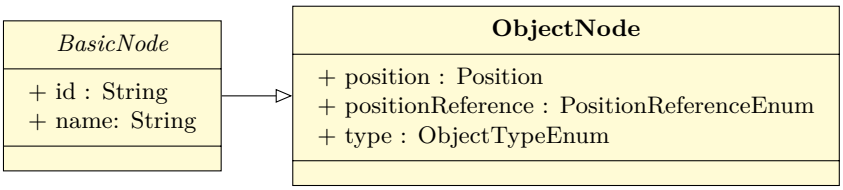


Figura 2.1: Diagramma delle classi - Nodo Oggetto

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: terna (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

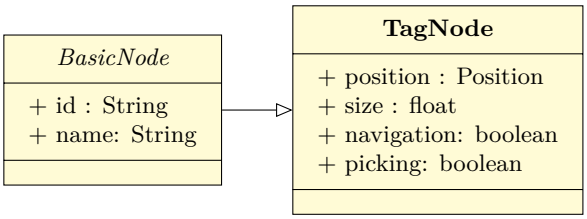


Figura 2.2: Diagramma delle classi - Nodo Tag

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.

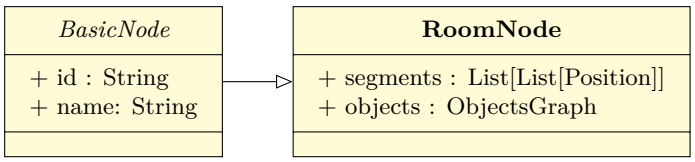


Figura 2.3: Diagramma delle classi - Nodo Stanza

2.1.2 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 quali vi è la necessità di rappresentare queste relazioni tra oggetti.

2.1.3 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 4, 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.

2.1.4 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 [10]. Successivamente, come verrà mostrato nel Capitolo 3, questa verrà fusa con la Mappa Semantica rendendo dunque la base di conoscenza coerente rispetto lo stato dell'ambiente attuale.

2.1.5 Esempio di Mappa Semantica

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

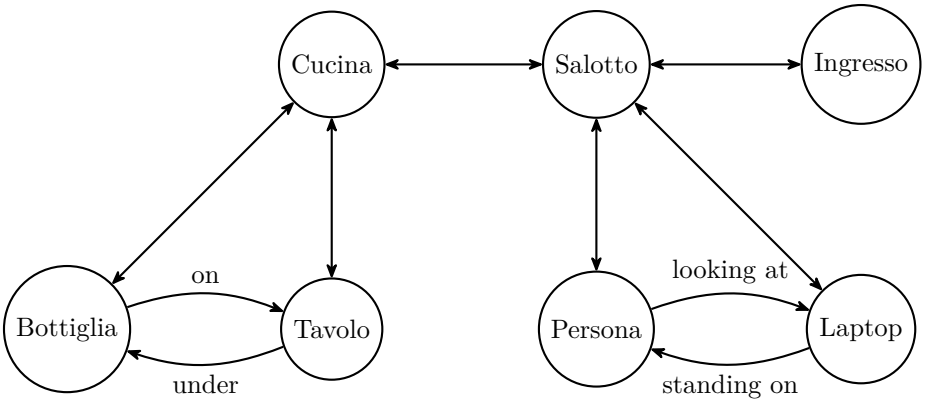
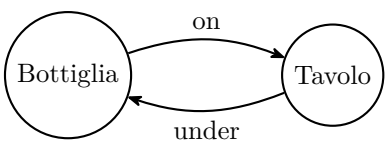
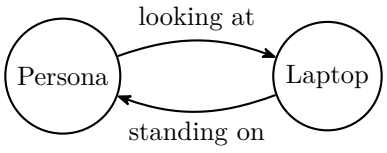


Figura 2.4: Mappa Semantica

(a) Grafo delle stanze





(b) Grafo degli oggetti della stanza Cucina (c) Grafo degli oggetti della stanza Salotto

2.2 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 3

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.

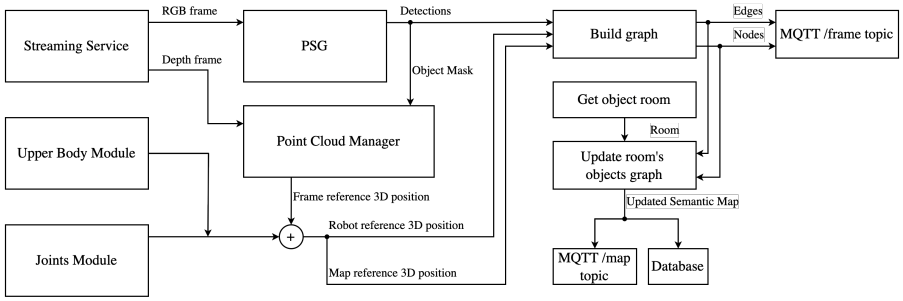


Figura 3.1: Schema dei flussi dati per la generazione del grafo di scena e aggiornamento della mappa semantica

3.1 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 composto 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.

3.1.1 Lettura del 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.

3.1.2 Inferenza

Ogni frame ricevuto dal feed video viene successivamente dato in input al modello PSGTr [10] 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 [10] è un modello di deep learning a singolo stato basato su architettura Transformer [4] 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 [2] e Visual Genome [3]. Per le relazioni hanno estratto e costruito un dataset di 56 predicati a partire da dataset come VG-150 [5], VrR-VG [7] and GQA [6].

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 [8] e HOI [9]. 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.

3.1.3 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

```
obj_classes ← [ ]
obj_masks ← [ ]
obj_labels_ids ← detectionResults.labels
for i = 0 to obj_labels_ids.length do
    obj_classes.append(PSG_CLASSES[obj_labels_ids[i]])
    obj_masks.append(detectionResults.masks[i])
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 cosí una posizione $P_{ipd} = (x_i, y_i, z_i)$.

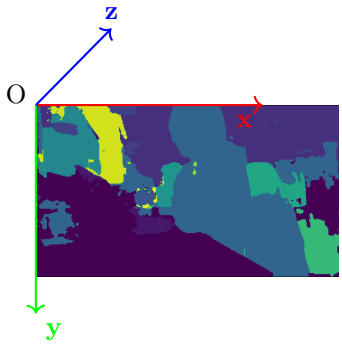


Figura 3.2: Sistema di coordinate pixel

Algorithm 2 Calcolo della posizione 3D nel sistema pixel

```
procedure GET_PIXEL_COORDS(depth_frame, obj_mask)
    z_ip ← median(depth_frame[obj_mask])          ▷ Point Cloud
    x_ip ← median(obj_mask[:, 0])
    y_ip ← median(obj_mask[:, 1])
    return x_ip, y_ip, z_ip
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 Intrinseca 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 Instrinseca 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

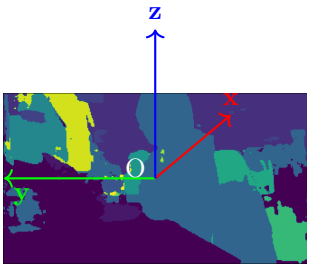


Figura 3.3: Sistema di coordinate camera

Algorithm 3 Calcolo della posizione 3D nel sistema camera

```
procedure GET_CAMERA_COORDS(P_ip)
    M_ic ← GET_CAMERA_INTRINSICS          ▷ Funzione libreria Robee
    P_ic ← M_ic^-1 × P_ip
    distance = norm(P_ic − P_camera)
    return P_ic, distance
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

```
procedure GET_MAP_COORDS(P_ic)
    M_ec ← GET_CAMERA_EXTRINSICS          ▷ Funzione libreria Robee
    P_im ← M_ec^-1 × P_ic
    return P_im
end procedure
```

Istanziamento dei nodi Con tutti i dati ora a dispozione 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

```
obj_ids ← [ ]
for i = 0 to obj_labels_ids.length do
    obj_pixel_coords ← GET_PIXEL_COORDS(depth_frame, obj_masks[i])
    obj_camera_coords, distance ← GET_CAMERA_COORDS(obj_pixel_coords)
    obj_coords ← GET_MAP_COORDS(obj_camera_coords)
    if distance > min_distance and distance < max_distance then
        node ← Node(i, obj_classes[i], obj_coords)
        obj_ids.append(i)
        semantic_scene.add_node(node)
        semantic_map.add_node(node)
    end if
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

```
for j = 0 to rel_labels.length do
    source_idx ← rel_pair_idxes[j][0]
    target_idx ← rel_pair_idxes[j][1]
    if source_idx in obj_ids and target_idx in obj_ids then
        source_node ← semantic_scene.get_node(source_idx)
        target_node ← semantic_scene.get_node(target_idx)
        if rel_dist[j] > rel_threshold then
            semantic_scene.add_edge(source_node, target_node, rel_labels[j])
        end if
    end if
end for
```

3.2 Aggiormento 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.

3.2.1 Trovare la stanza corrente del robot

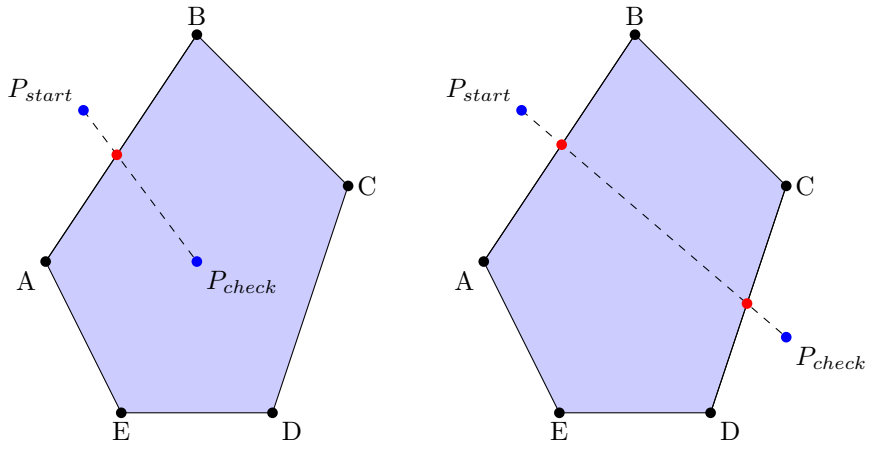
Trovare la stanza corrente del robot è uno step necessario: permette di aggiornare il Grafo degli Oggetti della sola la stanza in cui il robot si trova, evitando di aggiornare l'intera mappa, efficientando il processo.

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.



(a) Punto interno (b) Punto esterno

Algorithm 7 Ray Casting

```
1: procedure IS_INSIDE_POLYGON(P_start, P_check, segments)
2:     intersection_count ← 0
3:     for i = 0 to segments.length do
4:         A ← segments[i]
5:         B ← segments[(i + 1)%segments.length]
6:         if DO_INTERSECT(P_start, P_robot, A, B) then        ▷ Appendice A.2
7:             intersection_count ← intersection_count + 1
8:         end if
9:     end for
10:    return intersection_count%2 == 1
11: end procedure
```

Deteminare 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, poichè sia il riconoscimento delle stanze (*capitolo* 4) 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 ← robot.get_position()
2: current_room ← None
3: for s in semantic_map.get_rooms() do
4:     bb_x_min, bb_x_max, bb_x_min ← GET_BOUNDING_BOX(s.segments)
5:     P_start ← [MEAN(bb_x_min, bb_x_max), bb_y_min − ϵ]
6:     if IS_INSIDE_POLYGON(P_start, P_robot, s.segments) then
7:         current_room ← s
8:     end if
9: end for
```

3.2.2 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.

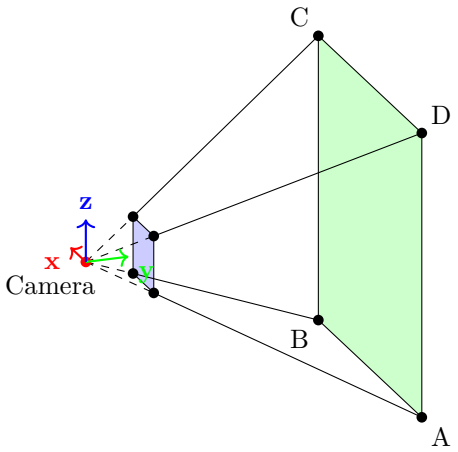


Figura 3.5: Camera Frustum. I vertici del piano verde rappresentano la proiezione degli estremi del frame nel sistema di coordinate mappa.

Il funzionamento alla base di questo processo è molto simile a quello affrontato nella sezione 3.1.3 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 P_{camera} normalizzato:

$$\vec{d} = (\vec{v} - \overrightarrow{P_{camera}})/norm(\vec{v} - \overrightarrow{P_{camera}}) \tag{3.1}$$

  - Viene calcolato il punto finale $\vec{v'}$ come la somma tra il vettore normalizzato $\vec{d}$ moltiplicato per uno scalare *distance* e $\overrightarrow{P_{camera}}$:

$$\vec{v'} = \vec{d} \cdot distance + \overrightarrow{P_{camera}} \tag{3.2}$$

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 ← [ ]
3:      M_ic ← GET_CAMERA_INTRINSICS
4:      M_ec ← GET_CAMERA_EXTRINSICS
5:      for i = 0 to frame_vertices.length do
6:          v⃗ ← M_ic^-1 × frame_vertices[i]
7:          v⃗' ← M_ec^-1 × v⃗
8:          d⃗ ← (v⃗' − P_camera⃗)
9:          d⃗_n ← d⃗/norm(d⃗)
10:         V ← d⃗_n * distance + P_camera⃗
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:

$$\begin{cases} \vec{n} = (\vec{v_1} - \vec{v_2}) \times (\vec{v_3} - \vec{v_2}) \\ d = \vec{n} \cdot \vec{v_1} = \vec{n} \cdot \vec{v_2} = \vec{n} \cdot \vec{v_3} \end{cases} \tag{3.3}$$

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.

$$\vec{n} = \begin{cases} \vec{n} & \text{se } \vec{n} \cdot \vec{c} - d \geq 0 \\ -\vec{n} & \text{altrimenti} \end{cases} \quad d = \begin{cases} d & \text{se } \vec{n} \cdot \vec{c} - d \geq 0 \\ -d & \text{altrimenti} \end{cases} \tag{3.4}$$

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.

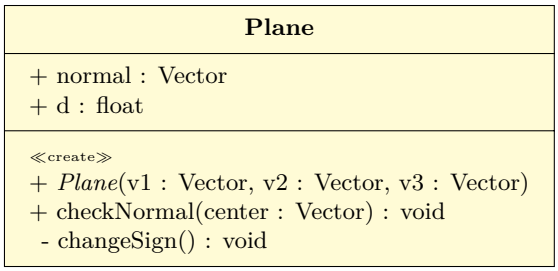


Figura 3.6: Classe Plane

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:

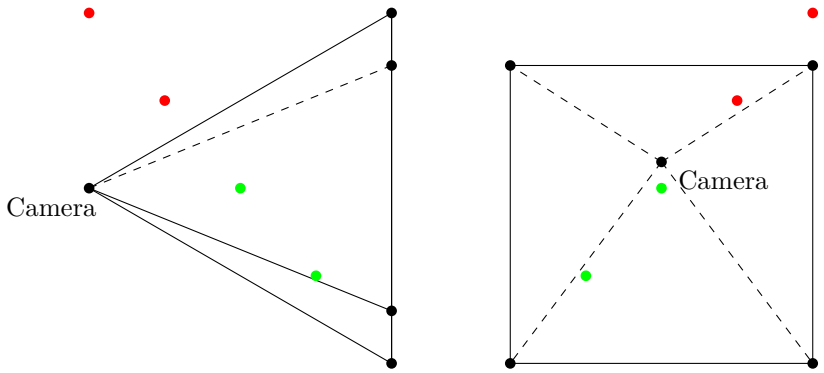
$$\vec{n} \cdot \vec{p} \geq d \tag{3.5}$$

Che in cardinate cartesiane diventa:

$$x_n \cdot x_p + y_n \cdot y_p + z_n \cdot z_p \geq d \tag{3.6}$$

Algorithm 10 Controllo appartenenza di punto al frustum

```
1:  procedure IS_INSIDE_FRUSTUM(P_i, planes)
2:      inside ← True
3:      for i = 0 to planes.length do
4:          n⃗ ← planes[i].normal
5:          d ← planes[i].d
6:          if n⃗ · P_i < d then
7:              inside ← False
8:          end if
9:      end for
10:     return inside
11: end procedure
```



(a) Prospettiva 1 (b) Prospettiva 2

Figura 3.7: 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

```
map_vertices ← PROJECT_CAMERA_FRUSTUM(P_camera, frame_vertices)
frustum_planes ← CREATE_FRUSTUM_PLANES(P_camera, map_vertices)
room_nodes ← currentRoom.get_nodes()
room_edges ← currentRoom.get_edges()
nodes_to_remove ← [ ]                                  ▷ Scelta nodi da rimuovere
for i = 0 to room_nodes.length do
    node ← room_nodes[i]
    if IS_INSIDE_FRUSTUM(node.position, frustum_planes) then
        nodes_to_remove.append(node)
    end if
end for
edges_to_remove ← [ ]                                  ▷ Scelta archi da rimuovere
for i = 0 to room_edges.length do
    edge ← room_edges[i]
    if nodes_to_remove[edge.source]ornodes_to_remove[edge.target] then
        edges_to_remove.append(edge)
    end if
end for
for i = 0 to nodes_to_remove.length do                 ▷ Rimozione nodi e archi
    currentRoom.remove_node(nodes_to_remove[i])
end for
for i = 0 to edges_to_remove.length do
    currentRoom.remove_edge(edges_to_remove[i])
end for
current.add_nodes(last_scene_graph.nodes)              ▷ Aggiunta nuovi risultati
current.add_edges(last_scene_graph.edges)
```

3.2.3 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.

3.3 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 istantaneamente.

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 [11] 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 4

Riconoscimento di Stanze

4.1 Analisi e risultati

4.2 Conclusioni

Capitolo 5

Analisi e Risultati

5.1 Errore della posizione degli oggetti

5.2 Punti di forza e svantaggi

5.2.1 Inferenza efficiente

5.2.2 Merging efficiente

Capitolo 6

Conclusioni e Sviluppi Futuri

6.1 Miglioramenti

6.1.1 Database a grafo

6.1.2 Finetuning OpenPSG

6.1.3 Object tracking

6.1.4 Utilizzo di OpenPVSG e Open4PSG

Appendice A

Appendice

A.1 RoBee System

A.1.1 Dashboard and Console

A.1.2 Infrastucture architecture, microservices and MQTT

A.1.3 Maps, navigation and LiDaRs

A.1.4 Joints and transformations

A.1.5 Cameras and point cloud

A.2 Codice

A.2.1 Costruzione Grafo di Scena

Costruzione Nodi

Costruzione Archi

A.2.2 Aggiornamento Mappa Semantica

Proiezione Camera Frustum

Controllo se l'oggetto appartiene al frustum

Recupero Stanza corrente Robot

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

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:

- $R_{11}, R_{12}, R_{13}, ..., R_{33}$ 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. indirizzo: https://www.sciencedirect.com/science/article/pii/B9780123361561500136.

[2] T.-Y. Lin, M. Maire, S. Belongie et al., *Microsoft COCO: Common Objects in Context*, 2014. arXiv: 1405.0312 [cs.CV].

[3] 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.

[4] A. Vaswani, N. Shazeer, N. Parmar et al., «Attention Is All You Need,» *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. indirizzo: http://arxiv.org/abs/1706.03762.

[5] 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.

[6] D. A. Hudson e C. D. Manning, «GQA: a new dataset for compositional question answering over real-world images,» *CoRR*, vol. abs/1902.09506, 2019. arXiv: 1902.09506. indirizzo: http://arxiv.org/abs/1902.09506.

[7] Y. Liang, Y. Bai, W. Zhang, X. Qian, L. Zhu e T. Mei, «Rethinking Visual Relationships for High-level Image Understanding,» *CoRR*, vol. abs/1902.00313, 2019. arXiv: 1902.00313. indirizzo: http://arxiv.org/abs/1902.00313.

[8] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov e S. Zagoruyko, «End-to-End Object Detection with Transformers,» *CoRR*, vol. abs/2005.12872, 2020. arXiv: 2005.12872. indirizzo: https://arxiv.org/abs/2005.12872.

[9] C. Zou, B. Wang, Y. Hu et al., «End-to-End Human Object Interaction Detection with HOI Transformer,» *CoRR*, vol. abs/2103.04503, 2021. arXiv: 2103.04503. indirizzo: https://arxiv.org/abs/2103.04503.

[10] J. Yang, Y. Z. Ang, Z. Guo, K. Zhou, W. Zhang e Z. Liu, «Panoptic Scene Graph Generation,» in *ECCV*, 2022.

[11] J. Yang, W. Peng, X. Li et al., «Panoptic Video Scene Graph Generation,» in *CVPR*, 2023.