Università degli Studi di Firenze

Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione

Tesi di Laurea Triennale in Ingegneria Informatica

Estrazione di informazioni contestuali da oggetti in planimetrie

Candidato
Francesco Amorosini

Relatore
Prof. Simone Marinai

Anno Accademico 2018-2019

Indice

Introduzione

In questo doumento viene presentato un procedimento atto ad **estrarre informazione contestuale da dataset di planimetrie**. Si procederà innanzitutto fornendo la definizione di contesto per un oggetto di una planimetria, dandone una rappresentazione tramite una struttura a grafo, e descrivendo tutti i passi che ci permetteranno di estrarlo. In particolare ci soffermeremo sul **riconoscimento dei muri** tramite l'elaborazione dell'immagine, che come vedremo è di fondamentale importanza per discriminare oggetti che appartengono a contesti diversi. L'accuratezza del riconoscimento dei muri verrà inoltre testata su un dataset di validazione. Le informazioni contenute nei grafi verranno aggregate in un dataset, grazie al quale potremo **inferire co-occorrenze di oggetti** in base alla loro frequenza e distanza. Infine, le informazioni ottenute verranno utilizzate per **definire sottoclassi di oggetti**, la cui riclassificazione verrà effettuata prima manualmente, ed in seguito tramite **l'addestramento di albero di decisione**.

Capitolo 1

Presentazione del problema

1.1 Definizione del contesto degli oggetti

Data una planimetria, per prima cosa bisogna stabilire quali oggetti al suo interno fanno parte di uno stesso contesto. Intuitivamente, il **contesto** di un oggetto deve contenere tutti gli oggetti nella sua prossimità che contengono informazioni utili per la sua classificazione. Un contesto è quindi un insieme di oggetti tra i cui elementi sussistono le seguenti proprietà:

- la loro distanza deve essere inferiore ad un certo valore.
- tra di essi non deve trovarsi alcun muro.

Per semplificare possiamo quindi dire che oggetti facenti parte dello stesso contesto si trovano nella stessa stanza.
Per la rappresentazione dei contesti è stata utlizzata una struttura a **grafo**, i cui nodi rappresentano gli oggetti e i cui archi indicano la sussistenza delle proprietà sopra elencate. Da tale definizione segue che, nel grafo di una planimetria, ad ogni **componente connessa** corrisponderà un contesto di

oggetti diverso. Osserviamo il frammento di planimetria nella Figura 1.1 e consideriamo ad esempio il letto colorato di rosa nella stanza centrale:

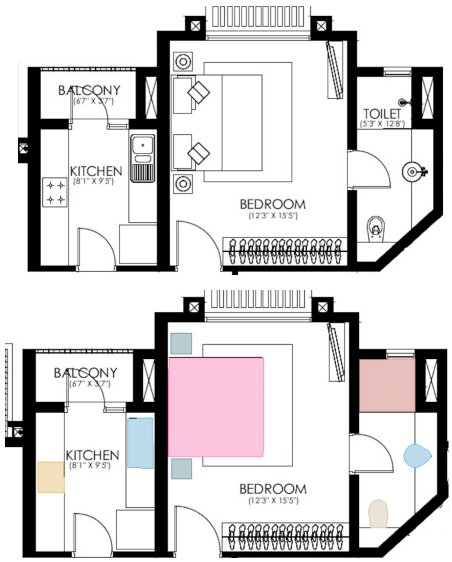


Figura 1.1: Un frammento di planimetria prima e dopo l'etichettatura degli oggetti al suo interno.

Il nostro obiettivo è fare in modo che il nodo ad esso associato sia collegato a quello dei due comodini, ma scollegato da ogni nodo riferito agli oggetti della cucina e del bagno: il grafo risultante dalla Figura 1.1 dovrà essere simile a quello illustrato nella Figura 1.2.

Figura 1.2: Grafo risultante degli oggetti nella Figura 1.1

1.2 Riconoscimento dei muri

Per verificare che tra due oggetti non si trovi alcun muro si potrebbe pensare di verificare che tra di essi (ad esempio tra i loro punti centrali) non si trovi alcun pixel nero dell'immagine. Questa soluzione si rivela in generale errata: quasi tutte le planimetrie contengono infatti dettagli, loghi, sketch, testo e rumore che rendono questa soluzione totalmente inaffidabile nella maggior parte dei casi. Per questo motivo, nel corso del progetto è sorta la necessità di sviluppare un algoritmo in grado di riconoscere i muri all'interno di una planimetria, in modo tale da ignorare tutti gli elementi che non servono al nostro scopo. Tale algoritmo funziona elaborando l'immagine tramite processi morfologici, ma la sua implementazione verrà discussa più avanti.

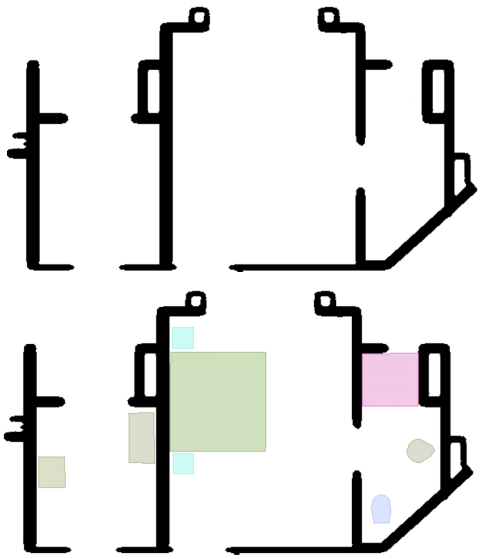


Figura 1.3: Applicazione del riconoscimento dei muri alla Figura 1.1.

Se avessimo elaborato la Figura 1.1 prima di procedere alla creazione del grafo avremmo ottenuto l'immagine 1.3, in cui si può facilmente verificare la presenza di muri tra due oggetti osservando il colore dei pixel che li separano.

L'algoritmo di rilevamento dei muri verrà testato su un dataset di validazione, in cui i muri sono stati precedentemente etichettati, in modo tale da poterne testare l'accuratezza.

1.3 Estrazione dei dati

Dal grafo di ogni planimetria verranno estratte informazioni riguardo la frequenza e la distanza di co-occorrenze di oggetti per ogni possibile classe, definendo eventualmente sottoclassi per particolari tipologie di oggetti. Grazie alle informazioni ottenute verranno costruiti dei **dataset** che, da una parte, permetteranno un'agevole visualizzazione dei risultati, mentre dall'altra saranno utili per addestrare **modelli di apprendimento** per automatizzare il processo di separazione degli oggetti in sottoclassi. Ad esempio in Figura 1.4 possiamo vedere le informazioni estratte dagli oggetti di tipo `chair`.

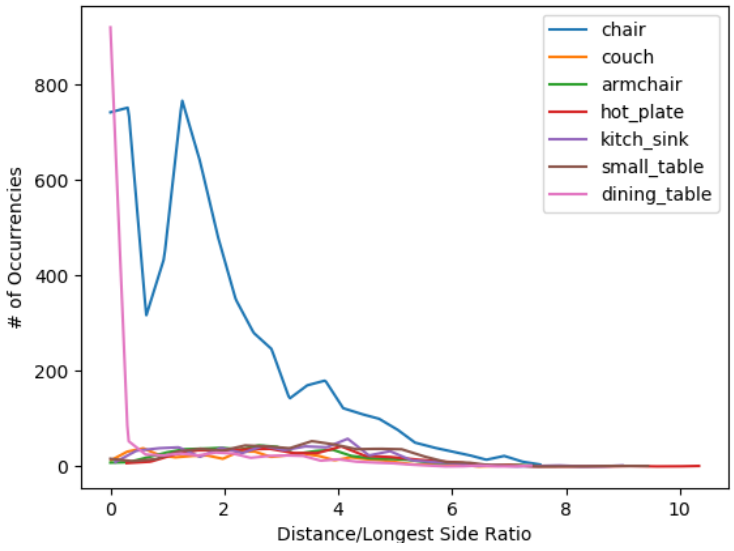


Figura 1.4: Visualizzazzione delle co-occerrenze per oggetti di tipo `chair`

Le curve riportate indicano quanto è frequente che un oggetto appartenente ad una data classe si trovi ad una certa distanza da un oggetto di tipo

`chair`. Dal grafico si evince che la grande maggioranza delle sedie si trova vicino ad un tavolo e nei pressi di altre sedie. Si noti in particolare che i due picchi della curva blu (che descrive le occorrenze `chair-chair`) rappresentano, rispettivamente, le sedie tra di loro adiacenti e le sedie che si trovano a lati opposti del tavolo. Da queste osservazioni possiamo quindi stimare lo spessore medio di un tavolo, che dovrebbe essere circa quello di una sedia e mezzo.

Capitolo 2

Vincolo di distanza

In questo capitolo vedremo come associare ad ogni planimetria le sue **annotazioni**, cioè i documenti che contengono le informazioni circa gli oggetti presenti in un'immagine. Le annotazioni possono essere salvate in vari formati a seconda del dataset utilizzato (`.json`, `.csv`, `.docx`, `.txt`, etc.), e verranno impiegate per implementare il vincolo di distanza che deve sussistere tra due oggetti per poter essere considerati all'interno dello stesso contesto.

2.1 Rappresentazione degli oggetti nel dataset

Per estrarre le annotazioni è stato utilizzato Flo2Plan Dataset Manager [2]. Flo2Plan è un tool che si occupa di recuperare le annotazioni corrispondenti alla planimetria scelta, salvarle in un dizionario, e tracciare sull'immagine i poligoni colorati corrispondenti ai vari oggetti. Se tra le annotazioni viene indicata la classe degli oggetti, grazie a Flo2Plan è possibile fare in modo che i poligoni e i nodi del grafo risultante abbiano lo stesso colore se associati ad oggetti dello stesso tipo. Ad esempio, alla seguente annotazione in formato

`.json` corrisponde il letto del frammento di planimetria vista nella Figura 1.1:

```
{
      "id": 4529,              id dell'oggetto
      "image_id": 204,         id dell'immagine
      "category_id": 11,       tipologia di oggetto
      "iscrowd": 0,
      "area": 14654.5,
      "bbox": [                coordinate del rettangolo
        490.0,                 circoscritto all'oggetto
        536.0,
        ...
      ],
      "segmentation": [        coordinate del perimetro
          490.0,                       dell'oggetto
          536.0,
          ...
      ]
    }
```

Figura 2.1: Esempio di annotazione per un oggetto della planimetria in Figura 1.1

Per l'implementazione del vincolo di distanza saranno di particolare interesse i campi `bbox` e `segmentation`: tramite una semplice media delle dimensioni del rettangolo circoscritto è infatti possibile calcolare il baricentro di ogni oggetto, mentre grazie ai dati sui perimetri riusciremo a calcolare la distanza effettiva che separa due oggetti.

2.2 Implementazione del vincolo

La **distanza massima** a cui si possono trovare due oggetti per poter essere considerati all'interno dello stesso contesto varia ovviamente da immagine a immagine. Per questo motivo, tale parametro deve essere stimato in base alle dimensioni di tutti gli altri oggetti: si procede raccogliendo in due vettori tutte le lunghezze e le larghezze degli oggetti presenti nella planimetria, calcolandone in seguito le medie. I due valori ottenuti descriveranno la grandezza media di tutti gli oggetti. La diagonale di questo rettangolo verrà utilizzata come unità di misura per la definizione della distanza minima. Idealmente, vorremmo che tra due oggetti dello stesso contesto ci sia al massimo lo spazio necessario per inserire un paio degli oggetti medi appena calcolati. Per questo motivo, la distanza massima consentita tra tali oggetti è stata fissata a fissata a circa due volte la diagonale del rettangolo medio.

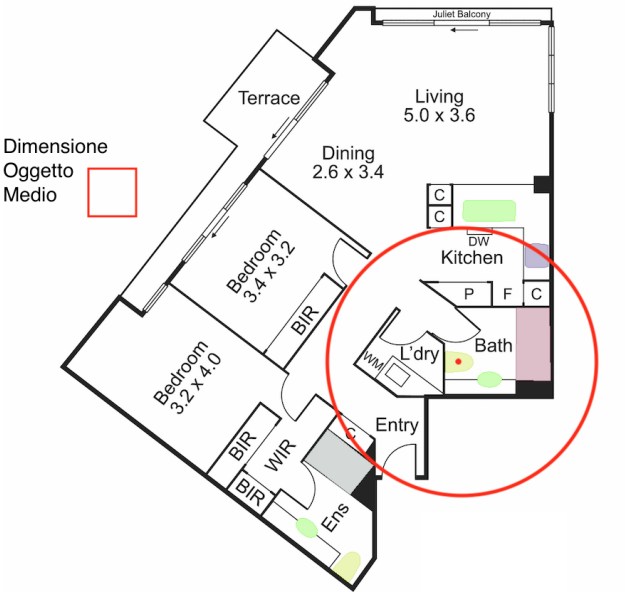


Figura 2.2: Planimetria su cui è stato indicato il rettangolo medio e una circonferenza di raggio pari a due volte la sua diagonale

Nella Figura 2.2 è riportata una planimetria in cui è stato disegnato il rettangolo medio ottenuto e la circonferenza di raggio pari al doppio della sua diagonale. Tutti gli oggetti i cui centri si trovano all'interno della circonferenza verranno presi in considerazione per definire il contesto dell'oggetto verde su cui quest'ultima è centrata. Bisogna però tenere conto del fatto che la distanza tra i centri di due oggetti in alcuni casi può essere ben lontana dalla loro **distanza effettiva**. Per questo motivo, tramite le informazioni sui perimetri degli oggetti calcoleremo le distanze effettive con un metodo simile a quello utilizzato per localizzare le scale all'interno di planimetrie [3]: partendo dalla retta che passa tra i due centri degli oggetti, si considerano i punti di intersezione di tale retta con i perimetri degli oggetti. Si calcola la distanza euclidea tra i due punti trovati, e si confronta tale valore con la distanza tra tutti i vertici dei perimetri. Il valore minore trovato sarà la distanza effettiva tra i due oggetti. Le distanze così ottenute verranno **normalizzate** sul lato più lungo di uno dei due oggetti e utilizzate per pesare gli archi del grafo risultante. Il grafo finale dovrà essere simile a quello in Figura 2.3

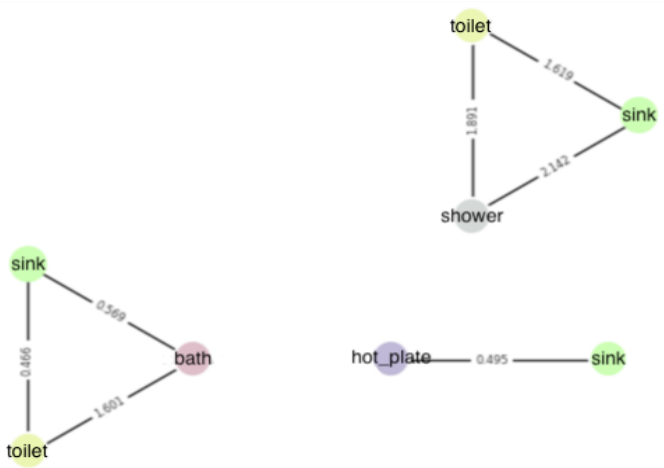


Figura 2.3: Grafo risultante dell'immagine 2.2

Capitolo 3

Algoritmo di rilevamento dei muri

In questo capitolo vedremo come rilevare i muri in una planimetria attraverso le operazioni morfologiche di erosione e dilatazione binaria. Il funzionamento del metodo proposto è garantito per planimetrie con muri neri e pieni, mentre si potrebbero verificare errori per planimetrie i cui muri risultano "vuoti" o troppo sottili.

3.1 Binarizzazione della planimetria

In morfologia binaria, un'immagine viene vista come un sottoinsieme dello spazio Euclideo $\mathbb{Z}^d$, dove d è la dimensione dell'immagine [4]. Operativamente, data un'immagine di dimensione (w, h), questa verrà convertita in una **matrice di pixel** $M_{w \times h}(\mathbb{Z}^p)$, dove p è il numero di valori che caratterizza un pixel dell'immagine. Naturalmente, il parametro p è strettamente legato al formato con cui viene aperta l'immagine, così come il dominio dei valori degli elementi della matrice. Nel corso dell'implementazione dell'algrotimo è stato utilizzato il modulo Python Imaging Library [5] per aprire le immagini nelle seguenti modalità:

- `RGBA`: ogni elemento della matrice è definito dalla tupla (`Red`, `Green`, `Blue`, `Alpha`), i cui valori sono contenuti nell'intervallo $[0, 255]$. In questo caso si ha $p = 4$.
- `L`: ogni elemento della matrice è definito da un valore di **grigio** nell'intervallo $[0, 255]$. In questo caso si ha $p = 1$.
- `1`: ogni elemento della matrice può assumere esclusivamente i valori nell'insieme $\{0, 1\}$, rispettivamente **nero e bianco**. Anche in questo caso si ha $p = 1$.

Il riconoscimento dei muri in una planimetria è implementato tramite operazioni morfologiche applicate a matrici binarie, per questo motivo prima di procedere è necessario convertire l'immagine nella modalità `1`. Per fare ciò, l'immagine selezionata viene aperta in modalità `L` e, dopo aver scelto un' appropriata **soglia di binarizzazione**, viene generata una nuova immagine in modalità `1` in cui ogni pixel nero indica un pixel nella prima immagine con valore inferiore alla soglia di binarizzazione scelta. Questo procedimento è preferibile ad aprire l'immagine direttamente in modalità `1` in quanto, talvolta, i muri dell'immagine sono di colore grigio più o meno scuro, e l'apertura dell'immagine direttamente in modalità `1` potrebbe portare a perdita di muri troppo chiari per essere rilevati. La soglia di binarizzazione è inizialmente fissata a 80, ma viene aumentata gradualmente in caso l'immagine rilevata risulti quasi completamente bianca.

3.2 Erosione e dilatazione binaria

L'idea alla base della morfologia binaria si basa sul confrontare un'immagine con una semplice forma predefinita, effettuando determinate operazioni in

caso la forma predefinita sia contenuta o meno nelle forme dell'immagine. La forma predefinita prende il nome di **elemento strutturante** o **kernel**, e deve essere rappresentata tramite una matrice binaria similmente all'immagine originale. L'**erosione binaria** di un'immagine secondo un elemento strutturante è il luogo dei punti dell'immagine tali che, se centrati sul kernel, questo risulti completamente contenuto nell'insieme di tutti gli elementi non nulli dell'immagine [6]. Per una migliore comprensione del funzionamento dell'erosione binaria possiamo osservare la Figura 3.1.

$$Image = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \qquad Kernel = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$Result = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figura 3.1: Esempio di applicazione dell'erosione binaria utilizzando un elemento strutturante 3×3

Viceversa, la **dilatazione binaria** è un'operazione morfologica che espande le forme di un'immagine. La dilatazione binaria di un'immagine secondo un elemento strutturante è il luogo dei punti che vengono coperti dal kernel quando il suo centro viene sovrapposto a un elemento non nullo dell'immagine [6]. Un esempio di applicazione della dilatazione binaria è riportato nella Figura 3.2.

$$Image = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \qquad Kernel = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$Result = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Figura 3.2: Esempio di applicazione della dilatazione binaria utilizzando un elemento strutturante 3×3

3.3 Definizione delle forme strutturanti per immagini di planimetrie

Poichè nelle planimetrie i muri sono rappresentati con dei rettangoli generalmente più lunghi di tutte le altre forme, l' elemento strutturante che andremo ad utilizzare dovrà essere un **segmento** abbastanza lungo da rimuovere nel processo di erosione tutte le forme che non fanno parte dei muri, ma che allo stesso tempo non cancelli eventuali muri troppo piccoli. Un passo fondamentale per la riuscita dell'algoritmo consiste nello **stimare la lunghezza corretta dei segmenti** utilizzati come forme strutturanti. In generale, la lunghezza dei segmenti dipenderà dalla **grandezza dell'immagine** e dalla **lunghezza di tutte le sue linee nere.** Per tenere conto del primo fattore è sufficiente calcolare la diagonale dell'immagine e utilizzare quest'ultima come misura della sua grandezza. Il secondo fattore è più complesso da stimare, e richiede che l'immagine venga analizzata lungo le sue righe e colonne, salvando in due vettori la lunghezza delle linee che vengono incontrate. In questi due vettori vengono contate le occorrenze di ogni lunghezza, in modo tale da poter tracciare gli istogrammi che vediamo in Figura 3.3.

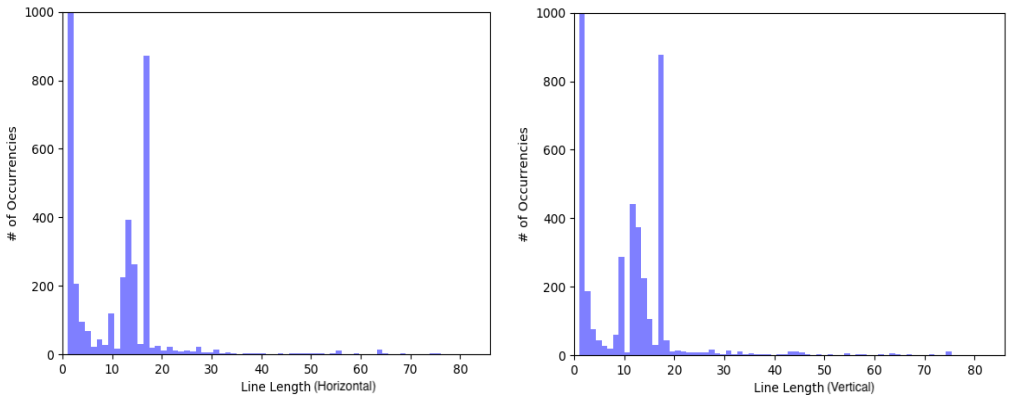


Figura 3.3: La lunghezza delle linee orizzontali e verticali in funzione della loro occorrenza nella planimetria

Sebbene la distribuzione delle linee nere cambi da planimetria a planimetria, in tutte le immagini possiamo trovare degli elementi comuni:

- la maggior parte delle linee nere nelle immagini sono lunghe meno di una decina di pixel. Tali linee costituiscono principalmente sottili dettagli e rumore, perciò verranno ignorate.
- dopo il picco più alto, il massimo locale successivo si troverà in corrispondenza dello spessore di muri particolarmente sottili (in genere i muri interni delle abitazioni) e delle lettere di eventuali didascalie.
- dopo i picchi appena citati si susseguono una serie di massimi dovuti alla lunghezza delle lettere e dei vari muri.

Conseguentemente alle considerazioni appena fatte, la stima della lunghezza dei segmenti da usare come elemento strutturante si svolge nella seguente modalità:

1. da ogni istogramma si cancellano tutti i valori fino al primo massimo compreso, in modo tale da trascurare i dettagli e il rumore.
2. dai valori rimanenti si estraggono i 3 picchi più alti.
3. il valore scelto sarà pari al secondo picco trovato nel passo precedente: questa scelta ci permetterà di erodere tutte le lettere, preservando allo stesso tempo la quasi totalità dei muri.

Il procedimento appena descritto ci fornirà informazioni sulla planimetria che risulteranno fondamentali per la buona riuscita dell'algoritmo di rilevamento dei muri, la cui implementazione è descritta nel paragrafo successivo.

3.4 Algoritmo di rilevamento dei muri

L'implementazione dell'algoritmo per il rilevamento dei muri avviene attraverso i seguenti passi:

1. Stimare la lunghezza corretta delle forme strutturanti tramite il procedimento visto nel paragrafo 3.3.

2. Generare un insieme di n forme strutturanti costituito da segmenti orientati in n angolazioni diverse. Nei nostri esperimenti abbiamo utilizzato segmenti orizzontali, verticali, a 30, 45 e 60 gradi.

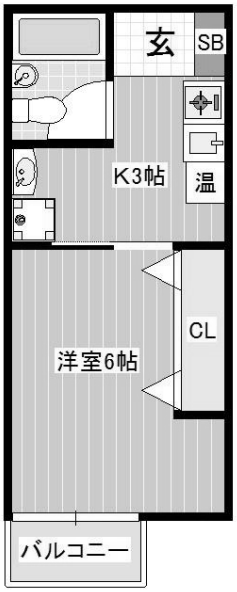
3. Utilizzare le forme strutturanti per generare n nuove immagini attraverso un processo di erosione.

4. Erodere ulteriormente le immagini con un elemento strutturante a croce 3×3. Il numero di cicli di erosione in questa fase è proporzionale alla lunghezza stimata dei segmenti. Questo step serve per eliminare eventuali residui dalle immagini ottenute nello step precedente.

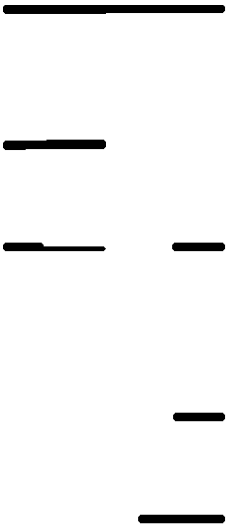
5. Dilatare le n immagini per riottenere lo spessore dei muri originali. Poichè il passo precedente dovrebbe idealmente aver cancellato tutte le forme non appartenenti a muri, a meno di eventuali errori le immagini risultanti dovrebbero contenere solo e soltanto i muri dell'immagine originale.

6. Combinare insieme le n immagini attraverso un'operazione di OR logico.

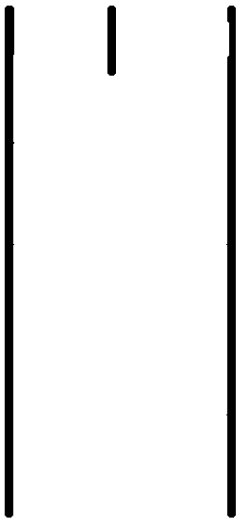
Nella Figura 3.4 possiamo osservare un esempio di applicazione dell'algoritmo. Per questioni di praticità sono state riportate solo le immagini che nel punto 3 sono state erose con forme strutturanti verticali ed orizzontali.



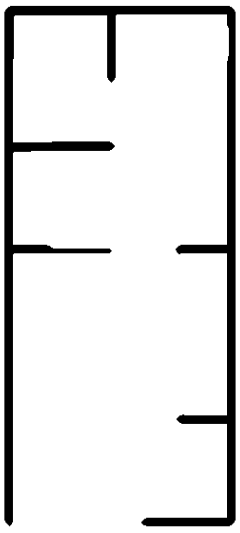
(a) Immagine originale



(b) Erosione e dilatazione con elemento strutturante orizzontale



(c) Erosione e dilatazione con elemento strutturante verticale



(d) Immagine finale dei muri rilevati

Figura 3.4: Alcuni snapshot dell'applicazione dell'algoritmo su un esempio di planimetria

Capitolo 4

Estrazione di informazioni dai contesti degli oggetti

Grazie all'algoritmo di riconoscimento dei muri discusso nel Capitolo 3 possiamo finalmente individuare gli oggetti appartenenti allo stesso contesto e inserirli in una struttura a grafo come quella in Figura 2.3. Il dataset utilizzato contiene 290 planimetrie e 7130 oggetti, i cui grafi sono stati implementati grazie alla librerie NetworkX [8] e Graphviz [9]. In questo capitolo vedremo come utilizzare i grafi per ottenere informazioni riguardo co-occorrenze di oggetti.

4.1 Istogrammi di co-occorrenze di oggetti

A partire da ogni planimetria possiamo estrarre informazioni riguardo co-occorrenze di oggetti analizzandone il grafo: per fare ciò per ogni immagine istanziamo un vettore contenente tante triplette (`oggetto1`, `oggetto2`, `distanza`) quanti sono gli archi del grafo degli oggetti. La soluzione proposta permette di linearizzare il grafo, semplificando in questo modo la creazione

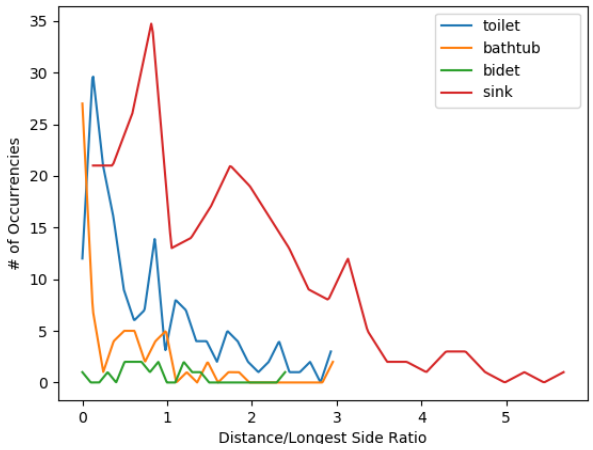
di un nuovo **dataset** contenente le informazioni aggregate ottenute da ogni planimetria. Tale dataset sarà un dizionario con tanti campi quante le possibili combinazioni di coppie di oggetti, dove ogni **key** sarà una tupla del tipo (`oggetto1`, `oggetto2`), ed ogni **value** sarà un vettore contenente tutte le distanze trovate nelle planimetrie tra i due oggetti del tipo chiave.

```
{
        ('armchair', 'armchair') : [2.592, 2.592, ...]
        ('armchair', 'sink') : [2.354, 1.672, ...]
        ('armchair', 'bathtub') : []
        ('armchair', 'table') : [0.804, 0.868, ...]
        ('armchair', 'couch') : [0.169, 0.188, ...]
        ...
        ('sink', 'armchair') : [2.354, 1.672, ...]
        ('sink', 'sink') : [0.656, 0.559, ...]
        ('sink', 'bathtub') : [0.102, 0.666, 0.243 ...]
        ('sink', 'table') : [2,643, 3.981, ...]
       ...
 }
```

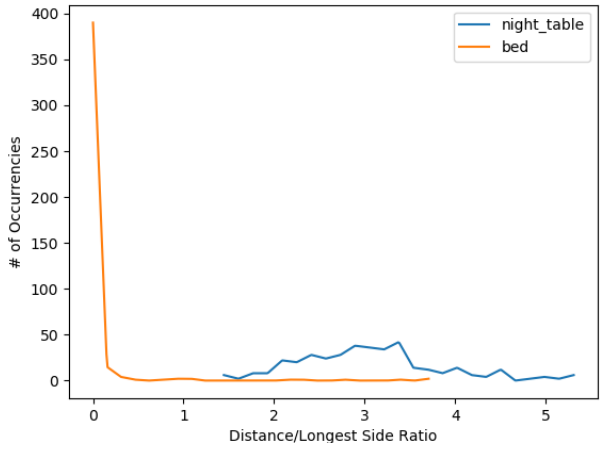
Figura 4.1: Alcuni campi del dataset delle co-occorrenze di oggetti

Consideriamo ad esempio la co-occorrenza (`sink-bathtub`): dalla Figura 4.1 si evince che, nelle planimetrie analizzate, c'è un lavandino a distanza 0.102 da una vasca, un altro a distanza 0.666, uno a distanza 0.243, e così via. Ricordiamo che le tutte le distanze sono normalizzate sul lato più lungo del primo oggetto usato come key. Un dataset del genere permette di definire, per ogni coppia di classi di oggetti, una **funzione istogramma** che mette in relazione la distanza di ogni occorrenza con il numero di occorrenze

che si trovano alla data distanza. Questo significa che, presa una qualsiasi classe di oggetti, possiamo tracciare un grafico contenente tutte le funzioni istogramma che coinvolgono tale classe, in modo tale da visualizzare quali tipi di oggetti di uno stesso contesto si trovano ad una certa distanza più frequentemente.



(a) Istogrammi dell'oggetto `shower`



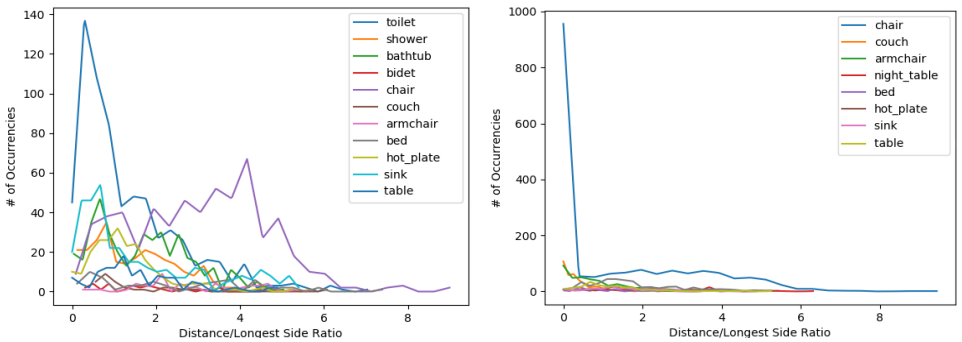
(b) Istogrammi dell'oggetto `night_table`

Figura 4.2: Esempi di istogrammi di co-occorrenze di oggetti per 2 classi diverse

Visualizzare grafici come quelli in Figura 4.2 permette spesso di trarre conclusioni interessanti. Osserviamo ad esempio il grafico in Figura 4.2(b): la curva `bed` in arancione ci mostra che è estremamente probabile che un letto si trovi a distanza 0 (cioè praticamente attaccato) rispetto ad un comodino. Inoltre, poichè i comodini si trovano in genere a lati opposti di un letto, grazie alla curva blu possiamo stimare che lo spessore medio di un letto vari tra i 2 (letto singolo) e i 3.5 (letto matrimoniale) comodini .
Nell' Appendice A sono stati inseriti tutti i grafici ottenuti nelle modalità descritte in questo capitolo, in modo tale da lasciare al lettore la possibilità di trovare, qualora lo desiderasse, altre interessanti relazioni tra classi di oggetti.

4.2 Definizione di sottoclassi di oggetti

Osservando per tutte le tipologie di oggetti i grafici come quelli in Figura 4.2 ci rendiamo conto che per alcune classi vengono tracciate un gran numero di curve: ciò si verifica quando una certa tipologia di oggetti è presente in svariati contesti molto diversi tra loro.



(a) Istogrammi dell'oggetto `sink` (b) Istogrammi dell'oggetto `table`

Figura 4.3: Istogrammi di co-occorrenze di oggetti con molte curve

Ad esempio in Figura 4.3 vediamo due classi i cui istogrammi presentano un gran numero di curve. Per risolvere questo problema è necessario definire **sottoclassi** di oggetti, in modo tale da tracciare ulteriori grafici su cui ridistribuire le curve. La soluzione proposta prevede la creazione delle classi `bathroom_sink`, `kitchen_sink`, `small_table` e `dining_table`. Poichè gli oggetti delle planimetrie utilizzate erano già stati precedentemente etichettati, all'introduzione di sottoclassi deve necessariamente seguire una fase di riclassificazione, in cui gli oggetti delle superclassi verranno etichettati con le classi appena create. E' importante notare che, se la fase di riclassificazione fosse eseguita senza commettere errori, sarebbe possibile addestrare un modello in grado di automatizzare il processo per le iterazioni future. Il processo di classificazione si svolgerà dunque nei seguenti passi:

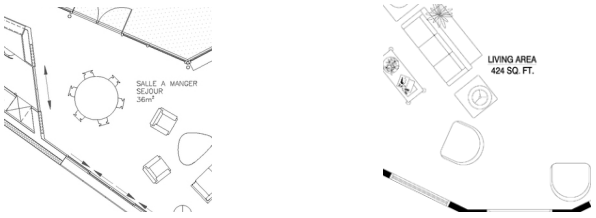
1. Viene lanciata una funzione che, in base alle informazioni ottenute dal dataset in Figura 4.1, cerca di classificare gli oggetti nelle loro sottoclassi. Tale funzione opera osservando la distanza minima dell'oggetto da classificare rispetto a oggetti di contesti significativi. Ad esempio, se un `sink` è molto vicino ad un `bathtub`, questo verrà etichettato come `bathroom_sink`.

2. Dalla planimetria dell'oggetto appena classificato viene ritagliata una piccola immagine che lo contiene. Questa immagine viene poi inserita in una cartella diversa a seconda della classificazione scelta al passo precedente.

3. Dopo aver iterato i passi precedenti su tutti gli oggetti delle planimetrie, l'utente osserva una per volta tutte le immagini salvate, correggendo all'occorrenza la classificazione spostando le immagini errate nella cartella corretta.

4. Viene lanciata una nuova funzione che etichetta nuovamente gli oggetti delle superclassi, questa volta tenendo conto esclusivamente della cartella in cui si trova l'oggetto da classificare. I risultati vengono poi salvati in un file `.pickle`.

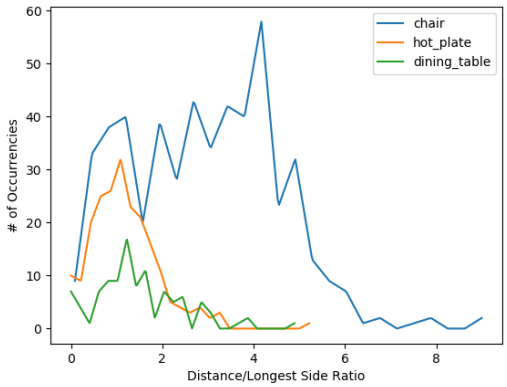
Se l'utente ha svolto il suo compito diligentemente, la classificazione dovrebbe terminare senza errori. Nella Figura 4.4 possiamo osservare due oggetti originariamente di classe `table`, correttamente etichettati secondo le loro sottoclassi.



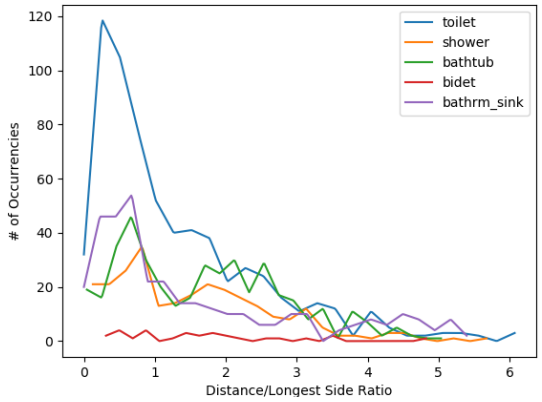
(a) `dining_table` (b) `small_table`

Figura 4.4: Oggetti della classe `table` correttamente classificati

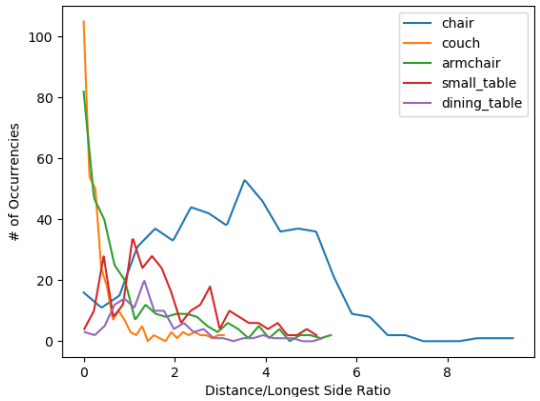
Al termine dell'intero processo, dalla Figura 4.3 sono stati ricavati 4 nuovi grafici, che possiamo vedere in Figura 4.5.



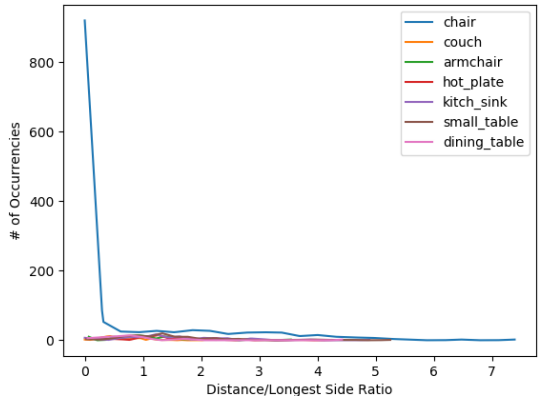
(a) Istogrammi dell'oggetto `kitchen_sink`



(b) Istogrammi dell'oggetto `bathroom_sink`



(c) Istogrammi dell'oggetto `small_table`



(d) Istogrammi dell'oggetto `dining_table`

Figura 4.5: Istogrammi di co-occorrenze per le sottoclassi appena definite

Osservando ad esempio gli istogrammi delle classi `bathroom_sink` e `kitchen_sink` vediamo che questi contengono tutte le curve che originariamente erano nell'istogramma della classe `sink`, dove però le varie curve sono state ridistribuite in modo da separare gli oggetti del bagno da quelli della cucina. Un ragionamento analogo può essere fatto per le classi `dining_table` e `small_table`. Si noti infine che in generale le sottoclassi potrebbero non essere mutualmente esclusive. E' questo il caso della classe `sink`: infatti in svariate planimetrie sono stati rinvenuti lavandini in stanze diverse da bagni e cucine (se ne trovano molti ad esempio in garage, lavanderie, e balconi). In questi casi è stato scelto di non specificare ulteriormente gli oggetti, lasciando invariata la classificazione originale.

Capitolo 5

Risultati sperimentali

In questo capitolo discuteremo i risultati degli esperimenti svolti nel corso del progetto. Gli esperimenti svolti si dividono in due categorie: quelli sull'algoritmo di rilevamento dei muri e quelli sugli alberi di decisione utilizzati per sottoclassificare di oggetti. In entrambi i casi gli esperimenti hanno lo scopo di quantificare l'accuratezza delle soluzioni proposte.

5.1 Valutazione dell'algoritmo di rilevamento dei muri

Per la valutazione dell'algoritmo di rilevamento dei muri è stato utilizzato un dataset contenente circa 800 planimetrie, la maggior parte delle quali è caratterizzata da muri neri pieni. Tale dataset, a differenza da quello utilizzato per estrarre co-occorrenze di oggetti, è associato ad annotazioni contenenti informazioni circa la posizione di tutti i muri. Queste informazioni verranno utilizzate per definire un'**immagine soluzione** da confrontare con l'output dell'algoritmo di rilevamento, in modo tale da poter trarre concluisioni circa la sua accuratezza.

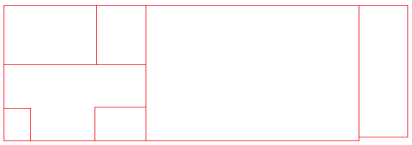
5.1.1 Definizione dell'immagine soluzione

Per il dataset utilizzato le annotazioni sono in formato `.txt` ed ogni muro viene rappresentato con la seguente formattazione:

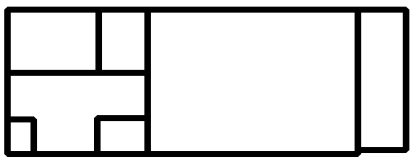
```
252 287 362 287 wall
```

I quattro numeri riportati costituiscono le coordinate di due punti dell'immagine, rappresentando effettivamente il segmento che costituisce l'anima del muro. Dopo aver tracciato tutti i segmenti dei muri su una nuova immagine, sarà possibile ottenere un'immagine simile alla Figura 5.1(c) tramite un numero opportuno di cicli di dilatazione (che dipende dalle informazioni ottenute dal procedimento descritto nella capitolo 3.3).

(a) Immagine originale



(b) Soluzione ottenuta dalle annotazioni



(c) Dilatazione dell'immagine precedente

Figura 5.1: Illustrazione dei passi necessari per trovare l'immagine soluzione di una planimetria

Osservando attentamente la soluzione trovata possiamo notare che questa presenta **due errori** non trascurabili:

- I muri tracciati in questo modo non tengono conto di eventuali porte.
- Se presenti, tra i muri delle annotazioni si trovano anche le ringhiere che delimitano i balconi della planimetria.

Fortunatamente, sia le porte che i balconi risultano essere oggetti etichettati nelle annotazioni della planimetria, ed è quindi possibile individuare i muri errati per escluderli o correggerli nella rappresentazione della soluzione.
Tenere conto delle porte è semplice: è sufficiente infatti controllare, per ogni muro, se esiste una porta le cui coordinate si trovano sul segmento corrispondente descritto nelle annotazioni.

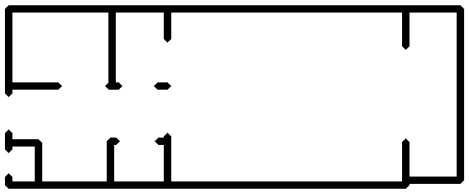


Figura 5.2: Soluzione della Figura 5.1(c) dopo aver rimosso le porte.

Per cui, ogni volta che viene trovato un muro contenente una porta, questo viene sostituito da due nuovi muri più corti in modo tale da lasciare un'apertura. Rimuovere i muri che delimitano i balconi è un problema più complesso, in cui entrano in gioco fattori come l'orientazione del balcone, la sua forma, e dove è situata l'etichetta che lo identifica. Il procedimento si può riassumere nei seguenti passi:

1. A partire dall'etichetta del balcone, si trovano i quattro muri più vicini nelle quattro direzioni.

2. Si calcolano le distanze di questi quattro muri dall'etichetta. Se l'etichetta è centrata rispetto al balcone, si otterranno quattro distanze simili a due a due. Nel caso una di queste distanze sia troppo grande, significa che il muro su cui è stata calcolata è un muro interno, e pertanto va ignorato (questo accade quando l'etichettà si trova nei pressi dell'entrata del balcone).

3. Si eliminano tutti i muri rimanenti e tutti i muri che hanno gli estremi in comune con i muri rimanenti. Questo passo dovrebbe eliminare il balcone anche se la sua forma non è rettangolare.

Si noti che questo algoritmo potrebbe commettere degli errori nel caso in cui il balcone presenti delle forme particolari, si trovi all'angolo della planimetria, o l'etichetta sia troppo decentrata rispetto al centro del balcone. Se l'algoritmo è andato a buon fine, l'immagine soluzione finale sarà simile alla Figura 5.3.

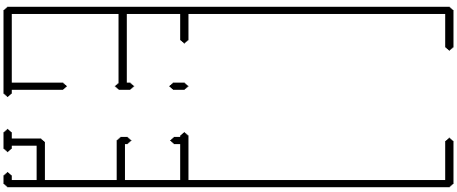


Figura 5.3: Soluzione della Figura 5.1(c) dopo aver rimosso le porte e il balcone.

5.1.2 Metriche di valutazione

Dopo aver definito l'immagine soluzione da confrontare con l'output dell'algoritmo di rilevamento, possiamo scegliere le metriche che utilizzeremo per valutare quest'ultimo e la procedura di calcolo dei relativi score. Data un'immagine di output e la sua corrispondente immagine soluzione, per ogni pixel possono verificarsi quattro casi:

- True Positive (TP): Il pixel fa parte di un muro sia nell'immagine di output che nell'immagine soluzione.
- False Positive (FP): Il pixel fa parte di un muro nell'immagine di output, ma non nell'immagine soluzione. In statistica questo caso è anche chiamato **errore di prima specie** (α).
- False Negative (FN): Il pixel fa parte di un muro nell'immagine soluzione, ma non nell'immagine di output. In statistica questo caso è anche chiamato **errore di seconda specie** (β).
- True Negative (TN): Il pixel non fa parte di un muro in entrambe le immagini.

In generale per controllare se un pixel appartenga o meno ad un muro basta semplicemente controllarne il colore. Nel campo del machine learning e dell'analisi predittiva, i quattro casi vengono spesso raccolti all'interno di una tabella chiamata **matrice di confusione** [7], che permette una visualizzazione generale delle performance dell'algoritmo. Nella Tabella 5.1 sono stati raccolti in una matrice di confusione i quattro casi appena descritti.

		Immagine Soluzione	
		Pixel Nero	Pixel Bianco
Immagine	Pixel Nero	TP	FP (α)
Output	Pixel Bianco	FN (β)	TN

Tabella 5.1: Matrice di confusione per l'algoritmo di rilevamento dei muri

Nel nostro caso, poichè quasi tutti i pixel di una planimetria sono bianchi, troveremo che la maggior parte di questi rientreranno nel caso TN. Per questo motivo le metriche scelte saranno quelle che non necessitano del pa-

rametro TN per il calcolo, e che quindi saranno più significative ai fini della valutazione:

- **Precision**: dati tutti i pixel neri nell'immagine di output, la precision indica quanti di questi sono neri anche nell'immagine soluzione. In statistica questa metrica viene anche chiamata **positive predictive value**, e viene calcolata con la seguente formula:

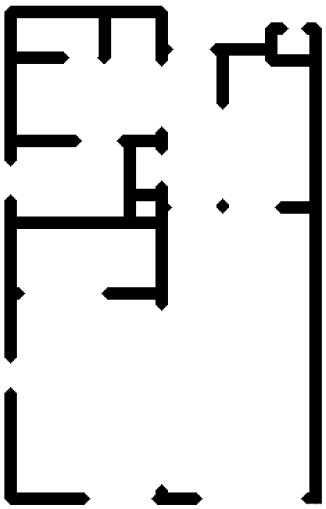
$$PPV = \frac{TP}{TP + FP}$$

- **Recall**: dati tutti i pixel neri nell'immagine soluzione, la recall indica quanti di questi sono neri anche nell'immagine di output. In statistica questa metrica viene anche chiamata hit-rate o **true positive rate**, e viene calcolata con la seguente formula:

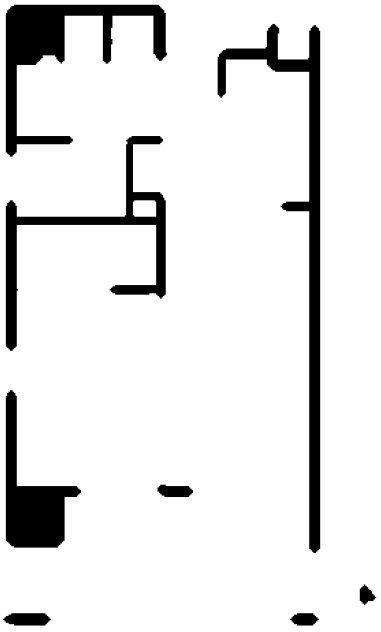
$$TPR = \frac{TP}{TP + FN}$$

Si noti inoltre che tutti i concetti appena descritti possono essere estesi anche ai singoli muri delle immagini, e non sono di certo limitati al semplice conteggio di pixel. In particolare, nel nostro caso un muro presente nell'immagine soluzione verrà considerato riconosciuto (TP) se almeno il 50% dei suoi pixel è presente anche nell'immagine di output. In caso contrario il muro verrà considerato non riconosciuto (FN). Per quanto riguarda il caso FP, basterà calcolare un'**immagine differenza** tra l'immagine soluzione e l'immagine di output per ottenere una nuova immagine contenente tutte le forme erroneamente classificate come muri. Dall'immagine differenza, qualsiasi **componente connessa** [6] sufficientemente grande verrà considerata un muro erroneamente rilevato. Nella Figura 5.5 vediamo un esempio di planimetria con relativa immagine di output, soluzione, e differenza.

(a) Immagine originale



(b) Immagine soluzione



(c) Immagine di output dell'algoritmo

(d) Immagine differenza

Figura 5.4: Immagini utili per la valutazione dell'algoritmo

5.1.3 Analisi dei risultati

La fase di validazione consiste nel lanciare sull'intero dataset l'algoritmo di riconoscimento dei muri, calcolando per ogni planimetria la Precision e la Recall riguardanti la classificazione dei pixel e il riconoscimento dei muri. Per i valori trovati viene calcolata una media, i cui risultati finali sono stati raccolti nella Tabella 5.2

	Precision	Recall
Pixel	72.1%	93.3%
Walls	83.4%	75.7%

Tabella 5.2: Risultati della fase di validazione dell'algoritmo

Dalla tabella emerge un dato particolarmente interessante: in media l'algoritmo è in grado di rilevare solamente 3 muri su 4 presenti nella soluzione, ma tali muri costituiscono il 93% dei pixel neri dell'immagine soluzione. Questa osservazione è diretta conseguenza del fatto che, poichè alla base del riconoscimento dei muri vi è un processo di erosione, può capitare che i muri più piccoli vengono erroneamente eliminati insieme al testo e agli altri particolari dell'immagine. Dopo aver osservato, per un gran numero di planimetrie, immagini del tipo della Figura 5.5, possiamo concludere che la maggior parte degli errori siano dovuti alle seguenti cause:

- Le porte ed i muri non sono correttamente etichettati nelle annotazioni.
- Nella planimetria sono presenti colonne e/o piloni che nelle annotazioni non vengono segnalati come muri.
- I muri della planimetria sono particolarmente sottili o le forme da erodere sono particolarmente grandi.

- I balconi non vengono correttamente rimossi dall'immagine soluzione.
- Il numero di cicli di erosione e dilatazione non è stimato correttamente in fase di preprocessing.

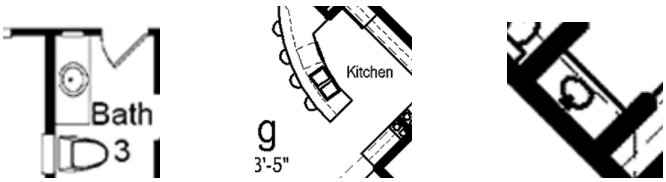
5.2 Classificazione tramite alberi di decisione

La classificazione manuale eseguita nel paragrafo 4.2 è un processo lungo, dispendioso, e di certo non è scalabile su grandi dataset. Ciò nonostante un controllo manuale permette di ridurre al minimo gli errori, e questo ci permette di utilizzare la classificazione risultante come base per creare un **training set** per un modello di apprendimento, in modo tale da automatizzare il processo di separazione in sottoclassi per le iterazioni successive. Il training set che andremo a utilizzare nel nostro caso avrà tanti campi quanti sono gli oggetti della superclasse, dove ogni oggetto avrà la seguente rappresentazione:

```
obj_id={ class1: (min_dist, #occ.)
         class2: (min_dist, #occ.)
         class3: (min_dist, #occ.)
         [...]
         }
```

Ogni oggetto sarà quindi identificato da un numero univoco e conterrà, per ogni classe, il numero di oggetti di quella classe nel suo stesso contesto insieme alla distanza dal più vicino fra questi. Si noti tuttavia che in alcune planimetrie vi sono oggetti quasi completamente circondati da muri (ad esempio il lavandino in Figura 5.5(c)), e che quindi questa rappresentazione non ci

fornisce alcuna informazione su tali oggetti.



(a) `bathroom_sink` (b) `kitchen_sink` (c) `sink`

Figura 5.5: 3 esempi di `sink` classificati dallo script. Si noti che la Figura 5.5(c) è in reatà un `bathroom_sink` che può essere riconosciuto solo ignorando i muri che lo circondano.

Per questo motivo, nella rappresentazione degli oggetti sono stati duplicati alcuni campi, i cui valori vengono calcolati senza tenere conto della presenza dei muri.

Il modello utilizzato in questa trattazione è un **albero di decisione** implementato grazie alla libreria Sci-Kit Learn [10]. Un albero di decisione è un metodo di apprendimento supervisionato non parametrico che funziona separando i dati in sottoinsiemi via via più piccoli e omogenei dal punto di vista delle classi presenti. Ogni volta che i dati vengono separati secondo un certo criterio (cioè viene effettuata una decisione), tale decisione viene aggiunta all'interno di una struttura ad albero. Un albero di decisione ottimale riesce a classificare dati correttamente utilizzando il minor numero possibile di decisioni. Un albero con troppi nodi, infatti, è un modello eccessivamente complesso che predice perfettamente ogni immagine del training set ma ottiene scarsi risultati sui dati di test (questo problema è noto con il nome di **ovrefitting**). Per questo motivo dobbiamo scegliere una **misura di impurità** che per ogni sottoinsieme ci permetta sia di individuare la migliore separazione possibile, sia di capire quando non è più conveniente continuare

a generare ulteriori sottoinsiemi (**pruning**). Nel nostro caso la misura di impurità utilizzata è l'**entropia**. Tramite quest'ultima è stato perciò specificato il parametro `min_impurity_decrease` come criterio di arresto: se la differenza di impurità a seguito di una decisione è minore di tale parametro non verranno effettuate ulteriori separazioni in quel sottoinsieme.
Bisogna tuttavia trovare un valore adatto per tale parametro, in modo tale che l'albero risultante mantenga delle performance soddisfacenti (e quindi un adeguato numero di nodi), aggirando allo stesso tempo il problema dell'overfitting. Lo studio dei macroparametri di un modello viene effettuato tramite l'analisi della sua **curva di validazione**:

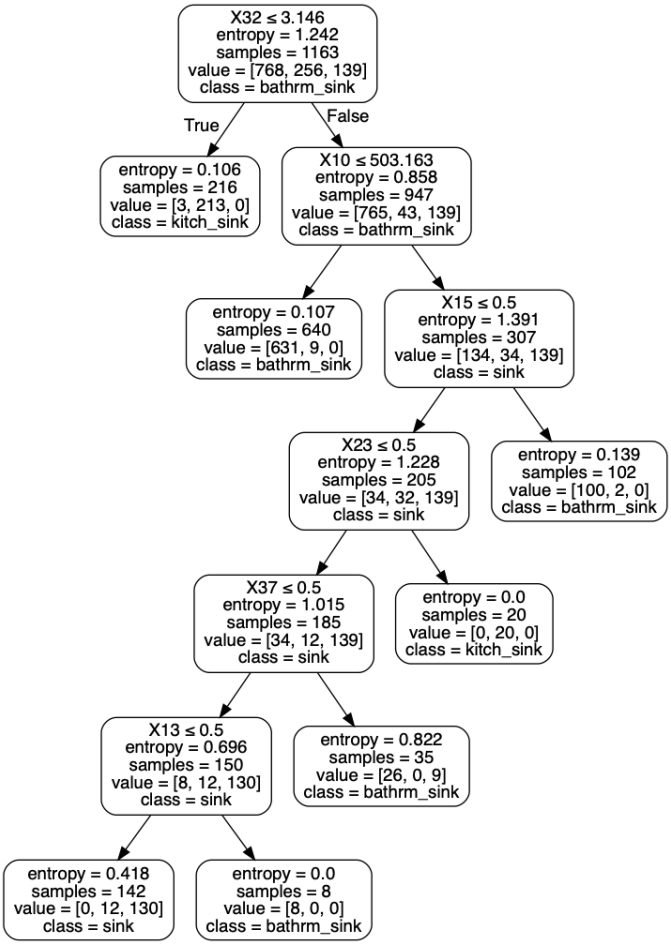
1. Si scelgono i valori del parametro `min_impurity_decrease` da testare. Nel nostro caso abbiamo preso 10 punti sull'intervallo $[10^{-7}, 1]$ distanziati secondo una scala logaritmica.

2. Si sceglie quale porzione di dati dovrà fungere da training set e quale da test set. Nel nostro caso è stata effettuata una **K-fold cross validation** con $K = 4$: il 75% dei dati è stato utilizzato per addestrare il modello, mentre la restante parte verrà utilizzata per valutarne le prestazioni. Per ogni valore del parametro da testare verranno quindi istanziati K alberi diversi, ognuno addestrato con una porzione diversa del dataset e testato sui dati rimanenti.

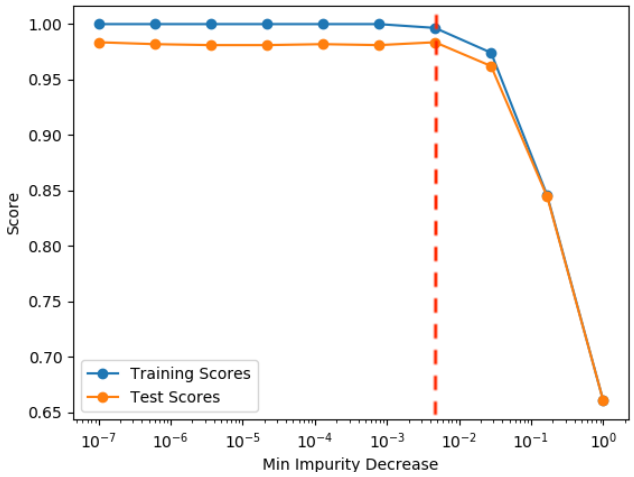
3. Per ogni valore del parametro da testare vengono salvati K score per il training set e K score per il test set, ottenuti dai K alberi definiti al passo precedente. Il risultato finale sarà costituito da due matrici $M_{(\#\ valori\ \times\ K)}$, dove sulla riga i-esima sono stati salvati i K score ottenuti per quel valore. Per ogni riga vengono calcolate le medie degli

score ottenuti, ottenendo un vettore di lunghezza pari al numero di valori da testare.

4. Vengono tracciate le curve di test e di train, che mettono in relazione il valore del parametro utilizzato con le medie degli score ottenuti. Poichè il nostro scopo è massimizzare lo score del modello per i dati non osservati, il valore di `min_impurity_decrease` che verrà scelto si troverà in corrispondenza del massimo della curva di test.

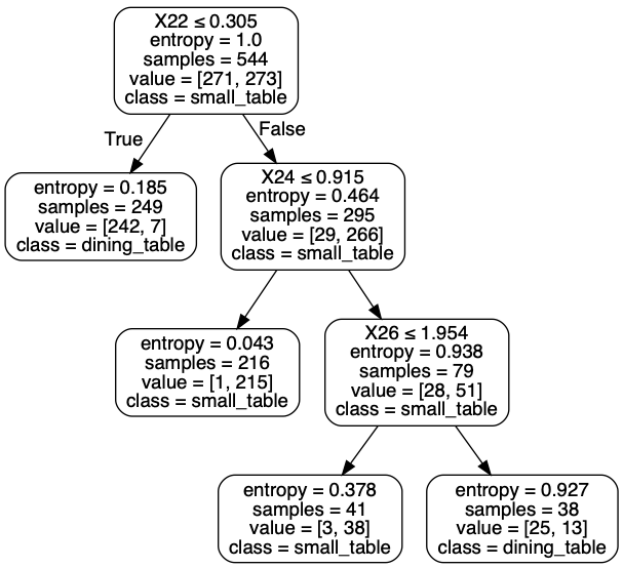


(a) Albero di decisione della classe `sink` (12 nodi)

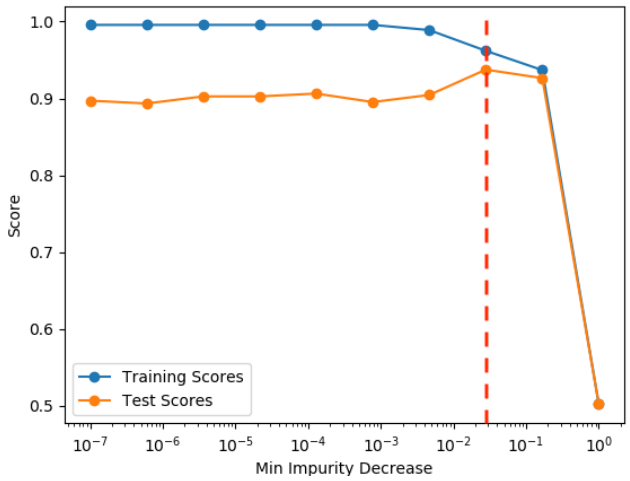


(b) Curva di validazione della classe `table` (`min_impurity_decrease` ≈ 0.005)

Figura 5.6: Albero di decisione con relativa curva di validazione per la specifica della classi `sink`



(a) Albero utilizzato per specificare la classe `sink` (7 nodi)



(b) Curva di validazione della classe `table` (`min_impurity_decrease` ≈ 0.027)

Figura 5.7: Albero di decisione con relativa curva di validazione per la specifica della classi `table`

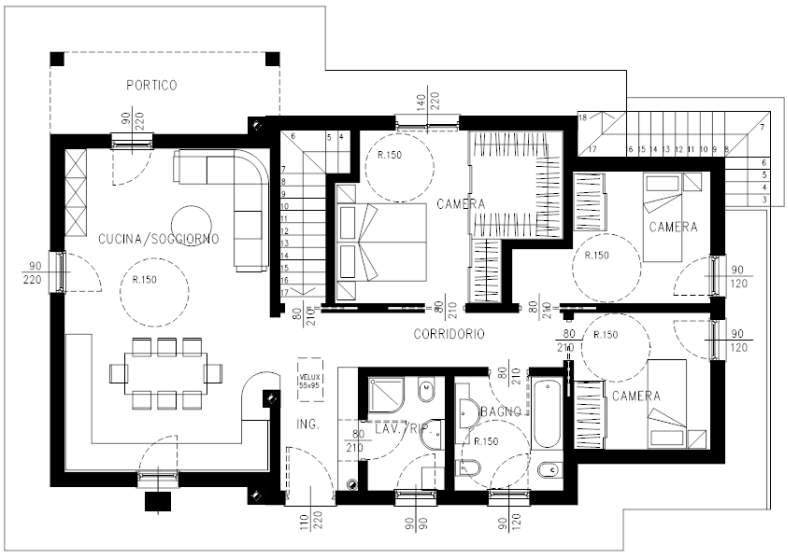
In linea con quanto detto, l'albero della classe `sink` contiene più nodi di quello della classe `table`: infatti, poichè quest'ultimo deve classificare solamente due tipi di oggetti (a differenza delle 3 classi dell'altro albero) si verifica che l'albero della classe `table` raggiunge uno score ottimale per un valore di `min_impurity_decrease` circa 6 volte superiore a quello dell'albero della classe `sink`. Una volta trovati gli alberi ottimali, questi vengono esportati in formato `.dot` per poter essere utilizzati nelle iterazioni future.

Capitolo 6

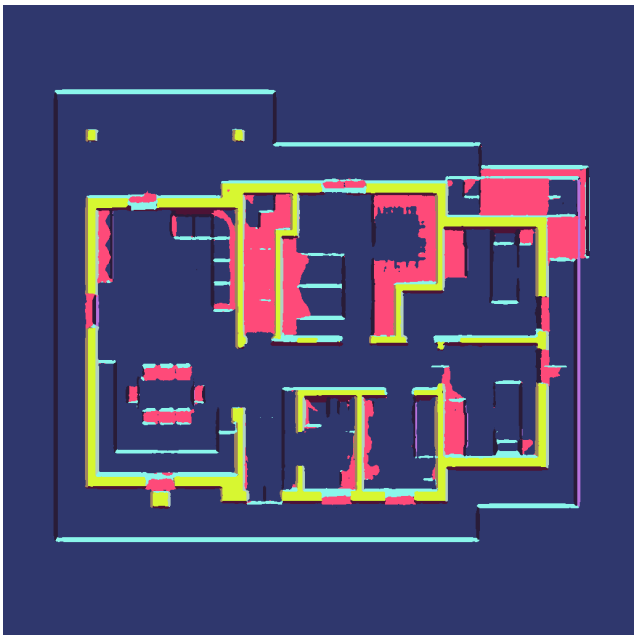
Futuri sviluppi e conclusioni

6.1 Migliorare il rilevamento dei muri

Uno dei modi per migliorare l'informazione estratta è sicuramente quello di rilevare i muri secondo un approccio più raffinato. Una metodo alternativo a quello trattato, ad esempio, potrebbe essere quello di **segmentare** l'immagine in modo unsupervised aggregando insieme regioni di pixel con caratteristiche simili (**clustering**). La segmentazione può essere affiancata ad altri metodi di elaborazione dell'immagine, ad esempio il Local Binary Pattern o la Trasformata Wavelet. I risultati ottenuti con quest'ultima (Figura 6.1) lasciano supporre che seguendo questa strada si possa trovare un modo per isolare i muri da tutte le altre forme dell'immagine.



(a) Immagine di input



(b) Segmentazione dell'immagine precedente

Figura 6.1: Esempio di segmentazione di una planimetria.

6.2 Applicazioni pratiche

L'informazione contestuale estratta secondo i metodi suggeriti in questa trattazione può essere utile per migliorare le performance di modelli esistenti per la classificazione degli oggetti in planimetria. Supponendo infatti di avere un classificatore in grado di distinguere oggetti in base alla loro forma e orientamento, ad esempio quello di *Ziran* e *Marinai* [11], tenere conto dei vari contesti di oggetti sicuramente porterebbe ad una classificazione più accurata.

Uno dei campi di applicazione più importanti del riconoscimento di oggetti riguarda l'accessibilità ad informazioni grafiche da parte di persone affette da cecità. Ad esempio, senza allontanarci troppo dal tema di questa trattazione, il tool GraVVITAS [12] applicato a schermi multitouch permette di restituire un feedback uditivo a partire da input tattili su planimetrie etichettate. Lo sviluppo di tecnologie simili potrebbe un giorno permettere a chiunque di accedere a qualsiasi forma di informazione bidimensionale, proprio come i sintetizzatori vocali sono diventati nel giro di pochi anni lo standard per la presentazione di informazione testuale.

6.3 Conclusioni

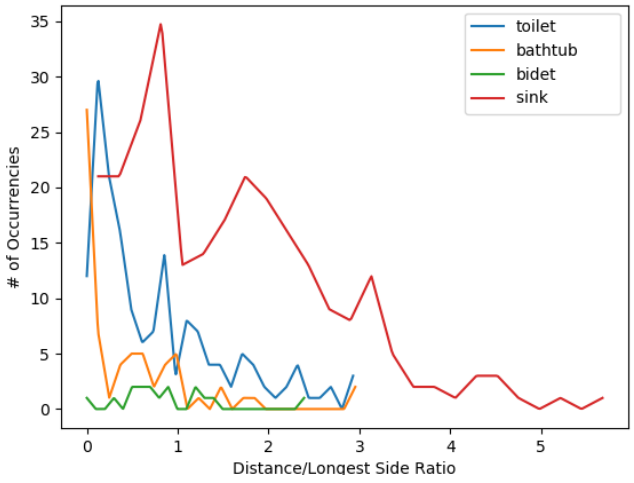
Abbiamo visto come ricavare informazioni contestuali da dataset di planimetrie elaborandone l'immagine. Le informazioni ottenute sono state visualizzate sottoforma di istogrammi, ed in seguito utilizzate per definire sottoclassi di particolari oggetti. Il risultato della sottoclassificazione, eseguita manualmente, è stata utilizzata per addestrare alberi di decisione che possono essere impiegati per automatizzare il processo. Il metodo presentato ci si auspica

possa migliorare il riconoscimento di oggetti in planimetrie, così da renderle un domani accessibili anche ad utenti affetti da disabilità.

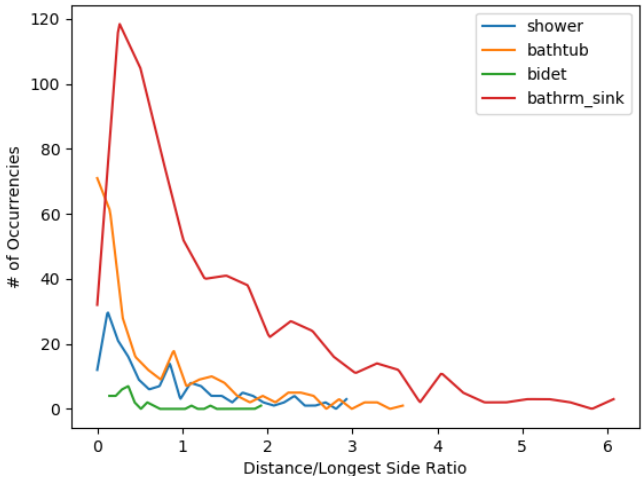
Appendices

Appendice A

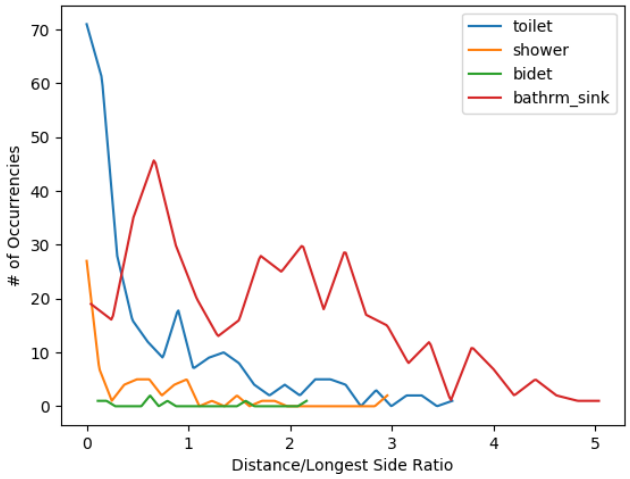
Istogrammi di tutte le classi di oggetti



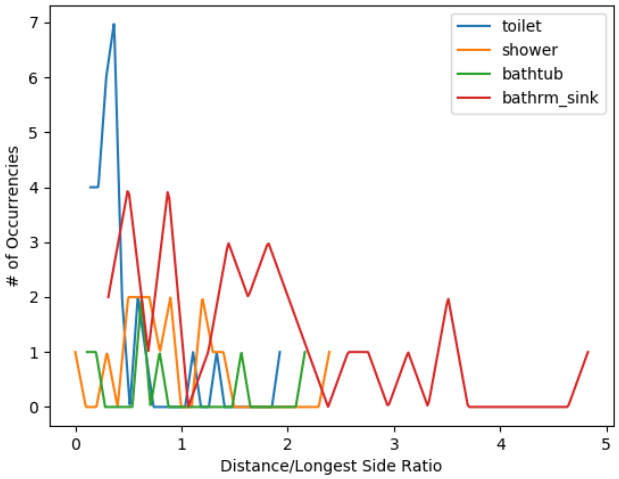
(a) Istogrammi dell'oggetto `shower`



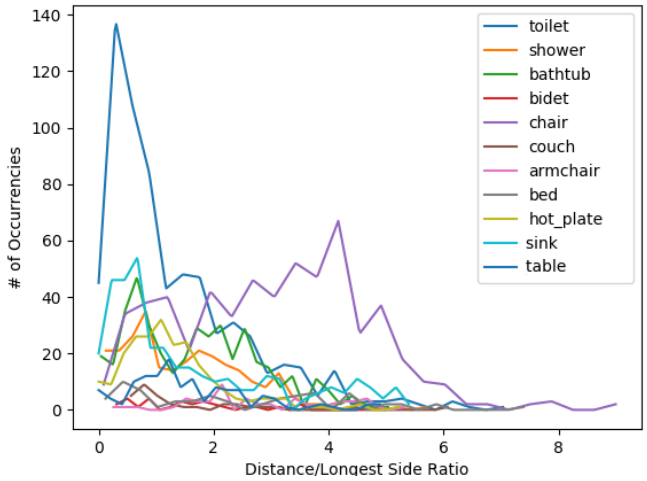
(b) Istogrammi dell'oggetto `toilet`



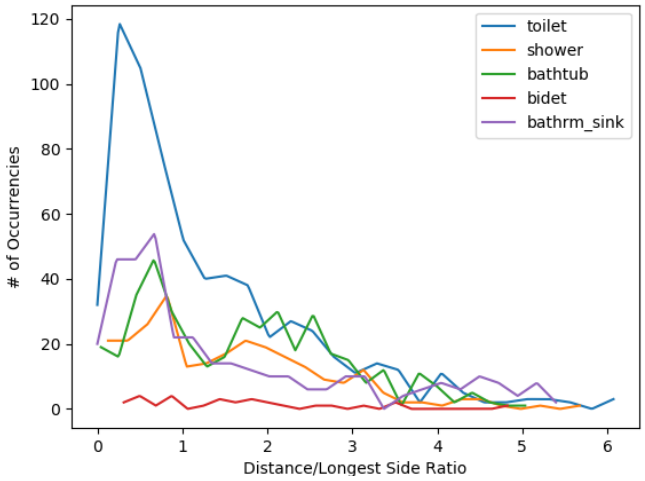
(c) Istogrammi dell'oggetto `bathtub`



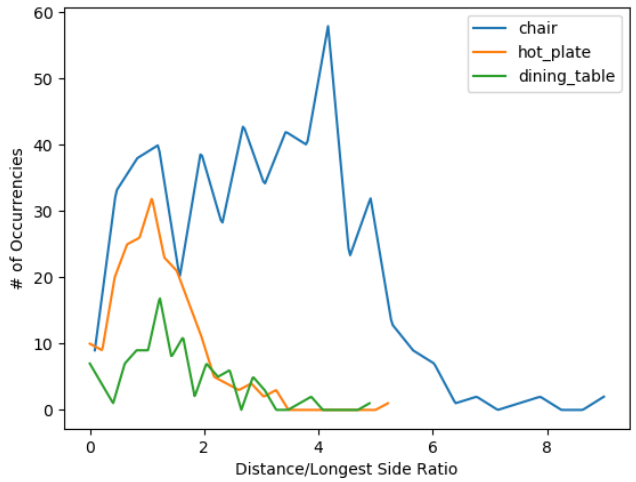
(d) Istogrammi dell'oggetto `bidet`



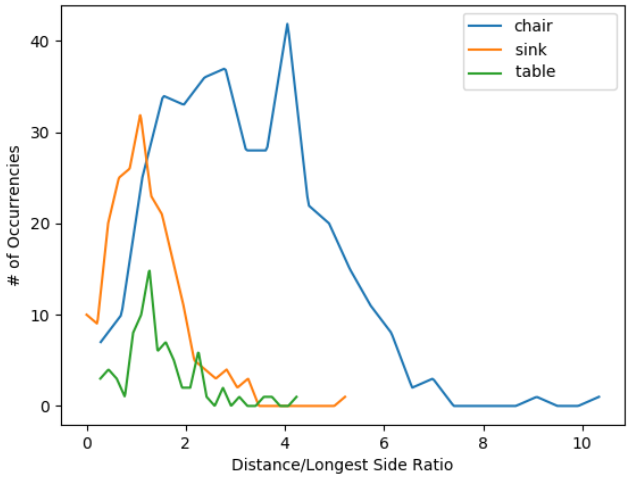
(e) Istogrammi dell'oggetto `sink`



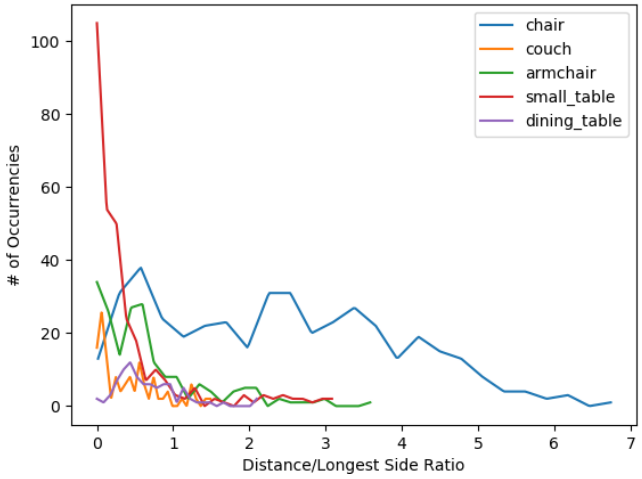
(f) Istogrammi dell'oggetto `bathroom_sink`



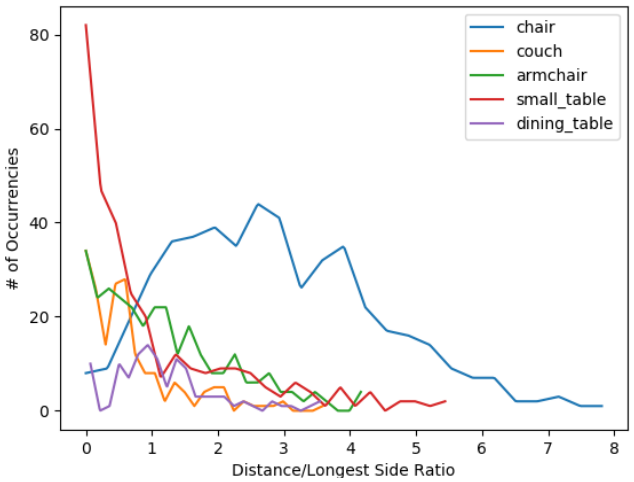
(g) Istogrammi dell'oggetto `kitchen_sink`



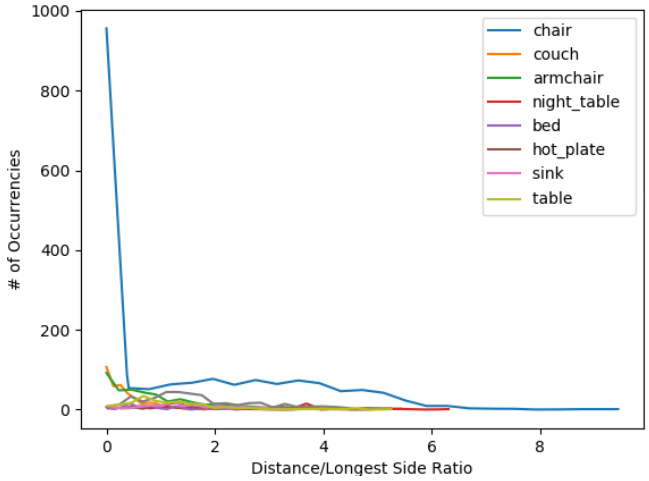
(h) Istogrammi dell'oggetto `hot_plate`



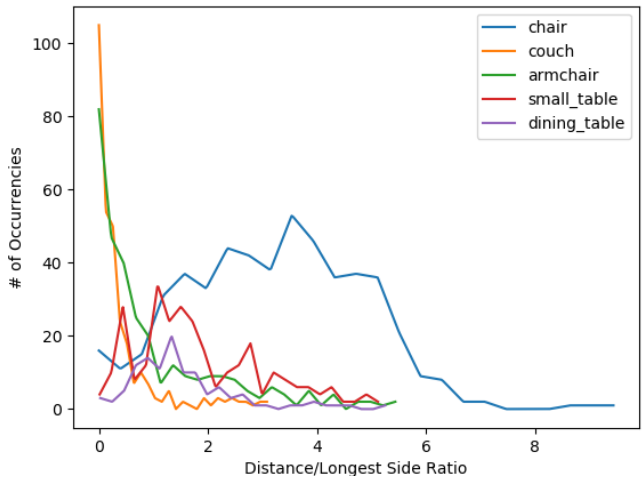
(i) Istogrammi dell'oggetto `couch`



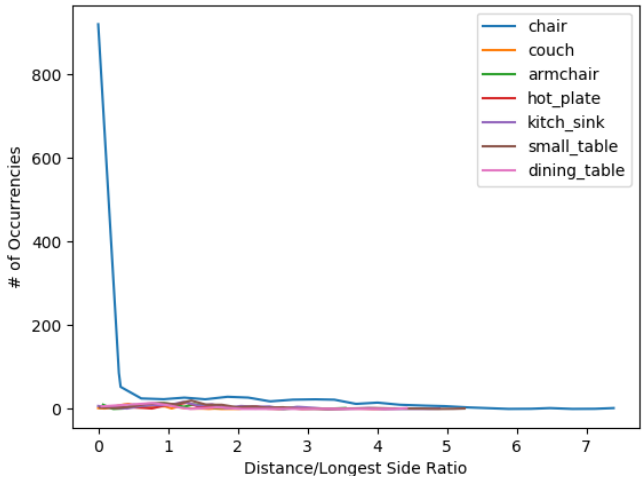
(j) Istogrammi dell'oggetto `armchair`



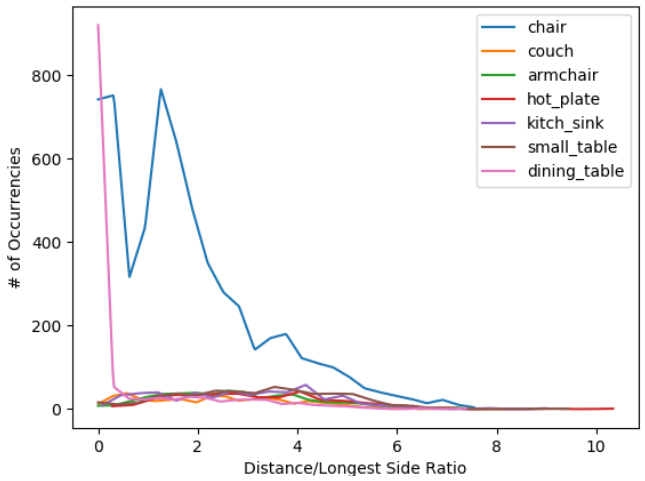
(k) Istogrammi dell'oggetto `table`



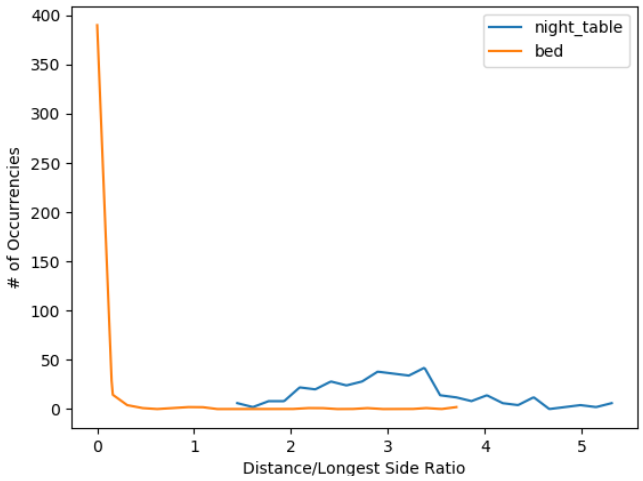
(l) Istogrammi dell'oggetto `small_table`



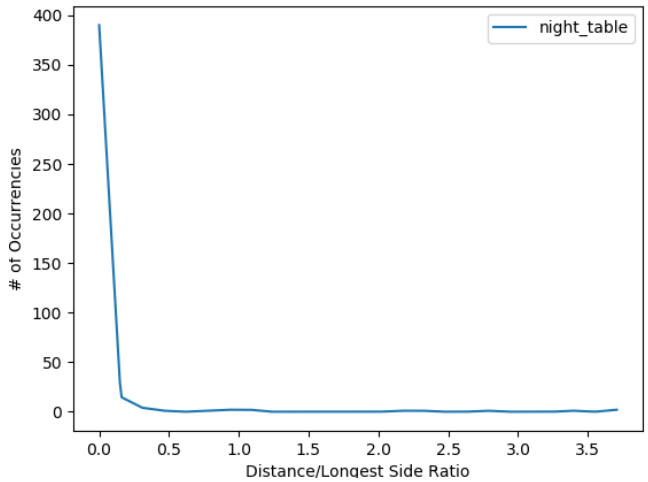
(m) Istogrammi dell'oggetto `dining_table`



(n) Istogrammi dell'oggetto `chair`



(o) Istogrammi dell'oggetto `night_table`



(p) Istogrammi dell'oggetto `bed`

Bibliografia

[1] K. Xu, K. Chen, H. Fu, W.-L. Sun, and S.-M. Hu, "Sketch2scene: Sketch-based co-retrieval and co-placement of 3d models," *ACM Transactions on Graphics*, vol. 32, no. 4, pp. 123:1–123:12, 2013.

[2] S. Capobianco, "Flo2plan dataset manager." `https://github.com/scstech85/Flo2PlanManager`, 2019.

[3] M. Salamino, "Localizzazione di scale in planimetrie," *Tesi di Laurea Triennale in Ingegneria Informatica*, pp. 12–13, 2017/2018.

[4] Wikipedia contributors, "Mathematical morphology — Wikipedia, the free encyclopedia," 2019. [Online; accessed 11-05-2019].

[5] A. Clark and Contributors, "Python imaging library." `https://pillow.readthedocs.io/en/stable/index.html`, 2009–. [Online; accessed 11-05-2019].

[6] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python." `http://www.scipy.org`, 2001–. [Online; accessed 11-05-2019].

[7] Wikipedia contributors, "Confusion matrix — Wikipedia, the free encyclopedia," 2019. [Online; accessed 22-May-2019].

[8] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference* (G. Varoquaux, T. Vaught, and J. Millman, eds.), (Pasadena, CA USA), pp. 11 – 15, 2008.

[9] J. Ellson, E. Gansner, L. Koutsofios, S. North, G. Woodhull, S. Description, and L. Technologies, "Graphviz — open source graph drawing tools," in *Lecture Notes in Computer Science*, pp. 483–484, Springer-Verlag, 2001.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[11] Z. Ziran and S. Marinai, *Object Detection in Floor Plan Images*, pp. 383–394. 09 2018.

[12] C. Goncu and K. Marriott, "Gravvitas: Generic multi-touch presentation of accessible graphics," in *Human-Computer Interaction – INTERACT 2011* (P. Campos, N. Graham, J. Jorge, N. Nunes, P. Palanque, and M. Winckler, eds.), (Berlin, Heidelberg), pp. 30–48, Springer Berlin Heidelberg, 2011.