Alma Mater Studiorum
Università di Bologna

Scuola di Ingegneria

Corso di laurea magistrale in Ingegneria Informatica

# SAFFIRE: System for Autonomous Feature Filtering for Intelligent ROI Estimation

Tesi di laurea magistrale in
Computer Vision and Image Processing M

*Relatore:*
Prof. Luigi Di Stefano

*Candidato:*
Marco Boschi

*Correlatore:*
Dott. Ing. Martino Alessandrini

Sessione III
Anno Accademico 2018–2019

# Indice

# Capitolo 1

# Introduzione

Nell'ambito della *machine vision*, in particolare nell'automazione industriale, sempre più attività vengono delegate a computer e robot per migliorare, aumentare e velocizzare la produzione. Una di queste attività è l'ispezione dei pezzi prodotti (Figura 1.1) per, ad esempio, controllare che non contengano difetti o che le stampe siano state effettuate correttamente. Un sistema di ispezione è collocato lungo il nastro trasportatore che muove i pezzi e utilizza una fotocamera per acquisire le immagini di questi ultimi, opportunamente illuminati, per far risaltare (ove presenti) i difetti da rilevare. Le immagini sono quindi processate da un software che decide se il pezzo soddisfa i requisiti richiesti e, in caso negativo, un robot può procedere a scartarlo.

Le tipologie di analisi possono essere le più disparate, dalla lettura del testo (OCR) alla misura delle dimensioni, passando dal rilevamento di difetti superficiali e molto altro ancora. Nel caso di *Optical Character Recognition* i sistemi attuali richiedono che le righe di testo siano orizzontali e l'isolamento del testo rimuovendo le aree inutili per rendere il processo più veloce. Il processo di misura necessita invece di punti di riferimento specifici nel pezzo in modo da definire un sistema di riferimento che permetta più facilmente di individuare le parti da misurare.

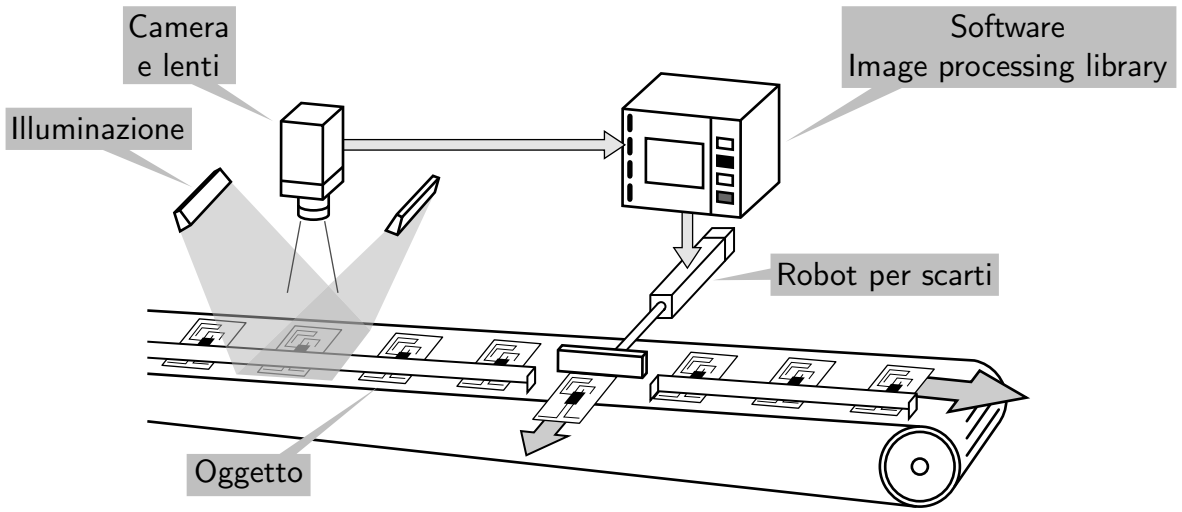



**Figura 1.1:** Schema di un sistema di ispezione per machine vision.

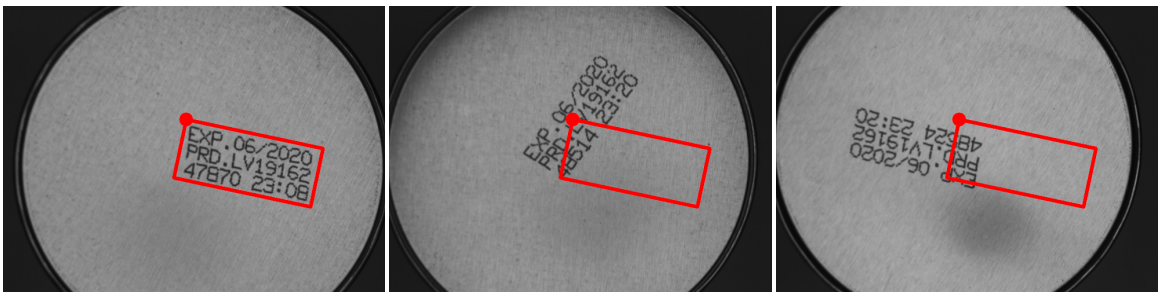



**Figura 1.2:** Esempio di un dataset dove mantenere una ROI fissa non permette di localizzare correttamente il testo di interesse.

Per soddisfare queste necessità occorre individuare all'interno delle immagini acquisite le *regioni di interesse* (ROI) che saranno poi analizzate. Queste non sono solo un'area dell'immagine, ma hanno anche un'orientazione che può essere utilizzata per raddrizzare il pezzo in modo che il software lo veda sempre nello stesso modo e perciò possa lavorare in maniera più semplice. Utilizzare una ROI statica rispetto alla posizione dell'immagine non è sempre una soluzione praticabile perché i pezzi potrebbero essere soggetti a rotazioni e spostamenti all'interno del campo visivo della fotocamera (Figura 1.2). Nel caso in cui una ROI statica non sia praticabile, l'individuazione delle ROI chiede l'intervento umano per posizionarla in ogni immagine oppure la scelta manuale di un pattern che abbia un'orientazione caratteristica e una posizione statica rispetto alla ROI, un pattern di questo tipo è detto *anchor pattern*.

Nel caso più tipico della scelta manuale di un'ancora, il software utilizzerà un componente detto *locatore* che andrà a ricercare nelle nuove immagini l'ancora specificata e rispetto a questa collocherà la ROI desiderata. La scelta dell'ancora non è però un processo semplice e presenta alcuni svantaggi:

- Richiede una conoscenza di che cosa sia un anchor pattern.
- Richiede tempo in quanto l'utente deve ispezionare un campione delle immagini per capire quale pattern, se presente, possa rappresentare un anchor pattern affidabile per l'ispezione richiesta. Questo processo spesso richiede numerosi tentativi provando e confrontando diverse opzioni per identificare la migliore.
- La scelta è soggettiva.

Il locatore proposto, denominato SAFFIRE[1] (System for Autonomous Feature Filtering for Intelligent ROI Estimation), solleva l'utente da questi compiti impegnativi e dispendiosi realizzando un sistema di ancoraggio completamente non supervisionato, basando la scelta solamente sui dati forniti in input. Il processo realizzato non solo è più semplice, ma anche più robusto e rimuove le limitazioni della scelta manuale dell'ancora.

Le caratteristiche principali di SAFFIRE sono:

[1]Pronunciato come *sapphire*, zaffiro in inglese.

- Utilizzando una selezione di immagini con indicata la ROI desiderata, il locatore "impara" quali sono le caratteristiche dell'immagine che possono essere utilizzate per stabilire la posizione della ROI.

- Il locatore sceglie caratteristiche ottimali per l'algoritmo poi utilizzato in fase di posizionamento della ROI su nuove immagini, la scelta è infatti basata su specifiche condizioni oggettive e non su una valutazione soggettiva.

- Le caratteristiche possono essere sparsamente distribuite in qualunque area dell'immagine e non far necessariamente parte di un singolo oggetto o pattern. La selezione di un insieme di caratteristiche di questo tipo sarebbe impossibile per un qualsiasi utente umano, anche esperto.

Nel Capitolo 2 vengono presentate tecniche affermate quali i metodi utilizzati per ricavare e confrontare *feature* delle immagini, le strategie di esplorazione di grafi, gli approcci per confrontare immagini a livello di contenuto e alcuni calcoli geometrici. Il Capitolo 3 presenta in dettaglio il funzionamento del locatore illustrando prima la struttura generale dell'algoritmo utilizzato come punto di partenza di questo lavoro e successivamente illustrando le modifiche proposte per migliorare la costruzione e ricerca del modello. I dataset utilizzati per valutare le capacità del locatore proposto sono presentati nel Capitolo 4 e i risultati ottenuti sono illustrati nel Capitolo 5.

# Capitolo 2

# Background

In questo capitolo vengono presentate tecniche appartenenti allo stato dell'arte della computer vision e non solo, che costituiscono la base del funzionamento di SAFFIRE.

Nella Sezione 2.1 sono presentati alcuni metodi per estrarre *feature* da un'immagine assieme ai processi necessari per assegnare loro un *descrittore* che svolga il ruolo di "impronta digitale" per riconoscerle e confrontarle. La Sezione 2.2 presenta due approcci per esplorare i grafi e individuare al loro interno un percorso migliore secondo un'opportuna definizione. Nella Sezione 2.3 sono presentati alcuni concetti volti all'analisi delle immagini utilizzando il loro contento in senso più globale, utili anche per confrontare le immagini stesse. Infine la Sezione 2.4 illustra calcoli geometrici per misurare la sovrapposizione di aree e il calcolo di trasformazioni tra due gruppi di dati per farli sovrapporre.

## 2.1 Ricerca di pattern

La ricerca di pattern all'interno delle immagini è uno dei metodi principali per la ricerca di oggetti e quello utilizzato dal locatore per individuare l'origine del sistema di riferimento usato per posizionare la ROI.

Di seguito sono presentate alcune delle tecniche che permettono questa ricerca. Il secondo metodo di ricerca è invece basato sul confronto delle immagini a livello più basso e di contenuto, confrontando per esempio i colori dei pixel, e verrà presentato successivamente.

### 2.1.1 Feature e descrittori

Uno dei metodi più versatili nella ricerca di oggetti all'interno di immagini consiste nell'utilizzo di *feature* locali robuste, permettendo di trovare applicazione anche in ambienti difficilmente controllabili. Il punto focale è che queste feature devono essere invarianti a diversi aspetti come l'uso di camere differenti, a diverse distanze e con condizioni di luce variabili.

L'uso di feature per riconoscere gli oggetti si basa quindi sull'individuare le stesse feature in immagini diverse. Questo è fatto estraendo in maniera indipendente le

feature da diverse immagini e confrontandole fra di loro. Le feature sono però solo delle caratteristiche rilevanti dell'immagine come le coordinate di un pixel, per confrontarle occorre perciò assegnare a ognuna una sua "impronta digitale" detta *descrittore*. Poiché le feature sono *locali* il descrittore deve essere calcolato non con tutto il contenuto dell'immagine, ma utilizzando solamente i pixel nell'intorno della feature stessa con un processo che rispetti il vincolo di invarianza.

Stabilire le corrispondenze tra due immagini è un processo che si articola in 3 fasi:

1. individuazione delle feature nelle immagini;
2. calcolo dei descrittori per ogni feature ottenendo un vettore di numeri o una stringa binaria per ognuna;
3. calcolo dei descrittori corrispondenti tra le immagini, noti come *match*.

#### 2.1.1.1 EDL

EDLines (EDL in breve)[3] è un estrattore di segmenti, noti come *key-line*, basato sul rilevatore di contorni *Edge Drawing* (ED)[36] che restituisce invece che una mappa binaria dei contorni, delle sequenze continue di pixel che determinano il contorno e non dei pixel che possono essere isolati. La rilevazione di questi contorni si articola in quattro fasi:

1. l'immagine viene processata utilizzando un filtro di Gauss per rimuove il rumore;
2. l'intensità e direzione del gradiente vengono calcolate per ogni pixel dell'immagine filtrata;
3. un sottoinsieme di pixel, detti ancore, viene individuato scegliendo tra quelli con una probabilità estremamente elevata di essere contorni, cioè i picchi della mappa del gradiente;
4. le ancore vengono connesse fra loro tracciando un percorso che si muove lungo i valori massimi del gradiente.

Una volta ottenute le sequenze di pixel che costituiscono i contorni, le linee vengono individuate utilizzando il metodo dei minimi quadrati aggiungendo un pixel alla volta dalla sequenza fintanto che l'errore non supera 1 pixel. A questo punto una nuova linea viene generata con i pixel aggiunti fino a quel momento e si comincia a costruire una nuova linea con i pixel rimanenti.

Le linee sono infine validate utilizzando un metodo presentato in [12] e basato su un modello probabilistico dove i segmenti sono eccezioni al modello. Per realizzare il filtro viene definito il numero di falsi allarmi: per un segmento di lunghezza $n$ in un'immagine di dimensione $N \times N$ e con $k$ pixel il cui gradiente è allineato alla direzione del segmento questo numero è

$$NF(n,k) = N^4 \sum_{i=k}^{n} p^i (1-p)^{n-i},$$

dove $p$ è la probabilità che un singolo pixel del segmento abbia il gradiente diretto lungo la direzione del segmento. Il segmento costruito viene considerato valido se $NF(n,k) \leq 1$.

Il descrittore utilizzato è noto come *Line Band Descriptor* (LBD)[39] calcolato partendo da una regione attorno al segmento nota come *line support region* (LSR) lunga quanto il segmento che è posizionato parallelo ai bordi a metà dell'altezza. La LSR è divisa in altezza in $m$ bande $\{B_1, \ldots, B_m\}$ parallele al segmento ognuna larga $w$ pixel, l'altezza totale è quindi $m \cdot w$.

All'interno della LSR è definito un sistema di riferimento usando la direzione del segmento $\boldsymbol{d}_L$, data dalla direzione del gradiente, e la sua perpendicolare in senso orario $\boldsymbol{d}_\perp$. Con questo sistema di riferimento è possibile riportare il gradiente di un pixel $\boldsymbol{g}$ in valori relativi alla LSR $\boldsymbol{g}'$.

Sulla base della posizione lungo $\boldsymbol{d}_\perp$, a ogni riga di pixel vengono assegnati due fattori ottenuti da funzioni gaussiane. Il primo fattore è globale e definito come

$$f_g(i) = \frac{1}{\sqrt{2\pi}\sigma_g} e^{-\frac{d_i}{2\sigma_g^2}}, \qquad \sigma_g = \frac{1}{2}(mw - 1),$$

dove $d_i$ è la distanza della $i$-esima riga da quella centrale. Il secondo fattore è locale a ogni banda e assegna a ogni $k$-esima riga appartenente a $B_j$ e le vicine $B_{j-1}$ e $B_{j+1}$ il valore

$$f_l(k) = \frac{1}{\sqrt{2\pi}\sigma_l} e^{-\frac{d'_k}{2\sigma_l^2}}, \qquad \sigma_l = w,$$

dove $d'_k$ è la distanza della $k$-esima riga dalla riga centrale di $B_j$.

Il descrittore del segmento è ottenuto concatenando i descrittori di ogni banda $BD_j$ ottenuti considerando le righe della banda stessa e le due vicine, con l'eccezione di $B_1$ e $B_m$ che hanno un solo vicino. Per una riga $k$ vengono definiti quattro valori accumulando le componenti dei gradienti dei pixel di quella riga:

$$v1_j^k = \lambda \sum_{\boldsymbol{g}'_{\boldsymbol{d}_\perp} > 0} \boldsymbol{g}'_{\boldsymbol{d}_\perp}, \quad v2_j^k = \lambda \sum_{\boldsymbol{g}'_{\boldsymbol{d}_\perp} < 0} -\boldsymbol{g}'_{\boldsymbol{d}_\perp}, \quad v3_j^k = \lambda \sum_{\boldsymbol{g}'_{\boldsymbol{d}_L} > 0} \boldsymbol{g}'_{\boldsymbol{d}_L}, \quad v4_j^k = \lambda \sum_{\boldsymbol{g}'_{\boldsymbol{d}_L} < 0} -\boldsymbol{g}'_{\boldsymbol{d}_L},$$

dove $\lambda = f_g(k) f_l(k)$.

Per costruire $BD_j$ viene prima definita la *band descriptor matrix* (BDM) composta aggregando i valori appena definiti:

$$BDM_j = \begin{pmatrix} v1_j^1 & v1_j^2 & \cdots & v1_j^n \\ v2_j^1 & v2_j^2 & \cdots & v2_j^n \\ v3_j^1 & v3_j^2 & \cdots & v3_j^n \\ v4_j^1 & v4_j^2 & \cdots & v4_j^n \end{pmatrix} \in \mathbb{R}^{4 \times n}, \qquad n = \begin{cases} 2w, & j \in \{1, m\} \\ 3w, & \text{altrimenti} \end{cases}.$$

Una volta calcolati i vettori media $M_j$ e deviazione standard $S_j$, il descrittore della banda è definito come la loro concatenazione $BD_j = (M_j^T, S_j^T) \in \mathbb{R}^8$ e la concatenazione di tutti i descrittori di banda costituisce il descrittore del segmento come vettore

in $\mathbb{R}^{8m}$.

Per velocizzare il calcolo dei match, i descrittori composti da numeri reali vengono trasformati in descrittori binari scegliendo 32 coppie di BD e ognuna è confrontata bin a bin ottenendo stringhe di 8 bit, per un descrittore totale composto da 256 bit.

Sulla base di quanto proposto in [30], ogni descrittore viene diviso in $m$ parti e ognuna di queste utilizzata per inserirlo in $m$ distinte tabelle di hashing. Questo permette di estrarre in maniera rapida tutti quei descrittori che hanno almeno una parte simile a un descrittore utilizzato come chiave di ricerca e quindi filtrare questi possibili descrittori simili mantenendo solo quelli che hanno una distanza di Hamming[20] non superiore a una soglia $r$, cioè che non differiscono per più di $r$ bit.

#### 2.1.1.2 LSD

Line Segment Detector (LSD)[19] utilizza un approccio diverso da EDL, ma comunque basato sull'idea che il gradiente possa essere usato per individuare i contorni degli oggetti, sulla base di questo i segmenti individuati sono i contorni localmente rettilinei.

Il processo comincia calcolando il gradiente per ogni pixel e costruendo una mappa, detta *level-line field*, in cui a ogni pixel è associata la direzione dalla linea (*level-line)* che individua, perpendicolare al gradiente stesso. Questo campo è quindi segmentato in regioni connesse i cui pixel condividono la stessa level-line a meno di una soglia $\tau$. Queste regioni sono dette *line support region* e sono i candidati per i segmenti restituiti.

A ogni regione viene associato il rettangolo di area minima con orientazione uguale alla direzione del principale asse d'inerzia della regione stessa. Detto $n$ il numero di pixel assegnati al rettangolo e $k$ il numero di questi pixel la cui level-line è uguale, a meno di $\tau$, all'orientazione del rettangolo, il segmento viene validato utilizzando il test proposto in [12] e illustrato per EDL in Sezione 2.1.1.1.

Una volta ottenuti i segmenti questi sono descritti utilizzando LBD[39] e comparati nello stesso metodo utilizzato per EDL.

#### 2.1.1.3 ORB

ORB[35] utilizza come feature dei singoli punti, detti *key-point*, ottenuti utilizzando FAST,[33,34] un riconoscitore di angoli, e una versione migliorata di BRIEF[9] per costruire il descrittore.

Detto $p$ un pixel dell'immagine in scala di grigi con intensità $I_p$, FAST considera un cerchio di raggio 3 individuando 16 pixel sulla circonferenza. $p$ è considerato una feature se in una sequenza continua di almeno $n = 9$ pixel, tutti sono più luminosi o più scuri del pixel centrale a meno di una soglia $t$. Detta $S$ una sequenza continua di $n$ pixel della circonferenza attorno a $p$, $p$ è una feature se una delle seguenti due condizioni è verificata:

- $\exists S, \forall x \in S, I_x > I_p + t$;
- $\exists S, \forall x \in S, I_x < I_p - t$.

Questa definizione risulta in molte feature concentrate lungo i contorni di aree con diverse intensità e FAST da solo non offre una misura dell'intensità dell'angolo nota come *cornerness*. Per poter filtrare le feature viene selezionato il numero massimo $N$ di feature desiderate e la soglia $t$ è impostata per ottenere più di $N$ feature, la cornerness viene calcolata usando la misura proposta da Harris in [21] e solo le prime $N$ feature, dopo averle ordinate per cornerness decrescente, vengono mantenute.

Questo processo viene applicato non solo all'immagine originale, ma a una piramide di immagini costruita generando scale sempre più piccole dell'immagine di partenza. Questa tecnica permette di rilevare feature che alla dimensione originale sembrano non rilevanti, ma che a dimensione ridotta invece lo sono.

Le feature così individuate non hanno però una nozione dell'orientazione, fondamentale per poter individuare oggetti soggetti a rotazioni. Per sopperire a questa mancanza viene utilizzato il *centroide dell'intensità*.[32]

Questo centroide è calcolato come il punto

$$\boldsymbol{C} = \left( \frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right),$$

dove $m_{ij}$ sono i momenti basati sull'intensità $I$

$$m_{ij} = \sum_{x,y} x^i y^j I(x, y).$$

Le coordinate utilizzate hanno come origine il punto $\boldsymbol{O}$ corrispondente al pixel $p$ e solo i pixel appartenenti a un intorno di $p$ con sia $x$ che $y$ appartenenti a $[-r, r]$ vengono considerati.

L'orientazione della feature è infine calcolata come la direzione del vettore $\overrightarrow{\boldsymbol{OC}}$:

$$\theta = \text{atan2}(m_{01}, m_{10}),$$

dove $\text{atan2}(y, x)$ è la variante dell'arcotangente che considera anche il quadrante dove si trova il punto per dare un angolo nell'intervallo $[-\pi, \pi]$.

Il descrittore utilizzato è una versione modificata di BRIEF per tenere in considerazione anche l'orientazione. BRIEF restituisce come descrittore una stringa di bit in cui ognuno rappresenta un confronto di valori di intensità di diversi pixel nell'intorno del pixel $p$ scelto come feature.

Per costruire il descrittore viene innanzitutto isolata una sezione dell'immagine di $31 \times 31$ pixel centrata intorno a $p$. Il test binario utilizzato per generare un bit del descrittore è definito come

$$\tau(q, r) = \begin{cases} 1, & I'(q) < I'(r) \\ 0, & I'(q) \geq I'(r) \end{cases},$$

dove $I'(q)$ indica non l'intensità del pixel $q$, ma la somma delle intensità dei pixel in una finestra di dimensione $5 \times 5$ centrata intorno a $q$. Il descrittore, di lunghezza $n$, è quindi costruito come

$$\sum_{i=1}^{n} 2^{i-1} \tau(q_i, r_i),$$

richiedendo pertanto di definire un pattern di $n = 256$ coppie di pixel su cui basare i test. Per rendere il descrittore legato all'orientazione occorre infine che il pattern sia ruotato secondo l'orientazione $\theta$, questo passaggio viene semplificato discretizzando l'orientazione con incrementi di 12° e calcolando a priori i pattern per ogni valore.

Il pattern utilizzato non può essere qualunque, ma deve garantire una media di 0.5, la massima varianza e i test non devono essere correlati fra di loro per garantire le capacità ottimali di ricerca.

Per determinare i match fra descrittori vengo utilizzate tecniche di Locality Sensitive Hashing (LSH)[17] che sfruttano la natura binaria del descrittore per semplificare la ricerca di match inserendo ogni descrittore in più tabelle di hashing. In fase di ricerca questo permette di isolare un sottoinsieme dei descrittori che, con alta probabilità, contengono quello più simile. I descrittori più simili sono quindi isolati utilizzando un metodo di forza bruta basato sulla distanza di Hamming.

### 2.1.2 Trasformata di Hough generalizzata

La trasformata di Hough (HT)[24] permette ricercare in un'immagine una forma geometrica descritta analiticamente non attraverso una ricerca globale, ma con una locale. Questo è possibile mappando la ricerca nello *spazio dei parametri* caratteristico della forma cercata, per esempio per una retta espressa come $y = mx + q$, lo spazio dei parametri è definito dall'asse $m$ e dall'asse $q$.

La ricerca parte dalla rilevazione dei contorni nell'immagine interessata, quindi per ogni punto rilevato viene tracciato nello spazio dei parametri il luogo dei punti corrispondenti a tutti i possibili parametri che determinano la forma cercata passante per il punto. Per individuare le forme presenti occorre cercare i punti dello spazio dei parametri dove si incontrano più luoghi di punti.

Le intersezioni cercate non saranno mai precise, quindi lo spazio dei parametri viene quantizzato creando così una struttura nota come *accumulation array* (AA) e ogni volta che un luogo di punti attraversa una cella, *vota* per quella combinazione di parametri. Dopo aver raccolto tutti i voti, le celle che risultano essere massimi locali dell'AA sono possibili forme cercate.

La generalizzazione della HT (GHT) può avvenire sia verso forme arbitrarie (sempre basata sulla rilevazione dei bordi) sia verso oggetti generici sfruttando feature locali, anche se i concetti di base sono molto simili, quest'ultimo è il caso di interesse per SAFFIRE.[6,18,26]

Per poter effettuare la ricerca dell'oggetto occorre una fase iniziale in cui viene creato il modello a partire dalle feature rilevate sull'immagine. Scelto un punto di riferimento $\boldsymbol{P}_C$ arbitrario sull'immagine, ad esempio il centro della stessa o il centro della ROI nel caso di questo lavoro, per ogni feature con centro $\boldsymbol{P}_i$ viene calcolato un vettore, detto *joining vector*

$$\boldsymbol{V}_i = \boldsymbol{P}_C - \boldsymbol{P}_i.$$

Tutti i vettori $\boldsymbol{V}_i$ compongono il cosiddetto *modello a stella*.

Per effettuare la ricerca vengono estratte dall'immagine le stesse feature usate per costruire il modello e poi vengono calcolati i match tra queste e quelle del modello. Per ogni match tra una feature $F_i$ del modello e $\tilde{F}_i$ dell'immagine, è possibile esprimere un voto per la posizione dell'oggetto, o meglio del punto di riferimento, nell'AA alla posizione data da

$$\tilde{\boldsymbol{P}}_{C_i} = \tilde{\boldsymbol{P}}_i + \boldsymbol{V}_i.$$

Questo voto non tiene però conto delle possibili rotazioni e scale che possono intervenire all'interno del match, che dunque devono essere compensate tenendo conto delle informazioni di orientamento e dimensione di ogni feature, dunque si può individuare l'angolo di rotazione e fattore di scala del match come

$$\Delta\varphi_i = \varphi_i - \tilde{\varphi}_i, \qquad s_i = \frac{S_i}{\tilde{S}_i}$$

dove $\varphi_i$ e $\tilde{\varphi}_i$ sono l'orientazione rispettivamente della feature del modello e dell'immagine, $S_i$ e $\tilde{S}_i$ la dimensione o scala caratteristica rispettivamente della feature del modello e dell'immagine.

Il voto può perciò essere corretto in

$$\tilde{\boldsymbol{P}}_{C_i} = \tilde{\boldsymbol{P}}_i + s_i \cdot \boldsymbol{R}(\Delta\varphi_i)\boldsymbol{V}_i,$$

dove $\boldsymbol{R}(\Delta\varphi_i)$ rappresenta la matrice di rotazione di un angolo pari a $\Delta\varphi_i$.

Come nel caso della HT semplice le possibili posizioni (complete di rotazione e scala) sono individuate dai massimi locali nell'AA. Per poter ottenere l'orientazione corretta dell'oggetto nell'immagine occorre infine calcolare una similitudine, cioè una composizione di traslazione, rotazione e scala uniforme, che riporti l'oggetto sull'immagine analizzata. Questa può essere ottenuta tenendo conto di tutti i match $\langle \boldsymbol{P}_i, \tilde{\boldsymbol{P}}_i \rangle$ che hanno votato per la posizione considerata e utilizzando un metodo di stima robusto come RANSAC[15] o i minimi quadrati.

## 2.2 Ricerca su grafi

L'elaborazione di grafi è un ambito molto vasto, per il locatore risulta di particolare interesse la ricerca di percorsi con certe caratteristiche all'interno di grafi. Per consentire questa ricerca nel grafo deve essere definito un nodo di partenza che funge da punto di origine e una condizione che stabilisca quali nodi siano da considerare come obiettivo da raggiungere. Il percorso desiderato è quello che minimizza il *costo di cammino*, una funzione definita sulla base di quali archi e nodi compongono il percorso.

### 2.2.1 Esaustiva

L'algoritmo denominato SumPath[14] effettua una ricerca esaustiva in un albero effettuando due esplorazioni dello stesso: la prima per individuare il nodo foglia dove il percorso migliore termina e la seconda per trovare il percorso che porta a questo.

La prima esplorazione procede dal nodo radice ed esplora tutti i suoi figli in ordine, l'esplorazione di un nodo altro non è che esplorare in maniera ricorsiva i figli in ordine, trasformandosi in una ricerca depth-first. Quando un nodo foglia viene incontrato, se il costo del cammino che l'ha raggiunta è il migliore, la foglia obiettivo viene aggiornata. Nell'implementazione di [14] il percorso migliore ha la somma dei valori di tutti i nodi sul percorso massima.

Utilizzando la foglia obiettivo come input, la seconda esplorazione esegue nuovamente una ricerca depth-first fino a ritrovare la foglia, quindi l'albero viene risalito fino alla radice per individuare il percorso. L'algoritmo può facilmente essere adattato a lavorare con la tipologia di grafi utilizzata da SAFFIRE.

In quanto ricerca esaustiva, il risultato fornito è certamente corretto, tuttavia il calcolo è molto dispendioso dovendo esplorare interamente il grafo. Utilizzando invece informazioni aggiuntive sul contenuto dei nodi è possibile ignorare alcuni nodi e così velocizzare la ricerca, tuttavia questo è compito di strategie di ricerca informate.

### 2.2.2 A*

A* è un algoritmo di ricerca informata[22] e uno dei migliori algoritmi di questa categoria.[38] Contrariamente a una ricerca esaustiva, non tutti i percorsi vengono valutati ma solo alcuni utilizzando una funzione *euristica* come guida.

L'euristica è un valore associato a ogni nodo che dà una stima ottimista del costo di cammino rimanente per raggiungere un nodo obiettivo. Nella ricerca, l'algoritmo è anche supportato da due strutture dati: la *lista dei nodi chiusi* e la lista dei nodi aperti o *frontiera*. La prima contiene un riferimento a tutti i nodi già valutati, la seconda una parte dei nodi non ancora valutati (non necessariamente tutti) che devono essere considerati successivamente; quest'ultima non considera però i soli nodi ma anche il percorso fatto per raggiungerli.

L'algoritmo procede ciclicamente eseguendo:

1. un nodo viene estratto dalla frontiera;
2. se il nodo è considerato un obiettivo viene restituito come soluzione e l'algoritmo termina;
3. altrimenti se il nodo è già parte dei nodi chiusi viene scartato e si passa all'iterazione successiva;
4. altrimenti il nodo viene aggiunto alla lista dei nodi chiusi e tutti i nodi raggiungibili vengono aggiunti alla frontiera.

In caso la frontiera diventi vuota allora non è possibile trovare una soluzione.

Affinché l'algoritmo risulti completo e corretto è necessario che l'euristica soddisfi certi requisti e che la scelta del nodo da espandere non sia casuale. Identificando il costo di cammino di un nodo $x$ come $g(x)$ e la sua euristica con $h'(x)$, a ogni nodo nella frontiera è quindi associato un valore, noto come *funzione di valutazione*,

$$f(x) = g(x) + h'(x).$$

Questo permette di tenere la frontiera ordinata per $f(x)$ crescente e di scegliere sempre il nodo con $f(x)$ minima come quello da espandere.

Per quanto riguarda l'euristica, questa deve essere *ammissibile* o *ottimistica*, cioè non sovrastimare l'effettiva distanza dall'obiettivo $h(x)$:

$$h'(x) \leq h(x).$$

Poiché l'algoritmo viene applicato a un grafo e non solo ad un albero si aggiunge anche il requisito di *consistenza* che può essere espresso in due vincoli per la funzione $h'(x)$:

1. $h'(x) = 0$ se $x$ è un obiettivo;
2. $h'(x) \leq c(x, a, x') + h'(x')$, cioè l'euristica per un nodo deve essere sempre minore (o al più uguale) al costo $c$ per raggiungere un qualsiasi nodo $x'$ direttamente raggiungibile da quello considerato (con una qualunque azione $a$) sommato all'euristica di $x'$.

Questo secondo requisito è fondamentale per la correttezza dell'algoritmo quando la lista dei nodi chiusi contiene solo lo stato corrispondente al nodo e non il percorso fatto per raggiungerlo. Sebbene questo renda la scelta dell'euristica più complessa, le strutture dati necessarie risultano più semplici e l'occupazione di memoria minore.

## 2.3 Confronto del contenuto delle immagini

La ricerca di oggetti basata su pattern è in grado di offrire strumenti più avanzati per individuare gli oggetti, tuttavia è possibile che vengano proposte più posizioni quando solo una è desiderata.

Di seguito sono presentate alcune tecniche per costruire dei descrittori che possono essere usati per confrontare il contenuto delle immagini, o in particolare delle ROI, anche se queste non hanno la stessa dimensione. Per creare un descrittore efficace per il modello della ROI è opportuno utilizzare tutte le immagini di training, pertanto occorre identificare anche un metodo per combinare più descrittori dello stesso tipo in uno unico mantenendo la stessa dimensione.

### 2.3.1 Istogrammi

Un primo e semplice descrittore è basato sull'istogramma dei livelli di grigio. Questo istogramma è basato su immagini in scala di grigi oppure su un solo canale di un'immagine a colori in modo che il valore di ogni pixel sia rappresentato da un singolo valore, solitamente i valori interi $\{0, 1, 2, \ldots, 255\}$.

L'istogramma è costruito ponendo sulle ascisse i possibili valori dei pixel e sulle ordinate il numero di pixel. Per ogni valore di grigio, detto *bin* dell'istogramma, è quindi assegnato il numero di pixel che ha quel preciso valore.

L'uso di questo descrittore così come appena descritto presenta un problema quando deve essere confrontato con l'istogramma ricavato da immagini di dimensioni diverse, infatti il numero di pixel differisce e le distanze tra bin corrispondenti non sarebbero rilevanti, per questo motivo solitamente si procede a normalizzare il contenuto di ogni bin dividendolo per il numero totale di pixel dell'immagine originaria. L'istogramma così ottenuto contiene in ogni bin valori in $[0, 1]$, cioè la probabilità di presentarsi del livello di grigio corrispondente, e l'istogramma stesso può essere visto come la *funzione di massa di probabilità* (PMF) di ogni pixel quando è visto come una variabile casuale il cui valore è il colore. Il descrittore così costruito è denominato *GLH*.

#### 2.3.1.1 Equalizzazione

Un secondo problema emerge nel confronto di istogrammi ricavati da immagini con differenti illuminazioni, cosa che può risultare in istogrammi traslati orizzontalmente uno rispetto all'altro, seppure con andamento simile. Per sopperire a questo problema, una possibilità è di ricorrere all'equalizzazione degli istogrammi che non richiede alcun parametro per migliorare il contrasto dell'immagine utilizzando tutto l'intervallo di valori disponibili per rappresentare immagini in scala di grigi.

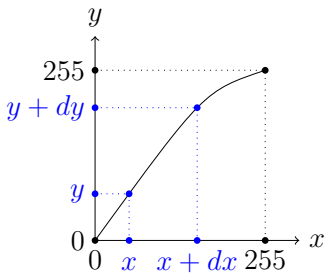



**Figura 2.1:** Trasformazione $T$ utilizzata per equalizzare l'istogramma.

Per massimizzare il contrasto, l'istogramma dovrebbe diventare piatto per usare effettivamente tutti i possibili valori, ma questo è possibile solo per istogrammi continui e non discreti come quelli dei livelli di grigio. Per procedere comunque anche in casi discreti si utilizza perciò un'approssimazione della funzione utilizzata per appiattire il caso continuo, che viene costruita utilizzando solamente elementi di probabilità.

Per trovare la trasformazione $T$ cercata viene considerata una variabile casuale $x$ e $T$ (Figura 2.1) viene considerata strettamente monotona crescente tale che

$$x \in [0, 1] \Leftrightarrow y = T(x) \in [0, 1].$$

La *funzione di densità di probabilità* (PDF) per $x$ può essere qualunque mentre $T$ verrà definita in modo che la PDF di $y$ sia piatta.

Data la PDF di una variabile, la probabilità che la variabile appartenga a un certo intervallo è l'area sottesa dalla PDF nel solo intervallo di interesse. Considerando la

corrispondenza fra $x$ e $y$ in Figura 2.1, segue dalla monotonicità di $T$ che un qualunque $\tilde{x} \in [x, x + dx]$ avrà un'immagine nell'intervallo $[y, y + dy]$, pertanto la probabilità che $x$ appartenga a $[x, x + dx]$ è la stessa che $y$ appartenga a $[y, y + dy]$. Queste probabilità possono essere espresse come

$$p_x(x)\,dx = p_y(y)\,dy \Rightarrow p_y(y) = p_x(x)\frac{dx}{dy},$$

dove $\frac{dx}{dy}$ è la derivata della funzione inversa $x = T^{-1}(y)$, che corrisponde al reciproco della derivata di $T$. La PDF in input è quindi ora legata a quella in output attraverso la derivata di $T^{-1}$.

È possibile considerare $T$ come la *funzione di distribuzione cumulativa* (CDF) di probabilità di $x$, cioè la probabilità che $x$ sia minore di un certo valore calcolabile come l'area sottesa dalla PDF cambia al variare del limite superiore e fissando quello inferiore a 0. $T$ così definita è una funzione monotona strettamente crescente dal dominio di $x$ a $[0, 1]$. Con questa scelta la trasformazione diventa

$$y = T(x) = \int_0^x p_x(\xi)\,d\xi$$

rendendo $y$ una variabile casuale uniforme. Questo avviene perché la derivata di una funzione integrale è la funzione all'interno dell'integrale, da cui segue che la PDF di $y$ sia come segue:

$$p_y(y) = p_x(x)\frac{dx}{dy} = p_x(x)\frac{1}{\frac{dy}{dx}} = p_x(x)\frac{1}{p_x(x)} = 1.$$

Discrettizzando questo risultato per applicarlo a un istogramma dei livelli di grigio non c'è più una teoria a sostegno della trasformazione ottenuta e l'istogramma non sarà appiattito, tuttavia l'intervallo di valori utilizzato si allarga migliorando il contrasto. La discretizzazione avviene convertendo l'integrale in una sommatoria e sostituendo la PMF alla PDF:

$$\begin{cases} N = \sum_{i=0}^{255} h(i) \\ p(i) = \dfrac{h(i)}{N} \end{cases} \Rightarrow j = T(i) = \sum_{k=0}^{i} p(i) = \frac{1}{N}\sum_{k=0}^{i} h(i) \Rightarrow j = \frac{255}{N}\sum_{k=0}^{i} h(i),$$

dove $p(i)$ rappresenta la PMF come normalizzazione del conteggio dei pixel $h(i)$ contenuto nell'istogramma e il passaggio finale annulla la normalizzazione riportando le nuove posizioni dei bin $j$ ai valori $\{0, 1, 2, \ldots, 255\}$.

L'ultimo passaggio per ottenere il descrittore denominato *EqGLH* consiste nel normalizzare l'istogramma equalizzato dividendo i valori per $N$.

#### 2.3.1.2 Misura della distanza

Il confronto che si si vuole attuare tra gli istogrammi, ricordando che possono essere visti come distribuzioni di probabilità, deve anche tenere conto di possibili traslazioni

dell'andamento. Una misura di distanza che soddisfa questo requisito è quella di Bhattacharyya:[8]

$$d(\boldsymbol{H}_1, \boldsymbol{H}_2) = -\ln\left[\sum_{i=0}^{N-1}\sqrt{\boldsymbol{H}_1(i)\cdot\boldsymbol{H}_2(i)}\right]$$

dove $N$ è il numero di bin degli istogrammi, $N = 256$ nel caso di istogrammi dei livelli di grigio.

Questa formulazione tuttavia non risulta essere una metrica in quanto non soddisfa la disuguaglianza triangolare, pertanto l'implementazione effettivamente utilizzata è

$$d(\boldsymbol{H}_1, \boldsymbol{H}_2) = \sqrt{1 - \frac{1}{\sqrt{\bar{\boldsymbol{H}}_1\bar{\boldsymbol{H}}_2 N^2}}\sum_{i=0}^{N-1}\sqrt{\boldsymbol{H}_1(i)\cdot\boldsymbol{H}_2(i)}},$$

nota come *distanza di Hellinger*,[23] dove $\bar{\boldsymbol{H}}_j$ è la media dei valori dei bin dell'istogramma.

#### 2.3.1.3 Combinazione

Tenendo in considerazione che questi istogrammi rappresentano conteggi di pixel il descrittore combinato può essere costruito sommando fra di loro tutti i possibili istogrammi prima di effettuare la normalizzazione o l'equalizzazione.

### 2.3.2 Filtri di Gabor

I filtri di Gabor sono filtri di convoluzione studiati per rilevare la presenza di informazioni a una specifica frequenza lungo una data direzione nella regione attorno al punto considerato. Utilizzando questa caratteristica, i filtri possono essere utilizzati per estrarre feature utili per lo studio delle texture di un'immagine.[31]

Il filtro è definito come

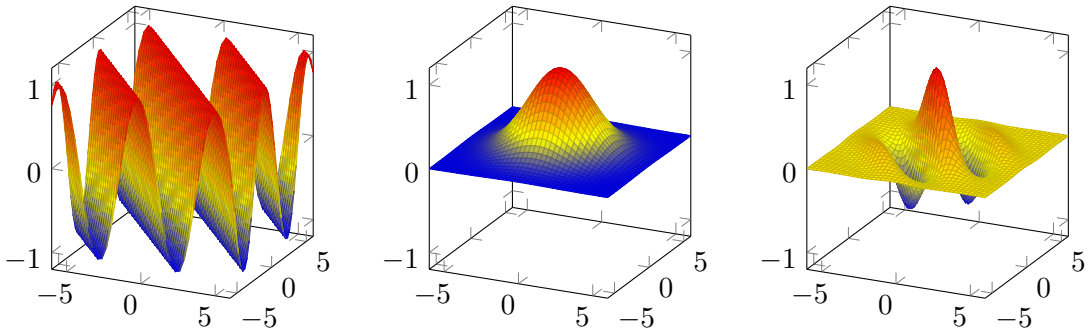
$$g(x, y) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right)\exp\left[i\left(2\pi\frac{x'}{\lambda} + \psi\right)\right],$$
$$x' = x\cos\theta + y\sin\theta, \qquad y' = -x\sin\theta + y\cos\theta,$$

ma per processare immagini al fine di rilevare le texture solo la parte reale è utilizzata e il filtro può essere visto come una sinusoide 2D modulata da una gaussiana 2D (Figura 2.2):

$$g(x, y) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right)\cos\left(2\pi\frac{x'}{\lambda} + \psi\right).$$

Il filtro utilizza i seguenti parametri:

- $\theta$ è la rotazione della sinusoide nel piano;
- $\lambda$ è un fattore di scala della sinusoide aumentandone il periodo tanto più il valore è grande;




**(a)** Sinusoide. **(b)** Gaussiana. **(c)** Filtro di Gabor.

**Figura 2.2:** Costruzione grafica di un filtro di Gabor.

- $\psi$ è la fase della sinusoide;
- $\sigma$ è la dimensione della gaussiana;
- $\gamma$ altera la dimensione della gaussiana allargandola tanto più il valore è piccolo lungo la direzione perpendicolare a $\theta$.

Un singolo filtro è in grado di rilevare la presenza di pattern lungo una direzione specifica a una determinata scala, per questo motivo il descrittore deve essere costruito utilizzando filtri a diverse orientazioni e scale. Per ottenere orientazioni distinte $\theta$ assume i valori

$$\theta = i \cdot \frac{\pi}{n}, \qquad i \in \{0, 1, 2, \ldots, n-1\}$$

dove $n$ è il numero di orientazioni da considerare. Per considerare più scale occorre variare in maniera coerente $\sigma$ e $\lambda$, il primo per determinare la dimensione del filtro e il secondo per aumentare in maniera proporzionale lo spessore delle bande. I valori sono quindi calcolati come

$$\sigma = \sigma_0 + j \cdot \Delta s, \quad \lambda = \lambda_0 + j \cdot \Delta s, \qquad i \in \{0, 1, 2, \ldots, m-1\},$$

dove $m$ è il numero di scale da considerare e $\Delta s$ l'incremento di $\sigma$ e $\lambda$ tra due scale consecutive.

Ne risulta un banco di $m \cdot n$ filtri che si può utilizzare per calcolare i valori del descrittore (Figura 2.3). Ogni filtro è usato per calcolare la convoluzione dell'immagine e il valor medio e varianza del risultato costituiscono 2 valori del descrittore, che in totale avrà $2mn$ valori, o bin. Il descrittore così ottenuto è stato denominato *Gabor*.

I valori dei bin sono indipendenti fra di loro pertanto l'uso di una norma $L_p$ come distanza risulta sufficiente, ma per un confronto più accurato è possibile utilizzare dei pesi $w_i$ per dare maggiore o minore importanza a singoli bin:

$$d(\boldsymbol{H}_1, \boldsymbol{H}_2, \boldsymbol{w}) = \left( \sum_{i=0}^{N-1} w_i \left| \boldsymbol{H}_1(i) - \boldsymbol{H}_2(i) \right|^p \right)^{\frac{1}{p}}.$$

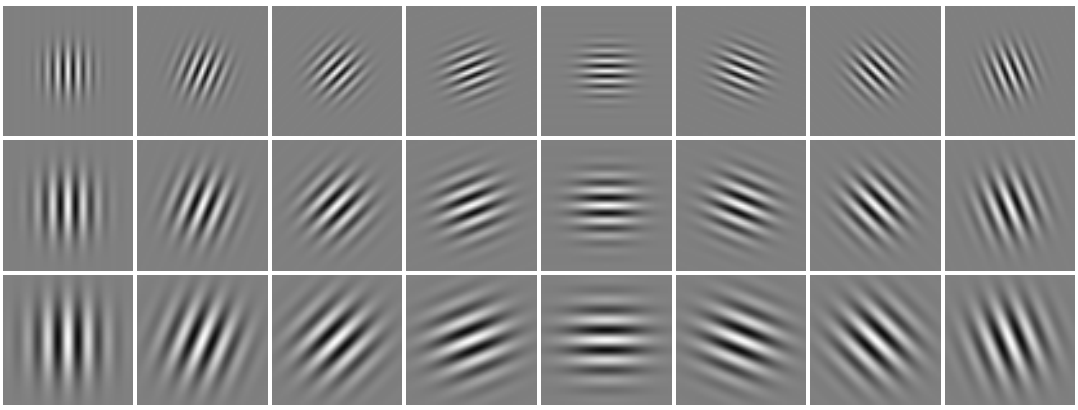


**Figura 2.3:** Esempio di banco di filtri con $n = 8$ orientazioni e $m = 3$ scale. Per fini di visualizzazione il valore $-1$ corrisponde al nero e 1 al bianco.

Nel caso si imponga $w_i = 1, \forall i$ ne risulta la normale norma $L_p$. In questo lavoro si è scelto $p = 2$ ottenendo come funzione di confronto tra i descrittori la distanza euclidea pesata.

#### 2.3.2.1 Combinazione

Il passo di combinazione di più descrittori risulta fondamentale anche per la definizione dei pesi da utilizzare per il calcolo della distanza. Sulla base di tecniche utilizzate per *content based image retrieval* (CBIR), il descrittore risultante è la media dei descrittori fatta a livello di bin, mentre la varianza sarà la base per la definizione dei pesi.[7] In CBIR i pesi sono normalmente calcolati come

$$w_i = \frac{1}{\sigma_i^2},$$

tuttavia viste le poche immagini usate per il training, si vuole evitare che si presenti il caso di $\sigma_i^2 = 0$ che porterebbe a calcolare $w_i = +\infty$. I pesi sono perciò calcolati trasformando i possibili valori di $\sigma_i^2$ nell'intervallo $[0, 1]$ con 1 assegnato a $\sigma_i^2$ minimo e 0 al massimo.

### 2.3.3 DCT

La trasformata discreta del coseno (DCT) permette di esprimere una funzione o segnale come somma di sinusoidi di diversa ampiezza e frequenza, utilizzando un numero limitato di valori della funzione presi a intervalli specifici. Questo permette di ottenere una compressione e approssimazione del segnale originale, inizialmente pensata per comprimere immagini.[2]

Utilizzando $N$ campioni $x_0, \ldots, x_{N-1}$ della funzione a intervalli regolari, la funzione viene rappresentata da altrettanti valori $X_0, \ldots, X_{N-1}$. Nel caso di un segnale 1D

questi termini vengono calcolati come

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \qquad k = 0, \ldots, N-1$$

e la formula può essere generalizzata a segnali 2D come immagini in scala di grigi applicando la DCT 1D a ogni riga e quindi a ogni colonna di pixel:

$$\begin{aligned} X_{k_1,k_2} &= \sum_{n_1=0}^{N_1-1}\left(\sum_{n_2=0}^{N_2-1} x_{n_1,n_2}\cos\left[\frac{\pi}{N_2}\left(n_2+\frac{1}{2}\right)k_2\right]\right)\cos\left[\frac{\pi}{N_1}\left(n_1+\frac{1}{2}\right)k_1\right] \\ &= \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x_{n_1,n_2}\cos\left[\frac{\pi}{N_1}\left(n_1+\frac{1}{2}\right)k_1\right]\cos\left[\frac{\pi}{N_2}\left(n_2+\frac{1}{2}\right)k_2\right]. \end{aligned}$$

Utilizzando tutti i pixel dell'immagine come input, i campioni ottenuti in output saranno lo stesso numero, tuttavia per semplificare il confronto del descrittore ottenuto da questi valori è preferibile che la dimensione sia indipendente da quella dell'immagine di partenza. Poiché la maggior quantità di informazioni è portata dalle basse frequenze,[4] vengono scelti come valori da inserire nel descrittore i primi 10 coefficienti della prima riga, prima colonna e opzionalmente della diagonale,[5] l'origine (appartenente a tutti e tre i gruppi) viene però considerata una sola volta. I descrittori sono denominati *DCT-VHD* e *DCT-VH* a seconda che venga considerata o meno la diagonale.

Come nel caso di *Gabor*, i bin del descrittore risultano essere indipendenti pertanto una norma $L_p$ è sufficiente come funzione di confronto, basandosi su quanto presentato in [5] viene perciò scelta la distanza di Manhattan pesata, ottenuta con $p = 1$:

$$d(\boldsymbol{H}_1, \boldsymbol{H}_2, \boldsymbol{w}) = \sum_{i=0}^{N-1} w_i \left|\boldsymbol{H}_1(i) - \boldsymbol{H}_2(i)\right|,$$

i pesi sono calcolati a seguito della combinazione di più descrittori come fatto per *Gabor*.

### 2.3.4 Distanza di Mahalanobis

Diversamente dalle già presentate norme $L_p$, la distanza di Mahalanobis[27,11] è una distanza quadratica e come tale tiene conto di tutte le possibili correlazioni tra i diversi bin dei possibili descrittori a cui viene applicata. Questa funzione trova applicazione nel misurare la distanza di un oggetto da una distribuzione di oggetti simili, pertanto trova una perfetta applicazione nel confrontare il descrittore di una possibile ROI con quello del modello ottenuto dalla combinazione di più descrittori.

L'insieme dei descrittori è rappresentato dal centroide, mentre le covarianze tra diversi bin vengono usate per dare una stima delle correlazioni tra questi. Una volta raccolte tutte le covarianze nella matrice simmetrica $N \times N$ delle covarianze $\boldsymbol{\Sigma}$, la distanza è calcolata come

$$d(\boldsymbol{H}_1, \boldsymbol{H}_2, \boldsymbol{\Sigma}) = \sqrt{(\boldsymbol{H}_1 - \boldsymbol{H}_2)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{H}_1 - \boldsymbol{H}_2)}.$$

È possibile utilizzare questa distanza quando è presente correlazione tra diversi bin degli istogrammi oppure per verificare se questa è presente.

## 2.4 Calcoli geometrici

### 2.4.1 Sovrapposizione di ROI

Il locatore ha necessità di valutare l'allineamento e sovrapposizione delle ROI delle immagini. Questo può essere espresso attraverso il concetto di IoU (Intersection over Union), applicabile a due (o più) ROI. Con la sua definizione, l'IoU è applicabile a qualunque forma di ROI ma SAFFIRE utilizza ROI rettangolari.

L'IoU è il rapporto tra l'area dell'intersezione e quella dell'unione di tutte le ROI considerate. Sebbene gli strumenti utilizzati per l'implementazione del locatore offrano metodi per il calcolo dell'intersezione e della sua area, occorre individuare un algoritmo che possa calcolare l'area dell'unione di più ROI in maniera efficiente.

Nel caso di due sole ROI il calcolo dell'IoU risulta semplificato è può essere fatto come

$$\text{IoU} = \frac{\mathcal{A}_\cap}{\mathcal{A}_\cup} = \frac{\mathcal{A}_1 + \mathcal{A}_2 - \mathcal{A}_\cap}{\mathcal{A}_\cup},$$

dove $\mathcal{A}_j$ è l'area di una ROI, $\mathcal{A}_\cap$ l'area dell'intersezione e $\mathcal{A}_\cup$ quella dell'unione. Si ha perciò $\text{IoU} \in [0, 1]$, dove 1 corrisponde a una perfetta sovrapposizione e le ROI coincidono mentre 0 indica che le ROI sono completamente separate, cioè l'intersezione è nulla.

La soluzione al calcolo dell'area di intersezione ed unione di più rettangoli, anche ruotati, rappresentanti le ROI è stata individuata nella libreria Clipper[1] che offre il calcolo di quattro operazioni tra coppie di poligoni arbitrari: AND, OR, NOT e XOR cioè intersezione, unione, differenza e differenza simmetrica.

Questa libreria implementa ed estende l'algoritmo illustrato in [37] utilizzando anche tecniche presentate in [1] e [10].

### 2.4.2 RANSAC

*Random sample consensus* (RANSAC)[15] è un algoritmo iterativo per la stima di un modello matematico a partire da un insieme di dati che contengono sia dati coerenti con il modello cercato, detti *inliers*, sia dati incoerenti, ad esempio rumore, detti *outliers*.

L'algoritmo procede eseguendo un certo numero di iterazioni e per ognuna:

1. il numero minimo di dati per stimare il modello cercato vengono estratti casualmente dall'insieme di input e costituiscono gli inliers;

2. gli inliers vengono usati per stimare un modello;

[1] `http://www.angusj.com/delphi/clipper.php`

3. tutti gli altri dati vengono testati sul modello definendo il *consensus set*, cioè l'insieme dei dati che supportano il modello;

4. se il consensus set è più grande del migliore trovato finora viene aggiornato.

Una volta terminate le iterazioni, RANSAC restituisce il modello corrispondente al più grande consensus set. Per SAFFIRE il modello è una similitudine con 4 gradi di libertà e i dati di input sono coppie corrispondenti di feature, pertanto il modello può essere stimato utilizzando solo due coppie nel caso di key-points o una sola coppia per key-lines. Il modello ottenuto dal solo RANSAC pertanto non è molto robusto e un successivo step di raffinamento viene usato per rendere la trasformazione robusta.

#### 2.4.2.1 Ridurre le iterazioni

RANSAC è basato sull'estrazione casuale di dati per generare il modello e perciò sono necessarie molte iterazioni, 2000 in questo progetto. Questo può richiedere molto tempo ma è possibile non eseguirle tutte introducendo il concetto di *confidenza* della bontà del modello per terminare prima ma è buona pratica lasciare un numero massimo di iterazioni per garantire la terminazione.

La confidenza è espressa come probabilità $p$ che il risultato fornito da RANSAC sia accettabile e la probabilità di scegliere un inlier per la stima del modello è

$$w = \frac{\text{n° di inliers nei dati}}{\text{n° di dati di input}},$$

questo valore tuttavia non è noto a priori ma è possibile calcolarne una stima sulla base del numero $n$ di inliers necessari per stimare il modello. $w^n$ è la probabilità che tutti i punti scelti siano inliers per il modello finale quindi $1 - w^n$ è la probabilità che almeno uno sia un outlier, risultando in un modello non buono. Detto $k$ il numero di iterazioni totali, $(1 - w^n)^k$ è la probabilità che in nessuna delle iterazioni vengano selezionati tutti inliers, che deve quindi essere uguale a $1 - p$:

$$1 - p = (1 - w^n)^k \Rightarrow k = \frac{\log(1 - p)}{\log(1 - w^n)},$$

permettendo di trovare una stima per $k$ che idealmente sarà molto minore del valore massimo fissato.

#### 2.4.2.2 Raffinamento della trasformazione

Il raffinamento della trasformazione può assumere varie forme: dal metodo dei minimi quadrati a metodi di raffinamento non lineari. L'approccio seguito utilizza l'algoritmo di Levenberg–Marquardt[25,28] ricevendo in input tutti gli inliers parte del consensus set e la trasformazione ottenuta da RANSAC, nel caso di key-lines però la transformazione in input viene ricalcolata come media aritmetica delle trasformazioni per ogni singola coppia di key-lines facente parte del consensus set.

Questo algoritmo è principalmente usato per risolvere problemi di minimi quadrati partendo da una soluzione iniziale accettabile ma non ottima e ricalcolando i parametri di un modello, nel caso specifico i 6 elementi della matrice che rappresenta la trasformazione, in modo che la somma dei quadrati delle differenze sia minima. Per la stima di una trasformazione le differenze sono calcolate come le distanze tra i punti di partenza trasformati e quelli corrispondenti.

# Capitolo 3

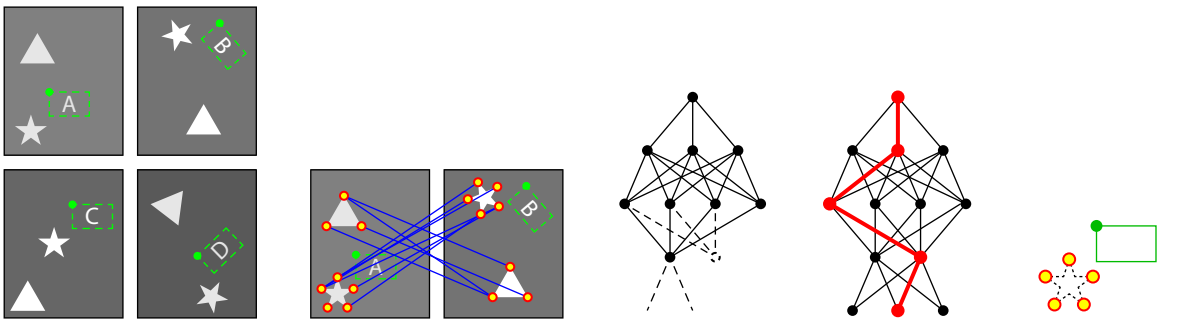
# Metodo sviluppato

In questo capitolo viene presentato il locatore alla base di questo progetto, iniziando dal punto di partenza e la struttura di base dell'algoritmo per poi passare alle modifiche proposte per migliorare il locatore sia in termini di capacità di ricerca che in velocità.

L'algoritmo risultante alla fine del processo di miglioramento si compone di diversi passaggi sia in fase di training che in ricerca. Tali passaggi sono sinteticamente illustrati rispettivamente in Figura 3.1 e Figura 3.2 e saranno illustrati in dettaglio in seguito.

## 3.1 Punto di partenza

In questa sezione viene presentata la versione iniziale del locatore prima dell'introduzione delle modifiche proposte in modo da delineare le diverse fasi del training e della ricerca. Per ogni fase vengono anche evidenziati possibili problemi che saranno affrontati nella sezione successiva.




**(a)** Input: un insieme di immagini con indicata la ROI.

**(b)** Estrazione delle feature e match.

**(c)** Valutazione di possibili modelli e costruzione del grafo.

**(d)** Ottimizzazione del grafo.

**(e)** Output: modello delle feature e ROI.

**Figura 3.1:** Passaggi dell'algoritmo durante il training.

**(a)** Input: un'immagine di test.

**(b)** Ricerca del modello nell'immagine.

**(c)** Filtro dei candidati usando il contenuto della ROI.

**(d)** Output: la ROI sulla nuova immagine.

**Figura 3.2:** Passaggi dell'algoritmo durante la ricerca.

### 3.1.1 Costruzione del modello

La fase di training richiede come input un dataset di immagini consistenti, ognuna corredata dalla ROI desiderata dove anche l'orientazione è rilevante (Figura 3.1a). Sulla base di queste immagini l'algoritmo procede come segue:

1. le feature vengono estratte da tutte le immagini e individuati i match con un'immagine di riferimento (Figura 3.1b);
2. i possibili match vengono valutati e aggregati per costruire un grafo (Figura 3.1c);
3. il grafo viene esplorato per individuare quali feature si ripropongono in tutte le immagini rimanendo consistenti con la posizione della ROI (Figura 3.1d).

Il risultato di questi tre passaggi è un sottoinsieme di feature dell'immagine di riferimento che costituiscono il modello da ricercare e la ROI corrispondente sempre sulla stessa immagine (Figura 3.1e).

#### 3.1.1.1 Estrazione delle feature e match

La costruzione del modello inizia scegliendo un'immagine di riferimento. Per semplicità, l'immagine di riferimento sarà la prima immagine del training set. Successivamente le stesse feature sono estratte da tutte le immagini di training, riferimento inclusa, e vengono individuati i match tra le feature dell'immagine di riferimento e ogni altra immagine di training cercando le feature più simili ad ogni feature dell'immagine di riferimento utilizzando una ricerca $k$NN con $k = 3$.

Le feature usate inizialmente sono segmenti estratti usando EDL[3] e LBD[39] come descrittore.

I match così trovati vengono filtrati individuando la distanza massima e minima fra tutti i match. Sulla base di queste distanze, vengono scartati tutti i match la cui distanza è superiore al seguente valore:

$$\text{minima} + 0.7 \cdot (\text{massima} - \text{minima}). \tag{3.1}$$

#### 3.1.1.2 Costruzione del grafo

Il grafo utilizzato per individuare il modello si compone di tanti livelli quante sono le immagini ricevute in input. Ogni livello contiene diversi nodi costruiti sulla base dei match individuati nella fase precedente, fa eccezione il primo livello che contiene un singolo nodo detto *radice* del grafo.

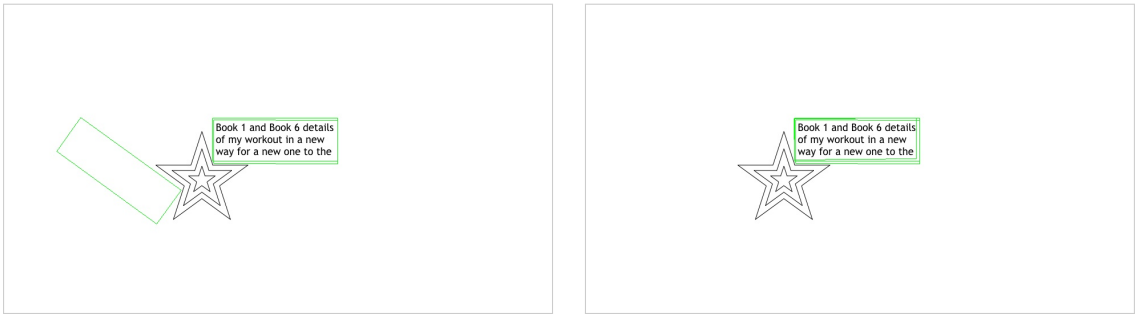
Per i livelli successivi al primo, ognuno dei match per l'immagine associata viene usato per calcolare una similitudine dall'immagine di riferimento a quella corrente considerando solamente la traslazione, rotazione e scala necessarie per allineare perfettamente il segmento alla base del match.

La trasformazione così ottenuta è poi utilizzata per prima cosa per calcolare l'IoU tra la ROI dell'immagine di riferimento trasformata e quella dell'immagine corrente; se è inferiore a una certa soglia, detta $t_m$, il match viene scartato. Come mostrato in Figura 3.3, questo secondo livello di filtraggio risulta importante nell'eliminare quei match tra segmenti simili che però non sono rilevanti per allineare le ROI, cosa che può avvenire quando il pattern desiderato risulta essere simmetrico o coincidente a seguito di una rotazione.

Il secondo passaggio consiste nell'individuare quali segmenti dell'immagine di riferimento sono sovrapposti a segmenti nell'immagine considerata dopo la trasformazione. L'allineamento è valutato controllando che gli estremi e il punto medio siano sovrapposti (entro una certa distanza massima) ai corrispondenti punti di un altro. Se il numero di sovrapposizioni supera una certa soglia, il match è considerato valido e l'insieme dei segmenti del modello che hanno sovrapposizioni e la trasformazione vengono conservati.

Con queste informazioni, nel livello viene inserito un nuovo nodo con associati la trasformazione appena calcolata, quali segmenti dell'immagine di riferimento hanno una sovrapposizione e a quali segmenti dell'immagine corrente questi si sovrappongono. Dopo aver ripetuto il procedimento per ogni match dell'immagine, ognuno avrà generato un proprio nodo a meno di essere stato scartato.

Il nodo radice è costruito in maniera diversa in quanto deve rappresentare la sovrap-




**(a)** Senza filtro. **(b)** Con filtro.

**Figura 3.3:** Effetti del filtro basato su $t_m$, le ROI di training riportate sull'immagine di riferimento sono rappresentate in verde.

posizione dell'immagine di riferimento su se stessa. Per motivi che emergeranno con la spiegazione dell'algoritmo usato per esplorare il grafo, gli unici dati rilevanti per questo nodo sono i segmenti sovrapposti che banalmente sono tutti i segmenti rilevati nell'immagine in quanto si può immaginare che siano sovrapposti a se stessi.

La costruzione del grafo si completa inserendo gli archi fra i livelli in modo che ogni nodo sia collegato a tutti i nodi del livello successivo (Figura 3.4). Sulla base della somiglianza a un albero, i nodi dell'ultimo livello sono detti *foglie*.

#### 3.1.1.3 Esplorazione del grafo e costruzione del modello

I segmenti che faranno parte del modello devono essere scelti in modo da massimizzare il numero di sovrapposizioni fra tutte le immagini di training. Per individuare quali segmenti soddisfano questa condizione viene impiegato l'algoritmo SumPath,[14] opportunamente modificato per massimizzare non la somma del valore dei nodi, ma il numero degli elementi dell'intersezione dei segmenti dell'immagine di riferimento assegnati ai nodi lungo il percorso. L'algoritmo presentato è pensato per alberi ma è facilmente adattabile a questo tipo di grafi e usa come punto di partenza il nodo radice precedentemente definito.

Questo algoritmo tuttavia è molto inefficiente in quanto richiede due esplorazioni esaustive del grafo, una prima per identificare la foglia di destinazione e una seconda per trovare il percorso migliore che porta a questa. Scegliere un algoritmo più efficiente per individuare il modello ed esplorare meno nodi del grafo è quindi un primo punto di miglioramento.

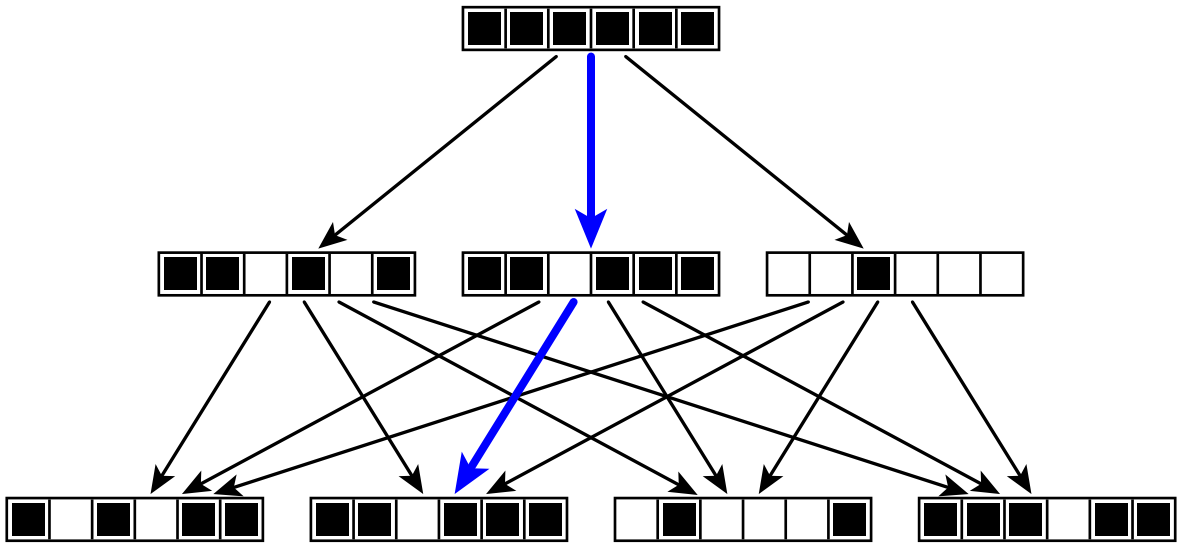


**Figura 3.4:** Un esempio del grafo usato per la costruzione del modello di un dataset con tre immagini di training, la seconda con 3 possibili sovrapposizioni e la terza con 4. Ogni nodo è rappresentato come la lista delle feature rilevate nell'immagine di riferimento, in nero quelle sovrapposta e in bianco quelle non sovrapposte. Il percorso migliore è rappresentato in blu.

Una volta trovato il percorso migliore nel grafo, le feature che costituiscono il modello sono state individuate come quei segmenti appartenenti all'intersezione dei segmenti del modello assegnati ai nodi lungo il cammino. Allo stesso modo si hanno le trasformazioni tra l'immagine di riferimento e le altre immagini di training che permettono di individuare la posizione del modello in queste ultime. Utilizzando queste trasformazioni è possibile riportare la posizione di tutte le ROI sull'immagine di riferimento e quindi costruire la ROI del modello. Questo deve essere fatto mantenendo l'informazione di rotazione delle singole ROI e al contempo costruire una singola ROI che modelli tutte quelle rappresentate lungo il percorso nel grafo. Per fare ciò occorre combinare opportunamente i centri, orientazioni e dimensioni di ogni ROI.

Il centro della ROI del modello viene calcolato come il centroide dei centri delle ROI, in modo analogo l'orientazione è la media delle orientazioni delle ROI, tuttavia essendo gli angoli delle quantità circolari occorre prestare particolare attenzione, la media di 5° e 355° infatti è 0° e non 180°, il risultato della media aritmetica. Il calcolo della media di quantità circolari avviene calcolando la media dei seni e dei coseni e quindi l'arcotangente:[29]

$$\bar{\theta} = \operatorname{atan2}\left(\frac{1}{n}\sum_{i=1}^{n}\sin\theta_i, \frac{1}{n}\sum_{i=1}^{n}\cos\theta_i\right).$$

Per calcolare la dimensione finale della ROI, tutte le ROI da combinare vengono allineate sullo stesso centro e ruotate di un angolo pari a $-\bar{\theta}$ in modo che siano (idealmente) dei rettangoli non ruotati, quindi viene calcolato il rettangolo non ruotato di dimensioni minime che contiene tutti i vertici delle ROI al suo interno, le sue dimensioni sono assegnate alla ROI del modello.

Oltre all'eccessiva complessità computazionale, questo approccio presenta un altro problema che deriva dai criteri di scelta del percorso sul grafo. Poiché si cerca di massimizzare il numero di feature sovrapposte fra tutte le possibili trasformazioni, se sono presenti pattern di disturbo con un numero maggiore di feature, è possibile che l'algoritmo scelga queste ultime come modello, quando invece sarebbero dovute essere ignorate.

### 3.1.2 Ricerca

La fase di ricerca richiede come input una singola immagine consistente con il dataset utilizzato per il training in modo che siano presenti quegli elementi le cui feature fanno parte del modello. La fase di ricerca iniziale si compone solo della ricerca del modello (Figura 3.2b) in quanto un solo candidato viene restituito che è quindi anche l'output, la fase di scelta (Figura 3.2c) è aggiunta solo in seguito.

La ricerca del candidato procede in maniera analoga al processo di costruzione del grafo utilizzato in training. Il processo comincia con il calcolo dei 3NN per le feature del modello ricercando tra quelle rilevate nell'immagine in input. Il passo successivo considera ogni singolo match e calcola per ciascuno come il modello si andrebbe a sovrapporre alle feature, il match con il maggior numero di sovrapposizioni è il candidato

cercato e la trasformazione corrispondente applicata alla ROI del modello per ottenere la ROI in output cercata.

Nonostante la semplicità di questo metodo, denominato *PerfectAlignment*, e i discreti risultati che offre, la ricerca non è in grado di tenere conto di alcune problematiche che possono emergere. Un primo problema è che la trasformazione è calcolata solo sulla base di un singolo match e pertanto non molto robusta.

Una seconda, e più importante, difficoltà emerge nel caso di un modello simmetrico o che si ripete dopo un certo angolo di rotazione. Questo può portare a posizionare correttamente il modello ma a sbagliare la sua orientazione di un valore variabile a seconda della simmetria del modello e quindi collocare in maniera errata la ROI.

## 3.2 Proposte per migliorare l'accuratezza

In questa e nella prossima sezione sono presentati tutti i miglioramenti proposti per migliorare le capacità di ricerca e di training del locatore. Sono introdotti inizialmente i miglioramenti all'accuratezza e successivamente quelli che riguardano la velocità di esecuzione.

### 3.2.1 Semplificare il grafo di training

Nell'ambito di questo lavoro tutte le immagini saranno di oggetti simili che si spostano su un piano parallelo alla camera come illustrato in Figura 1.1, è pertanto ragionevole ignorare trasformazioni prospettiche e assumere che le trasformazioni necessarie per individuare il modello siano solo traslazioni e rotazioni.

Per evitare di introdurre troppe limitazioni, una scala deve comunque essere considerata ma in maniera limitata. Si è pertanto deciso di aggiungere un ulteriore filtro ai match individuati basato sul fattore di scala che permetterebbe una perfetta sovrapposizione delle feature.

Il fattore di scala ideale è $s = 1$, perciò tutti i match con $s \notin [0.85, 1.15]$ vengono scartati. Questo ha l'effetto di semplificare il grafo costruito per il training e di ridurre i match individuati durante la ricerca.

### 3.2.2 Migliorare la costruzione del modello

Usando come base il metodo di ricerca iniziale, l'applicazione di A* per la ricerca del modello usa gli stessi vincoli, trasformati però dalla massimizzazione del numero di segmenti sovrapposti nella minimizzazione di segmenti non sovrapposti, questo perché A* cerca di raggiungere l'obiettivo con il minimo costo di cammino. Il costo per percorrere un arco del grafo perciò non è costante ma dipende dal cammino pregresso, infatti occorre considerare le sovrapposizioni dei precedenti nodi per cui si è passati e valutare come l'aggiunta del nuovo nodo riduce (o idealmente lascia inalterato) il numero di sovrapposizioni totali.

Una volta definito il costo di cammino, l'euristica deve essere costruita in modo da dare una stima del costo rimanente, tenendo conto dei vincoli che deve rispettare. Inoltre, per come il grafo è costruito (si veda Figura 3.4) un qualunque nodo dell'ultimo livello è considerato come obiettivo, pertanto questi nodi avranno euristica pari a 0. Per gli altri nodi la stima ottimistica del costo di cammino viene fatta ignorando tutti i livelli intermedi, valutando come cambia il numero di sovrapposizioni se il prossimo nodo aggiunto è una delle *foglie* del grafo, cioè un nodo dell'ultimo livello: questo viene fatto per tutte le foglie e l'euristica minore viene assegnata al nodo.

In relazione al fatto che le sovrapposizioni del costo di cammino saranno poi usate per costruire il modello da ricercare in nuove immagini e che i match di questo modello saranno usati per stimare una trasformazione, risulta inutile analizzare percorsi in cui il numero di sovrapposizioni è troppo ridotto. Per questo motivo un nuovo nodo appena generato sarà aggiunto alla frontiera se ha almeno 3 sovrapposizioni, altrimenti verrà scartato.

#### 3.2.2.1 Parallelizzazione

A* è un algoritmo facilmente parallelizzabile, tuttavia i calcoli aggiuntivi per sincronizzare i diversi thread paralleli fanno sì che non ci sia un guadagno in termini temporali, spesso si ha infatti una regressione. Per ottenere una riduzione dei tempi è necessario quindi sacrificare la correttezza dell'algoritmo ottenendo un percorso sub-ottimo.[16]

Questo è dovuto al fatto che in applicazioni comuni le varie fasi dell'algoritmo sono molto rapide, per quanto debbano essere ripetute molte volte. Per SAFFIRE tuttavia l'espansione di un nodo, in particolare l'aggiunta dei nuovi nodi alla frontiera e il calcolo della loro euristica, risulta particolarmente costosa in quanto richiede di valutare tutti i possibili nodi dell'ultimo livello del grafo sulla base dell'euristica scelta.

In questa situazione la parallelizzazione di A* può quindi portare a riduzione dei tempi senza sacrificare la correttezza. Questo è stato portato a termine lasciando invariata la struttura sequenziale dell'algoritmo tranne la fase di espansione del nodo estratto dalla frontiera.

Il thread principale si occupa di suddividere i nuovi nodi da aggiungere alla frontiera in parti più o meno uguali tra i thread, uno per core della CPU in uso. Ognuno di questi thread può ora procedere indipendentemente e calcolare costo di cammino ed euristica per ognuno dei nodi a lui assegnati prima di aggiungerli nuovamente alla frontiera.

Per ridurre ulteriormente il costo della sincronizzazione la frontiera non è un'unica lista, ma tante liste quanti sono i thread dedicati all'espansione dei nodi. Ogni thread può perciò inserire i suoi nodi nella frontiera a lui assegnata, richiedendo il solo calcolo aggiuntivo di valutare il primo nodo di ogni frontiera per decidere quale espandere all'interazione successiva. La condizione di fallimento dovuta a una frontiera vuota si traduce banalmente in avere tutte le frontiere vuote.

Questa è solo l'ultima fase del processo di training, ma anche le due precedenti possono essere parallelizzate. La prima fase richiede di calcolare le feature in ognuna delle immagini di training, che può essere parallelizzata facilmente assegnando ogni immagine a un thread diverso.

La seconda consiste nella costruzione del grafo che sarà poi esplorato e siccome ogni livello è costruito basandosi solo sulla corrispondente immagine di training e quella di riferimento, la parallelizzazione può avvenire in maniera analoga assegnando ogni livello a un thread dedicato.

### 3.2.3 Migliorare la resistenza a elementi di disturbo

Il metodo presentato fino ad ora si concentra solo su un aspetto del modello, le feature da ricercare nelle immagini. Queste tuttavia rappresentano solamente l'origine da usare come riferimento per posizionare la ROI sull'immagine, la parte realmente importante. Emerge dunque una nuova possibilità per valutare come scegliere i nodi del grafo e quindi quali feature inserire nel modello.

L'idea alla base di questo approccio migliorato è che il modello migliore è quello che permette la migliore sovrapposizione delle ROI di tutte le immagini sul cammino considerato, un fattore che può essere valutato in termini di IoU. Il valore dell'IoU appartiene all'intervallo $[0, 1]$ con il valore migliore, corrispondente a una perfetta sovrapposizione, pari a 1.

A* tuttavia lavora in termini di minimizzazione perciò occorre trasformare opportunamente il valore. Il costo di cammino può essere espresso semplicemente come

$$1 - \text{IoU},$$

l'euristica invece, in maniera analoga per le sole feature, valuta di quanto calerebbe l'IoU delle ROI correnti aggiungendo solamente la foglia che dà il calo minore.

Per ottenere la formulazione finale per $g(x)$ e $h'(x)$ è stata scelta una combinazione lineare dei costi ed euristiche basati sul numero di sovrapposizioni e IoU. Complessivamente come euristica per un nodo viene scelto il valore minimo, fra tutte le foglie, della combinazione lineare. Con questo approccio però emerge un problema dovuto al fatto che mentre la parte basata su IoU appartiene a $[0, 1]$, quella delle feature è un numero intero, per cui per una più facile gestione anche quest'ultima deve essere normalizzata e il valore massimo (usato per dividere i valori fino ad ora considerati) viene individuato nel numero di feature rilevante nell'immagine di riferimento, cioè la cardinalità massima degli insiemi delle sovrapposizioni.

Analogamente a quanto fatto per il numero di sovrapposizioni, se l'IoU di un percorso è troppo basso viene considerato non valido. Questo permette di evitare di lavorare su percorsi con IoU sotto la soglia del 65 % in quanto vengono scartati non appena sono rilevati.

La funzione risultante usata per esprimere il costo di cammino è dunque

$$\alpha \left( 1 - \frac{\text{n° sovrapposizioni}}{\text{n° feature di riferimento}} \right) + \beta \left( 1 - \text{IoU} \right). \tag{3.2}$$

Con questa formulazione si presentano due problemi: la scelta dei parametri $\alpha$ e $\beta$ e il fatto che più il peso dell'IoU aumenta più è possibile che il modello scelto non abbia il numero massimo di sovrapposizioni.

Per risolvere il secondo problema si è deciso di non terminare l'esecuzione di A* una volta trovato il percorso migliore, ma di proseguire a trovare i successivi fintanto che non ne viene restituito uno il cui IoU è inferiore a una percentuale, detta $t_{A^*}$, dell'IoU del percorso migliore. Tra i percorsi così accumulati viene scelto quello con il maggior numero di sovrapposizioni e il modello corrispondente restituito.

Il primo problema può essere risolto solamente in via sperimentale testando differenti combinazioni di $\alpha$ e $\beta$. Si può però notare che il denominatore che compare al primo termine è strettamente collegato al tipo di feature utilizzate e di immagini considerate, presentando un'alta variabilità. Da questa considerazione emerge la possibilità di non considerare questo termine, cioè impostare $\alpha = 0$ e $\beta = 1$, per evitare di impostare un parametro sulla base di valori molto variabili e non prevedibili in un contesto generale come quello in cui questo lavoro si colloca.

#### 3.2.3.1 Un migliore allineamento

Il semplice IoU appena presentato può andare bene se occorre individuare solo l'area della ROI, quando però anche l'orientazione è importante non è sufficiente per valutare l'allineamento di due o più ROI, occorre dunque definire un *IoU orientato* (oIoU).

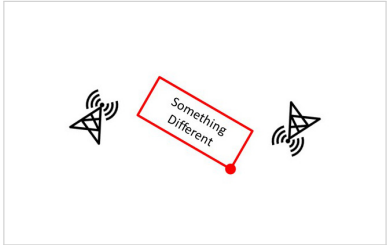
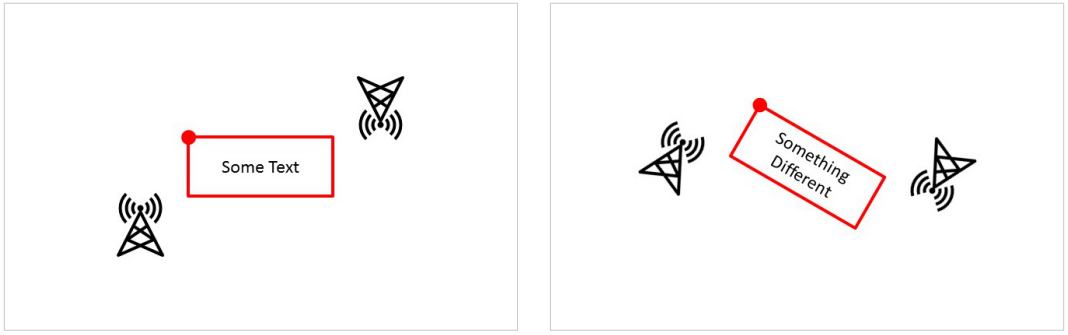



**Figura 3.5:** Ricerca fallita del pattern per il dataset in Figura 3.6.

L'uso di questa nuova tecnica risulta importante per la fase di training basata su A* nel caso in cui siano presenti dei pattern simmetrici che possono causare problemi. Considerando il dataset in Figura 3.6, il pattern simmetrico può essere individuato in due modi sull'immagine di training: correttamente oppure ruotato di 180° (Figura 3.5). In entrambi i casi l'IoU risulterebbe pari a 1 e perciò entrambi risultano nodi validi per far parte di un percorso significativo per A*, tuttavia il secondo caso è sbagliato e deve essere scartato: la definizione proposta per l'oIoU fa ciò ottenendo un valore di oIoU = 0 che poi A* scarterà.

Sulla base del prodotto scalare tra vettori per misurarne l'allineamento, si è scelto di utilizzare il coseno del minor angolo compreso tra le direzioni che danno l'orientazione di due ROI per dare una rappresentazione di quanto le direzioni siano simili. Estendendo il concetto a più di due ROI, la prima viene presa come riferimento e viene calcolato l'angolo $\Delta\theta_i$ tra questa e ogni altra ROI, infine viene calcolata la media dei coseni di tutti questi angoli. Per combinare la misura della sovrapposizione data dall'IoU tradizionale con la misura dell'allineamento dell'orientazione è preferibile avere entrambi questi valori nello stesso intervallo $[0, 1]$ pertanto la media dei coseni viene normalizzata e le due misure moltiplicate fra di loro creando una singola misura




**(a)** Immagine di riferimento. **(b)** Immagine di training.

**Figura 3.6:** Dataset per la verifica dell'oIoU, l'angolo in evidenza della ROI rappresenta il vertice in alto a sinistra.

denominata *IoU orientato* (oIoU):

$$\text{oIoU} = \text{IoU} \cdot \frac{1}{n-1} \sum_{i=1}^{n-1} \frac{1 + \cos \Delta\theta_i}{2}.$$

L'oIoU così definito viene sostituito direttamente all'IoU nell'Equazione 3.2 per il calcolo del costo di cammino e dell'euristica, tuttavia per garantire l'ottimalità di A* non può essere utilizzato così com'è. Questo accade perché l'utilizzo della media dei coseni potrebbe portare l'oIoU ad aumentare nel caso l'aggiunta di una nuova ROI porti la media ad avvicinarsi ad 1 e quindi il costo di cammino cala quando dovrebbe solo aumentare o, idealmente, rimanere invariato.

Per evitare questo problema occorre rendere la media una funzione monotona non crescente all'aumentare del numero di ROI. Questo è realizzando ipotizzando che nei livelli del grafo non ancora considerati per il percorso in analisi si verifichi il caso ideale di un allineamento perfetto con $\Delta\theta_i = 0 \Rightarrow \cos \Delta\theta_i = 1$, in questo modo quando si sostituisce l'angolo reale il valore del coseno calerà e così pure la media.

Considerando ad esempio un grafo composto da radice e 5 livelli con un possibile percorso che termina in un nodo del secondo livello, per il calcolo del costo di cammino pari a $1 - \text{oIoU}$ si avranno $n = 6$ ROI di cui solo 3 reali, pertanto l'oIoU sarà calcolato come

$$\text{oIoU} = \text{IoU} \cdot \frac{1}{5} \left( \sum_{i=1}^{2} \frac{1 + \cos \Delta\theta_i}{2} + \sum_{i=3}^{5} 1 \right),$$

dove il termine finale evidenziato è pari a 3, cioè il numero di livelli, il terzo, quarto e quinto, non considerati nel percorso. Per il calcolo dell'euristica si procede in maniera analoga considerando che viene aggiunta anche una ROI dall'ultimo livello del grafo:

$$\text{oIoU} = \text{IoU} \cdot \frac{1}{5} \left( \sum_{i=1}^{2} \frac{1 + \cos \Delta\theta_i}{2} + \sum_{i=3}^{4} 1 + \frac{1 + \cos \Delta\theta_5}{2} \right).$$

Questa definizione mantiene la proprietà che a due ROI perfettamente sovrapposte corrisponde oIoU = 1, tuttavia questo è vero solo se anche l'orientazione è la stessa. Nel caso del solo IoU se una delle due ROI subisce una rotazione di 180° il valore rimane invariato pari a 1, questo non è desiderato in quanto sebbene la posizione sia corretta il contenuto della ROI una volta isolata sarà diverso. Usando l'oIoU invece risulterà un valore pari a 0 che indica un pessimo allineamento come desiderato, equivalente a ROI non sovrapposte.

Questo nuovo oIoU è quindi sostituito all'IoU per la scelta del locatore migliore e introdotto all'interno di A* per il calcoli dei costi di cammino e dell'euristica, l'Equazione 3.2 diventa pertanto

$$\alpha \left(1 - \frac{\text{n° sovrapposizioni}}{\text{n° feature di riferimento}}\right) + \beta \left(1 - \text{oIoU}\right).$$

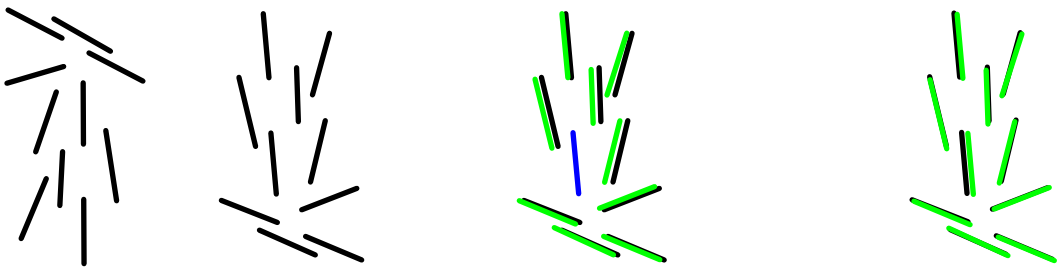
### 3.2.4 Migliorare la robustezza della trasformazione

Per migliorare la trasformazione, restituita dal metodo di ricerca iniziale, che dà la posizione della ROI è possibile tenere il metodo originale come punto di partenza ma sfruttare meglio i risultati della valutazione di come le feature si sovrappongono.

Dalla valutazione della sovrapposizione è possibile individuare gli *inliers*, cioè le feature sovrapposte e calcolare una seconda trasformazione, non basata sul perfetto allineamento di un singolo segmento, ma utilizzare tutti i punti degli inliers per stimare una similitudine robusta che sostituisca quella iniziale.

Questa stima è calcolata utilizzando RANSAC per trovare la miglior trasformazione tra le due immagini, applicata ai punti iniziali, medi e finali dei segmenti corrispondenti.

Questo metodo di ricerca è denominato *RobustSimilarity* (Figura 3.7).



**(a)** Riferimento. **(b)** Immagine di training. **(c)** Sovrapposizione del modello, in verde, basata su un solo segmento evidenziato in blu. **(d)** Sovrapposizione con stima robusta della trasformazione.

**Figura 3.7:** Rappresentazione grafica di come *RobustSimilarity* calcoli una stima robusta di una trasformazione, offrendo una sovrapposizione migliore.

### 3.2.5 Migliorare la ricerca del modello

Il metodo di ricerca iniziale offre come risultato solo una possibile posizione per la ROI, pertanto non è in grado di gestire correttamente pattern simmetrici. Per fare ciò occorre considerare più di una sola posizione e successivamente determinare quale di queste è quella che ha maggiore probabilità di essere quella corretta.

#### 3.2.5.1 Trovare possibili ROI

Per identificare possibili candidati per la ROI cercata, una possibile soluzione è l'applicazione della GHT. Come punto di riferimento $\boldsymbol{P}_C$ è stato scelto il centro della ROI del modello mentre l'AA non è implementato come una matrice densa, ma in maniera sparsa come un array di gruppi di voti.

Quando un nuovo voto deve essere aggiunto all'AA l'algoritmo esegue i seguenti passi:

1. se non ci sono gruppi di voti viene creato un nuovo gruppo contenente il voto (Figura 3.8a);
2. altrimenti vengono individuati tutti i gruppi il cui centroide ha una distanza dal voto corrente inferiore a una certa soglia:
   a) se nessun gruppo viene individuato, il voto viene inserito in un nuovo gruppo (Figura 3.8b e Figura 3.8c);
   b) altrimenti tutti i gruppi individuati vengono unificati fra di loro e il voto aggiunto al gruppo risultante (Figura 3.8d e Figura 3.8e).

Questa tecnica è ispirata all'algoritmo di clustering DBSCAN[13] ma segue un approccio semplificato in quanto non cerca quei gruppi con almeno un punto a distanza inferiore a una certa soglia, ma quelli il cui centro è entro questa distanza. Questo permette di ridurre l'occupazione di memoria in quanto si può mantenere solo il centroide e, al contempo, anche la complessità computazionale dovendo calcolare una sola distanza per gruppo e non una per ogni voto già registrato nell'AA.

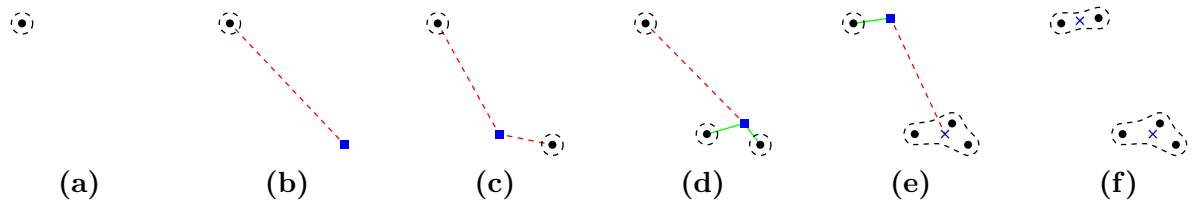



**Figura 3.8:** Rappresentazione grafica del raggruppamento dei voti della GHT aggiungendo un voto alla volta. I punti rappresentano voti già processati, il tratteggio i gruppi e le croci il loro centroide. I quadrati indicano un nuovo voto e sono collegati da un tratto rosso a gruppi troppo lontani e verde a quelli sufficientemente vicini.

Ogni gruppo di voti così ottenuto rappresenta una possibile posizione e le feature che hanno votato per un gruppo possono essere usate per stimare la trasformazione per posizionare la ROI.

#### 3.2.5.2 Scelta del risultato

Non tutte le possibili posizioni ottenute dal primo passaggio vanno considerate per evitare di considerare quei gruppi con troppi pochi voti. I gruppi di voti sono pertanto ordinati per numero di voti decrescente e solo quelli con un numero di voti pari o superiore al 40 % dei voti del massimo vengono considerati.

Si sono così trovate le possibili posizioni della ROI che vanno ora filtrate per individuare il risultato finale. Avendo effettuato la prima ricerca basandosi su pattern, il secondo passaggio si basa sul contenuto delle possibili ROI e lo confronta con quello del modello.

Questo confronto richiede innanzitutto di stabilire un descrittore per la ROI del modello basandosi sul contenuto di tutte le ROI nelle immagini di training, i cinque possibili descrittori sono $(Eq)GLH$, *Gabor* e $DCT\text{-}VH(D)$, opportunamente corredati dai pesi necessari per effettuare il confronto.

Lo stesso descrittore usato per il modello è calcolato anche per ogni possibili ROI e la sua distanza dal modello calcolata. La ROI che presenta la distanza minima viene scelta come risultato e restituita.

### 3.2.6 Migliorare la robustezza dei nodi

La trasformazione assegnata a ogni nodo del grafo è costruita utilizzando un singolo match tra feature, pertanto non è molto robusta, questa trasformazione è denominata *Single*. Si può quindi utilizzare la stessa tecnica presentata in Sezione 3.2.4 per individuare gli inliers di questa prima trasformazione e utilizzarli per stimarne una seconda con RANSAC, la trasformazione ottenuta è detta *Robust*.

### 3.2.7 Migliorare il descrittore del modello

Il modello costruito inizialmente utilizza solamente le feature e i relativi descrittori provenienti dall'immagine di riferimento, questo descrittore è detto *Simple*. Utilizzando la stessa idea alla base dei descrittori del contenuto delle ROI si è deciso di utilizzare informazioni da tutte le immagini di training costruendo ciò che è stato denominato "super-feature".

Un primo metodo, noto come *SuperDesc*, utilizza solo feature dell'immagine di riferimento, tuttavia ognuna di queste è accoppiata al suo descrittore proveniente dalla stessa immagine e un altro descrittore per ogni altra immagine di training. Un'immagine di training fornisce perciò a ogni feature del modello un descrittore aggiuntivo, il descrittore della feature che ha stabilito un match con la feature del modello secondo quanto riportato nei nodi del grafo che compongono il percorso alla base del modello stesso.

Un secondo metodo, detto *SuperFeatDesc*, utilizza non solo i descrittori come nel metodo precedente ma anche le feature dalle altre immagini di training. Nel caso di un modello con $n$ sovrapposizioni ottenuto da $m$ immagini di training saranno presenti $n \cdot m$ feature, invece che solo $n$ come nei due casi precedenti, e altrettanti descrittori. Le feature delle altre immagini non possono essere utilizzate così come sono, ma la trasformazione associata al nodo del grafo che le ha scelte viene usata per riportarle sull'immagine di riferimento correggendo quindi angolo, dimensione e posizione in modo che siano pressoché sovrapposte fra di loro.

Alla base della decisione di utilizzare anche le feature delle altre immagini si pone il metodo con cui la GHT offre la trasformazione necessaria per individuare la posizione della ROI: poiché questa trasformazione è una stima con RANSAC utilizzando coppie di punti corrispondenti, più coppie si hanno a disposizione più la trasformazione risultante è robusta.

### 3.2.8 Ridurre i match

La ricerca dei match è basata sulla ricerca dei $k$NN ma inizialmente per ogni segmento vengono tenuti i 3 migliori. Questo può permettere di individuare un match migliore per il modello nel suo complesso ma al contempo la larghezza del grafo può aumentare inutilmente, portando i tempi di training ad aumentare.

Per questo motivo si è deciso di valutare se la riduzione del parametro $k$ possa permettere di contenere i tempi di training senza intaccare la capacità di ricerca.

Un'ulteriore riduzione può essere richiesta in luce dei nuovi descrittori proposti in Sezione 3.2.7. Assegnando a ogni feature del modello più segmenti è possibile che nei match utilizzati per la ricerca la maggior parte dei punti appartengano a uno stesso segmento e i pochi rimanenti agli altri, questo farebbe sì che la stima della trasformazione con RANSAC dia maggiore importanza a quei molti punti tutti vicini fra di loro, rendendo la trasformazione poco robusta.

Il metodo iniziale che mantiene tutti i match è denominato *All*, mentre il nuovo metodo proposto *One* in quanto per ogni feature del modello mantiene un singolo match, quello con distanza minore fra tutti i match trovati per la feature considerata.

Una terza tecnica di riduzione dei match introduce il concetto di *greediness*, cioè il considerare soltanto le opzioni migliori e non tutte quelle disponibili. In questo lavoro viene introdotta sia nella costruzione del grafo sia nella ricerca della ROI andando ad agire sull'Equazione 3.1 per ridurre la distanza massima all'aumentare del fattore di greediness $g$.

La greediness è definita come un valore $g \in [0, 1]$ dove 0 corrisponde al comportamento normale e 1 alla massima riduzione della distanza. $g$ va ad agire sul fattore 0.7 abbassandolo fino a 0.2 in corrispondenza di $g = 1$, l'Equazione 3.1 è quindi modificata in

$$\text{minima} + [0.2 + (0.7 - 0.2)(1 - g)] \cdot (\text{massima} - \text{minima}).$$

### 3.2.9 Training incrementale

Nell'uso finale di SAFFIRE si prevede di non avere a disposizione tutto il dataset di training nello stesso momento ma l'utente acquisirà le immagini da usare una alla volta e il locatore dovrà essere in grado di dare una previsione della ROI anche con una sola immagine di training, questo per facilitare la fase iniziale di marcatura delle ROI nelle immagini. Le previsioni iniziali saranno non perfette se non sbagliate, ma l'aggiunta di altre immagini le porterà ad essere corrette.

Per supportare questa modalità di training, l'algoritmo deve essere rivisto in un'ottica incrementale mantenendo in memoria tutte le informazioni necessarie a individuare il modello finché non viene trovato quello finale, in particolare il grafo in quanto l'aggiunta di una nuova immagine di training può portare a cambiare il percorso necessario per minimizzare il costo di cammino. Aggiungere un nuovo nodo al percorso precedente può infatti rivelarsi non sufficiente a determinare il massimo IoU (e successivamente oIoU) tra le ROI.

L'algoritmo di training rimane pressoché invariato ma le tre fasi (estrazione delle feature, costruzione del grafo ed esplorazione) vengono ripetute ogni volta che una nuova immagine viene aggiunta. L'aggiunta della prima immagine vede in maniera banale l'estrazione delle proprie feature, la costruzione del nodo radice del grafo contenente tutte le feature appena estratte e la creazione del modello usando la ROI fornita e tutte le feature.

L'aggiunta di immagini successive procede all'espansione incrementale del grafo estraendo le feature dall'immagine e costruendo un singolo nuovo livello da aggiungere in fondo al grafo prima di eseguire da capo A* per individuare il nuovo modello. L'esecuzione di A* per ogni nuova immagine presenta il rischio di un tempo di training complessivo molto elevato, pertanto occorrono alcuni accorgimenti per limitarlo.

La riduzione del tempo avviene riducendo la complessità del grafo eliminando quei nodi considerati non più promettenti dopo aver estratto un modello. Per fare questo, dopo ogni esecuzione di A*, per prima cosa da ogni nodo vengono rimosse quelle sovrapposizioni che non fanno parte del modello appena estratto, quindi tutti i nodi che contengono meno del 50 % delle sovrapposizioni del modello vengono eliminati dal grafo. Questo processo è detto *semplificazione* del grafo.

#### 3.2.9.1 Terminazione automatica

Con l'aggiunta incrementale di immagini di training si vuole fornire un meccanismo che decida in maniera automatica quando non è più necessario aggiungere altre immagini in quanto il modello è sufficientemente buono. Le condizioni di terminazione individuate sono 3:

- la previsione della ROI è sufficientemente buona;
- il modello calcolato ha un numero troppo basso di feature;
- non è possibile calcolare un modello.

La prima condizione viene verificata prima di utilizzare la nuova immagine per trovare un nuovo modello. Per decidere se procedere il modello corrente viene utilizzato per fare una previsione della ROI, se questa previsione ha un IoU (e successivamente oIoU) sufficientemente elevato con la ROI fornita dall'utente la previsione è considerata valida e il sistema decide che non è necessario aggiungere l'immagine al dataset di training, altrimenti viene aggiunta e il modello ricalcolato. Per garantire che le feature assegnate al modello appartengano tutte a regioni interessanti delle immagini e non siano parte dello sfondo o elementi di disturbo questa condizione è valutata solamente quando al modello sono già state assegnate almeno 3 immagini, quando ciò accade il modello è detto *raffinato*.

Le altre due condizioni sono valutate dopo aver ricalcolato il modello. La seconda richiede di evitare l'aggiunta di ulteriori immagini di training quanto il modello appena calcolato contiene meno di un numero minimo di feature, impostato di poco maggiore al numero minimo per stimare una similitudine. La terza verifica che un modello valido sia stato calcolato, in caso contrario ripristina il modello precedente e richiede di non aggiungere ulteriori immagini di training.

Anche queste ultime due condizioni hanno effetto solo dopo aver aggiunto almeno tre immagini al modello, tuttavia in caso di fallimento nel calcolo del modello la terza ripristina in ogni caso quello precedente.

#### 3.2.9.2 Utilizzo di più ROI

Sulla base delle condizioni di terminazione appena introdotte alcune immagini di training non vengono usate in alcun modo per il modello. Le immagini vengono scartate perché le feature che contengono non permettono di ottenere un modello valido, tuttavia è possibile considerare solo la ROI fornita per migliorare quella del modello.

Per fare questo viene inizialmente calcolata la previsione della ROI sull'immagine e la trasformazione che riporta la ROI del modello sull'immagine viene invertita per riportare la ROI fornita dall'utente (e non quella prevista) sul modello. A questo punto la ROI appena riportata e quella del modello possono essere unificate in una sola da usare come modello.

Queste ulteriori ROI aggiunte senza passare dal grafo sono opportunamente salvate per essere considerate anche in caso che per immagini di training successive venga valutato di aggiungerle al modello.

### 3.2.10 Nuove tipologie di grafi

Gli approcci presentati fino ad ora sono basati su un grafo costruito solo sulla base di sovrapposizioni perfette di singoli coppie di segmenti in base ai match tra quelli dell'immagine di riferimento e le altre immagini di training (sia *Single* che *Robust*). I nodi così costruiti contengono una sola sovrapposizione con vicinanza sia fisica che a livello di descrittore, per tutte le altre è garantita solo la vicinanza fisica dei punti.

Un secondo svantaggio è che segmenti diversi che generano trasformazioni simili danno origine ognuno ad un proprio nodo, di conseguenza in un livello del grafo sono

presenti molti nodi simili fra di loro, sia per trasformazione sia per sovrapposizioni, che complicano inutilmente il grafo e la ricerca del modello.

Da questa prospettiva emerge come il grafo utilizzato per costruire il modello possa non essere sufficientemente robusto, di seguito è presentata una possibile alternativa per la costruzione del grafo.

#### 3.2.10.1 RANSAC

L'utilizzo di RANSAC permette di stimare quella trasformazione che ha il maggior numero di match a sostenerla, il che si traduce in un nodo con il maggior numero di sovrapposizioni, ognuna valida sia per vicinanza fisica sia per similitudine dei descrittori in quanto gli accoppiamenti vengono preservati da RANSAC.

Per garantire la scelta della trasformazione migliore, una sola esecuzione di RANSAC non è sufficiente, tuttavia se la trasformazione restituita è realmente robusta, esecuzioni successive restituiranno la stessa o una molto simile. Per ovviare a questo problema, e al contempo garantire che non ci siano nodi simili nello stesso livello del grafo, è possibili rimuovere gli inliers dell'ultima esecuzione dall'input dell'esecuzione successiva. Questo forzerà RANSAC a trovare una trasformazione e sovrapposizioni diverse che saranno poi valutate dall'esplorazione del grafo. Il grafo così costruito è denominato *gRANSAC*.

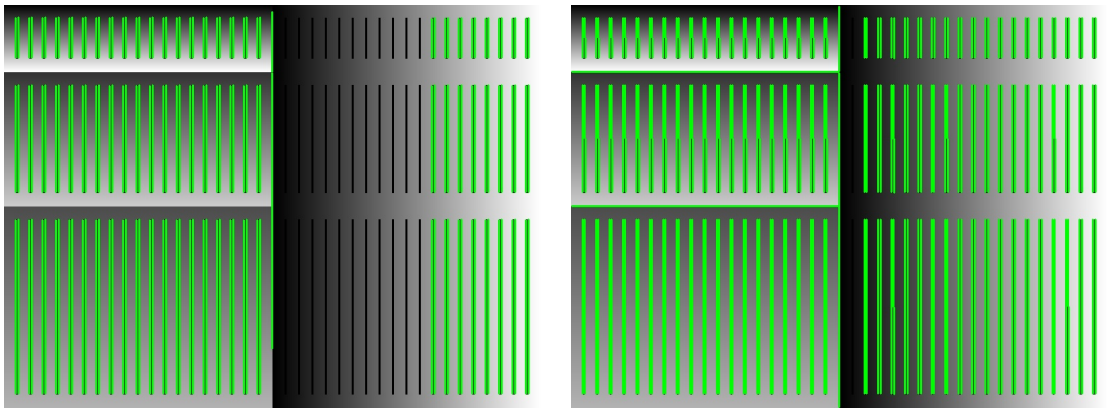
Ogni nodo così generato viene aggiunto al grafo se e solo se la trasformazione corrispondente richiede un fattore di scala $s$ sufficientemente vicino a 1, in maniera analoga a quanto fatto per il metodo originario (Sezione 3.2.1) tenendo solo quei nodi con $s \in [0.85, 1.15]$. L'iterazione di RANSAC termina quando non è più possibile trovare una trasformazione oppure il numero di inliers alla sua base è troppo basso, evitando così di generare nodi con sovrapposizioni di segmenti di disturbo provenienti ad esempio dallo sfondo.

### 3.2.11 Utilizzo di altre feature

Gli approcci visti fino a questo punto sono tutti basati sull'utilizzo di segmenti estratti da EDL, tuttavia esistono molte altre tipologie di feature.

Un altro possibile estrattore di segmenti è LSD[19] che ha il vantaggio di non richiedere alcun parametro di configurazione, un punto molto favorevole visto l'ampio campo di applicazione di questo lavoro. LSD inoltre è in grado di estrarre molti più segmenti rispetto ad EDL, soprattutto in casi di scarso contrasto come mostrato in Figura 3.9.

Abbandonando i segmenti, una delle feature più famose è SIFT[26] ma la complessità di estrazione, calcolo dei descrittori e loro confronto risulta essere eccessiva per questo lavoro. Altre feature in grado di soddisfare le condizioni necessarie sono quelle estratte da ORB[35] che avendo un descrittore binario permettono un calcolo dei match molto rapido, così come l'estrazione dei punti delle immagini che andranno a costituire le feature.



**(a)** EDL. **(b)** LSD.

**Figura 3.9:** Confronto dei segmenti, in verde, estratti in condizioni di contrasto variabile.

Per garantire che le feature assegnate al modello siano valide e supportate da match di descrittori tra le diverse immagini di training i metodi originari per la costruzione del grafo (*Single* e *Robust*) vengono abbandonati in favore del solo *gRANSAC*.

### 3.2.12 Scelta del descrittore migliore

L'ambito di utilizzo molto ampio di questo lavoro rende difficile scegliere una singola tipologia di feature che possa essere valida in ogni possibile applicazione. Per questo motivo occorre sacrificare la velocità in fase di training per determinare quale feature (LSD o ORB) risulti più adatta al caso corrente.

Questa scelta può essere fatta portando avanti in parallelo il training incrementale di un locatore basato su LSD e di uno basato su ORB, o più in generale di un numero arbitrario di locatori. Durante il training si può quindi valutare come i due modelli evolvano all'aggiunta di immagini di training e come cambino le loro capacità di ricerca, riutilizzando le immagini di training come dataset su cui effettuare la ricerca.

Un primo approccio è basato sulla continua aggiunta di immagini di training secondo il comportamento descritto in Sezione 3.2.9.1 con la differenza che una previsione sufficientemente buona non causa la fine del training per quel modello ma evita solamente l'aggiunta dell'immagine al training set, che viene però accumulata per valutazione.

Dopo ogni aggiunta i modelli vengono utilizzati per effettuare la ricerca della ROI nelle immagini accumulate e le ROI effettive ora disponibili utilizzate per calcolare l'IoU (e successivamente oIoU). Se l'IoU medio per un modello è superiore a una certa soglia e sufficienti immagini sono state aggiunte per scartare le feature di disturbo il modello è considerato valido è può essere restituito. Fra i vari modelli, quello restituito come migliore sarà il modello con IoU medio massimo tra tutti quelli validi, se nessun modello può essere ulteriormente migliorato perché il numero di feature è troppo basso o perché l'aggiunta di una nuova immagine causerebbe la perdita del modello, allora

la soglia minima sull'IoU medio viene scartata e il modello con valore massimo viene considerato migliore.

Una seconda iterazione (e quella effettivamente utilizzata) si basa sulla prima e smette di tentare di fare il training di un modello quando il training fallisce consecutivamente per un determinato numero di immagini, 2 nei test condotti.

L'aggiunta di nuove immagini al modello è ora più restrittiva: appena vengono aggiunte sufficienti immagini per filtrare le feature e l'IoU medio su tutte le immagini proposte è superiore a una data soglia, non viene più fatto il training del modello e ulteriori immagini, come in precedenza, contribuiscono solo a scegliere quale modello restituire.

## 3.3 Proposte per migliorare la velocità

### 3.3.1 Uso di immagini più piccole

Il processo di estrazione delle feature da un'immagine può risultare costoso per immagini di grandi dimensioni, segue pertanto che l'uso di immagini più piccole possa offrire una valida soluzione per ridurre i tempi sia in traning che in ricerca.

In caso non sia possibile acquisire direttamente immagini a dimensione ridotta si può sopperire utilizzando tecniche di sottocampionamento riducendo le dimensioni dell'immagine originale di un fattore intero. Questo permette di scartare dei pixel senza calcolare esplicitamente il valore dei rimanenti, ma riutilizzando parte dei pixel originali.

### 3.3.2 Greedy A*

L'esplorazione di grafi di grandi dimensioni può risultare costosa, per questo motivo all'aggiunta di una sola immagine di training si procede secondo l'ipotesi che il percorso che sarà trovato nel grafo corrisponda esattamente, a meno del nodo in più presente alla fine, al percorso precedentemente individuato.

Sulla base di questa ipotesi viene definita una versione greedy di A* che, utilizzando le stesse definizioni dell'algoritmo completo, identifica quale sia il nodo dell'ultimo livello del grafo che quando aggiunto al percorso precedente risulta nel miglior modello. L'ipotesi iniziale però non ha un fondamento, per questo motivo il risultato di questo approccio è considerato valido solamente se l'IoU (e successivamente l'oIoU) delle ROI supera una soglia $t_{gA^*}$.

### 3.3.3 Uso più efficiente della GHT

Il metodo di ricerca presentato richiede di calcolare il descrittore del contenuto della ROI per ognuna delle possibili posizioni, tuttavia se è presente una sola possibile posizione è possibile evitare questo calcolo e risparmiare così tempo. Inoltre è ragionevole

assumere che la ROI cercata (se presente) sia tra i primi gruppi di voti (quando ordinati per numero di voti), pertanto si procede limitando le posizioni considerate non solo per numero di voti minimo, ma anche anche imponendo un valore massimo alle posizioni valutate, limite fissato a 3.

Per guadagnare ulteriormente in velocità è possibile rendere più rapido il calcolo dei descrittori del contenuto delle ROI utilizzando più thread che si dividono il lavoro. A beneficiare di ciò sono $(Eq)GLH$ (anche se come si vedrà non verrà utilizzato) e *Gabor*. Nel primo caso è possibile assegnare a ogni thread il calcolo l'istogramma dei livelli di grigio per una specifica sezione della ROI e infine sommarli, per *Gabor* poiché le singole convoluzioni per calcolare due elementi del descrittore sono indipendenti è possibile suddividerle fra diversi thread ed eseguirle in maniera parallela.

### 3.3.4 Scegliere un locatore più veloce

Per avere tempi ridotti in ricerca, scegliere il locatore migliore solo sulla base dell'oIoU non offre la scelta migliore, per questo motivo il modello restituito non è più basato solamente su quello che offre le migliori capacità di ricerca, ma anche sulla velocità e il fatto che il modello sia stato raffinato. Per prima cosa viene individuato il miglior locatore, in termini di oIoU, con modello raffinato e non, se l'oIoU medio nel primo caso è sopra una soglia saranno valutati solamente quelli con modello raffinato, altrimenti possono essere considerati anche gli altri. A partire dal miglior oIoU medio identificato da $\text{oIoU}_{\max}$, tutti i locatori (con modello raffinato o meno sulla base della valutazione appena effettuata) con oIoU medio maggiore di

$$\text{oIoU}_{\max} - t_{\text{oIoU}}$$

sono considerati e quello con tempo medio di ricerca minore viene restituito come migliore.

Questo processo deriva dal fatto che sacrificare un minimo di precisione, scegliendo $t_{\text{oIoU}}$ sufficientemente piccolo, sia accettabile per guadagnare in velocità. Per garantire la maggior velocità possibile viene introdotto anche un "non-locatore" che non fa una ricerca di feature nell'immagine per posizionare la ROI, ma banalmente combina tutte le ROI proposte in training nello stesso modo usato per la costruzione del modello del locatore. Questo locatore trova applicazione in quei casi in cui ciò che va individuato ha sempre la stessa posizione e dunque si può risparmiare il tempo di estrazione di feature lasciando la ROI sempre nella stessa posizione, ma non sapendo se questo è il caso o meno la scelta tra il non-locatore e un vero locatore deve essere effettuata.

Per valutare al meglio se il non-locatore è applicabile, il concetto di "modello non raffinabile" viene modellato come ROI di training con un oIoU reciproco troppo basso e "feature di disturbo scartate" richiede semplicemente un certo numero di ROI di training. Questo permette di gestire il caso di ROI non statiche facendo sì che il coordinatore dei locatori in training parallelo lo riconosca come non raffinato e per questo lo scarti.

# Capitolo 4

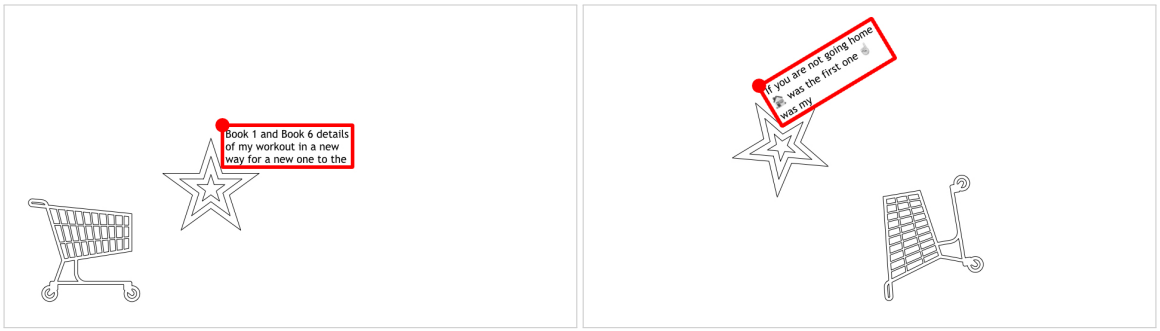
# Evaluation setup

In questo capitolo vengono presentati i dati e la metodologia utilizzati per valutare le capacità del locatore secondo i miglioramenti introdotti nel Capitolo 3.

## 4.1 Dataset

Per la valutazione dei metodi proposti sono stati usati sia dati reali sia sintetici. Questi ultimi si rendono necessari per isolare i singoli problemi individuati in modo da verificare che un singolo miglioramento si riveli efficace.


**(a)** Star.




**(b)** StarCart.

**Figura 4.1:** Dataset sintetici.

Per verificare la resistenza a **pattern simmetrici**, sia durante la creazione del modello secondo la verifica di $t_m$ sia durante la ricerca con l'uso della GHT è stato utilizzato il dataset in Figura 4.1a. Per verificare la resistenza a **pattern di disturbo** (che sopravvivono al test iniziale basato su $t_m$) che possono portare a costruire un modello errato è stato utilizzato il dataset in Figura 4.1b.

Per quanto riguarda i dati reali, sono stati utilizzati i dataset presentati in Figura 4.2 e Figura 4.3. In particolare *Lattine*, *Curry* e, successivamente, *Birra* hanno guidato l'evoluzione di SAFFIRE verso la configurazione finale. Il dataset *LattineBagnate* invece è stato usato per discriminare quale descrittore utilizzare per il contenuto delle ROI.

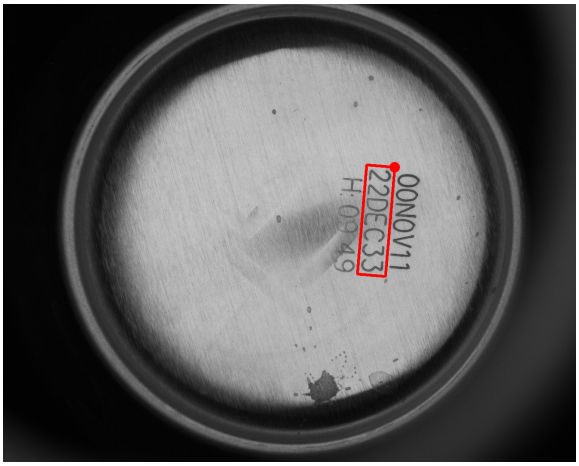
## 4.2 Metriche

I primi valori da valutare sono i tempi per il training e la ricerca. Questi tempi sono stati misurati su un processore Intel Core i5 7267U (3.1 GHz, 2 core fisici e 4 thread) come media di 5 esecuzioni. I tempi di ricerca sono la media dei tempi di ricerca fra tutte le immagini del dataset, il numero di immagini è riportato assieme ai tempi medi.

I risultati della ricerca sono presentati come distribuzione dell'IoU tra la ROI prevista e quella fornita manualmente. Per dare una rappresentazione della bontà delle ROI proposte per ogni dataset e metodo di ricerca vengono accumulati gli IoU per ogni immagine e i dati raccolti dai test condotti sono presentati sotto forma di box plot.

Un box plot è composto da un rettangolo i cui estremi rappresentano il primo $q_1$ e terzo quartile $q_3$ della distribuzione e al cui interno è indicata la mediana $q_2$. Esternamente al rettangolo sono presenti dei "baffi" che si estendono verso l'esterno del rettangolo fino al più piccolo valore con distanza massima $1.5 \cdot \text{IQR}$ dal primo quartile e fino al più grande valore con distanza massima $1.5 \cdot \text{IQR}$ dal terzo quartile, con $\text{IQR} = q_3 - q_1$. I punti che non rientrano nell'intervallo dei "baffi" sono considerati outliers e rappresentati come singoli punti.

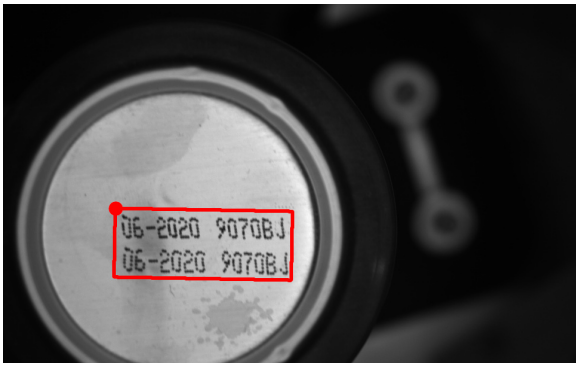
I risultati della configurazione finale, l'unica ad essere basata su oIoU e includere alcune funzionalità, sono presentati in termini di oIoU e sono eseguiti un un processore Snapdragon 660 (1.95–2.2 GHz, 8 core fisici) con 6 thread dedicati alla parallelizzazione degli algoritmi.



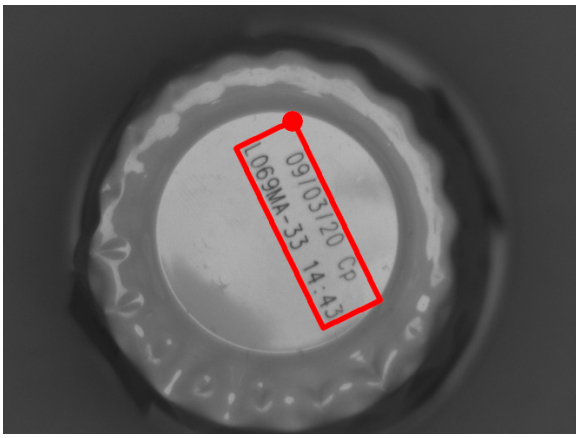

**(a)** Lattine.


**(b)** LattineBagnate.


**(c)** Curry.




**(d)** Birra.

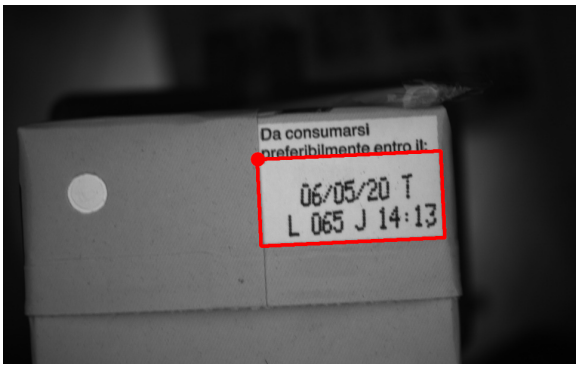



**(e)** Tea.

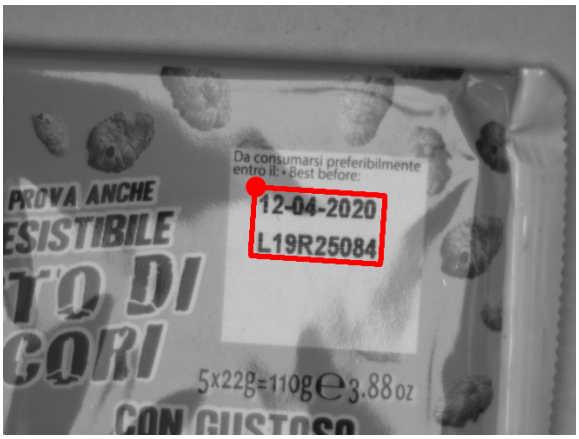

**(f)** Olio.

**Figura 4.2:** Dataset reali.

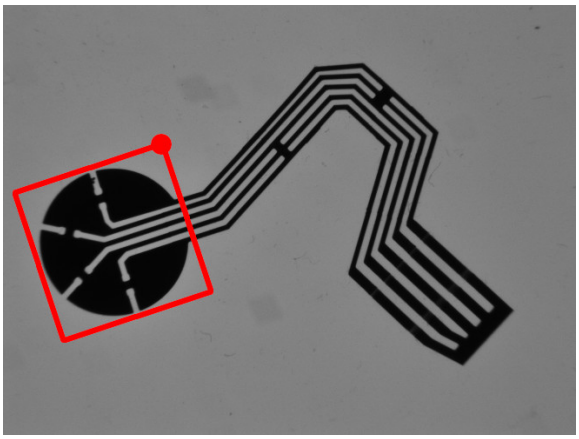

**(a)** Pasta.



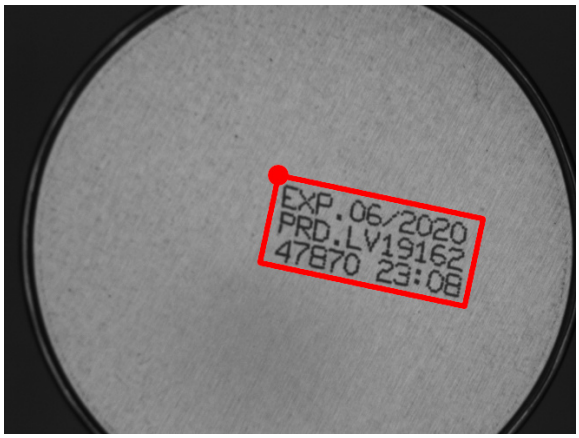

**(b)** Succhi.




**(c)** Ciocory.



**(d)** LeadFrame.




**(e)** Caprice.

**Figura 4.3:** Dataset reali.

# Capitolo 5

# Risultati

In questo capitolo sono presentati i risultati della valutazione delle capacità di SAFFIRE secondo i metodi indicati nel Capitolo 4. I risultati presentati rappresentano l'evoluzione di SAFFIRE con l'aggiunta successiva di nuove migliorie come riportato nel Capitolo 3. Alcune funzioni e l'utilizzo di oIoU fanno parte solo della configurazione finale presentata in Sezione 5.4.

## 5.1 Costruzione del modello

### 5.1.1 Costo computazionale

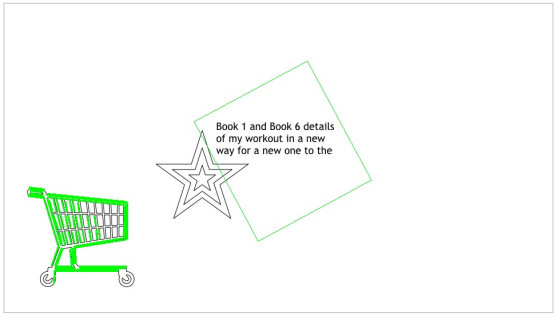
L'aggiunta dell'ulteriore controllo sulla scala durante la costruzione del grafo permette di ridurre il numero di nodi valutati dal metodo di ricerca iniziale, sulla ricerca del modello del dataset *Lattine*, da 5079 a 4187, una riduzione del 17.6 % che già contribuisce a ridurre il carico computazionale della costruzione del modello.

Per valutare se l'utilizzo di A* sia efficacie per aumentare questa riduzione si è continuato a usare il dataset *Lattine* per verificare come cambia il numero di nodi valutati tra il metodo iniziale e A* stesso.

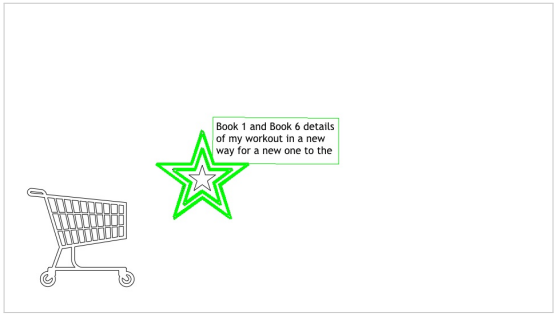
Mantenendo l'ulteriore filtro sulla costruzione del grafo, A* richiede 3392 valutazioni, una riduzione complessiva del 33.2 % che permette comunque di ottenere lo stesso modello e percorso nel grafo. Un'ulteriore riduzione dei nodi valutati è fornita anche dallo scartare quei percorsi con un numero non sufficiente di sovrapposizioni o il cui IoU è troppo basso, scendendo a solo 260, una riduzione totale del 94.9 %.

L'uso di altre euristiche non comporta un cambio significativo nel numero di nodi elaborati. La considerazione di più risultati da parte A* con diverse combinazioni di euristiche aumenta il numero di nodi valutati a valori tra 2 e 3 volte il numero di nodi considerati quando solo il percorso migliore viene usato e la riduzione di nodi valutati si attesta attorno all'85 %.

L'uso di A* come metodo di ricerca nel grafo è pertanto vincente, semplificando enormemente la costruzione del modello.




**(a)** Modello valutando solo le sovrapposizioni.




**(b)** Modello corretto considerando anche l'IoU delle ROI.

**Figura 5.1:** Effetti dell'aggiunta della seconda euristica basata su IoU sul dataset StarCart.

### 5.1.2 Pattern di disturbo ed euristiche

Nel caso del dataset StarCart il metodo iniziale basato solo sulle sovrapposizioni restituisce, come mostrato in Figura 5.1, un modello sbagliato che contiene solo feature appartenenti al carrello. L'aggiunta dell'euristica basata su IoU permette invece di trovare il modello corretto usando solo feature della stella.

Per verificare quantitativamente quale sia l'approccio migliore per la ricerca del percorso nel grafo e quindi la costruzione del modello sono state provate numerose configurazioni dei parametri $\alpha$, $\beta$, $t_m$ e $t_{A^*}$. I risultati, espressi in termini di feature assegnate al modello, sono riportati in Tabella 5.1.

| | | **Star** | | | | **StarCart** | | | | **Lattine** |
|---|---|---|---|---|---|---|---|---|---|---|
| | $t_m$ | 0.2 | 0.4 | 0.6 | 0.8 | 0.2 | 0.4 | 0.6 | 0.8 | 0.2 |
| **Metodo iniziale** | | 40 | 40 | 40 | 40 | 0* | 40 | 40 | 40 | 5 |
| **Euristica IoU** | $\alpha = 0.5, \beta = 0.5$ | 39 | 39 | 39 | 39 | 39 | 39 | 39 | 39 | 4 |
| | $0.7, 0.3$ | 39 | 39 | 39 | 39 | 40 | 40 | 40 | 40 | 5 |
| | $0.8, 0.2$ | 39 | 39 | 39 | 39 | 40 | 40 | 40 | 40 | 5 |
| | $0, 1$ | 30 | 30 | 30 | 30 | 39 | 38 | 38 | 38 | 4 |
| **A* più percorsi** $\alpha = 1, \beta = 0$ | $t_{A^*} = 0.8$ | 40 | 40 | 40 | 40 | 0* | 40 | 40 | 40 | 5 |
| | 0.85 | 40 | 40 | 40 | 40 | 0* | 40 | 40 | 40 | 5 |
| | 0.9 | 40 | 40 | 40 | 40 | 0* | 40 | 40 | 40 | 5 |
| **Euristica IoU, A* più percorsi** $t_{A^*} = 0.8$ | $\alpha = 0.5, \beta = 0.5$ | 40 | 40 | 40 | 40 | 39 | 39 | 39 | 40 | 5 |
| | $0.7, 0.3$ | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 5 |
| | $0.8, 0.2$ | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 5 |
| | $0, 1$ | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 5 |

**Tabella 5.1:** Risultati, espressi in termini di numero di feature nel modello, al variare dei parametri della creazione del modello. 0* indica che tutte le feature proposte per il modello risultano essere errate. $\alpha$ e $\beta$ regolano la costruzione dell'euristica (Sezione 3.2.3), $t_m$ il filtro dei match (Sezione 3.1.1.2) e $t_{A^*}$ l'esplorazione di percorsi oltre il migliore in A* (Sezione 3.2.3).

Dal confronto dei vari esperimenti con il modello restituito dal metodo iniziale emerge come varie combinazioni lineari delle due euristiche portino a scartare alcune delle feature. Considerare più percorsi restituiti da A* è in grado di ovviare parzialmente e raggiungere il numero massimo di sovrapposizioni, in particolare questo si verifica quando l'euristica sul numero di sovrapposizioni viene ignorata, confermando che la considerazione iniziale della problematica dovuta a individuare il parametro $\alpha$ fosse almeno parzialmente fondata. Si può tuttavia valutare che considerare una sola euristica invece che due abbia il vantaggio che si possono scartare tutti i calcoli e i dati legati all'euristica delle sovrapposizioni.

### 5.1.3 Parallelizzazione

Dai risultati in Tabella 5.2 emerge come su grafi limitati in larghezza i tempi di A* e SumPath siano comparabili mentre su grafi molto larghi, come quello di *Curry* che contiene molti segmenti, A* sia più performante.

L'introduzione del parallelismo all'interno di A* permette di migliorare di molto i tempi di esplorazione del grafo in training, la terza fase indicata come *Modello* in Tabella 5.2. Nel caso di grafi più ampi sono raggiunti tempi 2 volte più veloci della versione non parallela e fino a 3.4 volte di SumPath. Si è pertanto deciso di proseguire utilizzando A* nella versione parallelizzata.

| | | **SumPath** | **A*** | **A* parallelo** | | |
|---|---|---|---|---|---|---|
| | **N° immagini** | 4 | 4 | 2 | 3 | 4 |
| **Lattine** | **Feature** | 119.50 ms | 111.54 ms | 36.61 ms | 62.60 ms | 80.88 ms |
| | **Grafo** | 62.90 ms | 67.26 ms | 10.18 ms | 28.37 ms | 38.91 ms |
| | **Modello** | 137.75 ms | 140.12 ms | 56.11 ms | 73.31 ms | 128.40 ms |
| | **Totale** | 323.81 ms | 321.76 ms | 105.98 ms | 168.31 ms | 251.55 ms |
| **Curry** | **Feature** | 192.71 ms | 189.02 ms | 68.21 ms | 91.05 ms | 116.85 ms |
| | **Grafo** | 667.05 ms | 656.69 ms | 221.68 ms | 277.43 ms | 377.07 ms |
| | **Modello** | 40.79 s | 24.82 s | 44.78 ms | 555.64 ms | 11.90 s |
| | **Totale** | 41.65 s | 25.67 s | 339.53 ms | 928.99 ms | 12.40 s |

**Tabella 5.2:** Tempi di training per SumPath e A* al variare del numero di immagini e di calcolo seriale o in parallelo. I tempi sono suddivisi nelle tre fasi di training illustrate in Figura 3.1.

## 5.2 Ricerca della ROI

Per valutare l'efficacia dei descrittori impiegati e la bontà della ROI individuata, sono state individuate manualmente le ROI anche nelle immagini di test dei dataset.

### 5.2.1 Valutazione dei descrittori

Per valutare quale sia il descrittore migliore per valutare il contenuto delle ROI sono stati considerati i dataset *Lattine* e *LattineBagnate*. Per ogni immagine di test è stata considerata la ROI reale e una serie di ROI sbagliate create attorno a questa mantenendo inalterata la dimensione e applicando rotazioni e traslazioni. Le posizioni generate sono deterministiche e consistenti tra esecuzioni successive.

Per effettuare il confronto, tutte le immagini di training concorrono alla creazione del descrittore del modello secondo modalità diverse in base al singolo descrittore. Per ogni ROI, vera o errata, viene quindi calcolato lo stesso descrittore e confrontato con quello del modello. I risultati sono presentati in maniera grafica e sotto forma di curve di distanza in funzione di IoU decrescente tra possibile posizione e ROI effettiva in Figura 5.2 e Figura 5.3.

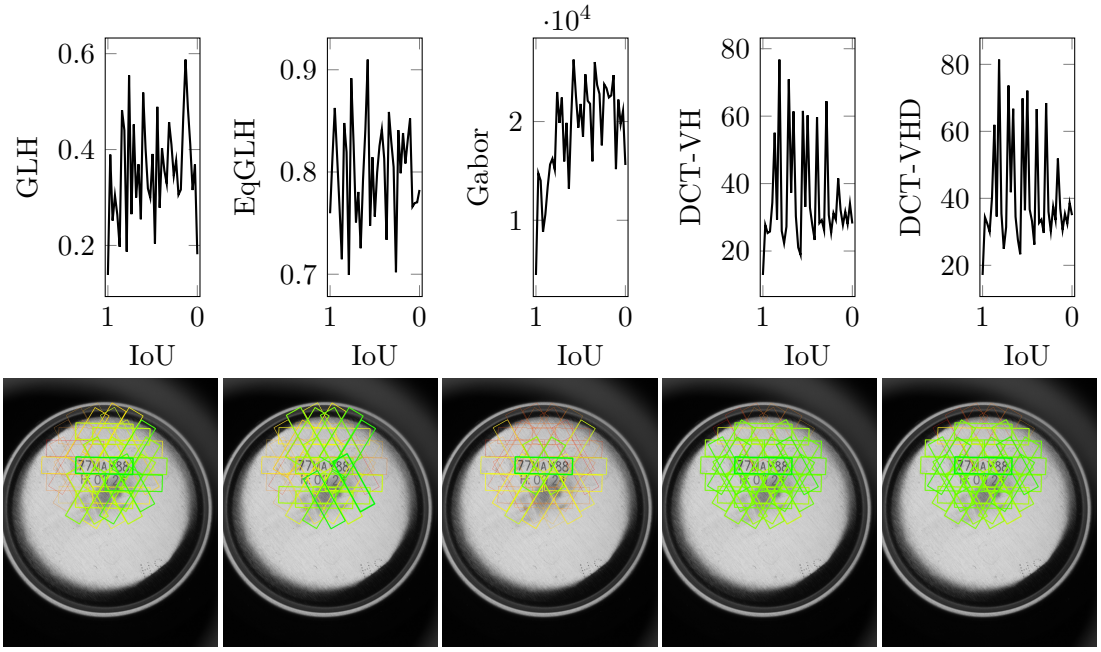
Il descrittore ideale è quello che ha valore minimo con IoU pari a 1 e a seguire i valori sono molto maggiori, possibilmente non decrescenti al calare dell'IoU. Confrontando questa considerazione con i grafici del dataset *Lattine* emerge come il descrittore che meglio rispetta ciò sia *Gabor* e seppure l'andamento non sia ideale è comunque molto buono. Considerando anche il dataset *LattineBagnate*, che presenta molti elementi di disturbo, *Gabor* si rivela ancora un descrittore molto buono in una buona parte dei casi rappresentati in Figura 5.3a, anche se in alcuni casi (Figura 5.3b) gli elementi di disturbo portano ROI con basso IoU ad avere la distanza minima. I due descrittori *DCT-VH*($D$) risultano da scartare in quanto l'andamento delle distanze non è quello desiderato, se non in alcuni casi. Anche se per ($Eq$)*GLH* c'è una leggera tendenza a crescere, l'andamento complessivo non è quello desiderato, tant'è che spesso la ROI originale non ha la distanza minima. Queste sono considerazioni generali e non tutte le ROI analizzate saranno proposte dall'algoritmo di ricerca del modello.

Un secondo indice di valutazione di un descrittore è il tempo medio per calcolarlo, in Tabella 5.3 sono riportati i tempi medi per calcolare il descrittore di una ROI e confrontarlo con quello del modello. I tempi sono calcolati su 195 ROI tra 5 immagini di test.

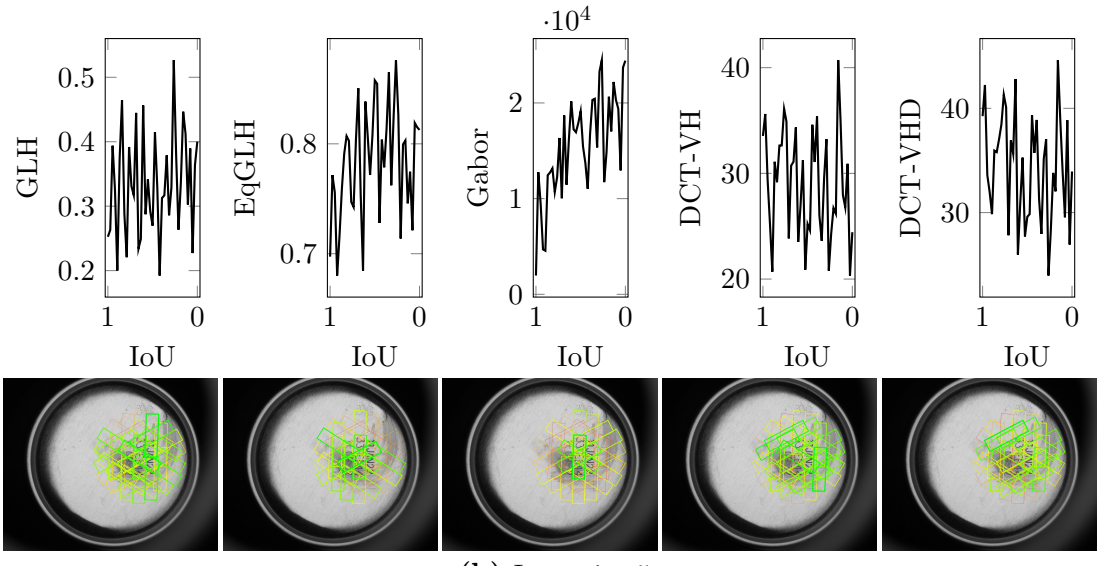
Dall'analisi dei tempi emerge come, con l'esclusione di *DCT-VH*($D$) che non offre l'andamento richiesto, *Gabor* sia il descrittore più veloce da calcolare e confrontare. Inoltre occorre tenere in considerazione che se i risultati della GHT, l'unico metodo di ricerca basato sui descrittori della ROI, indicano una sola possibile posizione, questi calcoli non vengono effettuati.

| | **GLH** | **EqGLH** | **Gabor** | **DCT-VH** | **DCT-VHD** |
|---|---|---|---|---|---|
| **Lattine** | 111.14 ms | 96.20 ms | 13.14 ms | 386.02 µs | 376.02 µs |
| **LattineBagnate** | 22.56 ms | 22.63 ms | 5.44 ms | 175.76 µs | 188.49 µs |

**Tabella 5.3:** Tempi medi per il calcolo e confronto di un descrittore.

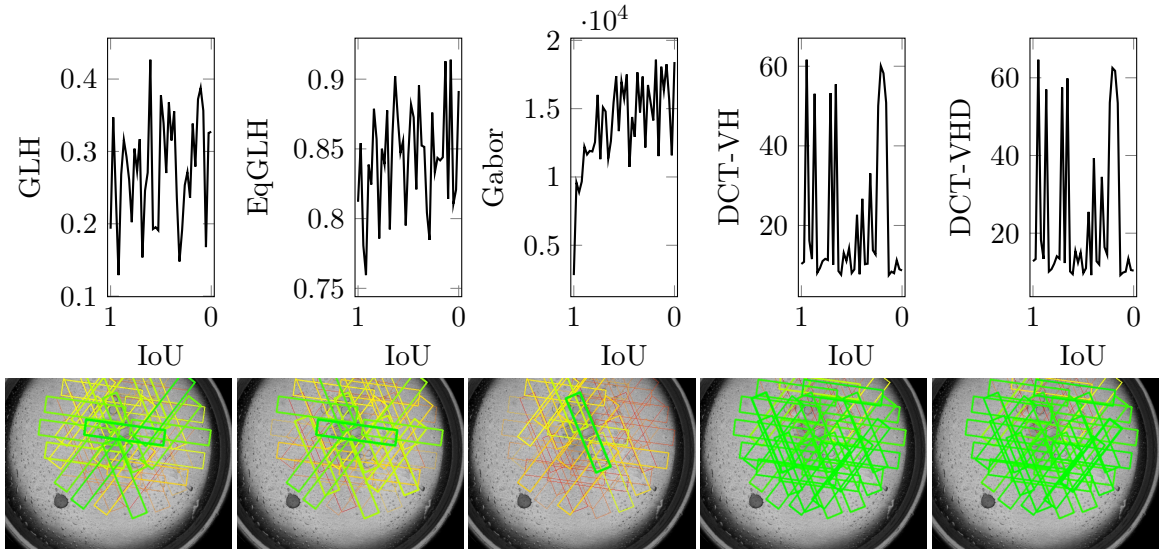



**(a)** Immagine 1.

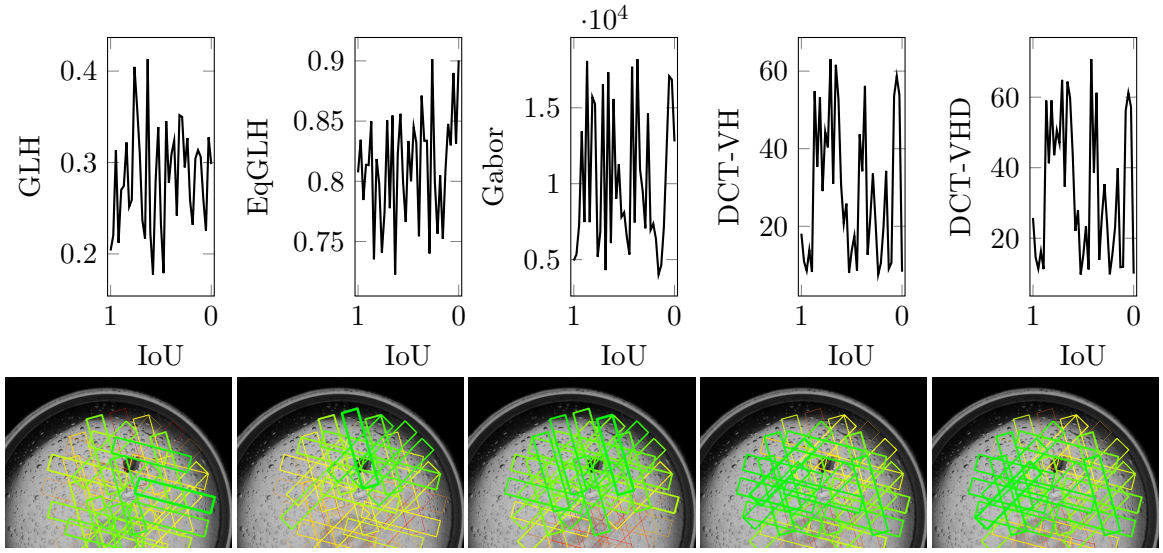



**(b)** Immagine 5.

**Figura 5.2:** Confronto dei descrittori in due immagini del dataset *Lattine*. Le possibili posizioni delle ROI sono evidenziate con spessore maggiore per quella a distanza minima, a distanza minima corrisponde il verde e massima il rosso.




**(a)** Immagine 1.




**(b)** Immagine 5.

**Figura 5.3:** Confronto dei descrittori in due immagini del dataset *LattineBagnate*.

#### 5.2.1.1 Distanza di Mahalanobis

Una possibile alternativa alle distanze presentate per i descrittori *Gabor* e *DCT-VH*, è di utilizzare la distanza di Mahalanobis per il confronto per valutare se è presente una correlazione tra i diversi bin degli istogrammi. Dai test è stato escluso *DCT-VHD* che non presenta differenze significative dal simile *DCT-VH*.

In Figura 5.4 sono riportati i confronti tra le due distanze ed emerge come la distanza di Mahalanobis offra più o meno lo stesso andamento, se non con un leggero peggioramento. Questo mostra come non sia presente una correlazione tra i differenti valori dei descrittori *Gabor* e *DCT-VH*, per questo motivo la distanza di Mahalanobis può essere accantonata dato che essendo una distanza quadratica richiede uno sforzo computazionale maggiore di una norma $L_p$. Esperimenti successivi sono pertanto eseguiti utilizzando la distanza inizialmente proposta.

### 5.2.2 Valutazione della posizione

Dalla valutazione dei risultati con diversi metodi di ricerca (Figura 5.5), emerge come l'uso della GHT sia fondamentale per introdurre robustezza a pattern simmetrici che portano *PerfectAlignment* e *RobustSimilarity* a sbagliare di molto le ROI proposte. Per quanto riguarda la scelta del migliore approccio per la selezione della ROI tra quelle proposte dalla GHT, tutti e tre i descrittori *(Eq)GLH* e *Gabor* risultano essere efficaci per restituire i risultati corretti ma solo *Gabor* permette di individuare correttamente tutte le posizioni. La scelta può dunque essere orientata verso *Gabor* in quanto offre anche l'andamento che più approssima quello ideale e il tempo di calcolo minore.

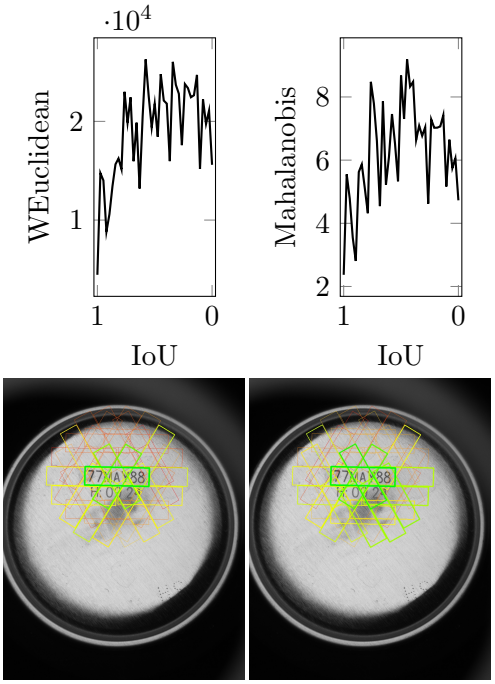
Confrontando con i tempi medi della ricerca in una singola immagine riportati in Tabella 5.3, emerge come la ricerca con GHT sia anche quella più veloce e, come accade nel dataset *Curry*, la velocità è ulteriormente ridotta in quanto la presenza di pattern non simmetrici permette di convergere verso un numero minore di candidati, idealmente uno solo.

| N° immagini | | | PerfectAlignment | RobustSimilarity | GHT+Gabor |
|---|---|---|---|---|---|
| **Lattine** | **87** | **Match** | 32.49 ms | 22.41 ms | 22.03 ms |
| | | **Ricerca** | 148.22 ms | 100.38 ms | 29.34 ms |
| | | **Totale** | 184.66 ms | 124.75 ms | 53.21 ms |
| **Curry** | **16** | **Match** | 45.41 ms | 45.07 ms | 46.41 ms |
| | | **Ricerca** | 130.80 ms | 188.37 ms | 11.54 ms |
| | | **Totale** | 178.50 ms | 235.52 ms | 60.00 ms |

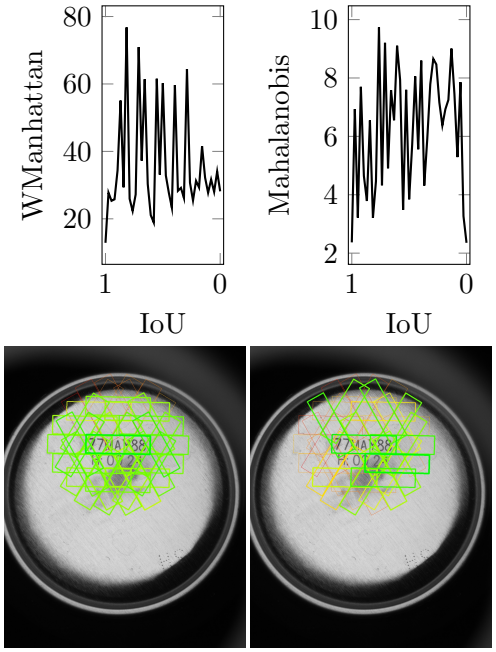
**Tabella 5.4:** Tempi medi per la ricerca della ROI con diversi metodi di ricerca su differenti dataset.

### 5.2.3 Altri miglioramenti

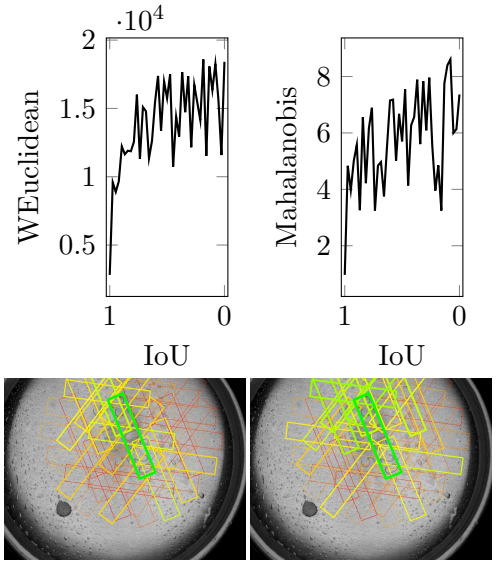
Dai risultati precedenti emerge come la struttura base del locatore sia composta dal modello costruito usando A* per esplorare il grafo e GHT per guidare la ricerca della ROI,



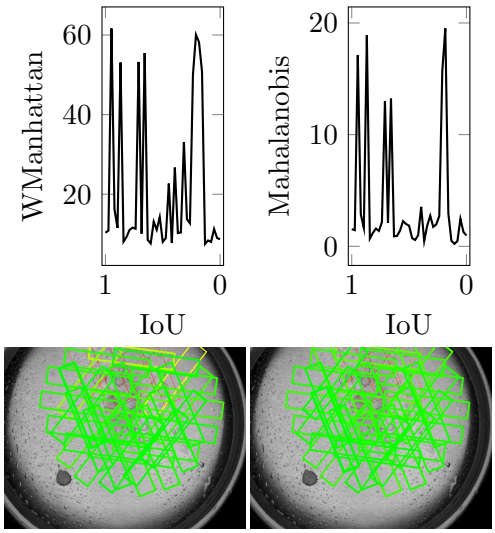

**(a)** Lattine, Gabor.




**(b)** Lattine, DCT-VH.




**(c)** LattineBagnate, Gabor.




**(d)** LattineBagnate, DCT-VH.

**Figura 5.4:** Confronto tra distanza di Mahalanobis e distanza euclidea/di Manhattan pesate, rispettivamente per *Gabor* e *DCT-VH*.

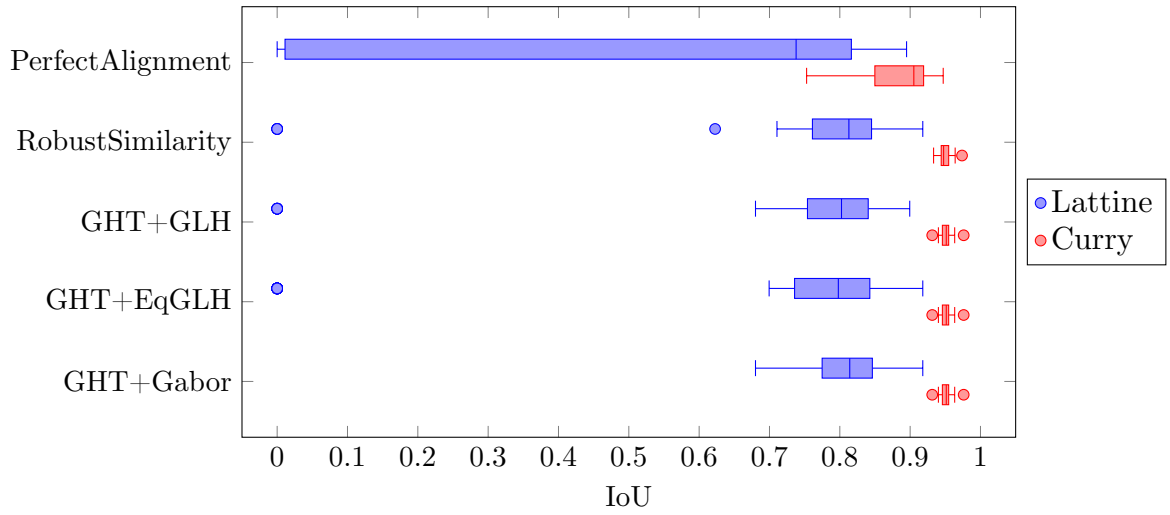



**Figura 5.5:** Distribuzione degli IoU calcolati con diversi metodi di ricerca su differenti dataset.

utilizzando *Gabor* come descrittore per discriminare possibili posizioni alternative.

Di seguito sono presentati ulteriori miglioramenti che partono da questa struttura per migliorare i tempi di training o i risultati della ricerca. Se non diversamente indicato ogni miglioramento utilizza la configurazione migliore fornita dal precedente.

#### 5.2.3.1 Trasformazioni robuste nel grafo

Dai risultati in Figura 5.6 emerge come ricalcolare la trasformazione con RANSAC offra un miglioramento, per quanto piccolo.

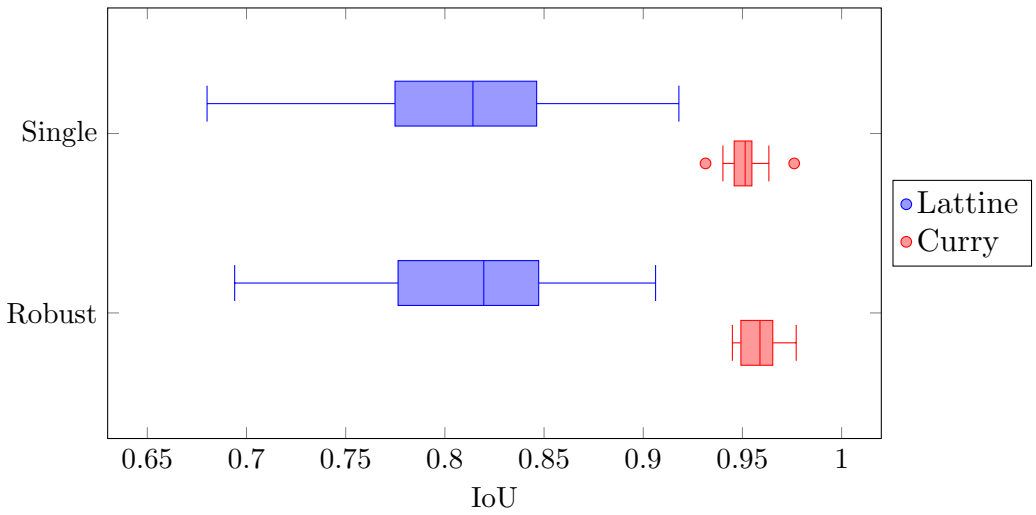



**Figura 5.6:** Distribuzione degli IoU calcolati con diverse trasformazioni per i nodi del grafo.

#### 5.2.3.2 Uso di un modello più ricco

L'uso di un modello più ricco con feature da tutte le immagini di training non offre cambiamenti significativi sia in termini di training che di ricerca come mostrato in Figura 5.7 e Tabella 5.5.

Nonostante questa strategia non causi peggioramenti, non ci sono nemmeno miglioramenti significati nella capacità di ricerca. Tuttavia, poiché la variabilità dei tempi di ricerca è accettabile, si è deciso di procedere con *SuperFeatDesc* in quanto l'utilizzo di descrittori da tutte le immagini di training offre più robustezza a possibili variabilità nelle immagini presentate.

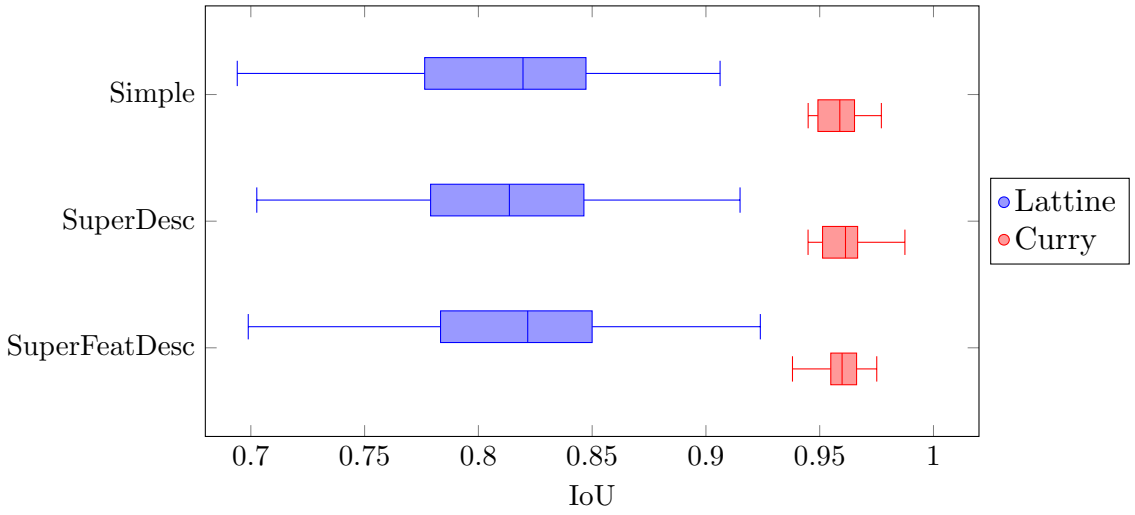



**Figura 5.7:** Distribuzione degli IoU calcolati con diversi metodi di costruzione del modello usando GHT+Gabor come ricerca.

| | **N° immagini** | | **Simple** | | **SuperDesc** | | **SuperFeatDesc** | |
|---|---|---|---|---|---|---|---|---|
| | Train | Ricerca | Train | Ricerca | Train | Ricerca | Train | Ricerca |
| **Lattine** | 4 | 87 | 355.85 ms | 97.40 ms | 335.81 ms | 69.31 ms | 351.96 ms | 55.58 ms |
| **Curry** | 4 | 16 | 50.82 s | 57.57 ms | 48.73 s | 74.11 ms | 48.43 s | 72.93 ms |

**Tabella 5.5:** Tempi medi di training di ricerca (GHT+Gabor) della ROI con diversi metodi di costruzione del modello su differenti dataset.

#### 5.2.3.3 Numero di immagini di training

Tutti i risultati mostrati fino ad ora utilizzano 4 immagini per il training, tuttavia occorre valutare se ridurre il numero di immagini possa avere effetti sul modello creato e le posizioni delle ROI individuate nelle immagini di test.

Dai risultati in Figura 5.8 emerge come l'uso di 4 immagini di training causi un peggioramento nella fase di ricerca rispetto a 3 immagini. Anche aggiungendo la terza immagine di training c'è un peggioramento, tuttavia in questo caso è trascurabile.

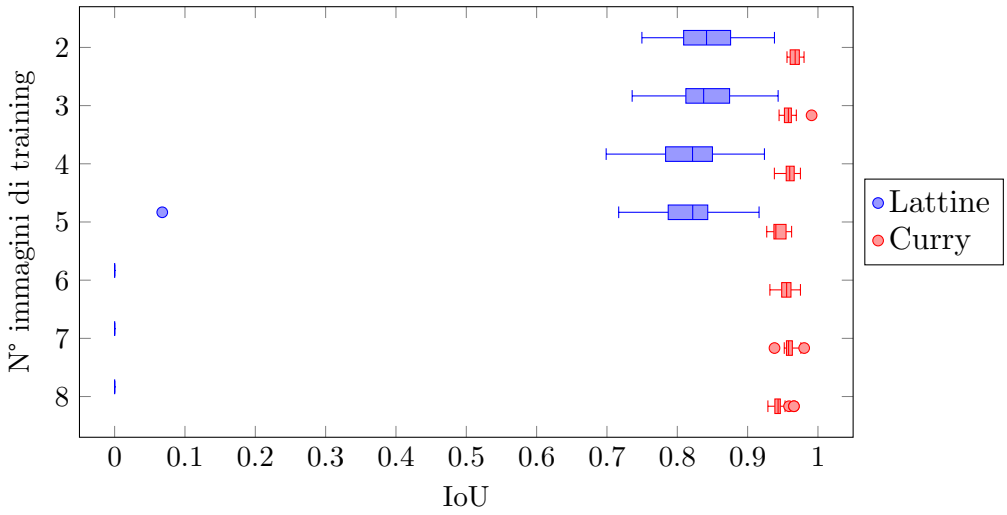
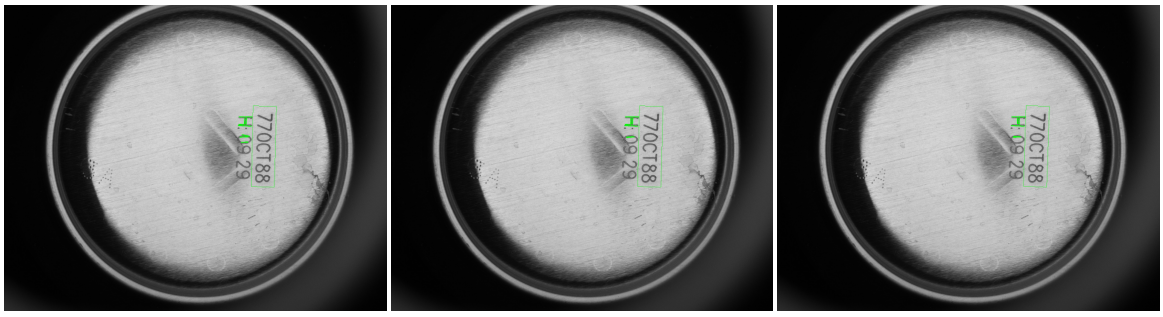
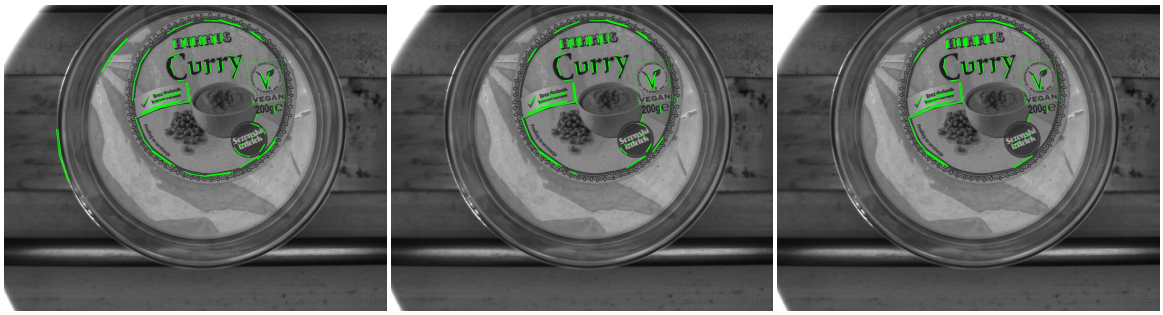



**Figura 5.8:** Distribuzione degli IoU calcolati al variare del numero di immagini di training.

Considerando anche i tempi di training (Tabella 5.6b), un aumento è atteso, tuttavia l'aggiunta della quarta immagine di training provoca un aumento eccessivo del tempo medio, in particolare per *Curry* dove le immagini più ricche di feature rendono più complessa l'esplorazione del grafo per la costruzione del modello. Una rappresentazione



**(a)** Lattine.




**(b)** Curry.

**Figura 5.9:** Cambiamento del modello all'aggiunta di immagini di training, sono presentati i modelli con 2, 3 e 4 immagini di training per ogni dataset.

| | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| **Lattine** | 8 | 6 | 5 | 3 | – | – | – |
| **Curry** | 96 | 78 | 67 | 56 | 50 | 47 | 43 |

**(a)** Numero di feature assegnate al modello.

| | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| **Lattine** | 155.71 ms | 259.89 ms | 349.27 ms | 467.25 ms | 503.23 ms |
| **Curry** | 839.72 ms | 2.72 s | 47.54 s | 106.58 s | 179.76 s |

**(b)** Tempi di training medi.

**Tabella 5.6:** Effetti della variazione del numero di immagini training.

più immediata dei risultati in Tabella 5.6a è presente in Figura 5.9.

Da risultati sperimentali sulle applicazione in cui SAFFIRE dovrà operare risulta che l'uso di sole 3 immagini di training sembra quindi essere la scelta migliore con un giusto compromesso tra precisione della ricerca e tempo di training. Questo risultato verrà poi confermato dalla versione del locatore in grado di terminare il training in maniera automatica come riportato in Sezione 5.2.4.2 e Sezione 5.2.5.

#### 5.2.3.4 Cambio dell'immagine di riferimento

Utilizzando la migliore configurazione identificata con i test precedenti, cioè la costruzione del modello secondo il modello *SuperFeatDesc* usando solo 3 immagini di training e la ricerca basata su GHT e *Gabor* per discriminare possibili alternative, si vuole verificare se e come cambia la ricerca e il modello al variare dell'immagine di riferimento e non tenere sempre la prima come tale.

Dai risultati di ricerca in Figura 5.10 emerge come non ci sia un cambio significativo nella distribuzione dell'IoU per *Curry* ma sia presente per *Lattine*, sembra però plausibile assumere che *Lattine* sia un caso fortuito in cui l'immagine numero 3 causi problemi quando è usata come riferimento. Il numero di feature assegnate al modello

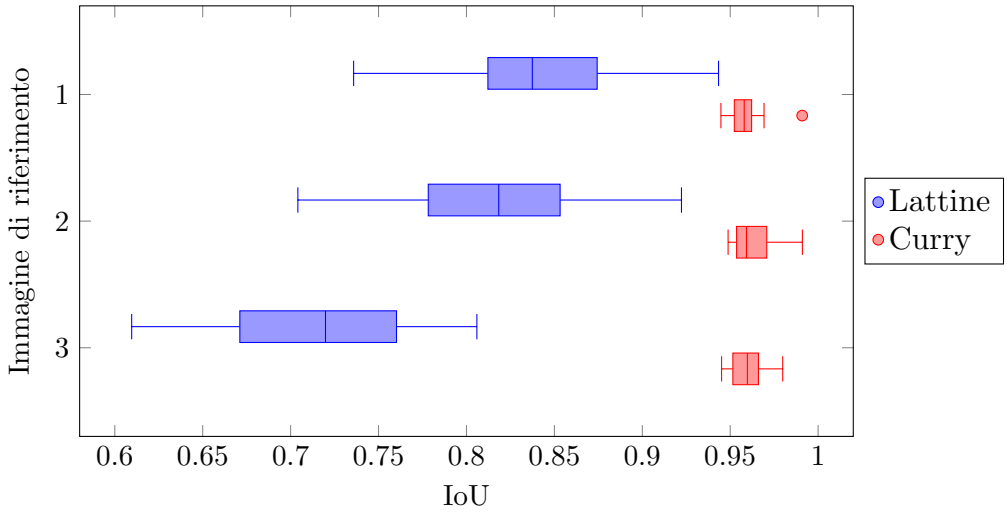



**Figura 5.10:** Distribuzione degli IoU calcolati al variare del numero dell'immagine di riferimento.

| | #1 | #2 | #3 |
|---|---|---|---|
| **Lattine** | 6 | 7 | 7 |
| **Curry** | 78 | 77 | 76 |

**Tabella 5.7:** Variazione del numero di feature assegnate al modello al variare dell'immagine di riferimento.

(Tabella 5.7) rimane costante perciò sembra che la scelta dell'immagine di riferimento sia indifferente per la buona riuscita della ricerca.

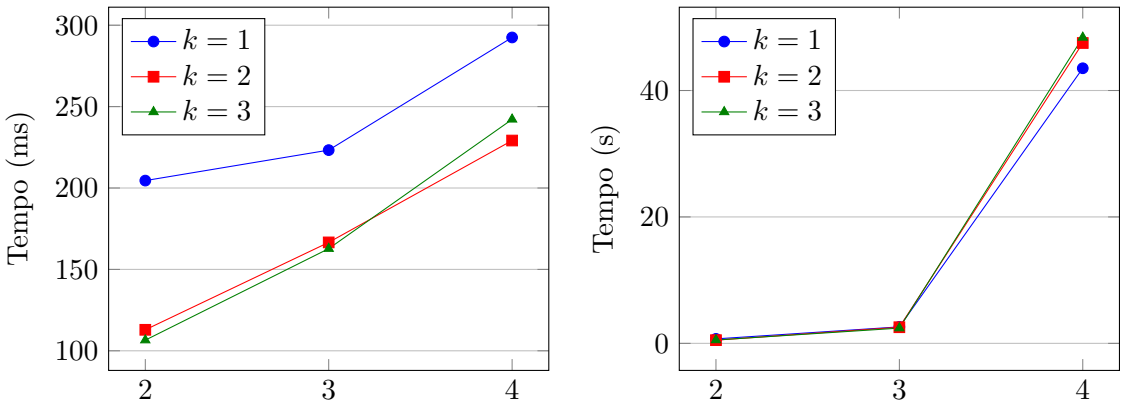
#### 5.2.3.5 Cambio di $k$

Si nota come al variare del parametro $k$ della ricerca $k$NN dei match non ci siano variazioni significative nell'accuratezza della ricerca (Figura 5.12), lo stesso vale per il numero di feature assegnate al modello (Tabella 5.8).

| | | 2 | 3 | 4 |
|---|---|---|---|---|
| **Lattine** | $k=1$ | 8 | 6 | 5 |
| | 2 | 8 | 6 | 5 |
| | 3 | 8 | 6 | 5 |
| **Curry** | $k=1$ | 96 | 78 | 67 |
| | 2 | 96 | 78 | 67 |
| | 3 | 96 | 78 | 67 |

**Tabella 5.8:** Variazione del numero di feature assegnate al modello al variare del numero immagini training.

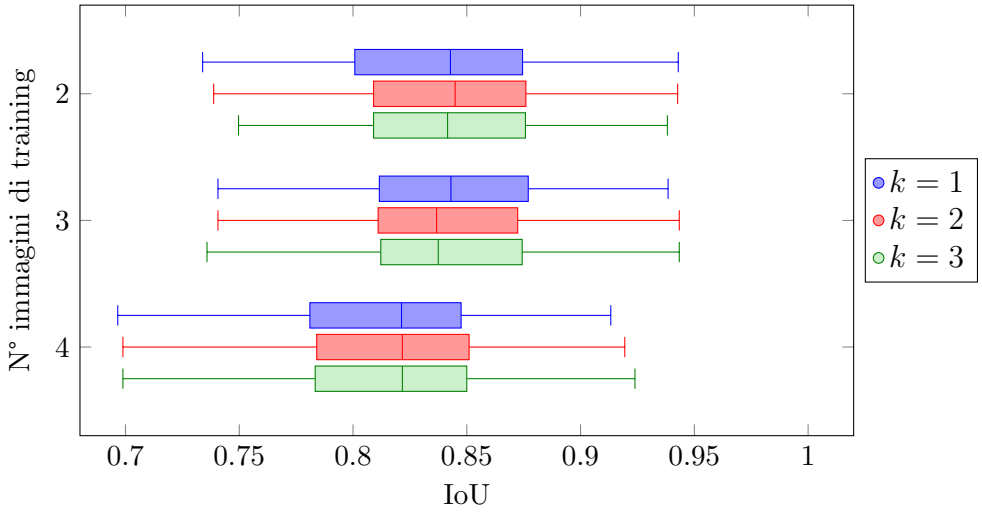
Anche il tempo di training non cambia in maniera significativa con un numero limitato di immagini di training (Figura 5.11), l'uso di più immagini è stato scartato



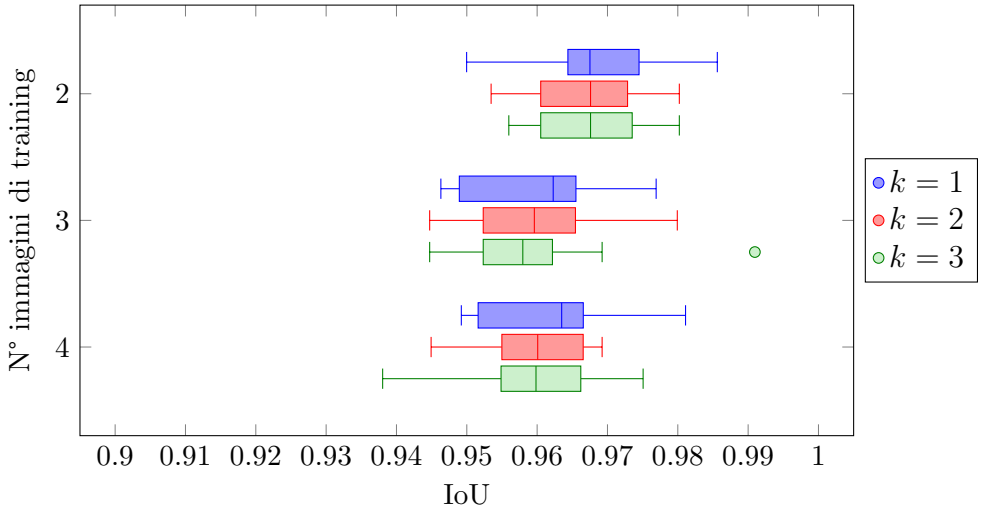

**(a)** Lattine. **(b)** Curry.

**Figura 5.11:** Tempi di training medi al variare di $k$ e del numero immagini training.

in quanto i tempi risulterebbero troppo elevati. Pertanto usare $k = 1$ risulta essere vantaggioso in quanto con meno match la fase di training presenterà un grafo semplificato e quella di ricerca dovrà processare meno voti per calcolare le posizioni secondo la GHT.




**(a)** Lattine.




**(b)** Curry.

**Figura 5.12:** Distribuzione degli IoU calcolati al variare di $k$ e del numero immagini training.

#### 5.2.3.6 Singolo match per segmento

Dalla distribuzione degli IoU ottenuti dalla ricerca (Figura 5.13) emerge come non ci siano vantaggi dal considerare per ogni segmento solamente il match a distanza minore, considerarli tutti offre invece una maggiore accuratezza nell'individuazione della ROI, dunque questo miglioramento viene scartato.

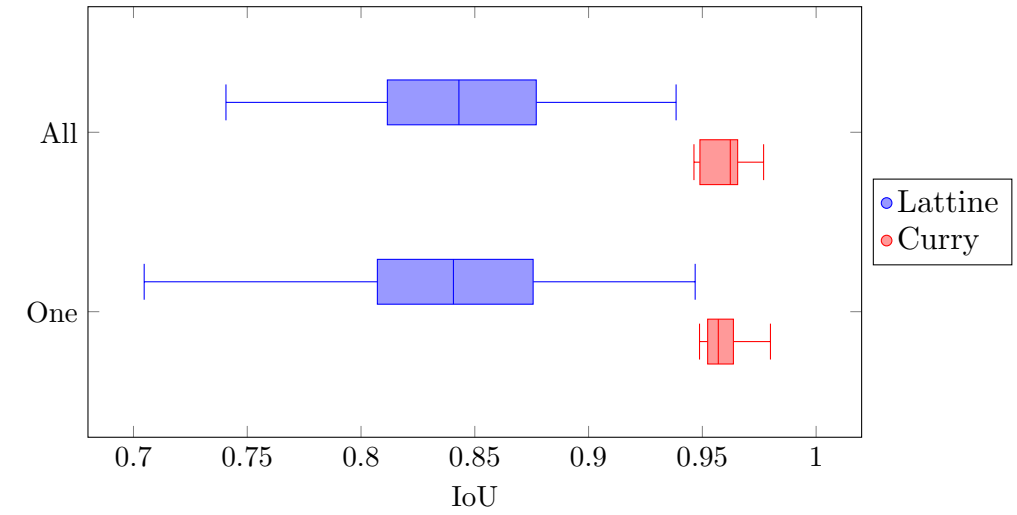



**Figura 5.13:** Distribuzione degli IoU calcolati al variare dei match considerati per la ricerca.

#### 5.2.3.7 Greediness

L'uso di un fattore di greediness maggiore di 0 permette di semplificare il grafo e quindi di ridurre i tempi di training (Tabella 5.9). L'uso di un fattore di greediness troppo elevato può però portare a non trovare più match validi, pertanto scegliere 0.75 risulta essere un buon compromesso. Valutando i risultati della ricerca in Figura 5.14 emerge

| | $g = 0$ | 0.25 | 0.5 | 0.75 | 1 |
|---|---|---|---|---|---|
| **Lattine** | 207.21 ms | 206.12 ms | 202.61 ms | 205.62 ms | 190.35 ms |
| **Curry** | 2.83 s | 2.16 s | 1.89 s | 1.99 s | 1.25 s |

**Tabella 5.9:** Tempi di training medi al variare del fattore di greediness.

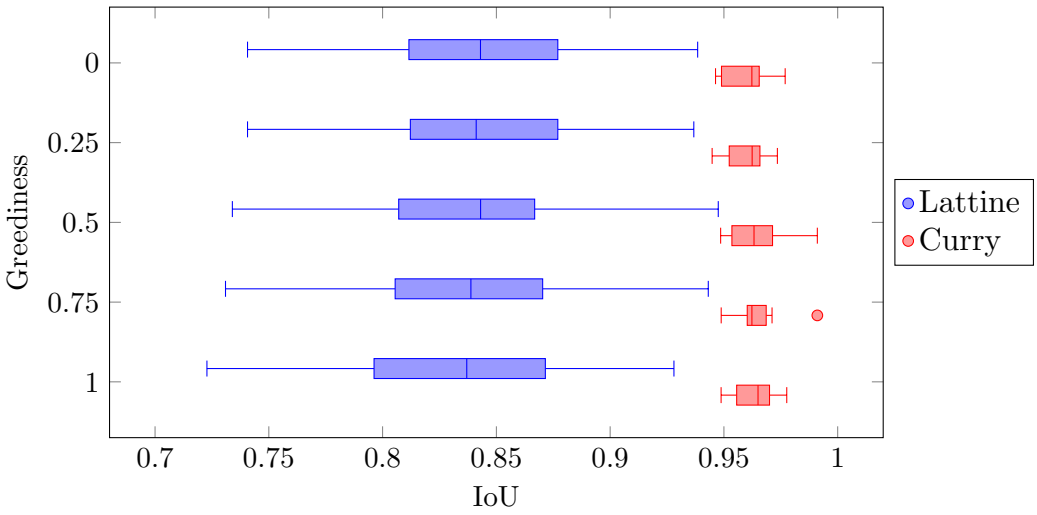



**Figura 5.14:** Distribuzione degli IoU calcolati al variare del fattore di greediness.

come non ci siano variazioni significative nella precisione della ricerca, quindi l'uso della greediness è vantaggioso.

### 5.2.4 Training incrementale

Con il passaggio al training incrementale, mantenendo invariata la configurazione, non si presenta alcuna differenza nei risultati della ricerca, tuttavia l'aggiunta di ogni nuova immagine richiede di effettuare nuovamente il training, pertanto i tempi complessivi sono più lunghi e sono riportati in Tabella 5.10.

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| **Lattine** | 82.94 ms | 131.76 ms | 184.99 ms | 235.30 ms | 268.70 ms | 167.06 ms |
| **Curry** | 75.28 ms | 402.56 ms | 1.83 s | 29.20 s | 65.76 s | 101.36 s |

**Tabella 5.10:** Baseline dei tempi di training incrementale all'aggiunta di immagini di training.

#### 5.2.4.1 Semplificazione del grafo

Valutando i risultati della ricerca (Figura 5.15) emerge come non ci siano differenze significative nei risultati della ricerca. Valutando anche i tempi richiesti per il training (Tabella 5.11) emerge però come la strategia offra un evidente vantaggio nel velocizzare l'aggiunta di un'immagine di training.

L'uso di questa tecnica fa emergere la possibilità che l'aggiunta di un'ulteriore immagine non faccia altro che estendere il percorso corrente nel grafo di un ulteriore nodo nel livello aggiuntivo. Se questo fosse vero l'uso di A* diventa eccessivo e una semplice

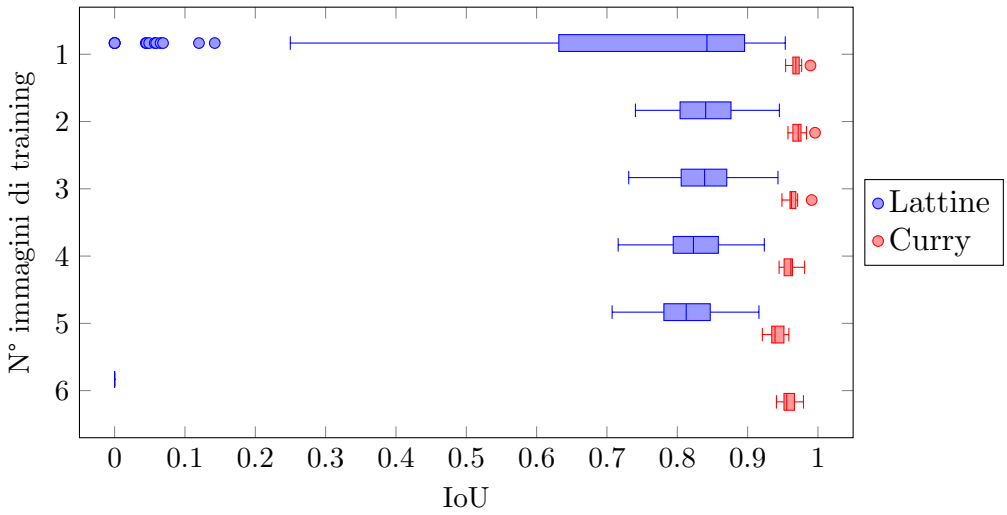



**Figura 5.15:** Distribuzione degli IoU calcolati all'aggiunta di immagini di training utilizzando la semplificazione del grafo.

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| **Lattine** | 77.31 ms | 129.63 ms | 186.44 ms | 274.92 ms | 256.66 ms | 51.51 ms |
| **Curry** | 75.61 ms | 418.47 ms | 1.59 s | 17.86 s | 42.38 s | 61.85 s |

**Tabella 5.11:** Tempi di training incrementale con semplificazione del grafo all'aggiunta di immagini di training.

valutazione di tutti i nodi del nuovo livello sarebbe sufficiente. Dall'analisi dei percorsi che generano i modelli a ogni livello di training risulta che, per quanto in alcuni dataset questo sia il caso, in altri non si verifica in quanto l'aggiunta di un'immagine può portare A* a variare parzialmente o interamente il percorso che segue nel grafo.

#### 5.2.4.2 Terminazione automatica

Nei risultati della ricerca (Figura 5.16) la mancanza della distribuzione in corrispondenza di un'immagine di training indica che non è stata aggiunta al modello, che pertanto è rimasto invariato e così i risultati della ricerca.

La decisione se non aggiungere un'immagine è basata sul fatto che la previsione col modello corrente è sufficientemente buona, valutata in termini di IoU tra previsione e vera ROI superiore a una soglia. Nell'utilizzo finale sarà l'utente a decidere se la previsione è buona, in questi test è simulato impostando una soglia opportuna per non aggiungere un'immagine e poi aggiungerne un'altra.

La condizione finale di terminazione è basata su un numero troppo basso di feature nel modello o, più semplicemente, se per tutte le immagini di training rimanenti la previsione è buona. Questa corrisponde alla prima configurazione per terminare il training,

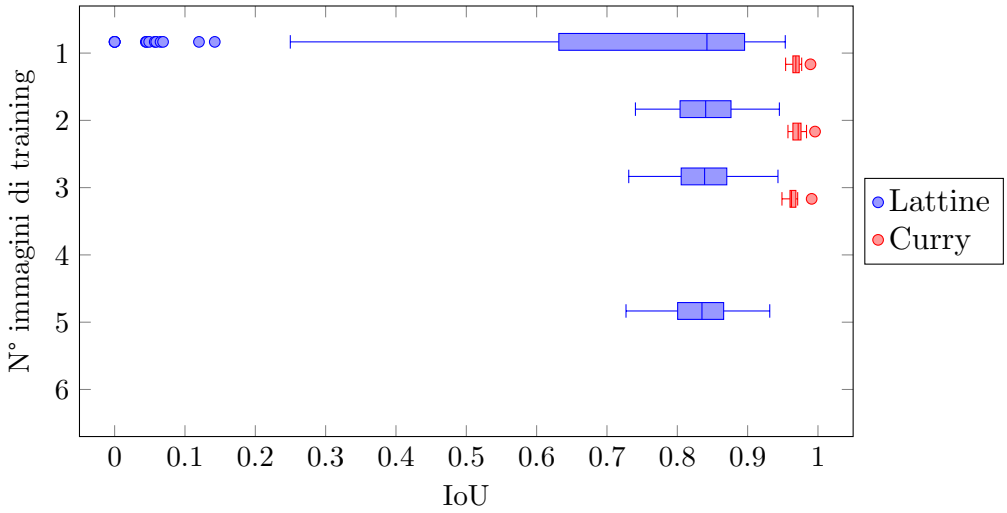



**Figura 5.16:** Distribuzione degli IoU calcolati all'aggiunta di immagini di training con stop automatico dell'aggiunta di immagini di training.

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **Lattine** | 85.66 ms | 61.17 ms | 63.25 ms | – | 50.56 ms |
| **Curry** | 73.35 ms | 67.84 ms | 64.89 ms | – | – |

**Tabella 5.12:** Tempi di ricerca medi dopo ogni aggiunta di un'immagine di training.

per i risultati forniti dalla seconda configurazione basata sull'utilizzo delle immagini di training anche per valutare le capacità di ricerca si rimanda alla Sezione 5.4.

#### 5.2.4.3 Utilizzo di più ROI

Dai risultati della ricerca (Figura 5.17) emerge come l'aggiunta indiscriminata della sola ROI dell'immagine di training, quando questa non viene usata per costruire il modello delle feature, porti a un peggioramento delle capacità, come si può notare nel caso di *Lattine* per la quarta immagine.

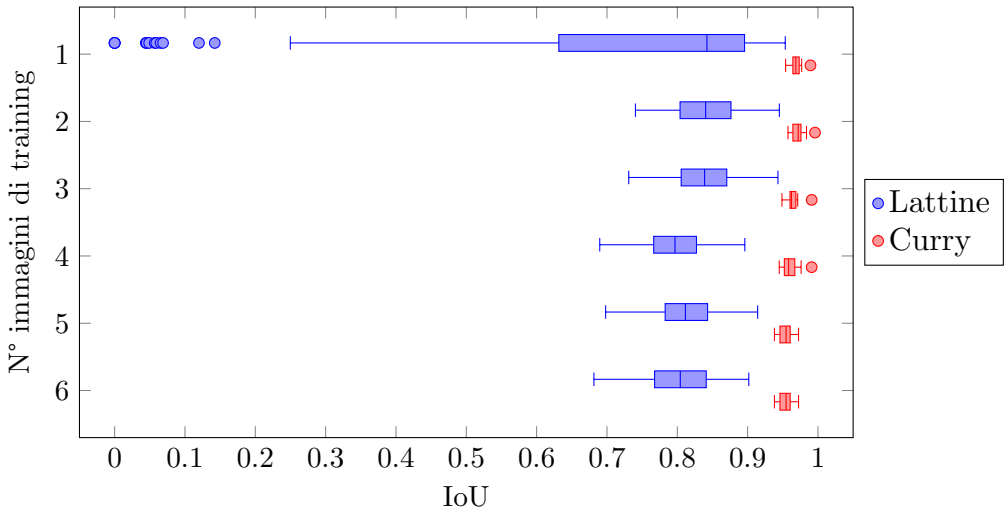



**Figura 5.17:** Distribuzione degli IoU calcolati all'aggiunta di immagini di training con aggiunta delle ROI di training quando l'immagine non viene usata per costruire il modello.

### 5.2.5 Nuovi grafi e feature

Dall'uso di EDL con *gRANSAC* emerge come l'introduzione del vincolo di un descrittore simile per considerare una sovrapposizione valida renda molto più difficile costruire un modello valido. Vari tentativi di regolazione dei parametri hanno portato ad aumentare nuovamente il parametro della ricerca a $k = 2$ e la riduzione sensibile della greediness. L'uso di super-feature composte sia da descrittori che feature stesse per il modello inoltre rende la trasformazione non troppo robusta, dunque le super-feature usate contengono solo descrittori multipli. L'uso delle sovrapposizioni basate anche su un descrittore simile semplifica però il training in quanto ora solo due immagini sono

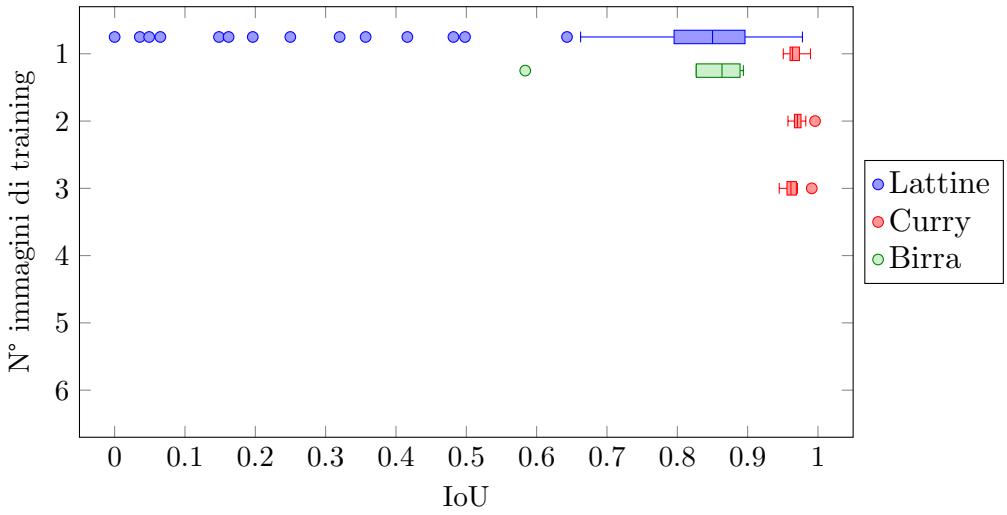



**Figura 5.18:** Risultati della ricerca con EDL basato su *gRANSAC*.

sufficienti per eliminare le feature di disturbo. I risultati con questi parametri sono mostrati in Figura 5.18, dove emerge come molte immagini di training siano scartate e come l'IoU medio sia più basso.

In Figura 5.19 sono presentati i risultati della ricerca all'aggiunta di nuove immagini di training usando *gRANSAC* basato su LSD e ORB. Dai risultati emerge come in due casi su tre ORB offra risultati migliori, mentre nel caso di *Lattine* sia LSD a risultare migliore. LSD riesce comunque a funzionare sufficientemente bene nella maggior parte dei casi nonostante presenti alcune difficoltà come un primo fallimento in training con

| | | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| **LSD** | **Lattine** | 574.06 ms | 1.08 s | 1.76 s | 255.41 ms | 301.89 ms | 254.53 ms |
| | **Curry** | 718.56 ms | 1.96 s | 468.80 s | 476.90 ms | 460.06 ms | 454.10 ms |
| | **Puntinato** | 142.07 ms | 477.14 ms | 142.98 ms | 142.52 ms | 115.77 ms | 146.66 ms |
| **ORB** | **Lattine** | 120.48 ms | 132.52 ms | 245.99 ms | 269.08 ms | – | – |
| | **Curry** | 84.84 ms | 218.41 ms | 108.48 ms | 102.05 ms | 96.11 ms | 98.05 ms |
| | **Puntinato** | 100.03 ms | 214.63 ms | 68.03 ms | 101.62 ms | 65.02 ms | 96.92 ms |

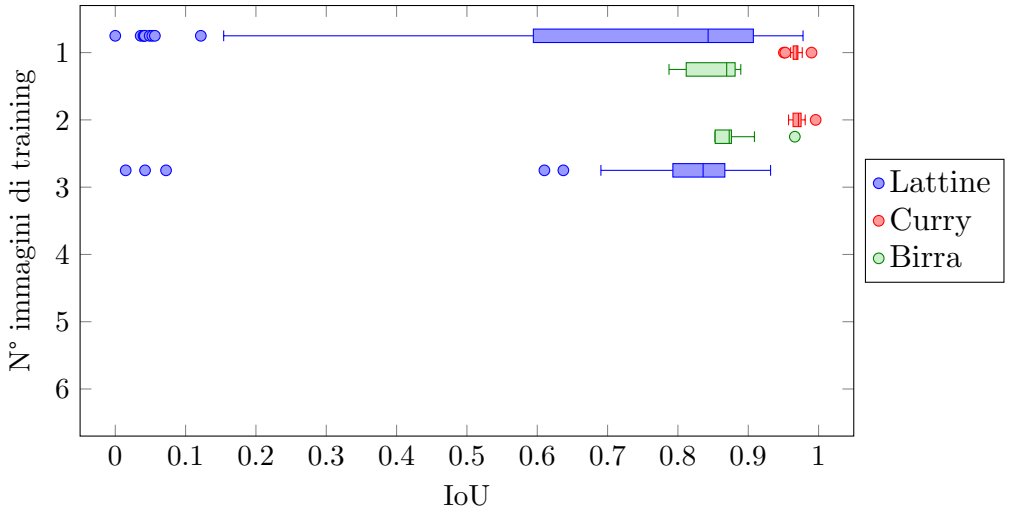
**(a)** Training.

| | | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| **LSD** | **Lattine** | 196.08 ms | – | 126.10 ms | – | – | – |
| | **Curry** | 331.12 ms | 234.73 ms | – | – | – | – |
| | **Puntinato** | 73.16 ms | 70.10 ms | – | – | – | – |
| **ORB** | **Lattine** | 46.34 ms | – | 46.01 ms | 43.87 ms | – | – |
| | **Curry** | 41.14 ms | 51.35 ms | – | – | – | – |
| | **Puntinato** | 53.27 ms | 38.84 ms | – | – | – | – |

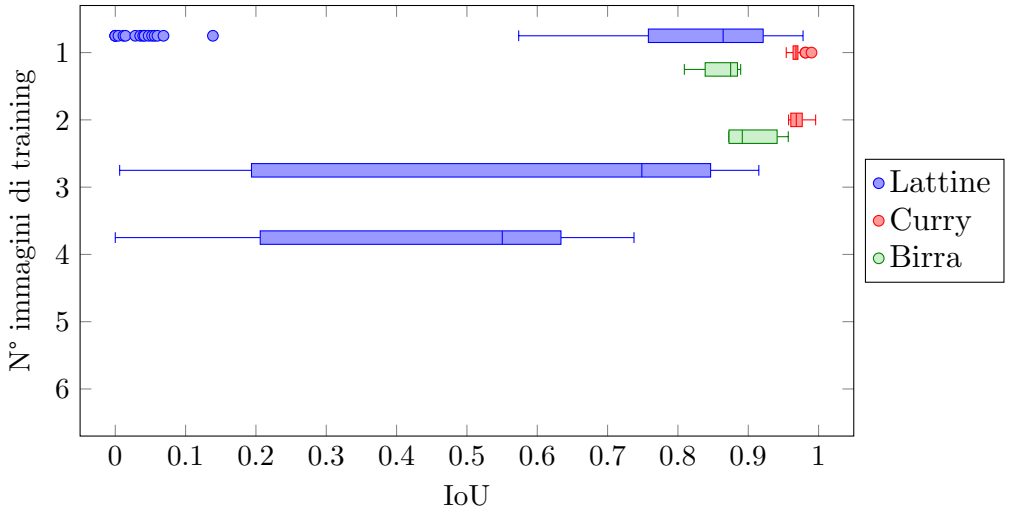
**(b)** Ricerca.

**Tabella 5.13:** Tempi medi dopo ogni aggiunta di un'immagine di training.

la seconda immagine del dataset. Come inizialmente ipotizzato LSD si rivela migliore di EDL in capacità di ricerca.




**(a)** LSD.




**(b)** ORB.

**Figura 5.19:** Distribuzione degli IoU calcolati all'aggiunta di immagini di training utilizzando un grafo *gRANSAC* al variare delle feature usate.

#### 5.2.5.1 Selezione del descrittore migliore

Per scegliere automaticamente il descrittore migliore vengono utilizzate non solo alcune immagini del dataset per effettuare il training, ma tutte quelle disponibili. Una volta completato il training tutte le immagini rimanenti vengono utilizzate per scegliere tra ORB e LSD in una fase denominata *Scelta* in cui i due modelli vengono utilizzati per cercare la ROI e confrontarla con quella reale.

I tempi di scelta così ottenuti sono pertanto dipendenti dal numero di immagini utilizzate e per questo elevati. Possono però essere ridotti utilizzando un numero minore di immagini per guidare la scelta. I risultati della ricerca e i tempi sono riportati in Figura 5.20 e Tabella 5.14.

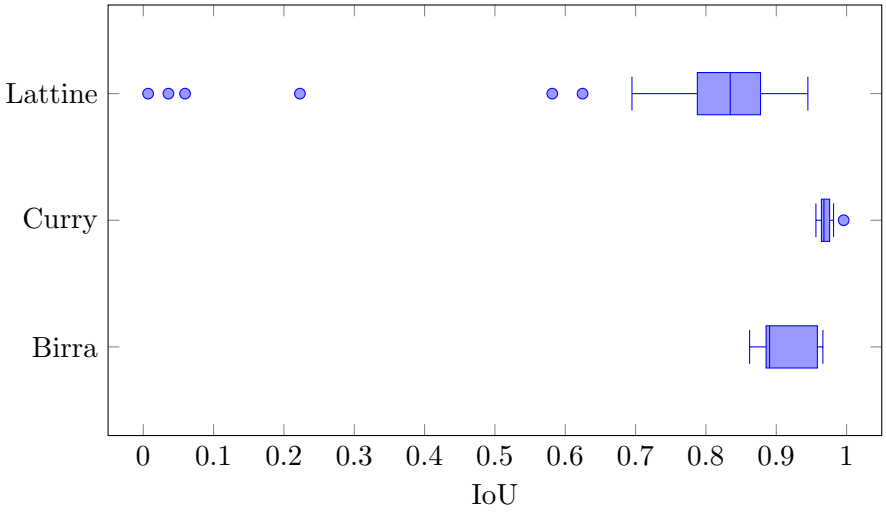



**Figura 5.20:** Distribuzione degli IoU con il locatore ottenuto da training e scelta automatici.

| | | Lattine | Curry | Birra |
|---|---|---|---|---|
| **Train** | **Tempo** | 5.24 s | 2.87 s | 917.11 ms |
| | **Scelta** | 2.62 s | 9.61 s | 1.37 s |
| | **Locatore** | LSD | ORB | ORB |
| **Ricerca** | **Match** | 109.44 ms | 22.06 ms | 9.66 ms |
| | **GHT+Gabor** | 21.98 ms | 29.07 ms | 32.01 ms |
| | **Totale** | 132.13 ms | 51.29 ms | 41.84 ms |

**Tabella 5.14:** Tempi medi di training e ricerca per il locatore ottenuto da training e scelta automatici.

## 5.3 Velocizzare training e ricerca

Di seguito è riportato, per ogni dataset, come varia l'accuratezza della ricerca e la velocità con l'uso, in maniera individuale, di diversi cambiamenti per migliorare la velocità senza compromettere l'accuratezza. Per alcuni dataset sono riportati i risultati sia per le immagini a dimensione intera, già visti in precedenza, sia a risoluzione ridotta, questo perché questo progetto si concentra principalmente su immagini di dimensione limitata e alcuni parametri sensibili alla dimensione delle immagini come quelli di RANSAC e il numero minimo di sovrapposizioni in ogni nodo sono stati ridotti per funzionare meglio su immagini piccole e vengono ora aumentati dinamicamente sulla

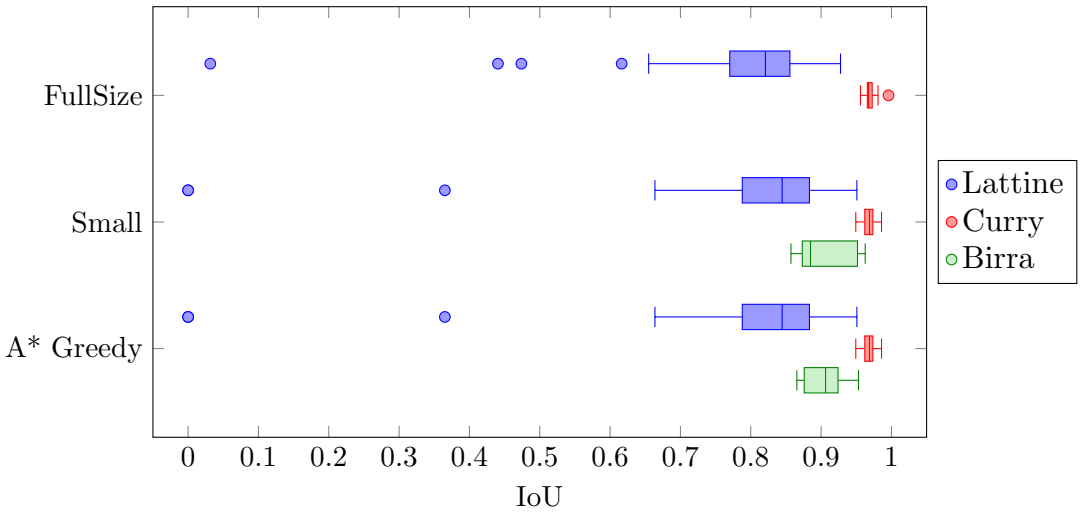



**Figura 5.21:** Distribuzione degli IoU secondo diversi approcci per velocizzare training e ricerca.

| | | Lattine | | | Curry | | | Birra | |
|---|---|---|---|---|---|---|---|---|---|
| | | FullSize | Small | A* Greedy | FullSize | Small | A* Greedy | Small | A* Greedy |
| **Train** | **Tempo** | 4.92 s | 2.52 s | 2.54 s | 8.28 s | 4.51 s | 4.62 s | 3.99 s | 3.98 s |
| | **Scelta** | 3.50 s | 747.14 ms | 790.18 ms | 9.79 s | 3.24 s | 3.26 s | 1.30 s | 1.32 s |
| | **Locatore** | LSD | LSD | LSD | LSD | ORB | ORB | ORB | ORB |
| **Ricerca** | **Match** | 106.92 ms | 51.08 ms | 52.44 ms | 174.06 ms | 10.33 ms | 10.62 ms | 9.26 ms | 8.73 ms |
| | **GHT+Gabor** | 19.40 ms | 8.46 ms | 8.96 ms | 41.59 ms | 21.99 ms | 22.50 ms | 26.53 ms | 31.23 ms |
| | **Totale** | 126.76 ms | 59.76 ms | 61.62 ms | 216.23 ms | 32.49 ms | 33.30 ms | 35.95 ms | 40.12 ms |

**Tabella 5.15:** Tempi medi secondo diversi approcci per velocizzare training e ricerca.

base della dimensione delle immagini fornite. Tutti i miglioramenti proposti sono applicati individualmente alle immagini a dimensione ridotta.

La fase di training è divisa nel training vero e proprio in cui vengono generati i modelli in base a diverse feature e una fase di scelta in cui ulteriori immagini fornite non vengono usate per migliorare i modelli ma usate solamente per decidere quale modello utilizzare. Questi tempi sono riportati ma dipendono molto da quante ulteriori immagini vengono fornite e risultano meno bloccanti per l'utente, per ogni immagine infatti questi tempi considerano il calcolo della proposta della ROI, seguita dalla correzione da parte dell'utente e infine la valutazione di quanto la previsione fosse corretta prima di passare all'immagine successiva.

Valutando i dati in Figura 5.21 e Tabella 5.15, si nota come evitare l'uso di A* e semplicemente aggiungere al percorso corrente il nodo che genera il percorso di costo minore, secondo la stessa definizione di A*, richieda tempi comparabili. Tuttavia poiché nei dataset presentati due immagini risultano sufficienti per costruire un modello valido, è possibile che l'uso di questa strategia non riesca a dimostrare appieno le proprie potenzialità.

## 5.4 Configurazione e piattaforma finale

La configurazione finale del progetto in esecuzione sullo Snapdragon è l'unica ad includere l'uso di oIoU, la riduzione dei picchi di GHT considerati, evitare il calcolo del descrittore delle possibili ROI e parallelizzarne il calcolo. Non sono invece utilizzati il subsampling e la versione greedy di A*.

Dai risultati della ricerca riportati in Figura 5.22 emergono le ottime capacità di ricerca anche in situazioni complesse come *Tea* e *Birra*. La strategia di ricerca richiede una prima parte di estrazione di feature e individuazione dei match e una seconda, la sola i cui tempi sono misurati direttamente, che aggrega i match con l'uso della GHT. Confrontando con i tempi rilevati (Tabella 5.16) risulta che i tempi richiesti dalla GHT sono mediamente molto limitati, anche quando confrontati con il tempo totale che include l'estrazione delle feature. Fa eccezione il dataset *Birra* in cui lo stile particolare del testo rende difficoltosa la ricerca richiedendo anche il calcolo del descrittore delle possibili ROI, passaggio ignorato nella maggior parte dei casi negli altri dataset, ma che se utilizzato rende il tempo della GHT pari alla metà del tempo totale.

I tempi richiesti in training sono molto più elevati, ma questa lentezza è facilmente trascurabile in quanto è un processo da eseguire una sola volta attraverso l'interazione

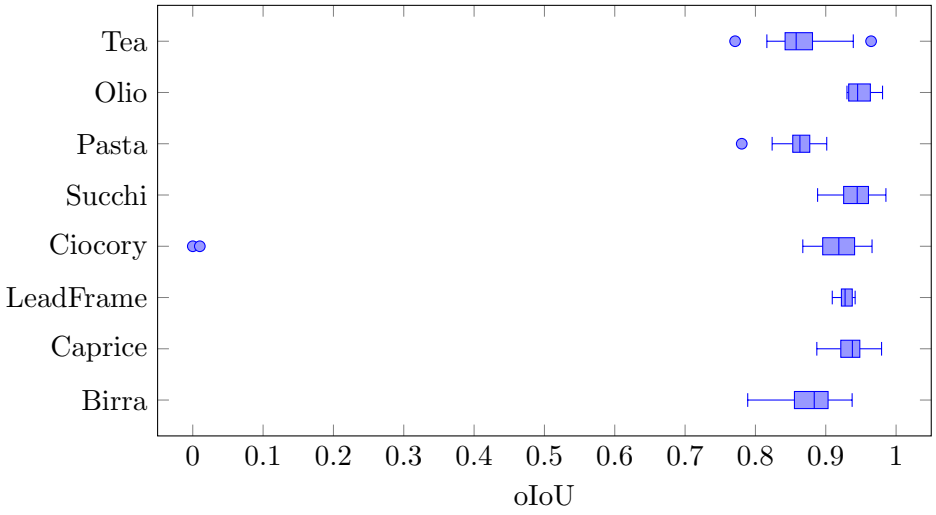



**Figura 5.22:** Distribuzione degli oIoU della configurazione finale su differenti dataset.

| | | Tea | Olio | Pasta | Succhi | Ciocory | LeadFrame | Caprice | Birra |
|---|---|---|---|---|---|---|---|---|---|
| **Train** | **Tempo** | 1.30 s | 983.33 ms | 8.63 s | 3.13 s | 3.83 s | 896.10 ms | 1.24 s | 2.78 s |
| | **Locatore** | ORB | ORB | LSD | LSD | ORB | ORB | ORB | ORB |
| **Ricerca** | **GHT+Gabor** | 8.62 ms | 2.90 ms | 744.38 µs | 1.98 ms | 774.97 µs | 4.89 ms | 1.11 ms | 45.05 ms |
| | **Totale** | 37.05 ms | 36.20 ms | 106.21 ms | 99.16 ms | 35.28 ms | 38.89 ms | 32.32 ms | 86.70 ms |

**Tabella 5.16:** Tempi medi per il training e la ricerca della configurazione finale su differenti dataset.

con l'utente, per cui il tempo totale riportato è in realtà suddiviso tra diverse fasi di training. Il numero di campioni utilizzato è stato il minimo necessario per stabilire quale descrittore utilizzare. Dai dati riportati (Tabella 5.16) emerge come ORB sia preferito nella maggior parte dei casi, questo segue dal fatto che (come si può rilevare dai tempi di ricerca) il tempo per l'estrazione delle feature è molto più breve che per LSD.

# Capitolo 6

# Conclusioni

Questo progetto dimostra come tecniche allo stato dell'arte di computer vision possano essere utilizzate assieme ad elementi innovativi per effettuare la ricerca di una ROI in un'immagine.

I risultati presentati confermano che l'utilizzo di tutte le feature da una singola immagini può pregiudicare la buona ricerca della ROI, confermando la validità del metodo proposto per filtrare le sole feature rilevanti per identificare la ROI. L'accoppiare i risultati della GHT con una seconda fase di verifica basata sul contenuto stesso della ROI risulta essere in molti casi superfluo, ma fondamentale quando non ci sono sufficienti elementi affinché la GHT converga su un consenso globale relativamente alla posizione della ROI.

La configurazione ottenuta per SAFFIRE risulta essere molto flessibile e riesce a funzionare su una vasta gamma di oggetti, ma non con tutti. È il caso del dataset *Lattine* che, quando analizzato utilizzando il grafo basato su RANSAC, risulta in scarse capacità di ricerca e difficoltà in training. Questo può essere imputato al descrittore utilizzato o alle feature stesse, lasciando aperti futuri sviluppi per individuare descrittori migliori per i segmenti o feature alternative a ORB che possano funzionare anche in questo caso.

L'uso di più descrittori allo stesso tempo è però un approccio vincente che permette di ottenere risultati migliori rispetto a scegliere solamente ORB, rivelatosi migliore nella maggior parte dei casi. I dataset reali utilizzati non hanno permesso di far risaltare l'efficacia del non-locatore, tuttavia non è da escludere che casi di questo tipo possano presentarsi. Per questo motivo, mantenere l'uso del non-locatore è una scelta valida, anche perché la complessità che aggiunge al training è trascurabile.

Il metodo di training ottenuto richiede in genere un tempo non trascurabile, ma l'approccio proposto risulta interattivo per l'utente che viene anche aiutato nella selezione delle ROI di training utilizzando i locatori con training parziale per guidare il posizionamento della ROI.

Essendo limitato alla sola individuazione della ROI, SAFFIRE può trovare le più disparate applicazioni come l'OCR per verificare che informazioni come data di scadenza e lotto siano state stampate correttamente fino alla verifica di difetti, per esempio controllando che l'etichetta incollata su una confezione sia integra e non presenti errori di stampa. Queste sono solo alcune delle possibili applicazioni, ma qualunque attività che

necessiti di validare o processare in qualunque modo una regione isolata dell'immagine può essere applicato a valle di SAFFIRE.

Oltre al già citato studio di possibili altre feature e/o descrittori, possibili sviluppi possono concentrarsi sull'individuare un descrittore del contenuto della ROI alternativo a *Gabor* che possa rendere più veloce la ricerca in quei casi in cui la sola GHT non sia sufficiente. Un altro punto che può beneficiare di miglioramento è l'ottimizzazione del grafo individuando ad esempio un'euristica meno costosa da calcolare oppure abbandonare l'uso di A* in favore di un algoritmo più efficiente per la tipologia di grafi in uso.

# Bibliografia


[1] Max K. Agoston. *Computer Graphics and Geometric Modelling*. Springer Science & Business Media, 4 gen. 2005. 859 pp. ISBN: 978-1-85233-818-3.

[2] Nasir Ahmed. «How I came up with the discrete cosine transform». In: *Digital Signal Processing* **1**.1 (1 gen. 1991), pp. 4–5. ISSN: 1051-2004. DOI: 10.1016/1051-2004(91)90086-Z.

[3] Cuneyt Akinlar e Cihan Topal. «Edlines: Real-time line segment detection by Edge Drawing (ED)». In: *International Conference on Image Processing (ICIP)*. 1 Set. 2011, pp. 2837–2840. DOI: 10.1109/ICIP.2011.6116138.

[4] Muzhir Al-Ani e Fouad Awad. «The JPEG Image Compression Algorithm». In: *International Journal of Advances in Engineering & Technology* **6** (1 mag. 2013), pp. 1055–1062.

[5] Cong Bai, Kidiyo Kpalma e Joseph Ronsin. «A new descriptor based on 2D DCT for image retrieval». In: *International Conference on Computer Vision Theory and Applications (VISAPP)*. Rome, Italy, feb. 2012. URL: https://hal.archives-ouvertes.fr/hal-00728076.

[6] D. H. Ballard. «Generalizing the Hough transform to detect arbitrary shapes». In: *Pattern Recognition* **13**.2 (1 gen. 1981), pp. 111–122. ISSN: 0031-3203. DOI: 10.1016/0031-3203(81)90009-1.

[7] Ilaria Bartolini. «Multimedia Data Management – Result Accuracy, User Interaction and Multimedia Applications». Diapositive delle lezioni. Università di Bologna, 3 giu. 2019. URL: http://www-db.disi.unibo.it/courses/MDM/PDF/9.01.ResultAccuracyMultimediaApplications.pdf (visitato il 17/05/2019).

[8] Anil Kumar Bhattacharyya. «On a measure of divergence between two statistical populations defined by their probability distributions». In: *Bulletin of the Calcutta Mathematical Society* **35** (1943), pp. 99–109.

[9] Michael Calonder, Vincent Lepetit, Christoph Strecha e Pascal Fua. «BRIEF: Binary Robust Independent Elementary Features». In: *Computer Vision – ECCV 2010*. A cura di Kostas Daniilidis, Petros Maragos e Nikos Paragios. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 778–792. ISBN: 978-3-642-15561-1. DOI: 10.1007/978-3-642-15561-1_56.

[10] Xiaorui Chen e Sara McMains. «Polygon Offsetting by Computing Winding Numbers». In: *Proceedings of the ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. Vol. 2: 31st Design Automation Conference, Parts A and B. Long Beach, California, USA, 24 set. 2005, pp. 565–575. ISBN: 0-7918-4739-X. DOI: 10.1115/DETC2005-85513.

[11] R. De Maesschalck, D. Jouan-Rimbaud e D. L. Massart. «The Mahalanobis distance». In: *Chemometrics and Intelligent Laboratory Systems* **50**.1 (4 gen. 2000), pp. 1–18. ISSN: 0169-7439. DOI: 10.1016/S0169-7439(99)00047-7.

[12] Agnès Desolneux, Lionel Moisan e Jean-Michel Morel. «Meaningful Alignments». In: *International Journal of Computer Vision* **40**.1 (1 ott. 2000), pp. 7–23. ISSN: 1573-1405. DOI: 10.1023/A:1026593302236.

[13] Martin Ester, Hans-Peter Kriegel, Jörg Sander e Xiaowei Xu. «A density-based algorithm for discovering clusters in large spatial databases with noise». In: AAAI Press, 1996, pp. 226–231.

[14] *Find the maximum sum leaf to root path in a Binary Tree*. GeeksforGeeks. 6 Apr. 2012. URL: https://www.geeksforgeeks.org/find-the-maximum-sum-path-in-a-binary-tree/ (visitato il 19/08/2019).

[15] Martin A. Fischler e Robert C. Bolles. «Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography». In: *Commun. ACM* **24**.6 (giu. 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692.

[16] Alex Fukunaga, Adi Botea, Yuu Jinnai e Akihiro Kishimoto. «A Survey of Parallel A*». In: *arXiv:1708.05296 [cs.AI]* (16 ago. 2017). arXiv: 1708.05296.

[17] Aristides Gionis, Piotr Indyk e Rajeev Motwani. «Similarity Search in High Dimensions via Hashing». In: *VLDB*. 1999.

[18] W. Eric L. Grimson. *Object Recognition by Computer: The Role of Geometric Constraints*. Cambridge, MA, USA: MIT Press, 1990. ISBN: 978-0-262-07130-7.

[19] Rafael Grompone von Gioi, Jérémie Jakubowicz, Jean-Michel Morel e Gregory Randall. «LSD: a fast line segment detector with a false detection control». In: *IEEE transactions on pattern analysis and machine intelligence* **32**.4 (apr. 2010), pp. 722–732. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2008.300.

[20] R. W. Hamming. «Error detecting and error correcting codes». In: *The Bell System Technical Journal* **29**.2 (apr. 1950), pp. 147–160. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1950.tb00463.x.

[21] Chris Harris e Mike Stephens. «A combined corner and edge detector». In: *Proceedings of the Fourth Alvey Vision Conference*. 1988, pp. 147–151.

[22] Peter E. Hart, Nils J. Nilsson e Bertram Raphael. «A Formal Basis for the Heuristic Determination of Minimum Cost Paths». In: *IEEE Transactions on Systems Science and Cybernetics* **4**.2 (lug. 1968), pp. 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136.

[23] Ernst D. Hellinger. «Neue Begründung der Theorie quadratischer Formen von unendlichvielen Veränderlichen». In: *Journal für die reine und angewandte Mathematik (Crelles Journal)* **136** (1909), pp. 210–271. ISSN: 0075-4102. DOI: 10.1515/crll.1909.136.210.

[24] Paul V. C. Hough. «Method and means for recognizing complex patterns». Brev. americano 3069654A. 18 Dic. 1962.

[25] Kenneth Levenberg. «A method for the solution of certain non-linear problems in least squares». In: *Quarterly of Applied Mathematics* **2**.2 (1944), pp. 164–168. ISSN: 0033-569X, 1552-4485. DOI: 10.1090/qam/10666.

[26] David G. Lowe. «Distinctive Image Features from Scale-Invariant Keypoints». In: *International Journal of Computer Vision* **60**.2 (1 nov. 2004), pp. 91–110. ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000029664.99615.94.

[27] Prasanta Chandra Mahalanobis. «On the Generalized Distance in Statistics». In: *Proceedings of National Institute of Sciences of India* **2**.1 (1936), pp. 49–55.

[28] Donald W. Marquardt. «An Algorithm for Least-Squares Estimation of Nonlinear Parameters». In: *Journal of the Society for Industrial and Applied Mathematics* **11**.2 (1 giu. 1963), pp. 431–441. ISSN: 0368-4245. DOI: 10.1137/0111030.

[29] *Mean of circular quantities*. In: *Wikipedia*. Page Version ID: 904656517. 3 Lug. 2019. URL: https://en.wikipedia.org/w/index.php?title=Mean_of_circular_quantities&oldid=904656517 (visitato il 15/11/2019).

[30] Mohammad Norouzi, Ali Punjani e David J. Fleet. «Fast search in Hamming space with multi-index hashing». In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919. Giu. 2012, pp. 3108–3115. DOI: 10.1109/CVPR.2012.6248043.

[31] A.G. Ramakrishnan, S. Kumar Raja e H.V. Raghu Ram. «Neural network-based segmentation of textures using Gabor features». In: *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*. Set. 2002, pp. 365–374. DOI: 10.1109/NNSP.2002.1030048.

[32] Paul L. Rosin. «Measuring Corner Properties». In: *Computer Vision and Image Understanding* **73**.2 (1 feb. 1999), pp. 291–307. ISSN: 1077-3142. DOI: 10.1006/cviu.1998.0719.

[33] Edward Rosten e Tom Drummond. «Machine Learning for High-Speed Corner Detection». In: *Computer Vision – ECCV 2006*. A cura di Aleš Leonardis, Horst Bischof e Axel Pinz. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 430–443. ISBN: 978-3-540-33833-8. DOI: 10.1007/11744023_34.

[34] Edward Rosten, Reid Porter e Tom Drummond. «Faster and Better: A Machine Learning Approach to Corner Detection». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* **32**.1 (gen. 2010), pp. 105–119. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2008.275.

[35] E. Rublee, V. Rabaud, K. Konolige e G. Bradski. «ORB: An efficient alternative to SIFT or SURF». In: *2011 International Conference on Computer Vision*. Nov. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[36] Cihan Topal, Cüneyt Akınlar e Yakup Genç. «Edge Drawing: A Heuristic Approach to Robust Real-Time Edge Detection». In: *2010 20th International Conference on Pattern Recognition*. ISSN: 1051-4651. Ago. 2010, pp. 2424–2427. DOI: 10.1109/ICPR.2010.593.

[37] Bala R. Vatti. «A Generic Solution to Polygon Clipping». In: *Commun. ACM* **35**.7 (lug. 1992), pp. 56–63. ISSN: 0001-0782. DOI: 10.1145/129902.129906.

[38] W. Zeng e R. L. Church. «Finding shortest paths on real road networks: the case for A*». In: *International Journal of Geographical Information Science* **23**.4 (1 apr. 2009), pp. 531–543. ISSN: 1365-8816. DOI: 10.1080/13658810801949850.

[39] Lilian Zhang e Reinhard Koch. «An efficient and robust line segment matching approach based on LBD descriptor and pairwise geometric consistency». In: *Journal of Visual Communication and Image Representation* **24**.7 (1 ott. 2013), pp. 794–805. ISSN: 1047-3203. DOI: 10.1016/j.jvcir.2013.05.006.

# Ringraziamenti

Ringrazio il Prof. Luigi Di Stefano per l'opportunità di partecipare al progetto che ha portato alla nascita di SAFFIRE e tutti i consigli durante la sua evoluzione.

Grazie anche a Martino che ha posto le basi per SAFFIRE e mi ha seguito durante tutto il suo sviluppo, ma anche a Nicoletta, Mariano, Fabricio, Lucrezia, Diego, Leonardo e Gregorio che mi hanno accolto in azienda e nel loro team permettendomi di avvicinarmi al mondo del lavoro con consigli e suggerimenti per migliorare le mie conoscenze.

Cristian, Nicola, Marco e Matteo. Grazie per avermi accompagnato anche in questi ultimi due anni come amici, colleghi e coinquilini. Anche se l'avventura universitaria è finita, e con questa un intero periodo della nostra vita, che sia l'inizio per un nuovo viaggio.

Il ringraziamento più importante va ai miei genitori che mi hanno supportato e lasciato seguire le mie passioni. Romeo, grazie per farmi sempre compagnia quando torno a casa e per avermi aspettato anche se non capivi dove andassi durante la settimana.