

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica

Sviluppo di un sistema di misura dell'accoppiamento tra componenti automobilistici mediante sistemi di visione

Progettazione di Visual Googles per uso industriale: approcci, errori e strategie di risoluzione

Candidato:
Rachele Campana

Relatore:
Prof.Paolo Castellini

Correlatore:
Dott.ssa Vittoria Medici

Anno Accademico 2025-2026



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica

Sviluppo di un sistema di misura dell'accoppiamento tra componenti automobilistici mediante sistemi di visione

Progettazione di Visual Googles per uso industriale: approcci, errori e strategie di risoluzione

Candidato:
Rachele Campana

Relatore:
Prof.Paolo Castellini

Correlatore:
Dott.ssa Vittoria Medici

Anno Accademico 2025-2026

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica
Via Brecce Bianche – 60131 Ancona (AN), Italy

刀の打ち方を知ってるか？
刀はな、叩いて叩いて叩き上げて、不純物や余分なものを飛ばし、
鋼の純度を高め、最強の刀を作るんだ。

泣いていい　逃げてもいい
ただ諦めるな
信じるんだ
地獄のような鍛錬に耐えた日々
お前は必ず報われる
極限まで叩き上げ
誰よりも最強の刃になれ！
一つのことを極めろ！

諦めるな!

辛くても苦しくても
楽な方へ逃げるな

Do you know what a process of forging a sword is? When making a sword, you hammer and hammer to get rid of impurities and anything else you don't need; the greater the purity of the blade, the more durable it becomes.

It's alright to cry or go off by yourself, just don't ever give up, believe in yourself. You endure all that hellish training, you'll be rewarded for that without question, work to forge yourself like a tempered blade and become the most resilient of all, focus on what you do best.

Don't ever give up! Even if it's painful, even if it's agonizing, don't try to take the easy way out.

Ringraziamenti

Desidero esprimere la mia profonda gratitudine al Professor Paolo Castellini e alla Dottoressa Vittoria Medici, per la disponibilità, la guida e la pazienza dimostrate durante la stesura di questo lavoro. Grazie alla loro guida e supervisione è stato possibile realizzare questo progetto, arricchendo notevolmente il mio percorso formativo e fornendomi strumenti preziosi sia dal punto di vista accademico che personale.

Ancona, Ottobre 2026

Rachele Campana

Sommario

Il presente lavoro di tesi si inserisce nel contesto dell'automazione industriale per l'accoppiamento di componenti meccanici mediante sistemi di visione artificiale, focalizzandosi in particolare sulla progettazione e sull'integrazione del modulo telecamera.

Nella fase iniziale dello studio, si è perseguito lo sviluppo e la traduzione in ambiente Python di algoritmi per l'estrazione delle feature, prendendo come riferimento modelli preesistenti in ambiente MATLAB. Tuttavia, l'approccio convenzionale ha riscontrato forti limiti esecutivi: in presenza di vetture inclinate, distorte o inquadrate da prospettive non perfettamente ortogonali e centrate, le tecniche classiche non garantivano risultati ottimizzati, compromettendo la stima dell'asse di simmetria e la successiva accuratezza del posizionamento.

Per superare tali criticità, si è reso necessario un riorientamento metodologico verso l'impiego di Reti Neurali Convoluzionali (CNN). È stata quindi addestrata una CNN dedicata al riconoscimento robusto delle feature della vettura anche in condizioni geometriche non ideali, consentendo di determinare con precisione l'asse di simmetria e le coordinate del baricentro della figura identificata. Tali parametri costituiscono il riferimento visivo fondamentale per il corretto orientamento e il posizionamento automatizzato del logo della vettura.

I risultati ottenuti dimostrano come il passaggio alle architetture convoluzionali conferisca al sistema di visione la tolleranza prospettica e la robustezza necessarie per operare efficacemente nelle reali condizioni operative della linea di assemblaggio.

Indice

1 Introduzione ai sistemi di visione: Visual Goggles **1**
1.1 Applicazione dei Visual Goggles in ambito industriale 5

2 Sviluppo e validazione di algoritmi per feature **9**
2.1 Approccio tramite IA 12
2.2 Approccio tramite Computer Vision Classica 20
2.2.1 Template Matching e Feature Matching 22
2.2.2 Multi-Scale Template Matching 30

3 Sperimentazione da algoritmi preesistenti **35**
3.1 Implementazione in Python ed analisi strutturale del codice 52

4 Stima dell'asse di simmetria **59**
4.1 Flip and Match in MATLAB 59
4.1.1 Costruzione del codice in ambiente MATLAB 64
4.2 Flip and Match in Python 66
4.3 feature matching e asse di simmetria 73
4.4 ottimizzazione per un calcolo rigoroso 80
4.4.1 Analisi del codice 81

5 Reti neurali **89**
5.1 Introduzione alle reti neurali 89
5.2 Raccolta dati e generazione del dataset 93
5.2.1 Training della rete e risultati ottenuti 100

6 Conclusioni **121**

Elenco delle figure

1.1 Visual Goggles [1] 2

2.1 Problema del parallasse 9
2.2 Risultato vista ortogonale 11
2.3 Template Matching [11] 24
2.4 Confronto degli algoritmi di template matching per la complessità computazionale basata sulla misura del match [11] 26
2.5 Esito sperimentale 28
2.6 Esito sperimentale con keypoints 28
2.7 Riferimento dato al sistema 29
2.8 Corrispondenza trovate dal sistema 29
2.9 Mappe di aspettativa [10] 33
2.10 Mappe di aspettativa del BBS e SDS per una gaussiana 2D nel caso di rotazione [10] 33
2.11 Evidenza sperimentale del Template Matching 34

3.1 Esempio di rilevamento angoli [20] 39
3.2 originale [20] 43
3.3 originale ruotato e scalato [20] 43
3.4 matched points candidati[20] 45
3.5 matching points [20] 46
3.6 confronto [20] 46
3.7 immagine di allenamento 49
3.8 immagine di richiesta 49
3.9 risultato con ORB 49
3.10 Corrispondenze trovate 51
3.11 Corrispondenze finali 51
3.12 Confronto finale 58

4.1 rilevamento di una coppia di punti [23] 61
4.2 Mappa di simmetria [23] 63
4.3 Risultato in Python 68
4.4 Risultato migliorato in Python 69
4.5 Risultato vettura distorta 70
4.6 Risultato vettura inclinata 2 70
4.7 Risultato vettura inclinata 71

4.8 Risultato Rettificazione Proiettiva 72
4.9 Risultato 2 Rettificazione Proiettiva 72
4.10 Risultato 3 Rettificazione Prospettica 72
4.11 Risultato 4 Rettificazione Prospettica 73
4.12 Risultato 5 Rettificazione Prospettica 73
4.13 Risultato codici combinati 77
4.14 Risultato 2 codici combinati 77
4.16 input e risultato pe vettura distorta 78
4.15 Risultato 3 codici combinati 78
4.17 input e riusltato per vettura frontale 79
4.18 input e risultato per vettura molto distorta 79
4.19 Risultato dell'asse e heat map 88
4.20 Risultato dell'asse e edge map 88

5.1 Rete neurale [12] 90
5.2 Rete neurale artificiale[12] 91
5.3 Modello in ambiente Blender 94
5.4 Posizione Camera in Blender 95
5.5 Keypoint 95
5.6 keypoint 99
5.7 Dataset 100
5.8 Coordinate (esempio foto 32) 100
5.9 Fase di addestramento 112
5.10 Analisi delle previsioni del modello 116

Elenco delle tabelle

3.1 Tabella criteri [20] 41
3.2 Rilevatori tabella [20] 42
3.3 Descrittori tabella [20] 42

4.1 Corrispondenze tra operazioni MATLAB e Python 66

Capitolo 1

Introduzione ai sistemi di visione: Visual Goggles

Negli ultimi anni, la diffusione di dispositivi digitali ha portato a nuove opportunità nel campo dell'industria dell'interazione Uomo-Macchina. Nel campo automobilistico, dove i sistemi complessi di hardware e software installati sui veicoli moderni sono sempre più importanti per gli utenti finali, la disponibilità di telecamere economiche e miniaturizzate ha permesso lo sviluppo di interfacce utente naturali basate sulla visione artificiale, aprendo a nuove opportunità nell'Interazione Uomo-Veicolo.

La visione artificiale consente di eseguire misure estremamente precise, ripetibili e affidabili su un oggetto. Per questo, nei moderni processi industriali i sistemi di visione artificiale vengono sempre più utilizzati trovando applicazione in quasi tutti i settori. Tra i fronti di evoluzione troviamo

- le telecamere, sempre più performanti dal punto di vista della velocità e del numero di pixel
- tecnologie di visione 3D rappresentano un settore in crescita in alcune nicchie di mercato come quella del "pick and place" di oggetti particolari

Attualmente il problema di queste tecnologie non è tanto l'acquisizione dell'immagine quanto l'elaborazione successiva, che richiede processori sempre più veloci e performanti abbinati a software molto evoluti per il riconoscimento nello spazio tridimensionale.

Gli Smart glasses, o Visual Goggles, sono un tipo di dispositivo indossabile sul viso , rivolto dall'utente, destinati a rivoluzionare il modo in cui interagiamo con le informazioni e con l'ambiente circostante poichè raggiungono l'obiettivo originale di permettere una visione più chiara oltre a funzionare come un computer. Sono dispositivi informatici da testa a forma di montatura che integrano un display, sensori e un processore connesso wireless, permettendo all'utilizzatore di percepire informazioni digitali sovrapposte o adiacenti al campo visivo naturale mantenendo le mani libere. A differenza di smartphone o tablet, gli smartglass forniscono informazioni senza costringere l'utente a distogliere lo sguardo dall'ambiente immediato, rendendoli particolarmente rilevanti per applicazioni in cui la consapevolezza situazionale continua e il funzionamento a mani libere sono essenziali. La tecnologia è nata dalla



Figura 1.1: Visual Goggles [1]

ricerca nei display montati sulla testata e sul calcolo indossabile negli anni '90 ed è arrivata al mercato commerciale con prodotti come Google Glass nel 2013, seguita da generazioni successive di produttori industriali e di consumo. Gli smartglass occupano uno spazio di design tra due estremi: montature minimaliste che mostrano una piccola sovrapposizione informativa nel campo periferico, e visori di realtà aumentata completamente immersivi che riempiono un ampio campo visivo con immagini renderizzate. I compromessi ingegneristici tra capacità di display, durata della batteria, peso e accettabilità sociale definiscono il panorama del prodotto. Gli smartglass hanno applicazioni in una vasta gamma di ambiti professionali e di consumo, tra cui:

- Assemblaggio industriale e manutenzione con istruzioni di sovrapposizione passo dopo passo
- Sovrapposizione di navigazione chirurgica e imaging medico nelle sale operatorie
- Ritiro degli ordini in magazzino e logistica guidati da segnali visivi
- Ingegneria del servizio sul campo con collaborazione video da esperti a distanza
- Assistenza personale al consumatore con informazioni contestuali generate dall'IA ambientale [5]

La Figura 1 1.1 mostra un esempio di occhiali intelligenti, che forniscono le informazioni desiderate agli utenti tramite un display sotto forma di occhiali esterni o binocoli. Supportano tecnologie di comunicazione wireless, come Bluetooth e Wi-Fi, e possono essere utilizzate per cercare e condividere informazioni in tempo reale tramite una connessione internet. Inoltre, fornendo una funzione di tracciamento della posizione tramite il Sistema di Posizionamento Globale (GPS), è possibile sviluppare varie applicazioni basate sulle informazioni di localizzazione. Come interfaccia per la comunicazione tra smart glasses e utenti, viene utilizzato un pulsante touch o un metodo di elaborazione dei comandi in linguaggio naturale basato sul riconoscimento vocale. Utilizzando una fotocamera montata davanti al dispositivo, è possibile acquisire fotografie o dati video dell'ambiente circostante in tempo reale.

Gli occhiali intelligenti sono una tecnologia basata su HMD ottici (OHMD), che consistono in oggetti di plastica posizionati all'altezza degli occhi, permettendo all'utente di visualizzare un mondo digitale online, un mondo offline e il mondo fisico . A differenza degli smartphone e di altri dispositivi indossabili, gli occhiali intelligenti

permettono agli utenti di svolgere compiti senza uno sforzo fisico significativo; Ad esempio, non è necessario usare ripetutamente le mani o gli occhi per interagire con un occhiale intelligente . Inoltre, gli smart glasses vengono utilizzati in vari modi, incluso lo sviluppo di applicazioni. Esempi includono la visualizzazione delle informazioni, l'analisi dei dati delle immagini, l'elaborazione dei dati, la navigazione, la trasmissione e la condivisione delle informazioni, e il rilevamento e l'avviso del rischio.

Non sono semplicemente uno schermo da indossare; sono un sistema complesso di componenti interconnessi progettato per acquisire, elaborare e proiettare informazioni. L'architettura fondamentale può essere suddivisa in diversi sottosistemi chiave che lavorano in sinergia:

- Il motore ottico: la proiezione del mondo digitale. Il componente più critico e determinante di qualsiasi paio di occhiali intelligenti è il sistema ottico. Questa è la tecnologia responsabile della visualizzazione delle immagini: ciò si ottiene combinando un micro-display e una serie di guide d'onda o combinatori. Il processo inizia spesso con un minuscolo micro-display, come un pannello a cristalli liquidi su silicio (LCoS), un micro-display a diodo organico a emissione di luce (OLED) o uno scanner laser MEMS (Micro-Electro-Mechanical System). Questi display sono incredibilmente piccoli, spesso delle dimensioni di una gomma da cancellare o anche più piccoli, ma sono in grado di generare immagini ad alta risoluzione.

  L'immagine generata viene quindi diretta in una guida d'onda ottica. Si può immaginare una guida d'onda come un pezzo di vetro o plastica trasparente che funge da condotto per la luce. Sfruttando i principi della diffrazione (attraverso reticoli di diffrazione) o della riflessione, la guida d'onda "piega" la luce proveniente dal micro-display, incanalandola attraverso la lente e infine direttamente nell'occhio dell'utente. La magia di questa tecnologia sta nel fatto che la lente stessa rimane trasparente, consentendo all'utente di vedere il mondo reale, mentre l'immagine proiettata si sovrappone perfettamente ad essa. Questo crea l' effetto di realtà aumentata (AR), fondendo in modo impeccabile i contenuti digitali con la realtà fisica.

- Sensori: gli occhi e le orecchie degli occhiali Affinché gli occhiali intelligenti siano in grado di percepire il contesto e di interagire, si affidano a una serie di sensori che fungono da organi percettivi. Questa serie di sensori è ciò che differenzia un semplice display a sovrimpressione da un dispositivo indossabile veramente intelligente.

- Fotocamere: una o più fotocamere ad alta risoluzione catturano dati visivi dalla prospettiva dell'utente. Ciò consente funzionalità come la registrazione video in prima persona, l'acquisizione di foto e, soprattutto, la visione artificiale. Gli

occhiali possono analizzare questo flusso video per identificare oggetti, leggere testi, riconoscere volti e comprendere l'ambiente in 3D.

- Unità di misura inerziale (IMU): si tratta di una combinazione di accelerometri e giroscopi che traccia il movimento, la rotazione e l'orientamento precisi della testa dell'utente. Questo è fondamentale per stabilizzare la sovrapposizione digitale, garantendo che un oggetto virtuale rimanga "ancorato" su un tavolo reale anche se si muove la testa.

- Sensori di profondità: alcuni modelli avanzati includono sensori specializzati come telecamere time-of-flight (ToF) o proiettori di luce strutturata. Questi sensori mappano attivamente l'ambiente proiettando schemi di luce infrarossa e misurando il tempo o la distorsione necessari al ritorno della luce. Questo crea una mappa 3D precisa dell'ambiente circostante, essenziale per posizionare oggetti digitali che possano occludersi correttamente dietro oggetti reali.

- Sensori di luce ambientale: questi regolano automaticamente la luminosità del display proiettato in base alle condizioni di illuminazione, garantendo una visibilità ottimale sia all'interno che all'esterno in presenza di luce solare.

- Telecamere di eye tracking: i sensori a infrarossi possono tracciare lo sguardo dell'utente e la dilatazione della pupilla. Ciò consente un controllo intuitivo (ad esempio, la selezione di un elemento guardandolo), un'autenticazione biometrica avanzata e un rendering più efficiente (rendering dei dettagli più elevati solo dove l'utente sta guardando).

- Il cervello integrato: elaborazione dei dati. I dati grezzi provenienti da tutti questi sensori costituiscono un flusso continuo di informazioni che devono essere elaborate in tempo reale. Questo è il compito del System on a Chip (SoC), l'unità di elaborazione centrale degli occhiali. Non si tratta di un processore standard per smartphone; è un chip altamente specializzato progettato per un'estrema efficienza energetica e gestione termica. Questo SoC in genere contiene una Central Processing Unit (CPU) per le attività generali, una Graphics Processing Unit (GPU) per il rendering delle immagini e un Digital Signal Processor (DSP) per la gestione dei sensori. [1]

Molti fattori determinano il successo di un motore di ricerca visiva, ma tutto si riduce alla visione artificiale e al riconoscimento delle immagini. La visione artificiale è ciò che permette a macchine come telefoni cellulari, tablet e laptop di "vedere". Ma semplicemente vedere un oggetto non significa che lo capisca. Ed è qui che entra in gioco il potere del riconoscimento delle immagini. Il riconoscimento delle immagini permette ai computer di identificare, comprendere e categorizzare oggetti specifici all'interno di un'immagine. Per farlo con precisione, l'IA visiva richiede addestramento. Ad esempio, si possono fornire a un motore di ricerca immagini migliaia di sedie di diversi stili, dimensioni e colori. Il motore AI elabora e impara

da ogni pixel di ogni immagine per poter affinare ed espandere la sua comprensione delle diverse sedie nel tempo. . Quindi, quando nel sistema viene caricata l'immagine di una sedia, un motore di ricerca visiva può confrontare ogni pixel con ogni sedia che ha mai visto, e sa automaticamente di cosa si tratta. [2]

1.1 Applicazione dei Visual Goggles in ambito industriale

I cambiamenti rapidi e profondi che hanno colpito l'industria manifatturiera hanno ovviamente influenzato anche il personale manifatturiero. In particolare, per gli operatori nel reparto industriale ci sono margini di errore inferiori, metodi di lavoro in cambiamento e nuove tecnologie. Questo ha portato sia a nuove richieste che a nuove possibilità. Attualmente non ci sono indicazioni che questo processo rallenterà. Le previsioni su ciò che l'industria 4.0 può portare indicano piuttosto che la tendenza sarà l'opposto e il cambiamento accelererà. In questo scenario guidato dalla tecnologia, è importante che il lavoro e le condizioni di lavoro degli operatori siano considerati come parte dello sviluppo generale della produzione futura. Con l'aumento dei dati di produzione disponibili e delle variazioni di prodotto nel reparto industriale, aumenta anche la pressione sugli operatori di assemblaggio per gestire questo vasto flusso di informazioni, superando ciò che è umanamente possibile. Gli operatori di assemblaggio quindi necessitano di supporto decisionale che semplifichi queste informazioni. La realtà aumentata (AR) può fornire un supporto efficace agli operatori di assemblaggio per aiutarli a visualizzare le informazioni e a inserirle nel loro contesto. Gli smart glasses sono un tipo di visore per la realtà virtuale (HMD) con un grande potenziale negli ambienti di lavoro dell'Industria 4.0, dove gli operatori in fabbrica devono essere in grado di ricevere informazioni critiche in modo tempestivo, accessibile e sicuro per essere il più produttivi possibile. Gli occhiali intelligenti raccolgono dati da una rete wireless e li proiettano su un piccolo schermo davanti agli occhi dell'utente.

Utilizzando applicazioni software di realtà aumentata e assistita dotate di occhiali intelligenti, i lavoratori del reparto ricevono informazioni fondamentali, incluse istruzioni complesse, documenti di riferimento, diagrammi, checklist, dati di macchine, immagini e video, direttamente nella loro linea di vista. La disponibilità e l'accessibilità di informazioni vitali favoriscono una maggiore produttività complessiva, permettendo ai lavoratori di svolgere il proprio lavoro con qualità al primo intervento e risolvere problemi mantenendo le mani libere. Anche se molti vedono il potenziale dell'AR nell'industria, le aziende stanno ancora in gran parte eseguendo 'proof of concept' e le tecnologie abilitanti per l'AR (occhiali e cuffie) non sono ancora sufficientemente robuste per un uso continuo in ambienti di lavoro difficili (l'AR industriale presenta vincoli specifici rispetto ad altre aree applicative che possono portare a barriere applicative; gli aspetti che hanno riscontrato particolarmente

carenti sono l'affidabilità, il rispetto della sicurezza sul lavoro e l'accuratezza delle sovrapposizioni). [4] [7]

In questa tesi, si propone uno studio di tecniche di visione artificiale in ambito industriale, nello specifico nel settore automobilistico, che non si limita al preciso accoppiamento dei componenti automobilistici, ma si estende alla definizione del posizionamento e del centramento del logo e degli elementi grafici in rilievo sulla carrozzeria.

Per elaborare un sistema di visione in grado di posizionare virtualmente il logo in maniera tale da consentire all'operatore che indossa l'occhiale (dotato di telecamera e proiettore) il posizionamento corretto del logo o di eventuali elementi grafici, siamo partiti dalla progettazione della telecamera con algoritmi dedicati all'individuazione di due gruppi di feature con l'obiettivo di determinare il baricentro complessivo necessario per la centratura del logo.

In seguito ci siamo interessati allo sviluppo delle Reti Neurali Convoluzionali (CNN) e la crescita dell'Internet delle Cose (Internet of Things o IoT), che hanno reso gli occhiali intelligenti una piattaforma potenzialmente rivoluzionaria per l'assistenza visiva. Questa ricerca introduce un nuovo metodo per il rilevamento e l'assistenza alla navigazione in tempo reale utilizzando la connessione IoT e algoritmi CNN. La soluzione proposta migliora la capacità degli utenti di vedere e comprendere il proprio ambiente combinando una piccola fotocamera con occhiali intelligenti. Gli occhiali intelligenti aiutano l'utente identificando gli oggetti nel suo campo visivo utilizzando algoritmi di riconoscimento degli oggetti basati su CNN e fornendo informazioni contestuali. Utilizzando la connessione IoT, il sistema consente una facile integrazione con i servizi di navigazione, offrendo agli utenti assistenza e indicazioni in tempo reale in nuove aree. [8] Il sottosistema di visualizzazione degli smart glasses affronta vincoli fondamentali non presenti negli schermi mobili: il percorso ottico deve proiettare un'immagine su o attraverso una piccola guida d'onda o un combinatore montato alla periferia di una cornice standard per occhiali, mantenendo la messa a fuoco a una distanza focale confortevole. Le ottiche a guida d'onda che utilizzano reticoli diffrattivi o olografici sono l'approccio dominante negli occhiali intelligenti moderni per realtà aumentata; convogliano la luce da un microdisplay, tipicamente un microLED o un proiettore a cristallo liquido su silicio (LCoS), attraverso la lente verso l'occhio senza richiedere un elemento combinatore separato visibile dall'esterno. Gli smartglass incorporano una varietà di sensori che consentono sia la percezione ambientale sia l'input dell'utente. I sensori di tracciamento oculare monitorano la direzione dello sguardo, abilitando interfacce di selezione basate sullo sguardo e informando il rendering del display per concentrare le risorse computazionali sulla regione di messa a fuoco attuale. Le unità di misura inerziale tracciano l'orientamento della testa, che deve essere noto con precisione per mantenere una registrazione stabile della sovrapposizione mentre l'utente si muove [5]. Più che del motore ottico, in questa tesi ci si focalizzerà sulla progettazione di una telecamera rivolta verso l'esterno, che cattura immagini e video per compiti di visione artificiale, inclusi generalmente il

riconoscimento di oggetti, la scansione di codici a barre e la mappatura geometrica necessaria per una registrazione accurata della sovrapposizione in realtà aumentata.

Capitolo 2

Sviluppo e validazione di algoritmi per l'estrazione e l'analisi delle feature

In una prima fase del lavoro, si è adottata un'analisi che partisse da dati reali raccolti sul campo tramite un esperimento, anziché da modelli matematici teorici o simulazioni al computer, basata sull'acquisizione diretta di un'immagine relativa alla vista posteriore di una vettura. Per contenere la complessità dello studio e agevolare la formulazione del modello, il problema è stato inizialmente formulato secondo un'ipotesi di lavoro bidimensionale (2D). L'ipotesi di modellazione bidimensionale (2D) potrà essere estesa a livello universale laddove il sistema di proiezione e il sistema di acquisizione visiva condividano un'architettura coassiale: la motivazione principale riguarda l'assenza di parallasse (o l'assenza di offset geometrico) tra il punto da cui viene proiettata la luce e il punto da cui l'immagine viene acquisita. Nel momento in cui la fotocamera e il proiettore condividono lo stesso asse ottico (configurazione coassiale), le proiezioni geometriche si semplificano drasticamente. Quando un proiettore e una telecamera sono separati da una certa distanza (triangolazione), la luce proiettata e il punto di vista della telecamera formano un triangolo. L'immagine catturata dalla telecamera subisce una distorsione prospettica e di profondità dovuta all'angolo tra i due assi ottici e per capire dove si trova un punto nella scena, serve un modello 3D basato sulla triangolazione .Quando sono coassiali il centro di proiezione e il centro di presa della fotocamera coincidono esattamente nello spazio. In questo

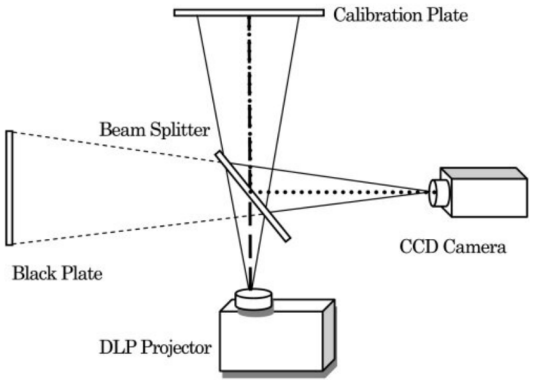


Figura 2.1: Problema del parallasse

modo ogni raggio di luce proiettato percorre lo stesso identico cammino ottico del raggio di luce ripreso dalla telecamera e di conseguenza, non c'è parallasse: le variazioni di profondità non introducono uno spostamento laterale o una distorsione relativa tra la luce proiettata e l'immagine catturata. Se proiettore e fotocamera non sono coassiali, la geometria tridimensionale della vettura crea delle zone d'ombra: il proiettore illumina una superficie che la telecamera non vede, o viceversa (occlusioni). In un sistema coassiale invece la telecamera vede esattamente ciò che il proiettore illumina; non si verificano in questo modo zone d'ombra o punti ciechi dovuti alla tridimensionalità dell'oggetto. L'intero dominio di analisi risponde quindi come una superficie piana bidimensionale proiettata.

Se proiettore e telecamera hanno la stessa risoluzione e sono perfettamente allineati e coassiali, il pixel (x, y) proiettato dal proiettore corrisponde sempre e univocamente allo stesso pixel (x, y) visto dalla telecamera (corrispondenza biunivoca $(1 : 1)$ tra il piano immagine del proiettore e quello del sensore) indipendentemente da quanto sia vicina o lontana la vettura, da cui l'immagine catturata dalla telecamera è già perfettamente sovrapponibile alla matrice del proiettore. Il problema si riduce matematicamente a una trasformazione piano-piano (2D) senza dover gestire matrici 3D complesse o triangolazioni volumetriche.

Si assume pertanto l'ipotesi semplificativa per cui la superficie posteriore del veicolo sia considerata planare. Sulla base di tale approssimazione, l'immagine acquisita viene trattata come un dominio bidimensionale, dal quale procedere con l'estrazione e l'individuazione dei principali gruppi di feature caratteristici della vettura: ai fini della generalizzazione del sistema di visione e per garantirne l'invarianza rispetto a differenti modelli di autoveicoli, occorre superare l'approccio basato su misure geometriche rigide. Misure puntuali, come la distanza specifica tra i gruppi ottici o l'ampiezza del portellone posteriore per il posizionamento del logo, rappresentano infatti parametri strettamente vincolati alla singola vettura e, pertanto, non generalizzabili. Anziché calcolare il punto medio a partire da coordinate puntuali rigide e fisse, l'approccio proposto si fonda sull'estrazione e il raggruppamento di cluster di feature (gruppi di feature), identificati mediante caratteristiche cromatiche e geometriche intrinseche. Sfruttando la naturale simmetria assiale dell'autoveicolo, la determinazione della coordinata di mezzeria tra i gruppi di feature omologhi si riduce all'individuazione del loro asse di simmetria relativo, definendo così la retta di allineamento lungo la quale verrà successivamente posizionato il logo.

Un ulteriore vincolo operativo riguarda la prospettiva di acquisizione del veicolo, la quale non si presenta necessariamente ortogonale e centrata, bensì frequentemente soggetta ad inclinazioni o rotazioni prospettiche. In tale scenario, l'individuazione dell'asse di simmetria a partire da gruppi di feature (punti di interesse) speculari costituisce un approccio estremamente solido, in quanto consente di ridurre la complessità dell'elaborazione dell'intera scena a un insieme contenuto di elementi geometrici chiave in maniera tale da lavorare sintetizzando la visione globale in una rappresentazione geometrica essenziale ed efficace.



Figura 2.2: Risultato vista ortogonale

L'algoritmo, implementato in ambiente Python mediante l'IDE Visual Studio Code, ha prodotto risultati ampiamente soddisfacenti nell'elaborazione della vista frontale della vettura 2.2.Al contrario, l'introduzione di distorsioni prospettiche e angolazioni variabili nella vista ha evidenziato i limiti funzionali del metodo, compromettendone le prestazioni e generando un'identificazione errata dell'asse di simmetria a causa della perdita della simmetria geometrica sul piano immagine.

Infatti se l'immagine ha una prospettiva o un'angolazione variabile, dobbiamo ragionare in termini di geometria epipolare o, più semplicemente, di trasformazioni rigide nel piano se l'inclinazione è solo una rotazione 2D. Se l'oggetto è ruotato nell'immagine, l'asse di simmetria non sarà perfettamente verticale o orizzontale

- Se la vista è frontale ma ruotata l'approccio dei punti medi funziona perfettamente. La retta risultante sarà inclinata dell'angolo corretto
- Se c'è distorsione prospettica,ovvero se l'immagine è scattata "di lato", la simmetria geometrica pura si perde. In quel caso, si dovrebbe prima calcolare la matrice di omografia per "raddrizzare" l'oggetto (rectification) e poi procedere.

Questo perchè se scattiamo una foto a un oggetto simmetrico (come una bottiglia, un volto o un'automobile), trovare il suo asse di simmetria nell'immagine sembra un compito facile: basta trovare la linea che divide l'oggetto in due metà speculari.

Tuttavia, il modo in cui la fotocamera è posizionata rispetto all'oggetto cambia completamente le regole della geometria. Si distinguono due scenari principali, dal più semplice al più complesso: si immagini di aver posizionato una foto su un tavolo e di averla semplicemente ruotata di 30 gradi. Quello che succede è che la foto è vista perfettamente frontalmente ma l'oggetto al suo interno non è dritto. L'asse di simmetria non sarà più una linea verticale perfetta , ma una linea inclinata ;Tuttavia la geometria dell'oggetto non cambia affatto, ovvero le distanze e le proporzioni nell'immagine rimangono identiche a quelle reali. In termini tecnici si parla di trasformazione rigida nel piano: il calcolo è molto semplice e se si usa il metodo dei punti medi (ovvero si trovano i punti centrali tra i bordi opposti dell'oggetto e li si uniscono con una retta), si troverà subito l'asse di simmetria. La retta che si ottiene sarà semplicemente inclinata dell'angolo giusto. Si immagini ora di fotografare

una macchina non di fronte, ma partendo da uno dei fari posteriori : per effetto della prospettiva, la parte dell'oggetto più vicina alla fotocamera apparirà molto più grande, mentre la parte più lontana apparirà più piccola e stretta.

In questa situazione la prospettiva "distrugge" la simmetria geometrica sul piano 2D della foto e le due metà dell'oggetto non sono più speculari sulla foto, anche se lo sono nella realtà. Se piegassimo la foto a metà lungo l'asse visivo, i due lati non combacerebbero affatto. I metodi classici 2D (come i punti medi) qui falliscono del tutto e bisogna ricorrere alla geometria epipolare (la geometria che regola come il mondo 3D si proietta sulle fotocamere).

Per trovare la simmetria, la fotocamera deve "matematicamente annullare" la prospettiva, si calcola una trasformazione matematica chiamata matrice di omografia per fare la cosiddetta rettificazione (rectification). In pratica, si ricostruisce come apparirebbe l'oggetto se lo guardassimo di fronte. Solo dopo aver "raddrizzato" l'immagine sul piano frontale si può applicare la ricerca dell'asse di simmetria.

Il limite fondamentale dell'approccio iniziale risiede nella sensibilità del sistema alle variazioni angolari e prospettiche dell'inquadratura. In uno scenario reale, una volta integrata la tecnologia negli smart glasses, imporre all'operatore di mantenere una posizione rigida o un'angolazione prefissata non è praticabile, in quanto introdurrebbe vincoli operativi persino più gravosi dei processi attuali.

Per superare tale criticità, si è sviluppato un approccio volto a svincolare il sistema di acquisizione e proiezione dalla specifica direzione di visualizzazione. L'obiettivo è rendere l'architettura di visione indipendente dall'inclinazione della vista: individuando i due gruppi di feature come oggetti geometricamente noti, è possibile calcolare l'asse di simmetria 'analitico' dello spazio reale, superando i limiti della mera simmetria bidimensionale estratta dall'immagine acquisita.

2.1 Progettazione telecamera: approccio tramite IA

Riconoscere oggetti specifici per nome (come "fanali") e poi usarli per calcolare la geometria richiede un passaggio di Object Detection o Instance Segmentation. Per riconoscere i "fanali", non si possono usare semplici algoritmi matematici sui pixel; si ha bisogno di un modello che "sappia" cos'è un fanale: il modello più utilizzato è YOLO (You Only Look Once).

Il rilevamento degli oggetti, compito centrale nella visione artificiale, ha registrato notevoli progressi negli ultimi anni grazie allo sviluppo continuo di algoritmi più efficienti e accurati. Una delle scoperte più significative in questo campo è il framework YOLO, un algoritmo pionieristico di rilevamento di oggetti a uno stadio che ha attirato ampia attenzione per la sua capacità di ottenere rilevamento in tempo reale con alta precisione.

I rilevatori di oggetti a singolo stadio rappresentano una classe di modelli progettati per rilevare oggetti in un'immagine attraverso un singolo passaggio avanti della rete neurale . A differenza dei rivelatori a due stadi, che prevedono passaggi separati per

la proposta di regione e la classificazione degli oggetti, i rivelatori a singolo stadio svolgono entrambi i compiti simultaneamente, semplificando il processo di rilevamento . Questo approccio ha guadagnato popolarità grazie alla sua semplicità, efficienza computazionale e capacità di elaborazione in tempo reale, rendendolo particolarmente adatto per applicazioni che richiedono un'inferenza rapida, come veicoli autonomi e sistemi di sorveglianza. Un tipico rivelatore a singolo stadio prevede direttamente le probabilità di classe degli oggetti e le scatole di delimitazione senza la necessità di una rete di proposta di regione: nei sistemi tradizionali (two-stage), una prima rete analizza l'immagine per cercare "punti d'interesse" e proporre delle regioni (region proposal), e una seconda rete classifica cosa c'è in quelle regioni.

Un rilevatore a singolo stadio elimina del tutto la rete di proposta: prende l'intera immagine e la divide in una griglia e per ogni cella della griglia, la rete calcola in un unico calcolo dove si trova l'oggetto(le coordinate del rettangolo (bounding box)) e che cosa è l'oggetto (probabilità delle varie classi: ad esempio 85% auto,100% persona,...)

Il grande vantaggio è che saltando la fase di proposta delle regioni, l'algoritmo riduce drasticamente i calcoli matematici. Il risultato è una velocità di elaborazione elevatissima, rendendo questi modelli ideali per l'uso in tempo reale (come video a molti FPS o sistemi integrati su smart glasses). Le caratteristiche che rendono i rivelatori a singolo stadio uno strumento fondamentale nel rilevamento degli oggetti, offrendo un equilibrio tra velocità e precisione supportando al contempo una vasta gamma di applicazioni reali sono

- per adattarsi a scale e rapporti d'aspetto diversi degli oggetti, i rivelatori a singolo stadio utilizzano scatole di ancoraggio (note anche come scatole predefinite) . Queste scatole predefinite permettono alla rete di effettuare aggiustamenti per adattarsi meglio a oggetti di diverse forme e dimensioni.

- i rivelatori a singolo stadio sono composti da due componenti principali: una testa di regressione per prevedere le coordinate della scatola delimitante e una testa di classificazione per determinare le classi di oggetti. Entrambe le teste operano sulle feature map estratte dall'immagine di input.

- l'obiettivo di addestramento del modello consiste nel minimizzare una combinazione di tre perdite: perdita di localizzazione (per previsioni accurate nella scatola di confine), perdita di fiducia (per presenza o assenza di oggetti) e perdita di classificazione (per accuratezza delle etichette di classe).

- dopo aver previsto più bounding box, viene applicata la soppressione non massima (Non-Maximum Suppression (NMS)) per filtrare le previsioni a bassa fiducia e sovrapposte. Questo garantisce che vengano mantenute solo le scatole delimitanti più sicure e non ridondanti.

- impiegano un'architettura di rete neurale unificata che prevede simultaneamente le scatole di delimitazione e le probabilità di classe, eliminando la necessità di una fase di proposta regionale separata
- efficienza computazionale, che li rende adatti all'elaborazione in tempo reale. L'assenza di un passaggio separato di proposta regionale riduce il sovraccarico computazionale, permettendo una rilevazione rapida
- i rivelatori a singolo stadio trovano applicazioni in vari ambiti, inclusi veicoli autonomi, sorveglianza, robotica e riconoscimento di oggetti in immagini e video. La loro velocità e semplicità li rendono particolarmente adatti in scenari in cui il rilevamento in tempo reale è cruciale.

Il framework YOLO, ideato da Joseph Redmon, ha rivoluzionato il rilevamento in tempo reale degli oggetti introducendo un approccio a griglia per prevedere simultaneamente le bounding box e le probabilità di classe . Questo design innovativo facilita un processo di rilevamento altamente efficiente, rendendo YOLO particolarmente adatto per applicazioni che richiedono prestazioni in tempo reale. Dalla sua nascita, l'architettura YOLO ha subito un'evoluzione continua, con ogni versione, dalla YOLOv1 all'ultima YOLOv9, che ha introdotto miglioramenti in accuratezza, velocità ed efficienza . Progressi significativi includono l'introduzione di scatole di ancoraggio, che migliorano la capacità del modello di rilevare oggetti di forme e dimensioni diverse, oltre a nuove funzioni di perdita volte a ottimizzare le prestazioni. Inoltre, ogni iterazione YOLO presenta backbone ottimizzati che contribuiscono a tempi di elaborazione più rapidi e a migliori capacità di rilevamento. Ogni revisione ha fatto progressi nel minimizzare la latenza e massimizzare l'accuratezza del rilevamento come obiettivi centrali nello sviluppo continuo delle tecnologie di rilevamento degli oggetti in tempo reale [6].

Per identificare rapidamente e con precisione gli oggetti, YOLO divide un'immagine in una griglia e prevede contemporaneamente sia le caselle di delimitazione sia le probabilità di classe. Le coordinate della scatola delimitante e le probabilità di classe sono generate da strati convoluzionali, a seguito dell'estrazione delle caratteristiche tramite una profonda Rete Neurale Convoluzionale (CNN). YOLO migliora il rilevamento di oggetti di dimensioni variabili utilizzando scatole di ancoraggio su più scale. Le rilevazioni finali vengono perfezionate utilizzando la Non-Maximum Suppression (NMS), che filtra le previsioni ridondanti e a bassa fiducia, rendendo YOLO un metodo altamente efficiente e affidabile per il rilevamento degli oggetti. Ogni variante YOLO introduce innovazioni distinte volte a ottimizzare sia la velocità che la precisione. Ad esempio, YOLOv4 e YOLOv5 hanno integrato backbones avanzati e funzioni di perdita come Complete Intersection over Union (CIoU) per migliorare la localizzazione degli oggetti. I tre componenti principali dell'architettura YOLO, ovvero dorsale, collo e testa, subiscono modifiche significative nelle sue varianti per migliorare le prestazioni:

- Backbone: responsabile dell'estrazione delle funzionalità dai dati di input , la backbone è tipicamente una CNN pre-addestrata su grandi dataset come ImageNet. Le backbone comuni nelle varianti YOLO includono ResNet50, ResNet101 e CSPDarkNet53.

- Neck: il manico elabora e affina ulteriormente le feature map generate dalla spina dorsale. Spesso impiega tecniche come le Reti a Piramide di Caratteristiche (FPN) e i Moduli di Attenzione Spaziale (SAM) per migliorare la rappresentazione delle caratteristiche.

- Head: la testa processa le caratteristiche fuse dal manico per prevedere le scatole di delimitazione e le probabilità di classe. La testa di YOLO utilizza tipicamente scatole di ancoraggio multiscala, garantendo una rilevazione efficace degli oggetti a diverse scale .

Nelle applicazioni industriali, YOLO è diventato uno dei modelli di rilevamento di oggetti in tempo reale più utilizzati grazie alla sua alta velocità di elaborazione e alle capacità efficienti di identificazione degli oggetti. L'architettura a singolo stadio di YOLO gli consente di rilevare e classificare oggetti in un unico passaggio attraverso la rete neurale, rendendolo particolarmente adatto a ambienti che richiedono decisioni rapide, come linee di produzione, controllo qualità automatizzato e rilevamento di anomalie. La sua adattabilità a una vasta gamma di compiti in diversi settori, dalla lavorazione alimentare all'edilizia, ha reso YOLO uno strumento versatile. Nella produzione , YOLO viene utilizzato per migliorare la precisione e l'efficienza dei sistemi automatizzati. Che si tratti di rilevare difetti nei prodotti o monitorare la sicurezza in tempo reale, YOLO contribuisce a risultati di qualità superiore e a costi operativi ridotti.

Uno degli usi più efficaci di YOLO nell'industria è nella produzione, dove le sue capacità di rilevamento in tempo reale sono sfruttate per il controllo automatico della qualità, il rilevamento di difetti e l'ottimizzazione della produzione. La velocità e la precisione di YOLO permettono ai produttori di individuare rapidamente i problemi sulla linea di produzione, riducendo i tempi di inattività e minimizzando i risultati difettosi.

Nel nostro caso specifico è stato pensato all'applicazione di YOLOv8 / YOLOv9:

- YOLOv8: ottimizzazione per l'efficienza delle risorse. YOLOv8 ha ulteriormente perfezionato l'architettura, concentrandosi sull'ottenere una migliore efficienza delle risorse senza sacrificare la precisione. Uno dei progressi più notevoli di YOLOv8 è stata la sua capacità di scalare in modo efficiente tra diverse configurazioni hardware, rendendolo uno strumento flessibile sia per dispositivi a basso consumo che per ambienti di calcolo ad alte prestazioni. YOLOv8 ha semplificato il processo di addestramento e introdotto ottimizzazioni che hanno migliorato la sua capacità di generalizzare su una vasta gamma di compiti di rilevamento oggetti. YOLOv8 ha inoltre migliorato la gestione degli scenari

di rilevamento multi-oggetto, dove più oggetti di dimensioni e forme diverse appaiono in un unico fotogramma. Nonostante questi miglioramenti, YOLOv8 ha affrontato sfide in architetture più profonde, in particolare per questioni di convergenza. Questi problemi sono nati dalla difficoltà del modello nel bilanciare la complessità computazionale richiesta per reti più profonde con la necessità di inferenza in tempo reale. Di conseguenza, le prestazioni di YOLOv8 su dataset complessi, specialmente quelli che coinvolgono oggetti piccoli, sovrapposti o occlusi, non sono sempre state coerenti.

- L'introduzione di YOLOv9 , YOLO-NAS ha segnato un cambiamento significativo nell'evoluzione del framework YOLO. Con l'espansione più ampia dei modelli di rilevamento oggetti nelle applicazioni reali, è cresciuta la necessità di soluzioni in grado di bilanciare precisione ed efficienza computazionale, specialmente su dispositivi edge con potenza di calcolo limitata. In risposta alle sfide incontrate dai modelli precedenti, YOLOv9 ha introdotto diverse tecniche rivoluzionarie progettate per migliorare il flusso di gradiente, gestire l'accumulo di errori e facilitare una migliore convergenza durante l'addestramento. Queste innovazioni hanno permesso a YOLOv9 di estendere la sua applicabilità a una gamma più ampia di compiti reali di rilevamento oggetti.

Dopo YOLOv9, YOLOv10 e YOLOv11 hanno portato ulteriori miglioramenti al framework YOLO, facendo notevoli progressi sia in velocità che in precisione. YOLOv10 ha introdotto innovazioni rivoluzionarie che hanno migliorato prestazioni ed efficienza, costruendo i punti di forza dei suoi predecessori e spingendo i confini del rilevamento degli oggetti in tempo reale. Inoltre, i meccanismi di attenzione migliorati del modello hanno permesso una migliore rilevazione di oggetti piccoli e occlusi, permettendo a YOLOv10 di superare le versioni precedenti in compiti come il rilevamento della maschera facciale e applicazioni di veicoli autonomi. La sua capacità di bilanciare efficienza computazionale con precisione di rilevamento ha stabilito un nuovo standard di precisione nel riconoscere gli oggetti.

Con l'introduzione di YOLOv9, YOLOv10 si utilizzano dataset ben consolidati come Roboflow o CVAT, che devono contenere la classe "headlight" (fanale), oppure, in mancanza di un dataset pronto, si dovrebbe "etichettare" un centinaio di foto di auto per addestrare il modello [6].

Il processo si divide in tre fasi:

- Detection: il modello individua i fanali nell'immagine e restituisce dei "Bounding Boxes" (rettangoli) o delle "Masks" (poligoni precisi)
- Identificazione delle Coppie: se il modello trova due fanali, li classifichiamo come L (sinistro) e R (destro) in base alla loro posizione relativa nel piano.
- Calcolo Geometrico: usiamo i centri di questi fanali per tracciare l'asse

Se l'auto è inclinata o vista di tre quarti, la distanza percepita tra i fanali cambia a causa della prospettiva

- Asse in 2D: se si vuole l'asse di simmetria dell'immagine (come appare a schermo), il calcolo dei centri è sufficiente.
- Asse nel mondo reale (3D): se si vuole l'asse di simmetria reale dell'auto nonostante la prospettiva, si dovrebbe usare un modello di Pose Estimation (che identifica i "keypoints" dell'auto) o calcolare la matrice di proiezione.

Tuttavia l'approccio tramite il "Deep Learning" (IA) non sempre è consigliato nella progettazione rispetto alla Computer Vision classica: il Deep Learning (DL) viene utilizzato nel campo dell'elaborazione delle immagini digitali per risolvere problemi difficili (ad esempio colorazione delle immagini, classificazione, segmentazione e rilevamento). I metodi DL come le Reti Neurali Convoluzionali (CNN) migliorano soprattutto le prestazioni predittive utilizzando grandi quantità di dati e abbondanti risorse di calcolo, spingendo i limiti di ciò che era possibile. Problemi che una volta si pensava fossero irrisolvibili ora vengono risolti con una precisione superiore a quella umana [9].

L'analisi descrittiva consiste nel definire un modello matematico comprensibile che descriva il fenomeno che vogliamo osservare. Questo comporta la raccolta di dati su un processo, la formulazione di ipotesi sui modelli presenti nei dati e la convalida di queste ipotesi confrontando il risultato dei modelli descrittivi che creiamo con il risultato reale . Creare tali modelli è comunque rischioso perché c'è sempre il rischio di variabili non modellate che scienziati e ingegneri non includono per ignoranza o per non aver compreso fenomeni complessi, nascosti o non intuitivi. L'analisi predittiva prevede la scoperta di regole che stanno alla base di un fenomeno e la creazione di un modello predittivo che minimizzi l'errore tra il risultato reale e quello previsto, considerando tutti i possibili fattori interferenti . Il Machine Learning rifiuta il paradigma di programmazione tradizionale, sostituendo l'analisi del problema con un framework di addestramento in cui al sistema vengono forniti un gran numero di pattern di addestramento (insiemi di input per i quali gli output desiderati sono noti) che esso impara e usa per calcolare nuovi pattern . Il DL è un sottoinsieme del machine learning. Il DL si basa in gran parte sulle Reti Neurali Artificiali (ANN), un paradigma di calcolo ispirato al funzionamento del cervello umano. Come il cervello umano, è composto da molte celle di calcolo o 'neuroni' che eseguono ciascuno un'operazione semplice e interagiscono tra loro per prendere una decisione. Il Deep Learning riguarda tutto l'apprendimento o il 'credit assignment' attraverso molti strati di una rete neurale in modo accurato, efficiente e senza supervisione ed è di recente interesse grazie ai progressi nella tecnologia hardware L'auto-organizzazione e lo sfruttamento delle interazioni tra piccole unità si sono dimostrati più efficaci rispetto al controllo centrale, soprattutto per modelli di processi complessi e non lineari, poiché è possibile ottenere maggiore tolleranza ai guasti e adattabilità a nuovi dati.

I rapidi progressi nel DL e i miglioramenti nelle capacità dei dispositivi, inclusi potenza di calcolo, capacità di memoria, consumo energetico, risoluzione del sensore d'immagine e ottiche, hanno migliorato le prestazioni e la convenienza economica accelerando ulteriormente la diffusione delle applicazioni basate sulla visione. Rispetto alle tecniche tradizionali di CV, il DL permette agli ingegneri CV di ottenere maggiore accuratezza in attività come classificazione delle immagini, segmentazione semantica, rilevamento degli oggetti e Localizzazione e Mappatura Simultanea (SLAM). Poiché le reti neurali utilizzate nel DL vengono addestrate invece che programmate, le applicazioni che usano questo approccio richiedono spesso meno analisi esperta e perfezionamenti e sfruttano la quantità enorme di dati video disponibili nei sistemi odierni. Il DL offre anche una flessibilità superiore perché i modelli e i framework CNN possono essere riaddestrati usando un dataset personalizzato per qualsiasi caso d'uso, al contrario degli algoritmi CV, che tendono ad essere più specifici per dominio.

Prendendo come esempio il problema del rilevamento degli oggetti su un robot mobile, possiamo confrontare i due tipi di algoritmi per la visione artificiale: l'approccio tradizionale è usare tecniche CV consolidate come i descrittori di caratteristiche (Scale Invariant Feature Transform (SIFT),Speeded Up Robust Features (SURF),Binary Robust Independent Elementary Features (BRIEF), ecc. . .) per il rilevamento degli oggetti. Prima dell'emergere del DL, si effettuava un passaggio chiamato estrazione delle caratteristiche per compiti come la classificazione delle immagini. Le caratteristiche sono piccole aree "interessanti", descrittive o informative nelle immagini. Diversi algoritmi CV, come il rilevamento dei bordi, il rilevamento degli angoli o la segmentazione per soglia, possono essere coinvolti in questo passaggio. Quante più caratteristiche possibili vengono estratte dalle immagini e queste caratteristiche formano una definizione (nota come bag-of-words) di ciascuna classe di oggetti. Nella fase di deploy, queste definizioni vengono cercate in altre immagini. Se un numero significativo di caratteristiche di un bag-of-words si trova in un'altra immagine, l'immagine viene classificata come contenente quell'oggetto specifico (ad es. sedia, cavallo, ecc.). La difficoltà di questo approccio tradizionale è che è necessario scegliere quali caratteristiche sono importanti in ciascuna immagine. Man mano che il numero di classi da classificare aumenta, l'estrazione delle caratteristiche diventa sempre più complicata. Spetta al giudizio dell'ingegnere CV e a un lungo processo di tentativi ed errori decidere quali caratteristiche descrivono meglio le diverse classi di oggetti. Inoltre, ogni definizione di caratteristiche richiede di gestire una miriade di parametri, che devono tutti essere regolati finemente dall'ingegnere CV. Il DL ha introdotto il concetto di apprendimento end-to-end, dove alla macchina viene semplicemente fornito un dataset di immagini annotate con le classi di oggetti presenti in ciascuna immagine. Così, un modello DL viene "addestrato" sui dati forniti, dove le reti neurali scoprono i modelli sottostanti nelle classi di immagini e determinano automaticamente le caratteristiche più descrittive e salienti rispetto a ciascuna classe di oggetti. È ormai ben noto che le CNN performano molto meglio degli algoritmi tradizionali, sebbene con compromessi rispetto ai requisiti di calcolo e ai tempi di

addestramento. Con tutti gli approcci all'avanguardia nella CV che adottano questa metodologia, il flusso di lavoro dell'ingegnere CV è cambiato drasticamente, e la conoscenza e l'esperienza nell'estrarre caratteristiche fatte a mano è stata sostituita dalla conoscenza e dall'esperienza nel lavorare con architetture di deep learning.

Tuttavia ci sono delle sfide introdotte dal DL. Gli ultimi approcci di DL possono raggiungere una precisione significativamente migliore ma questo salto ha un costo in termini di miliardi di operazioni matematiche aggiuntive e un aumento della richiesta di potenza di calcolo. Il DL richiede queste risorse di calcolo per l'addestramento e in misura minore per l'inferenza. È essenziale avere hardware dedicato (ad esempio GPU ad alte prestazioni e TPU per l'addestramento e piattaforme AI accelerate come VPU per l'inferenza) per gli sviluppatori di AI. I risultati dell'elaborazione visiva usando il DL dipendono anche dalla risoluzione delle immagini. Ottenere prestazioni adeguate nella classificazione degli oggetti, per esempio, richiede immagini o video ad alta risoluzione, con il conseguente aumento della quantità di dati che deve essere elaborata, memorizzata e trasferita. La risoluzione delle immagini è particolarmente importante per applicazioni in cui è necessario rilevare e classificare oggetti a distanza, ad esempio nei filmati delle telecamere di sicurezza. Le tecniche di riduzione dei fotogrammi discusse in precedenza, come l'uso delle caratteristiche SIFT o del flusso ottico per oggetti in movimento, per identificare prima una regione di interesse, sono utili sia rispetto alla risoluzione delle immagini sia rispetto alla riduzione del tempo e dei dati necessari per l'addestramento.

Inoltre il DL ha bisogno di big data. Spesso sono necessari milioni di record di dati. Ad esempio, il PASCAL VOC Dataset consiste in 500.000 immagini con 20 categorie di oggetti , ImageNet consiste in 1,5 milioni di immagini con 1000 categorie di oggetti e Microsoft Common Objects in Context (COCO) consiste in 2,5 milioni di immagini con 91 categorie di oggetti. Quando grandi dataset o potenti strutture di calcolo non sono disponibili, tornano utili i metodi tradizionali. Un'altra difficoltà risiede nel fatto che addestrare una CNN richiede moltissimo tempo. A seconda della disponibilità dell'hardware di calcolo, l'addestramento può richiedere ore o giorni. Inoltre, addestrare per una determinata applicazione spesso richiede molte iterazioni, visto che implica tentativi ed errori con diversi parametri di addestramento. Tuttavia, va detto che compiti più semplici e specifici rispetto alla classificazione generale delle immagini non richiederanno così tanti dati (nell'ordine delle centinaia o migliaia anziché milioni). Questo resta comunque un quantitativo considerevole di dati e le tecniche di CV vengono spesso utilizzate per aumentare i dati di addestramento tramite data augmentation o per ridurre i dati a un tipo specifico di feature attraverso altri passaggi di pre-elaborazione. La pre-elaborazione comporta la trasformazione dei dati (di solito con tecniche tradizionali di CV) per permettere che relazioni e pattern siano interpretati più facilmente prima di addestrare il modello. La data augmentation, utilizzata come vedremo nell'approccio tramite reti neurali, è un'attività comune di pre-elaborazione usata quando c'è poco materiale d'addestramento. Può includere rotazioni casuali, spostamenti, inclinazioni, ecc. sulle

immagini del nostro set di addestramento per aumentare effettivamente il numero di immagini. Ci sono molti altri problemi in evoluzione nella CV come robotica, realtà aumentata, cucitura automatica dei panorami, realtà virtuale, modellazione 3D, stima del movimento, stabilizzazione video, motion capture, elaborazione video e comprensione della scena, che non possono essere implementati facilmente in modo differenziabile con il deep learning, ma devono essere risolti usando altre tecniche "tradizionali". Il DL eccelle nel risolvere problemi di classificazione chiusi, in cui un'ampia gamma di segnali potenziali deve essere mappata su un numero limitato di categorie, a condizione che ci siano sufficienti dati disponibili e che il set di test somigli molto al set di training. Tuttavia, le deviazioni da queste assunzioni possono causare problemi ed è fondamentale riconoscere i problemi in cui il DL non è bravo. Il deep learning deve essere integrato con altre tecniche se vogliamo raggiungere l'intelligenza artificiale generale . Tra i problemi in cui il DL funziona male, c'è la limitata capacità degli algoritmi DL di apprendere relazioni visive, cioè capire se più oggetti in un'immagine siano uguali o diversi. Questa limitazione è dovuta al fatto che i meccanismi di feedback, inclusi attenzione e raggruppamento percettivo, possono essere i componenti computazionali chiave per realizzare il ragionamento visivo astratto. Inoltre i modelli di ML trovano difficile gestire i priori, cioè non tutto può essere appreso dai dati, quindi alcuni priori devono essere inseriti nei modelli. Le soluzioni legate alla CV 3D hanno infatti bisogno di forti priori per funzionare bene, ad esempio la modellazione 3D basata su immagini richiede informazioni di continuità, silhouette e illuminazione[9].

In conclusione, l'adozione del modello basato su intelligenza artificiale risulta inattuabile nell'ambito del presente studio. La metodologia presenta marcate criticità operative, riconducibili ai requisiti di potenza di calcolo sproporzionati rispetto alla dotazione hardware disponibile per l'elaborazione dei dati, alla necessità di un supporto software superiore rispetto a quello compatibile con la nostra infrastruttura sperimentale, e alle complessità analitiche inerenti al processo di risoluzione.

2.2 Progettazione telecamera: approccio tramite Computer Vision Classica

Gli approcci tradizionali, si dimostrano utili nel migliorare le prestazioni nelle attività di CV, grazie alle seguenti caratteristiche :

- Scale Invariant Feature Transform (SIFT)
- Speeded Up Robust Features (SURF)
- Features from Accelerated Segment Test (FAST)
- Trasformate di Hough
- Geometric hashing

I descrittori di caratteristiche come SIFT e SURF sono generalmente combinati con algoritmi di classificazione di machine learning tradizionali, come Support Vector Machines e K Nearest Neighbours, per risolvere i problemi di CV menzionati prima. Il DL a volte è esagerato, poiché spesso le tecniche tradizionali di CV possono risolvere un problema in modo molto più efficiente e con meno righe di codice rispetto al DL. Algoritmi come SIFT e anche semplici algoritmi di sogliatura dei colori e conteggio dei pixel non sono specifici per le classi, cioè sono molto generali e funzionano allo stesso modo per qualsiasi immagine. Al contrario, le caratteristiche apprese da una rete neurale profonda sono specifiche del dataset di addestramento che, se non ben costruito, probabilmente non funzioneranno bene con immagini diverse dal set di addestramento. Pertanto, SIFT e altri algoritmi sono spesso usati per applicazioni come cucitura di immagini/ricostruzione di mesh 3D che non richiedono conoscenze specifiche delle classi. Questi compiti si sono dimostrati realizzabili addestrando grandi dataset, tuttavia questo richiede un enorme sforzo di ricerca e non è pratico affrontare questo sforzo per un'applicazione chiusa. Bisogna usare il buon senso quando si sceglie quale strada prendere per una determinata applicazione di CV. Ad esempio, classificare due classi di prodotto su un nastro trasportatore in una linea di assemblaggio, uno con vernice rossa e uno con vernice blu. Una rete neurale profonda funzionerebbe, dato che si possono raccogliere abbastanza dati per l'addestramento. Tuttavia, lo stesso può essere ottenuto usando una semplice sogliatura dei colori. Alcuni problemi possono essere affrontati con tecniche più semplici e più veloci.Inoltre se il dataset di addestramento è limitato, la macchina potrebbe adattarsi troppo ai dati di training e non essere in grado di generalizzare per il compito richiesto. Sarebbe troppo difficile modificare manualmente i parametri del modello perché una CNN ha milioni di parametri al suo interno, ciascuno con complesse interrelazioni. In questo senso, i modelli di DL sono stati criticati come una scatola nera . La CV tradizionale ha piena trasparenza e permette di giudicare se la soluzione funzionerà anche fuori dall'ambiente di addestramento. L'ingegnere di CV può avere intuizioni su un problema che può trasferire al proprio algoritmo e se qualcosa fallisce, i parametri possono essere modificati per funzionare meglio su una gamma più ampia di immagini. Oggi, le tecniche tradizionali sono usate quando il problema può essere semplificato in modo da essere implementato su microcontrollori a basso costo o per limitare il problema per le tecniche di deep learning evidenziando certe caratteristiche nei dati, aumentando i dati o aiutando nell'annotazione del dataset . Infine, ci sono molti problemi più complessi in CV come robotica , realtà aumentata , cucitura automatica di panorami , realtà virtuale , modellazione 3D , stima del movimento , stabilizzazione video , motion capture , elaborazione video e comprensione delle scene , che non possono essere semplicemente implementati in modo differenziabile con il deep learning, ma traggono vantaggio dalle soluzioni che utilizzano tecniche "tradizionali"[9].

2.2.1 Template Matching e Feature Matching

Nella maggior parte dei problemi di visione artificiale e analisi delle immagini, è necessario definire una misura di somiglianza tra due o più oggetti o immagini diversi presentati al sistema. Un approccio molto comune per rilevare oggetti e trovare la misura di somiglianza è il template matching. Il template matching cerca di rispondere a una delle domande più basilari su un'immagine: se c'è un certo oggetto in una data immagine, e dove si trova . Il template è una descrizione di quell'oggetto (e quindi è un'immagine a sua volta), ed è usato per cercare nell'immagine calcolando una misura di differenza tra il template e tutte le possibili porzioni dell'immagine che potrebbero corrispondere al template. Se uno di questi passaggi produce una piccola differenza, allora viene considerato come un possibile caso dell'oggetto.

Il template matching è una tecnica di visione artificiale ad alto livello che permette l'identificazione delle parti di un'immagine che corrispondono al modello immagine fornito ed è un aspetto fondamentale di molte questioni nella visione artificiale, incluso il riconoscimento di oggetti o scene, la determinazione della struttura 3D da più immagini, la corrispondenza stereo e il tracciamento del movimento. La distinzione tra i diversi primitivi di corrispondenza è probabilmente la differenza più evidente tra i vari algoritmi di matching. Il template matching è una tecnica per trovare aree di un'immagine che corrispondono (o sono simili) a un'immagine template che richiede due immagini:

- Immagine sorgente (I): l'immagine in cui ci aspettiamo di trovare una corrispondenza con l'immagine template
- Immagine template (T): il frammento d'immagine che sarà confrontato con l'immagine template

L'obiettivo finale è rilevare la tecnica migliore per l'area di matching più alta.

Le misure di corrispondenza e il template matching nelle immagini bidimensionali o tridimensionali richiedono una misura di corrispondenza tra due immagini che indichi il grado di somiglianza o differenza tra di esse . La scelta dell'algoritmo di matching dipende dalle immagini di riferimento e dal template. Un metodo particolare classifica gli approcci di template o image matching in approcci basati sul template o sull'area e approcci basati sulle caratteristiche (feature)[11].

Approccio basato sulle caratteristiche (feature matching)[11]

Un approccio basato sulle caratteristiche è appropriato quando sia le immagini di riferimento sia il template contengono più corrispondenze rispetto a caratteristiche e punti di controllo. In questo caso, le caratteristiche includono punti, curve o un modello di superficie per eseguire il template matching. In questa categoria, l'obiettivo finale è localizzare le connessioni a coppie tra l'immagine target o cosiddetta di riferimento e l'immagine template usando relazioni spaziali o caratteristiche. In

questo approccio, le relazioni spaziali, i descrittori invarianti, le piramidi, le wavelet e i metodi di rilassamento giocano un ruolo importante nell'estrazione delle misure di corrispondenza.

Approccio basato sull'area[11]

I metodi basati sull'area, solitamente conosciuti come metodi di correlazione o template matching si basano su un algoritmo combinato di rilevamento delle caratteristiche e corrispondenza delle caratteristiche. Questo metodo funziona molto bene quando i template non hanno caratteristiche forti con un'immagine, poiché operano direttamente sui valori dei pixel. Le corrispondenze vengono misurate utilizzando i valori di intensità sia dell'immagine che del template. I punteggi di corrispondenza vengono estratti calcolando le differenze al quadrato nelle intensità fisse, metodi basati sulla correzione, metodi di ottimizzazione e informazioni mutue. In alcuni problemi di template matching, l'abbinamento diretto tra il template e le immagini target è impossibile. Pertanto, il valore proprio e lo spazio proprio delle immagini fornite sono utilizzati nel matching dei template. Questi valori forniscono i dettagli necessari per confrontare le immagini in varie condizioni, come illuminazione, contrasto dei colori o pose di matching adeguate. Ad esempio, si cerca un esemplare in una determinata immagine; lo spazio proprio può consistere in template dell'esemplare in diverse posizioni rispetto alla telecamera usando varie condizioni di illuminazione o espressioni. È possibile che l'immagine da confrontare sia occlusa da un oggetto o presenti problemi legati al movimento e diventi ambigua. Una delle possibili soluzioni a questo è dividere il template in più sotto-immagini e effettuare il matching su queste sotto-immagini.

L'approccio basato sull'area sarà proprio quello che utilizzeremo nel nostro studio a causa delle possibili diverse condizioni di illuminazione, contrasti cromatici , luci e ombre delle immagini usate come riferimento-obiettivo, poichè senza un confronto specifico di pixel potrebbero mandare in errore il sistema.

Na¨ıve Template Matching[11]

La corrispondenza di template è uno dei metodi base per estrarre un elemento dato identico al template dall'immagine target. In questo approccio, con o senza ridimensionamento (di solito senza ridimensionamento), l'immagine target viene scansionata dal template e vengono calcolate le misure di somiglianza. Infine, le posizioni con le somiglianze più forti vengono identificate come potenziali posizioni del pattern. La Figura 2.3 illustra un esempio di template matching. Come mostrato, i sotto-immagini sono considerati come modelli, e viene eseguito il metodo di matching semplice per estrarre i modelli e i pattern dalle immagini target. Poiché le sotto-immagini provengono originariamente dall'immagine target senza alcuna scala o manipolazione, questo algoritmo di matching diretto funziona in modo efficiente. Uno dei parametri di errore utilizzati per calcolare le differenze tra le immagini target

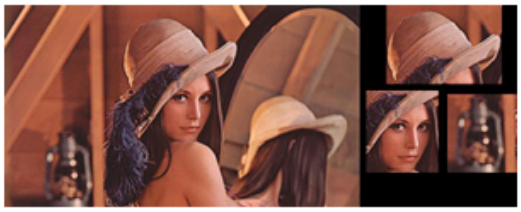


Figura 2.3: Template Matching [11]

e quelle modello è la somma delle differenze al quadrato

$$SSD = \sum_{x,y}[f(x,y) - t(x-u, y-v)]^2$$

Matching delle Immagini tramite Correlazione [11]

In questo classico metodo di template matching, si misura la similarità tra il target e il template. A differenza del semplice algoritmo di template matching, il target e il template potrebbero avere intensità dell'immagine o livelli di rumore diversi. Tuttavia, queste immagini devono essere allineate. La misura di similarità usata in questo approccio si basa sulla correlazione tra il target e il template.

Cross-Correlation[11]

Nel trattamento delle immagini, la cross-correlazione è una misura della somiglianza tra due immagini quando le immagini hanno dimensioni diverse. Scorrendo la prima immagine (template) sulla seconda immagine (target), viene misurata la correlazione tra le due immagini. Il metodo di cross-correlazione è simile per natura alla convoluzione di due funzioni. Inoltre, la cross-correlazione di un dato segnale o immagine con se stesso è chiamata auto-correlazione.

Normalized Cross-Correlation [11]

La cross-correlazione normalizzata è una versione aggiornata dell'approccio di cross-correlazione che è stata migliorata per due motivi principali:

- I risultati della cross-correlazione normalizzata sono invarianti ai cambiamenti globali di luminosità e, teoricamente, le variazioni di intensità dell'immagine hanno meno effetto su questa metrica rispetto al metodo classico.
- Il valore di correlazione ottenuto per una data finestra di scorrimento è normalizzato nell'intervallo $[-1, +1]$. Pertanto, i valori di cross-correlazione normalizzata per due immagini simili tendono a $+1$, mentre quelli per due immagini completamente diverse tendono a -1.

La cross-correlazione normalizzata (NCC) è ampiamente utilizzata come misura operativa di somiglianza per problemi di template matching.

Matching Filters[11]

I metodi di corrispondenza dei modelli con ricerca estesa possono essere spiegati come una sorta di filtro di corrispondenza, dove il modello $f(x, y)$ deve essere rilevato in un'area data di $s(x, y)$. Questo filtro è sviluppato per produrre una risposta massima quando una regione dell'area di ricerca corrisponde al modello.

Algoritmi di Rilevamento della Similarità Sequenziale[11]

Gli algoritmi di rilevamento della similarità sequenziale (SSDA) sono un'alternativa più efficiente ai metodi basati sulla correlazione, inclusi i filtri adattati per la registrazione traslazionale. La misura della corrispondenza viene calcolata indirettamente in base a un errore per i pixel corrispondenti in f e g nelle immagini in confronto a qualsiasi stadio del processo di registrazione .

Misure di distorsione [11]

Queste misure sono simili alle misure di distorsione usate per valutare le prestazioni delle tecniche di comprensione o ricostruzione delle immagini. Sono anche simili alla comprensione di immagini frattali o alla quantizzazione vettoriale (VQ), poiché il confronto tra immagini viene usato per definire le relative mappature e per selezionare il codice VQ. Un modello di spazio di funzioni per derivare un filtro di ricostruzione è ottimale quando le immagini originali e ricostruite sono il più simili possibile rispetto a una determinata misura di somiglianza.

Coefficiente di Dice[11]

Il Coefficiente di Dice (DC) è stato comunemente usato negli studi fMRI per valutare la somiglianza di due mappe spaziali. Il volume-overlap è una misura semplice per capire quanti dei voxel (volumetric pixel, ovvero pixel "tridimensionali") sopra la soglia di due mappe t si trovano nello stesso punto . A seconda del fattore di normalizzazione, ci sono diverse varianti della metrica di overlap; quella più comune è il Coefficiente di Dice mostrato qui sotto

$$DC = \frac{2 \times |A \cap B|}{|A| + |B|}$$

Metodi Efficienti di Template Matching [11]

La complessità di ciascun algoritmo di template matching dovrebbe essere calcolata. Questo calcolo spiega l'efficienza del metodo in termini di tempo. La Tabella 2.4 mostra la complessità computazionale delle misure di corrispondenza per alcuni algoritmi. La complessità fornita qui di seguito è definita in termini del numero di

Measure of Match	Operations				
	Multiplication	Division	Addition	Subtraction	Other
Normalized Cross-Correlation	3rc+1	1	3(rc-1)	0	1 sqrt
Cross Covariance	3rc+8	1	5rc-3	1	1 sqrt
Root Mean Square	rc+1	0	rc-1	rc	1 sqrt
Sum Absolute Valued Diffs	0	0	rc-1	rc	rc abs
Variance of the Differences	rc+2	1	2(rc-1)	rc+1	0
Stochastic Sign Change	r(c-1)	1	(r-1)(c-2)	rc	r(c-1) sgn
Deterministic Sign Change	r(c-1)	1	rc+(r-1)(c-2)	rc	r(c-1) sgn

Figura 2.4: Confronto degli algoritmi di template matching per la complessità computazionale basata sulla misura del match [11]

operazioni durante il calcolo della metrica di corrispondenza (cioè la cross-correlazione) tra due immagini con r righe e c colonne. Per confrontare i metodi, abbiamo assunto che il template f e l'immagine g siano immagini digitali (digitalizzate) rc. L'immagine data g viene estratta da ogni posizione in un'area di ricerca chiamata s, che ha r righe e c colonne. Il template matching viene eseguito mediante una ricerca esaustiva e la traslazione del template in ciascuna delle $(R-r+1)(C-c+1)$ posizioni nell'immagine. Successivamente, viene calcolata una misura di corrispondenza tra il template e il campione nella posizione di sovrapposizione. Il numero di operazioni richiesto per un determinato metodo di template matching è stato calcolato moltiplicando il numero di operazioni richieste per la dimensione dell'area di ricerca. Tuttavia, è possibile aumentare l'efficienza perché il template non cambia durante il template matching. È anche possibile utilizzare la ridondanza nel calcolo dei termini per misure di corrispondenza adiacenti per migliorare l'efficienza .

Nel presente studio sperimentale è stata adottata la tecnica del Template Matching tramite un algoritmo, implementato in ambiente Python, in grado di ricevere in ingresso un'immagine di riferimento raffigurante il "fanale tipo" (il template) ed effettuare una scansione traslazionale , dell'immagine della vettura,facendola scorrere lungo l'intera immagine dell'autoveicolo, identificando la regione che presenta il massimo grado di correlazione (o somiglianza).

Tuttavia una delle sfide nell'usare qualsiasi tecnica di template matching è scegliere una misura di corrispondenza che sia invariabile all'intensità e alla scala dell'immagine. Per superare questa sfida, alcuni scienziati hanno sperimentato con diverse scale di un template così come alcune versioni ruotate di un template. Tuttavia, questo può risolvere solo il problema della scalabilità del template matching. Le variazioni di intensità dell'immagine rimangono. Utilizzando la normalizzazione dell'intensità dell'immagine locale o globale, le variazioni di intensità in certi problemi possono essere risolte. Tuttavia, questa tecnica non è espandibile alla maggior parte dei problemi di template matching.

Invarianza del livello di intensità media

Per il template matching con livello di intensità media invariato, è stata sviluppata una misura di corrispondenza basata sulla varianza delle differenze pixel per pixel

tra il template e la sotto-immagine dell'immagine target.

Invarianza alla rotazione

Una rappresentazione del modello e della sotto-immagine invarianti alla rotazione e alla scala può essere generata semplicemente precedendo la procedura descritta nel paragrafo precedente con una trasformazione in coordinate polari. L'origine si trova al centro del modello o della sotto-immagine, e questo converte la rotazione in uno spostamento circolare. L'asse radiale,cioè l'asse che rappresenta la distanza dal centro dell'immagine (raggio r), nel sistema di coordinate polari viene poi deformato su una scala logaritmica, e la scala diventa traslazione. Quindi l'immagine viene convertita da coordinate cartesiane (x, y) a polari. L'asse r rappresenta la distanza dal centro, mentre l'asse θ rappresenta l'angolo. Una rotazione dell'immagine originale diventa un semplice scorrimento (shift) lungo l'asse degli angoli θ. Applicando una scala logaritmica all'asse del raggio r (ottenendo una trasformazione Log-Polare), anche la variazione di scala (ingrandimento o rimpicciolimento dell'immagine) si trasforma in una semplice traslazione (uno shift lineare lungo l'asse $\log r$). La magnitudine della DFT (Trasformata di Fourier discreta)viene quindi calcolata per avere una rappresentazione dell'immagine invariabile a rotazione e scala. Viene descritto un approccio simile in termini di mappatura conforme logaritmica complessa. L'immagine viene mappata su un piano complesso, dove ogni pixel è rappresentato matematicamente, come mostrato dall'equazione

$$z = x + jy$$

La mappatura logaritmica complessa che trasforma un'immagine da coordinate rettangolari a coordinate esponenziali polari è fornita nell'equazione seguente

$$w = \ln(z) = \ln(|z|) + j\theta_z$$

Rotazione e scala diventano traslazione nel dominio della trasformazione.

Come abbiamo detto in precedenza, il matching di punti chiave (Feature Matching) è il modo in cui la Computer Vision classica risolve il problema della "rigidità" del Template Matching. Invece di cercare l'intera "figurina" del fanale, Python cerca dei punti matematicamente unici (chiamati keypoints) come angoli, spigoli o variazioni brusche di contrasto che rimangono riconoscibili anche se l'oggetto è ruotato, rimpicciolito o visto di lato. Gli esiti sperimentali di questo approccio sono riportati in figura 2.5 e in figura 2.6 esplicitando i keypoints ottenuta dando al sistema come template di riferimento l'immagine del fanale da cui il sistema sviluppa il risultato tramite

- Detection: l'algoritmo (es. ORB o SIFT) analizza entrambe le immagini e trova centinaia di punti distintivi



Figura 2.5: Esito sperimentale



Figura 2.6: Esito sperimentale con keypoints

- Description: per ogni punto, crea una "carta d'identità" matematica (un vettore) che descrive l'area intorno a quel punto 2.6
- Matching: confronta le carte d'identità del template con quelle dell'immagine grande e unisce i punti che si somigliano di più.

Come evidenziato dai risultati ottenuti, l'approccio basato sui keypoint (feature matching), per quanto inizialmente ritenuto il più idoneo , è da scartare. Tale metodo non esegue una ricerca mirata della feature di riferimento per individuarne la corrispettiva struttura speculare; al contrario, correla i punti chiave dell'intera immagine con quelli estratti dal modello fornito. Di conseguenza, qualora il riferimento (sia esso un fanale) presenti singolarità visive simili a quelle di regioni estranee, l'algoritmo includerà nel calcolo anche queste corrispondenze "bugiarde" (outliers). L'inclusione di tali elementi altera l'estrazione delle caratteristiche rilevanti, conducendo infine a un'errata stima dell'asse di simmetria 2.6.

Tra le principali criticità riscontrate nel nostro caso figura l'estrazione di punti chiave in aree estranee al target, quali il lunotto o la targa. La presenza di gradienti di contrasto simili a quelli della feature di riferimento hanno generato falsi positivi determinando un sensibile errore nel calcolo dell'asse di simmetria. Si procede quindi con il template matching semplice, abbandonando i "punti chiave" (ORB) per usare un metodo a scorrimento (Sliding Window). In questo approccio, l'algoritmo esegue una scansione dell'immagine dell'autoveicolo traslando la regione di interesse (ROI) del fanale pixel per pixel, al fine di valutare la correlazione matematica tra le due aree. Tale metodo (visto in precedenza come metodo delle aree) garantisce un'elevata accuratezza nella localizzazione dell'elemento target, pur presentando una marcata rigidità geometrica: variazioni di scala o di inclinazione del fanale ne riducono drasticamente l'efficacia. Confrontando l'area quindi A differenza dei punti



Figura 2.7: Riferimento dato al sistema



Figura 2.8: Corrispondenza trovate dal sistema

chiave che guardano solo "angoli", questo metodo confronta tutta l'area del fanale (forma, colori, sfumature). La targa ha una forma rettangolare bianca e nera, mentre il fanale è rosso/trasparente con una trama interna: per l'algoritmo di scorrimento sono oggetti completamente diversi. Questo approccio funziona in maniera più rigida grazie al vincolo dei due massimi: invece di dire "trova tutto quello che somiglia", abbiamo detto: "Dimmi qual è il punto assolutamente più simile nella parte sinistra e il punto assolutamente più simile nella parte destra". Questo forza il sistema a scegliere solo i due candidati migliori (i fanali).

Invarianza alla scala

Una delle principali sfide nella generalizzazione delle tecniche di template matching è fornire un operatore invariante alla scala per poter estrarre il template dall'immagine target. Partendo dall'assunzione che gli oggetti target nell'immagine siano lontani tra loro e che lo sfondo sia uniforme, abbiamo visto la trasformata di Fourier Mellin (FM) che può essere utilizzata come rappresentazione o operatore del template e delle sotto-immagini in ogni posizione dell'immagine target prima di calcolare la misura di corrispondenza . La trasformata di Fourier Mellin comprende quattro passaggi, che sono i seguenti:

- Si calcola la Trasformata Discreta di Fourier (DFT) dell'immagine. La proprietà di traslazione circolare della DFT è una rappresentazione invariante alla traslazione centrale dell'immagine originale. Tuttavia, questo implica che i limiti della traslazione sono i bordi dell'immagine.
- L'immagine della magnitudine ricostruita dall'operazione DFT viene normalizzata rispetto al suo valore all'origine per rimuovere l'effetto moltiplicativo dei cambiamenti di scala nell'immagine originale.

- Gli assi del quadrante positivo dell'immagine normalizzata vengono deformati su una scala logaritmica. Questo converte la scala nell'immagine originale in traslazione.
- Viene calcolata la magnitudine DFT dell'immagine scalata logaritmicamente, fornendo una rappresentazione invariante alla scala dell'immagine originale. Le variazioni di scala tollerate dalla trasformata FM sono limitate dalla risoluzione dell'immagine e dalla dimensione della finestra dell'immagine.

2.2.2 Multi-Scale Template Matching

Nel nostro caso se un oggetto rigido/non rigido si muove nello spazio 3D, con uno sfondo variabile/invariabile e l'oggetto può subire deformazioni rigide/non rigide e occlusioni parziali, come la vettura del nostro studio, si è voluta studiare come soluzione al problema il Multi-Scale template Matching. Essendo la tecnica più cruciale nei compiti di template matching, la misura di similarità è stata studiata per decenni e si traduce in vari metodi, dal classico "somma delle differenze assolute" (SAD), alla "somma delle distanze al quadrato" (SSD) fino alle più recenti "best buddies similarity" (BBS) e "deformable diversity similarity" (DDIS) . Tuttavia, ci sono ancora diversi aspetti da migliorare: la maggior parte delle applicazioni reali preferisce mostrare i risultati del matching con box di dimensioni variabili per includere le regioni dell'oggetto anziché una dimensione fissa. Tuttavia, impostare parametri geometrici può comportare un aumento dei candidati da valutare, il che richiede una misura di similarità distintiva contro i cambiamenti di scala.

Il template matching di solito è denso e tutti i pixel/caratteristiche all'interno del template e di una finestra candidata vengono presi in considerazione per misurare la somiglianza anche quando alcune parti non sono desiderabili (ad esempio, occlusioni, cambiamenti di aspetto causati da deformazioni), per cui è necessario un metodo di misurazione della somiglianza in grado di gestire rumori e valori anomali. A causa delle possibili deformazioni rispetto al modello di riferimento, ci si aspetta che una buona misura di somiglianza sia indipendente dalla correlazione spaziale (ad esempio, quando l'oggetto all'interno di una finestra candidata è fortemente ruotato rispetto al modello, la correlazione spaziale tra il modello e il candidato nell'ordine di scansione a raster può diventare inaffidabile). Viene proposta la similarità diversificata scalabile (SDS) per affrontare i problemi sopra descritti. SDS può essere applicata con la finestra mobile multi-scala e non è limitata da alcun modello di deformazione parametrico specifico. Sia BBS che DDIS si concentrano nel risolvere il problema sfruttando le proprietà del vicino più vicino (NN). Ogni NN è definito da una coppia di patch tra il template e il target. In BBS, se e solo se ogni patch in una coppia di patch è il NN dell'altra, viene definito un match e il numero di questi match determina il punteggio BBS. DDIS migliora ulteriormente il BBS introducendo la diversità rilevante dei sottoinsiemi di patch tra il target e il template, il che rende BBS più robusto rispetto a ostruzioni e deformazioni. Anche

se questi metodi possono gestire le deformazioni all'interno di una finestra fino a un certo punto, ci sono limitazioni. SDS ha i seguenti due vantaggi

- Il primo è che SDS permette di misurare la somiglianza tra due insiemi di punti di dimensioni diverse, e l'entità del punteggio è robusta rispetto alla scala. Di solito, l'entità del punteggio DDIS o BBS cresce con l'aumentare della scala dell'insieme di punti, il che rende le finestre candidate più grandi più favorevoli da selezionare come risultati finali. Per ridurre questa ingiustizia, SDS introduce la diversità rilevante bidirezionale e penalizza i cambiamenti di scala per rendere possibile l'uso della finestra scorrevole multi-scala, e il punteggio può convergere alla scala corretta.

- Il secondo vantaggio di SDS è la sua robustezza alla rotazione intensa. Sia BBS che DDIS implicano un termine di distanza spaziale nella ricerca dei vicini più vicini (NN) o nel calcolo finale della somiglianza, il che comporta una limitazione: il NN di un punto deve essere spazialmente vicino. Questa limitazione è un forte vincolo che può effettivamente ridurre il numero di outlier, ma allo stesso tempo diminuisce la robustezza contro rotazioni intense. In questo caso, invece delle coordinate cartesiane, sfruttiamo l'angolo polare delle coordinate polari per il calcolo della distanza spaziale, il che elimina la limitazione imposta dal vincolo. Inoltre, l'informazione di rango dell'aspetto all'interno di un cerchio locale viene utilizzata per cercare i NN insieme all'aspetto locale, il che aiuta a trovare NN più affidabili e porta a un miglioramento significativo nei casi di rotazione intensa [10].

Il meccanismo del template matching è semplice: viene campionato un gran numero di finestre candidate nell'immagine target, seguito da una misura di somiglianza tra ogni finestra candidata e il template. Il punteggio di somiglianza gioca un ruolo fondamentale nel misurare la fiducia e nel distinguere il vero target dagli altri candidati. Le misure pronte all'uso più comuni sono metodi pixel-per-pixel come la somma delle differenze (SSD), la somma delle differenze assolute (SAD) e la correlazione incrociata normalizzata (NCC), grazie alla loro semplicità ed efficienza. Per affrontare i cambiamenti geometrici del target, molti lavori hanno considerato di estendere la fase di campionamento delle finestre con trasformazioni parametriche planari, come la traslazione, la trasformazione di similitudine , la trasformazione affine e la trasformazione proiettiva . Tuttavia, questi metodi di solito falliscono in deformazioni complesse perché il metodo di somiglianza pixel-per-pixel dipende dalle corrispondenze corrette tra i pixel nel template e nella finestra candidata, il che è molto limitato dai modelli geometrici planari. In ambienti non vincolati, per affrontare trasformazioni non rigide e altri tipi di disturbi, coinvolgere informazioni globali invece di informazioni locali pixel-per-pixel per progettare una similarità robusta è un elemento chiave. L'abbinamento degli istogrammi (HM) che misura principalmente la similarità tra due istogrammi di colore, non è limitato da trasformazioni geometriche.

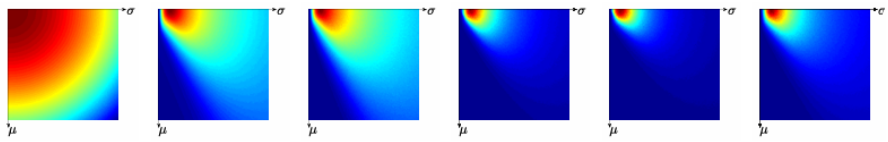
Tuttavia, di solito non è una buona scelta quando compaiono ingombri di sfondo e occlusioni all'interno delle finestre. La distanza del trasportatore della terra (EMD) è proposta per misurare la similarità tra due distribuzioni di probabilità. Inoltre, un approccio più robusto è stato proposto utilizzando una rappresentazione spaziale-esterna per misurare l'EMD. La misura di similarità della mappatura tonale è proposta per gestire il rumore, che è approssimato da una funzione a tratti costante/lineare. La correlazione asimmetrica è proposta per gestire sia il rumore che i cambiamenti di illuminazione. Una famiglia di misure di somiglianza interessante negli ultimi anni esplora una proprietà statistica globale sui due insiemi di punti. La similarità bidirezionale (BDS) propone che due insiemi di punti siano considerati simili se tutti i punti di un insieme sono contenuti nell'altro, e viceversa. La similarità dei ""best buddies " (BBS) conta i NN a due facce come statistica di similarità. La similarità della diversità deformabile (DDIS) misura la diversità delle corrispondenze tra i due insiemi ed è riportato che supera BBS rivelando la "deformazione" del campo dei NN. Nonostante la robustezza di BBS e DDIS contro le trasformazioni all'interno delle finestre di ricerca, scala e rotazione sull'intera finestra di ricerca non sono state considerate.

Dato un template (modello) ritagliato da un'immagine di riferimento e un'immagine di destinazione (target) legati da trasformazioni geometriche e fotometriche sconosciute, l'obiettivo è progettare una misura di similarità in grado di localizzare in modo distintivo, trovandone il valore massimo, una regione nell'immagine target che includa lo stesso oggetto presente nel template.

Ogni regione candidata nell'immagine target è definita da una finestra rettangolare, e le finestre candidate nell'immagine target vengono generate secondo una modalità a finestra scorrevole multi-scala (multiple-scale sliding window).

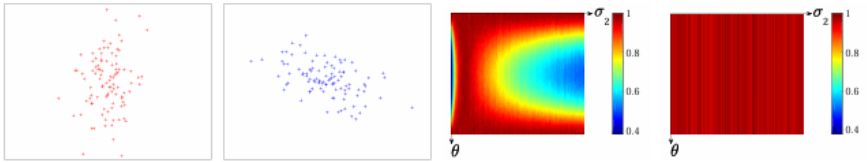
Prendendo come input l'immagine template e una finestra candidata estratta dall'immagine target , è possibile calcolare un punteggio SDS espresso come numero reale.

Per dimostrare l'efficacia di SDS nel misurare la somiglianza tra insiemi di punti con scale diverse, prima si fornisce un'analisi statistica 1D. Le aspettative di somiglianza tra due insiemi di punti estratti da due modelli Gaussiani 1D sono calcolate a scopo di confronto, dove gli insiemi di punti sono considerati come finestre modello/candidato e i punti come patch. Il nostro obiettivo è mostrare che l'aspettativa di SDS è massima quando i due modelli Gaussiani sono identici e diminuisce più rapidamente quando i modelli si separano. Analizziamo inoltre le aspettative degli insiemi di punti in diverse dimensioni per mostrare la robustezza di SDS rispetto alla scala (s). La Figura 2.9 (a) fino a (d) mostrano l'illustrazione delle mappe di aspettativa approssimata quando i due insiemi di punti hanno la stessa dimensione ($s = 1$). Si può osservare chiaramente che l'aspettativa di SDS diminuisce più rapidamente rispetto a SSD, BBS o DDIS quando i parametri della seconda Gaussiana (μ e σ) si allontanano dai parametri della prima Gaussiana ($\mu = 0$ e $\sigma = 1$). Le Figure 2.9 (d) fino a (f) mostrano il confronto delle mappe di aspettativa quando i due insiemi di



(a) SSD, $s = 1$ (b) BBS, $s = 1$ (c) DDIS, $s = 1$ (d) SDS, $s = 1$ (e) SDS, $s = 0.5$ (f) SDS, $s = 2$

Figura 2.9: Mappe di aspettativa [10]



(a) Example of T (b) Example of Q (c) $E(BBS)$ (d) $E(SDS)$

Figura 2.10: Mappe di aspettativa del BBS e SDS per una gaussiana 2D nel caso di rotazione [10]

punti hanno dimensioni diverse (s = 1, 0,5, 2), fornendo una forte evidenza che SDS è altamente robusto rispetto alla scala, poiché le mappe di aspettativa rimangono quasi invariate nonostante il cambiamento di scala. Per mostrare la robustezza rispetto alla rotazione, si analizza il valore atteso della similarità tra due insiemi T e Q (template T e finestra candidata Q) generati da modelli gaussiani 2D; come mostrato in Figura 2.10, fissiamo i parametri tranne θ e σ_2 per verificare l'effetto dell'angolo di rotazione insieme alla forma della gaussiana. Nel caso di BBS, come possiamo osservare dalla Figura 2.10 (c), quando σ_2 è estremamente piccolo, i punti generati tendono a formare una linea, la quale è sensibile alla rotazione poiché le linee si sovrappongono poco dopo la rotazione. Questo avviene anche quando $\sigma_2 \gg \sigma_1$, poiché si può osservare che il valore atteso diminuisce gradualmente all'aumentare di σ_2. Inoltre, si presuppone che una gaussiana isotropa non sia influenzata dalla rotazione, come confermato dalla Figura 2.10 (c) in cui, quando $\sigma_1 = \sigma_2 = 1$, il valore atteso si mantiene stabile rispetto alla rotazione. D'altra parte, SDS mostra un invarianza alla rotazione nonostante il cambiamento di forma della distribuzione nella Figura 2.10(d) [10].

Sulla base delle premesse generali del template matching multi-scala, questo approccio è stato adattato ed esattamente contestualizzato per risolvere il problema della dimensione: se infatti il ritaglio del fanale preso come modello è stato fatto da una foto scattata a 1 metro e l'auto nella foto è a 5 metri, il sistema non lo troverà, a meno del Multi-scale template matching, grazie alla cui implementazione nel codice, Python rimpicciolisce il nostro template diverse volte finché non trova la dimensione che si incastra perfettamente nell'auto. Invece di cercare il fanale una volta sola, creiamo un ciclo che rimpicciolisce l'immagine dell'auto (o ingrandisce il template) a vari step (es. dal 100% al 20% della dimensione originale). Per ogni step, Python controlla se il "pezzo del puzzle" si incastra meglio. I vantaggi di questo approccio

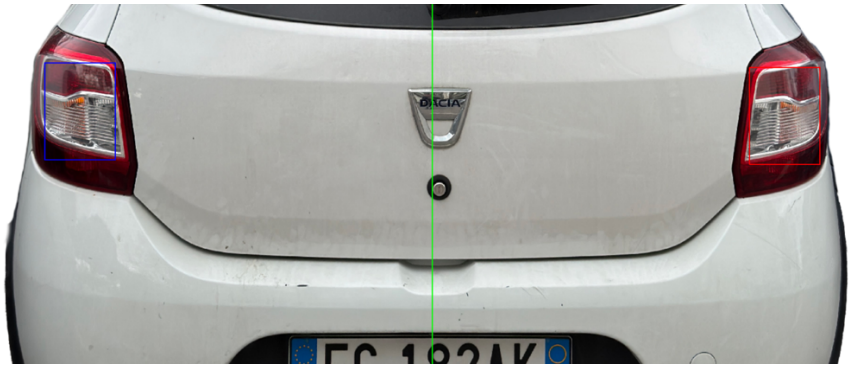


Figura 2.11: Evidenza sperimentale del Template Matching

sono

- Adattabilità: se il nostro template è un'immagine ad alta risoluzione e la macchina è lontana, l'analisi multi-scala troverà il momento in cui l'auto "diventa piccola quanto il template" e farà scattare il match
- Precisione: usando l'intera immagine del fanale come riferimento (e non solo dei punti chiave), è quasi impossibile che si confonda con la targa, perché la targa non ha la forma "rettangolo-rosso-con-plastica-trasparente" del fanale.
- Divisione Speculare: dividendo l'immagine a metà durante la ricerca, obblighiamo il sistema a trovare un solo fanale per lato, evitando che trovi due volte lo stesso fanale e ignori l'altro

Un'importante considerazione riguarda la ricerca su un'immagine dimezzata: affinché l'algoritmo funzioni correttamente, è necessario istruire l'algoritmo a cercare anche il riflesso del nostro template, operazione facilmente implementabile tramite la funzione cv2.flip. Integrando la funzione di flip nella sezione di codice che analizza il fanale, il sistema è in grado di individuare una corrispondenza quasi perfetta (con valori di similarità vicini a 1.0) anche sulla metà opposta dell'auto, poiché le caratteristiche del template e dell'immagine ora coincidono.

Dall'analisi condotta emerge un'evidente discrepanza tra il risultato ottenuto e quello teorico atteso. Sebbene l'asse di simmetria — calcolato a partire dalle feature del fanale sinistro e della sua controparte destra individuata dall'algoritmo — appaia visivamente corretto, i valori massimi di correlazione restituiti dal sistema per entrambi i lati rimangono pari a 0. Tale comportamento evidenzia l'assenza di un'effettiva corrispondenza matematica, a fronte di una sovrapposizione geometrica solo apparente.

Questo limite è attribuibile alla natura stessa del template matching il quale, pur integrando accorgimenti per le variazioni di scala, risulta inefficace in presenza di inclinazioni e distorsioni prospettiche. Tale vincolo rende l'approccio corrente inadeguato a risolvere appieno il problema.

Capitolo 3

Progettazione e sperimentazione di algoritmi per l'individuazione e la caratterizzazione di descrittori visivi a partire da architetture software preesistenti

Mentre nell'ambito delle Misure Meccaniche e Termiche lo strumento di programmazione di riferimento è tradizionalmente MATLAB, il presente lavoro, specifico nella Computer Vision,si focalizza sulla progettazione e sulla gestione della telecamera in ambiente Python. Ciononostante, la familiarità con l'utilizzo e le logiche di MATLAB è stata impiegata come base concettuale per individuare le funzioni e i comandi equivalenti in Python.

MATLAB (Matrix Laboratory) è un ambiente di calcolo numerico ad alte prestazioni e un linguaggio di programmazione sviluppato da MathWorks. È ampiamente usato in vari campi, tra cui ingegneria, scienza, finanza e accademia, per analisi dei dati, sviluppo di algoritmi e visualizzazione. MATLAB si distingue per la facilità d'uso, le potenti capacità di manipolazione di matrici e array e le ampie toolbox per applicazioni diverse. MATLAB offre un IDE che combina un editor di codice, un debugger e strumenti di visualizzazione, facilitando l'intero flusso di lavoro dalla prototipazione iniziale all'implementazione finale. L'IDE offre un'interfaccia intuitiva, rendendolo accessibile sia ai principianti che agli utenti esperti. Uno dei punti di forza di MATLAB è il suo linguaggio di programmazione ad alto livello che sfrutta operazioni su matrici e array, permettendo uno sviluppo di codice conciso ed efficiente per calcoli scientifici e ingegneristici. Il linguaggio MATLAB è progettato per eseguire calcoli numerici in modo efficiente, con supporto integrato per algebra lineare, elaborazione dei segnali, elaborazione delle immagini e altre funzioni matematiche. Fornisce anche potenti capacità di visualizzazione dei dati, permettendo agli utenti di creare grafici, plot e animazioni 2D e 3D di alta qualità per esplorare e comunicare i propri dati in modo efficace.

MATLAB è altamente estensibile attraverso l'uso di toolbox, che sono raccolte di funzioni e applicazioni specializzate, pensate per domini specifici, come i sistemi di controllo, il machine learning, l'ottimizzazione e la finanza. Questi toolbox ampliano le funzionalità di MATLAB, permettendo agli utenti di affrontare una vasta gamma

di problemi e di sfruttare algoritmi e tecniche all'avanguardia. Inoltre, MATLAB si integra perfettamente con altri linguaggi di programmazione e tecnologie, inclusi C, C++, Fortran, Java, .NET e Python, consentendo interoperabilità e riutilizzo del codice su diverse piattaforme e sistemi. Questa flessibilità rende MATLAB uno strumento prezioso, soprattutto nel nostro studio, per sviluppare e distribuire applicazioni in ambienti diversi, dai sistemi embedded alle piattaforme di cloud computing[19].

La funzione di MATLAB che prendiamo a riferimento nel nostro caso studio è il "Feature Matching" [18]: si tratta del rilevamento, estrazione e corrispondenza di feature, ovvero nello specifico si occupa di rilevare punti di interesse, estrarre descrittori di feature, trovare corrispondenze tra le feature, registrare e recuperare immagini.

Le feature locali e i relativi descrittori costituiscono i blocchi costitutivi di molti algoritmi di visione artificiale. Tra le loro applicazioni figurano la registrazione delle immagini, il rilevamento e la classificazione degli oggetti, il tracking, la stima del movimento e il recupero di immagini basato sul contenuto (CBIR). Questi algoritmi utilizzano feature locali per gestire meglio le variazioni di scala, la rotazione e l'occlusione. Gli algoritmi Computer Vision Toolbox™ includono i rilevatori di angoli FAST, Harris e Shi & Tomasi, nonché i rilevatori di blob SIFT, SURF, KAZE e MSER. La toolbox include i descrittori SIFT, SURF, FREAK, BRISK, LBP, ORB e HOG. È possibile combinare i rilevatori e i descrittori in base ai requisiti dell'applicazione. Nello specifico le funzioni di questo comando sono

Rilevamento di feature	
`detectBRISKFeatures`	Detect BRISK features
`detectFASTFeatures`	Detect corners using FAST algorithm
`detectHarrisFeatures`	Detect corners using Harris–Stephens algorithm
`detectKAZEFeatures`	Detect KAZE features
`detectMinEigenFeatures`	Detect corners using minimum eigenvalue algorithm
`detectMSERFeatures`	Detect MSER features
`detectORBFeatures`	Detect ORB keypoints
`detectSIFTFeatures`	Detect scale invariant feature transform (SIFT) features *(Da R2021b)*
`detectSURFFeatures`	Detect SURF features

Estrazione di feature	
`extractFeatures`	Extract interest point descriptors
`extractLBPFeatures`	Extract local binary pattern (LBP) features
`extractHOGFeatures`	Extract histogram of oriented gradients (HOG) features

Corrispondenza di feature	
`matchFeatures`	Find matching features
`matchFeaturesInRadius`	Find matching features within specified radius

Registrazione di immagini	
`estgeotform2d`	Estimate 2-D geometric transformation from matching point pairs (*Da R2022b*)
`estgeotform3d`	Estimate 3-D geometric transformation from matching point pairs (*Da R2022b*)
`imwarp`	Apply geometric transformation to image
`imblend`	Blend two images (*Da R2024b*)
`vision.BlockMatcher`	Estimate motion between images or video frames
`vision.TemplateMatcher`	Locate template in image

Visualizzazione e riproduzione	
`insertMarker`	Insert markers in image or video
`insertShape`	Insert shapes in image or video
`showMatchedFeatures`	Display corresponding feature points
`showShape`	Display shapes on image, video, or point cloud
`insertObjectAnnotation`	Annotate truecolor or grayscale image or video
`insertObjectKeypoints`	Insert object keypoints in image (*Da R2023b*)
`insertText`	Insert text in image or video
`imshow`	Display image
`imshowpair`	Compare differences between images
`vision.ChromaResampler`	Downsample or upsample chrominance components of images

Nome	Memorizzazione di feature
`binaryFeatures`	Object for storing binary feature vectors
`BRISKPoints`	Object for storing BRISK interest points
`cornerPoints`	Object for storing corner points
`KAZEPoints`	Object for storing KAZE interest points
`MSERRegions`	Object for storing MSER regions
`ORBPoints`	Object for storing ORB keypoints
`SIFTPoints`	Object for storing SIFT interest points *(Da R2021b)*
`SURFPoints`	Object for storing SURF interest points

Nome	Trasformazione di oggetti
`rigidtform2d`	2-D rigid geometric transformation *(Da R2022b)*
`simtform2d`	2-D similarity geometric transformation *(Da R2022b)*
`affinetform2d`	2-D affine geometric transformation *(Da R2022b)*
`projtform2d`	2-D projective geometric transformation *(Da R2022b)*
`rigidtform3d`	3-D rigid geometric transformation *(Da R2022b)*
`simtform3d`	3-D similarity geometric transformation *(Da R2022b)*

Nome	Creazione di un database di riferimento
`bagOfFeatures`	Bag of visual words object
`invertedImageIndex`	Search index that maps visual words to images

Nome	Recupero delle immagini
`retrieveImages`	Search image set for similar image
`imageDatastore`	Datastore for image data
`evaluateImageRetrieval`	Evaluate image search results

Nome	Recupero delle immagini usando la rete CLIP
`clipNetwork`	Create pretrained CLIP deep learning neural network for vision-language tasks *(Da R2026a)*
`extractImageEmbeddings`	Extract feature embeddings from image using CLIP network image encoder *(Da R2026a)*
`extractTextEmbeddings`	Extract text embeddings from search text using CLIP network text encoder *(Da R2026a)*

Per il rilevamento e l'estrazione locale delle caratteristiche dobbiamo prima capire che cosa sono le funzionalità locali: le caratteristiche locale si riferiscono a un

motivo o a una struttura distinta trovata in un'immagine, come punti, bordo o piccoli patch di immagine. Di solito sono associate a una patch immagine che si differenzia dall'ambiente immediato per consistenza, colore o intensità. Quale sia la caratteristica che in realtà rappresenta non importa, basta solo che sia distinta dall'ambiente circostante. Esempi tra le caratteristiche locali sono macchie, angoli e pixel ai bordi.

Esempio di rilevamento degli angoli[20]

3.1

```
I = imread("circuit.tif");
corners = detectFASTFeatures(I,MinContrast=0.1);
J = insertMarker(I,corners,"circle");
imshow(J)
```

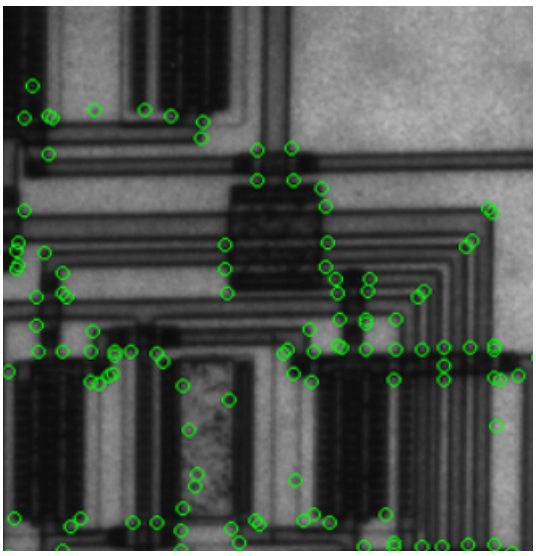


Figura 3.1: Esempio di rilevamento angoli [20]

Ciò che rende buona una funzione locale è che i rilevatori che si basano su approcci basati su gradiente e variazione di intensità rilevano buone features locali. Queste caratteristiche includono bordi, macchie e regioni. Le buone caratteristiche locali mostrano le seguenti proprietà:

- Rilevamenti ripetibili: quando somministrati due immagini della stessa scena, la maggior parte delle caratteristiche che il rilevatore trova in entrambe le immagini sono le stesse. Le caratteristiche sono robuste ai cambiamenti delle condizioni di visione e al rumore
- Distinzione: l'intorno attorno al centro della feature varia abbastanza da permettere un confronto affidabile tra le caratteristiche

- Localizzabile: la funzione ha una posizione unica assegnata. I cambiamenti nelle condizioni di visione non influenzano la sua posizione

Il rilevamento delle caratteristiche identifica le regioni di un'immagine che contengono caratteristiche distintive, come angoli o punti. Il rilevamento delle caratteristiche viene utilizzato per identificare punti di interesse da utilizzare nell'elaborazione successiva. Questi punti non corrispondono necessariamente a strutture fisiche, come gli angoli di un tavolo. L'elemento chiave del rilevamento delle caratteristiche è l'identificazione di caratteristiche che rimangono localmente invarianti, in modo che possano essere rilevate anche in presenza di rotazioni o variazioni di scala.

L'estrazione delle caratteristiche comporta il calcolo di un descrittore, che viene tipicamente calcolato su regioni centrate attorno alle caratteristiche rilevate. I descrittori si basano sull'elaborazione delle immagini per trasformare un vicinato locale di pixel in una rappresentazione vettoriale compatta. Questa nuova rappresentazione consente il confronto tra vicinati indipendentemente dai cambiamenti di scala o di orientamento. I descrittori, come SIFT o SURF, si basano sul calcolo dei gradienti locali. I descrittori binari, come BRISK, ORB o FREAK, si basano su coppie di differenze di intensità locali, che vengono poi codificate in un vettore binario.

La scelta del rilevatore di feature e del descrittore va fatta tenendo conto dei criteri della nostra applicazione e della natura dei nostri dati. La prima tabella 3.1 aiuta a comprendere i criteri generali su cui basare la scelta. Le due tabelle successive 3.23.3 forniscono dettagli sui rilevatori e sui descrittori disponibili nel Computer Vision Toolbox [20].

Considerazioni per la scelta di un rilevatore e di un descrittore

Criterio	Suggerimento
Tipo di feature presenti nell'immagine	Utilizzare un rilevatore appropriato per i propri dati. Ad esempio, se l'immagine contiene cellule batteriche, usare il rilevatore di blob anziché il rilevatore di angoli. Se l'immagine è una vista aerea di una città, è possibile usare il rilevatore di angoli per individuare strutture artificiali.
Contesto in cui si utilizzano le feature: • Corrispondenza di punti chiave • Classificazione	I descrittori HOG, SURF e KAZE sono adatti per attività di classificazione. Al contrario, i descrittori binari, come ORB, BRISK e FREAK, sono generalmente utilizzati per trovare corrispondenze tra punti in immagini diverse, utilizzate per la registrazione.
Tipo di distorsione presente nell'immagine	Scegliere un rilevatore e un descrittore che tengano conto della distorsione presente nei dati. Ad esempio, se non è presente alcun cambiamento di scala, si può considerare un rilevatore di angoli che non gestisce la scala. Se i dati contengono un livello di distorsione più elevato, come scala e rotazione, utilizzare un rilevatore e descrittore di feature SIFT, SURF, ORB o KAZE. I metodi SURF e KAZE sono computazionalmente onerosi.
Requisiti prestazionali: • Prestazioni in tempo reale richieste • Accuratezza contro velocità	I descrittori binari sono generalmente più veloci ma meno accurati rispetto ai descrittori basati sul gradiente. Per una maggiore accuratezza, utilizzare più rilevatori e descrittori contemporaneamente.

Tabella 3.1: Tabella criteri [20]

Scegliere una funzione di rilevamento in base al tipo di feature

Rilevatore	Tipo di feature	Funzione	Indipendente dalla scala
FAST	Angolo	`detectFASTFeatures`	No
Algoritmo autovalore minimo [4]	Angolo	`detectMinEigenFeatures`	No
Rilevatore di angoli [3]	Angolo	`detectHarrisFeatures`	No
SIFT	Blob	`detectSIFTFeatures`	Sì
SURF	Blob	`detectSURFFeatures`	Sì
KAZE	Blob	`detectKAZEFeatures`	Sì
BRISK	Angolo	`detectBRISKFeatures`	Sì
MSER	Regione con intensità uniforme	`detectMSERFeatures`	Sì
ORB	Angolo	`detectORBFeatures`	No

Tabella 3.2: Rilevatori tabella [20]

Descrittori: invarianza e uso tipico

Descrittore	Binario	Funzione e Metodo	Invarianza		Uso tipico	
			Scala	Rotazione	Ricerca di corrispondenze tra punti	Classificazione
HOG	No	`extractHOGFeatures(I, ...)`	No	No	No	Sì
LBP	No	`extractLBPFeatures(I, ...)`	No	Sì	No	Sì
SIFT	No	`extractFeatures(I,points,Method="SIFT")`	Sì	Sì	Sì	Sì
SURF	No	`extractFeatures(I,points,Method="SURF")`	Sì	Sì	Sì	Sì
KAZE	No	`extractFeatures(I,points,Method="KAZE")`	Sì	Sì	Sì	Sì
FREAK	Sì	`extractFeatures(I,points,Method="FREAK")`	Sì	Sì	Sì	No
BRISK	Sì	`extractFeatures(I,points,Method="BRISK")`	Sì	Sì	Sì	No
ORB	Sì	`extractFeatures(I,points,Method="ORB")`	No	Sì	Sì	No
• Blocco • Semplice vicinato di pixel attorno a un punto chiave	No	`extractFeatures(I,points,Method="Block")`	No	No	Sì	Sì

Tabella 3.3: Descrittori tabella [20]

Esempio di uso delle features locali [20]

L'allineamento di due immagini è un modo semplice per comprendere le caratteristiche locali. Questo esempio individua una trasformazione geometrica tra due immagini. Utilizza le caratteristiche locali per individuare punti di ancoraggio ben localizzati.

La prima immagine 3.2 è l'immagine originale



Figura 3.2: originale [20]

La seconda immagine 3.3corrisponde all'originale ruotata e scalata

```
original = imread('cameraman.tif');
figure
imshow(original);
scale = 1.3;
J = imresize(original,scale);
theta = 31;
distorted = imrotate(J,theta);
figure
imshow(distorted)
```



Figura 3.3: originale ruotato e scalato [20]

- Rilevare feature corrispondenti tra l'immagine originale e quella distorta: rilevare le feature SURF corrispondenti è il primo passo per determinare la trasformazione necessaria a correggere l'immagine distorta.

```
1 ptsOriginal = detectSURFFeatures(original);
2 ptsDistorted = detectSURFFeatures(distorted);
```

- Estrarre le feature e confrontare i blob rilevati tra le due immagini: il passaggio di rilevamento ha individuato diverse strutture a blob approssimativamente corrispondenti in entrambe le immagini.

- Confrontare le feature rilevate. Questo processo è facilitato dall'estrazione delle feature, che determina un descrittore di patch locale.

```
[featuresOriginal,validPtsOriginal] = ...
          extractFeatures(original,ptsOriginal);
[featuresDistorted,validPtsDistorted] = ...
          extractFeatures(distorted,ptsDistorted);
```

  È possibile che non tutti i punti originali siano stati utilizzati per estrarre i descrittori. I punti potrebbero essere stati scartati se troppo vicini al bordo dell'immagine. Pertanto, i punti validi vengono restituiti insieme ai descrittori delle feature.

  La dimensione della patch utilizzata per calcolare i descrittori viene determinata durante la fase di estrazione delle feature. La dimensione della patch corrisponde alla scala alla quale la feature viene rilevata. Indipendentemente dalla dimensione della patch, i due vettori di feature, `featuresOriginal` e `featuresDistorted`, vengono calcolati in modo da avere la stessa lunghezza. I descrittori consentono di confrontare le feature rilevate, indipendentemente dalla loro dimensione e rotazione.

- Trovare le corrispondenze candidate: ottenere le corrispondenze candidate tra le feature fornendo i descrittori alla funzione "match features". Le corrispondenze candidate implicano che i risultati possano contenere alcune corrispondenze non valide. Due patch che corrispondono possono indicare feature simili ma potrebbero non essere una corrispondenza corretta. L'angolo di un tavolo può assomigliare all'angolo di una sedia, ma le due feature non sono ovviamente una corrispondenza.

```
indexPairs = matchFeatures(featuresOriginal,featuresDistorted);
```

- Trovare le posizioni dei punti in entrambe le immagini: ogni riga della variabile restituita `indexPairs` contiene due indici delle corrispondenze candidate tra le feature delle due immagini.

- Utilizzare gli indici per raccogliere le posizioni effettive dei punti da entrambe le immagini.

```
matchedOriginal = validPtsOriginal(indexPairs(:,1));
matchedDistorted = validPtsDistorted(indexPairs(:,2));
```

- Visualizzare le corrispondenze candidate 3.4

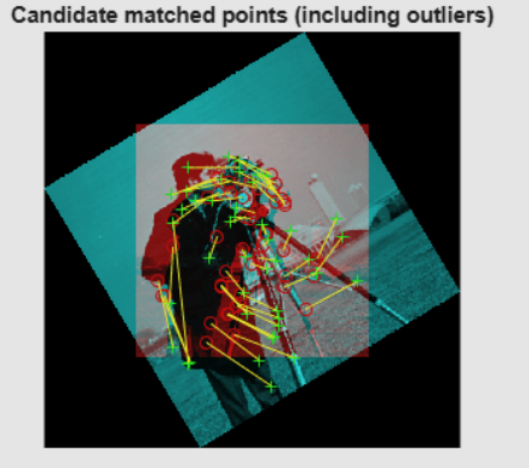


Figura 3.4: matched points candidati[20]

```
figure
showMatchedFeatures(original,distorted,matchedOriginal,matchedDistorted)
title('Candidate matched points (including outliers)')
```

- Analizzare le posizioni delle feature: se è presente un numero sufficiente di corrispondenze valide, rimuovere le corrispondenze errate. Una tecnica efficace per questo scenario è l'algoritmo RANSAC. La funzione `estgeotform2d` implementa l'M-estimator sample consensus (MSAC), che è una variante dell'algoritmo RANSAC. MSAC trova una trasformazione geometrica e separa gli inlier (corrispondenze corrette) dagli outlier (corrispondenze spurie).

```
[tform,inlierIdx] = estgeotform2d(matchedDistorted, ...
    matchedOriginal,'similarity');
inlierDistorted = matchedDistorted(inlierIdx,:);
inlierOriginal = matchedOriginal(inlierIdx,:);
```

- Visualizzare i punti corrispondenti 3.5

```
figure
showMatchedFeatures(original,distorted,inlierOriginal,inlierDistorted)
title('Matching points (inliers only)')
legend('ptsOriginal','ptsDistorted')
```

- Verificare la trasformazione geometrica calcolata: applicare la trasformazione geometrica calcolata all'immagine distorta.

```
outputView = imref2d(size(original));
recovered = imwarp(distorted,tform,OutputView=outputView);
```

- visualizzare l'immagine recuperata e l'originale 3.6



Figura 3.5: matching points [20]



Figura 3.6: confronto [20]

```
outputView = imref2d(size(original));
recovered = imwarp(distorted,tformTotal,OutputView=outputView);

figure
imshowpair(original,recovered,'montage')
```

In Python, l'equivalente del comando matchFeatures di MATLAB si trova principalmente nella libreria OpenCV (cv2), che è lo standard de facto per la computer vision. OpenCV (Open Source Computer Vision Library) è una libreria orientata all'elaborazione e all'analisi delle immagini. Sviluppata inizialmente da Intel nel 1999, ha guadagnato rapidamente popolarità, diventando oggi una delle più utilizzate nello sviluppo di applicazioni di visione artificiale. I suoi ambiti di applicazione sono molti, e tra questi spiccano:

- Riconoscimento facciale
- Identificazione di oggetti o persone
- Ispezione e sorveglianza

- Giochi e controlli
- Conteggio di oggetti
- Analisi di immagini mediche
- Robotica
- Realtà aumentata

A partire dalla versione OpenCV 2.3.1, include un'interfaccia di programmazione C++, Java e Python. Inoltre, è disponibile su diverse piattaforme, tra cui Windows, Linux, Mac OS X e Android.

Oltre a essere multipiattaforma e multilinguaggio, è degno di nota il fatto che OpenCV sia un prodotto con licenza BSD (Berkeley Software Distribution), con meno restrizioni rispetto ad altre come la GPL (General Public License, Licenza Pubblica Generale GNU), poiché permette persino l'uso del suo codice sorgente in software commerciale. Come detto, OpenCV è disponibile nei principali linguaggi di programmazione: tra tutti questi, si utilizzerà Python, forse quello impiegato dal maggior numero di programmatori al mondo. I motivi dell'enorme successo di questo linguaggio risiedono sicuramente nella sua semplicità di apprendimento e facilità d'uso [22].

Non solo permette la classica modalità di programmazione imperativa, ma supporta anche la programmazione orientata agli oggetti, aprendosi così all'uso di un'infinità di librerie che, come OpenCV, si basano su questo paradigma.

In OpenCV vengono incluse le seguenti librerie correlate:

- **NumPy**: questa è una dipendenza dei binding Python di OpenCV. Fornisce funzionalità di calcolo numerico, inclusi array efficienti.
- **SciPy**: questa è una libreria per il calcolo scientifico strettamente correlata a NumPy. Non è richiesta da OpenCV, ma è utile per manipolare i dati nelle immagini di OpenCV.
- **OpenNI**: questa è una dipendenza opzionale di OpenCV. Aggiunge il supporto per determinate telecamere di profondità, come Asus XtionPRO.
- **SensorKinect**: questo è un plugin di OpenNI e una dipendenza opzionale di OpenCV. Aggiunge il supporto per la telecamera di profondità Microsoft Kinect.

Mentre MATLAB offre una funzione spesso "tutto in uno", in Python il processo è leggermente più articolato ma molto più flessibile. Il workflow tipico si divide in tre fasi: rilevamento (detection), estrazione dei descrittori e matching. Gli algoritmi per le caratteristiche si dividono in due categorie: rivelatori di caratteristiche e descrittori di caratteristiche.

Un rilevatore di caratteristiche individua le regioni di interesse in un'immagine. L'ingresso in un rilevatore di caratteristiche è un'immagine, e l'output sono coordinate pixel delle aree significative dell'immagine.

Un descrittore di caratteristica codifica quella caratteristica in una "impronta digitale" numerica. La descrizione delle caratteristiche rende una caratteristica unicamente identificabile rispetto ad altre caratteristiche presenti nell'immagine. Come algoritmi di rilevamento delle caratteristiche si utilizzano, a seconda della features [21]

- Rilevamento degli angoli di Harris: un angolo è un'area di un'immagine che presenta una grande variazione nei valori di intensità del colore dei pixel in tutte le direzioni. Un algoritmo popolare per rilevare gli angoli in un'immagine è chiamato Harris Corner Detector
- Rilevatore di angoli Shi-Tomasi
- Trasformata delle caratteristiche invarianti in scala (SIFT): è in grado di rilevare caratteristiche in un'immagine indipendentemente dai cambiamenti di dimensione o orientamento. Questa proprietà di SIFT conferisce a OpenCV un vantaggio rispetto ad altri algoritmi di rilevamento delle caratteristiche che falliscono quando si effettuano trasformazioni su un'immagine.
- Funzionalità robuste e accelerate (SURF): SURF è una versione più veloce di SIFT. È un altro modo per trovare le caratteristiche in un'immagine.
- Caratteristiche di Accelerated Segment Test (FAST): molti degli algoritmi di rilevamento delle caratteristiche che abbiamo esaminato finora funzionano bene in diverse applicazioni. Tuttavia, non sono abbastanza veloci per alcuni casi d'uso della robotica (ad esempio SLAM).

  L'algoritmo FAST è un algoritmo molto veloce per rilevare gli angoli in un'immagine.
- Rilevatori di blob con LoG, DoG e DoH: un blob è un altro tipo di caratteristica in un'immagine. Un blob è una regione in un'immagine con valori di intensità pixel simili, una sorta di macchia, un'area chiara su scuro o scura su chiara di un'immagine.

  Tre algoritmi popolari di rilevamento dei blob sono il Laplaciano di Gaussiano (LoG), la Differenza di Gaussiano (DoG) e il Determinante dell'Assiano (DoH).

Come algoritmi di Descrittore di Caratteristiche si utilizzano[21]

- Istogramma dei gradienti orientati: l'algoritmo HoG scompone un'immagine in piccole sezioni e calcola il gradiente e l'orientamento in ciascuna sezione. Queste informazioni vengono poi raccolte in contenitori per calcolare gli istogrammi. Questi istogrammi forniscono a un'immagine "impronte digitali" numeriche che la rendono unicamente identificabile

- Caratteristiche elementari binarie robuste e indipendenti (BRIEF): BRIEF è un'alternativa veloce ed efficiente a SIFT
- FAST orientato e BRIEF ruotato (ORB): SIFT è stato brevettato per molti anni, e SURF è ancora un algoritmo brevettato. ORB è stato creato nel 2011 come alternativa gratuita a questi algoritmi. Combina gli algoritmi FAST e BRIEF. Si può usare ORB per localizzare le caratteristiche in un'immagine e poi abbinarle a quelle di un'altra immagine.

Ad esempio [21], considero questo logo di Whole Foods3.7. Questo logo sarà la nostra immagine di allenamento. Voglio trovare questo logo Whole Foods all'interno



Figura 3.7: immagine di allenamento



Figura 3.8: immagine di richiesta

della nostra immagine di richiesta 3.8. Utilizzando l'algoritmo di rilevazione ORB quello che otteniamo sarà 3.9



Figura 3.9: risultato con ORB

Grazie a queste funzioni è chiaro quindi come il feature matching riesca a rilevare gli stessi punti in immagini anche inclinate o distorte, soluzione quindi che sembra fare al caso nostro dal momento in cui non si può prevedere un approccio frontale e perfettamente ortogonale da parte di tutti gli utilizzatori degli smart glasses. Tuttavia ci sono alcune distinzioni importanti da fare in base agli algoritmi visti che decidiamo di usare: il "matching" non è un semplice confronto tra pixel, ma si basa su descrittori matematici progettati proprio per essere robusti ai cambiamenti geometrici. Sulla base dei dati visti 3.3 in precedenza possiamo classificare gli algoritmi (sia in MATLAB che in Python) in base a come gestiscono queste situazioni:

Inclinazione (Rotazione)

- SIFT
- SURF
- ORB

sono invarianti alla rotazione. Quando l'algoritmo individua un punto di interesse, calcola un "orientamento dominante" per quel punto. Tutte le informazioni estratte attorno a quel punto vengono poi "ruotate" matematicamente verso una direzione standard prima di creare il descrittore. Il risultato è che anche se l'immagine è ruotata di 90° o 180°, il descrittore sembrerà lo stesso.

Distorsione prospettica e affine

Se si inclina la fotocamera rispetto al piano dell'oggetto (distorsione prospettica), i cerchi diventano ellissi e le proporzioni cambiano.

- SIFT e SURF tollerano bene inclinazioni fino a circa 30-45 gradi (resistenza moderata)
- Se l'angolo è molto acuto, i descrittori standard iniziano a fallire (distorsioni estreme). In questi casi si usano varianti come ASIFT (Affine SIFT), che simula diverse inclinazioni dell'immagine per trovare match anche in condizioni estreme.

Cambiamento di scala (Zoom)

Questi algoritmi sono invarianti alla scala. Se ci allontaniamo o ci avviciniamo all'oggetto, l'algoritmo costruisce una "piramide" di immagini a diverse risoluzioni per trovare il punto alla dimensione corretta.

Tuttavia sia in MATLAB che in Python il comando matchFeatures da solo spesso produce dei "falsi positivi" (linee che collegano punti sbagliati). Come mostrato in figura 3.103.11, utilizzando la funzione del matching feature in Python otteniamo due risultati: il primo 3.10 definisce le corrispondenze trovate comprendendo anche i falsi positivi, il secondo risultato 3.11 ci fornisce invece le corrispondenze finali filtrate dai falsi positivi.

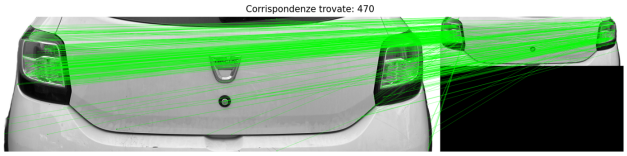


Figura 3.10: Corrispondenze trovate

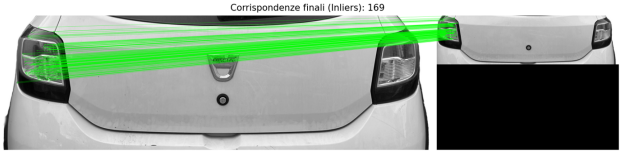


Figura 3.11: Corrispondenze finali

Per gestire distorsioni e inclinazioni in modo professionale, abbiamo usato un secondo passaggio chiamato RANSAC. La stima robusta del modello è fondamentale nella visione artificiale, essenziale per compiti come la localizzazione visiva, la Struttura da Movimento (SfM), la Localizzazione e Mappatura Simultanea (SLAM) e la stima della posa degli oggetti. L'algoritmo Random Sample Consensus (RANSAC) è l'approccio standard per una stima robusta in presenza di dati rumorosi o contaminati da outlier (falsi positivi). RANSAC seleziona iterativamente campioni di dati minimi, stima i parametri del modello e valuta la qualità del modello contando gli inlier. Nonostante la sua popolarità, la sensibilità di RANSAC alle distribuzioni di rumore del mondo reale, ai suoi parametri (ad esempio, la soglia tra inlier e outlier) e la suscettibilità a campioni mal condizionati hanno motivato ampie ricerche. I miglioramenti includono nuovi metodi di campionamento per la scoperta rapida di buoni modelli, nuove funzioni di valutazione che modellano meglio le distribuzioni di rumore, algoritmi di raffinamento per mitigare gli effetti dei campioni rumorosi e controlli di degenerazione. I metodi recenti esplorano anche l'apprendimento di componenti specifici di RANSAC (ad esempio, probabilità di inlier, valutazione/campionamento) o la possibilità di una stima differenziabile end-to-end per l'addestramento delle reti. Data l'importanza fondamentale di RANSAC e la sua ampia applicazione, sono state dedicate molte ricerche, che non vedremo nei dettagli in questo studio, per migliorare i suoi aspetti principali: robustezza nei confronti di alti tassi di outlier, accuratezza del modello stimato ed efficienza computazionale [17]. Questi sforzi possono essere ampiamente suddivisi in progressi nelle strategie di campionamento, nei meccanismi di valutazione del modello, nelle tecniche di ottimizzazione e verifica locali e, più recentemente, nell'eliminazione degli outlier tramite apprendimento. Sorvolando i numerosi miglioramenti implementati nella funzione per ampliarne l'efficienza, non essendo oggetto principale del nostro studio, mentre in MATLAB useremo la funzione "estimateGeometricTransform", in Python come schema di omografia veloce e robusto per il rilevamento in tempo reale di bersagli planari RANSAC capisce qual è la "regola geometrica" (es. una prospettiva) che lega la maggior parte dei punti e scarta automaticamente tutti i match che non seguono quella regola (gli outlier), pulendo il

risultato finale.

3.1 Implementazione in Python ed analisi strutturale del codice

L'obiettivo di questa sezione è stabilire una corrispondenza puntuale tra la funzione MATLAB per il feature matching tra due immagini e la sua controparte in Python. Attraverso un'analisi dettagliata dello script originale, l'algoritmo viene scomposto nelle sue componenti fondamentali per individuare, per ciascuna di esse, l'equivalente funzionale e sintattico in ambiente Python.

```
%% Registrazione automatica di immagini (rotazione + scala)
% Script MATLAB basato sull'esempio "Automatically Find Image Rotation and
    Scale" (Computer Vision Toolbox),
% adattato per allineare un'immagine a colori disallineata rispetto a
    un'immagine di riferimento (es. IR).

clear all; close all; clc;

%% --- Step 1: Lettura Immagini ---
original = imread('test_IR.bmp');      % Immagine di riferimento (es.
    infrarosso)
distorted_rgb = imread('test_color.bmp'); % Immagine distorta a colori
    (ruotata/scalata)
filewrite = 'filerisultato.bmp';       % Nome file salvataggio finale

figure; imshow(original); title('Originale (riferimento)');
figure; imshow(distorted_rgb); title('Distorta (da allineare)');

%% --- Step 2: Conversione in scala di grigi ---
% Ci riduciamo delle feature lavoro su immagini a singolo canale
distorted_gray = rgb2gray(distorted_rgb);

%% --- Step 3: Ricerca e Abbinamento delle Feature (SURF) ---
ptsOriginal = detectSURFFeatures(original);
ptsDistorted = detectSURFFeatures(distorted_gray);

[featuresOriginal, validPtsOriginal] = extractFeatures(original,
    ptsOriginal);
[featuresDistorted, validPtsDistorted] = extractFeatures(distorted_gray,
    ptsDistorted);

% Abbiniamo i punti in base ai descrittori
indexPairs = matchFeatures(featuresOriginal, featuresDistorted, ...
    'MatchThreshold', 40, 'MaxRatio', 0.7, 'Unique', true);

matchedOriginal = validPtsOriginal(indexPairs(:,1));
```

```
matchedDistorted = validPtsDistorted(indexPairs(:,2));
fprintf('Trovati %d punti in comune (prima del filtraggio).\n',
    size(indexPairs,1));

figure; showMatchedFeatures(original, distorted_gray, matchedOriginal,
    matchedDistorted);
title('Corrispondenze trovate (incluse le errate)');

%% --- Step 4: Stima della Trasformazione (MSAC / RANSAC) ---
% 'similarity' cerca traslazione + rotazione + scala (scarta outlier via
    MSAC)
[tform, inlierIdx] = estgeotform2d(matchedDistorted, matchedOriginal,
    'similarity');

inlierDistorted = matchedDistorted(inlierIdx,:);
inlierOriginal = matchedOriginal(inlierIdx,:);
fprintf('Corrispondenze rimaste dopo il filtraggio (inlier): %d\n',
    nnz(inlierIdx));

figure; showMatchedFeatures(original, distorted_gray, inlierOriginal,
    inlierDistorted);
title('Corrispondenze finali (solo inlier)'); legend('ptsOriginal',
    'ptsDistorted');

%% --- Step 5: Estrazione di Angolo e Scala ---
% tform e' una 'distorted' a 'original' -> invertiamo per la distorsione
    originale
invTform = invert(tform);

Ainv = invTform.A;
ss = Ainv(2,1); sc = Ainv(1,1);
scaleRecovered = hypot(ss, sc);
fprintf('Scala recuperata: %.4f\n', scaleRecovered);

thetaRecovered = atan2d(-ss, sc); % Angolo positivo = rotazione in senso
    orario
fprintf('Angolo recuperato: %.4f gradi\n', thetaRecovered);

%% --- Step 6: Ripristino e Salvataggio ---
% Applichiamo la trasformazione all'immagine a COLORI originale
outputView = imref2d(size(original));
recovered = imwarp(distorted_rgb, tform, 'OutputView', outputView);

figure; imshowpair(original, recovered, 'montage'); title('Originale vs
    Recuperato');

imwrite(recovered, filewrite);
```

```
fprintf('Immagine allineata salvata come "%s".\n', filewrite);
```

Listing 3.1: Codice MATLAB per il matching feature

```
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt

# --- Step 1: Lettura delle Immagini ---
# Carichiamo l'immagine di riferimento in scala di grigi
original = cv2.imread('auto.jpg', cv2.IMREAD_GRAYSCALE)
# Carichiamo l'immagine distorta a colori
distorted_rgb = cv2.imread('vettura_inclinata.jpeg')

# Gestione di eventuali errori di caricamento
if original is None or distorted_rgb is None:
    raise FileNotFoundError('Immagini non trovate. Verifica i percorsi.')

# Convertiamo in scala di grigi l'immagine distorta per la ricerca dei
    punti
distorted_gray = cv2.cvtColor(distorted_rgb, cv2.COLOR_BGR2GRAY)
filewrite = 'pippo.txt'

# --- Step 2: Ricerca e Abbinamento delle Feature (SIFT) ---
sift = cv2.SIFT_create()

kp1, des1 = sift.detectAndCompute(original, None)
kp2, des2 = sift.detectAndCompute(distorted_gray, None)

bf = cv2.BFMatcher(norm=cv2.NORM_L2)
matches = bf.knnMatch(des1, des2, k=2)

good_matches = []
for m, n in matches:
    if m.distance < 0.75 * n.distance:
        good_matches.append(m)

print(f'Trovati {len(good_matches)} punti in comune validi.')

img_matches = cv2.drawMatches(
    original, kp1,
    distorted_gray, kp2,
    good_matches, None,
    matchColor=(0, 255, 0),
    singlePointColor=(255, 0, 0),
    flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
)
```

```
plt.figure(figsize=(15, 7))
plt.imshow(img_matches, cmap='gray')
plt.title(f'Corrispondenze trovate: {len(good_matches)}')
plt.axis('off')
plt.show()

# --- Step 4: Stima della Trasformazione ---
if len(good_matches) > 4:
    src_pts = np.float32([kp1[m.trainIdx].pt for m in
        good_matches]).reshape(-1, 1, 2)
    dst_pts = np.float32([kp2[m.queryIdx].pt for m in
        good_matches]).reshape(-1, 1, 2)
    T, inliers = cv2.estimateAffinePartial2D(src_pts, dst_pts,
        method=cv2.RANSAC)
    matches_mask = inliers.ravel().tolist()

    img_matches = cv2.drawMatches(
        original, kp1,
        distorted_gray, kp2,
        good_matches, None,
        matchColor=(0, 255, 0),
        singlePointColor=None,
        matchesMask=matches_mask,
        flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
    )

    plt.figure(figsize=(15, 7))
    plt.imshow(img_matches, cmap='gray')
    plt.title(f'Corrispondenze finali (Inliers): {sum(matches_mask)}')
    plt.axis('off')
    plt.show()

    # --- Step 5: Estrazione di Angolo e Scala ---
    sc = T[0, 0]
    ss = T[1, 0]

    scale_recovered = math.sqrt(sc**2 + ss**2)
    theta_recovered = math.atan2(ss, sc) * 180 / math.pi

    print(f'Scala recuperata: {scale_recovered:.4f}')
    print(f'Angolo recuperato: {theta_recovered:.4f} gradi')

    # --- Step 6: Otteniamo le dimensioni dell'immagine originale ---
    h, w = original.shape
    recovered = cv2.warpAffine(distorted_rgb, T, (w, h))
    cv2.imwrite(filewrite, recovered)
```

```
    recovered_rgb_plot = cv2.cvtColor(recovered, cv2.COLOR_BGR2RGB)

    fig, ax = plt.subplots(1, 2, figsize=(12, 6))
    ax[0].imshow(original, cmap='gray')
    ax[0].set_title('Originale (Riferimento)')
    ax[0].axis('off')

    ax[1].imshow(recovered_rgb_plot)
    ax[1].set_title('Recuperata (Colore Allineato)')
    ax[1].axis('off')

    plt.tight_layout()
    plt.show()

else:
    print('Non ci sono abbastanza punti in comune per allineare le
        immagini.')
```

Listing 3.2: Codice Python per il matching feature

Si parte un classico codice MATLAB di Computer Vision3.1 utilizzato per la registrazione di immagini, ovvero il processo di allineamento di due immagini diverse della stessa scena (in questo caso, un'immagine a infrarossi e una visibile a colori) calcolando quanto una è ruotata e scalata rispetto all'altra. Ecco l'analisi di cosa fa il codice, divisa per blocchi logici:

- Lettura e Preparazione delle Immagini (Step 1 e 2): Il codice inizia pulendo l'ambiente di lavoro (clear all, close all, clc) e poi carica due immagini:l'immagine di riferimento (original), caricata da TestIR.bmp. È l'immagine fissa (infrarossi) su cui vogliamo allineare l'altra, e l'immagine distorta (distorted.rgb),caricata da test.color.bmp. È l'immagine a colori che risulta disallineata (ruotata o scalata). Viene anche definita la stringa "filewrite='filerisultato.bmp'" che sarà il nome del file salvato alla fine. L'immagine a colori viene convertita in scala di grigi (rgb2gray) perché gli algoritmi di rilevamento dei punti di interesse lavorano tipicamente su immagini a singolo canale

- Estrazione e Abbinamento delle Feature (Step 3): questa è la parte centrale dell'algoritmo. Invece di sovrapporre le immagini a tentativi, il codice cerca punti salienti (come angoli o spigoli molto netti):
  - Rilevamento: Usa detectSURFFeatures per trovare questi punti chiave (feature SURF) in entrambe le immagini
  - Estrazione: Con extractFeatures, il codice crea una sorta di "impronta digitale" (descrittore matematico) per ogni punto trovato, in modo da poterlo riconoscere

  - Abbinamento: matchFeatures confronta le impronte digitali delle due immagini e crea delle coppie di punti che ritiene uguali (indexPairs)
  - Infine, visualizza a schermo le due immagini con le linee che collegano i punti trovati. In questa fase ci sono molti "falsi positivi" (outlier), ovvero punti abbinati in modo errato 3.10

- Calcolo della Trasformazione Geometrica (Step 4): qui il codice pulisce gli errori del passaggio precedente e trova la vera relazione matematica tra le due immagini: usa la funzione estimateGeometricTransform impostata su 'similarity'. Questo significa che dice al sistema di cercare una trasformazione che includa solo spostamento (traslazione), rotazione e ridimensionamento (scala), ma non distorsioni prospettiche. Sotto il cofano, utilizza l'algoritmo MSAC (una variante avanzata del RANSAC). Questo algoritmo prende punti a caso, calcola una trasformazione di prova e scarta automaticamente tutti i punti di abbinamento che non seguono quella logica (gli outlier)Il risultato è tform, un oggetto che contiene la matrice di trasformazione esatta per allineare le immagini

- Estrazione di Angolo e Scala (Step 5): avendo calcolato la trasformazione, il codice usa un po' di trigonometria per estrarre i valori esatti di quanto l'immagine era stata ingrandita/rimpicciolita e ruotata. Poiché la trasformazione calcolata va dall'immagine distorta all'originale, il codice prende la matrice inversa (Tinv).

$$T_{inv} = \begin{bmatrix} sc & -ss & 0 \\ ss & sc & 0 \\ tx & ty & 1 \end{bmatrix}$$

  Dove i valori sono definiti come:

  - $sc = s \cdot \cos(\theta)$
  - $ss = s \cdot \sin(\theta)$
  - s è il fattore di scala
  - θ è l'angolo di rotazione

  Il codice estrae sc e ss dalla matrice e calcola la scala e l'angolo (convertito in gradi) stampandoli nella console

- Ripristino e Salvataggio (Step 6): l'ultimo passaggio è puramente applicativo; Si usa la funzione "imwarp" per applicare la trasformazione geometrica tform trovata. Applica questa trasformazione all'immagine a colori originale (distorted.rgb), non a quella in scala di grigi usata per i calcoli. In questo modo, crea una nuova immagine (recovered) che è l'immagine a colori ma perfettamente ruotata e scalata per combaciare con l'infrarosso. Infine mostra un confronto finale(nel caso del nostro studio si ottiene l'immagine 3.12) a

schermo (imshowpair) e salva l'immagine allineata sul disco rigido con il nome "filerisultato.bmp" usando la funzione imwrite. In Python, lo standard di fatto per questo tipo di operazioni è ,come detto, la libreria OpenCV (cv2), spesso usata in combinazione con NumPy e Matplotlib: il codice che otteniamo come corrispondente di quello in MATLAB è quello in figura 3.2



Figura 3.12: Confronto finale

Le differenze principali tra il nostro MATLAB e il nostro Python sono

- Gestione dei Falsi Positivi: In OpenCV, è prassi comune usare il Ratio Test di Lowe (il ciclo for sui matches). Serve a filtrare i punti sbagliati prima di passarli all'algoritmo geometrico, rendendo il tutto molto più veloce e robusto. MATLAB lo fa automaticamente dietro le quinte in alcune funzioni.
- Matrice di Trasformazione: MATLAB calcolava la trasformazione in un verso, poi faceva l'inversa tform.invert.T. OpenCV, usando estimateAffinePartial2D(src, dst), calcola direttamente la matrice giusta per andare dall'immagine distorta all'originale. Non c'è bisogno di invertirla manualmente.
- Spazio e Colore: OpenCV quando legge immagini a colori con imread, le carica nel formato BGR (Blu-Verde-Rosso) invece del classico RGB. Quando salviamo l'immagine con imwrite non ci sono problemi, ma per visualizzarla a schermo con Matplotlib va inserita una piccola conversione altrimenti i colori sembrerebbero inaccurati.

Abbiamo poi, modificando il codice, trovato una corrispondenza grafica tra le due immagini che carichiamo per vedere se il comando che abbiamo dato funziona, ovvero se da un immagine originale riesco a trovare il matching corretto tra i punti in una immagine dello stesso soggetto ma distorta, inclinata, ruotata, . . . , che corrispondono all'indagine fatta riferita a 3.103.113.2.

Capitolo 4

Stima dell'asse di simmetria calcolato a partire dai gruppi di features

Quanto discusso nei capitoli precedenti riguardo al feature matching e alle relative soluzioni adottate ha lo scopo di calcolare l'asse di simmetria del veicolo, sfruttando i rispettivi gruppi di caratteristiche (features). In questo modo si ottiene un riferimento geometrico preciso su cui posizionare loghi e scritte in rilievo. Questa sezione si concentra sull'individuazione e sull'analisi di un comando specifico, utile per identificare i punti corrispondenti a una determinata feature all'interno di un'immagine e per calcolarne successivamente i rispettivi punti specchiati.

4.1 Flip and Match in MATLAB

Per trovare punti corrispondenti che siano l'uno lo specchiato dell'altro all'interno della stessa immagine, il comando principale rimane matchFeatures, ma lo useremo all'interno di una strategia chiamata "Flip and Match". MATLAB non ha un comando singolo come "findMirrorPoints" perché per il computer un punto specchiato ha un aspetto diverso (una "L" diventa una "J" ad esempio), tuttavia quello che andiamo a fare è ingannare l'algoritmo.

Il concetto di base del rilevamento della simmetria nelle immagini del mondo reale si basa sull'idea semplice di abbinare coppie simmetriche di punti caratteristici. Ci sono molti metodi robusti ed efficienti per il rilevamento dei punti di interesse nel moderno campo dell'elaborazione delle immagini . L'importanza della simmetria di ciascuna coppia viene valutata sulla base della posizione, dell'orientamento e, opzionalmente, della scala delle caratteristiche. I metodi per il rilevamento dei punti caratteristici spesso producono un elenco di record relativi ai punti di interesse rilevati. Ogni record rappresenta un punto caratteristico e di solito contiene un quartetto di valori: coordinate dell'immagine x e y, orientamento ϕ e scala s del punto. Gli ultimi due valori di orientamento e scala sono normalizzati per garantire l'indipendenza di questi parametri. Un insieme di punti caratteristici invarianti a orientamento e scala, rilevati in modo robusto ed efficace e con alta ripetibilità, è un input molto adatto per il rilevamento delle simmetrie. Esistono due principali classi di simmetria nelle immagini del mondo reale. La prima è la cosiddetta simmetria diretta, che riguarda le coppie di punti caratteristici simili solo sotto trasformazioni

traslazionali e rotazionali. Un semplice esempio di simmetria diretta è una ruota di automobile. I punti caratteristici simili sono distribuiti intorno al perimetro della ruota. La cosiddetta simmetria speculare è l'altro tipo di simmetria generale, che riguarda coppie di punti caratteristici diversi dove il primo punto corrisponde solo alla versione speculare dell'altro. I volti umani e le ali delle farfalle sono buoni esempi di simmetria speculare, perché quasi ogni punto caratteristico rilevato ha il suo gemello speculare sull'altro lato dell'oggetto. Un descrittore del punto caratteristico speculare è definito come un descrittore di una copia speculare del frammento locale dell'immagine associato al punto caratteristico originale. Le coppie corrispondenti di punti caratteristici generano poi un insieme di coppie abbinate per il passo successivo di elaborazione.

Un punto caratteristico è simile all'idea di punto di interesse, cioè ogni punto caratteristico può essere definito come un pixel con una piccola area locale molto diversa dall'intorno più vicino. I punti caratteristici vengono generalmente calcolati tramite un metodo robusto e invariante alla rotazione con alta ripetibilità, come il metodo SIFT. Oltre ai valori di coordinata x e y del punto caratteristico, l'orientamento ϕ di ciascun punto caratteristico deve sempre essere determinato in relazione al valore di scala s, che è importante solo se si utilizza un metodo invariabile alla scala per il rilevamento dei punti caratteristici. Ciò nonostante, si assume che l'i-esimo punto di interesse nell'immagine contenga generalmente tutti e quattro i valori, formando così un vettore di punti $\mathbf{p}_i = (x_i, y_i, \phi_i, s_i)$. Il vettore $\mathbf{p}_i$ descrive in modo univoco la posizione, l'orientamento e la scala dell'i-esimo punto di interesse rilevato nell'immagine. Inoltre, per ogni vettore $\mathbf{p}_i$ viene solitamente calcolato un descrittore, indicato con $\mathbf{k}_i$. Tali descrittori rappresentano l'aspetto delle porzioni locali dell'immagine attorno alle coordinate x_i e y_i. Esistono molti metodi di descrizione delle caratteristiche progettati per scopi di corrispondenza (matching) e basati su teorie molto diverse.

Il caso della simmetria speculare viene indicato come bilaterale per enfatizzare la somiglianza tra un determinato punto di interesse $\mathbf{p}_i$ in un'immagine e un secondo punto di interesse $\mathbf{p}_j$ nella stessa immagine, ma ribaltato rispetto a un dato asse. Ciò significa che due punti di interesse sono trattati come una coppia di caratteristiche $(\mathbf{p}_i, \mathbf{p}_j)$ solo se la porzione di immagine originale del primo punto corrisponde alla porzione di immagine specchiata dell'altro. Esiste un unico vero punto di interesse $\mathbf{p}_j$ correlato all'altro punto di interesse $\mathbf{p}_i$ nell'immagine contenente un oggetto dotato di simmetria speculare. Per scopi di corrispondenza (matching), per ogni descrittore noto $\mathbf{k}_i$ viene calcolato un descrittore specchiato $\mathbf{m}_i$. Per ottenere una versione specchiata della porzione di immagine originale possiamo scegliere un asse di riflessione arbitrario grazie alla normalizzazione dell'orientamento. La costruzione delle versioni normalizzate e specchiate del punto di interesse originale è mostrata schematicamente in figura 4.1. La formazione della caratteristica specchiata $\mathbf{m}_i$ nella figura menzionata è solo illustrativa. In realtà, esistono due modi fondamentalmente diversi per calcolare i descrittori delle caratteristiche specchiate $\mathbf{m}_i$. Il primo approccio consiste semplicemente nel ribaltare la porzione di immagine originale relativa al

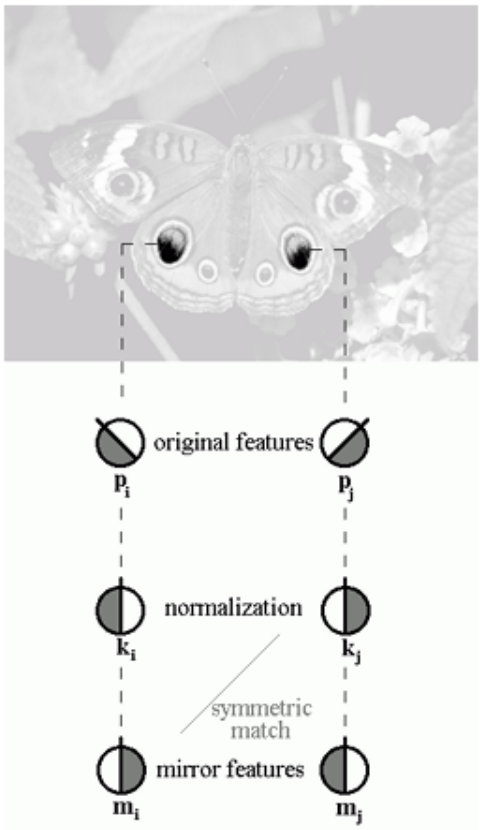


Figura 4.1: rilevamento di una coppia di punti [23]

descrittore $\mathbf{k}_i$ rispetto all'asse x o y e calcolare un descrittore (specchiato) aggiuntivo $\mathbf{m}_i$. Questo approccio è chiaramente molto semplice e ingenuo, ma non è sempre efficiente

L'altro approccio richiede la conoscenza della struttura del descrittore di caratteristiche $\mathbf{k}_i$, poiché il descrittore specchiato $\mathbf{m}_i$ viene generato direttamente modificando tutti i valori appropriati in $\mathbf{k}_i$. Chiaramente, questo metodo è sostanzialmente più efficiente rispetto al precedente, ma non è così immediato. Inoltre, è utilizzabile solo con una struttura del descrittore conveniente; ad esempio, il descrittore SIFT soddisfa pienamente queste esigenze. Il passaggio di elaborazione successivo consiste nel riconoscere le caratteristiche potenzialmente simmetriche e nel formare un insieme di coppie $(\mathbf{p}_i, \mathbf{p}_j)$ di punti di interesse correlati rispettivamente ai descrittori $\mathbf{k}_i$ e $\mathbf{m}_j$. L'abbinamento del descrittore $\mathbf{k}_i$ con il descrittore $\mathbf{m}_j$ produce lo stesso risultato dell'abbinamento del descrittore $\mathbf{k}_j$ con il descrittore $\mathbf{m}_i$. Ne consegue che la coppia $(\mathbf{p}_i, \mathbf{p}_j)$ è sempre formalmente intercambiabile con la coppia invertita $(\mathbf{p}_j, \mathbf{p}_i)$, poiché sono realmente equivalenti. Il grado di simmetria di una coppia arbitraria $(\mathbf{p}_i, \mathbf{p}_j)$

viene quantificato dall'ampiezza di simmetria M_{ij}, definita come:

$$M_{ij} = \begin{cases} \Phi_{ij} \cdot S_{ij} \cdot D_{ij} & \text{se } \Phi_{ij} > 0 \\ 0 & \text{altrimenti} \end{cases} \quad (1)$$

dove Φ_{ij} indica il peso della simmetria angolare, S_{ij} indica il peso della scala e infine D_{ij} indica il peso della distanza. In accordo con l'equazione precedente, il valore di simmetria di ciascuna coppia $(\mathbf{p}_i, \mathbf{p}_j)$ viene valutato come una funzione della posizione relativa dei punti di interesse nell'immagine, del loro orientamento e della loro scala. A questo punto, a ogni coppia $(\mathbf{p}_i, \mathbf{p}_j)$ di tutti i punti di interesse rilevati nell'immagine è stata assegnata la corrispondente ampiezza di simmetria M_{ij}. Maggiore è la somiglianza dei descrittori $\mathbf{k}_i$ e $\mathbf{m}_j$, maggiore diventa il valore M_{ij}. Ciò significa che l'ampiezza di simmetria quantifica la forza della simmetria di una singola coppia di punti di interesse. Infine, è possibile generare una mappa di simmetria accumulando i singoli valori dell'ampiezza di simmetria M_{ij} per tutti i punti di interesse rilevati. Il processo di accumulazione viene eseguito nello stile Hough di uno spazio di votazione. Ogni coppia di caratteristiche corrispondenti in un'immagine rappresenta un piccolo contributo di un asse di simmetria immaginario. Più precisamente, i due punti scuri nell'immagine 4.1 definiscono in modo univoco il loro punto medio, che si trova sull'asse maggiore della farfalla. In questo modo, l'asse di simmetria dominante nell'immagine può essere calcolato tramite la trasformata lineare di Hough applicata a tutti i punti medi delle coppie simmetriche. Infatti, ogni coppia simmetrica esprime un voto in un accumulatore di Hough ponderato dalla corrispondente magnitudine di simmetria M_{ij}. L'accumulatore viene poi elaborato con una procedura di soppressione dei non-massimi e gli assi di simmetria dominanti vengono quindi rilevati come picchi isolati.

Il rilevamento della simmetria rotazionale invece (talvolta radiale) è leggermente meno complicato rispetto alla simmetria speculare, poiché non vi è alcun bisogno di generare la versione specchiata $\mathbf{m}_i$ del descrittore di caratteristiche originale $\mathbf{k}_i$. La simmetria rotazionale viene rilevata direttamente confrontando ciascun descrittore di caratteristiche $\mathbf{k}_i$ con tutti i descrittori rimanenti $\mathbf{k}_j$. Se i vettori normalizzati di una coppia arbitraria di punti di interesse $(\mathbf{p}_i, \mathbf{p}_j)$ non sono paralleli, esiste un punto rispetto al quale essi sono simmetrici per rotazione. Inoltre, se nell'immagine sono presenti più coppie di questo tipo, esse definiscono un punto centrale della simmetria rotazionale.

Sostanzialmente l'idea di base del Flip&Match non è confrontare due immagini distinte, ma cercare corrispondenze all'interno della stessa immagine tra le sue parti originali e le loro versioni "ribaltate" (specchiate):

- Estrazione delle features: si estraggono i punti di interesse (keypoints) dall'immagine originale (usando algoritmi come SIFT, ORB o FAST).
- Generazione delle features specchiate: per ogni feature estratta, se ne crea una

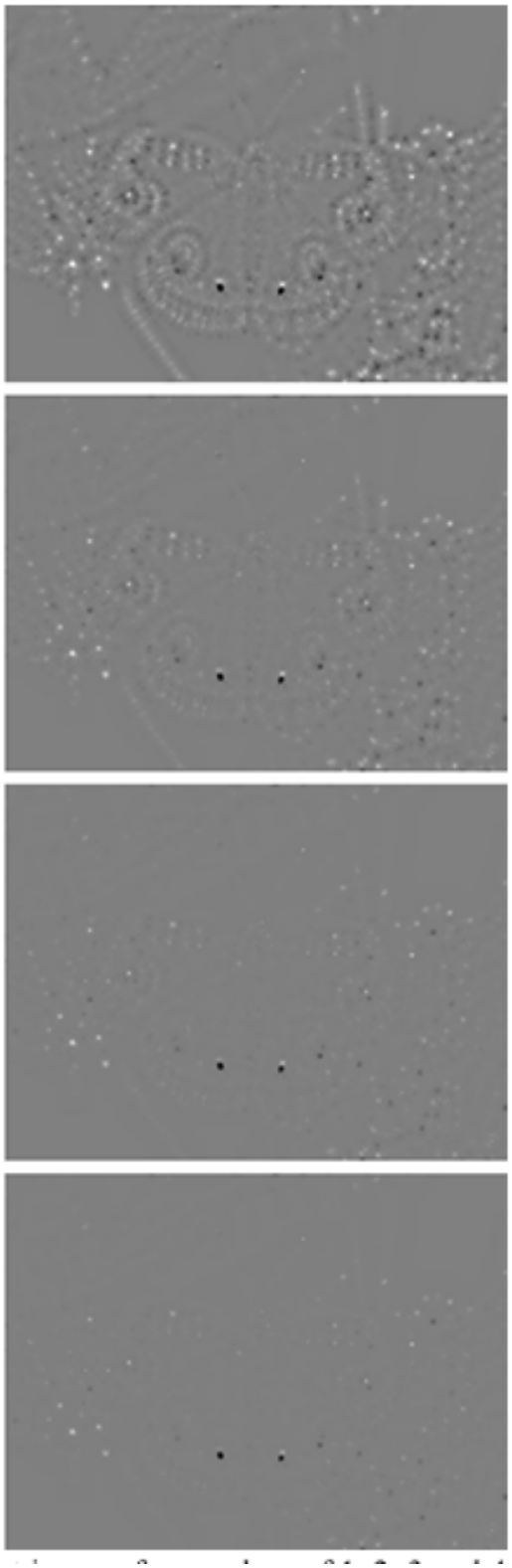


Figura 4.2: Mappa di simmetria [23]

"virtuale" o "specchiata" riflettendo il descrittore o il patch di immagine rispetto a un asse di simmetria ipotetico.

- Matching: si tenta di far corrispondere i descrittori originali con quelli specchiati. Se una feature originale in una posizione A corrisponde bene a una feature specchiata in posizione B, significa che il punto B è il riflesso del punto A rispetto all'asse di simmetria.
- Votazione (Hough Transform): tutte le coppie di punti che mostrano una corrispondenza valida "votano" per una posizione e un orientamento dell'asse di simmetria all'interno di uno spazio di accumulazione (spesso una Hough Space). L'asse che riceve il maggior numero di voti è considerato l'asse di simmetria del soggetto (nel tuo caso, la vettura)[23]

Nel nostro caso di studio confrontiamo l'immagine originale con una sua versione ribaltata. Se un punto nell'originale è simile a un punto nella versione ribaltata, allora abbiamo trovato una simmetria.

4.1.1 Costruzione del codice in ambiente MATLAB

- **Preparazione delle immagini:** caricare l'immagine originale e generarne una versione specchiata (ad esempio, lungo l'asse orizzontale).

```
I = imread('immagine.jpg');
I_flip = flip(I, 2); % Ribaltamento orizzontale
```

- **Estrazione delle feature:** estrarre i punti di interesse e i relativi descrittori da entrambe le immagini.

```
points = detectSURFFeatures(I);
points_flip = detectSURFFeatures(I_flip);

[feat, valid_points] = extractFeatures(I, points);
[feat_flip, valid_points_flip] = extractFeatures(I_flip, points_flip);
```

- **Matching:** utilizzare la funzione `matchFeatures` per identificare le corrispondenze tra l'immagine originale e quella specchiata.

```
indexPairs = matchFeatures(feat, feat_flip, 'Unique', true, 'MaxRatio', 0.8);

matchedPoints = valid_points(indexPairs(:,1));
matchedPointsFlip = valid_points_flip(indexPairs(:,2));
```

- **Riporto delle coordinate:** i punti in `matchedPointsFlip` hanno coordinate riferite all'immagine ribaltata. Per ricondurli al sistema di riferimento originale, è necessario trasformare la coordinata x. Data una larghezza dell'immagine W, la relazione è:

$$x_{originale} = W - x_{flipped} + 1$$

  Implementazione in MATLAB:

```
W = size(I, 2);
locsFlip = matchedPointsFlip.Location;
locsFlip(:,1) = W - locsFlip(:,1) + 1; % Trasformazione coordinata X
```

Riassumendo i comandi da usare in sequenza:

- flip: per creare la versione speculare
- detect...Features (es. detectSIFTFeatures): per trovare i punti
- extractFeatures: per ottenere i descrittori
- matchFeatures: Per trovare le coppie specchiate

4.2 Flip and Match in Python

In Python, il punto di riferimento per la Computer Vision è, come detto in precedenza, la libreria OpenCV (cv2).

Il passaggio da MATLAB a Python è piuttosto diretto, anche se OpenCV richiede qualche riga di codice in più per la gestione esplicita degli oggetti.

Operazione (MATLAB)	**Corrispettivo Python (OpenCV + NumPy)**
`flip(I, 2)`	`cv2.flip(img, 1)` oppure `img[:, ::-1]`
`detectSURFFeatures`	`sift = cv2.SIFT_create()` poi `sift.detect()`
`extractFeatures`	`sift.compute()` (o `detectAndCompute` per farli insieme)
`matchFeatures`	`cv2.BFMatcher()` o `cv2.FlannBasedMatcher()`
`showMatchedFeatures`	`cv2.drawMatches()`

Tabella 4.1: Corrispondenze tra operazioni MATLAB e Python

Come esempio pratico del Flip&Match in Python prendiamo il codice in figura 4.1

```
import cv2
import numpy as np

# 1. Carica l'immagine e crea la versione specchiata
img = cv2.imread('immagine.jpg', 0) # Carica in scala di grigi
h, w = img.shape
img_flip = cv2.flip(img, 1) # 1 = flip orizzontale

# 2. Inizializza il detector (SIFT)
sift = cv2.SIFT_create()

# 3. Trova punti chiave e descrittori
kp1, des1 = sift.detectAndCompute(img, None)
kp2, des2 = sift.detectAndCompute(img_flip, None)
```

```
# 4. Matching con BFMatcher (Brute Force)
bf = cv2.BFMatcher()
# Usiamo knnMatch per poter applicare il Ratio Test di Lowe
matches = bf.knnMatch(des1, des2, k=2)

# 5. Applica il Ratio Test (il MaxRatio di MATLAB)
good_matches = []
for m, n in matches:
    if m.distance < 0.75 * n.distance:
        good_matches.append(m)

# 6. Trasforma le coordinate dei punti specchiati
# In Python, i punti sono oggetti 'KeyPoint'. Dobbiamo estrarre .pt (x, y)
points_orig = []
points_mirrored = []
for m in good_matches:
    # Punto nell'immagine originale
    pt_orig = kp1[m.queryIdx].pt

    # Punto nell'immagine flipped
    pt_flip = kp2[m.trainIdx].pt

    # Riporta il punto flipped alle coordinate dell'originale
    # x_originale = Larghezza - x_flipped
    pt_mirrored_back = (w - pt_flip[0], pt_flip[1])

    points_orig.append(pt_orig)
    points_mirrored.append(pt_mirrored_back)

# 7. Visualizzazione (opzionale, disegna i match classici)
res = cv2.drawMatches(img, kp1, img_flip, kp2, good_matches, None, flags=2)
cv2.imshow('Symmetry Detection', res)
cv2.waitKey(0)
```

Listing 4.1: Flip&Match in Python

Adottando opportuni accorgimenti nel codice modello in figura 4.1 per ottimizzare il risultato ottenuto e immettendo gli input opportuni al nostro studio, un primo risultato è visibile nell'immagine 4.3



Figura 4.3: Risultato in Python

dove

- Punti Verdi: sono le feature "reali" rilevate da SIFT sulla nostra immagine.
- Punti Rossi: sono i punti corrispondenti trovati nell'immagine ribaltata, ma disegnati sulla posizione corretta della nostra immagine originale.
- Linee Blu: uniscono le coppie. In una foto perfettamente frontale, queste linee saranno perfettamente orizzontali.
- Linea Gialla: è l'asse verticale calcolato come valore mediano di tutti i centri di quelle linee blu.

Come visibile in figura 4.3 ci sono numerosi errori nel risultato ottenuto.

Problema relativo ai punti specchiati

Per capire se i punti sono corretti ("inliers") o errati ("outliers"), dobbiamo osservare la geometria delle coppie che abbiamo appena graficato. In una simmetria speculare verticale, la matematica ci impone delle regole rigide: se una coppia le viola, è quasi certamente un errore. I 3 criteri principali con cui analizziamo il risultato ottenuto sono:

- L'Orizzontalità delle linee (Il test più rapido): poiché abbiamo ribaltato l'immagine orizzontalmente, ogni punto $P_1(x_1, y_1)$ deve avere il suo corrispettivo $P_2(x_2, y_2)$ alla stessa altezza. Considereremo giuste le linee blu che uniscono il punto verde e quello rosso in maniera perfettamente orizzontale, mentre considereremo sbagliate le linee blu oblique o diagonali.
- L'Equidistanza dall'asse: l'asse giallo deve trovarsi esattamente nel punto medio di ogni linea blu. Considereremo giusto se il punto verde e il punto rosso sono alla stessa distanza dall'asse giallo, mentre considereremo sbagliato se un punto è molto vicino all'asse e l'altro è molto lontano.
- Coerenza della struttura: i punti dovrebbero formare una sorta di "riflesso" logico. Se vediamo un punto verde sullo specchietto sinistro, dovremmo avere

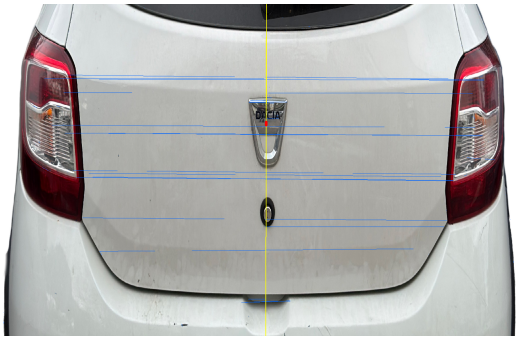


Figura 4.4: Risultato migliorato in Python

un punto rosso sullo specchietto destro. Se vediamo un punto rosso in un'area dove non c'è nulla di simmetrico (es. una macchia sull'asfalto solo da un lato), è un errore.

Possiamo quindi migliorare il codice per "pulire" i punti sbagliati integrando un doppio sistema di sicurezza: un filtro verticale (Y-check) che scarta automaticamente le coppie di punti che non si trovano alla stessa altezza (con una tolleranza di 5 pixel)(questo elimina quasi tutti i match "diagonali" errati) e una statistica robusta (Mediana), ovvero si utilizza la mediana per calcolare l'asse X, ignorando eventuali errori residui che sono sfuggiti al primo filtro 4.4.

Problema relativo all'input del codice

Quando si utilizza come input l'immagine di un veicolo distorto o inclinato, è necessario definire una metodologia che consenta di individuare l'asse di simmetria reale del segmento. In particolare, dopo aver estratto le feature e le rispettive controparti specchiate, occorre calcolare tale asse tenendo conto della distorsione prospettica dell'immagine, superando l'approssimazione di una vista ortogonale. L'obiettivo è ottenere questo risultato possibilmente senza ricorrere a un'esplicita procedura di calibrazione (ovvero senza la necessità di mappare i pixel in millimetri).

Il problema si sposta in questo caso dalla Geometria 2D alla Geometria Proiettiva: quando guardiamo un oggetto inclinato (distorto dalla prospettiva), le proprietà euclidee come la "perpendicolarità" e il "punto medio" spariscono. Un asse che nella realtà è perpendicolare, nell'immagine può apparire obliquo. Nel nostro caso, sia con il codice più generale che con quello migliorato del Flip&Match in ambiente Python, otteniamo due risultati errati 4.54.6. Per calcolare l'asse di simmetria reale (quello nel mondo 3D) partendo da un'immagine 2D distorta, abbiamo tre strade principali:

La strada Geometrica (Punti di Fuga)

In prospettiva, tutte le linee parallele tra loro convergono in un punto di fuga. Tutti i segmenti che uniscono i punti specchiati (P_i con P_i') sono, nella realtà, paralleli tra loro e perpendicolari all'asse. Se l'immagine è distorta, queste linee non saranno

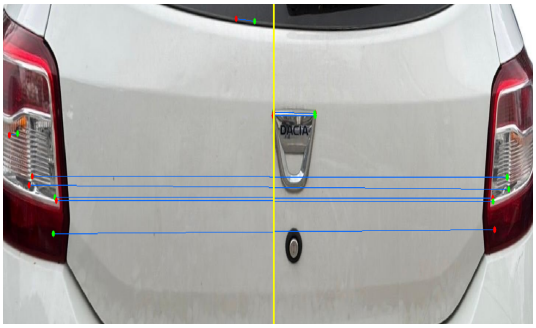


Figura 4.5: Risultato vettura distorta

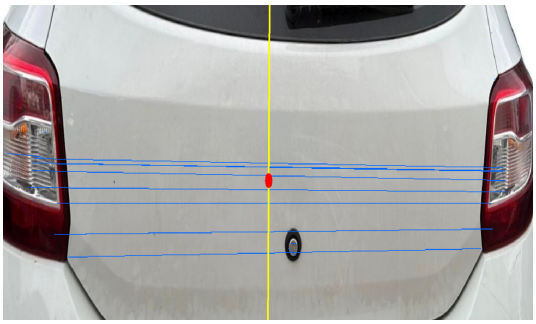


Figura 4.6: Risultato vettura inclinata 2

parallele nel 2D, ma convergeranno verso un punto di fuga V. L'asse di simmetria reale deve passare per il punto di fuga delle linee verticali (se l'asse è verticale) e deve rispettare la trasformazione proiettiva.

La strada della Rettificazione (Homography)

Questa è la tecnica più usata in Computer Vision. Invece di cercare di calcolarci l'asse sull'immagine storto, raddrizziamo l'immagine. Troviamo almeno 4 punti di cui conosciamo la geometria (es. i quattro angoli della targa di un'auto o le quattro ruote) e calcoliamo una matrice di omografia (H) che trasforma il piano distorto in un piano frontale (top-down view). Applichiamo poi la trasformazione:

```
img_rettificata = cv2.warpPerspective(img, H, ...)
```

Su questa nuova immagine, il punto medio e la perpendicolare tornano a essere validi.

La strada matematica (Vanishing Line)

Se non possiamo raddrizzare l'immagine, dobbiamo considerare che il punto medio in prospettiva non è a metà del segmento. Esiste un rapporto chiamato Cross-Ratio (Rapporto Incrociato) che si conserva nelle proiezioni.

Per calcolare l'asse reale senza raddrizzare tutto dobbiamo individuare la linea dell'orizzontale dell'immagine (definita dai punti di fuga). L'asse di simmetria reale

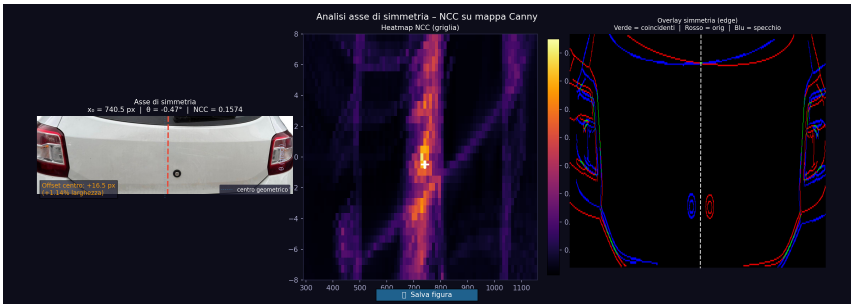


Figura 4.7: Risultato vettura inclinata

è la linea che passa per il punto di fuga verticale e taglia i segmenti secondo le regole della prospettiva.

L'approccio di MATLAB si basa sulla ricerca dell'asse di simmetria ottimale esplorando sia la posizione che l'inclinazione (per gestire la prospettiva), usando le mappe di edge come feature robuste. Il problema ha due gradi di libertà: la posizione orizzontale dell'asse x0 e la sua inclinazione θ (l'effetto prospettico ruota leggermente l'asse rispetto alla verticale).

Il cuore del metodo è la funzione symmetry_score: per ogni coppia (x0, θ) candidata, ogni pixel viene mappato nel suo corrispondente rispecchiato rispetto all'asse inclinato (con un campionamento bilineare per la sub-pixel accuracy), e la somiglianza tra le due metà viene misurata con la correlazione normalizzata NCC — che è robusta alle variazioni di luminosità tra i due lati.

Il flusso di esecuzione è:

- Pre-elaborazione : riduzione a scala di lavoro, blur gaussiano, estrazione della mappa di Canny. Lavorare sugli edge anziché sui pixel grezzi rende il metodo molto più stabile al rumore e alle differenze di illuminazione.
- Peso gaussiano verticale : le righe centrali dell'immagine (paraurti, targa, luci) vengono enfatizzate perché portano più informazione di simmetria rispetto al cielo o all'asfalto.
- Ricerca su griglia : si calcola lo score su una griglia 80x33 di candidati (x0, θ) con $\theta \in [-8°, +8°]$, sufficiente per coprire i tipici errori di inquadratura.
- Raffinamento con fminsearch : a partire dal massimo della griglia, l'ottimizzatore Nelder-Mead converge al minimo sub-pixel con precisione $\tilde{0}.1$ px.
- Visualizzazione a tre pannelli: l'immagine originale con l'asse, la heatmap dello score, e una sovrapposizione RGB che mostra quanto le due metà coincidono dopo il ribaltamento. Nel nostro caso di studio il risultato ottenuto è riportato in figura 4.7

Se la distorsione è nota (ad esempio sappiamo che l'auto è inclinata di 30 gradi), possiamo correggere le coordinate dei punti prima di calcolare l'asse.

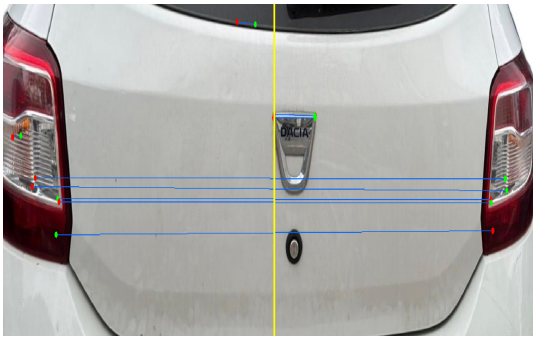


Figura 4.8: Risultato Rettificazione Proiettiva

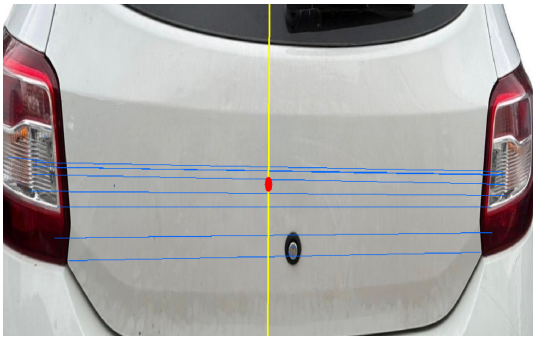


Figura 4.9: Risultato 2 Rettificazione Proiettiva

Se invece la distorsione è ignota, l'approccio più semplice è il RANSAC proiettivo: Invece di usare cv2.fitLine (che cerca una linea retta semplice), dovremmo usare le coppie di punti per stimare una trasformazione che renda i segmenti paralleli. Il codice precedente che aveva esiti visibilmente errati 4.54.6 fallisce perché assume che l'asse sia una retta verticale che taglia a metà segmenti orizzontali. In prospettiva, però, le linee che uniscono i punti specchiati non sono più parallele e l'asse non è più necessariamente verticale.

Per correggere questo senza dover conoscere la posizione della fotocamera, dobbiamo usare il concetto di Punto di Fuga e Rettificazione Proiettiva. Il risultato precedente 4.7 è solo un esempio dei tanti codici utilizzati in questa fase del nostro studio, di seguito mostriamo ulteriori risultati da codici leggermente modificati ma costruiti tutti sulla stessa linea di principio 4.84.94.104.114.12.

Nonostante i ripetuti tentativi di ottimizzazione dei vari script, i risultati ottenuti

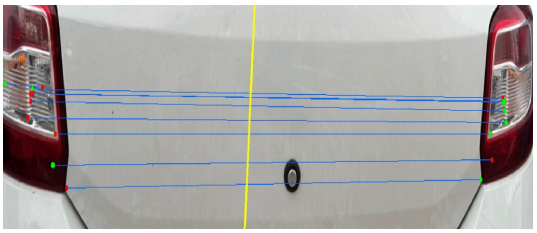


Figura 4.10: Risultato 3 Rettificazione Prospettica

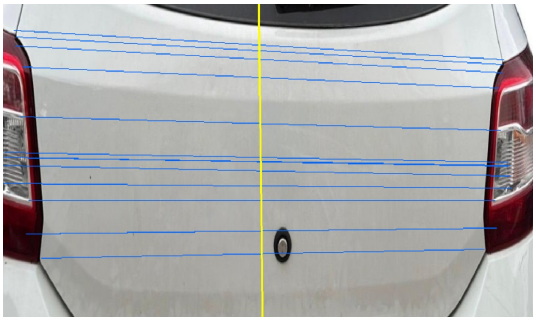


Figura 4.11: Risultato 4 Rettificazione Prospettica

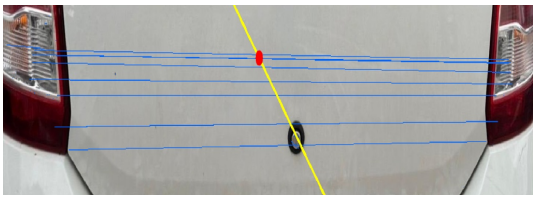


Figura 4.12: Risultato 5 Rettificazione Prospettica

si sono rivelati costantemente insoddisfacenti. Di conseguenza, l'approccio finora adottato e la struttura attuale del codice stanno evidenziando una complessità eccessiva, rendendo necessaria una revisione complessiva dell'impostazione.

4.3 Integrazione tra il modulo di feature matching e l'algoritmo di calcolo dell'asse.

In questa sezione l'obiettivo è integrare il modulo di feature matching, capace di operare su immagini distorte o inclinate, con l'algoritmo di calcolo dell'asse di simmetria. L'integrazione deve consentire al sistema di gestire correttamente la deformazione prospettica: anziché basarsi su un calcolo bidimensionale basato sul semplice punto medio (intrinsecamente impreciso in presenza di distorsioni), il sistema dovrà identificare l'asse di simmetria reale, considerando le deformazioni geometriche intrinseche all'immagine.

In termini di Computer Vision, quello che stiamo proponendo è una normalizzazione geometrica dell'immagine prima dell'analisi. Questo approccio risulta sensato perchè il codice che cerca l'asse di simmetria tramite il flip speculare soffre molto se l'auto è inclinata o vista in prospettiva, perché il concetto di "specchiata" funziona solo se l'asse è perfettamente verticale. Passiamo così alla costruzione di un codice costituito da

- un codice di allineamento: raddrizza l'auto rispetto a un'immagine di riferimento "dritta"
- un codice per l'asse di simmetria: trova l'asse su un'immagine ora perfettamente centrata

Tuttavia dobbiamo seguire una certa logica di integrazione rispetto a come li abbiamo presentati : partendo da un'immagine distorta fornita come input, il processo si sviluppa in due fasi principali. Nella prima fase di trasformazione, si utilizzano l'algoritmo SIFT e la funzione estimateAffinePartial2D per raddrizzare l'immagine distorta confrontandola con quella di riferimento . Successivamente, nella seconda fase dedicata alla simmetria, si opera sull'immagine così rettificata (memorizzata nella variabile recovered), eseguendo l'operazione di flip per ricercare e calcolare l'asse di simmetria prospettico. Questo approccio è superiore al precedente per diversi motivi:

- Riduzione del rumore: Il calcolo della pendenza nel primo codice è un'approssimazione geometrica per correggere la prospettiva "al volo". Se raddrizziamo l'immagine prima con SIFT, quella parte di codice diventa quasi inutile e l'asse sarà quasi perfettamente verticale
- Precisione SIFT: SIFT è molto robusto alle rotazioni. Usarlo per allineare l'auto a un modello di riferimento garantisce che il calcolo della simmetria avvenga in condizioni ideali
- Ispezione visiva: Se l'auto raddrizzata non sembra dritta a occhio nudo, sapremo subito che il problema è nel matching iniziale e non nel calcolo della simmetria

L'intenzione quindi è creare una pipeline di elaborazione a due stadi: il primo stadio corregge la geometria macroscopica (allineamento tramite SIFT con l'immagine di riferimento) e il secondo stadio affina la ricerca della simmetria locale (asse prospettico tramite auto-matching speculare). Questa combinazione è ottima perché SIFT raddrizza l'auto, ma il calcolo dell'asse di simmetria finale basato sui punti medi è molto più preciso per capire se l'auto è "centrata" rispetto a se stessa. Unendo i due codici otteniamo i seguenti vantaggi

- Doppio Filtro di Errore: SIFT raddrizza l'auto rispetto al database, ma il calcolo dell'asse (perpendicolare alla pendenza dei match speculari) corregge eventuali piccoli errori di prospettiva rimasti dopo l'allineamento
- Robustezza Statistica: usando la np.median sia per la posizione dell'asse che per la sua inclinazione, ignori i "falsi match" che SIFT potrebbe generare su riflessi della carrozzeria.
- Coerenza Geometrica: l'immagine finale che vediamo è "standardizzata". Questo significa che se dobbiamo misurare distanze o asimmetrie (es. danni alla carrozzeria), i pixel avranno un rapporto di scala costante.

Correggendo inizialmente la geometria macroscopica e ricercando poi la simmetria otteniamo anche in questo caso dievrsi risultati, basati su miglioramenti del codice modello in figura 4.2, che sembrano riuscire a calcolare l'asse di simmetria anche in ambiente prospettico 4.134.144.15.

```
import cv2
import numpy as np

# --- STEP 1: Allineamento con Omografia (Correzione Prospettica) ---
original = cv2.imread('autonologo.jpg', cv2.IMREAD_GRAYSCALE)
distorted_rgb = cv2.imread('vettura_inclinata_4.jpeg')
distorted_gray = cv2.cvtColor(distorted_rgb, cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT_create()
kp_ref, des_ref = sift.detectAndCompute(original, None)
kp_dist, des_dist = sift.detectAndCompute(distorted_gray, None)

bf = cv2.BFMatcher()
matches = bf.knnMatch(des_ref, des_dist, k=2)
good_matches = [m for m, n in matches if m.distance < 0.75 * n.distance]

if len(good_matches) > 10:
    # Estraiamo i punti per l'omografia
    src_pts = np.float32([kp_dist[m.trainIdx].pt for m in
        good_matches]).reshape(-1, 1, 2)
    dst_pts = np.float32([kp_ref[m.queryIdx].pt for m in
        good_matches]).reshape(-1, 1, 2)

    # --- MODIFICA CHIAVE: findHomography invece di Affine ---
    # RANSAC pulisce i match errati causati dai riflessi sulla carrozzeria
    H, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

    h, w = original.shape
    # --- MODIFICA CHIAVE: warpPerspective per gestire la matrice 3x3 ---
    img_aligned = cv2.warpPerspective(distorted_rgb, H, (w, h))

if len(good_matches) > 10:
    # Calcolo Omografia
    src_pts = np.float32([kp_dist[m.trainIdx].pt for m in
        good_matches]).reshape(-1, 1, 2)
    dst_pts = np.float32([kp_ref[m.queryIdx].pt for m in
        good_matches]).reshape(-1, 1, 2)
    H, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

    h, w = original.shape
    img_aligned = cv2.warpPerspective(distorted_rgb, H, (w, h))

    # --- NUOVA LOGICA: Centratura Dinamica ---
    # Prendiamo solo i punti 'inliers' (quelli buoni che hanno confermato
        l'omografia)
    inlier_pts = src_pts[mask.ravel() == 1]
```

```
    # Proiettiamo questi punti nell'immagine raddrizzata
    pts_transformed = cv2.perspectiveTransform(inlier_pts, H)

    # Troviamo il centro reale dell'auto basandoci sui punti trasformati
    x_min = np.min(pts_transformed[:, 0, 0])
    x_max = np.max(pts_transformed[:, 0, 0])
    centro_reale_x = (x_min + x_max) / 2

    # Se il centro dell'auto e' lontano dal centro dell'immagine (w/2),
        traslamo
    shift_x = (w / 2) - centro_reale_x
    M_shift = np.float32([[1, 0, shift_x], [0, 1, 0]])
    img_aligned = cv2.warpAffine(img_aligned, M_shift, (w, h))
    # ----------------------------------------
else:
    img_aligned = distorted_rgb.copy()

# --- STEP 2: Analisi Simmetria (Flip e Matching Speculare) ---
gray_aligned = cv2.cvtColor(img_aligned, cv2.COLOR_BGR2GRAY)
img_flip = cv2.flip(gray_aligned, 1)

kp_r, des_r = sift.detectAndCompute(gray_aligned, None)
kp_f, des_f = sift.detectAndCompute(img_flip, None)
matches_sym = bf.knnMatch(des_r, des_f, k=2)

punti_medi_x = []
punti_medi_y = []
pendenze = []

for m, n in matches_sym:
    if m.distance < 0.70 * n.distance:
        p1 = kp_r[m.queryIdx].pt
        p2 = (w - kp_f[m.trainIdx].pt[0], kp_f[m.trainIdx].pt[1])

        dx = p1[0] - p2[0]
        dy = p1[1] - p2[1]

        # Filtro nervoso per match orizzontali
        if abs(dy) < 10:
            punti_medi_x.append((p1[0] + p2[0]) / 2)
            punti_medi_y.append((p1[1] + p2[1]) / 2)
            if abs(dx) > 20:
                pendenze.append(dy / dx)

# --- STEP 3: Calcolo Asse Robusto ---
if len(punti_medi_x) > 4:
    x_mediana = np.median(punti_medi_x)
```

```
    y_mediana = np.median(punti_medi_y)
    m_blu = np.median(pendenze) if len(pendenze) > 0 else 0

    # Se m_blu e' quasi zero, l'asse e' perfettamente verticale
    if abs(m_blu) < 0.01:
        m_asse = -1 / m_blu
        x_top = x_mediana - (0 - y_mediana) / m_asse
        x_bot = x_mediana - (h - y_mediana) / m_asse
    else:
        x_top = x_bot = x_mediana

    # Disegno Risultati
    cv2.line(img_aligned, (int(x_top), 0), (int(x_bot), h), (0, 255, 0), 2)
        # Asse Verde
    cv2.circle(img_aligned, (int(x_mediana), int(y_mediana)), 7, (0, 0,
        255), -1) # Centro Rosso

# Visualizzazione
cv2.namedWindow('Raddrizzamento con Omografia', cv2.WINDOW_NORMAL)
cv2.imshow('Raddrizzamento con Omografia', img_aligned)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Listing 4.2: Codice modello



Figura 4.13: Risultato codici combinati



Figura 4.14: Risultato 2 codici combinati



Figura 4.16: input e risultato pe vettura distorta



Figura 4.15: Risultato 3 codici combinati

Si riesce quindi a calcolare l'asse di simmetria in maniera prospettica, anche mettendo in crisi il sistema, quindi ad esempio andando a sostituire al posto della vettura inclinata, l'immagine di una vettura dritta , da cui indipendentemente dall'input il sistema funziona e riesce a calcolare correttamente l'asse di simmetria.

Nota importante per il codice modello

Tenendo in considerazione il codice modello 4.2 risulta di fondamentale importanza la parte del codice in figura 4.3 per permettere la distorsione in ogni direzione.

```
# ... (mantenere lo Step 1 fino a warpPerspective) ...

if len(good_matches) > 10:
    # Calcolo Omografia
    src_pts = np.float32([kp_dist[m.trainIdx].pt for m in
        good_matches]).reshape(-1, 1, 2)
    dst_pts = np.float32([kp_ref[m.queryIdx].pt for m in
        good_matches]).reshape(-1, 1, 2)
    H, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
```



Figura 4.17: input e riusltato per vettura frontale



Figura 4.18: input e risultato per vettura molto distorta

```
    h, w = original.shape
    img_aligned = cv2.warpPerspective(distorted_rgb, H, (w, h))

    # --- NUOVA LOGICA: Centratura Dinamica ---
    # Prendiamo solo i punti "inliers" (quelli buoni che hanno confermato
        l'omografia)
    inlier_pts = src_pts[mask.ravel() == 1]
    # Proiettiamo questi punti nell'immagine raddrizzata
    pts_transformed = cv2.perspectiveTransform(inlier_pts, H)

    # Troviamo il centro reale dell'auto basandoci sui punti trasformati
    x_min = np.min(pts_transformed[:, 0, 0])
    x_max = np.max(pts_transformed[:, 0, 0])
    centro_reale_x = (x_min + x_max) / 2

    # Se il centro dell'auto e' lontano dal centro dell'immagine (w/2),
        trasliamo
    shift_x = (w / 2) - centro_reale_x
    M_shift = np.float32([[1, 0, shift_x], [0, 1, 0]])
    img_aligned = cv2.warpAffine(img_aligned, M_shift, (w, h))
    # ----------------------------------------
else:
    img_aligned = distorted_rgb.copy()

# ... (procedere con Step 2: Flip e Asse Simmetria) ...
```

Listing 4.3: parte di codice per le distorsioni

4.4 Ottimizzazione e modellazione di codici per un calcolo rigoroso dell'asse

Lo script Python ottenuto dalla combinazione di codici nella sezione precedente riesce a individuare l'asse di simmetria e a correggere con discreta efficacia le distorsioni prospettiche delle foto, a prescindere dalla loro direzione. Ciononostante, il calcolo dell'asse rimane sempre un'approssimazione: prendendo come riferimento visivo la serratura del portellone posteriore, che rappresenta il vero centro dell'autovettura, si nota chiaramente che l'asse stimato non coincide mai con essa4.16 4.174.18.

Non è tuttavia possibile vincolare il sistema alla serratura come punto fisso, poiché non tutti i veicoli ne sono provvisti. Per comprendere l'origine di questa imprecisione, abbiamo analizzato direttamente i punti di interesse (keypoints) usati dal sistema per il feature matching da cui viene ricavato l'asse. Tornando indietro al calcolo dei punti sorgente, sono emersi due problemi principali

- il sistema prende punti non solo casuali ma piuttosto illogici

- applicando un algoritmo di filtraggio per eliminare i falsi positivi o i match dubbi, il numero di punti validi residui risulta insufficiente per calcolare un'interpolazione attendibile dell'asse

Di conseguenza, le cause della mancata accuratezza nella determinazione dell'asse di simmetria possono essere ricondotte a diversi fattori:

- errore di matching: le corrispondenze tra i punti sono errate, inficiando il successivo calcolo dell'asse;
- stima errata della distorsione: la lettura e la successiva correzione delle distorsioni prospettiche risultano imprecise;
- selezione inadeguata dei keypoint e scarsità di dati: il sistema non individua i punti realmente significativi e, a seguito del filtraggio dei falsi positivi, i punti residui risultano insufficienti per un'interpolazione attendibile.

Altro problema potrebbe essere la risoluzione dell'immagine data come input, che però ancora non consideriamo.

4.4.1 Analisi del codice

L'approccio attuale nasce da una revisione del codice precedente, le cui prestazioni iniziali erano ritenute accettabili, al fine di analizzare nel dettaglio la logica di elaborazione del sistema. La strategia consiste nel fornire in ingresso un'immagine contenente approssimativamente una metà del veicolo, per poi individuare i punti di corrispondenza (feature matching) rispetto alla metà specchiata, sfruttando le funzionalità di MATLAB.

- Lettura di due immagini
- Rilevamento delle feature tramite SURF
- Estrazione dei descrittori delle feature
- Corrispondenza delle feature (Feature matching)
- Recupero dei punti corrispondenti (Matched points)
- Visualizzazione dei punti corrispondenti figure

Si eseguono i diversi punti sopra elencati per individuare i punti delle feature di una metà nell'altra metà specchiata, visualizzare quelli corretti, per poi riportare la seconda metà della macchina, dove abbiamo fatto il matching e trovato i punti corrispondenti rispetto alla prima metà, nella sua posizione originaria, con i punti specchiati, che dovremmo ribaltare, e visualizzare così insieme ai punti della prima metà e individuare poi l'asse in base a questi keypoints.

Sulla base di questa impostazione, abbiamo avviato la conversione dei comandi da MATLAB a Python per analizzare il comportamento e le prestazioni del sistema con questo linguaggio di programmazione.

Il processo di sviluppo e coversione procede per blocchi funzionali:

- stima dell'asse su immagini frontali: in prima istanza, la determinazione dell'asse di simmetria viene testata su immagini frontali o con inclinazioni contenute. Per valutare la precisione dei diversi algoritmi candidati, confrontiamo l'inclinazione dell'asse calcolato rispetto all'asse verticale della foto
- integrazione della correzione prospettica: successivamente, l'algoritmo verrà esteso per identificare automaticamente le distorsioni, applicare il raddrizzamento dell'immagine e stimare l'asse finale (avvalendosi anche del supporto di risorse come MATLAB)

Il codice modello in figura 4.4 che sviluppiamo quindi per questa sezione presenta

- un sistema che consente in individuare l'asse di simmetria quando l'immagine è pressochè frontale
- un sistema che consente di individuare, rispetto a un'immagine data di riferimento, idealmente frontale, il raddrizzamento dell'immagine distorta o inclinata , e procede poi allo stesso modo del primo codice per l'individuazione dell'asse

```
import os
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, color, transform, feature, img_as_float
from scipy.ndimage import gaussian_filter, map_coordinates
from scipy.optimize import minimize
import tkinter as tk
from tkinter import filedialog

# ============================================================
#  TROVA_ASSE_SIMMETRIA.py
#  Trova l'asse di simmetria del posteriore di un'auto in un'immagine
# ============================================================

def trova_asse_simmetria(image_path=None):
    # --- 0. Caricamento immagine -------------------------------------
    print('\n--- AVVIO SCRIPT ---')
    if image_path is None or image_path == '':
        print('Nessun percorso fornito, in attesa di selezione manuale del
            file...')
        root = tk.Tk()
        root.withdraw() # Nasconde la finestra principale
```

```
    filetypes = (('Immagini', '*.jpg *.png *.jpeg *.bmp *.tif'),
        ('Tutti i file', '*.*'))
    image_path = filedialog.askopenfilename(title='Seleziona immagine
        posteriore auto', filetypes=filetypes)

    if not image_path:
        print('Operazione annullata dall\'utente.')
        return

print(f'[1/5] Caricamento immagine in corso: {image_path}')
img_orig = io.imread(image_path)
print(f'    -> Successo! Risoluzione originale: {img_orig.shape[1]} x
    {img_orig.shape[0]}')

# --- 1. Pre-elaborazione ----------------------------------------
print('\n[2/5] Avvio pre-elaborazione (scala di grigi, resize, edge
    map)...')
if img_orig.ndim == 3:
    img_gray = color.rgb2gray(img_orig)
else:
    img_gray = img_as_float(img_orig)

max_w = 600
H, W = img_gray.shape
scale = min(1.0, max_w / W)

if scale < 1.0:
    # Usiamo np.ceil per forzare l'arrotondamento verso l'alto come fa
        MATLAB
    img_small = transform.resize(img_gray, (int(np.ceil(H * scale)),
        int(np.ceil(W * scale))), anti_aliasing=True)
else:
    img_small = img_gray.copy()

Hs, Ws = img_small.shape
print(f'    -> Immagine ridimensionata per velocita: {Ws} x {Hs}
    (scala scale:.2f})')

sigma_blur = 1.5
img_blur = gaussian_filter(img_small, sigma=sigma_blur)
edge_map = feature.canny(img_blur, low_threshold=0.05,
    high_threshold=0.15)
edge_map = edge_map.astype(float)

y_center = Hs / 2.0
y_sigma = Hs * 0.35
Y_col = np.arange(Hs).reshape(-1, 1)
```

```
y_weight = np.exp(-((Y_col - y_center)**2) / (2 * y_sigma**2))
edge_weighted = edge_map * y_weight
print('     -> Pre-elaborazione completata.')

# --- 2. Griglia di ricerca ----------------------------------------
x_range = np.linspace(0.20 * Ws, 0.80 * Ws, 33)
ang_range = np.linspace(-8, 8, 33)
score_grid = np.zeros((len(ang_range), len(x_range)))

print(f'\n[3/5] Avvio ricerca su griglia sull\'asse...')
print(f'     -> Calcolo di {len(x_range)} posizioni x {len(ang_range)}
    angoli = {score_grid.size} combinazioni totali.')

for ai, a in enumerate(ang_range):
    # Stampa il progresso ogni 5 step per far capire che il codice sta
        lavorando
    if ai % 5 == 0 or ai == len(ang_range) - 1:
        print(f'     ... Processando angolo {ai+1}/{len(ang_range)}
            (theta: {a:.1f} gradi)')

    for xi, x0 in enumerate(x_range):
        score_grid[ai, xi] = symmetry_score(edge_weighted, x0, a)

ai_best, xi_best = np.unravel_index(np.argmax(score_grid),
    score_grid.shape)
x0_init = x_range[xi_best]
ang_init = ang_range[ai_best]
print(f'     -> Griglia conclusa. Massimo provvisorio: x0={x0_init:.1f}
    px, theta={ang_init:.2f} gradi')

# --- 3. Raffinamento con ottimizzatore
    ----------------------------------------
print('\n[4/5] Avvio raffinamento con ottimizzatore Nelder-Mead...')

def obj_fun(p):
    return -symmetry_score(edge_weighted, p[0], p[1])

def callback_opt(x_curr):
    print(f'     ... Affinamento in corso: x0={x_curr[0]:.2f},
        theta={x_curr[1]:.2f}')

options = {'xatol': 0.1, 'fatol': 1e-6, 'maxiter': 500, 'disp': False}
res = minimize(obj_fun, [x0_init, ang_init], method='Nelder-Mead',
    options=options, callback=callback_opt)

x0_opt = res.x[0]
ang_opt = res.x[1]
```

```
x0_orig = x0_opt / scale
ang_orig = ang_opt

print('\n=== RISULTATO FINALE ===')
print(' Asse di simmetria:')
print(f' Posizione orizzontale : x0 = {x0_orig:.1f} px (immagine orig)')
print(f' Inclinazione        : theta = {ang_orig:.3f} gradi')
print(f'   (theta > 0 -> asse inclinato a destra, theta < 0 -> asse
    inclinato a sinistra)\n')

# --- 4. Visualizzazione ----------------------------------------
print('[5/5] Preparazione della visualizzazione dei grafici in
    corso...')

fig1 = plt.figure('Asse di Simmetria - Posteriore Auto', figsize=(15,
    5))
fig1.patch.set_facecolor('white')
ax1 = plt.subplot(1, 3, 1)
ax1.imshow(img_orig)
disegna_asse(ax1, x0_orig, ang_orig, img_orig.shape[0], 'yellow', 2.5)
ax1.set_title('Asse di simmetria (rotato x0={x0_orig:.1f} px,
    theta={ang_orig:.2f} gradi)', fontsize=11)
ax1.set_xlabel(f'x_0 = {x0_orig:.1f} px, theta = {ang_orig:.2f} gradi')

ax2 = plt.subplot(1, 3, 2)
ax2.imshow(edge_weighted, cmap='gray')
disegna_asse(ax2, x0_opt, ang_opt, Hs, 'red', 1.5)
ax2.set_title('Edge map pesata (spazio di lavoro)', fontsize=11)
ax2.set_xlabel('Regione enfatizzata al centro verticale')

ax3 = plt.subplot(1, 3, 3)
extent = [x_range[0]/scale, x_range[-1]/scale, ang_range[-1],
    ang_range[0]]
im = ax3.imshow(score_grid, aspect='auto', extent=extent,
    origin='lower', cmap='hot')
plt.colorbar(im, ax=ax3)
ax3.plot(x0_orig, ang_orig, 'c+', markersize=14, markeredgewidth=2,
    label='Ottimo')
ax3.set_xlabel('x_0 [px immagine originale]')
ax3.set_ylabel('theta [gradi]')
ax3.set_title('Score di simmetria (griglia)', fontsize=11,
    fontweight='bold')
ax3.legend(loc='lower right')

plt.suptitle('Ricerca asse di simmetria - effetto prospettico
    corretto', fontsize=13, fontweight='bold')
```

```
    plt.tight_layout()

    # --- 5. Immagine specchiata per verifica visiva
        ----------------------------------------
    fig2 = plt.figure('Verifica simmetria', figsize=(12, 5))
    fig2.patch.set_facecolor('white')

    img_gray_orig = color.rgb2gray(img_orig) if img_orig.ndim == 3 else
        img_as_float(img_orig)
    img_mirrored_orig = rispecchia_immagine(img_gray_orig, x0_orig,
        ang_orig)

    ax4 = plt.subplot(1, 2, 1)
    ax4.imshow(img_orig)
    disegna_asse(ax4, x0_orig, ang_orig, img_orig.shape[0], 'yellow', 2)
    ax4.set_title('Originale + asse', fontsize=11)

    ax5 = plt.subplot(1, 2, 2)
    overlay = np.stack([img_gray_orig, (img_mirrored_orig +
        img_gray_orig)/2, img_mirrored_orig], axis=-1)
    ax5.imshow(overlay)
    disegna_asse(ax5, x0_orig, ang_orig, img_orig.shape[0], 'yellow', 2)
    ax5.set_title('Sovrapposizione (rosso) - spec. (blu)', fontsize=11)

    print('     -> Finito! Visualizzazio ne grafici su schermo (chiudi le
        finestre grafiche per terminare lo script).')
    plt.show()

# ============================================================
#  FUNZIONI DI SUPPORTO
# ============================================================

def symmetry_score(img, x0, theta_deg):
    H, W = img.shape
    theta = np.deg2rad(theta_deg)
    tan_t = np.tan(theta)

    X, Y = np.meshgrid(np.arange(W), np.arange(H))
    x_rel = (X - x0) - tan_t * (Y - H/2.0)
    X_mirror = X - 2 * x_rel
    Y_mirror = Y

    valid = (X_mirror >= 0) & (X_mirror <= W - 1) & (Y_mirror >= 0) &
        (Y_mirror <= H - 1)

    coords = np.array([Y_mirror.ravel(), X_mirror.ravel()])
    img_flip_flat = map_coordinates(img, coords, order=1, mode='constant',
```

```
        cval=0.0)
    img_flip = img_flip_flat.reshape(H, W)

    A = img[valid]
    B = img_flip[valid]

    if A.size < 100:
        return 0.0

    A = A - np.mean(A)
    B = B - np.mean(B)

    denom = np.sqrt(np.sum(A**2) * np.sum(B**2))
    if denom < 1e-10:
        return 0.0
    else:
        return np.sum(A * B) / denom

def rispecchia_immagine(img, x0, theta_deg):
    H, W = img.shape
    theta = np.deg2rad(theta_deg)
    tan_t = np.tan(theta)

    X, Y = np.meshgrid(np.arange(W), np.arange(H))
    x_rel = (X - x0) - tan_t * (Y - H/2.0)
    X_mirror = X - 2 * x_rel

    coords = np.array([Y.ravel(), X_mirror.ravel()])
    img_out_flat = map_coordinates(img, coords, order=1, mode='constant',
        cval=0.0)
    return img_out_flat.reshape(H, W)

def disegna_asse(ax, x0, theta_deg, H, colore, lw):
    theta = np.deg2rad(theta_deg)
    y1 = 0
    y2 = H - 1
    x1 = x0 + np.tan(theta) * (y1 - H/2.0)
    x2 = x0 + np.tan(theta) * (y2 - H/2.0)

    ax.plot([x1, x2], [y1, y2], '-', color=colore, linewidth=lw)
    ax.plot(x0, H/2.0, 'o', color=colore, markersize=8, markeredgewidth=lw)

# ============================================================
#  Esecuzione
# ============================================================
if __name__ == '__main__':
    # Ho mantenuto il tuo percorso, ma nota che ora lo passo correttamente
```

```
    alla funzione
image_path = r'autonologo_non_ta_targa.jpg'

trova_asse_simmetria(image_path)
```

Listing 4.4: Codice modello

Il risultato dello script 4.4 oltre a restituire l'asse di simmetria individuato sull'immagine di input (sia essa frontale o inclinata), genera ulteriori output analitici: una heatmap che mappa su griglia i punteggi di simmetria calcolati e una edge map che evidenzia i contorni della vettura all'interno dello spazio di lavoro 4.19 4.20.

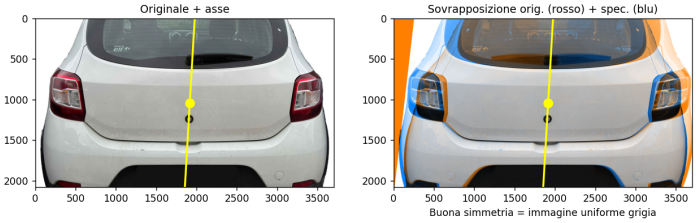


Figura 4.19: Risultato dell'asse e heat map

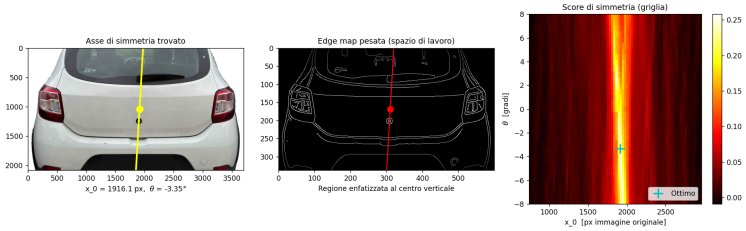


Figura 4.20: Risultato dell'asse e edge map

Questo script rappresenta la soluzione attualmente più vicina al risultato desiderato per la stima dell'asse di simmetria, indipendentemente dalla prospettiva dell'immagine di input (sia essa frontale o distorta). Ciononostante, pur a fronte dei recenti miglioramenti del codice, l'asse identificato diverge da quello reale: esso presenta infatti un'inclinazione anomala, sebbene passi correttamente per il centro della serratura del portellone.

Ulteriori modifiche all'algoritmo attuale risulterebbero eccessivamente artificiose, senza garantire il raggiungimento della precisione richiesta. Di conseguenza, si è scelto di esplorare un nuovo approccio basato sull'impiego di una rete neurale.

Capitolo 5

Reti neurali e Deep Learning: approccio tramite sviluppo e training

Le reti neurali stanno diventando uno strumento essenziale per molte aziende e organizzazioni. Il deep learning è una tecnica potente nell'ambito del machine learning che utilizza reti neurali con più livelli. Queste reti più profonde eccellono nell'identificare modelli complessi all'interno di vasti set di dati, rivoluzionando aree come l'elaborazione delle immagini e del linguaggio.

MATLAB stesso include il Deep Learning Toolbox, che fornisce algoritmi, funzioni e applicazioni per la creazione, l'addestramento, la visualizzazione e la simulazione di reti neurali. La distinzione fondamentale tra le reti neurali tradizionali e il deep learning risiede nella loro profondità, con le reti di deep learning che incorporano molti più strati. Questa maggiore profondità consente loro di affrontare problemi complessi ed elaborare grandi quantità di dati che travolgerebbero le reti più semplici. Consente la classificazione, la regressione, il clustering, la riduzione della dimensionalità, la previsione di serie temporali e la modellazione e il controllo di sistemi dinamici. Prima di utilizzare uno qualsiasi dei metodi, il primo passo è definire il problema selezionando un set di dati (dataset).

L'intelligenza artificiale analizza grandi volumi di dati utilizzando reti neurali con più livelli nascosti. Qualche anno fa, costruire un sistema di rilevamento delle frodi con cinque livelli nascosti era praticamente impossibile. Tutto questo è cambiato grazie all'incredibile potenza di calcolo e ai Big Data. Per addestrare i modelli di deep learning è necessaria un'enorme quantità di dati, poiché apprendono direttamente da essi. Più dati vengono forniti, maggiore è la loro accuratezza. L'intelligenza artificiale raggiunge un'accuratezza incredibile attraverso le reti neurali profonde (deep learning), qualcosa che in precedenza era impossibile [12].

5.1 Introduzione alle reti neurali

Una rete neurale è un tipo di algoritmo di machine learning ispirato al cervello umano. È uno strumento potente che eccelle nella risoluzione di problemi complessi più difficili da gestire per gli algoritmi informatici tradizionali, come il riconoscimento delle immagini e l'elaborazione del linguaggio naturale. La teoria delle reti neurali si ispira alla rete neurale naturale del sistema nervoso umano 5.1. È possibile definire

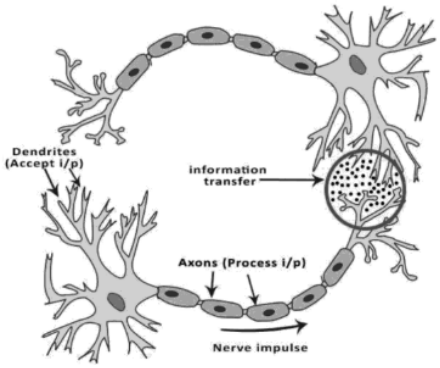


Figura 5.1: Rete neurale [12]

una rete neurale come un sistema di calcolo costituito da una serie di elementi di elaborazione semplici e altamente interconnessi, che elaborano informazioni in base alla loro risposta dinamica agli input esterni.

L'idea delle RNA (Reti Neurali Artificiali) si basa sulla convinzione che il funzionamento del cervello umano, creando le giuste connessioni, possa essere imitato utilizzando silicio e fili come neuroni e dendriti viventi.

Il cervello umano è composto da 100 miliardi di cellule nervose chiamate neuroni. Esse sono collegate ad altre migliaia di cellule tramite gli assoni. Gli stimoli provenienti dall'ambiente esterno o gli input degli organi sensoriali vengono recepiti dai dendriti. Questi input creano impulsi elettrici che viaggiano rapidamente attraverso la rete neurale. Un neurone può quindi inviare il messaggio ad altri neuroni per gestire il problema oppure non lo invia.

Le RNA sono composte da più nodi, che imitano i neuroni biologici del cervello umano. I neuroni sono collegati da link e interagiscono tra loro. I nodi possono ricevere dati in ingresso ed eseguire semplici operazioni sui dati. Il risultato di queste operazioni viene passato ad altri neuroni. L'output di ciascun nodo è chiamato attivazione o valore del nodo.

A ogni collegamento è associato un peso. Le RNA sono in grado di apprendere, il che avviene modificando i valori dei pesi. La seguente illustrazione mostra una semplice RNA 5.2.

Nel diagramma topologico mostrato, ogni freccia rappresenta una connessione tra due neuroni e indica il percorso del flusso di informazioni. Ogni connessione ha un peso, un numero intero che controlla il segnale tra i due neuroni.

Se la rete genera un risultato "buono o desiderato", non è necessario regolare i pesi. Se invece la rete genera un risultato "scarso o indesiderato" o un errore, il sistema modifica i pesi per migliorare i risultati successivi. Le RNA sono in grado di apprendere e devono essere addestrate. Esistono diverse strategie di apprendimento:

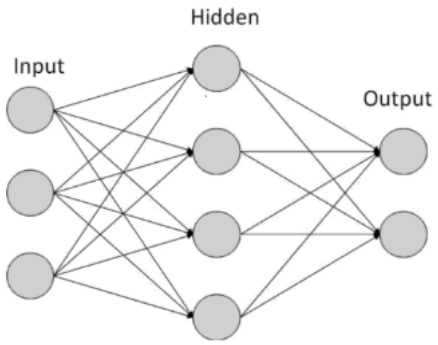


Figura 5.2: Rete neurale artificiale[12]

Apprendimento supervisionato

Implica un insegnante più esperto della RNA stessa. Ad esempio, l'insegnante alimenta alcuni dati di esempio di cui conosce già le risposte. Ad esempio, il riconoscimento di modelli. La RNA formula delle ipotesi durante il riconoscimento. Poi l'insegnante fornisce alla RNA le risposte. La rete confronta quindi le sue ipotesi con le risposte "corrette" dell'insegnante e apporta modifiche in base agli errori.

Apprendimento non supervisionato

È necessario quando non esistono set di dati di esempio con risposte note. Ad esempio, la ricerca di un modello nascosto. In questo caso, il clustering, cioè la suddivisione di un insieme di elementi in gruppi secondo un modello sconosciuto, viene effettuato sulla base dei set di dati esistenti.

Apprendimento per rinforzo

Questa strategia si basa sull'osservazione. La RNA prende una decisione osservando il suo ambiente. Se l'osservazione è negativa, la rete regola i suoi pesi per essere in grado di prendere una decisione diversa la volta successiva[12].

La strategia di apprendimento che utilizzeremo noi è incentrata sull'apprendimento supervisionato, ovvero faremo partire il processo dalla creazione di un dataset primario contenente n immagini del veicolo. Attraverso tecniche di data augmentation (distorsioni geometriche e variazioni di angolazione), la dimensione del dataset viene ampliata a n^m campioni. Su questa base dati, il sistema viene incaricato di stimare la posizione precisa del logo della vettura. Una volta estratte le relative coordinate (il centro del logo vedremo), si passa alla fase di addestramento (training) della rete neurale: il modello apprende così a predire correttamente la posizione del riferimento del logo su nuove immagini in base alle informazioni apprese dal dataset aumentato.

Il passaggio da n a n^m immagini tramite distorsioni, rotazioni e cambi di angolazione è l'anima del progetto. Il ragionamento è che il sistema non deve imparare a memoria

una singola foto, ma deve capire la geometria del veicolo. Introducendo distorsioni educhiamo la rete a riconoscere la zona corretta a prescindere dal punto di vista dell'osservatore o dalle condizioni di luce e più il dataset è variegato, meno il sistema sarà fragile di fronte a imprevisti (ad esempio una foto scattata dal basso o in un'officina buia). A differenza della semplice classificazione (dire "c'è un logo" o "non c'è"), qui chieddiamo al sistema di calcolare delle coordinate (x, y, z o u, v): in questo modo il sistema impara a mappare i pixel dell'immagine (input) in un sistema di riferimento spaziale (output) (Keypoint Detection o Object Pose Estimation). Questo tipo di apprendimento supervisionato di una rete neurale, rispetto a un metodo tradizionale, che potrebbe essere l'utilizzo di righelli o sensori laser, precisi ma rigidi, impara i riferimenti "semantici" capendo che il logo va esattamente al centro delle coordinate individuate. È un approccio molto più flessibile e simile a quello umano.

La classificazione delle reti neurali prevede [16]:

- Reti neurali feedforward (Feedforward Neural Networks): questo tipo di base elabora i dati in modo lineare dall'input all'output, senza cicli. Sono comunemente usate per attività semplici come la classificazione e la regressione.
- Reti neurali ricorrenti (RNN, Recurrent Neural Networks): progettate per dati sequenziali come serie temporali o linguaggio naturale, le RNN utilizzano un ciclo di feedback per conservare la memoria degli input precedenti, rendendole adatte per dati con una componente temporale.
- Reti neurali convoluzionali (CNN, Convolutional Neural Networks): progettate specificatamente per attività relative alle immagini, le CNN estraggono caratteristiche dalle immagini utilizzando un livello convoluzionale, che identifica modelli come bordi e forme.
- Reti avversarie generative (GAN, Generative Adversarial Networks): composte da un generatore e un discriminatore, le GAN mettono questi due componenti l'uno contro l'altro. Il generatore crea i dati mentre il discriminatore ne valuta l'autenticità. Questo processo contraddittorio fa sì che il generatore produca dati sempre più realistici, spesso utilizzati per generare immagini, video e audio.

Di nostro interesse saranno chiaramente le reti neurali convoluzionali, delle quali non entreremo nel dettaglio teorico per evitare confusione, ma saranno lo strumento del quale ci avvalremo.

I passi standart per la progettazione di reti neurali per la risoluzione di problemi sono i seguenti:

- raccogliere i dati
- creare la rete
- configurare la rete

- addestrare la rete
- convalidare la rete
- utilizzare la rete

Quattro sono le aree di applicazione tipiche delle reti neurali: adattamento di funzioni, riconoscimento di modelli, clustering e analisi delle serie temporali.

Quello di cui ci occuperemo in questa sezione è la raccolta dati e la generazione di un dataset con cui andare successivamente ad allenare la rete neurale.

5.2 Raccolta dati e generazione del dataset

Nel presente studio, la raccolta dei dati richiede un'ampia gamma di immagini del veicolo che coprano differenti direzioni e angolazioni di visuale, garantendo così che la rete neurale possa operare efficacemente in qualsiasi contesto operativo, indipendentemente dalla posizione dell'operatore. Le diverse trasformazioni geometriche che la rete deve implementare servono a simulare diversi punti di vista della telecamera o diverse posizioni dell'auto sulla linea di produzione:

- Rotazione: ruotare l'immagine di piccoli angoli (es. $\pm 5°$) o grandi angoli se l'auto può arrivare inclinata.
- Traslazione: spostare l'auto orizzontalmente o verticalmente nel frame.
- Scaling : simulare una telecamera più vicina o più lontana.
- Flipping: specchiare l'immagine (attenzione: se il logo non è simmetrico, occorrerà ricalcolare opportunamente le coordinate di riflessione).

Inoltre la rete dovrà riconoscere anche Trasformazioni Fotometriche, ovvero le diverse condizioni ambientali, fondamentali se il sistema deve lavorare con luci diverse (officina illuminata, penombra, riflessi sulla carrozzeria):

- Variazione di Luminosità e Contrasto: simula diverse ore del giorno o esposizioni della camera
- Color Jittering: modifica leggermente la saturazione e la tinta (utile se il colore dell'auto varia)

Infine le trasformazioni più avanzate che la rete deve riconoscere sono quelle prospettiche, che simulano l'angolazione (l'auto infatti non è mai perfettamente piatta di fronte all'obiettivo):

- Shearing: distorsione dell'immagine lungo l'asse
- Perspective Transform: cambia i punti di fuga dell'immagine, simulando una ripresa angolata

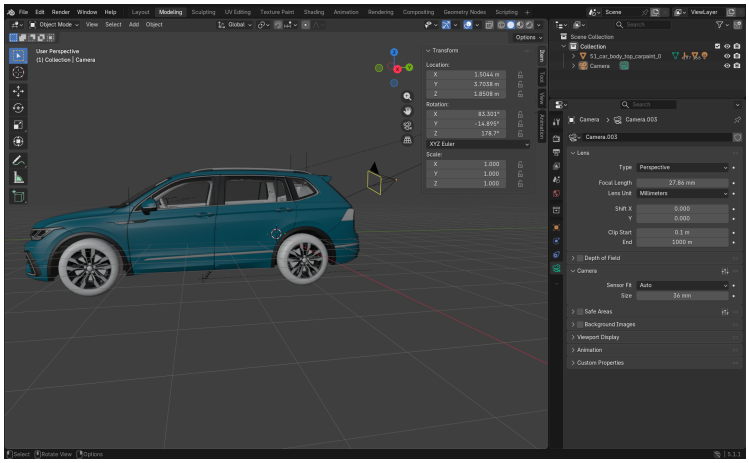


Figura 5.3: Modello in ambiente Blender

Poiché un approccio empirico diretto risulterebbe eccessivamente complesso, artificioso e dispendioso in termini di tempo, si è optato per una metodologia basata sulla modellazione tridimensionale. Nello specifico, è stato preso come riferimento un modello 3D di una Volkswagen Tiguan, importato e analizzato all'interno dell'ambiente Blender. Se abbiamo accesso al modello 3D (CAD) dell'auto, non abbiamo bisogno di scattare ad esempio 500 foto ma sfruttando le funzionalità della camera virtuale del software, è stato possibile generare in modo controllato un dataset completo di immagini riprese da molteplici prospettive di osservazione 5.3. Possiamo usare software come Blender o motori di gioco come Unity/Unreal Engine per posizionare l'auto in una scena virtuale: Blender è un software di rendering open-source ampiamente utilizzato in molti campi . Le capacità di modellare la geometria 3D e rendere il risultato lo rendono anche adatto a simulare la vettura del nostro studio.

Per scopi di test con la fotocamera, la geometria del bersaglio non deve essere completamente corretta. È sufficiente che la dimensione e la forma complessive corrispondano al bersaglio. BlenderProc è una libreria che "comanda" Blender tramite script Python senza che si debba aprire l'interfaccia grafica ogni volta. Viene utilizzato quindi come strumento che permette di creare automaticamente immagini 3D molto vicine alla realtà [13]. Insieme a ogni immagine, il programma genera in modo impeccabile anche i dati "invisibili" che servono alle reti neurali per capire la scena: la mappa delle distanze (profondità), l'orientamento delle superfici (normali) e il ritaglio preciso degli oggetti (segmentazione). Grazie a questa funzionalità, è possibile creare immagini perfette per l'addestramento dell'intelligenza artificiale, anche per situazioni complesse o difficili da fotografare nella realtà. Con uno script Python quindi possiamo ordinare al software di scattare 500 foto cambiando automaticamente l'angolazione della camera, le luci e il colore della carrozzeria. Nel nostro contesto, la gestione della camera virtuale di Blender (rappresentata dal rettangolo giallo in figura5.4) richiede l'impostazione di precisi vincoli di posizionamento spaziale e di ampiezza angolare. Tale perimetrazione evita l'inutile appesantimento computazionale derivante dalla

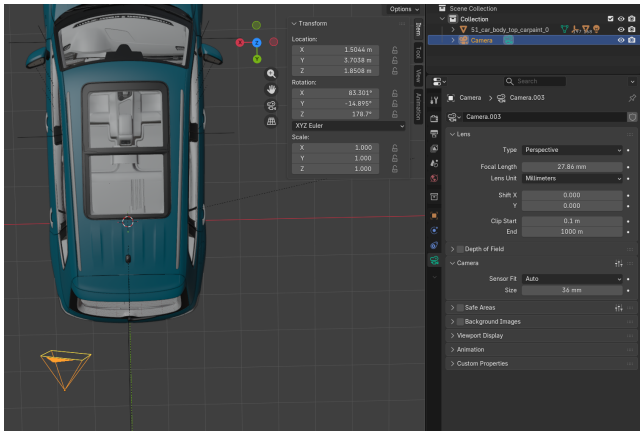


Figura 5.4: Posizione Camera in Blender



Figura 5.5: Keypoint

generazione di viste da distanze o prospettive non plausibili rispetto al reale ambiente operativo.

Nello specifico, i parametri spaziali stabiliti prevedono un'altezza da terra allineata a quella media di un operatore umano e una distanza dal veicolo compresa tra 60 cm e 1 metro, riflettendo accuratamente le reali condizioni di lavoro. Il range di angolazione è stato esteso a tutte le rotazioni capaci di mantenere l'autoveicolo all'interno dell'inquadratura, escludendo a priori le posizioni esterne non informative ai fini dell'addestramento.

Dato il modello 3D e l'ambiente di lavoro, per la generazione del dataset va individuato il keypoint che vogliamo la nostra rete neurale si alleni a riconoscere: a questo scopo, volendo individuare il punto per la corretta collocazione del logo utilizziamo come keypoint (tramite la funzione "group vertex" di Blender) il centro del logo, come visualizzato in figura 5.5. Lo Script che viene utilizzato per individuare le coordinate del keypoint fissato e acquisire le immagini della vettura per allenare la rete neurale è riportato di seguito 5.1

```
import bpy
import os
import random
import math
```

```
from mathutils import Vector

# --- CONFIGURAZIONE ---
n_images = 200
# In Python usa / o \\ per i percorsi Windows
output_dir = "C:\\dataset_12"
obj_name = "45_trunk_black_plastic_1_plastic_0.001" # Il nome dell'oggetto
    auto
                                                    # nell'outliner
group_name = "keypoint" # Il nome del Vertex Group che hai creato

# Assicurati che la cartella esista
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

def get_vertex_world_pos(obj_name, v_group_name):
    """Trova la posizione globale del primo vertice nel gruppo
        specificato"""
    obj = bpy.data.objects.get(obj_name)
    if not obj:
        print(f"Errore: Oggetto '{obj_name}' non trovato!")
        return None

    # Verifica se il gruppo esiste
    if v_group_name not in obj.vertex_groups:
        print(f"Errore: Vertex Group '{v_group_name}' non trovato su
            {obj_name}!")
        return None

    v_group_index = obj.vertex_groups[v_group_name].index

    # Cerchiamo il vertice che appartiene a questo gruppo
    for v in obj.data.vertices:
        for g in v.groups:
            if g.group == v_group_index:
                # L'operatore @ moltiplica la matrice del mondo per la
                    coordinata locale
                # e restituisce la posizione X,Y,Z reale nello spazio 3D
                return obj.matrix_world @ v.co
    return None

def get_2d_coords(camera, world_point):
    """Converte una coordinata 3D in pixel 2D dell'immagine renderizzata"""
    from bpy_extras.object_utils import world_to_camera_view
    scene = bpy.context.scene
    coords = world_to_camera_view(scene, camera, world_point)
    render_scale = scene.render.resolution_percentage / 100
```

```
    width = scene.render.resolution_x * render_scale
    height = scene.render.resolution_y * render_scale
    # Invertiamo la Y perche' Blender conta dal basso, i sensori dall'alto
    return (coords.x * width, (1 - coords.y) * height)

# --- LOOP DI GENERAZIONE ---
print(f'Inizio generazione dataset...')
for i in range(n_images):
    camera = bpy.data.objects['Camera']

    # 1. RECUPERO POSIZIONE VERTICE
    target_pos = get_vertex_world_pos(obj_name, group_name)
    if target_pos is None:
        print('Interruzione: Punto di ancoraggio non trovato.')
        break

    # 2. POSIZIONAMENTO CAMERA (Ravvicinato per il retro)
    # Angolo centrato sull'asse Y positivo
    angle = random.uniform(math.pi / 2 - 0.5, math.pi / 2 + 0.5)

    # Distanza ravvicinata (es. tra 60cm e 1.2 metri dal punto)
    dist = random.uniform(0.6, 1.2)
    z_height = random.uniform(0.5, 1.5) # Altezza variabile

    # Calcolo posizione camera rispetto al target (non allo zero del mondo)
    camera.location.x = target_pos.x + dist * math.cos(angle)
    camera.location.y = target_pos.y + dist * math.sin(angle)
    camera.location.z = target_pos.z + z_height

    # Punta la camera verso il vertice
    direction = target_pos - camera.location
    camera.rotation_mode = 'QUATERNION'
    camera.rotation_quaternion = direction.to_track_quat('-Z', 'Y')

    # 3. RENDER
    scene = bpy.context.scene
    scene.render.filepath = os.path.join(output_dir, f'img_{i}.png')
    bpy.ops.render.render(write_still=True)

    # 4. SALVATAGGIO COORDINATE PIXEL
    px_coords = get_2d_coords(camera, target_pos)
    with open(os.path.join(output_dir, f'img_{i}.txt'), 'w') as f:
        f.write(f'{px_coords[0]}, {px_coords[1]}')

print(f'Dataset di {n_images} immagini generato con successo in:
    {output_dir}')
```

Listing 5.1: Script Blender

Lo script che stiamo creando trasforma Blender in una "fabbrica di dati". Invece di scattare foto manualmente, chiediamo a Python di muovere la telecamera e le luci matematicamente, salvando ogni volta dove finisce il nostro punto, del quale vogliamo calcolare le coordinate, nell'immagine finale.

- Caricamento: lo script apre il file blend in "background" (senza interfaccia grafica). Riconosce l'auto e il punto vuoto del quale vogliamo le coordinate
- Campionamento (Semisfera): immaginiamo una cupola invisibile sopra la macchina. Lo script sceglie un punto a caso su questa cupola, ci mette la telecamera e la punta verso il logo. Questo garantisce che avremmo foto dall'alto, di lato, da 45°, ecc...
- Luci Casuali: per ogni scatto, lo script crea o sposta delle luci. Cambia la loro potenza (Watt) e il colore (luce calda/fredda), così l'IA impara a ignorare i riflessi fastidiosi sulla vernice.
- Proiezione Matematica: Blender conosce la posizione 3D del logo e la posizione della camera. Usando la trigonometria della lente, calcola esattamente su quali pixel (x, y) cade il logo nella foto 2D.

In questo modo il sistema operativo (Python) legge le istruzioni dello script, apre Blender, calcola le luci, muove la telecamera e scatta le foto per poi salvare tutto nella cartella destinata al dataset.

Perchè l'utilizzo del terminale di Blender ?

Invece di usare il CMD di Windows che potrebbe non trovare i comandi, useremo direttamente l'eseguibile di Blender già installato. Eseguire uno script direttamente nel terminale interno di Blender (la Python Console o il Text Editor) offre vantaggi decisivi rispetto al prompt dei comandi di Windows:

- Accesso immediato al contesto di lavoro : quando si esegue un codice dentro Blender, lo script agisce istantaneamente sulla scena 3D aperta, sugli oggetti selezionati e sui materiali attivi. Se lanciamo lo script da cmd, rischiamo di non avere il contesto dell'interfaccia o di dover avviare una nuova istanza del programma in background.
- Autocompletamento intelligente (Auto-completion): nella Python Console di Blender, premendo il tasto Tab il programma suggerisce classi, metodi e proprietà dell'API di Blender (bpy), facilitando la scrittura senza dover consultare continuamente la documentazione.
- Feedback visivo e debug in tempo reale: l'effetto dello script è visibile immediatamente nella vista 3D. Se modifichiamo una posizione o creiamo un vertice, lo vediamo a schermo all'istante.



Figura 5.6: keypoint

- Modifica integrata con il Text Editor: possiamo scrivere lo script nell'editor di testo di Blender e premere semplicemente Run Script per testare al volo senza passare da una finestra all'altra.

- Salvataggio dello script nel file blend: i file di testo creati dentro Blender possono essere salvati direttamente all'interno della scena 3D, rendendo il progetto completo e portatile.

Un dettaglio chiave da considerare è che il punto target, le cui coordinate saranno oggetto di stima da parte della rete neurale, sia fissato sul corpo della vettura. A tale scopo, non basterà vedere il nostro kypoint nell'immagine, ma dovremo anche estrarne le coordinate bidimensionali (x, y). Per ancorare fisicamnete questo punto alla carrozzeria e far si che non sia solo un oggetto che fluttua vicino ad essa, il metodo migliore è utilizzare il Vertex Group (Gruppo di Vertici), come già detto, in quanto questa funzione garantisce che il punto sia vincolato alla geometria della macchina, seguendone ogni movimento o deformazione 5.6. In seguito con il Vertex Parenting, eviteremo l'errore umano di posizionamento. Se decidiamo di cambiare l'angolazione dell'auto per generare 1000 foto diverse (Data Augmentation), il punto rimarrà proiettato sulla carrozzeria in modo matematicamente perfetto, permettendoci di generare un dataset di training di altissima qualità in pochi minuti. Una volta create le immagini 5.7, la rete neurale non sa cosa sta guardando finché non le diamo le coordinate di verità (Ground Truth).

Fissare quel punto (keypoint) in Blender ci serve a creare le etichette (labels). Senza quel punto, avremmo solo 200 foto di una macchina. Con quel punto, abbiamo 200 foto + 200 coordinate che dicono alla rete: "In questa foto, il logo si trova esattamente qui"5.8. Se dovessimo scattare 200 o 500 foto reali a una macchina, dovremmo aprire ogni foto in un software (come LabelImg) e cliccare a mano sul logo per dire al computer dove si trova. Un lavoro enorme e impreciso. Avendo fissato il Vertex Group in Blender, lo script può calcolare automaticamente dove finisce quel punto nello spazio 2D dell'immagine per ogni singolo render. In questo modo abbiamo un file (JSON, XML o TXT) per ogni immagine con scritto: logo_pos: [x, y]5.8.



Figura 5.7: Dataset

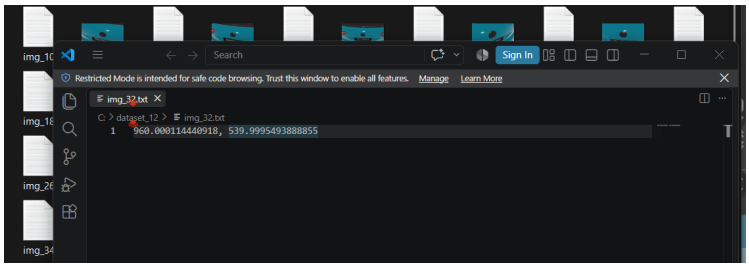


Figura 5.8: Coordinate (esempio foto 32)

5.2.1 Training della rete e risultati ottenuti

Le reti neurali artificiali abbiamo detto essere modelli matematici composti da unità di calcolo (neuroni) connesse tra loro, organizzate in strati (layer). Il deep learning è un sottoinsieme del machine learning che usa reti neurali con molti strati (layer) nascosti (da qui il "deep"). In pratica: deep learning = reti neurali profonde. Quando sentiamo parlare di deep learning, stiamo parlando di reti neurali con molti layer.

Non è un dettaglio tecnico astratto: il numero di layer cambia drasticamente cosa la rete riesce a fare. Una rete con due layer può riconoscere forme semplici, una con duecento layer può riconoscere centinaia di razze di cani in milioni di foto diverse.

Ogni neurone riceve input, li elabora con una semplice operazione matematica e passa il risultato al neurone successivo, il che significa che il suo valore di output diventa l'input per i prossimi neuroni. Da questo meccanismo elementare nasce la capacità di riconoscere immagini, tradurre lingue, generare testi. Ogni connessione tra neuroni ha un peso: un numero che dice quanto quel segnale è importante e influenza quanto input viene passato tra un neurone all'altro. Questo comportamento segue la formula input $\times$ weights. Durante l'addestramento, la rete aggiusta milioni ,o miliardi, di pesi finché non riesce a produrre l'output corretto sui dati di training. È in quei pesi che vive la "conoscenza" della rete. Una volta che un neurone riceve

gli input da tutti gli altri neuroni connessi ad esso, viene aggiunto un bias, un valore costante che va sommato al calcolo che coinvolge il peso menzionato. Anche il bias è un fattore che viene modificato durante l'addestramento. Una rete neurale è in grado di generalizzare e modellare un problema nel mondo reale (che non è altro che una funzione matematica) proprio grazie al costante aggiustamento di pesi e bias, che modulano l'output e l'input di ogni singolo neurone finché non la rete non approccia ad una soluzione accettabile. L'output di un neurone è espresso dalla formula output = inputs × weights + bias. L'aggiustamento di pesi e bias viene fatto nelle hidden layers (strati nascosti), che sono gli strati presenti tra lo strato di input e quello di output. Sono detti "nascosti" proprio perché non vediamo il comportamento di aggiustamento di pesi e bias.

La caratteristica che rende complesse le reti neurali è proprio l'enorme mole di calcoli che avviene a livello sia di rete che di singolo neurone.

Insieme ai pesi e bias ci sono le funzioni di attivazione, che aggiungono una ulteriore complessità matematica e influenzano enormemente la performance di una rete neurale. Pesi e bias possono essere interpretati come un sistema di manopole che possiamo ruotare per ottimizzare il nostro modello (come quando cerchiamo di sintonizzare la nostra radio ruotando le manopole per cercare la frequenza gradita).

La differenza sostanziale è che in una rete neurale, abbiamo centinaia se non migliaia di manopole da girare per raggiungere il risultato finale. Poiché pesi e bias sono dei parametri della rete, questi saranno oggetto del cambiamento generato dalla rotazione della manopola immaginaria.

Visto che i pesi sono moltiplicati all'input, questi influenzano la magnitudine dell'input. Il bias, invece, poiché è sommato all'espressione dell'output sposterà la funzione nel piano dimensionale.

Allenare una rete neurale significa farla sbagliare migliaia di volte e correggerla a ogni errore. Il processo si chiama training ed è il cuore del deep learning. Si parte con pesi casuali. Si dà alla rete un dato di input (un'immagine di un gatto, per esempio) e si guarda cosa risponde. Quasi sempre risponde male. La differenza tra la risposta sbagliata e quella corretta si chiama loss (perdita). Un algoritmo chiamato backpropagation aggiusta poi i pesi in modo da ridurre la loss, iterazione dopo iterazione. Il problema più subdolo del training è l'overfitting: la rete impara a memoria gli esempi ma fallisce su dati nuovi. Tecniche come dropout, regularization e data augmentation servono proprio a prevenire questo problema [14] [15].

```
import os
import math
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from pathlib import Path
from PIL import Image
```

```
from torch.utils.data import Dataset, DataLoader, Subset
from torchvision import transforms, models
import matplotlib.pyplot as plt

#
    ----------------------------------------------------------------------------
# 1. Configurazione Kaggle
#
    ----------------------------------------------------------------------------
class Config:
    # --- MODIFICA: Due percorsi separati ---
    # Inserisci il percorso esatto della cartella che contiene i file .png
    images_dir =
        "/kaggle/input/datasets/vittoriamedici/virtualgauges-5000/Immagini"

    # Inserisci il percorso della cartella che contiene il file poses.csv
    csv_dir = "/kaggle/input/datasets/vittoriamedici/virtualgauges-5000"
    csv_name = "poses.csv"

    epochs = 40
    batch_size = 16
    lr = 1e-4
    beta = 1.0

    val_split = 0.15
    test_split = 0.10
    checkpoint_name = "best_pose_model.pt"

#
    ----------------------------------------------------------------------------
# 2. Dataset e Trasformazioni
#
    ----------------------------------------------------------------------------
def get_transforms(train=False):
    ops = [transforms.Resize((224, 224))]
    if train:
        ops += [transforms.ColorJitter(brightness=0.1, contrast=0.1,
            saturation=0.1)]

    ops += [
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
            0.225]),
    ]
    return transforms.Compose(ops)

class PoseDataset(Dataset):
```

```
    # --- MODIFICA: Ora accetta images_dir e csv_dir ---
    def __init__(self, images_dir, csv_dir, csv_name, transform=None):
        self.images_dir = Path(images_dir)
        self.csv_dir = Path(csv_dir)

        # Legge il CSV dalla cartella del CSV
        self.df = pd.read_csv(self.csv_dir / csv_name)
        self.transform = transform or get_transforms()

        pos = self.df[["x", "y", "z"]].values.astype(np.float32)
        self.pos_mean = pos.mean(axis=0)
        self.pos_std = pos.std(axis=0) + 1e-6

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        row = self.df.iloc[idx]

        # --- MODIFICA: Cerca l'immagine nella cartella delle immagini ---
        img_path = self.images_dir / row["filename"]

        img = Image.open(img_path).convert("RGB")
        img = self.transform(img)

        pos = np.array([row["x"], row["y"], row["z"]], dtype=np.float32)
        pos_norm = (pos - self.pos_mean) / self.pos_std

        angles = np.array([row["roll"], row["pitch"], row["yaw"]],
            dtype=np.float32)
        trig = np.concatenate([np.sin(angles),
            np.cos(angles)]).astype(np.float32)

        target = np.concatenate([pos_norm, trig]).astype(np.float32)

        return img, torch.from_numpy(target)

#
    ------------------------------------------------------------------------
# 3. Architettura del Modello e Loss
#
    ------------------------------------------------------------------------
class PoseRegressionNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.backbone = models.resnet18(weights="IMAGENET1K_V1")
        feat_dim = self.backbone.fc.in_features
```

```
        self.backbone.fc = nn.Identity()

        self.head = nn.Sequential(
            nn.Linear(feat_dim, 256),
            nn.ReLU(inplace=True),
            nn.Dropout(0.3),
            nn.Linear(256, 128),
            nn.ReLU(inplace=True),
            nn.Linear(128, 9),
        )

    def forward(self, x):
        return self.head(self.backbone(x))

class PoseLoss(nn.Module):
    def __init__(self, beta=1.0):
        super().__init__()
        self.beta = beta
        self.mse = nn.MSELoss()

    def forward(self, pred, target):
        loss_pos = self.mse(pred[:, :3], target[:, :3])
        loss_ang = self.mse(pred[:, 3:], target[:, 3:])
        return loss_pos + self.beta * loss_ang, loss_pos.item(),
            loss_ang.item()

#
    ----------------------------------------------------------------------------
# 4. Esecuzione: Train, Validation e Test Finale
#
    ----------------------------------------------------------------------------
if __name__ == "__main__":
    cfg = Config()
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Avvio su: {device.type.upper()}")

    # --- CREAZIONE DEI TRE SPLIT (Train, Val, Test) ---
    # Creiamo due "versioni" del dataset per gestire correttamente le
        trasformazioni
    base_dataset_train = PoseDataset(cfg.images_dir, cfg.csv_dir,
        cfg.csv_name, transform=get_transforms(train=True))
    base_dataset_eval = PoseDataset(cfg.images_dir, cfg.csv_dir,
        cfg.csv_name, transform=get_transforms(train=False))

    pos_mean, pos_std = base_dataset_train.pos_mean,
        base_dataset_train.pos_std
```

```
total_len = len(base_dataset_train)
n_test = int(total_len * cfg.test_split)
n_val = int(total_len * cfg.val_split)
n_train = total_len - n_val - n_test

# Generiamo indici casuali ma fissi (seed 42) per poter replicare
    l'esperimento
generator = torch.Generator().manual_seed(42)
indices = torch.randperm(total_len, generator=generator).tolist()

train_idx = indices[:n_train]
val_idx = indices[n_train:n_train + n_val]
test_idx = indices[n_train + n_val:]

# Assegniamo gli indici alle versioni corrette del dataset
train_ds = Subset(base_dataset_train, train_idx)
val_ds = Subset(base_dataset_eval, val_idx)
test_ds = Subset(base_dataset_eval, test_idx)

print(f"Dataset diviso in: Train ({len(train_ds)}), Val
    ({len(val_ds)}), Test ({len(test_ds)})")

train_loader = DataLoader(train_ds, batch_size=cfg.batch_size,
    shuffle=True, num_workers=0)
val_loader = DataLoader(val_ds, batch_size=cfg.batch_size,
    shuffle=False, num_workers=0)
test_loader = DataLoader(test_ds, batch_size=cfg.batch_size,
    shuffle=False, num_workers=0)

model = PoseRegressionNet().to(device)
criterion = PoseLoss(beta=cfg.beta)
optimizer = torch.optim.Adam(model.parameters(), lr=cfg.lr)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
    patience=5, factor=0.5)

best_val_loss = float("inf")
storia_train_loss = []
storia_val_loss = []

# --- CICLO DI ADDESTRAMENTO ---
for epoch in range(1, cfg.epochs + 1):
    model.train()
    train_loss = 0.0
    for imgs, targets in train_loader:
        imgs, targets = imgs.to(device), targets.to(device)
        optimizer.zero_grad()
        preds = model(imgs)
```

```
            loss, lp, la = criterion(preds, targets)
            loss.backward()
            optimizer.step()
            train_loss += loss.item() * imgs.size(0)

        train_loss /= len(train_ds)

        model.eval()
        val_loss = 0.0
        with torch.no_grad():
            for imgs, targets in val_loader:
                imgs, targets = imgs.to(device), targets.to(device)
                preds = model(imgs)
                loss, lp, la = criterion(preds, targets)
                val_loss += loss.item() * imgs.size(0)

        val_loss /= len(val_ds)
        scheduler.step(val_loss)

        storia_train_loss.append(train_loss)
        storia_val_loss.append(val_loss)

        print(f"Epoca {epoch:03d}/{cfg.epochs} | Train Loss:
            {train_loss:.4f} | Val Loss: {val_loss:.4f}")

        if val_loss < best_val_loss:
            best_val_loss = val_loss
            torch.save({
                "model_state": model.state_dict(),
                "pos_mean": pos_mean,
                "pos_std": pos_std,
                # --- AGGIUNTO: Salviamo anche la storia della loss! ---
                "storia_train_loss": storia_train_loss,
                "storia_val_loss": storia_val_loss,
                "test_idx": test_idx
            }, cfg.checkpoint_name)
            print(f"Salvato! (Loss: {best_val_loss:.4f})")

    print("\nAddestramento completato!")

    # --- VALUTAZIONE FINALE SUL TEST SET ---
    print("\n" + "="*40)
    print("APERTURA CASSAFORTE: VALUTAZIONE SUL TEST SET")
    print("="*40)

    # Carichiamo il miglior modello salvato
    ckpt = torch.load(cfg.checkpoint_name, map_location=device,
```

```
    weights_only=False)
model.load_state_dict(ckpt["model_state"])
model.eval()

test_loss = 0.0
with torch.no_grad():
    for imgs, targets in test_loader:
        imgs, targets = imgs.to(device), targets.to(device)
        preds = model(imgs)
        loss, lp, la = criterion(preds, targets)
        test_loss += loss.item() * imgs.size(0)

test_loss /= len(test_ds)
print(f"Errore finale (Loss) sui dati mai visti: {test_loss:.4f}")
print("="*40 + "\n")

# --- GRAFICO CURVA DI APPRENDIMENTO ---
plt.figure(figsize=(10, 5))
plt.plot(storia_train_loss, label='Errore su Addestramento (Train)')
plt.plot(storia_val_loss, label='Errore su Validazione (Val)')
plt.title('Curva di Apprendimento del Modello')
plt.xlabel('Epoche')
plt.ylabel('Loss (Errore)')
plt.legend()
plt.grid(True)
plt.show()
```

Listing 5.2: Script del training

Lo script in figura 5.5 imposta una pipeline di deep learning per la stima della posa (pose estimation): dato un'immagine, il modello deve predire la posizione 3D (x, y, z) e l'orientamento (roll, pitch, yaw) di un oggetto. Immaginiamo un'auto in uno showroom, o meglio, la sua carrozzeria vista attraverso un paio di smart glasses. L'obiettivo finale è che chi indossa gli occhiali veda, in realtà aumentata, esattamente dove va posizionato il logo sulla carrozzeria (non un punto approssimativo, ma la posizione precisa, alla giusta angolazione, indipendentemente da dove si trovi la persona rispetto all'auto in quel momento).

Il problema è che gli occhiali si muovono: la persona cammina intorno alla macchina, si china, inclina la testa. Per sapere dove disegnare il logo in sovrimpressione, il sistema deve prima rispondere a una domanda più basilare: "in questo istante, dove si trovano gli occhiali nello spazio rispetto alla carrozzeria, e come sono orientati?" Questo è esattamente il problema di pose estimation che il codice affronta.

Per insegnare a una rete neurale a rispondere a questa domanda, serve un dataset "sintetico": migliaia di scatti generati virtualmente che ritraggono la carrozzeria da tantissime posizioni e angolazioni diverse. Per ogni scatto si conosce esattamente,

perché generato a computer, la posizione della "telecamera virtuale" (x, y, z) e il suo orientamento (roll, pitch, yaw), cioè la posa esatta degli occhiali in quel momento.

Lo script comincia con la "Config", il pannello di controllo: dove stanno le immagini, dove sta il CSV con le pose vere, quante epoche di addestramento, batch size, ecc.

Poi c'è il "Dataset" (PoseDataset): l'operatore che, immagine per immagine, prepara i dati. Ridimensiona ogni foto a 224×224 pixel, e in fase di training aggiunge piccole variazioni di luce/contrasto (utile perché in un ambiente reale le condizioni di luce sullo showroom cambieranno rispetto al rendering virtuale, quindi la rete deve imparare a non fossilizzarsi sui dettagli fotometrici esatti).

Un dettaglio chiave è che la posizione (x, y, z) viene normalizzata (media/deviazione standard) per stabilità numerica. Gli angoli invece vengono trasformati in seno e coseno (un trucco necessario perché un angolo di 359° e uno di 1° sono quasi identici come orientamento reale degli occhiali, ma numericamente sembrerebbero lontanissimi se usati "grezzi"). Con sin/cos la rete impara la ciclicità corretta.

Il "modello" (PoseRegressionNet) prende una ResNet18 pre-addestrata (che "sa già vedere" forme e texture in generale), le toglie la parte finale pensata per classificare oggetti, e ci attacca una piccola testa su misura che restituisce 9 numeri: i 3 di posizione + 6 di seno/coseno degli angoli, cioè la posa completa degli occhiali rispetto alla carrozzeria in quell'istante.

Infine la "loss" (PoseLoss) giudica separatamente quanto la rete sbaglia in posizione e quanto in orientamento, e le combina con un peso regolabile (beta) (utile perché magari per posizionare bene il logo l'orientamento conta di più ,o di meno, della posizione esatta in centimetri).

```
import torch
from torchvision import models
import torch.nn as nn

# 1. Ricrea l'architettura esatta usata in addestramento
class PoseRegressionNet(nn.Module):
    def __init__(self):
        super().__init__()
        # Non serve scaricare i pesi ImageNet, li sovrascriveremo
        self.backbone = models.resnet18(weights=None)
        feat_dim = self.backbone.fc.in_features
        self.backbone.fc = nn.Identity()

        self.head = nn.Sequential(
            nn.Linear(feat_dim, 256),
            nn.ReLU(inplace=True),
            nn.Dropout(0.3),
            nn.Linear(256, 128),
            nn.ReLU(inplace=True),
            nn.Linear(128, 9),
        )
```

```
    def forward(self, x):
        return self.head(self.backbone(x))

# 2. Configura il dispositivo
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = PoseRegressionNet().to(device)

# 3. Leggi e carica il checkpoint
print("Lettura del file .pt in corso...")
ckpt = torch.load(r"C:\Users\vitto\OneDrive - Universita Politecnica delle
    Marche\MANUFACTOR\Virtual Gauges\Risultati 5000\best_pose_model.pt",
    map_location=device, weights_only=False)

model.load_state_dict(ckpt["model_state"])
model.eval() # Fondamentale: imposta il modello in modalita di predizione

# 4. Estrai i parametri e le logiche salvate
pos_mean = ckpt["pos_mean"]
pos_std = ckpt["pos_std"]

print("Pesi del modello caricati con successo!")
print(f"Media Posizioni (X, Y, Z): {pos_mean}")

if "test_idx" in ckpt:
    print(f"Trovati {len(ckpt['test_idx'])} indici di test blindati nel
        file.")
```

Listing 5.3: Script di inferenza

Mentre il primo file costruiva l'infrastruttura per addestrare la rete, questo script 5.3serve a riportare in vita quella rete già addestrata, per usarla concretamente (ad esempio dentro l'applicazione che gira sugli occhiali o in un test su PC prima di portarla sul dispositivo).

weights=None

Qui non serve scaricare i pesi pre-addestrati su ImageNet come si faceva in training, perché verranno sovrascritti con i pesi specifici che la rete ha imparato guardando migliaia di viste sintetiche della carrozzeria. È come recuperare lo scheletro vuoto del modello, pronto per essere "riempito" con la conoscenza già acquisita.

La struttura (backbone + testa a 9 output) deve essere identica a quella usata in fase di addestramento, altrimenti i pesi salvati non si "incastrerebbero" correttamente nei layer. Il modello, ormai addestrato, è come un fotografo esperto che ha imparato a memoria come appare la carrozzeria vista da migliaia di angolazioni diverse. Ma quel fotografo, per ora, esiste solo "sulla carta", ossia nel codice, e prima di poterlo

far lavorare bisogna svegliarlo e rimettergli in testa tutto ciò che ha imparato. Prima di tutto, bisogna decidere su quale "banco di lavoro" farlo operare. Un conto è far ragionare il fotografo su un tavolo normale (la CPU del computer), un conto è dargli un tavolo super attrezzato con strumenti potenti (la GPU, la scheda grafica). Semplicemente se c'è una GPU disponibile, la andremo ad usare perché è molto più veloce; altrimenti lavoriamo con la CPU. A questo punto la nostra rete ha le mani e gli occhi (l'architettura della rete), ma non ricorda ancora nulla. Il file definibile come il suo "diario di memoria" contiene tutto ciò che ha imparato durante l'addestramento, cioè migliaia e migliaia di numeri (i pesi della rete) che rappresentano la sua esperienza visiva accumulata guardando le foto sintetiche della carrozzeria. Quindi il codice "apre il diario" e trasferisce tutta quella memoria dentro la rete appena creata. Da questo momento, la rete non è più un fotografo qualunque: è quello specifico fotografo che ha imparato a riconoscere la nostra carrozzeria da ogni prospettiva. Durante l'addestramento, per non fargli "imparare a memoria" invece che capire davvero (overfitting), ogni tanto gli si bendava un occhio a caso, o gli si copriva un orecchio, in modo casuale (è il meccanismo del Dropout). Ora che deve lavorare sul serio, ovvero dare sempre la stessa risposta corretta alla stessa foto, questi bendaggi casuali vanno tolti. Ricordiamo che durante l'addestramento le posizioni x, y, z non erano state date alla rete "grezze", ma erano state ridotte a numeri più piccoli e maneggevoli (sottraendo la media e dividendo per la variazione tipica, cioè normalizzandole). Questo significa che il fotografo, quando risponde "dove pensa di trovarsi", non dirà mai un numero reale tipo "80 cm a destra della portiera". Dirà invece un numero "in codice", tipo "0.3", un valore nella scala ridotta usata in addestramento.

$$pos_mean = ckpt[\text{"}pos_mean\text{"}]$$

$$pos_std = ckpt[\text{"}pos_std\text{"}]$$

sono i due numeri necessari per fare il percorso inverso: prendere la risposta in codice del fotografo e ritrasformarla in una vera distanza reale in centimetri/metri, utilizzabile per capire davvero dove disegnare il logo sulla carrozzeria vista dagli occhiali. Inoltre durante l'addestramento, alcune foto erano state messe da parte apposta, come una specie di "esame a sorpresa": il fotografo non le aveva mai viste durante l'allenamento, e servivano solo per controllare, alla fine, se avesse davvero imparato bene o solo "studiato a memoria" le foto di training. Il diario si porta dietro anche l'elenco di quali foto erano quelle riservate all'esame , così in futuro si può ripetere lo stesso identico esame senza rischiare di sbagliare e usare per errore foto che il fotografo aveva già visto. Quindi questo secondo script prende il fotografo "addestrato" , che prima esisteva solo durante il training , e lo richiama in vita per l'uso reale: gli restituisce tutta la memoria (i pesi), lo mette in modalità seria , e recupera il "righello" necessario per tradurre le sue risposte in numeri reali utilizzabili sugli occhiali.

```
Avvio su: CUDA
Dataset diviso in: Train (3750), Val (750), Test (500)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth"
    to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
100%| 44.7M/44.7M [00:00<00:00, 214MB/s]
Epoca 001/40 | Train Loss: 0.4111 | Val Loss: 0.1421
Salvato! (Loss: 0.1421)
Epoca 002/40 | Train Loss: 0.1886 | Val Loss: 0.1124
Salvato! (Loss: 0.1124)
Epoca 003/40 | Train Loss: 0.1552 | Val Loss: 0.0868
Salvato! (Loss: 0.0868)
Epoca 004/40 | Train Loss: 0.1287 | Val Loss: 0.0862
Salvato! (Loss: 0.0862)
Epoca 005/40 | Train Loss: 0.1202 | Val Loss: 0.0782
Salvato! (Loss: 0.0782)
Epoca 006/40 | Train Loss: 0.1128 | Val Loss: 0.0714
Salvato! (Loss: 0.0714)
Epoca 007/40 | Train Loss: 0.1042 | Val Loss: 0.0640
Salvato! (Loss: 0.0640)
Epoca 008/40 | Train Loss: 0.0936 | Val Loss: 0.0652
Epoca 009/40 | Train Loss: 0.0898 | Val Loss: 0.0962
Epoca 010/40 | Train Loss: 0.0835 | Val Loss: 0.0656
Epoca 011/40 | Train Loss: 0.0781 | Val Loss: 0.0618
Salvato! (Loss: 0.0618)
Epoca 012/40 | Train Loss: 0.0769 | Val Loss: 0.0596
Salvato! (Loss: 0.0596)
Epoca 013/40 | Train Loss: 0.0731 | Val Loss: 0.0841
Epoca 014/40 | Train Loss: 0.0694 | Val Loss: 0.0740
Epoca 015/40 | Train Loss: 0.0663 | Val Loss: 0.0563
Salvato! (Loss: 0.0563)
...
========================================
Errore finale (Loss) sui dati mai visti: 0.0642
========================================
```

Listing 5.4: Output dell'addestramento del modello

```
# --- GRAFICO CURVA DI APPRENDIMENTO ---
plt.figure(figsize=(10, 5))
plt.plot(storia_train_loss, label='Errore su Addestramento (Train)')
plt.plot(storia_val_loss, label='Errore su Validazione (Val)')
plt.title('Curva di Apprendimento del Modello')
plt.xlabel('Epoche')
plt.ylabel('Loss (Errore)')
plt.legend()
plt.grid(True)

# --- AGGIUNTO: Salva la figura in alta risoluzione ---
```

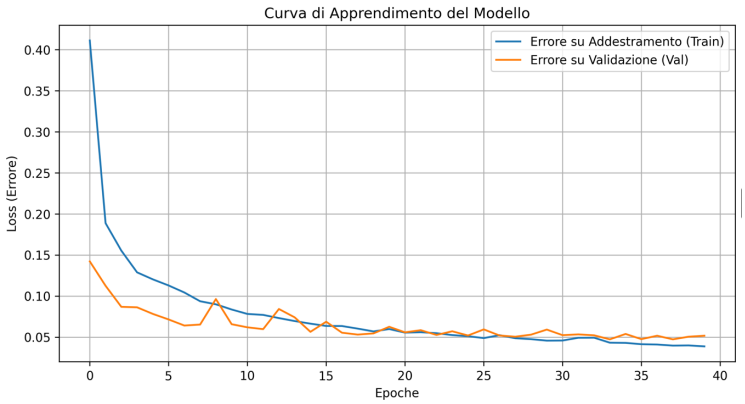


Figura 5.9: Fase di addestramento

```
plt.savefig("curva_di_apprendimento.png", dpi=300, bbox_inches='tight')

plt.show()
```

Listing 5.5: Grafico della curva di apprendimento

In figura 5.9 viene raccontata la fase di addestramento: sull'asse orizzontale ci sono le epoche, ogni epoca è un "giro completo" in cui il fotografo ha visto tutte le foto di training almeno una volta. Sull'asse verticale c'è la loss, l'errore calcolato dalla PoseLoss che avevamo visto. Più il valore è basso, meno il fotografo sbaglia. Ci sono due linee:

- Blu (Train): l'errore misurato sulle foto che il fotografo sta usando per imparare.
- Arancione (Val): l'errore misurato su un gruppo di foto separato, il "validation set", che il fotografo non usa per imparare ma solo per essere controllato di tanto in tanto.

Il grafico può essere suddiviso in intervalli:

Fase iniziale (epoche 0-5)

Fase di apprendimento rapido. All'inizio l'errore blu parte altissimo (0.41) e crolla vertiginosamente. È normale: all'inizio il fotografo non sa praticamente nulla di specifico sulla nostra carrozzeria (ha solo la conoscenza generica di ImageNet), quindi i primi "sguardi" alle foto gli insegnano moltissimo in poco tempo .

Fase intermedia (epoche 5-20)

Fase diapprendimento più lento e raffinato. Le due curve continuano a scendere ma più dolcemente, e iniziano ad avvicinarsi tra loro. Il fotografo sta ancora migliorando, ma ora sta affinando dettagli sempre più sottili, non più concetti grossolani.

Fase finale (epoche 20-40)

Fase di stabilizzazione (plateau). Entrambe le curve si appiattiscono attorno a valori bassi (0.04-0.06) e restano lì, con piccole oscillazioni. Questo indica che il modello ha raggiunto un buon punto di convergenza: continuare ad allenarlo oltre non porta più benefici significativi.

Il segnale più importante è che train e validation restano vicine.

Questo è l'aspetto più rassicurante del grafico. Le due curve non si separano man mano che le epoche avanzano (la linea blu (train) scende leggermente sotto l'arancione (val), ma di poco, e senza un divario che si allarga progressivamente).

Se invece la linea blu fosse continuata a scendere molto mentre l'arancione fosse rimasta ferma o fosse addirittura risalita, sarebbe stato un chiaro segnale di overfitting: il fotografo starebbe "memorizzando a memoria" le foto specifiche di training (dettagli irrilevanti, rumore) invece di imparare davvero a generalizzare il concetto di "posa della carrozzeria" e si comporterebbe male su foto mai viste.

Qui invece questo non succede: il fatto che l'errore di validazione scenda parallelamente (seppur con più "rumore"/oscillazioni, normale perché il validation set è più piccolo) è un buon segnale che il modello sta generalizzando correttamente.

Andamento della curva di validazione

Le piccole oscillazioni della curva di validazione (i picchi su e giù) sono normali: il validation set è più piccolo del training set, quindi la misura della loss è statisticamente più "rumorosa", ovvero basta qualche foto un po' più difficile in un certo batch per far salire temporaneamente il valore, senza che questo indichi un vero problema.

Questo grafico certifica che l'addestramento è andato bene: il modello ha imparato in modo stabile, senza segni di overfitting, e ha raggiunto un plateau di errore basso entro le 40 epoche previste.

```
import matplotlib.pyplot as plt
import torch
import numpy as np
from torch.utils.data import DataLoader, Subset

def visualizza_risultati_test(cfg, checkpoint_path="best_pose_model.pt"):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # 1. Carica il modello addestrato e i parametri di normalizzazione
        salvati
    ckpt = torch.load(checkpoint_path, map_location=device,
        weights_only=False)
    model = PoseRegressionNet().to(device)
    model.load_state_dict(ckpt["model_state"])
    model.eval()
```

```
# 2. Ricrea l'ESATTO Test Set usato in addestramento (utilizzando lo
    stesso seed 42)
# Assicurati che cfg.images_dir e cfg.csv_dir siano valorizzati
    correttamente nella tua classe Config
base_dataset_eval = PoseDataset(cfg.images_dir, cfg.csv_dir,
    cfg.csv_name, transform=get_transforms(train=False))

# --- MODIFICA CRITICA: Estrai gli indici di test esatti dal checkpoint
    ---
if "test_idx" in ckpt:
    test_idx = ckpt["test_idx"]
    print("Indici di test caricati con successo dal checkpoint.")
else:
    # Fallback nel caso il checkpoint sia vecchio e non abbia gli
        indici salvati
    print("Indici non trovati nel checkpoint, uso il seed 42 come
        fallback...")
    total_len = len(base_dataset_eval)
    n_test = int(total_len * cfg.test_split)
    n_val = int(total_len * cfg.val_split)
    n_train = total_len - n_val - n_test
    generator = torch.Generator().manual_seed(42)
    indices = torch.randperm(total_len, generator=generator).tolist()
    test_idx = indices[n_train + n_val:]

# 3. Ricrea il Test Set e il DataLoader
test_ds = Subset(base_dataset_eval, test_idx)
test_loader = DataLoader(test_ds, batch_size=cfg.batch_size,
    shuffle=False)

veri_pos, pred_pos = [], []
veri_ang, pred_ang = [], []

print(f"Generazione dei grafici sulle {len(test_ds)} immagini mai viste
    (Test Set)...")

# 3. Inferenza sui dati di test
with torch.no_grad():
    for imgs, targets in test_loader:
        imgs = imgs.to(device)
        preds = model(imgs).cpu().numpy()
        targets = targets.numpy()

        for i in range(len(preds)):
            # Decodifica POSIZIONE (de-normalizziamo le coordinate X, Y,
                Z)
            p_pred = preds[i, :3] * ckpt["pos_std"] + ckpt["pos_mean"]
```

```
            p_vera = targets[i, :3] * ckpt["pos_std"] + ckpt["pos_mean"]
            pred_pos.append(p_pred)
            veri_pos.append(p_vera)

            # Decodifica ANGOLI (da seno/coseno torniamo a radianti, e
                poi a gradi per leggibilita)
            a_pred = np.degrees(np.arctan2(preds[i, 3:6], preds[i, 6:9]))
            a_vera = np.degrees(np.arctan2(targets[i, 3:6], targets[i,
                6:9]))
            pred_ang.append(a_pred)
            veri_ang.append(a_vera)

# Convertiamo in array NumPy per facilitare il plotting
veri_pos = np.array(veri_pos)
pred_pos = np.array(pred_pos)
veri_ang = np.array(veri_ang)
pred_ang = np.array(pred_ang)

# 4. Disegniamo i grafici (Griglia 2 righe x 3 colonne)
fig, axs = plt.subplots(2, 3, figsize=(18, 10))

titoli_pos = ['Posizione X', 'Posizione Y', 'Posizione Z']
titoli_ang = ['Rotazione Roll', 'Rotazione Pitch', 'Rotazione Yaw']

for i in range(3):
    # --- RIGA 0: Grafici della Posizione ---
    axs[0, i].scatter(veri_pos[:, i], pred_pos[:, i], alpha=0.7,
        color='dodgerblue')
    # Calcoliamo i limiti per disegnare la linea retta ideale a 45 gradi
    min_p = np.min([veri_pos[:, i], pred_pos[:, i]])
    max_p = np.max([veri_pos[:, i], pred_pos[:, i]])
    axs[0, i].plot([min_p, max_p], [min_p, max_p], 'r--',
        label='Previsione Ideale')

    axs[0, i].set_title(titoli_pos[i], fontweight='bold')
    axs[0, i].set_xlabel('Coordinata Reale (Blender)')
    axs[0, i].set_ylabel('Coordinata Prevista (AI)')
    axs[0, i].legend()
    axs[0, i].grid(True)

    # --- RIGA 1: Grafici della Rotazione ---
    axs[1, i].scatter(veri_ang[:, i], pred_ang[:, i], alpha=0.7,
        color='darkorange')
    # Calcoliamo i limiti per la linea retta ideale
    min_a = np.min([veri_ang[:, i], pred_ang[:, i]])
    max_a = np.max([veri_ang[:, i], pred_ang[:, i]])
    axs[1, i].plot([min_a, max_a], [min_a, max_a], 'r--',
```

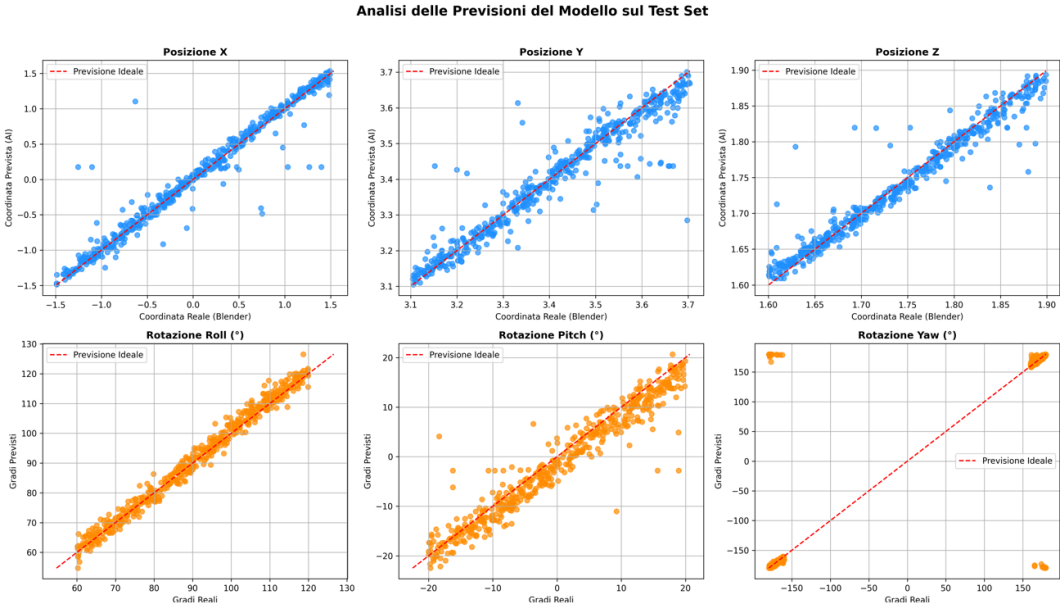


Figura 5.10: Analisi delle previsioni del modello

```
            label='Previsione Ideale')

        axs[1, i].set_title(f"{titoli_ang[i]} (gradi)", fontweight='bold')
        axs[1, i].set_xlabel('Gradi Reali')
        axs[1, i].set_ylabel('Gradi Previsti')
        axs[1, i].legend()
        axs[1, i].grid(True)

    plt.suptitle("Analisi delle Previsioni del Modello sul Test Set",
        fontsize=16, fontweight='bold', y=1.02)
    plt.tight_layout()
    plt.savefig("risultati_test_scatter.png", dpi=300, bbox_inches='tight')
    plt.show()

# Eseguiamo la funzione passando la configurazione attuale
cfg = Config()
visualizza_risultati_test(cfg)
```

Listing 5.6: Visualizzazione dei risultati sul Test Set

```
import zipfile
import os

# 1. Elenca i file esatti che vuoi scaricare
file_da_esportare = [
    "best_pose_model.pt",
    "curva_di_apprendimento.png",
    "risultati_test_scatter.png"
```

```
]

nome_archivio = "tutto_il_progetto.zip"

# 2. Crea l'archivio ZIP
with zipfile.ZipFile(nome_archivio, 'w', zipfile.ZIP_DEFLATED) as zipf:
    for file in file_da_esportare:
        if os.path.exists(file):
            zipf.write(file)
            print(f"Aggiunto all'archivio: {file}")
        else:
            print(f"Attenzione, file non trovato: {file}")

print(f"\nArchivio '{nome_archivio}' pronto per il download!")
```

Listing 5.7: Script per l'esportazione dei file del progetto in un archivio ZIP

In figura 5.10 abbiamo rappresentato un grafico a dispersione (perfetto per confrontare previsto vs reale perché la sua forma dice immediatamente quanto un modello sia preciso) di predicted vs ground truth sul test set, per ciascuna delle 6 componenti della posa stimata dalla rete (3 traslazionali + 3 rotazionali). La bisettrice tratteggiata rossa (y = x) rappresenta l'accuratezza perfetta; la dispersione dei punti attorno ad essa è una proxy visiva della varianza dell'errore residuo, componente per componente. Da notare è che questi plot mostrano i valori già denormalizzati, cioè dopo aver invertito la normalizzazione con pos_mean/pos_std per la posizione, e dopo aver riconvertito sin/cos in gradi per gli angoli.

Posizione X,Y,Z

Dalle immagini 5.10riscontriamo una buona correlazione lineare, residui contenuti e omoschedastici (varianza dell'errore relativamente costante lungo il range), con pochi outlier isolati. Le coordinate di posizione (x, y, z) sono numeri che si comportano in modo "normale": possono aumentare o diminuire, ma non "si mordono la coda". Se la telecamera è a x = 1.49 e poi si sposta leggermente a x = 1.51, il valore numerico cambia di poco (+0.02) e questo piccolo cambiamento corrisponde davvero a un piccolo spostamento fisico. A livello di angoli se siamo a 179° e ci spostiamo leggermente, potremmo ritrovarci a -179°. Numericamente sembra un salto enorme (quasi 360° di differenza), ma fisicamente ci siamo mossi pochissimo. Questo è un target periodico (si ripete ciclicamente ogni 360°). La posizione invece non ha questo problema: è non periodica, il che significa che "vicino nei numeri" corrisponde sempre a "vicino nella realtà". Quando la rete si allena, in pratica sta cercando di minimizzare la loss (l'errore) modificando i suoi pesi interni. Possiamo immaginare questo processo come una pallina che rotola giù per un paesaggio collinare, cercando la valle più bassa (il punto di minimo errore). Questo "paesaggio" è la superficie di errore (error surface / loss landscape). Dai risultati possiamo definire la superficie di errore che la

rete ottimizza è ben condizionata, ovvero questo "paesaggio" è relativamente liscio e regolare, senza nessuna discontinuità da gestire (una discontinuità sarebbe, in questo paesaggio metaforico, un salto a picco improvviso, un punto in cui spostandosi di pochissimo nei parametri della rete, l'errore calcolato schizza da un valore piccolo a uno enorme, senza vie di mezzo. Questo succederebbe se si usassero gli angoli grezzi in gradi come target: due situazioni fisicamente quasi identiche (179° e -179°) genererebbero un errore matematico gigantesco, creando uno "strappo" nel paesaggio della loss. Per la posizione questo non accade mai: l'errore cresce sempre in modo proporzionale e prevedibile alla distanza reale, quindi l'ottimizzatore ha "vita facile", ovvero la discesa verso il minimo è più stabile e la rete converge in modo più pulito e affidabile). Quindi la posizione è un tipo di target "comportato bene" da ottimizzare, perché piccoli errori numerici corrispondono sempre a piccoli errori reali, a differenza degli angoli grezzi, che invece nascondono questi "strappi" matematici dovuti alla loro natura ciclica. Questo spiega, tra l'altro, perché nello script il target angolare non venga usato in gradi grezzi, ma trasformato in seno/coseno, proprio per eliminare quella discontinuità e rendere anche la loss angolare "ben condizionata" quanto quella di posizione.

Rotazione: Roll e Pitch

Anche qui abbiamo una buona correlazione, con eteroschedasticità leggermente maggiore su Pitch (varianza del residuo cresce visibilmente verso gli estremi del range, ±20°). Questo può indicare che il training set ha meno campioni in quelle regioni della distribuzione (sparsità nei bin estremi), oppure che quell'asse di rotazione produce cambiamenti visivi più sottili/ambigui nell'immagine 2D, rendendo la regressione meno vincolata.

Rotazione: Yaw

Guardiamo il grafico della rotazione Yaw leggermente più nello specifico essendo il più difficile di interpretazione. L'asse delle x indica l'angolo di yaw reale (da Blender), in gradi, che va da -180° a +180°, mentre l'asse delle y indica l'angolo di yaw previsto dal modello, stessa scala.

Se il modello indovinasse sempre perfettamente, tutti i punti sarebbero sulla diagonale rossa. Invece i punti sono raggruppati in quattro zone agli angoli/estremi del grafico, e la diagonale centrale è quasi vuota. La scala dei gradi da -180° a +180° è in realtà un cerchio tagliato e steso in linea retta. Immaginiamo un orologio: le 11:59 e le 00:01 sono vicinissime nel tempo reale, ma se stendiamo il quadrante in una linea da 0 a 24 ore, quei due istanti finiscono agli estremi opposti della linea.

Con lo yaw succede la stessa cosa: +179° e -179° sono fisicamente quasi la stessa direzione (differiscono di soli 2°), ma sulla scala stesa in linea retta stanno ai due capi opposti. I due gruppi in alto a sinistra e in basso a destra sono punti dove, ad esempio, il valore reale è vicino a +170°/+180° e il modello ha previsto qualcosa vicino a

-170°/-180° (o viceversa). Numericamente sembra un errore enorme (quasi 350° di differenza), ma l'errore angolare reale è piccolissimo (il modello ha probabilmente previsto quasi correttamente, solo "dalla parte sbagliata del taglio del cerchio). I due cluster isolati vicino a (150,150)/(-150,-150) invece sono probabilmente casi in cui previsto e reale stanno dallo stesso lato del taglio (lì il modello è effettivamente accurato, ed è per questo che appaiono vicini alla diagonale). Il grafico non sta dicendo "il modello sbaglia molto lo yaw" ma sta mostrando un artefatto di visualizzazione dovuto al fatto che un angolo ciclico è stato disegnato su un asse lineare che ha un "punto di rottura" a ±180°, ed è proprio in quella zona che si concentrano molti dei nostri dati di test.

Il modello di pose estimation raggiunge quindi una buona accuratezza su quasi tutte le componenti della posa, dimostrando che l'approccio è efficace per il nostro caso d'uso (localizzare gli smart glasses rispetto alla carrozzeria per il posizionamento del logo in AR). Il modello stima con buona precisione la posizione 3D e l'orientamento (roll, pitch) della telecamera rispetto alla carrozzeria. La componente di yaw richiede un'analisi dell'errore corretta per la natura ciclica dell'angolo, poiché la metrica di errore lineare standard sovrastima l'imprecisione reale in prossimità del punto di discontinuità (±180°). Complessivamente, i risultati indicano che il modello è promettente per il compito di localizzazione necessario al posizionamento AR del logo, con un margine di miglioramento identificato specificamente nella valutazione (e potenzialmente nel training) della componente yaw. Questo, insieme ai buoni risultati nella fase di training , indica in dominio sintetico che l'approccio è promettente per il compito di localizzazione. La validazione è però limitata a immagini renderizzate di un singolo modello di veicolo: il trasferimento al dominio reale e la generalizzazione ad altre vetture restano da verificare sperimentalmente e costituiscono il naturale sviluppo del lavoro.

Capitolo 6

Conclusioni

Attraverso l'indagine condotta in questo studio, è stato possibile delineare il ruolo dei diversi metodi di sviluppo e progettazione nel campo della Computer Vision e la loro efficacia nella nostra ricerca specifica. Il presente lavoro non si configura originariamente come una linea guida prescrittiva per la progettazione della telecamera da integrare negli smart glasses. Al contrario, esso intende documentare il percorso di esplorazione e brainstorming relativo ai differenti approcci concepiti, evidenziando le criticità riscontrate e le soluzioni progressivamente adottate. L'elevato numero di idee vagliate è stato costantemente confrontato con gli obiettivi prefissati; tuttavia, l'analisi dei limiti emersi da ciascuna ipotesi non ha condotto a una mera forzatura o al semplice adattamento dei codici esistenti, bensì al raggiungimento di un'architettura software flessibile e capace di rispondere in modo robusto alle problematiche affrontate. Nel panorama attuale, l'integrazione delle reti neurali ha segnato un cambio di paradigma definitivo per le tecniche di Computer Vision, superando i limiti degli approcci tradizionali basati sull'estrazione manuale delle caratteristiche. L'evoluzione dalle architetture convoluzionali (CNN) ai più recenti modelli basati su Vision Transformers (ViT) ha dimostrato come l'apprendimento profondo consenta di raggiungere livelli di accuratezza, robustezza e generalizzazione senza precedenti in compiti complessi quali la classificazione, la segmentazione e il tracciamento di oggetti. Anche nel contesto di questo studio, l'adozione delle reti neurali si è rivelata determinante per il raggiungimento di un primo risultato pienamente soddisfacente. Ciò nonostante, fondamentale è stato il contributo degli approcci tradizionali, analizzati e sviluppati in maniera graduale secondo un livello di complessità crescente, orientato alle specifiche esigenze della ricerca. Quanto esposto in questo elaborato rappresenta solo una sintesi dell'articolato processo di progettazione e sviluppo: un percorso che ha permesso di comprendere come l'analisi degli errori riscontrati sia preziosa tanto quanto i successi ottenuti. L'esame critico dei limiti emersi da ciascuna tentata soluzione ha infatti ampliato la prospettiva verso strategie risolutive più articolate e robuste, fornendo le basi metodologiche per guidare i futuri sviluppi verso il raggiungimento dell'obiettivo finale.

Bibliografia

[1] InAir Space, *Smart Glasses How They Work: A Deep Dive Into Wearable Visual Computing*, https://inairspace.com/blogs/learn-with-inair/smart-glasses-how-they-work-a-deep-dive-into-wearable-visual-computing

[2] Syte, *Visual Search*, https://www.syte.ai/glossary/visual-search/

[3] C. Randieri, *Rassegna Sistemi di visione*, https://www.researchgate.net/profile/Cristian-Randieri-2/publication/280621291_Rassegna_Sistemi_di_visione_Intellisystem_Technologies/links/5948dc38a6fdcc70635b82c1/Rassegna-Sistemi-di-visione-Intellisystem-Technologies.pdf

[4] B. Gajsek et al., *Visual and optometric issues with smart glasses in Industry 4.0 working environment*, https://www.researchgate.net/profile/Brigita-Gajsek/publication/329882815_Visual_and_optometric_issues_with_smart_glasses_in_Industry_40_working_environment/links/5c3cca3192851c22a3749b4a/Visual-and-optometric-issues-with-smart-glasses-in-Industry-40-working-environmen
pdf

[5] IEEE Technology Navigator, *Smartglasses*, https://technav.ieee.org/topic/smartglasses/

[6] *The YOLO Framework: A Comprehensive Review of Evolution, Applications, and Benchmarks in Object Detection*, MDPI, https://www.mdpi.com/2073-431X/13/12/336

[7] *Augmented reality smart glasses in industrial assembly: Current status and future challenges*, ScienceDirect, https://www.sciencedirect.com/science/article/abs/pii/S2452414X20300509

[8] *A Smart Eyewear using IoT and CNNs for Visual Assistance*, IEEE Xplore, https://ieeexplore.ieee.org/abstract/document/10696418

[9] *Deep Learning vs. Traditional Computer Vision*, arXiv, https://arxiv.org/pdf/1910.13796

[10] *Multi-scale Template Matching with Scalable Diversity Similarity in an Unconstrained Environment*, arXiv, https://arxiv.org/pdf/1907.01150

[11] *Template Matching Advances and Applications in Image Analysis*, arXiv, https://arxiv.org/pdf/1610.07231

[12] *Applicazioni di Deep Learning con MATLAB: Reti Neurali Attraverso Esempi*, Google Books, https://books.google.it/books?hl=it&lr=&id=xBm4EQAAQBAJ&oi=fnd&pg=PA3&dq=deep+learning+con+reti+neurali+dataset&ots=fti8AloWiW&sig=0SbVhoIkQRfHl_MnM0x4KT2Pt8Y&redir_esc=y#v=onepage&q&f=false

[13] *BlenderProc*, arXiv, https://arxiv.org/pdf/1911.01911

[14] H-FARM College, *Reti Neurali: cosa sono e come funzionano davvero*, https://college.h-farm.com/news/reti-neurali-cosa-sono-come-funzionano-ai/

[15] *Introduzione alle reti neurali - pesi, bias e attivazione*, Diario di un Analista, https://www.diariodiunanalista.it/posts/introduzione-alle-reti-neurali-pesi-bias-e-attivazione/

[16] Google Cloud, *Che cos'è una rete neurale?*, https://cloud.google.com/discover/what-is-a-neural-network?hl=it

[17] *SupeRANSAC: One RANSAC to Rule Them All*, arXiv, https://arxiv.org/pdf/2506.04803

[18] MathWorks, *Feature Matching*, https://it.mathworks.com/help/vision/ref/matchfeatures.html#doc_center_content

[19] Y. Chen, L. Huang, *MATLAB Roadmap to Applications* https://link.springer.com/content/pdf/10.1007/978-981-97-8788-3.pdf

[20] MathWorks, *Local Feature Detection and Extraction*, https://it.mathworks.com/help/vision/ug/local-feature-detection-and-extraction.html

[21] Automatic Addison, *Image Feature Detection, Description, and Matching in OpenCV*, https://automaticaddison.com/image-feature-detection-description-and-matching-in-opencv/

[22] *Visión artificial: Aplicaciones prácticas con OpenCV - Python*, Google Books, https://books.google.it/books?hl=it&lr=&id=FE1OEAAAQBAJ&oi=fnd&pg=PT5&dq=opencv+python&ots=3zFoBj0iY2&sig=mwyHKPduEt7WA_AnXYchBVMjGc8&redir_esc=y#v=onepage&q=opencv%20python&f=false

[23] Z. Horak, *Detection of Symmetric Features in Images* https://www.researchgate.net/publication/261268823_Detection_of_Symmetric_Features_in_Images