ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE - DIN

CORSO DI LAUREA IN INGEGNERIA MECCANICA LM

TESI DI LAUREA

in

Meccanica delle Macchine

Sviluppo di un software basato sull'analisi di immagini per la pianificazione di traiettoria di un robot a cavi

CANDIDATO:
Alessandro Iaia

RELATORE:
Prof. Ing. Marco Carricato

CORRELATORI:
Edoardo Idà
Simone Comari

Anno accademico 2019/2020

Sessione II

Introduzione

La marcatura è uno dei processi tecnologici più diffusi nel mondo industriale, tanto che, fra le applicazioni laser, è quella che vanta il maggior numero di sistemi installati. Il progetto *Laser Engraver* è nato con lo scopo di realizzare un prototipo di marcatrice presso i laboratori del GRAB (Group of Robotics, Automation and Articular Biomechanics) dell'Università di Bologna. L'obiettivo è quello di poter incidere materiali come legno, carta e cartone, contenendo i costi e rendendo il prototipo facilmente scalabile e trasportabile.
Sulla base di queste specifiche si è deciso di adottare, per questa macchina, l'architettura dei robot a cavi, i quali si distinguono dai più classici manipolatori paralleli per via dell'adozione di elementi flessibili, che collegano il membro terminale della catena cinematica al telaio. L'impiego di cavi come organi di trasmissione comporta, tuttavia, una serie di problematiche dovute alla natura elastica degli stessi. Di conseguenza, questi robot sono ancora, al momento, poco diffusi in ambito industriale. Viceversa, essi risultano di grande interesse nella ricerca scientifica, poiché il superamento dei limiti dei manipolatori tradizionali può portare a ottenere macchine molto più versatili e meno costose da produrre e manutenere.
Il software che gestisce il Laser Engraver ha lo scopo di generare, a partire da un'immagine digitale, un file di testo contenente le istruzioni in GCode per la realizzazione della marcatura dell'immagine stessa sul pezzo. Il GCode è un linguaggio normato, basato su sigle alfanumeriche, utilizzato per assegnare istruzioni all'unità di controllo di una macchina CNC. Dal punto di vista delle strutture dati, un'immagine digitale (o *bitmap*) è una matrice, in cui ciascun punto (*pixel*) ha un valore che dipende dal suo colore. Elaborando la bitmap secondo opportuni algoritmi, è possibile trasformarne il contenuto in modo tale da ricavarne una traiettoria di punti, a ciascuno dei quali è associato un valore che dipende dal colore di quel particolare pixel. Sulla base di quest'ultimo parametro, la potenza del laser può essere regolata, punto per punto, per ottenere una profondità di passata adeguata a replicare sul pezzo la scala di grigi dell'immagine di partenza.
Questa tesi si concentra sullo sviluppo del codice appena descritto, che costituisce il back-end di un'applicazione desktop corredata di un'interfaccia grafica, anch'essa oggetto dell'elaborato. L'intero pacchetto software è stato sviluppato in MATLAB.

Nel capitolo 1 vengono richiamati i processi di marcatura laser e si introduce il progetto Laser Engraver. Il capitolo 2 fornisce una panoramica sul mondo della robotica a cavi, con esempi applicativi e modelli di controllo. Il capitolo 3 offre un compenso delle nozioni di *image processing* necessarie alla comprensione degli algoritmi implementati. Il capitolo 4 è dedicato al software per la pianificazione di traiettoria, che viene dettagliatamente descritto, sotto forma di pseudocodice, in tutte le sue parti. Nel capitolo 5 si illustra, infine, l'interfaccia sviluppata per l'applicazione.

Indice

Capitolo 1

La marcatura laser

La marcatura laser permette di incidere in maniera indelebile scritte e disegni su un oggetto solido, anche se non necessariamente rigido. È possibile lavorare un'ampia gamma di materiali: acciaio, plastica, vetro, legno, carta ecc.
Il basso costo dell'attrezzatura necessaria a questo processo ha portato a una diffusione su larga scala di macchine da marcatura, tanto che, fra le applicazioni laser, questa è quella che vanta il maggior numero di sistemi installati.
La marcatura ha diverse varianti, che si differenziano principalmente per meccanismo fisico e modalità di movimentazione della testa laser. Prima di affrontare questi aspetti, viene dedicato breve spazio a dei richiami generali sui sistemi laser.

1.1 Generalità sui sistemi laser

La radiazione laser è resa possibile dal fenomeno dell'*emissione stimolata* (fig. 1.1), che si verifica quando un fotone (o, equivalentemente, un'onda elettromagnetica) investe un atomo che si trova in uno stato energetico già eccitato da una fonte esterna. Il risultato è l'emissione, da parte dello stesso atomo, di un altro fotone, avente la medesima frequenza, direzione e fase della radiazione incidente.

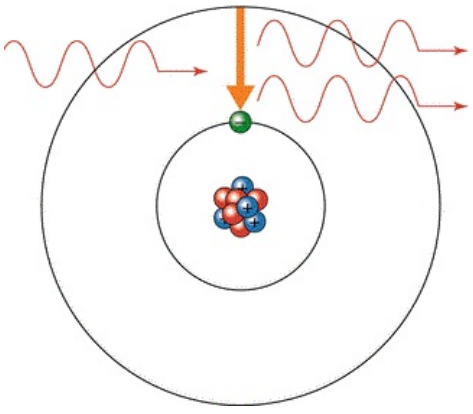


Figura 1.1: Emissione stimolata di fotoni da parte di un atomo.

Un fascio laser è una radiazione elettromagnetica prodotta mediante l'emissione stimolata di fotoni da parte di un materiale solido o gassoso (detto *mezzo attivo*), opportunamente eccitato. Tale eccitazione (*pompaggio*) consente agli atomi del mezzo attivo di compiere il salto energetico necessario all'innesco dell'emissione stimolata,

meccanismo che, se avviene con continuità, è in grado di generare un flusso di particelle dirette lungo l'asse della cavità contenente il mezzo attivo (detta *risonatore*). In sostanza, sulla base di quanto appena descritto, i componenti fondamentali di una sorgente laser sono:

- **Mezzo attivo**: è l'elemento a partire dal quale viene generata la radiazione.
- **Sorgente energetica di pompaggio**: ha lo scopo di eccitare gli atomi del mezzo attivo. La sua scelta dipende proprio dalle caratteristiche di quest'ultimo. In genere, il pompaggio avviene mediante una scarica elettrica o una radiazione luminosa.
- **Risonatore**: assicura che i fotoni restino nel mezzo attivo per un tempo sufficiente a determinare l'emissione spontanea e conferisce una direzione preferenziale alla radiazione. La tipologia di risonatore più semplice è costituita da un cilindro cavo, avente due specchi piani alle estremità. Esistono però svariate configurazioni possibili, a seconda delle esigenze dell'impianto.

Come mostrato in figura 1.2, la radiazione laser si distingue da quella propria di altre sorgenti (naturali o artificiali) per via di alcune importanti caratteristiche:

- **Monocromaticità**: le onde elettromagnetiche che costituiscono il fascio possiedono tutte la stessa lunghezza d'onda, la quale dipende unicamente dal mezzo attivo impiegato.
- **Coerenza spaziale e temporale**: le radiazioni componenti il fascio oscillano tutte con la stessa fase, che si conserva sia nello spazio (*coerenza spaziale*) che nel tempo (*coerenza temporale*).
- **Bassa divergenza**: il raggio laser in uscita dal risonatore ha una bassa divergenza, quindi può essere facilmente focalizzato (mediante un opportuno sistema ottico), in modo da poter concentrare tutta la sua energia su aree molto piccole, ottenendo così elevati valori di *irradianza* (potenza per unità di superficie).

Una volta generato dalla sorgente, il raggio deve essere prima trasportato e poi focalizzato sul pezzo in lavorazione. Il processo tecnologico necessita, inoltre, di un moto relativo fra utensile e pezzo. Di conseguenza, le altre componenti fondamentali di un impianto laser industriale sono:

- **Sistema di trasporto**: in genere si utilizzano degli specchi in rame, opportunamente orientati, oppure, se la lunghezza d'onda della radiazione lo consente, una fibra ottica, che tuttavia provoca un aumento della divergenza del fascio.
- **Sistema di focalizzazione**: la focalizzazione può avvenire tramite una lente (*per trasmissione*) o uno specchio concavo (*per riflessione*). La scelta si basa sul tipo di sorgente impiegata e sulla sua potenza.
- **Sistema di movimentazione**: crea un moto relativo fra laser e pezzo. Esistono sia sistemi a pezzo fisso e fascio mobile che viceversa.

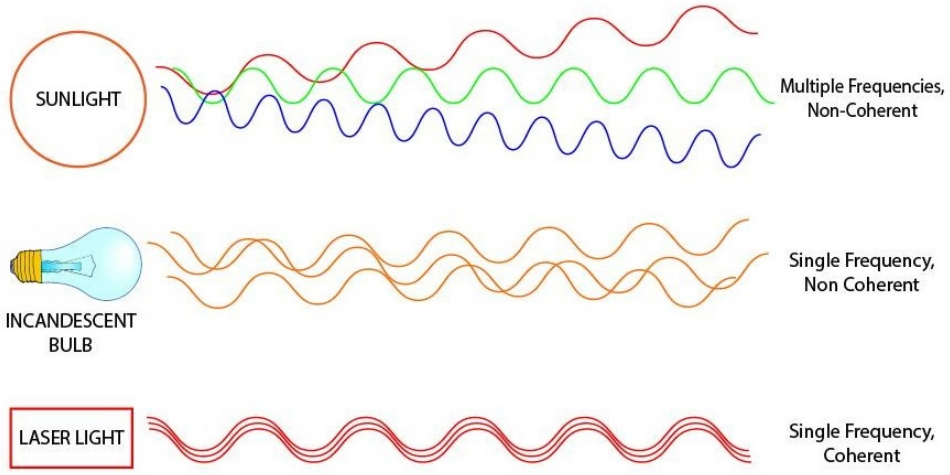


Figura 1.2: Confronto fra le caratteristiche delle radiazioni proprie del sole, delle lampadine a incandescenza e delle sorgenti laser.

Siccome la generazione del laser è un processo a bassa efficienza energetica, la sorgente necessita anche di un'unità frigorifera che sia in grado di smaltire il calore sviluppato.

La scelta della miglior sorgente per una specifica applicazione dipende da fattori di natura tecnico-economica (materiale da lavorare, costo dell'impianto, tipo di processo tecnologico ecc...). A tal proposito, la tabella di figura 1.3 dà un quadro delle principali sorgenti laser di impiego industriale.

	CO_2	**Nd:YAG**	**Diodi**	**Fibra**
Potenza Max [kW]	40	6÷15	10	100
Irradianza Max [W/cm^2]	10^8	10^7÷10^9	10^6	10^9
Lunghezza d'onda [μm]	10.6	1.06	0.80÷0.95	1.03
Rendimento [%]	20	5÷20	30÷40	30
Investimento iniziale	Medio	Elevato	Basso	Elevato

Figura 1.3: Caratteristiche delle principali sorgenti laser [6].

Il laser a diodi è solitamente legato a impieghi low-cost, in cui non è richiesta un'elevata densità di potenza. A differenza delle altre sorgenti, che generano un raggio a sezione circolare, per le sorgenti a diodi la sezione risulta rettangolare. Il fascio, inoltre, non è facilmente focalizzabile, rendendo questo tipo di sorgente inadatto alle lavorazioni di alta precisione. La sua struttura, comunque, è compatta e l'elevato rendimento elimina la necessità di un raffreddamento ad acqua.
Il laser Nd:YAG e quello in fibra sono entrambi in grado di lavorare materiali metallici, plastici e ceramici, ma il primo richiede maggior manutenzione e, in genere, ha un tempo di vita inferiore. Il laser in fibra ha una struttura compatta e non necessita di un sistema di raffreddamento. In più, il fascio è focalizzabile su aree più piccole rispetto a quelle ottenibili con le altre sorgenti, rendendolo particolarmente adatto ad applicazioni in cui è richiesta elevata precisione.
Il laser a CO_2 è ideale per la lavorazione di materiali polimerici e organici. Fra le

sorgenti citate, questa è l'unica che non permette il trasporto attraverso una fibra ottica.

L'accoppiamento sorgente-materiale è vincolato, per qualunque sorgente, dall'interazione laser-materia che si verifica quando il raggio, dopo la focalizzazione, giunge a contatto con il pezzo. La radiazione incidente sulla superficie viene in parte riflessa e in parte assorbita dal materiale. La percentuale di potenza termica assorbita rispetto a quella totale incidente sul pezzo è detta *coefficiente di assorbimento* ed è funzione della lunghezza d'onda del laser. La frazione di potenza assorbita è quella che, esercitando una forza elettromagnetica sulle particelle del materiale dotate di carica, le mette in movimento, provocando un aumento macroscopico della temperatura del pezzo, che scaturisce nell'ossidazione, fusione o vaporizzazione dello stesso, a seconda dei valori di irradianza in gioco. La figura 1.4 mostra l'andamento del coefficiente di assorbimento per materiali metallici e non metallici ed evidenzia come, confrontando i due casi, il comportamento sia molto diverso per lunghezze d'onda elevate.

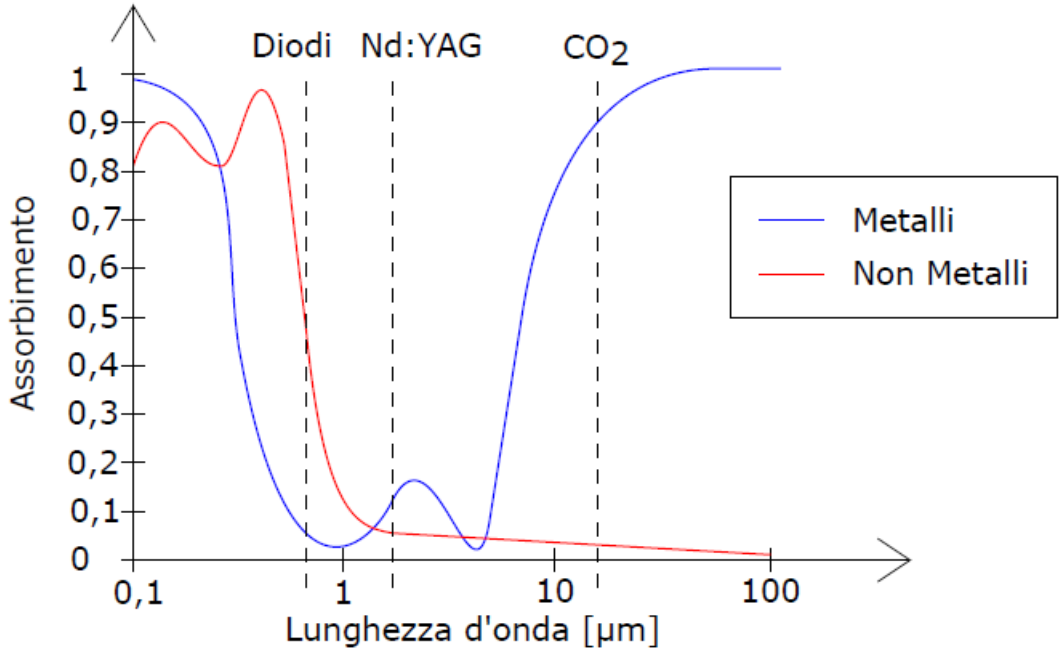


Figura 1.4: Spettro di assorbimento delle radiazioni laser per materiali metallici (blu) e non metallici (rosso). Le linee tratteggiate verticali rappresentano la lunghezza d'onda propria del laser della tipologia indicata nella didascalia.

1.2 Implementazione del processo

Il termine *marcatura* si riferisce, in realtà, a una serie di processi, facenti capo a due macro-categorie:

- **Incisione**: il meccanismo di rimozione del materiale è la sublimazione. Il solco ha una profondità stabilita ed è solitamente di colore bruno, per via dell'ossidazione indotta dal calore sulle pareti. Quando il materiale è costituito da più strati e la rimozione interessa solo parte di essi, il processo prende il

nome di *ablazione*. In questo caso, la radiazione deve essere assorbita solo dalla porzione di materiale da sublimare e riflessa da quella sottostante.

- **Marcatura** propriamente detta: il laser scalda il materiale al di sotto del punto di fusione, provocando modifiche alla sua microstruttura. Il risultato è che le zone marcate hanno un indice di rifrazione diverso da quelle non lavorate. Siccome le proprietà di riflessione della luce sono legate a tale indice, la zona esposta al fascio risulta visibile all'occhio umano. La marcatura, a differenza dell'incisione, non crea un intaglio sul pezzo, quindi è particolarmente indicata per quei componenti che sono soggetti a sollecitazioni cicliche durante il loro impiego.

Spesso questi due termini vengono accorpati nella più generica definizione di marcatura (vedi [7], ad esempio). In effetti, come si vedrà nel capitolo 4, questa distinzione non è affatto importante ai fini della scrittura del software di pianificazione della traiettoria, obiettivo di questa tesi. Per questo motivo, d'ora in poi si eviterà questo formalismo e si parlerà, più semplicemente, di marcatura in senso lato.
Di maggior interesse è, invece, la classificazione dei processi in funzione del sistema di movimentazione della testa laser rispetto al pezzo. Le possibilità sono tre:

- **Marcatura lineare**: il fascio viene movimentato come una sorta di matita, tracciando un solco della dimensione dello *spot*[1] del laser sul pezzo. Le linee di spessore maggiore vengono ricavate con più passate affiancate oppure defocalizzando il fascio. Questa modalità è la più flessibile e, di conseguenza, anche la più diffusa.

- **Marcatura a scansione**: il laser si muove scansionando un'area rettangolare e viene attivato solo nei punti che devono essere anneriti. Nel caso in cui la sorgente sia in grado di controllare la potenza istantanea, è possibile, variando tale parametro punto per punto, ottenere marcature in scala di grigi. Il limite alla ricchezza di dettagli riproducibili dipende dalla risoluzione del sistema di scansione.

- **Marcatura d'area**: il fascio defocalizzato investe una maschera, rimanendone parzialmente oscurato. La parte trasmessa viene poi focalizzata da una lente sul pezzo, in modo che tutta la superficie da marcare venga esposta nel medesimo istante alla radiazione. Non vi è, dunque, moto relativo fra testa e pezzo, ma, d'altra parte, è importante che la distribuzione di potenza sulla sezione del fascio sia uniforme, al fine di evitare disuniformità nel colore della zona lavorata. Inoltre, le sorgenti che adoperano questa modalità devono essere in grado di generare un'elevata potenza di picco, così da poter garantire un tempo di marcatura contenuto.

La modalità implementata nel software oggetto dell'elaborato è la seconda.
I parametri di processo sono:

- potenza del laser;

- velocità relativa laser-pezzo;

[1] Intersezione tra il fascio laser e la superficie del pezzo (vedi fig. 1.5).

- diametro dello spot e distanza focale;
- frequenza della radiazione.

La frequenza del laser viene solitamente scelta nell'intorno dei picchi dello spettro di assorbimento del materiale che si desidera trattare. Il diametro dello spot corrisponde alla larghezza del solco lasciato sul pezzo. Maggiore è il diametro, minore è la risoluzione ottenibile nella marcatura. Con riferimento alla figura 1.5, il fascio, dopo essere stato focalizzato da una lente convergente, abbandona l'andamento cilindrico per assumere quello di un iperboloide iperbolico di rotazione. Il diametro minimo della sezione trasversale del raggio viene chiamato *diametro focale*, mentre la sua distanza dalla lente è detta *distanza focale*. Il punto del fascio a sezione minima è detto *fuoco*. In genere, si fa in modo di collocare quest'ultimo in corrispondenza della superficie del pezzo, così da ottenere il massimo valore di irradianza e limitare l'estensione della zona termicamente alterata. Ci sono però contesti in cui si preferisce posizionare il fuoco al di sotto della superficie del pezzo, in particolare nel caso dell'incisione di materiali fragili.

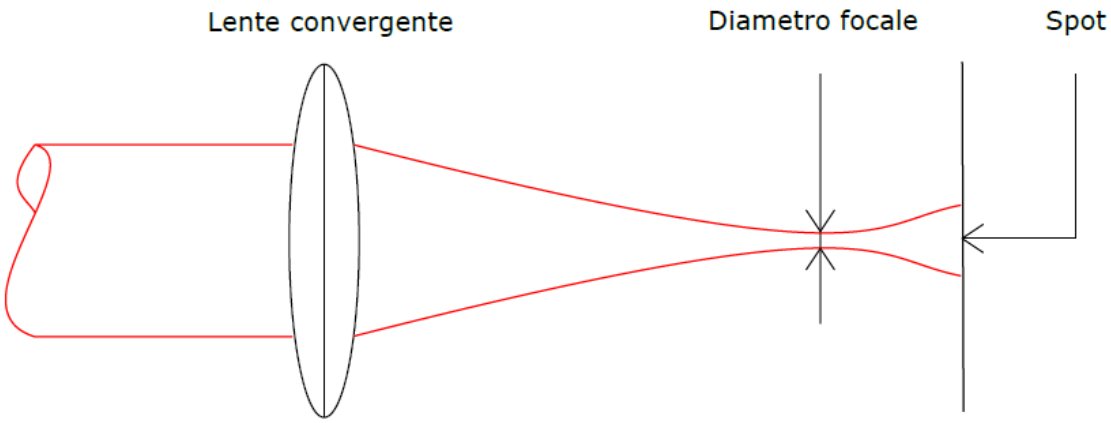


Figura 1.5: Schema della conformazione del fascio laser prima e dopo la focalizzazione tramite lente convergente.

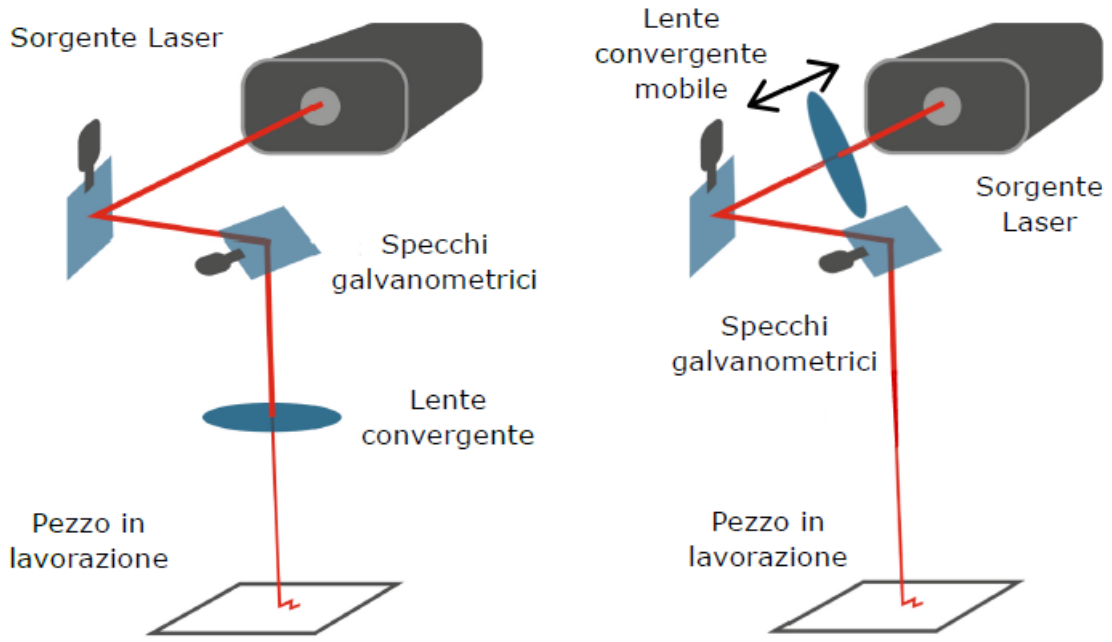
I principali parametri di output della lavorazione sono la profondità del solco e la rugosità delle sue pareti. L'interazione fra grandezze di input e di output è riassumibile nei seguenti punti [17]:

- la profondità aumenta incrementando la potenza del laser oppure diminuendo la velocità di processo;
- la rugosità diminuisce aumentando la potenza del laser oppure diminuendo la velocità di processo;
- per garantire una buona visibilità del solco è bene utilizzare laser a frequenza e potenza media relativamente basse.

In base a queste osservazioni sperimentali, è evidente che, se si vuole ottenere una profondità uniforme, è necessario che la velocità di processo vari proporzionalmente alla potenza. Se, invece, l'obiettivo è quello di una marcatura con profondità variabile, si può controllare quest'ultima variando la potenza e mantenendo la velocità

di processo costante. La seconda strategia è esattamente quella implementata nel codice presentato nel capitolo 4.

Le sorgenti impiegate per la marcatura sono quelle a CO_2 e Nd:YAG. I laser a diodi sono raramente usati in ambito industriale, per via delle grandi dimensioni dello spot. Le sorgenti a CO_2 hanno, tipicamente, potenze comprese fra 10 e 200 W e sono particolarmente indicate per la marcatura propriamente detta. Le sorgenti Nd:YAG, invece, sono più consone all'incisione e hanno potenze mediamente comprese fra 10 e 50 W. In generale, il costo di un impianto laser aumenta con la potenza dello stesso, motivo che spiega il perché i sistemi per la marcatura siano molto più economici di quelli per il taglio.



(a) Architettura classica (b) Architettura alternativa

Figura 1.6: Schema funzionale delle più comuni teste galvanometriche da marcatura.

Il sistema di trasporto, focalizzazione e movimentazione del fascio impiega tre elementi ottici: due specchi galvanometrici e una lente convergente. Questi componenti vengono racchiusi in una testina, insieme alla sorgente, in modo da proteggerli dai fumi prodotti durante la lavorazione. I due specchi, posizionati ad assi sghembi con inclinazione relativa di 90°, possono essere ruotati da due motori elettrici. L'architettura classica, illustrata in figura 1.6a, prevede il posizionamento della lente a valle degli specchi, parallela e centrata rispetto al supporto sul quale è appoggiato il pezzo in lavorazione. Cambiando la posizione angolare degli specchi, è possibile far spazzare al laser l'intera area di lavoro. La quota della lente può essere regolata prima della lavorazione, così da poter definire la posizione del fuoco rispetto alla superficie del pezzo.
Il principale punto debole di quest'architettura è la difficoltà nella lavorazione di aree estese. Quando, infatti, il laser si muove verso la periferia dell'area di lavoro, il fuoco, a sua volta, si sposta verso l'alto, allontanandosi dalla superficie del pezzo, poiché la distanza focale resta invariata, mentre il tratto lente-pezzo aumenta. In

secondo luogo, per via della non ortogonalità tra il fascio e la superficie, la forma dello spot risulta sempre più distorta man mano che ci si muove verso la periferia. A motivo delle carenze appena esposte, sistemi di questo tipo non possono essere impiegati su aree maggiori di 2x2 metri. A questo inconveniente si aggiunge l'impossibilità di lavorare superfici tridimensionali, dato che non è possibile regolare la quota della lente di focalizzazione durante il processo.
Una valida alternativa è offerta dalla soluzione in figura 1.6b, in cui la lente è anteposta agli specchi e dotata di un grado di libertà traslatorio lungo la direzione del fascio. È così possibile regolare istantaneamente la posizione del fuoco rispetto alla superficie del pezzo, sia essa parallela al supporto o inclinata, purché non verticale. Questo sistema permette di estendere l'area di lavoro fino a 4x4 metri, anche se l'aumento del numero di moti controllati rallenta la lavorazione, inficiando di conseguenza la produttività.
Esistono anche architetture in grado di eliminare la deformazione prospettica dello spot e lavorare, quindi, superfici di grandi dimensioni, anche verticali o free-form. In questo caso non viene più movimentato il raggio in uscita dalla sorgente, ma la sorgente stessa. Gli specchi galvanometrici non sono più necessari, quindi la testina racchiude al suo interno solo la sorgente e la lente di focalizzazione. Se si vogliono realizzare lavorazioni planari su regioni parallele al supporto, sono sufficienti due guide ortogonali per il posizionamento nel piano e un sistema di regolazione della quota della lente (o dell'intera testina) per gestire la posizione del fuoco in fase di calibrazione. Questo sistema è, tuttavia, più lento e meno preciso di quello di figura 1.6a, poiché movimenta masse maggiori su percorsi più lunghi. Per ottenere risultati ottimali su pezzi di geometria complessa, si adotta un approccio ibrido. L'area di lavoro viene virtualmente divisa in zone. Per ciascuna di esse, la testa galvanometrica (fig. 1.6a oppure 1.6b) opera da una differente posizione, che non varia finché la marcatura di quella particolare zona non è stata completata. Ne risulta che i movimenti delle ottiche, che sono successivi a quelli di appostamento, sono di entità minore rispetto al caso in cui la testa operi da una posizione fissa durante l'intera lavorazione.

1.3 Progetto Laser Engraver

Il progetto Laser Engraver è nato con lo scopo di realizzare un prototipo di marcatrice laser presso il GRAB (Group of Robotics, Automation and Articular Biomechanics) dell'Università di Bologna. Le specifiche che hanno guidato le scelte progettuali sono:

- area di lavoro fissa, piana e verticale, di dimensione 1x1 metro;
- marcatura di carta, cartone e legno;
- contenimento dei costi;
- possibilità di apportare agevolmente modifiche al prototipo in fasi successive, ad esempio scalarlo, renderlo portatile o variare l'orientamento del piano di lavoro da verticale a orizzontale.

Il progetto si articola su due fronti: quello dell'hardware meccanico ed elettrico/elettronico e quello del software. Più nello specifico:

- Hardware meccanico ed elettrico/elettronico:
  - telaio e supporto;
  - testina (laser, componenti ottici, case ecc.);
  - sistema di movimentazione della testina;
  - sistema di controllo;
  - sistema di alimentazione;
- Software:
  - GCode generator;
  - GCode parser;
  - firmware per la gestione del processo e il controllo degli attuatori;

Una buona parte della progettazione hardware è già stata affrontata da altri studenti. La parte rilevante per questa tesi è invece, come noto, quella software.

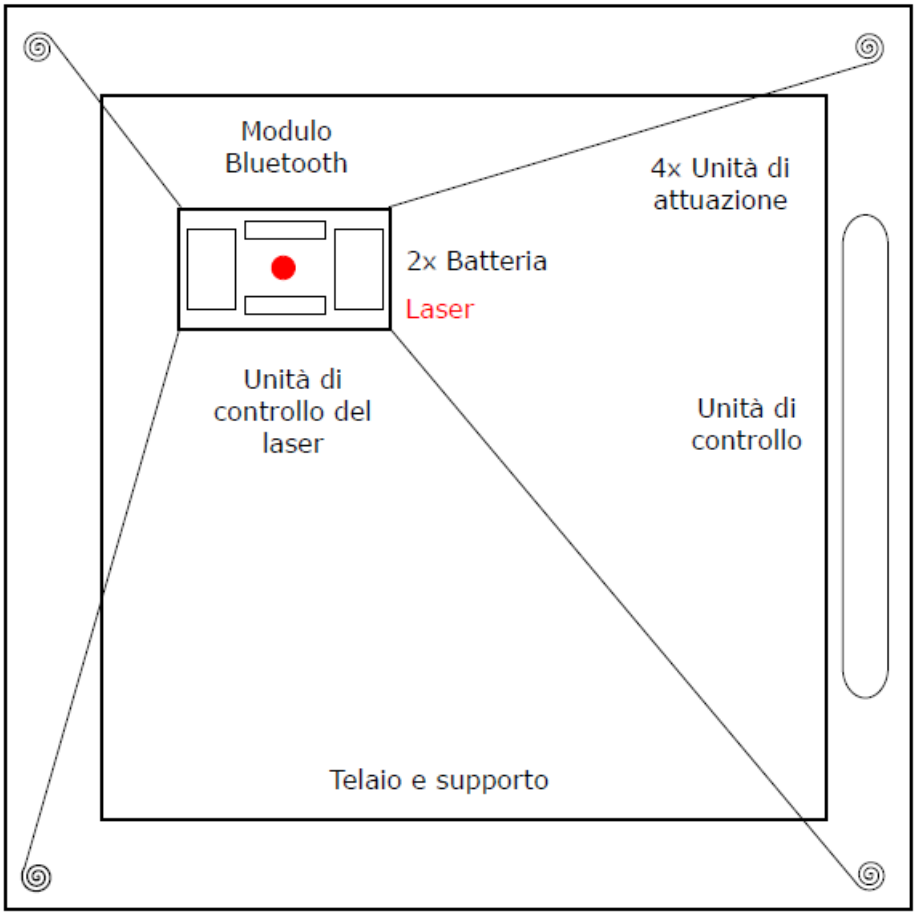


Figura 1.7: Rappresentazione schematica del prototipo di marcatrice laser.

In figura 1.7 si mostra un disegno schematico del prototipo. Per asservire l'intera area di lavoro in modo agevole, si è scelto di adoperare una piattaforma mobile

dotata di una sorgente laser e una lente di focalizzazione. Per venire incontro all'esigenza del contenimento dei costi, la sorgente scelta è a diodi e ha una potenza di 2.5 W. La lente è già presente all'interno dell'involucro che contiene la sorgente e la sua posizione è regolabile tramite una rotella esterna al case. Il laser scelto è in grado di raggiungere valori di irradianza sufficienti a marcare legno, carta e cartone con una risoluzione accettabile. Il suo funzionamento è impulsato, ovvero la potenza media è regolabile tramite un segnale di controllo (ad esempio PWM). Tale segnale è un'onda quadra periodica che può assumere soli valori binari. Il parametro che la caratterizza è il *duty cycle*, ovvero il rapporto fra il tempo di accensione in un ciclo e il periodo dell'onda. Il duty cycle ha dunque, per definizione, valori compresi fra 0 (potenza nulla) e 1 (potenza massima). Variandolo è possibile ottenere tutti i valori di potenza intermedi; si possono quindi realizzare marcature in scala di grigi, secondo la logica di controllo già introdotta nella sezione 1.2.
Per garantire la scalabilità del prototipo, imposta dalle specifiche di progetto, si è scelto di adottare l'architettura dei robot a cavi. In questo modo, infatti, il sistema può essere ridimensionato a piacere senza dover riprogettare il sistema di attuazione. Inoltre, rispetto a una più tradizionale attuazione lineare, all'aumentare delle dimensioni dell'area di lavoro si ottengono rendimenti maggiori e costi minori.
A eccezione di tre componenti, oltre a quelli commerciali (viti, profilati ecc.), tutti gli altri sono stati realizzati mediante FDM (*Fused Deposition Modeling*), ancora una volta in ottica di prototipazione low-cost. Fra questi vi è la testina, formata da una piattaforma che ospita il laser, un modulo bluetooth per la comunicazione col firmware di controllo e delle batterie, che alimentano i due dispositivi appena menzionati. Il telaio (fig. 1.8) è costituito da profilati commerciali in alluminio estruso, fissati tramite bulloni, e da altri due set di componenti progettati *ad hoc*: delle staffe a 45° e dei "piedi", che conferiscono maggior stabilità alla struttura. Sul telaio sono alloggiati il sistema di alimentazione, gli attuatori e le loro unità di controllo.

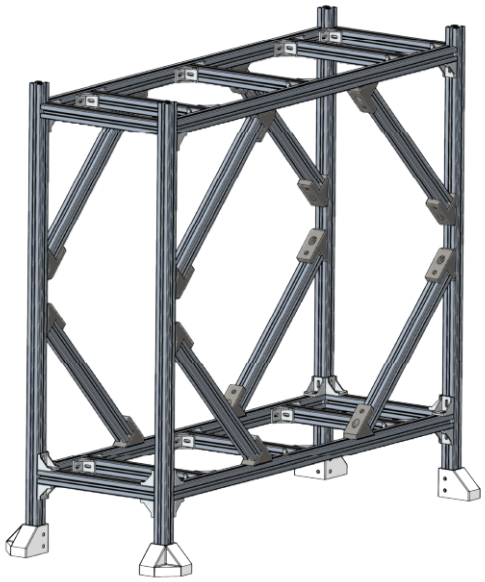


Figura 1.8: Modello CAD del telaio della macchina.

Per la marcatura di superfici bidimensionali adagiate su un supporto fisso, la piattaforma può essere dotata di moto esclusivamente piano. Se la quota della lente

viene correttamente tarata in fase di settaggio, non si osservano distorsioni indesiderate dello spot sul materiale, il che semplifica notevolmente il controllo del moto relativo laser-pezzo. Il robot si muove nel piano mediante l'avvolgimento e svolgimento di quattro cavi, rinviati alle pulegge situate in corrispondenza degli spigoli del telaio. La piattaforma è, quindi, sovravincolata e sovrattuata, in quanto sarebbero stati sufficienti due soli cavi.

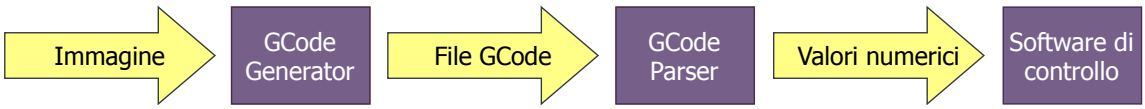


Figura 1.9: Diagramma di flusso del software della marcatrice.

Per quanto riguarda il software, si dà ora qualche informazione generale sulle parti che lo compongono. Il GCode generator è la sezione del codice che converte un'immagine digitale in un file di testo, contenente le istruzioni necessarie alla marcatura della stessa immagine su un pezzo mediante una macchina CNC. Tali istruzioni sono espresse secondo un linguaggio normato, il GCode (vedi sez. 4.1), per l'appunto. Ogni riga è costituita da codici alfanumerici che, secondo determinate regole di sintassi, forniscono informazioni di alto livello alla macchina, ad esempio: "accendi il laser", "imposta la potenza al 60%", "muovi il laser dalla posizione attuale al punto di coordinate (X,Y) alla velocità V" ecc. Istruzioni di questo tipo non possono essere direttamente inviate al controllo della macchina, che ha bisogno di comandi di più basso livello. In altri termini, è necessaria una lista di valori numerici che ciascun parametro controllabile (velocità e posizione della testina, potenza del laser ecc.) assume durante la lavorazione, istante per istante. A svolgere questo compito di interpretazione e traduzione è il GCode parser. Le istruzioni in uscita da quest'ultimo arrivano, infine, al firmware di controllo, il quale, risolvendo l'analisi cinematica e dinamica inversa e comunicando con i sensori installati sulla marcatrice, determina i valori delle variabili di giunto necessari a compiere le operazioni richieste.
Si fa notare come, di questi tre step (schematizzati in figura 1.9), i primi due siano compiuti offline, mentre il terzo avviene in real-time. Questo disaccoppiamento temporale è reso possibile dall'implementazione di un buffer circolare, nel quale vengono virtualmente stoccate le istruzioni decodificate, in attesa di essere prese in carico dal software di controllo della macchina.

Capitolo 2

La robotica a cavi

Lo scopo di una qualunque macchina è quello di generare un ben preciso movimento. Un robot non è altro che una macchina il cui moto può essere programmato liberamente. La caratteristica che lo differenzia da una macchina automatica è la sua capacità di adattarsi a situazioni differenti, con una (anche parziale) autonomia. Gli elementi funzionali di un robot sono, in genere, tre: il telaio, il membro terminale (*end-effector*) e gli organi di trasmissione intermedi. A seconda dell'architettura del manipolatore, che influisce sulla modalità con cui i suoi componenti si trasmettono forze e coppie, si distinguono diverse tipologie di robot. Per quanto complesse, esse sono solitamente riconducibili a due macro-categorie: robot seriali e robot paralleli. I compiti che un manipolatore può gestire sono stabiliti, oltre che dalla sua architettura, anche dai suoi gradi di libertà (GDL), che sanciscono il numero di variabili necessarie e sufficienti a descrivere la posa dell'end-effector.
Nel corso degli ultimi decenni, molti ricercatori hanno approfondito gli aspetti legati all'ottimizzazione delle performance dei robot, proponendo soluzioni volte a superare le criticità dei modelli precedenti. Questi studi hanno portato, a partire dagli anni ottanta, all'introduzione di sistemi che adoperano elementi flessibili (cavi, in particolare) per la movimentazione dell'organo terminale. Nel seguito si approfondiscono le ragioni alla base di questa scelta e si illustra il più semplice modello di controllo implementabile su questi manipolatori.

2.1 Dai robot seriali ai robot a cavi

I robot seriali, anche detti *bracci robotici* per via della loro architettura, sono costituiti da una serie di membri rigidi collegati in catena aperta mediante coppie cinematiche, tipicamente rotoidali o prismatiche, ciascuna attuata da un motore. La posa dell'end-effector viene determinata dalla combinazione delle variabili di giunto di ciascuno degli attuatori.
Il robot SCARA (*Selective Compliance Articulated Robot Arm*) è un classico esempio di sistema seriale: i suoi quattro GDL sono ottenuti tramite tre coppie rotoidali e una prismatica (fig. 2.1).
Questi manipolatori sono dotati di un'elevata mobilità, che si traduce in grandi aree e volumi di lavoro. D'altra parte, però, la loro capacità di carico è bassa (rispetto al peso proprio) e la precisione di posizionamento dell'organo terminale è limitata. Ciascuna coppia cinematica, infatti, introduce una tolleranza dovuta all'imprecisione nell'accoppiamento meccanico, che sfocia in una scarsa accuratezza e ripetibilità dei

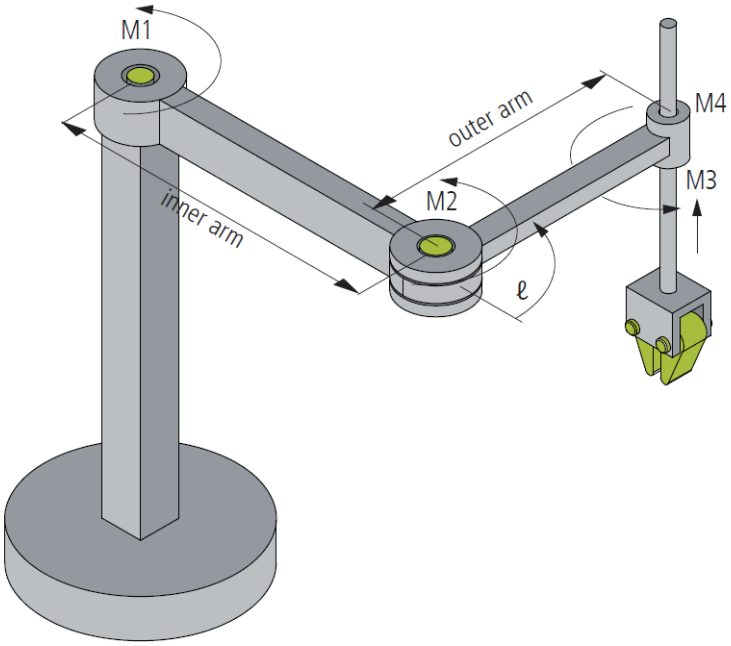


Figura 2.1: Il robot seriale SCARA.

movimenti del braccio. Ad aggravare ulteriormente la situazione, i membri, dovendosi sorreggere in successione, presentano una struttura rigida e massiccia, essendo sollecitati a flessione.

Aumentando il numero di bracci che collegano il telaio all'end-effector si ottiene un robot parallelo. In questo caso, diversamente dai sistemi seriali, sono presenti una o più catene cinematiche chiuse (fig. 2.2).

Si fa notare come i termini *seriale* e *parallelo* non si riferiscano alla geometria della macchina, bensì alla sua topologia, ovvero al modo in cui i membri si scambiano forze e coppie [19]. Nel caso dei manipolatori paralleli, infatti, le forzanti vengono ripartite su più "gambe", mentre un robot seriale costringe il suo unico braccio a sostenere tutti i carichi interni ed esterni. Le gambe dei robot paralleli, inoltre, sono soggette a soli sforzi di trazione e compressione. A fronte di ciò, questi manipolatori hanno strutture più snelle e dimensioni più contenute rispetto a quelli

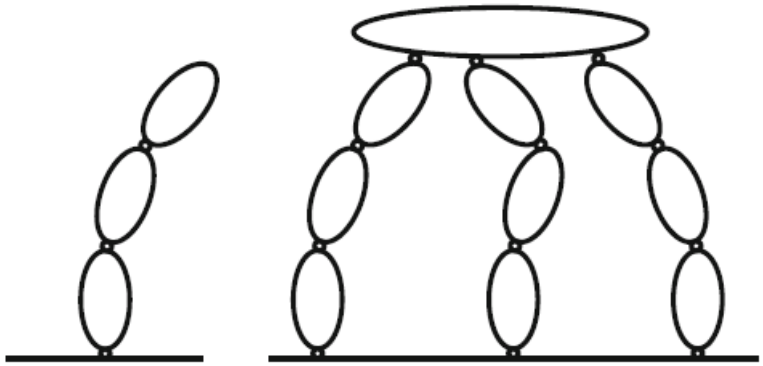


Figura 2.2: Schema delle due principali architetture di robot.

seriali, pur riuscendo a garantire una buona rigidezza complessiva della struttura. Dal momento che non tutti i giunti richiedono attuazione, i motori possono essere fissati direttamente al telaio. Per le ragioni appena citate, l'architettura parallela conferisce maggiori capacità di carico e permette dinamiche elevate rispetto ai sistemi seriali. Per via delle possibili interferenze fra gambe adiacenti, tuttavia, lo spazio di lavoro diminuisce.
Il più tipico esempio di robot parallelo è la piattaforma di Gough-Stewart (fig. 2.3), comunemente impiegata nei simulatori di volo.

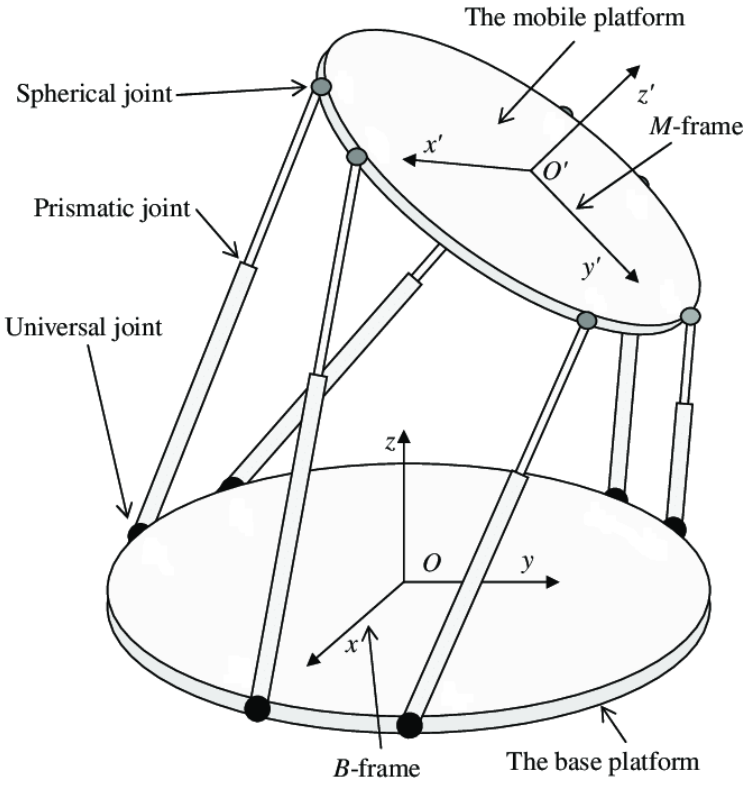


Figura 2.3: La piattaforma di Gough-Stewart.

Il passaggio dai robot paralleli ai robot a cavi è concettualmente semplice: è sufficiente, infatti, pensare di sostituire le gambe rigide di un manipolatore parallelo con degli elementi flessibili. In pratica, il funzionamento di queste macchine, note anche come CDPR (*Cable-Driven Parallel Robot*), è assimilabile a quello di una gru a molti GDL.
Sebbene i CDPR rientrino nella famiglia dei robot paralleli, essi presentano caratteristiche molto particolari, che consentono di racchiuderli in una categoria a sè stante.

2.2 Caratteristiche dei robot a cavi

I CDPR sono costituiti da una piattaforma mobile, un telaio e m cavi, connessi alla piattaforma in corrispondenza dei cosiddetti *punti distali* e al telaio in quelli *prossimali* (fig. 2.4). La lunghezza dei cavi viene regolata da un sistema di attuazione, detto anche *winch* (argano).

L'adozione di elementi flessibili comporta una serie di vantaggi rispetto alle architetture parallele tradizionali:

- Grandi spazi di lavoro
- Semplicità costruttiva
- Trasportabilità
- Versatilità
- Elevate accelerazioni
- Efficienza energetica

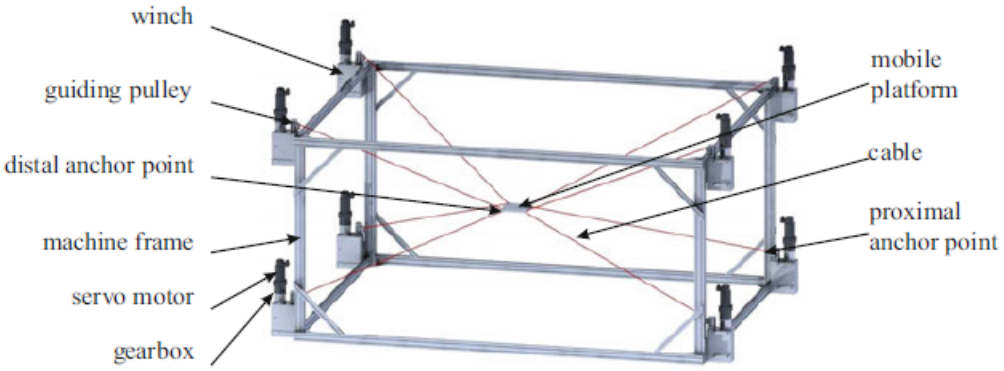


Figura 2.4: Rappresentazione schematica di un robot a cavi.

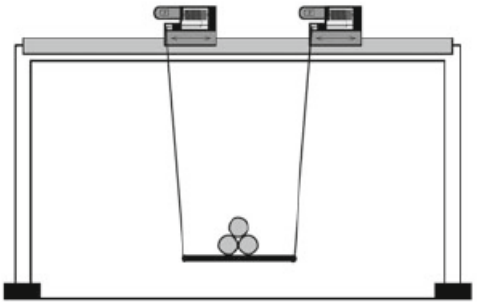
Per queste macchine, le corse degli attuatori sono molto grandi, in quanto dipendono unicamente dalla variazione di lunghezza dei cavi, resa possibile dall'avvolgimento e svolgimento di una bobina. Di conseguenza, la sola limitazione all'estensione dello spazio di lavoro risiede nella dimensione del tamburo, che pone un limite superiore alla quantità di cavo che esso può alloggiare.
La semplicità costruttiva rende questi sistemi particolarmente economici rispetto ai robot paralleli più classici e consente di trasportarli e adattarli facilmente ad aree di lavoro di dimensione pressoché arbitraria.
I cavi, data l'elevata resistenza a trazione, hanno sezione (e quindi massa) ridotta, motivo per cui i CDPR possono sopportare carichi elevati a fronte del peso proprio. La significativa riduzione delle inerzie consente, inoltre, il raggiungimento di elevate velocità e accelerazioni della piattaforma mobile, con prestazioni dinamiche mediamente difficili da ottenere con un robot parallelo tradizionale.
D'altra parte, però, l'adozione dei cavi comporta complicazioni legate alla natura elastica degli stessi, di cui bisogna in molti casi tenere conto per implementare un modello di controllo efficace.

2.2.1 Classificazione

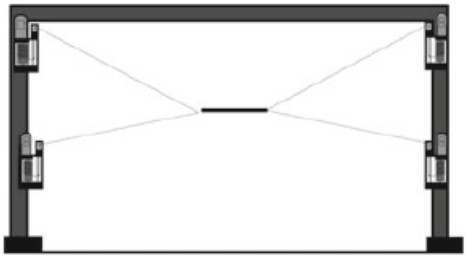
Secondo [19], i CDPR possono essere classificati in base a:

- Numero di cavi (m)

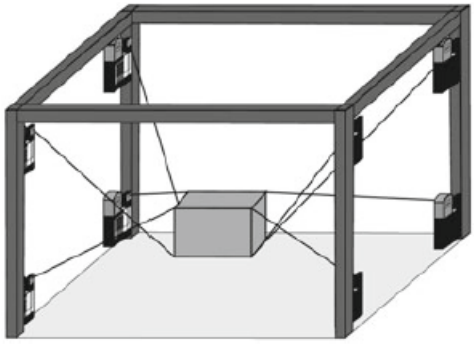
- Numero di GDL (n)
- Pattern di moto
- Sistema di attuazione
- Funzione



(a) IRPM piano ($m = 2$, $n = 3$)



(b) CRPM piano ($m = 4$, $n = 3$)



(c) RRPM spaziale ($m = 8$, $n = 6$)

Figura 2.5: Classificazione cinematica dei robot a cavi.

Una possibile classificazione cinematica è basata sul *grado di ridondanza* $r = m - n$ e suddivide i robot a cavi nelle seguenti categorie [16]:

- ***Incompletely Restrained Positioning Mechanism*** (IRPM, fig. 2.5a): appartengono a questo gruppo i sistemi per cui $m \leq n \leq 6$. Nel caso $m < n$, il robot non è completamente vincolato e non può, quindi, sostenere carichi esterni arbitrari. Non tutti i GDL sono controllabili, ma, a seconda delle forze e coppie esterne applicate, possono esistere configurazioni di equilibrio stabile o instabile.
  Se invece $m = n$, il robot è cinematicamente vincolato, ma l'equilibrio delle forze continua a dipendere dai carichi esterni (ad esempio la gravità).
- ***Completely Restrained Positioning Mechanism*** (CRPM, fig. 2.5b): il grado di ridondanza è pari a 1 ($m = n+1$). Il robot può essere completamente vincolato in determinate pose attraverso la sola azione dei cavi ed è in grado di reggere qualunque tipo di carico esterno, purché compatibile con la resistenza dei cavi.

- ***Reduntantly Restrained Positioning Mechanism*** (RRPM, fig. 2.5c): il grado di ridondanza è maggiore di 1 ($m > n + 1$). In realtà, nonostante il numero di cavi superi quello dei GDL, esiste una soluzione univoca al problema cinematico inverso. Ciò che resta indeterminato è, invece, la statica del robot.

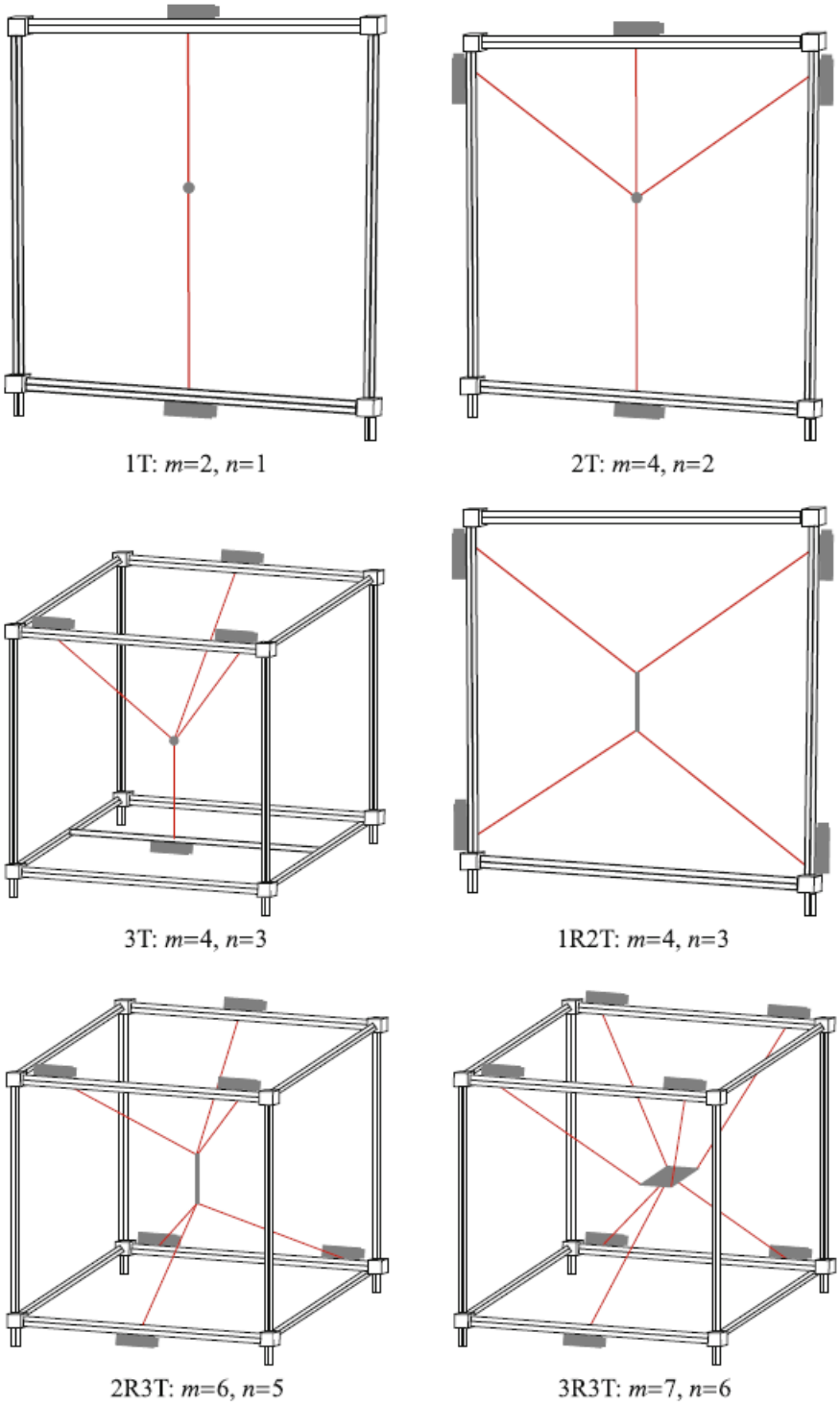


Figura 2.6: Classificazione dei robot a cavi in base al pattern di moto.

Un esempio di classificazione basata sul pattern di moto è, invece, quella introdotta da Verhoeven [21]. Per pattern di moto si intende la combinazione di

movimenti elementari, ossia rotazioni e traslazioni, che la piattaforma mobile è in grado di compiere rispetto a un sistema di riferimento cartesiano. Indicando con R i GDL rotazionali e con T quelli traslazionali, i CDPR sono suddivisibili nelle seguenti classi:

- **1T**: un solo GDL di traslazione (moto di un punto su una retta).
- **2T**: due GDL di traslazione (moto di un punto nel piano).
- **1R2T**: un GDL rotazionale e due traslazionali (moto di un corpo rigido nel piano).
- **2R3T**: due GDL rotazionali e tre traslazionali (moto di una trave nello spazio).
- **3R3T**: tre GDL rotazionali e tre traslazionali (moto di un corpo rigido nello spazio).

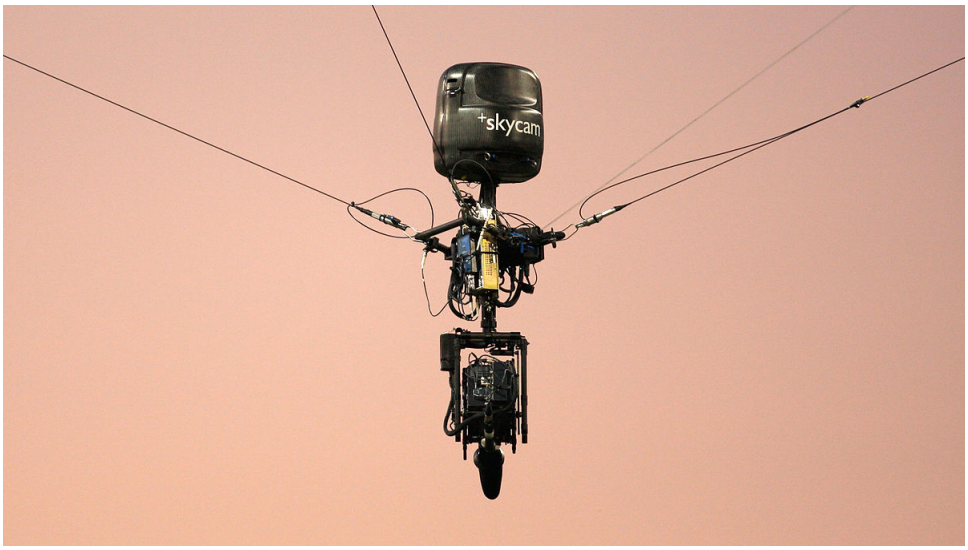
La figura 2.6 chiarisce le differenze di architettura legate ai possibili pattern di moto di un CDPR.

2.3 Applicazioni

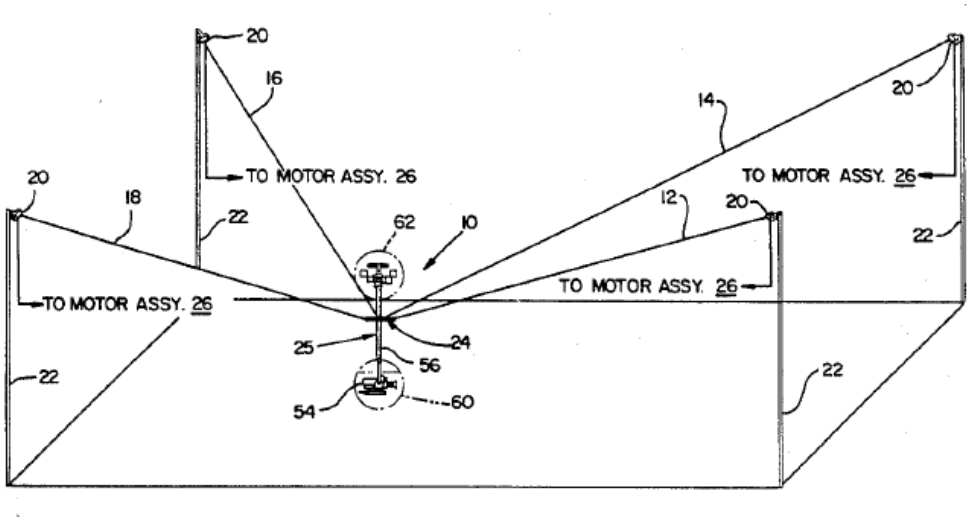
Per via delle loro particolari caratteristiche, i robot a cavi trovano impiego in un'ampia gamma di settori, quali ad esempio la logistica, la manifattura, la metrologia e l'intrattenimento.
Il primo CDPR a essere stato commercializzato risale al 1986 e consiste in un sistema di video-ripresa aerea noto come *Skycam* [5], illustrato in figura 2.7. Si tratta di un robot di tipo CRPM ($m = 4$, $n = 3$) ad architettura sospesa, principalmente impiegato per il trasporto di telecamere o apparecchi fotografici per eventi sportivi o d'intrattenimento. L'adozione di elementi flessibili permette alla macchina di muoversi coprendo le aree di stadi e arene senza intralciare la visibilità dell'evento. La buona stabilità della piattaforma, inoltre, consente di ottenere immagini di alta qualità. Quest'ultima caratteristica è resa possibile dall'adozione di un sistema a tre *gimbal* (componenti analoghi a dei giunti cardanici), che collega la telecamera alla piattaforma, assicurando riprese stabili anche in presenza di vibrazioni.

Più recentemente, nel 2016, KITE Robotics ha commercializzato un sistema (brevettato da [8]) per la pulizia delle facciate esterne di edifici (fig. 2.8). Questo CDPR è piano e ridondante ($m = 4$, $n = 2$), vincolato dal contatto diretto con la facciata dello stabile. L'installazione della macchina prevede la collocazione dei sistemi di svolgimento dei cavi sopra e sotto l'edificio. È presente, inoltre, una linea di approvvigionamento dell'acqua, collegata all'end-effector dall'alto (fig. 2.9a). Gli aspetti più innovativi di questo brevetto stanno nel sistema di bilanciamento della piattaforma e nel design della spazzola (fig. 2.9b), che consente il lavaggio di superfici curve e spigoli altrimenti complicati da raggiungere.

Oltre ai sistemi commerciali, ne esistono anche molti altri ideati da università e centri di ricerca. La Ohio University, ad esempio, ha sviluppato un prototipo di *rescue robot* [4], nato per compiere operazioni di salvataggio in situazioni di emergenza, specialmente in ambiente urbano. La macchina ha architettura sospesa e un numero di cavi riconfigurabile (fig. 2.10a), con la possibilità di ottenere un sistema sottovincolato o ridondante a seconda delle esigenze. La scelta di posizionare



(a) Foto del sistema installato



(b) Schema da brevetto

Figura 2.7: Skycam.

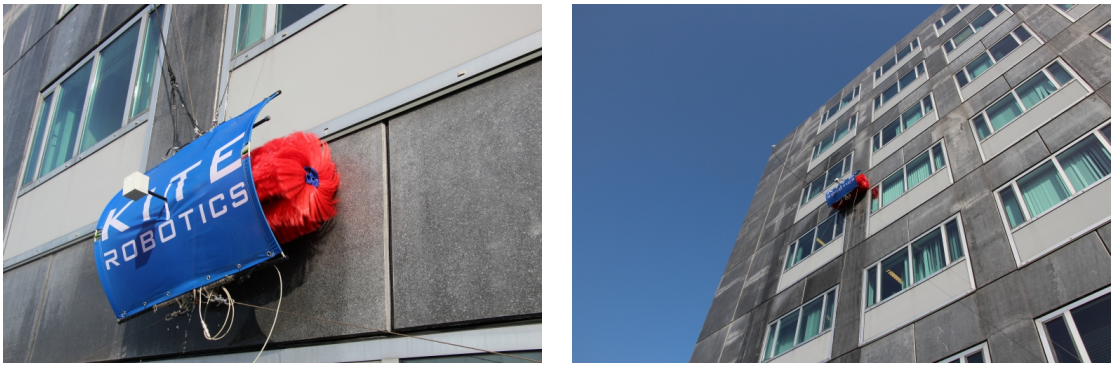
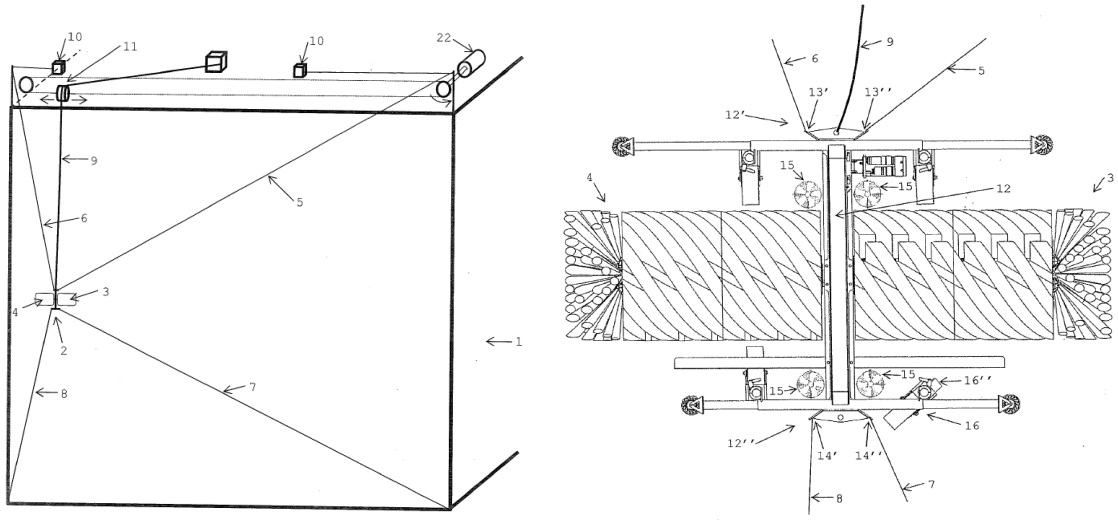


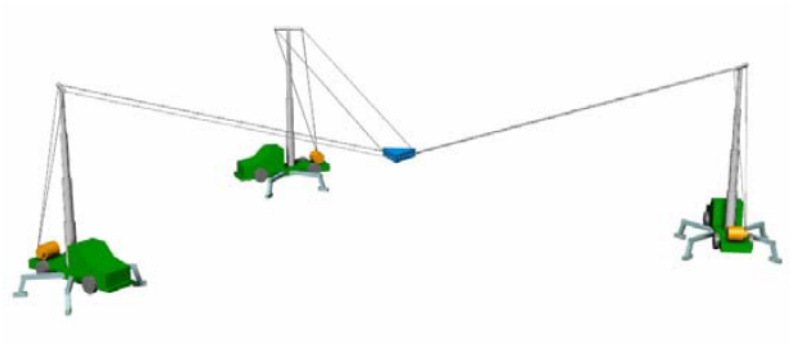
Figura 2.8: Sistema per la pulizia di edifici commercializzato da KITE Robotics.

il gruppo di attuazione e rinvio dei cavi su un veicolo (fig. 2.10b), poi, rende lo spazio di lavoro arbitrariamente flessibile, esigenza fondamentale per questo tipo di applicazioni.

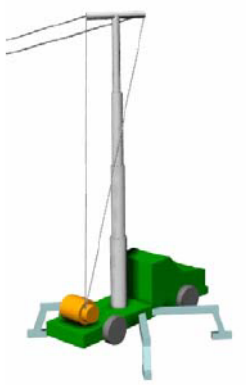


(a) Vista frontale dell'impianto (b) Design della spazzola

Figura 2.9: Schemi da brevetto [8].



(a) Schema del prototipo



(b) Veicolo di supporto

Figura 2.10: Rescue robot dell'Ohio University.

2.4 Modello standard per il controllo

Il cosiddetto *modello standard* per il controllo dei CDPR segue le ipotesi di un sistema multi-body costituito da membri rigidi: i cavi sono trattati come semplici

distanze fra ciascun punto distale e il punto prossimale omologo. Tali distanze sono variabili, ovvero sono modellate come coppie prismatiche, mentre alle estremità dei cavi sono applicati dei giunti sferici. Queste approssimazioni trascurano una serie di aspetti cruciali, di seguito elencati, che pongono inevitabilmente dei limiti all'applicabilità di questo modello [19]:

- **Vincoli unilaterali**: i cavi possono solo tirare, e non spingere, la piattaforma.
- **Elasticità dei cavi**: la lunghezza del cavo dipende dalla tensione a cui esso è soggetto.
- **Effetti termici**: la lunghezza del cavo dipende da variazioni di temperatura che possono verificarsi per cause esterne o per via dell'attrito sulle pulegge e sull'argano.
- **Avvolgimento irregolare sul tamburo**: i cavi, non essendo guidati in fase di avvolgimento, possono sovrapporsi, generando aumenti di tensione indesiderati e variazioni di lunghezza impreviste.
- **Complessità cinematiche di argano e piattaforma**: in alcuni casi la lunghezza di un cavo dipende anche dal suo orientamento rispetto all'argano e dalla posa della piattaforma mobile.
- **Peso dei cavi**: i cavi tendono a inflettersi per effetto della gravità, discostandosi dall'andamento rettilineo ideale.
- **Effetti di isteresi**: la lunghezza dei cavi può dipendere da stati di tensione precedenti a quello corrente.
- **Vibrazioni dei cavi**: le vibrazioni trasversali e longitudinali possono causare uno scostamento del cavo dalla linea che congiunge i punti di ancoraggio; è un fenomeno particolarmente significativo per robot di grandi dimensioni.
- **Reazioni elastiche del telaio e della piattaforma**: attuatori molto performanti possono generare accelerazioni talmente elevate da provocare reazioni elastiche della piattaforma e, talvolta, persino del telaio, introducendo ulteriore incertezza sulla lunghezza effettiva dei cavi.

Nonostante queste limitazioni, il modello standard, data la sua semplicità implementativa, costituisce ancora un buon compromesso per molte applicazioni industriali. Di seguito vengono presentate le soluzioni al problema cinematico, statico e dinamico.

2.4.1 Cinematica

2.4.1.1 Cinematica inversa

Il problema cinematico inverso consiste nel determinare, data la posa dell'end-effector, le variabili di giunto (oppure le lunghezze e l'orientamento dei cavi). Per i CDPR, la risoluzione è immediata e analoga a quella dei tradizionali robot paralleli. La geometria di un robot a cavi è definita dai punti di attacco dei cavi al telaio A_i e dai corrispondenti punti sulla piattaforma mobile B_i, con $i = 1,...,m$. Si definiscono

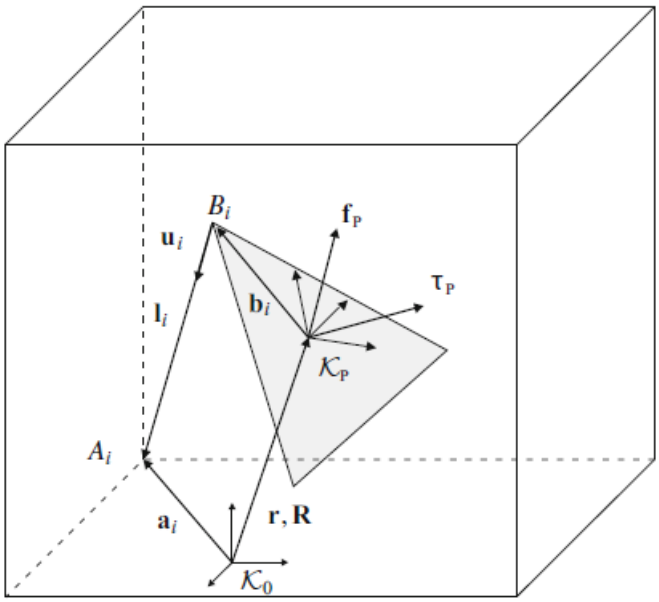


Figura 2.11: Rappresentazione del modello cinematico di un robot a cavi.

due sistemi di riferimento: K_0, solidale al telaio, e K_P, solidale alla piattaforma. Il vettore che collega l'origine di K_0 con A_i viene indicato con $\boldsymbol{a_i}$; esprimendolo nel sistema K_0 si ottiene ${}^{K_0}\boldsymbol{a_i}$. Analogamente, il vettore che collega l'origine di K_P con B_i viene indicato con $\boldsymbol{b_i}$; esprimendolo nel sistema K_P si ottiene ${}^{K_P}\boldsymbol{b_i}$. A questo punto, detto $\boldsymbol{r}$ il vettore che collega l'origine di K_0 con quella di K_P e facendo riferimento alla figura 2.11, si deduce che la lunghezza $\boldsymbol{l_i}$ del cavo i-esimo è data da:

$$\boldsymbol{l_i} = \boldsymbol{a_i} - \boldsymbol{r} - \boldsymbol{b_i} \qquad \text{per } i = 1,...,m. \tag{2.1}$$

Se $\boldsymbol{R_{0P}}$ è la matrice di rotazione che descrive l'orientamento della piattaforma rispetto al telaio fisso, allora il vettore $\boldsymbol{l_i}$ può essere espresso nelle coordinate del sistema K_0 come:

$${}^{K_0}\boldsymbol{l_i} = {}^{K_0}\boldsymbol{a_i} - {}^{K_0}\boldsymbol{r} - \boldsymbol{R_{0P}}\,{}^{K_0}\boldsymbol{b_i} \qquad \text{per } i = 1,...,m. \tag{2.2}$$

Dal rapporto fra il vettore e la sua norma si ottiene il versore

$$\boldsymbol{u_i} = \frac{\boldsymbol{l_i}}{\|\boldsymbol{l_i}\|} \qquad \text{per } i = 1,...,m, \tag{2.3}$$

che, per convenzione, è diretto dalla piattaforma al telaio, in modo tale che le forze di trazione applicate ai cavi assumano valore positivo. Una volta calcolate le lunghezze dei cavi, esse possono essere fornite in input ai motori per ottenere la posa desiderata. Come è già stato discusso, questo modello di calcolo trascura gli effetti delle pulegge di guida e degli elementi di fissaggio sulla piattaforma, ovvero, in altri termini, non considera la dipendenza dei vettori $\boldsymbol{a_i}$ e $\boldsymbol{b_i}$ dalla posa dell'end-effector. Un modello che valuta anche questi fenomeni è stato proposto da [18].

2.4.1.2 Cinematica diretta

Il problema diretto ha l'obiettivo di determinare la posa della piattaforma per data lunghezza dei cavi, o, equivalentemente, per date variabili di giunto. La posa $\boldsymbol{x}$ è

definita dalle grandezze contenute nel vettore posizione $\boldsymbol{r}$ e nella matrice $\boldsymbol{R_{0P}}$:

$$\boldsymbol{x} = [r_{x0}, r_{y0}, r_{z0}, \theta_x, \theta_y, \theta_z]^T \tag{2.4}$$

con θ_x, θ_y e θ_z angoli di Eulero nel sistema K_0.
In generale questo problema non può essere risolto in forma chiusa, fatta eccezione per rari casi. Di solito si procede per via numerica: se $n = m$ si può applicare il metodo di Newton-Raphson (come fatto da [10]); se, invece, $n < m$, si ottiene un sistema di m equazioni in n incognite ed è necessario implementare un processo di ottimizzazione nello spazio dei giunti o in quello operazionale.

2.4.2 Statica

Lo studio della statica di un CDPR è fondamentale per la definizione del suo spazio di lavoro, il quale è, di fatto, l'insieme delle pose che garantiscono, al netto dei carichi esterni, l'equilibrio della piattaforma. Indicando con f_i il modulo della forza agente sul cavo i-esimo e, rispettivamente, con $\boldsymbol{f_P}$ e $\boldsymbol{\tau_P}$ i vettori delle forze e delle coppie applicate alla piattaforma, si può scrivere il sistema di equazioni

$$\begin{bmatrix} \boldsymbol{u_1} & \dots & \boldsymbol{u_m} \\ \boldsymbol{b_1} \times \boldsymbol{u_1} & \dots & \boldsymbol{b_m} \times \boldsymbol{u_m} \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix} = \begin{bmatrix} \boldsymbol{f_P} \\ \boldsymbol{\tau_P} \end{bmatrix}, \tag{2.5}$$

ovvero, in forma compatta

$$\boldsymbol{A}^T \boldsymbol{f} = \boldsymbol{w}, \tag{2.6}$$

dove $\boldsymbol{f}$ è il vettore m-dimensionale delle forze agenti sui cavi e $\boldsymbol{w}$ il vettore n-dimensionale dei carichi esterni (forze e coppie) applicati sulla piattaforma. $\boldsymbol{A}^T$, che è funzione della posa $\boldsymbol{x}$, è la trasposta della matrice Jacobiana del sistema, che trasforma le forze dei cavi dallo spazio dei giunti a quello operazionale.
In base al grado di ridondanza, il sistema lineare 2.5 può essere sottovincolato ($r > 0$), completamente vincolato ($r = 0$) oppure sovravincolato ($r < 0$). In ogni caso, la sua risoluzione non è immediata e richiede tecniche algebriche idonee.

2.4.3 Dinamica

Per la risoluzione del problema dinamico secondo il modello semplificato, si scrivono le equazioni di Eulero applicate alla piattaforma. Supponendo che l'origine del sistema di riferimento K_P coincida col baricentro G della piattaforma, si ottiene

$$\boldsymbol{A}^T \boldsymbol{f} = \begin{bmatrix} m_P \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{J_G} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{r}} \\ \ddot{\boldsymbol{\Omega}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} \\ \dot{\boldsymbol{\Omega}} \times (\boldsymbol{J_G} \dot{\boldsymbol{\Omega}}) \end{bmatrix} - \begin{bmatrix} \boldsymbol{f_E} \\ \boldsymbol{\tau_E} \end{bmatrix}, \tag{2.7}$$

con m_P massa della piattaforma, $\boldsymbol{I}$ matrice identità, $\boldsymbol{J_G}$ matrice d'inerzia della piattaforma rispetto al suo centro di massa, $\dot{\boldsymbol{\Omega}}$ e $\ddot{\boldsymbol{\Omega}}$ velocità e accelerazione angolare della piattaforma, $\boldsymbol{f_E}$ e $\boldsymbol{\tau_E}$ forze e coppie esterne applicate in G. In caso di sovravincolamento ($r > 0$) il sistema diventa indeterminato, con infinite soluzioni per le tensioni dei cavi.

Capitolo 3

L'elaborazione digitale delle immagini

Un'immagine può essere definita come una funzione bidimensionale $f(x, y)$, la quale associa a ogni punto del piano un valore (o un vettore tridimensionale), che ne descrive l'intensità in scala di grigi (o il colore). Quando x, y e la norma di f sono tutte quantità finite e discrete, l'immagine è detta *digitale*. Spesso ci si riferisce a queste immagini coi termini *bitmap* o *immagine raster*, distinguendole dalle cosiddette *immagini vettoriali*, che sono invece definite tramite relazioni matematiche fra gli enti geometrici che le compongono e sono tipiche, ad esempio, dei sistemi CAD. Nel seguito della trattazione si farà riferimento solo alle bitmap, poiché il software sviluppato in questa attività di tesi non è in grado di gestire immagini vettoriali.
Il *digital image processing* è la disciplina che si occupa dello studio delle immagini digitali. In particolare, i processi che essa tratta sono di tre tipi [11]:

- ***Low-level***: comprendono tutte le operazioni di *pre-processing*, come ad esempio la regolazione del contrasto e il filtraggio del rumore. In ogni caso, sia l'input che l'output sono immagini.
- ***Mid-level***: comprendono operazioni di partizione dell'immagine in regioni o oggetti (*segmentazione*), conversione degli stessi in una forma idonea a essere processata e riconoscimento di oggetti. Sono processi caratterizzati dal fatto che, mentre l'input è sempre una bitmap, l'output è, in genere, un attributo estratto dall'immagine stessa (ad esempio un contorno o un oggetto).
- ***High-level***: sono tutte quelle procedure che determinano il "senso" di un'immagine, a partire dall'analisi delle parti che la compongono. Sono operazioni che, di fatto, svolgono la funzione cognitiva della visione.

In questo capitolo, dopo una breve trattazione sulla rappresentazione delle immagini, viene offerta una descrizione di alcune operazioni, prevalentemente di basso livello, necessarie alla comprensione degli algoritmi presentati nel capitolo 4.

3.1 Le immagini digitali

Un'immagine digitale è costituita da punti (*pixel*) a distanza regolare, a ciascuno dei quali è associato un valore. Di conseguenza, essa può essere rappresentata matematicamente tramite una matrice $\boldsymbol{I}$ di dimensione $m \times n$, che, per ogni pixel,

assume un valore $I(i, j)$, che ne descrive il colore, dove $i = 1, ..., m$ e $j = 1, ..., n$ (fig. 3.1). I valori di m e n dipendono dalla risoluzione spaziale del sensore che cattura e campiona l'immagine. $I(i, j)$, invece, varia su una scala che va da un valore minimo, che rappresenta la tonalità più scura (nero), a uno massimo, che rappresenta quella più chiara (bianco). L'ampiezza di questa scala dipende dal numero di *bit* destinati alla rappresentazione binaria del valore di grigio di ciascun pixel. Il sistema più comune è quello a 8 bit, che consente di ottenere $2^8 = 256$ tonalità differenti. Per convenzione, la scala varia da 0 (nero) a 255 (bianco).

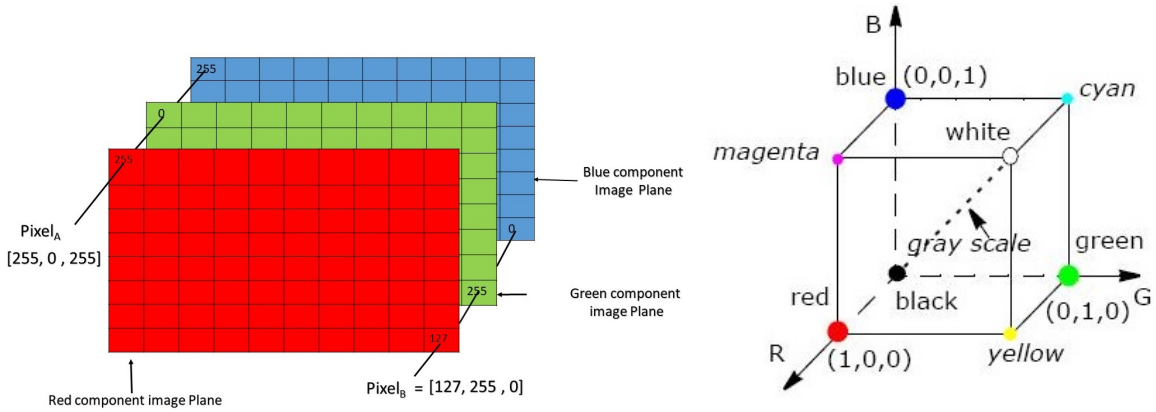
Figura 3.1: Rappresentazione matriciale di un'immagine raster.

Un modello di questo tipo è in grado di descrivere solo immagini in scala di grigi (o *grayscale*), poiché i colori sono dati da una combinazione di più tonalità (e quindi valori) e non da un singolo numero. Per le immagini a colori, quindi, è necessario associare a ogni pixel una terna di numeri, ciascuno riferito all'intensità (a 8 bit) di un particolare colore di riferimento in quel punto. Il modello RGB (rosso, verde, blu) è il più diffuso e può essere interpretato, dal punto di vista matematico, in due modi equivalenti:

- l'immagine è rappresentata da una matrice e a ciascun pixel è associato un vettore tridimensionale RGB;
- l'immagine è modellata come una matrice tridimensionale RGB, costituita da tre sotto-matrici, ciascuna riferita a un singolo colore o *canale*.

La seconda modalità (fig. 3.2a) è quella implementata in MATLAB e, perciò, quella di maggior interesse nell'ambito di questa tesi.
Lo spazio dei colori può essere rappresentato mediante un cubo di spigoli R, G e B unitari, coincidenti con gli assi di un sistema cartesiano (fig. 3.2b). Ogni punto interno al cubo è ottenibile tramite una combinazione dei valori di ciascun canale; la retta che collega l'origine con il vertice di coordinate $(1, 1, 1)$ coincide con la scala dei grigi.

Le immagini *binarie* (ovvero in bianco e nero) costituiscono una terza tipologia di bitmap. La loro rappresentazione è ottenuta tramite una matrice i cui soli valori



(a) Matrice tridimensionale (b) Rappresentazione cartesiana

Figura 3.2: Modello RGB.

(a) A colori (b) In scala di grigi

(c) Binaria

Figura 3.3: La stessa immagine nelle tre diverse versioni.

ammissibili sono 0 (nero) e 1 (bianco), da cui la loro denominazione. In figura 3.3 si mostra la stessa immagine nelle sue tre versioni: a colori, in scala di grigi e binaria.

Esistono semplici algoritmi che permettono di convertire un'immagine RGB in una grayscale o binaria, oppure una grayscale in una binaria.

La conversione da RGB a scala di grigi si basa, generalmente, su una combinazione dei valori di ciascun canale in un determinato punto della matrice, a cui vengono attribuiti pesi diversi a seconda dell'algoritmo. I metodi implementati nel software sviluppato sono elencati di seguito [1]:

- **Media aritmetica**

$$I_{grayscale}(i,j) = \frac{R(i,j) + G(i,j) + B(i,j)}{3} \tag{3.1}$$

- **Media pesata**

$$I_{grayscale}(i,j) = \alpha R(i,j) + \beta G(i,j) + \gamma B(i,j) \tag{3.2}$$

$$\text{con} \qquad \alpha = 0.2989 \qquad \beta = 0.5870 \qquad \gamma = 0.1140$$

- **Desaturazione**

$$I_{grayscale}(i,j) = \frac{MAX(i,j) + MIN(i,j)}{2} \tag{3.3}$$

$$\text{con} \qquad \begin{aligned} MAX(i,j) &= max(R(i,j), G(i,j), B(i,j)) \\ MIN(i,j) &= min(R(i,j), G(i,j), B(i,j)) \end{aligned}$$

- **Scomposizione secondo il massimo**

$$I_{grayscale}(i,j) = MAX(i,j) \tag{3.4}$$

- **Scomposizione secondo il minimo**

$$I_{grayscale}(i,j) = MIN(i,j) \tag{3.5}$$

- **Canale singolo** (ad esempio R)

$$I_{grayscale}(i,j) = R(i,j) \tag{3.6}$$

Le differenze fra questi algoritmi sono illustrate in figura 3.4, che fa ancora riferimento all'immagine di figura 3.3a.
La formula più comunemente utilizzata è la 3.2, i cui pesi α, β e γ sono stati stabiliti dall'NTSC (*National Television System Committee*) per compensare le differenze nella percezione dei colori da parte dell'occhio umano, che è più sensibile alla luce rossa e verde rispetto a quella blu.
Si fa notare come, trattandosi di operazioni fra numeri interi a 8 bit, i risultati rappresentabili siano sempre compresi fra 0 e 255. Se, ad esempio, il numeratore a secondo membro della 3.3 dovesse dare un numero maggiore di 255, la sua rappresentazione a 8 bit sarebbe comunque 255.
La conversione da RGB a grayscale è un processo irreversibile: la perdita di informazione non consente, cioè, di riottenere i colori originali a partire dal solo valore di luminosità.

La trasformazione di un'immagine a colori in una binaria avviene, solitamente, a valle di una conversione in scala di grigi. Le bitmap in scala di grigi, invece, possono essere direttamente convertite in bianco e nero tramite un'operazione detta *thresholding*, che consiste nell'assegnare il valore 1 (bianco) ai soli pixel con intensità superiore a una certa soglia e 0 (nero) agli altri. Esistono due differenti modalità di impostazione della soglia (fig. 3.5):

(a) Media aritmetica

(b) Media pesata

(c) Desaturazione

(d) Scomposizione (max)

(e) Scomposizione (min)

(f) Canale singolo R

(g) Canale singolo G

(h) Canale singolo B

Figura 3.4: Algoritmi di conversione in scala di grigi; l'immagine di partenza è quella in fig. 3.3a.

(a) Global (b) Adaptive

Figura 3.5: Conversione in bianco e nero secondo due diversi algoritmi di thresholding; l'immagine di partenza è quella in fig. 3.3b.

- ***Global*** (fig. 3.5a): la soglia è la stessa per tutti i pixel.

- ***Adaptive*** (fig. 3.5b): per ciascun pixel, la soglia varia a seconda dei valori di intensità degli altri pixel che lo circondano.

Ulteriori applicazioni del thresholding sono presentate nella sezione 3.2.

I principali formati per immagini raster sono riassunti nella tabella di figura 3.6. La scelta del formato più idoneo dipende dalla particolare applicazione, che detta esigenze diverse in termini di precisione e compressione dell'informazione. Quest'ultima si rende necessaria per una rapida trasmissione delle immagini, per la quale occorre rimuovere informazioni visive ridondanti dalla bitmap, il cui aspetto generale, tuttavia, viene preservato. Si distingue la compressione *lossy*, che rende impossibile la ricostruzione dei dettagli eliminati dall'immagine, da quella *lossless*, che, al contrario, consente un totale recupero dell'informazione.

Acronimo	**Nome**	**Proprietà**
GIF	Graphics interchange format	Limitato a 8 bit Compressione lossless
JPG/JPEG	Joint Photographic (Experts) Group	Compressione lossy Esistono varianti lossless
BMP	Bit map picture	Compressione lossless limitata Esistono varianti lossy
PNG	Portable network graphics	Nuovo formato lossless Rimpiazza formato GIF
TIF/TIFF	Tagged image (file) format	Formato dettagliato e flessibile Esistono varianti lossy/lossless

Figura 3.6: Caratteristiche dei principali formati digitali per immagini raster [20].

3.2 Operazioni sulle immagini

3.2.1 Operatori punto

Il più semplice tipo di operatore applicabile a una bitmap è una trasformazione punto-punto, che associa al valore di ciascun pixel dell'immagine in input l'intensità del corrispondente pixel dell'immagine in output. Queste procedure sono caratterizzate dal fatto che il valore associato a un pixel dell'output non dipende dal valore dei pixel a esso adiacenti nell'immagine di partenza.
In questa categoria ricadono le operazioni aritmetiche di base, che possono essere impiegate per:

- **Regolazione del contrasto** (fig. 3.7): sommando o sottraendo un valore costante a tutti i punti della bitmap è possibile aumentare o diminuire, rispettivamente, la luminosità dell'immagine. Un effetto simile può essere ottenuto anche tramite la moltiplicazione e la divisione per una costante.

- ***Blending*** (fig. 3.8): sommando due immagini è possibile generare una bitmap composita; la luminosità di ciascuna delle matrici di partenza può essere regolata con un fattore di moltiplicazione.

Sottraendo due immagini fra loro, si ottiene una nuova bitmap che mostra le differenze fra le due (fig. 3.9); in alternativa, anche la divisione fra immagini (pixel per pixel) è un operatore valido, seppur meno efficiente dal punto di vista computazionale [20].

(a) $C < 0$ (b) $C > 0$

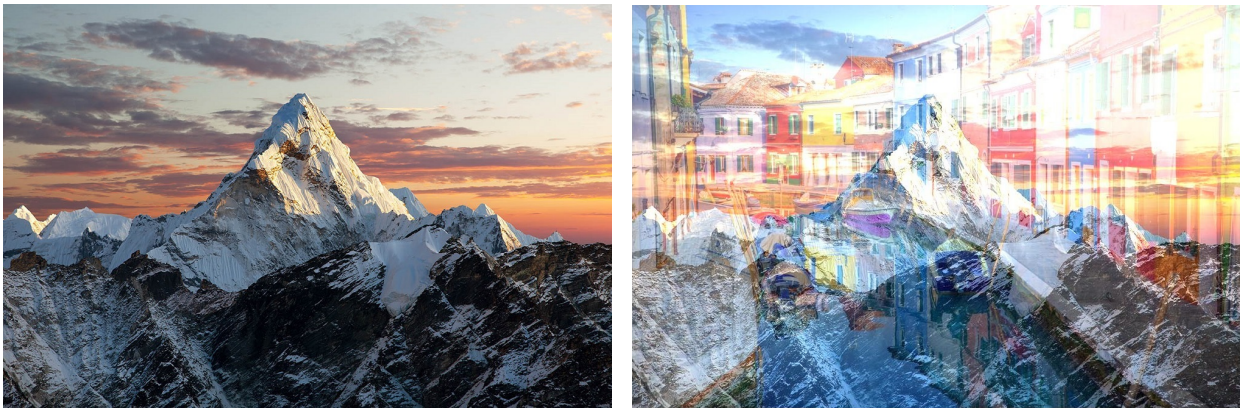
Figura 3.7: Regolazione del contrasto tramite l'aggiunta di un valore costante C a tutta la bitmap; l'immagine di partenza è quella in fig. 3.3a.

Altre operazioni possibili sono quelle logiche:

- **NOT**: provoca un'inversione del colore (fig. 3.10). Se l'immagine è binaria, i pixel bianchi diventano neri e viceversa; se la bitmap è in scala di grigi o a colori, l'operazione sui pixel è data da

$$I_{out}(i,j) = MAX - I_{in}(i,j), \tag{3.7}$$

dove MAX indica il massimo valore di intensità dato dalla risoluzione in bit (ad esempio, $MAX = 255$ per immagini a 8 bit).



(a) Immagine di partenza (b) Blend con fig. 3.3a

Figura 3.8: Esempio di blending di immagini.

(a) Immagine di partenza (b) Differenza con fig. 3.3a

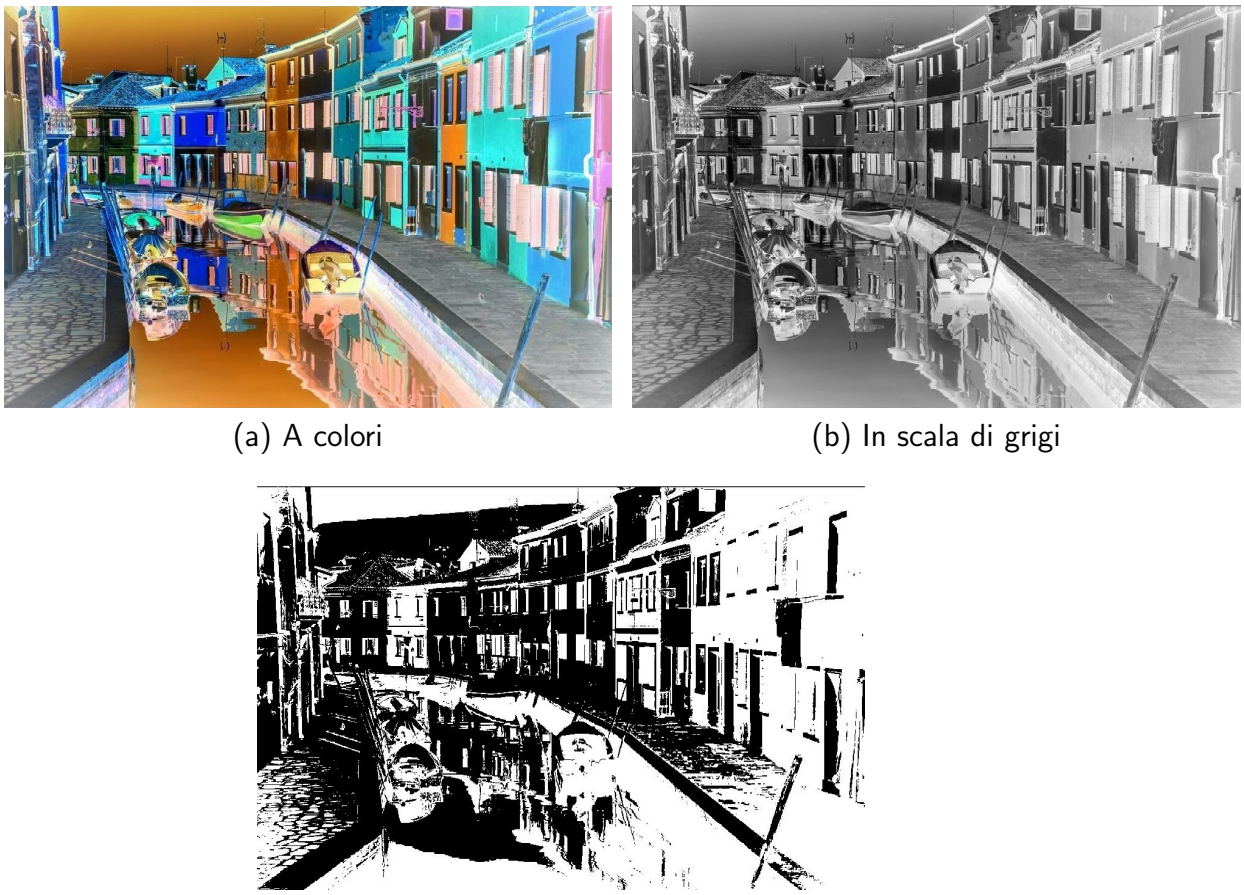
Figura 3.9: Differenza fra due immagini.

- **OR/XOR**: queste operazioni sono utilizzate su immagini binarie, soprattutto per individuare movimenti fra un frame e l'altro di un video.
- **AND**: molto impiegato per l'applicazione di maschere binarie a immagini; è possibile, ad esempio, selezionare tutti i pixel entro un certo range di valori per implementarvi dei filtri. Si tratta, di fatto, di un'altra applicazione del thresholding.

Analogamente, si può costruire e applicare a una bitmap un qualunque operatore logico ibrido, tipo NAND, NOR o NXOR.

3.2.2 Operatori spaziali

Gli operatori spaziali agiscono sul valore di un pixel considerando anche l'intensità dei pixel che si trovano in un suo intorno, detto *finestra* o *neighborhood*. Facendo riferimento alla figura 3.11, si parla di *connettività 4* quando i pixel inclusi nella finestra sono quelli N,E,S e W; quando, invece, la neighborhood comprende anche i pixel diagonali (NW, NE, SE e SW), si parla di *connettività 8*. Quest'ultima è la modalità impiegata di default dalla maggior parte degli operatori spaziali.



(a) A colori

(b) In scala di grigi

(c) Binaria

Figura 3.10: Inversione del colore (NOT logico) delle immagini in fig. 3.3.

La dimensione di un intorno può essere controllata variando il suo ordine $M \times N$, dal quale viene calcolato un offset k (di solito pari a $M/2$ o $N/2$). In questo modo, i pixel appartenenti alla finestra sono quelli di coordinate $(i \pm k, j \pm k)$.
La maggior parte delle operazioni di questo tipo viene svolta in real-time sui processori moderni, purché la dimensione della neighborhood non sia eccessivamente grande. Fra queste si approfondiscono diversi tipi di filtri e alcune trasformazioni geometriche.

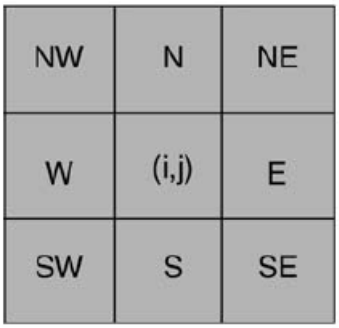


Figura 3.11: Esempio di neighborhood 3×3 centrata nel pixel di coordinate (i, j).

3.2.2.1 Filtri

Il filtraggio è un'operazione che, tipicamente, viene impiegata per la rimozione del rumore o per l'individuazione dei contorni di un'immagine. Nei filtri lineari, che sono i più comunemente utilizzati, il nuovo valore (filtrato) di un pixel è determinato da una combinazione lineare dei valori di intensità dei pixel nella sua neighborhood. Qualunque altro tipo di filtro è non lineare.
La specifica combinazione lineare è determinata dal cosiddetto *kernel di filtraggio*, che è una matrice avente la stessa dimensione della neighborhood definita per svolgere l'operazione. A ogni pixel del kernel è associato un peso, che viene attribuito al punto corrispondente della neighborhood nella combinazione lineare. Il processo di filtraggio avviene traslando il kernel su tutta la bitmap, centrandolo, di volta in volta, in ciascun punto (i, j) d'interesse dell'immagine originale e moltiplicando ogni pixel del kernel per il corrispettivo peso; i nuovi valori vengono calcolati tramite combinazione lineare e, infine, copiati nella stessa posizione, ma in una nuova immagine filtrata f (fig. 3.12).

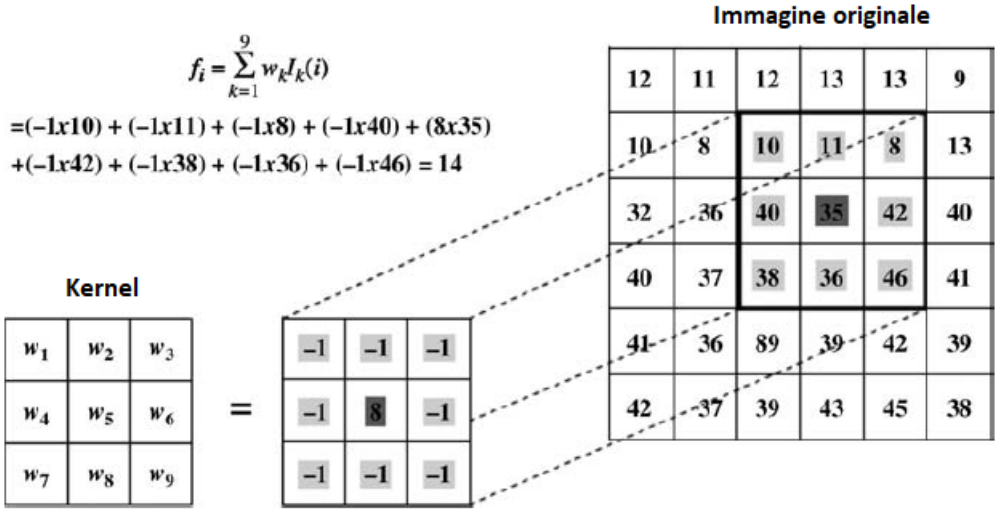


Figura 3.12: Procedura di filtraggio con un kernel 3×3.

In termini matematici, il processo di filtraggio è esprimibile con la seguente formula:

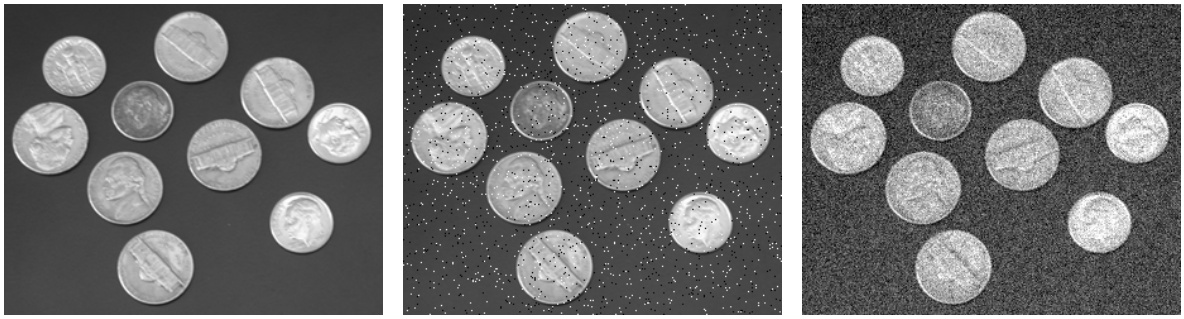
$$f_i = \sum_{k=1}^{N} w_k I_k(i), \tag{3.8}$$

dove $I_k(i)$ rappresenta il pixel k-esimo della neighborhood centrata nel pixel i-esimo dell'immagine, con k indice lineare che si muove lungo le righe (o le colonne) della finestra. Il termine w_k è il valore del k-esimo pixel del kernel, mentre f_i rappresenta il valore filtrato risultante da $I_k(i)$.
Alcuni problemi possono sorgere nel filtraggio in prossimità dei bordi della bitmap, dove parte del kernel fuoriesce dai confini dell'immagine. Di solito si sceglie uno dei seguenti approcci:

- la zona di confine non viene filtrata e resta, quindi, uguale a quella dell'immagine di partenza;

- vengono filtrati solo i pixel che rientrano nel kernel, modificando eventualmente i loro pesi;
- i pixel del kernel che rimangono "vuoti" vengono riempiti, specchiando i valori degli altri pixel rispetto al bordo della bitmap.

Le ultime due opzioni sono spesso preferite alla prima. Eventualmente è anche possibile tagliare l'immagine, in modo da escludere i pixel di confine che non sono stati soggetti al filtraggio.



(a) Immagine originale (b) Rumore "sale e pepe" (c) Rumore gaussiano

Figura 3.13: Due tipi di rumore applicati a un'immagine.

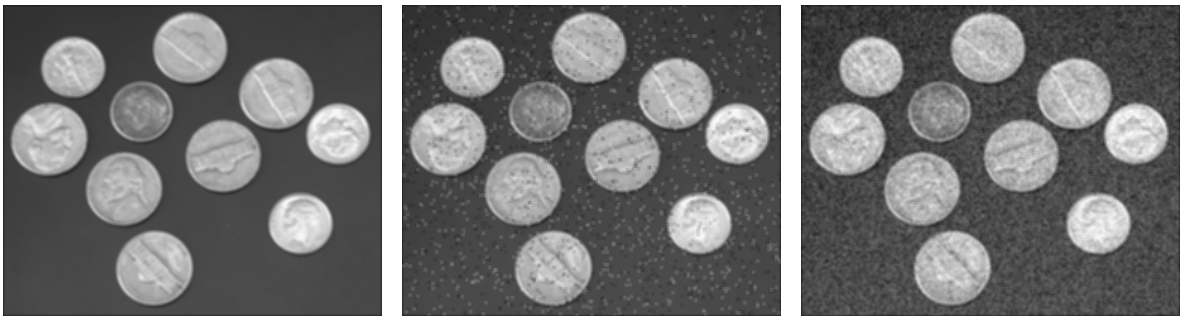
Si passa ora alla descrizione di alcuni dei filtri più utilizzati per la rimozione del rumore. In figura 3.13 vengono presentati due diversi tipi di disturbo:

- **"Sale e pepe"**: rumore che si manifesta con la presenza di pixel bianchi e neri sparsi per l'immagine.
- **Gaussiano**: rumore statistico che ha una densità di probabilità $p(g)$ con distribuzione normale (o gaussiana), data da

$$p(g) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(g-\mu)^2}{2\sigma^2}}, \tag{3.9}$$

  dove g indica il livello di grigio, σ la deviazione standard e μ il valor medio di grigio della distribuzione.

Lo strumento più elementare per affrontare questa problematica è il *filtro medio* (fig. 3.14), che assegna lo stesso peso $w_k = 1/(MN)$ a tutti i pixel del kernel. Come evidente dalla figura 3.14c, questo filtro è abbastanza efficace nel rimuovere il rumore gaussiano, seppure il suo impiego comporti una perdita di definizione dei contorni dell'immagine. Confrontando la bitmap filtrata con l'originale di figura 3.13c, si nota come il disturbo sia, tuttavia, ancora presente. Aumentando la dimensione del kernel si può ulteriormente ridurre il rumore, a discapito, però, di un ulteriore scadimento della definizione dell'immagine.
Dall'analisi della figura 3.14b si deduce che il filtro medio è inadeguato per la riduzione del rumore "sale e pepe", poiché quest'ultimo perturba in modo significativo il valor medio dei pixel della neighborhood. I punti soggetti al rumore sono, in pratica, esterni alla distribuzione statistica dei valori di grigio di ciascun intorno. È quindi necessario implementare un filtro che sia più robusto dal punto di vista statistico.



(a) Immagine originale (b) Rumore "sale e pepe" (c) Rumore gaussiano

Figura 3.14: Filtro medio (3×3) applicato a ciascuna delle immagini di figura 3.13.

Il *filtro mediano* (fig. 3.15) supera alcuni dei limiti appena discussi, anche se comporta un maggior costo dal punto di vista computazionale. In questo caso, i pesi w_k sono tutti pari al valore mediano della neighborhood. La mediana m di un insieme di numeri è quel valore per cui metà della distribuzione è inferiore e l'altra metà superiore a m. In altri termini, m è il punto medio della distribuzione ordinata dei valori considerati.
Il filtro mediano funziona molto bene per la rimozione del rumore "sale e pepe" (fig. 3.15b), consentendo, al tempo stesso, di mantenere una buona definizione dei contorni. Per quel che riguarda, invece, il rumore gaussiano, il comportamento è simile a quello del filtro medio: la riduzione del disturbo è parziale ed è accompagnata da un peggioramento della qualità dell'immagine (fig. 3.15c).

(a) Immagine originale (b) Rumore "sale e pepe" (c) Rumore gaussiano

Figura 3.15: Filtro mediano (3×3) applicato a ciascuna delle immagini di figura 3.13.

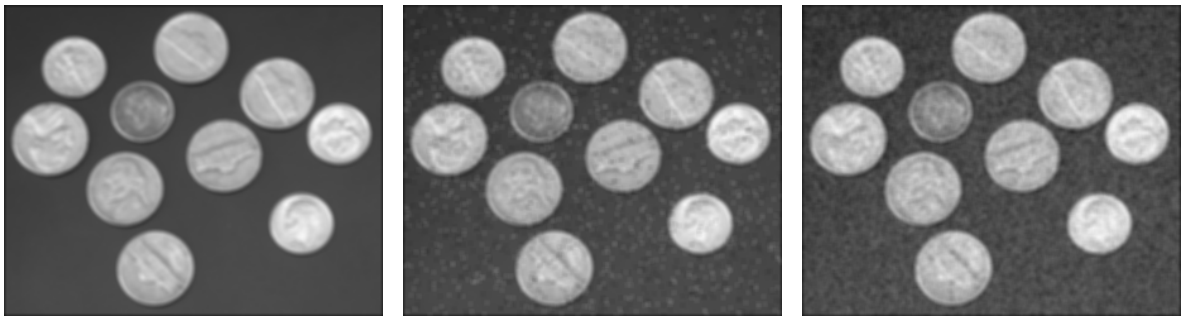
Il *filtro gaussiano* utilizza un kernel derivato dalla discretizzazione di una versione della funzione gaussiana nel piano avente simmetria polare. La sua definizione è data da

$$f(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{3.10}$$

L'approssimazione discreta di $f(x, y)$ si ottiene specificando due parametri:

- la dimensione del kernel;
- il valore della deviazione standard σ.

La scelta di queste grandezze deve permettere un buon compromesso fra un corretto campionamento della funzione e il tempo di calcolo richiesto per implementarlo.
Il filtro gaussiano tende a ridurre le disomogeneità dell'immagine a cui è applicato (fig. 3.16). A differenza del filtro medio, il suo effetto non viene controllato agendo sulla dimensione del kernel, bensì sul valore della deviazione standard. Esso tende, inoltre, a sfumare o eliminare i contorni più marcati, agendo come filtro passa-basso nel dominio della frequenza. Questa sua caratteristica si nota, ad esempio, confrontando la figura 3.16a con la 3.13a.



(a) Immagine originale (b) Rumore "sale e pepe" (c) Rumore gaussiano

Figura 3.16: Filtro gaussiano (5×5 con $\sigma = 2$) applicato a ciascuna delle immagini di figura 3.13.

3.2.2.2 Trasformazioni geometriche

Una trasformazione geometrica è un'operazione che modifica la relazione spaziale fra i pixel di un'immagine. Nel digital image processing, una procedura di questo tipo si suddivide sempre in due fasi:

1. **Trasformazione spaziale**: definisce il riarrangiamento dei pixel nella bitmap.
2. **Interpolazione**: assegna ai pixel trasformati un determinato livello di grigio (o di colore) secondo una certa logica.

Il concetto alla base di una trasformazione è quello di *forma*, che può essere descritta come un vettore contenente le coordinate di determinati punti dell'immagine:

$$\boldsymbol{x} = [x_1, y_1, x_2, y_2, \ldots, x_N, y_N] \tag{3.11}$$

Nel caso di un'immagine semplice, si può pensare che $\boldsymbol{x}$ sia costituito da tutti i punti che individuano i contorni degli oggetti raffigurati. Se, invece, la bitmap dovesse rappresentare una fotografia o, in generale, un'immagine complessa, una descrizione matematicamente accettabile della sua forma dovrebbe comprendere anche punti interni o esterni ai contorni individuati.
In qualunque caso, se si adotta un approccio di questo tipo è possibile descrivere, con una singola formula, le principali operazioni spaziali lineari, grazie all'adozione di un sistema di coordinate omogeneo. In primo luogo, la forma viene definita come

$$\boldsymbol{S} = \begin{bmatrix} x_1 & x_2 & \ldots & x_N \\ y_1 & y_2 & \ldots & y_N \\ 1 & 1 & \ldots & 1 \end{bmatrix}, \tag{3.12}$$

mentre la generica trasformazione lineare è descritta dalla matrice

$$\boldsymbol{T} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.13}$$

Assegnando ai termini α_{ij} i valori indicati in tabella 3.1, è possibile ottenere operazioni di traslazione, rotazione, *scaling* (o *resizing*) oppure *shearing* (fig. 3.17). Queste trasformazioni, dette *affini*, hanno la caratteristica di conservare il parallelismo fra gli enti geometrici.

Trasformazione	α_{11}	α_{12}	α_{13}	α_{21}	α_{22}	α_{23}
Traslazione di (x, y)	1	0	x	0	1	y
Rotazione di θ	$\cos\theta$	$\sin\theta$	0	$\sin\theta$	$\cos\theta$	0
Scaling di s	s	0	0	0	s	0
Shearing verticale di s	1	s	0	0	1	0
Shearing orizzontale di s	s	0	0	s	1	0

Tabella 3.1: Valori dei coefficienti α_{ij} necessari a ottenere operazioni di traslazione, rotazione, scaling o shearing in coordinate omogenee.

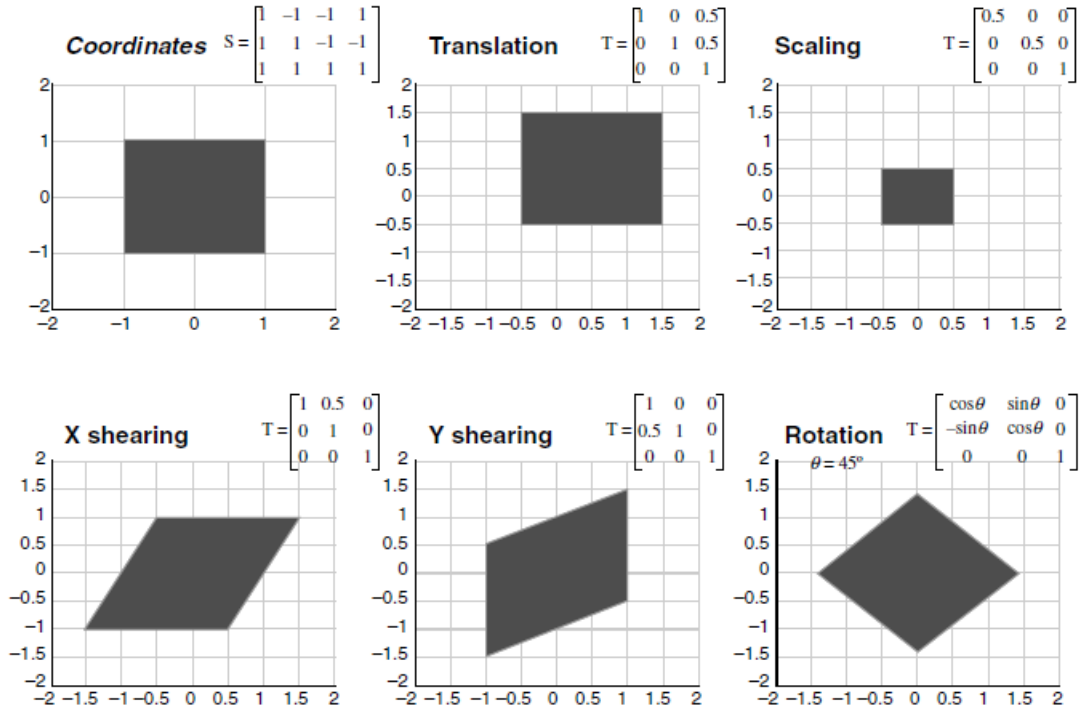


Figura 3.17: Trasformazioni affini ottenibili inserendo i valori della tabella 3.1 nella matrice $\boldsymbol{T}$ definita nella formula 3.13.

A questo punto, la nuova matrice di forma (a trasformazione avvenuta) è data da:

$$\hat{\boldsymbol{S}} = \boldsymbol{T}\boldsymbol{S}. \tag{3.14}$$

Tramite questo approccio, qualunque sequenza di trasformazioni lineari, descritte ciascuna da una matrice $(\boldsymbol{T_1}, \boldsymbol{T_2}, ..., \boldsymbol{T_N})$, può essere ricondotta alla formula 3.14 assemblando una matrice globale $\boldsymbol{T}$, tale che

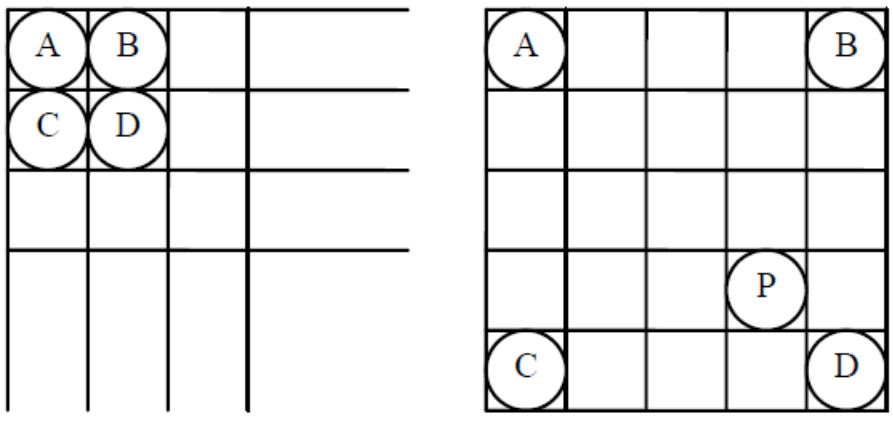
$$\boldsymbol{T} = \boldsymbol{T_1}\boldsymbol{T_2}...\boldsymbol{T_N}. \tag{3.15}$$

Con riferimento all'equazione 3.14, si fa notare come, a seconda dei valori interni alla matrice $\boldsymbol{T}$, le coordinate $\hat{x}_i$ e $\hat{y}_i$ contenute nelle prime due righe di $\hat{\boldsymbol{S}}$ possano assumere valori frazionari. Trattandosi, però, di indici di una matrice, essi devono necessariamente essere numeri naturali e vanno, pertanto, approssimati all'intero più vicino.
A questa approssimazione, relativa al posizionamento dei pixel, si accompagna quella, già accennata, che ha lo scopo di ridefinire il valore di grigio di tali punti. Il problema è esemplificato in figura 3.18, con riferimento a un'operazione di ingrandimento, ma il ragionamento si può estendere a qualunque altra trasformazione affine. Per definire l'intensità da assegnare al punto P (fig. 3.18b) esistono diversi metodi, di cui i più comuni sono [12]:

1. **Interpolazione *nearest neighbor* (NN)**: il valore di grigio di P coincide con quello del punto della matrice originale a esso più vicino (in questo caso D).

2. **Interpolazione bilineare**: l'intensità di P viene ricavata da una combinazione bilineare dei valori di A, B, C e D.

3. **Interpolazione bicubica**: l'intensità di P viene ricavata da una combinazione bicubica dei valori di una neighborhood 4×4 dell'immagine di partenza.

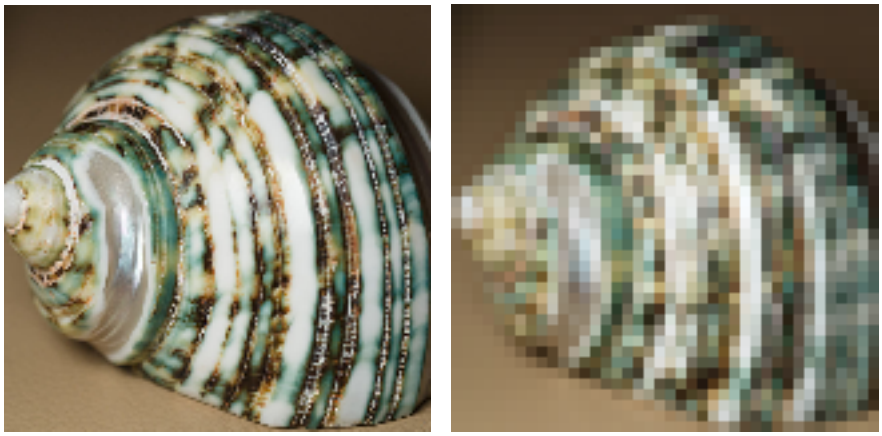


(a) Matrice originale (b) Matrice ingrandita

Figura 3.18: Schema esemplificativo di un'operazione di ingrandimento di una bitmap. I punti A, B, C e D dell'immagine di partenza non sono più fra loro adiacenti ed è necessario definire i valori dei pixel interni alla neighborhood da essi delimitata.

Una comparazione fra questi algoritmi, nel caso di un resizing, viene offerta in figura 3.19. Il metodo NN è il più semplice e rapido, ma può portare a distorsioni indesiderate dei contorni rettilinei in immagini ad alta risoluzione (fig. 3.19b).

L'interpolazione bilineare è più precisa e non genera discontinuità nette di colore; fungendo anche da filtro passa-basso, però, essa tende a sbiadire i contorni degli oggetti (fig. 3.19c), in modo simile a un filtro gaussiano. Dal punto di vista della qualità, solitamente i risultati migliori si ottengono con un'interpolazione bicubica (fig. 3.19d), che tuttavia comporta un maggior carico computazionale.



(a) Originale (b) Nearest neighbor

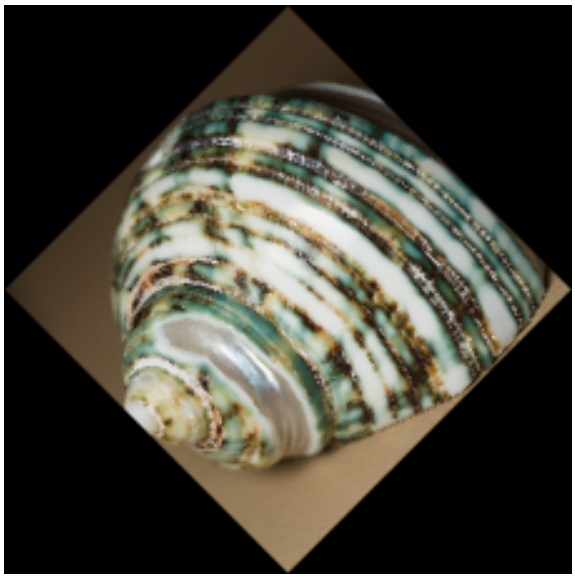
(c) Bilineare (d) Bicubica

Figura 3.19: Confronto tra i diversi algoritmi di interpolazione nel resizing di una bitmap. L'immagine (a), di dimensione 160×160, è stata prima ridotta a una risoluzione di 40×40 e, poi, nuovamente ingrandita alla dimensione originale secondo i tre diversi metodi.

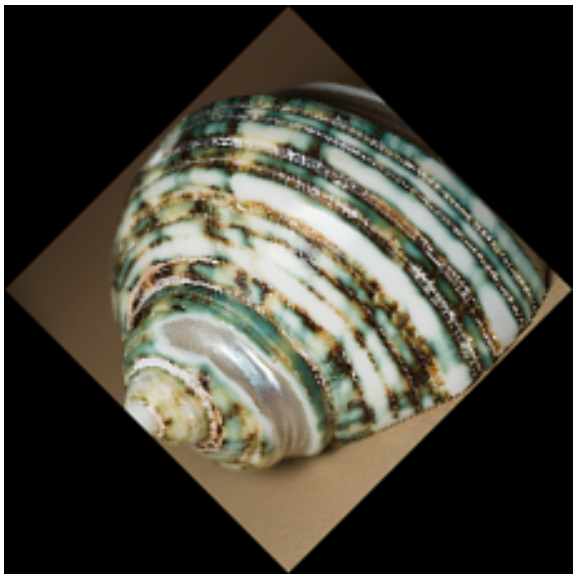
Considerazioni analoghe possono essere fatte nel caso della rotazione, di cui viene proposto un esempio in figura 3.20.

(a) Originale

(b) Nearest neighbor



(c) Bilineare



(d) Bicubica

Figura 3.20: Confronto tra i diversi algoritmi di interpolazione nella rotazione di 45° dell'immagine (a).

Capitolo 4

Il software per la pianificazione di traiettoria

Questo capitolo è dedicato alla descrizione del GCode Generator, ovvero del software per la pianificazione di traiettoria del Laser Engraver. Esso è corredato di un'interfaccia utente (*front-end*), oggetto del capitolo 5, che permette di selezionare ed, eventualmente, modificare l'immagine in input. Successivamente, una volta inseriti i dati tecnologici e geometrici necessari alla lavorazione (velocità di avanzamento, diametro dello spot, dimensione del disegno ecc.), è possibile generare un file di testo contenente le istruzioni in GCode necessarie alla marcatura dell'immagine, per poi, infine, elaborarne un'anteprima grafica.
Il GCode Generator, che costituisce il *back-end* del software, è posto a valle delle operazioni di pre-processing e di inserimento dei dati e si suddivide in tre parti:

1. **Image processing**: la bitmap viene manipolata in modo da potervi estrarre le informazioni necessarie alla generazione del GCode. In questa fase viene implementato un algoritmo di ottimizzazione della traiettoria, che permette di minimizzare il tempo ciclo della lavorazione.

2. **Generazione del GCode**: l'informazione elaborata nello step precedente viene convertita in un file di testo contenente righe di GCode.

3. **Parsing grafico del GCode**: il file GCode viene interpretato e riconvertito in informazioni che permettono di visualizzare graficamente il risultato, in modo che l'utente possa valutarne la qualità e, se lo ritiene necessario, tornare indietro per cambiare alcuni parametri di input.

L'intero pacchetto software è stato sviluppato in MATLAB, poiché si tratta di un linguaggio di alto livello che non richiede conoscenze avanzate di programmazione. Questo aspetto è particolarmente importante nell'ambito del progetto Laser Engraver, al quale continueranno a collaborare anche altri tesisti, che potranno, così, contribuire ai futuri sviluppi del codice senza la necessità di approfondire strumenti e linguaggi più complessi.
Nella prima parte del capitolo viene presentata la particolare implementazione di GCode scelta per questo progetto. Successivamente vengono descritti nel dettaglio, in forma di pseudocodice, tutti gli algoritmi di back-end. Nella sezione finale si mostrano alcuni risultati e si analizzano, infine, i pregi e i limiti del software.

4.1 Implementazione del GCode

Il GCode è un linguaggio alfanumerico utilizzato per assegnare istruzioni all'unità di controllo di una macchina CNC. Nonostante la normativa ISO 6983 [3] ne regolamenti la formattazione, esistono numerose varianti di questo linguaggio implementate dai costruttori.
Il file contenente tutte le informazioni necessarie alla lavorazione, detto *part program*, è costituito da righe (*blocchi*), ciascuna contenente un'istruzione composta da più comandi, eseguiti secondo un ordine stabilito dall'unità di controllo. Ogni comando elementare è individuato da una coppia di caratteri: una lettera (*indirizzo*) e un numero. Terminata l'analisi e l'esecuzione di un blocco, si passa al successivo, fino ad arrivare alla fine del part program.
Si distinguono comandi modali e non modali: i primi restano validi finché non vengono rimpiazzati o resettati da un comando alternativo, mentre i secondi hanno valenza limitata al blocco di cui fanno parte.
Per ulteriori dettagli sul GCode classico si rimanda a [6]. Ci si concentra ora, invece, sulla particolare implementazione scelta per questa attività, che fa uso solo di alcuni dei comandi previsti dalla norma ISO già citata. Essi si distinguono in diverse tipologie, sulla base dell'indirizzo usato:

- **M**: sono dette *funzioni ausiliarie*. Non gestiscono movimenti della macchina, ma, nel caso specifico, comandano l'accensione (M4) e lo spegnimento (M5) del laser.

- **G**: indica una *funzione preparatoria*, che gestisce, cioè, i movimenti della testa laser. Il numero che segue l'indirizzo G permette di distinguere vari possibili modalità di movimentazione (ad esempio G0, G1, G2 ecc.). Le funzioni preparatorie impiegate in questa sede sono:

  - G0: moto di appostamento rapido.
  - G1: moto di avanzamento a laser acceso.
  - G11: macro-istruzione per la scansione di una riga (viene approfondita nel seguito di questa sezione).
  - G28: ciclo di *homing*, comprendente le operazioni che servono a riportare gli assi in una posizione nota ai sensori all'inizio della lavorazione.

- **X, Y**: corrispondono ai nomi degli assi controllati. Ciascuno è seguito da un numero, che indica la coordinata a cui deve giungere lo spot del laser nella modalità indicata dal comando G precedente.

- **F**: preceduto da G0 o G1 nello stesso blocco, designa la velocità (in mm/s) associata a quel particolare comando.

- **S**: regola la potenza a 8 bit del laser, secondo una scala che va da 0 (potenza nulla) a 255 (potenza massima). Se il comando è preceduto da una funzione G, ad esempio:

```
G1 X10 Y20 S220
```

allora lo spostamento al punto indicato deve essere effettuato solo dopo aver regolato il laser alla potenza designata (220, in questo caso).

Tutti i comandi appena elencati sono modali, fatta eccezione per G11 e G28.

In base a quanto anticipato nella sezione 1.2, l'operazione che il Laser Engraver deve compiere è una marcatura a scansione. Ciò significa che la testa laser deve muoversi lungo delle linee rette (arbitrariamente inclinate, come si vedrà), che, percorse l'una dopo l'altra, permettono la scansione dell'intero disegno. Alla luce di ciò, si è pensato di creare un particolare comando (il già citato G11[1]) dedicato a questa azione, che costituisce, di fatto, l'operazione elementare alla base del processo tecnologico. Questo accorgimento permette di semplificare il part program ed evitare di sovraccaricare la memoria dell'unità di controllo con informazioni superflue.
G11 è, più precisamente, una macro-istruzione, ovvero un comando che ne comprende altri al suo interno, ma che, avendo una sintassi più compatta, consente di risparmiare blocchi di codice. Nel GCode da norma ISO esistono svariate macro dedicate a operazioni molto comuni, quali la foratura, la tornitura cilindrica e la fresatura di asole, solo per citarne alcune.
Se non si utilizzasse una macro-istruzione, l'operazione di scansione di una singola riga richiederebbe la seguente sintassi:

```
G0 X(primo punto) Y(primo punto) S0
G1 X(secondo punto) Y(secondo punto) S(0÷255)
X(terzo punto) Y(terzo punto) S(0÷255)
...
X(i-esimo punto) Y(i-esimo punto) S(0÷255)
...
X(ultimo punto) Y(ultimo punto) S(0÷255)
```

Il primo blocco è relativo al moto di appostamento in corrispondenza del primo punto della riga di scansione e avviene, quindi, a laser spento. Tutti gli altri blocchi, invece, indicano un valore di potenza che dipende dall'intensità di grigio del punto di coordinate (X,Y) indicate. Si fa notare come G1, essendo un comando modale, non abbia bisogno di essere ripetuto in ciascun blocco.
Una prima osservazione che si può fare è che, dovendo il laser percorrere traiettorie rettilinee, queste possono essere definite dalle sole coordinate dei punti di inizio e fine della riga, senza la necessità di specificare X e Y per ogni punto interno a essa. Ha quindi maggior senso adottare delle coordinate parametriche, comprese fra 0 (punto di inizio della riga) e 1 (punto finale), seguite, per ciascun blocco, dal comando S corrispondente. Un blocco iniziale, invece, identifica l'operazione di scansione G11 e riporta le coordinate cartesiane (X,Y) per i due punti estremi della riga. In pratica, la macro assume la forma seguente:

```
G11 X(primo punto) Y(primo punto) X(ultimo punto) Y(ultimo punto)
(coordinata parametrica secondo punto) S(0÷255)
(coordinata parametrica terzo punto) S(0÷255)
...
(coordinata parametrica i-esimo punto) S(0÷255)
...
```

[1]La scelta del numero 11 per la designazione della macro-istruzione è totalmente arbitraria e volta a evitare confusione, dal momento che, nel GCode originale, non esiste un comando G11.

```
1 S(0÷255)
```

Questa sintassi è, evidentemente, più snella e compatta di quella precedente, soprattutto nel caso in cui si lavori su grandi aree, che richiederebbero di specificare coordinate X e Y con molte cifre.

Ora che sono stati descritti tutti i comandi implementati per questa applicazione, si fa accenno alla struttura del part program, che è sempre suddivisibile in tre sezioni:

1. **Parte introduttiva**: definisce le velocità assegnate ai comandi G0 e G1, compie il ciclo di homing e comanda l'accensione del laser con potenza nulla. Un esempio è il seguente:

   ```
   G0 F2000
   G1 F1000
   G28 X450.0 Y450.0
   M4 S0
   ```

   dove G0 e G1, in realtà, non comandano un movimento, ma servono a riferire la velocità definita da F a ciascuna delle due funzioni preparatorie. Tale velocità resta valida per tutto il seguito del part program, poiché è stata designata tramite dei comandi modali.

2. **Parte esecutiva**: contiene tutte le macro G11, ciascuna riferita a una diversa riga del disegno. Ad esempio:

   ```
   G11 X470.0 Y450.0 X473.0 Y450.0
   0.33 S132
   0.67 S62
   1.00 S26
   G11 X473 Y450.5 X470.0 Y450.5
   0.33 S190
   0.67 S232
   1.00 S58
   ...
   G11 X516.0 Y525.4 X517.0 Y525.4
   1.00 S20
   ```

3. **Parte conclusiva**: comanda lo spegnimento del laser con la funzione M5. Si fa notare come il comando M5 sia, teoricamente, equivalente al blocco 'M4 S0'. Per chiarezza, tuttavia, M5 viene utilizzato sempre e solo alla fine del part program, mentre 'M4 S0' all'inizio.

4.2 Image processing

Gli algoritmi di image processing implementati nel back-end prendono in input un'immagine, dunque una matrice, e la trasformano in una *cell array* di righe di scansione, inclinate secondo una direzione scelta dall'utente (compresa fra $-89°$ e $90°$ rispetto al limite inferiore della bitmap). Per "cell array" si intende, in MATLAB, una struttura dati che ha per elementi dei vettori o delle matrici, anche di

dimensioni diverse fra loro.
La sezione di image processing è contenuta nello script *ImageProcessingFull* e si articola in due funzioni principali, eseguite una in sequenza all'altra: *scaleImage* e *imageToCellMm*. Quest'ultima, a sua volta, contiene numerose sotto-funzioni, le cui dipendenze sono riassunte nel diagramma di figura 4.1. In vetta si trova lo script di più alto livello (il già citato *ImageProcessingFull*), mentre alla base è collocata la funzione di più basso livello (*normalizeRow*). Quando una funzione f_1 è collegata con una linea a una funzione f_0 che si trova a un livello superiore, allora significa che f_1 è inclusa in f_0.
La funzione *scaleImage* si occupa di modificare la risoluzione dell'immagine, in modo coerente rispetto alla dimensione dello spot laser e alla risoluzione del sensore di posizione montato sul robot.
Si passa poi a *imageToCellMm*, nell'ambito della quale, in base alle scelte che l'utente compie nell'interfaccia, si prospetta uno dei seguenti scenari:

A. l'angolo di scansione viene scelto dall'utente ed è o 0° o 90°;

B. l'angolo di scansione viene scelto dall'utente ed è di modulo compreso fra 0° e 90°, estremi esclusi;

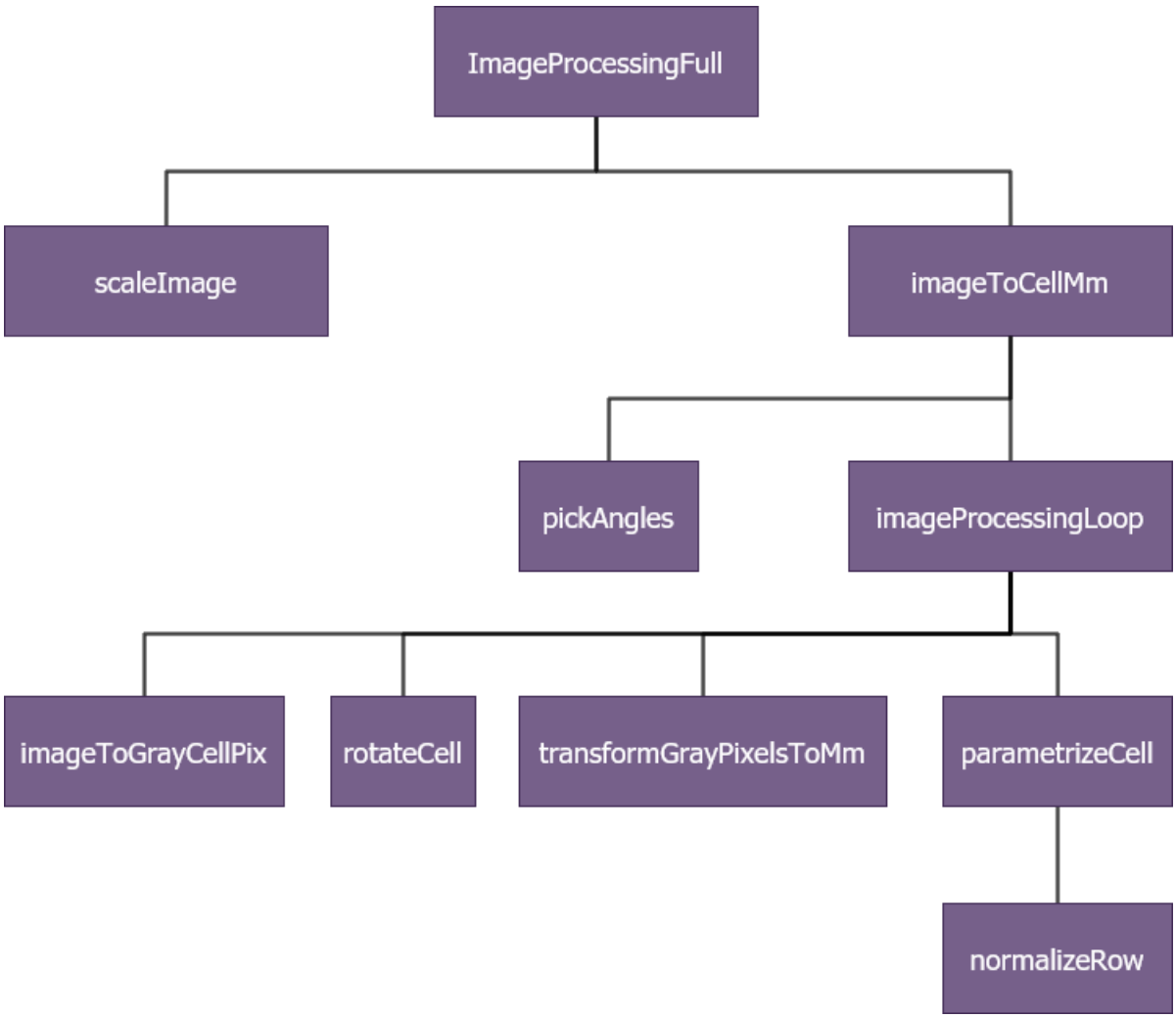


Figura 4.1: Diagramma delle dipendenze per le funzioni di image processing.

C. l'angolo di scansione non viene esplicitamente scelto dall'utente, ma viene ottimizzato da un algoritmo che, dopo aver determinato, secondo alcuni criteri geometrici, tutti gli angoli candidabili alla selezione, individua quello che consente di realizzare la marcatura nel minor tempo possibile.

A gestire questa casistica è la funzione *pickAngles*, che fornisce in output un angolo (o un vettore di angoli) per il quale svolgere l'operazione successiva di estrazione delle righe di scansione dalla bitmap, demandata a *imageProcessingLoop*. Lo scenario A è il più semplice, in quanto, per motivi che saranno chiari più avanti, la scansione orizzontale e quella verticale sono sempre attuabili, a prescindere dall'immagine in input e dalla sua risoluzione. Diversamente, il caso B richiede di valutare se l'angolo scelto dall'utente sia compatibile con i criteri geometrici implementati nell'algoritmo; in caso contrario, il sistema imposta come angolo di scansione quello più vicino al valore di input che soddisfa tali criteri. A differenza degli scenari A e B, nel caso C l'output di *pickAngles* non è un angolo, ma un vettore di angoli.
In qualunque dei tre casi, *imageProcessingLoop* prende in carico il vettore $\boldsymbol{\alpha}$ (eventualmente monodimensionale) degli angoli e, per ciascuna delle sue componenti α_i, procede alla seguente manipolazione:

1. Se $\alpha_i \neq 0$, la bitmap viene ruotata in senso orario di tale angolo.

2. *imageToGrayCellPix* estrae dalla bitmap (già ruotata dell'angolo α_i) le sue righe, inserendole in una nuova cell array.

3. Se $\alpha_i \neq 0$, ciascun elemento della cell array viene nuovamente ruotato (dalla funzione *rotateCell*) di un angolo α_i, ma stavolta in senso antiorario, in modo da riportare l'immagine all'inclinazione originale.

4. L'informazione contenuta nella cell array viene convertita da coordinate in pixel a coordinate cartesiane (in millimetri), tramite la funzione *transformGrayPixelsToMm*, che si occupa anche di riferire correttamente il disegno rispetto al centro dell'area di lavoro, secondo le indicazioni fornite dall'utente.

5. *parametrizeCell*, infine, effettua la parametrizzazione delle coordinate, secondo quanto previsto dalla macro G11; in questa fase viene ottimizzato, inoltre, il punto di inizio della marcatura.

Nello scenario C, l'ottimizzazione dell'angolo di scansione avviene al livello di *imageProcessingLoop*.

I dati di input a *ImageProcessingFull* sono elencati di seguito, con denominazione coerente a quella impiegata successivamente nello pseudocodice:

- ***img***: bitmap di input, già convertita in scala di grigi o in bianco e nero nell'interfaccia; si tratta, dunque, di una matrice $m \times n$.

- ***sensRes***: risoluzione del sensore di posizione (in mm) installato sul robot.

- ***Dbeam***: diametro dello spot. Si tratta, in realtà, di un'approssimazione della dimensione reale dello spot, visto che, per i laser a diodi, l'intersezione del fascio con la superficie del pezzo non è circolare.

- ***mode***: stringa che riporta la modalità di scansione. Può essere *'horizontal'*, *'vertical'*, *'diagonal'* oppure *'optimized'*.

- ***speed***: vettore riga contenente le velocità (in mm/s) associate ai comandi G0 (prima componente) e G1 (seconda componente).

- ***sizeI***: vettore riga contenente larghezza (prima componente) e altezza (seconda componente) desiderate per l'immagine marcata.

- ***sizeWP***: vettore riga contenente larghezza e altezza del pezzo in lavorazione.

- ***offset***: vettore riga contenente l'eventuale offset (orizzontale e verticale) fra il centro del disegno e quello del pezzo in lavorazione. Se l'offset è nullo, il centro della bitmap viene posizionato in corrispondenza del centro del pezzo.

- ***Slim***: vettore riga contenente i valori limite (minimo e massimo) di S per il GCode, ovvero i valori limite di potenza del laser, compresi fra 0 e 255.

- ***modeR***: stringa che riporta la modalità di resizing da implementare nell'algoritmo. Può essere *'bicubic'*, *'bilinear'* oppure *'nearest'*, in corrispondenza ai metodi di interpolazione descritti nella sezione 3.2.2.2.

- ***varargin***: è una variabile che può essere inserita o meno fra gli argomenti di *imageProcessingFull* e di varie altre funzioni. Viene utilizzata nel caso B per indicare l'angolo (in gradi) scelto dall'utente per la scansione, mentre negli altri due scenari è una variabile priva di contenuto.

Si fa notare che, qui e nel seguito, tutte le lunghezze sono sempre espresse in millimetri, tranne dove diversamente indicato.

4.2.1 Resizing

La necessità dell'operazione di scaling dell'immagine deriva dal bisogno di adeguare la dimensione (in pixel) della bitmap originale a quella (in millimetri) del disegno marcato sul pezzo. È importante sottolineare che il resizing avviene sull'immagine di partenza, cioè non ancora ruotata secondo la direzione di scansione. In caso contrario, infatti, si verificherebbe una distorsione irreversibile della bitmap, che renderebbe il risultato completamente sfalsato rispetto alle aspettative. Questa scelta, in altri termini, fa sì che lo scaling dell'immagine avvenga come se la scansione dovesse essere orizzontale oppure verticale, non obliqua. Ciò ha ripercussioni importanti sulle condizioni geometriche di accettabilità di un angolo per la scansione diagonale (vedi sez. 4.2.3).
Osservando che, nell'immagine marcata, la dimensione di un pixel coincide con quella dello spot del laser, è ragionevole imporre le seguenti condizioni:

- la distanza fra due pixel orizzontalmente adiacenti deve essere maggiore o uguale al diametro dello spot; in caso contrario, infatti, il laser investirebbe più di un pixel in uno stesso punto dell'area di lavoro, rendendo impossibile la differenziazione del loro valore di grigio.

- la distanza fra due pixel orizzontalmente adiacenti deve essere compatibile con la risoluzione del sensore di posizione installato sul robot. Ad esempio, con una risoluzione di 50 μm e un diametro dello spot di 20 μm, è evidente che il limite inferiore alla distanza fra pixel coincide con la risoluzione del sensore.

Algoritmo 4.1: scaleImage

Input: img, Dbeam, sensRes, sizeI, sizeWP, offset, Slim, modeR

```
1  rows_0 = numero di righe di img;
2  columns_0 = numero di colonne di img;
3  img2 = img ruotata di 90° in senso orario;
4  dist_x = sizeI(1)/columns_0;
5  dist_y = sizeI(2)/rows_0;
6  r = ⌊dist_x/sensRes⌋ - 2;
7  if r < 0 then r = 0 ;
8  n = ⌊dist_x/Dbeam⌋ - 1;
9  if n < 0 then n = 0 ;
10 if r > n then
11 |  points_col = n;
12 else
13 |  points_col = r;
14 points_row = ⌊dist_y/Dbeam⌋;
15 points_col_90 = points_row;
16 points_row_90 = points_col;
17 if Dbeam ≤ 1 then
18 |  rows = sizeI(2) + (sizeI(2) - 1)*points_row;
19 |  columns = sizeI(1) + (sizeI(1) - 1)*points_col;
20 |  rows_90 = sizeI(2) + (sizeI(2) - 1)*points_row_90;
21 |  columns_90 = sizeI(1) + (sizeI(1) - 1)*points_col_90;
22 else
23 |  rows = sizeI(2)/Dbeam;
24 |  columns = sizeI(1)/Dbeam;
25 |  rows_90 = sizeI(2)/Dbeam;
26 |  columns_90 = sizeI(1)/Dbeam;
27 img3 = resizing di img2 a dimensioni (columns×rows) e con interpolazione
     modeR;
28 img_90 = resizing di img2 a dimensioni (columns_90×rows_90) e con
     interpolazione modeR;
29 xScaleFactor = sizeI(1)/columns;
30 yScaleFactor = sizeI(2)/rows;
31 pix2M = [xScaleFactor,yScaleFactor];
32 xScaleFactor_90 = sizeI(1)/columns_90;
33 yScaleFactor_90 = sizeI(2)/rows_90;
34 pix2M_90 = [xScaleFactor_90,yScaleFactor_90];
35 center = [sizeWP(1),sizeWP(2)]/2;
36 drawingOrigin = center - [sizeI(1),sizeI(2)]/2 + offset;
37 S_min = 255 - Slim(2);
38 S_max = 255 - Slim(1);
39 if img3 è binaria then
40 |  img4 = img3, ma con S_max al posto di 1 e S_min al posto di 0;
41 else
42 |  img4 = S_min + (S_max - S_min)/255*img3;
```

Output: img4, img_90, pix2M, pix2M_90, drawingOrigin

- La distanza fra due pixel verticalmente adiacenti deve essere minore o uguale al diametro dello spot, per evitare discontinuità del disegno lungo la direzione ortogonale a quella di scansione.

- La distanza fra due pixel verticalmente adiacenti deve avere un limite inferiore, al di sotto del quale il grado di sovrapposizione fra passate adiacenti diventa talmente alto da generare profondità di passata eccessive, dovute al passaggio del laser su tratti già precedentemente incisi. Il limite inferiore imposto è pari alla metà del diametro dello spot.

La funzione *scaleImage* (algoritmo 4.1) implementa queste condizioni (righe 4–28) e fornisce, in output, due immagini: *img4*, relativa alla scansione orizzontale, e *img_-90*, relativa a quella verticale. Il motivo per cui l'algoritmo produce due diverse bitmap è legato al processo di ottimizzazione successivo e sarà più chiaro nel seguito. Le altre grandezze di output sono i fattori di scala delle due immagini (*pix2M* e *pix2M_90*), ovvero i vettori aventi per componenti la distanza orizzontale e verticale (in mm) fra pixel adiacenti, e *drawingOrigin*, punto che serve a riferire l'immagine all'area di lavoro ed è situato in corrispondenza del vertice inferiore sinistro della bitmap.
Si fa notare come, alla riga 3, la bitmap venga preventivamente ruotata di 90° in senso orario. Quest'operazione è legata alla successiva estrazione delle righe dall'immagine. Per compiere questa procedura, infatti, occorre tener presente che, in una matrice, le righe hanno indice i crescente da sinistra verso destra, mentre l'indice di colonna j aumenta dall'alto verso il basso. Ciò corrisponde, d'altronde, al sistema di riferimento illustrato in figura 3.1, la cui origine è situata nel vertice superiore sinistro della matrice. Nel caso delle coordinate cartesiane, al contrario, l'origine del sistema di riferimento è posta nel vertice inferiore sinistro della bitmap, con asse x orientato verso destra e asse y verso l'alto. È evidente che, mentre le coordinate x e gli indici di riga i hanno lo stesso orientamento, y e m hanno verso opposto e origine non coincidente. Per ovviare a questa problematica, che porterebbe, dopo l'estrazione delle righe, a ottenere un'immagine ribaltata verticalmente rispetto all'originale, si impone alla bitmap una rotazione di 90° orari in testa all'algoritmo.
La sezione finale di *scaleImage* (righe 37–42) è dedicata all'adeguamento della bitmap a eventuali vincoli imposti dai dati di input. In particolare, se si vuole ottenere una marcatura binaria, si fa in modo che *img4* diventi un'immagine a due soli valori (*S_min* e *S_max*), anche se, formalmente, si tratta ancora di una bitmap grayscale. Questo stratagemma consente di evitare le complicazioni implementative che risulterebbero dal dover adattare gli algoritmi successivi al caso delle immagini binarie.

4.2.2 Generazione della traiettoria

La generazione della traiettoria è lo scopo di *imageToCellMm*, funzione che si articola in una serie di parti che vengono ora approfondite, muovendosi dal livello più basso a quello più alto, continuando a fare riferimento al diagramma di figura 4.1.
Lo pseudocodice di *imageToGrayCellPix* è riportato nell'algoritmo 4.2. Nella cell array di output ciascun punto è descritto da una terna di valori, corrispondenti alle sue coordinate x e y (in pixel) e al valore di grigio. Si noti che l'immagine di input ha subito la preventiva rotazione di 90° già discussa nella sezione precedente, quindi

il ciclo alla riga 5, che è sulle colonne di *img*, si muove in realtà da una riga all'altra dell'immagine di partenza.
Un aspetto che merita di essere commentato è l'esclusione dalle righe della cell array di alcuni punti bianchi. La procedura prevede, infatti, che vengano considerati solamente quei punti che, lungo una certa riga, vanno dal primo pixel non bianco all'ultimo pixel non bianco. Questa scelta deriva dal fatto che, nell'esecuzione della marcatura, non ha senso che il laser si muova lungo righe della stessa lunghezza quando, in realtà, buona parte di esse non deve essere incisa. Il risultato è che, a vantaggio di una riduzione del tempo di lavorazione, ogni riga ha, in generale, lunghezza, punto di inizio e punto di fine differenti dalle altre.
Altro aspetto da sottolineare riguarda la lunghezza delle righe estratte: non vengono considerate righe con un numero di punti non bianchi inferiore a 2. L'errore commesso rispetto all'immagine di partenza, infatti, è di dimensione pari allo spot del laser ed è stato, quindi, reputato ininfluente sul risultato finale.

Algoritmo 4.2: imageToGrayCellPix

Input: img

```
1  m = numero di colonne di img;
2  img_double = img convertita in doppia precisione;
3  inizializza la cell array grayCellPix;
4  k = 0;
5  for ogni colonna i = 1:m do
6      ind = vettore degli indici di tutti i punti non bianchi nella colonna
        i-esima di img;
7      if ci sono almeno 2 punti non bianchi then
8          x = vettore n_i×1 con tutti gli interi compresi fra ind(1) e ind(fine);
9          y = vettore n_i×1 con ogni componente uguale a i;
10         k = k + 1;
11         grayCellPix{k,1} = [x,y,img_double(x,i)];
```

Output: GrayCellPix

Algoritmo 4.3: rotateCell

Input: inp, angle

```
1  center = media delle coordinate x e y (baricentro) dei punti di inp;
2  t = angle in radianti;
3  R = [cos(t),-sin(t);sin(t),cos(t)];
4  m = numero di righe di inp;
5  inizializza la cell array outCell;
6  for ogni riga i = 1:m do
7      row = inp{i};
8      rowT = prime due colonne di row trasposte (matrice 2×m);
9      p0_rot = R*(rowT - center);
10     p_rot = trasposta di (p0_rot + center);
11     outCell{i} = [p_rot,row(:,3)];
```

Output: outCell

Algoritmo 4.4: transformGrayPixelsToMm

Input: grayCellPix, pix2M, drawingOrigin

```
1  xMinPix = coordinata x minima fra quelle dei punti appartenenti a
     grayCellPix;
2  yMinPix = coordinata y minima fra quelle dei punti appartenenti a
     grayCellPix;
3  m = numero di righe di grayCellPix;
4  inizializza la cell array grayCellMm (m×1);
5  for ogni riga i = 1:m do
6  |  row = grayCellPix{i};
7  |  coordsPix = prime due colonne di row;
8  |  coordsMeters = pix2M.*(coordsPix - [xMinPix,yMinPix]) +
   |    drawingOrigin;
9  |  grayCellMm{i} = [coordsMeters,row(:,3)];
```

Output: grayCellMm

Algoritmo 4.5: normalizeRow

Input: row

```
1  n = numero di righe di row;
2  inizializza z: matrice con prime due colonne nulle e terza colonna uguale a
     quella di row;
3  x_max = max(row(:,1));
4  x_min = min(row(:,1));
5  y_max = max(row(:,2));
6  y_min = min(row(:,2));
7  if row è verticale then
8  |  z(:,1) = (row(:,1) - x_min)/(x_max - x_min);
9  else
10 |  z(:,1) = |(row(:, 2) − y_min)|/(y_max - y_min);
11 if z(1,1) = 1 e z(n,1) = 0 then
12 |  z2(:,1) = 1 - z(:,1);
13 |  z2(:,2) = z(:,2);
14 else
15 |  z2 = z;
```

Output: z2

L'algoritmo 4.3, riferito alla funzione *rotateCell*, applica a ciascuna riga della cell array di input una matrice di rotazione. Questa operazione non prevede interpolazioni del tipo di quelle viste nella sezione 3.2.2.2, nè per il posizionamento dei pixel, nè per il livello di grigio. Di conseguenza, molti punti risultano avere dei valori di coordinate frazionari, che sono tuttavia accettabili, dal momento che l'informazione sulla posizione di un pixel non è più data dagli indici di una matrice, che, per definizione, devono essere interi, ma da dei numeri contenuti in un vettore di una cell array, i quali sono, in generale, valori reali.

transformGrayPixelsToMm (algoritmo 4.4) converte le coordinate dei punti della cell array di input da pixel a millimetri, effettuando anche una traslazione del siste-

ma di riferimento (riga 8). Il nuovo sistema di assi, infatti, non è più centrato nel vertice inferiore sinistro dell'immagine, ma, invece, nel vertice omologo del pezzo in lavorazione (che si suppone essere di forma rettangolare).
In analogia con MATLAB, si è indicata con '.*' l'operazione di moltiplicazione ordinata fra matrici, elemento per elemento. Ad esempio, dati un vettore riga $\boldsymbol{v} = (v_1, v_2)$ e una matrice

$$\boldsymbol{M} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \\ m_{31} & m_{32} \end{bmatrix}, \tag{4.1}$$

allora il loro prodotto ordinato è dato da

$$\boldsymbol{v}.*\boldsymbol{M} = \begin{bmatrix} v_1 m_{11} & v_2 m_{12} \\ v_1 m_{21} & v_2 m_{22} \\ v_1 m_{31} & v_2 m_{32} \end{bmatrix}. \tag{4.2}$$

La funzione *normalizeRow* (algoritmo 4.5) riceve in input una matrice $n \times 3$, che rappresenta un singolo elemento della cell array contenente tutte le righe dell'immagine, e ne parametrizza le coordinate (contenute nelle prime due colonne) fra 0 e 1, come richiesto dalla macro G11.

4.2.3 Ottimizzazione della traiettoria

Il problema di ottimizzazione ha lo scopo di determinare la traiettoria che permette di realizzare l'operazione di marcatura di un'immagine nel minor tempo possibile. In base agli algoritmi descritti finora, ciascun tratto di percorso lungo il quale si muove lo spot laser può essere legato o a un moto di avanzamento (spostamento a laser acceso lungo una riga) o a uno di appostamento (spostamento a laser spento fra la fine di una riga e l'inizio della successiva). Fatta eccezione per i moti di appostamento, minoritari rispetto a quelli di avanzamento, il processo tecnologico richiede di essere eseguito a velocità costante. Di conseguenza, non si commette un errore significativo se, ai fini dell'ottimizzazione, si trascurano i tratti di interpolazione rapida G0 e si suppone, quindi, che l'intero tragitto venga percorso alla velocità assegnata al comando G1. In questo modo, determinare il percorso che minimizza il tempo ciclo equivale a calcolare il tragitto di lunghezza complessiva minima.
Fatte queste premesse, l'ottimizzazione si articola in due fasi:

1. individuazione dell'angolo di scansione che minimizza la distanza percorsa a laser acceso;

2. per dato angolo di scansione, individuazione del tragitto che minimizza la distanza percorsa a laser spento o, in altri termini, ottimizzazione del punto di inizio della marcatura.

Lo step 1 viene implementato a livello della funzione *imageToCellMm* solo nel caso in cui sia l'utente a richiederlo (*mode = 'optimized'*), mentre il punto 2 fa parte di *parametrizeCell* ed è attivo per qualunque modalità di scansione. Prima, però, di descrivere gli algoritmi appena citati, si approfondisce la formulazione del problema dal punto di vista teorico.

Entrambe le fasi dell'ottimizzazione sono riconducibili a un problema noto, in letteratura, come *traveling salesman problem* (TSP) [15]. Dati un insieme di punti

e una variabile di costo da ottimizzare (tempo, distanza, costo economico o altro), il TSP si pone l'obiettivo di determinare il percorso che, oltre a far tappa in tutti i punti dell'insieme, minimizza anche il costo del "viaggio". Nonostante la sua impostazione appaia intuitivamente semplice, il problema è estremamente complesso sotto il profilo matematico, specialmente quando si ha a che fare con un elevato numero di punti. Delle varie formulazioni esistenti, viene qui proposta quella di Danzig–Fulkerson–Johnson.
Dato un insieme di n punti, detta $d_{ij} > 0$ la distanza fra il punto i e il punto j, se si definiscono i valori binari

$$x_{ij} = \begin{cases} 1 & \text{se il percorso va da } i \text{ verso } j \\ 0 & \text{altrimenti} \end{cases} \tag{4.3}$$

allora il TSP può essere espresso con il seguente problema di programmazione lineare:

$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} d_{ij} x_{ij} : \begin{cases} x_{ij} \in \{0,1\} & i,j = 1,...,n \\ \sum_{i=1, i \neq j}^{n} x_{ij} = 1 & j = 1,...,n \\ \sum_{j=1, j \neq i}^{n} x_{ij} = 1 & i = 1,...,n \\ \sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1 & \forall Q \subset \{1,...,n\}, |Q| \geq 2 \end{cases} \tag{4.4}$$

La seconda condizione del sistema 4.4 assicura che ciascun punto sia immediatamente successivo, nel tragitto stabilito, a un solo altro punto. La terza, analogamente, stabilisce che ogni punto deve precederne uno e uno solo. L'ultima condizione fa sì che non si creino dei loop aggiuntivi costituiti da un sottoinsieme degli n punti.

Come in tutti i problemi di ricerca operativa, la soluzione può essere ricavata in modo esatto, euristico oppure meta-euristico. Per il TSP, in genere, una soluzione analitica può essere ottenuta in tempi accettabili solo per poche decine di punti, che diventano alcune centinaia nel caso si impieghi un metodo meta-euristico. Nel caso in esame, l'insieme dei punti in cui il percorso deve fare tappa è costituito da quelli di inizio e fine di ciascuna riga della cell array contenente le informazioni relative all'immagine. Non avrebbe alcun senso, infatti, includere anche i punti interni, poiché il laser deve comunque percorrerli, muovendosi sempre su traiettorie rettilinee. Il calcolo risulterebbe, peraltro, inutilmente più complesso, aggiungendo al sistema risolutivo una mole di variabili ed equazioni ininfluenti sul risultato finale. Alla luce di queste considerazioni, è facile immaginare che il numero di punti in input al TSP, calcolabile come il doppio del numero di righe della cell array, possa facilmente sforare le migliaia, nel caso più generale. È quindi evidente che gli unici metodi impiegabili sono, in questo contesto, quelli euristici, basati su approssimazioni che permettono di giungere a una soluzione sub-ottimale, ma comunque parzialmente ottimizzata.

Essendo un problema di grande interesse pratico, il TSP finora descritto è stato studiato e ridefinito sulla base delle sue particolari applicazioni. In particolare, risulta interessante, per il seguito della trattazione, approfondire le sue seguenti generalizzazioni:

- ***Generalized TSP*** (**GTSP**): i punti sono suddivisi in gruppi e il percorso ottimizzato deve contenere un solo punto per ciascun gruppo (fig. 4.2a). In figura 4.2b è mostrato un esempio legato a un'applicazione di taglio laser [2]: tutti i punti del perimetro di ciascuna lavorazione fanno parte di un gruppo e, siccome il fascio compie un percorso chiuso lungo il bordo di taglio, per ottimizzare il tempo ciclo complessivo è sufficiente minimizzare i moti di appostamento, dato che quelli di lavoro hanno lunghezza prefissata e dipendente dal bordo di taglio stesso.

- ***Precedence-constrained TSP*** (**PCTSP**): il percorso deve rispettare delle precedenze fra i punti da visitare. È una problematica che trova applicazione, ad esempio, in operazioni di foratura complesse [9], per le quali, ai fini dell'ottimizzazione, è necessario conciliare le fasi di movimentazione e di sostituzione dei diversi utensili.

- ***Precedence-constrained generalized TSP*** (**PCGTSP**): si tratta di un'unione dei due casi precedenti, introdotta per la prima volta da [14]. I punti sono, quindi, suddivisi in gruppi ed esistono dei vincoli di precedenza fra un gruppo e l'altro.

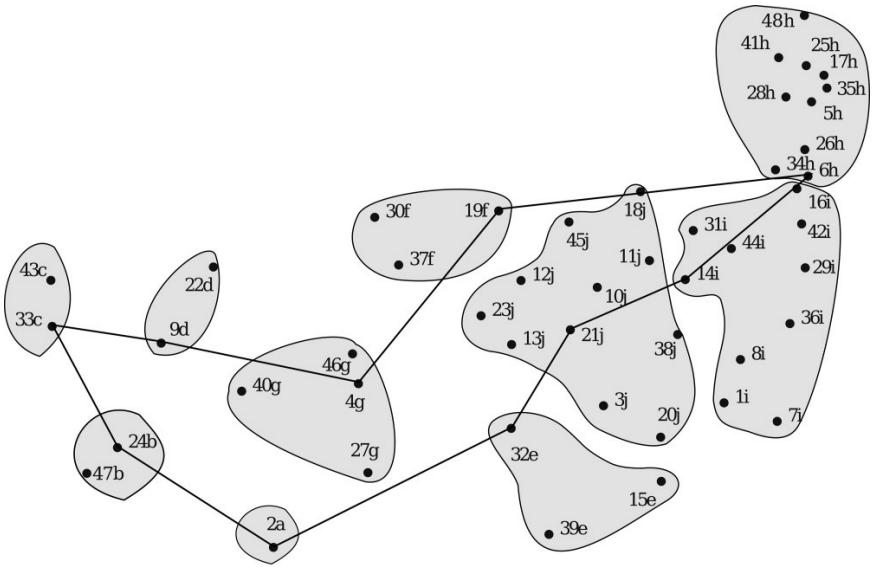
Trattandosi di generalizzazioni, queste tre casistiche presentano complessità ancora maggiore rispetto al TSP standard, sia dal punto di vista della formulazione che della risoluzione.

Per quel che riguarda il problema di ottimizzazione in esame, si parte dall'analisi del punto 2, che è schematizzabile secondo uno dei seguenti modelli:

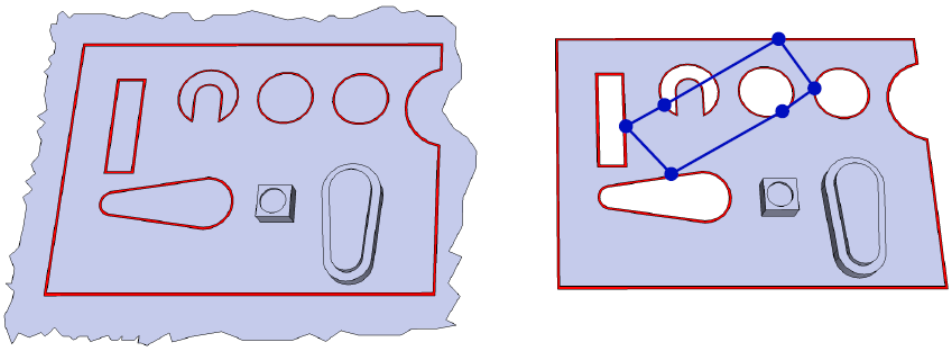
a. un PCGTSP in cui ciascun gruppo è composto da due punti (inizio e fine della riga di scansione), ma il loop, a differenza dell'esempio di figura 4.2b, è aperto, poiché il laser percorre ciascuna riga in una sola direzione, senza ritornare al suo punto d'inizio, per poi spostarsi a quella successiva;

b. un PCTSP in cui i vincoli di precedenza sono variabili a seconda della direzione in cui ciascuna riga viene scansionata.

Dall'analisi della letteratura scientifica non è emerso un modello assimilabile a nessuno dei due appena descritti. Conseguentemente, l'unica strada per arrivare a una soluzione è quella di definire un metodo euristico ragionevolmente valido per questa specifica applicazione.
L'approssimazione adottata si basa sulla logica nearest neighbor (NN), già introdotta nella sezione 3.2.2.2. Una volta scelto il punto di partenza fra gli n che costituiscono la traiettoria, il metodo NN prevede di muoversi sempre, volta per volta, verso il punto più vicino fra quelli non ancora visitati. Secondo [13], questa euristica consente di ottenere, nel TSP standard, un tragitto che è il 25% più lungo di quello minimo. Nel caso di questa tesi, tuttavia, il problema affrontato non è il TSP standard, bensì una sua generalizzazione, per la quale non è detto che valga la stessa stima della vicinanza del risultato alla soluzione ottima.
L'implementazione dell'ottimizzazione del punto di inizio della marcatura è esemplificata in figura 4.3, che mostra quattro possibili alternative. Si può infatti affermare che, volendo evitare moti di appostamento superflui, la lavorazione possa partire, a prescindere dall'angolo di scansione, da uno dei seguenti punti:



(a) Schema concettuale



(b) Esempio applicativo [2]

Figura 4.2: Il generalized traveling salesman problem.

- primo punto della prima riga (fig. 4.3a);
- ultimo punto della prima riga (fig. 4.3b);
- primo punto dell'ultima riga (fig. 4.3c);
- ultimo punto dell'ultima riga (fig. 4.3d).

Riferendosi, per semplicità, a una scansione orizzontale, questo approccio equivale a decidere se iniziare l'operazione dal punto più in alto a sinistra, in alto a destra, in basso a sinistra o in basso a destra dell'immagine. La scelta influisce in modo significativo sui successivi cambi di riga, effettuati secondo la logica NN, e, quindi, sulla lunghezza dell'intero percorso che il laser segue. Per ciascun angolo di scansione considerato, dunque, l'algoritmo valuta quale sia la migliore fra le quattro alternative.

Anche lo step 1 dell'ottimizzazione è un problema riconducibile, in un certo senso, al TSP. In questo caso, tuttavia, si aggiunge la complicazione legata al fatto che, al variare dell'angolo di scansione α, cambia anche l'immagine in input all'algoritmo (che, a partire dall'originale, viene ruotata preventivamente di $-\alpha$).

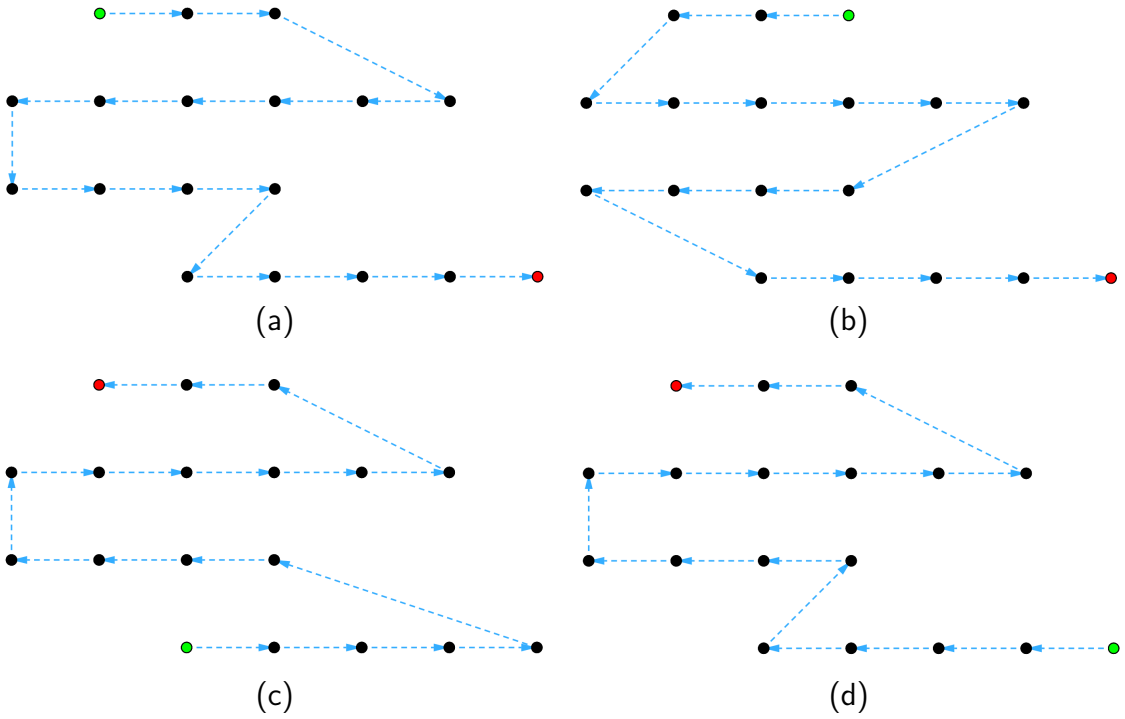


Figura 4.3: Confronto fra quattro possibili percorsi di marcatura (azzurro), differenti in base ai punti di inizio (verde) e fine (rosso). Il passaggio da una riga alla successiva sfrutta la logica NN.

Dal momento che non sono stati trovati modelli idonei a trattare questo caso, è molto difficile, se non impossibile, definire il problema in modo matematicamente rigoroso. Non resta dunque che determinare, ancora una volta, una logica empirica che consenta di ottenere una soluzione ragionevolmente valida.
L'euristica implementata si articola nei seguenti passi:

I. viene definito un insieme di possibili angoli di scansione, ovvero, per convenzione, tutti quelli compresi fra -90° e 90° con salti di 10° (ad esempio -90°, -80°, -70°, ..., 0°, 10°, ecc.);

II. gli angoli definiti vengono filtrati sulla base di criteri geometrici;

III. per ciascuno degli angoli rimasti, si calcola il percorso di lavorazione e si individua quello (sia α_0) che consente di minimizzare il tempo ciclo;

IV. si ripete il passo precedente per tutti gli angoli interi di valore compreso nell'intorno $\alpha_0 \pm 5°$ che rispettano i criteri geometrici al punto 2, ricavando, infine, l'angolo α_{ott} cercato.

L'ottimizzazione è, quindi, articolata in due stadi, di cui il primo serve a individuare un range di valori in cui approfondire la ricerca dell'angolo ottimale, mentre il secondo è quello che, a tutti gli effetti, determina α_{ott}. L'implementazione dei criteri di filtraggio, ai punti II e IV, ha un duplice scopo:

- consentire una corretta resa grafica della marcatura;
- ridurre il tempo di calcolo.

La seconda esigenza è giustificata dal fatto che, per un algoritmo brute-force come questo, ciascuna iterazione del processo, corrispondente a un particolare angolo, influisce in maniera considerevole sul tempo di calcolo complessivo. Il primo aspetto, invece, è direttamente legato ai criteri di selezione degli angoli, i quali fissano un range di valori ammissibili per la distanza (verticale e orizzontale) fra pixel adiacenti, in maniera del tutto analoga a quanto visto per la funzione *scaleImage* nella sezione 4.2.1. L'immagine di partenza, infatti, subisce un resizing che è esatto solo per angoli di scansione di 0° oppure 90°, mentre, nel caso in cui essa venga ruotata e poi scalata in mm, occorre assicurarsi che le distanze fra i punti che la costituiscono siano compatibili col diametro dello spot.
Per valutare queste condizioni, allora, è necessario calcolare le distanze d_x (orizzontale) e d_y (verticale) fra pixel adiacenti. Ricapitolando, le operazioni geometriche che l'immagine subisce nel corso dell'algoritmo sono le seguenti:

i. la bitmap viene ruotata di un angolo $-\alpha$ e ne vengono estratte le righe, che costituiscono gli elementi di una nuova cell array;

ii. ciascuna riga della cell array risultante viene ruotata di α, in modo che l'immagine torni all'orientamento originale;

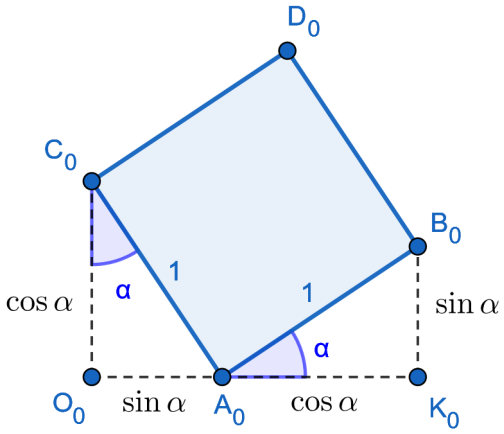
iii. vengono applicati all'immagine i fattori di scala x e y, cioè viene moltiplicata ciascuna distanza orizzontale per x e ciascuna distanza verticale per y.

Si consideri la figura 4.4a, che riporta, schematicamente, quattro pixel adiacenti di un'immagine che ha subito le prime due trasformazioni. I punti A_0 e B_0 giacciono sulla stessa riga di scansione, così come i punti C_0 e D_0. Tutte le distanze lungo le righe e le colonne di scansione sono unitarie, poiché la rotazione (realizzata da *rotateCell*) preserva le distanze fra i pixel appartenenti alla matrice di partenza. Una volta applicati i fattori di scala, come da punto iii, la situazione è quella di figura 4.4b, nella quale si nota che il quadrilatero $\square$ABCD si è trasformato in un parallelogramma e l'angolo $\angle$BAK ha assunto un nuovo valore β. La differenza fra quest'ultimo e α è che, mentre il secondo rappresenta un angolo di scansione "virtuale" dell'immagine, al fine dell'estrazione delle sue righe, β coincide, invece, con l'angolo fisico di scansione del laser durante la marcatura. Dal punto di vista pratico, dunque, β è il valore di maggior interesse per l'utente e, per questo motivo, è l'unico che viene mostrato nell'interfaccia grafica.
La relazione fra i due angoli è data da

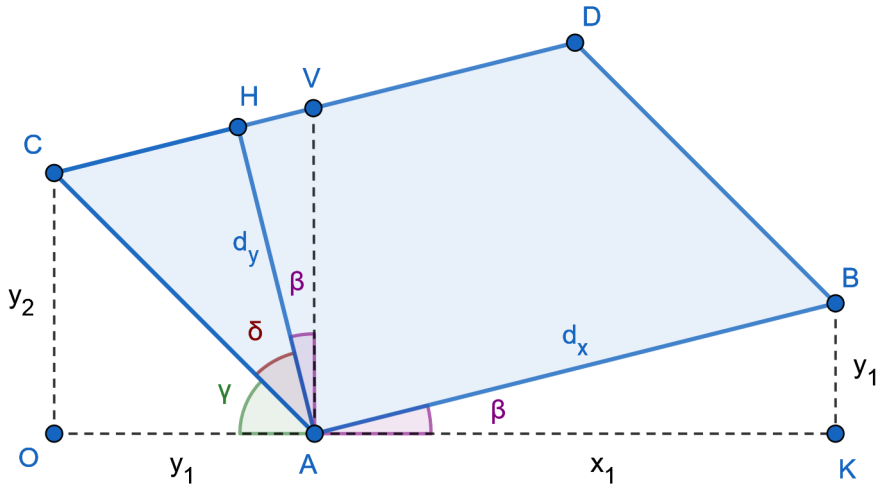
$$\beta = \arctan(\frac{y}{x}\tan\alpha), \tag{4.5}$$

mentre il valore di d_x si ricava tramite i seguenti passaggi:

$$x_1 = x\cos\alpha \tag{4.6}$$
$$x_2 = y\sin\alpha \tag{4.7}$$
$$d_x = \sqrt{{x_1}^2 + {x_2}^2} \tag{4.8}$$



(a) Immagine non scalata



(b) Immagine scalata

Figura 4.4: Schematizzazione del processo di applicazione dei fattori di scala all'immagine.

Indicando con γ l'angolo $\angle$OAC e con δ l'angolo $\angle$CAH, è possibile ottenere d_y come segue:

$$y_1 = x \sin \alpha \tag{4.9}$$

$$y_2 = y \cos \alpha \tag{4.10}$$

$$\overline{\mathrm{AC}} = \sqrt{{y_1}^2 + {y_2}^2} \tag{4.11}$$

$$\gamma = \arctan(\frac{y}{x} \cot \alpha) \tag{4.12}$$

$$\delta = \frac{\pi}{2} - \gamma - \beta \tag{4.13}$$

$$d_y = \overline{\mathrm{AC}} \cos \delta \tag{4.14}$$

A questo punto, indicando con D il diametro dello spot, si impongono i criteri geometrici:

$$D \leq d_x < 2D \tag{4.15}$$

$$\frac{D}{2} < d_y \leq D \tag{4.16}$$

La condizione 4.16 equivale a quella già imposta nel resizing, mentre la 4.15 presenta anche un limite superiore, fissato a $2D$, per fare in modo che la densità di punti (e quindi di informazione) lungo una riga sia sufficiente a garantire una corretta resa dell'immagine marcata.

Algoritmo 4.6: parametrizeCell

```
   Input: inp, speed
 1 inp2 = inp con la prima riga invertita (primo punto diventa ultimo);
 2 inp3 = inp ribaltata (prima riga diventa ultima);
 3 inp4 = inp3 con la prima riga invertita;
 4 n = numero di righe di inp;
 5 Out = cell array n×1×4, contenente {inp,inp2,inp3,inp4};
 6 inizializza la cell array parOut (n×1×4);
 7 inizializza la matrice flipRow0 (n×4);
 8 for i = 1:4 do
 9     for ogni riga j = 1:n do
10         row = Out{j,:,i};
11         m = numero di righe di row;
12         if j > 1 then
13             row_pr = Out{j-1,:,i};
14             dist0 = ‖row(1,1:2) - row_pr(m,1:2)‖;
15             dist1 = ‖row(end,1:2) - row_pr(m,1:2)‖;
16         if j > 1 e dist1 < dist0 then
17             row2 = normalizeRow(row);
18             row2(:,1) = 1 - row2(:,1);
19             row_par = row2 invertita dall'alto al basso;
20             Out{j,:,i} = Out{j,:,i} invertita dall'alto al basso;
21             flipRow0(j,i) = 1;
22         else
23             row_par = normalizeRow(row);
24         parOut{j,:,i} = row_par;
25 inizializza d_on = 0;
26 for ogni riga j = 1:n di inp do
27     r = inp{j,:};
28     s = ‖r(m,1:2) - r(1,1:2)‖;
29     d_on = d_on + s;
30 inizializza d_off (vettore 4×1);
31 for i = 1:4 do
32     v1 = matrice (n - 1)×2 contenente le coordinate (x,y) di tutti i punti
        iniziali di ogni riga (eccetto la prima) di Out{:,:,i};
33     v2 = matrice (n - 1)×2 contenente le coordinate (x,y) di tutti i punti
        finali di ogni riga (eccetto l'ultima) di Out{:,:,i};
34     norm0 = ‖v2 - v1‖ fatta secondo le righe, vettore (n - 1)×1;
35     d_off(i) = somma degli elementi di norm0;
36 ind = indice corrispondente al valore minimo di d_off;
```

37 **if** *ind è un vettore* **then** ind = ind(1) ;
38 t_cycle = 60*(d_on + d_off(ind))/speed(2);
39 parCell = parOut(:,:,ind);
40 Cell = Out(:,:,ind);
41 flipRow = flipRow0(:,ind);
Output: parCell, Cell, flipRow, t_cycle

L'implementazione di queste logiche avviene a partire dalla funzione *parametrizeCell* (algoritmo 4.6), la quale, come anticipato, oltre a parametrizzare la traiettoria (contenuta nella cell array *inp*) in vista della sua trasformazione in GCode, ottimizza il punto di inizio della marcatura, applicando l'euristica brute-force basata sul metodo NN. Sulla base di quest'ultima, ciascuna riga può essere mantenuta invariata oppure ribaltata di ordine, in modo che il primo punto diventi l'ultimo e viceversa. A questa operazione viene associato il vettore binario di output *flipRow*, che vale 1 in caso di ribaltamento e 0 altrimenti. Il suo ruolo è quello di permettere, nella successiva fase di parsing, la ricostruzione della cell array di righe a partire dal listato GCode. Gli altri output sono la cell array ottimizzata *Cell*, quella parametrizzata *parCell* e il tempo di lavorazione *t_cycle*, espresso in secondi.

Tutte le operazioni geometriche sull'immagine, comprendenti buona parte delle funzioni fin qui descritte, sono raggruppate all'interno di *imageProcessingLoop* (algoritmo 4.7). La variabile di input *angle* è il vettore degli angoli di scansione fisica β in gradi.
Prima di effettuare le trasformazioni sulla bitmap, l'algoritmo prevede due operazioni preliminari:

- conversione da β ad α (righe 9 e 17);
- scelta dell'immagine con risoluzione per scansione orizzontale (*img*) oppure verticale (*img_90*), a seconda del valore di *angle* (righe 7, 12 e 15).

Il secondo punto è stato implementato poiché si è notato che l'algoritmo tende, in base ai criteri geometrici già discussi, a preferire *img* per angoli di modulo inferiore a 45° e *img_90* per gli altri.

Il filtraggio degli angoli viene effettuato da *pickAngles* (algoritmo 4.8). In caso di scansione a 0° o 90° (scenario A), questa funzione viene bypassata. Se l'utente sceglie, invece, di ottimizzare l'angolo di scansione (scenario C), l'algoritmo applica le condizioni 4.15 e 4.16 secondo quanto già visto. Nello scenario B, infine, nel caso in cui l'angolo indicato manualmente non dovesse rispettare i criteri su d_x e d_y, il sistema seleziona automaticamente l'angolo intero più vicino a quello designato dall'utente.

Algoritmo 4.7: imageProcessingLoop
Input: img, img_90, angle, pix2M, pix2M_90, speed, drawingOrigin
1 angle2 = angle in radianti;
2 n = numero di righe di angle;
3 inizializza le cell array Gray, parGray e flipRow (tutte 1×1×n);
4 inizializza t_cycle, vettore n×1;

```
for ogni angolo i = 1:n do
    if angle(i) = 0 then
        img_i = img;
        pix2M_i = pix2M;
        alpha = atan(tan(angle2(i))*pix2M_i(1)/pix2M_i(2)), in gradi;
    else
        if |angle(i)| > 45 then
            img2 = img_90;
            pix2M_i = pix2M_90;
        else
            img2 = img;
            pix2M_i = pix2M;
        alpha = atan(tan(angle2(i))*pix2M_i(1)/pix2M_i(2)), in gradi;
        img_i = img2 ruotata di alpha in senso orario (interpolazione NN);
    grayCellPix = imageToGrayCellPix(img_i);
    if angle(i) = 0 then
        grayCellPix2 = grayCellPix;
    else
        grayCellPix2 = rotateCell(grayCellPix,alpha);
    grayCellMm =
     transformGrayPixelsToMm(grayCellPix2,pix2M_i,drawingOrigin);
    [parGray{:,:,i},Gray{:,:,i},flipRow{:,:,i},t_cycle(i)] =
     parametrizeCell(grayCellMm,speed);
Output: parGray, Gray, flipRow, t_cycle
```

Algoritmo 4.8: pickAngles

```
Input: pix2M, pix2M_90, Dbeam, mode, varargin
x = pix2M(1);
y = pix2M(2);
x_90 = pix2M_90(1);
y_90 = pix2M_90(2);
if mode = 'diagonal' then
    beta = vettore colonna con tutti i numeri interi da 1 a 89, convertiti in
     radianti;
else
    if varargin non viene inserito then
        beta = vettore colonna con i numeri da 10 a 80 (a salti di 10),
         convertiti in radianti;
    else
        beta = varargin in radianti;
alpha = atan(tan(beta)*x/y);
alpha_90 = atan2(tan(beta)*x_90/y_90);
```

14 dx = vettore della dimensione di alpha dato dalla norma di [x*cos(alpha),y*sin(alpha)] fatta per righe;
15 ind_x = vettore binario che vale 1 per gli indici di dx per cui D $\leq$ dx $<$ 2D;
16 dx_90 = vettore della dimensione di alpha dato dalla norma di [x_90*cos(alpha_90),y_90*sin(alpha_90)] fatta per righe;
17 ind_x_90 = vettore binario che vale 1 per gli indici di dx_90 per cui D $\leq$ dx_90 $<$ 2D;
18 gamma = atan(y/x*cot(alpha));
19 delta = $\pi/2$ - gamma - beta;
20 dy = vettore della dimensione di alpha dato dalla norma di [x*sin(alpha),y*cos(alpha)].*cos(delta) fatta per righe;
21 ind_y = vettore binario che vale 1 per gli indici di dy per cui D/2 $<$ dy $\leq$ D;
22 gamma_90 = atan(y_90/x_90*cot(alpha_90));
23 delta_90 = $\pi/2$ - gamma_90 - beta;
24 dy_90 = vettore della dimensione di alpha_90 dato dalla norma di [x_90*sin(alpha_90),y_90*cos(alpha_90)].*cos(delta_90) fatta per righe;
25 ind_y_90 = vettore binario che vale 1 per gli indici di dy_90 per cui D/2 $<$ dy_90 $\leq$ D;
26 ind = AND logico fra ind_x e ind_y;
27 ind_90 = AND logico fra ind_x_90 e ind_y_90;
28 ind_tot = OR logico fra ind e ind_90;
29 beta2 = beta(ind_tot) in gradi;
30 **if** *mode = 'diagonal'* **then**
31 | ang1 = vettore [-beta2;beta2] ordinato e senza ripetizioni;
32 | ang2 = valore di ang1 più vicino a varargin;
33 **else**
34 | **if** *ang2 è vuoto* **then**
35 | | ang = il numero più vicino a varargin fra 0 e 90 (o l'uno o l'altro);
36 | **else**
37 | | ang = ang2(1);
38 | **if** *varargin non viene inserito* **then**
39 | | ang = vettore [0;-beta2;beta2;90] ordinato e senza ripetizioni;
40 | **else**
41 | | ang = vettore beta2 ordinato e senza ripetizioni;

Output: ang

Algoritmo 4.9: imageToCellMm

Input: img, img_90, pix2M, pix2M_90, Dbeam, mode, speed, drawingOrigin, varargin

1 **if** *mode = 'horizontal'* **then**
2 | angle1 = 0;
3 **else if** *mode = 'vertical'* **then**
4 | angle1 = 90;

```
else if mode = 'diagonal' then
    angle1 = pickAngles(pix2M,pix2M_90,Dbeam,mode,varargin);
else if mode = 'diagonal' then
    angle1 = pickAngles(pix2M,pix2M_90,Dbeam,mode,varargin);
else
    angle1 = pickAngles(pix2M,pix2M_90,Dbeam,mode);
[parGray1,Gray1,flipRow1,t_cycle1] = imageProcessingLoop(img,
 img_90,angle1,pix2M,pix2M_90,speed,drawingOrigin);
ind_opt1 = indice corrispondente al valore minimo del vettore t_cycle;
if mode = 'optimized' then
    angle_opt1 = angle1(ind_opt1);
    angle2_0 = vettore colonna contenente i numeri interi compresi fra
     (angle_opt1 - 5) e (angle_opt1 + 5), escluso angle_opt1;
    if angle2_0 contiene almeno un angolo > 90° then
        angle2_1 = angle2_0 con tutti gli angoli convertiti nel range [-89°,90°];
    else
        angle2_1 = angle2_0;
    angle2 = pickAngles(pix2M,pix2M_90,Dbeam,mode,angle2_2);
    if angle2 non è vuoto then
        [parGray2,Gray2,flipRow2,t_cycle2] =
         imageProcessingLoop(img,img_90,angle2,pix2M,
         pix2M_90,speed,drawingOrigin);
        ind_opt2 = indice corrispondente al valore minimo del vettore
         t_cycle2;
        if min(t_cycle1) ≤ min(t_cycle2) then
            ind_opt = ind_opt1;
            Gray_opt = Gray1{:,:,ind_opt};
            parGray_opt = parGray1{:,:,ind_opt};
            flipRow_opt = flipRow1{:,:,ind_opt};
            angle_opt = angle1(ind_opt);
            t_cycle_opt = t_cycle1(ind_opt);
        else
            ind_opt = ind_opt2;
            Gray_opt = Gray2{:,:,ind_opt};
            parGray_opt = parGray2{:,:,ind_opt};
            flipRow_opt = flipRow2{:,:,ind_opt};
            angle_opt = angle2(ind_opt);
            t_cycle_opt = t_cycle2(ind_opt);
    else
        ind_opt = ind_opt1;
        Gray_opt = Gray1{:,:,ind_opt};
        parGray_opt = parGray1{:,:,ind_opt};
        flipRow_opt = flipRow1{:,:,ind_opt};
        angle_opt = angle1(ind_opt);
        t_cycle_opt = t_cycle1(ind_opt);
```

```
45 else
46 |   ind_opt = ind_opt1;
47 |   Gray_opt = Gray1{:,:,ind_opt};
48 |   parGray_opt = parGray1{:,:,ind_opt};
49 |   flipRow_opt = flipRow1{:,:,ind_opt};
50 |   angle_opt = angle1(ind_opt);
51 |   t_cycle_opt = t_cycle1(ind_opt);
```

Output: parGray_opt, Gray_opt, flipRow_opt, angle_opt, t_cycle_opt

L'ottimizzazione dell'angolo di scansione è realizzata da *imageToCellMm*, il cui pseudocodice è riportato nell'algoritmo 4.9.

L'ultimo tassello della sezione di image processing è rappresentato da *ImageProcessingFull* (algoritmo 4.10), che esegue, in sequenza, le operazioni di resizing e di generazione e ottimizzazione della traiettoria. Il termine *full*, impiegato per la denominazione della funzione, fa riferimento a uno dei nomi commerciali comunemente utilizzati per indicare la marcatura a scansione (*full mode*).

Algoritmo 4.10: ImageProcessingFull

Input: img, sensRes, Dbeam, mode, speed, sizeI, sizeWP, offset, Slim, modeR, varargin

```
1 [img2,img_90,pix2M,pix2M_90,drawingOrigin] =
   scaleImage(img,Dbeam,sensRes,sizeI,sizeWP,offset,Slim,modeR);
2 if varargin non viene inserito then
3 |   [parGray,Gray,flipRow,ang,t_cycle] = imageToCellMm(img2,
  |    img_90,pix2M,pix2M_90,Dbeam,mode,speed,drawingOrigin);
4 else
5 |   [parGray,Gray,flipRow,ang,t_cycle] = imageToCellMm(img2,
  |    img_90,pix2M,pix2M_90,Dbeam,mode,speed,drawingOrigin,varargin);
```

Output: Gray, parGray, flipRow, drawingOrigin, t_cycle, ang

4.3 Generazione del GCode

Una volta determinata la traiettoria, contenuta nelle cell array *Gray* e *parGray* in output a *ImageProcessingFull*, essa viene convertita in GCode dalla funzione *GCodeFull*, i cui dati di input sono, oltre alle variabili citate, anche *drawingOrigin*, *speed* e un numero indicato con *thresh*. Quest'ultimo rappresenta un valore di soglia, impostato dall'utente, che ha lo scopo di stabilire la differenza minima fra i valori di grigio di due *set-point* adiacenti. Dal punto di vista del controllo della macchina, infatti, dover comandare una variazione della potenza punto per punto può diventare complicato nel caso in cui siano richieste fluttuazioni ad alta frequenza di tale parametro. A determinare questo limite tecnologico è la potenza di calcolo dell'unità di controllo, la cui efficacia nella gestione dei segnali dipende anche dalla velocità di avanzamento richiesta per il processo. Se la frequenza di variazione della potenza è troppo alta per l'unità di controllo, allora è necessario ridurre la velocità di processo. Per quel che riguarda il risultato finale, in realtà, variazioni minime di potenza non

Algoritmo 4.11: simplifyGCodeFull

Input: parRow, thresh

```
1  n = numero di righe di parRow;
2  inizializza parOut (vettore 1×2);
3  i = 1 (indice delle righe di parRow);
4  r = 0 (contatore della lunghezza del vettore v);
5  k = 0 (contatore della lunghezza di parOut);
6  while true do
7      r = r + 1;
8      v(1,:) = parRow(i,:);
9      if i < n then
10         for j = i+1:n do
11             if (j = i + 1 e |parRow(i,2) - parRow(j,2)| ≤ thresh) oppure (j >
                i + 1 e il massimo fra |m - parRow(j,2)| e |M - parRow(j,2)| è
                ≤ thresh) then
12                 r = r + 1;
13                 v(r,:) = parRow(j,:);
14                 m = minimo della seconda colonna di v;
15                 M = massimo della seconda colonna di v;
16             else
17                 i = j;
18                 esci dal loop (solo il for, non il while);
19         if v ha una sola riga then
20             v2 = v;
21         else
22             val = media dei valori della seconda colonna di v;
23             if almeno il 70% dei punti di v sono bianchi then
24                 mean_val = 255;
25             else if almeno il 70% dei punti di v sono neri then
26                 mean_val = 0;
27             else
28                 mean_val = val;
29             v2 = [v(ultima riga,1),mean_val arrotondato];
30     else
31         v2 = v;
32         i = i + 1;
33     if v2(1,1)≠0 then
34         k = k + 1;
35         parOut(k,:) = v2;
36     reinizializza v = [0,0];
37     reinizializza r = 0;
38     if v2(1,1) = 1 then esci dal loop;
```

Output: parOut

corrispondono, spesso, a differenze percettibili nel colore della marcatura. Per questo si è pensato di introdurre, nell'interfaccia grafica (vedi cap. 5), un parametro che stabilisce la qualità del GCode, il quale controlla, a sua volta, il valore di *thresh*, che può variare fra 0 (qualità massima) e 60 (qualità minima).

Complessivamente, la struttura di questa sezione del software è piuttosto semplice, in quanto *GCodeFull* contiene al proprio interno una sola sotto-funzione, denominata *simplifyGCodeFull* (algoritmo 4.11), che si occupa della semplificazione dell'input secondo la logica a cui si è appena fatto riferimento. Tale funzione riceve in input il parametro *thresh* e una matrice a due colonne (*parRow*), che rappresenta un singolo elemento di *parGray*. Se due o più punti consecutivi di *parRow* hanno valori con differenza minore o uguale alla soglia impostata, allora essi vengono sostituiti dal solo ultimo punto della serie, al quale viene assegnato il valore di grigio medio fra quelli dei set-point considerati. Nel caso in cui, invece, almeno il 70% della riga sia costituito da punti bianchi, l'intera riga viene considerata bianca. La stessa logica viene applicata per il nero, poiché, in entrambi i casi, avrebbe poco senso sostituire le tonalità più nette della scala di grigi con valori di poco differenti. Nel caso del bianco, tra l'altro, ciò implicherebbe dover lavorare una zona che, originariamente, sarebbe dovuta rimanere non marcata, con possibili conseguenze negative sul risultato finale.
L'output della funzione è ancora una matrice a due colonne, ma con numero di righe $r_i < n_i$, dove n_i è il numero di punti contenuti in *parRow*.

Algoritmo 4.12: GCodeFull

Input: Gray, parGray, speed, drawingOrigin, thresh
if *non esiste un file denominato 'GCode_temp.txt' nella cartella di lavoro* **then**
    crea il file;
rendi il file scrivibile;
rendi il file invisibile nella cartella di lavoro;
scrivi nel file: 'G0 F(speed(1)) – G1 F(speed(2)) – G28 X(drawingOrigin(1)) Y(drawingOrigin(2)) – M4 S0' (trattino = inizio di una nuova riga del file);
m = numero di righe di Gray;
for *ogni elemento i = 1:m di Gray* **do**
    Row = Gray{i} (vettore $n_i \times 3$);
    parRow = parGray{i} (vettore $n_i \times 2$);
    x0 = Row(1,1);
    y0 = Row(1,2);
    x1 = Row(ultima riga,1);
    y1 = Row(ultima riga,2);
    scrivi nel file: 'G11 X(x0) Y(y0) X(x1) Y(y1) –';
    parRow2 = simplifyGCodeFull(parRow,thresh) (matrice $r_i \times 2$);
    v = prima colonna di parRow2 trasposta (vettore $1 \times r_i$);
    p = trasposta di (255 - parRow2(:,2)) (vettore $1 \times r_i$);
    scrivi nel file le r_i righe: '(v) S(p) –';
scrivi nel file: 'M5';
chiudi il file;

Si fa notare come la macro G11 preveda che la luminosità del primo punto della riga (di coordinata parametrica 0) venga ignorata. Per questo motivo, il punto iniziale viene automaticamente escluso da *parOut*.

Al livello superiore dell'implementazione, *GCodeFull* (algoritmo 4.12) crea un file di testo nella directory in cui viene installato il software, all'interno del quale scrive il part program, sulla base delle informazioni contenute in *Gray* e *parGray*. Come si vedrà anche nel capitolo successivo, questo file, che è temporaneo ed è mantenuto nascosto, viene eliminato alla chiusura dell'applicazione. Nel caso in cui l'utente voglia salvarlo, esso viene copiato, rinominato e memorizzato in una cartella selezionabile tramite una finestra di dialogo.
Alla riga 5 viene compilata la parte introduttiva del part program. Alle righe 14 e 18 si trovano, rispettivamente, l'inizio di ciascun comando G11 e la lista di coordinate parametriche e valori di potenza associati. Si noti come la scala di grigi sia invertita rispetto a quella di potenza (riga 17). Infine, il file viene completato con lo spegnimento del laser alla riga 19.

Algoritmo 4.13: inverseParametrize

Input: coords, mat
1 x0 = mat(1,1);
2 y0 = mat(1,2);
3 x1 = mat(2,1);
4 y1 = mat(2,2);
5 n = numero di righe di coords;
6 inizializza out (vettore n×3);
7 out(:,1) = (x1 - x0)*coords(:,1) + x0;
8 out(:,2) = (y1 - y0)*coords(:,1) + y0;
9 out(:,3) = coords(:,2);
Output: out

Algoritmo 4.14: GCodePlot

Input: ax, input, Dbeam
1 m = numero di righe di elementi della cell array input;
2 elimina eventuali plot precedenti da ax;
3 **for** *ogni elemento i = 1:m di input* **do**
4 row = input{i};
5 n = numero di righe di row;
6 **for** *gli elementi j = 1:n-1 di row* **do**
7 **if** *row(j+1,3) ≠ 255* **then**
8 plotta su ax un segmento di spessore Dbeam che congiunge i punti di coordinate row(j,1:2) e row(j+1,1:2), con colore dato dal valore row(j+1,3) in scala di grigi;

Algoritmo 4.15: GCodeParserFull

```
Input: ax, flipRow, Dbeam
apri in sola lettura il file 'GCode_temp.txt' nella cartella di lavoro;
newline = riga del file di testo corrispondente al primo comando G11;
inizializza parserCell (cell array 1×1);
inizializza j = 0 (contatore delle righe di parserCell);
inizializza k = 0 (contatore delle righe di coords);
while newline non comincia con 'M5' do
    if newline comincia con 'G11' then
        x0 = coordinata x del punto di inizio riga;
        y0 = coordinata y del punto di inizio riga;
        x1 = coordinata x del punto di fine riga;
        y1 = coordinata y del punto di fine riga;
        inizializza coords = [0,0];
        k = 0;
    else
        x = valore di coordinata parametrica alla riga newline;
        g = 255 - (valore di potenza S alla riga newline);
        k = k + 1;
        coords(k,:) = [x,g];
        if coords(k,1) = 1 then
            v = [x0,y0;x1,y1];
            j = j + 1;
            coords2 = [0,coords(1,2);coords];
            if flipRow(j) = 1 then
                v = v ribaltato verticalmente (prima riga diventa ultima e
                  viceversa);
                coords2(:,1) = 1 - coords2(:,1);
                coords2 = coords2 ribaltato verticalmente;
                row = inverseParametrize(coords2,v);
                row2 = row ribaltato verticalmente;
            else
                row2 = inverseParametrize(coords2,v);
            parserCell{j,1} = row2;
    newline = riga successiva a newline nel file di testo;
GCodePlot(ax,parserCell,Dbeam);
chiudi il file;
```

4.4 Parsing grafico

La sezione che si occupa di interpretare e produrre un plot del GCode è gestita da *GCodeParserFull*, che riceve in input *Dbeam*, il vettore *flipRow* e l'oggetto *ax*, il quale identifica gli assi cartesiani sui quali visualizzare l'anteprima grafica. Questa parte del codice impiega due sotto-funzioni:

- ***inverseParametrize*** (algoritmo 4.13): i suoi input sono una matrice $n \times 2$ (*coords*) di coordinate parametriche e una 2×2 (*mat*), in cui ciascuna riga

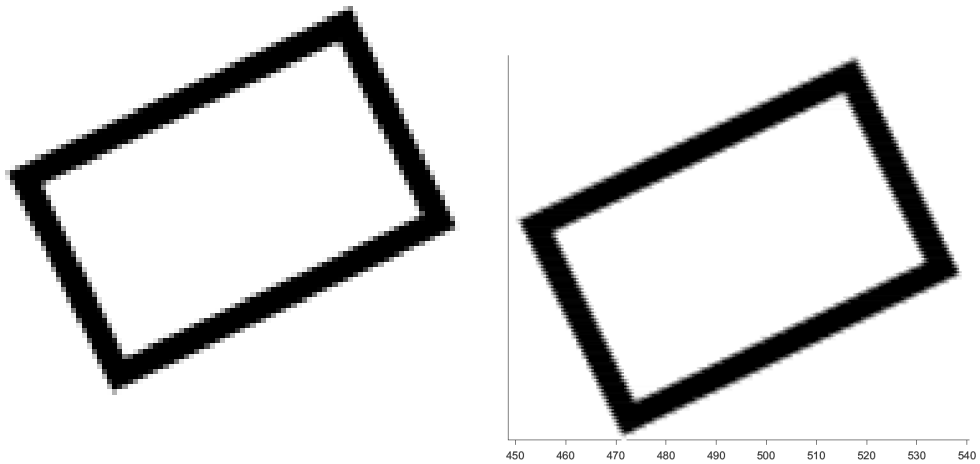
contiene le coordinate (x, y) dei punti di inizio e fine di una linea di scansione. La funzione converte, per interpolazione lineare, le coordinate parametriche di *coords* in quelle cartesiane del vettore *out*, associando a ciascun punto il corrispondente valore di grigio.

- ***GCodePlot*** (algoritmo 4.14): ricava un grafico a partire da una cell array di coordinate e valori di grigio, in cui ogni punto di una riga di scansione è collegato al successivo con un segmento grigio, la cui luminosità è pari a quella del secondo dei due punti.

Lo pseudocodice del parser è riportato, invece, nell'algoritmo 4.15. La funzione legge le informazioni contenute nel file di testo creato da *GCodeFull* e ne estrae dei valori che vengono raggruppati nella cell array *parserCell*, la quale viene, infine, plottata sugli assi *ax* (riga 33). Si fa notare che il valore di grigio assegnato al punto iniziale della linea di scansione (riga 22) è totalmente arbitrario, poiché, in base a quanto già visto, il GCode trascura la luminosità di quel punto.

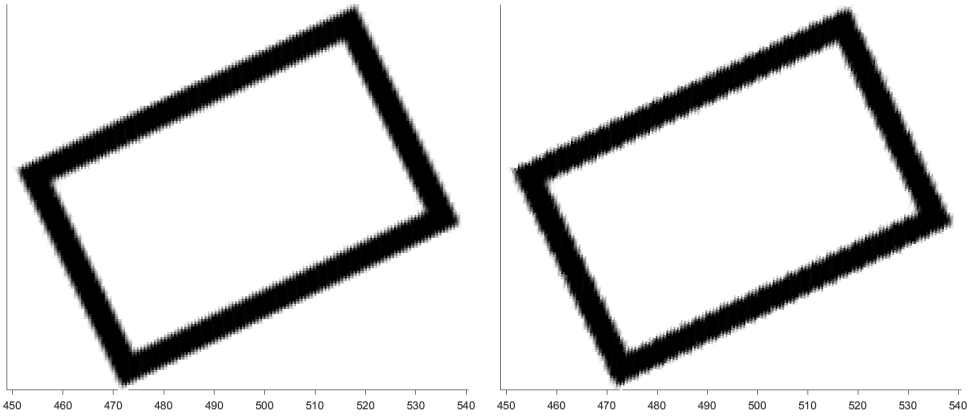
4.5 Commento dei risultati

La figura 4.5 mostra l'elaborazione di una semplice immagine, realizzata secondo tre diverse direzioni di scansione: orizzontale, verticale e diagonale a 85°. Per tutte e tre la soglia *thresh* è impostata al valore intermedio di 30. È evidente che, essendo le tre immagini di partenza diverse, in quanto ruotate di angoli differenti, ed essendo la rotazione e il resizing due operazioni di interpolazione, alcune piccole differenze sono riscontrabili nel risultato finale. La resa grafica più fedele all'originale è data dalla scansione orizzontale (fig. 4.5b), poiché essa non richiede la rotazione preventiva della bitmap appena citata, ma solo lo scaling.
Al di là di alcuni piccoli dettagli, comunque, il risultato complessivo, per immagini semplici come quella di figura 4.5a, è piuttosto soddisfacente e rimane relativamente invariato se *thresh* cambia valore.



(a) Immagine originale (b) Scansione a 0°

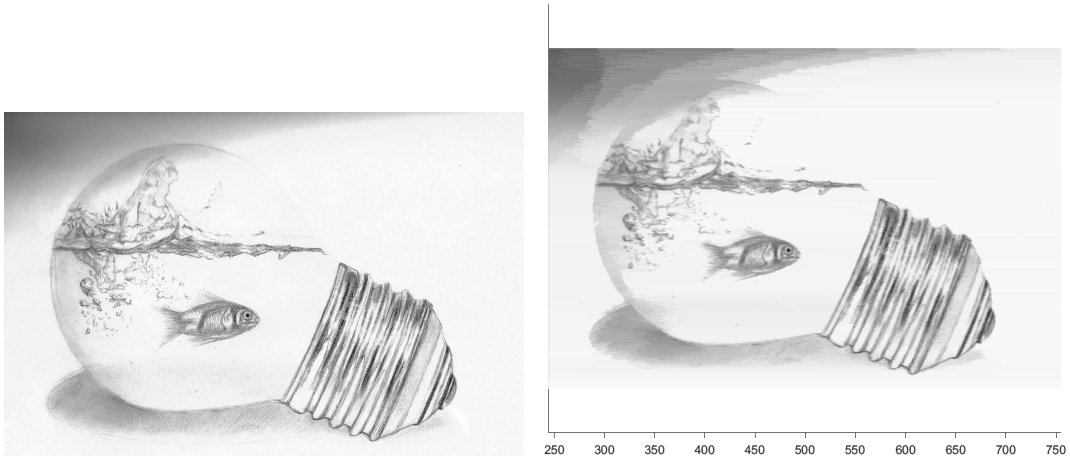
Figura 4.5: Esempio di elaborazione di un'immagine da parte del software secondo diverse direzioni di scansione.



(c) Scansione a 90° (d) Scansione a 85°

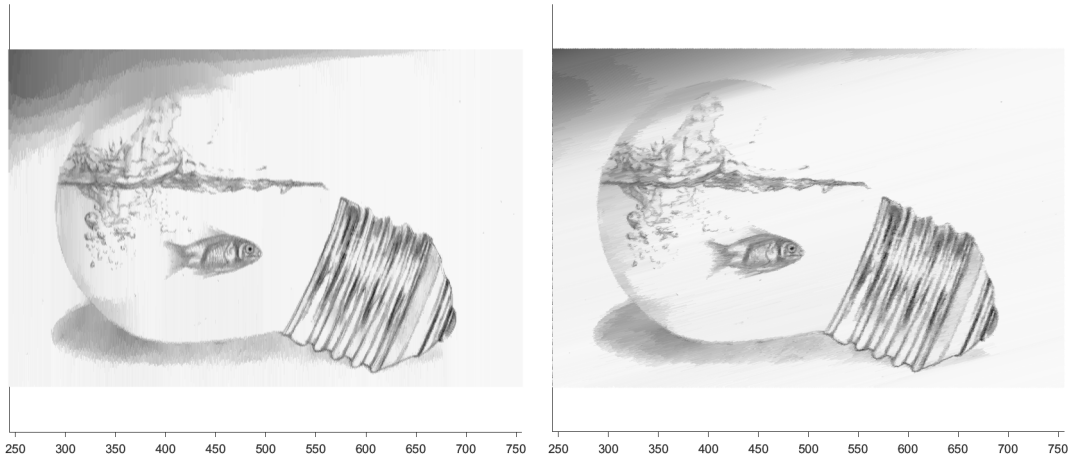
Passando, invece, a bitmap più complesse, la direzione di scansione e la soglia del GCode tendono a influire in maniera significativa sulla resa finale. In figura 4.5, ad esempio, si nota come la scansione verticale dia risultati nettamente peggiori rispetto a quella orizzontale o diagonale, a parità di *thresh*. Queste osservazioni non possono essere, tuttavia, generalizzate a qualunque immagine e vanno, perciò, riformulate caso per caso.

La figura 4.6 mostra le differenze provocate da un diverso valore di *thresh* sull'immagine processata, che è ancora quella di figura 4.5a (coincidente col risultato se $thresh = 0$). Oltre al risultato grafico, è importante che l'utente consideri la differenza, in termini di numero di righe, tra i file di testo contenenti il part program, in quanto essa dà un'idea qualitativa della complessità del GCode generato dal software nei vari casi. In base all'istogramma di figura 4.5, al crescere di *thresh* l'andamento del numero di righe del file di testo è approssimativamente di tipo esponenziale decrescente. In altri termini, è sufficiente imporre bassi valori di soglia per ottenere un drastico calo del numero di set-point del GCode. Nell'esempio considerato, inoltre, il numero di righe di codice varia di due ordini di grandezza lungo l'intera scala di valori che *thresh* può assumere.



(a) Immagine originale (b) Scansione a 0°

Figura 4.5: Effetto della direzione di scansione sul risultato finale.



(c) Scansione a 90° (d) Scansione a 15°

L'algoritmo di semplificazione del GCode è evidentemente rudimentale, ma si potrebbe pensare di migliorarlo implementando un'ulteriore sezione del codice che riduca le transizioni nette di luminosità, creando una maggior continuità nei valori di grigio in zone che rispettano alcune condizioni prestabilite.

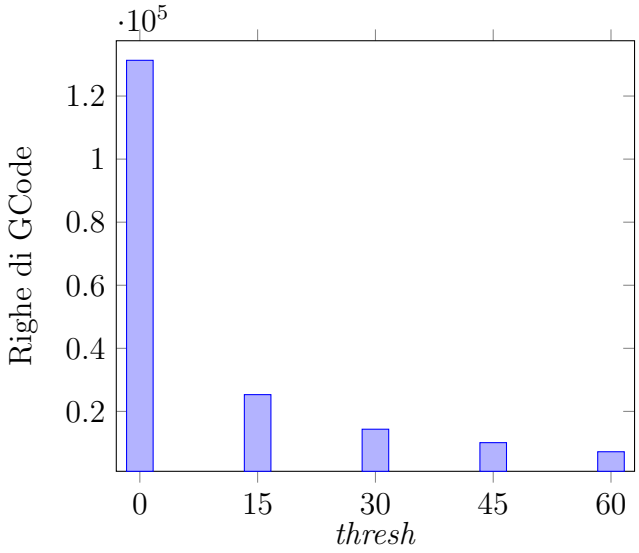
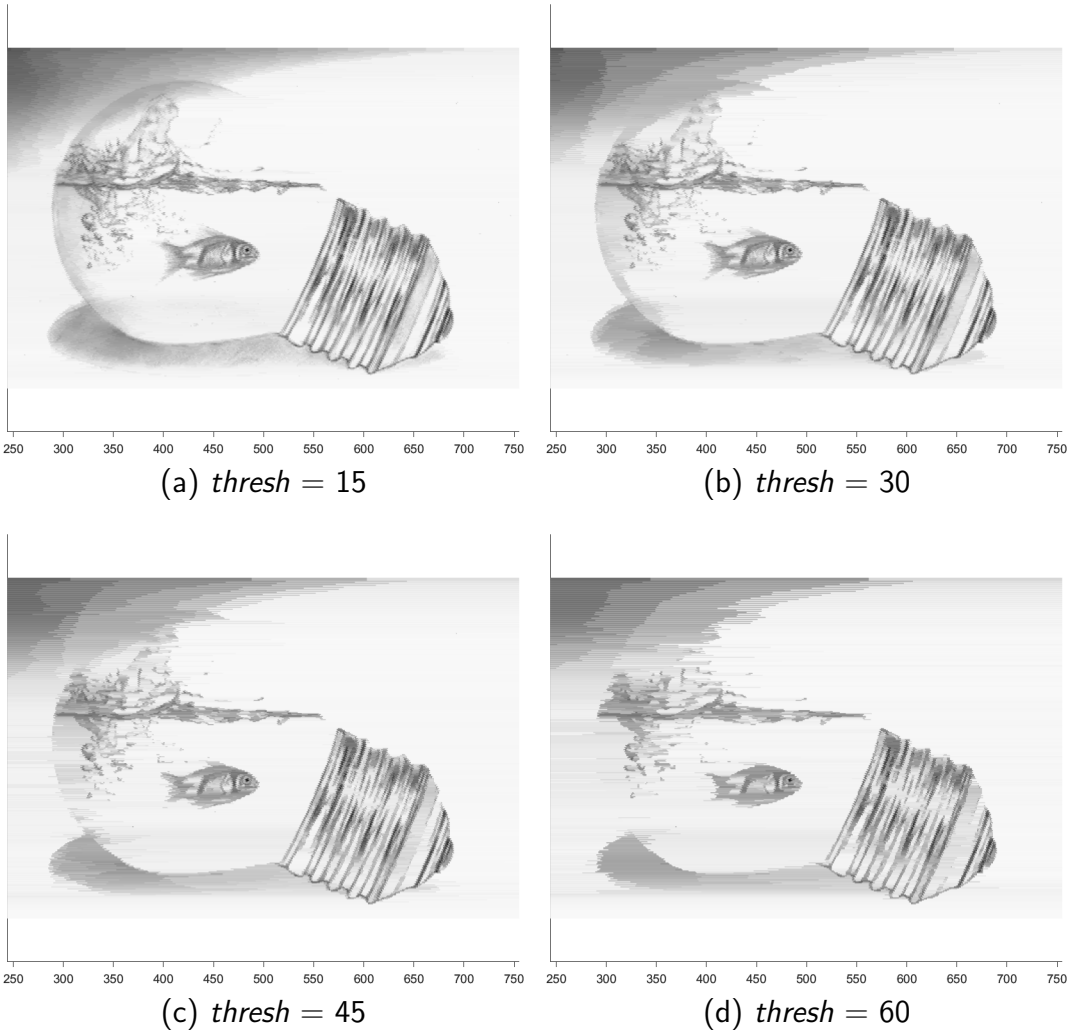


Figura 4.5: Numero di righe del GCode generato dal software a fronte di diversi valori di *thresh*. Le immagini di riferimento sono quelle di figura 4.6.

La logica di ottimizzazione dell'angolo di scansione in due stadi si rivela, all'atto pratico, ben posta, dal momento che, nella totalità dei casi testati, è corretto assumere che gli angoli nell'intorno di quello ottimale portino a ottenere, man mano che ci si avvicina a quest'ultimo, tempi ciclo sempre più bassi. È quindi sensato individuare, nel primo stadio, un valore di tentativo attorno al quale approfondire, solo in una seconda fase, la ricerca dell'ottimo. Lo stesso algoritmo risulta, tuttavia, abbastanza lento e macchinoso nel caso in cui l'immagine sia costituita da un numero elevato di punti. Questo aspetto era comunque prevedibile dal fatto che il software è basato su una serie di procedure brute-force, i cui tempi di calcolo crescono in modo considerevole all'aumentare delle soluzioni alternative da confrontare.
Una criticità di questo algoritmo è quella relativa ai criteri geometrici di filtraggio dei



(a) *thresh* = 15 (b) *thresh* = 30

(c) *thresh* = 45 (d) *thresh* = 60

Figura 4.6: Differenze provocate da diversi valori della soglia di differenza di grigio ammissibile nel GCode. L'immagine di riferimento è quella di figura 4.5a.

possibili angoli di scansione. Essi tendono, infatti, a rendere difficile l'accettabilità di angoli lontani da 0° e 90°. La scansione orizzontale e quella verticale, in effetti, costituiscono dei casi particolari, per i quali il resizing dell'immagine è calcolato in modo esatto. In questo senso, esse rispettano sempre le condizioni di filtraggio, come già si è visto, mentre, allontanandosi verso angoli di modulo nell'intorno dei 45°, diventa sempre più difficile soddisfare le relazioni 4.15 e 4.16.

Un ultimo aspetto che si sottolinea riguarda la lentezza del parser grafico e, in particolare, della funzione *GCodePlot*, specialmente all'aumentare del numero di punti e al diminuire di *thresh*.

Capitolo 5

L'interfaccia grafica

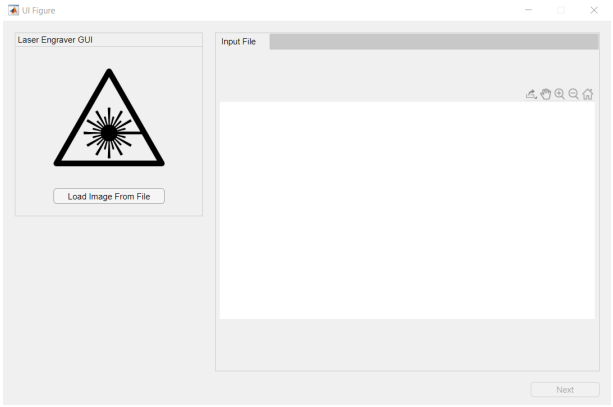
Il software descritto nel capitolo 4 costituisce, come già anticipato, il back-end di un'applicazione *stand-alone*, che è corredata di un'interfaccia utente. Grazie a quest'ultima è possibile selezionare l'immagine da elaborare, effettuare operazioni di pre-processing, inserire i dati geometrici e tecnologici necessari alla lavorazione e, infine, generare il GCode e valutarne un'anteprima grafica.
In questo breve capitolo conclusivo viene offerta una descrizione della GUI e delle sue funzionalità.

5.1 Descrizione delle funzionalità

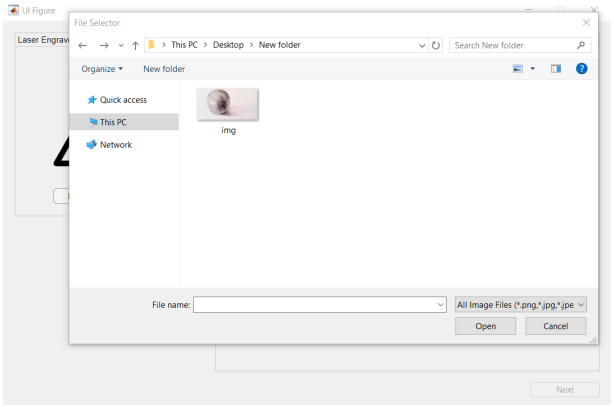
L'interfaccia è stata sviluppata tramite App Designer, l'ambiente di sviluppo di GUI integrato in MATLAB. Esso permette di gestire in modo semplice e intuitivo l'aspetto grafico, associando a ciascun elemento (pulsante, slider, menù a tendina ecc...) delle righe di codice che entrano in funzione nel momento in cui l'utente interagisce col componente stesso.
Se si escludono le finestre di dialogo e quelle legate a salvataggi e caricamenti, la GUI è composta da un'unica finestra, sulla quale si alternano quattro differenti schermate. Questa scelta consente di velocizzare l'applicazione e rendere più fluide le transizioni, in quanto l'apertura e la chiusura di finestre è un'operazione molto più lenta rispetto alla comparsa e scomparsa di singoli elementi grafici. La dimensione dell'interfaccia è fissata a 547×866 pixel e non è modificabile, dal momento che il resizing è molto complesso da gestire in App Designer. Integrare delle logiche di scaling avrebbe richiesto, infatti, tempi di sviluppo del codice non compatibili con la durata complessiva dell'attività di tesi, quindi si è deciso di dimensionare la GUI assumendo che l'utente impieghi uno schermo full HD (1080×1920 pixel).

Una volta avviata l'applicazione (fig. 5.1a), per proseguire è necessario caricare una bitmap da un file di formato JPG/JPEG o PNG, cliccando sull'apposito pulsante e selezionando l'immagine da una nuova finestra (fig. 5.1b). Fatto ciò, viene mostrata un'anteprima della figura sulla destra e il pulsante 'Next' diventa attivo (fig. 5.1c), consentendo di procedere alla seconda schermata (fig. 5.2a), che è dedicata alle operazioni di pre-processing.
I parametri controllabili in questa fase sono:

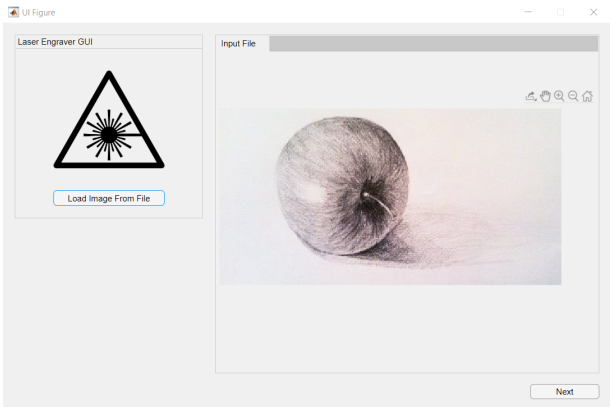
- la modalità di conversione in scala di grigi, secondo i metodi elencati nella sezione 3.1;



(a) Aspetto iniziale



(b) Selezione del file

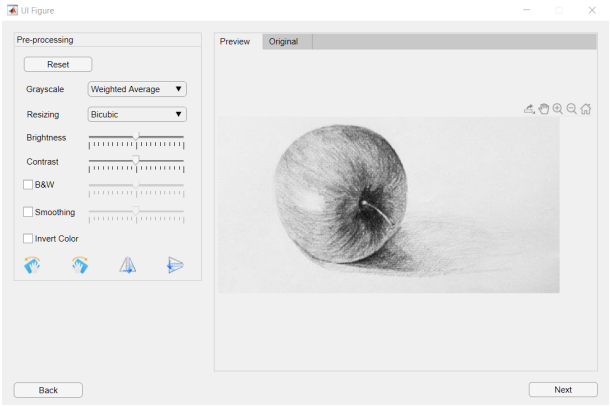


(c) Anteprima del file caricato

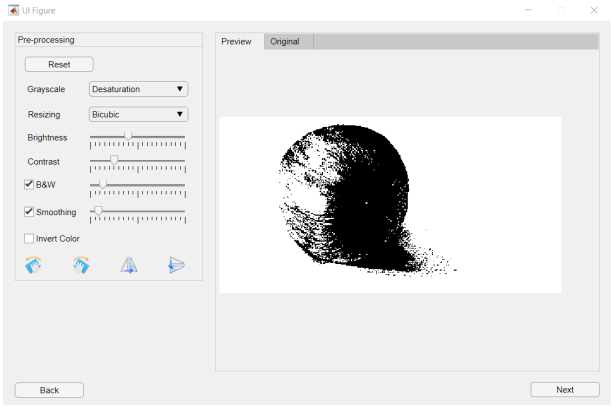
Figura 5.1: Prima schermata dell'interfaccia.

- il metodo di interpolazione per il resizing, a scelta fra quelli descritti nella sezione 3.2.2.2;
- la luminosità, attraverso un comando di tipo additivo;
- il contrasto, attraverso un comando di tipo proporzionale;
- l'eventuale soglia per la conversione in bianco e nero (vedi sez. 3.1);
- la deviazione standard per il filtro gaussiano (vedi sez. 3.2.2.1), se desiderato;
- l'eventuale inversione di colore (NOT logico, vedi sez. 3.2.1);
- l'orientamento dell'immagine, che può essere ruotata, cliccando sulle icone in basso, in senso orario o antiorario e può essere specchiata orizzontalmente o verticalmente.

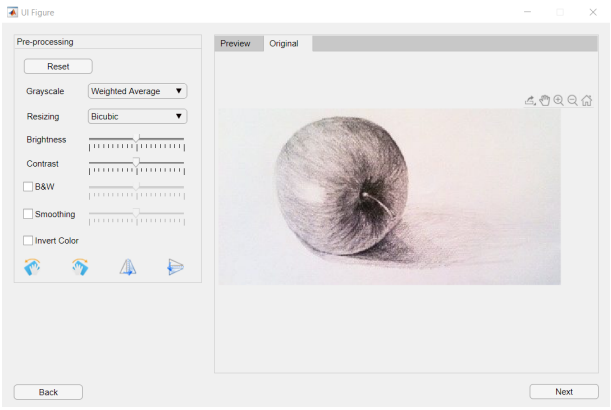
Variando una qualunque di queste scelte, l'anteprima grafica viene aggiornata in tempo reale, come da figura 5.2b, ed è possibile confrontarla con l'immagine originale, che è rappresentata nella seconda scheda del grafico sulla destra (fig. 5.2c). È possibile tornare alle impostazioni di partenza cliccando sul pulsante 'Reset'. Se si preme 'Back', invece, si torna alla situazione di figura 5.1a ed è necessario riselezionare la bitmap dalla finestra di dialogo apposita.



(a) Parametri di default



(b) Modifica dei parametri



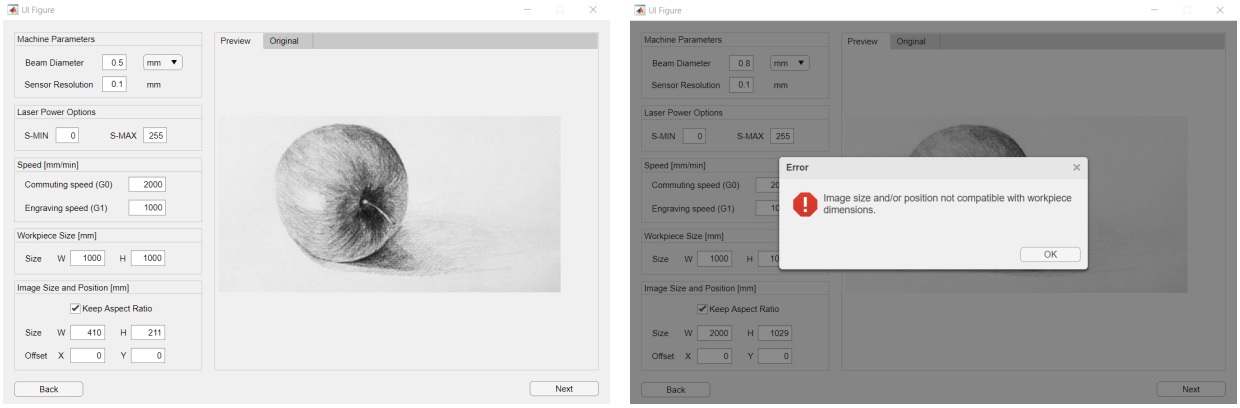
(c) Confronto con l'immagine originale

Figura 5.2: Seconda schermata dell'interfaccia.

Procedendo, invece, tramite 'Next', si apre la sezione dedicata all'input dei dati tecnologici (fig. 5.3a). Le grandezze da specificare sono:

- il diametro dello spot, la cui unità di misura (mm oppure μm) può essere scelta tramite un menù a tendina;
- la risoluzione del sensore di posizione;
- la potenza minima e massima del laser sulla scala a 8 bit ($0 \div 255$), che influenzano anche l'anteprima dell'immagine sulla destra;
- le velocità di avanzamento (G1) e di appostamento (G0);
- la dimensione del pezzo lavorato;
- la dimensione dell'immagine marcata sul pezzo, con la possibilità di mantenere il rapporto di aspetto originale tramite la spunta dell'apposita casella;
- l'offset orizzontale e verticale dell'immagine rispetto al centro del pezzo.

Se, dopo aver premuto 'Next', il sistema verifica che, in base alle dimensioni inserite, l'immagine è collocata (anche solo parzialmente) fuori dall'area del pezzo, un



(a) Input dei dati (b) Messaggio di errore

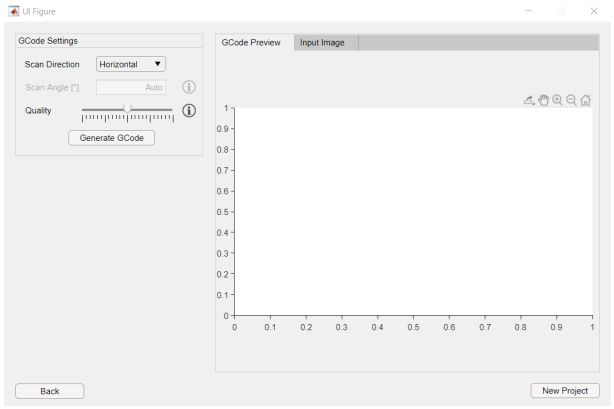
Figura 5.3: Terza schermata dell'interfaccia.

messaggio di errore compare in primo piano e impedisce di procedere finché non vengano adeguatamente corretti i dati di input (fig. 5.3b).

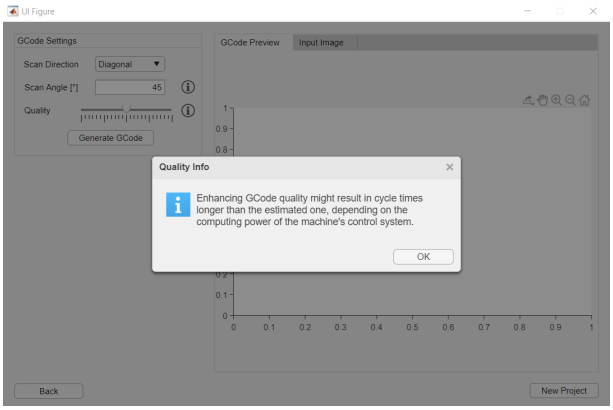
Una volta risolti eventuali problemi sui valori inseriti, cliccando su 'Next' si apre l'ultima schermata (fig. 5.4a), che permette di scegliere la direzione di scansione (orizzontale, verticale, diagonale oppure ottimizzata) e la qualità del GCode. In caso di scansione diagonale, viene attivato un ulteriore campo di modifica, relativo all'inserimento manuale dell'angolo di scansione, come da scenario B (vedi sez. 4.2). Sono presenti, inoltre, due icone interattive: una volta cliccate, si apre una finestra di dialogo che fornisce ulteriori informazioni sulla qualità (fig. 5.4b) e sulla designazione dell'angolo (fig. 5.4c), rispettivamente.
Il valore dello slider 'Quality' rappresenta il complemento a 60 del parametro *thresh*, introdotto nella sezione 4.3, il quale regola il numero di set-point del GCode lungo una riga dell'immagine. Se l'unità di controllo della macchina non è in grado di gestire il processo alla velocità assegnata al comando G1, può essere che la lavorazione venga automaticamente rallentata, in modo che si riesca comunque a garantire la qualità richiesta (in termini di numero di set-point). Per quel che riguarda l'angolo di scansione, poi, esso viene scelto sulla base dei criteri geometrici inclusi nella funzione *pickAngles* (algoritmo 4.8) e, come noto, potrebbe non coincidere con quello richiesto dall'utente.

Se si preme su 'Generate GCode', vengono eseguite in sequenza le funzioni *ImageProcessingFull* (algoritmo 4.10) e *GCodeFull* (algoritmo 4.12), che prendono in input i valori forniti nella terza schermata e l'immagine già processata nella seconda. Nella prima delle due funzioni, la bitmap viene convertita in una traiettoria di righe di scansione, mentre, nella seconda, i punti di tale tragitto vengono utilizzati per la scrittura del part program su un file di testo temporaneo. Durante l'esecuzione di queste due sezioni di back-end, una barra di caricamento avvisa del calcolo in corso (fig. 5.4d).
Terminato il processo, compare sull'interfaccia un ulteriore pannello grafico, che mostra l'angolo di scansione risultante dall'algoritmo e il tempo ciclo stimato (fig. 5.4e), il cui calcolo è basato sull'assunzione di velocità costante e pari a quella assegnata a G1 (vedi sez. 4.2.3). Da questo momento in poi, se si clicca su 'Back' si apre una finestra (fig. 5.4f) che avvisa l'utente della perdita del GCode generato, nel caso in cui si voglia confermare di tornare alla schermata precedente.

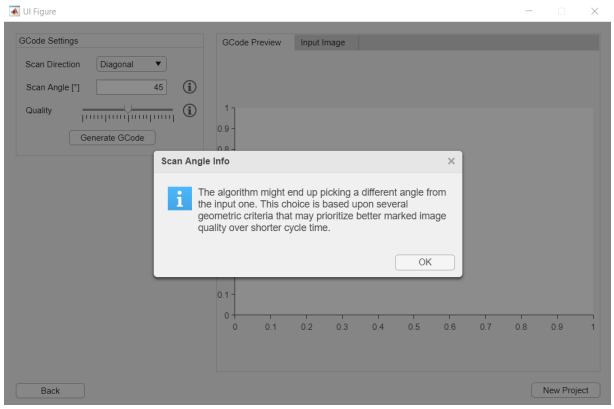
Il pulsante 'GCode Preview' permette l'elaborazione dell'anteprima grafica del part program (fig. 5.4g), tramite l'esecuzione di *GCodeParserFull* (algoritmo 4.15). Anche in questo caso, il processo di calcolo è accompagnato da una barra di caricamento (fig. 5.4h). Il file di testo temporaneo, creato dalle funzioni di back-end e utilizzato per generare l'anteprima, può essere copiato, rinominato e salvato in una cartella a scelta cliccando su 'Save GCode To File' (fig. 5.4i).



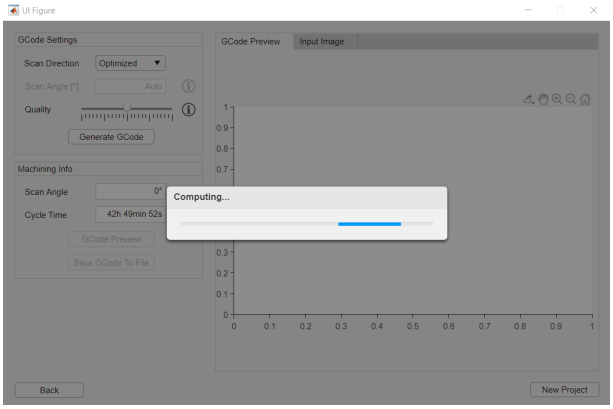
(a) Aspetto iniziale



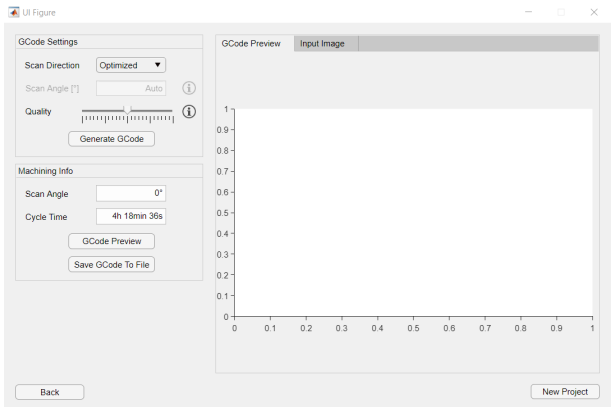
(b) Informazioni sullo slider 'Quality'



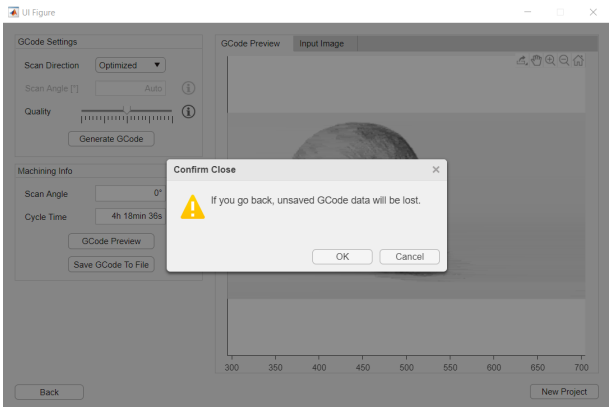
(c) Informazioni sull'angolo di scansione



(d) Barra di caricamento per la generazione del GCode

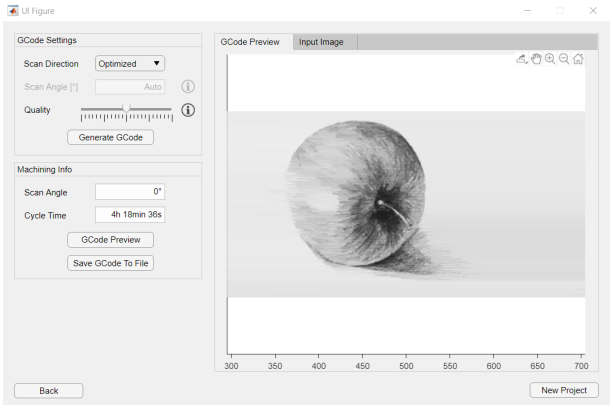


(e) Aspetto dopo la generazione del GCode

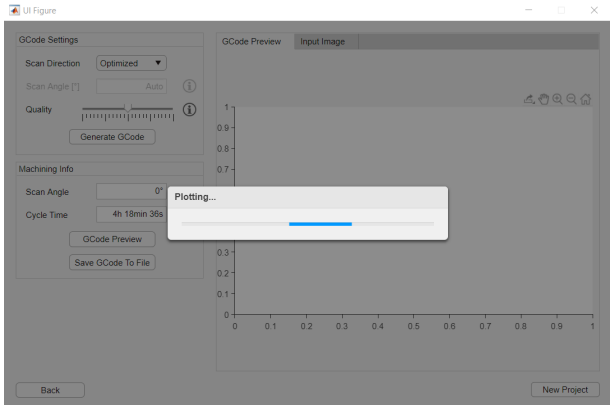


(f) Finestra di dialogo associata a 'Back'

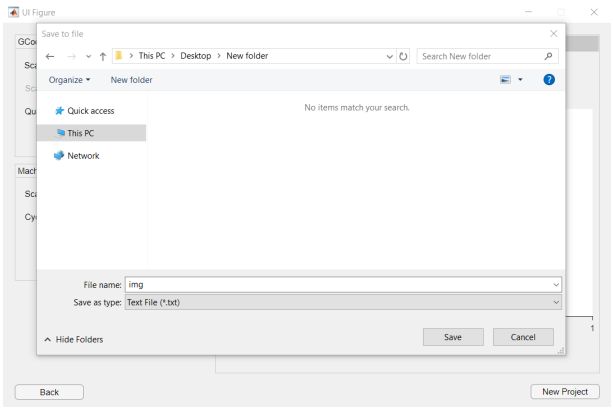
Figura 5.4: Quarta schermata dell'interfaccia.



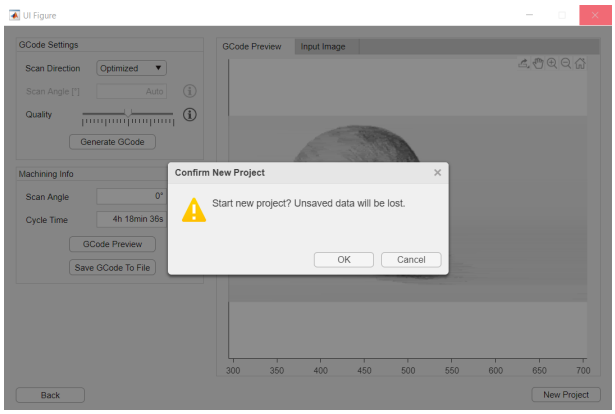
(g) Anteprima grafica del GCode



(h) Barra di caricamento per il parsing grafico del GCode



(i) Salvataggio del GCode su file di testo



(j) Finestra di dialogo associata a 'New Project'

Si sottolinea come, a seconda della directory nella quale è stata installata, l'applicazione potrebbe necessitare di essere eseguita con diritti da "amministratore", poiché, in caso contrario, risulta impossibile la creazione di alcuni file temporanei fondamentali per il corretto funzionamento del software.

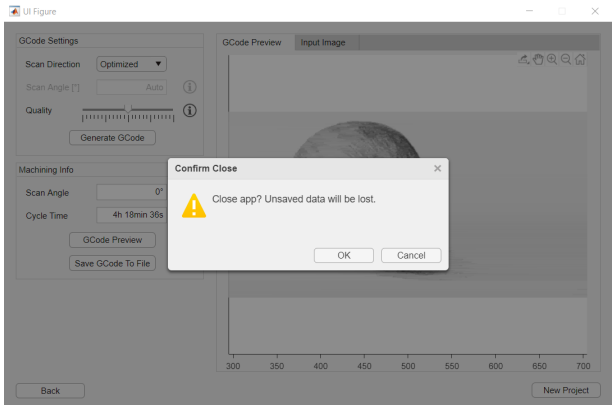


Figura 5.4: Finestra di dialogo per la conferma della chiusura dell'applicazione.

'New Project' consente di tornare alla schermata di partenza, dopo aver confermato la scelta attraverso una finestra di dialogo (fig. 5.4j). In modo analogo, infine, se l'utente chiude la finestra dell'applicazione (da una qualunque delle quattro scher-

mate), un messaggio di conferma avvisa della perdita di eventuali dati non salvati (fig. 5.4).

Conclusioni

Questa attività di tesi è nata con l'obiettivo di sviluppare un pacchetto software per la pianificazione di traiettoria di un robot a cavi. Il prototipo del manipolatore è stato realizzato nell'ambito del progetto Laser Engraver del GRAB (Group of Robotics, Automation and Articular Biomechanics) dell'Università di Bologna. Per contestualizzare la macchina e il processo tecnologico, sono state approfondite nozioni riguardanti la marcatura laser e la robotica a cavi, che hanno consentito di focalizzare il lavoro su aspetti più mirati.
Ci si è concentrati, infatti, sulla marcatura a scansione, la quale prevede che il laser realizzi l'immagine riga per riga, variando puntualmente il valore della potenza per agire sulla profondità del solco marcato. Gli algoritmi di image processing implementati hanno permesso di tradurre l'informazione contenuta nella bitmap di partenza in blocchi di GCode. È stato studiato, poi, il problema dell'ottimizzazione della traiettoria, che, nell'ottica di determinare il percorso di lavorazione più breve, ha portato a sviluppare un'euristica brute-force a due step: dopo aver calcolato l'angolo di scansione ottimale, l'algoritmo sceglie il punto di inizio della lavorazione che minimizza i moti di appostamento del laser. Si è riscontrato che le logiche implementate forniscono risultati soddisfacenti, seppure nei limiti evidenziati. L'interfaccia grafica, inoltre, consentirà all'utente, quando il prototipo sarà in funzione, di poter usufruire del software tramite una semplice applicazione stand-alone, nell'ambito della quale è possibile specificare i dati di processo e le preferenze che influenzano il risultato finale.

Bibliografia

[1] https://tannerhelland.com/2011/10/01/grayscale-image-algorithm-vb6.html.

[2] S. Alatartsev et al. «On optimizing a sequence of robotic tasks». In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 217–223.

[3] *Automation systems and integration – Numerical control of machines – Program format and definitions of address words – Part 1: Data format for positioning, line motion and contouring control systems.* Norma ISO 6983-1:2009. Geneva, CH: International Organization for Standardization, 2009.

[4] P. Bosscher, R. L. Williams e M. Tummino. «A Concept for Rapidly-Deployable Cable Robot Search and Rescue Systems». In: *ASME. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Volume 7: 29th Mechanisms and Robotics Conference, Parts A and B*. 2005.

[5] W. G. Brown. «Suspension system fot supporting and conveying equipment, such as a camera». US patent 4,710,819. 1987.

[6] G. Campana e M. Mele. *Sistemi integrati di lavorazione. Appunti delle lezioni, esercizi risolti e domande per l'autovalutazione.* A cura di S. E. Esculapio. Società Editrice Esculapio, 2019.

[7] E. Capello. *Le lavorazioni industriali mediante laser di potenza. La tecnologia, le applicazioni e i sistemi.* A cura di M. Editore. Maggioli Editore, 2009.

[8] D. De Zeeuw, S. A. J. Spanjer e K. Zeggelaar. «Cleaning system for a façade of a building structure». WO 2016/085330. 2016.

[9] R. Dewil et al. «A Critical Review of Multi-hole Drilling Path Optimization». In: *Archives of Computational Methods in Engineering* (gen. 2018). DOI: 10.1007/s11831-018-9251-x.

[10] S. Fang et al. «Motion control of a tendon-based parallel manipulator using optimal tension distribution». In: *IEEE/ASME Transactions on Mechatronics* 9.3 (2004), pp. 561–568.

[11] R. C. Gonzalez e R. E. Woods. *Digital Image Processing*. Prentice Hall, 2008.

[12] D. Han. «Comparison of Commonly Used Image Interpolation Methods». In: *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering* (mar. 2013).

[13] D. Johnson e L. A. McGeoch. «The Traveling Salesman Problem: A Case Study in Local Optimization». In: 2008.

[14] M. Kubo e H. Kasugai. «THE PRECEDENCE CONSTRAINED TRAVELING SALESMAN PROBLEM». In: *Journal of the Operations Research Society of Japan* 34.2 (1991), pp. 152–172. DOI: 10.15807/jorsj.34.152.

[15] R. Matai, S. Singh e M. Mittal. «Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches». In: nov. 2010. ISBN: 978-953-307-426-9. DOI: 10.5772/12909.

[16] A. Ming e T. Higuchi. «Study on multiple degree-of-freedom positioning mechanism using wires, (part 1) - Concept, design and control». In: *International Journal of the Japan Society for Precision Engineering, 28(2):131-138* (1994).

[17] C. Patel, A. Patel e R. Patel. «A Review on Laser Marking Process for Different Materials». In: *International Journal for Scientific Research & Development* 5.1 (2017).

[18] A. Pott. «Influence of Pulley Kinematics on Cable-Driven Parallel Robots». In: *Latest Advances in Robot Kinematics*. A cura di J. Lenarcic e M. Husty. Dordrecht: Springer Netherlands, 2012, pp. 197–204.

[19] A. Pott. *Cable-driven Parallel Robots: Theory and Application*. A cura di B. Siciliano e O. Khatib. Springer, Cham, mag. 2018.

[20] C. Solomon e T. Breckon. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. Wiley-Blackwell, 2010.

[21] R. Verhoeven. «Analysis of the Workspace of Tendon-based Stewart Platforms». Tesi di dott. University of Duisburg-Essen, 2004.