Alma Mater Studiorum · Università di Bologna

CAMPUS DI CESENA

Dipartimento di Informatica - Scienza e Ingegneria

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

STUDIO E REALIZZAZIONE DI UN PLUGIN PER L'ALLINEAMENTO DI IMMAGINI MICROSCOPICHE

Tesi in
Web Semantico

Relatore:
Antonella Carbonaro

Presentata da:
Stefano Belli

Correlatore:
Filippo Piccinini

Sessione
2018/2019

PAROLE CHIAVE
Istologia
Microscopia
Immunoistochimica sequenziale
Allineamento roto-traslativo
ImageJ Plugin

Introduction

Immunoistochemistry is a diagnostic technique used by anatomo pathologists to identify the structure of the cells when morphologyc analisys are not sufficient. The tecnique is based on the study of the protein expression in the tissues; thanks to an antigen-antibody reaction it is possible to find out whether the target protein is present in the analyzed tissue and where it is located (cell membrane, cytoplasm, nucleus, extra-cellular tissues). Immunoistochemistry, exploiting tissue-specific markers, is largely used to identify malignant tumors (f.i. lymphoma, carcinomas, melanomas), to detect neoplasms (f.e. lymphomas or leukemia), to discover the origin of the metastasis and to identify microbial agents such as viruses, bacteria and protozoa. This technique it's also useful to collect diagnostic data and therapeutic response forecast data concerning neoplasms treatment (breast carcinomas, lymphomas).

Human tissues and cells are primarily made of water and they commonly appear transparent to human sight. For this reason, colouring agents are usually used to outline their target components. However, those markers particles are usually bigger than the cellular components, making impossible for the biologist to add many of in a single step of the analysis. The standard procedure expect to apply sequentially the immunoistochemical tecnique, coloring the sample with a restrict number of markers, capturing pictures with the help of an optical microscope and then washing away these markers in order to repeat the above steps with a new set of dyes. This iterative procedure allows a precise analysis of the characeristics of the tissue and the subcellular area. However, in order to get a complete overview of the sample, the images needs to be aligned by the same reference point. This task can be difficult as the sample can suffer elastic or even permanent deformations during the different steps of the operation. To date, a gold standard software able to perform images sequence alignment (considering different resolutions and sizes of the picture) is yet to be found.

The aim of this thesis project is to fill this gap by developing a user-friendly application which allows doctors and biologists to align histological examination images of patients in order to value the cross-correlation between different cancer markers.

The project was developed within the Research Group "Data Science for Health" (DS4H), which is working to provide new methods and technologies to solve current problems in the

world of health and healthcare, in cooperation with “Istituto Scientifico Romagnolo per lo Studio e la cura dei Tumori” (IRST) based in Meldola (FC). This work led to the creation of “DS4H Image Alignment”, an open-source application which allows to overlay images of a sample tissue captured in different times and with different methods. Thanks to this application, clinical specialists can conduct histological studies otherwise hardly achievable.

This dissertation is structured in this way: Chapter 1 describes the most known methods to capture images in Light/Optical Microscopy and some the most common techniques of cellular staining and shows off some of the issues frequently encountered during the application of those operations, such as deformation and misalignment of slides due to the treatments. Chapter 2 introduces some widespread co-registration algorithms used in microscopy and the mathematical principles behind them. Some of those algorithms will be later used by “DS4H Image Alignement” software tool. Chapter 3 focuses on the development of plugins for ImageJ, which is the most common open-source software suite for image processing among life sciences researchers. It summarizes the principal developments of JAVA applications compatible with ImageJ, exploring above all SciJava, the framework on which it is based. Chapter 4 explain the main features of “DS4H Image Alignment” and its architectural structure, focusing on his working principles and how the functional requirements of the customer are fulfilled. Finally, some critical issues of the application are presented, explaining how part of them have been solved, and some feasible solutions to the residual problems are provided, too. Final considerations are included in this dissertation final chapter, alongside some ideas of improvements for the usability and efficiency of the program.

Introduzione

L'immunoistochimica è una tecnica utilizzata dall'anatomo-patologo per chiarire la natura delle strutture cellulari, laddove la pura morfologia risulti insufficiente, e lo fa tramite la valutazione dell'espressione di specifiche proteine presenti nei tessuti. Il metodo si basa sulla reazione antigene-anticorpo ed evidenzia se la proteina che target sia presente nel campione in esame e, se sì, dove essa sia localizzata(membrana cellulare, citoplasma, nucleo, tessuti extracellulari). Una delle applicazioni più diffuse dell'immunoistochimica è rappresentata dall'identificazione della natura di tumori maligni indifferenziati (es: linfomi, carcinomi, melanomi), nella caratterizzazione di neoplasie (es: linfomi, leucemie), nell'individuazione dell'origine di una metastasi, nell'identificazione di agenti microbici (es: virus, batteri, protozoi), il tutto utilizzando marcatori tessuto-specifici. Essa, inoltre, viene utilizzata nella produzione di dati di utilizzo prognostico e predittivo della risposta terapeutica per alcune neoplasie (es: carcinoma della mammella, linfomi).

Siccome tessuti e cellule umane sono tessuti principalmente composti da acqua, risultano tipicamente trasparenti all'osservazione ad occhio nudo. Per poter caratterizzare le cellule, quindi, vengono tipicamente utilizzati coloranti: tuttavia, a causa della dimensione ridotta delle componenti cellulari di interesse in confronto alle dimensioni delle particelle del colorante, non sempre è possibile utilizzare in parallelo un numero elevato di marcatori. Quindi la procedura standard prevede di ricorrere all'immunoistochimica sequenziale, dove si colora il campione con un numero limitato di marcatori, si acquisiscono immagini con ausilio del microscopio ottico, si eliminano i suddetti marcatori attraverso dei lavaggi, si evidenziano le cellule con un nuovo set di marcatori, si acquisiscono nuove immagini, etc. Questa procedura iterativa permette l'analisi accurata di varie caratteristiche del tessuto in esame e regioni sub-cellulari, in maniera sequenziale. Tuttavia, per avere una visione d'insieme del campione, occorre infine allineare le immagini per studiare la co-localizzazione dei segnali avendo un unico riferimento. Questo non è un compito semplice, perché durante le diverse fasi di colorazione il campione può subire deformazioni elastiche o addirittura permanenti. Oltretutto, ad oggi non esiste un software gold standard per allineare sequenze di immagini che potrebbero avere anche dimensioni differenti fra loro.

Questo progetto di Tesi mira a colmare tale lacuna, proponendo l'analisi e l'implementazio-

ne di un applicativo user-friendly per permettere a medici e biologi di allineare immagini di provini istologici, provenienti da pazienti, al fine di valutare la cross-correlazione tra differenti marcatori tumorali.

In particolare, il progetto è stato sviluppato all'interno del gruppo di ricerca "Data Science for Health" (DS4H), attivo nel proporre metodi ed applicativi per risolvere problemi aperti nel mondo della salute e sanità ed è stato svolto in collaborazione con l'Istituto Scientifico Romagnolo per lo Studio e la cura dei Tumori (IRST) di Meldola (FC).

La Tesi ha portato alla creazione di "DS4H Image Alignment", un applicativo open-source che permette di sovrapporre immagini dello stesso tessuto, acquisite in tempi e modalità differenti, abilitando così gli specialisti Clinici ad analisi istologiche del tessuto del paziente altrimenti difficilmente eseguibili.

In particolare, la Tesi è così organizzata: Il Capitolo 1 descrive le principali modalità di acquisizione di immagini in microscopia ottica e alcune delle tecniche di colorazione cellulari, oltre a introdurre il lettore alle problematiche principali derivanti da queste operazioni, quali deformazioni e disallineamenti dei vetrini a seguito di ogni trattamento. Il Capitolo 2 approfondisce le principali modalità di co-registrazione applicabili ad immagini microscopiche bidimensionali, come nel caso di utilizzo di *DS4H Image Alignment.*, enunciandone i principi matematici che ne costituiscono la base di funzionamento. Il Capitolo 3 approfondisce la procedura di sviluppo di moduli addizionali (chiamati Plugins) di ImageJ, la software suite open-source per elaborazioni di immagini più diffusa tra i ricercatori nel campo delle scienze della vita. Esso riassume le principali modalità di sviluppo di applicativi JAVA compatibili con il programma, approfondendo il framework base sulla quale si erge il suo ecosistema, cioè SciJava. Infine, il Capitolo 4 riporta nello specifico le caratteristiche di "DS4H Image Alignment", spiegando le ragioni che hanno spinto alla sua realizzazione ed enunciando i requisiti funzionali richiesti dai committenti, oltre ad illustrarne l'architettura e le modalità di utilizzo. Il Capitolo si conclude con l'analisi dei punti critici del programma, esplicitando gli accorgimenti impiegati per la loro eliminazione e gli attuali problemi residui, proponendo alcune implementabili soluzioni. I risultati ottenuti dallo sviluppo del programma sono infine riassunti nelle conclusioni, contestualmente a una serie di prospettive di miglioramento e ampliamento del progetto.

Indice

Capitolo 1

Microscopia ottica in ambito istologico

Questo capitolo riporta alcune nozioni alla base della microscopia ottica in ambito istologico, introducendo il lettore al campo di applicazione del software *DS4H Image Alignment* sviluppato in questo progetto di Tesi. Più specificatamente, la sezione 1.1 descrive le principali modalità di analisi di campioni istologici, dettagliando le operazioni di Sezionamento e Colorazione; nella stessa sezione vengono descritte le tipologie di coloranti più comunemente utilizzati in istologia e la relativa strumentazione impiegata per l'analisi dei campioni. Nella sezione 1.2, invece, vengono approfondite alcune delle principali problematiche che possono insorgere con l'attuazione di queste operazioni, come ad esempio la generazioni di aberrazioni ottiche (a causa della strumentazione) o di errori derivanti da fattori umani.

1.1 Tecniche istologiche

L'istologia (dal greco *istos* cioè "tela", e *logos*, cioè "conoscenza") è quella branca della biologia dedicata all'osservazione diretta e allo studio dei tessuti, cioè agglomerati di cellule uniformi dal punto di vista morfologico e funzionale; i tessuti possono essere divisi in quattro categorie principali: tessuto epiteliale, connettivo, muscolare e nervoso. Questa scienza è uno dei cardini fondamentali dell'anatomia patologica, poiché permette di analizzare l'alterazione della morfologia delle cellule, stanti tipicamente a indicare la presenza di un'anomalia funzionale. Più specificatamente, l'istologia richiede l'esecuzione di una serie di operazioni per l'osservazione di un campione: innanzitutto si rende necessario sezionare una porzione di tessuto (fase di *Sezionamento*), evidenziarne poi le componenti di interesse attraverso metodi colorimetrici o fluorimetrici (fase di *Colorazione*). Infine, si procede con l'osservazione del tessuto al microscopio. Questa serie di operazioni viene solitamente raggruppata sotto il nome di "preparazione del campione" e, sopratutto la fase di colorazione delle cellule, è suscettibile di molte variazioni di protocollo. Nelle prossime sezioni, verranno presentati i principali step di preparazione del campione per analisi istologiche.

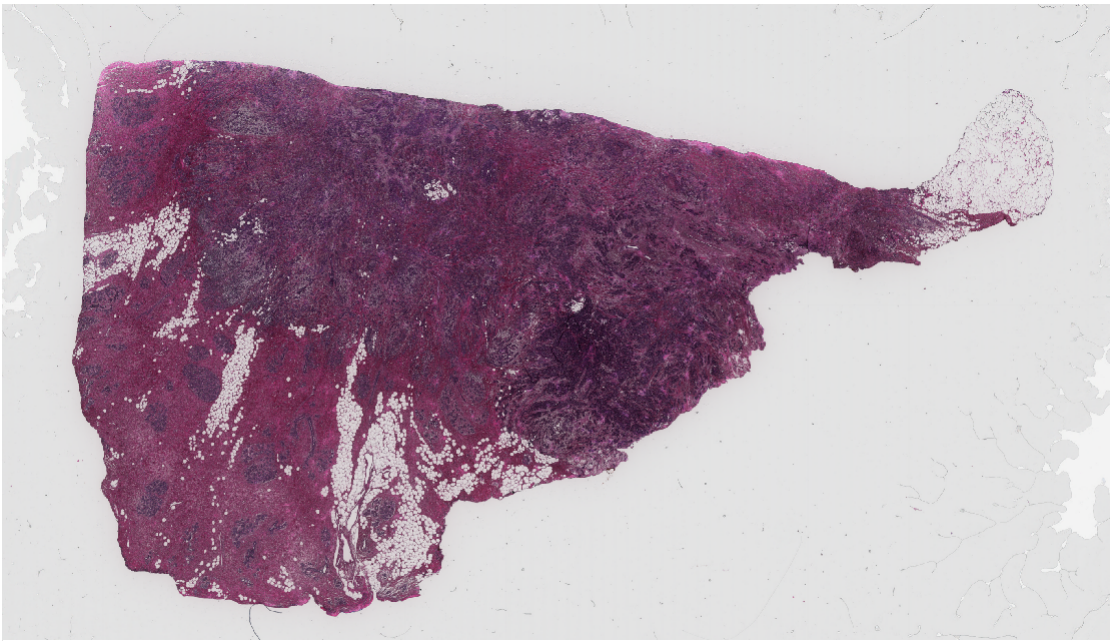

Figura 1.1: Esempio di campione istologico umano marcato con coloranti ottici per evidenziare strutture di interesse.

1.1.1 Sezionamento

Innanzitutto, è bene precisare che i i campioni analizzati in istologia sono tipicamente campioni biologici provenienti da essere vivente. Come tale, sarà imperativo impedire al campione di andare incontro a decomposizione durante la fase di analisi, come normalmente avverrebbe alle cellule dopo il prelievo. Per impedire questa eventualità, sono state messe a punto nei secoli alcune metodiche chiamate ”fissazione” o ”inclusione” che prevedono il trattamento con specifici reagenti (alcol o alcoli deidrogenati) al fine di evitare il loro deterioramento. Si può, inoltre, essiccare o disidratare il tessuto per favorire l'ingresso di sostanze utili al mantenimento della ”corposità” o consistenza del campione, come ad esempio la paraffina.

Dopo l'esecuzione di queste procedure atte a preservare le cellule dalla degradazione, l'operatore può procedere con la fase di sezionamento. Nello specifico, occorrerà ottenere una frazione di tessuto il più sottile possibile attraverso l'utilizzo di uno strumento chiamato ”microtomo”; la misura del taglio è generalmente nell'ordine del micrometro (tipicamente la dimensione di una cellula). Il risultato ottimale di questa fase è quello di ottenere un unico strato di cellule omogenee e non sovrapposte tra loro, garantendo allo specialista Clinico una visione opportuna del campione.

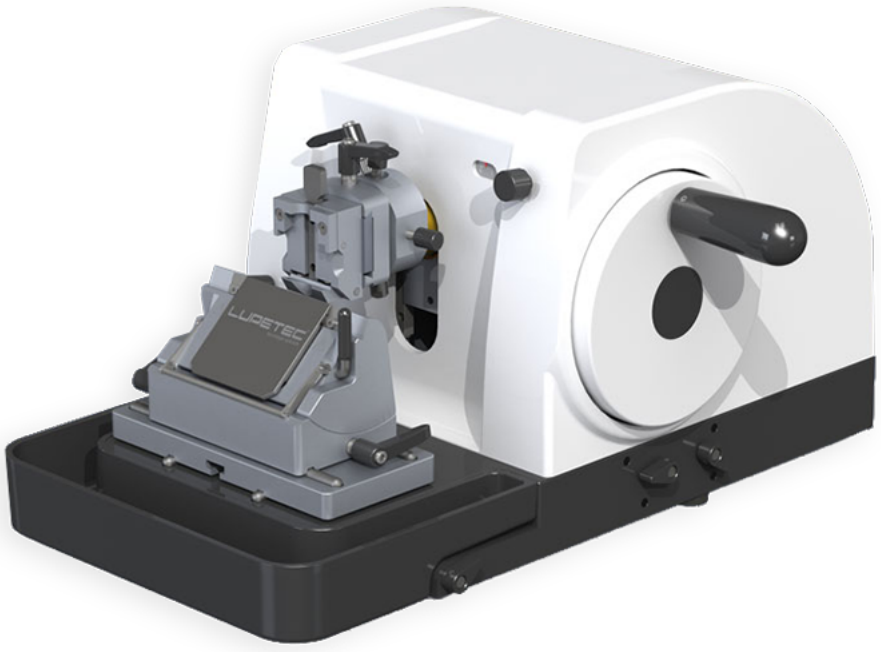

Figura 1.2: Esempio di microtomo manuale.

1.1.2 Colorazione

I tessuti di origine animale sono perlopiù costituiti d'acqua, e quindi la loro analisi al microscopio ottico può risultare difficoltosa data la naturale trasparenza delle cellule. Per ovviare a questo problema, al campione vengono addizionati uno o più coloranti in grado di risaltare determinati elementi delle cellule; in particolare si utilizzano colorazioni basate su reazioni acido-base. Nel dettaglio, per evidenziare gli acidi nucleici del DNA e, di conseguenza, le strutture nucleari vengono utilizzati coloranti basici, mentre invece per la colorazione delle proteine (che compongono il citoplasma) si impiegano di norma coloranti acidi.

Una variante più complessa delle tecniche di colorazione è l'immunofluorescenza, tecnica sulla quale principalmente si basa l'immunoistochimica, cioè l'analisi di tessuti mediante rapporto antigene-anticorpo. Per schematizzare banalmente questo tipo di legame, si immagini l'antigene come un particolare tipo di chiave presente nel campione; l'anticorpo è una serratura selezionata (o in alcuni casi ingegnerizzata) per riconoscere e incastrarsi specificatamente con la suddetta chiave. Nella immunofluorescenza, l'anticorpo selezionato è legato ad una sostanza che emette fluorescenza (ad esempio fluoresceina e cianina), cioè un fluorocromo; queste molecole, una volta eccitate, emetteranno luce colorata a determinati intervalli di lunghezza d'onda che metteranno in rilievo le componenti prese in esame della cellula. In passato si impiegavano radioisotopi per svolgere la funzione di fluorocromi, ma comportavano un rischio maggiore per l'operatore e metodiche più dispendiose. Fortunatamente, al giorno d'oggi il loro utilizzo è stato notevolmente ridotto in favore di composti che non emettono radiazioni dannose.

In maniera semplicistica, la colorazione avviene tramite l'inserimento di colorante sul vetrino contenente il campione da analizzare (tramite strumentazione specifica, come ad esempio una pipettatrice). A seconda della tipologia di analisi, i coloranti possono essere aggiunti e lavati via più volte e subire periodi di incubazione in successione (come nel caso dell'immunofluorescenza) anche piuttosto lunghi.

Esempi di coloranti

In questa sottosezione si presentano alcuni dei principali coloranti a fluorescenza utilizzati in microscopia ottica:

- **Cianina**: con il termine cianina si definisce un gruppo di coloranti polimetinici, che emettono radiazioni luminosi nello spettro che va dal'infrarosso all'ultravioletto. Questo insieme di molecole viene denominato Cy2, Cy3, Cy5, Cy7 in base al numero di gruppi funzionali che possiede; fra le più utilizzate risulta sicuramente Cy3 [1], la cui fluorescenza acquista un colore tendenzialmente giallo-verdognolo ed è facilmente individuabile ad occhio nudo. Una caratteristica di questo gruppo di coloranti è quella di poter essere utilizzato per evidenziare alternativamente sia gli acidi nucleici che le proteine.

- **DAPI**: il 4',6-diamidino-2-fenilindole è un colorante utilizzato per la sua capacità di legarsi ai segmenti di DNA in cui abbondino timina e adenina. Una sua caratteristica interessante è la sua capacità di attraversare la membrana cellulare in maniera non distruttiva (grazie alla sua particolare struttura chimica), fornendo una colorazione tipicamente azzurrognola.

- **FITC**: la fluorescina-5-isotiocianato è caratterizzata, come già anticipato dal nome, dalla presenza di un gruppo isotiocianato estremamente reattivo. Indubbiamente il suo utilizzo più frequente è nel legame con le proteine dotate di gruppi solfidrilici e amminici liberi. Uno dei suoi limiti più noti è la sua relativa instabilità, che può portare all'effetto di *photobleaching*.

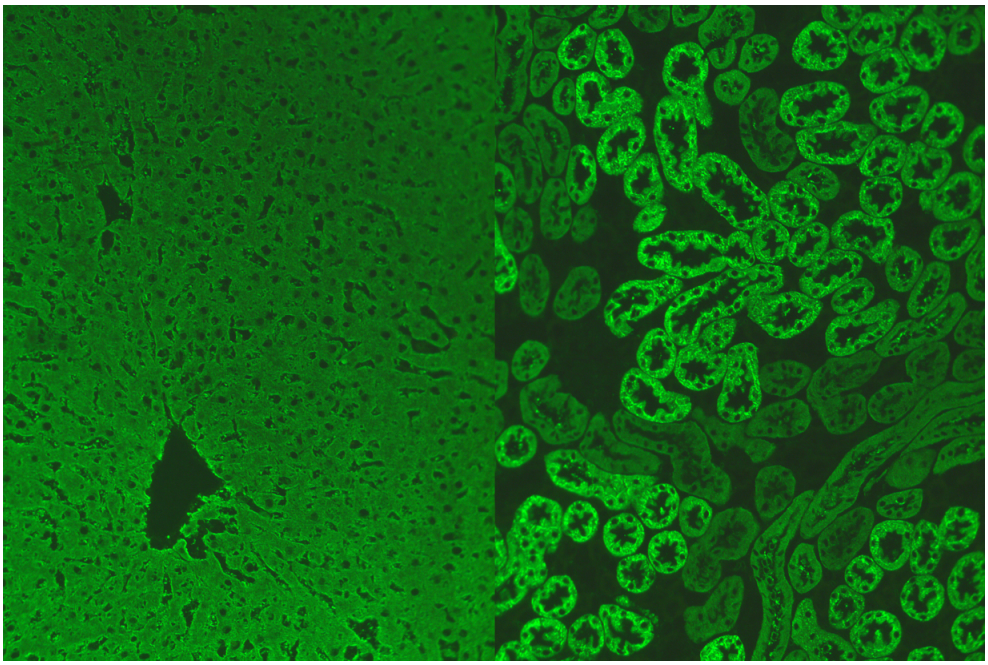

Figura 1.3: Esempio di emissione FITC

- **Eosina**: L'eosina è una tetrabromofluoresceina molto utilizzata come colorante citoplasmatico, in contrasto all'ematossilina (che, come vedremo, è impiegata come colorante degli acidi nucleici delle cellule); la sua componente acida è infatti indicata alla colorazione del citoplasma cellulare. La tecnica che vede la combinazione dei due pigmenti è abbreviata con la sigla EE ed è permanente nel tessuto dopo la sua applicazione. Il pigmento caratteristico di questa sostanza è il rosa-rosso.

- **Ematossilina**: l'ematossilina è un estratto di origine vegetale ed è una delle sostanze più utilizzate in istologia in coppia con l'eosina sopra descritta. Questa sostanza dona un colore blu-violaceo alle cellule, evidenziando gli acidi nucleici e la membrana cellulare, e per questo motivo viene spesso definita "colorante basico". Data la sua incapacità di

permeare attraverso il tessuto viene associata a sali metallici (come ad esempio ferro, alluminio e piombo).

L'impiego dei suddetti coloranti (e più generalmente della quasi totalità dei fluorocromi) prevede più lavaggi tramite PBS (*tampone fosfato salino*, una soluzione in grado di mantenere il ph costante nel campione) e periodi di incubazione che possono comportare periodi di lavoro di alcuni giorni. Come vedremo, queste manipolazioni possono causare deformazioni della morfologia del campione.

1.1.3 Microscopia ottica

L'iter di preparazione del campione si conclude con l'allestimento di un vetrino portaoggetti sul quale viene depositato il materiale biologico da analizzare e la successiva scansione da parte dell'operatore. Al di là della differenziazione tra i vari modelli a disposizione in commercio, il microscopio ottico risulta generalmente così composto:

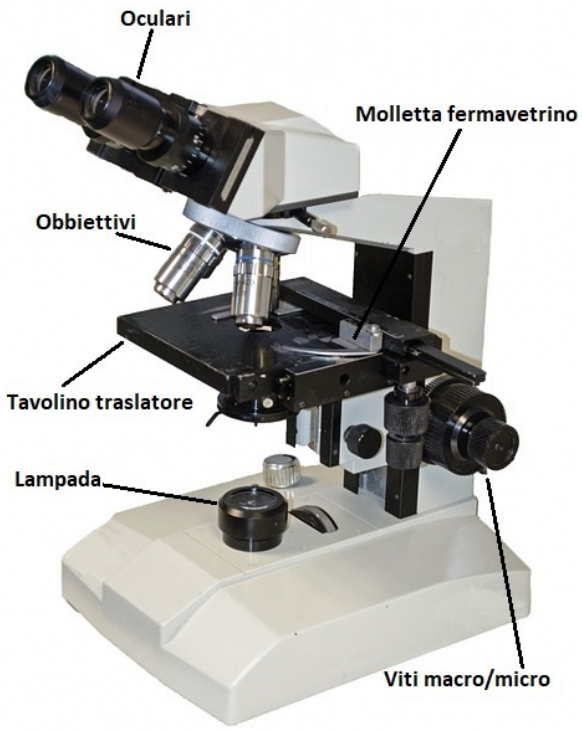

Figura 1.4: Principali componenti di un microscopio ottico.

Per funzionare il microscopio ottico sfrutta due coppie di lenti biconvesse che rifrangono la luce proveniente dalla lampada (posta di norma sotto al tavolino); il raggio di luce viene così deviato, ricostruendo un'immagine amplificata rispetto all'originale. Al di là dell'ingrandimento, è indispensabile fornirsi di uno strumento capace di una buona risoluzione delle immagini prodotte:

ciò permetterà di distinguere tutti gli elementi di un campione posti a distanze microscopiche fra di loro.

Grazie all'utilizzo del microscopio, l'operatore può quindi osservare il campione con risoluzione sub-cellulare ed acquisire immagini di esso per successive analisi. Tuttavia, non sempre è possibile marcare in una unica battuta tutti gli elementi sub-cellulari di interesse per potere eseguire una corretta valutazione del tessuto del paziente e, quindi, poter valutare patologie o deformazioni. Tipicamente si eseguono quindi acquisizioni di immagini in sequenza, smontando il campione dal vetrino e colorando in sequenza con tutti i marcatori di interesse che, per vari motivi meglio conosciuti dagli esperti di biochimica con il nome tecnico di *limiti di carico stechiometric*o, non possono essere usati contemporaneamente. Infine, nella procedura standard lo specialista Clinico osserva le immagini in parallelo al fine di avere una visione di insieme del campione. DS4H Image Alignment consente invece di allineare le immagini in un unico stack, consentendo quindi non solo la visione contemporanea di tutti i coloranti utilizzati, ma anche di eseguire accurati studi di co-relazione tra i diversi marcatori.

1.2 Deformazioni e disallineamenti

In condizioni ideali l'immagine ottenuta dal microscopio ottico è una fedele riproduzione ingrandita del tessuto originale. I fattori che possono comportare, invece, alterazioni dell'immagine sono molteplici e possono essere suddivisi in strumentali ed umani.
Tra i primi si ricordano:

- **Cause tecniche:** esse sono per lo più legate alla manutenzione del microscopio, magari per un'errata pulizia o l'usura dei vetrini portaoggetti;
- **Cause fisiche:** spesso sono la causa di diffrazioni, cioè legato alla natura ondulatoria della radiazione luminosa;
- **Cause geometriche:** in questo caso il motivo è da ricercarsi nella rifrazione aria-vetro;

I fattori umani, invece, dipendono dalle frequenti ed inevitabili manipolazioni da parte dell'operatore. Si pensi, infatti, a:

- **Trattamenti in successione con reagenti**: lavaggi a base di PBS nella fase di colorazione, utilizzo di coloranti diversi sullo stesso campione di cellule;
- **Prolungati tempi di incubazione degli anticorpi:** soprattutto in caso di tecniche di immunofluorescenza indiretta, l'incubazione con anticorpi secondari può rendere necessari più giorni di lavoro per uno stesso campione. Per eseguire la stessa analisi, quindi, potrebbe verificarsi la successione di diversi operatori che lavorano su uno stesso campione, ciascuno con una diversa manualità.
- **Riposizionamento dei vetrini:** le operazioni di colorazione, incubazione e lavaggio richiedono lo spostamento fisico del vetrino dal microscopio al piano di lavoro. Nel momento in cui uno stesso campione viene tolto e poi reinserito sul tavolino portaoggetti, le immagini ottenute potrebbero vedere la traslazione delle cellule: se infatti l'obbiettivo dello strumento di acquisizione immagini è fisso non si può dire lo stesso del vetrino, che può spostarsi all'interno della fotografia. Quelli che, a prima vista, potrebbero apparire come impercettibili variazioni nel posizionamento delle cellule, si traducono in disallineamenti che rendono difficoltosa l'individuazione di aree target per l'analisi istologica.

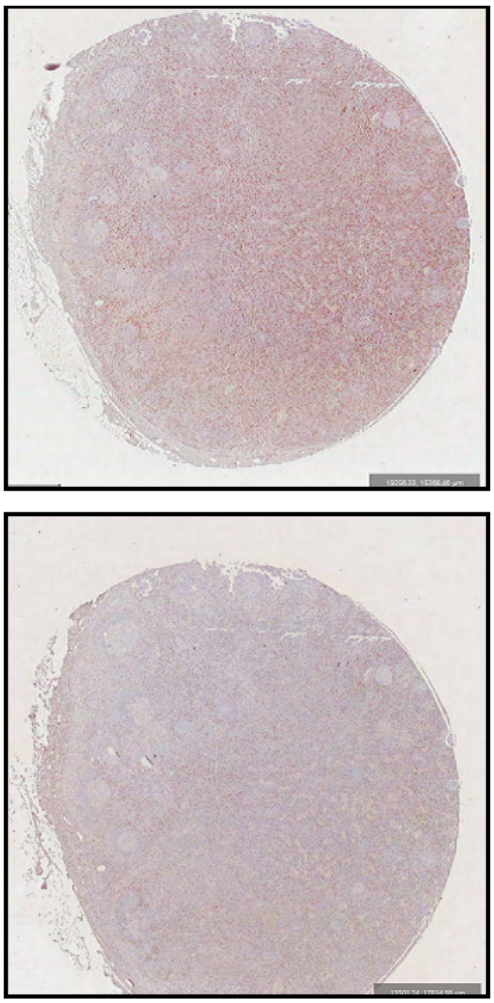

Figura 1.5: Esempio di disallineamento di vetrini tra fasi di colorazioni diverse. Si noti come lo stesso campione risulti traslato rispetto al medesimo riferimento.

Si renderà necessario, quindi, eseguire delle elaborazioni delle immagini post-produzione che riallineino precisamente le sequenze dei campioni, e in questo frangente viene posto lo sviluppo di *DS4H Image Alignment*, come si vedrà nel Capitolo 4.

Capitolo 2

Metodi di co-registrazione di immagini microscopiche

In questo capitolo vengono approfondite le principali modalità di co-registrazione applicabili ad immagini microscopiche bidimensionali, come nel caso di utilizzo di *DS4H Image Alignment.* Più specificatamente, nella sezione 2.1 ne vengono introdotte alcune nozioni fondamentali, presentando dapprima i principali algoritmi di registrazione riscontrabili in letteratura (sottosezione 2.1.1) e poi illustrando i modelli di trasformazione lineare applicabili e le relative differenze in termini di resa grafica (sottosezione 2.1.2).
La sezione 2.2 approfondisce invece il principio matematico che sta alla base della deformazione di immagini, introducendo nella sottosezione 2.2.1 la metodologia di minimizzazione MLS (*Moving Least Squares*) utilizzata in questo progetto di tesi e, nella sottosezione 2.2.2, alcune strategie di ottimizzazione impiegate tramite *meshing* dell'immagine.

2.1 Definizione e Concetti

La registrazione di immagini è definita come il processo di allineamento geometrico tra due o più rappresentazioni che si riferiscono allo stesso oggetto (o scena), grazie al quale è possibile sovrapporre immagini provenienti da prospettive o strumentazioni diverse [2] (come ad esempio nel già trattato Capitolo 1 con la colorazione tramite immunofluorescenza). Storicamente, l'utilizzo di tecniche di registrazione di immagini è documentato già nei primi anni del 1900, quando lo sviluppo della fotografia aerea ha reso la creazione di mosaici di immagini un'attività comune per la costruzione di mappe topografiche. Successivamente, i campi di applicazione sono diventati molteplici e attualmente variano dalla costruzione di scene panoramiche tramite apparecchi fotografici al tracciamento di oggetti in movimento, oltre all'allineamento di immagini microscopiche bidimensionali come trattato in questo progetto di Tesi.

L'impiego di questa metodologia prevede la selezione di due diverse immagini, una delle quali fungerà da riferimento e una che subirà un processo di *resampling* di allineamento; generalmente, esse vengono denominate rispettivamente *reference* (oppure *source*) e *target* [31].

2.1.1 Algoritmi di Co-registrazione

La diversità delle immagini trattate e i molteplici campi di applicazione rendono impossibile l'individuazione di un unico metodo di co-registrazione applicabile universalmente; ne sono stati pertanto proposti diversi nel corso del tempo, ognuno dei quali specializzato in un preciso ambito di applicazione. Uno delle classificazioni più comuni si basa sulle diverse modalità di confronto tra le immagini date in input [2]. Secondo questa logica, possono essere individuate le seguenti categorie di algoritmi:

- **Co-registrazione manuale:** questo genere di co-registrazione richiede un'interazione completa da parte dell'utente, dato che sarà suo compito allineare visivamente i soggetti interessati tramite operazioni trascinamento (traslazione) delle immagini fino all'ottenimento di un risultato soddisfacente. Com'è intuibile, questo tipo di registrazione ha dei limiti importanti, sia dal punto di vista dell'accuratezza del risultato (dipenderanno principalmente infatti dal grado di esperienza e dalle capacità dell'utente) che dal punto di vista dell'usabilità e facilità d'uso. ImageJ TrackEM2 risponde a questa tipologia di applicazione[32]: grazie a un flessibile utilizzo di contrasti tra *Source* e *Template* l'utente è visivamente facilitato nell'avvicinamento e sequenziazione delle immagini.

- **Co-Registrazione tramite Landmark**: questo tipo di registrazione si basa sul posizionamento da parte dell'operatore di una serie di marcatori sulle immagini prescelte come *source* e *target*, e possono assumere la forma di semplici punti [33], linee [34] o griglie polinomiali [35]. Una volta posizionati i marcatori su entrambe le immagini (che rappresentiamo con gli insiemi $P = \{P_i\}$ e $Q = \{Q_i\}$), si attuerà una tecnica di ottimizzazione

(o regressione) per ricercare una trasformazione T che minimizzi ad esempio la radice quadrata delle distanze dei punti accoppiati. Nella sezione 2.2 vengono opportunamente dettagliate le tecniche, anche chiamate metodologie dei minimi quadrati o *Ordinary Least Squares*, utilizzate durante lo sviluppo di questo progetto di Tesi. Lo svantaggio principale di questa metodologia è che, come per la co-registrazione manuale, richiede la partecipazione di personale con un certo grado di conoscenza dell'ambito in cui opera; il posizionamento dei marcatori, infatti, deve essere congruente su entrambi le immagini e, sopratutto nei casi di microscopia ottica, ciò richiede una buona capacità di distinzione tra punti caratteristici di uno stesso tessuto. Uno dei più immediati esempi di applicazione di registrazione landmark-based può essere trovata nel plugin *Align Images by Line ROI* [36] di ImageJ, mentre invece *DS4H Image Alignment* è un esempio di applicazione che sfrutta la registrazione basata su punti.

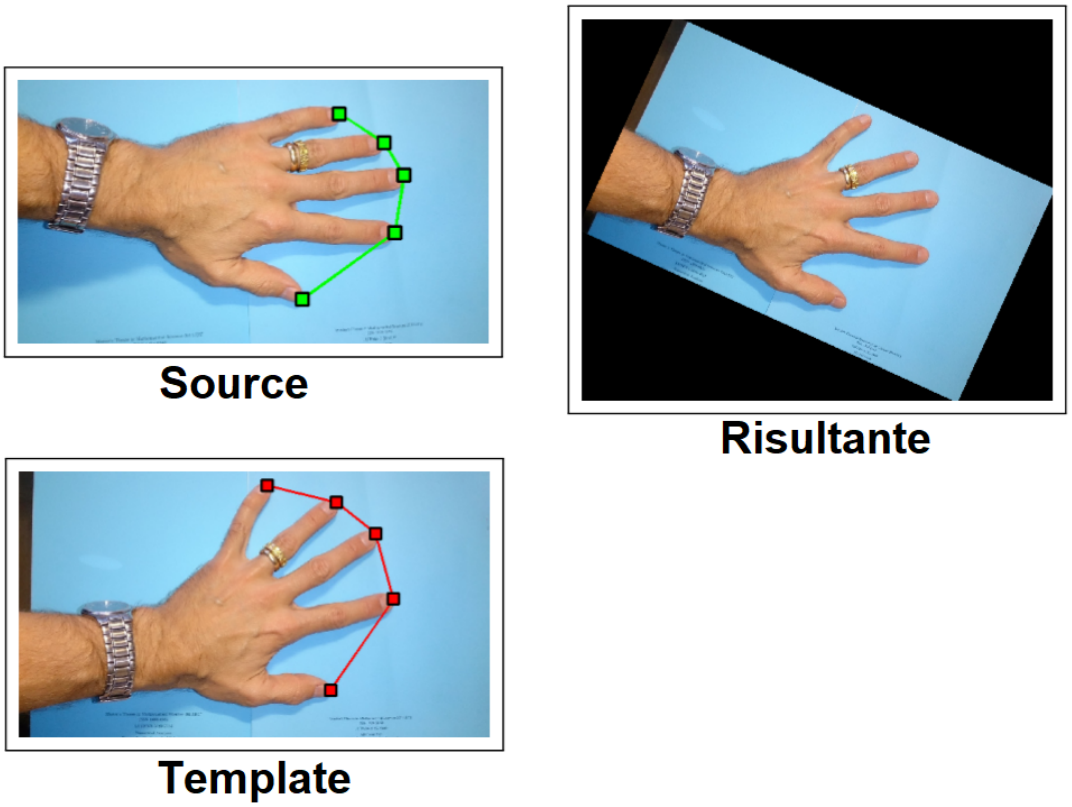

Figura 2.1: Esempio di registrazione basata su punti. Tratto da: *http://www.maths.lu.se/*

- **Co-registrazione surface-based**: si tratta di una categoria di algoritmi che si basano sulla ricerca di una trasformazione T che minimizzi la distanza sulle superfici individuate tra *Source* e *Template*. Le superfici possono essere individuate tramite tecniche di segmentazione delle immagini, atte cioè a partizionare immagini in regioni significative (che diverranno così le superfici di interesse per la registrazione). Tra le più utilizzate in ambito biomedicale [38] citiamo:

  - Thresholding based segmentation: si tratta di una modalità di segmentazione molto semplice, che si basa sulla contornazione di tutti gli oggetti che possiedono una colorazione in scala di grigi entro un limite specificato (*treshold*). È applicabile solo dove vi è una netta distinzione tra sfondo e oggetti da selezionare, e non è adeguata a situazioni in cui è necessario fare una distinzione più precisa tra gli oggetti presenti nell'immagine.
  - Edge based segmentation: questa tecnica si basa sull'individuazione dei contorni di oggetti basandosi sulla differenza locale di scala di grigi e colori. L'insieme dei punti di interesse individuato viene così concatenato fino a formare una *edge chain*, in modo così da eliminare eventuali contorni spuri.
  - Region based segmentation: la region based segmentation è incentrata sul principio di omogeneità, suddividendo automaticamente regioni locali con pixel dalle proprietà simili (solitamente valori nella scala di grigi).

  Sebbene tecniche di surface registration siano state teorizzate già da tempo [40] solo recentemente esse hanno avuto un notevole impulso di ricerca; tra i framework più innovativi del settore si cita MSM [39] (*Multimodal Surface Matching*), specializzato nell'allineamento di immagini neurali.

- **Co-registrazione intensity-based**: questa tipologia di registrazione si basa sulla misura della similarità delle immagini da allineare, confrontando eventuali regolarità grazie a metriche di correlazione. Una delle metriche più intuitive, basata sulla somma dei quadrati delle differenze di intensità dei *voxel* di *Source* e *Template* è:

$$\sum_{p\in\Omega} [I_s(p) - I_t(p)]^2$$

  Dove Ω rappresenta la regione di intersezione fra le immagini e p la posizione dei pixel sulla regione. La somma risulterà zero nel caso di immagini perfettamente allineate. Questa tecnica (abbreviata tramite l'acronimo *SSD*) risulta facilmente calcolabile, differenziabile e capace di fornire un'ottima misura di similarità in caso di immagini che differiscono solo per un rumore gaussiano. Tuttavia essa è notevolmente influenzata da forti cambiamenti di intensità tra le immagini, rendendola praticamente inutilizzabile in contesti istologici (ad esempio tramite l'utilizzo di un semplice agente di contrasto). Altre tecniche di co-registrazione sono invece basate su informazioni dedotte da variabili casuali impostate su entrambe le immagini (metodo della Mutua Informazione) oppure su relazioni nello spazio delle frequenze delle immagini (*phase correlation*). Infine, è senza dubbio meritevole di menzione il software Elastix [41] come tool di registrazione basato su SSD, principalmente dedicato all'elaborazione di immagini biomedicali.

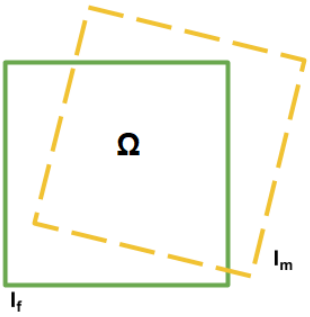

Figura 2.2: Spazio di intersezione tra due immagini

2.1.2 Modelli di trasformazione

L'applicazione di un algoritmo di co-registrazione prevede l'utilizzo di uno specifico modello di trasformazione tra *Source* e *Template*, cioè una funzione che opera una rilocazione dei punti di un'immagine rispetto a un sistema di riferimento. Innanzitutto, le trasformazioni possono essere divise in "rigide" oppure "elastiche". Le prime rientrano nella categoria delle *isometrie*, cioè trasformazioni geometriche che lasciano invariate le lunghezze, gli angoli e le aree dell'immagine campione. Questo tipo di operazioni sono generalmente definite da una singola equazione che viene applicata all'intera immagine, e sono pertanto ritenute più concise e immediate.

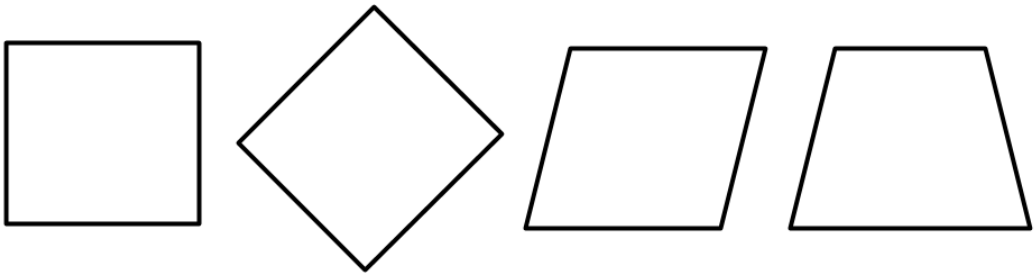

Figura 2.3: Modelli globali. Da sinistra a destra: immagine originale, rappresentazione a seguito di trasformazione roto-traslativa rigida, a seguito di trasformazione affine, a seguito di trasformazione proiettiva.

Le trasformazioni elastiche sono invece rappresentate da un insieme di equazioni, ognuna dedicata ad una specifica regione dell'immagine campione. La loro applicazione può essere semplicisticamente immaginata come la deformazione di un corpo elastico sottoposto a una serie di forze di corpi esterni (a cui ognuna corrisponde una differente equazione). Date le loro particolari modalità di applicazione, questo genere di trasformazioni vengono anche chiamate, rispettivamente, "globali" o "locali"; dato comunque che l'ambito di applicazione di *DS4H Image Alignment* è rappresentato da oggetti di natura rigida, questo progetto di tesi approfondirà soltanto trasformazioni globali, considerate più adatte e performanti per lo scopo. A tal fine, vengono presi in esame i principali modelli presentati in letteratura [42]:

Modello roto-traslativo rigido

Le isometrie (o movimenti rigidi) sono trasformazioni del piano che mantengono invariate le distanze tra i punti dell'immagine traslata. In altre parole, prendendo $\overline{AB}$ come la distanza tra i punti A e B e $\overline{A'B'}$ come la distanza tra i punti trasformati, si avrà allora che:

$$\overline{AB} = \overline{A'B'}$$

Ne consegue che tutte le operazioni isometriche trasformano un immagine in una figura congruente, cambiandone solo la posizione sul piano. Concettualmente, la trasformazione rigida può essere descritta come l'applicazione di un'operazione di traslazione + rotazione; considerando $p = [x,y]^T$ un qualsiasi punto dell'immagine s e $p^{'} = [x^{'},y^{'}]^T$ il medesimo punto posto nell'immagine t, allora le coordinate cartesiane delle due osservazioni saranno legate tramite l'equazione:

$$\begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & a \\ sin\theta & cos\theta & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix}$$

Dove θ è l'angolo di rotazione dell'immagine da applicare, mentre la coppia a,b rappresentano rispettivamente la traslazione sull'asse delle ascisse e l'asse delle ordinate. Dato il ristretto numero di parametri di cui è composto, questo modello risulta adeguato solo in alcuni limitati casi nei quali due sequenze di immagini differiscono solo in termini di rotazione e traslazione; nondimeno, la registrazione di immagini microscopiche digitali tramite questo modello può risultare adeguata, dato che il dispositivo di acquisizione rimane sempre perpendicolare e fisso alla stessa distanza rispetto ai campioni.

Modello affine

Il modello affine può essere considerato come la combinazione di trasformazioni di similarità con l'applicazione di *shearing* sull'immagine; precisamente, esso prevede l'utilizzo di funzioni di Identità, Traslazione, Scala, Omotetia, Similarità, Riflessione, Composizione e *Shearing* in un qualsiasi ordine e sequenza. A livello visivo questo tipo di trasformazione può essere immaginata come l'allungamento di un oggetto parallelamente al suo piano dopo aver applicato una forza sui suoi bordi. Per tale motivo, questo modello ha la particolarità di mantenere parallele tutte le linee che erano parallele prima della sua applicazione; a differenza del modello rigido, tuttavia, le affinità non sono necessariamente delle isometrie, dato che tutte le sue applicazioni non danno garanzia sulla preservazione di angoli e distanze tra i punti.
Basandosi sulla matrice introdotta con il modello rigido, il modello affine è legato dall'equazione:

$$\begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix} = \begin{bmatrix} A\ cos\theta & B\ sin\theta & a \\ C\ sin\theta & D\ cos\theta & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix}$$

dove A, B, ,C, D sono numeri reali. Si può dimostrare che ogni operazione di affinità gode delle seguenti proprietà:

- Collinearità tra i punti: se tre punti P, Q, R sono allineati, i loro corrispondenti in un'affinità P', Q', R' sono anch'essi allineati.
- Parallelismo: a rette parallele corrispondono rette parallele.
- Rapporto dei segmenti paralleli: il punto medio di un segmento corrisponde sempre al punto medio di un segmento omologo.
- se la figura S' è l'immagine corrispondente di una figura S, allora $\frac{Area\ (S')}{Area\ (S)} = |det\ A|$, dove $det\ A = ad - bc$

Modello Proiettivo

La trasformazione proiettiva, anche chiamata omografica o prospettica, è una delle più utilizzate tecniche nel campo della registrazione d'immagine, ed è considerata una delle più diffuse tipologie di trasformazione lineare. In poche parole, questo tipo di operazione può essere immaginata come il posizionamento di un occhio nello spazio: in questo modo, le grandezze degli oggetti non saranno quantificabili (poiché non vi saranno informazioni sulla profondità degli oggetti stessi) e l'orizzonte verrà considerato parte integrante del piano. A livello pratico, una delle principali differenze rispetto alle operazioni di affinità è che la trasformazione non dà garanzie sul mantenimento del parallelismo delle linee o il rapporto dell'immagine.
Una tipica trasformazione omografica assume una forma matriciale di questo tipo:

$$\begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix} = \begin{bmatrix} A\ cos\theta & B\ sin\theta & a \\ C\ sin\theta & D\ cos\theta & b \\ G & H & 1 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix}$$

Rimangono validi i formalismi già precedentemente introdotti per il modello affine.

2.2 Deformazioni minimi quadrati

In questa sezione viene approfondito il principio matematico che sta alla base della deformazione dell'immagine in *DS4H Image Alignment*, cioè lo sfruttamento di una funzione di trasformazione lineare affine (sebbene vengano utilizzate anche trasformazioni traslative, esse non verranno approfondite nel dettaglio, dato che sono ritenuti più semplici e triviali).

Specificatamente, verrà preso in analisi il metodo dei minimi quadrati, cioè una tecnica di ottimizzazione che permette di individuare una funzione che si avvicini il più possibile a un insieme di dati (in questo caso, punti in un piano); la funzione individuata minimizza la differenza al quadrato dello spostamento sull'asse dei punti delle immagini e la loro rotazione, grazie all'utilizzo di una matrice di affinità. Il metodo approfondito (e utilizzato nel software tool oggetto di questa Tesi) è il *Moving Least Squares*, molto utilizzato in ambito di computer grafica assieme alle sue controparti Generalizzate (*Generalized least squares*) e Pesate (*Weighted least squares*).

2.2.1 Moving Least Squares

Concettualmente, la deformazione può essere riassunta come l'utilizzo di una funzione f per ogni punto presente nell'immagine originale: la risultante sarà infatti il frutto dell'applicazione di f per ogni punto v dell'originale. Prendendo p come l'insieme dei punti selezionati dall'algoritmo di registrazione (oppure inseriti manualmente dall'utente), la funzione f si potrà ritenere soddisfacente solo se le seguenti proprietà si considereranno rispettate:

- **Interpolazione**: i punti p dovranno corrispondere esattamente agli stessi punti q una volta applicata la trasformazione (tali per cui $f(p_i) = q_i$)
- **Identità**: se i punti q dell'immagine deformata sono identici ai punti p dell'immagine originale, allora la deformazione sarà pari all'applicazione di una funzione di identità, tale per cui $q_i = p_i \Rightarrow f(v) = v$

Ovviamente, supponendo l'implementazione di un algoritmo di registrazione landmark-based, i punti di controllo p_i sono da considerare come i landmark manualmente posizionati dall'utente sull'immagine *Template*.

Basandosi sull'articolo di Levin [43], ripreso anche dall'articolo [26] che pone le basi per la libreria grafica utilizzata in questi progetto, possiamo riportare la definizione di una funzione f che rispetti le precedenti proprietà e produca un risultato con un accettabile livello di rumore e di realismo per modelli rigidi. Dato un punto v nell'immagine, si ricerca la migliore

trasformazione lineare $l_v(x)$ che minimizza:

$$\sum_i w_i \left| l_v(p_i) - q_i \right|^2$$

dove p_i e q_i sono righe vettoriali e le variabili w_i hanno la forma:

$$w_i = \frac{1}{|p_i - v|^{2a}}$$

È possibile ottenere una differente funzione di trasformazione $l_v(x)$ per ogni valore v dato il peso w_i, e possiamo definire la nostra funzione di deformazione f come $f(v) = l_v(v)$. Inoltre, dal momento che v si avvicina al valore p_i, il valore risultante wi tenderà all'infinito (e pertanto $f(p_i) = q_i$) Nel caso quindi in cui tutte le variabili q e p siano uguali, allora la funzione di trasformazione sarà sempre $l_v(x) = x$, cioè la sua applicazione sarà pari a una funzione di identità.

Dato che $l_v(x)$ è una trasformazione affine si può scomporre in due parti separate, cioè matrice di trasformazione lineare M e una traslazione T:

$$l_v(x) = xM + T$$

T può essere riscritta come:

$$T = q_* - p_* M$$

dove q_* e p_* sono baricentri pesati pari a:

$$p_* = \frac{\sum_i w_i p_i}{\sum_i w_i} \qquad q_* = \frac{\sum_i w_i q_i}{\sum_i w_i}$$

e pertanto possiamo sostituire T presente nell'equazione di $l_v(x)$ riscrivendola:

$$l_v(x) = (x - p_*)M + q$$

L'equazione da minimizzare potrà quindi essere specificata in questo modo:

$$\sum_i w_i \left| (p_i - p_*)M - (q_i - q_*) \right|^2$$

2.2.2 Applicazione e Mesh Deformation

Teoricamente, la deformazione di un immagine richiede l'applicazione della funzione affine su ogni pixel presente nell'immagine bidimensionale (nel caso di immagini tridimensionali, invece, dovrà essere applicata ad ogni *voxel*). Nell'ambito della microscopia digitale (soprattutto nel caso in cui il file sia in formato *wholeslide*) il numero di pixel può tuttavia facilmente superare le 2^{31} unità, rendendo la deformazione particolarmente onerosa in termini di costi com-

putazionali (e talvolta sostanzialmente impossibile da attuare su elaboratori meno performanti).

Una strategia largamente impiegata in questi contesti è quella di suddividere un immagine in una serie di *mesh*, cioè poligoni di una stessa dimensione, e di calcolare la funzione affine solo nei loro vertici. Molto utilizzati in ambito di computer grafica, i mesh possono essere di varia dimensione e forma (come ad esempio piramidi o cubi) ma in ambito bidimensionale è comune creare griglie di oggetti rettangolari. Grazie a questa suddivisione, la quantità di calcoli per applicare la deformazione viene notevolmente ridotta, pur mantenendo un risultato visivo praticamente impercettibile all'occhio umano (a patto che l'area di ogni poligono sia sufficientemente ridotta).

A titolo di esempio, *DS4H Image Alignment* suddivide le immagini sottoposte a registrazione a un processo di *meshing* con una risoluzione di 32x32 pixel. Questo consente di ridurre il numero di calcoli per un immagine di 2^{31} pixel (tipicamente un' immagine con una risoluzione di 46k x 46k pixel) di due ordini di grandezza, cioè equivalenti a 2^{26} operazioni.

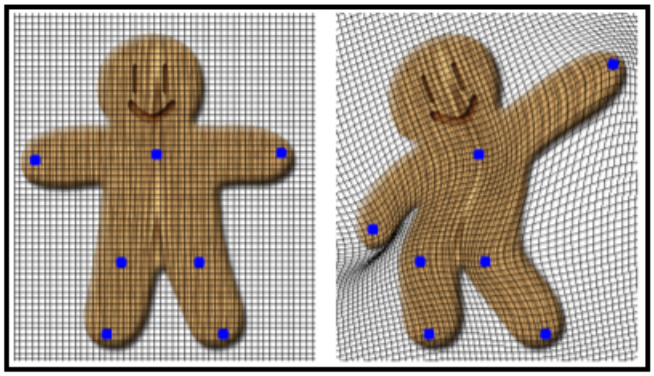

Figura 2.4: Mesh grid e relativa deformazione. Si noti la curvatura accentuata sui punti di controllo. Tratto da *Image Deformation Using Moving Least Squares* [26]

Capitolo 3

ImageJ: Descrizione e Plugins

3.1 ImageJ

ImageJ [9] è una software suite di *image processing* originariamente sviluppato dall'NIH (*National Institutes of Health*) americano nel 1997, in sostituzione all'ormai datato *NIH Image*. Similarmente ad alcuni programmi già visti nelle sezioni precedenti (come *Icy*, *Cellprofiler* e *QuPath*), ImageJ viene considerato [10] un *"Generalist image analysis tool"*, capace cioè di ben adattarsi a un ampio set di problemi di natura diversa (come la diagnostica per immagini, la scienza dei materiali e la microscopia ottica); la sua flessibilità (riconosciuta come una delle chiavi del successo di questo programma) viene garantita da un framework che consente di estendere agevolmente le funzionalità del programma attraverso l'introduzione di moduli aggiuntivi sviluppati dagli utenti (in JAVA, Kotlin, Javascript, Scala, MATLAB ecc.)

Trattandosi di un software open-source mantenuto da una comunità di persone piuttosto eterogenea (ricercatori, biologi, sviluppatori ecc.), esistono molteplici distribuzioni parallele di ImageJ (comunemente chiamati *"Flavors"*), ognuna dedicata alla risoluzione di una categoria di problemi. A fianco alla build "classica" di ImageJ (ImageJ e ImageJ2), infatti, vengono sviluppate versioni dedicate allo studio e analisi di immagini astronomiche (chiamato *AstroImageJ* [11]) o che utilizzano l'engine JavaFX [12] come interfaccia (ambendo a creare una nuova versione che possa essere più intuitiva e adatta agli standard della UX odierna). La build tuttavia più utilizzata in ambito di analisi istologica è tuttavia Fiji [13], che include una serie di plugins predefiniti e certificati in ogni versione. Essendo la versione più comunemente utilizzata e mantenuta, questo elaborato si concentrerà sullo sviluppo di un applicativo compatibile con la versione Fiji di ImageJ.

3.1.1 Software tools analoghi

Attualmente esiste una buona disponibilità di software che permettano un analisi automatizzata di immagini biologiche. Tra i più utilizzati dalla comunità, ricordiamo:

- **CellProfiler**: CellProfiler [3] è un software gratuito open-source che permette, grazie a una serie di algoritmi già inclusi all'interno del pacchetto software, di effettuare analisi quantitative di immagini biologiche; strutturando gli algoritmi come moduli indipendenti, CellProfiler consente di creare delle *pipelines* di elaborazione (sarà infatti sufficiente ordinare in maniera sequenziale gli algoritmi scelti). I campi di applicazione del software sono molteplici (ad esempio, classificazione e conta delle celle, calcolo delle dimensioni e stima dell'intensità del colore) e possono essere ulteriormente estesi tramite l'introduzione di nuovi moduli da parte della comunità (sotto forma di plugin); l'elaborazione non è ristretta a singole immagini ma sono disponibili anche *pipelines* per intere sequenze (ad esempio per finalità di object tracking).

  In questi casi la quantità di informazioni prodotte può risultare ingente, rendendo difficile la loro analisi tramite strumenti non appositamente progettati; per agevolare questa operazione è stato sviluppato un software complementare chiamato CellProfiler Analyst [4], in grado di agevolare notevolmente l'esplorazione di dati multi-dimensionali grazie alla sua interfaccia semplificata. Inoltre, esso fornisce strumenti di analisi tramite *machine learning* con sistemi di classificazione supervisionata dall'utente. Infine, gli ultimi sviluppi del software hanno visto l'integrazione di feature che si basano su Reti neurali Convoluzionali [5] (CNN) per l'elaborazione delle immagini sottoposte, grazie all'integrazione di pacchetti software terze parti come *TensorFlow*.

- **Citomyne**: Cytomine [7] è un applicativo web open-source che fornisce un ambiente di analisi collaborativa di immagini biologiche *wholeslide*. Specificatamente, questo tool è stato sviluppato per cambiare le modalità tipiche di lavoro degli ambienti di analisi di immagini biomedicale, dove il numero di ricercatori è spesso limitato e ristretto allo stesso dominio; le sue funzionalità promuovono attivamente la collaborazione attiva di gruppi di scienziati eterogenei come Biologi, Informatici e Citizen Scientists [8].

  A livello di funzionalità, il software supporta nativamente immagini multi-gigapixel ad alta risoluzione, fornisce strumenti di controllo di accesso e di condivisione delle annotazioni degli utenti e permette di eseguire algoritmi di analisi automatizzata (come ad esempio algoritmi di *object couinting*, *object classification* e *landmark detection*) con minime preconfigurazioni necessarie da parte dell'utente.

  L'architettura di *Cytomine* si suddivide in quattro componenti che comunicano tramite API RESTful, cioè *Citomyne core* (modulo principale del tool, responsabile dell'inpu-

t/output dei *Data Models* del progetto nonché del funzionamento del Database di progetto), *Cytomine-IMS* (componente che fornisce una serie di *web services* utilizzati per l'upload di immagini 5D multi-gigapixel), *Cytomine-WebUI* (Componente frontend dell'applicazione, scritta principalmente in *angularJs e python*) e *Cytomine-DataMinining* (modulo responsabile del funzionamento di algoritmi di image recognition basati su machine learning).

- **Qupath**: QuPath [6] è un software open source desktop studiato per facilitare l'analisi e l'esplorazione di immagini *wholeslide* biologiche. La maggior parte dei software di digital imaging (come *Fiji*, *Icy* e *CellProfiler*), infatti, falliscono nel gestire efficacemente immagini generate da scanner *Whole slide* a causa della loro dimensione (in alcuni casi tali immagini decompresse possono contenere 40GB di dati); molto spesso l'unica possibilità di elaborarli è di effettuare un *downsampling* o *cropping* dell'immagine, oppure di utilizzarne piccole sezioni (*tiles*) per volta. Grazie a una completa automatizzazione dell'operazione di lettura con *tiles*, QuPath consente la visualizzazione di file whole-slide anche su macchine dalle modeste prestazioni; come in tutti gli altri software di *digital imaging*, inoltre, è possibile aggiungere annotazioni e ROI (*Region of Interest*) su porzioni di immagini, oltre che consentire la loro modifica e sezione. Oltre a ciò, lo strumento fornisce anche una serie di strumenti utili alla classificazione e individuazione di punti di interesse all'interno delle immagini, utilizzando anche algoritmi basati sul machine learning; a tal fine, si evidenzia come l'aggiunta di annotazioni *object-based* supportate dal programma facilitano enormemente l'operazione di learning per questi algoritmi (sarà infatti possibile creare legami di ereditarietà tra di esse).

3.2 Sviluppo di moduli addizionali

Le funzionalità di ImageJ possono essere ampliate tramite lo sviluppo di tre categorie di strumenti:

- **Macro**: le macro sono programmi atti a automatizzare una serie di operazioni di ImageJ. Data la loro semplicità e limitatezza (non possono essere infatti programmate per svolgere operazioni al di fuori di quelle già presenti nel core di ImageJ), vengono spesso utilizzati per eliminare le procedure più ripetitive dell'image processing. Le macro possono essere programmate attraverso l'uso di un linguaggio di scripting dedicato chiamato *IJM* [14] (The ImageJ Macro Language) oppure registrate attraverso una serie di strumenti dedicati del programma.

- **Script**: Qualora la semplice reiterazione di istruzioni sia troppo limitante, ImageJ permette l'esecuzione di piccole porzioni di codice (per l'appunto, *scripts*) in vari linguaggi predefiniti (tra i quali ricordiamo *Groovy*, *Python*, *Javascript* e *Ruby*) Tutti gli script possono essere eseguiti direttamente nella stessa istanza di ImageJ, grazie all'incluso *Script Editor*; ad eccezione dei programmi scritti in Clojure, tutti gli script sono interpretati a run-time, sacrificando così le prestazioni per i task più impegnativi.

- **Plugin**: nel caso invece in cui sia necessaria una completa flessibilità all'interno del framework e una performance di esecuzione pari a quella nativa, ImageJ permette l'inclusione di interi moduli applicativi chiamati Plugins. In termini generali, i Plugins si rendono obbligatori qualora sia necessario renderizzare nuove interfacce utente od eseguire elaborazioni che richiedono computazioni su molteplici thread.

L'elevata interattività progettata per DS4H Image Alignment (aggiunta/rimozione di immagini allo stack in maniera dinamica, impostazione dei corner points o loro rimozione, possibilità di preview delle immagini ecc.) lo rendono inadatto allo sviluppo tramite macro IJM; inoltre, la prospettiva di ottimizzare le elaborazioni di co-registrazione tramite multi-threading e la necessità di creare UI dedicate hanno portato il team a scegliere i Plugin come categoria di strumenti per questo progetto.

3.2.1 SciJava Framework

SciJava è un framework che standardizza la creazione di estensioni in vari programmi di ambito biomedico (tra i quali ricordiamo ImageJ, Fiji, CellProfiler, Icy ecc.) in modo da garantire completa interoperabilità tra programmi di diversa natura e provenienza. Nel dettaglio, SciJava impone che la creazione di moduli applicativi aggiuntivi (in ImageJ chiamati Plugins) debba avvenire attraverso l'implementazione di alcune interfacce predefinite, in modo da fornire una base comune adatta a tutti i programmi che adottano lo standard *SciJava*. A titolo di esempio, si

noti come la creazione di un semplice plugin in ImageJ debba necessariamente avvenire tramite l'aggiunta dell'annotazione `@Plugin`:

```
@Plugin(type = Command.class)
public class HelloWorld implements Command {
    public static void main(final String... args) {
        final ImageJ ij = new ImageJ();
        ij.launch(args);
        ij.command().run(HelloWorld.class, true);
    }

   @Override
   public void run() {
       // ...
   }
}
```

Un plugin così composto potrà essere aggiunto e riconosciuto automaticamente dall'applicazione (ad es. ImageJ) grazie al suo inserimento nel `context`, cioè lo stato globale dell'applicazione mantenuto da *SciJava*. Una volta inserito al suo interno sarà poi possibile effettuare operazioni di *Plugin discovery*, cioè di ricerca e riutilizzo degli altri plugin riconosciuti. La tipologia del plugin potrà essere specificata tramite l'inclusione del parametro `type` all'interno dell'annotazione; tra le principali tipologie fornite da *SciJava* (e pertanto disponibili all'utilizzo con ImageJ) ricordiamo:

- **Service**: all'interno dello *SciJava* framework i `Service` fungono da metodi di *utility* all'interno dello stesso `context` dell'applicazione. Eventuali inter-dipendenze tra gli stessi `Service` possono essere risolte tramite un meccanismo di *dependency injection* che il framework fornisce.

- **Op**: i plugins di tipologia `Op` sono dedicati alla creazione di algoritmi di image processing riutilizzabili. La loro conformazione garantisce infatti facilità d'uso e di estensione (l'interfaccia dedicata permette di creare facilmente pipelines di Ops sequenziali) e performance (grazie all'integrazione con la libreria *imgLib2*).

- **Command**: i plugin che invece richiedono un'interazione con l'utente e la visualizzazione di un risultato a schermo ricadono nella categoria `Command`. Sono generalmente la soluzione adatta qualora si debbano fornire funzionalità o algoritmi a utenti di ImageJ; è possibile comunque lanciare la loro esecuzione in modalità *headless*.

All'interno dello *SciJava* framework sono presenti anche altre tipologie di plugin (tra cui ricordiamo gli `Image Formats` usati ad esempio dal framework `SCIFIO`, i `Converters` e gli

`Input Preprocessors`) ma possono essere considerati non in linea con gli obiettivi di *DS4H Image Alignment* e quindi non verranno approfonditi. Data la necessità di mostrare un interfaccia utente completa all'utente per questo progetto, si è deciso di adottare la tipologia di plugin *Command* per lo svolgimento di questo progetto.

3.2.2 SciJava Project Conventions

Lo sviluppo di plugins compatibili *SciJava* richiede la conformazione ad alcuni principi generali di gestione del progetto; sebbene alcuni di essi non siano obbligatori per il loro corretto funzionamento sono comunque consigliati così da mantenere una certa uniformità.

Uno dei principi obbligatori a cui i plugin si devono adeguare è l'utilizzo di *Maven* [15] come tool di gestione del progetto. Grazie ad esso è possibile:

- Risolvere agevolmente (e in maniera del tutto automatica) le sue dipendenze esterne;
- Definire in maniera standard il suo versioning;
- Incapsulare il tutto in un file .jar eseguibile compatibile con il programma di destinazione (in questo caso ImageJ).

In altri termini, *Maven* è un programma eseguibile che permette di automatizzare una serie di operazioni di gestione del progetto che spaziano dalla risoluzione delle dipendenze alla creazione di file eseguibili (sotto forma di file .jar); queste operazioni sono evocabili in qualsiasi momento tramite riga di comando una volta posizionati all'interno della cartella del progetto e installato *Maven* nel sistema operativo:

- `mvn clean`: questo comando rimuove tutti i file generati dalle precedenti build di *Maven* e riporta il progetto a uno stato "pulito" e indipendente da qualsiasi tipo di cache.
- `mvn compile`: questo comando lancia la compilazione dei sorgenti del progetto e riporta eventuali errori riscontrati durante la sua esecuzione. Ogni esecuzione di questo comando ricontrollerà inoltre le dipendenze esterne remote indicate dal file `pom.xml` e, se più aggiornate rispetto alle locali, le sostituirà prima di lanciare la compilazione.
- `mvn install`: questo comando compila e crea un file eseguibile del progetto all'interno della cartella del sistema `UserHomeDirectory/.m2/repository/` (supponendo un operazione di `install` per DS4H-Image-Alignment su macchina windows 10 e nome utente "MarioRossi", l'output del progetto sarà riportato in `C:\Users\MarioRossi\.m2\repository\DS4H-Image-Alignment`).

Le modalità di esecuzione di questi comandi possono essere modificate attraverso la creazione di un file di configurazione `pom.xml` posto all'interno della root del progetto. La sua predisposizione è facilitata dall'adozione del principio *Convention Over Configuration* [16] che il tool

adotta per la maggior parte delle opzioni disponibili: l'uso estensivo di *implicit defaults* richiede infatti di specificarne soltanto un esiguo numero valori obbligatori (come ad esempio l'id di progetto e le dipendenze esterne).Un esempio di file `pom.xml` compatibile *Maven* può essere estratto da quello utilizzato per *DS4H Image Alignment*:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    ...
    <modelVersion>4.0.0</modelVersion>
   <groupId>DS4H-Image-Alignment</groupId>
   <artifactId>DS4H-Image-Alignment</artifactId>
   <version>1.0.4</version>

   <parent>
     <groupId>org.scijava</groupId>
     <artifactId>pom-scijava</artifactId>
       <version>25.0.0</version>
     <relativePath />
   </parent>

    <repositories>
        <repository>
            <id>imagej.public</id>
            <url>http://maven.imagej.net/content/groups/public</url>
        </repository>
    </repositories>

   <dependencies>
     <dependency>
        <groupId>net.imagej</groupId>
        <artifactId>imagej</artifactId>
     </dependency>
   </dependencies>
</project>
```

In questo esempio sono state rimossi tutti i componenti non obbligatori e che il tool non può determinare come *implicit default*, dato che sono caratteristici di ogni progetto: si riporta infatti la sezione `<groupId>` (necessaria per identificare univocamente il plugin), il parametro `<parent>` (utilizzata per ereditare le dipendenze e le configurazioni già predisposte per *SciJava* e rendere il progetto compatibile con ImageJ) ed infine `<dependencies>` (utilizzato per definire

altre dipendenze esterne che il plugin deve includere per il suo funzionamento). Fortunatamente, l'inclusione di *Maven* all'interno di un progetto Java viene notevolmente semplificata dai moderni IDE compatibili Java, come ad esempio Intellij Idea o Eclipse: a titolo di esempio, l'esecuzione di comandi tramite *command line interface* non è obbligatoria perché già fornita da questi programmi, e sarà così possibile eseguire `mvn install`, `mvn clean` ecc. tramite delle GUI fornite appositamente.

Le linee guida sullo sviluppo di questi progetti prevede anche l'utilizzo di *Git* [17] come strumento di controllo del codice sorgente: questa scelta si ricollega alla preferenza di utilizzo di *GitHub* [18] (uno dei più famosi siti di hosting per repository *Git)* per la gestione dell'*issue tracking* (ogni problema o proposta sul plugin deve essere discussa sull'apposito strumento del repository).

3.2.3 Esempio di sviluppo di plugin ImageJ

Come già anticipato nelle sezioni precedenti, l'intero ecosistema di ImageJ si basa sui contributi open-source di una folta schiera di programmatori, biologi e scienziati. Sebbene i *core components* siano mantenuti con una certa regolarità da team dell'università di Wisconsin-Madison e di Dresda, lo stesso non si può dire per contenuti *community-based* come la wiki dedicata allo sviluppo di nuovi plugins. Per questo motivo, riteniamo utile l'inserimento di sottosezione che possa fungere da *reference* per la inizializzazione, sviluppo e installazione di moduli aggiuntivi in ImageJ Fiji. Nelle pagine seguenti verranno illustrate le procedure necessarie per lo sviluppo di un plugin che, dato un file immagine in input, ne restituisce una copia in scala di grigi.

Strumenti necessari

Innanzitutto, lo sviluppo di un plugin ImageJ non può avvenire senza l'installazione su una macchina dedicata di:

- **Java SDK**: essendo i plugin scritti in Java, è indispensabile che nel sistema sia installata il *Source Development Kit* più recente di Java.
- **Maven**: dato che le procedure di compilazione, installazione e testing devono essere eseguite tramite *Maven*, è indispensabile che esso venga aggiunto al sistema operativo (in ambienti windows ciò sarà possibile tramite l'aggiunta dell'eseguibile del programma alla variabile $PATH$).

Concettualmente, i Plugin per ImageJ potrebbero essere sviluppati anche senza l'ausilio di IDE specifici, tuttavia si consiglia caldamente il loro utilizzo data la loro quasi universale integrazione con *Maven*: in questo modo, il lancio di comandi specifici di build, testing e installazione

sarà semplificata. Più specificatamente, il team ha optato per utilizzare *IntelliJ Idea* [19] di *JetBrains* come strumento di sviluppo e compilazione, ritenendolo moderno e adatto alle esigenze richieste.

Inizializzazione e Build

La procedura per la creazione di un nuovo progetto compatibile *SciJava* con IntelliJ IDEA versione 2018.3.5 può essere così riassunta:

1. Creare un nuovo progetto Maven tramite `File` → `Project` → `Maven`.
2. All'interno del file `pom.xml` specificare le sezioni obbligatorie analogamente a quelle indicate dal Listato 3.2.2. Si ricoda che come `GroupId` si dovrà inserire un nome che identifichi univocamente il progetto mentre `ArtifactId` servirà a specificare il nome del file .jar possiederà ad ogni compilazione
3. Creare una nuova classe di partenza del progetto implementando l'interfaccia specifica della tipologia di plugin che si vorrà creare, in maniera analoga al Listato 3.2.1.

Una volta fatto ciò sarà necessario configurare la modalità di Debug dell'IDE, navigando il menù tramite `Run` → `Edit Configurations`. Da qui basterà aggiungere una voce `Application` con il valore di `Main class` corrispondente alla classe creata nel punto 3 (che, seguendo il listato 3.2.1, sarà pari ad `HelloWorld`).

La procedura di Build per finalità di distribuzione del plugin richiede necessariamente l'utilizzo di *Maven*, precisamente tramite il comando `nvm install`. Utilizzando IntelliJ IDEA, la procedura completa di distribuzione può essere così riassunta:

1. Aprire il progetto tramite *IntelliJ Idea* e apportare le modifiche al codice ritenute necessarie.
2. Nel menù dedicato a *Maven*, lanciare un'operazione di `install` e attendere il completamento dell'operazione (essa avrà successo solo nel caso in cui non siano presenti errori di compilazione). I file compilati saranno disponibili nella cartella `target` del progetto o, alternativamente, potranno essere ricavati dalla cartella predefinita di build di *Maven*, cioè `UserHomeDirectory/.m2/repository/`
3. Il plugin potrà essere installato in qualsiasi istanza ImageJ semplicemente trascinando all'interno della finestra dell'applicazione l file .jar compilato o utilizzando il menù *Plugins* → `Install`.

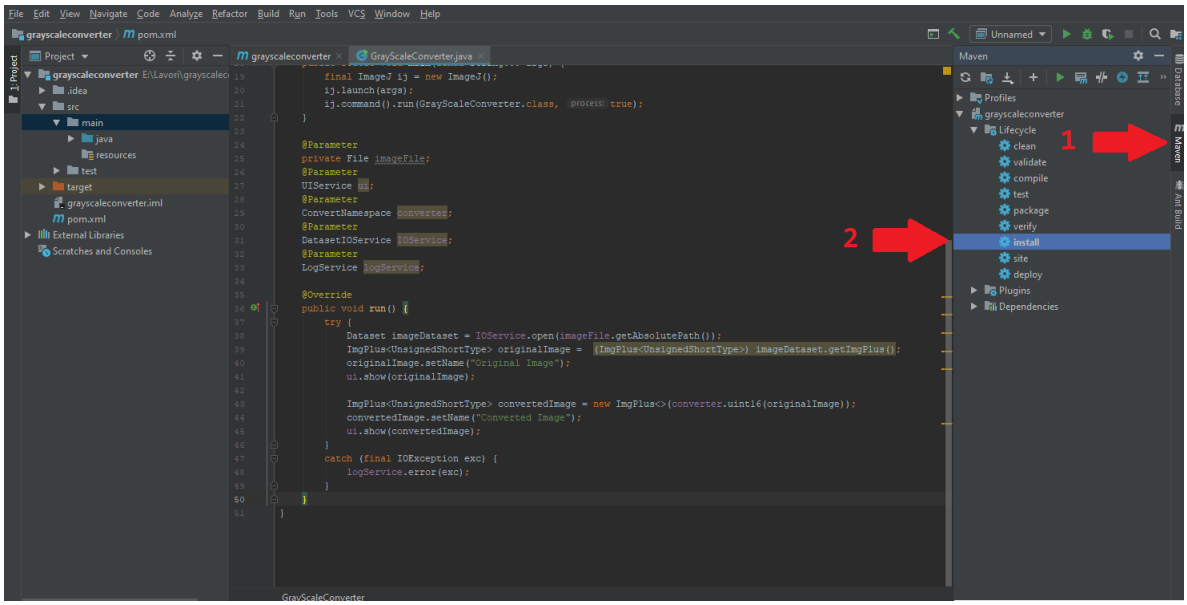

Figura 3.1: Procedura di `mvn install` su *IntelliJ Idea*

Sviluppo del plugin

Come indicato dalla sottosezione 3.2.1, i plugins ImageJ possono essere divisi in una serie di categorie predefinite (tra le principali ricordiamo `Commands`, `Op` e `Services`). Dato che l'obiettivo principale del plugin presentato in questa sezione è quello di convertire un immagine nel corrispettivo in scala di grigi, si ritiene che la più adatta sia la `Op`. Il codice presentato di seguito utilizza le nuove API sperimentali ImageJ 2.x, per il quale non è ancora consigliato l'adozione dato il loro frequente *refactoring* da parte del team di sviluppo:

```
@Plugin(type = Op.class)
public class GrayScaleConverter implements Op {
    public static void main(final String... args) {
        final ImageJ ij = new ImageJ();
        ij.launch(args);
        ij.command().run(GrayScaleConverter.class, true);
    }
    @Parameter
    private File imageFile;
    @Parameter
    UIService ui;
    @Parameter
    ConvertNamespace converter;
    @Parameter
    DatasetIOService IOService;
```

```
    @Parameter
    LogService logService;

    @Override
    public void run() {
        try {
            Dataset imageDataset = IOService.open(imageFile.getAbsolutePath());
            ImgPlus<UnsignedShortType> originalImage =
                (ImgPlus<UnsignedShortType>) imageDataset.getImgPlus();
            originalImage.setName("Original Image");
            ui.show(originalImage);

            ImgPlus<UnsignedShortType> convertedImage = new
                ImgPlus<>(converter.uint16(originalImage));
            convertedImage.setName("Converted Image");
            ui.show(convertedImage);
        }
        catch (final IOException exc) {
            logService.error(exc);
        }
    }
}
```

Si noti l'inserimento di parametri grazie alla *Dependency Injection* garantita dal framework (annotazione `@Parameter`) e il largo uso di *Generics* rispetto alle API ImageJ 1.x.

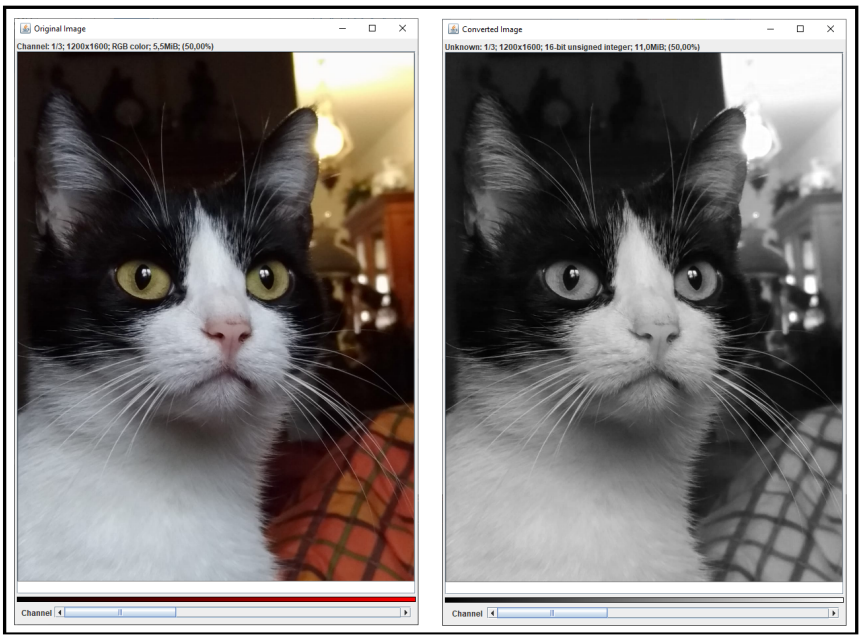

Figura 3.2: Output prodotto dal plugin di esempio, `GrayScaleConverter`

Capitolo 4

Caso di studio: DS4H Image Alignment

Questo capitolo descrive le scelte architetturali impiegate per lo sviluppo di DS4H Image Alignment, un tool creato per permettere l'allineamento di immagini microscopiche bidimensionali. La sezione 4.1 presenta una panoramica dei problemi che possono insorgere con l'impiego di tecniche di immunoistochimica sequenziale e gli obiettivi generali del progetto. La sezione 4.2 riepiloga l'insieme delle scelte architetturali adottate, descrivendone nel dettaglio l'insieme delle interfacce e classi sviluppate. La sezione 4.3 introduce le modalità di importazione dei campioni nel tool, presentando un formato file standard per l'archiviazione di immagini microscopiche bidimensionali, cioè `OME-XML`. La sezione 4.4 presenta le principali modalità di co-registrazione delle immagini adottate dal software e le possibilità di personalizzazione da parte dell'utente. Infine, la sezione 4.5 riepiloga l'insieme di punti critici del programma e ne propone alcune possibili soluzioni.

4.1 Analisi del problema

Tessuti e cellule umane sono tessuti a componente maggioritaria acquosa, pertanto risultano tipicamente trasparenti alla visione ad occhio nudo. Per poter caratterizzare le cellule, vengono quindi tipicamente utilizzati coloranti. In particolare, in immunoistochimica è normale procedura analizzare il campione sfruttando coloranti specifici per alcuni tipi di cellule, al fine ad esempio di identificare cellule tumorali, oppure alcune regioni subcellulari, per capire ad esempio effetti di farmaci o trattamenti. Tuttavia, a causa della dimensione ridotta dei componenti cellulari di interesse in confronto alla dimensioni delle particelle del colorante, non è sempre possibile utilizzare in parallelo un numero elevato di marcatori.

È quindi procedura standard fare ricorso alla immunoistochimica sequenziale, dove si colora il campione con un numero limitato di marcatori, si acquisiscono immagini con ausilio di un microscopio, si eliminano i marcatori attraverso dei lavaggi, si marcano le cellule con un nuovo set di marcatori, si acquisiscono nuove immagini, etc. Questa procedura iterativa permette quindi l'analisi accurata di varie caratteristiche delle cellule e regioni sub-cellulari, in maniera sequenziale. Tuttavia, per avere una visione di insieme del campione, occorre alla fine allineare le immagini per studiare la colocalizzazione dei segnali avendo un unico riferimento. Questo non è un compito semplice, perchè durante le diverse fasi di colorazione il campione potrebbe subire deformazioni elastiche o addirittura permanenti. Oltretutto, ad oggi non esiste un software gold standard per allineare sequenze di immagini che potrebbero avere anche dimensioni differenti.

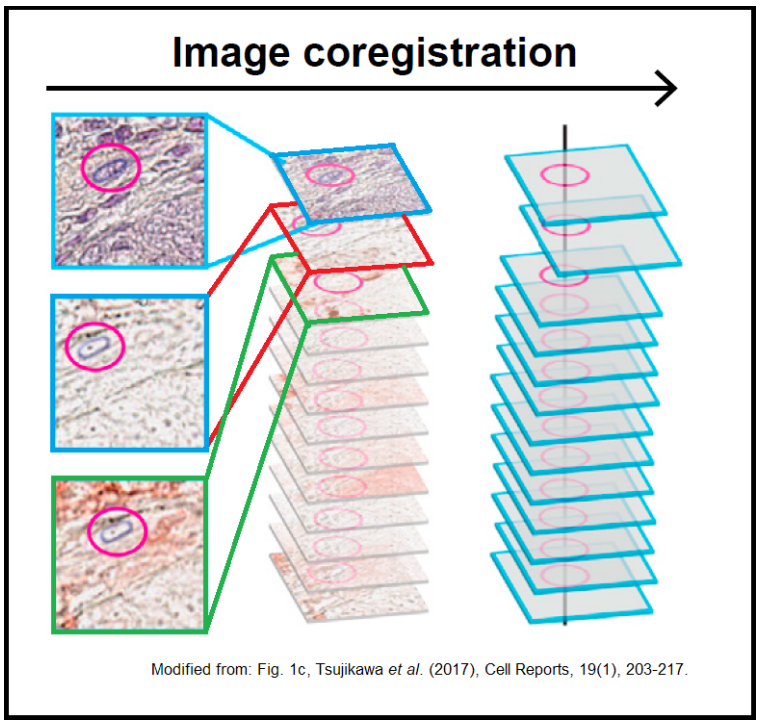

Figura 4.1: Operazione di co-registrazione di immagini microscopiche bidimensionali.

Lo scopo di DS4H Image Alignment è colmare questa lacuna, fornendo all'operatore una soluzione estremamente user-friendly per risolvere il problema dell'allineamento di immagini.

4.1.1 Obiettivi

Lo scopo di questo progetto di tesi è quello di fornire alla comunità uno strumento che consenta l'elaborazione di immagini microscopiche di vari formati (supportando quindi anche i formati file proprietari, per il quale è spesso necessario utilizzare tool di conversione di terze parti) e di permetterne la co-registrazione delle stesse attraverso l'impostazione di una serie di punti di riferimento *landmarks*. Una descrizione più dettagliata dei requisiti è deducibile dal seguente diagramma dei casi d'uso:

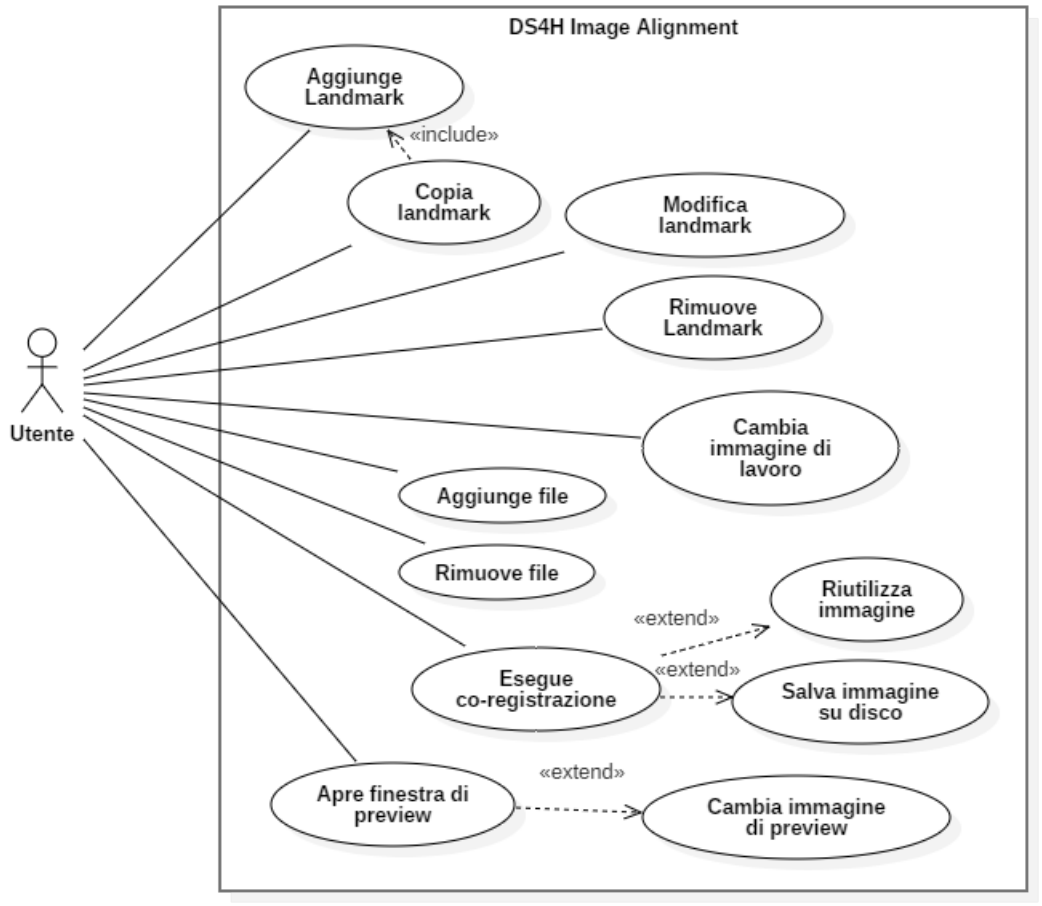

Figura 4.2: Diagramma dei casi d'uso del progetto.

Si tenga presente che i modelli di trasformazione previsti sono designati per immagini bidimensionali: l'applicazione di questi modelli su immagini tridimensionali non è prevista allo stato corrente di sviluppo di DS4H.

4.2 Architettura del sistema

La struttura di progetto di *DS4H Image Alignment* segue una concezione *Package By Feature* in un'ottica di una migliore coesione, modularità e decoupling tra i vari package presenti: le principali feature previste dell'applicazione sono state infatti incluse in un package dedicato all'interno del più generale package `DS4H`. Questo approccio si può considerare completamente diverso da un stile più tradizionale come il *Package by Layer* dove ogni classe java presente sarebbe stata suddivisa in base alla sua categoria di appartenenza (`model`, `view`, `service` ecc.).

Il pattern MVC è stato ampiamente implementato per gestire tutte le interazioni tra utente e dati da elaborare: a titolo di esempio, la classe `MainDialog` (incaricata di creare l'interfaccia utente della finestra principale) e di gestirne le interazioni non può alterare in nessun modo la parte `model` dell'applicazione (ogni evento viene notificato al controller dedicato nonché classe principale del programma, cioè `ImageAlignment.java`).

Di seguito viene riportata una descrizione generale dei componenti del sistema, suddividendoli in Viste, Controller e Modelli.

4.2.1 Gestione delle view

Come vedremo nelle prossime sezioni, tutte le immagini visualizzate su ImageJ sono incapsulate all'interno di una classe fornita dal framework chiamata `ImagePlus`; ogni istanza di tali classi può essere mostrata direttamente (tramite la direttiva `show()`) oppure inclusa all'interno di una finestra personalizzata (che deve estendere la classe `ImageWindow`). Questa "rigida" filosofia è dovuta principalmente alla necessaria compatibilità con il framework *SciJava*: ogni immagine scambiata con un programma che implementa la libreria *SciJava-Common* sarà quindi compatibile, perché utilizza la stessa classe per la sua visualizzazione. A livello pratico, questa particolarità comporta un diffuso utilizzo di *Dialogs* (finestre modali) per l'apertura di immagini di ImageJ; in questo modo ognuna di esse può essere modificata ed elaborata al suo interno, garantendo un completo supporto con i tool di editing del programma (ad esempio, strumenti di disegno, scala, selezione ecc.).

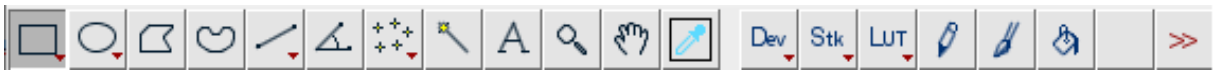

Figura 4.3: Alcuni degli strumenti predefiniti di ImageJ.

Il team di sviluppo di DS4H Image Alignment ha seguito così questa filosofia implementando la totalità delle interfacce utente come *Dialogs* (classi che estendono, a vari livelli di gerarchia, la classe `Frame`); tali classi possono essere così riassunte:

- `MainDialog`: finestra principale dell'applicazione, è incaricata di mostrare all'utente l'immagine attuale di lavoro e di permettere l'inserimento di *Corners* (cioè di landmarks) per la co-registrazione di immagini. Inoltre, la *dialog* consente l'accesso a tutte le altre funzionalità previste dal sistema: aggiunta di immagini, finestra di preview, rimozioni di File ecc.

- `PreviewDialog`: funzionalità che mostra una finestra di anteprima delle immagini inserite nel sistema. Il suo scopo è di facilitare l'inserimento di *Corners Points* da parte dell'utente, segnalando dove i markers siano già stati inseriti (ogni punto, se selezionato nella `MainDialog`, verrà evidenziato su entrambe le finestre).

- `RemoveDialog`: funzionalità che permette di rimuovere file di immagini già inseriti all'interno del sistema e di scartarne i *Corner Points* associati. Ogni file elencato è corredato dalle anteprime delle immagini che lo compongono.

- `AlignDialog`: finestra che viene riportata a schermo ogni una volta che la co-registrazione delle immagini inserite viene completata. Attraverso di essa è possibile salvare manualmente l'immagine in file dedicato `TIFF` specificandone la destinazione di scrittura, oppure riutilizzare la stessa immagine per un nuovo ciclo di co-registrazione in *DS4H Image Alignment*.

- `AboutDialog`: finestra di presentazione del progetto, comprendente gli indirizzi mail dei contributori e licenza associata al software.

Tutte le build di ImageJ (eccezion fatta per ImageJFX, che utilizza JavaFX) utilizzano `Swing` come framework per la gestione delle UI; ne consegue che tutte le funzionalità di *DS4H* sono state codificate con tale framework.

4.2.2 Controller

All'interno del progetto *DS4H Image Alignment* la classe che ricopre un ruolo centrale nella gestione del flow applicativo utente e delle sue interazioni è sicuramente `ImageAlignment.java`. Innanzitutto, fungendo da controller per i *models* e le *views* dell'applicazione, tutti gli eventi generati dall'utente (ad esempio come l'aggiunta di un *Landmark*, cambio di immagine, eliminazione di un file...) sono elaborati da `ImageAlignment`. Cioè è reso possibile attraverso l'implementazione di una serie di interfacce specifiche per ogni dialog, chiamate `Listeners`: a titolo di esempio, il monitoraggio degli eventi generati dal *MainDialog* è possibile solamente implementando l'interfaccia `OnMainDialogEventListener`.

Un altro compito di primaria importanza di questa classe è la sua configurazione come punto di partenza del plugin, specificando la sua tipologia e path di accesso in maniera compatibile con gli standard *SciJava* (tramite le già trattate annotazioni `@Plugin()`). Inoltre, la classe si deve assicurare di porre l'esecuzione del plugin all'interno del `context` *SciJava* principale

dell'applicazione: in caso contrario, potrebbe non essere possibile intercettare alcuni eventi o interazioni utente più specifiche come l'utilizzo di tool esterni o cambi di parametri (cambio del tipo di immagine da RGB a grayscale, ridimensionamento, sezione ecc.).

```
@Plugin(type = Command.class, headless = true,
      menuPath = "Plugins>Registration>DSH4 Image Alignment")
public class ImageAlignment extends AbstractContextual implements Plugin,
    OnMainDialogEventListener, OnPreviewDialogEventListener,
    OnAlignDialogEventListener, OnRemoveDialogEventListener {
   public static void main(final String... args) {
      ImageJ ij = new ImageJ();
      ij.ui().showUI();
      ImageAlignment plugin = new ImageAlignment();
      plugin.setContext(ij.getContext());
      plugin.run();
   }
```

4.2.3 Data Models

La sezione *models* di *DS4H Image Alignment* è principalmente incentrata sulla gestione dei file di immagini da parte dell'utente e dei ROI (landmarks) aggiunti in vista della co-registrazione.
Innanzitutto, è bene premettere che l'interfaccia principale dell'applicazione (cioè quella dedicata all'aggiunta di *Landmarks* tramite la `MainDialog`) è stata studiata per rendere l'apertura e l'elaborazione delle immagini il più semplice possibile; pertanto, tutti i file aggiunti all'applicazione durante la sessione utente vengono concatenati e mostrati come se fossero appartenenti ad un unico stack. A tal fine, è stata sviluppata una classe dedicata alla gestione dell'insieme di file caricati all'interno dell'applicazione, chiamata `ImagesManager`; tramite essa è infatti possibile "scorrere" tra le possibili immagini aggiunte al sistema (essa infatti implementa l'interfaccia java `ListIterator`, specifica per la navigazione di collezioni di oggetti) e inserire/rimuovere nuove collezioni (sotto forma di oggetti `ImageFile`). Ogni istanza di `ImageFile` permette l'estrazione delle immagini grazie a dei `BufferedImageManager` forniti dalla libreria `bio-formats`; oltre a ciò, la classe estrae un insieme di informazioni e metadati utili come ad esempio numero di immagini, dimensione massima ecc. Inoltre, ogni istanza di `ImageFile` contiene una collezione di `RoiManager`, cioè classi che mantengono una lista di ROI aggiunti dall'utente (che saranno così estratti durante la fase di co-registrazione). Infine, la visualizzazione dell'immagine a schermo viene garantita invece dalla classe `BufferedImage`, principalmente grazie all'estensione della classe `ImagePlus` di ImageJ. Come si può notare dallo schema,

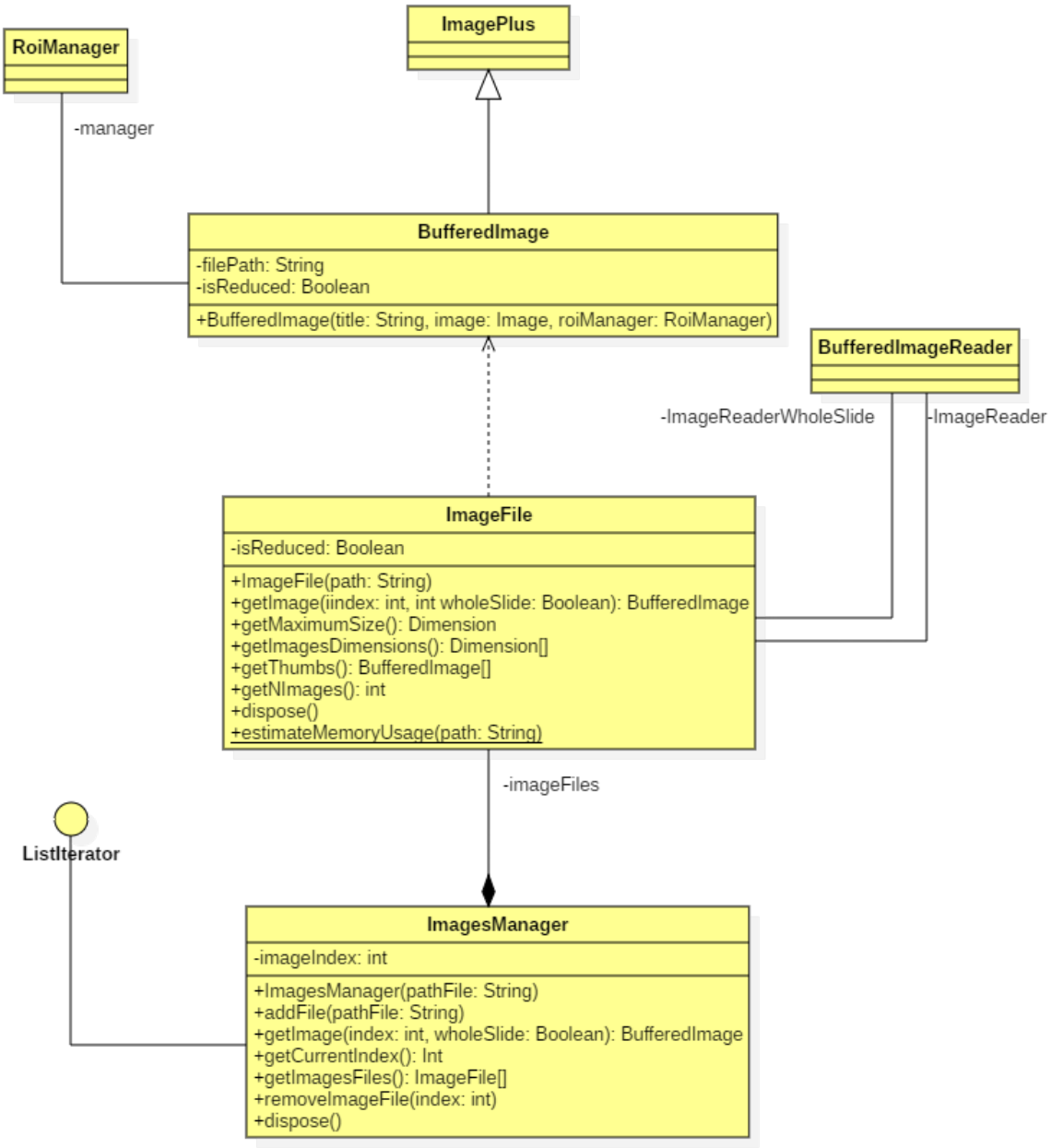

Figura 4.4: Diagramma UML delle classi dati principali di DS4H.

ogni istanza viene generata a richiesta dal rispettivo `ImageFile`: in questo modo si riduce la quantità di memoria utilizzata in quanto ne potrà essere attiva soltanto una alla volta.

4.3 Importazione di immagini microscopiche bidimensionali

Uno dei requisiti fondamentali assegnati allo sviluppo di *DS4H Image Alignment* è la garanzia di un supporto completo alla maggior parte dei formati immagine correntemente utilizzati nella microscopia biomedicale. Nonostante siano stati proposti come standard vari formati [22] immagine nel corso del tempo (ad esempio `Analyze`, `Nifti`, `Minc` e `Dicom`), la loro adozione

da parte dei principali produttori dei dispositivi di acquisizione biomedicale è stata insufficiente. Al momento, infatti, si contano almeno [23] 150 differenti formati immagine comunemente utilizzati dal personale in ambito di ricerca; la compatibilità da parte di programmi di editing di immagini, tuttavia, è garantita solo per un numero molto ridotto di essi (a titolo di esempio, ImageJ garantisce il supporto nativo a solo circa 15 di questi).

Una soluzione frequentemente adottata da parte dei team di ricerca è quella di incorrere in conversioni (molto spesso grazie all'utilizzo di tools esterni *open-source* progettati appositamente) che trasformano i formati immagine meno frequentemente utilizzati in altri più popolari (ad esempio, file `.TIFF`). Così facendo, tuttavia, la maggior parte dei metadata allegati (che includono ad esempio la descrizione dell'esperimento effettuato, la metodologia di acquisizione, data, utente ecc.) vengono modificati o addirittura eliminati dal processo di conversione. È evidente quindi che questo approccio non si può considerare sostenibile, dato l'enorme sviluppo che l'imaging biomedicale si prospetta di ricevere [24] negli anni a venire: è necessario abbracciare soluzioni che possano essere compatibili con i futuri sviluppi del settore.

4.3.1 OME Data model

L'*Open Microscopy Environment* (OME) è un consorzio di università, laboratori di ricerca e aziende che collaborano con l'obiettivo di creare nuovi formati e software open-source per l'analisi biomedicale. Oltre alla creazione di prodotti del calibro di OMERO (uno stimato software web-based di editing di immagini scientifiche in ambito medico), uno dei più rimarchevoli contributi alla comunità scientifica di questo consorzio è senza dubbio la creazione di un nuovo standard per l'archiviazione e consultazione di immagini microscopiche bidimensionali, cioè *OME Data Model* [21][25].

Attraverso l'utilizzo di una ontologia capace di codificare i diversi formati dati e relazioni prodotte dalle operazioni di imaging più comuni, *OME Data Model* permette di fornire un elevata interoperabilità tra dispositivi di acquisizione diversi. L'ontologia è scritta in XML Schema (.xsd) ed è liberamente consultabile attraverso i canali ufficiali del consorzio.
Nel dettaglio, lo standard integra al suo interno i dati binari delle immagini e tutte le informazioni riguardanti la sua acquisizione, trasformazione e analisi (cioè i suoi metadati). Gruppi di immagini possono essere organizzati all'interno di oggetti `Datasets` (gruppi definiti dall'utente che racchiudono immagini tra loro interconnesse, come ad esempio sequenze prodotte da uno stesso esperimento) oppure `Projects` (collezioni di `Dataset`). Ulteriori suddivisioni delle immagini sono possibili attraverso l'utilizzo di `Plates`, `Screens`, `Wells` e `Samples`, ognuno con il proprio insieme di identificatori comuni e la propria gerarchia (ad esempio, possono essere assegnati più `Samples` allo stesso `Wells`). Qualora il set predefinito di oggetti non sia sufficiente a soddisfare le esigenze di un dispositivo di acquisizione, *OME Data Model* permette di

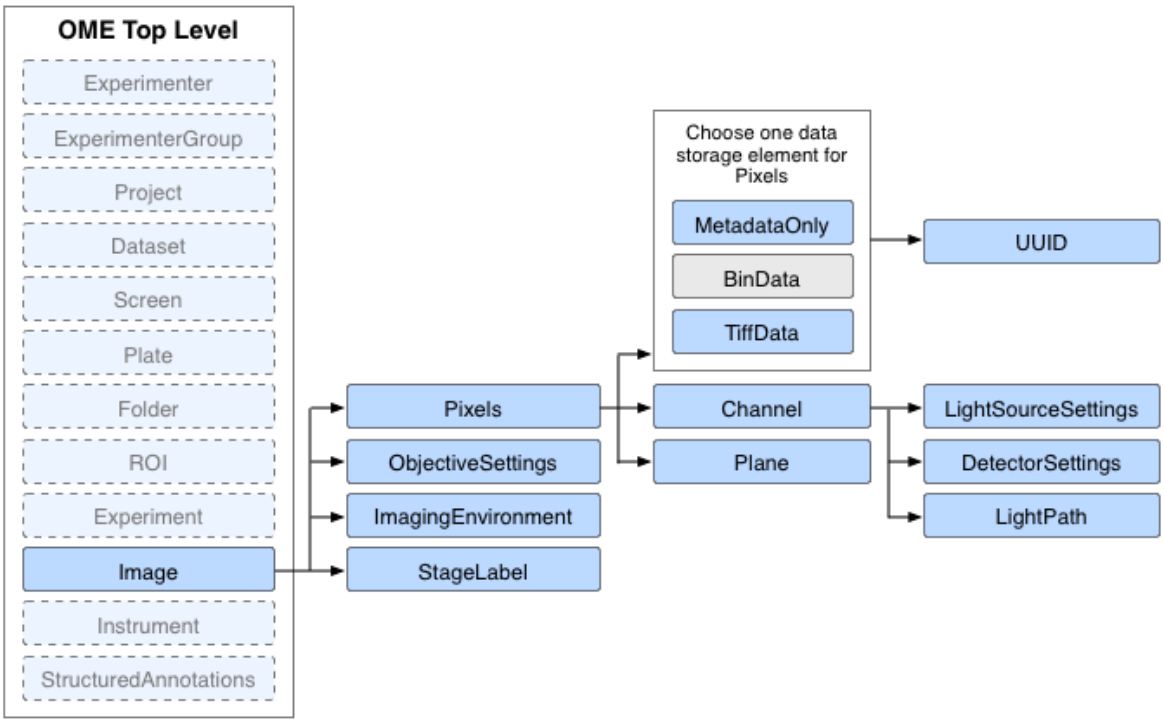

Figura 4.5: Estratto della sezione ”Image” dello standard.

aggiungere informazioni personalizzate attraverso le `StructuredAnnotations`.

OME-XML è il primo formato file reso pubblico dal consorzio che rispetta totalmente lo standard *OME Data Model*; la sua elevata flessibilità permette di immagazzinare agevolmente tutti i metadati ottenibili da sperimentazioni di microscopia. Un altro formato file compatibile di più recente introduzione è *OME-TIFF*, caratterizzato da un *TIFF* multi-plane e metadati *OME* all'interno del proprio header; l'elevata compatibilità dei programmi di digital imaging con i file di tipo *TIFF* rendono *OME-TIFF* molto utilizzato in ambito di imaging biomedicale.

4.3.2 Bio-formats

Bio-formats è una libreria sviluppata dall'*Open Microscopy Environment* (lo stesso consorzio autore dello standard *OME Data Model*) che garantisce il supporto I/O di più di 150 formati file immagine diversi. Molto conosciuta nell'ambito del biological imaging (il suo utilizzo è documentato su software come *Icy*, *CellProfiler* e *JCB Data Viewer*) viene spesso utilizzata in ImageJ in forma di Plugin per convertire formati immagine proprietari come `Aperio .SVS`, `ZEISS CZI` e `Hamamatsu ndpi` in formati standard come il già citato OME-TIFF.

La libreria (scritta in Java, ma che è compatibile con una serie di linguaggi come c++, Python e MATLAB) rende possibile l'apertura di un elevato numero di formati file immagine grazie all'utilizzo di classi di decodifica dedicate (ognuna derivante da un'unica classe astratta

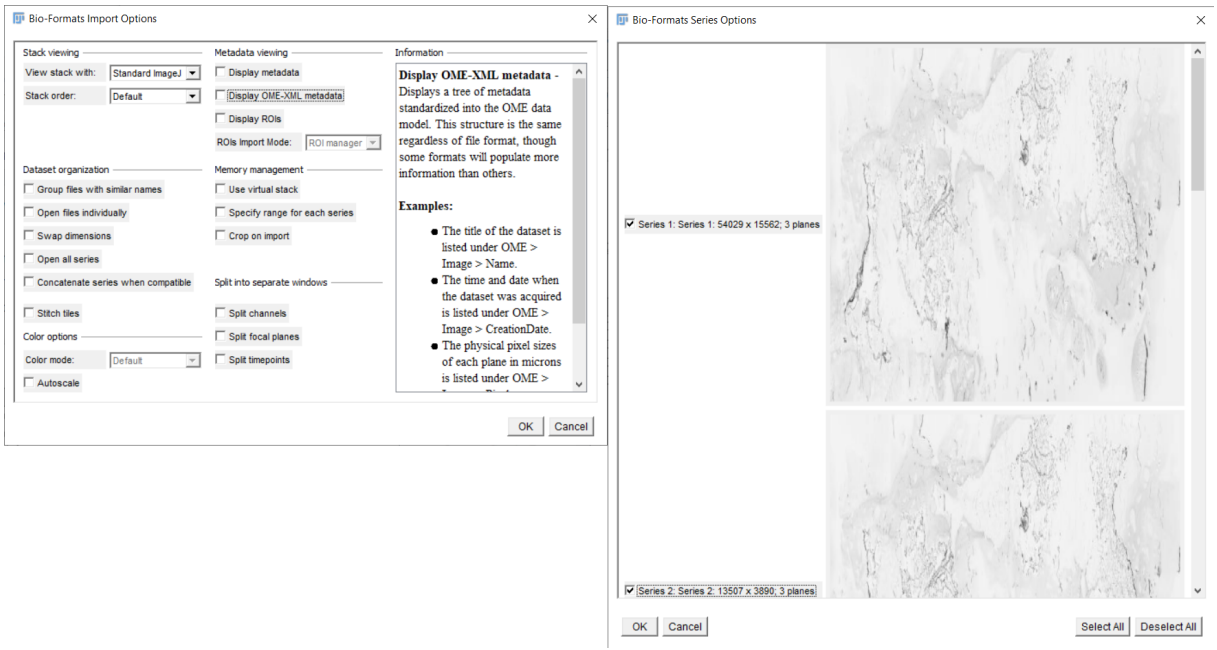

Figura 4.6: Interfaccia di importazione per il plugin Bio-formats di ImageJ.

chiamata `FormatReader`). Ogni `FormatReader` specializzato, infatti, conterrà una serie di algoritmi atti alla decodifica del particolare formato file (ad esempio, i già citati file `.SVS` vengono processati da un istanza di `SVSReader`) che verranno eseguiti in maniera del tutto automatica. Generalmente, la creazione di un `Reader` personalizzato non risulta particolarmente difficile per la parte binaria del file (cioè dei pixel dell'immagine, colori e canali), dato che si può considerare piuttosto simile anche per formati diversi; ciò che richiede un certo lavoro di adattamento è invece l'implementazione della logica di conversione dei metadati in un formato OME-XML compatibile, data la totale mancanza di uno standard uniforme tra i vari produttori. Una volta processato, il file sarà disponibile per la scrittura in una modalità compatibile con l'*OME Data Model* (solitamente in *OME-TIFF*).

Integrazione in DS4H Image Alignment

L'integrazione di *Bio-formats* in *DS4H Image Alignment* è fondamentale per garantire il funzionamento del plugin sulla quasi totalià delle immagini istologiche producibili in laboratorio. Il suo inserimento all'interno del progetto può avvenire attraverso la specificazione, all'interno del file `pom.xml`, di un `artifactId` denominato `bio-formats_plugins`. Una volta soddisfatte le dipendenze richieste l'apertura e la decodifica dei metadati dei file potrà venire così delegata alla libreria appena inserita; di seguito viene mostrata la procedura necessaria per processare un file `.svs` tramite la libreria:

```
// Impostazioni opzioni di importazione
ImporterOptions options = new ImporterOptions();
options.setVirtual(true);
```

```
options.setId("c:\example.svs");
options.setColorMode(ImporterOptions.COLOR_MODE_DEFAULT);
options.setSeriesOn(0, true);
ImportProcess process = new ImportProcess(options);
process.execute();

// Creazione di un reader di immagini tramite buffer di memoria
BufferedImageReader reader =
    BufferedImageReader.makeBufferedImageReader(importProcess.getReader());
return reader.openImage(imageIndex);
```

Tralasciando i parametri di configurazione per l'importazione del file (come ad esempio le modalità di gestione del colore, tramite `setColorMode()`), un aspetto di sicuro interesse a fini di studio è la selezione della risoluzione dell'immagine piramidale tramite il comando `setSeries()`. Come già trattato durante il Capitolo 1, infatti, l'elaborazione di immagini *whole-slide* comporta un costo computazionale che può essere ingente per macchine non specializzate in questo compito; a causa di ciò, la quasi totalità di tali file è composta da sotto-immagini di risoluzione inferiore in modo da facilitare la loro visualizzazione anche con risorse hardware normalmente insufficienti.Per ridurre il carico complessivo di memoria RAM utilizzata e velocizzare l'apertura delle immagini, quindi, *DS4H Image Alignment* utilizza due diversi `ImageReader` per uno stesso file di immagini: uno utilizzato per fornire all'utente un immagine di risoluzione ridotta per l'attività di *landmark placement* (cioè quella mostrata dal già citato `MainDialog`) e un altro per ricavare una *whole slide image* durante l'attività finale di co-registrazione (il cui risultato verrà mostrato nell'`AlignDialog`). L'utilizzo di un `ImageReader` (tramite le primitive `openImage()`, ad esempio) permetterà così di istanziare delle `ImagePlus`, cioè delle classi che consentono a utenti di ImageJ di visualizzare e modificare agevolmente immagini *Raster*.

4.4 Co-registrazione tramite landmarks

Una volta garantito il supporto alla quasi totalità di formati file per bioimmagini, il team di sviluppo di DS4H si è dedicato all'implementazione delle funzionalità di co-registrazione indicate dai requisiti raccolti in fase di analisi. Lo sviluppo di questa importante sezione del progetto è stata divisa in due parti: la prima si è incentrata sull'individuazione di modalità di inserimento e modifica di punti di controllo da parte dell'utente, la seconda sull'attuazione di algoritmi di co-registrazione basati sui *landmark* inseriti. La prossima sezione presenta le principali soluzioni tecniche adottate per la loro implementazione, oltre ad approfondire le modalità con il quale l'utente può modificare le modalità di elaborazione delle immagini.

4.4.1 Registrazione dei punti di controllo

Come anticipato dal Capitolo 2, l'applicazione di algoritmi di co-registrazione richiede l'impostazione di alcuni specifici punti di controllo (anche chiamati *landmarks*) su ogni immagine trattata. Nel caso di *DS4H Image Alignment*, la loro definizione è possibile generando dei ROI (*Region of Interest*) su ogni ogni punto interessato dall'utente, tramite la pressione del tasto "C" nella finestra `MainDialog`; fortunatamente, ImageJ predispone già nativamente il supporto alla loro aggiunta e modifica per ogni `ImagePlus` mostrata a schermo, e non è stato quindi necessario riprogettare completamente questa funzionalità.

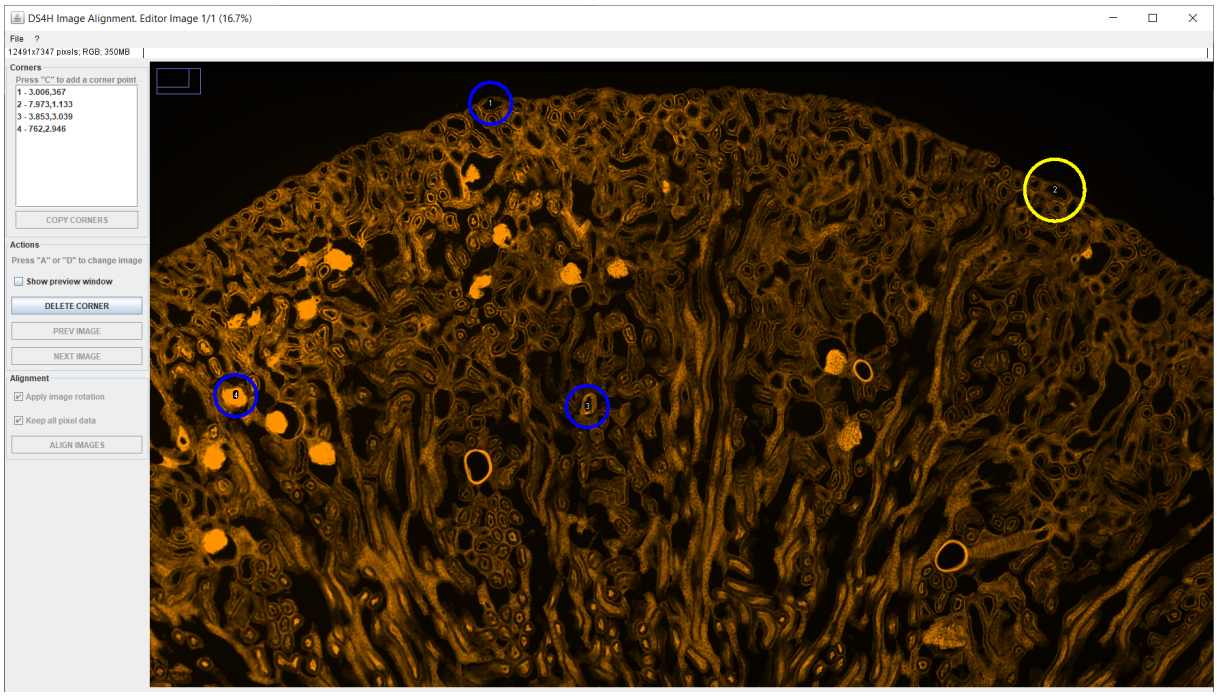

Figura 4.7: Registrazione dei punti di controllo per un immagine in fluorescenza Cy3.

Come indicato dalla Figura 4.4, ogni `BufferedImage` mantiene in memoria l'insieme dei punti inseriti dall'utente in un oggetto di gestione chiamato `ROIManager`, con il quale si rende possibile l'eliminazione e la modifica di ogni ROI aggiunto.

Come si può notare, la classe possiede anche un flag booleano denominato `isReduced`, il cui scopo è quello di indicare se l'immagine è stata soggetta a una riduzione della sua risoluzione a causa della sue dimensioni eccessive. In tal caso, come indicato dalla sezione 4.3.2, la grandezza effettiva del campione mostrato nella `MainDialog` sarà differente dalla dimensione del campione soggetto a co-registrazione (l'immagine viene infatti ricaricata utilizzando un livello differente della piramide): ne consegue che le coordinate dei ROI inserite non risulteranno proporzionali tra le due fasi di lavoro.Per questo motivo, la fase di registrazione prevede l'applicazione una funzione di "aggiustamento" dei valori di X e Y per ogni ROI elaborato, applicando una scala basata sulla differenza di dimensione di ogni asse.

4.4.2 Applicazione tramite Moving Least Squares

L'integrazione di algoritmi di co-registrazione in *DS4H Image Alignment* è stata possibile grazie all'utilizzo di una libreria che raccoglie un insieme di algoritmi utilizzati per la trasformazione, registrazione e interpretazione di immagini, cioè *mpicbg* [27]. La libreria, pienamente compatibile con la distribuzione di ImageJ/Fiji, è sviluppata e mantenuta dall'istituto *Max Planck Institute of Molecular Cell Biology and Genetics* da cui ne deriva l'acronimo. Come anticipato, la libreria contiene un vasto numero di algoritmi per applicazioni di *image processing* (ad esempio sono inclusi algoritmi di *feature trasform* come il SIFT); per le finalità di questo progetto si è deciso di utilizzare algoritmi basati su metodi MLS (*Moving Least Squares*) [26].

Una volta aggiunti un numero minimo di punti focali per ogni immagine (cioè tre), l'interfaccia del plugin informa l'utente che è possibile avviare l'operazione di co-registrazione; la selezione/deselezione di una serie di parametri presenti nel pannello di controllo permettono di cambiare radicalmente l'operazione di elaborazione effettuata sui campione:

- **Apply image Rotation**: questa opzione cambia il modello di trasformazione applicato sulle immagini da *Affine* (opzione abilitata) a semplice *Traslazionale* (opzione disabilitata).

  Come già trattato durante il Capitolo 2, esiste una profonda differenza tra queste due tipologie di trasformazioni: mentre la traslazionale consiste nel semplice spostamento sugli assi X e Y dell'immagine, la trasformazione affine effettua delle vere e proprie deformazioni atte a cambiare radicalmente l'immagine stessa. Consentendo di selezionare queste due basilari tipologie si ritiene di riuscire a coprire efficacemente i casi d'uso nel campo della immunoistochimica sequenziale, dove i campioni che hanno subito un intenso processo di colorazione (e conseguente lavaggio) possono essere caratterizzati da importanti trasformazioni tra un passaggio di colorazione e il seguente; i campioni invece che hanno subito minime alterazioni possono essere semplicemente riassestati con il metodo *Traslazionale*.

- **Keep all pixel data**: L'operazione di co-registrazione può talvolta provocare una perdita di informazioni dei campioni (generalmente viene provocato un "taglio" di alcune porzioni esterne delle suddette immagini). La causa di questo effetto indesiderato è da imputare alla stessa operazione di deformazione: prendendo il caso più semplice (modello traslazionale), un immagine spostata sugli assi X e Y può uscire dai confini della sorgente dalla quale si basa. In caso di deformazione Affine l'effetto è ulteriormente aggravato dall'utilizzo di *shearing* e rotazione sull'immagine: in questo caso, la possibilità di perdere alcune sezioni dell'immagine si fa concreta. Lanciando l'elaborazione con questa opzione abilitata, DS4H Image alignment ricalcola la dimensione dello stack per ogni asse con la seguente formula:

$$dimensioneAsse = dimensioneMassimaAsse + massimoScostamentoAsse$$

In altre parole, per ogni asse lo stack si adegua alla dimensione più grande riscontrata e ne aggiunge la massima differenza tra i punti dei landmark dello stesso ordine. Applicando una traslazione per ogni immagine pari al suo scostamento l'immagine risulterà così perfettamente allineata e corredata da tutti i suoi dati originali; uno degli effetti indesiderati è ovviamente un incremento delle dimensioni totali dello stack risultante che, oltre ad implicare un maggior costo computazionale dell'elaborazione, può portare l'immagine a raggiungere la dimensione massima gestita da ImageJ (questo concetto verrà approfondito nel corso della sottosezione 4.5.2).

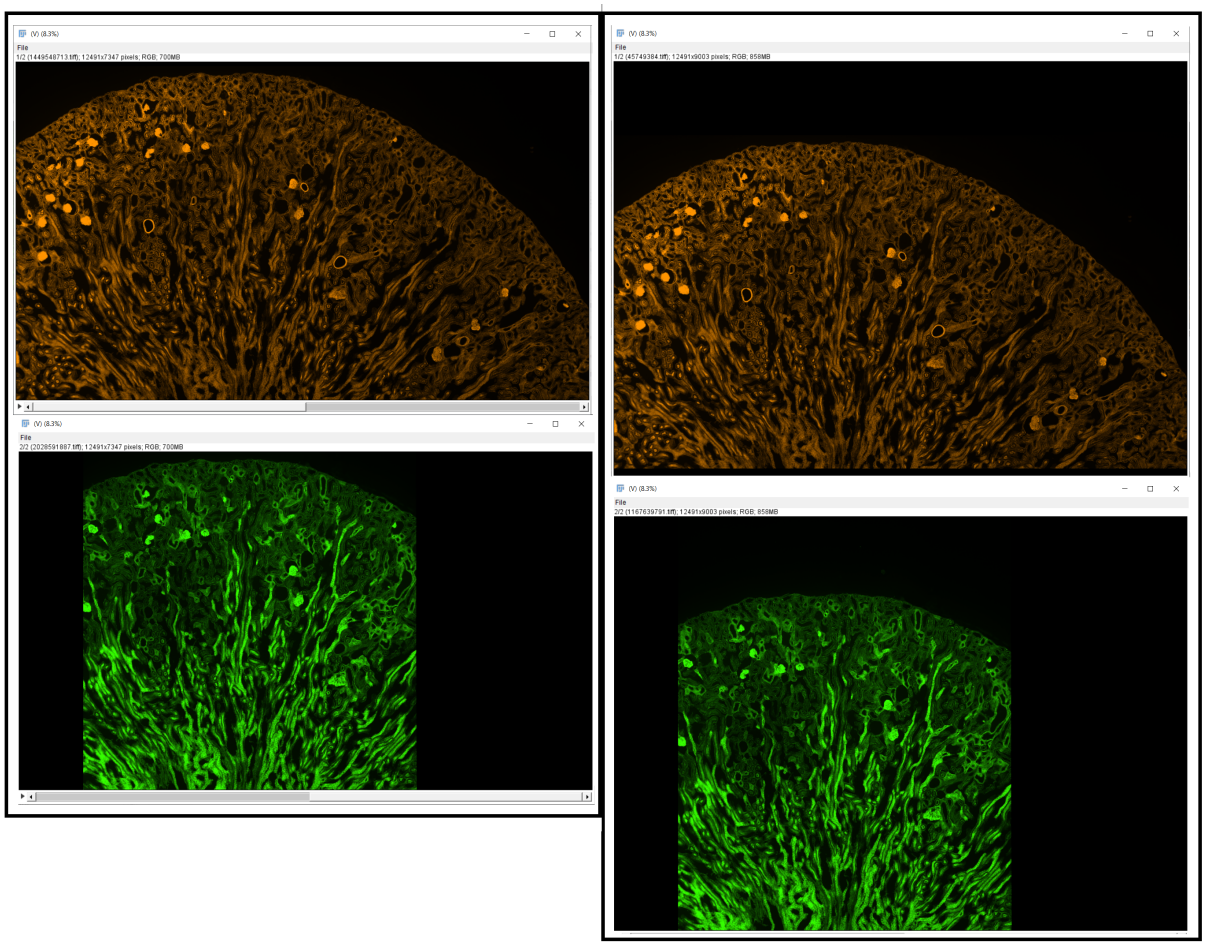

Figura 4.8: Comparazione di elaborazione con ”keep all pixel data” disabilitata (a sinistra) e abilitata (a destra). Notare come le immagini di destra abbiano mantenuto tutte le informazioni.

4.5 Criticità

Nonostante i componenti centrali di *DS4H Image Alignment* siano basati sull'utilizzo di meccanismi ampiamente conosciuti dalla comunità e con una certa garanzia di stabilità (come *ImageJ*, *bio-formats* e *mpicbg*), non è stato possibile evitare completamente l'insorgere di alcune problematiche di funzionamento dell'applicazione. Le sottosezioni seguenti approfondiscono le principali criticità emerse durante lo sviluppo, presentando i vari accorgimenti impiegati alla loro eliminazione e riduzione.

4.5.1 Utilizzo di memoria

Come già precedentemente anticipato, la microscopia fa largo uso di immagini di grandi dimensioni: maggiore sarà infatti la loro risoluzione (e qualità), minore sarà la difficoltà di analisi del campione. Ovviamente, questa caratteristica comporta un costo ingente in termini di dimensioni delle immagini, in quanto un file `.tiff` decompresso può facilmente contenere 5GB di dati; l'utilizzo di molteplici immagini nello stack di *DS4H Image Alignment* può portare a un rapido esaurimento della memoria del dispositivo. Allo stato attuale, il plugin riduce il più possibile questa eventualità applicando i seguenti accorgimenti:

- **Utilizzo di immagini scalate in MainDialog**: come già anticipato nella sezione 4.3.2, la struttura piramidale compresa nella quasi totalità dei formati commerciali consente di sfruttare agevolmente risoluzioni minori della stessa immagine (nell'ordine di 1/2, 1/4 e così via). Durante la fase di inserimento dei punti di controllo (cioè durante la visualizzazione della `MainDialog`), infatti, vengono mostrate soltanto immagini scalate prelevate dalla suddetta piramide (ovviamente se presenti nel file); in questo modo, questa fase potrà vantare un minimo utilizzo della memoria e caricamenti ridotti, che gioveranno all'interattività della UX.

- **Utilizzo di cache su disco**: Il precedente Listato 4.3.2 ha indicato le modalità di estrazione e inizializzazione delle immagini inserite all'interno del plugin, nella forma di `BufferedImageReader`. Questo fondamentale oggetto si occupa, per ogni file composto da stack di immagini, di occupare in memoria soltanto un singolo campione per volta; in questo modo è così possibile renderli uguali, in termine di impatto sulla RAM, alle controparti mono-immagine. Un meccanismo simile è stato utilizzato durante la fase di co-registrazione con elementi *wholeslide*: ogni campione elaborato viene scritto su disco (precisamente nella directory dei file temporanei dell'utente) e prelevato *on demand* durante la sua visualizzazione come `ImagePlus` nella `AlignDialog`.

Nonostante le precedenti misure, l'applicazione richiede il completo immagazzinamento in memoria di almeno due immagini *wholeslide* durante l'applicazione di algoritmi di deformazione, richiedendo così un quantitativo di memoria che può non essere sufficiente su macchine non

specializzate. Per evitare che l'utente avvii elaborazioni comunque destinate al fallimento a causa di memoria insufficiente, ogni file aggiunto viene dapprima analizzato per stimarne l'impatto sulla memoria RAM (tramite il metodo `estimateMemoryUsage()` di `ImageFile`). Qualora si rilevi che l'insieme delle immagini non sia sostenibile per l'elaborazione, l'interfaccia utente mostra a schermo una schermata di errore (con allegate delle istruzioni per aumentare la memoria allocata nella JVM).

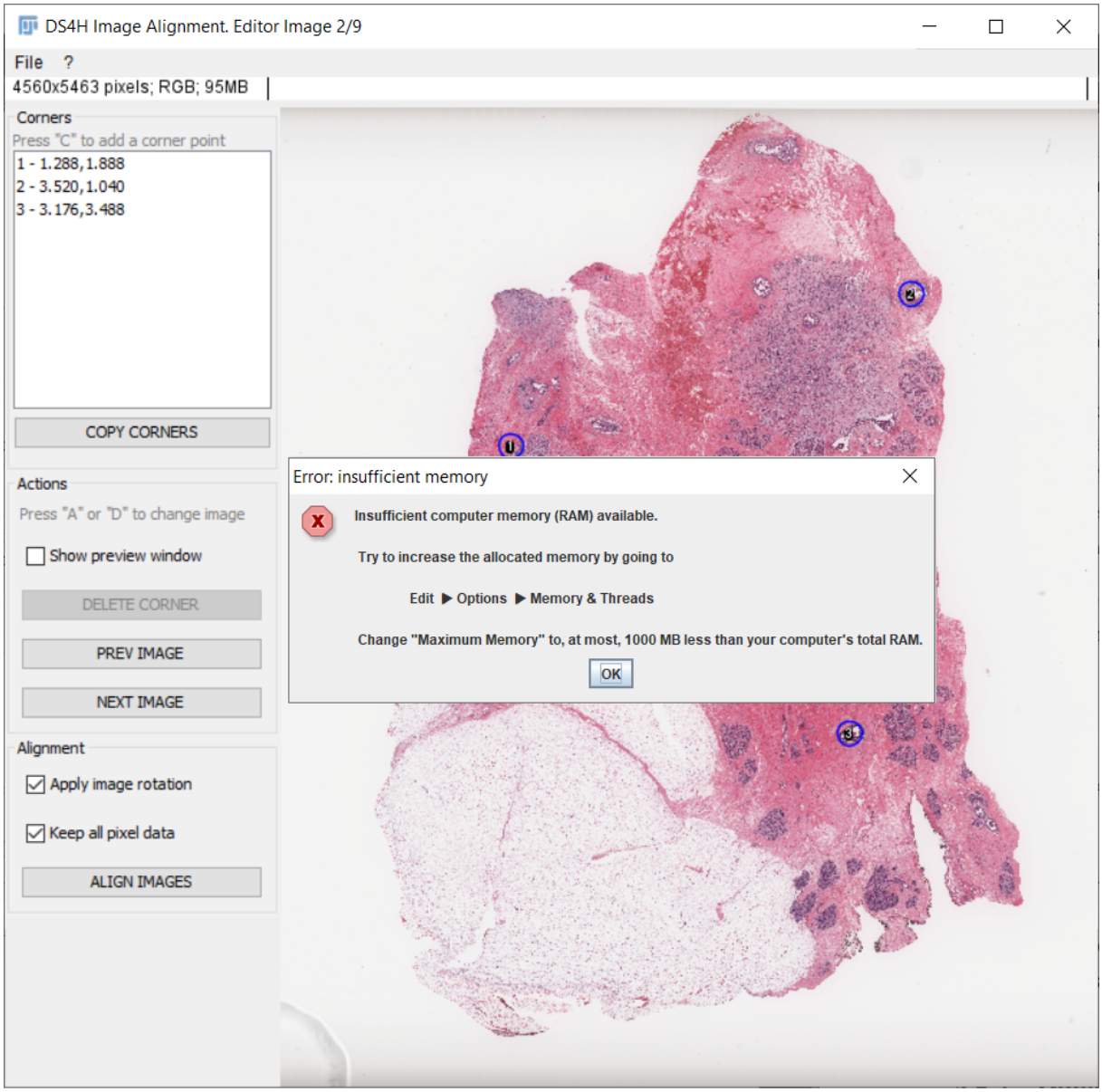

Figura 4.9: Messaggio di errore in caso di memoria prevista insufficiente.

4.5.2 Risoluzione massima supportata

Come indicato nella sezione 3.1, lo sviluppo di moduli aggiuntivi in *ImageJ* può avvenire tramite la creazione di *Plugin* compatibili *SciJava*, attraverso lo stesso linguaggio con cui il quale il *core* del programma è stato implementato, cioè JAVA. Ai tempi della creazione del

programma (circa 1997), questo linguaggio era uno dei migliori candidati per lo sviluppo di un applicativo open-source: si trattava di un linguaggio moderno, con capacità di portabilità promettenti e performance accettabili.

L'aumento graduale della qualità delle bioimmagini, tuttavia, ha comportato il raggiungimento del limite della risoluzione massima processabile dal programma, cioè 2^{31} pixel (indicativamente pari a una risoluzione di $46000x46000$); la ragione per cui questo valore massimo è ancora oggi presente nel programma è dovuto al fatto che è provocato da un limite intrinseco di JAVA, relativo alla dimensione massima [28] (in termine di indici) allocabile per un array. Tutte le immagini mostrate a schermo in ImageJ, infatti, sono istanze di oggetti `ImagePlus` i cui pixel vengono mantenuti in memoria in un array monodimensionale; ne consegue è impossibile, allo stato attuale, processare immagini ultra-larghe sul programma tramite le API 1.x.

Purtroppo, al momento non esistono soluzioni univoche per la risoluzione di questo problema: gli sviluppatori stessi del programma consigliano di aprire soltanto porzioni di immagini (*regions*), in modo di rimanere al di sotto del limite di due miliardi di pixel per volta (tramite la primitiva di `ImageProcessor.openImage(int no, int x, int y, int width, int height)`). Ovviamente, questo utilizzo complica inevitabilmente le operazioni che devono essere eseguite su immagini intere come le deformazioni di *DS4H Image Alignment*, e potrebbero essere completate solo eseguendo dello *stitching* tra le varie sezioni. Un altra soluzione sembra risiedere nelle nuove API previste per ImageJ2, che promettono di risolvere questo problema utilizzando una diversa organizzazione dei pixel all'interno nuovi `ImagePlus` previsti. Tuttavia, la loro al momento scarsa retrocompatibilità con Fiji ne sconsiglia ancora l'adozione.

Conclusioni

I capitoli precedenti hanno illustrato le numerose difficoltà che caratterizzano le analisi istologiche, in particolar modo di quelle immunoistologiche: l'alternanza di molteplici fasi di colorazione e lavaggio per uno stesso campione comporta, infatti, la deformazione e disallineamento dei vetrini rispetto allo strumento di acquisizione impiegato. Una possibile soluzione a questo problema è quella della co-registrazione delle immagini acquisite così da permettere il loro riallineamento e facilitare l'analisi da parte dell'operatore.

Ad oggi, però, non esiste un software gold standard dedicato alla risoluzione di questo problema. In questo contesto, il gruppo di ricerca DS4H ha sviluppato un Plugin ImageJ pensato per lo svolgimento di questo compito, *DS4H Image Alignment*. Il software, sviluppato dopo un'attenta fase di analisi derivante da esigenze reali espresse dai ricercatori dell'Istituto Scientifico Romagnolo per lo Studio e la cura dei Tumori (IRST) di Meldola (FC), permette la co-registrazione di immagini microscopiche digitali tramite posizionamento di *Landmark* manuali. Le figure così prodotte sono inserite all'interno di un Tiff Multi-page per facilitarne la consultazione.

In questa sede, si ritiene importante indicare alcuni miglioramenti che potrebbero essere inseriti all'interno del programma per agevolarne l'utilizzo e aumentarne le potenzialità. Sebbene risulti compatibile con tutte le versioni disponibili di ImageJ Fiji, l'applicazione potrebbe essere adattata alle nuove Api di ImageJ 2.x non appena verranno rilasciate in versione definitiva. Ciò consentirebbe di risolvere le problematiche relative alla dimensione massima delle immagini elaborabili, come indicato nella sottosezione 4.5.2.

Inoltre, lo sviluppo di un meccanismo automatico di inserimento di punti di controllo potrebbe snellire notevolmente la procedura manuale richiesta all'operatore, ad esempio tramite algoritmi di segmentazione *edge-based* o *tresholding-based*. L'impiego di marcatori di diverse tinte in microscopia, tuttavia, rendono difficile la loro implementazione, dato che tali tecniche si basano principalmente sull'intensità di uno stesso colore. Dovranno quindi essere ricercate segmentazioni meno legate alla cromaticita delle immagini [44].

In ultima analisi, potrebbe essere rilevante approfondire delle tecniche di registrazione che richiedano un quantitativo minore di memoria per lo svolgimento dell'operazione. Sebbene un

intensivo lavoro di ottimizzazione sia già stato svolto con *DS4H Image Alignment*, una modalità di registrazione che richieda significativamente meno memoria potrebbe rendere ancora più accessibile l'utilizzo del programma.

Infine, esempi di utilizzo di "DS4H Image Alignment" sono stati riportati nel seguente articolo scientifico: J. Bulgarelli, M. Tazzari, A.M. Granato, L Ridolfi, S. Maiocchi, F. De Rosa, M. Petrini, E. Pancisi, G. Gentili, B. Vergani, F. PICCININI, A. CARBONARO, B.E. Leone, G. Foschi, V. Ancarani, M. Framarini, M. Guidoboni, Dendritic cell vaccination in metastatic melanoma turns "non-T cell inflamed" into "T-cell inflamed" tumors, Frontiers in Immunology, 2019. IF2019: 4.716. DOI: 10.3389/fimmu.2019.02353 "DS4H Image Alignment" e oggi diffuso come modulo aggiuntivo di ImageJ. (Codice sorgente, versioni eseguibili, video tutorial, immagini di esempio sono disponibili al link [37] riportato in bibliografia).

Elenco delle figure

Bibliografia

[1] Lauren A. Ernst, Ravinder K. Gupta, Ratnakar B. Mujumdar, and Alan S. Waggoner. Cyanine dye labeling reagents for sulfhydryl groups. *Cytometry*, 10(1):3–10, 1989.

[2] Barbara Zitova and Jan Flusser. Image registration methods: a survey. *Image and vision computing*, 21(11):977–1000, 2003.

[3] Claire McQuin, Allen Goodman, Vasiliy Chernyshev, Lee Kamentsky, Beth A. Cimini, Kyle W. Karhohs, Minh Doan, Liya Ding, Susanne M. Rafelski, Derek Thirstrup, Winfried Wiegraebe, Shantanu Singh, Tim Becker, Juan C. Caicedo, and Anne E. Carpenter. Cellprofiler 3.0: Next-generation image processing for biology. *PLOS Biology*, 16(7):1–17, 07 2018.

[4] Thouis R. Jones, In Han Kang, Douglas B. Wheeler, Robert A. Lindquist, Adam Papallo, David M. Sabatini, Polina Golland, and Anne E. Carpenter. Cellprofiler analyst: data exploration and analysis software for complex image-based screens. *BMC Bioinformatics*, 9(1):482, 2008.

[5] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436 EP –, May 2015.

[6] Peter Bankhead, Maurice B. Loughrey, José A. Fernández, Yvonne Dombrowski, Darragh G. McArt, Philip D. Dunne, Stephen McQuaid, Ronan T. Gray, Liam J. Murray, Helen G. Coleman, Jacqueline A. James, Manuel Salto-Tellez, and Peter W. Hamilton. Qupath: Open source software for digital pathology image analysis. *Scientific Reports*, 7(1):16878, 2017.

[7] Raphaël Marée, Benjamin Stévens, Loïc Rollus, Natacha Rocks, Xavier Moles Lopez, Isabelle Salmon, Didier Cataldo, and Louis Wehenkel. A rich internet application for remote visualization and collaborative annotation of digital slides in histology and cytology. *Diagnostic Pathology*, 8(Suppl 1):S26–S26, Sep 2013. PMC3849538[pmcid].

[8] Finalizing a definition of ”citizen science” and ”citizen scientists”. `http://www.openscientist.org/2011/09/finalizing-definition-of-citizen.html`.

[9] Johannes Schindelin, Curtis T. Rueden, Mark C. Hiner, and Kevin W. Eliceiri. The imagej ecosystem: An open platform for biomedical image analysis. *Molecular Reproduction and Development*, 82(7-8):518–529, 2015.

[10] Kevin W. Eliceiri, Michael R. Berthold, Ilya G. Goldberg, Luis Ibáñez, B. S. Manjunath, Maryann E. Martone, Robert F. Murphy, Hanchuan Peng, Anne L. Plant, Badrinath Roysam, Nico Stuurman, Jason R. Swedlow, Pavel Tomancak, and Anne E. Carpenter. Biological imaging software tools. *Nature methods*, 9(7):697–710, Jun 2012. 22743775[pmid].

[11] Karen A Collins, John F Kielkopf, Keivan G Stassun, and Frederic V Hessman. Astroimagej: image processing and photometric extraction for ultra-precise astronomical light curves. *The Astronomical Journal*, 153(2):77, 2017.

[12] Imagejfx, an enhanced interface for imagej.
`http://www.imagejfx.net/`.

[13] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, Jean-Yves Tinevez, Daniel James White, Volker Hartenstein, Kevin Eliceiri, Pavel Tomancak, and Albert Cardona. Fiji: an open-source platform for biological-image analysis. *Nature Methods*, 9:676 EP –, Jun 2012. Perspective.

[14] Imagej macro language programmer's reference guide.
`https://imagej.nih.gov/ij/docs/macro_reference_guide.pdf/`.

[15] Frederic P Miller, Agnes F Vandome, and John McBrewster. Apache maven. 2010.

[16] Nicholas Chen. Convention over configuration.
http://softwareengineering.vazexqi.com/files/convention_over_configuration.pdf/, 2006.

[17] Scott Chacon and Ben Straub. *Pro git*. Apress, 2014.

[18] Github.
`https://Github.com/`.

[19] Intellij idea.
`https://www.jetbrains.com/idea/`.

[20] Ds4h image alignment github repository.
`https://github.com/Luxor001/DS4H-Image-Alignment/`.

[21] Ilya Goldberg, Chris Allan, Jean-Marie Burel, Doug Creager, Andrea Falconi, Harry Hochheiser, Josiah Johnston, Jeff Mellen, Peter Sorger, and Jason R Swedlow. The open microscopy environment (ome) data model and xml file: Open tools for informatics and quantitative analysis in biological imaging. *Genome biology*, 6:R47, 02 2005.

[22] Michele Larobina and Loredana Murino. Medical image file formats. *Journal of Digital Imaging*, 27(2):200–206, Apr 2014.

[23] Supported file formats by bio formats. `https://docs.openmicroscopy.org/bio-formats/5.8.2/supported-formats.html/`.

[24] Erik Meijering, Anne E. Carpenter, Hanchuan Peng, Fred A. Hamprecht, and Jean-Christophe Olivo-Marin. Imagining the future of bioimage analysis. *Nature Biotechnology*, 34:1250 EP –, Dec 2016.

[25] Melissa Linkert, Curtis T. Rueden, Chris Allan, Jean-Marie Burel, Will Moore, Andrew Patterson, Brian Loranger, Josh Moore, Carlos Neves, Donald MacDonald, Aleksandra Tarkowska, Caitlin Sticco, Emma Hill, Mike Rossner, Kevin W. Eliceiri, and Jason R. Swedlow. Metadata matters: access to image data in the real world. *The Journal of Cell Biology*, 189(5):777–782, 2010.

[26] Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. *ACM Trans. Graph.*, 25:533–540, 07 2006.

[27] Libreria mpicbg per trasformazione, registrazione e interpretazione di immagini. `https://github.com/axtimwalde/mpicbg/`.

[28] Java array access specification: *An attempt to access an array component with a long index value results in a compile-time error*. `https://docs.oracle.com/javase/specs/jls/se12/html/jls-10.html#jls-10.4/`.

[29] Roberta Di Pietro. *Elementi di istologia*. Edises, 2012.

[30] Microscopia a fluorescenza. `https://elearning.uniroma1.it/pluginfile.php/601011/mod_resource/content/1/23%20Tecniche%20microcopiche%20avanzate.pdf/`.

[31] Arthur Ardeshir Goshtasby. *2-D and 3-D image registration: for medical, remote sensing, and industrial applications*. John Wiley & Sons, 2005.

[32] Albert Cardona. Trakem2: an imagej-based program for morphological data mining and 3d modeling. 01 2006.

[33] Fred L. Bookstein. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Transactions on pattern analysis and machine intelligence*, 11(6):567–585, 1989.

[34] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. *SIGGRAPH Comput. Graph.*, 26(2):35–42, July 1992.

[35] Ron MacCracken and Kenneth I Joy. Free-form deformations with lattices of arbitrary topology. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 181–188. ACM, 1996.

[36] Plugin *Align Image By Line ROI* di imagej. `https://imagej.net/Align_Image_by_line_ROI/`.

[37] Data science for health (ds4h) image alignment. `https://imagej.net/DS4H_Image_Alignment/`.

[38] Neeraj Sharma and Lalit M Aggarwal. Automated medical image segmentation techniques. *Journal of medical physics/Association of Medical Physicists of India*, 35(1):3, 2010.

[39] Emma C Robinson, Saad Jbabdi, Matthew F Glasser, Jesper Andersson, Gregory C Burgess, Michael P Harms, Stephen M Smith, David C Van Essen, and Mark Jenkinson. Msm: a new flexible framework for multimodal surface matching. *Neuroimage*, 100:414–426, 2014.

[40] Lisa Gottesfeld Brown. A survey of image registration techniques. *ACM computing surveys (CSUR)*, 24(4):325–376, 1992.

[41] Stefan Klein, Marius Staring, Keelin Murphy, Max A Viergever, and Josien PW Pluim. Elastix: a toolbox for intensity-based medical image registration. *IEEE transactions on medical imaging*, 29(1):196–205, 2009.

[42] Jacques Feldmar and Nicholas Ayache. Rigid, affine and locally affine registration of free-form surfaces. *International journal of computer vision*, 18(2):99–119, 1996.

[43] David Levin. The approximation power of moving least-squares. *Mathematics of computation*, 67(224):1517–1531, 1998.

[44] Max Mignotte. Segmentation by fusion of histogram-based k-means clusters in different color spaces. *IEEE Transactions on image processing*, 17(5):780–787, 2008.

Ringraziamenti

Vorrei innanzitutto ringraziare la prof.ssa Antonella Carbonaro e il prof. Filippo Piccinini per avermi seguito come, rispettivamente, Relatrice e Correlatore per lo sviluppo di questa tesi. Il loro aiuto è stato costante, puntuale e collaborativo, facendomi sentire parte di un team allineato verso uno scopo condiviso.

Ringrazio inoltre il Dott. Massimo Giudoboni, la Dott.ssa Marcella Tazzari, la Dott.ssa Jenny Bulgarelli e tutti gli altri membri del team dell'Immunotherapy and Somatic Cell Therapy Unit dell'Ospedale IRST IRCCS Meldola (FC) per aver fornito il razionale alla base del progetto di Tesi ed immagini con cui testare l'algoritmo sviluppato.

Ringrazio tutti i miei amici che in questi anni mi hanno supportato condividendo pareri, materiale e consigli sugli esami che avrei dovuto affrontare e che hanno dimostrato un'enorme pazienza con me: Cozzolino, Laura, Gioele, Andrea, Tania, Fabio, Nicos sono solo alcune delle persone a cui devo offrire una cena. O forse due.

Una dedica speciale va ai miei colleghi di Dynaset che mi hanno accompagnato in questi anni di studio. È anche grazie ai vostri incoraggiamenti (e comprensione...) se sono riuscito in questa impresa, rimanendo comunque uno stimolo fondamentale per la mia crescita professionale.

Un grazie va alla mia famiglia: per aspettarmi la sera tardi al ritorno dalla biblioteca, per regalarmi i nippon dopo un esame andato bene e per la tanta, tanta pazienza.
Non so se con questa laurea vi avrò reso fiero di me, ma io sono fiero di voi come genitori.
Ale, ti prometto che ci vedremo più spesso. Mi dispiace aver dovuto sacrificare qualche serata con te e Sara, ma è finalmente finita. Voglio dimostrarti che se avessi potuto avrei passato molte più serate insieme con una delle Forst che ti piacciono tanto.

Un grazie anche alla mia famiglia allargata: Urbano, Graziella e Marco. La vostra comprensione e fiducia mi ha dato forza, e anche qui siete stato un pezzo fondamentale del mosaico. Mi sento fortunato ad avervi nella mia famiglia, e mi sento di darti ragione Graziella: uniti possiamo fare tutto.

E infine, un grazie a te Sara. Hai condiviso con me gioie e dolori di questo percorso, e senza il tuo aiuto non ce l'avrei mai e poi mai fatta. Non potrò mai restituirti il tempo che ho rubato dalla nostra coppia, e ti ringrazio per avermelo dedicato.
Come sempre sei stata un'insostituibile compagna di viaggio, e ora il nostro viaggio ha un nuovo corso, insieme.