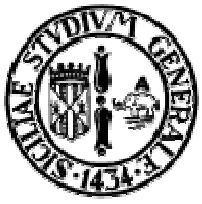


Dipartimento di Matematica e Informatica

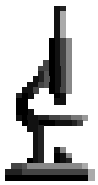
Corso di Laurea Specialistica in Informatica

MULTIMEDIA

Prof. S.Battiato

Relazione del progetto :

Studio del software JAVA per l' Image Processing ImageJ



a cura di:

Bosco Camillo, camillo.bosco@studenti.unict.it
Cappello Annalisa, annalisa.cap@katamail.com

Indice

1 Introduzione

1.0 Presentazione

La presente relazione costituisce un manuale dettagliato di riferimento sul progetto ImageJ per l'utente e per lo sviluppatore. L'ambiente di ImageJ, nelle sue varie sfaccettature, è stato studiato dagli studenti Bosco Camillo e Cappello Annalisa al fine di realizzare dei plugins per l'enhancement nel dominio spaziale e delle frequenze nell'ambito del corso di Multimedia tenuto dal Prof. Sebastiano Battiato presso il Dipartimento di Matematica ed Informatica dell'Università di Catania nell'A.A. 2004-2005.

1.1 Generalità su ImageJ

ImageJ è un software *open source*, programmato in JAVA, che nasce con l'obiettivo di emulare le funzionalità dei più comuni software commerciali per l'image processing.

Originariamente progettato dalla NIH Image per il Macintosh, eredita dal linguaggio JAVA i vantaggi derivanti dalla portabilità su differenti piattaforme: può infatti essere eseguito, sia sotto forma di applet (online) sia sotto forma di applicazione JAVA, su un qualsiasi computer su cui sia installata la Java Virtual Machine versioni 1.1 o successive.

ImageJ consente di visualizzare, modificare, analizzare, processare, salvare e stampare immagini a 8-bit, 16-bit e 32-bit. I formati supportati sono TIFF, GIF, JPEG, BMP, DICOM, FITS e "raw".

E' basato sul multithreading ovvero è possibile effettuare delle operazioni in parallelo con altre poiché a ciascun processo è associato un thread. Inoltre supporta le cosiddette "stacks", cioè serie di immagini che condividono la stessa finestra.

ImageJ offre anche la possibilità di calcolare l'area e le statistiche sui valori dei pixel relativamente a delle regioni (ROI = Region Of Interest) selezionate dall'utente. E' inoltre possibile misurare

distanze e angoli, plottare grafici ed istogrammi, effettuare le più semplici operazioni di Image Enhancement.

ImageJ supporta inoltre le più comuni trasformazioni geometriche come *scaling* e rotazione e operazioni di zooming.

I principali obiettivi legati al design di ImageJ sono:

- il progetto di una architettura aperta che sia estendibile mediante l'aggiunta di nuove funzionalità tramite plugins scritti in JAVA; i plugins possono essere creati, modificati e visualizzati all'interno dell'editor integrato di ImageJ e del compilatore JAVA;
- lo sviluppo su piattaforma Mac OS X (esistono anche opportune versioni per Windows e Linux); il codice sorgente è free e liberamente scaricabile all'indirizzo http://rsb.info.nih.gov/ij/index.html. L'autore Wayne Rasband lavora presso il National Institute of Mental Health, Maryland, USA.

1.2 Installazione di ImageJ

Relativamente alla installazione di ImageJ sono richiesti:

- i file .class e i file di configurazione di ImageJ;
- il Java Runtime Environment (JRE);
- al fine di creare plugins per ImageJ è necessario un compilatore JAVA incluso nella JAVA 2 SDK Standard Edition liberamente scaricabile dal sito http://java.sun.com.

La distribuzione più recente di ImageJ può essere scaricata dal sito http://rsb.info.nih.gov/ij/download.html. E' possibile distinguere due modalità di installazione:

- "***modalità utente***": se si vuole utilizzare ImageJ in maniera standard cioè come un software per l'Image Processing, allora basta scaricare, in ambiente Windows, l'apposito eseguibile denominato `ijversione-setup.exe` dove la stringa versione è appositamente sostituita da un numero indicante la versione di ImageJ. Basterà effettuare un doppio click

su tale eseguibile per avviare il wizard di installazione; similmente è possibile ciccare due volte sul file `ij.jar` per avviare il software.

- "***modalità sviluppatore***": se si vuole estendere ImageJ mediante l'implementazione di plugins, bisognerà anzitutto provvedere alla installazione di JAVA 2 SDK Standard Edition che comprende anche il JRE; quindi, al fine di poter sviluppare e compilare i plugins, si consiglia di utilizzare l'editor integrato di ImageJ (`Plugins/New`); in alternativa è possibile sviluppare i plugins utilizzando un editor esterno: in tale ottica, al fine di poter importare i packages di ImageJ, è necessario copiare il file `ij.jar` e le cartelle `plugins` e `macros` nella cartella `jre\lib\ext` situata all'interno della cartella principale in cui è stato installato l'SDK (esempio `C:\j2sdkversione\jre\lib\ext`). Per avviare il software basterà ciccare due volte sul file `ij.jar`.
- Ulteriori dettagli circa le modalità di installazione si trovano sul sito internet.

1.3 Aggiornamento di ImageJ

E' possibile aggiornare la versione correntemente utilizzata di ImageJ semplicemente sostituendo il file JAR `ij.jar`. L'ultima versione disponibile di ImageJ è reperibile all'indirizzo http://rsb.info.nih.gov/upgrade/index.html. Basterà sostituire il vecchio file JAR con il nuovo file appena scaricato per espletare l'aggiornamento di ImageJ.

2 Guida all'utilizzo di ImageJ

2.1 Concetti basilari

Di seguito verranno presentati alcuni concetti basilari relativi al semplice utilizzo di ImageJ.

Avviando il software compare una finestra (figura 1) contenente una barra dei menu, una barra degli strumenti, una barra di stato e una progress bar.

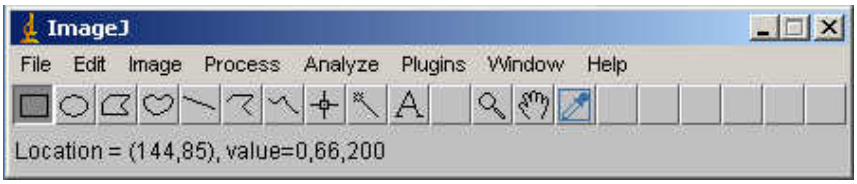


Figura 1 La finestra principale di ImageJ

Le immagini, gli istogrammi, i grafici, etc . . . sono visualizzati in altre finestre (figura 2). Si noti che gli istogrammi e i grafici sono considerati come vere e proprie immagini e come tali possono essere copiate nella clipboard, modificate, stampate e salvate.

Particolari finestre sono quelle che visualizzano risultati relativi a misurazioni effettuate sulle immagini ("results windows" – figura 3).

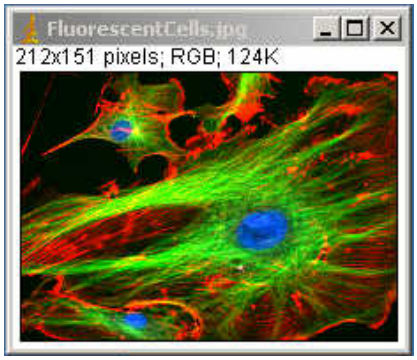
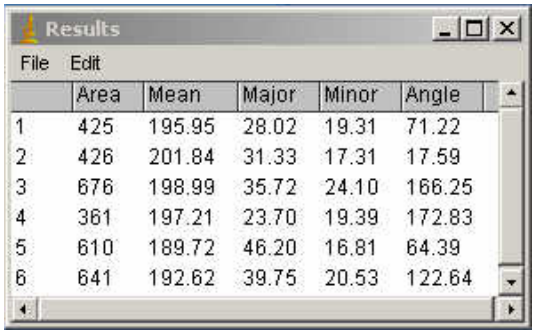


Figura 2 Una finestra in cui è mostrata una immagine



	Area	Mean	Major	Minor	Angle
1	425	195.95	28.02	19.31	71.22
2	426	201.84	31.33	17.31	17.59
3	676	198.99	35.72	24.10	166.25
4	361	197.21	23.70	19.39	172.83
5	610	189.72	46.20	16.81	64.39
6	641	192.62	39.75	20.53	122.64

Figura 3 Una "results window"

La barra degli strumenti (figura 4) contiene gli strumenti per effettuare selezioni, operazioni di zooming e scrolling su immagini, per cambiare i colori, etc . . . . Cliccando su un particolare strumento viene visualizzata la descrizione ad esso relativa nella barra di stato.

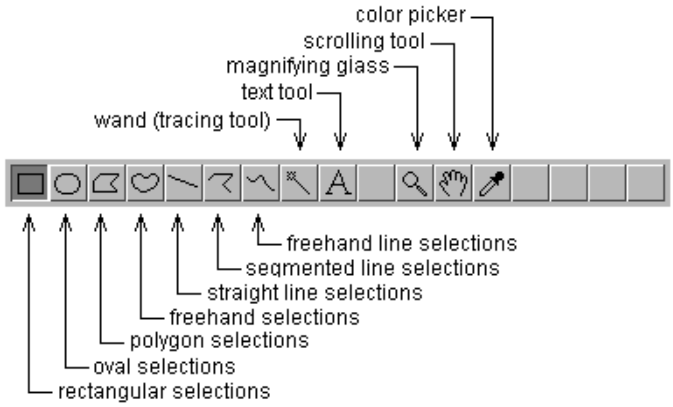


Figura 4 La barra degli strumenti

Di seguito viene riportata una breve descrizione degli strumenti disponibili:

Rectangle Selection tool : consente di effettuare selezioni rettangolari. Tenendo premuto il tasto shift è possibile selezionare quadrati; inoltre premendo i tasti freccia e contemporaneamente il tasto alt è possibile variare le dimensioni della selezione. La locazione e le dimensioni della selezione sono visualizzate nella barra di stato.

Oval Selection tool : consente di effettuare selezioni a forma di ellisse. Tenendo premuto il tasto alt è possibile specificare una selezione circolare; inoltre premendo i tasti freccia e contemporaneamente il tasto alt è possibile variare le dimensioni della selezione. La locazione e le dimensioni della selezione sono visualizzate nella barra di stato.

Polygon Selection tool : consente di effettuare selezioni di forma irregolare. Per creare una selezione bisogna cliccare ripetutamente con il mouse per disegnare i lati del poligono. Alla fine si clicca sul piccolo box in corrispondenza del punto iniziale e ImageJ automaticamente disegna l'ultimo segmento mancante.

Freehand Selection tool  : consente di effettuare selezioni di forma irregolare trascinando (dragging) il mouse.

Wand Selection tool  : consente di effettuare selezioni di oggetti di colore uniforme o sogliati all'interno di una immagine.

Straight Line Selection tool  : consente di creare una linea retta. Tenendo premuto il tasto alt è possibile specificare la direzione (orizzontale o verticale) della linea.

Segmented Line Selection tool  : consente di creare una spezzata la cui forma è scelta dall'utente mediante click successivi del mouse: ogni click definirà un nuovo segmento della linea. Un doppio click indica la fine della linea.

Freehand Line Selection tool  : consente di creare una linea la cui forma è scelta dall'utente.

Mark & Count tool  : consente di contare oggetti; cliccando su un punto dell'immagine vengono registrate le coordinate del punto e l'intensità e viene disegnato un simbolo di mark. Poiché i simboli di mark modificano l'immagine potrebbe essere opportuno lavorare su una copia della stessa. Un doppio click sulla icona crosshair consente di cambiare la dimensione di un simbolo di mark; settando la larghezza del mark a zero verrà disabilitato il marking. L'effetto di questo tool è immediato e non necessita dell'uso dei comandi *Draw* o *Fill*.

Text tool  : consente di aggiungere del testo alle immagini; crea una selezione rettangolare contenente una o più linee di testo. Utilizzando il comando *Edit/Draw* il testo viene permanentemente "disegnato" sulla immagine. Utilizzando *Edit/Options/Fonts* è possibile specificare dimensione, stile e famiglia del testo.

Magnifying Glass tool  : consente di effettuare operazioni di zooming sull'immagine. Un doppio click sull'icona del tool consente di riportare l'immagine alle dimensioni originali 1:1. Il livello di zooming corrente è mostrato nella barra del titolo dell'immagine. Vi sono 11 possibili livelli di zooming: 1:32, 1:16, 1:8, 1:4, 1:2, 1:1, 2:1, 4:1, 8:1, 16:1 e 32:1.

Scrolling tool  : consente di visualizzare totalmente una immagine le cui dimensioni sono superiori a quelle della finestra che la contiene (scrolling).

Color Picker tool  : consente di settare il colore di foreground e quello di background. Una palette di colori da scegliere viene visualizzata cliccando sull'item *Colors* del menu *Image*.

La barra di stato (figura 5), quando il cursore è sopra una immagine, visualizza le coordinate e i valori del pixel corrente. Dopo aver effettuato un filtraggio su una immagine la barra di stato visualizza il tempo impiegato e la velocità di elaborazione espressa in pixels/secondo.

Figura 5 La barra di stato

La progress bar (figura 6), situata a destra della barra di stato, mostra lo stato di avanzamento (progress) delle operazioni. Essa non compare nel caso di operazioni che richiedono meno di un secondo per essere espletate.

Figura 6 La progress bar

ImageJ consente la visualizzazione simultanea di più immagini di cui ciascuna è visualizzata in una apposita finestra. La finestra attiva ha la sua barra del titolo evidenziata. Tutte le operazioni saranno eseguite sulla immagine attiva. ImageJ supporta immagini a toni di grigio a 8-bit, 16 bit e 32 bit e immagini a colori a 8-bit e 32-bit.

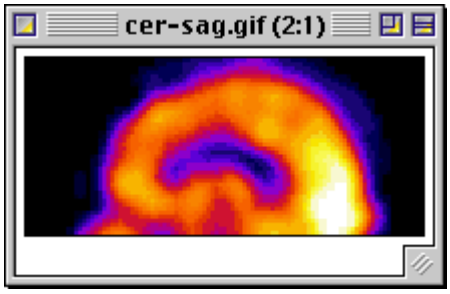


Figura 7 Una immagine

Inoltre ImageJ supporta la visualizzazione di più immagini, correlate dal punto di vista spaziale o temporale, in una singola finestra. Questi insiemi di immagini sono denominati *stacks* (figura 8). Le immagini che costituiscono uno stack sono chiamate *slices*. Tutte le slices in uno stack devono essere immagini con le stesse dimensioni e con la stessa profondità. Una barra di scorrimento consente di visualizzare le diverse slices.

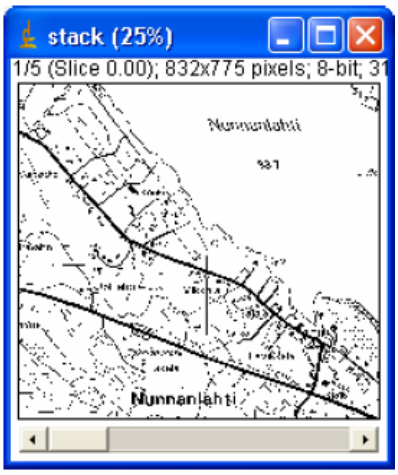


Figura 8 Un esempio di stack

Le selezioni (figura 9) sono aree o linee definite dall'utente. Solo una selezione per volta può essere attiva. Le selezioni sono create utilizzando i corrispondenti tools della barra degli strumenti. Le selezioni di aree possono essere misurate (*Analyze/Measure*), filtrate, riempite (*Edit/Fill*) o disegnate (*Edit/Draw*). Si utilizza *Edit/Draw* per disegnare una linea del colore correntemente usato. La linea può essere misurata utilizzando *Analyze/Measure*.

Figura 9 Le selezioni

ImageJ supporta immagini dei seguenti formati: TIFF, GIF, JPEG, DICOM, BMP e FITS. Esistono inoltre dei plugins che consentono di leggere files “raw”, immagini in formato ASCII, e per caricare immagini in rete specificando un URL.

I plugins possono essere considerati come moduli JAVA che consentono di espandere le funzionalità di ImageJ. Mediante i plugins è possibile implementare qualsiasi algoritmo di enhancement. Tutte le funzionalità di base (es. lettura, visualizzazione, zooming di immagini) saranno gestite da ImageJ. I plugins possono essere creati o modificati utilizzando l’editor integrato di ImageJ accessibile da *Plugins/Edit*. Alternativamente è possibile sviluppare plugins all’esterno dell’ambiente di ImageJ (ad esempio utilizzando un IDE). L’integrazione dei plugins all’interno di ImageJ può avvenire semplicemente inserendoli dentro la cartella di ImageJ denominata “plugins”: in tal modo essi saranno installati nel menu Plugins. Alternativamente i plugins possono essere installati in altri menu utilizzando il comando *Plugins/Hot Keys/Install Plugins*. Più di 150 plugins sono disponibili all’indirizzo http://rsb.info.nih.gov/ij/plugins/index.html.

2.2 Descrizione della barra dei menu

La barra dei menu contiene diversi menu che offrono molteplici funzionalità. Di seguito si riporta una descrizione generica delle funzioni offerte da ogni menu:

menu File: contiene degli items per:

la creazione di una nuova immagine o di un nuovo stack (item New);

la lettura e la visualizzazione di una immagine in una apposita finestra (item Open);

l’apertura e la visualizzazione di alcune immagini di esempio scaricabili da http://rsb.info.nih.gov/ij/images/ (item Open Samples);

l’importazione di immagini, di insiemi di immagini (stacks) collocate nella stessa cartella, di LUT, di file di testo aperti come immagini, di immagini “raw” ovvero in un formato che non è direttamente supportato da ImageJ (item Import);

la chiusura dell’immagine visualizzata nella finestra correntemente attiva (item Close);

il salvataggio della immagine attiva o dello stack in formato TIFF con possibilità di salvare solo l’area selezionata mediante una selezione rettangolare e mediante l’utilizzo del comando *Image/Duplicate* (item Save);

il salvataggio della immagine attiva in formato TIFF, GIF, JPEG o “raw” con possibilità di salvare selezioni, risultati statistici (measurements), lookup tables, ecc . . . (item Save As);

la sostituzione della immagine corrente con la sua ultima versione salvata (item Revert);

la possibilità di definire il layout di stampa impostando le relative opzioni (stampa centrata sul foglio, fattore di scala, rotazione di 90°, ecc. . .) di stampa (item Page Setup);

la stampa della immagine corrente con le impostazioni di stampa correnti (item Print);

l’uscita da ImageJ (item Quit).

menu Edit: contiene degli items per:

l’annullamento dell’ultima operazione di editing o filtering effettuata sulla immagine corrente (item Undo);

la copia dei contenuti della corrente selezione di immagine nella clipboard ed il relativo riempimento con il colore di background corrente (item Cut);

la copia dei contenuti della corrente selezione di immagine nella clipboard; se nessuna area della immagine corrente è stata selezionata viene copiata l’intera immagine correntemente attiva; la quantità di dati copiata è mostrata nella status bar; le operazioni

di *Cut*, *Copy* e *Paste* di immagini verso altre applicazioni non è correntemente supportato; selezionando l'opzione *Fill With Clipboard*, accessibile dall'item *File/New*, è possibile creare una nuova finestra con i contenuti della clipboard (item Copy);

l'inserimento dei contenuti della clipboard nella immagine attiva (item Paste);

la possibilità di effettuare, dopo una operazione di Cut, il Paste di una immagine risultante da una operazione (AND, OR, XOR, Add, ecc . . .) tra la immagine di partenza e quella memorizzata nella clipboard (item Paste Control . . .);

la cancellazione di una area selezionata di una immagine e la relativa sostituzione con una area omogenea di colore uguale a quello correntemente impostato come "background color" (item Clear);

la cancellazione di una area esterna ad una selezione e la relativa sostituzione con una area omogenea di colore uguale a quello correntemente impostato come "background color" (item Clear Outside);

il riempimento di una selezione con il colore correntemente impostato come "foreground color" (item Fill);

la possibilità di colorare i bordi di una selezione usando il colore di foreground corrente (item Draw);

la creazione della immagine inversa di quella corrente o di una selezione (simile ad un negativo fotografico) (item Invert);

la possibilità di creare, cancellare o modificare selezioni (item Selection):

1. **Select All**: crea una selezione rettangolare della stessa dimensione dell'immagine;
2. **Select None**: cancella la selezione nella immagine attiva;
3. **Restore Selection**: ripristina la precedente selezione alla sua posizione originale;
4. **Fit Spline**: effettua il fitting di una spline cubica ad un poligono o ad una selezione costituita da più linee;

5. **Fit Ellipse**: sostituisce una area selezionata con la migliore ellisse adattata; l'ellisse ha la stessa area, orientazione e centroide come la selezione originale;
6. **Convex Hull**: sostituisce un poligono di selezione a mano libera con la sua buccia convessa che può essere pensata come un nastro di gomma avvolta ermeticamente intorno ai punti che definiscono la selezione.

la possibilità di cambiare varie impostazioni di ImageJ in base alle preferenze dell'utente (item Options):

1. **Line Width . . .**: visualizza un dialog box che consente di modificare la larghezza delle linee generate dal comando Draw;
2. **JPEG Quality . . .**: questo dialogo consente di specificare il livello di compressione usato dal comando *File/Save As/Jpeg*; specificando un basso valore si richiederà un grado più alto di compressione;
3. **Fonts . . .**: apre una piccola finestra con tre combo box per la scelta della famiglia, della dimensione e dello stile del font utilizzato dal text tool;
4. **Profile Plot Options . . .**: questo dialogo controlla come vengono visualizzati i grafici generati con il comando *Analyze/Plot Profile*;
5. **Conversions . . .**: questo dialogo consente di settare le opzioni che controllano come le immagini vengono convertite da un tipo all'altro; l'opzione *Scale When Converting* consente di scalare i valori dei pixels nel range [0, 255] quando si converte da 16 bits o 32 bits a 8 bits oppure di scalare i valori dei pixels nel range [0, 65535] quando si converte da 32 bits a 16 bits. Un'altra interessante opzione è Unweighted RGB to Grayscale Conversion: se attivata, la formula *gray=(red+green+blue)/3* è usata per convertire immagini RGB a toni di grigio. Se questa opzione invece non è attivata viene utilizzata la formula *gray=0.299*red+0.587*green+0.114*blue*.

6. **Memory . . .**: questo dialogo consente di specificare la quantità massima di memoria disponibile per ImageJ. Mediante questo dialogo è possibile aumentare la quantità di memoria allocata di default per ImageJ (128 MB).
7. **Misc . . .**: questo dialogo contiene diverse opzioni: l'opzione *Divide By Zero Value* specifica il valore utilizzato quando *Process/Image Calculator* individua una divisione per zero mentre si divide una immagine a 32 bit per un'altra; il valore di default per questa opzione è infinito (altri possibili valori sono "max" e "NaN"). L'opzione *Use Pointer Cursor* consente di modificare l'aspetto del cursore impostando un cursore a freccia. L'opzione *Hide "Process Stack?" Dialog* consente di sopprimere il dialogo che chiede: "Processare tutte le xx slices ?" (solo la slice corrente sarà processata). L'opzione *Antialiased Text* consente di smussare il testo creato con il Text tool. L'opzione *Open/Save Using JFileChooser* consente di abilitare *File/Open* e *File/Save As/* all'utilizzo dell'oggetto JAVA JFileChooser al posto delle finestre di apertura e salvataggio dei files fornite dal sistema operativo. L'opzione *Debug Mode* consente di abilitare la visualizzazione dei messaggi di debug.

menu Image: contiene degli items per:

la determinazione del tipo della immagine attiva e la sua conversione in un altro tipo (item Type):

1. **8 bit**: converte l'immagine attiva, che deve essere a toni di grigio a 16 bit, 32 bit o a colori con 8 bit o RGB, in una a toni di grigio a 8 bit;
2. **16 bit**: converte l'immagine attiva, che deve essere a toni di grigio a 8 bit o 32 bit, in una a toni di grigio a 16 bit;
3. **32 bit**: converte l'immagine attiva, che deve essere a toni di grigio a 8 bit o 16 bit, in una a toni di grigio a 32 bit;

4. **8 bit Color**: converte l'immagine attiva, che deve essere a colori RGB, in una indicizzata a colori a 8 bit mediante l'algoritmo di color quantization di Heckbert;
5. **RGB Color**: converte l'immagine attiva, che deve essere a toni di grigio, a colori a 8 bit, RGB (red, green, blue), HSB, in una RGB a 32 bit;
6. **RGB Stack**: converte l'immagine attiva, che deve essere RGB, in uno stack con tre slices (una immagine per ogni canale);
7. **HSB Stack**: converte l'immagine attiva, che deve essere RGB, in uno stack con tre slices (una immagine per ogni canale);

aggiustare contrasto e luminosità, livelli di soglia e dimensione della immagine attiva (item Adjust):

1. **Brightness/Contrast . . .**: consente di alterare interattivamente luminosità e contrasto della immagine attiva;
2. **Threshold . . .**: consente di impostare interattivamente il più piccolo ed il più grande valore di soglia, segmentando l'immagine nelle features di interesse e nel background. I pixel con valore di luminosità maggiore o uguale alla soglia più bassa e minore o uguale alla soglia più alta sono visualizzati in rosso.
3. **Size . . .**: consente di alterare le dimensioni della immagine attiva o della selezione corrente;

mostrare una finestra contenente informazioni sulla immagine attiva (titolo, larghezza, altezza, bit per pixel, ecc . . .) (item Show Info . . .);

gestire i colori della immagine attiva (item Color):

1. **Convert Stack to RGB**: converte uno stack con 2 o 3 slices, s toni di grigio a 8 o 16 bit, in una immagine RGB;

2. **Color Picker . . .**: consente all'utente di selezionare i colori di background e foreground; viene mostrata una palette di colori basata sul modello HSB tra cui scegliere i colori di background e foreground.

lavorare con stacks di immagini (item Stacks):

1. **Add slice**: aggiunge uno slice vuoto dopo quello correntemente visualizzato;
2. **Delete slice**: cancella lo slice correntemente visualizzato;
3. **Next slice**: visualizza lo slice che segue quello correntemente visualizzato;
4. **Previous slice**: visualizza lo slice che precede quello correntemente visualizzato;
5. **Set slice . . .**: visualizza lo slice specificato mediante uno slice number inserito dall'utente;
6. **Convert Images to Stack**: crea un nuovo stack che consiste di tutte le immagini correntemente visualizzate in finestre separate; le immagini devono essere dello stesso tipo e dimensione;
7. **Convert Stack to Images**: converte le slices nello stack corrente in finestre separate che visualizzano ciascuna una diversa slice dello stack;
8. **Make Montage**: produce una singola immagine che contiene le immagini (slices) dello stack visualizzate in una griglia; ciò facilita il confronto tra immagini;
9. **Reslice . . .**: ricostruisce una o più slices ortogonali attraverso il volume dell'immagine rappresentata dallo stack corrente;
10. **ZProject . . .**: proietta uno stack di immagine lungo l'asse perpendicolare al piano dell'immagine (asse "z");
11. **3D Project . . .**: genera una sequenza animata proiettando attraverso una rotazione un insieme di dati in 3D su un piano; ogni frame della sequenza è il risultato della proiezione da una differente angolazione visuale;

12. **Plot Z-axis Profile**: plotta il valor medio di grigio della selezione ROI rispetto al numero di slice; richiede una selezione;
13. **Start Animation**: anima lo stack attivo visualizzando ripetutamente le sue slices (immagini) in sequenza;
14. **Stop Animation**: termina l'animazione dello stack attivo;
15. **Animation Options . . .**: questo dialogo consente di settare la velocità di animazione espressa in frames al secondo oppure di abilitare l'animazione "oscillante".

effettuare il crop dell'immagine o dello stack sulla base della selezione rettangolare corrente (item Crop);

creare una nuova finestra contenente una copia della immagine attiva o una selezione rettangolare (item Duplicate);

ridimensionare l'immagine o la selezione attiva orizzonatalmente e/o verticalmente riscalandola secondo valori specificati dall'utente (item Scale . . .);

ruotare l'immagine attiva o lo stack attivo attraverso una serie di comandi (item Rotate):

1. **Flip Vertical**: volge a rovescio l'immagine o la selezione;
2. **Flip Horizontal**: sostituisce l'immagine o la selezione con una immagine specchio dell'originale;
3. **Rotate 90 Degrees Right**: ruota l'intera immagine o stack di 90 gradi in senso orario;
4. **Rotate 90 Degrees Left**: ruota l'intera immagine o stack di 90 gradi in senso antiorario;
5. **Arbitrarily . . .**: questo dialogo consente di ruotare in senso orario l'immagine o la selezione attiva di un angolo specificato in gradi dall'utente.

applicare alle immagini a toni di grigio una selezione di lookup tables di colore per produrre immagini a falsi colori (item Lookup Tables):

1. **Invert LUT**: inverte la lookup table corrente. Per le immagini a 8 bit ciò equivale a sostituire ogni entry della tabella che ha valore v con il valore 255-v;
2. **Apply LUT**: applica la lookup table corrente ad ogni pixel dell'immagine o della selezione e ripristina la funzione identità di default.

menu Process: contiene degli items per:

smussare l'immagine o la selezione attiva; questo filtro sostituisce ad ogni pixel la media del suo intorno 3x3 (item Smooth);

aumentare il contrasto ed evidenziare i dettagli dell'immagine o della selezione corrente, accentuando eventualmente il rumore. Questo filtro utilizza il seguente kernel $\begin{vmatrix} -1 & -1 & -1 \\ -1 & 12 & -1 \\ -1 & -1 & -1 \end{vmatrix}$ e sostituisce ogni pixel con la media pesata dell'intorno 3x3 (item Sharpen);

evidenziare i cambiamenti acuti di intensità nella immagine o nella selezione attiva mediante il filtro Sobel edge detector ($\begin{vmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix}, \begin{vmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{vmatrix}$) (item Find Edges);

aumentare il contrasto dell'immagine usando l'histogram stretching o l'histogram equalization. Entrambi i metodi sono descritti in dettaglio in Hypermedia Image Processing Reference all'URL http://www.dai.ed.ac.uk/HIPR2/. Questo comando non altera i valori dei pixels finchè le opzioni *Normalize* o *Equalize Histogram* non sono attivate:

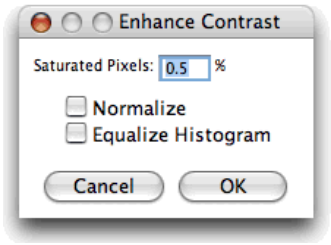


Figura 10 Il dialogo per contrast enhancement

Il campo *Satured Pixels* determina il numero di pixels nella immagine che possono essere saturati. Aumentare tale valore aumenterà il contrasto. Se si attiva l'opzione *Normalize* ImageJ ricalcolerà i valori dei pixel dell'immagine in modo che il range sia uguale al massimo range per quel tipo di dati, o a [0, 1.0] per le immagini float. Il massimo range è [0, 255] per immagini a 8 bit e [0, 65535] per immagini a 16 bit. La normalizzazione di immagini RGB non è supportata. Se viene attivato *Equalize Histogram*, l'immagine verrà migliorata equalizzando l'istogramma (item Enhance Contrast);

aggiungere rumore alle immagini o rimuoverlo (item Noise):

1. **Add noise**: aggiunge rumore random all'immagine o alla selezione. Il rumore è normalmente Gaussiano distribuito con media pari a 0 e deviazione standard pari a 25;
2. **Add more noise**: aggiunge rumore Gaussiano con media pari a 0 e deviazione standard pari a 75;
3. **Salt and Pepper**: aggiunge rumore salt e pepper alla immagine o alla selezione sostituendo in maniera random il 2.5 % dei pixel con pixel neri e il 2.5 % con pixel bianchi. Questo comando funziona solo con immagini a 8 bit;
4. **Despeckle**: è un filtro mediano; sostituisce ogni pixel con il valore mediano nel suo intorno 3x3; i 9 pixels della finestra devono essere ordinati e il pixel centrale deve essere sostituito con il valore mediano (il quinto). Il filtro mediano è utile per la rimozione di rumore salt and pepper.

produrre un effetto ombra con la luce che proviene dalla direzione corrispondente al nome del comando. I comandi usano la funzione di convoluzione di ImageJ *Convolve3x3* (item Shadows);

processare immagini binarie; si assume che gli oggetti siano neri ed il background bianco (item Binary):

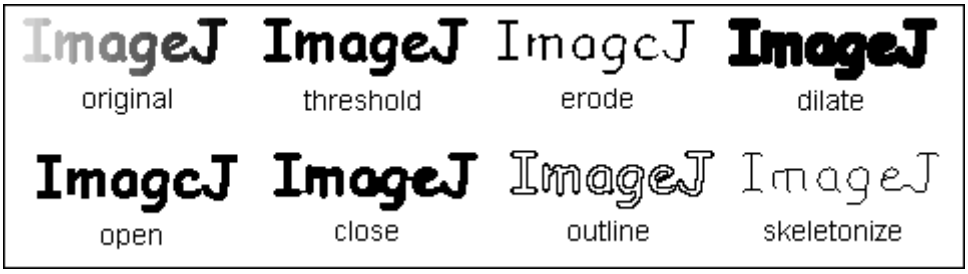


Figura 11 Varie opzioni del menu Binary

1. **Threshold**: converte immagini a toni di grigio in immagini binarie;
2. **Erode**: sostituisce ogni pixel con il minimo (il più chiaro) valore nell'intorno 3x3. Nelle immagini binarie rimuove pixel dai lati degli oggetti neri;
3. **Dilate**: sostituisce ogni pixel con il massimo (il più scuro) valore nell'intorno 3x3. Nelle immagini binarie aggiunge pixel ai lati degli oggetti neri;
4. **Open**: effettua una operazione di erosione seguita da una dilazione. Nelle immagini binarie smussa gli oggetti e rimuove pixel isolati;
5. **Close**: effettua una operazione di dilazione seguita da una erosione. Nelle immagini binarie smussa gli oggetti e riempie piccoli buchi;
6. **Set Iterations . . .**: consente di specificare quante volte devono essere eseguite le suddette operazioni. Il valore di default è 1;
7. **Outline**: genera un bordo di larghezza un pixel in corrispondenza degli oggetti neri in una immagine binaria;
8. **Skeletonize**: rimuove pixel dai lati degli oggetti di una immagine binaria finchè essi non vengono ridotti a scheletri di larghezza pari a un pixel;
9. **Distance Map**: genera una mappa basata sulla distanza Euclidea. Ogni pixel di foreground (nero) nella immagine binaria è sostituito con un pixel di colore grigio uguale alla distanza del pixel dal suo pixel di background (bianco) più vicino;
10. **Ultimate Points**: genera gli ultimi punti "eroded" della mappa Euclidea;

11. **Watershed**: la segmentazione Watershed è un modo per separare automaticamente oggetti che si toccano.

effettuare operazioni matematiche sulla immagine attiva (item Math):

1. **Add . . .**: aggiunge una costante all'immagine;
2. **Subtract . . .**: sottrae una costante dall'immagine;
3. **Multiply . . .**: moltiplica l'immagine per la costante reale specificata;
4. **Divide . . .**: divide l'immagine per la costante reale specificata;
5. **AND . . .**: effettua un AND bit a bit dell'immagine con la costante binaria specificata;
6. **OR . . .**: effettua un OR bit a bit dell'immagine con la costante binaria specificata;
7. **XOR . . .**: effettua uno XOR bit a bit dell'immagine con la costante binaria specificata;
8. **Min . . .**: tutti i pixel che hanno valore minore della costante specificata sono sostituiti con la costante stessa;
9. **Max . . .**: tutti i pixel che hanno valore maggiore della costante specificata sono sostituiti con la costante stessa;
10. **Gamma . . .**: applica la funzione *f(p) = (p/255)^gamma*255* ad ogni pixel (*p*) nell'immagine o nella selezione dove 0.1 <= *gamma* <= 5.0. Per le immagini RGB questa funzione è applicata a tutti e tre i canali. Per le immagini a 16 bit il valore minimo ed il massimo sono usati per lo scaling al posto di 255.
11. **Log . . .**: per le immagini a 8 bit, applica la funzione *f(p) = log(p) * 255/log(255)* ad ogni pixel (*p*) nell'immagine o nella selezione. Per le immagini RGB questa funzione è applicata a tutti e tre i canali. Per le immagini a 16 bit il valore minimo ed il massimo sono usati per lo scaling al posto di 255. Per le immagini

float, nessuno scaling è effettuato. Per calcolare log10 dell'immagine, moltiplicare il risultato di questa operazione per 0.4343 (1/log(10));

12. **Reciprocal**: genera la reciproca della immagine attiva o della selezione. Funziona solo con immagini float a 32 bit;
13. **NaN Background**: setta i pixel non sogliati, in una immagine float a 32 bit, al valore NaN (Not a Number).
14. **Abs**: genera il valore assoluto della immagine o della selezione attiva. Funziona solo con immagini float a 32 bit.

effettuare visualizzazioni ed operazioni nel dominio di Fourier basandosi sulla implementazione della 2D Fast Hartley Transform (FHT) (item FFT):

1. **FFT**: computa la trasformata di Fourier e visualizza lo spettro di potenza.
2. **Inverse FFT**: computa la trasformata di Fourier inversa.
3. **Redisplay Power Spectrum**: ricomputa lo spettro di potenza a partire dall'immagine nel dominio delle frequenze (32-bit FHT). Questo comando consente di ricominciare nel caso in cui sia stato modificato lo spettro di potenza dell'immagine a 8 bit.
4. **Bandpass Filter . . .**: si tratta di un plugin per l'implementazione del filtraggio passa banda.
5. **Custom Filter . . .**: consente di effettuare un filtraggio nel dominio di Fourier dell'immagine attiva usando un filtro specificato dall'utente.

effettuare operazioni di filtraggio nel dominio spaziale mediante una serie di filtri e plugins installati mediante il comando *Plugins/Utilities/Install Plugin* (item Filters):

1. **Convolve . . .**: effettua la convoluzione spaziale usando un kernel specificato dall'utente; non vi è limite alla dimensione del kernel ma esso ovviamente deve essere quadrato; l'opzione *Normalize Kernel* divide ogni coefficiente del kernel per la somma di tutti i coefficienti, preservando la luminosità dell'immagine;

2. **Gaussian Blur . . .**: smussa l'immagine corrente effettuando la convoluzione mediante un kernel quadrato Gaussiano; la larghezza del kernel, in pixel, è 2**radius*+1, dove *radius* è inserito mediante una dialog box;
3. **Median . . .**: riduce il rumore nella immagine attiva sostituendo ogni pixel con il mediano dei valori dei pixel dell'intorno;
4. **Mean . . .**: smussa l'immagine corrente sostituendo ogni pixel con la media del suo intorno; la dimensione dell'intorno è specificata inserendo il raggio in un dialog box;
5. **Minimum . . .**: questo filtro effettua l'erosione a toni di grigio sostituendo ogni pixel nell'immagine con il più piccolo valore nell'intorno del pixel;
6. **Maximum . . .**: questo filtro effettua la dilazione a toni di grigio sostituendo ogni pixel nell'immagine con il più grande valore nell'intorno del pixel;
7. **Unsharp Mask . . .**: evidenzia i lati sottraendo all'immagine originale una versione smussata (blurred) dell'immagine (unsharp mask). L'immagine unsharp mask è creata effettuando un blurring Gaussiano dell'immagine originale e moltiplicando l'immagine così ottenuta per il parametro "Mask Weight". Aumentando il raggio del blurring Gaussiano, aumenterà il contrasto e il valore "Mask Weight" con l'effetto di un edge enhancement aggiuntivo;
8. **Variance . . .**: evidenzia i lati nell'immagine sostituendo ogni pixel con la varianza del vicinato;
9. **Show Circular Masks**: genera uno stack contenente esempi delle maschere circolari usate dai filtri *Median*, *Mean*, *Minimum*, *Maximum* e *Variance* per varie dimensioni dell'intorno.

effettuare operazioni aritmetiche e logiche tra due immagini selezionate mediante popup menus (item Image Calculator . . .);

Add	img1 = img1+img2
Subtract	img1 = img1-img2
Multiply	img1 = img1*img2
Divide	img1 = img1/img2
AND	img1= img1 AND img2
OR	img1 = img1 OR img2
XOR	img1 = img1 XOR img2
Min	img1 = min(img1,img2)
Max	img1 = max(img1,img2)
Average	img1 = (img1+img2)/2
Difference	img1 = |img1-img2|
Copy	img1 = img2

Tabella 1 Operazioni che è possibile eseguire usando l'item Image Calculator . . .

rimuovere sfondi continui smussati da gels e altre immagini (item Subtract Background . . .);

ripetere l'esecuzione del comando precedente (item Repeat Command);

menu Analyze: contiene degli items per:

calcolare e visualizzare statistiche relative all'area o alla lunghezza delle linee di una selezione (item Measure);

contare e misurare gli oggetti in immagini binarie o sogliate (item Analyze Particles . . .);

calcolare e visualizzare la media, la deviazione standard, il minimo e il massimo dei valori di ciascuna colonna della tabella dei risultati (item Summarize);

cancellare la tabella dei risultati e reimpostare il contatore di misure (item Clear Results);

specificare quali misure (area, deviazione standard, minimo e massimo livello di grigio, centro di massa, bounding rectangle, circolarità, valor medio di grigio, valore di grigio modale, centroide, perimetro, fit ellipse, diametro di Feret, visualizza etichetta, limite

alla threshold, visualizza etichetta) considerare nei comandi *Analyze/Measure* e *Analyze/Analyze Particles* (item Set Measurements . . .);

definire una scala spaziale dell'immagine attiva in modo tale che i risultati relativi alle misure possano essere presentati in unità calibrate come ad esempio i millimetri (item Set Scale . . .);

calibrare una immagine ad un set di standard di densità, per esempio gli standard dell'isotopo radioattivo o una tavoletta con passo di densità ottica calibrata (item Calibrate . . .);

calcolare e visualizzare un istogramma della distribuzione dei livelli di grigio nella immagine o nella selezione attiva; è possibile anche salvare gli istogrammi (item Histogram);

visualizzare un grafico bidimensionale delle intensità dei pixel lungo una linea (item Plot Profile);

visualizzare un grafico tridimensionale delle intensità dei pixel in una immagine a toni di grigio o a pseudo colori (item Surface Plot);

visualizzare un grafico della lookup table dell'immagine attiva; le immagini a colori RGB non utilizzano una lookup table (item Show LUT);

analizzare gels elettroforetici in una dimensione utilizzando il plugin Gel Analyzer (item Gels);

analizzare le immagini mediante l'uso di opportuni plugins (item Tools):

1. **Save XY Coordinates . . .**: scrive in un file di testo le coordinate X e Y e i valori dei pixel per tutti i pixel che non fanno parte del background dell'immagine attiva;
2. **Box Count . . .**: conta il numero di box di dimensione crescente necessari a coprire il confine ad un pixel di un oggetto binario;

3. **Analyze Line Graph**: consente di ricostruire dati relativi a coordinate numeriche a partire da grafici a linea analizzata;
4. **ROI Manager**: consente di lavorare con selezioni di area multiple;
5. **Calibration Bar**: crea una copia RGB dell'immagine corrente e visualizza una calibration bar etichettata su essa.

menu Plugins: vedi paragrafo 2.3.

menu Window: contiene degli items per:

portare in primo piano la finestra di ImageJ premendo il tasto enter (item ImageJ [enter]);

visualizzare l'immagine aperta successiva; mediante il tasto tab è possibile scorrere tutte le immagini aperte (item PutBehind [tab]).

2.3 Descrizione dei plugins, delle macro e del menu Plugins

I plugins e le macro sono moduli di codice che consentono di estendere le potenzialità di ImageJ. I plugins sono scritti in linguaggio JAVA e vengono compilati producendo files .class. Le macro, scritte in un linguaggio JAVA-like, sono memorizzate in files .txt. I plugins vengono eseguiti più velocemente e sono più flessibili mentre le macro sono più facili da scrivere e debuggare. I plugins e le macro nella cartella `plugins`, con un underscore nel loro nome, sono visualizzate nella parte bassa del menu *Plugins*. I plugins e le macro nelle sottocartelle della cartella `plugins` sono visualizzati nei sottomenu del menu *Plugins*.

Molti comandi in ImageJ sono implementati come plugins ma questi plugins interni sono collocati all'interno del file `ij.jar`, non nella cartella `plugins`. Il file `ij.jar` contiene anche un file che specifica le proprietà (IJ_Props.txt) che ImageJ usa per installare plugins interni nei menu. Utilizzando un programma di compressione/decompressione (es. WinZip) è possibile estrarre i contenuti del file `ij.jar`. E' inoltre possibile convertire un plugin interno in un plugin creato

dall’utente semplicemente copiando il codice sorgente cartella `plugins` e aggiungendo un underscore al nome del file e al nome della classe e inoltre cambiando l’intestazione `package` in `import`. Ad esempio per cambiare il comando RoiManager (Analyze/Tools/ROI Manager) facendolo diventare un plugin scritto dall’utente bisogna effettuare i seguenti passi:

1. Copiare `ij/plugin/frame/RoiManager.java` nella cartella `plugins`;
2. Cambiare il nome del file a `Roi_Manager.java`;
3. Aprire `Roi_Manager.java` utilizzando il comando *Plugins/Edit* e cambiare tutte le occorrenze (4) di "RoiManager" in "Roi_Manager";
4. Cambiare la prima linea da "package ij.plugin.frame;" a "import ij.plugin.frame.*;".
5. Eseguire il nuovo plugin usando il comando *File/Compile and Run*.

In tal modo comparirà nel menu *Plugins* l’item *Roi Manager* al successivo riavvio di ImageJ.

E’ inoltre possibile cambiare la locazione della cartella `plugins` alla quale corrisponde la proprietà "plugins.dir". Questa proprietà può essere modificata sia da linea di comando sia all’interno del codice del programma JAVA che avvia ImageJ. Per esempio se si esegue ImageJ con il comando:

java –D plugins.dir=/Applications/ImageJ -cp ../ij.jar:. ij.ImageJ

la cartella `plugins` verrà cercata in /Applications/ImageJ. Questa proprietà può essere modificata anche all’interno del codice del programma che lancia ImageJ aggiungendo il seguente comando:

System.getProperties().setProperty("plugins.dir", "/Applications/ImageJ");

new ImageJ(null);

Il **menu Plugins** contiene items per:

installare, eseguire e registrare macro (item Macros):

1. **Install . . .**: aggiunge una o più macro contenute in un file nella parte bassa del sottomenu Macros; per installare un insieme di macro e nel frattempo visualizzare il loro codice sorgente, aprire il file della macro con il comando *File/Open* e usare il comando *Macros/Install Macros*. Le macro nel file

ImageJ/macros/StartupMacros.tx sono automaticamente installate all'avvio di ImageJ;

2. **Run . . .**: carica ed esegue una macro senza aprirla nell'editor di ImageJ; per eseguire una macro e contemporaneamente visualizzare il codice aprirla con *File/Open* e usare il comando *File/Run Macro*;
3. **Record . . .**: apre il registratore di comandi di ImageJ; per creare una macro aprire il registratore, usare uno o più comandi di ImageJ ed infine cliccare su "Create". Quando il registratore è aperto, ogni comando utilizzato produce una chiamata alla funzione run() che ha una o due argomenti: il primo è il nome del comando mentre il secondo, opzionale, contiene un dialogo con la specifica di parametri;

creare collegamenti comando-tasto, installare e rimuovere plugins (item Shortcuts):

1. **Create Shortcut . . .**: associa un tasto ad un comando di ImageJ ed inserisce questa nuova associazione (collegamento) nel sottomenu Shortcuts;
2. **Install Plugin . . .**: installa un plugin in un sottomenu specificato dall'utente. I plugins con un metodo showAbout() sono anche automaticamente aggiunti al sottomenu *Help/About Plugins*. Come illustrato in figura 12, si utilizza il primo menu a popup per selezionare il plugin ed il secondo menu a popup per selezionare il sottomenu in cui si vuole installare il plugin; il comando deve essere diverso da uno già esistente; il collegamento (Shortcut) è opzionale e deve essere una singola lettera o un valore da "F1" a "F12"; l'argomento, anche esso opzionale, è la stringa che sarà passata in input al metodo run del plugin.

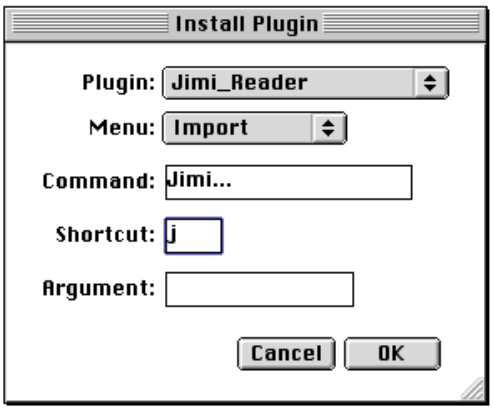


Figura 12 dialogo per l'installazione dei plugins

3. **Remove . . .**: rimuove i comandi aggiunti al sottomenu *Shortcuts* mediante il comando *Create Shortcuts*. Rimuove anche comandi aggiunti mediante *Install Plugin* e rimuove plugins installati nel menu *Plugins*.

usufruire di varie utilità (item Utilites):

1. **Control Panel . . .**: questo comando apre una finestra contenente i comandi di ImageJ in una struttura gerarchica ad albero;
2. **Monitor Memory . . .**: visualizza un grafico continuamente aggiornato sull'utilizzo di memoria di ImageJ;
3. **Capture Screen**: copia la schermata corrente in una immagine RGB e visualizza questa immagine in una nuova finestra.

aprire una nuova finestra di testo contenente un prototipo (in termini di codice JAVA) per uno dei tre tipi di plugins supportati da ImageJ (item New . . .);

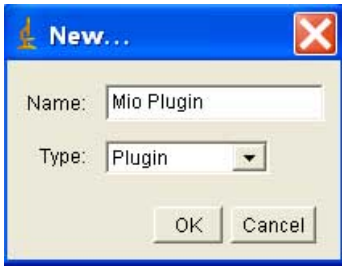


Figura 13 Finestra di dialogo per la creazione di un nuovo plugin

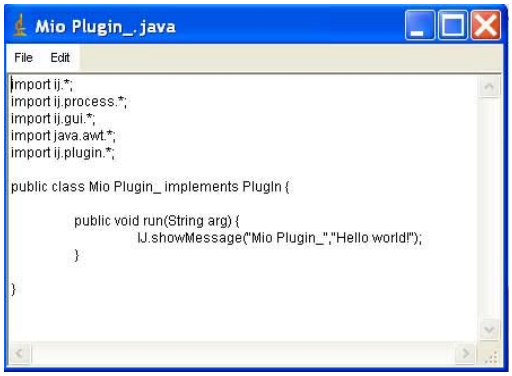


Figura 14 Prototipo JAVA per uno dei tre tipi di plugins

aprire una finestra di testo che consente di modificare, compilare ed eseguire plugins (item Edit . . .);

compilare ed eseguire plugins; richiede la presenza di un compilatore javac; (item Compile and Run . . .);

2.4 Tabella dei collegamenti (Keyboard Shortcuts)

La tabella 2 mostra alcune associazioni tra tasti e comandi di ImageJ:

Comando	Tasto/i	Descrizione
New	N	Crea una nuova immagine o stack
Open	O	Apre una TIFF, GIF, JPEG, BMP, DICOM or FITS
Open Samples	Shift-B	Apre immagini di esempio "Blobs"
Close	W	Chiude la finestra attiva
Save	S	Salva l’immagine attiva in formato Tiff
Revert	R	Ritorna alla ultima versione salvata dell’immagine
Print	P	Stampa l’immagine attiva
Undo	Z	Annulla l’effetto dell’ultima operazione
Cut	X	Copia l’immagine nella clipboard e cancella la selezione
Copy	C	Copia l’immagine nella clipboard
Paste	V	Incolla la selezione della clipboard nell’immagine attiva
Clear	backspace	Cancella la selezione sostituendola con un’area avente colore pari a quello di background
Select All	A	Seleziona l’intera immagine
Select None	Shift-A	Cancella la selezione
Restore Selection	Shift-E	Ripristina la ROI

Fill	F	Riempie la selezione con il colore di foreground
Draw	D	Disegna la selezione
Invert	Shift-I	Inverte l'immagine o la selezione
Adjust Contrast	Shift-C	Aggiusta luminosità e contrasto
Adjust Threshold	Shift-T	Aggiusta i livelli di soglia
Show Info	I	Visualizza informazioni sull'immagine attiva
Next Slice	>	Avanza alla prossima slice nello stack
Previous Slice	<	Torna alla precedente slice nello stack
Start Animation	=	Inizia/interrompe l'animazione dello stack
Duplicate	Shift-D	Duplica l'immagine o la selezione attiva
Scale	E	Scala l'immagine o la selezione attiva
Smooth	Shift-S	Applica un kernel 3x3 di smmothing
Find Edges	Shift-F	Effettua la Sobel edge detection
Repeat Command	Shift-R	Ripete il comando precedente
Measure	M	Visualizza statistiche sulla immagine o sulla selezione
Histogram	H	Visualizza l'istogramma della finestra o della selezione attiva
Plot Profile	K	Visualizza un grafico del profile di densità della selezio ne corrente
ImageJ	enter	Porta in primo piano la finestra di ImageJ
Put Behind	tab	Passa alla finestra che visualizza l'immagine successiva

Tabella 2 Tabella dei keyboard Shortcuts

3 Struttura di ImageJ

3.1 Struttura delle classi di ImageJ

Il programma è strutturato in packages. Ecco di seguito un breve descrizione delle classi che ogni package contiene:

- **Package ij**
    1. **ImageJApplet**

ImageJ può essere eseguito come applet o come applicazione. Questa è la sua classe applet. Il vantaggio dell'esecuzione come applet è la possibilità di far girare il programma (in remoto) all'interno di un browser. Lo svantaggio più grosso è l'accesso limitato ai files sul disco a causa del concetto di sicurezza delle applet Java (se l'applet non è firmata).

2. **ImageJ**

   La classe main dell'applicazione ImageJ. Questa classe contiene il metodo *run*, che è l'entry point del main, e la finestra principale di ImageJ.

3. **Executer**

   Una classe per eseguire i comandi del menu in threads separati (senza bloccare il resto del programma).

4. **IJ**

   Una classe che contiene molte utilities.

5. **ImagePlus**

   La rappresentazione di un'immagine in ImageJ, che è basata su *ImageProcessor*.

6. **ImageStack**

   Una *ImageStack* è un'array espandibile di immagini.

7. **WindowManager**

   Questa classe organizza e controlla la lista delle finestre aperte.

- **Package ij.gui**

  1. **ProgressBar**

     Una barra nella finestra principale di ImageJ che informa graficamente sul progresso dell'operazione in esecuzione.

  2. **GenericDialog**

     Un dialogo che può essere customizzato e attivato, per esempio, per caricare l'input dell'utente prima di mandare in esecuzione il plugin.

3. **NewImage**

   Una classe per creare una nuova immagine di un certo tipo.

4. **Roi**

   Una classe che rappresenta una regione di interesse dell'immagine. Se supportato dal plugin, è possibile effettuare una certa operazione solo sulla ROI e non sull'intera immagine.

5. **ImageCanvas**

   Canvas deriva da *java.awt.Canvas*. Si tratta di una sorta di "tela" virtuale su cui l'immagine è dipinta.

6. **ImageWindow**

   Un frame, derivante da *java.awt.Frame*, che permette di visualizzare l'immagine.

7. **StackWindow**

   Una ImageWindow progettata per visualizzare stack di immagini.

8. **HistogramWindow**

   Una ImageWindow progettata per visualizzare istogrammi.

9. **PlotWindow**

   Una ImageWindow progettata per visualizzare plots.

- **Package ij.io**

  Questo package contiene per la lettura/decodifica e per la scrittura/codifica di files immagine.

- **Package ij.measure**

  Contiene le classi per le misure.

- **Package ij.plugin**

  Molti comandi del menu di ImageJ sono implementati come plugin e, quindi, possono essere trovati nelle classi del package ij.plugin e dei suoi sotto-packages.

1. **PlugIn**

E' un'interfaccia che deve essere implementata da plugins che non necessitano di un'immagine in input.

2. **Converter**

Implementa un metodo per convertire una *ImagePlus* da un formato ad un altro.

- **Package ij.plugin.filters**

1. **PlugInFilter**

E' un'interfaccia che deve essere implementata da plugins che necessitano di un'immagine in input.

- **Package ij.plugin.frame**

1. **PlugInFrame**

Una classe per la finestra che può essere suddivisa in ulteriori classi da un plugin.

- **Package ij.process**

1. **ImageConverter**

Una classe che contiene metodi per convertire la immagini da un formato ad un altro.

2. **ImageProcessor**

Una superclasse astratta di processori immagini per certi formati di immagine. Un processore di immagine fornisce metodi per lavorare sui dati forniti da essa forniti**.**

3. **StackConverter**

Una classe per convertire stacks di immagini da un formato ad un altro.

4. **StackProcessor**

Una classe per processare stacks di immagini.

- **Package ij.text**

Questo package contiene classi per visualizzare ed editare testo.

3.2 Il concetto di Plugin in ImageJ

Le funzionalità fornite dai comandi dal menu di ImageJ possono essere estesi con plugin scritti dall'utente. Questi plugin sono classi java che implementano le interfacce necessarie e collocati in una certa cartella. I plugin possono essere scritti attraverso un editor di testo (accessibile dal menu "Plugins/New..." e "Plugins/Edit...") o attraverso un recorder di plugins. In ogni caso i nuovi plugins, trovati da ImageJ, vengono automaticamente inseriti nel menu Plugin o in sottomenu.
Di base esistono 2 tipi di plugins: quelli che non necessitano di un'immagine in input (che implementano l'interfaccia *PlugIn*) e i *plugin filters*, che necessitano un'immagine in input (che implementano l'interfaccia *PlugInFilter*)

3.2.1 Plugin

Questa interfaccia ha solo un metodo

- *void run(java.lang.String arg)*

  Questo metodo manda in esecuzione il plugin. L'argomento arg è una stringa (che può essere anche vuota). E' possibile istallare plugins più di una volta, cosicché ogni comando può chiamare la stessa classe con diversi argomenti.

3.2.2 Plugin Filter

Questa interfaccia ha anche un metodo

- *void run(Image Processor ip)*

  Questo metodo manda in esecuzione il plugin. L'input passato come argomento è il processore di immagine. Il processore può essere modificato direttamente o può fornire dati per creare un nuovo processore o una nuova immagine, in modo che l'immagine originale rimanga invariata. L'immagine rimane bloccata fino a quando l'esecuzione del plugin non è terminata.

L'argomento può essere passato attraverso il metodo

- *int setup(java.lang.String arg, ImagePlus imp)*

  Questo metodo setta il *plugin filter* per l'uso. La stringa *arg* ha la stessa funzione che ha nel metodo *run* dell'interfaccia *PlugIn*. L'utente non si deve preoccupare dell'argomento *imp*: è ImageJ che se ne occupa passando in automatico l'immagine attiva. Il metodo setup restituisce un flag che rappresenta le capacità del filtro.

I flag definiti in *PlugInFilter* sono:

- *static int DOES 16* Il *plugin filter* tratta immagini a toni di grigio a 16 bit.
- *static int DOES 32* Il *plugin filter* tratta immagini a toni di grigio a 32 bit (floating point).
- *static int DOES 8C* Il *plugin filter* tratta immagini a colori a 8 bit.
- *static int DOES 8G* Il *plugin filter* tratta immagini a toni di grigio a 8 bit.
- *static int DOES ALL* Il *plugin filter* tratta tutti i tipi di immagine.
- *static int DOES RGB* Il *plugin filter* tratta immagini RGB.
- *static int DOES STACKS* Il *plugin filter* tratta stacks; ImageJ lo invoca per ogni immagine che ne fa parte.
- *static int DONE* Se il metodo setup restituisce DONE, il metodo run non verrà invocato.
- *static int NO CHANGES* Il *plugin filter* non modifica i dati dell'immagine.
- *static int NO IMAGE REQUIRED* Il *plugin filter* non necessita che venga aperta un'immagine.
- *static int NO UNDO* Il *plugin filter* non necessita di undo.
- *static int ROI REQUIRED* Il *plugin filter* ha bisogno di una regione di interesse (ROI).
- *static int STACK REQUIRED* Il *plugin filter* ha bisogno di uno stack.
- *static int SUPPORTS MASKING* Il *plugin filter* lavora sempre all'interno della regione rettangolare della ROI.

3.3 Istallazione dei plugins

I plugin creati dall'utente devono essere collocati all'interno di una cartella chiamata *plugins*, che è una sotto-cartella di *ImageJ*. Ma solo i files in *plugins* con almeno un underscore nel loro nome appariranno nel menu. Dalla versione 1.20 è anche possibile creare sotto-cartelle. Le sotto-cartelle verranno visualizzate come sottomenu del menu *Plugins*.

Per istallare un plugin bisogna copiare il file .class nella cartella *plugins* o in una delle sue sotto-cartelle. Il plugin apparirà nel menu appena verrà mandato in esecuzione. In alternativa, è possibile aggiungerlo direttamente al menu selezionando "Plugins / Shortcut / Install plugin".

3.4 Un semplice plugin (esempio)

Consideriamo come esempio il plugin Inverter_, che inverte l'immagine corrente in un'immagine a toni di grigio a 8 bit. All'inizio vengono importati i packages necessari: *ij* per le classi di base di ImageJ, *ij.process* per i processori immagine e *ij.plugin.filter.PlugInFilter* per l'interfaccia:

```
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;
import java.awt.*;
```

La parola chiave *package* non deve essere usata dentro le classi del plugin, ma solo nel package di default.

Il plugin ha bisogno un'immagine in input, quindi deve implementare *PlugInFilter* :

```
public class Inverter_ implements PlugInFilter {
```

A questo punto vengono invocati i metodi per il settaggio. Nel caso in cui si deve passare "about" come argomento, viene invocato *showAbout* che mostra una finestra di dialogo. In quel caso

restituisce DONE in modo che il *run* non venga chiamato. Altrimenti viene restituito un altro flag. *Inverter_* opera su immagini a toni di grigio a 8 bit e riesce a gestire sia stacks che ROI.

```
public int setup(String arg, ImagePlus imp) {
        if (arg.equals("about")) {
                showAbout();
                return DONE; }
        return DOES_8G+DOES_STACKS+SUPPORTS_MASKING;
}
```

Il *run* implementa la funzione principale del plugin: viene caricato il processore dell'immagine e ne ricava un array di pixels (256 possibili valori usando un array di bytes). Dall'array, che è unidimensionale, viene calcolata la larghezza dall'immagine e i limiti del rettangolo della ROI.

```
public void run(ImageProcessor ip) {
        byte[] pixels = (byte[])ip.getPixels();
        int width = ip.getWidth();
        Rectangle r = ip.getRoi();
```

Dopo vengono dichiarate 2 variabili per evitare di calcolare ogni volta la posizione dell'array unidimensionale dell'immagine. Nel loop più esterno si itera dalla prima linea della ROI fino all'ultima, si calcola l'offset (la posizione del primo pixel della corrente linea) e si procede dal pixel più a sinistra a quello più a destra (con il ciclo più interno). Si assegna *i* alla corrente posizione e si inverta il valore del pixel sottraendogli 255.

```
int offset, i;
for (int y=r.y; y<(r.y+r.height); y++) {
        offset = y*width;
        for (int x=r.x; x<(r.x+r.width); x++) {
                i = offset + x;
                pixels[i] = (byte)(255-pixels[i]); }
}
```

showAbout usa il metodo statico *showMessage* della classe *IJ* per visualizzare un messaggio. Il primo parametro specifica il titolo, il secondo il testo:

```
void showAbout() {
    IJ.showMessage("About Inverter_...",
    "This sample plugin filter inverts 8-bit images. Look\n" +
    "at the 'Inverter_.java' source file to see how easy it is\n" +
    "in ImageJ to process non-rectangular ROIs, to process\n" +
    "all the slices in a stack, and to display an About box."
    );
```

3.5 Registrazione di un plugin

Se il plugin deve solo eseguire una sequenza di comandi dal menu di ImageJ, non è necessario scrivere un file. "Plugins/Record" apre una finestra e registra le azioni dell'utente fino a quando "Record" è selezionato. Lo pseudocodice delle operazioni sarà visualizzato nella finestra. Il tasto "Create Plugin" genera una classe java a partire dallo pseudocodice attraverso i metodi della classe *IJ*. La classe del plugin viene aperta mediante un editor di testo che permette anche di compilarla ed eseguirla.

3.6 Compilazione ed esecuzione di plugins

Se l'ambiente runtime di java che si sta usando include un compilatore java (come, per esempio, in ImageJ) oppure sulla macchina su cui si sta lavorando è già istallato un compilatore java, è possibile compilare ed eseguire i plugins dall'interno stesso di ImageJ. Di base esistono 2 modalità:

- usare il menu "Plugins/Compile and run...", che apre un file di dialogo con cui si può caricare il file .java e che sarà compilato all'interno di un file .class ed eseguito come plugin.

- usare "File/Compile and run..." dall'editor built-in del plugin che compilerà ed eseguirà il codice della finestra di editor.

Se il plugin necessita altre librerie oltre a quelle standard Java e a quelle di ImageJ, l'utente deve modificare il classpath dell'ambiente in modo da renderle disponibili durante la compilazione e l'esecuzione del plugin.

4 Rappresentazione delle immagini

4.1 Tipi di immagini

Le immagini sono grandi array di pixels. E' importante sapere come il valore di questi pixels deve essere interpretato. Ciò è specifico del tipo dell'immagine; ne esistono 5:

- **immagine a toni di grigio (a 8 bit).** Si possono visualizzare 256 livelli di grigio. Ogni pixel è rappresentato in formato *byte*.
- **immagine a colori (a 8 bit).** Si possono visualizzare 256 colori specificati in una lookup table o LUT. Ogni pixel è rappresentato in formato *byte*.
- **immagine a toni di grigio (a 32 bit).** Si possono visualizzare 65.536 livelli di grigio. Ogni pixel è rappresentato in formato *short*.
- **immagine a colori RGB.** Si possono visualizzare 256 valori per canale. Ogni pixel è rappresentato in formato ***int.***
- **immagine a toni di grigio (a 32 bit, floating point).** Ogni pixel è rappresentato in formato *float*.

4.2 Immagini

Una *ImagePlus* è un oggetto che rappresenta un'immagine e che si basa su un *ImageProcessor*.

Un *ImageProcessor* è un oggetto che permette di lavorare con l'*array* di pixel e, quindi, effettuare operazioni sull'immagine. Il tipo di *ImageProcessor* usato dipende dal tipo di immagine. I tipi di immagine sono rappresentati da costanti dichiarate in *ImagePlus* :

- *COLOR_256* Un'immagine a colori a 8 bit con una LUT.
- *COLOR_RGB* Un'immagine a colori RGB.
- *GRAY16* Un'immagine a toni di grigio a 16 bit.
- *GRAY32* Un'immagine a toni di grigio a 32 bit (floating point).
- *GRAY8* Un'immagine a toni di grigio a 8 bit.

ImageJ visualizza immagini usando una classe chiamata *ImageWindow*, che può anche effettuare repaint, zooming, cambiamenti di maschere, ecc.

Per costruire una *ImagePlus* si usa uno dei seguenti costruttori:

- *ImagePlus()*

  Costruttore di default. Crea una nuova immagine vuota e non effettua nessuna inizializzazione.

- *ImagePlus(java.lang.String urlString)*

  Costruisce una nuova *ImagePlus* caricando l'immagine dall'URL specificato.

- *ImagePlus(java.lang.String title, java.awt.Image img)*

  Costruisce una nuova *ImagePlus* basata su un'immagine java AWT. Il primo argomento è il titolo dell' *ImageWindow* che visualizzerà l'immagine.

- *ImagePlus(java.lang.String title, ImageProcessor ip)*

  Costruisce una nuova *ImagePlus* che usa l'*ImageProcessor* specificato. Il primo argomento è il titolo dell'*ImageWindow* che visualizzerà l'immagine**.**

- *ImagePlus(java.lang.String title, ImageStack stack***)**

  Costruisce una nuova *ImagePlus* a partire da una *ImageStack*. Il primo argomento è il titolo dell'*ImageWindow* che visualizza l'immagine.

Il tipo di un'immagine può essere recuperato usando il metodo *int getType().*

Metodi simili esistono per recuperare le dimensioni dell'immagine, il titolo (= nome dell'*ImageWindow* che visualizza l'immagine), l'immagine AWT che rappresenta l'ImagePlus e informazioni sul file:

- *int getHeight()*
- *int getWidth()*
- *java.lang.String getTitle()*
- *java.awt.Image getImage()*
- *ij.io.FileInfo getFileInfo()*

L'immagine AWT sulla quale l'ImagePlus si basa e il titolo dell'immagine possono essere settati usando:

- *void setImage(java.awt.Image img)*
- *void setTitle(java.lang.String title)*

Una ImagePlus può avere una serie di proprietà aggiuntive che possono essere definiti dall'utente. Queste proprietà possono assumere qualsiasi valore e possono essere indicizzate usando una stringa. I metodi che permettono di leggerle e settarle sono:

- *java.util.Properties getProperties()*

  Restituisce le proprietà dell'immagine.
- *java.lang.Object getProperty(java.lang.String key)*

  Restituisce la proprietà associata alla chiave.
- *void setProperty(java.lang.String key, java.lang.Object value)*

  Aggiunge una coppia chiave-valore alle proprietà dell'immagine.

4.3 Processori

Ogni immagine è basata su un processore di immagine. Il tipo di processore dipende dal tipo di immagine. L'utente può recuperare e settare il processore usando i seguenti 2 metodi di ImagePlus:

- *ImageProcessor getProcessor()*

  Restituisce un riferimento all'*ImageProcessor* dell'immagine.

- *void setProcessor(java.lang.String title, ImageProcessor ip)*

  Setta il processore a quello specificato dall'argomento.

Quando si lavora con *pluginfilters*, non c'è bisogno di occuparsi di recuperare il prcessore dall'*ImagePlus*, perchè viene passato direttamente come argomento al metodo *run*.

ImageProcessor è una classe astratta. In base al tipo di immagine, viene usata la corrispondente sotto-classe di *ImageProcessor*. Ne esistono 5:

- *ByteProcessor*

  Usato per le immagini a toni di grigio a 8 bit e quelle a colori. Contiene la sotto-classe *BinaryProcessor* per le immagini a toni di grigio con i valori dei pixels che vanno da 0 a 255.

- ShortProcessor

  Usato per le immagini a toni di grigio a 16 bit.

- ColorProcessor

  Usato per le immagini intere a 32 bit (RGB con 8 bit per canale).

- FloatProcessor

  Usato per le immagini a 32 bit floating point.

4.4 Come accedere al valore dei pixels

Per lavorare con un'immagine è necessario accedere ai suoi pixels. Per recuperare i valori dei pixels esiste il metodo *java.lang.Object getPixels()* che restituisce un riferimento all'array dei pixels dell'immagine. Dato che il tipo di questo array dipende dal tipo di immagine, è necessario (dopo aver caricato i dati) effettuare un cast al tipo appropriato, per esempio:

int[] pixels = (int[]) myProcessor.getPixels()

Questo esempio indica che si vuole lavorare con un'immagine RGB.

Sappiamo che un'array è una struttura unidimensionale. Per convertire una posizione dell'array ad una coordinata (x,y) dell'immagine bisogna conoscerne la larghezza.

E' possibile conoscere la larghezza e l'altezza di un'immagine attraverso i metodi

- *int getHeight()*
- *int getWidth()*

A questo punto si può iterare attraverso tutti i pixels dell'immagine usando due cicli annidati.

In alcuni casi bisogna leggere i pixels da *ByteProcessor*, *ShortProcessor* o *ColorProcessor*. I dati byte in Java assumono valori nel range [-128,127], mentre ci si aspetta un'immagine a toni di grigio a 8 bit con i valori dei pixels nel range [0,255]. Se si effettua un cast da byte ad un altro tipo si elimina sicuramente il segno. Ciò può essere effettuato usando l'operazione binaria AND (&):

```
int pix = 0xff & pixels[i];
...
pixels[i] = (byte) pix;
```

La stessa situazione si presenta con il tipo short. In questo caso valori in [-32768,32767] devono essere mappati nell'intervallo [0,65535].

Si procede allo stesso modo:

```
int pix = pixels[i] & 0xffff;
...
pixels[i] = (short) pix;
```

ColorProcessor restituisce l'array di pixels come array di interi. I valori delle 3 componenti del colore sono compattati in un unica variabile *int*. Ogni singola componente si può calcolare nel seguente modo:

```
int red = (int)(pixels[i] & 0xff0000)>>16;
int green = (int)(pixels[i] & 0x00ff00)>>8;
int blue = (int)(pixels[i] & 0x0000ff);
pixels[i] = ((red & 0xff)<<16)+((green & 0xff)<<8) + (blue & 0xff);
```

L'array di pixels con cui si lavora è un riferimento all'array di pixels dell'*ImageProcessor*. Quindi ogni modifica ha immediatamente effetto. Se si vuole che l'ImageProcessor usi un altro array, si può usare il metodo

- *void setPixels(java.lang.Object pixels)*

Spesso non si vuole lavorare con l'intero array. ImageProcessor offre alcuni metodi per recuperare o impostare singoli pixels:

- *int getPixel(int x, int y)*

  Restituisce il valore del pixel specificato (come tipo *int*).

- *void putPixel(int x, int y, int value)*

  Imposta il pixel alla posizione (x,y) al valore specificato.

- *float getPixelValue(int x, int y)*

  Restituisce il valore del pixel specificato (nel formato *float*).

- *void getColumn(int x, int y, int[] data, int length)*

  Restituisce la colonna di pixels sotto il pixel di posizione (x,y) in data.

- *void putColumn(int x, int y, int[] data, int length)*

  Inserisce i pixels contenuti in data all'interno della colonna che inizia dalla posizione (x,y).

- *void getRow(int x, int y, int[] data, int length)*

  Restituisce i pixels lungo la riga che inizia dalla posizione (x,y) in data.

- *void putRow(int x, int y, int[] data, int length)*

  Inserisce i pixels contenuti in *data* all'interno della riga che inizia nella posizione (x,y).

- *double[] getLine(int x1, int y1, int x2, int y2)*

  Restituisce i pixels lungo la linea che inizia nel punto (x1,y1) e finisce nel punto (x2,y2).

Il metodo di ImagePlus

- *int[] getPixel(int x, int y)*

  Restituisce il valore del pixel nella posizione (x,y) in un array di 4 elementi.

Tutti questi metodi devono essere usati solo se si vogliono modificare pochi pixels. Se si vogliono modificare ampie regioni dell'immagine è molto più conveniente lavorare con l'intero array.

4.5 Regioni di interesse

Un plugin filter non deve sempre lavorare sull'intera immagine. ImageJ supporta le cosiddette "region of interest" o ROI, che possono essere selezioni di regioni dell'immagine rettangolari, ovali, poligonali, senza forma o di testo**.**

Si può accedere al confine del rettangolo della ROI corrente usando il metodo di *ImageProcessor*

- *java.awt.Rectangle getRoi()*

  Permette di recuperare solo i pixels che si trovano all'interno del rettangolo.

E' anche possibile settare la ROI del processore con

- *void setRoi(int x, int y, int rwidth, int rheight)*

  Questo metodo setta la ROI al rettangolo che inizia dalla posizione (x,y) con la larghezza e l'altezza specificati.

Altri metodi per lavorare con le ROI si trovano in ImagePlus:

- *void setRoi(int x, int y, int width, int height)*

  Seleziona un rettangolo che inizia nel punto (x,y) con la larghezza e l'altezza specificate.

- *void setRoi(java.awt.Rectangle r)*

  Seleziona un rettangolo.

- *void setRoi(Roi roi)*

  Effettua una selezione basata sull'oggetto ROI specificato.

- *Roi getRoi()*

Restituisce un oggetto ROI che rappresenta la selezione corrente.

Le classi che rappresentano i diversi tipi di ROI si trovano nel package *ij.gui*.

Queste classi sono:

- *FreehandROI*
- *OvalROI*
- *PolygonROI*
- *ROI*
- *TextROI*

4.6 Creare nuove immagini

Spesso ha più senso che un plugin non modifichi l'immagine originale, ma ne crei una nuova contenente le modifiche apportate. I metodi di *ImagePlus* per fare questo sono:

- *ImagePlus createImagePlus()*

  Restituisce una nuova *ImagePlus* con gli attributi di una *ImagePlus*, ma nessuna immagine.

Una funzione simile è fornita da *ImageProcessor*

- *ImageProcessor createProcessor(int width, int height)*

  Restituisce un nuovo processore bianco con la larghezza e l'altezza specificate e che può essere usato per creare una nuova *ImagePlus* usando il costruttore:

- *ImagePlus(java.lang.String title, ImageProcessor ip)*

La classe *NewImage* fornisce alcuni metodi statici molto utili per creare una nuova *ImagePlus* di un certo tipo.

- *static ImagePlus createByteImage(java.lang.String title, int width, int height, int slices, int fill)*

Crea una nuova immagine a colori o a toni di grigio (a 8 bit) con il titolo, la larghezza, l'altezza e il numero di slice specificati. *fill* è una costante che determina come deve essere riempita inizialmente l'immagine.

- *static ImagePlus createFloatImage(java.lang.String title, int width, int height, int slices, int fill)*

  Crea una nuova immagine a 32 bit floating point con il titolo, la larghezza, l'altezza e il numero di slice specificati. *fill* è una costante che determina come deve essere riempita inizialmente l'immagine.

- *static ImagePlus createRGBImage(java.lang.String title, int width, int height, int slices, int fill)*

  Crea una nuova immagine RGB con il titolo, la larghezza, l'altezza e il numero di slices specificati. *fill* è una costante che determina come deve essere riempita inizialmente l'immagine.

- *static ImagePlus createShortImage(java.lang.String title, int width, int height, int slices, int fill)*

  Crea una nuova immagine a toni di grigio (a 16 bit) con il titolo, la larghezza, l'altezza e il numero di slice specificati. *fill* è una costante che determina come deve essere riempita inizialmente l'immagine.

Nella classe *NewImage* esistono 5 possibili valori che l'argomento *fill* può assumere:

- FILL BLACK Riempie l'immagine di colore nero.
- FILL WHITE Riempie l'immagine di colore bianco.
- FILL RAMP Riempie l'immagine con una scala di grigi in orizzontale.

Ci sono 2 metodi per copiare i valori dei pixels fra 2 diversi *ImageProcessors*:

- *void copyBits(ImageProcessor ip, int xloc, int yloc, int mode)*

  Copia l'immagine rappresentata da ip in (xloc,yloc) usando la modalità specificata. Si tratta di una delle seguenti costanti definite nell'interfaccia Blitter:

- ✓ ADD destinazione = destinazione + sorgente
- ✓ AND destinazione = destinazione AND sorgente
- ✓ AVERAGE destinazione = (destinazione + sorgente)/2
- ✓ COPY destinazione = sorgente
- ✓ COPY INVERTED destinazione = 255 - sorgente
- ✓ COPY TRANSPARENT si assume che i pixels bianchi siano transparenti.
- ✓ DIFFERENCE destinazione = | destinazione - sorgente |
- ✓ DIVIDE destinazione = destinazione / sorgente
- ✓ MAX destinazione = maximum(destinazione, sorgente)
- ✓ MIN destinazione = minimum(destinazione, sorgente)
- ✓ MULTIPLY destinazione = destinazione * sorgente
- ✓ OR destinazione = destinazione OR sorgente
- ✓ SUBTRACT destinazione = destinazione - sorgente
- ✓ XOR destinazione = destinazione XOR sorgente

- *void insert(ImageProcessor ip, int xloc, int yloc)*

  Inserisce in (*xloc, yloc*) l'immagine in contenuta in ip.

Se l'utente ha bisogno di un'immagine AWT, la si può ottenere usando il metodo di *ImageProcessor*

- *java.awt.Image createImage()*

La stessa funzione è fornita dal metodo di *ImagePlus*

- *java.awt.Image getImage()*

4.7 Visualizzare immagini

Per rendere visibili le modifiche apportate ai pixels dell'immagine ImageJ si usa una classe di *ImagePlus* chiamata *ImageWindow*. *ImagePlus* contiene tutto ciò che è necessario per aggiornare o mostrare un'immagine nuova:

- *void draw()*

  Visualizza l'immagine.

- *void draw(int x, int y, int width, int height)*

Visualizza l'immagine e disegna la ROI usando il rettangolo selezionato.

- *void updateAndDraw()*

  Aggiorna l'immagine attraverso l'*ImageProcessor* associato e la visualizza.

- *void updateAndRepaintWindow()*

  Invoca *updateAndDraw* per aggiornare i dati e visualizzare l'immagine. Il metodo effettua anche un repaint della finestra in modo da forzare le informazioni visualizzate sull'immagine da aggiornare (dimensione, tipo e misura).

- *void show()*

  Apre una finestra per visualizzare l'immagine e cancellare la barra di stato.

- *void show(java.lang.String statusMessage)*

  Apre una finestra per visualizzare l'immagine e uno *statusMessage* nella barra di stato.

- *void hide()*

  Chiude la finestra (eventualmente aperta) e visualizza l'immagine.

4.8 Il PlugIn ColorInverter (esempio)

Supponiamo di voler modificare il plugin *Inverter_* in modo che riesca a trattare immagini RGB. In particolare, vogliamo che inverta i colori dei pixels della ROI e mostri il risultato in una nuova finestra.

Innanzitutto modifichiamo il nome della classe ricordando di modificare anche il nome del file:

```
import ij.*;
import ij.gui.*;
import ij.process.*;
import ij.plugin.filter.PlugInFilter;
import java.awt.*;
public class ColorInverter_ implements PlugInFilter {
...
```

Siccome vogliamo trattare solo con immagini RGB, non con stacks; vogliamo supportare le ROI non rettangolari e visualizzare i risultati in una nuova immagine senza modificare l'originale, cambiamo la capacità restituite dal metodo setup in DOES_RGB + SUPPORTS_MASKING + NO_CHANGES.

```
public int setup(String arg, ImagePlus imp) {
    if (arg.equals("about")) {
        showAbout();
        return DONE;
    }
    return DOES_RGB+SUPPORTS_MASKING+NO_CHANGES;
}
```

Il metodo run ha la seguente firma:

```
public void run(ImageProcessor ip) {
```

Prima di tutto conviene salvare le dimensioni e la ROI dell'immagine originale.

```
int w = ip.getWidth();
int h = ip.getHeight();
Rectangle roi = ip.getRoi();
```

Vogliamo salvare il risultato in una nuova immagine, quindi creiamo una nuova immagine RGB della stessa dimensione inizialmente bianca, con una slice e carichiamo il corrispondente processore.

```
ImagePlus inverted = NewImage.createRGBImage("Inverted image", w, h, 1,
                                                    NewImage.FILL_BLACK);
ImageProcessor inv_ip = inverted.getProcessor();
```

Dopo copiamo l'immagine dal processore originale a partire dalla posizione (0,0) della nuova immagine, usando la modalità COPY (che semplicemente sovrascrive i pixels del processore di destinazione) e prendiamo i dati della nuova immagine dall'array (che ancora è uguale al vecchio). Essendo un'immagine RGB, sarà un array di interi.

```
inv_ip.copyBits(ip,0,0,Blitter.COPY);
int[] pixels = (int[]) inv_ip.getPixels();
```

Adesso attraversiamo tutti i pixels della ROI con 2 cicli annidati. Il ciclo più esterno attraversa le righe, quello più interno scandisce tutte le colonne di ogni riga. L'offset dell'array rappresenta l'inizio della riga corrente (= alla larghezza dell'immagine × il numero di linee).

```
for (int i=roi.y; i<roi.y+roi.height; i++) {
    int offset =i*w;
    for (int j=roi.x; j<roi.x+roi.width; j++) {
```

Nel ciclo più interno, viene calcolata la posizione del corrente pixel (salvata in una variabile per essere riutilizzabile). Dopo ne viene calcolato il valore.

```
int pos = offset+j;
int c = pixels[pos];
```

Si estraggono le 3 componenti del colore

```
int r = (c&0xff0000)>>16;
int g = (c&0x00ff00)>>8;
int b = (c&0x0000ff);
```

Invertiamo ogni componente sottraendo il suo valore a 255. Dopo li ricompattiamo di nuovo in un unica variabile di tipo int.

```
        r=255-r;
        g=255-g;
        b=255-b;
        pixels[pos] = ((r & 0xff) << 16) + ((g & 0xff) << 8) + (b & 0xff);
    }
}
```

Adesso ci rimane solo di mostrare l'immagine invocando il metodo show per aprire una *ImageWindow* che la visualizzi.

Dopo invochiamo *updateAndDraw* per forzare l'array dei pixel ad essere letto e l'immagine ad essere aggiornata.

```
        inverted.show();
        inverted.updateAndDraw();
    }
}
```

4.9 Stacks

ImageJ supporta array espandibili di immagini chiamati stacks, che consistono in immagini della stessa dimensione (slice). In un *pluginfilter* è possibile accedere allo stack aperto, a partire dalla ImagePlus corrente, attraverso il metodo

- *ImageStack getStack()*

  ImagePlus offre anche un metodo per creare un nuovo stack:

- *ImageStack createEmptyStack()*

  Restituisce uno stack vuoto che ha la stessa larghezza, altezza e colore dell'immagine.

Alternativamente si può creare una ImageStack usando uno dei seguenti costruttori

- *ImageStack(int width, int height)*

  Crea un nuovo stack vuoto con la larghezza e l'altezza specificati.

- *ImageStack(int width, int height, java.awt.image.ColorModel cm)*

  Crea un nuovo stack vuoto con la larghezza, l'altezza e il ColorModel specificati.

Per settare lo stack appena creato come uno stack di immagini, si usa

- *void setStack(java.lang.String title, ImageStack stack)*

Il numero di slices di uno stack viene restituito dai metodi

- *int getSize()*

  della classe *ImageStack* oppure

- *int getStackSize()*

  della classe *ImagePlus*.

Lo slice di una *ImagePlus* visualizzato correntemente può essere recuperato e settato rispettivamente dai metodi

- *int getCurrentSlice()*
- *void setSlice(int index)*

Uno stack offre molti metodi per recuperare e settare le sue proprietà:

- *int getHeight()*

  Restituisce l'altezza dello stack.

- *int getWidth()*

  Restituisce la larghezza dello stack.

- *java.lang.Object getPixels(int n)*

  Restituisce l'array di pixels dello slice specificato, dove *n* è il valore compreso fra 1 e il numero totale di slices.

- *void setPixels(java.lang.Object pixels, int n)*

  Assegna l'array di pixels allo slice specificato, dove *n* è il valore compreso fra 1 e il numero totale di slices.

- *ImageProcessor getProcessor(int n)*

  Restituisce un *ImageProcessor* per lo slice specificato, dove *n* è il valore compreso fra 1 e il numero totale di slices.

- *java.lang.String getSliceLabel(int n)*

  Restituisce l'etichetta dello slice specificato, dove *n* è il valore compreso fra 1 e il numero totale di slices.

- *void setSliceLabel(java.lang.String label, int n)*

  Setta l'etichetta *label* come titolo dello slice specificato, dove *n* è il valore compreso fra 1 e il numero totale di slices.

- *java.awt.Rectangle getRoi()*

  Restituisce il rettangolo della ROI dello stack.

- *void setRoi(java.awt.Rectangle roi)*

  Setta la ROI dello stack nel rettangolo specificato.

Gli slices possono essere aggiunti o rimossi dall'*ImageStack* usando questi metodi:

- *void addSlice(java.lang.String sliceLabel, ImageProcessor ip)*

  Aggiunge l'immagine rappresentata da *ip* alla fine dello stack.

- *void addSlice(java.lang.String sliceLabel, ImageProcessor ip, int n)*

  Aggiunge l'immagine rappresentata da *ip* dopo l' n-esimo slice.

- *void addSlice(java.lang.String sliceLabel, java.lang.Object pixels)*

  Aggiunge un'immagine rappresentata dal suo array di pixels alla fine dello stack.

- *void deleteLastSlice()*

  Cancella l'ultimo slice dello stack.

- *void deleteSlice(int n)*

  Cancella lo slice specificato, dove n è il valore compreso fra 1 e il numero totale di slices.

4.10 Il PlugIn StackAverage (esempio)

Il seguente esempio mostra come trattare gli stacks. Calcola la somma dei valori dei pixels che si trovano nella stessa posizione in ogni slice dello stack ed aggiunge uno slice che mostra la somma calcolata alla fine dello stack.

Prima di tutto importiamo i packages necessari. Vogliamo lavorare solo sullo stack corrente, quindi dobbiamo implementare un *PlugInFilter*.

```
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;
public class StackAverage_ implements PlugInFilter {
```

Definiamo lo stack come una variabile istanza perchè ne avremo bisogno anche nel metodo *setup* e *run*.

```
protected ImageStack stack;
```

In questo metodo prendiamo lo stack dall'immagine corrente a restituiamo le capacità del plugin. Specifichiamo che tratta 8 immagini a toni di grigio a 8 bit e che necessita di uno stack in input.

```
public int setup(String arg, ImagePlus imp) {
```

```
        stack = imp.getStack();
        return DOES_8G + STACK_REQUIRED;
    }
```

Nel metodo *run* dichiariamo un array di byte che conterrà, di volta in volta, i pixels dello slice corrente. Dopo recuperiamo larghezza e altezza dello stack e calcoliamo la lunghezza dell'array dei pixels di ogni slice come il prodotto della larghezza per l'altezza. *sum* è l'array che conterrà i valori dei pixels sommati.

```
    public void run(ImageProcessor ip) {

        byte[] pixels;
        int dimension = stack.getWidth()*stack.getHeight();
        int[] sum = new int[dimension];
```

Nel loop più esterno iteriamo per ogni slice dello stack e prendiamo l'array di pixels da ognuno di essi. Nel loop più interno attraversiamo tutti i pixels dell'array dello slice corrente e aggiungiamo il valore del pixel al pixel corrispondente nell'array della somma.

```
    for (int i=1;i<=stack.getSize();i++) {
        pixels = (byte[]) stack.getPixels(i);
        for (int j=0;j<dimension;j++) {
            sum[j]+=0xff & pixels[j];
        }
    }
```

L'immagine risultante è ancora un'immagine a toni di grigio a 8 bit, quindi creiamo un corrispondente array di *byte*. Dopo iteriamo per tutti i pixels dell'array della somma e dividiamo ciascuno di essi per il numero di slice in modo da ottenere valori nell'intervallo [0,255].

```
    byte[] average = new byte[dimension];
    for (int j=0;j<dimension;j++) {
        average[j] = (byte) ((sum[j]/stack.getSize()) & 0xff);
    }
```

Alla fine aggiungiamo una nuova slice allo stack attraverso l'array della somma e che chiamiamo "*Average*".

```
        stack.addSlice("Average",average);
    }
```

4.11 Riferimenti aggiuntivi

Windows

- *void setWindow(ImageWindow win)*

  Setta la finestra che visualizza l'immagine.

- *ImageWindow getWindow()*

  Restituisce la finestra usata per visualizzare l'immagine.

- *void mouseMoved(int x, int y)*

  Mostra le coordinate del cursore e il valore del pixel nella barra di stato.

Multithreading

- *boolean lock()*

  Blocca l'immagine in modo da non essere più accessibile da altri thread.

- *boolean lockSilently()*

  Simile a *lock*, ma non effettua nessun suono, né visualizza un messaggio di errore se l'operazione di bloccaggio dell'immagine fallisce.

- *void unlock()*

  Sblocca l'immagine.

Lookup Tables

- *LookUpTable createLut()*

  Crea la LUT basata sull'immagine.

Statistics

- *ij.process.ImageStatistics getStatistics()*

Restituisce un oggetto *ImageStatistics* object generato usando le opzioni standard di misura (area, media, modalità, minimo and massimo).

- *ij.process.ImageStatistics getStatistics(int mOptions)*

  Restituisce un oggetto *ImageStatistics* generato usando le opzioni di misura specificate.

- *ij.process.ImageStatistics getStatistics(intmOptions, int nBins)*

  Restituisce un oggetto *ImageStatistics* generato usando le opzioni di misura specificate e il numero di bin dell'istogramma.

Calibration

- *void setCalibration(ij.measure.Calibration cal)*

  Setta la calibrazione dell'immagine.

- *void setGlobalCalibration(ij.measure.Calibration global)*

  Setta la calibrazione dell'intero sistema.

- *ij.measure.Calibration getCalibration()*

  Restituisce la calibrazione dell'immagine.

Geometric Transforms

- *void flipHorizontal()*

  Sposta l'immagine orizzontalmente.

- *void flipVertical()*

  Sposta l'immagine verticalmente.

- *void rotate(double angle)*

  Ruota l'immagine dell'angolo specificato (in gradi) in senso orario.

- *void scale(double xScale, double yScale)*

  Scala l'immagine del fattore specificato.

- *ImageProcessor crop()*

  Taglia l'immagine lasciando solo il rettangolo della ROI corrente. Restituisce un nuovo processore che rappresenta l'immagine ritagliata.

- *ImageProcessor resize(int dstWidth, int dstHeight)*

  Reimposta le dimensioni dell'immagine. Restituisce un nuovo processore che rappresenta l'immagine con le nuove dimensioni.

- *ImageProcessor rotateLeft()*

  Ruota l'immagine di 90 gradi in senso anti-orario Restituisce un nuovo processore che rappresenta l'immagine ruotata.

- *ImageProcessor rotateRight()*

  Ruota l'immagine di 90 gradi in senso orario Restituisce un nuovo processore che rappresenta l'immagine ruotata.

- *void setInterpolate(boolean interpolate)*

  Settare *interpolate* a vero fa in modo che *scale*(), *resize*() and *rotate*() effettuino una interpolazione bilineare, altrimenti viene effettuata un'interpolazione al più vicino.

Filters

- *void convolve3x3(int[] kernel)*

  Effettua la convoluzione dell'immagine con la matrice di convoluzione 3×3 specificata.

- *void sharpen()*

  Effettua lo sharpening dell'immagine usando un kernel di convoluzione 3×3.

- *void smooth()*

  Rimpiazza ogni pixel con la media di ogni maschera 3×3.

- *void filter(int type)*

  Effettua un'operazione di filtraggio 3×3, l'argomento definisce il tipo di filtro.

- *void dilate()*

  Dilata l'immagine usando un filtro minimo 3×3.

- *void erode()*

  Effettua l'erosione dell'immagine usando un filtro massimo 3×3.

- *void findEdges()*

Trova i contorni dell'immagine attraverso l'operatore Sobel.

- *void medianFilter()*

  Applica un filtro mediano 3×3.

- *void gamma(double value)*

  Effettua la correzione gamma.

- *void invert()*

  Inverte l'immagine.

- *void add(int value)*

  Aggiunge l'argomento ad ogni pixel.

- *void multiply(double value)*

  Moltiplica l'argomento ad ogni pixel.

- *void and(int value)*

  Effettua l'AND binario fra l'argomento ed ogni pixel.

- *void or(int value)*

  Effettua l'OR binario fra l'argomento ed ogni pixel.

- *void xor(int value)*

  Effettua l'OR esclusivo binario fra l'argomento ed ogni pixel.

- *void log()*

  Calcola i valori dei pixels su scala logaritmica.

- *void noise(double range)*

  Aggiunge un rumore casuale (scelto all'interno di *range*) all'immagine.

Drawing

- *void setColor(java.awt.Color color)*

  Setta il colore del foreground. Ciò setterà il valore fill/draw di default con il valore che rappresenta il colore.

- *void setValue(double value)*

Setta il valore fill/draw di default.

- *void setLineWidth(int width)*

  Setta lo spessore della linea.

- *void moveTo(int x, int y)*

  Setta la corrente locazione per disegnare a *(x,y).*

- *void lineTo(int x2, int y2)*

  Disegna una linea dalla corrente locazione *(x2,y2).*

- *void drawPixel(int x, int y)*

  Setta il pixel nella posizione *(x,y)* con il colore corrente.

- *void drawDot(int xcenter, int ycenter)*

  Disegna un punto usando il corrente spessore della linea e colore.

- *void drawDot2(int x, int y)*

  Disegna un dot 2×2 del corrente colore.

- *void fill()*

  Riempie la corrente ROI con il colore corrente.

- *void fill(int[] mask)*

  Riempie i pixels che si trovano all'interno della ROI e parte della maschera (per esempio i pixels che hanno valore (= nero) nell'array della maschera).

- *void drawString(java.lang.String s)*

  Disegna la stringa s nella locazione corrente con il colore corrente.

- *int getStringWidth(java.lang.String s)*

  Restituisce la lunghezza della stringa specificata in pixels.

Colors

- *int getBestIndex(java.awt.Color c)*

  Restituisce l'indice della LUT che rappresenta meglio il colore specificato.

- *java.awt.image.ColorModel getColorModel()*

Restituisce il color model del processore.

- *void invertLut()*

  Inverte i valori nella LUT.

Minimum, Maximum and Threshold

- *double getMin()*

  Restituisce il più piccolo valore fra i pixels visualizzati.

- *double getMax()*

  Restituisce il più grande valore fra i pixels visualizzati.

- *void setMinAndMax(double min, double max)*

  Mappa i pixels nell'intervallo relativo al tipo di immagine.

- *void resetMinAndMax()*

  Per immagini di tipo short o float, ricalcola il valore minimo e massimo necessari per visualizzare bene l'immagine

- *void autoThreshold()*

  Calcola l'auto soglia di un'immagine e la applica.

- *double getMinThreshold()*

  Restituisce la soglia minima.

- *double getMaxThreshold()*

  Restituisce la soglia massima.

- *void setThreshold(double minThreshold, double maxThreshold, int lutUpdate)*

  Setta il livello di soglia minimo e massimo. Il terzo parametro specifica se la LUT potrà essere ricalcolata.

Histograms

- *int[] getHistogram()*

  Restituisce l'istogramma dell'immagine. Questo metodo restituirà un istogramma di luminosità per le immagini RGB e nessuno per le immagini floating point.

- *int getHistogramSize()*

  La dimensione dell'istogramma è 256 per le immagini a 8 bit e RGB e maxmin+1 per le immagini di tipo *int*.

Snapshots (Undo)

- *void snapshot()*

  Salva lo stato corrente del processore come un'istantanea.

- *java.lang.Object getPixelsCopy()*

  Restituisce un riferimento all'istantanea dell'immagine, per esempio l'array di pixel prima dell'ultima modifica.

- *void reset()*

  Resetta il processore allo stato salvato nell'istantanea.

- *void reset(int[] mask)*

  Resetta il processore allo stato salvato nell'istantanea, escluso i pixels che fanno parte della maschera.

Accessing Images

- *java.lang.Object[] getImageArray()*

  Restituisce lo stack come un array di oggetti *ImagePlus*.

Color

- *boolean isHSB()*

  Restituisce vero se si tratta di uno stack HSB a 3-slice.

- *boolean isRGB()*

  Restituisce vero se si tratta di uno stack RGB a 3-slice.

- *java.awt.image.ColorModel getColorModel()*

  Restutisce il color model dello stack.

- *void setColorModel(java.awt.image.ColorModel cm)*

  Assegna un nuovo color model allo stack.

5 Utility Methods e Conversione di immagini

L' API di ImageJ contiene una classe chiamata IJ che implementa alcuni metodi statici molto utili. Questi metodi sono usati anche nei plugin generati da "Plugins/Record".

5.1 Messaggi di errore

Spesso è necessario che un plugin mostri un messaggio (di errore o per informazioni). Nel primo caso si usa

- *static void error(java.lang.String msg)*

che mostra un messaggio nella casella di dialogo chiamata “Error”, nel secondo caso

- *static void showMessage(java.lang.String msg)*

che mostra un messaggio nella casella di dialogo chiamata “Message”.

E' possibile specificare il titolo della casella dei messaggi usando il metodo

- *static void showMessage(java.lang.String title, java.lang.String msg)*

Tutti questi metodi visualizzano massaggi che l'utente deve accettare. Se si vuole fare in modo che l'utente sia libero di scegliere di cancellare il plugin o di continuare, si può usare il metodo

- *static boolean showMessageWithCancel(java.lang.String title, java.lang.String msg)*

Questo metodo restituisce falso se l'utente clicca su cancel, vero altrimenti.

Ci sono anche latri messaggi predefiniti:

- *static void noImage()*

  Mostra il messaggio “no images are open”.

- *static void outOfMemory(java.lang.String name)*

  Mostra “out of memory” nella firnestra di ImageJ.

- *static boolean versionLessThan(java.lang.String version)*

Mostra un messagiio di errore e restituisce falso se la versione ImageJ è anteriore a quella specificata.

5.2 ImageJ Window, Status Bar e Progress Bar

La figura 15 mostra la finestra principale di ImageJ e le sue componenti

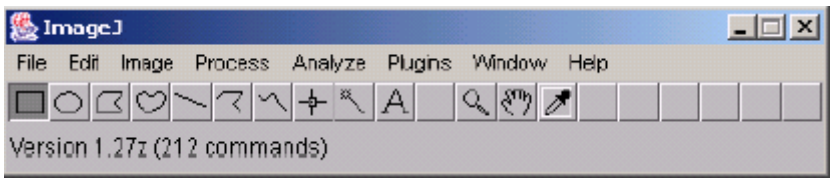


Figura 15 Finestra principale di ImageJ e sue componenti: Menu Bar, Tool Bar e Status Bar

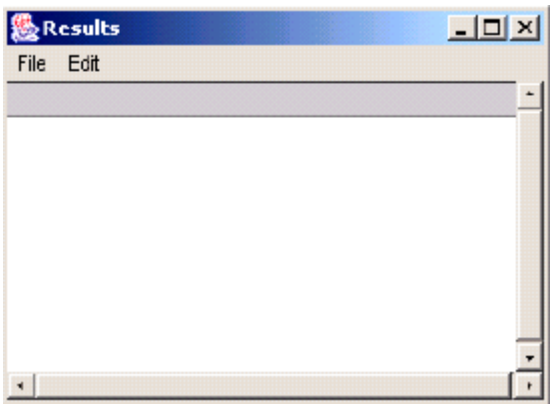


Figura 16 Finestra dei risultati di ImageJ; si apre automaticamente quando si scrive del testo

Per visualizzare una linea di testo nella finestra dei risultati (Figura 2) si usa

- *static void write(java.lang.String s)*

E' possibile usare il pannello di testo della finestra dei risultati come una tabella (per esempio, per visualizzare statistiche, misure, ecc.). In questo caso ImageJ permette di settare l'intestazione delle colonne usando

- *static void setColumnHeadings(java.lang.String headings)*

Si noti che questo metodo cancella l'intero pannello di testo.

Se si vogliono visualizzare dei numeri, si possono usare i seguenti metodi la formattazione dell'output:

- *static java.lang.String d2s(double n)*

  Converte un numero in una stringa formattata usando 2 cifre decimali a destra della virgola.

- *static java.lang.String d2s(double n, int precision)*

  Converte un numero in una stringa formattata arrotondata.

Il testo può anche essere visualizzato nella barra di stato (nella parte più bassa della finestra principale) usando il metodo

- *static void showStatus(java.lang.String s)*

Può essere utile visualizzare il tempo che è stato necessario per effettuare un'operazione:

- *static void showTime(ImagePlus imp, long start, java.lang.String str)*

Questo metodo visualizza l'argomento specificato, seguito dal tempo trascorso dal valore iniziale e il rate di pixels processati al secondo.

Il progresso dell'operazione corrente può essere visualizzato usando la barra di progresso di ImageJ.

- *static void showProgress(double progress)*

  aggiorna la posizione della barra di progresso al valore specififcato (nell'intervallo da 0.0 a 1.0).

5.3 Input dell'utente

Spesso in un plugin è necessario l'input dell'utente (per esempio un parametro). ImageJ offre 2 semplici metodi a questo scopo:

- *static double getNumber(java.lang.String prompt, double defaultNumber)*

  Permette all'utenete di inserire un numero nella casella di dialogo.

- *static java.lang.String getString(java.lang.String prompt, java.lang.String defaultString)*

  Permette all'utente di inserire una stringa nella finestra di dialogo.

5.4 Chiamare comandi del menu

Si può accedere ai comandi di tutto il menu da un plugin. Esistono 2 diversi metodi:

- *static void doCommand(java.lang.String command)*

  Comincia ad eseguire un comando di menu in un thread separato e ritorna immediatamente. Eseguire un comando in un thread separato significa che il programma procede subito invece di aspettare che il comando sia stato eseguito. Ciò ha il vantaggio che il programma non viene bloccato mentre il comando è in esecuzione.

- *static void run(java.lang.String command)*

  Esegue un comando di menu nel thread corrente, il programma continuerà dopo che il comando sarà terminato.

5.5 Chiamare gli altri plugins

Così come i comandi di menu, si possono eseguire anche altri plugins.

- *static java.lang.Object runPlugIn (java.lang.String className, java.lang.String arg)*

  Esegue il plugin specificato dal corrispondente nome della classe e lo inizializza con l'argomento specificato.

5.6 Il PlugIn MessageTest (esempio)

Vediamo adesso un plugin che usa alcuni dei metodi appena presentati.

Siccome non abbiamo bisogno di nessuna immagine, implementiamo il plugin come interfaccia. Dobbiamo importare il package *ij* perchè abbiamo bisogno della classe *IJ*.

```
import ij.*;
```

```
import ij.plugin.PlugIn;
public class Message_Test implements PlugIn {
```

Dobbiamo implementare solo il metodo *run*. Non è necessario nessun argomento. Prima di tutto visualizziamo una stringa nella barra di stato che informa l'utente che il plugin è iniziato. Dopo, impostiamo la barra di progresso allo 0% e mostriamo un messaggio di errore.

```
    public void run(String arg) {
        IJ.showStatus("Plugin Message Test started.");
        IJ.showProgress(0.0);
        IJ.error("I need user input!");
```

Vogliamo che l'utente inserisca una stringa di input e impostiamo la barra di progresso al 50%. In seguito scriviamo un messaggio nella finestra principale dicendo che sta iniziando il semplice plugin *RedAndBlue* (plugin che crea una nuova immagine con un gradiente rosso/blu e la visualizza) ed eseguiamo il plugin. Alla fine settiamo la barra di progresso al 100% e mostriamo un messaggio.

```
        String name = IJ.getString("Please enter your name: ","I.J. User");
        IJ.showProgress(0.5);
        IJ.write("Starting sample plugin RedAndBlue ... ");
        IJ.runPlugIn("RedAndBlue_","");
        IJ.showProgress(1.0);
        IJ.showMessage("Finished.",name+", thanks for running this plugin");
    }
}
```

5.7 Altre Utilities

Keyboard & Sound

- *static void beep()*

  Emette una beep audio.

- *static boolean altKeyDown()*

  Restituisce vero se il tasto Alt è premuto.

- *static boolean spaceBarDown()*

  Restituisce vero se la barra spaziatrice è premuta.

Accessing GUI Elements

- *static ImageJ getInstance()*

  Restituisce una riferimento al frame “ImageJ”.

- *static java.applet.Applet getApplet()*

  Restituisce l'applet che ha creato la corrente ImageJ o null se è eseguita come applicazione.

- *static TextPanel getTextPanel()*

  Restituisce un riferimento al pannello di testo nella finestra dei risultati di ImageJ.

Misc

- *static boolean isMacintosh()*

  Restituisce vero se la macchina sulla quale è eseguita ImageJ è un Macintosh.

- *static void wait(int msecs)*

  Ritarda *msecs* millisecondi.

- *static java.lang.String freeMemory()*

  Restituisce il totale di memoria libera in kByte sottoforma di stringa.

5.8 Conversione dei formati delle immagini

Il modo più semplice per convertire un'immagine da un tipo ad un altro è usare i metodi di conversione dell'*ImageProcessor* di un'immagine.

- *ImageProcessor convertToByte(boolean doScaling)*

Converte il processore in un *ByteProcessor* (8 bit a toni di grigio). Se doScaling è settato, i valori dei pixels vengono scalati nell'intervallo [0,255], altrimenti vengono arrotondati.

- *ImageProcessor convertToFloat()*

  Converte il processore in un *FloatProcessor* (3 bit a toni di grigio). Se la tabella di calibrazione è stata settata, viene usata la funzione di calibrazione.

- *ImageProcessor convertToRGB()*

  Converte il processore in un *ColorProcessor* (immagine RGB).

- *ImageProcessor convertToShort(boolean doScaling)*

  Converte il processor in un *ShortProcessor* (16 bit a toni di grigio). Se *doScaling* è settato, i valori dei pixels vengono scalati nell'intervallo [0,65.536], altrimenti vengono arrotondati.

La classe *ImageConverter* in *ij.process* fornisce molti metodi per la conversione dei tipi di immagine e metodi per convertire immagini RGB e HSB in stacks e viceversa. E' possibile accedervi direttamente o usando la classe *ij.plugin.Converter* come interfaccia. Un'istanza di un converter può essere costruita usando il metodo *Converter()* che lavora sull'immagine corrente.

L'unico metodo di questa classe è

- *public void convert(java.lang.String item)*

  dove *item* è una stringa che specifica il tipo di destinazione. Può avere uno dei seguenti valori: “8-bit”, “16-bit”, “32-bit”, “8-bit Color”, “RGB Color”, “RGB Stack” and “HSB Stack”.

Allo stesso modo, una istanza di *ImageConverter* può essere creata usando

- *ImageConverter(ImagePlus imp)*

I metodi per la conversione sono:

- *public void convertToGray8()*

  Converte l'*ImagePlus* in una scala di grigi a 8 bit.

- *public void convertToGray16()*

  Converte l'*ImagePlus* in una scala di grigi a 16 bit.

- *public void convertToGray32()*

  Converte l'*ImagePlus* in una scala di grigi a 32 bit.

- *public void convertToRGB()*

  Converte l'*ImagePlus* in RGB.

- *public void convertToRGBStack()*

  Converte un'immagine RGB in uno stack RGB (per esempio uno stack a 3 slices rappresentanti i canali rosso, verde e blu).

- *public void convertToHSB()*

  Converte un'immagine RGB in uno stack HSV (per esempio uno stack a 3 slices rappresentanti i canali tinta, saturazione e luminanza).

- *public void convertRGBStackToRGB()*

  Converte uno stack a 8 bit con 2 o 3 slice in RGB.

- *public void convertHSBToRGB()*

  Converte uno stack a 8 bit con 3-slice (tinta, saturazione e luminanza) in RGB.

- *public void convertRGBtoIndexedColor(int nColors)*

  Converte un'immagine RGB in un colore indicizzato a 8-bit. nColors deve essere maggiore di 1 e minore o uguale a 256.

Per scalare nell'intervallo [0,255] durante la conversione di immagini *short* o *float* in immagini byte o nell'intervallo [0,65535] durante la conversione di immagini *short* in immagini *float*, bisogna settare scaling a vero. Si può usare

- *public static void setDoScaling(boolean scaleConversions)*
- *public static boolean getDoScaling()*

  restituisce vero se *scaling* è abilitato.

ImageConverter non coverte stacks. A questo scopo esiste *StackConverter*, che può essere istanziato usando

- *StackConverter(ImagePlus img)*

che contiene i seguenti metodi:

- *void convertToGray16()*

  Converte lo stack nella scala di grigi a 16 bit.

- *void convertToGray32()*

  Converte lo stack nella scala di grigi a 32 bit (float).

- *void convertToGray8()*

  Converte lo stack nella scala di grigi a 8 bit.

- *void convertToRGB()*

  Converte lo stack in RGB.

6 Finestre

Di default i plugins lavorano con oggetti *ImagePlus* visualizzati in *ImageWindows*. Essi possono restituire informazioni nella finestra di ImageJ, ma non possono controllarla. Alcune volte, però, questo può essere necessario (soprattutto per caricare l'input dell'utente).

6.1 PlugInFrame

Un *PlugInFrame* è una sotto-classe di un frame AWT che implementa l'interfaccia *PlugIn*. Il plugin è implementato come una sotto-classe di *PlugInFrame*.

Esite un costruttore per costruire un *PlugInFrame*. Prende il titolo della finestra come argomento:

- *PlugInFrame(java.lang.String title)*

Dato che questa classe è un plugin, il metodo

- *void run(java.lang.String arg)*

dichiarato nell'interfaccia PlugIn interface, è implementato e può essere sovrascritto dal metodo run del plugin scritto dall'utente.

Naturalmente tutti i metodi dichiarati in *java.awt.Frame* e nelle sue superclassi possono essere sovrascritti. I dettagli si trovano nella documentazione API AWT.

6.2 GenericDialog

E' già stato spiegato il modo in cui caricare l'input dell'utente. Se si hanno bisogno di dati più complessi rispetto ad un numero o una stringa, *GenericDialog* aiuta a costruire un dialogo AWT detto "*modal*" (per esempio programmi che procedono solo dopo che l'utente ha risposto al dialogo). Un *GenericDialog* può essere costruito usando uno dei 2 costruttori:

- *GenericDialog(java.lang.String title)*

  Crea un nuovo *GenericDialog* con il titolo specificato.

- *GenericDialog(java.lang.String title, java.awt.Frame parent)*

  Crea un nuovo *GenericDialog* usando il titolo specificato e il frame parent (per esempio la classe plugin derivante dal PlugInFrame).

Il frame ImageJ può essere recuperato usando

- *IJ.getInstance().*

Il dialogo può essere visualizzato usando

- *void showDialog()*

GenericDialog offre diversi metodi per aggiungere controlli standard al dialogo:

- *void addCheckbox(java.lang.String label, boolean defaultValue)*

  Aggiunge una checkbox con l'etichetta e il valore di default specificati.

- *public void addCheckboxGroup(int rows, int columns, java.lang.String[] labels, boolean[] defaultValues)*

Aggiunge un gruppo di checkboxes usando un grid layout con il numero di righe e colonne specificati. Gli array contengono le etichette e i valori di default delle checkboxes.

- *void addChoice(java.lang.String label, java.lang.String[] items, java.lang.String defaultItem)*

  Adds una lista (popup menu) con l'etichetta, gli elementi e il valore di default specificati.

- *void addMessage(java.lang.String text)*

  Aggiunge un messaggio che consiste di una o più linee di testo.

- *void addNumericField(java.lang.String label, double defaultValue, int digits)*

  Aggiunge un campo numerico con l'etichetta, il valore di default e il numero di cifre specificati.

- *void addStringField(java.lang.String label, java.lang.String defaultText)*

  Aggiunge un campo di testo a 8 colonne con l'etichetta e il valore di default specificati.

- *void addStringField(java.lang.String label,java.lang.String defaultText, int columns)*

  Aggiunge un campo di testo con l'etichetta, il valore di default e il numero di colonne specificato.

- *void addTextAreas(java.lang.String text1, java.lang.String text2, int rows, int columns)*

  Aggiunge una o due aree di testo (fianco a fianco) con il contenuto iniziale e il numero di righe e colonne specificate. Se *text2* è null, la seconda area di testo non verrà visualizzata.

Dopo che l'utente ha chiuso la finestra di dialogo, si può accedere ai valori dei controlli attraverso i metodi elencati sotto. Esiste un metodo per ogni tipo di controllo. Se il dialogo contiene più di un controllo dello stesso tipo, ogni chiamata restituirà il valore del successivo controllo dello stesso tipo nell'ordine in cui sono stati aggiunti al dialogo.

- *boolean getNextBoolean()*

  Restituisce lo stato della successiva checkbox..

- *java.lang.String getNextChoice()*

  Restituisce l'elemento selezionato nella lista (popup menu).

- *int getNextChoiceIndex()*

  restituisce l'indice dell'elemento selezionato nella lista (popup menu).

- *double getNextNumber()*

  Restituisce il contenuto del successivo campo numerico.

- *java.lang.String getNextString()*

  Restituisce il contenuto del successivo campo di testo.

- *java.lang.String getNextText()*

  Restituisce il contenuto della successiva area di testo.

Il metodo

- *boolean wasCanceled()*

  restituisce vero se l'utente ha chiuso il dialogo usando il tasto “Cancel”, falso se l'utente ha cliccato su “OK”.

Se il dialogo contiene campi numerici, si usa il metodo

- *boolean invalidNumber()*

per controllare che i valori immessi nel campo sono numeri validi. Questo metodo restituisce vero se almeno un campo numerico non contiene un numero valido.

Inoltre, dato che *GenericDialog* estende *java.awt.Dialog*, anche tutti i metodi di *java.awt.Dialog* o di una sua superclasse possono essere usati.

6.3 Il PlugIn FrameDemo (esempio)

Questa demo mostra l'uso di *GenericDialog* e *PlugInFrame*. Viene creato un dialogo che lascia all'utente la scelta della larghezza e altezza del *PlugInFrame* che sarà visualizzato dopo la chiusura del dialogo.

Importiamo i packages *ij, ij.process, ij.gui* (dove si trova *GenericDialog*) e le classi *PlugInFrame* e *AWT label*.

```
import ij.*;
import ij.gui.*;
import ij.plugin.frame.PlugInFrame;
import java.awt.Label;
```

Il plugin che stiamo creando è una sotto-classe di *PlugInFrame* che implementa l'interfaccia *PlugIn*, quindi dobbiamo implementare qui un'interfaccia.

```
public class FrameDemo_ extends PlugInFrame {
```

Sovrascriviamo il costruttore di default della nuova classe. Se avessimo invocato il costruttore di default *PlugInFrame*() della superclasse, avremmo causato un errore in quanto non esiste. Quindi dobbiamo chiamare il costruttore della superclasse e specificare un titolo per il nuovo frame.

```
public FrameDemo_() {
        super("FrameDemo");
}
```

Nel metodo *run* creiamo un *GenericDialog* dal titolo “FrameDemo settings”. Dopo aggiungiamo 2 campi numerici a 3 cifre con valore di default uguale a 200.

```
public void run(String arg) {
        GenericDialog gd = new GenericDialog("FrameDemo settings");
        gd.addNumericField("Frame width:",200.0,3);
        gd.addNumericField("Frame height:",200.0,3);
```

Facciamo visualizzare il dialogo. Dato che è modale, il programma viene bloccato fino a quando l'utente non chiude il dialogo. Se l'utente clicca su “Cancel”, facciamo visualizzare un messaggio di errore e blocchiamo il metodo *run*.

```
gd.showDialog();
if (gd.wasCanceled()) {
        IJ.error("PlugIn canceled!");
```

```
            return;
        }
```

Recuperiamo il valore dei campi numerici con 2 chiamate a *getNextNumber*(). Settiamo la dimensione della finestra *FrameDemo* a queste dimensioni e aggiungiamo una etichetta AWT centrata con il testo “PlugInFrame demo”. Alla fine mostriamo il frame.

```
            this.setSize((int) gd.getNextNumber(),(int) gd.getNextNumber());
            this.add(new Label("PlugInFrame demo",Label.CENTER));
            this.show();
        }
}
```

6.4 ImageWindow

Una *ImageWindow* è un frame (derivato da *java.awt.Frame*) che visualizza una *ImagePlus*. Il frame contiene una *ImageCanvas* sulla quale viene dipinta l'immagine e una linea di testo (in alto). Ogni *ImagePlus* è associata ad una *ImageWindow*, che viene creata quando il metodo *show*() viene invocato per la prima volta. Una *ImageWindow* può essere anche creata usando uno dei costruttori:

- *ImageWindow(ImagePlus imp)*

  Crea una nuova *ImageWindow* che contiene l'immagine specificata.

- *ImageWindow(ImagePlus imp, ImageCanvas ic)*

  Crea una nuova *ImageWindow* che contiene l'immagine specificata e che verrà dipinta sul canvas specificato.

ImageJ mantiene la lista di tutte le finestre aperte usando la classe *WindowManager*. Quando il costruttore dell'*ImageWindow* viene invocato, la finestra viene aggiunta alla lista delle finestre aperte.

- *boolean close()*

Chiude la finestra e la rimuove dalla lista. Se l'immagine è stata cambiata, questo metodo chiede all'utente se vuole salvare l'immagine modificata. Se l'utente vuole salvarla, il metodo restituisce falso. Altrimenti restituisce vero e cancella l'immagine.

- *boolean isClosed()*

  Restituisce vero se *close*() è stato già chiamato, falso altrimenti.

Si può accedere all'immagine visualizzata in una *ImageWindow* e alla canvas sulla quale l'immagine è disegnata usando

- *ImagePlus getImagePlus()*
- *ImageCanvas getCanvas()*

ImageWindow fornisce metodi per tagliare, copiare e incollare comandi:

- *void copy(boolean cut)*

  Copia la ROI corrente (che deve essere rettangolare) nella clipboard. Se l'argomento cut è vero, la ROI viene tagliata la non copiata.

- *void paste()*

  Incolla il contenuto della clipboard nell'immagine corrente. Il contenuto della clipboard non deve essere più largo dell'immagine corrente e deve essere dello stesso tipo.

Così come l'*ImagePlus*, l'*ImageWindow* ha il metodo

- *void mouseMove(int x, int y)*

  Visualizza le coordinate specificate e il valore del pixel dell'immagine nella barra di stato della finestra di ImageJ.

ImageWindow ha anche una variabile booleana pubblica molto utile chiamata *running*, che viene settata a falso se l'utente clicca nella finestra, preme *escape* o chiude la finestra. Questa varabile può essere usata in un plugin, come nel frammento che segue, per dare all'utente la possibilità di interrompere il plugin.

```
...
win.running = true;
```

```
while (win.running) {
        // do computation
}
```

6.5 ImageCanvas

Ogni *ImageWindow* ha una *ImageCanvas* sulla quale l'immagine è disegnata. Si tratta di una sottoclasse di *java.awt.Canvas* che implementa una *MouseListener* e un *MouseMotionListener*. Può essere utile per trattare gli eventi e per ottenere informazioni su come l'immagine è visualizzata o può essere modificata.

Alcuni dei metodi utili di *ImageCanvas* sono:

- *java.awt.Point getCursorLoc()*

  Restituisce la locazione del corrente cursore.

- *double getMagnification()*

  Restituisce il corrente fattore di grandezza dell'immagine.

- *java.awt.Rectangle getSrcRect()*

  Restituisce il rettangolo dell'immagine della grandezza corrente.

- *int offScreenX(int x)*

  Converte uno screen a coordinate x in un offscreen a coordinate x.

- *int offScreenY(int y)*

  Converte uno screen a coordinate y in un offscreen a coordinate y.

- *int screenX(int x)*

  Converte uno screen a coordinate x in uno screen a coordinate x.

- *int screenY(int y)*

  Converte uno screen a coordinate y in un screen a coordinate y.

- *void setCursor(int x, int y)*

Setta il cursore con la posizione x e y specificati.

- *void setImageUpdated()*

  ImagePlus.updateAndDraw chiama questo metodo per trovare il metodo che disegna in modo da aggiornare l'immagine attraverso l'*ImageProcessor*.

- *void setMagnification(double magnification)*

  Setta un nuovo ordine di grandezza per le immagini.

- *void zoomIn(int x, int y)*

  Ingrandisce rendendo la finestra più grande.

- *void zoomOut(int x, int y)*

  Rimpicciolisce rendendo *srcRect* più grande.

6.6 Sottoclassi di ImageWindow

Uno *StackWindow* è un frame per visualizzare gli *ImageStacks*. E' derivata da *ImageWindow* ed ha una scrollbar per navigare all'interno dello stack.

- *void showSlice(int index)*

  Visualizza lo slice specificato e aggiorna la scrollbar dello stack.

- *void updateSliceSelector()*

  Aggiorna la scrollbar dello stack.

HistogramWindow è una sotto-classe di *ImageWindow* progettata per visualizzare istogrammi. Esistono 2 costruttori:

- *HistogramWindow(ImagePlus imp)*

  Visualizza l'istogramma (256 bins) dell'immagine specificata. La finestra ha il titolo “Histogram”.

- *HistogramWindow(java.lang.String title, ImagePlus imp, int bins)*

Visualizza l'istogramma dell'immagine usando il titolo e il numero di bins specificati.

- *void showHistogram(ImagePlus imp, int bins)*

  Visualizza l'istogramma dell'immagine usando il numero di bins specificati nell'*HistogramWindow*.

PlotWindow è una sottoclasse di *ImageWindow* progettata per visualizzare plot nel piano (x, y)

- *PlotWindow(java.lang.String title, java.lang.String xLabel, java.lang.String yLabel, float[] xValues, float[] yValues)*

  Costruisce una nuova finestra di plot con il titolo e le etichette per gli assi x e y. Dopo aggiunge punti con le coordinate (x,y).

- *void addLabel(double x, double y, java.lang.String label)*

  Aggiunge una nuova etichetta con il testo specificato alla posizione (x,y).

- *void addPoints(float[] x, float[] y, int shape)*
- *void addPoints(double[] x, double[] y, int shape)*

  Questi 2 metodi aggiungono punti con le coordinate (x, y) specificate al plot. Il numero di punti è dato dalla lunghezza dell'array. L'argomento *shape* determina la forma di un punto. Fino ad ora sono supportati solo i cerchi, specificati passando la costante PlotWindow.CIRCLE.

- *void setLimits(double xMin, double xMax, double yMin, double yMax)*

  Setta i limiti del piano del plot.

6.7 Gestione degli eventi (Esempio)

ImageWindow e *ImageCanvas* sono derivati dalla classi AWT *Frame* e *Canvas* e quindi supportano la gestione degli eventi. Ciò è utile soprattutto per caricare l'input dell'utente attraverso eventi del mouse e della tastiera.

La gestione degli eventi nelle AWT Java è basata su interfacce chiamati *listeners*. Esiste un'interfaccia *listener* per ogni tipo di evento. Una classe che implementa un'interfaccia *listener* è capace di trattare ogni tipo di evento. La classe può essere aggiunta alla lista di componenti di listeners ed essere notificata quando si verifica l'evento che gestisce.

Un plugin che deve gestire un certo tipo di evento deve implementare l'interfaccia appropriata. Inoltre deve poter accedere alla finestra dell'immagine e al canvas sul quale l'immagine è dipinta. Quindi deve essere aggiunta a questi componenti come un *listener*.

Per esempio, vogliamo scrivere un plugin che gestisce il click del mouse sull'immagine aperta. Le interfacce *listener* sono definite in *java.awt.event*, quindi dobbiamo importare questo package.

```
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*; import ij.gui.*;
import java.awt.event.*;
```

Dobbiamo accedere all'immagine e al *canvas* in più di un metodo, quindi dobbiamo dichiararli come variabili istanza:

```
ImagePlus img;
ImageCanvas canvas;
```

Il plugin deve implementare l'interfaccia appropriata:

```
public class Mouse_Listener implements PlugInFilter, MouseListener {
...
```

Nel metodo *setup* dobbiamo accedere all'*ImagePlus*, quindi dobbiamo salvarla come variabile istanza. Dobbiamo anche settare le capacità del plugin

```
public int setup(String arg, ImagePlus img) {
        this.img = img;
        return DOES_ALL+NO_CHANGES;
}
```

Nel metodo *run* recuperiamo l'*ImageWindow* che visualizza l'immagine e il *canvas* sul quale è stata disegnata. Vogliamo che il plugin venga notificato quando l'utente clicca sul *canvas*, quindi aggiungiamo il plugin al *MouseListeners* del *canvas*.

```
public void run(ImageProcessor ip) {
        ImageWindow win = img.getWindow();
        canvas = win.getCanvas();
        canvas.addMouseListener(this);
}
```

Per implementare l'interfaccia dobbiamo implementare i 5 metodi che dichiara. Vogliamo solo che gestisca il click del mouse, quindi lasciamo tutti gli altri vuoti. Prendiamo le coordinate del punto dove si è cliccato col mouse dall'oggetto evento che è passato al metodo. L'immagine deve essere scalata nella finestra, quindi usiamo i metodi di *ImageCanvas offScreenX*() e *offScreenY*() per recuperare le coordinate esatte.

```
public void mouseClicked(MouseEvent e) {
        int x = e.getX();
        int y = e.getY();
        int offscreenX = canvas.offScreenX(x);
        int offscreenY = canvas.offScreenY(y);
        IJ.write("mousePressed: "+offscreenX+","+offscreenY);
}
public void mousePressed(MouseEvent e) {}
public void mouseReleased(MouseEvent e) {}
public void mouseEntered(MouseEvent e) {}
public void mouseExited(MouseEvent e) {}
```

Un mouse *listener* più avanzato (che evita di assegnare il *listener* alla stessa immagine 2 volte) e un esempio simile che gestisce gli eventi della tastiera si trova nella pagina dei plugins di ImageJ.

Così come i *mouse listeners* e i *key listeners*, un plugin può implementare ogni *listener* di eventi, per esempio un *mouse motion listener*. Per aggiungere un *mouse motion listener*, devono essere apportate le seguenti modifiche al *mouse listener*.

La classe deve implementare l'interfaccia listener di eventi:

```
public class Mouse_Listener implements PlugInFilter, MouseListener,
                                                    MouseMotionListener {
```

Nel metodo *setup*, aggiungiamo il plugin come *listener* al *canvas* dell'immagine.

```
canvas.addMouseMotionListener(this);
```

Naturalmente dobbiamo implementare il metodo definito nell'interfaccia:

```
public void mouseDragged(MouseEvent e) {
    IJ.write("mouse dragged: "+e.getX()+","+e.getY());
}
public void mouseMoved(MouseEvent e) {
    IJ.write("mouse moved: "+e.getX()+","+e.getY());
}
```

7 Importazione/Esportazione di dati

Grazie alla sua capacità di gestire gli stacks di immagini, ImageJ può essere usata anche per processare movies. E' possibile importare ed esportare in formati di movies comuni. I plugin che effettuano queste operazioni sono disponibili nella pagina dei plugins di ImageJ. Sono basati sul QuickTime dell'Apple per la libreria Java, quindi per usarli si deve prima istallare QuickTime per Java.

References

[1] Documentazione ufficiale di ImageJ - http://rsb.info.nih.gov/ij/docs/index.html

[2] Werner Bailer, “Writing ImageJ Plugins – A Tutorial”, Version 1.6, 26 June 2003 - http://mtd.fh-hagenberg.at/depot/imaging/imagej/ijtutorial.pdf

[3] API JAVA di ImageJ - http://rsb.info.nih.gov/ij/developer/api/index.html