# Elaborazione di immagini

Informazioni e appunti sul software utilizzato

# Software Java per l’Image Processing ImageJ

## Introduzione

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

Può sfruttare il multitithreading ovvero è possibile effettuare delle operazioni in parallelo con altre.

ImageJ offre la possibilità di calcolare l’area e le statistiche sui valori dei pixel relativamente a delle regioni.

Si possono caricare dei file immagini e analizzarle o trattarle. Il software viene esteso attraverso i plugin.

## Plugin

Concettualmente, un plugin è una nuova funzionalità aggiunta a ImageJ. Un plugin si può sviluppare in:

* Macro
* Java

Nel nostro caso svilupperemo in Java. Quando il programma in esecuzione, le opzioni su come completare un’operazione richiesta saranno determinate da quali plugin sono disponibili in quel momento.

Un plugin, tecnicamente, è una classe java annotata con l’annotazione @Plugin. Le classi annotate in questo modo vengono quindi automaticamente rilevate e indicizzate in fase di esecuzione. Quando l’applicazione viene avviata da un utente, invece che in fase di compilazione.

### Tipologie di plugin

Non c’è limite al numero di plugin che possono essere scoperti in fase di rutime. Per consentire un recupero efficiente dei plugin, ogni classe viene annotata con un tipo specifico, in genere un’interfaccia Java, tramite il quale il plugin viene segnato.

#### Esempi

@Plugin(type=Service.class)

public class MyService implements Service { }

@Plugin(type=SpecialService.class)

public class SpecialService implements Service { }

### Proprietà dei plugin

I plugin vengono considerati in base alla proprietà dell’annotazione @Plugin della classe. Si può impostare tramite la classe Priority.

#### Esempi

@Plugin(priority=Priority.HIGH\_PRIORITY)

public class MyService implements Service { }

@Plugin(priority=224)

public class SpecialService implements Service { }

### Un semplice plugin

Consideriamo un plugin Inverte\_ che inverte un’immagine a toni di grigio a 8 bit.

1. All’inizio vengono importati i package necessari:
   1. Ij per le classi di base ImageJ
   2. Ij.process per i processori immagine
   3. Ij.plugin.filter.PlugInFilter per l’interfaccia
2. Il plugin ha bisogno di un’immagine in ingresso -> PlugInFilter
3. Nel caso 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.
4. Il run implementa la funzione principale del plugin. Viene caricato il processore dell’immagine e ne ricava un array di pixel, 256. Dall’array che è unidirezionale viene calcolata la larghezza dell’immagine e i limiti del rettangolo della ROI.

analisi documentazione

# Plugin Landini

## Setup

Analisi dell’immagine: verifica se è possibile analizzarla o meno

Per trattarla ottiene il suo numero ID, getID e la immette in un intero.

Attua la calibrazione; gli oggetti di calibrazione contengono i dati di calibrazione spaziale e di densità dell'immagine.

Se non è quadrata non continua.

Recupera delle statistiche globali dell’immagine con ImageStatistcs (classe) e getStatistics. I metodi utilizzati sono sempre invocati da imp ovvero dalla classe ImagePlus.

Verifica se l’immagine è binaria ovvero se è dotata solo da due tonalità. Questo lo verifica attraverso stats variabile che contiene le statistiche (istogrammi). Attraverso un if se i pixel della prima posizione sommati a quelli dell’ultima (0 e 255) sono uguali a quello che restituisce pixelCount stampa errore e return Done.

Se questo non è vero il controllo è passato.

Procede salvando su istk un immagine stack attraverso il metodo getStack(). Immagine Stack – imageStack -> altro formato di ImageJ.

Crea una lista di int, wList il metodo di WindowManager “getIDList”. WindowManager è necessaria per usare la finestra di ImageJ. Questo metodo restituisce una lista di ID di apertura di immagini.

Crea un altro vettore di String di lunghezza del precedente vettore appena creato. Nella prima posizione mette il valore stringa “none”.

Controlla per ogni elemento della wList se è diversa di thisisimage che ha l’ID della imp, l’ImmaginePlus. Se questo è vero incrementa j e imp prende l’immagine presente nella posizione di wList di quel momento (grazie l’ID) e il titolo (gestito l’indice da J) viene aggiunto dal nuovo imp.

Messaggi vari ed eventuali.

Viene creata una finestra di GenericDialog -> dialog box. Essa si chiama come il nome della classe e come frame l’istanza di IJ.

Vengono aggiunti svariati messaggi.

Vengono anche visualizzati.

Se essa viene cancellata (si presume chiusa) si passa al return done.

Setta svariati booleani attraverso il metodo della finestra di dialogo getNextBoolean; Returns the state of the next checkbox.

Si presume che si possa fare qualcosa con questa finestra tipo interagirci. Quindi quello fatto fino ad ora è creare una finestra di dialogo attraverso il setup.

## Run

Fino ad adesso si è denotato che il metodo run è il cuore di questa tipologia di plugin. Prende in ingresso un ImageProcessor ip. Viene incrementato stk, dichiarato protected ad inizio classe.

Vengono dichiarati e inizializzati due interi:

1. Xe che è la larghezza
2. Ye che è l’altezza di ip

Nella stringa slabel viene posta “l’etichetta della fetta”. Se questa stringa ha una dimensione maggiore di 0, viene le stringhe \\n

Viene dichiarata una variabile X che prende xe -1

Idem per Y.

XY è la moltiplicazione tra le due.

Si dichiarano diversi vettori. Hanno dei valori specifici. E anche un vettore di risultati inizializzato a zero.

int x=0, y=0, x1, y1, q, uy, dy, rx, lx, pixs, fig, x2, y2, py=0, px=0, nn, newcol, f1, fx1=-1, fx2=-1, fy1=-1, fy2=-1, vl, pb1, pb2;

        long m10, m01;

        int pxgrey=0, pxred=0,  pxgreen=0, pxblue=0;

        double nz, narea, mcx, mcy, feret, tferet, rads=180/Math.PI, angle=0.0, breadth1, breadth2, minr, maxr;

vengono inizializzate tutte queste.

int pxgrey=0, pxred=0,  pxgreen=0, pxblue=0;

queste sembrerebbero per i colori.

Ci sono alcune variabili che sembrano richiamare i features.

Narea -> area

MinR e maxR.

Viene creato un vettore a testa per la x e la y e per i colori, r g b.

Sembra che cerca di sistemare l’immagine.

Area -> area del parassita numero dei pixels

Perimetro -> perimetro del parassita

Convex hull nuvola di punti è la minima forma non convessa che racchiude tutti i punti.

CHull -> scafo convesso di poligono convesso calcolato da centri di pixel (??)

Convex Hull da capire

Lunghezza del massimo diametro

Larghezza dell’asse perpendicolare del minimo diametro

MinR -> raggio del cerchio iscritto centrato nel mezzo della massa

Idem per max racchiuso del cerchio

## Run

Il metodo all’interno del plugin, (classe java), prende in ingresso un ImageProcessor. E’ una classe astratta per processare i quattro tipi di dati principali: byte, short, float e RGB. Un ImageProcessor contiene i pixel dell’immagine 2D e alcuni metodi basi per trattarli.

### Fase uno:

Viene incrementato un contatore, stk, precedentemente inizializzato e creato come variabile globale. Sicuramente è un controllo. Viene restituita la larghezza e l’altezza dell’immagine e posta rispettivamente in xe e ye, attraverso i metodi getWidth e getHeight. Viene dichiarata e inizializzata una stringa. Viene creata a partire da istk che è un ImageStack con il metodo getSliceLabel. La classe rappresenta un espandibile array di immagini. Con il metodo getSliceLabel viene restituito un label di uno specifico slice. Infatti viene usato come indice il contatore nominato poco fa. Se la dimensione della stringa è maggiore di 0 vengono sostituiti i simboli di chiusura “\n”.

Vengono definiti gli assi: X, Y e XY. Sia X e Y sono uguali rispettivamente alla larghezza e alla altezza, meno 1. XY è il prodotto dei due assi.

Vengono definiti dei nuovi array: xd, yd, g, z, e chresults. Vengono inizializzati con cifre precise. Chresults ospiretà i risultati:

1. Convenx hull
2. Cx
3. Cy
4. Radius of minimal bouding circle
5. Convex area
6. Feret
7. Fx1
8. Fy1
9. Fy2

Vengono dichiarate e inizializzate alcune nuove variabili: x e y a zero, x1, y1,, q, uy, dy, rx, lx, pixs, fig, x2, y2, py e px a zero, nn, newcol, f1, fx1 a meno 1, idem per fx2, fy1, fy2 e infine vl, pb1, pb2. Sono tutte degli int.

Vengono dichiarate le variabili long m10 e m01.

Vengono dichiarate delle variabili double nz, narea, mcx, mcy, feret, tferet, rads che è uguale a 180 diviso pigreco, angle inizializzato a 0., breadth1, breadth2, minr e maxr.

Tra queste variabili volte alla morphological features in particolare abbiamo:

* Narea: l’area dell’oggetto che si sta analizzando ovvero il numero dei pixel da cui è composta.
* Feret e tferet: il diametro massimo dell’immagine (altezza).
* breadth1 e breadth2: per il calcolo del diametro minimo dell’immagine.
* MinR e MaxR: raggio della circonferenza inscritta e “esterna” rispetto al centro di massa.

Vengono dichiarati tre vettori float parsR, parsG, parsB riferendosi al colore.

Vengono dichiarati dei vettori della tipologia Vector; vx, vy, vr, vg, vb. I primi due si riferiscono alle dimensioni x e y, gli ultimi tre ai colori.

Se l’ImmageProcessor ip2 è diverso da null a imp2 viene assegnato setSlice(stk). Anticipazione de lo stack è reindirizzato.

Viene creato un nuovo FloodFiller ff con in ingresso ip ovvero l’ImageProcessor.

Se non ci sono particelle bianche vengono invertite attraverso un if che controlla doIwhite se è uguale a falso. Se questo è vero viene impostato doppio for che analizza tutti i pixel (con xe e ye) e ad ogni pixel date quelle coordinate imposta putPixel. Quest’ultimo metodo di ImageProcessor prende in ingresso x e y e un valore intero. Memorizza uno specifico valore. In questo caso il valore intero è dato da 255 – getPixel date coordinate. Sta appunto invertendo il colore.

Con un secondo if in cui si analizza doIdeleteborders si entra nell’ImageProcessor e si settano i colori a zero attraverso setColor; si sta impostando il colore di riempimento a zero. Sempre all’interno dello stesso if si entra nei “bordi”; rima in ye e si procede con due if. Si controllano i due bordi “verticali” se sono uguali a 255 viene aggiornato ff.

|  |
| --- |
| FloodFiller: questa classe, dedita al riempimento, è usata nella funzione macro floodFill() e dall’analizzatore di particelle L’algoritmo determina un’area connessa a un determinato nodo in una matrice multidimensionale. |

Ff è appunto un FloodFiller. Viene usato il metodo filler8, per il primo bordo (infatti zero per x e per y la y stessa). Con questo metodo si usa il valore di riempimento corrente definito da ImageProcessor.SetValue(). Si procede allo stesso modo per il secondo bordo ma usando la X per la x.

Stesso ragionamento si fa per l’asse x (invece di usare ye si imposta per xe).

Si risetta il colore a 254 con ip.setColor(254).

Si imposta fig a zero.

### Fase due

Si scansione una prima volta i pixel.

Si procede con un for riguardante ye (asse y). Per ogni y, se stk è uguale ad 1 e y%25 è uguale a 0, si richiama uno showProcess con IJ passandogli le coordinate y e ye-1.

Che poi in realtà è il contrario, avanzando con la y si sta analizzando l’asse x e avanzando con la y si analizza l’asse x.

|  |  |  |
| --- | --- | --- |
| Per ogni pixel-all’interno dei due for Quando si tratta con il secondo for, innestato al primo, se nella posizione corrente si ha un valore uguale a 255 si procede. In ogni caso si scrolla pixel per pixel, usando la x e y e come valori massimi xe e ye calcolati inizialmente.  Vengono puliti i vari vettori e ai vettori vx e vy vengono aggiunti i valori di x e y.  Vengono aggiornati i valori di x1 che prende x, y1 che prende y, uy prende y e dy prende zero insieme a rx. Lx prende xe.  Nz, narea e pixs prendono zero. Primo if ip.getPixel(x+1,y)+ip.getPixel(x,y+1)+ip.getPixel(x-1,y+1)+ip.getPixel(x+1,y+1)==0  Se la somma dei pixel che contornano il pixel x y è uguale a zero si controlla con un ulteriore if se il booleano doIminmax && mi>0 sia vera. Se è vero si procede con putPixel x y e valore zero. Else Altrimenti, si procede con putPixel con x e y e valore (fig % 251)+1. Fig è un intero ed è stato settato a zero prima del primo ciclo for. Verrà poi incrementato di uno. Dal commento si evince “nuovo colore se etichettatura” per quanto riguarda il metodo appena nominato.  Si controlla se ip2 sia diverso da null, se è vero pxgrey prende il pixel ip2.getPixel(x,y) ovvero del pixel analizzato ma dato da ip2 che è l’ImageProcessor “globale”. (Precedentemente viene immesso attraverso WindowManager con getProcessor  imp2=WindowManager.getImage(redirected);              ip2 = imp2.getProcessor();  )  Se i bits sono 24 vengono attribuiti a pxred, pxgreen e pxblue dei valori specifici;  if (ip2!=null){                                  pxgrey = ip2.getPixel(x,y);// look at px,py of redirected                                  if (bits==24){                                      pxred = (pxgrey & 0xff0000)>>16;                                      pxgreen = (pxgrey & 0x00ff00)>>8;                                      pxblue = (pxgrey & 0x0000ff);                                  }                              }  Pixs prende il valore 1 e fig viene incrementato. Rt, resultTable viene inizializzato. Dopo aver immesso i valori base viene controllato “doImorpho” e si procede con il resto dell’inizializzazione: infatti i valori prendono generalmente x o y, 0, -1.  //333  Viene controllato se ip2 sia diverso da null e se i bits sono uguali a 24. Se questo è vero vengono addatti nuovi valori, per lo più di colore. Altrimenti se bits è uguale a 8 o a 16 vengono setati i valori di Gr. Vengono addati i risultati. Else del primo if Il commento sostiene “go arround the edge ”, gira attorno al bordo. Si deduce che al primo if si lavora sul singolo if. Nell’else si lavora al contorno (che non è uguale a 0).  X1 e y1 prendono rispettivamente x y, coordinate analizzate. Si procede con putPixel imponendo 254 nelle date coordinate. Q viene impostato a zero e si procede con un while sempre vero.   While(true) Il primo if verifica se nelle coordinate date da x1 sommata a xd (posizione q) { int [] xd = {1,1,1,0,-1,-1,-1,0};} e stessa cosa per y1 ma sommato a yd medesima posizione, il pixel sia maggiore di zero.  ip.getPixel(x1+xd[q],y1+yd[q])>0  All’interno di questo if, vengono utilizzate x2 e y2 e settate con i valori di x1 e y2. Infatti a x1 verrà sommato al valore di xd nella posizione q e stessa cosa con y1 ma con yd.  Viene calcolato narea. Narea è un double dunque viene castato a double. Si sommano y1 + y2 moltiplicati a x2 – x1 e divisi due.   |  | | --- | | Sembrerebbe che prende il seme per triangolo. Perché A=b\*h / 2 |   Successivamente in ip.viene settato 254 alla posizione y1 e x1.  Se x1 è maggiore di rx, quest’ultimo prenderà il valore di x1.  Se x1 è minore di lx quest’ultimo prenderà x1  Idem per dy e uy ma con y1.  Il perimetro nz è sommato ad z[q]  Inizialmente:  double [] z = {1.414213538169861, 1.0, 1.414213538169861, 1.0, 1.414213538169861, 1.0, 1.414213538169861, 1.0};  nz era precedentemente settato a zero  q prende g[q]  int [] g = {6,0,0,2,2,4,4,6};  per poter porre fine al while viene posto un check quasi alla fine.  Viene controllato se x1 ha lo stesso valore della x stessa cosa per y1 con y.  Se questo è vero si controlla se si è arrivato all’ultimo pixel con getPixel x-1 e y+1 sia uguale 255. Se è vero break.  Poca prima di chiudere si adda ai due vettori vx e vy integer x1 e y1. Prima di chiudere il while si chiude con un else il principale if facendo prendere a q= q+1 mod 8.  Usciti dal ciclo vl prende vx.size e convh viene eseguito con vx, vy, vl chresult. //calcolo del feret 435 diametro massimo // feret diameter here                          feret=chresults[5];                          fx1=(int)chresults[6];                          fy1=(int)chresults[7];                          fx2=(int)chresults[8];                          fy2=(int)chresults[9];  vengono immessi direttamente dalla chresults dei risultati a tali posizioni. Se fx2 e fx1 sono uguali l’angolo è uguale a 90.  Altrimenti viene calcolato con una formula da Math usando fy2 e fy1 e fx2 e fx1.  se è minore di zero allora viene sommato 180. //calcolo del breadth diametro minimo Vengono azzerati breadth1 e breadth2 a zero insieme a pb1 e pb2.  Viene effettuato un ciclo for:   |  | | --- | | ciclo for  f1 prende zero e cicla fino a che è minore di vl.  Tferet prende  tferet=((fy1-fy2)\*((Integer)(vx.get(f1))).intValue() +(fx2-fx1)\*((Integer)(vy.get(f1))).intValue()+(fx1\*fy2 -fx2\*fy1))/feret;  dopo questo calcolo se tferet è maggiore di breadth1 allora breadth1 prende tferet e pb1 prende f1.  Se tferet è minore di breadth2, stessa cosa di sopra ma con pb2 e breadth2 |  Scansione Commento:  //scan bounding box from top left to bottom right and {flood) with 254 if pixel is 255 and any neighbour pixel is 254 // scansione bounding box da in alto a sinistra in basso a destra e {flood) con 254 se pixel è 255 e qualsiasi pixel adiacente è 254  Si utilizzano due cicli for. Il primo scorre py il secondo px. A tali coordinate, si verifica se il controlla se almeno un contorno è uguale a 254. Se è vero ff.fill8(px, py).  Si esce dal ciclo for e si in newcol si aggiunge un nuovo colore. Si azzerano i valori di m10, m01 mcx e mcy. Nuovo test 490  Se doIminmax è vero la variabile nn viene settata a 253. Altrimenti viene settata a newCol, il nuovo colore. //momento calcolato Si creano due cicli sempre per scorrere l’immagine che in realtà è trattata come matricione. Vengono usati py e px per lo scorrimento. Py viene confrontato con dy e px con rx. (rx dovrebbe aver preso il valre di x1 al primo ciclo e dy y1). Py viene inizializzato a uy e px con lx.  All’interno del ciclo for viene fatto un controllo. Se il pixel ha valore 254 allora pixs viene incrementato, a m10 viene sommato il valore di px e m01 viene sommato il valore di py. A ip viene applicato putPixel con px e py e valore nn.  Se ip2 è diverso da null e bits è uguale a 24, vengono addati i valori ai vettori di colore tramite getPixel di px e py. Altrimenti se bits non è uguale a 24 a vr viene aggiunto il valore di pixel di ip2 con le coordinate px e py.  Uscita dal ciclo for.  Si verifica se ip2 è diverso da null, se è vero si controlla se bits è uguale 24. Se è vero a stats viene passato pixs vr, parsR e idem per G e B.  Altrimenti viene passato solo R, stats pixs vr parsR.  Mcx prende m10 fratto pixs e mcy idem ma con m10 (?).  //minimal maximal radii  532  Minr prende il massimo valore e maxr prende -1.  Si crea un ciclo for  for (f1=0;f1<vl;f1++){                              tferet=Math.pow(mcx-((Integer)(vx.get(f1))).intValue(),2) +Math.pow(mcy-((Integer)(vy.get(f1))).intValue(),2);                              if (tferet>maxr){                                  maxr=tferet;                              }                              if (tferet<minr){                                  minr=tferet;                              }                          }  A tferet viene applicata una formula particolare e i valori di maxr e minr aggiornato volta per volta.   minr=Math.sqrt(minr);   maxr=Math.sqrt(maxr);  rt. addValue("Pixels", pixs);  rt. addValue("MinR", (float)minr);  rt. addValue("MaxR", (float)maxr);  rt. addValue("Breadth",(float)(Math.abs(breadth1)+Math.abs(breadth2)));  rt. addValue("CountCorrect", (float)((double)XY/((X-(1+rx-lx))\*(Y-(1+dy-uy)))));  rt. addValue("AspRatio", (((Math.abs(breadth1)+Math.abs(breadth2))<0.01)?-1:(float)(feret/(double)(Math.abs(breadth1)+Math.abs(breadth2)))));  rt. addValue("Roundness", (float) (4.0\*narea/(Math.PI\*Math.pow(feret,2))));  rt. addValue("ArEquivD", (float) Math.sqrt((4.0/Math.PI)\*narea));  rt. addValue("PerEquivD", (float) (narea/Math.PI));  rt. addValue("EquivEllAr", (float)(Math.PI\*feret\*(Math.abs(breadth1)+Math.abs(breadth2))/4.0));  rt. addValue("Compactness", (float) (Math.sqrt((4.0/Math.PI)\*narea)/feret));  rt. addValue("Solidity", ((chresults[4]>0)?(float)(narea/chresults[4]):-1));  rt. addValue("Concavity", (float) (chresults[4]-narea));  rt. addValue("Convexity", (float) (chresults[0]/nz));  rt. addValue("Shape", ((narea>0)?(float) ((nz\*nz)/narea):-1));  rt. addValue("RFactor", (float) (chresults[0]/(feret\*Math.PI)));  rt. addValue("RFactor", (float) (chresults[0]/(feret\*Math.PI)));    rt. addValue("ModRatio", (float) ((2.0\*minr)/feret));    rt. addValue("Sphericity", (float) (minr/maxr));    rt. addValue("ArBBox", (float) (feret\*(Math.abs(breadth1)+Math.abs(breadth2))));                                      rt. addValue("Rectang",   (((Math.abs(breadth1)+Math.abs(breadth2))<0.01)?-1:(float)(narea/(feret\*(Math.abs(breadth1)+Math.abs(breadth2)))))); |

## Analisi Papers - Appunti

### A Review of Automatic Malaria Parasites Detection and Segmentation in Microscopic Images

Luís Rosado, José M. Correia da Costa, Dirk Elias1 and Jaime S. Cardoso

La malaria è una delle cause principali di morte in numerose zone. I gruppi colpiti maggiormente sono i bambini e le donne incinte. Nel 2012, sono state stimate 207 milioni casi di malaria con 627 000 di morti approssimativamente. Intorno al 80% dei casi, si verificano in Africa, a causa di una non possibilità di accedee ad una diagnosi spesso basata solo sui sintomi. Vi è un urgente bisogno di nuovi strumenti per cui è semplice e rapido fare una diagnosi della malaria soprattutto dove l’assistenza sanitaria è carente.

##### Metodi

Sono stati proposti varie metodologie di analizzi di immagini attraverso la detezione di immagini di sangue contenenti i parassiti della malaria. Questa nuova metodologia mira ad sostenere l’aumento dell’interessamento ad utilizzare strumenti a basso costo per facilitare la diagnosi rapida e facile della malattia.

La maggior parte del materiale utilizzato in questo lavoro è stati fatto con studio di dataset significatamente grandi per la valutazione degli approcci proposti.

#### Introduzione

[wikipedia]

La maggior parte delle infezioni malariche sono dovute a quattro specie di Plasmodium:

1. P.falciparum
2. P.vivax
3. P.ovale
4. P.malarie

Delle quattro la P.falciparum è potenzialmente mortale.

L’ interesse per i CAD, ovvero computer che si occupano di diagnosi. Riducono la dipendenza del microscopio manuale e riduce tempo e costi.

#### Meccanismo di instaurazione

Nella maggior parte dei casi vi è un equilibro tra infezione risposta immunitaria ma nel caso di infezione da parte di P.falciparum l’equilibrio non è stabile. La malattia può precipitare in qualsiasi momento.

#### Ciclo vitale (tre fasi)

La forma infettante del plasmodio viene trasferita dalle zanzare femmine appartenenti al genere Anophels. Dopo una breve permanenza nel ciclo ematico, entro 45 minuti dal pasto ematico, gli sporozoiti invadono gli epatociti, cellule del fegato per il quale hanno un tropismo elevato.

Vi è una prima fase di moltiplicazione asessuata. Dopo questa fase si riversano in circolo migliaia di merozoiti mononucleati che vanno a infettare gli eritrociti.

Comincia un nuovo ciclo di riproduzione asessuata.

Il merozoite, infatti, si muta in

* Trofozoite, fase vegetativa. Si parla di
  + Immaturo con una forma ad anello
  + Maturo con citoplasma più allargato e accumulo di emozoina.
* La fase schizont inizia quando il trofozoita ha raggiunto la sua piena capacità e il parassita inizia a dividere in cellule figlie chiamate merozoiti. Molte altre divisioni della cromatina seguono, fino a quando ci sono molti corpi di cromatina ciascuno con il suo citoplasma. Il numero di cromatina e le divisioni di merozoite aiutano a identificare la specie. Questi nuovi parassiti chiaramente delineati sono ora pronti a lasciare la cellula ospite per invadere nuovi globuli rossi.
* I gametociti sono rotondi o a forma di “banana”. In questa fase il parassita prende colore e aiuta a identificare il sesso del parassita, grazie a “film sottili” (??)

La malaria è causata da parassiti che si instaurano nel sangue. Per poterli visualizzare, solitamente si utilizza l’analisi attraverso un microscopio. Quando il microscopista vede i parassiti colorati, la malattia è confermata e si può visualizzare la densità dell’infezione.

Ci sono varie specie di parassiti che causano la malaria:

1. P.vivax: è la specie più comune nelle parte relative ai tropici. Sono i più “colpevoli” per assenteismo da lavoro e scuola.
2. P.ovale: è considerato una rara specie. Si trova specialmente nel West Africa e altre parti dell’Africa. È morfologicamente simile al P.vivax.
3. P.malarie; si trova in tutto il mondo a causa della infezione cronica che in alcuni casi può durate tutta la vita. In alcuni pazienti causa complicazione come la sindrome nefrosica.
4. P.knowlesi; si trova in natutra nei macachi e infezioni umane acquistate
5. P.falciparum;

#### Algoritmo

1. Segmentazione
2. Feature extraction
3. Feature selection
4. Classification

##### Segmentation

La segmentazione è il processo in cui si procede con la suddivisione dell’immagine digitale in disgiunte, non sovrapposte, regioni (con dentro un parassita). Per ognuna di esse bisogna procedere con le misurazioni e con la classificazione -> ***Thresholding.***

# Differenze tra il PlugIn di Landini e PlugIns Basic di Image

ImageJ ha di base alcuni strumenti utili per l’analisi di immagini e si presume per il calcolo delle Measures, cosa che fa anche Landini nel suo PlugIn.

|  |  |  |
| --- | --- | --- |
| Measures | Particles8 | ImageJ Basic |
| Area | ✔ | ✔ |
| Perim | ✔ | ✔ |
| CHull | ✔ |  |
| CAree | ✔ |  |
| CPerim |  |  |
| Feret | ✔ | ✔ |
| Width | ✔ | ✔ |
| MinR | ✔ |  |
| MaxR | ✔ |  |
| MBCRadius | ✔ |  |
| AspRatio | ✔ |  |
| Circ | ✔ | ✔ |
| Roundness |  |  |
| ArEquivD | ✔ |  |
| PerEquivD | ✔ |  |
| EquivEllAr | ✔ |  |
| Compactness | ✔ |  |
| Solidity | ✔ | ✔ |
| Concavity | ✔ |  |
| Convexity | ✔ |  |
| Shape |  |  |
| RFactor |  |  |
| ArBBox | ✔ |  |
| Rectang | ✔ |  |
| ModRatio | ✔ |  |
| Sphericity | ✔ |  |
| Pixels | ✔ | ✔ sotto forma di elements |
| IS |  |  |
| CG |  |  |
| DS |  |  |
| Haralick ratio |  |  |
| Elongation |  |  |
| Skeleton lenght mm |  |  |
| Mean fibre width |  |  |
| Width standard deviation |  |  |
| GLCM features |  |  |
| Mean Intesity |  |  |
| Intensity std deviation |  |  |
| Skewness or third moment |  |  |
| Kurtosis |  |  |
| Entropy |  |  |
| Insentisy sum |  |  |
| Square intesity sum |  |  |
| Uniformity |  |  |
| Smothness R |  |  |
| Mode |  | ✔ |
| Median |  | ✔ |
| Max |  | ✔ |
| Hu moments |  |  |
|  |  |  |

# AGGIORNAMENTO SUL FUNZIONAMENTO DI iMAGEJ

## INTRODUZIONE

Dopo un’analisi più approfondita, per ricercare le Features, nella documentazione viene riportato il seguente link:

<https://imagescience.org/meijering/software/featurej/>

che invita a scaricare due pacchetti jar:

FeatureJ: <https://imagescience.org/meijering/software/download/FeatureJ_.jar>

ImageScience: <https://imagescience.org/meijering/software/download/imagescience.jar>

Sempre nella ricerca di materiale utile, si procede nello scaricare le seguenti classi:

<https://imagej.nih.gov/ij/plugins/particle-remover.html>

<https://imagej.nih.gov/ij/plugins/moments.html>

<https://imagej.nih.gov/ij/plugins/cell-counter.html>

<https://imagej.nih.gov/ij/plugins/enclose.html>

<https://imagej.nih.gov/ij/plugins/measure-label.html>

<https://imagej.nih.gov/ij/plugins/hull-circle.html>

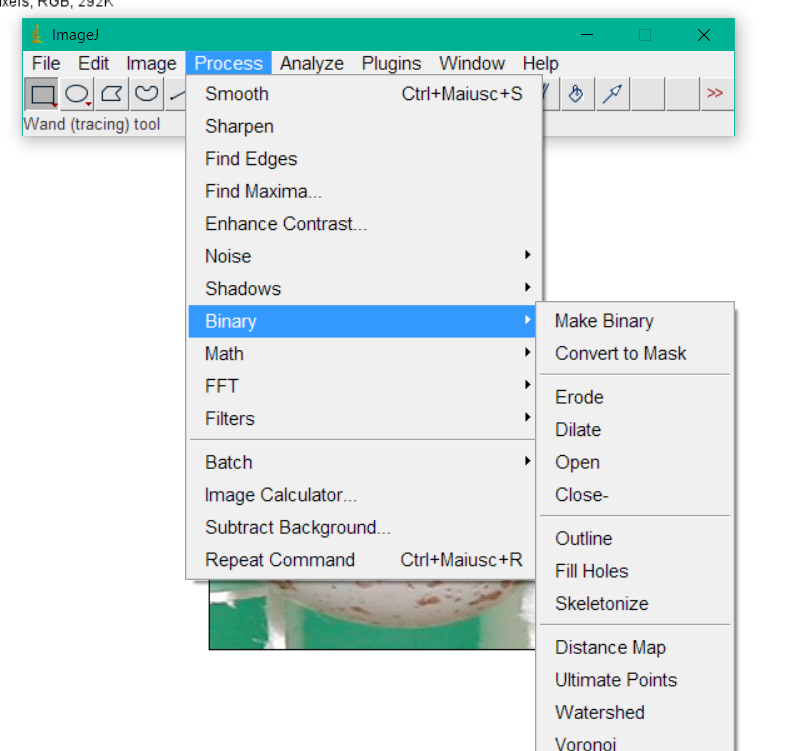
Si prova a prendere una immagine da analizzare e utilizzare il materiale scaricato.

## featureJ

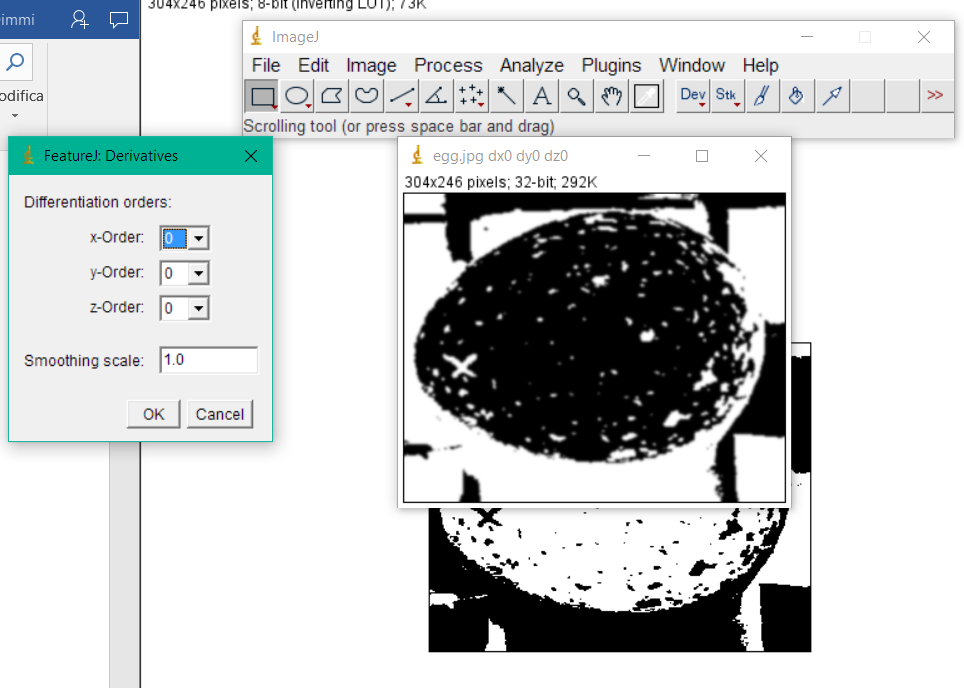
Si apre l’immagine esempio che segue:



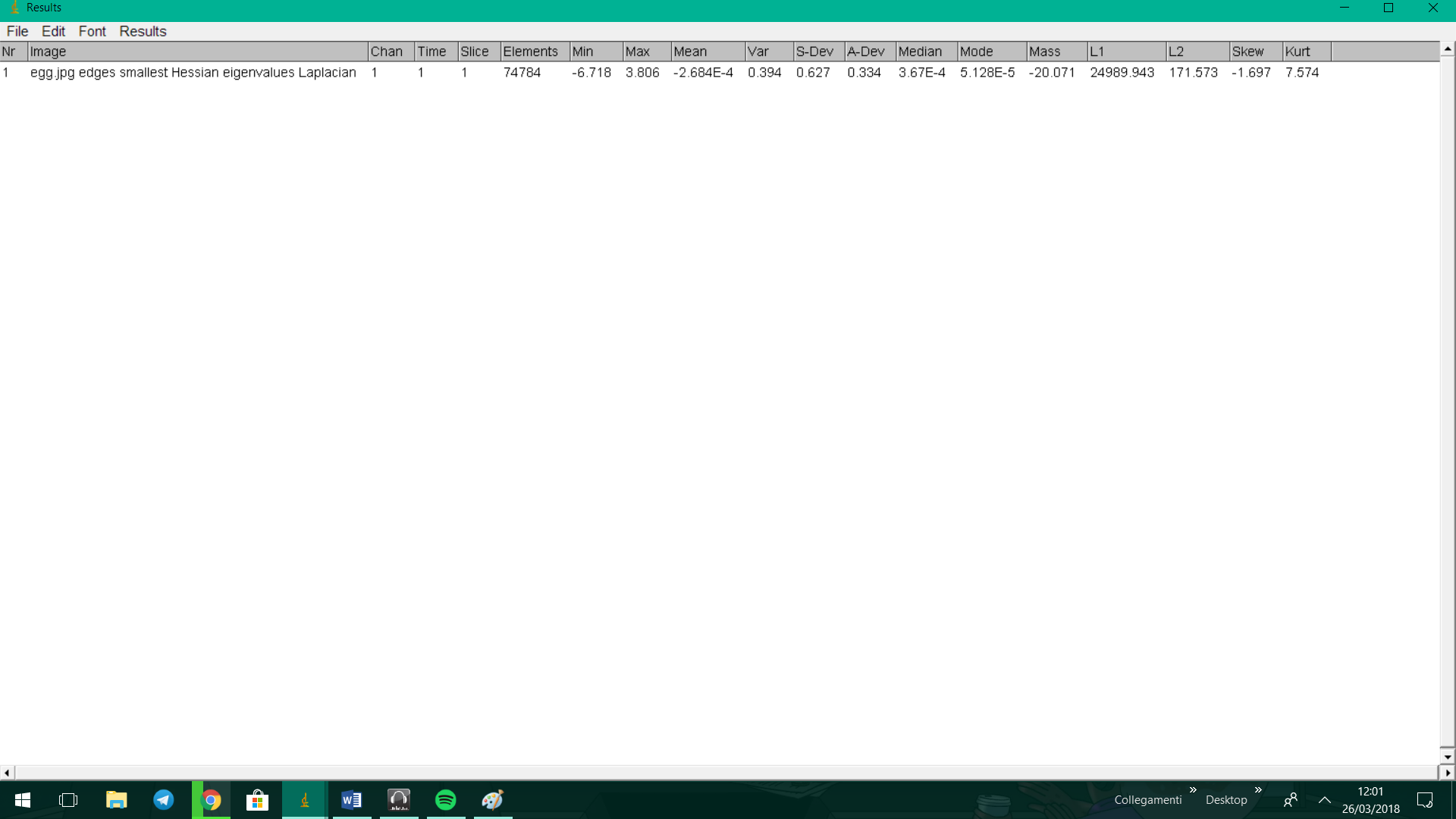
Viene richiesto che sia una grey-image.



Si seleziona MakeBinary e si riprova.



Le prime quattro danno una modifica dell’immagine ma in statics viene dato una tabella con valori.



## iMAGESCIENCE

Sono le librerie già presenti. Però si ricava il codice sorgente:

<https://github.com/imagescience/FeatureJ/tree/master/src/main/java/featurej>

<https://github.com/imagescience/ImageScience>

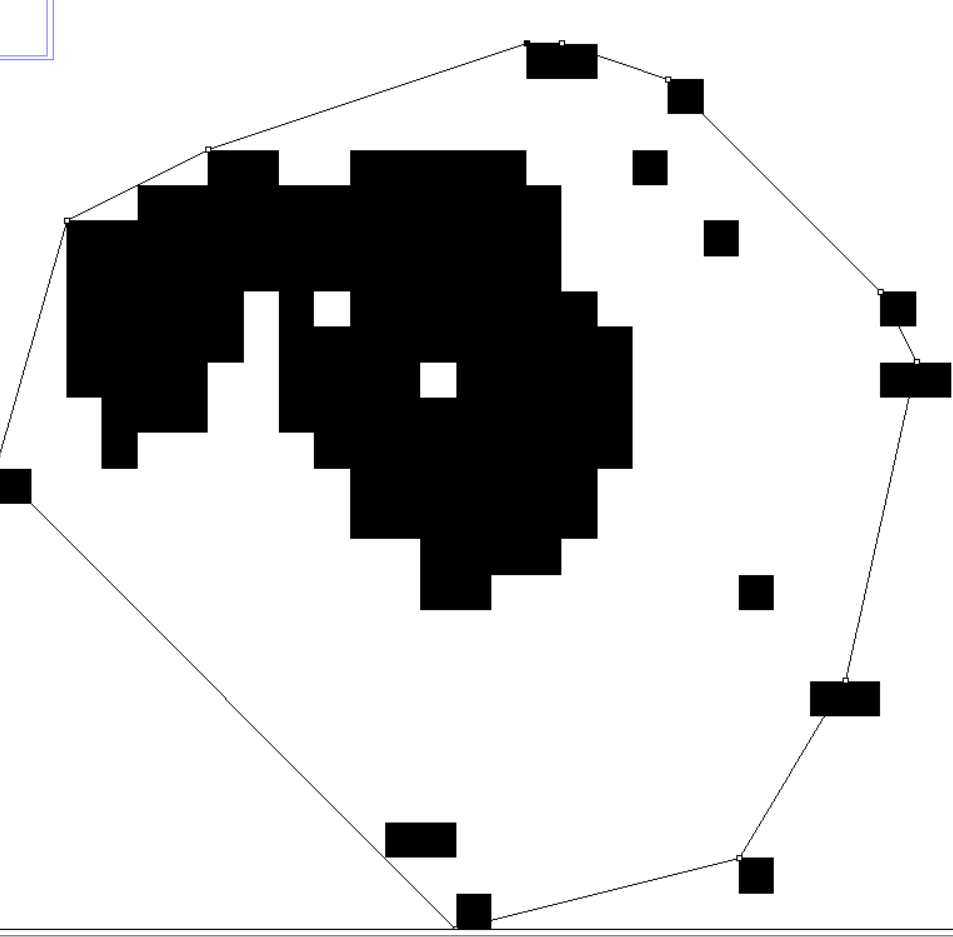
# Punto della situazione

L’obbiettivo è arricchire il Plugin Personale per calcolare tutte i features morfologiche. Estraendole da Landini o da ImageJ.

<http://www.mecourse.com/landinig/software/software.html>

### convex hull

Ho trovato un plugin fatto da Landini per esprimere il convex hull:



Questo è risultato ottenuto.

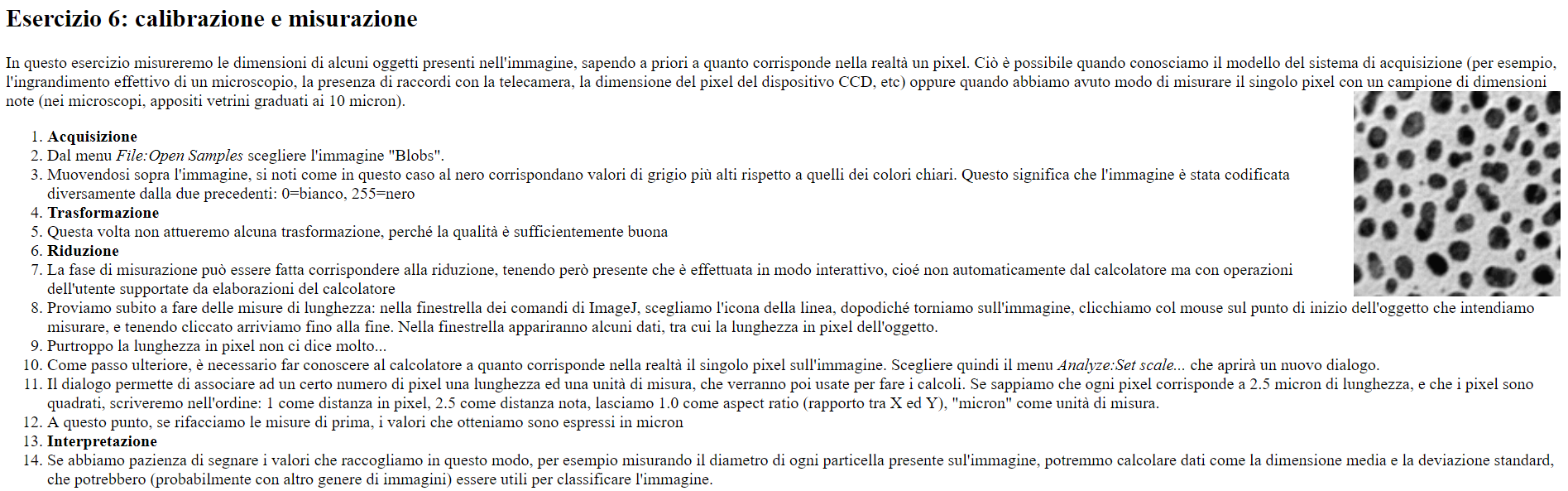
# Novità

Sono alla ricerca degli strumenti già implementati da ImageJ. Mi imbatto in una pagina in italiano in cui da esercizi per ImageJ stesso.

<http://mitel.dimi.uniud.it/lavim/ImageJ/>

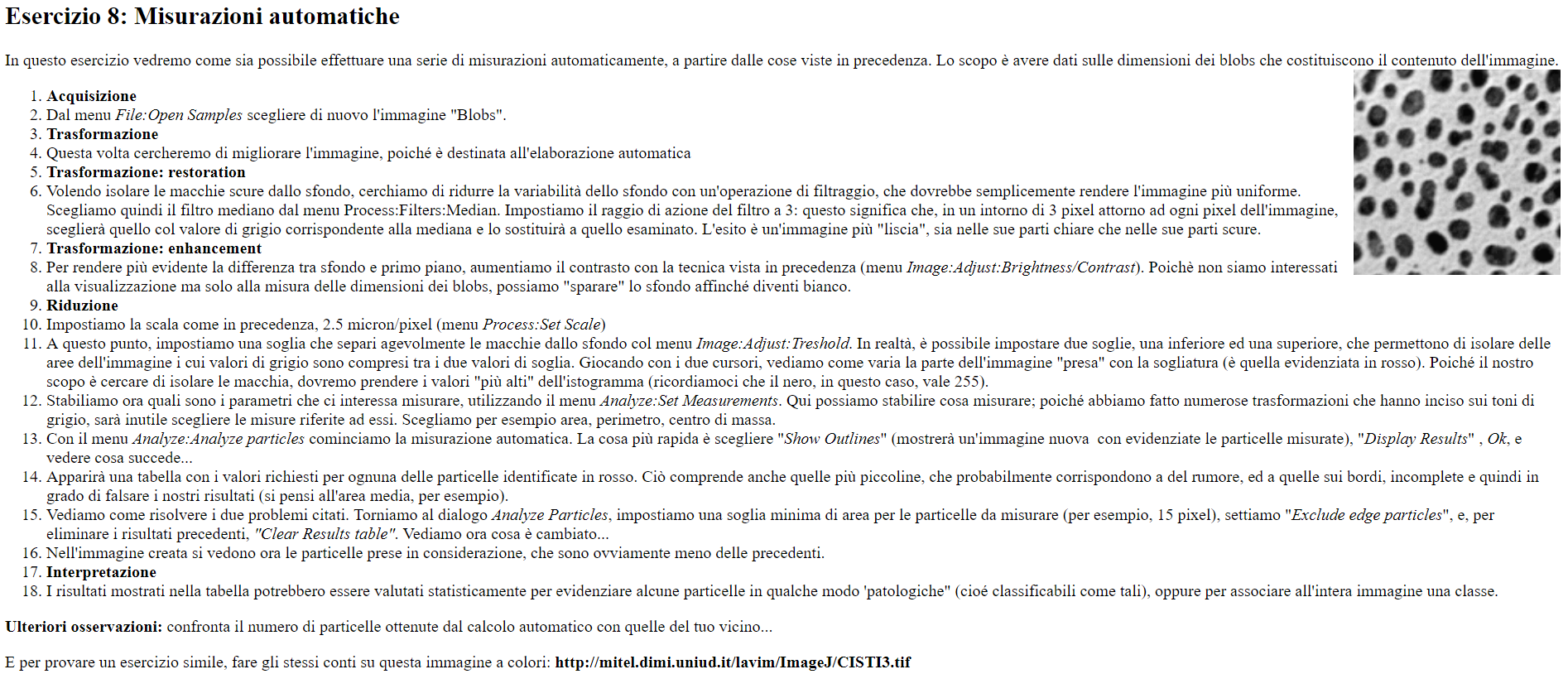
La cosa interessante sono due esercizi in particolare.

Il primo è il seguente:



|  |
| --- |
| Esercizio 6: calibrazione e misurazione In questo esercizio misureremo le dimensioni di alcuni oggetti presenti nell'immagine, sapendo a priori a quanto corrisponde nella realtà un pixel. Ciò è possibile quando conosciamo il modello del sistema di acquisizione (per esempio, l'ingrandimento effettivo di un microscopio, la presenza di raccordi con la telecamera, la dimensione del pixel del dispositivo CCD, etc) oppure quando abbiamo avuto modo di misurare il singolo pixel con un campione di dimensioni note (nei microscopi, appositi vetrini graduati ai 10 micron).test image   1. **Acquisizione** 2. Dal menu File:Open Samples scegliere l'immagine "Blobs". 3. Muovendosi sopra l'immagine, si noti come in questo caso al nero corrispondano valori di grigio più alti rispetto a quelli dei colori chiari. Questo significa che l'immagine è stata codificata diversamente dalla due precedenti: 0=bianco, 255=nero 4. **Trasformazione** 5. Questa volta non attueremo alcuna trasformazione, perché la qualità è sufficientemente buona 6. **Riduzione** 7. La fase di misurazione può essere fatta corrispondere alla riduzione, tenendo però presente che è effettuata in modo interattivo, cioé non automaticamente dal calcolatore ma con operazioni dell'utente supportate da elaborazioni del calcolatore 8. Proviamo subito a fare delle misure di lunghezza: nella finestrella dei comandi di ImageJ, scegliamo l'icona della linea, dopodiché torniamo sull'immagine, clicchiamo col mouse sul punto di inizio dell'oggetto che intendiamo misurare, e tenendo cliccato arriviamo fino alla fine. Nella finestrella appariranno alcuni dati, tra cui la lunghezza in pixel dell'oggetto. 9. Purtroppo la lunghezza in pixel non ci dice molto... 10. Come passo ulteriore, è necessario far conoscere al calcolatore a quanto corrisponde nella realtà il singolo pixel sull'immagine. Scegliere quindi il menu Analyze:Set scale... che aprirà un nuovo dialogo. 11. Il dialogo permette di associare ad un certo numero di pixel una lunghezza ed una unità di misura, che verranno poi usate per fare i calcoli. Se sappiamo che ogni pixel corrisponde a 2.5 micron di lunghezza, e che i pixel sono quadrati, scriveremo nell'ordine: 1 come distanza in pixel, 2.5 come distanza nota, lasciamo 1.0 come aspect ratio (rapporto tra X ed Y), "micron" come unità di misura. 12. A questo punto, se rifacciamo le misure di prima, i valori che otteniamo sono espressi in micron 13. **Interpretazione** 14. Se abbiamo pazienza di segnare i valori che raccogliamo in questo modo, per esempio misurando il diametro di ogni particella presente sul'immagine, potremmo calcolare dati come la dimensione media e la deviazione standard, che potrebbero (probabilmente con altro genere di immagini) essere utili per classificare l'immagine. |

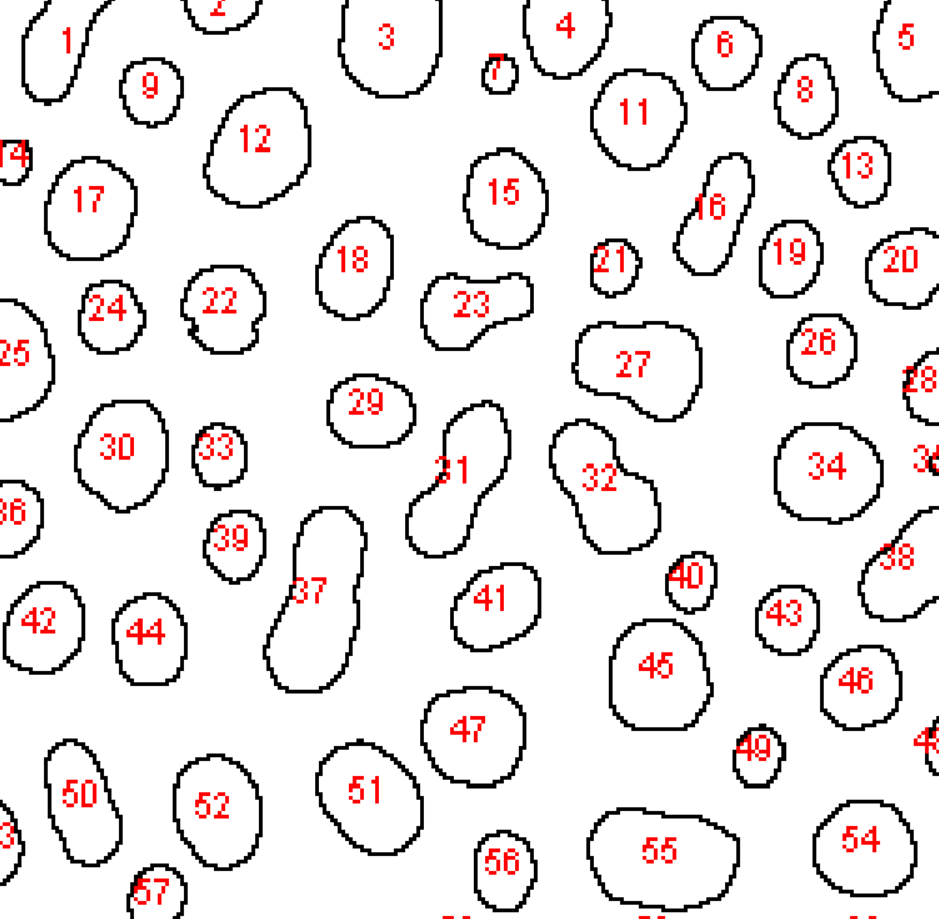
E anche questo esercizio che segue:



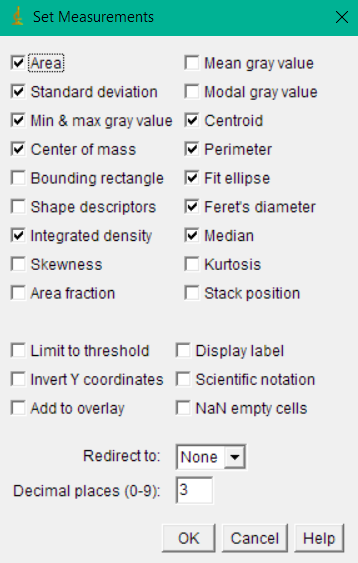
|  |
| --- |
| Esercizio 8: Misurazioni automatiche In questo esercizio vedremo come sia possibile effettuare una serie di misurazioni automaticamente, a partire dalle cose viste in precedenza. Lo scopo è avere dati sulle dimensioni dei blobs che costituiscono il contenuto dell'immagine.test image   1. **Acquisizione** 2. Dal menu File:Open Samples scegliere di nuovo l'immagine "Blobs". 3. **Trasformazione** 4. Questa volta cercheremo di migliorare l'immagine, poiché è destinata all'elaborazione automatica 5. **Trasformazione: restoration** 6. Volendo isolare le macchie scure dallo sfondo, cerchiamo di ridurre la variabilità dello sfondo con un'operazione di filtraggio, che dovrebbe semplicemente rendere l'immagine più uniforme. Scegliamo quindi il filtro mediano dal menu Process:Filters:Median. Impostiamo il raggio di azione del filtro a 3: questo significa che, in un intorno di 3 pixel attorno ad ogni pixel dell'immagine, sceglierà quello col valore di grigio corrispondente alla mediana e lo sostituirà a quello esaminato. L'esito è un'immagine più "liscia", sia nelle sue parti chiare che nelle sue parti scure. 7. **Trasformazione: enhancement** 8. Per rendere più evidente la differenza tra sfondo e primo piano, aumentiamo il contrasto con la tecnica vista in precedenza (menu Image:Adjust:Brightness/Contrast). Poichè non siamo interessati alla visualizzazione ma solo alla misura delle dimensioni dei blobs, possiamo "sparare" lo sfondo affinché diventi bianco. 9. **Riduzione** 10. Impostiamo la scala come in precedenza, 2.5 micron/pixel (menu Process:Set Scale) 11. A questo punto, impostiamo una soglia che separi agevolmente le macchie dallo sfondo col menu Image:Adjust:Treshold. In realtà, è possibile impostare due soglie, una inferiore ed una superiore, che permettono di isolare delle aree dell'immagine i cui valori di grigio sono compresi tra i due valori di soglia. Giocando con i due cursori, vediamo come varia la parte dell'immagine "presa" con la sogliatura (è quella evidenziata in rosso). Poiché il nostro scopo è cercare di isolare le macchia, dovremo prendere i valori "più alti" dell'istogramma (ricordiamoci che il nero, in questo caso, vale 255). 12. Stabiliamo ora quali sono i parametri che ci interessa misurare, utilizzando il menu Analyze:Set Measurements. Qui possiamo stabilire cosa misurare; poiché abbiamo fatto numerose trasformazioni che hanno inciso sui toni di grigio, sarà inutile scegliere le misure riferite ad essi. Scegliamo per esempio area, perimetro, centro di massa. 13. Con il menu Analyze:Analyze particles cominciamo la misurazione automatica. La cosa più rapida è scegliere "Show Outlines" (mostrerà un'immagine nuova  con evidenziate le particelle misurate), "Display Results" , Ok, e vedere cosa succede... 14. Apparirà una tabella con i valori richiesti per ognuna delle particelle identificate in rosso. Ciò comprende anche quelle più piccoline, che probabilmente corrispondono a del rumore, ed a quelle sui bordi, incomplete e quindi in grado di falsare i nostri risultati (si pensi all'area media, per esempio). 15. Vediamo come risolvere i due problemi citati. Torniamo al dialogo Analyze Particles, impostiamo una soglia minima di area per le particelle da misurare (per esempio, 15 pixel), settiamo "Exclude edge particles", e, per eliminare i risultati precedenti, "Clear Results table". Vediamo ora cosa è cambiato... 16. Nell'immagine creata si vedono ora le particelle prese in considerazione, che sono ovviamente meno delle precedenti. 17. **Interpretazione** 18. I risultati mostrati nella tabella potrebbero essere valutati statisticamente per evidenziare alcune particelle in qualche modo 'patologiche" (cioé classificabili come tali), oppure per associare all'intera immagine una classe.   **Ulteriori osservazioni:** confronta il numero di particelle ottenute dal calcolo automatico con quelle del tuo vicino...  E per provare un esercizio simile, fare gli stessi conti su questa immagine a colori: **http://mitel.dimi.uniud.it/lavim/ImageJ/CISTI3.tif** |

In particolare l’ultimo esercizio, misura SINGOLARMENTE, ogni particella e ciò è molto interessante:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Area | XM | YM | Perim. |
| 1 | 2.656.250 | 51.409 | 33.715 | 226.066 |
| 2 | 1.131.250 | 158.640 | 11.775 | 138.033 |
| 3 | 4.100.000 | 272.188 | 32.454 | 241.317 |
| 4 | 2.687.500 | 387.558 | 25.547 | 197.782 |
| 5 | 2.981.250 | 618.341 | 34.982 | 215.104 |
| 6 | 1.725.000 | 494.375 | 40.688 | 151.317 |
| 7 | 437.500 | 343.893 | 55.750 | 74.497 |
| 8 | 1.656.250 | 548.024 | 70.618 | 153.388 |
| 9 | 1.375.000 | 112.614 | 66.534 | 136.317 |
| 10 | 162.500 | 2.981 | 67.500 | 66.213 |
| 11 | 3.075.000 | 436.113 | 84.172 | 205.208 |
| 12 | 4.062.500 | 184.454 | 104.412 | 242.279 |
| 13 | 1.375.000 | 583.977 | 119.114 | 136.317 |
| 14 | 550.000 | 20.426 | 113.409 | 85.962 |
| 15 | 2.756.250 | 347.372 | 138.262 | 194.602 |
| 16 | 2.443.750 | 488.053 | 149.274 | 213.744 |
| 17 | 3.193.750 | 72.091 | 144.841 | 208.137 |
| 18 | 2.606.250 | 248.474 | 184.182 | 191.066 |
| 19 | 1.606.250 | 536.766 | 177.515 | 147.782 |
| 20 | 2.168.750 | 614.773 | 183.656 | 171.924 |
| 21 | 925.000 | 420.000 | 183.108 | 112.175 |
| 22 | 2.450.000 | 160.810 | 210.657 | 191.673 |
| 23 | 2.581.250 | 325.475 | 210.209 | 211.066 |
| 24 | 1.575.000 | 85.486 | 216.518 | 146.317 |
| 25 | 3.125.000 | 20.735 | 245.165 | 219.853 |
| 26 | 1.731.250 | 557.568 | 238.488 | 152.782 |
| 27 | 4.168.750 | 438.203 | 248.941 | 255.208 |
| 28 | 1.068.750 | 628.458 | 263.969 | 133.033 |
| 29 | 2.200.000 | 258.253 | 279.467 | 171.924 |
| 30 | 3.331.250 | 93.825 | 307.076 | 215.208 |
| 31 | 3.737.500 | 317.763 | 324.958 | 274.099 |
| 32 | 3.931.250 | 413.400 | 331.091 | 258.492 |
| 33 | 1.137.500 | 157.679 | 308.365 | 124.246 |
| 34 | 3.643.750 | 561.207 | 319.736 | 221.673 |
| 35 | 87.500 | 634.464 | 314.107 | 34.749 |
| 36 | 1.643.750 | 19.083 | 349.910 | 149.246 |
| 37 | 5.537.500 | 224.046 | 406.558 | 324.957 |
| 38 | 2.931.250 | 615.243 | 383.889 | 218.137 |
| 39 | 1.443.750 | 168.350 | 367.733 | 139.853 |
| 40 | 981.250 | 471.760 | 392.126 | 115.711 |
| 41 | 2.518.750 | 342.106 | 408.657 | 187.530 |
| 42 | 2.525.000 | 41.603 | 422.853 | 183.995 |
| 43 | 1.462.500 | 535.524 | 417.222 | 139.246 |
| 44 | 2.300.000 | 111.617 | 430.985 | 176.924 |
| 45 | 4.012.500 | 449.622 | 455.362 | 233.137 |
| 46 | 2.306.250 | 583.953 | 462.564 | 175.459 |
| 47 | 3.537.500 | 327.063 | 494.377 | 219.602 |
| 48 | 393.750 | 634.623 | 502.560 | 90.355 |
| 49 | 975.000 | 516.090 | 508.542 | 114.246 |
| 50 | 2.875.000 | 66.424 | 539.929 | 216.673 |
| 51 | 3.750.000 | 258.217 | 535.821 | 228.492 |
| 52 | 3.343.750 | 156.600 | 544.516 | 215.208 |
| 53 | 1.262.500 | 11.027 | 565.891 | 157.175 |
| 54 | 3.381.250 | 586.490 | 569.323 | 211.673 |
| 55 | 5.300.000 | 451.848 | 575.834 | 282.279 |
| 56 | 1.675.000 | 346.493 | 584.002 | 151.317 |
| 57 | 1.287.500 | 116.262 | 602.597 | 131.317 |
| 58 | 525.000 | 320.208 | 629.554 | 113.891 |
| 59 | 462.500 | 447.939 | 629.459 | 98.891 |
| 60 | 300.000 | 586.875 | 630.260 | 78.891 |
| 61 | 281.250 | 185.583 | 631.083 | 81.820 |



### OSSERVAZIONI

In questo modo si possono calcolare alcune delle features necessarie ma non tutte. L’ideale sarebbe rendere automatico il procedimento “pre” ovvero settaggio ecce cc.

Il fulcro è la selezione dell’area selezionata che potrebbe facilitare le cose; questo avviene attraverso il comando “threshold”.

## analisi comando threshold

È sotto la voce di Ajust e in inglese vuol dire “soglia” o “confine”.

### SORGENTE

<https://imagej.nih.gov/ij/source/ij/plugin/frame/ThresholdAdjuster.java>

# due strade

24-04-2018

Dopo un colloquio con la docente, si sono aperte due strade in particolare per il continuo della tesi.

## prima opzione-implementazione di analyze particle e set measures

L’idea sarebbe quella di aggiungere al menù “Set Measurements”, almeno una check box in più corrispondenti a delle nuove scelte e inserire le nuove features mancanti già implementate nel plugin di prova “Primitive Measures”, inserendole in Analyze Particles.



Per fare questo, dato che si parla di un elemento fondamentale del tool di ImageJ e non di un elemento installato e aggiuntivo, bisognerebbe prendere il codice sorgente di ImageJ, dopo un backup opportuno e modificare i due elementi aggiungendo quello che ci serve.

## seconda opzione-implementazione di utilizzo di THRESHOULD NEL PLUGIN DI PROVA “PRIMITIVE MEASURES”

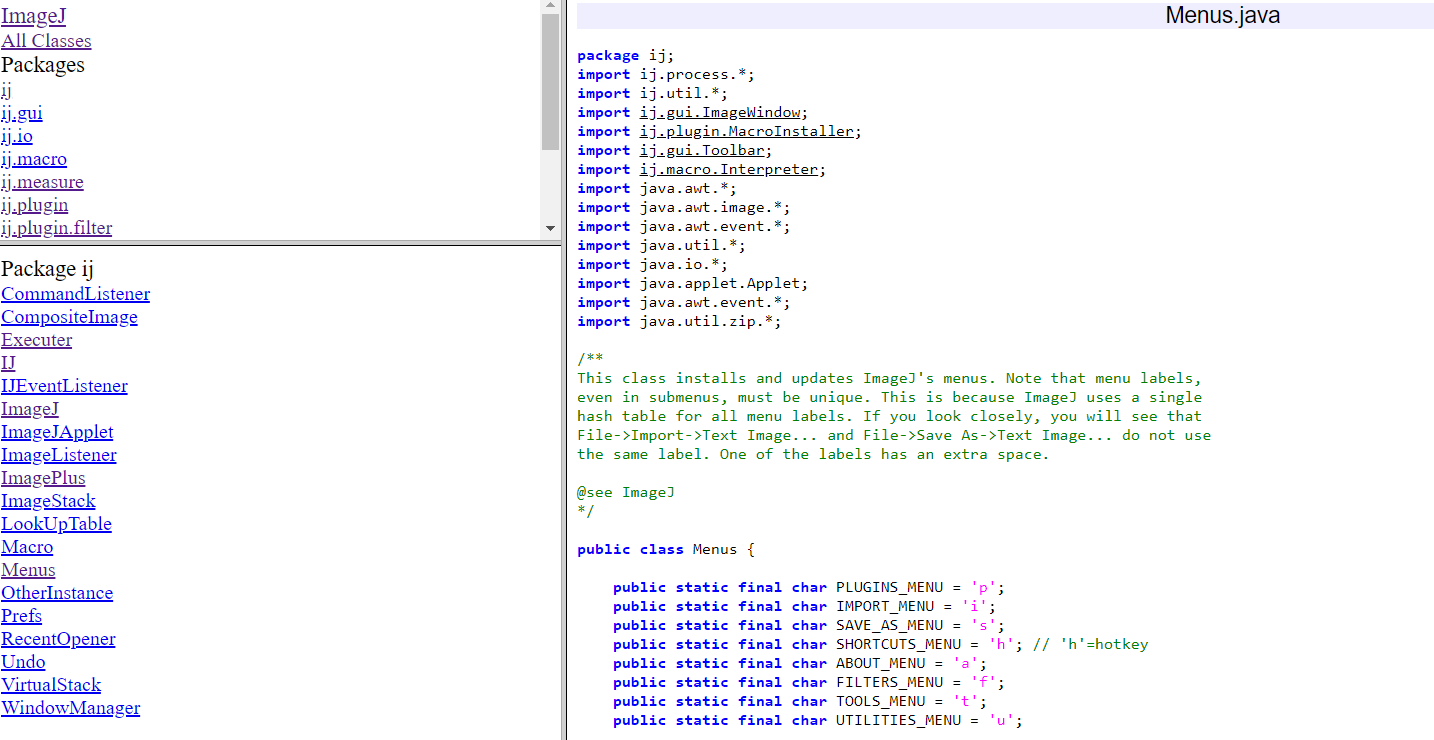
L’idea sarebbe quella di implementare l’utilizzo delle “soglie”, “Threashould” nel plugin di prova, in maniera tale che le i metodi usati si possano applicare a più oggetti in una immagine invece di applicarli solo in un solo oggetto. Questa implementazione la possiamo ricavare da Landini o da Analize Particles.

### Prima esperimentazione applicando la prima opzione

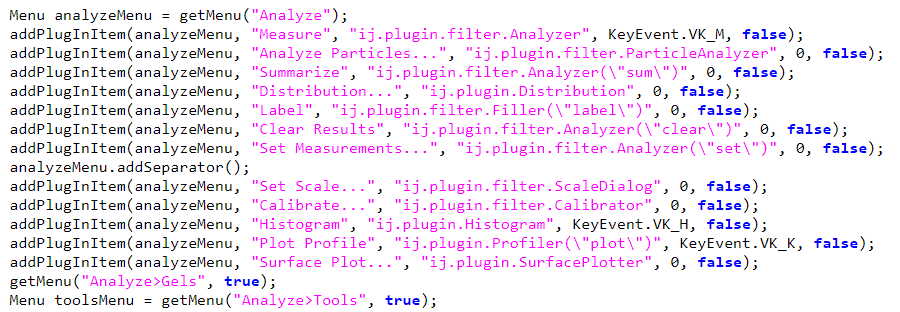
#### suddivisone dell’obbiettivo in più task

1. Entrare nel codice sorgente di ImageJ
2. Entrare nel menù Analyze
3. Modificare “Set Measurements” aggiungendo l’opzione “Roundness”
4. Aggiungere l’elemento “Roundness” in Analyze Particles

##### codice sorgente di Imagej



Dalla pagina <https://imagej.nih.gov/ij/developer/source/> riesco a ricavare il punto in cui viene richiamato Set Measurements.



Che a quanto pare fa parte della stessa classe Analyzer, che comporta un menù (appunto). Un’altra osservazione importante è che Analyzer è già di per sé un plugin filter. Il codice sorgente che segue <https://imagej.nih.gov/ij/developer/source/ij/plugin/filter/Analyzer.java.html> è di Analyzer. L’immagine che segue descrive in codice la GenericDialog di Set Measurements.

analisi del funzionamento del settaggio delle misure all’interno di imagej

All’interno del menù Analyze, ci si può avvicinare al funzionamento delle misure all’interno di ImageJ. In particolare, Set Measurements, oggetto dell’analisi in particolare, è una Generic Dialog con un insieme di checkbox spuntabili a piacimento. Per poterne capire il funzionamento, considero solo l’opzione riguardante l’area. Vengono dichiarati due vettori, labels di tipo stringa e states di tipo booleano. In particolare:

states[0]=(systemMeasurements&AREA)!=0;

per cui:

**private** **static** **int** systemMeasurements = Prefs.getInt(MEASUREMENTS,AREA+MEAN+MIN\_MAX);

SystemMeasurements prende valori già settati per cui vengono confrontati con le checkbox appena selezionate.

Infatti:

// Order must agree with order of checkboxes in Set Measurements dialog box

**private** **static** **final** **int**[] list = {AREA,MEAN,STD\_DEV,MODE,MIN\_MAX,

CENTROID,CENTER\_OF\_MASS,PERIMETER,RECT,ELLIPSE,SHAPE\_DESCRIPTORS, FERET,

INTEGRATED\_DENSITY,MEDIAN,SKEWNESS,KURTOSIS,AREA\_FRACTION,STACK\_POSITION,

LIMIT,LABELS,INVERT\_Y,SCIENTIFIC\_NOTATION,ADD\_TO\_OVERLAY,NaN\_EMPTY\_CELLS};

Che corrisponde a quello che si trova in SetMeasurements.

Dopo che vengono settate le misure, vengono salvate nelle Measurements utilizzare (old measurements). In particolare si denoti che è un int.

La problematica che si presenta è che è rischioso dover cambiare qualche riga di codice senza poterla testare; si potrebbe bloccare completamente il tool di ImageJ. Inoltre queste Measurements sono ben connesse con altri strumenti di ImageJ, dunque non si capisce se è possibile aggiungere qualcosa.

### note

Ricercando sul web mi sono imbattuta su una domanda nel blog riguardante imageJ. Partendo dal presupposto che la domanda posta mi potesse interessare solo in parte, mi sono imbattuta su una risposta in particolare che ho trovato molto interessante.

<http://imagej.1557.x6.nabble.com/Details-on-calling-a-plugin-from-another-one-td3696579.html>

|  |
| --- |
| You can do this in a plugin by extending the ParticleAnalyzer class and overriding the saveResults() method. I have attached an example plugin that measures the mean pixel value along the perimeter of particles. It displays the perimeter mean as "PMean" and the number of points sampled as "N". Here is an example of the Results table output:             Area      Mean    Perim.    PMean   N  1         81        161.5     32.4      123.2     40  2         30        147.2     30.5      127.8     34  3         99        165.3     35.8      124.4     44  4         14        138.9     12.5      119        16  5         22        147.6     17.3      105.8     22  6         69        185.7     39.0      84.7       46  7         3          130.7     5.7        122          8  8         1          128        2.8        120          4  9         81        183.4     42.4      84.3       50  10       90        181.5     49.6      82.9       56  11       53        188.4     35.6      84.8       42  12       49        172.9     36.7      80          42    import ij.\*;  import ij.measure.\*;  import ij.plugin.\*;  import ij.plugin.filter.\*;  import ij.process.\*;  import ij.gui.\*;  import java.awt.\*;   public class Custom\_Particle\_Analyzer implements PlugIn {       public void run(String arg) {          ImagePlus imp = IJ.getImage();          analyzeParticles(imp);      }       public void analyzeParticles(ImagePlus imp) {          CustomAnalyzer pa = new CustomAnalyzer();          int flags = pa.setup("", imp);          if (flags==PlugInFilter.DONE)              return;          pa.run(imp.getProcessor());          Analyzer.getResultsTable().show("Results");      }       class CustomAnalyzer extends ParticleAnalyzer {          @Override          protected void saveResults(ImageStatistics stats, Roi roi) {              super.saveResults(stats, roi);              ImageProcessor ip = imp.getProcessor();              ip.setRoi(roi);              FloatPolygon p = roi.getInterpolatedPolygon(1, false);              int n = p.npoints;              double sum = 0;              for (int i=0; i<n; i++) {                  int x = (int)Math.round(p.xpoints[i]);                  int y = (int)Math.round(p.ypoints[i]);                  sum += ip.getPixel(x, y);              }              double mean = sum/n;              rt.addValue("PMean", mean);              rt.addValue("N", n);          }      }       } |

Ho testato il pezzo di codice su un nuovo plugin ed effettivamente funziona. Viene aggiunta una misura al SetMeasure (senza dover modificare la tabella). Sarebbe una estensione di Analyze Particles.

Ho provato anche un secondo plugin trovato nel web: <https://imagej.nih.gov/ij/plugins/download/Custom_Particle_Analyzer.java>

Che porta lo stesso nome ma da risultati diversi. Sarebbe molto interessante poter sfruttare questa strada per poter potenziare Analyze Particles, senza doverlo toccare o rovinare, usare allo stesso tempo SetMeasurement, e poter aggiungere più misure senza porsi troppi problemi.

Detto questo i task 3 e 4 non vengono implementati ma ci si concentra in un nuovo task:

procedere con la creazione di un plugin che estenda Analyze particles,

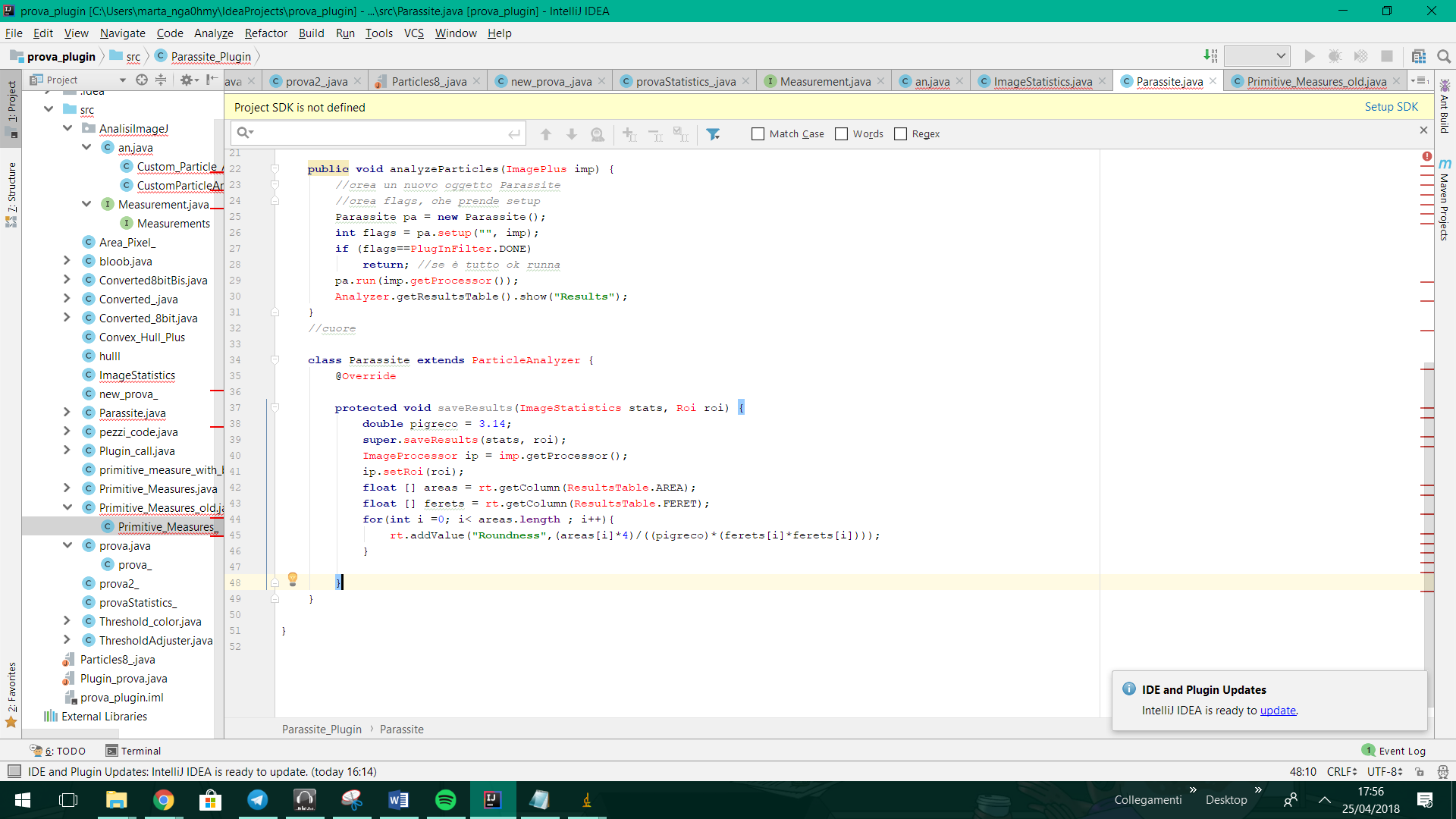
# prova plugin che estenda analyze particles

25-04-18

Dato il materiale trovato in rete, il tentativo è quello di creare un plugin che aggiunga una features in analyze particles senza doverla alterare.

Ho preso come base un plugin che realizzava questo. Sostanzialmente si estende la classe ParticlesAnalyzer. Dunque le features settate da Set Mesurement sono salve e calcolabili.

Il problema che si poneva era estrarre alcuni dati necessari, come l’area ecc dalla results table. Bastava usare il metodo “getCoulmn” allegato con la parola chiave.



Eureka! Sono riuscita con poche righe di codice ad aggiungere la features, roundness..

# approfondimento su plugin parassite

02-05-18

Il plugin “Parassite\_Plugin” creato recentemente, dal punto di vista tecnico, è una classe che estende Plugin, in cui al suo interno si possono individuare tre parti principali:

## prima parte

Il metodo pubblico run, presente in tutti i plugin, riceve l’immagine caricata da ImageJ e richiama il metodo analyzeParticles passandogli l’immagine stessa.

## seconda parte

AnalyzeParticles crea una genericDialog attraverso un metodo e setta Analyze (invocando dunque ParticlesAnalyzer) attraverso la stringa pa.setup e run. Pa è un oggetto Parassite.

## terza parte

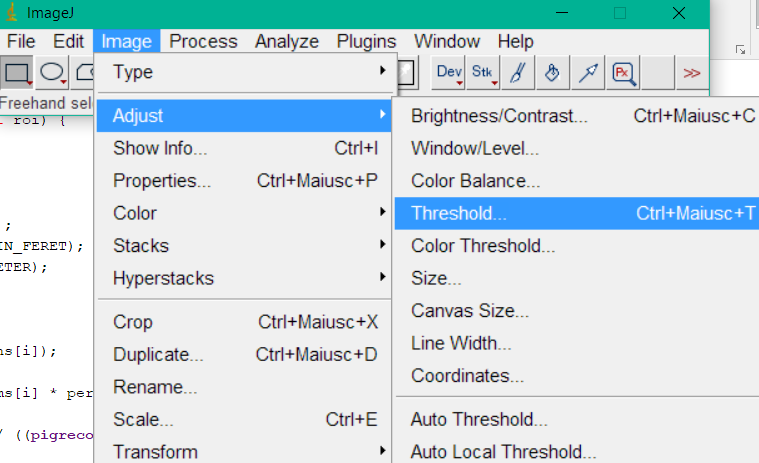
Viene creata una inner class che estende ParticleAnalyzer creando un override di saveResults. Qui vengono risalvati in svariati vettori, alcuni valori fondamentali per aggiungere delle misure mancanti.

I miei prossimi obbiettivi sono:

1. Settare il TheareShould in maniera automatica
   1. Aprile la “finestrella” che la lo setta
      1. OPPURE fare in modo che si setti in automatico
2. Aggiungere le misure:
   1. CHull
   2. Carea
   3. Cperim
   4. MinR
   5. MaxR
   6. MBCRadius
   7. Solidy
   8. Concavity
   9. Convecity
   10. ModRatio
   11. Sphericity
   12. Pixelds
3. Settare AnalyzeParticles in maniera automatica senza che si apra la finestra

# TheareShould

Anche in questo casa la metodica unica e migliore per poter procedere, è dover ricercare nel codice sorgente la chiamata di:



Al momento non sono disponibili le pagine di codice sorgente >-<.

# aggiungere le misure

Per poter aggiungere le misure, sotto consiglio di Andrea, mi imbatto in “Shape Filter”

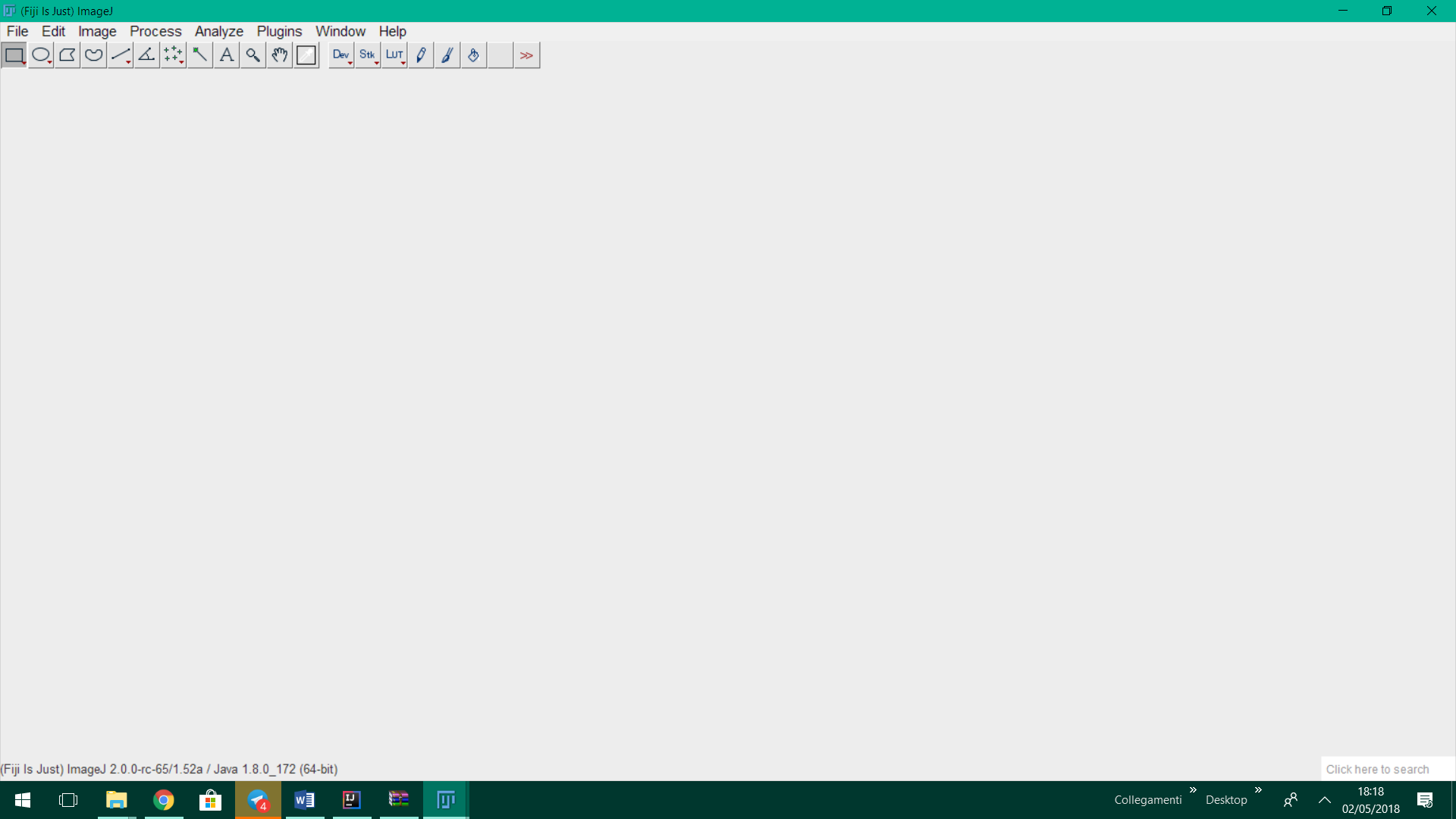
<https://imagej.net/Shape_Filter>

Ci rendiamo conto che per poterlo utilizzare e analizzare ci serbe Fiji, che si presenta come una sorta di “miglioramento” di imageJ.

<https://fiji.sc/>

## fiji

La schermata si presenta identica a ImageJ:



Sicuramente è diverso la metodica di Update e la gestione dei Plugin.

## shape filter

Si presenta subito abbastanza ostico da utilizzare e sembra quasi che a volte NON VOGLIA FUNZIONARE. La prima cosa da ricordare è che utilizza Blob, una libreria e inoltre usa altre classi collegate.

il plugin Parassite\_Plugin funziona anche su Fiji, grazie alla presenta della classe estesa Particle\_Analyzer.

Il prossimo obbiettivo è prelevare da Shape Filter il calcolo delle hull (convex, area, perimetro).

# aggiornamenti importanti

1. RI-abbandono di Fiji. Instabile e creatore di problemi inutili.
2. Shape Filter risulta inule.
3. Rianalizi del codice sorgente di Analyzer (menù e base per SET MEASUREMENT e Particle Analyzer, a cui si appoggia il mio plugin di prova).

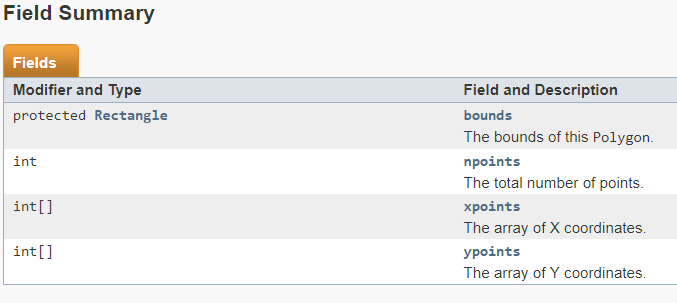
## osservazione codice sorgente di ANALYZER

A seguire la parte di codice che mi interessava particolarmente:

|  |
| --- |
| **if** ((**measurements**&SHAPE\_DESCRIPTORS)!=0) {  **double** circularity = perimeter==0.0?0.0:4.0\*Math.PI\*(stats.area/(perimeter\*perimeter));  **if** (circularity>1.0) circularity = 1.0;  **rt**.addValue(ResultsTable.CIRCULARITY, circularity);  Polygon ch = **null**;  **boolean** isArea = roi==**null** || roi.isArea();  **double** convexArea = roi!=**null**?getArea(roi.getConvexHull()):stats.pixelCount;  **rt**.addValue(ResultsTable.ASPECT\_RATIO, isArea?stats.major/stats.minor:0.0);  **rt**.addValue(ResultsTable.ROUNDNESS, isArea?4.0\*stats.area/(Math.PI\*stats.major\*stats.major):0.0);  **rt**.addValue(ResultsTable.SOLIDITY, isArea?stats.pixelCount/convexArea:Double.NaN);  **if** (**rt**.size()==1) {  **rt**.setDecimalPlaces(ResultsTable.CIRCULARITY, *precision*);  **rt**.setDecimalPlaces(ResultsTable.ASPECT\_RATIO, *precision*);  **rt**.setDecimalPlaces(ResultsTable.ROUNDNESS, *precision*);  **rt**.setDecimalPlaces(ResultsTable.SOLIDITY, *precision*);  }  *//rt.addValue(ResultsTable.CONVEXITY, getConvexPerimeter(roi, ch)/perimeter);* } |

Se in SET MEASUREMENT viene “checckata” la voce SHAPE .. vengono calcolati valori legati al convex hull area, che non viene stampato. Da qui la scoperta che ROI (classe che indica la zona di interesse) ha un metodo che viene appunto chiamato getConvexHull(). Questo rende un oggetto appartenente alla classe “Polygon” classe per la geometria in Java (dal 7). Questo oggetto ha 4 attributi in particolare.

<https://docs.oracle.com/javase/7/docs/api/java/awt/Polygon.html>



Quello che interessava maggiormente era il calcolo dell’area e del perimetro del convexHull.

Il calcolo dell’area è ripreso dal codice sorgente stesso ma con qualche modifica di tipo tecnico (fa un ciclo a vuoto dunque viene diminuito di uno) e il calcolo del perimetro è completamente nuovo fatto attraverso le distanze, da notare nel codice sorgente che questo è commentato (evidentemente nessuno aveva voglia di implementarlo eheh).

Al momento sono presenti le seguenti misure:

Morphological features:

Area

Perim

CArea

CPerim

Feret or Lenght (from ImageJ)

Breadth or Width (from Imagej)

AspRatio

Circ

Roundness

ArEquivD

PerEquivD

EquivEllAr

Compactness

Solidity (from ImageJ)

Concavity

Convexity

Shape

Rectang or Extent

Elongation

È da accertare la presenza di MinR e MaxR e conseguentemente di MBC radius.

Ho implementato il settaggio delle misure in automatico in maniera tale da non avere il disturbo di andare a settarlo nell’altro menù. Le prossime cose da settare in automatico solo le finestre di Analyze Particles; è possibile farlo solo attraberso l’Override dei metodi già presenti nella classe, dato che il mio plugin è una estensione di essa, anzi, è una inner class all’interno di un Plugin normale (??).

Da settare in automatico, inoltre, è l’avvio di Threshould.

Bisogna aggiungere le misure di:

MinR.

MaxR.

MBC.

IS

..ec

# aggiornamenti

Ho dato per scontato che MinR e MaxR fossero la metà del feret e del min feret ma i dati non mi sembrano dare questa conferma. MBC non è necessario calcolarlo.

Ho provato a settare in automatico il Threshould creando un nuovo oggetto ma si “incasinano” le finestre.

Sono state aggiunte e tolte alcune misure dalla stesura del Plugin.

Le misure da eliminare sono IS, CD, e DS.

La nuova misura più importante da implementare si chiama Bending Energy.

In più si aggiunge Nucleo, citoplasma ecc.

# prova classe singola

Ho provato a creare un PlugIn che non dovesse creare una inner class per poter implementare le nuove misure. Ci sarebbero dei problemi perché le nuove misure non vengono implementate al meglio.

# Riunione

23-05-2018

I prossimi task da compiere sono:

1. Test su Immagini con un oggetto
2. Test su Immagini con complete binarie
3. Capire se si può settare un threshould in automatico

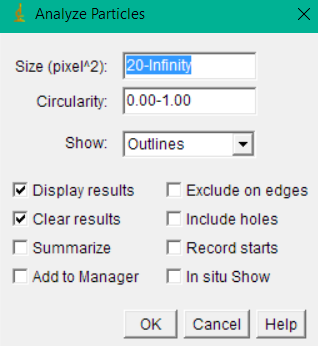
## Ultime misure da implementare per immagini binarie

**Pixels:**

**Heralick ratio:** mean radius / standard devition of radii

Mean radius inteso come la media dei diametri/2 che compone l’oggetto. Standard devition la deviazione standard dei raggi.

### threshould

Il problema posto è l’autosettaggio delle soglie per cui un oggetto deve essere inserito nella valutazione o meno. La soluzione era davanti gli occhi: dalla prima schermata si può semplicemente settare la grandezza minima del pixel per cui deve essere grande un oggetto. Dunque l’ideale sarebbe creare una finestra di dialogo unica per cui si può settare la grandezza dell’oggetto e la circularuty. Tutto il resto si potrebbe anche eliminare.

#### prototipo

|  |  |
| --- | --- |
| Size (pixel^2) |  |
| Circularity |  |
| Misure |  |

Purtroppo non è possibile implementare e modificare la finestra di dialogo di AnalyzeParticles. Avendo alcuni variabili private non è possibile trattarle.

Una possibile soluzione sarebbe creare una nuova classe ParticleAnalyzer uguale ma con le variabili pubbliche o protette. A parere mio sarebbe un approccio brutale.

# completamento misure binarie (calcolabili)

Con l’aggiunta del Haralick Ratio ho quasi finito le misure su immagini binarie su più elementi. Ricordo che il plugin estende la classe Particles Analyzer da cui mi sono presa le misure già implementate.

* Area
* Perim
* CHull
* CArea
* CPerim
* Feret
* Breadth
* MinR
* MaxR
* AspRatio
* Circ
* Roundness
* ArEquivD
* PerEquivD
* EquivEllAr
* Compactness
* Solidity
* Concavity
* Convexity
* Shape
* RFactor
* ArBBox
* Rectang
* ModRatio
* Sphericity
* Haralick ratio
* Elongation

La prossima tappa sarà fare i test su singole immagini e su più immagini.

# aggiornamento

11-06-2018

## Sistemazione del codice

Il codice ha passato una fase di “ridefinizione” e “abbellimento”. L’eliminazione del metodo che pareva fondamentale per il “run” della classe più esterna lo rende più elegante e snello. Il concetto per cui non si possa fare una classe che estende direttamente ParticleAnalyzer, sfugge ancora.

In questo momento il Plugin, si presenta con due finestre. La prima originale di ParticleAnalyzer. La seconda, del Plugin stesso, nuova e ridefinita per indicare all’utente le nuove misure che saranno implementate.

## Inizio delle misure in grigio

Inizialmente non comprendevo per quale assurdo motivo nelle sezioni di Mode, Max, e Mean non potessi settare una immagine in grigio attraverso il Threshold. Riuscivo solo con le immagini a colori e questo dava molto fastidio. Fino a che non ho approfondito la conoscenza con esso; in passato mi sono lamentata del fatto che bisognasse lasciarlo aperto per poterlo far aprire del PlugIn. Con la chiusura di esso l’immagine diventa in B&W.

# Aggiornamento

13-06-2018

## Il problema del min e del max

Ho riscontrato un problema non banale. Il Plugin non calcolava alcune misure in grey. Dopo una giornata quasi persa nel trovare la soluzione, la conclusione era non usare una funzione aggiuntiva per aggiungere le misure ma implementare nella funzione (attraverso Override) stessa. Evidentemente se trattiamo una immagine in B&W questo vincolo può essere superato facilmente anche perché veniva utilizzato un ciclo for da cui venivano estratti i valori della tabella già esistente e per poter aggiungere formule e dunque misure nuove. A causa di questo inconveniente, sono arrivata alla conclusione che il Plugin ParticleAnalyzer calcola roi per roi, come avevo già sospettato, e alcune misure le calcola attraverso ImageStatistics (che si può creare attraverso la roi stessa) e altre sono implementate nella roi. Per esempio area, min, max ecc sono calcolate da “stats”, oggetto ImageStatistics. Perimeter invece è calcolato con getLeght() di Roi. Dunque la conclusione sarebbe per poter calcolare alcune misure è necessario ricavarsi l’oggetto ImageStatistics da Roi e per altre basta semplicemente Roi. In questo modo ho *elimato* il ciclo for che sicuramente appesantiva il Plugin e usato direttamente le funzioni che mi offrono Roi e ImageStatistics.

## il problema del entropy

Per il calcolo della Entropia mi serviva sapere quante volte un pixel grigio di valore x, si ripetesse all’interno della roi. Dopo una serie di tentativi, scopro;

|  |
| --- |
| **int**[] hist = stats.**histogram**; ImageStatistics |

Dato l’oggetto ImageStatistics collegato alla roi, possiamo ricavarne il numero di volte in cui appare un certo valore del pixel (in grey) attraverso l’histogram.