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 |  |  |
|  |  |  |