Photographing Translucent Biological Samples*

Malaghan Institute of Medical Research

David Eccles, Gringene Bioinformatics

January 2014

Contents

*Document Version 2014-Jul-29-0

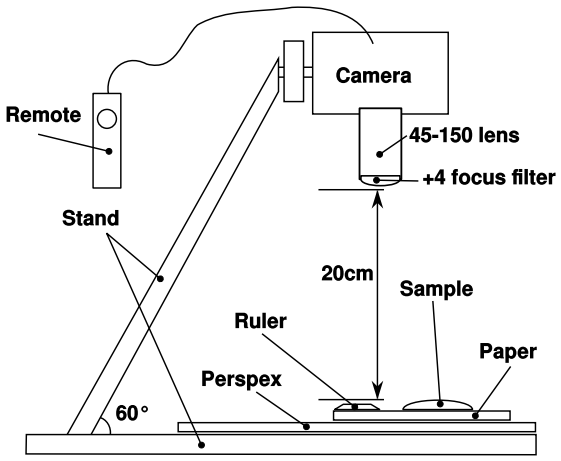

Figure 1: The setup for photography. A LUMIX G VARIO is used with a 45-150 zoom lens, placed about 20cm away from the sample.

1 Imaging Setup

Pictures are taken using the Lumix G Vario camera with 45-150mm lens and attached +4 close focus filter (see Figure 1). The camera is set up on a stand mount to be directly above the sample, at a height of approximately 20cm. A perspex sheet is placed on top of the stand to make it easier to move the sample underneath the camera. A remote trigger is used to reduce camera shake. Ambient diffuse fluorescent lighting is used to provide light to the sample.

The sample is mounted horizontally on blue seed germination paper[1], and a ruler is placed beneath the sample for scale. The camera is set to minimum zoom, or alternatively the highest zoom that still allows focusing (check by listening for the beep when half-pressing the camera shutter button). Because a remote trigger is used to prevent camera shake, a good image can be produced with ISO can be set to as low as possible (ISO 160), with shutter speed and aperture automatically controlled.

2 Stitching Images

When taking images, the stitching process is made simpler by laying out the sample horizontally, and shifting the paper, sample and ruler horizontally from right to left (fixed camera, moving sample). Images should overlap by about 30% so that matching points can be easily found on adjacent images. Small changes in vertical position and/or small rotations should not make the stitching more difficult (see Figure 2).

Stitching is carried out using Hugin. While this program is primarily designed for stitching photos taken from a fixed point (where the camera is kept in the same place and rotated to cover the subject), it has translation options that allow it to work effectively with shifted images.

[1]Anchor Paper, 19"x24" Blue Blotter Paper, cut into A5-sized sheets

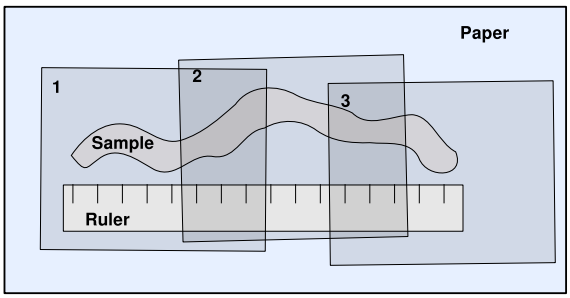

Figure 2: Images are easier to stitch using Hugin when laid out horizontally. The sample is shifted from right to left underneath a fixed camera.

If there are errors in the stitching process, it may be necessary to remove control points from the ruler and stitch based on the sample alone – Hugin can get a bit confused when it sees too many points that look the same.

3 Automatic Scale Calculation

An ImageJ macro has been written to automatically detect a [white] ruler in the image, and calculate a scale. The image is pre-processed by thresholding at 192/255, so that only extreme black/white colour changes are preserved. The process works by creating a line that is one third the image width and sliding it down the centre of the image at intervals of $1/100$ $height$. Once it detects at least 30 dark regions at regular intervals ($CV < 20\%$) along this line profile, it works out the pixel distance between the first and last region, then divides that by the number of regions to get the image resolution in px/mm. The scale calculator can handle small changes in the rotation of the ruler on the image, and rotates the profile line appropriately:

ImageJ Macro

```
function findMaxima(xx, tolerance){
  // based on "official" findMaxima function from ImageJ 1.48c+
  len = xx.length;
  if (len == 0) {
    return newArray();
  }
  if (tolerance < 0) {
    tolerance = 0;
  }
  maxPositions = Array.copy(xx);
  Array.fill(maxPositions,-1);
  max = xx[0];
  min = xx[0];
  maxPos = 0;
  lastMaxPos = -1;
  leftValleyFound = false;
  maxCount = 0;
  for (jj = 1; jj < len; jj++) {
    val = xx[jj];
    if (val > min + tolerance)
      leftValleyFound = true;
    if (val > max && leftValleyFound) {
      max = val;
```

```
24        maxPos = jj;
25      }
26      if (leftValleyFound)
27        lastMaxPos = maxPos;
28      if (val < max - tolerance && leftValleyFound) {
29        maxPositions[maxCount] = maxPos;
30        maxCount++;
31        leftValleyFound = false;
32        min = val;
33        max = val;
34      }
35      if (val < min) {
36        min = val;
37        if (!leftValleyFound)
38          max = val;
39      }
40    }
41    return Array.trim(maxPositions,maxCount);
42  }
43
44  requires("1.44i"); // needed for Array.sort
45
46  run("Select None");
47  ht = getHeight();
48  wd = getWidth();
49  oldName = getInfo("image.filename");
50  fName = "dup_" + oldName;
51  run("Duplicate...", "title=[" + fName + "]");
52  selectWindow(fName);
53  run("Enhance Contrast...", "saturated=0 normalize equalize");
54  run("8-bit");
55  setAutoThreshold("Default");
56  setThreshold(0, 192);
57  setOption("BlackBackground", false);
58  run("Convert to Mask");
59  run("Remove Outliers...", "radius=10 threshold=50 which=Bright");
60  found = false;
61  // Calculate image rotation using vertical profile
62  makeRectangle(wd/3, 100, wd/6, ht-100);
63  setKeyDown("alt");
64  p1 = getProfile();
65  setKeyDown("none");
66  makeRectangle(wd/3+wd/6, 100, wd/6, ht-100);
67  setKeyDown("alt");
68  p2 = getProfile();
69  setKeyDown("none");
70  startGr =  p1[floor(p1.length/100)] > p1[(p1.length-1) - floor(p1.length/100)];
71  ep1 = -1;
72  ep2 = -1;
73  for(i = 0; i < p1.length; i++){
74        edgeValue = 160; // point where ruler edge should be
75        if((ep1 == -1) && ((startGr && p1[i] < edgeValue) ||
76                           (!startGr && p1[i] > edgeValue))){
77          ep1 = i;
78        }
79        if((ep2 == -1) && ((startGr && p2[i] < edgeValue) ||
80                           (!startGr && p2[i] > edgeValue))){
81          ep2 = i;
82        }
83  }
84  ep1 += 10;
85  ep2 += 10;
86  rulerSlope = 0;
87  if((ep1 != -1) && (ep2 != -1)){
88    rulerSlope = (ep2 - ep1) / (wd / 6);
89    makeLine(wd/3+wd/12,ep1,wd/3+wd/6+wd/12,ep2);
```

```
}
// Calculate scale using line profile
for(i = 0; i < ht; i+= ht/100){
  deviation = wd/6 * rulerSlope;
  makeLine(wd/2-wd/6, i+ht/20-deviation, wd/2+wd/6, i+ht/20+deviation);
  profile = getProfile();
  minLocs = findMaxima(profile, 10);
  Array.sort(minLocs);
  diffs = Array.copy(minLocs);
  if(minLocs.length > 1){
    for(j = 0; j < (minLocs.length-1); j++){
      diffs[j] = minLocs[j+1] - minLocs[j];
    }
    diffs = Array.trim(diffs,minLocs.length-1);
    Array.sort(diffs);
    Array.getStatistics(diffs,min,max,mean,std);
    med = diffs[diffs.length/2];
    cv = abs(std / mean);
    //print("i="+i+",n="+diffs.length+",min="+min+",max="+max+",mean="+mean+
    //      ",med="+med+",std="+std+",cv="+cv*100+"%");
    //Array.print(diffs);
    if((cv < 0.2) && (diffs.length > 30)){
      // print("min="+min+",max="+max+",mean="+mean+
      //   ",med="+med+",std="+std+",cv="+cv*100+"%");
      // Array.print(diffs);
      // print(diffs.length + " peaks between " + minLocs[0] +
      //   " and " + minLocs[minLocs.length-1]);
      selectWindow(oldName);
      makeLine(minLocs[0]+wd/2-wd/6, i+ht/20-deviation,
               (minLocs[minLocs.length-1]-minLocs[0])+wd/2-wd/6,
                i+ht/20+deviation);
      run("Set Scale...", "distance="+(minLocs[minLocs.length-1]-minLocs[0])+
        " known="+diffs.length+" pixel=1 unit=mm");
      selectWindow(fName);
      found = true;
      i += ht; // end outer loop
    }
  }
  //wait(500);
}
close();
if(!found){
  exit("Error: Cannot determine scale, ruler could not be found. " +
    "Are you using a colour image with a horizontal white ruler?");
}
```

4 Background Subtraction

After the image is taken and stitched (see Figure 3), the background is removed from the image. By using a blue background, the amount of useful information that is removed in the background subtraction process is substantially reduced. The process of background subtraction is carried out in ImageJ as follows:

1. Split the image into Red, Green, and Blue channels
2. Normalise each channel to reduce colour split effects
3. combine and Z-project the Red and Green channels using average intensity [optional]
4. Use the Image Calculator to subtract the Blue channel from either the Red channel or

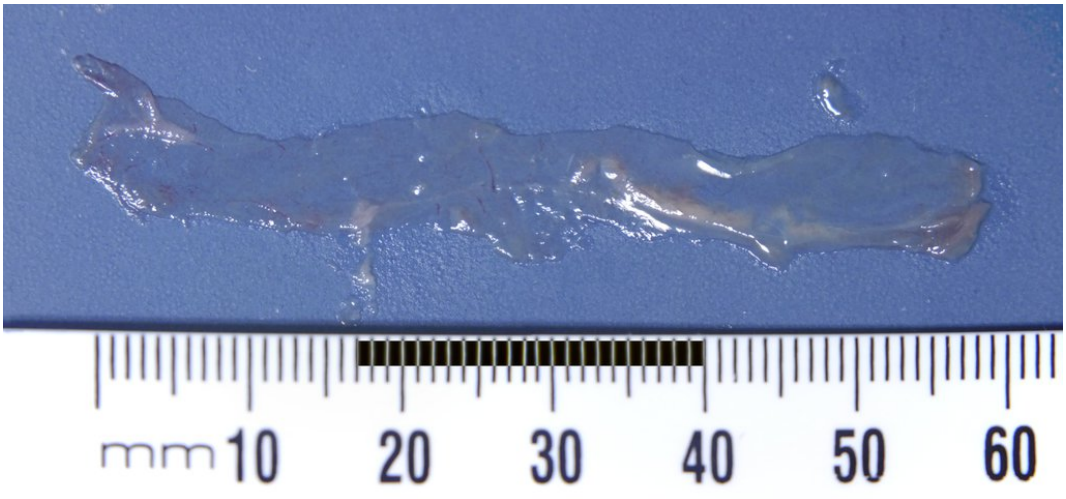

Figure 3: An image of a normal sample that has been stitched using Hugin. The region identified as containing a ruler section for calculating the scale (see Section 3) has been inverted in this image.

the combined Red+Green channel

A short ImageJ macro has been written to carry out this background subtraction process on an arbitrary image. The result of this proces on Figure 3 can be seen in Figure 4:

ImageJ Macro

```
1  // subtract the blue channel from an image
2  run("Select None");
3  oldName = getInfo("image.filename");
4  fName = "dup_" + oldName;
5  run("Duplicate...", "title=[" + fName + "]");
6  selectWindow(fName);
7  run("Split Channels");
8  selectWindow(fName + " (green)");
9  close();
10 selectWindow(fName + " (red)");
11 run("Enhance Contrast...", "saturated=0 normalize equalize");
12 selectWindow(fName + " (blue)");
13 run("Enhance Contrast...", "saturated=0 normalize equalize");
14 imageCalculator("Subtract create", fName + " (red)",fName + " (blue)");
15 selectWindow(fName + " (red)");
16 close();
17 selectWindow(fName + " (blue)");
18 close();
19 selectWindow("Result of " + fName + " (red)");
20 rename("R-B_" + oldName);
```

5 Feature Extraction

An ImageJ macro has been written to extract features from a background-subtracted image that look similar to tumours. The extraction process is roughly the following:

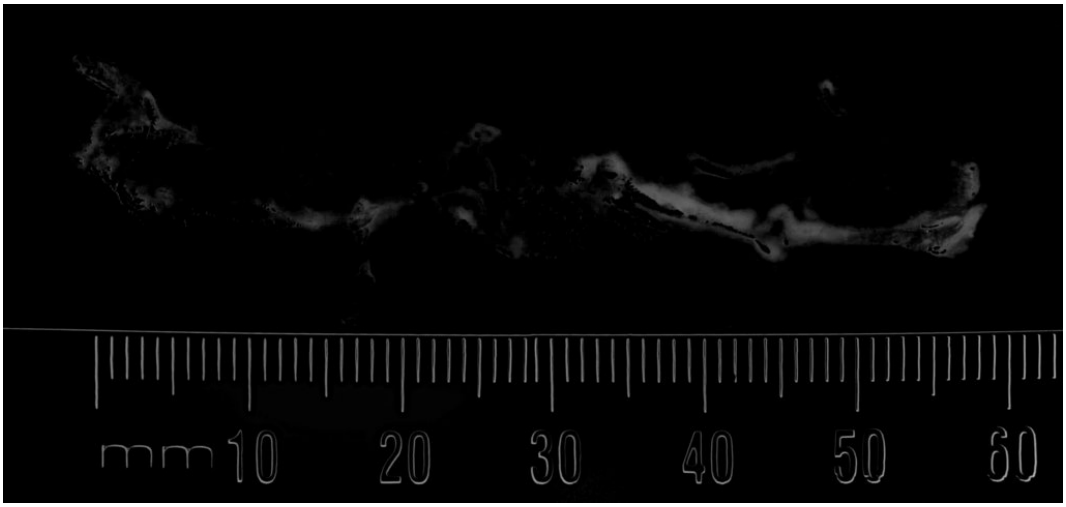

Figure 4: A sample with blue background subtracted from the image

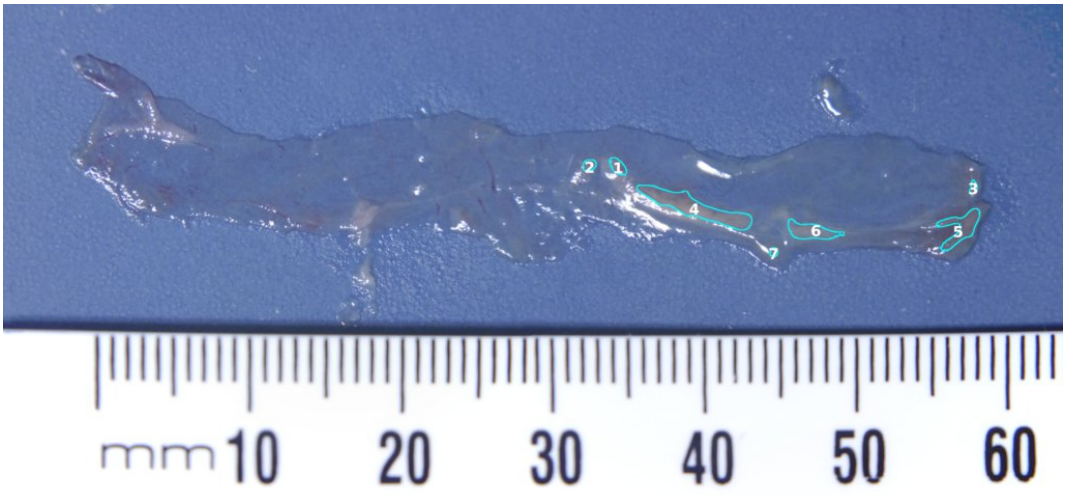

Figure 5: A sample with tumor-like regions identified by a feature extraction macro

1. Threshold the image to black/white
2. Despeckle to remove noisy areas
3. Remove dark outliers (larger noisy bits outside blobs)
4. Remove light outliers (light areas inside blobs, e.g. bubbles, light reflections)
5. Apply a watershed filter to separate joined blobs
6. Select remaining regions, then measure those regions on the original background-subtracted image

Results relating to the shape and appearance of the features are stored in the ImageJ results table. The features are outlined and transferred back to the original (non-subtracted) image as an overlay.

The result of this process on Figure 3 can be seen in Figure 5. Note that this was a normal sample, but tumour-like features were detected in it.

ImageJ Macro

```
requires("1.47o");

oldName = getInfo("image.filename");
oldDir = getDirectory("image");

if(oldDir == ""){
  oldDir = getDirectory("home");
}

if(selectionType != 0){
  run("Select None");
}
getSelectionBounds(selX, selY, selW, selH);

if(!is("grayscale")){
  // background subtraction hasn't been carried out first, so do that
  run("Autoscale");
  run("SubtractBlue");
} else {
  oldName = replace(oldName,"^R-B-","");
}

getPixelSize(unit, pw, ph, pd);
if(unit != "mm"){
  exit("Error: scale is not in millimetres (mm). " +
    "Has the scale been set?");
}

selectWindow("R-B-" + oldName);
greyName = "grey-" + oldName;
run("Duplicate...", "title=[" + greyName + "]");

wait(100);
selectWindow("R-B-" + oldName);
setAutoThreshold("Default bright");
setOption("BlackBackground", false);
run("Convert to Mask");
run("Invert");
run("Despeckle");
run("Remove Outliers...", "radius=25 threshold=50 which=Dark");
run("Remove Outliers...", "radius=25 threshold=50 which=Bright");
run("Watershed");
```

```
43 run("Set Measurements...", "area mean standard modal min centroid center perimeter "+
44     "bounding fit shape feret's integrated median skewness kurtosis display "+
45     "redirect=None decimal=3");
46 makeRectangle(selX, selY, selW, selH);
47 run("Clear Results");
48 run("Analyze Particles...", "size=0.20-Infinity circularity=0-1.00 "+
49     "show=[Overlay Outlines] display exclude clear");
50 if(Overlay.size > 0){
51   Overlay.copy();
52 //  close("R-B-" + oldName);
53   selectWindow(oldName);
54   Overlay.paste();
55   // re-analyse particle regions using greyscale image
56   selectWindow(greyName);
57   Overlay.paste();
58   run("To ROI Manager");
59   run("Clear Results");
60   selectWindow("Results");
61   roiManager("Measure");
62   selectWindow("ROI Manager");
63   run("Close");
64 //  close(greyName);
65   wait(100);
66   selectWindow("Results");
67   oldBase = replace(oldName,"(\\.tiff?|\\.jpe?g)$","");
68   for (i = 0; i < nResults; i++) {
69     setResult("FileName",i,oldBase);
70     setResult("BoundX",i,selX);
71     setResult("BoundY",i,selY);
72     setResult("BoundW",i,selW);
73     setResult("BoundH",i,selH);
74     setResult("CountMethod",i,"Auto");
75     setResult("Location",i,"");
76     if(startsWith(oldBase,"SI") || startsWith(oldBase,"Small")){
77       setResult("Location",i,"SI");
78     }
79     if(startsWith(oldBase,"LI") || startsWith(oldBase,"Large")){
80       setResult("Location",i,"LI");
81     }
82     setResult("TumourID",i,i+1);
83     setResult("Tumour",i,"");
84     setResult("Comments",i,"");
85     setResult("Label",i,i+1);
86   }
87   setOption("ShowRowNumbers", false);
88   updateResults();
89   if(!File.exists(oldDir + oldBase + "_macroOutput.csv")){
90     saveAs("results",oldDir + oldBase + "_macroOutput.csv");
91   } else {
92     increment = 1;
93     while(File.exists(oldDir + oldBase + "_macroOutput." +
94                       increment + ".csv")){
95       increment++;
96     }
97     saveAs("results",oldDir + oldBase + "_macroOutput." +
98            increment + ".csv");
99   }
100 } else { // no overlay, so don't try to copy it
101 //  close("R-B-" + oldName);
102 //  close(greyName);
103   selectWindow(oldName);
104   showMessage("Blob analysis is finished; no suitable blobs were found.");
105 }
```