

Image Processing

CS 110

Medical Images

 The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

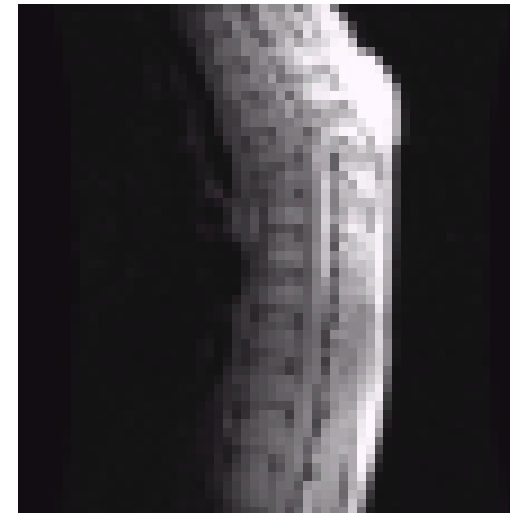
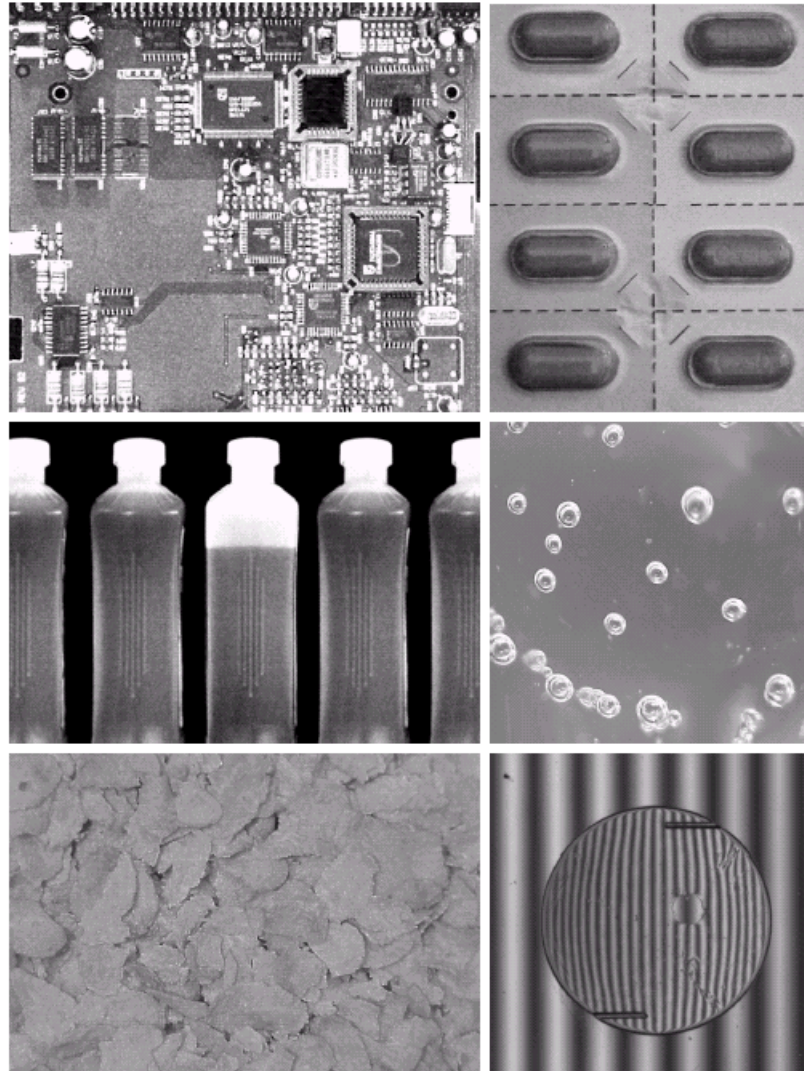


Image Processing in Manufacturing

a b
c d
e f

FIGURE 1.14

Some examples of manufactured goods often checked using digital image processing. (a) A circuit board controller. (b) Packaged pills. (c) Bottles. (d) Bubbles in clear-plastic product. (e) Cereal. (f) Image of intraocular implant. (Fig. (f) courtesy of Mr. Pete Sites, Perceptics Corporation.)



What can you do with Image Processing?

Inspect, Measure, and Count using Photos and Video

<http://www.youtube.com/watch?v=KsTtNWVhpgI>

Image Processing Software

<http://www.youtube.com/watch?v=1WJp9mGnWSM>

Thresholding for Image Segmentation

- Pixels below a cutoff value are set to black
- Pixels above a cutoff value are set to white

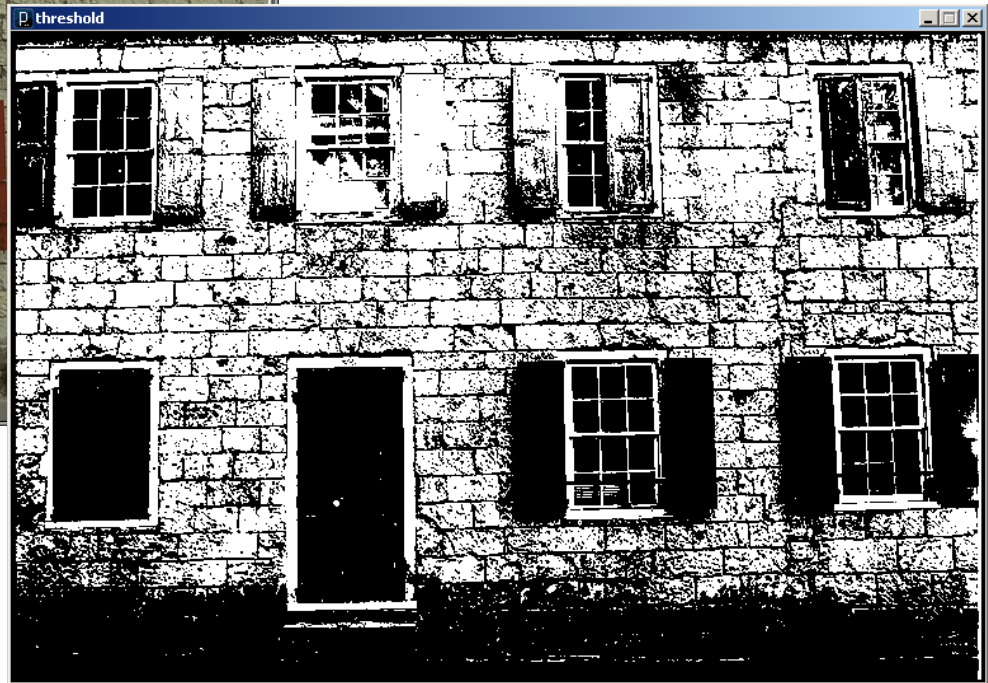
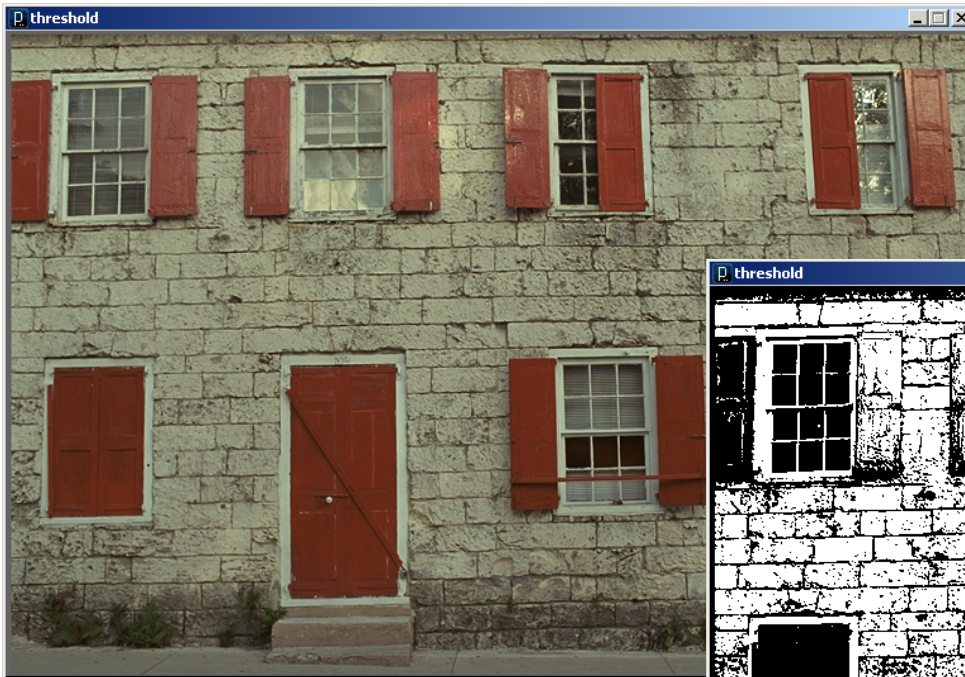
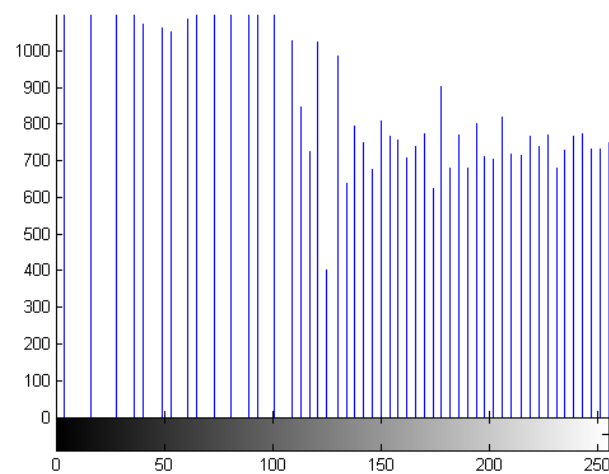
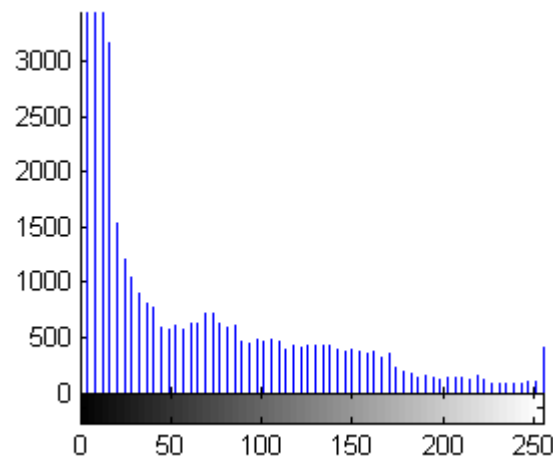


Image Enhancement

- Color and intensity adjustment
- Histogram equalization



Implementing a Color Histogram in Processing

```
// Histogram
```

```
// Arrays to hold histogram values
int[] aa = new int[256];
int[] ra = new int[256];
int[] ga = new int[256];
int[] ba = new int[256];
```

```
PImage img;
```

```
void setup() {
  size(516, 516);
  img = loadImage("kodim02.png");
  img.loadPixels();
```

```
// Sum up all pixel values
for (int i=0; i<img.pixels.length; i++) {
  float r = red(img.pixels[i]);
  float g = green(img.pixels[i]);
  float b = blue(img.pixels[i]);
```

```
// Increment histogram item amounts
ra[ int(r) ]++;
ga[ int(g) ]++;
ba[ int(b) ]++;
aa[ int((r+g+b)/3.0) ]++;
}
```

```
// Find max value
float max = 0.0;
for (int i=0; i<256; i++) {
  if (ra[i] > max) max = ra[i];
  if (ga[i] > max) max = ga[i];
  if (ba[i] > max) max = ba[i];
  if (aa[i] > max) max = aa[i];
}
```

```
// Draw scaled histogram
background(255);
noFill();
```

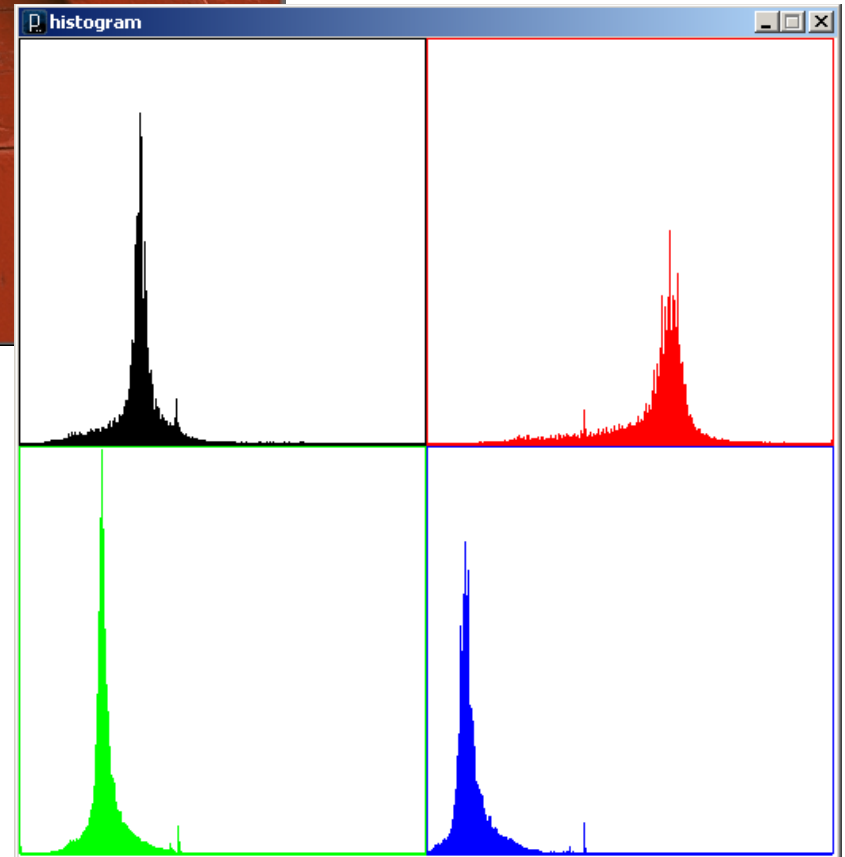
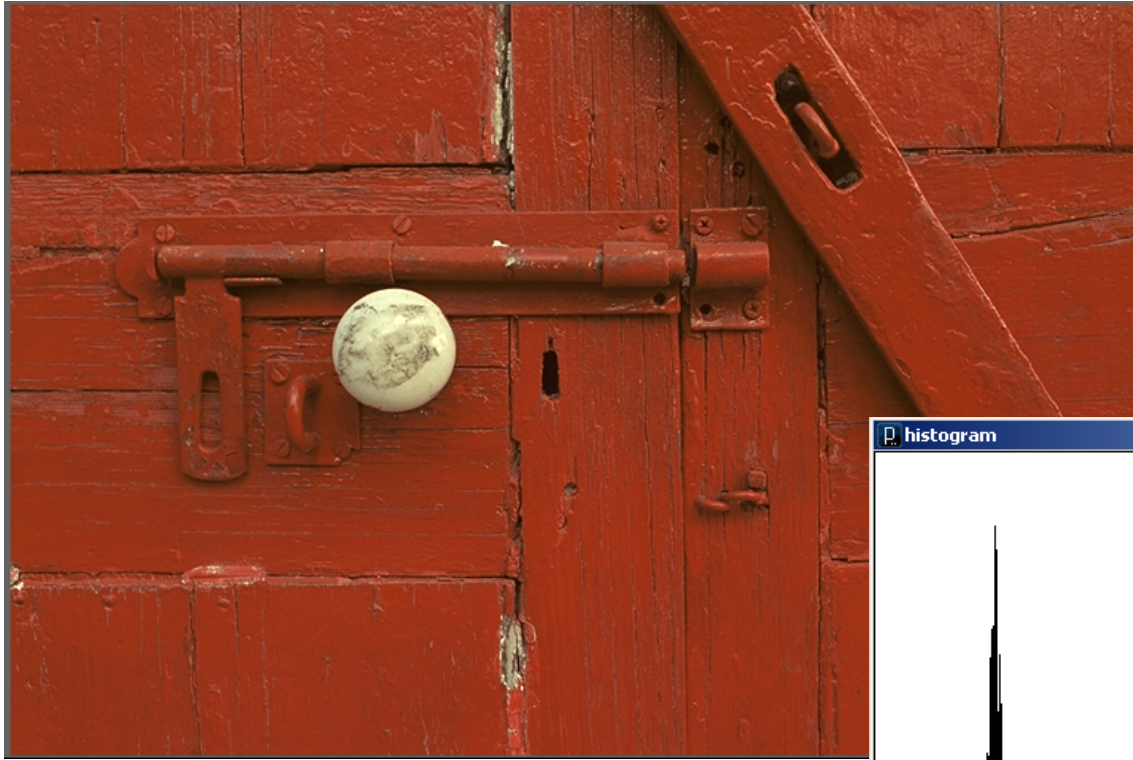
```
// Borders
stroke(0);
rect(0, 0, 256, 256);
stroke(255,0,0);
rect(257, 0, 256, 256);
stroke(0,255,0);
rect(0, 257, 256, 256);
stroke(0,0,255);
rect(257, 257, 256, 256);
```

```
// Lines
float h;
for (int i=0; i<256; i++) {
  // all
  stroke(0);
  h = map(aa[i], 0, max, 0, 255);
  line(i, 255, i, 255-h);
```

```
// red
stroke(255,0,0);
h = map(ra[i], 0, max, 0, 255);
line(257+i, 255, 257+i, 255-h);
```

```
// green
stroke(0,255,0);
h = map(ga[i], 0, max, 0, 255);
line(i+1, 514, i+1, 514-h);
```

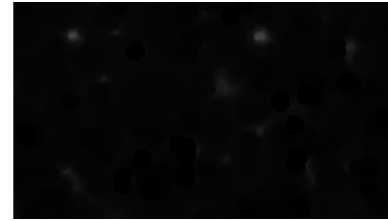
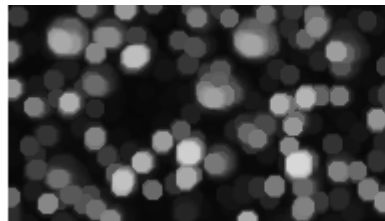
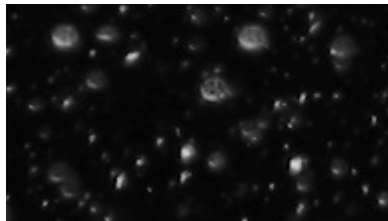
```
// blue
stroke(0,0,255);
h = map(ba[i], 0, max, 0, 255);
line(257+i, 514, 257+i, 514-h);
}
}
```



Feature Extraction

- Region detection – morphology manipulation

- Dilate and Erode



- Open

- Erode → dilate

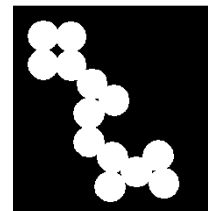
- Small objects are removed



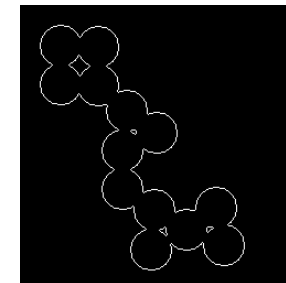
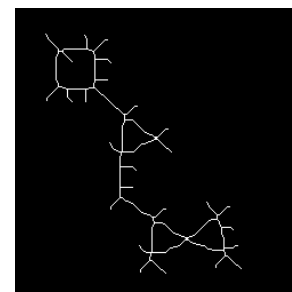
- Close

- Dilate → Erode

- Holes are closed



- Skeleton and perimeter



Erode + Dilate to Despeckle

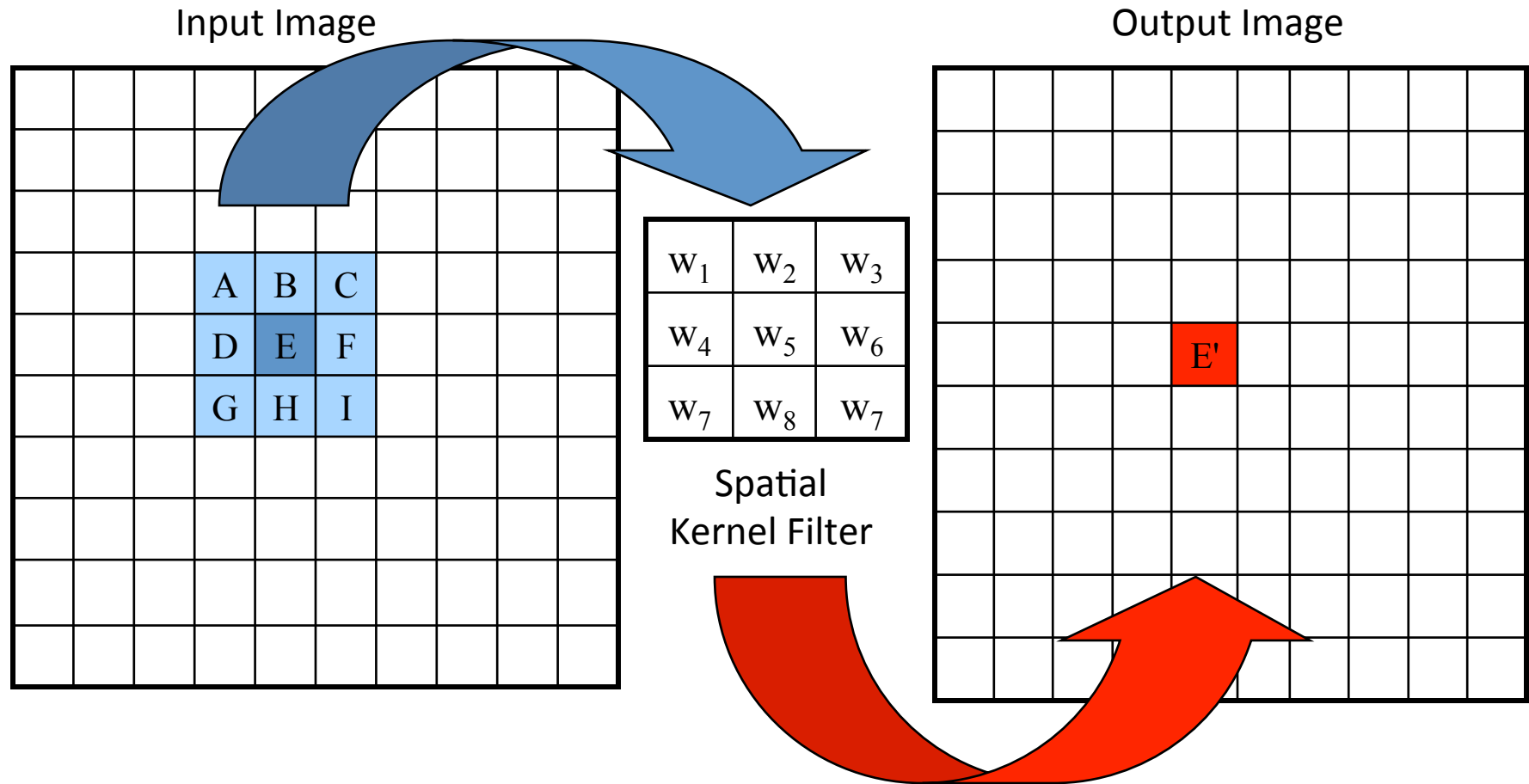


Erode



Dilate

Spatial Filtering



$$E' = w_1A + w_2B + w_3C + w_4D + w_5E + w_6F + w_7G + w_8H + w_7I$$

Image Enhancement

- Denoise

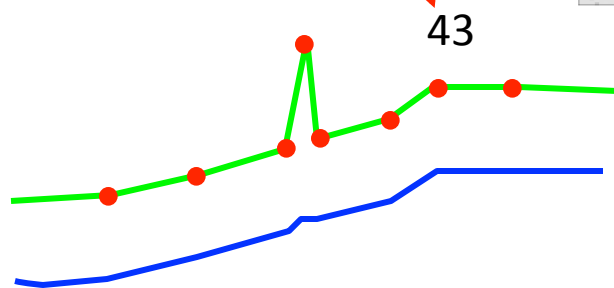
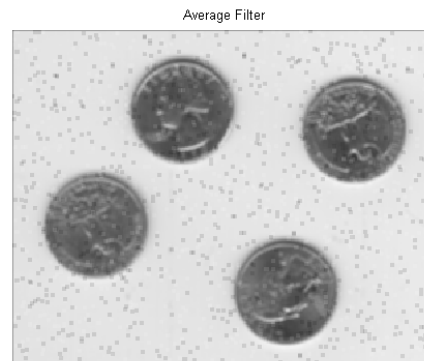
- Averaging

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9



- Median filter

20	5	43
78	3	22
115	189	200



// Spatial Filtering

```
PImage img;  
PImage filt;  
int w = 100;  
int msize = 3;
```

// Sharpen

```
float[][] matrix = {{ -1., -1., -1.},  
                    { -1., 9., -1.},  
                    { -1., -1., -1.}};
```

// Laplacian Edge Detection

```
//float[][] matrix = {{ 0., 1., 0. },  
//                    { 1., -4., 1. },  
//                    { 0., 1., 0. }};
```

// Average

```
//float[][] matrix = {{ 1./9., 1./9., 1./9.},  
//                    { 1./9., 1./9., 1./9.},  
//                    { 1./9., 1./9., 1./9.}};
```

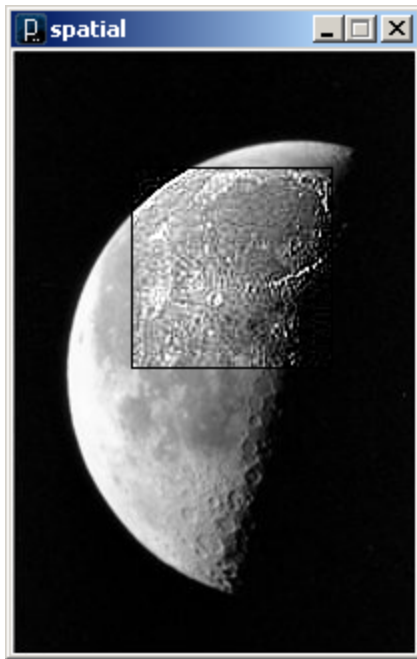
// Gaussian Blur

```
//float[][] matrix = {{ 1./16., 2./16., 1./16. },  
//                    { 2./16., 4./16., 2./16. },  
//                    { 1./16., 2./16., 1./16. }};
```

```
void setup() {  
  //img = loadImage("bmc3.jpg");  
  img = loadImage("moon.jpg");  
  size( img.width, img.height );  
  filt = createImage(w, w, RGB);  
}
```

```
void draw() {  
  // Draw the image on the background  
  image(img,0,0);  
  
  // Get current filter rectangle location  
  int xstart =  
    constrain(mouseX-w/2,0,img.width);  
  int ystart =  
    constrain(mouseY-w/2,0,img.height);  
  
  // Filter rectangle  
  loadPixels();  
  filt.loadPixels();  
  
  for (int i=0; i<w; i++) {  
    for (int j=0; j<w; j++) {  
      int x = xstart + i;  
      int y = ystart + j;  
      color c =  
        spatialFilter(x, y, matrix, msize, img);  
      int loc = i+j*w;  
      filt.pixels[loc] = c;  
    }  
  }  
  
  filt.updatePixels();  
  updatePixels();  
  
  // Add rectangle around convolved region  
  stroke(0);  
  noFill();  
  image(filt, xstart, ystart);  
  rect(xstart, ystart, w, w);  
}
```

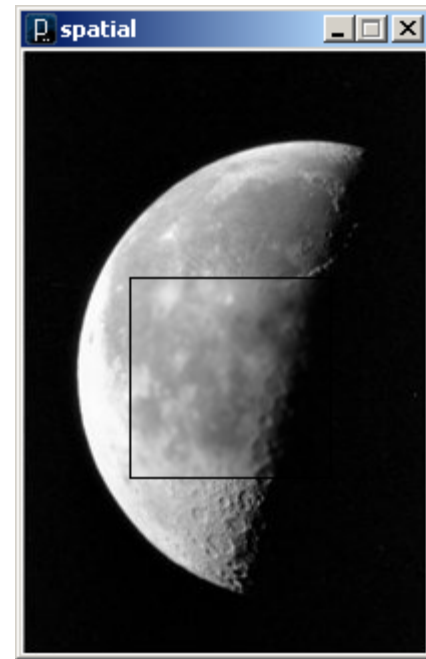
```
// Perform spatial filtering on one pixel location  
color spatialFilter(int x, int y, float[][] matrix,  
                    int msize, PImage img) {  
  float rtotal = 0.0;  
  float gtotal = 0.0;  
  float bttotal = 0.0;  
  int offset = msize/2;  
  
  // Loop through filter matrix  
  for (int i=0; i<msize; i++) {  
    for (int j=0; j<msize; j++) {  
  
      // What pixel are we testing  
      int xloc = x+i-offset;  
      int yloc = y+j-offset;  
      int loc = xloc + img.width*yloc;  
  
      // Make sure we haven't walked off  
      // the edge of the pixel array  
      loc = constrain(loc,0,img.pixels.length-1);  
  
      // Calculate the filter  
      rttotal += (red(img.pixels[loc]) * matrix[i][j]);  
      gtotal += (green(img.pixels[loc]) * matrix[i][j]);  
      bttotal += (blue(img.pixels[loc]) * matrix[i][j]);  
    }  
  }  
  
  // Make sure RGB is within range  
  rttotal = constrain(rtotal,0,255);  
  gtotal = constrain(gtotal,0,255);  
  bttotal = constrain(bttotal,0,255);  
  
  // return resulting color  
  return color(rtotal, gtotal, bttotal);  
}
```



Sharpen



Edge
Detection



Gaussian
Blur

Image Processing in Processing

tint() modulate individual color components

blend() combine the pixels of two images in a given manner

filter() apply an image processing algorithm to an image

Blend Command

```
img = loadImage("colony.jpg");  
mask = loadImage("mask.png");  
image(img, 0, 0);  
blend(mask, 0, 0, mask.width, mask.height,  
0, 0, img.width, img.height, SUBTRACT);
```

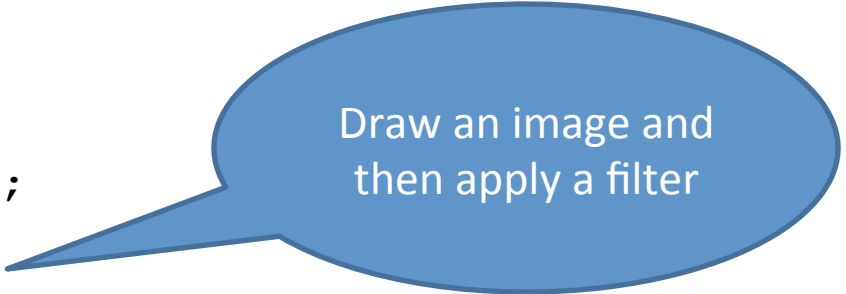


Draw an image and
then blend with
another image

BLEND	linear interpolation of colours:	$C = A * \text{factor} + B$
ADD	additive blending with white clip:	$C = \min(A * \text{factor} + B, 255)$
SUBTRACT	subtractive blending with black clip:	$C = \max(B - A * \text{factor}, 0)$
DARKEST	only the darkest colour succeeds:	$C = \min(A * \text{factor}, B)$
LIGHTEST	only the lightest colour succeeds:	$C = \max(A * \text{factor}, B)$
DIFFERENCE	subtract colors from underlying image.	
EXCLUSION	similar to DIFFERENCE, but less extreme.	
MULTIPLY	Multiply the colors, result will always be darker.	
SCREEN	Opposite multiply, uses inverse values of the colors.	
OVERLAY	A mix of MULTIPLY and SCREEN. Multiplies dark values, and screens light values.	
HARD_LIGHT	SCREEN when greater than 50% gray, MULTIPLY when lower.	
SOFT_LIGHT	Mix of DARKEST and LIGHTEST. Works like OVERLAY, but not as harsh.	
DODGE	Lightens light tones and increases contrast, ignores darks.	
BURN	Darker areas are applied, increasing contrast, ignores lights.	

Filter Command

```
PImage b;  
b = loadImage("myImage.jpg");  
image(b, 0, 0);  
filter(THRESHOLD, 0.5);
```



Draw an image and
then apply a filter

- THRESHOLD** converts the image to black and white pixels depending if they are above or below the threshold defined by the level parameter. The level must be between 0.0 (black) and 1.0(white). If no level is specified, 0.5 is used.
- GRAY** converts any colors in the image to grayscale equivalents
- INVERT** sets each pixel to its inverse value
- POSTERIZE** limits each channel of the image to the number of colors specified as the level parameter
- BLUR** executes a Gaussian blur with the level parameter specifying the extent of the blurring. If no level parameter is used, the blur is equivalent to Gaussian blur of radius 1.
- OPAQUE** sets the alpha channel to entirely opaque.
- ERODE** reduces the light areas with the amount defined by the level parameter.
- DILATE** increases the light areas with the amount defined by the level parameter.

```
// Threshold
```

```
PImage img;
```

```
void setup() {
```

```
  img = loadImage("kodim01.png");
```

```
  size(img.width, img.height);
```

```
  image(img, 0, 0);
```

```
}
```

```
void draw() {}
```

```
void drawImg(float thresh) {
```

```
  image(img, 0, 0);
```

```
  filter(THRESHOLD, thresh);
```

```
}
```

```
void mouseDragged() {
```

```
  float thresh = map(mouseY, 0, height, 0.0, 1.0);
```

```
  println(thresh);
```

```
  drawImg(thresh);
```

```
}
```

```
// Posterize
PImage img;

void setup() {
  img = loadImage("andy-warhol2.jpg");
  size(img.width, img.height);
  image(img, 0, 0);
}

void draw() {}

void drawImg(float val {
  image(img, 0, 0);
  filter(POSTERIZE, val);
}

void mouseDragged() {
  float val = int(map(mouseY, 0, height, 2, 10));
  val = constrain(val, 2, 10);
  println(val);
  drawImg(val);
}
```



Image Processing Applications

Manual Colony Counter

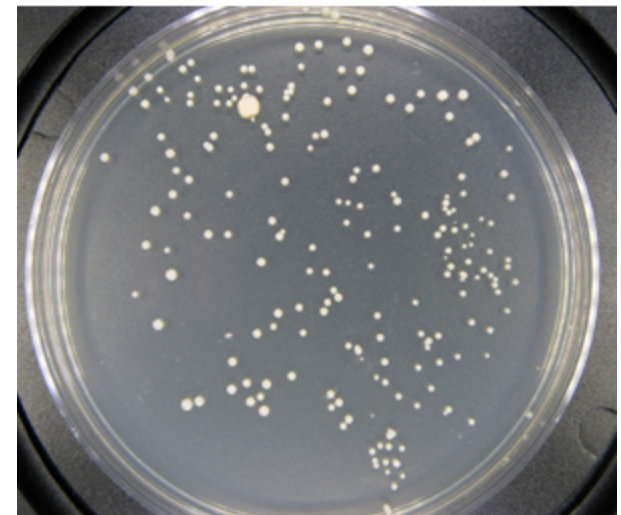
<http://www.youtube.com/watch?v=7B-9Wf6pENQ>

Automated Colony counter

<http://www.youtube.com/watch?v=qtJmQqRHHag>

Measuring Confluency in Cell Culture Biology

- Refers to the coverage of a dish or flask by the cells
- 100% confluency = completely covered
- Image Processing Method
 1. Mask off unimportant parts of image
 2. Threshold image
 3. Count pixels of certain color

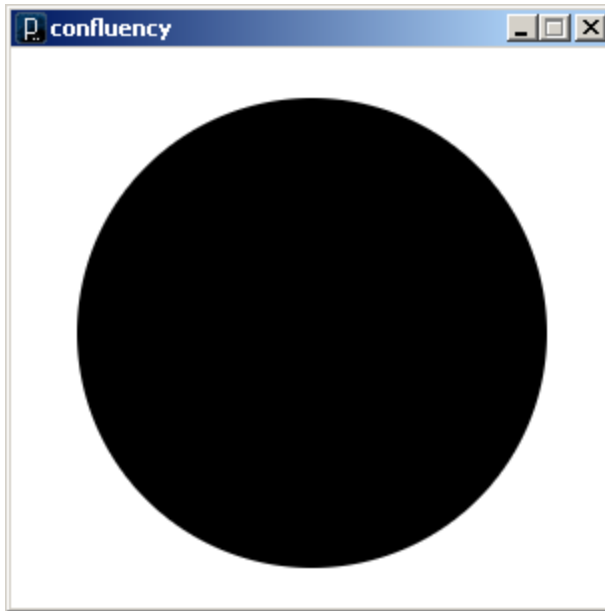


Blend: Subtract

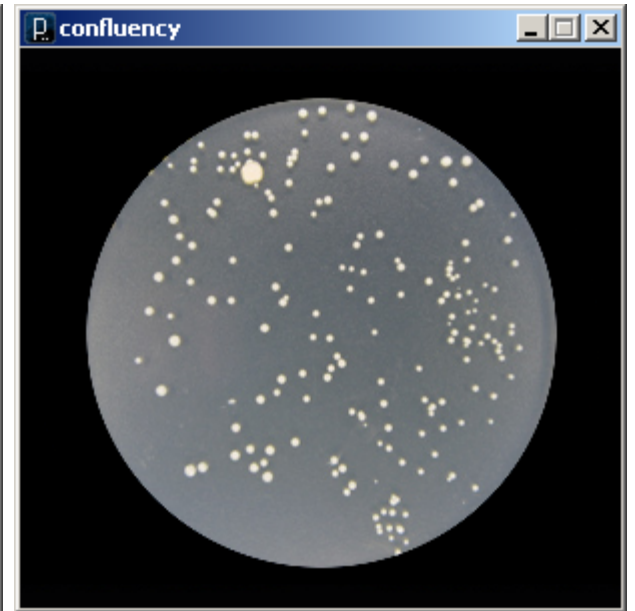
Original



Mask

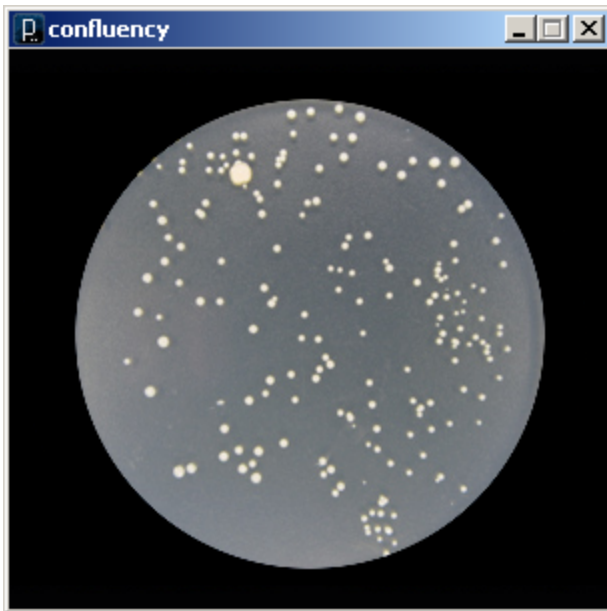


Subtracted

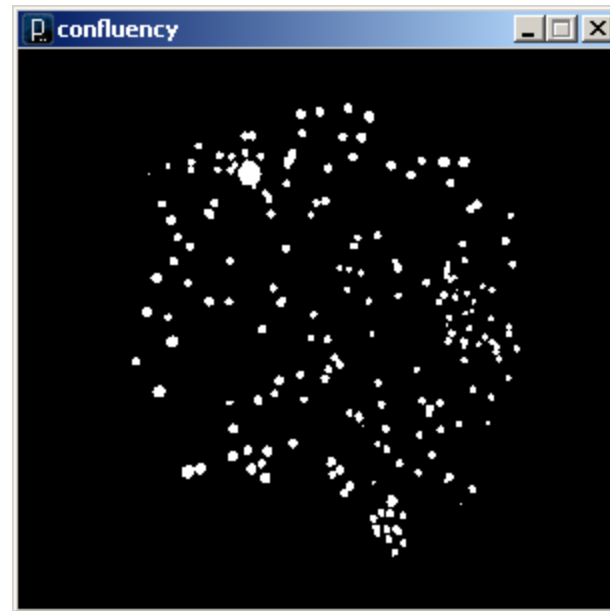


Filter: Theshold

Subtracted



Threshold



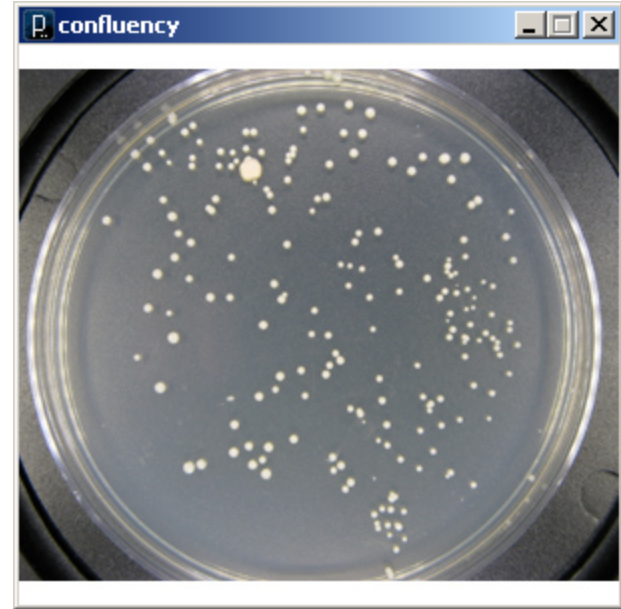
Count Fraction of Pixels to Quantitate

```
// Colony Confluency
PImage img;
PImage mask;

void setup() {
  img = loadImage("colony.jpg");
  mask = loadImage("mask.png");
  size(img.width, img.height);
}

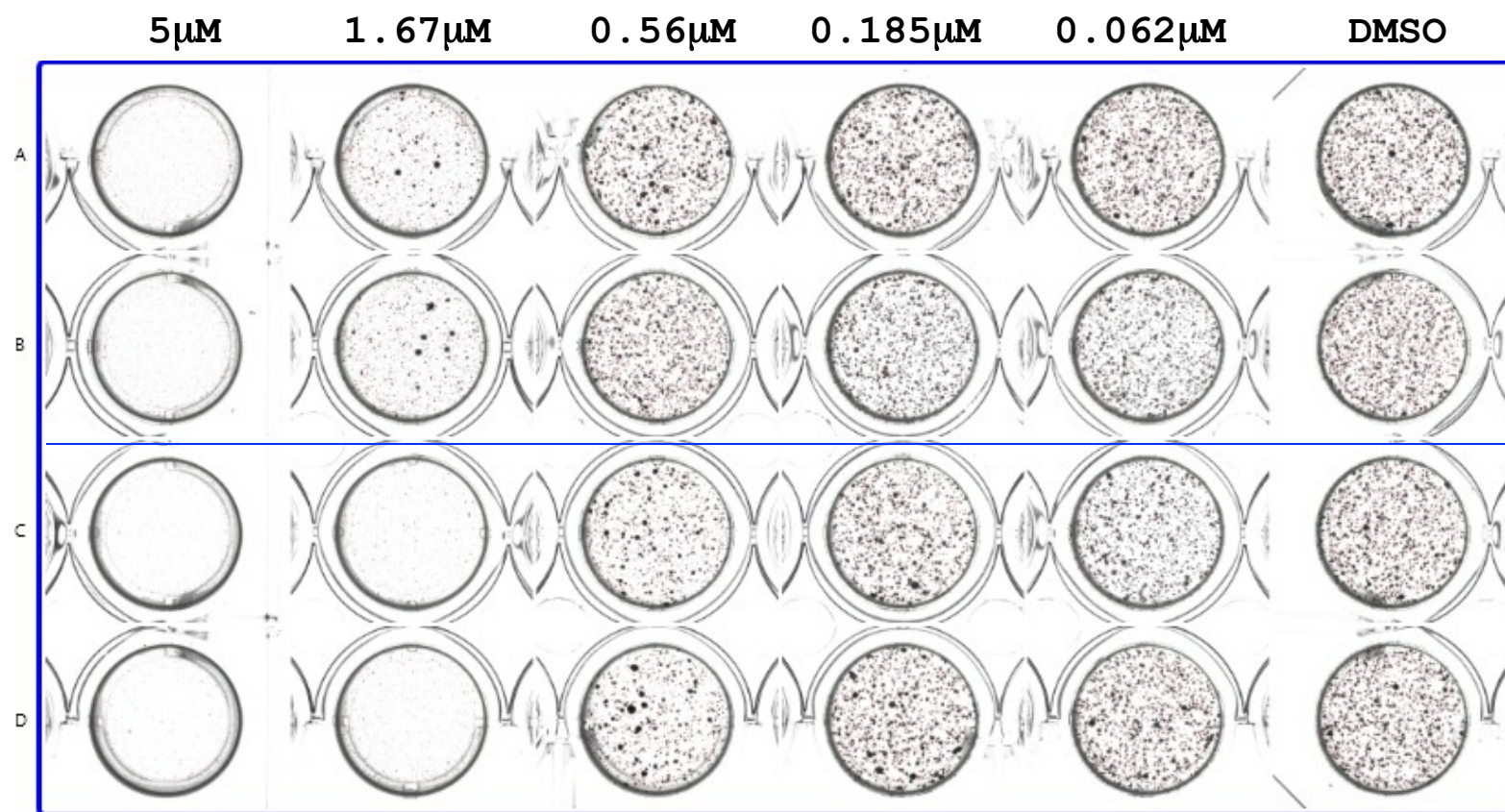
void draw() {
  image(img, 0, 0);
  blend(mask, 0, 0, mask.width, mask.height,
        0, 0, img.width, img.height, SUBTRACT);
  filter(THRESHOLD, 0.6);
}

void mousePressed() {
  loadPixels();
  int count = 0;
  for (int i=0; i<pixels.length; i++)
    if (red(pixels[i]) == 255) count++;
  println(count/42969.0);
}
```



5.3 % Confluency

IC₅₀ determination



Vision Guided Robotics

Colony Picking

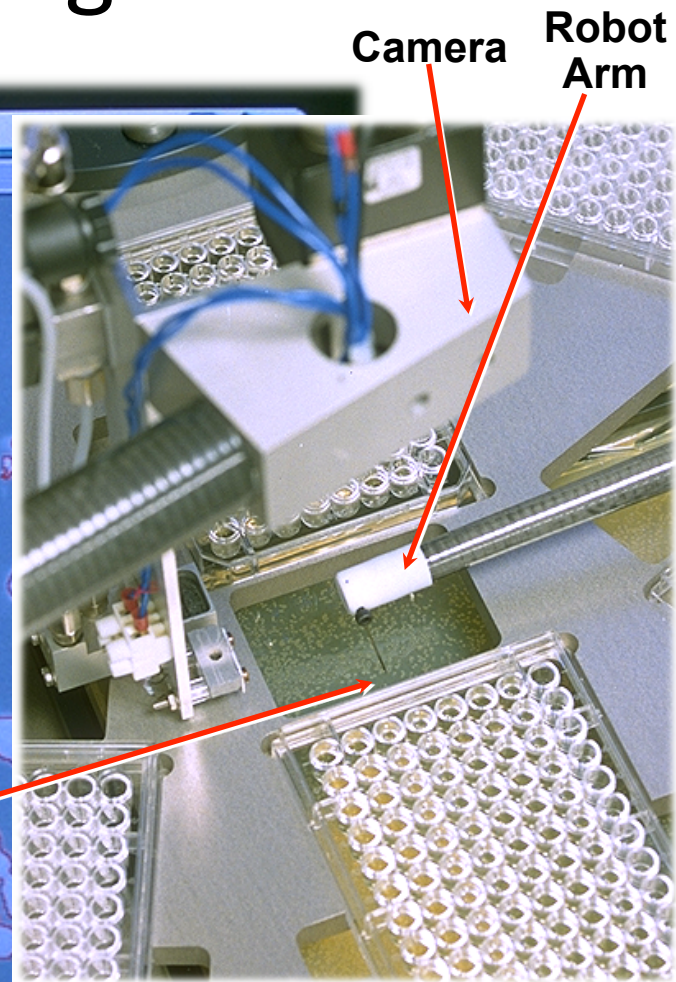
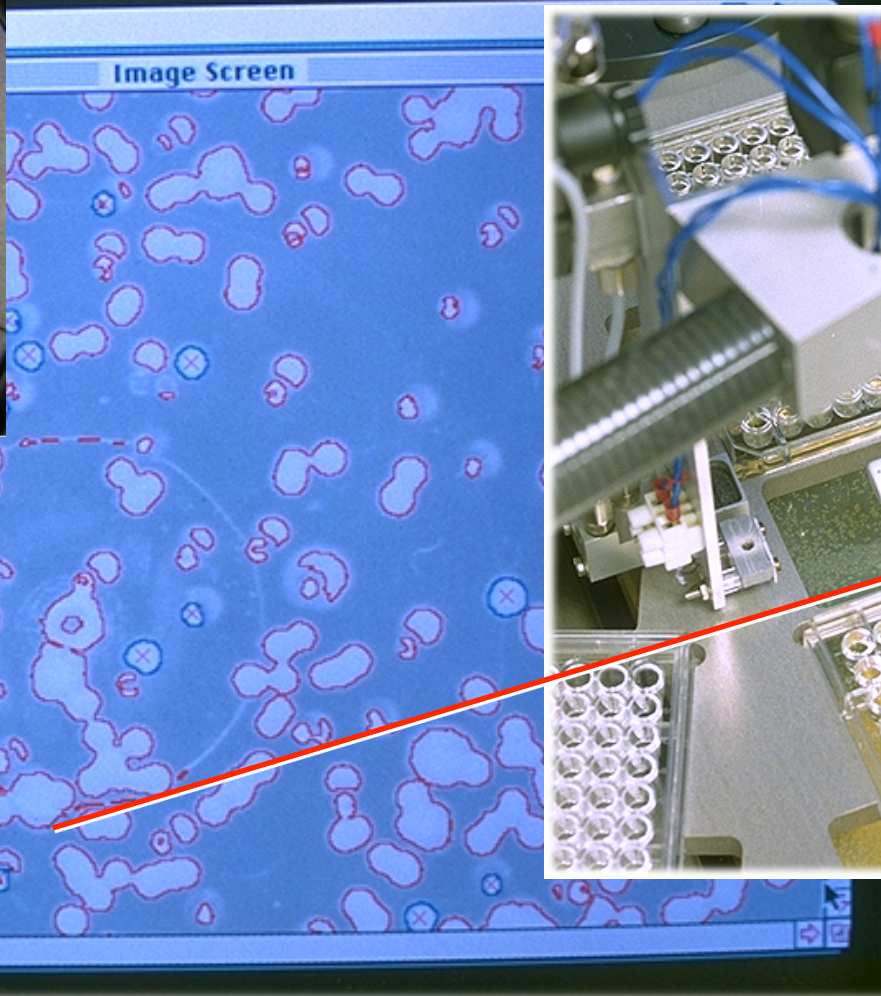
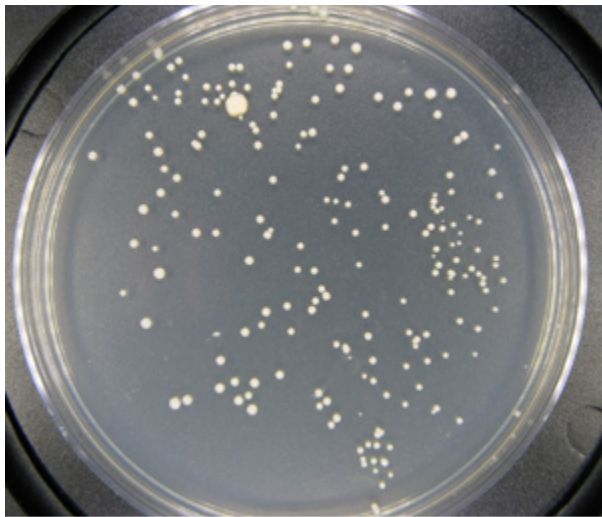
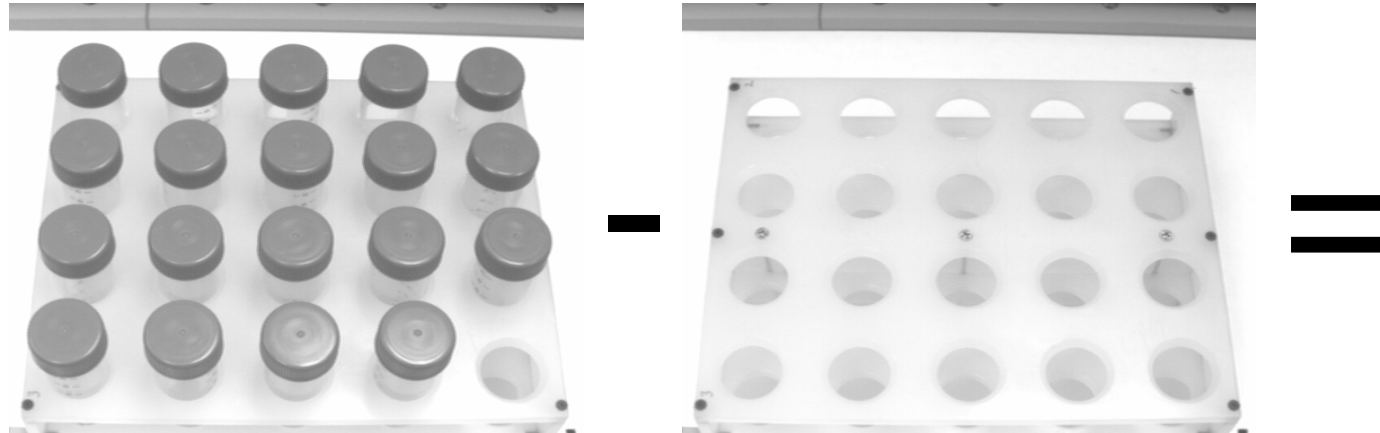


Image Processing



Compute the
presence of objects
or “particles”



Image Processing

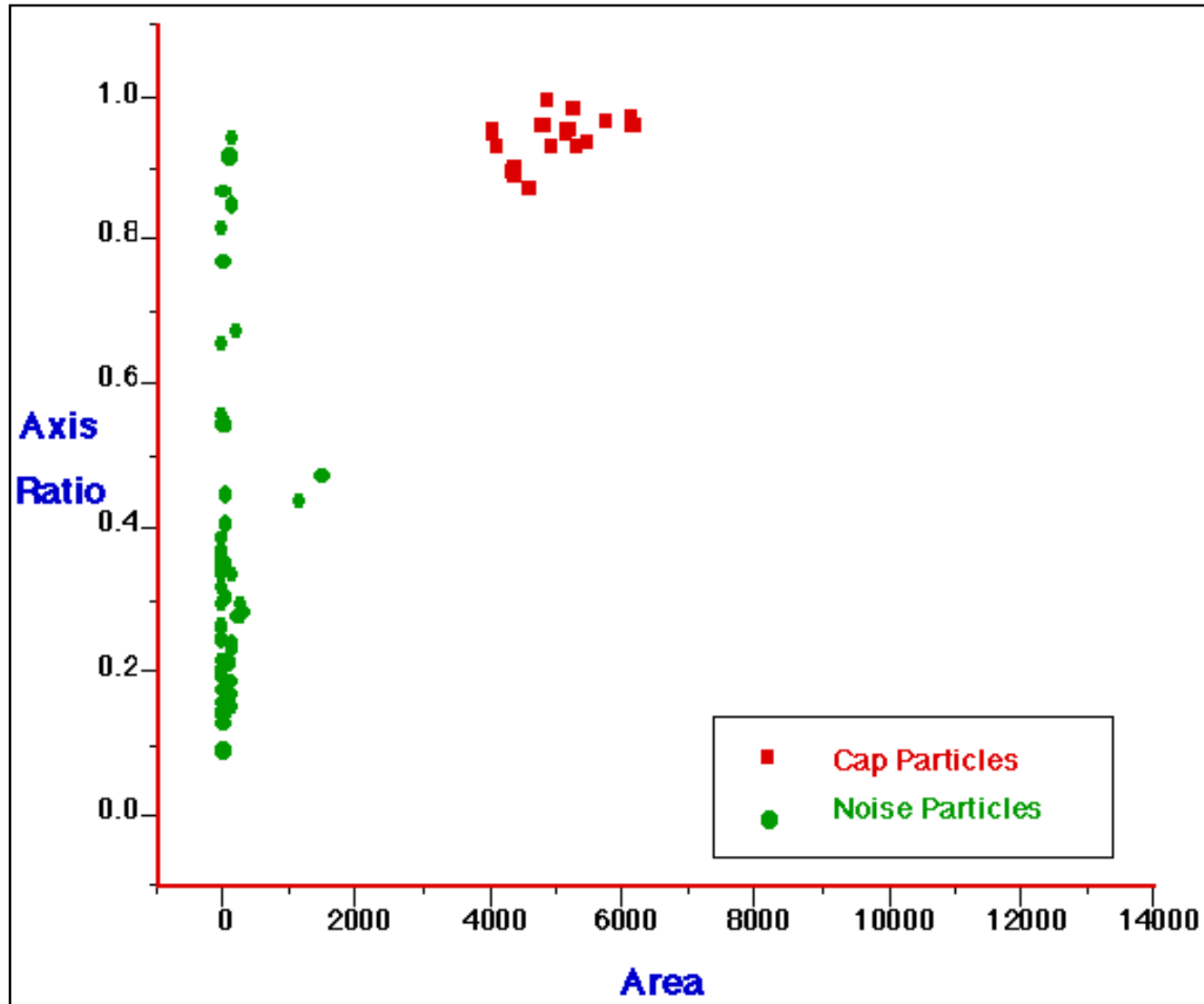


Image Processing

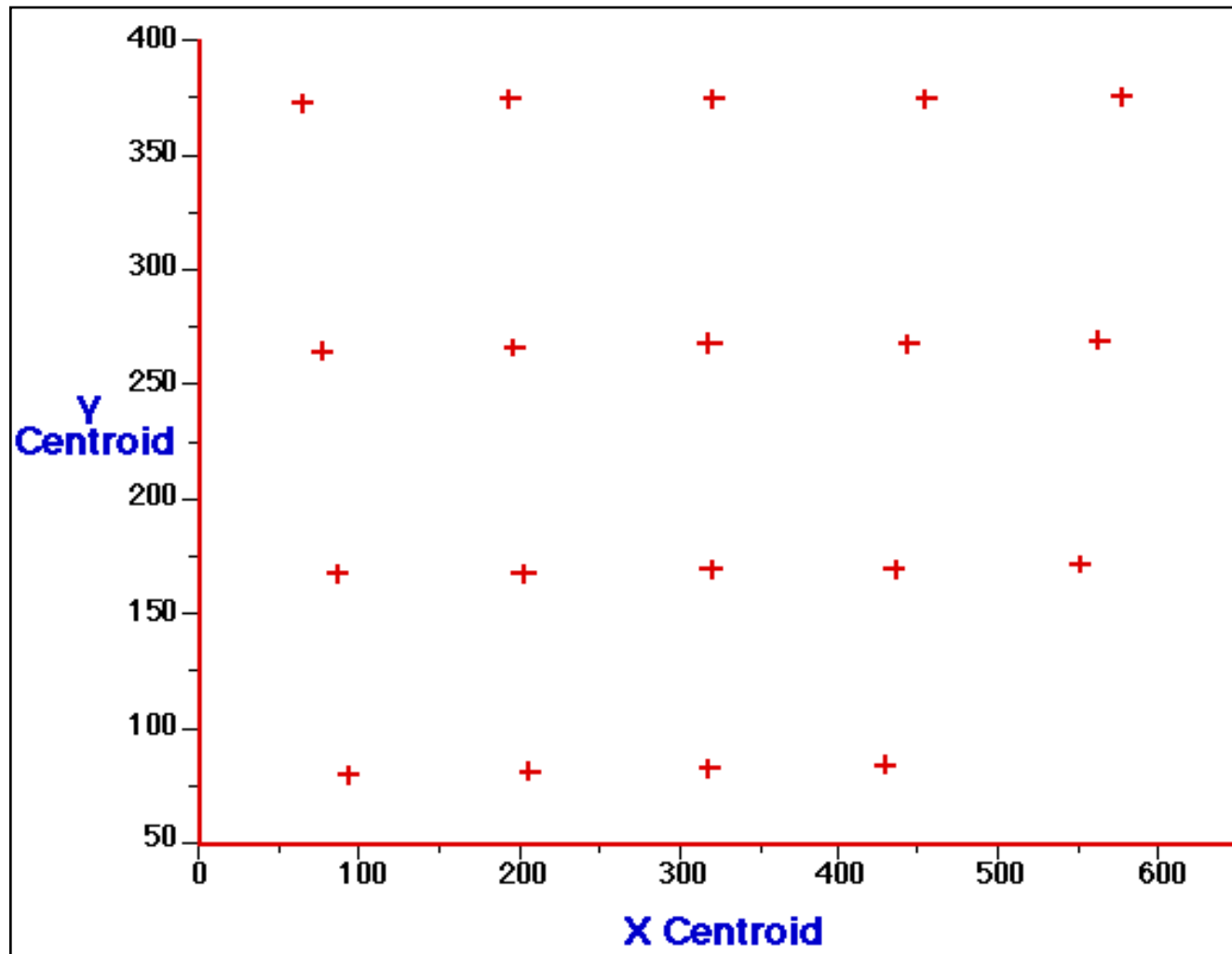


Image Processing

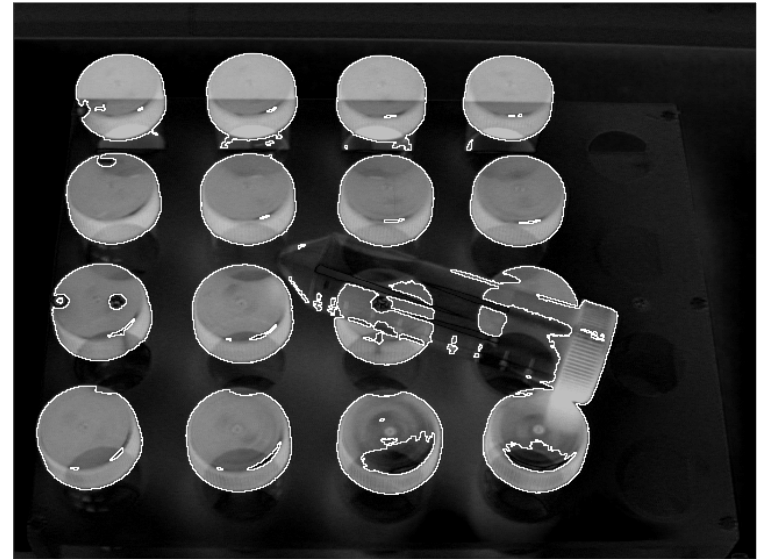
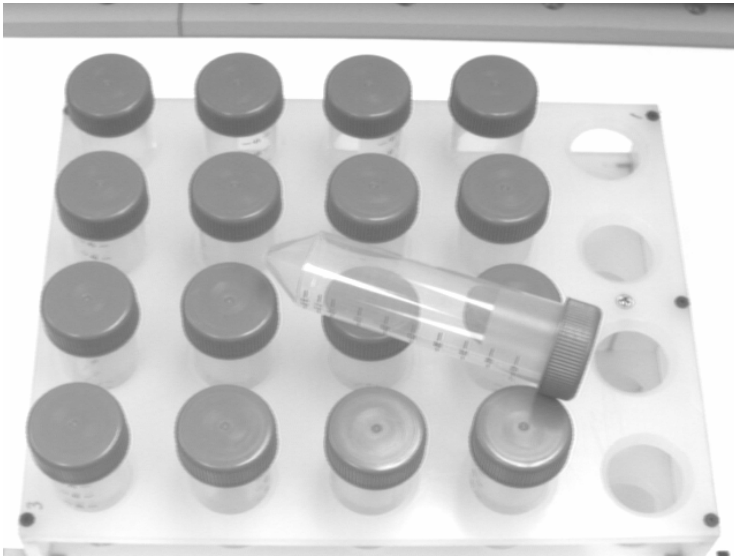
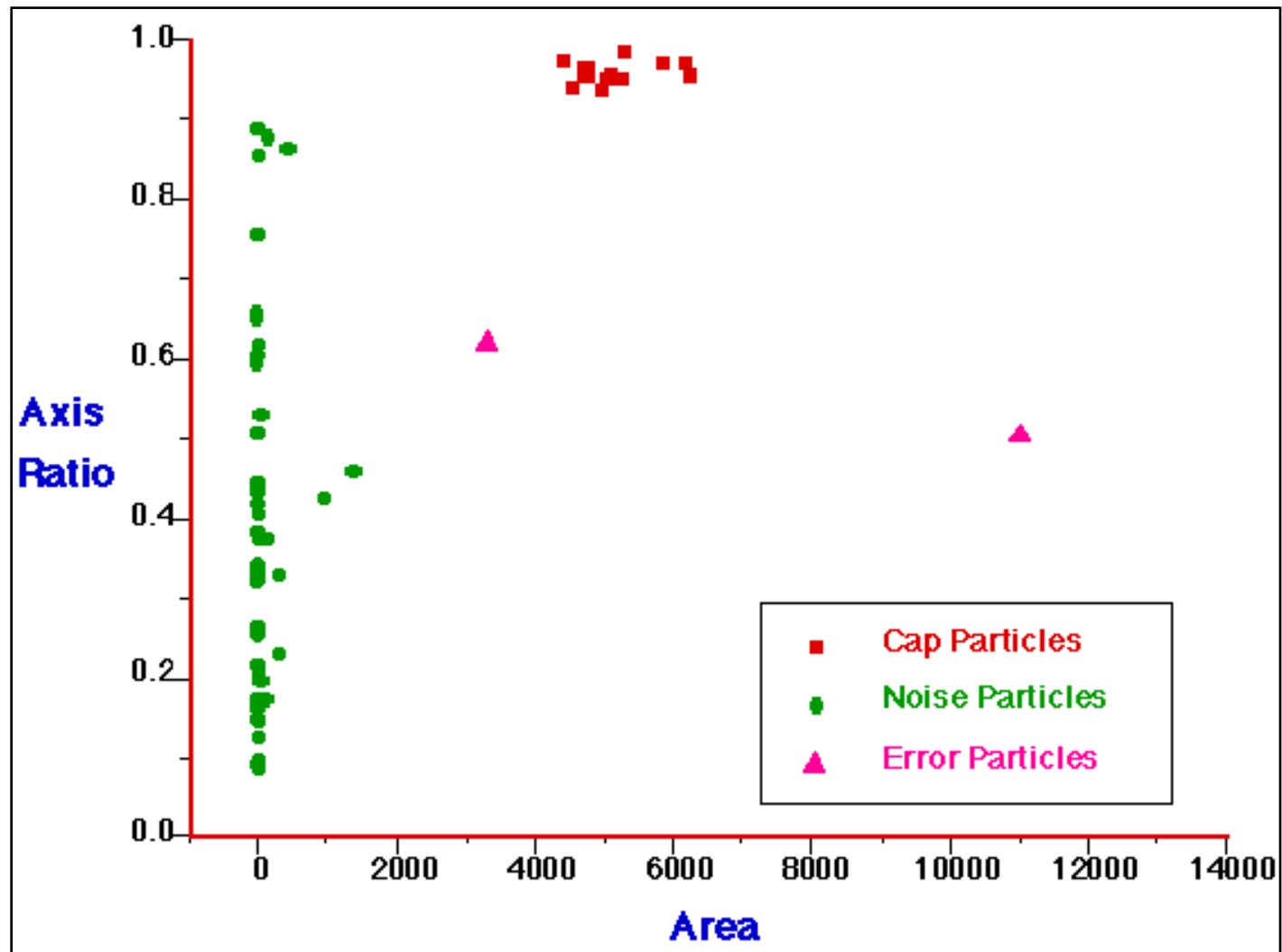


Image Processing



Predator algorithm for object tracking with learning

<http://www.youtube.com/watch?v=1GhNXHCQGSM>

Video Processing, with Processing

<http://www.niklasroy.com/project/88/my-little-piece-of-privacy/>

<http://www.youtube.com/watch?v=rKhbUjVyKlc>