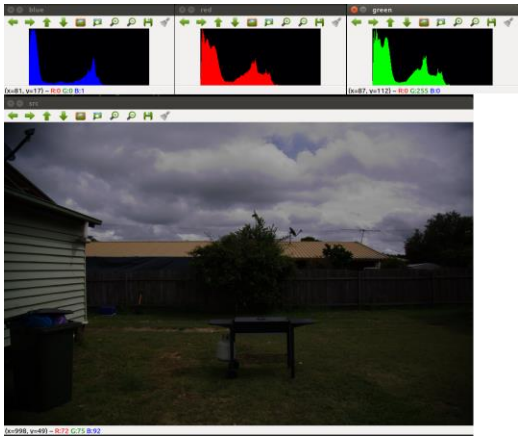
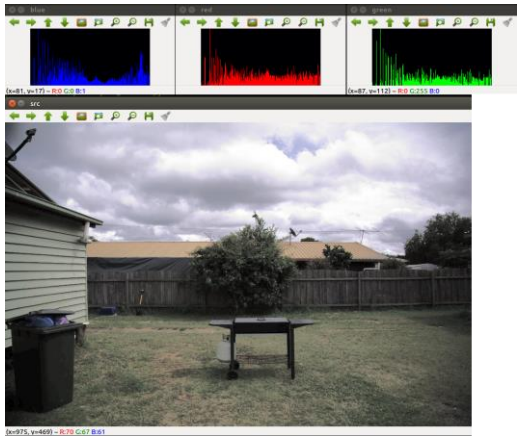
LAB3:
Image Processing

Computer Vision 2018

P. Zanuttigh

LAB3: Image Processing





Two main tasks (*and an optional one*)

1. Compute the histogram of an image and equalize it
2. Remove the noise using image filtering

- *Optional Task*: use morphological operators to remove thin structures (*not required for the homework*)

Histogram of an Image

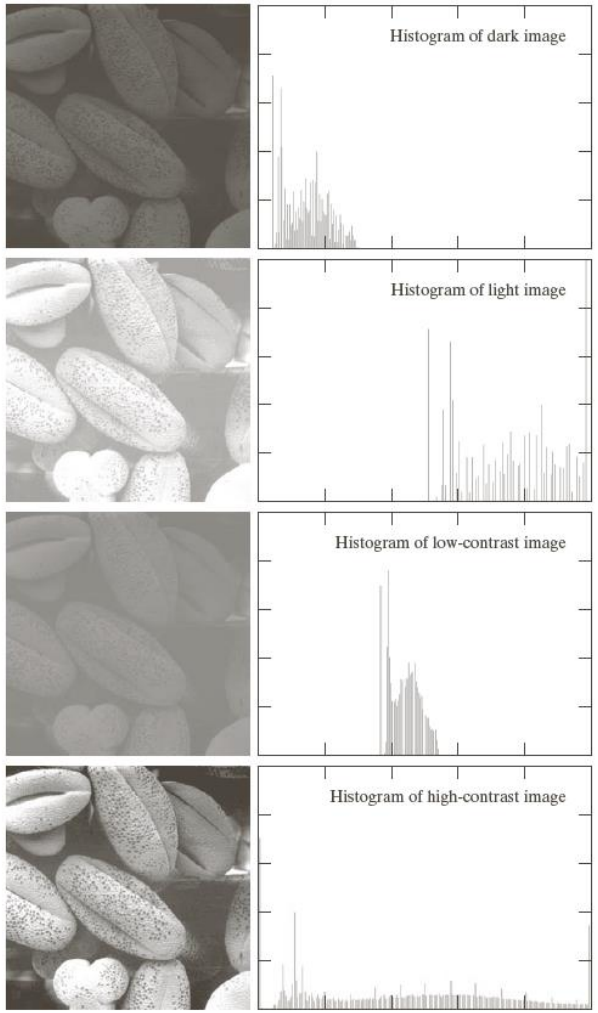


FIGURE 3.16 Four basic image types: dark, light, low contrast, high contrast, and their corresponding histograms.

Normalized Histogram

$$p(r_k) = \frac{h(r_k)}{MN} = \frac{n_k}{MN}$$

$h(r_k) = n_k$ = number of pixels with intensity equal to r_k

Can be viewed as a *probability density*

Usage:

1. Image statistics
2. Compression
3. Segmentation
4. Image enhancement

```
void cv::calcHist (     const Mat *       images,
                        int               nimages,
                        const int *       channels,
                        InputArray        mask,
                        OutputArray       hist,
                        int               dims,
                        const int *       histSize,
                        const float **    ranges,
                        bool              uniform = true,
                        bool              accumulate = false
                    )
```

The "*const*" Keyword

```
const int Constant1 = 96;
```

- Declare a constant as if it was a variable but add "*const*" before it
- Adding "*const*" before a variable means that it can't be modified
- One has to initialise it immediately in the constructor
  - because setting the value later would be as changing it
- Such constants are useful for parameters which are used in the program but do not need to be changed after the program is compiled

- It is also possible to declare pointers that can access the pointed value to read it, but not modify it

```
int x;
int y = 10;
const int * p = &y;
x = *p;    // ok: reading p
*p = x;    // error: modifying p, which is const-qualified
```

- Here p points to a variable, but points to it in a const-qualified manner, meaning that it can read the value pointed, but it cannot modify it
- Used to pass parameters to methods that should only read the data

cv::calcHist()

```
void cv::calcHist    (  const Mat *      images,          // input image
                        int              nimages,         // =1 for a single image
                        const int *      channels,        // =0 (first channel, split and work
                                                          on each channel by itself)
                        InputArray       mask,            // cv::Mat()  (do not use mask)
                        OutputArray      hist,            // output, data is put in a cv::Mat
                        int              dims,            // 1
                        const int *      histSize,        // number of bins, use an array of
                                                          size 1 with the # in the first elem
                        const float **   ranges,          // array of array with min and
                                                          max values
                        bool             uniform = true,  // true
                        bool             accumulate = false  // false
                     )
```

The function with more than one channel computes an n-dimensional histogram, not 3 histograms for the 3 channels. To work on color images split the image into the 3 channels and work on each channel by itself (you can use the cv::split function)

Histogram Equalization

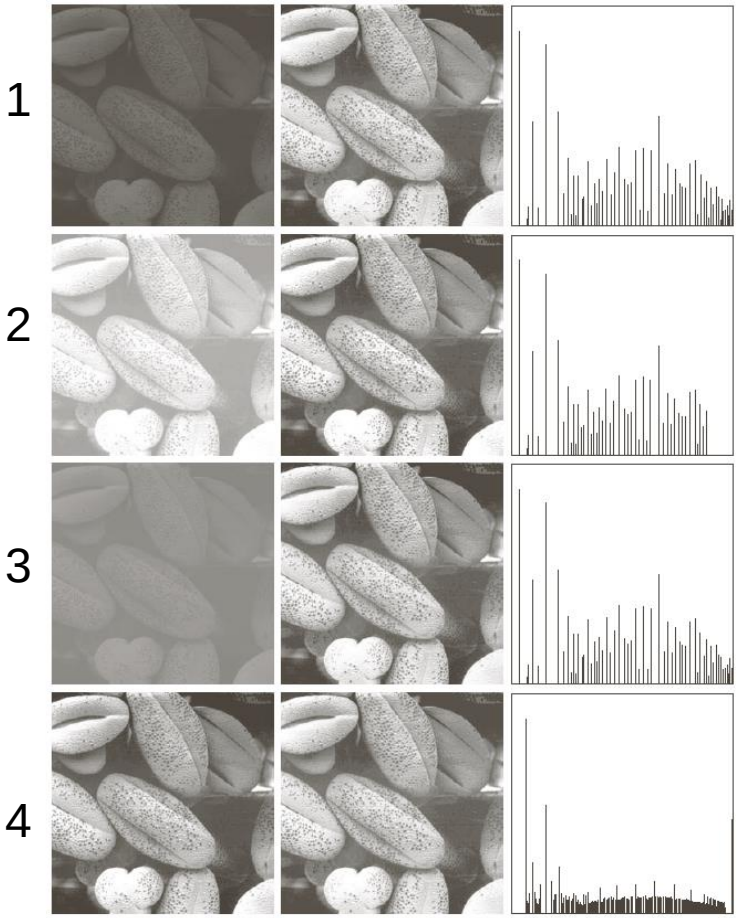


FIGURE 3.20 Left column: images from Fig. 3.16. Center column: corresponding histogram-equalized images. Right column: histograms of the images in the center column.

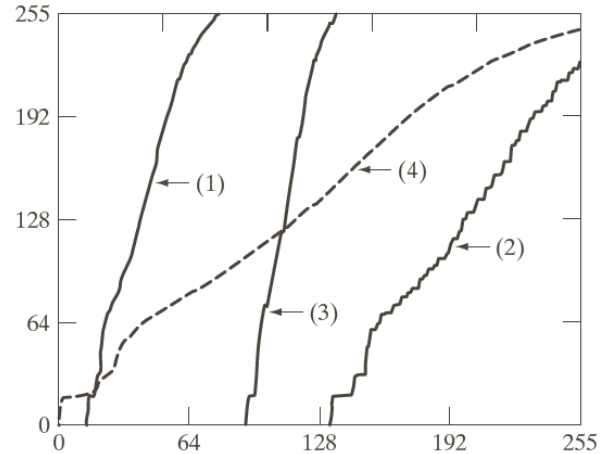


FIGURE 3.21 Transformation functions for histogram equalization. Transformations (1) through (4) were obtained from the histograms of the images (from top to bottom) in the left column of Fig. 3.20 using Eq. (3.3-8).

$$s = T(r) = (L-1)\int_0^r p_r(w)dw$$

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j)$$

Try to change the color space !

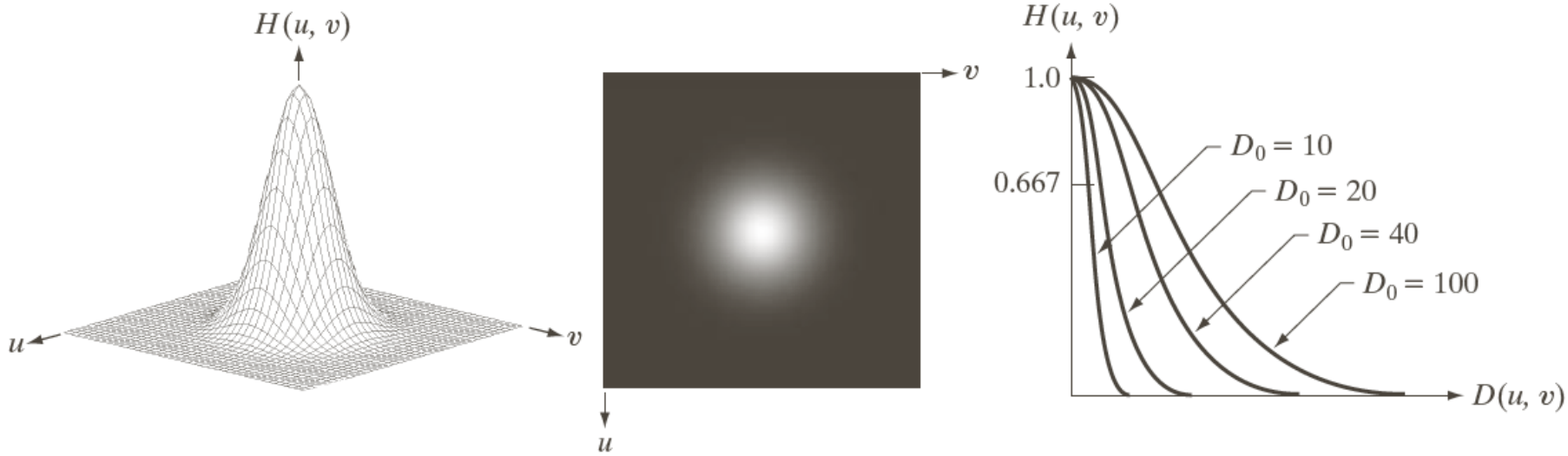
```
void cv::equalizeHist( InputArray        src,
                       OutputArray       dst
                     )
```

Image Denoising

- Generate a denoised version of the equalized image
- You should try different filters and parameter values (*see table*)
- Write a program that shows the result with some trackbars to vary the parameters
- To pass the image and the parameters to the callback of the trackbar create a class containing the image and parameters
- Extend the base filter class creating subclasses for the various filters

cv::medianFilter()	• Kernel size (square and odd)
cv::GaussianBlur()	• Kernel size (square and odd) • Sigma (sigmaX =sigmaY)
cv::bilateralFilter()	• Kernel size (trackbar not required, use a fixed value or use the $6\sigma_s$ rule) • Sigma range • Sigma space

Gaussian Low Pass Filter



a b c

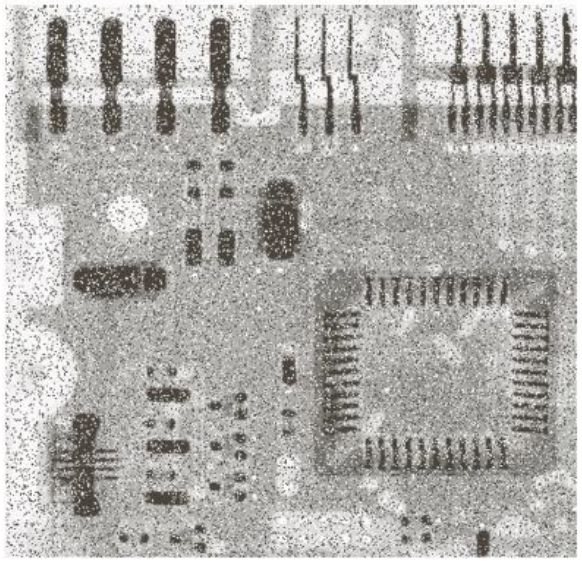
FIGURE 4.47 (a) Perspective plot of a GLPF transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections for various values of D_0.

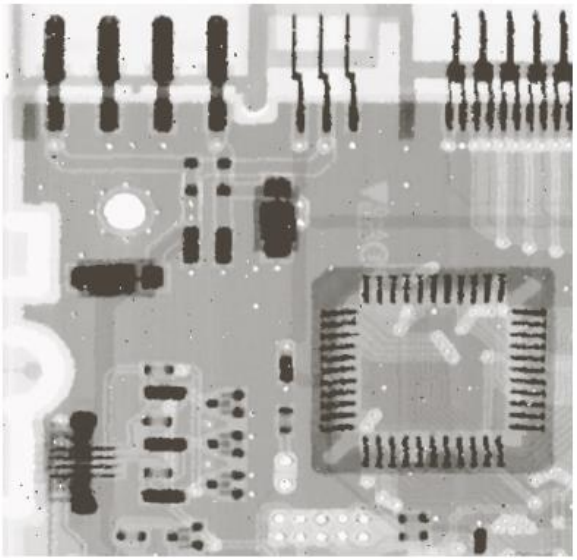
$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

There's no ringing !

```
void cv::GaussianBlur  (
                InputArray    src,
                OutputArray   dst,
                Size          ksize,
                double        sigmaX,
                double        sigmaY = 0, // 0: σx= σy
                int           borderType = BORDER_DEFAULT
                )
```

Median Filter





$$g(x,y) = \underset{(s,t)\in R}{median}\{f(s,t)\}$$

3	7	4
8	2	92
10	10	5

2	3	4	5	7	8	10	10	92

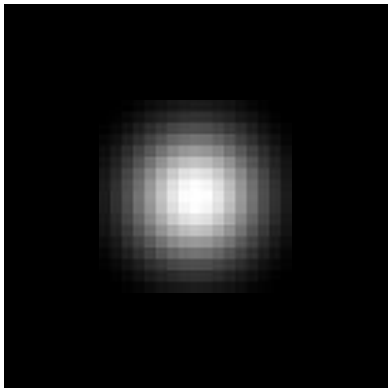
```
void cv::medianBlur    (   InputArray    src,
                           OutputArray   dst,
                           int           ksize
                       )
```

Bilateral Filter

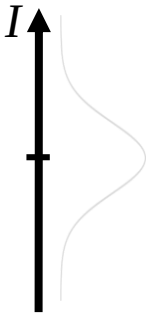
$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}\left(\|\mathbf{p}-\mathbf{q}\|\right) G_{\sigma_r}\left(|I_{\mathbf{p}} - I_{\mathbf{q}}|\right) I_{\mathbf{q}}$$

normalization factor

space weight



range weight



- Weighted average of pixels, but the weights depend on both the spatial distance and the color/intensity difference
- Space σ_s : spatial extent of the kernel
- Range σ_r : “minimum” amplitude of an edge

Extend the Filter Class

```
class Filter{

public:

// constructor
Filter(cv::Mat input_img, int
filter_size);

// perform filtering (in base class
// does nothing)
void doFilter();

// get output of the filter
cv::Mat getResult();

//get-set methods for params
void setSize(int size);//check if odd!
int getSize();

protected:

// input image
cv::Mat input_image;

// output image (filter result)
cv::Mat result_image;

// window size
int filter_size;
};
```

Extend the filter class and implement the 3 filters inside the derived classes

1. class MedianFilter : public Filter { …….
2. class GaussianFilter : public Filter { …….
3. class BilateralFilter : public Filter { …….

Derived classes

- ❑ Extra parameters:
  - Median : no new parameters, only the size !
  - Gaussian: contains also sigma
  - Bilateral: sigma_space, sigma_range
- ❑ Redefine *doFilter()* with the corresponding operations

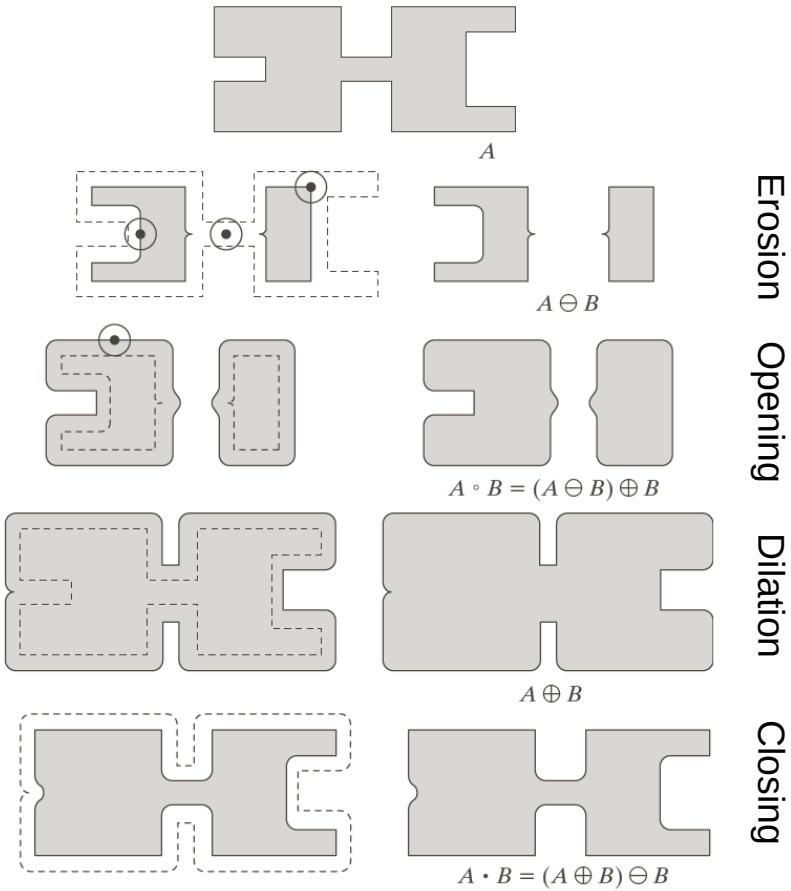
Trackbars in OpenCV

```
int cv::createTrackbar    (  const String &        trackbarname,

                             const String &        winname,
                             int *                 value,            //data written here
                             int                   count,            //max value
                             TrackbarCallback      onChange = 0,     //function to be called
                             void *                userdata = 0      //as for mouse callback
                          )
```

```
int cv::getTrackbarPos    (  const String &        trackbarname,
                             const String &        winname
                          )
```

Morphological Operators




```
cv::Mat element =
cv::getStructuringElement(cv::MORPH_RECT, cv::Size(3,3));
```

```
void cv::dilate (
    InputArray      src,
    OutputArray     dst,
    InputArray      kernel,
    Point           anchor = Point(-1,-1),
    int             iterations = 1,
    int             borderType = BORDER_CONSTANT,
    const Scalar &  borderValue = morphologyDefaultBorderValue()
)
```

```
void cv::morphologyEx (
    InputArray      src,
    OutputArray     dst,
    int             op,
    InputArray      kernel,
    Point           anchor = Point(-1,-1),
    int             iterations = 1,
    int             borderType = BORDER_CONSTANT,
    const Scalar &  borderValue = morphologyDefaultBorderValue()
)
```