Pixel Processing

SS 2008 – Image Processing

Multimedia Computing, Universität Augsburg
Rainer.Lienhart@informatik.uni-augsburg.de
www.multimedia-computing.{de,org}
Reference


Chapter on *Pixel Processing*  
(chap 8 in 4\textsuperscript{th} edition)

Most Figures are taken from that book
Point Operations

:= modification of gray value at individual pixels dependent only on the gray value and possibly on the position of the pixel

\[ I'(x, y) = P_{x,y}(I(x, y)) \]
Homogeneous Point Operations

\[ I'(x, y) = P(I(x, y)) \]

- Map set of gray values onto itself
- Generally not invertible:
  - Example: Thresholding
    \[ P(q) = \begin{cases} 
    0 & q < \text{threshold} \\
    255 & q \geq \text{threshold} 
    \end{cases} \]
    → Not invertible
  - Example: Inversion
    \[ P_{\text{negation}}(q) = 255 - q, \quad q = I(x, y) \]
    → Invertible
Image Thresholding

Two principles
- Fixed threshold;
- Adaptive threshold;

```c
void cvThreshold( const CvArr* src, CvArr* dst, double threshold, double max_value, int threshold_type );
```

- `src` = Source array (single-channel, 8-bit of 32-bit floating point).
- `dst` = Destination array; must be either the same type as `src` or 8-bit.
- `threshold` = threshold value.
- `max_value` = Maximum value to use with `CV_THRESH_BINARY` and `CV_THRESH_BINARY_INV` thresholding types.
- `threshold_type` = Thresholding type (see the discussion)
Adaptive Thresholding

void cvAdaptiveThreshold( const CvArr* src, CvArr* dst, double max_value, int adaptive_method=CV_ADAPTIVE_THRESH_MEAN_C, int threshold_type=CV_THRESH_BINARY, int block_size=3, double param1=5 );

Src = source image.
Dst = Destination image.
max_value = Maximum value that is used with CV.THRESH_BINARY and CV.THRESH_BINARY_INV.
adaptive_method = Adaptive thresholding algorithm to use: CV_ADAPTIVE_THRESH_MEAN_C or CV_ADAPTIVE_THRESH_GAUSSIAN_C (see the discussion).
threshold_type = Thresholding type; must be one of CV.THRESH_BINARY, CV.THRESH_BINARY_INV
block_size = The size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, ...
Param1 = The method-dependent parameter. For the methods CV_ADAPTIVE_THRESH_MEAN_C and CV_ADAPTIVE_THRESH_GAUSSIAN_C it is a constant subtracted from mean or weighted mean (see the discussion), though it may be negative.

The function cvAdaptiveThreshold transforms grayscale image to binary image according to the formulae:

threshold_type.CV.THRESH_BINARY:
• \( \text{dst}(x,y) = \max_{value} \), if \( \text{src}(x,y) > T(x,y) \) 0, otherwise

threshold_type.CV.THRESH_BINARY_INV:
• \( \text{dst}(x,y) = 0 \), if \( \text{src}(x,y) > T(x,y) \) \max_{value} \), otherwise where \( T \) is a threshold calculated individually for each pixel.

For the method CV_ADAPTIVE_THRESH_MEAN_C it is a mean of block_size \( \times \) block_size pixel neighborhood, subtracted by param1.

For the method CV_ADAPTIVE_THRESH_GAUSSIAN_C it is a weighted sum (gaussian) of block_size \( \times \) block_size pixel neighborhood, subtracted by param1.
LUT Operations

:= Look-up tables operations \( \rightarrow \) efficient way to implement homogeneous point operations on large images

Example:
- Homogeneous point operation on 1920x1080 8u_C1 image \( \rightarrow \) Need an LUT array of size 256.
- \( P(q) := \log (q) \rightarrow \) only 256 log operations with LUT vs. 1920*1080 log operations with out LUT

```
void cvLUT( const CvArr* src, CvArr* dst, const CvArr* lut );
\rightarrow \) Performs look-up table transform of array
```

\textit{src} = Source array of 8-bit elements.
\textit{dst} = Destination array of arbitrary depth and of the same number of channels as the source array.
\textit{lut} = Look-up table of 256 elements; should have the same depth as the destination array.

The function \texttt{cvLUT} fills the destination array with values from the look-up table. Indices of the entries are taken from the source array. That is, the function processes each element of \texttt{src} as following:

\[ \text{dst}(I)=\text{lut}[\text{src}(I)+\text{DELTA}] \]

where \text{DELTA}=0 if \texttt{src} has depth CV\_8U, and \text{DELTA}=128 if \texttt{src} has depth CV\_8S.
Figure 8.3: Illustration of a nonlinear look-up table with mapping of multiple values onto one and missing output value leading to uneven steps.
Figure 8.4: a The irradiance is gradually decreasing from the top to the bottom, which is almost not recognized by the eye. The gray scale of this floating-point image computed by averaging over 100 images ranges from 160 to 200. b Histogram of a; c and d (contrast enhanced, gray scale 184.0–200.0): Edges artificially produced by a staircase LUT with a step height of 1.0 and 2.0 make contours of constant irradiance easily visible (exercise 8.1).
Detection of Under-/Overflow

Figure 8.5: Detection of underflow and overflow in digitized images by histograms (exercise 8.2): a) image with underflow and b) its histogram; c) image with overflow and d) its histogram.
Figure 8.6: Contrast enhancement (exercise 8.3); a underexposed image and b its histogram; c interactively contrast enhanced image and d its histogram.
Contrast Enhancement (2)

Example 1:

```c
Void
NormalizeImageTo255 ( IplImage *in, IplImage *out )
{
    double InputMin, InputMax;
    cvMinMaxLoc( in, &InputMin, &InputMax );
    cvConvertScale ( in, in, 1, -InputMin);
    cvConvertScale ( in, out, 255.0 / (InputMax - InputMin), 0);
}
```

Example 2:

```c
Void
NormalizeImageTo255 ( IplImage *in, IplImage *out, double cutOffPercent=2.5)
{
    Ipp8u InputMin=0, InputMax=255;
    Ipp32u sum=0, sumT= in->width * (in->height * cutOffPercent);
    Ipp32u histo[256] = {0};
    calcHisto( in, histo );

    for (sum=0 ; InputMin<256 && sum<sumT; InputMin++) sum+=histo[i];
    InputMin--;
    for (sum=0 ; InputMax>=0 && sum<sumT; InputMax--) sum+=histo[i];
    InputMax++;

    cvConvertScale ( in, in, 1, -InputMin);
    cvConvertScale ( in, out, 255.0 / (InputMax - InputMin), 0);
}
```

:= Point operation dependent of position of pixel

\[ I'(x, y) = P_{x,y}(I(x, y)) \]

• Mostly related to calibration procedures

• Example:
  – Subtraction of a background image without objects
    \[ I'(x, y) = P_{x,y}(I(x, y)) = I(x, y) - B(x, y) \]
  – Temporal image averaging
    \[ I'(x, y, t) = P_{x,y}(I(x, y, t)) = \sum_{\tau=-T}^{T} w(\tau) I(x, y, \tau) \]
Figure 8.9: Noise reduction by image averaging (exercise 8.4): a single thermal image of small temperature fluctuations on the water surface cooled by evaporation; b same, averaged over 16 images; the full gray value range corresponds to a temperature range of 1.1 K.
Illumination Correction

Where is always

- Uneven illumination
- Dust on lenses
- Uneven spatial CCD sensitivity

→ Introduce systematic errors

\[ I'(x, y) = c \frac{I(x, y)}{R(x, y)} \]

Ref image taken with full illumination

\[ I'(x, y) = c \frac{I(x, y) - B(x, y)}{R(x, y) - B(x, y)} \]

BG image taken without illumination (fixed pattern noise on CCD)

Figure 8.10: Correction of uneven illumination with an inhomogeneous point operation (exercise 8.5): a original image and b its histogram; c background image and d its histogram; e division of the image by the background image and f its histogram.
Figure 8.13: Effect of windowing on the discrete Fourier transform (exercise 8.7): a original image; b DFT of a without using a window function; c image multiplied with a cosine window; d DFT of c using a cosine window.