OpenCV – Histogram 활용

김성영교수
금오공과대학교
컴퓨터공학과

학습 내용

- Histogram Scaling 구현
- Histogram Equalization 구현
- Histogram Backprojection
- Histogram Comparison

Histogram scaling 구현

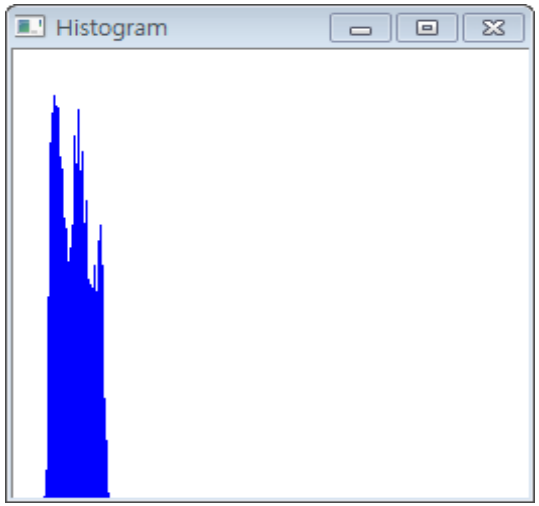
$$I'(x,y) = \left[\frac{S_{MAX} - S_{MIN}}{I_{MAX} - I_{MIN}}\right][I(x,y) - I_{MIN}] + S_{MIN}$$

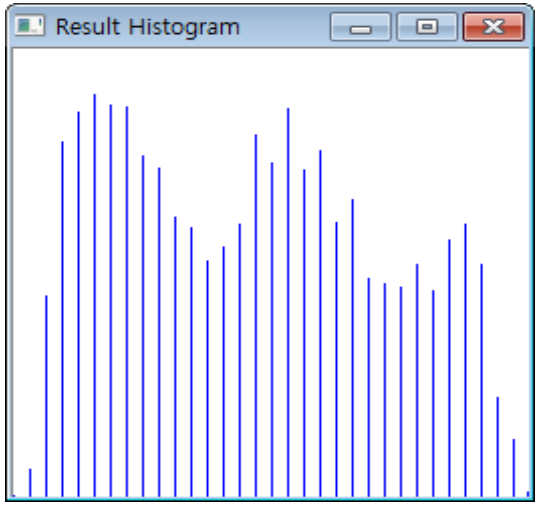
I_{MAX} : largest gray-level value in the image $I(x,y)$

I_{MIN} : smallest gray-level value in $I(x,y)$

S_{MAX} : maximum gray-level values possible
(for an 8-bit image this is 255)

S_{MIN} : minimum gray-level values possible
(for an 8-bit image this is 0)


Histogram


Result Histogram

```
MatND getHistogram( const Mat &image, int bins=256 );
Mat   createHistImage( const MatND &hist, int bins=256 );
void  histScaling( const Mat &image, Mat &result,
                   float minValue=0.0001f, float minScale=0,
                   float maxScale=255 );

int main(void)
{
   Mat image = imread( "input.jpg", 0 );
   if( image.data == NULL ) return -1;

   // histogram scaling
   Mat result;
   histScaling( image, result );

   // create histogram image & display
   MatND orgHist    = getHistogram( image );
   MatND resultHist = getHistogram( result );
   Mat orghistImg    = createHistImage( orgHist );
   Mat resulthistImg = createHistImage( resultHist );
```

```
    // Display the images
    namedWindow( "Image" );
    namedWindow( "Result" );
    namedWindow( "Histogram" );
    namedWindow( "Result Histogram" );

    imshow( "Image",  image );
    imshow( "Result", result );
    imshow( "Histogram", orghistImg );
    imshow( "Result Histogram", resulthistImg );

    waitKey();

    return 0;
}
```

```
void histScaling( const Mat &image, Mat &result,
        float minValue, float minScale, float maxScale )
{
   // Compute histogram
   int bins = 256;
   orgHist = getHistogram( image, bins );

   minValue *= (image.rows*image.cols);

   // find left extremity of the histogram
   int lbin;
   for( lbin=0; lbin<bins; lbin++ )
   {
      if( orgHist.at<float>(lbin) > minValue ) break;
   }

   // find right extremity of the histogram
   int ubin;
   for( ubin=bins-1; ubin>=0; ubin-- )
   {
      if( orgHist.at<float>(ubin) > minValue )  break;
   }
```

```
   result.create( image.rows, image.cols, image.type() );

   int numOfLines  = image.rows;
   int numOfPixels = image.cols;
   // do scaling
   for( int r=0; r<numOfLines; r++ ) {
      const uchar *data_in = image.ptr<uchar>( r );
      uchar *data_out = result.ptr<uchar>( r );
      for( int c=0; c<numOfPixels; c++ ) {
         data_out[ c ] = saturate_cast<uchar>(
                              ((maxScale-minScale)/(ubin-lbin))*
                              (data_in[c]-lbin)+minScale+0.5 );
      }
   }
}
```

$$I'(x,y) = \left[\frac{S_{MAX} - S_{MIN}}{I_{MAX} - I_{MIN}}\right][I(x,y) - I_{MIN}] + S_{MIN}$$

Method 2

$$I'(x,y) = \left[\frac{S_{MAX} - S_{MIN}}{I_{MAX} - I_{MIN}}\right][I(x,y) - I_{MIN}] + S_{MIN}$$

```
result = ((maxScale-minScale) / (ubin-lbin))
         * (image-lbin) + minScale + 0.5;
```

Method 3

$$dst(I) = \text{saturate\_cast} < \text{uchar} > \left(\left| src(I) * \alpha + \beta \right| \right)$$

```
void convertScaleAbs(
        InputArray src,  // source
        OutputArray dst, // destination
        double alpha=1,  // scale factor
        double beta=0    // added to the scaled values
)
```

```
convertScaleAbs( image-lbin,
                 result,
                 (maxScale-minScale)/(ubin-lbin),
                 minScale
)
```

$$I'(x,y) = \left[\frac{S_{MAX} - S_{MIN}}{I_{MAX} - I_{MIN}} \right] [I(x,y) - I_{MIN}] + S_{MIN}$$

Method 4

$$dst(I) = \text{saturate\_cast} < \text{rtype} > \left(\left| src(I) * \alpha + \beta \right| \right)$$

```
Mat::convertTo(
      OutputArray m,   // destination
      int rtype,       // desired dest. type
      double alpha=1, // scale factor
      double beta=0    // added value
)
```

```
Mat image2 = image - lbin;
image2.convertTo( result, result.type(),
                  (maxScale-minScale)/(ubin-lbin),
                  minScale
);
```

$$I'(x,y) = \left[\frac{S_{MAX} - S_{MIN}}{I_{MAX} - I_{MIN}} \right] [I(x,y) - I_{MIN}] + S_{MIN}$$

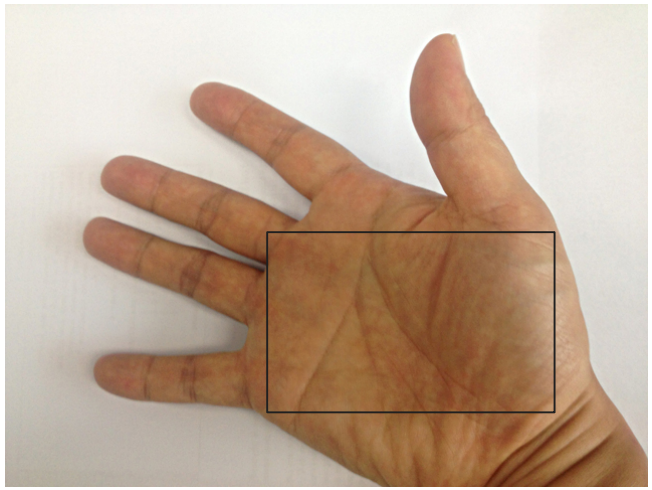
Histogram Equalization 구현

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

src – Source 8-bit single channel image

dst – Destination image of the same size and type as **src**

Histogram Backprojection



```
void calcBackProject(
        const Mat* images,
        int nimages,
        const int* channels,
        InputArray hist,
        OutputArray backProject,
        const float** ranges,
        double scale=1,
        bool uniform=true
)
```

```
void normalize(
        InputArray src,
        OutputArray dst,
        double alpha=1,
        double beta=0,
        int norm_type=NORM_L2,
        int dtype=-1,
        InputArray mask=noArray()
)
```

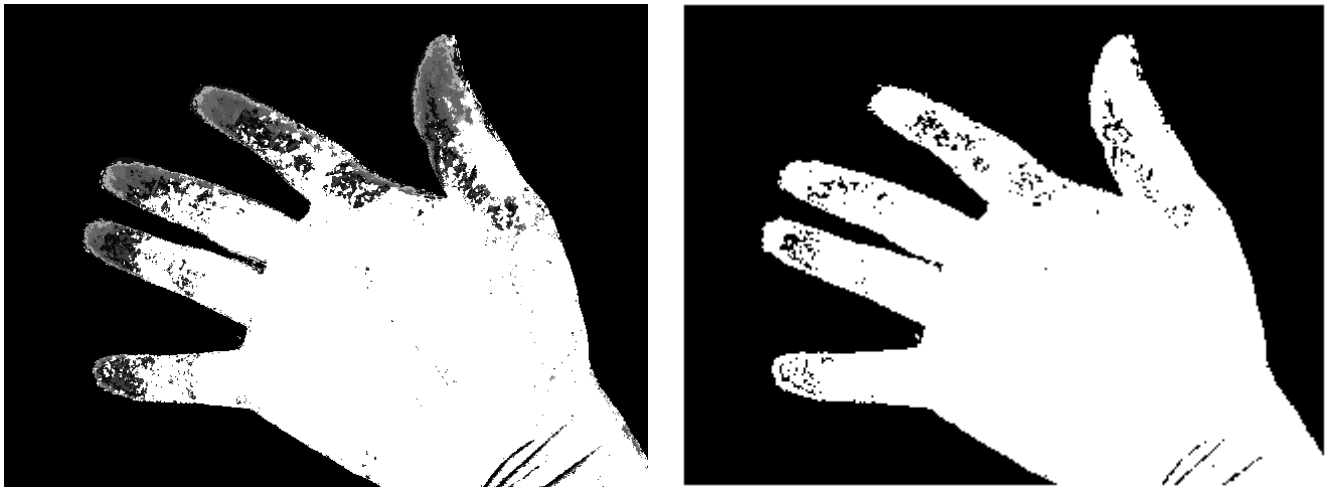
$$\begin{cases} \|src\|_{L_\infty} = \max|src(I)| = \alpha & if\ \ normType = NORM\_INF \\ \|src\|_{L_1} = \sum|src(I)| = \alpha & if\ \ normType = NORM\_L1 \\ \|src\|_{L_2} = \sqrt{\sum src(I)^2} = \alpha & if\ \ normType = NORM\_L2 \\ \|src\|_{MM} = [alpha, beta] & if\ \ normType = NORM\_MINMAX \end{cases}$$

norm_type

```
#define CV_8U  0
#define CV_8S  1
#define CV_16U 2
#define CV_16S 3
#define CV_32S 4
#define CV_32F 5
#define CV_64F 6
```

dtype

```
double threshold(
        InputArray src,    // source
        OutputArray dst,   // destination
        double thresh,     // threshold value
        double maxval,     // maximum value
        int type           // threshoding type
)
```



thresholding with 30

- THRESH_BINARY

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- THRESH_BINARY_INV

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$$

- THRESH_TRUNC

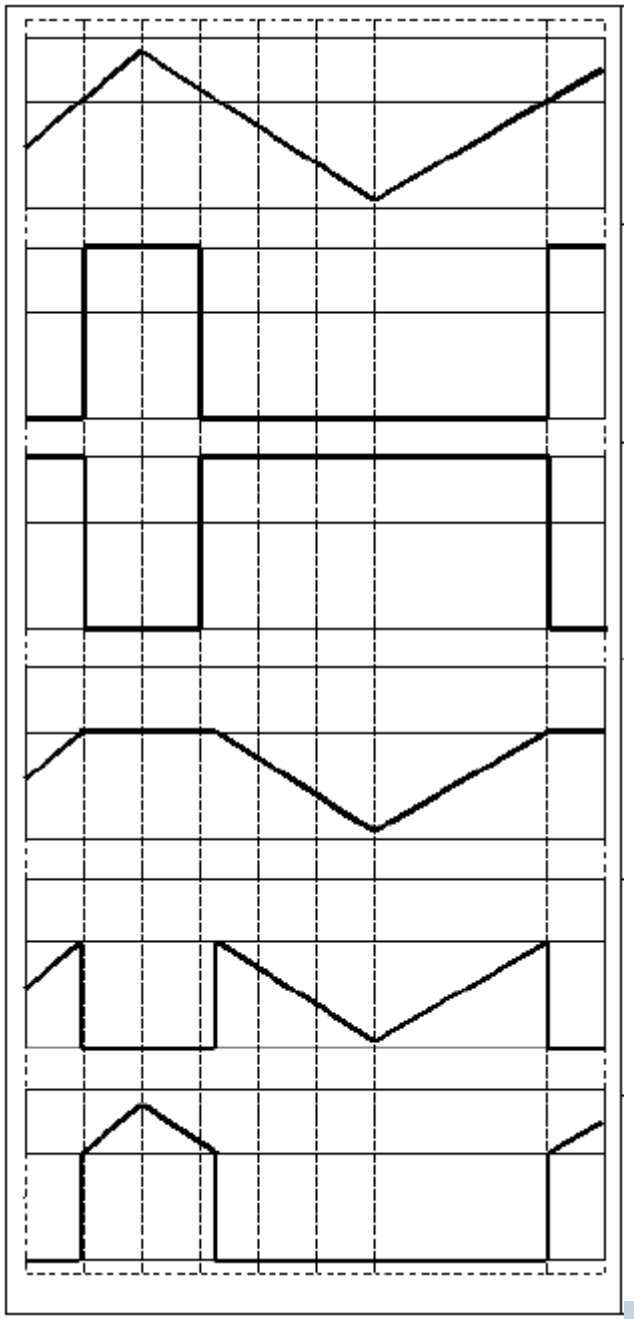
$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

- THRESH_TOZERO

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- THRESH_TOZERO_INV

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$



```
MatND getHistogram( const Mat &image, int hbins=30, int sbins=32 );
void  backprojectHist( const Mat &image, Mat &result, MatND hist );

int main(void)
{
   Mat image1 = imread( "hand.jpg", -1 );
   if( image1.data == NULL ) return -1;
   Mat image2 = imread( "hand2.jpg", -1 );
   if( image2.data == NULL ) return -1;

   Mat hsv1, hsv2;
   cvtColor( image1, hsv1, CV_BGR2HSV );
   cvtColor( image2, hsv2, CV_BGR2HSV );

   // get histogram from ROI
   Mat hand = hsv1( Rect(270, 230, 280, 180) );
   MatND hist = getHistogram( hand );

   Mat result1, result2;
   backprojectHist( hsv1, result1, hist );
   backprojectHist( hsv2, result2, hist );
```

```
    // Display the images
    namedWindow( "Image1" );
    namedWindow( "Result1" );
    namedWindow( "Image2" );
    namedWindow( "Result2" );

    imshow( "Image1", image1 );
    imshow( "Result1", result1 );
    imshow( "Image2", image2 );
    imshow( "Result2", result2 );

    waitKey();

    return 0;
}
```

```
MatND getHistogram( const Mat &image, int hbins, int sbins )
{
   int histSize[] = { hbins, sbins };
   // hue: 0 ~ 179, saturation: 0 ~ 255
   float hranges[] = { 0, 179 };
   float sranges[] = { 0, 255 };
   const float* ranges[] = { hranges, sranges };

   MatND hist;
   // we compute the histogram from the 0-th and 1-st channels
   int channels[] = { 0, 1 };

   calcHist( &image, 1, channels, Mat(), // do not use mask
         hist, 2, histSize, ranges,
         true, // the histogram is uniform
         false );

   return hist;
}
```

```
void backprojectHist( const Mat &image, Mat &result, MatND hist )
{
   // hue: 0 ~ 179, saturation: 0 ~ 255
   float hranges[] = { 0, 179 };
   float sranges[] = { 0, 255 };
   const float* ranges[] = { hranges, sranges };
   // we compute the histogram from the 0-th and 1-st channels
   int channels[] = { 0, 1 };

   normalize( hist, hist, 0, 255, NORM_MINMAX, -1, Mat() );

   calcBackProject( &image,
      1,              // use one image at a time
      channels,       // specifying image channels
      hist,           // the histogram we are using
      result,         // the resulting back projection image
      ranges,         // the range of values, for each dimension
      255.0           // the scaling factor
   );
}
```

Histogram Comparison



```
double compareHist(
      InputArray H1,
      InputArray H2,
      int method
)
```

Intersection (**method=CV_COMP_INTERSECT**)

$$d(H_1, H_2) = \sum_I \min(H_1(I), H_2(I))$$

Chi-Square (**method=CV_COMP_CHISQR**)

$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

Reference

- R. Laganière, **OpenCV2 Computer Vision: Application Programming Cookbook,** PACKT Publishing, 2011
- G. Bradski and A. Kaebler, **Learning OpenCV: Computer Vision with the OpenCV Library**, O'REILLY, 2008
- http://docs.opencv.org