OpenCV – Geometric Transform

김성영교수
금오공과대학교
컴퓨터공학과

학습 내용

- Reflecting images
- Scaling images
- Rotating images

Reflecting images

```
void remap(
      InputArray src,
      OutputArray dst,
      InputArray map1,
      InputArray map2,
      int interpolation,
      int borderMode=BORDER_CONSTANT,
      const Scalar& borderValue=Scalar()
)
```

- Applies a generic geometrical transformation to an image
- Using backward mapping

$$\mathrm{dst}(x, y) = \mathrm{src}(\mathrm{map}_x(x, y), \mathrm{map}_y(x, y))$$

Reflect an image from left right

$$(x', y') = (\text{width} - 1 - x, y)$$

Turn an image upside down

$$(x', y') = (x, \text{height} - 1 - y)$$

Combination

$$(x', y') = (\text{width} - 1 - x, \text{height} - 1 - y)$$

```cpp
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace cv;

void flip( const Mat &image, Mat &result, int type );

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

    Mat result;
    flip( image, result, 1 ); // vertical

    // Display the images
    namedWindow( "Image" );
    namedWindow( "Result" );
    imshow( "Image", image );
    imshow( "Result", result );

    waitKey();

    return 0;
}
```

```
void flip( const Mat &image, Mat &result, int type ) {
    result.create( image.size(), image.type() );

    Mat map_x, map_y;
    map_x.create( image.size(), CV_32FC1 );
    map_y.create( image.size(), CV_32FC1 );

    for( int j = 0; j < image.rows; j++ ){
        for( int i = 0; i < image.cols; i++ ){
            switch( type ) {
            case 1:
                map_x.at<float>(j,i) = i ;
                map_y.at<float>(j,i) = image.rows-1 - j ;
                break;
            case 2:
                map_x.at<float>(j,i) = image.cols-1 - i ;
                map_y.at<float>(j,i) = j ;
                break;
```

```
            case 3:
                map_x.at<float>(j,i) = image.cols-1 - i ;
                map_y.at<float>(j,i) = image.rows-1 - j ;
                break;
            } // end of switch
        }
    }

    remap( image, result,
        map_x, map_y,
        CV_INTER_LINEAR, BORDER_CONSTANT, Scalar(0,0,0) );
}
```

Scaling images

```cpp
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace cv;

void scale( const Mat &image, Mat &result,
        double scaleFactor, int mode );
BYTE InterpolationGray( const Mat &image,
        double x, double y, int mode );

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

    Mat result;
    scale( image, result, 2.0, 1 );

    namedWindow( "Image" );
    imshow( "Image", image );
    namedWindow( "Result" );
    imshow( "Result", result );

    waitKey();
    return 0;
}
```

```
void scale( const Mat &image, Mat &result,
        double scaleFactor, int mode ){
    result.create( image.size(), image.type() );

    // Performs Image Scaling
    int   x, y;
    float tmpx, tmpy;
    BYTE  L;

    for( y=0; y<image.rows; y++){
        const uchar *data_in = image.ptr<uchar>( y );
        uchar *data_out = result.ptr<uchar>( y );
        for( x=0; x<image.cols; x++ ){
            // 역사상에 의해 결과 좌표에 대응하는 원본 좌표 계산
            tmpx = x / scaleFactor;
            tmpy = y / scaleFactor;
            // 보간법에 의한 픽셀 값 계산
            uchar L = InterpolationGray( image,
                tmpx, tmpy, mode );
            // 픽셀 값 저장
            data_out[x] = L;
        }
    }
}
```

```
BYTE InterpolationGray( const Mat &image,
            double x, double y, int mode )
{
    // -----------------------------------------------------------
    // 픽셀 값을 계산
    // -----------------------------------------------------------
    // Interpolation
    //   mode 1: Nearnest Neighbor
    //   mode 2: Bilinear
    uchar L;
    if ( mode == 1 )
    {
        int orgx, orgy;
        orgx = (int)( x+0.5 );
        orgy = (int)( y+0.5 );
        orgx = __max( __min(image.cols-1, orgx), 0 );
        orgy = __max( __min(image.rows-1, orgy), 0 );
        L = (uchar)image.data[ orgy*image.step + orgx ];
    }
```

```
else if ( mode == 2 )
{
    int tmpx1, tmpx2, tmpy1, tmpy2;
    tmpx1 = (int)x;
    tmpy1 = (int)y;
    tmpx2 = (int)(x + 1.0f);
    tmpy2 = (int)(y + 1.0f);

    // 원본 영상의 경계를 벗어나는 것을 방지
    tmpx1 = __max( __min( image.cols-1,  tmpx1 ), 0 );
    tmpy1 = __max( __min( image.rows-1, tmpy1 ), 0 );
    tmpx2 = __max( __min( image.cols-1,  tmpx2 ), 0 );
    tmpy2 = __max( __min( image.rows-1, tmpy2 ), 0 );

    float alpha, beta;
    alpha = x - tmpx1;
    beta  = y - tmpy1;
```

```
        uchar L1, L2, L3, L4, L;
        L1 = (uchar)image.data[ tmpy1*image.step + tmpx1 ];
        L2 = (uchar)image.data[ tmpy1*image.step + tmpx2 ];
        L3 = (uchar)image.data[ tmpy2*image.step + tmpx1 ];
        L4 = (uchar)image.data[ tmpy2*image.step + tmpx2 ];

        L = (uchar)( (1-alpha)*(1-beta)*L1 +
                alpha*(1-beta)*L2 + (1-alpha)*beta*L3 +
                alpha*beta*L4 );
    }

    return L;
}
```

```cpp
void scale( const Mat &image, Mat &result, float sfactor )
{
    result.create( image.size(), image.type() );

    Mat map_x, map_y;
    map_x.create( image.size(), CV_32FC1 );
    map_y.create( image.size(), CV_32FC1 );
    for( int j = 0; j < image.rows; j++ )
    {
        for( int i = 0; i < image.cols; i++ )
        {
            map_x.at<float>(j,i) = i*sfactor;
            map_y.at<float>(j,i) = j*sfactor;
        }
    }

    remap( image, result,
        map_x, map_y,
        CV_INTER_LINEAR, BORDER_CONSTANT, Scalar(0,0,0) );
}
```

```
void warpAffine(
        InputArray src,
        OutputArray dst,
        InputArray T,
        Size dsize,
        int flags=INTER_LINEAR,
        int borderMode=BORDER_CONSTANT,
        const Scalar& borderValue=Scalar()
)
```

① $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

② $$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2^{-1} & 0 & 0 \\ 0 & 2^{-1} & 0 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

Applies a backward affine transformation to an image

$$\mathbf{I}(x, y) = \mathbf{D}(T_x(x, y), T_y(x, y))$$

If flags have a value of **WARP_INVERSE_MAP**

$$\mathbf{D}(x, y) = \mathbf{I}(T_x(x, y), T_y(x, y))$$

```
void scale( const Mat &image, Mat &result, double sfactor )
{
    result.create( image.size(), image.type() );

    Mat scaleMat( 2, 3, CV_64FC1 );
    scaleMat.at<double>(0, 0) = sfactor;
    scaleMat.at<double>(0, 1) = 0.0;
    scaleMat.at<double>(0, 2) = 0.0;
    scaleMat.at<double>(1, 0) = 0.0;
    scaleMat.at<double>(1, 1) = sfactor;
    scaleMat.at<double>(1, 2) = 0.0;

    // Apply the Affine Transform just found to the image
    warpAffine( image, result, scaleMat, result.size() );
}
```

```
Mat getAffineTransform(
      InputArray src,
      InputArray dst
)
```

calculates the 2x3 matrix of an affine transform from triangle points

$$\begin{bmatrix} x'_i \\ y'_i \end{bmatrix} = \texttt{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$dst(i) = (x'_i, y'_i), src(i) = (x_i, y_i), i = 0, 1, 2$$

예) $(0, 0) \Rightarrow (0, 0)$

$(10, 0) \Rightarrow (20, 0)$

$(0, 10) \Rightarrow (0, 20)$

$$x' = 2x \qquad \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}$$
$$y' = 2y$$

$(0, 0) \Rightarrow (0, 0)$

$(width, 0) \Rightarrow (2*width, 0)$

$(0, height) \Rightarrow (0, 2*height)$

```cpp
void scale( const Mat &image, Mat &result, float sfactor )
{
    result.create( image.size(), image.type() );

    Point2f srcTri[3];
    Point2f dstTri[3];

    // Set three points to calculate the Affine Transform
    srcTri[0] = Point2f( 0, 0 );
    srcTri[1] = Point2f( image.cols, 0 );
    srcTri[2] = Point2f( 0, image.rows );

    dstTri[0] = Point2f( 0*sfactor, 0*sfactor );
    dstTri[1] = Point2f( image.cols*sfactor, 0*sfactor );
    dstTri[2] = Point2f( 0*sfactor, image.rows*sfactor );

    // Get the Affine Transform
    Mat scaleMat; // will be  2, 3, CV_64FC1
    scaleMat = getAffineTransform( srcTri, dstTri );

    // Apply the Affine Transform just found to the image
    warpAffine( image, result, scaleMat, result.size() );
}
```

Rotating images

```
void rotate( const Mat &image, Mat &result, float angle )
{
    result.create( image.size(), image.type() );

    double pi = 3.14159265359;
    double cosVal = cos( angle * (pi/180) );
    double sinVal = sin( angle * (pi/180) );

    Mat scaleMat( 2, 3, CV_64FC1 );
    scaleMat.at<double>(0, 0) = cosVal;
    scaleMat.at<double>(0, 1) = -sinVal;
    scaleMat.at<double>(0, 2) = 0.0;
    scaleMat.at<double>(1, 0) = sinVal;
    scaleMat.at<double>(1, 1) = cosVal;
    scaleMat.at<double>(1, 2) = 0.0;

    // Apply the Affine Transform just found to the image
    warpAffine( image, result, scaleMat, result.size() );
}
```

```
void rotateCntr( const Mat &image, Mat &result, float angle )
{
    result.create( image.size(), image.type() );

    double pi = 3.14159265359;
    double cosVal = cos( angle * (pi/180) );
    double sinVal = sin( angle * (pi/180) );

    double cx = result.cols/2.0;
    double cy = result.rows/2.0;

    Mat scaleMat( 2, 3, CV_64FC1 );
    scaleMat.at<double>(0, 0) = cosVal;
    scaleMat.at<double>(0, 1) = -sinVal;
    scaleMat.at<double>(0, 2) = cx*(1-cosVal)+cy*sinVal;
    scaleMat.at<double>(1, 0) = sinVal;
    scaleMat.at<double>(1, 1) = cosVal;
    scaleMat.at<double>(1, 2) = cy*(1-cosVal)-cx*sinVal;

    // Apply the Affine Transform just found to the image
    warpAffine( image, result, scaleMat, result.size() );
}
```

```
Mat getRotationMatrix2D(
      Point2f center,
      double angle,
      double scale
)
```

Calculates an affine matrix of 2D rotation and scale

```cpp
void rotateNscale( const Mat &image, Mat &result,
        float angle, float scale ) {
    result.create( image.size(), image.type() );

    // Compute a rotation matrix with respect to
    // the center of the image
    Point center = Point( image.cols/2, image.rows/2 );

    // Get the rotation matrix with the specifications above
    Mat rotMat = getRotationMatrix2D( center, angle, scale );

    /// Rotate the warped image
    warpAffine( image, result, rotMat, image.size() );
}
```

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