Geometric Transformations of Images

Goals

- Learn how to apply different geometric transformation to images like translation, rotation, affine transformation etc.
- You will learn these functions: **cv.resize**, **cv.warpAffine**, **cv.getAffineTransform** and **cv.warpPerspective**

Transformations

Scaling

Scaling is just resizing of the image. OpenCV comes with a function **cv.resize()** for this purpose. The size of the image can be specified manually, or you can specify the scaling factor. Different interpolation methods are used. Preferable interpolation methods are **cv.INTER_AREA** for shrinking and **cv.INTER_CUBIC** (slow) & **cv.INTER_LINEAR** for zooming.

We use the function: **cv.resize (src, dst, dsize, fx = 0, fy = 0, interpolation = cv.INTER_LINEAR)**

Parameters

src input image

dst output image; it has the size dsize (when it is non-zero) or the size computed from src.size(), fx, and fy; the type of dst is the same as of src.

dsize output image size; if it equals zero, it is computed as:

$$\texttt{dsize = Size(round(fx*src.cols), round(fy*src.rows))}$$

Either dsize or both fx and fy must be non-zero.

fx scale factor along the horizontal axis; when it equals 0, it is computed as

$$\texttt{(double)dsize.width/src.cols}$$

fy scale factor along the vertical axis; when it equals 0, it is computed as

$$\texttt{(double)dsize.height/src.rows}$$

interpolation interpolation method(see **cv.InterpolationFlags**)

Try it

Affine Transform Example

<canvas> elements named **canvasInput** and **canvasOutput** have been prepared.
Click **Try it** button to see the result. You can choose another image.
You can change the code in the <textarea> to investigate more.

Try it

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let M = cv.matFromArray(2, 3, cv.CV_64FC1, [1, 0, 50, 0, 1, 100]);
let dsize = new cv.Size(src.rows, src.cols);
// You can try more different parameters
cv.warpAffine(src, dst, M, dsize, cv.INTER_LINEAR, cv.BORDER_CONSTANT, new cv.Scalar());
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); M.delete();
```

Translation

Translation is the shifting of object's location. If you know the shift in (x,y) direction, let it be (t_x, t_y), you can create the transformation matrix $\mathbf{M}$ as follows:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

We use the function: **cv.warpAffine (src, dst, M, dsize, flags = cv.INTER_LINEAR, borderMode = cv.BORDER_CONSTANT, borderValue = new cv.Scalar())**

Parameters

src	input image.
dst	output image that has the size dsize and the same type as src.
Mat	2 × 3 transformation matrix(cv.CV_64FC1 type).
dsize	size of the output image.
flags	combination of interpolation methods(see **cv.InterpolationFlags**) and the optional flag WARP_INVERSE_MAP that means that M is the inverse transformation ($\texttt{dst}\rightarrow\texttt{src}$)
borderMode	pixel extrapolation method (see **cv.BorderTypes**); when borderMode = BORDER_TRANSPARENT, it means that the pixels in the destination image corresponding to the "outliers" in the source image are not modified by the function.
borderValue	value used in case of a constant border; by default, it is 0.

rows.

Try it

Rotate Transform Example

<canvas> elements named **canvasInput** and **canvasOutput** have been prepared.
Click **Try it** button to see the result. You can choose another image.
You can change the code in the <textarea> to investigate more.

Try it

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let dsize = new cv.Size(src.rows, src.cols);
let center = new cv.Point(src.cols / 2, src.rows / 2);
// You can try more different parameters
let M = cv.getRotationMatrix2D(center, 45, 1);
cv.warpAffine(src, dst, M, dsize, cv.INTER_LINEAR, cv.BORDER_CONSTANT, new cv.Scalar());
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); M.delete();
```

Rotation

Rotation of an image for an angle θ is achieved by the transformation matrix of the form

$$M = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

But OpenCV provides scaled rotation with adjustable center of rotation so that you can rotate at any location you prefer. Modified transformation matrix is given by

$$\begin{bmatrix} \alpha & \beta & (1-\alpha) \cdot center.x - \beta \cdot center.y \\ -\beta & \alpha & \beta \cdot center.x + (1-\alpha) \cdot center.y \end{bmatrix}$$

where:

$$\begin{array}{l} \alpha = scale \cdot \cos\theta, \\ \beta = scale \cdot \sin\theta \end{array}$$

We use the function: **cv.getRotationMatrix2D (center, angle, scale)**

Parameters

- **center** center of the rotation in the source image.
- **angle** rotation angle in degrees. Positive values mean counter-clockwise rotation (the coordinate origin is assumed to be the top-left corner).
- **scale** isotropic scale factor.

Try it

Get Affine Transform Example

<canvas> elements named **canvasInput** and **canvasOutput** have been prepared.
Click **Try it** button to see the result. You can choose another image.
You can change the code in the <textarea> to investigate more.

Try it

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
// (data32F[0], data32F[1]) is the first point
// (data32F[2], data32F[3]) is the sescond point
// (data32F[4], data32F[5]) is the third point
let srcTri = cv.matFromArray(3, 1, cv.CV_32FC2, [0, 0, 0, 1, 1, 0]);
let dstTri = cv.matFromArray(3, 1, cv.CV_32FC2, [0.6, 0.2, 0.1, 1.3, 1.5, 0.3]);
let dsize = new cv.Size(src.rows, src.cols);
let M = cv.getAffineTransform(srcTri, dstTri);
```

Affine Transformation

In affine transformation, all parallel lines in the original image will still be parallel in the output image. To find the transformation matrix, we need three points from input image and their corresponding locations in output image. Then **cv.getAffineTransform** will create a 2x3 matrix which is to be passed to **cv.warpAffine**.

We use the function: **cv.getAffineTransform** **(src, dst)**

Parameters

- **src** three points([3, 1] size and cv.CV_32FC2 type) from input imag.
- **dst** three corresponding points([3, 1] size and cv.CV_32FC2 type) in output image.

Try it

Perspectiv Transform Example

<canvas> elements named **canvasInput** and **canvasOutput** have been prepared.
Click **Try it** button to see the result. You can choose another image.
You can change the code in the <textarea> to investigate more.

Try it

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let dsize = new cv.Size(src.rows, src.cols);
// (data32F[0], data32F[1]) is the first point
// (data32F[2], data32F[3]) is the sescond point
// (data32F[4], data32F[5]) is the third point
// (data32F[6], data32F[7]) is the fourth point
let srcTri = cv.matFromArray(4, 1, cv.CV_32FC2, [56, 65, 368, 52, 28, 387, 389, 390]);
let dstTri = cv.matFromArray(4, 1, cv.CV_32FC2, [0, 0, 300, 0, 0, 300, 300, 300]);
```

Perspective Transformation

For perspective transformation, you need a 3x3 transformation matrix. Straight lines will remain straight even after the transformation. To find this transformation matrix, you need 4 points on the input image and corresponding points on the output image. Among these 4 points, 3 of them should not be collinear. Then transformation matrix can be found by the function **cv.getPerspectiveTransform**. Then apply **cv.warpPerspective** with this 3x3 transformation matrix.

We use the functions: **cv.warpPerspective (src, dst, M, dsize, flags = cv.INTER_LINEAR, borderMode = cv.BORDER_CONSTANT, borderValue = new cv.Scalar())**

Parameters

src	input image.
dst	output image that has the size dsize and the same type as src.
Mat	3 × 3 transformation matrix(cv.CV_64FC1 type).
dsize	size of the output image.
flags	combination of interpolation methods (**cv.INTER_LINEAR** or **cv.INTER_NEAREST**) and the optional flag WARP_INVERSE_MAP, that sets M as the inverse transformation ($\texttt{dst}\rightarrow\texttt{src}$).
borderMode	pixel extrapolation method (**cv.BORDER_CONSTANT** or **cv.BORDER_REPLICATE**).
borderValue	value used in case of a constant border; by default, it is 0.

cv.getPerspectiveTransform (src, dst)

Parameters

src	coordinates of quadrangle vertices in the source image.
dst	coordinates of the corresponding quadrangle vertices in the destination image.

Try it

Perspectiv Transform Example

<canvas> elements named **canvasInput** and **canvasOutput** have been prepared.

Click **Try it** button to see the result. You can choose another image.

You can change the code in the <textarea> to investigate more.

Try it

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let dsize = new cv.Size(src.rows, src.cols);
// (data32F[0], data32F[1]) is the first point
// (data32F[2], data32F[3]) is the sescond point
// (data32F[4], data32F[5]) is the third point
// (data32F[6], data32F[7]) is the fourth point
let srcTri = cv.matFromArray(4, 1, cv.CV_32FC2, [56, 65, 368, 52, 28, 387, 389, 390]);
let dstTri = cv.matFromArray(4, 1, cv.CV_32FC2, [0, 0, 300, 0, 0, 300, 300, 300]);
```

Generated on Wed Jan 15 2025 23:15:45 for OpenCV by doxygen 1.8.13