UNIVERSITÀ DEGLI STUDI DI PADOVA

Image mapping

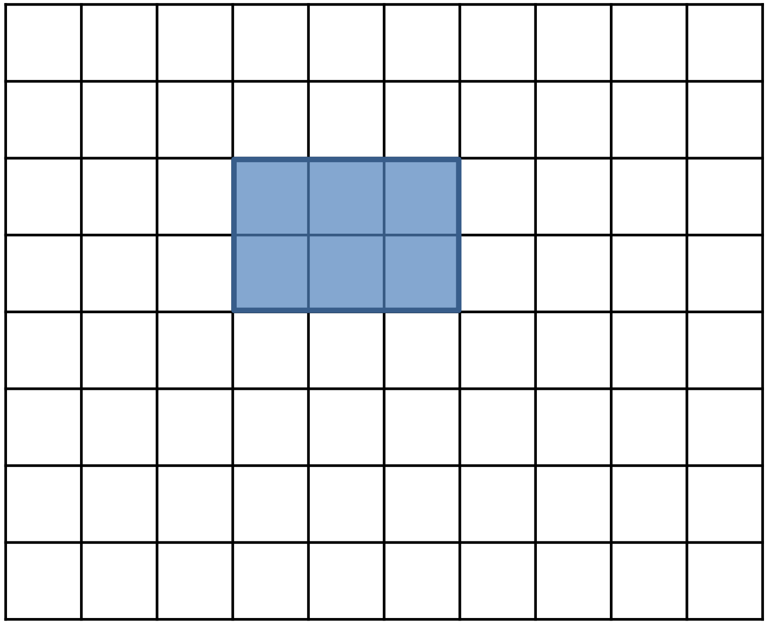
Stefano Ghidoni

Agenda

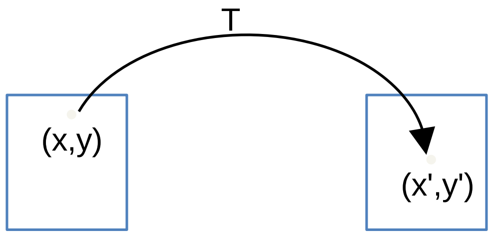
- Moving points vs moving pixels
- Forward & backward mappings
- How to code mapping

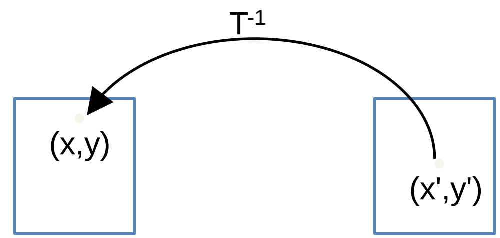
Geometric transforms

- A geometric transform is a modification of the spatial relationship among pixels
- Two steps
  - Coordinate transform $(x', y') = T\{(x, y)\}$
  - Image resampling
- Coord transform works on **geometrical points**
- Mapping/resampling goes back to pixels



Forward vs backward mapping





Forward mapping	***Backward mapping***
(x′,y′) = T{(x,y)}	*(x,y) =* T^{-1}*{(x′,y′)}*
For each (*x,y*) compute corresponding (*x′,y′*)	For each (*x′,y′*) compute corresponding (*x,y*)
Ambiguity issue: multiple points on the same (*x′,y′*)	Find each (*x′,y′*) only once
(*x′,y′*) decimals -> resampling is needed (e.g., bilinear)	(*x,y*) decimals -> resampling is needed (e.g., bilinear)
Missing pixels	Fills all pixels
Less used	Better!

not integers

Mapping images

- Images are usually remapped using the backward mapping
  - Check cv::warpAffine() function
- The low-level details of backward mapping are managed automatically by OpenCV
- We shall only provide the transformation matrix
- Other ways of specifying a transformation are provided!

cv::warpAffine

Original

Rotated

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

```
void cv::warpAffine(
cv::InputArray src, // input image
cv::OutputArray dst, // output image
cv::InputArray M, // 2x3 transform matrix
cv::Size dsize, // destination image size
int flags = cv::INTER_LINEAR, // interpolation, inverse
int borderMode = cv::BORDER_CONSTANT, // handling of missing pixels
const cv::Scalar& borderValue = cv::Scalar() // constant borders
);
```

Affine transform – tutorial

OpenCV 3.3.1
Open Source Computer Vision

Main Page | Related Pages | Modules | Namespaces | Classes | Files | Examples | Search

OpenCV Tutorials > Image Processing (imgproc module)

Affine Transformations

Goal

In this tutorial you will learn how to:

- Use the OpenCV function **cv::warpAffine** to implement simple remapping routines.
- Use the OpenCV function **cv::getRotationMatrix2D** to obtain a 2×3 rotation matrix

Theory

What is an Affine Transformation?

1. A transformation that can be expressed in the form of a *matrix multiplication* (linear transformation) followed by a *vector addition* (translation).
2. From the above, we can use an Affine Transformation to express:
   a. Rotations (linear transformation)
   b. Translations (vector addition)
   c. Scale operations (linear transformation)

   you can see that, in essence, an Affine Transformation represents a **relation** between two images.
3. The usual way to represent an Affine Transformation is by using a 2×3 matrix.

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}_{2\times 2} \quad B = \begin{bmatrix} b_{00} \\ b_{10} \end{bmatrix}_{2\times 1}$$

$$M = \begin{bmatrix} A & B \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & b_{00} \\ a_{10} & a_{11} & b_{10} \end{bmatrix}_{2\times 3}$$

Considering that we want to transform a 2D vector $X = \begin{bmatrix} x \\ y \end{bmatrix}$ by using A and B, we can do the same with:

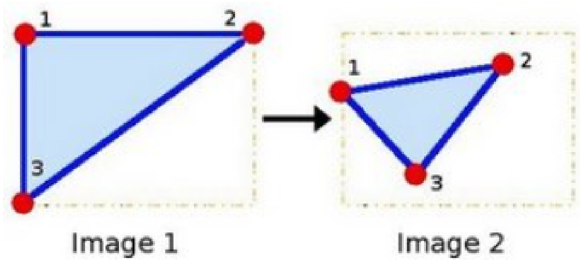
$T = A \cdot \begin{bmatrix} x \\ y \end{bmatrix} + B$ or $T = M \cdot [x, y, 1]^T$

$$T = \begin{bmatrix} a_{00}x + a_{01}y + b_{00} \\ a_{10}x + a_{11}y + b_{10} \end{bmatrix}$$

Affine transform – tutorial

How do we get an Affine Transformation?

1. We mentioned that an Affine Transformation is basically a **relation** between two images. The information about this relation can come, roughly, in two ways:
   a. We know both X and T and we also know that they are related. Then our task is to find M
   b. We know M and X. To obtain T we only need to apply $T = M \cdot X$. Our information for M may be explicit (i.e. have the 2-by-3 matrix) or it can come as a geometric relation between points.
2. Let's explain this in a better way (b). Since M relates 2 images, we can analyze the simplest case in which it relates three points in both images. Look at the figure below:



the points 1, 2 and 3 (forming a triangle in image 1) are mapped into image 2, still forming a triangle, but now they have changed notoriously. If we find the Affine Transformation with these 3 points (you can choose them as you like), then we can apply this found relation to all the pixels in an image.

Affine transform – tutorial

Code

1. **What does this program do?**
   - Loads an image
   - Applies an Affine Transform to the image. This transform is obtained from the relation between three points. We use the function **cv::warpAffine** for that purpose.
   - Applies a Rotation to the image after being transformed. This rotation is with respect to the image center
   - Waits until the user exits the program
2. The tutorial's code is shown below. You can also download it here here

```
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>

using namespace cv;
using namespace std;

const char* source_window = "Source image";
const char* warp_window = "Warp";
const char* warp_rotate_window = "Warp + Rotate";

int main( int, char** argv )
{
  Point2f srcTri[3];
  Point2f dstTri[3];

  Mat rot_mat( 2, 3, CV_32FC1 );
  Mat warp_mat( 2, 3, CV_32FC1 );
  Mat src, warp_dst, warp_rotate_dst;

  src = imread( argv[1], IMREAD_COLOR );

  warp_dst = Mat::zeros( src.rows, src.cols, src.type() );
```

Old-fashioned

Affine transform – tutorial


```
    srcTri[0] = Point2f( 0,0 );
    srcTri[1] = Point2f( src.cols - 1.f, 0 );
    srcTri[2] = Point2f( 0, src.rows - 1.f );

    dstTri[0] = Point2f( src.cols*0.0f, src.rows*0.33f );
    dstTri[1] = Point2f( src.cols*0.85f, src.rows*0.25f );
    dstTri[2] = Point2f( src.cols*0.15f, src.rows*0.7f );

    warp_mat = getAffineTransform( srcTri, dstTri );
```

Get affine transform from point correspondence

```
    warpAffine( src, warp_dst, warp_mat, warp_dst.size() );

    Point center = Point( warp_dst.cols/2, warp_dst.rows/2 );
    double angle = -50.0;
    double scale = 0.6;

    rot_mat = getRotationMatrix2D( center, angle, scale );

    warpAffine( warp_dst, warp_rotate_dst, rot_mat, warp_dst.size() );


    namedWindow( source_window, WINDOW_AUTOSIZE );
    imshow( source_window, src );

    namedWindow( warp_window, WINDOW_AUTOSIZE );
    imshow( warp_window, warp_dst );

    namedWindow( warp_rotate_window, WINDOW_AUTOSIZE );
    imshow( warp_rotate_window, warp_rotate_dst );

    waitKey(0);

    return 0;
}
```

Affine transform – tutorial

Result

1. After compiling the code above, we can give it the path of an image as argument. For instance, for a picture like:

after applying the first Affine Transform we obtain:

and finally, after applying a negative rotation (remember negative means clockwise) and a scale factor, we get:

Affine transform

- Tutorial:
  - Select matching points
  - Automatic calculation of the transformation matrix
- Pure transformations are also available
  - E.g.: rotation

getRotationMatrix2D()

§ getRotationMatrix2D()

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

Calculates an affine matrix of 2D rotation.

The function calculates the following matrix:

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

where

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

The transformation maps the rotation center to itself. If this is not the target, adjust the shift.

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.

See also

getAffineTransform, warpAffine, transform

UNIVERSITÀ DEGLI STUDI DI PADOVA

Image mapping

Stefano Ghidoni

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

INTELLIGENT AUTONOMOUS SYSTEMS LAB