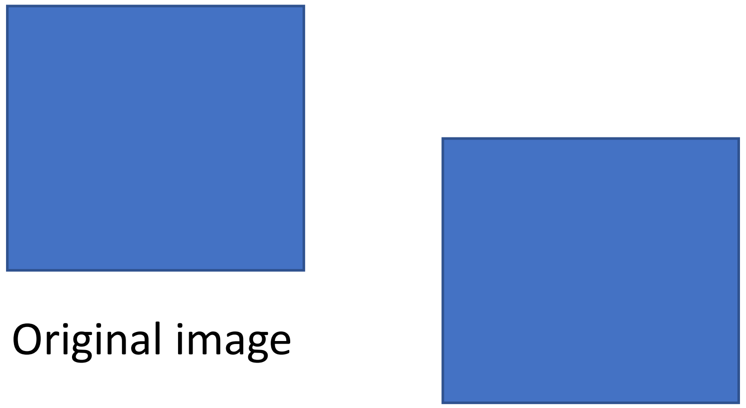
Image Transformations

Types of transformations

- Translation
- Euclidean
- Affine
- Homography

Translation

- The first image is shifted (translated) by (x , y) to obtain the second image.

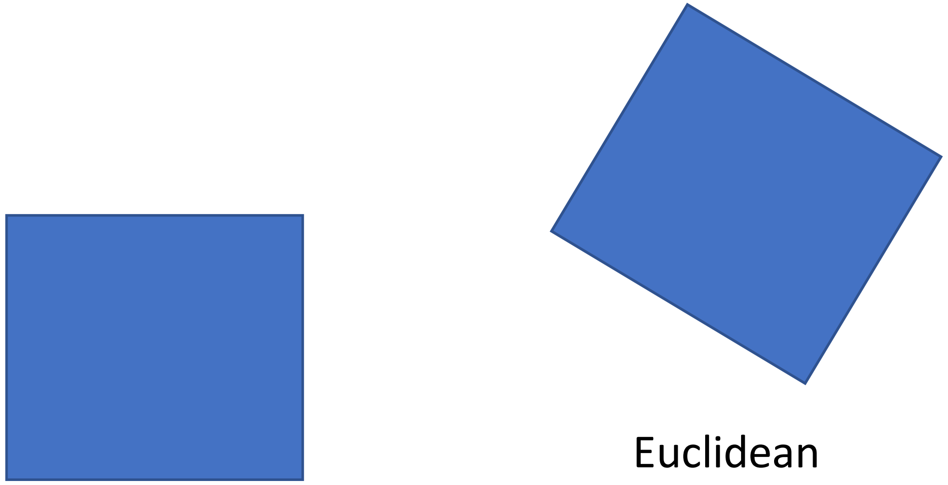


Original image

translation

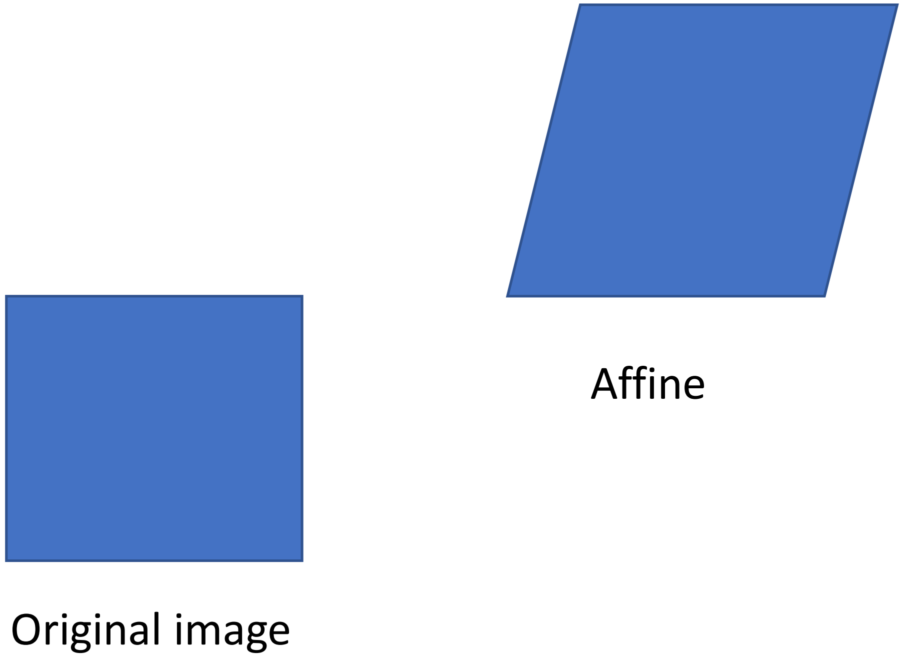
Euclidean Transformation

- A combination of translation and rotation
  - size does not change
  - parallel lines remain parallel
  - right angles remain right



Affine transformation

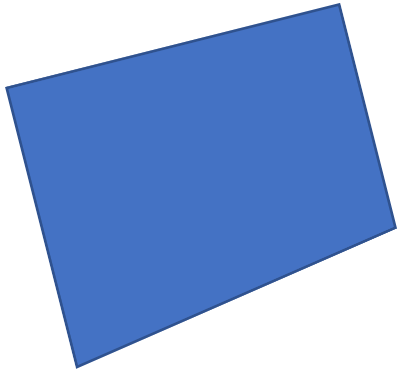
- A combination of rotation, translation, scale, and shear
  - parallel lines remain parallel
  - right angles are not preserved



Homography

- Account for a 3D effects (transform from one plane to another)

Original image



Homography

OpenCV Implementation

- Translation and Euclidean transforms are special cases of the Affine transform
  - Affine transform is stored in 2D matrix
  - Once the transform matrix is estimated, we can use cv2.warpAffine

```
dst=cv2.warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]])
```

src	input image.
dst	output image that has the size dsize and the same type as src .
M	2×3 transformation matrix.
dsize	size of the output image.
flags	combination of interpolation methods (see InterpolationFlags) and the optional flag WARP_INVERSE_MAP that means that M is the inverse transformation (dst→src).
borderMode	pixel extrapolation method (see 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.

OpenCV implementation

retval=cv2.getRotationMatrix2D(center, angle, scale)

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.

The function calculates the following matrix:

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

$$\alpha = \texttt{scale} \cdot \cos \texttt{angle},$$
$$\beta = \texttt{scale} \cdot \sin \texttt{angle}$$

OpenCV implementation

```
img = cv2.imread(sourceFilename)
# rotation
rows, cols = img.shape[:2]
M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 30, 0.5)
dst = cv2.warpAffine(img, M, (cols, rows))
```

Original image

Result

OpenCV implementation

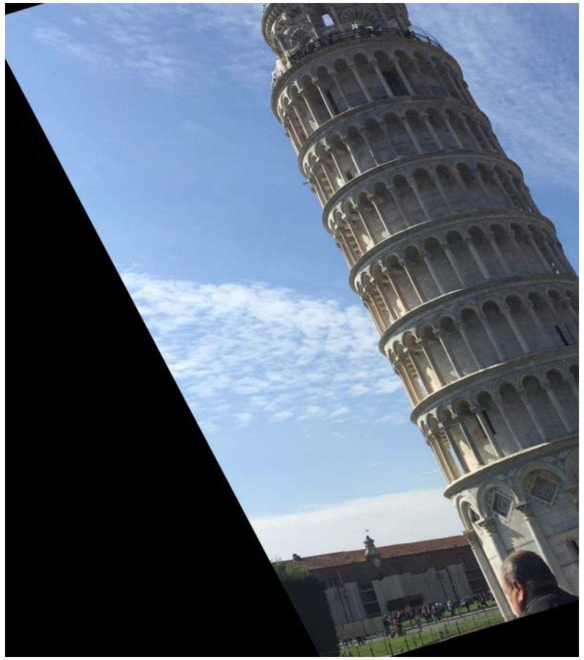
```
img = cv2.imread(sourceFilename)
rows, cols = img.shape[:2]


# affine transformation
pts1 = np.float32([[50,50],[200,50],[50,200]])
pts2 =
np.float32([[10,100],[200,50],[100,250]])

M = cv2.getAffineTransform(pts1,pts2)
dst = cv2.warpAffine(img,M,(cols,rows))

cv2.imwrite('Affine.jpg', dst)
```

Original image



Result

OpenCV Implementation

- Homography is stored in a 3 x 3 matrix.
- Once the Homography is estimated, the images can be brought into alignment using cv2.warpPerspective.

```
dst=cv2.warpPerspective(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]])
```

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

```
pts1 = np.float32([[993, 1312],[215, 2801], [3104, 813], [3848, 2367]])
pts2 = np.float32([[0, 0],[0,rows], [cols, 0], [cols, rows]])

M = cv2.getPerspectiveTransform(pts1,pts2)
dst = cv2.warpPerspective(img,M,(cols, rows))
```