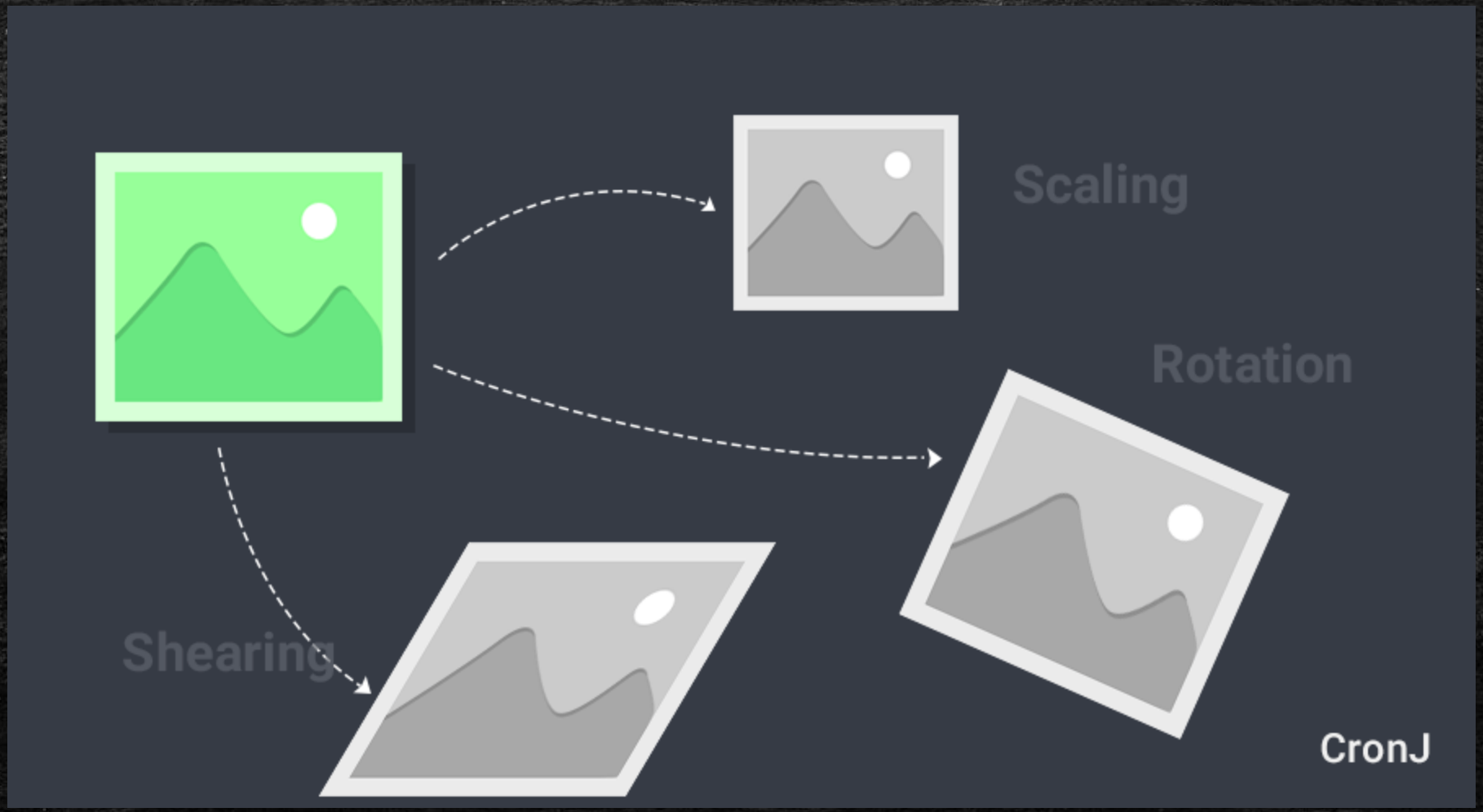
Transformations

Image Geometric Transformations



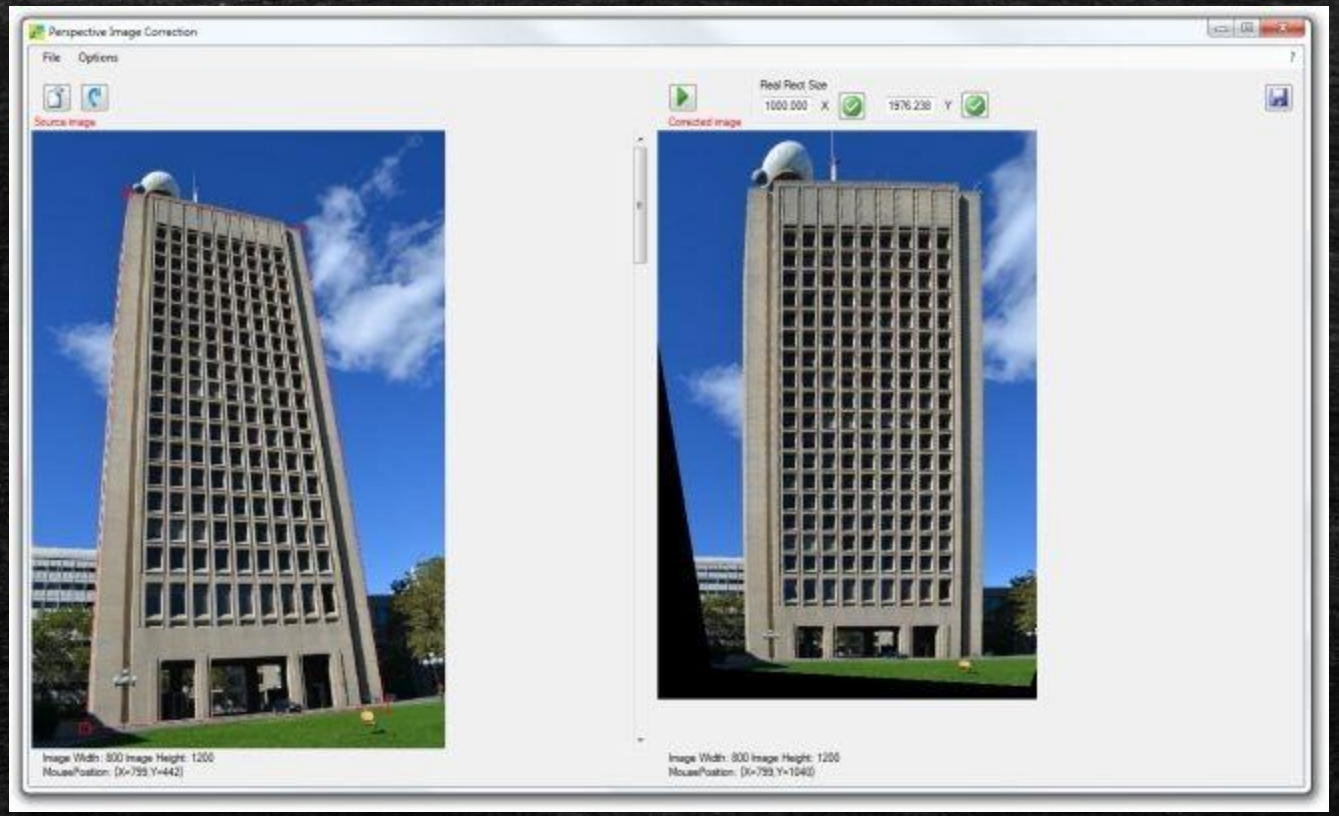
Why transformations?

Geometric transformations are widely used for a lot of applications including:

- image registration (joining/aligning images)

Why transformations?

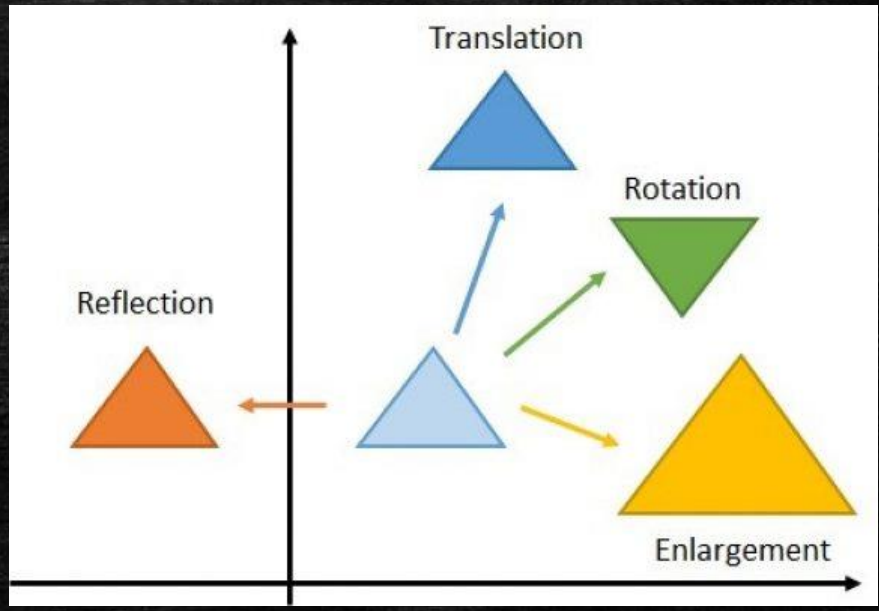
- Perspective correction



Affine Transformations

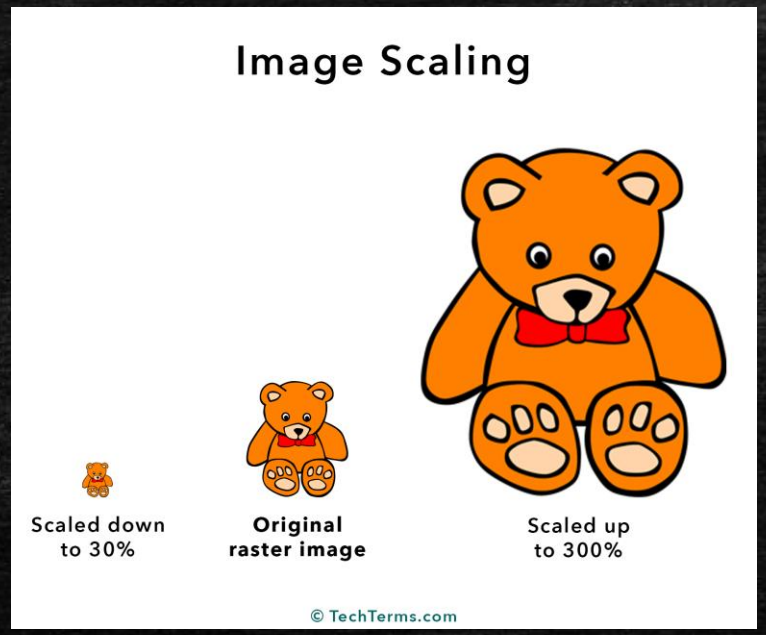
An affine transformation is any transformation that preserves collinearity (i.e., all points lying on a line initially still lie on a line after transformation) and parallelism.

Examples of affine transformations include translation, scaling, reflection, rotation, and compositions of them in any combination and sequence.



Affine - Scaling

- Scaling is just resizing of the image.
- OpenCV comes with a function cv2.resize() for this purpose. The size of the image can be specified manually, or you can specify the scaling factor.



Affine - Translation

Translation is the shifting of object's location

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 M as follows:

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

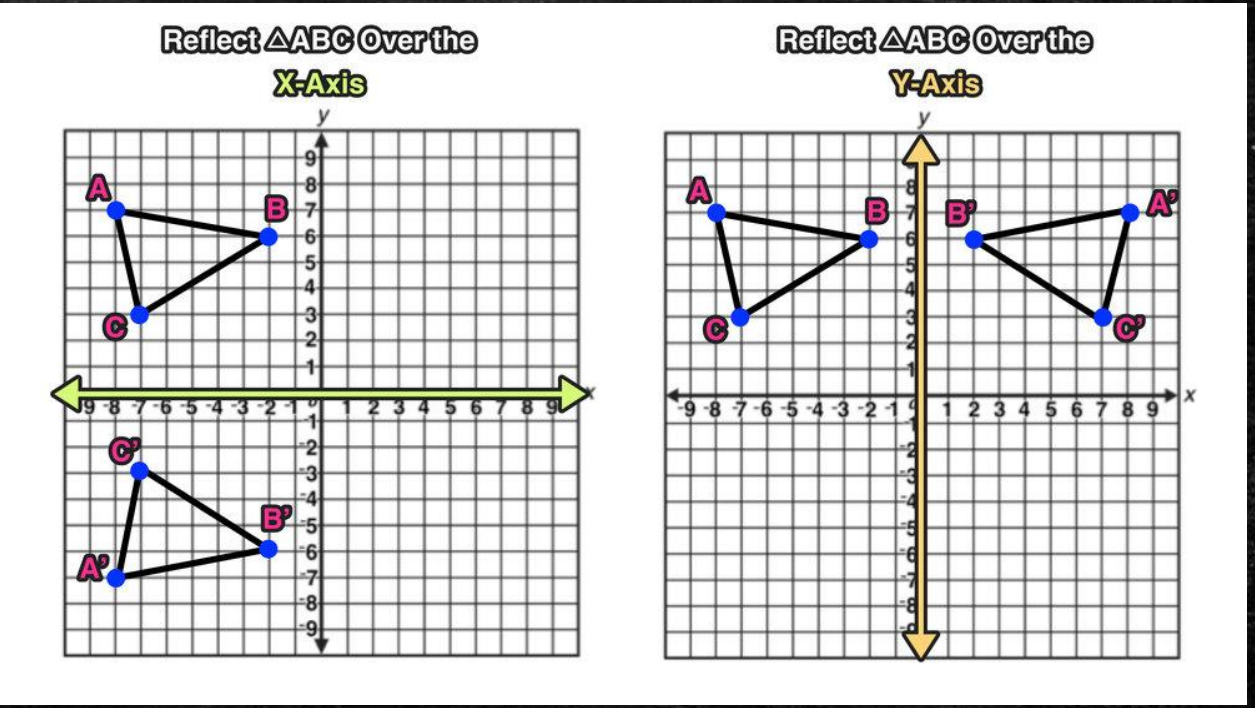
And then use the **cv2.warpAffine()** function.

Affine - Rotation

Rotation of the image
OpenCV provides a function, **cv2.getRotationMatrix2D**. You can pass to it the pivot and angle of rotation.
Note that scaling might be an issue here!

Affine - Reflection

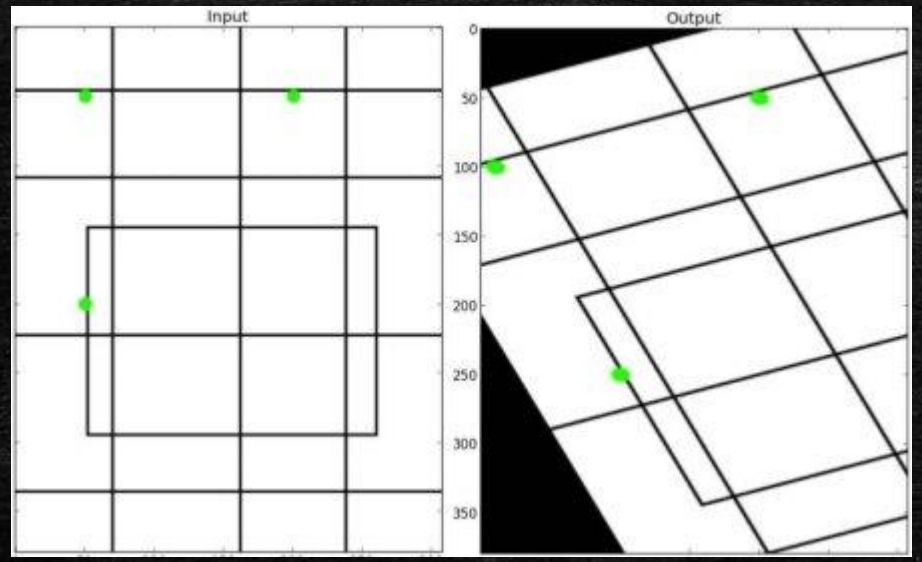
Image is reflected in x or y axis, as follows



This is also called image flipping. We have a special cv2.flip() function.

Affine Transformation

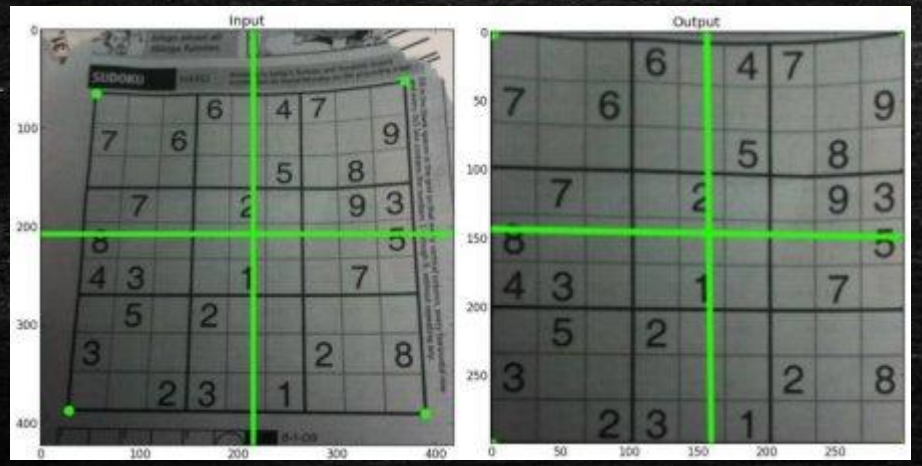
In affine transformation, all parallel lines in the original image will still be parallel in the output image. To find the transformation matrix, we need three points from input image and their corresponding locations in output image. Then cv2.getAffineTransform will create a 2x3 matrix which is to be passed to cv2.warpAffine.



Perspective Transformation

(this is NOT affine)

For perspective transformation, you need a 3x3 transformation matrix. 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 **cv2.getPerspectiveTransform**. Then apply **cv2.warpPerspective** with this 3x3 transformation matrix.



Questions?