Affine Transformation

Definition

Any transformation that can be expressed in the form of a *matrix multiplication* (linear transformation) followed by a *vector addition* (translation).

$$T = A \cdot \begin{bmatrix} x \\ y \end{bmatrix} + B$$

In which:

$$A = \begin{bmatrix} a_{00} + a_{01} \\ a_{10} + a_{11} \end{bmatrix} ; B = \begin{bmatrix} b_{00} \\ b_{10} \end{bmatrix}$$

When concatenated horizontally, this can be expressed in a larger Matrix:

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

By the definition above (*matmul + vector addition*), affine transformation can be used to achieve:

- Scaling (linear transformation)
- Rotations (linear transformation)
- Translations (vector additions)

We represent an Affine Transformation using a **2x3 matrix**.

Mathematical Definitions

Consider the goal of transforming a 2D vector $X = \begin{bmatrix} x \\ y \end{bmatrix}$ using A and B to obtain T, we can do it like such:

$$T = A \cdot \begin{bmatrix} x \\ y \end{bmatrix} + B$$

Or equivalently:

$$T = M \cdot [x, y, 1]^T = \begin{bmatrix} a_{00}x + a_{01}y + b_{00} \\ a_{10}x + a_{11}y + b_{10} \end{bmatrix}$$

Practical Examples

In `scale_04.py` from the **Examples and Illustrations** section, you'll see that the 2x3 matrix M is simple defined as such:
`np.float32([[3, 0, 0], [0, 3, 0]])`

Motivation

1. Imaging systems in the real-world are often subject to **geometric distortion**. The distortion may be introduced by perspective irregularities, physical constraints (e.g camera placements), or other reasons.
2. In the field of GIS (geographic information systems), routinely one would use affine transformation to "convert" geographic coordinates into screen coordinates such that it can **be displayed and presented** on our handheld / navigational devices.

3. One may also overlay coordinate data on a spatial data that reference a different coordinate systems; Or to **"stitch" together** different sources of data using a series of transformation

These are but a handful of examples where one may expect to see routine use of affine transformations. If you're spending any amount of time in computer vision, a high degree of familiarity with these remapping routines in OpenCV will come in very handy.

In your learn-by-building section, you will find a less-than-perfectly-digitalized map, `belitung_raw.jpg`. Your job is to apply what you've apply the necessary affine transformation to correct its perspective distortion and the resize the map accordingly.

Getting Affine Transformation

Given the importance of such a relation between two images, it should come as no surprise that `opencv` packs a number of convenient functions to help us specify this transformation. The two common use-cases are:

- 1. We **specify** our 2D vector representing the original image, X and our 2x3 transformation matrix M constructed in `numpy`.
  - Example code:

```
img = cv2.imread("our_image.png")
mat = np.float32([[3, 0, 0], [0, 3, 0]])
result = cv2.warpAffine(img, M=mat, dsize=(600, 600))
cv2.imshow("Transformed", result)
```

- 2. We **obtain** our 2x3 transformation matrix M by deriving the geometric relation using three points. Three points form a triangle, which is the minimal case required to find the affine transformation before applying the transformation to the whole image.
  - Example code:

```
img = cv2.imread("our_image.png")
coords_s = np.float32([[10, 10], [80, 10], [10, 80]])
coords_d = np.float32([[10, 10], [95, 10], [10, 80]])
mat = cv2.getAffineTransform(src=coords_s, dst=coords_d)
result = cv2.warpAffine(img, M=mat, dsize=(200, 200))
cv2.imshow("Transformed", result)
```

Have we printed out `mat` from the snippet of code above, we would see a 2x3 matrix that looks like this:

```
[[ 1.21428571  0.         -2.14285714]
 [ 0.          1.          0.        ]]
```

Dive Deeper

Let's also look at another application of `getAffineTransform` to strengthen our understanding.

Supposed we specify M to be `mat = np.float32([[1, 0, 0], [0, 1, 0]])`, what do you expect the transformation to be?

Take a minute to discuss with your classmates or refer back to the Mathematical Definitions section above and try to internalize this before moving forward.

To verify your answer, run `scale_03.py` and see if your hunch was right.

For an extra challenge, let's assume `our_image.png` is an image of 200x200. Pay attention to the specification of `mat` (M), what do you expect the outcome `result` to be?

Take a minute to discuss before moving forward.

```
img = cv2.imread("assets/our_image.png")
cv2.imshow("Original", img)

# custom transformation matrix
mat = np.float32([[3, 0, 0], [0, 3, 0]])
print(mat)
result = cv2.warpAffine(img, M=mat, dsize=(200, 200))
```

You may have expected the 2x3 matrix `mat` to have a scaling effect on our original image. However, the required argument of `dsize` in our `warpAffine()` call constrained the output to its original dimension, 200x200, thus "cropping out" only the top left corner of the image.

Supposed we'll like to see the transformed image (scaled by 3x) in its entirety, how would we have changed the value passed to the `dsize` argument?
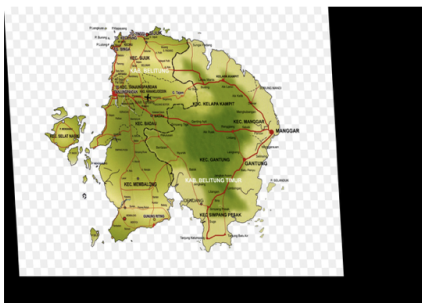
Refer to `scale_04.py` to verify that you've got this right.

Primer on Matrix Multiplication

Examples and Illustrations

Learn-by-Building

In the `homework` directory, you'll find a digital map `belitung_raw.jpg`. Your job is to apply what you've learned in this lesson to restore the map by correcting its skew and resize it appropriately.

affine transformation

Assignment 1:
Restore a digital map
of Belitung