Intro to Computer Vision with OpenCV-Python (Extra)

2017-05

Juyong Kim(juyong.kim@vision.snu.ac.kr)

SEOUL NATIONAL UNIV.
VISION & LEARNING

Contents

- Feature Extraction & Matching
  - Intro to Features / SIFT
  - SIFT Feature Extraction and Feature Matching in OpenCV
- Homography
  - Geometric Transformations
  - Homography from SIFT Matching

Feature Extraction & Matching

Understanding Features

- Feature detection / description
- SIFT

Feature Extraction in OpenCV-Python

Feature Matching

- Feature Matching
- Homography

Understanding Features

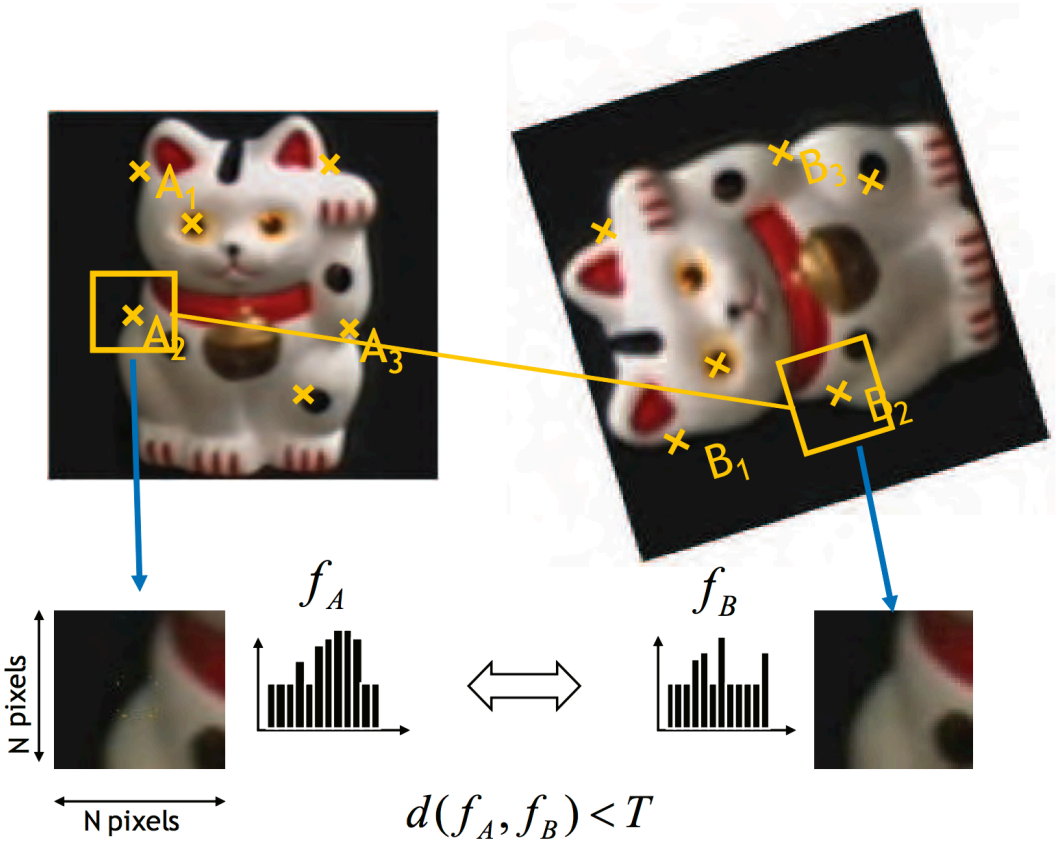
- Feature : A piece of information relevant for solving task.
  - Specific patterns which are unique, which can be easily tracked, which can be easily compared
  - Local visual feature: **Salient point** and its **representation**

- An example: Find patches A~F in the picture

Understanding Features

- Local Visual Features
  - Feature detection / Feature description

- Keypoint Matching
  - Find distinctive points
  - Define local regions around the points
  - Compute local descriptors from the region
  - Match local descriptors of two images



Feature Detection / Description

- Keypoint(Local Feature) Detection
  - Finding keypoints / interest points
  - Usually corners and blob centers
  - Harris corner detector, DoG, MSER

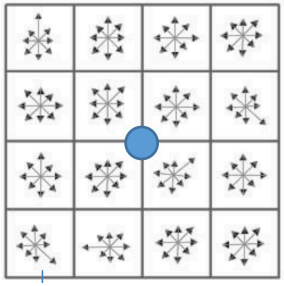
- Good Feature Detection
  - Repeatable: Robust to scaling / rotation / viewpoint change
  - Distinctive: Different features should look differently

Wikipedia, *Scale-invariant feature transform* [https://en.wikipedia.org/wiki/Scale-invariant_feature_transform]
Krystian Mikolajczyk and Cordelia Schmid, "*A Performance Evaluation of Local Descriptors*"

Feature Detection / Description

- Feature Description
  - Represent keypoints or local regions around them as a vector
  - SIFT, SURF

- ex) SIFT descriptor – 128-D vector

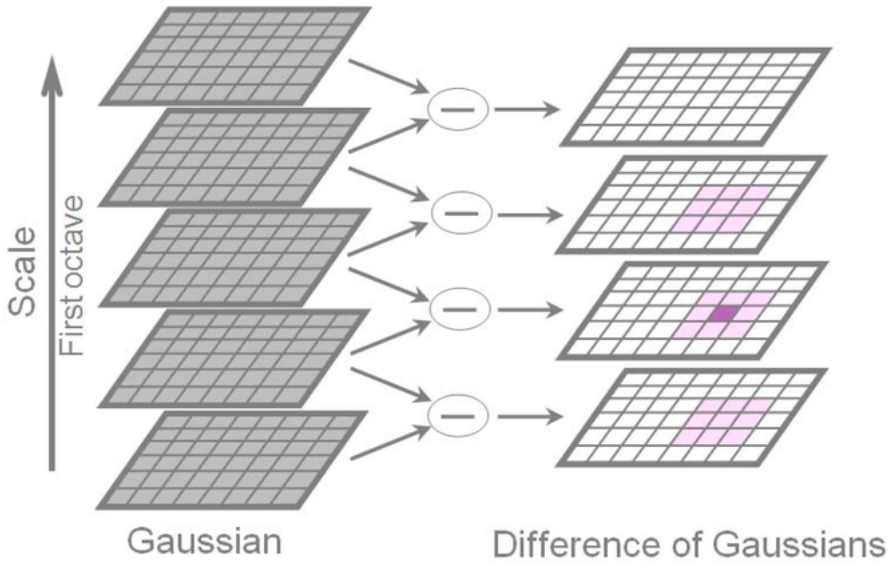


(7.0, 0.0, 0.0, 0.0, 128.0,
86.0, …, 4.0 2.0 0.0)

SIFT(Scale-Invariant Feature Transform)

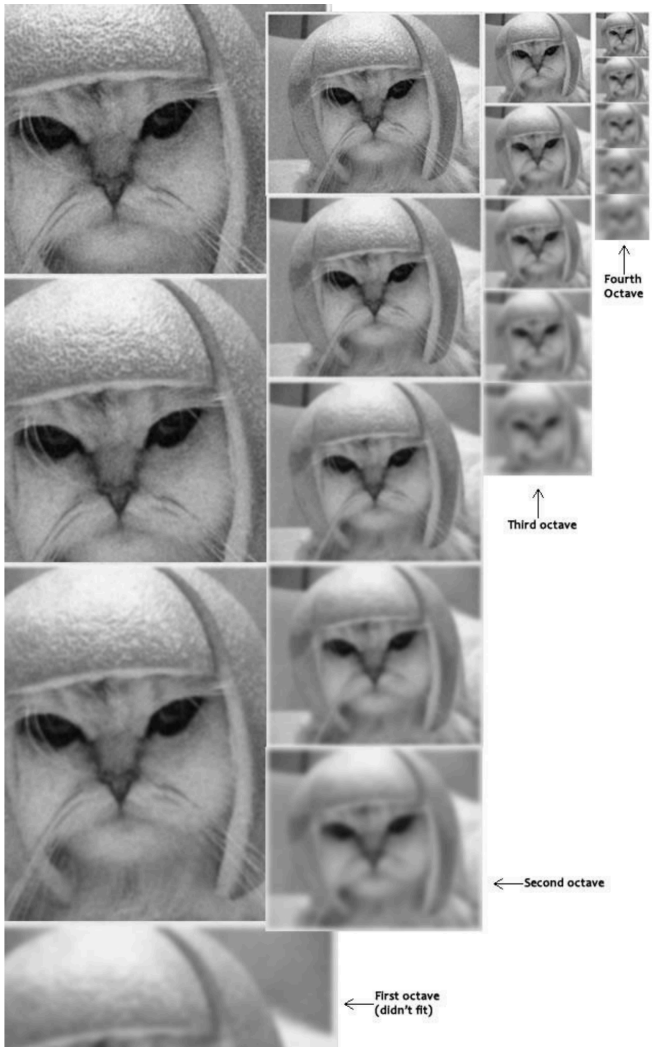
- An algorithm to **detect** and **describe** local features in images
  - Feature detector + Feature descriptor
  - Published by David Lowe in 1999

- SIFT Overview
  - Construct Scale Space
  - Take Difference of Gaussians
  - Locate DoG Extrema
  - Assign Keypoints Orientations
  - Build Keypoint Descriptors

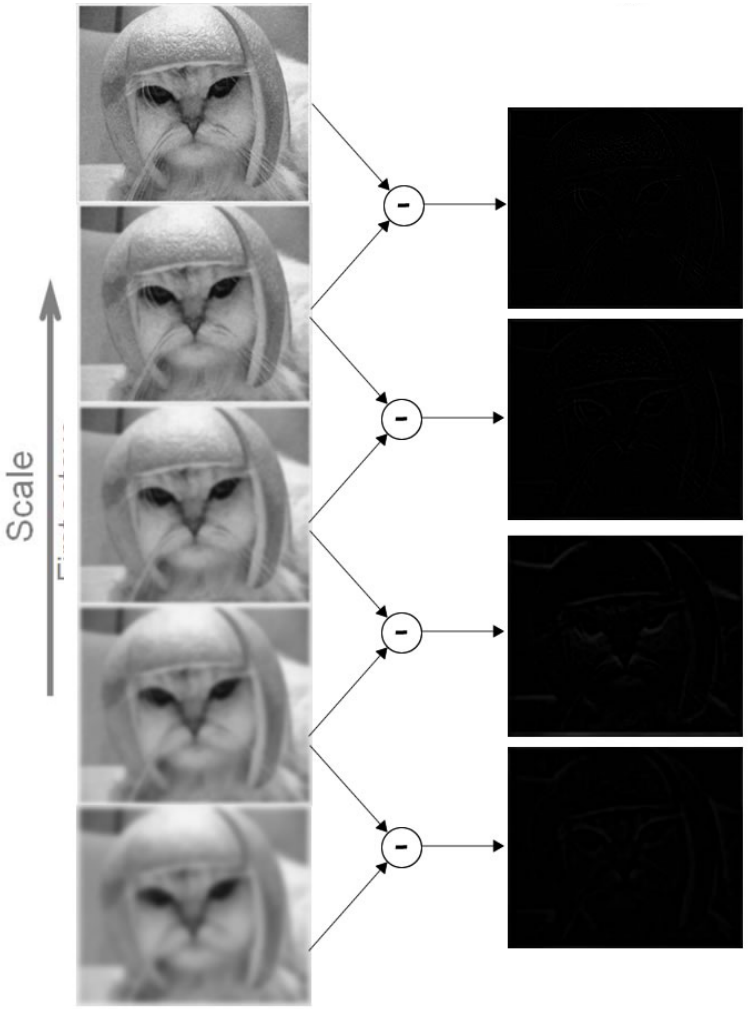


SIFT

- Construct Scale Space
  - Octave(size) / Scale(blur) / $\frac{1}{\sqrt{2}}$ factor

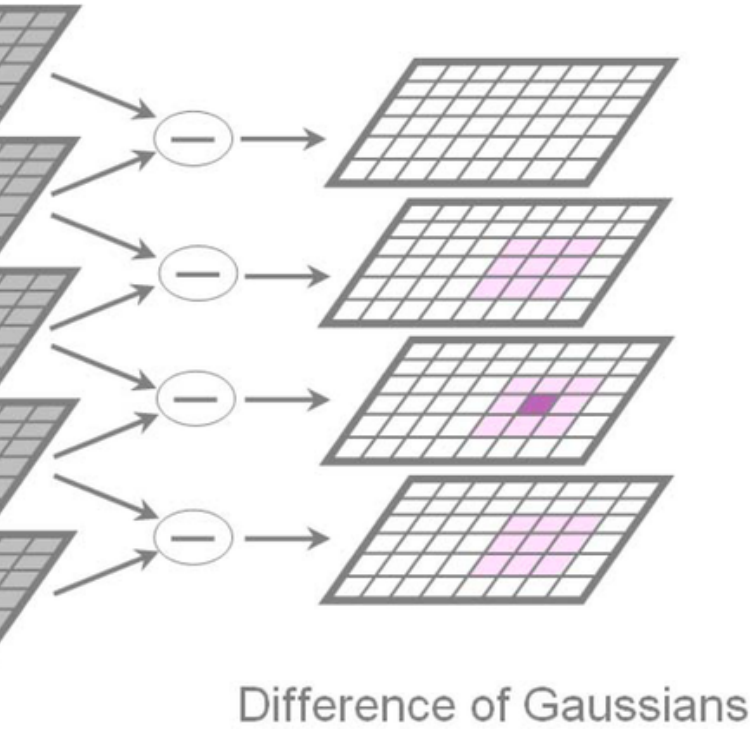


- Take Difference of Gaussian
  - For each octave



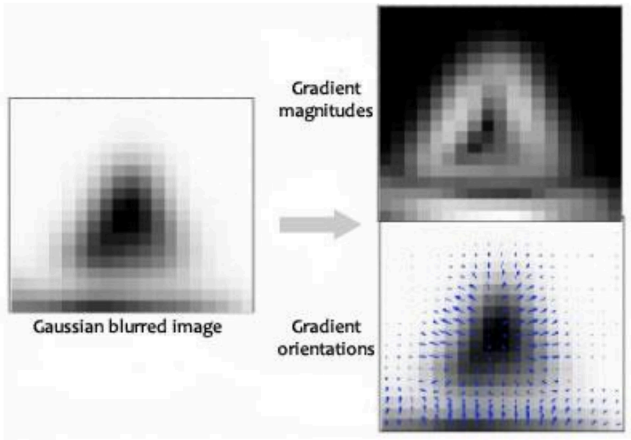
SIFT

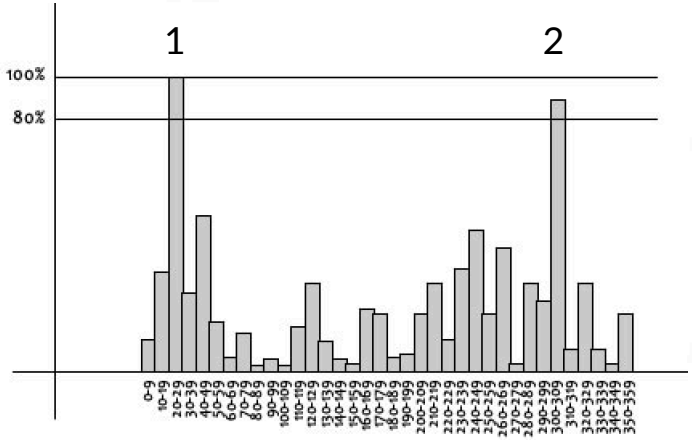
- Locate DoG Extrema
  - Look at all neighboring points (including scale)
  - Identify Min & Max (26 Comparisons)
  - Sub-pixel Localization
  - Assign Keypoints Scale

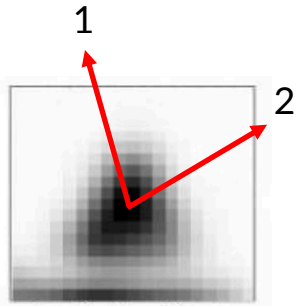


Difference of Gaussians

- Assign Keypoints Orientation
  - Histogram of gradient orientation (weighted by magnitudes)
  - All peaks with value >= 0.8×max

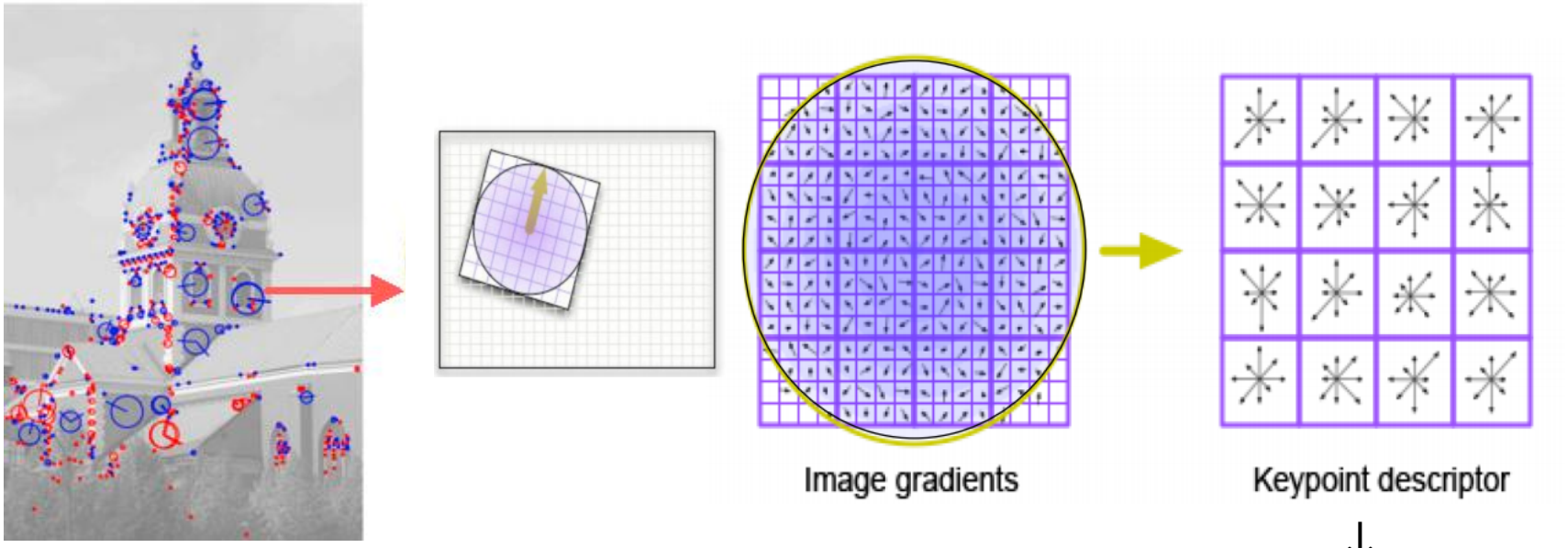






SIFT

- Build Keypoint Descriptor
  - Given a **keypoint** and corresponding **scale** and **orientation**(detection results)
  - Local patch around the keypoint
  - Generate 8-bin histogram for 4×4 grid
  - 128-D vector descriptor



(SIFT detector result)

↓

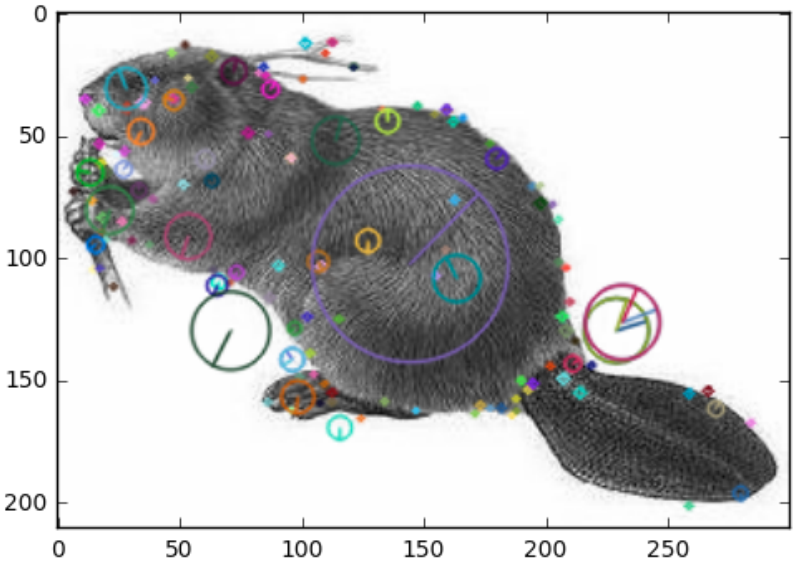
128-D vector
(SIFT descriptor)

Feature Extraction in OpenCV

- `cv2.xfeatures2d`
  - Sub-module for feature detectors / descriptors (opencv_contrib package)
  - SIFT, SURF, BRIEF ...
- Module for extracting and computing SIFT

```
# SIFT feature detector/descriptor
sift = cv2.xfeatures2d.SIFT_create()
kp, des = sift.detectAndCompute(gray, kp)
```

- Output
  - `kp`: A list of N `'cv2.KeyPoint'`
    - position, scale, orientation
  - `des`: Descriptors(N×128 Numpy array)



Feature Matching

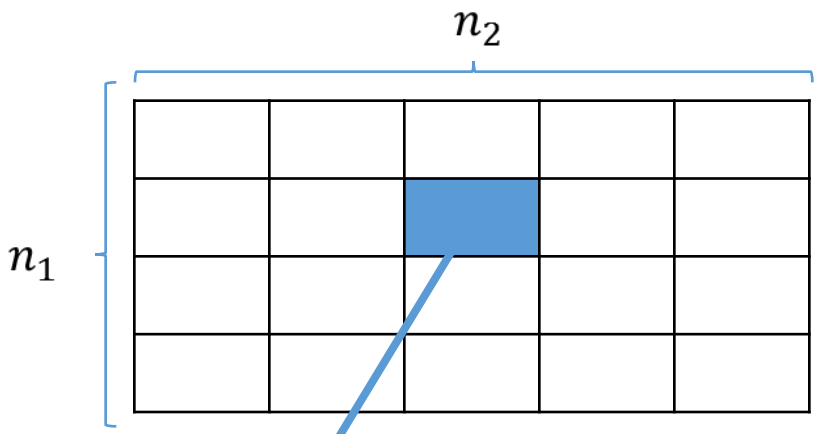
- Brute Force Matching

128-D vector v

n_1

n_2

Distance Matrix M



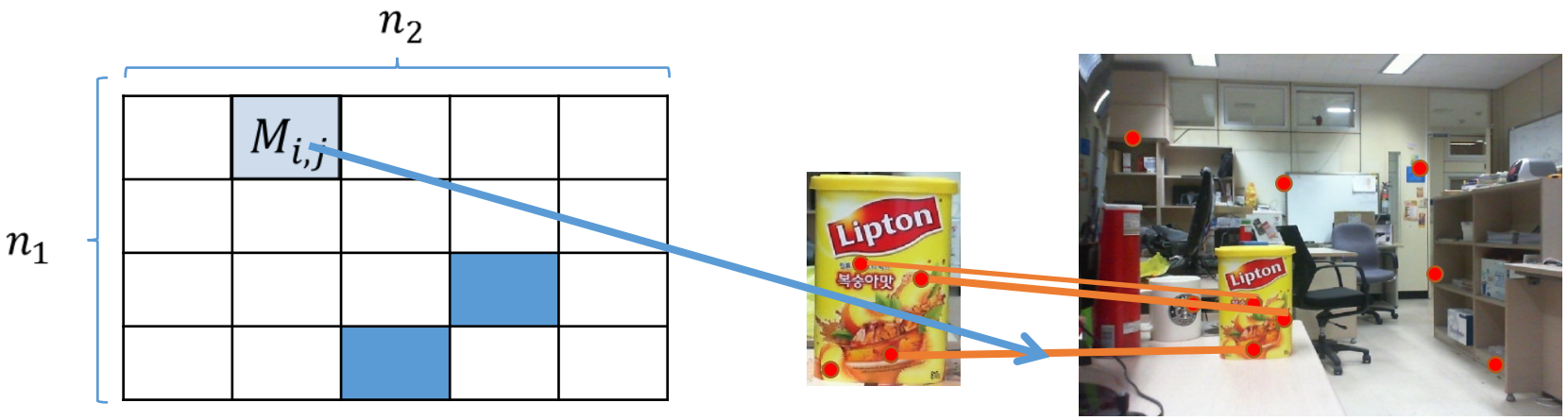
$$M_{i,j} = d(u_i, v_j)$$

Feature Distance

- L2-norm: $\|u_i - v_j\|$
- Hellinger distance: $\sqrt{1 - \sqrt{\bar{u}_i \bar{v}_j}}$

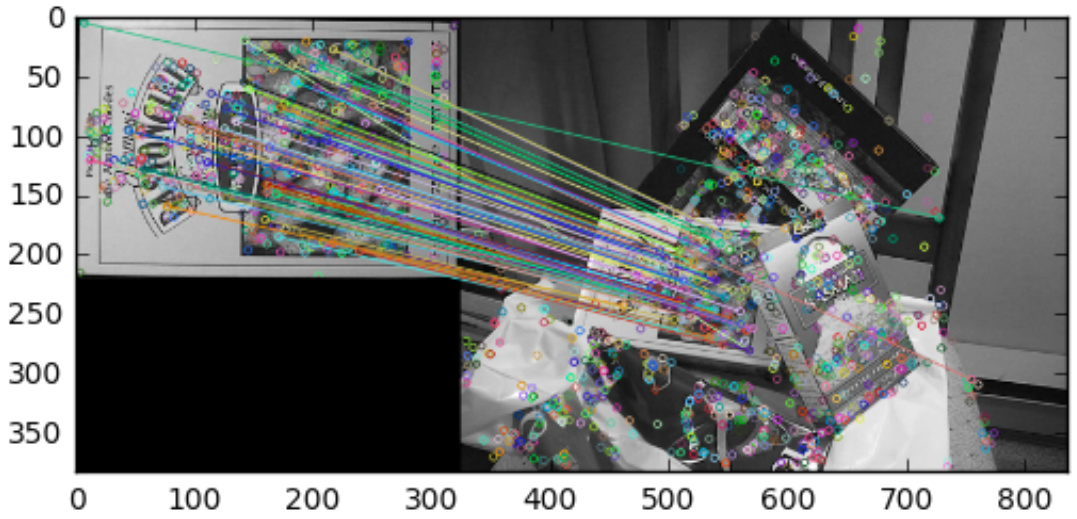
Feature Matching

- Brute Force Matching

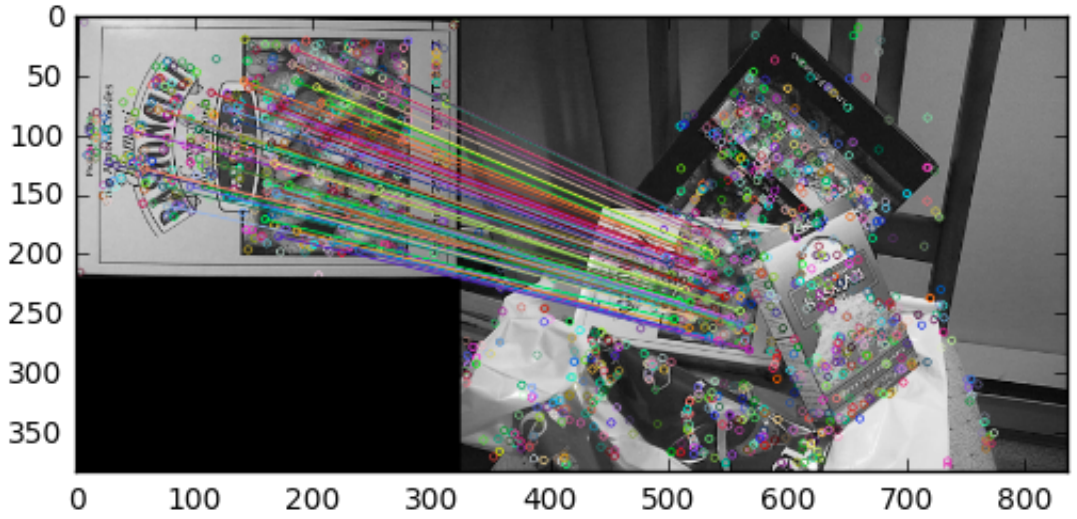


Feature Matching

- Euclidean distance



- Hellinger distance



Let’s Check the Code

extra_feature_matching.ipynb

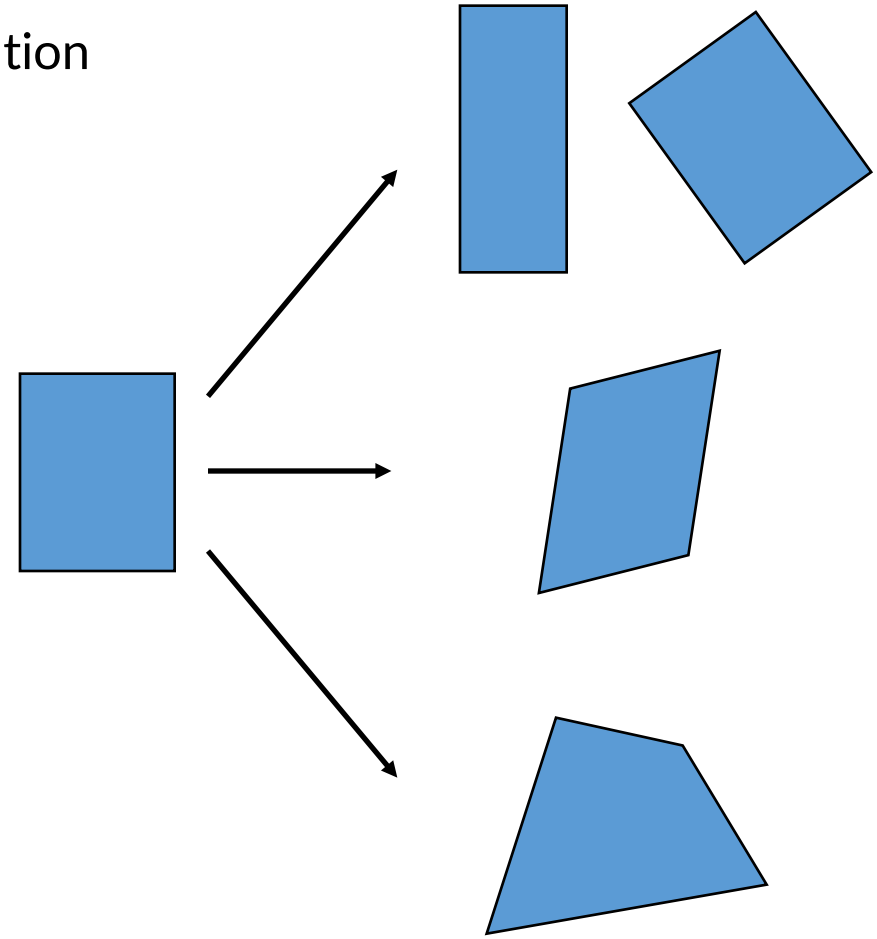
Homograpy

Geometric Transformations

Homography from SIFT Matching

Geometric Transformations

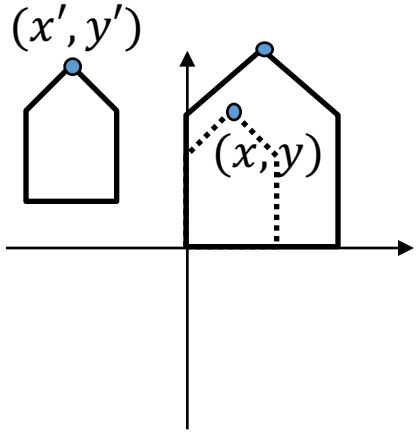
- Similarity Perspective Transformation
  - Scale / Translation / Rotation
- Affine Transformation
- Perspective Transformation
  - Homography



Geometric Transformations

- Scale

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

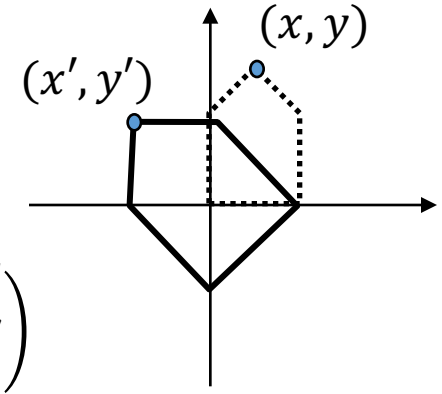


- Translation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- Rotation
  - No scale, translation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$



  - With scale, translation(rotation center on (C_x, C_y))

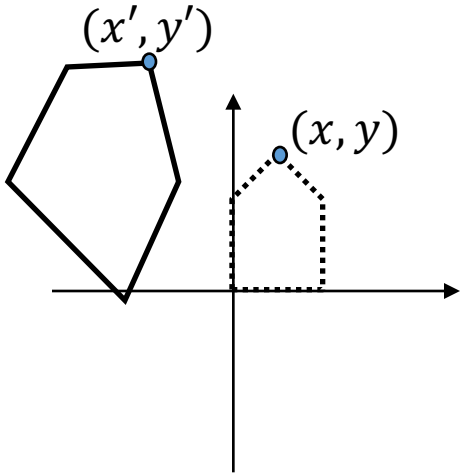
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} S\cos\theta & -S\sin\theta & C_x(1-\cos\theta) + C_y\sin\theta \\ S\sin\theta & S\cos\theta & C_x\sin\theta + C_y(1-\cos\theta) \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Geometric Transformations

- Affine transformation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b & T_x \\ c & d & T_y \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

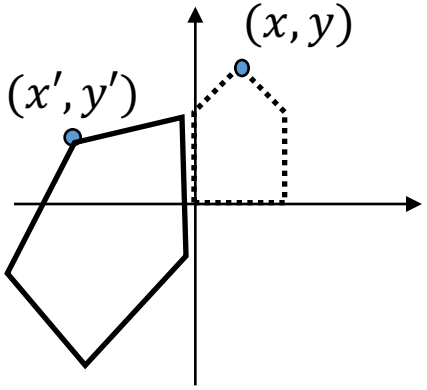
  - 6 free parameters



- Perspective Transformation

$$\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \qquad \begin{aligned} x' &= u'/w' \\ y' &= v'/w' \end{aligned}$$

  - 8 free parameters

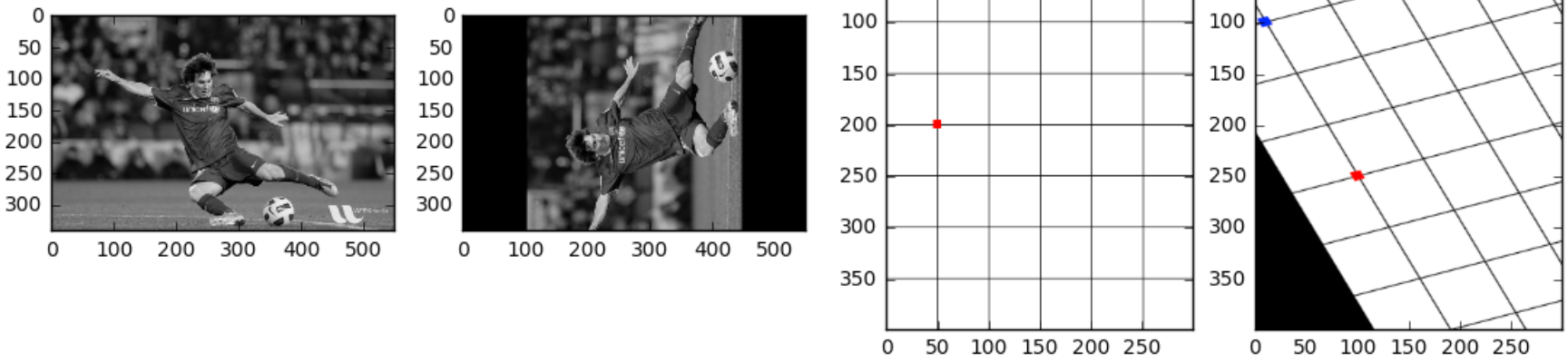


Geometric Transformations

- Affine transformation

```
pts2 = cv2.transform(pts1, M, ...) # points
dst = cv2.warpAffine(src, M, (cols,rows), ...) # image
```

  - `M`: Affine matrix(2×3)
  - `M` Can be obtained manually, or by `cv2.getAffineTransform()` and `cv2.getRotationMatrix2D()`

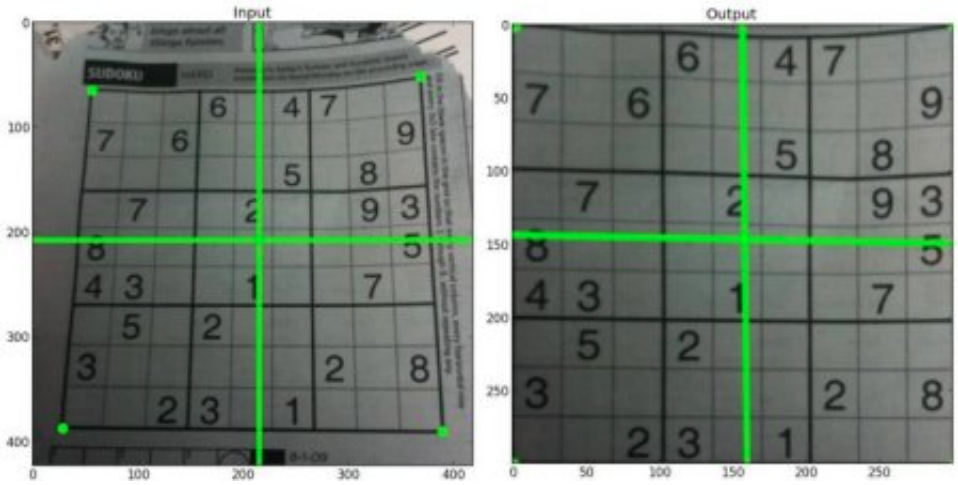


Geometric Transformations

- Perspective Transformation

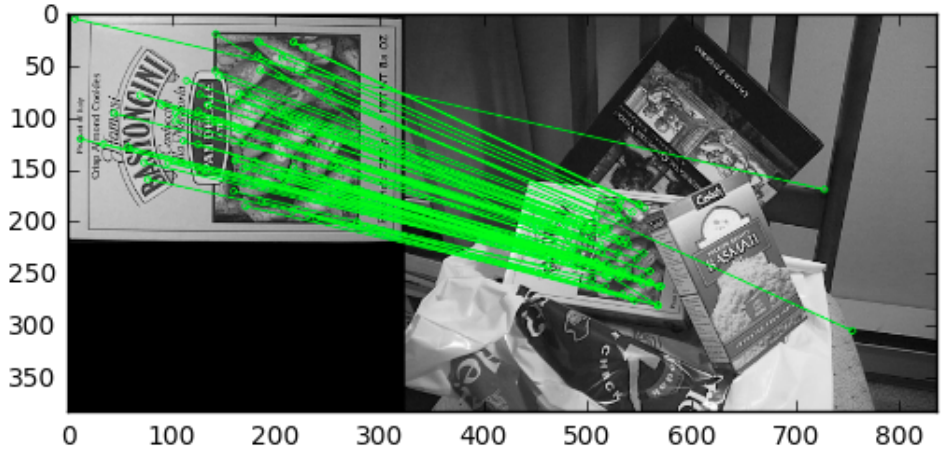
```
pts2 = cv2.perspectiveTransform(pts1, M, ...) # points
dst = cv2.warpPerspective(src, M, (cols,rows), ...) #
image
```

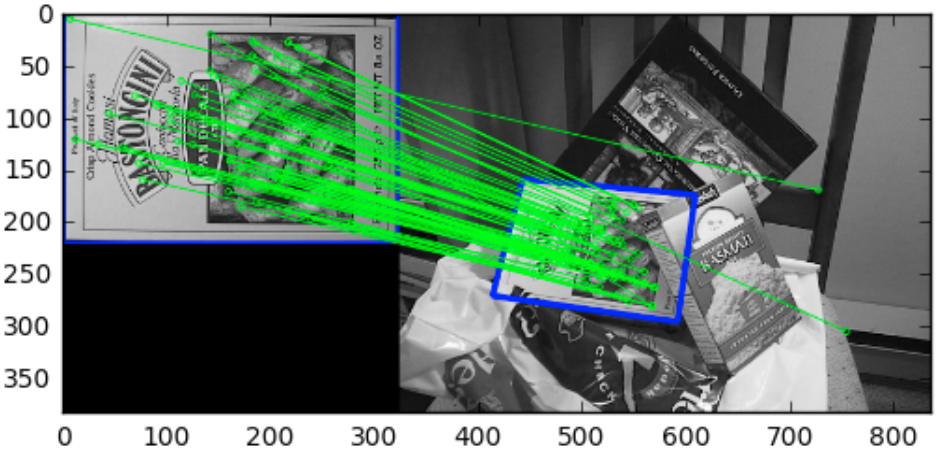
  - `M`: Homography matrix(3×3 with $M_{3,3} = 1$)
  - `M` Can be obtained manually, or by `cv2.getPerspectiveTransform()` and `cv2.findHomography()`



Homography from SIFT Matching

- Given SIFT feature matching
- Finding homography - `cv2.findHomography()`
  - Find a transformation that transforms points of a image to matched points
  - True matching → inlier
    False matching → outlier
  - `cv2.findHomography()` Uses RANSAC to find homography





RANSAC - Random sample consensus
: estimate parameters from noisy data, maintaining a consensus set

Let’s Check the Code

extra_homography.ipynb