
بسم الله الرحمن الرحيم

مبانی بینایی کامپیوتر

مدرس: محمدرضا محمدی

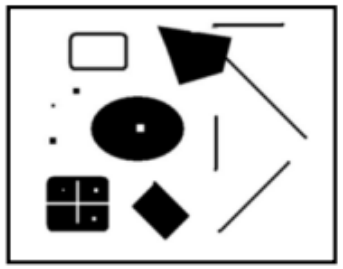
بهار ۱۴۰۳

تناظر و هم‌ترازی تصاویر

Correspondence and Image Alignment

آشکارساز Harris

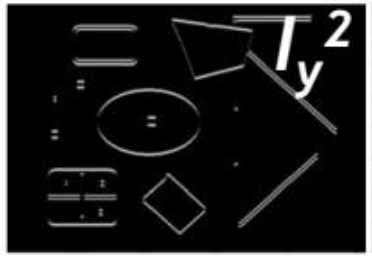
- محاسبه مشتق افقی و عمودی
- محاسبه مربع مشتق‌ها
- اعمال اثر پنجره w
- محاسبه مقادیر R
- حذف مقادیر غیر بیشینه

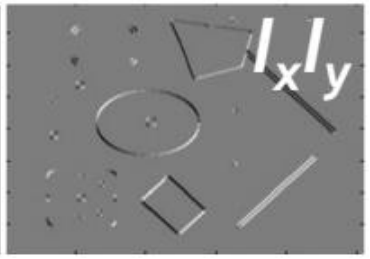


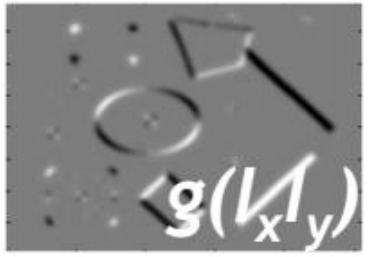
$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

$$M = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

$$R = det(M) - k\big(trace(M)\big)^2$$

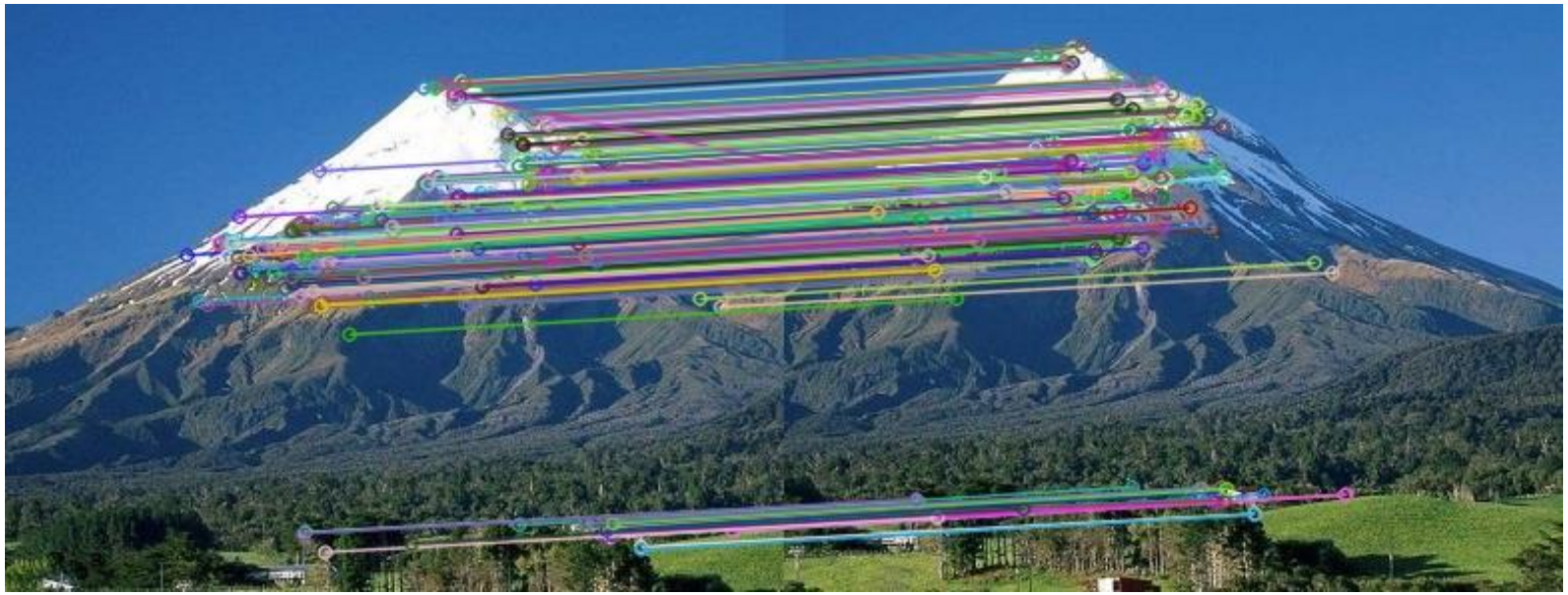






انطباق نقاط کلیدی

- پس از استخراج نقاط کلیدی از دو تصویر، نیاز است تا نقاط متناظر با یکدیگر مشخص شوند
- برای این منظور، ابتدا برای هر نقطه ویژگی یک توصیفگر محاسبه می‌شود
- سپس، دو به دوی توصیفگرها از دو تصویر مقایسه می‌شوند و مشابه‌ترین توصیفگرها به عنوان نقاط متناظر انتخاب می‌شوند
- برای جلوگیری از تناظریابی اشتباه، حد آستانه‌ای بر روی میزان مشابهت گذاشته می‌شود

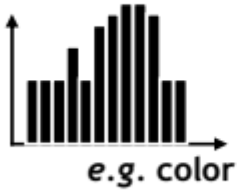


از نقاط به ناحیه‌ها

- آشکارساز Harris نقاط کلیدی را مشخص می‌کنند
  - مکان‌یابی دقیق
  - تکرارپذیری بالا
- به منظور مقایسه این نقاط، نیاز داریم تا هر نقطه توسط یک توصیفگر بر روی ناحیه اطراف خود بازنمایی شود
  - چطور می‌توانیم یک ناحیه مستقل از مقیاس تعریف کنیم؟

مقایسه ناحیه‌ها

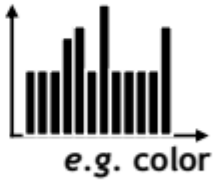
f_A



Similarity measure

$\neq$

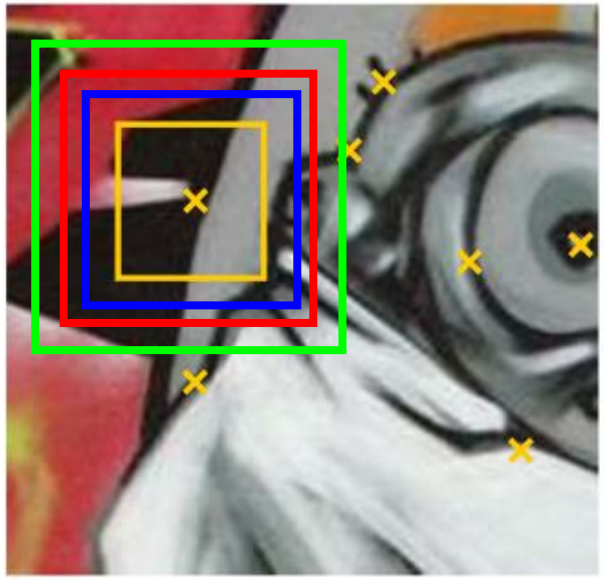
f_B

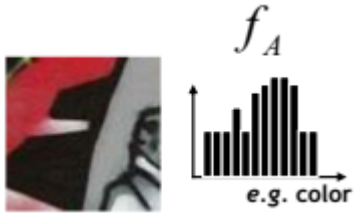


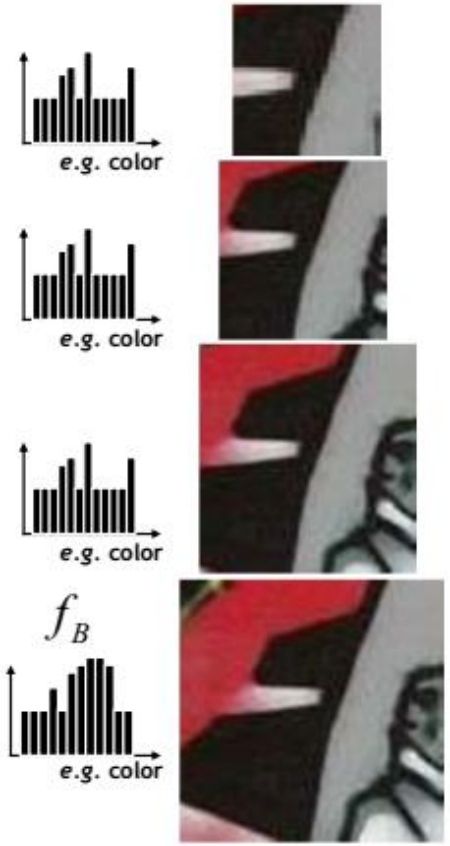
$d(f_A, f_B)$

رویکرد Naïve: جستجوی کامل

- روش چند مقیاسه:
  - توصیفگرها برای ناحیه‌های با ابعاد متفاوت محاسبه و مقایسه شوند
  - این محاسبات برای هر جفت نقاط از دو تصویر بسیار هزینه‌بر خواهد بود

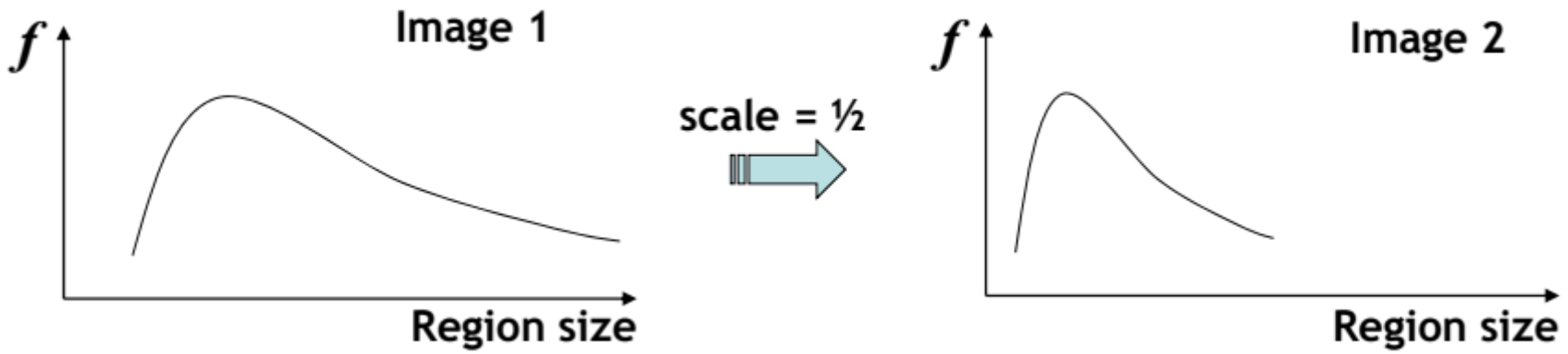






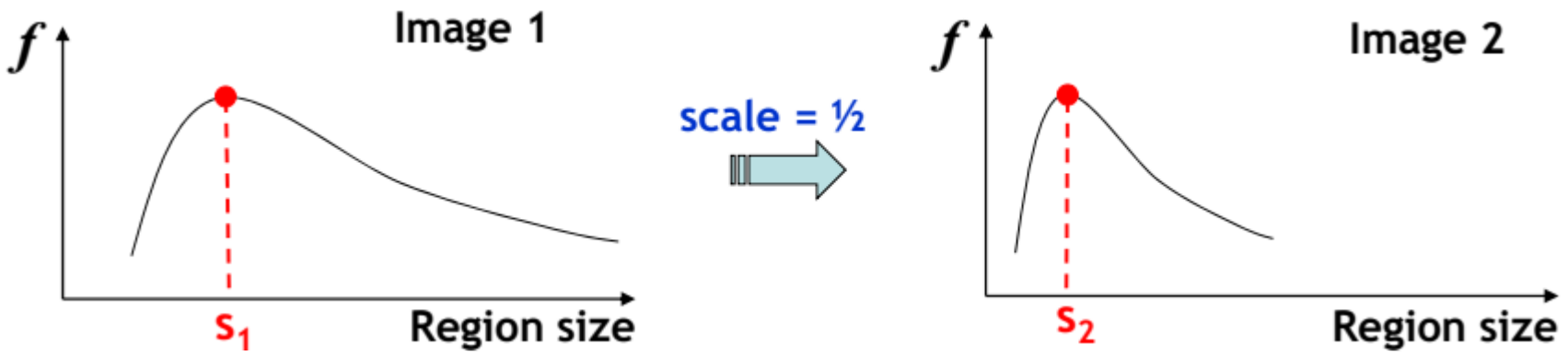
انتخاب خودکار مقیاس

- تابعی طراحی کنیم که مستقل از مقیاس باشد
  - برای ناحیه‌های متناظر یکسان باشد حتی اگر مقیاس متفاوتی داشته باشند
  - به عنوان مثال، میانگین شدت روشنایی مستقل از مقیاس است و برای دو ناحیه متناظر مقدار یکسانی دارد
- برای یک نقطه در یک تصویر، می‌توان میانگین شدت روشنایی را به صورت تابعی از ابعاد ناحیه لحاظ کرد

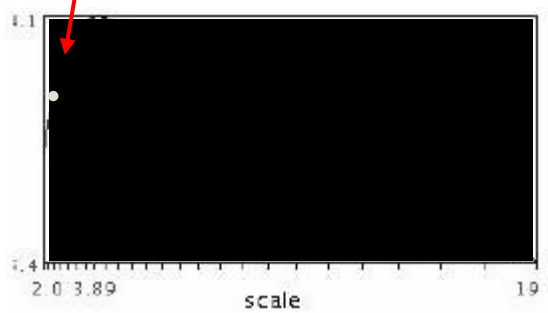


انتخاب خودکار مقیاس

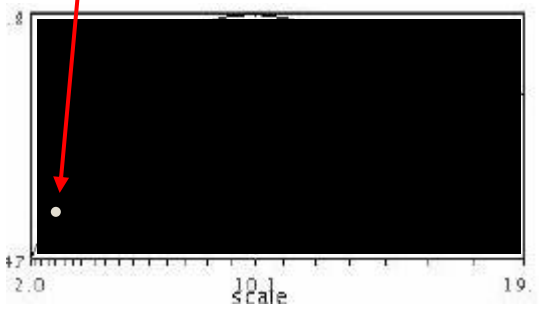
- ابعاد مربوط به بیشینه محلی در این منحنی متناسب با مقیاس خواهد بود
- نکته مهم این است که محاسبات مربوط به یافتن اندازه ناحیه در هر تصویر و برای هر نقطه کلیدی به صورت مستقل انجام می‌شود



انتخاب خودکار مقیاس

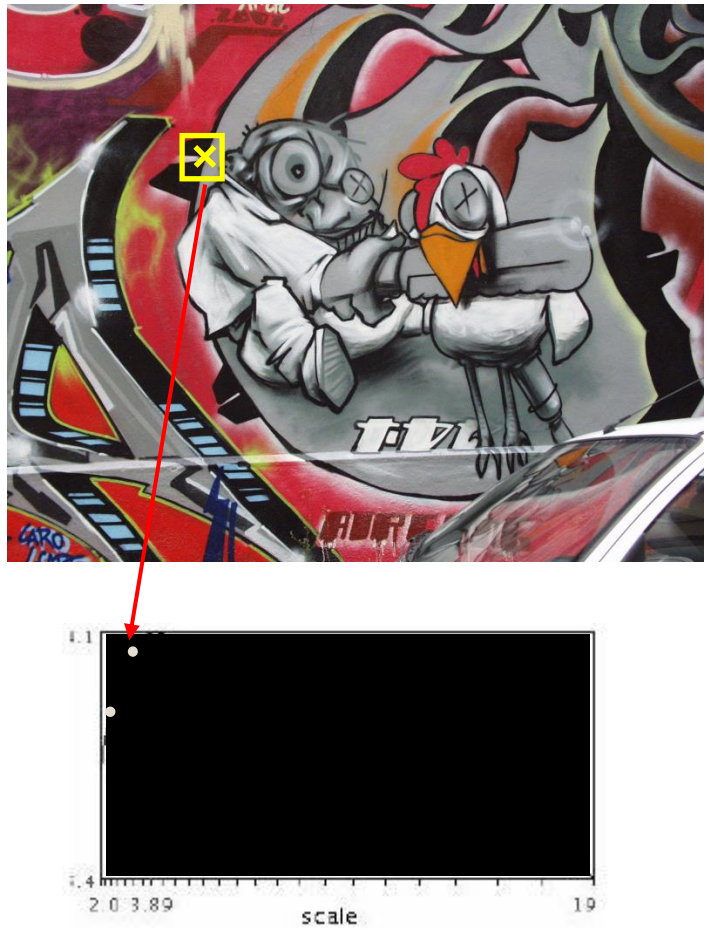


$f(I_{i_1 \ldots i_m}(x,\sigma))$



$f(I_{i_1 \ldots i_m}(x',\sigma))$

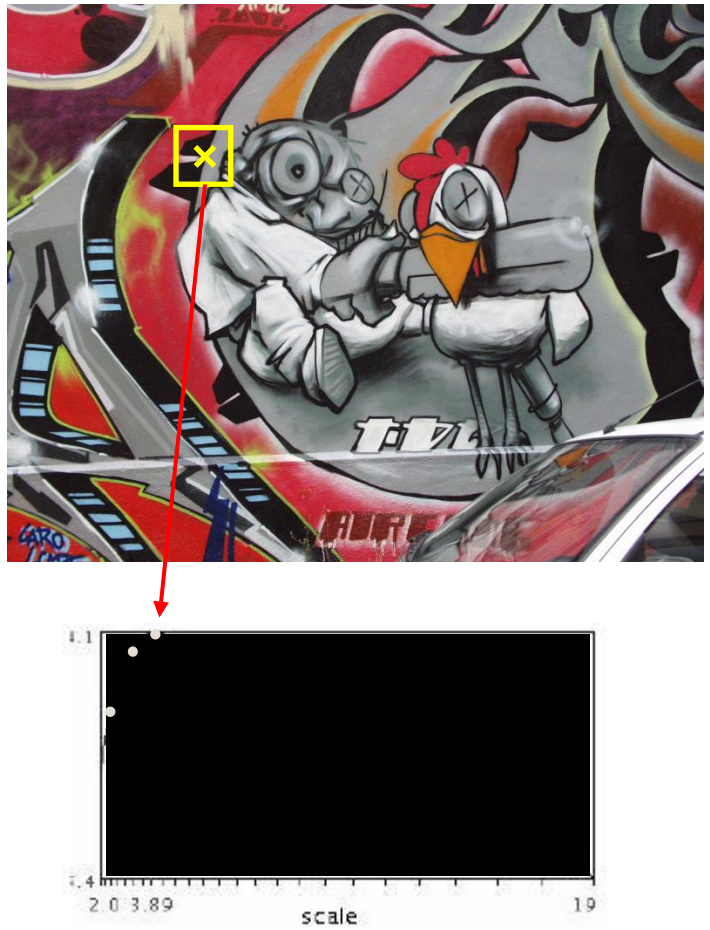
انتخاب خودکار مقیاس



$f(I_{i_1 \dots i_m}(x,\sigma))$

$f(I_{i_1 \dots i_m}(x',\sigma))$

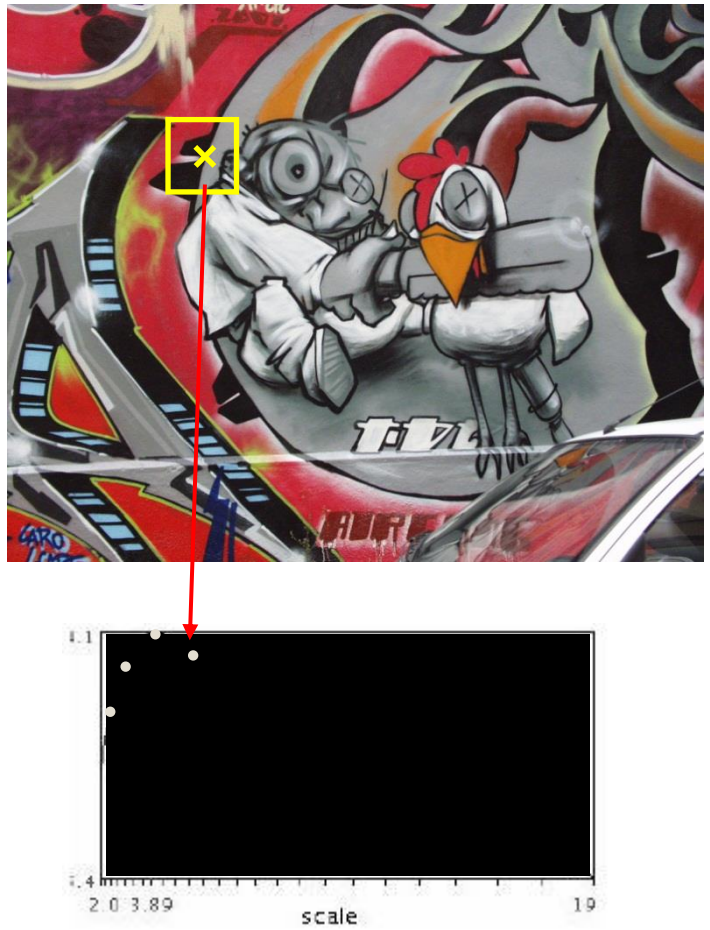
انتخاب خودکار مقیاس



$f(I_{i_1 \dots i_m}(x, \sigma))$

$f(I_{i_1 \dots i_m}(x', \sigma))$

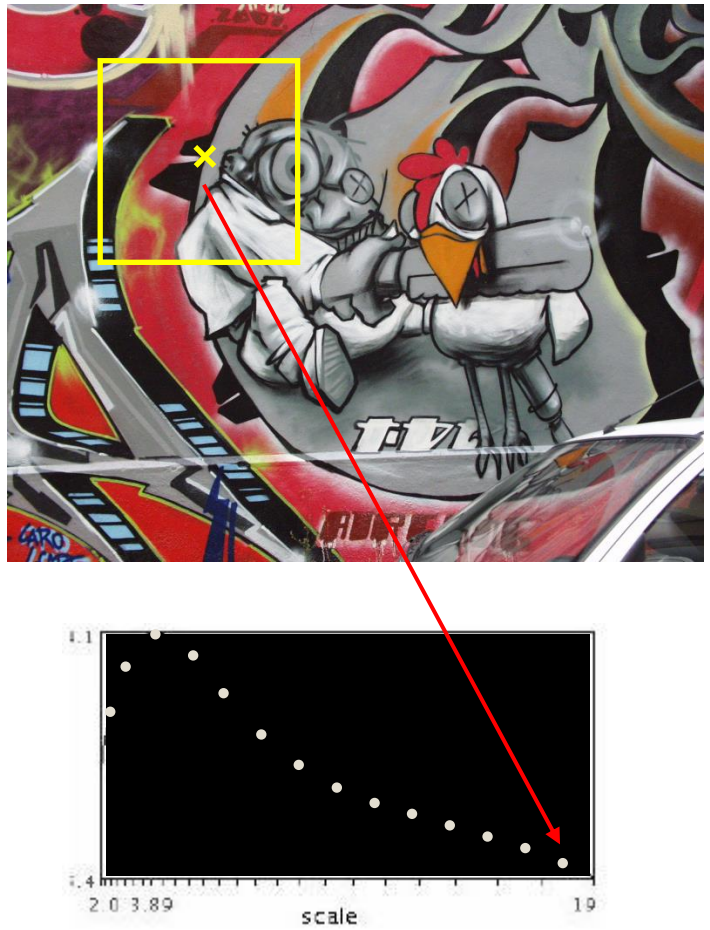
انتخاب خودکار مقیاس



$f(I_{i_1 \dots i_m}(x, \sigma))$

$f(I_{i_1 \dots i_m}(x', \sigma))$

انتخاب خودکار مقیاس

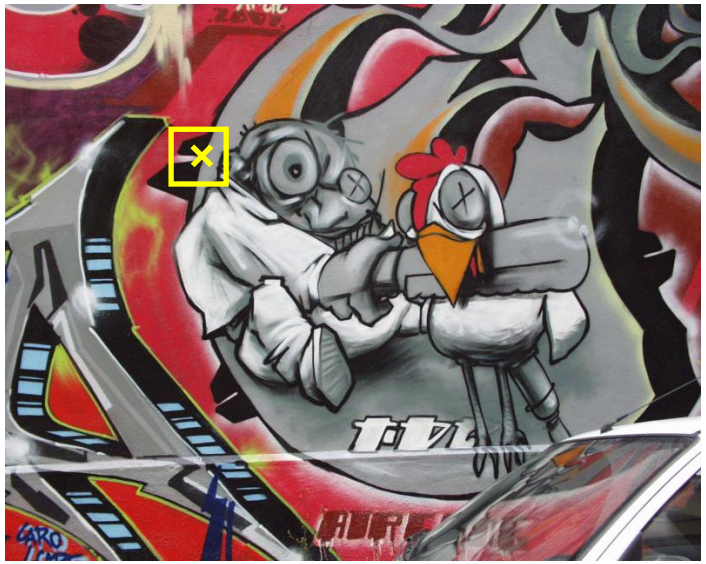


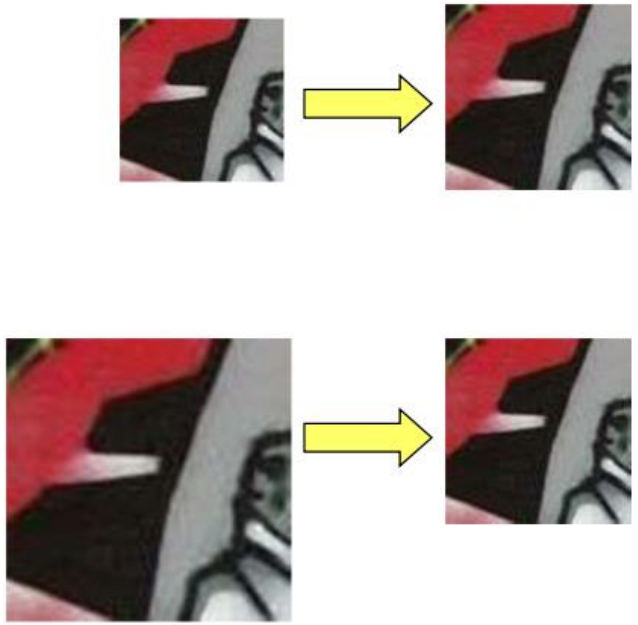
$f(I_{i_1 \dots i_m}(x,\sigma))$

$f(I_{i_1 \dots i_m}(x',\sigma))$

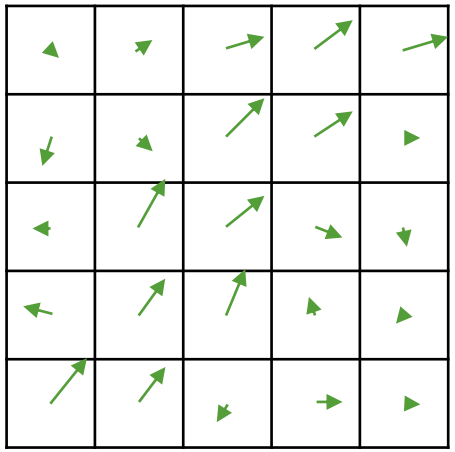
انتخاب خودکار مقیاس

- پس از انتخاب ابعاد مناسب، ناحیه‌ها را به یک اندازه مشخص نرمالیزه می‌کنیم تا به خوبی قابل مقایسه باشند

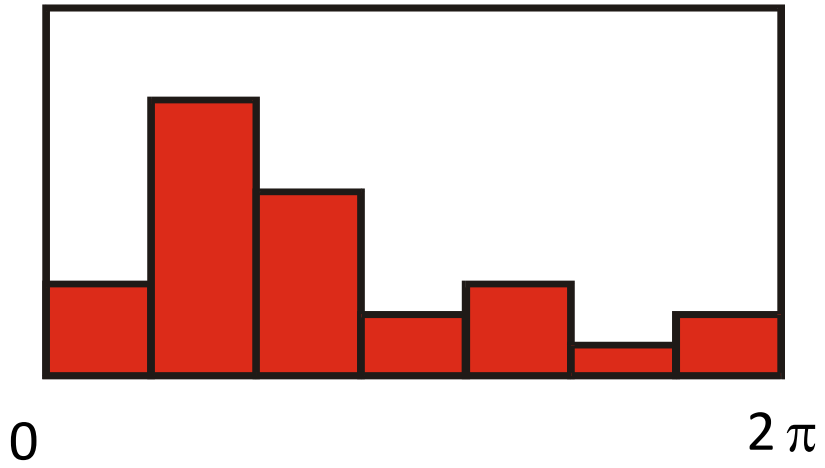




انتخاب خودکار جهت

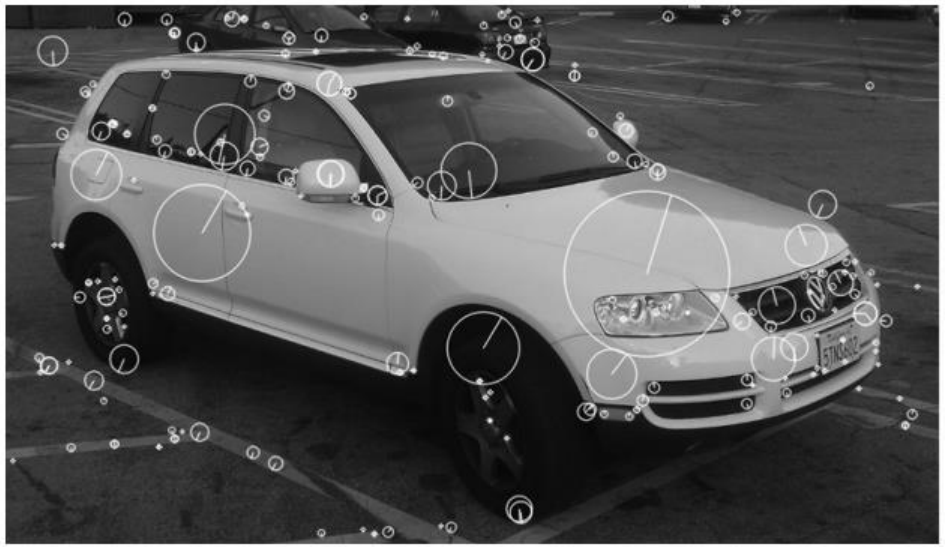


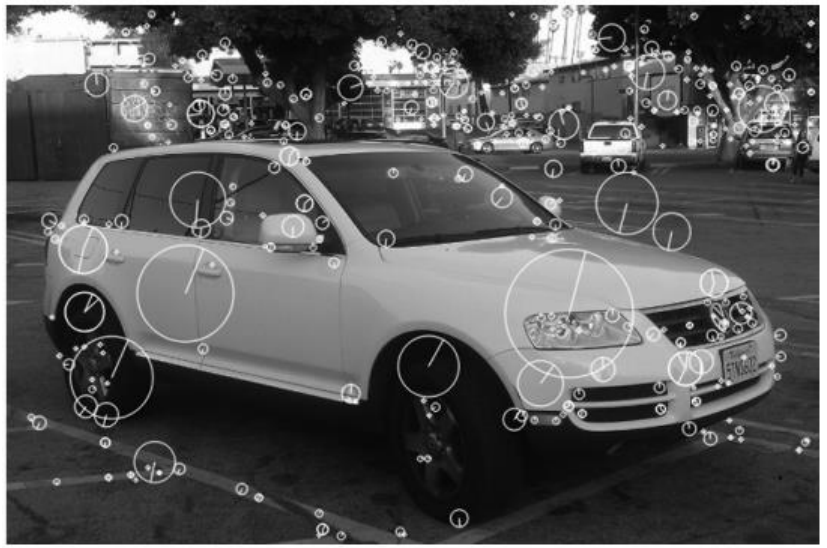
- نیاز است تابعی طراحی کنیم که متناسب با مقدار چرخش تصویر، تغییر کند
- می‌توان ابتدا هیستوگرام جهت گرادیان را محاسبه کرد
- سپس، جهت غالب در این هیستوگرام را انتخاب کرد



نقاط کلیدی

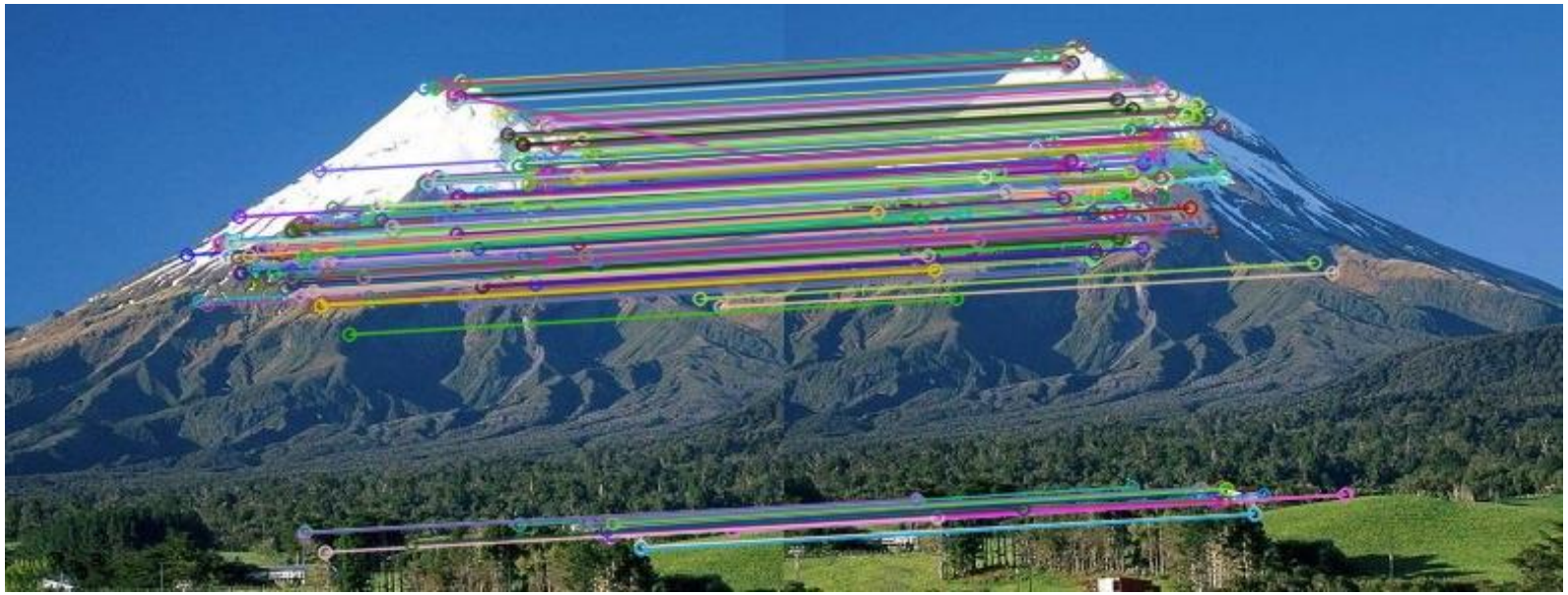
- روش‌های پرکاربرد برای اسخراج نقاط کلیدی و توصیفگرهای آنها عبارتند از:
  - SIFT
  - SURF
  - ORB





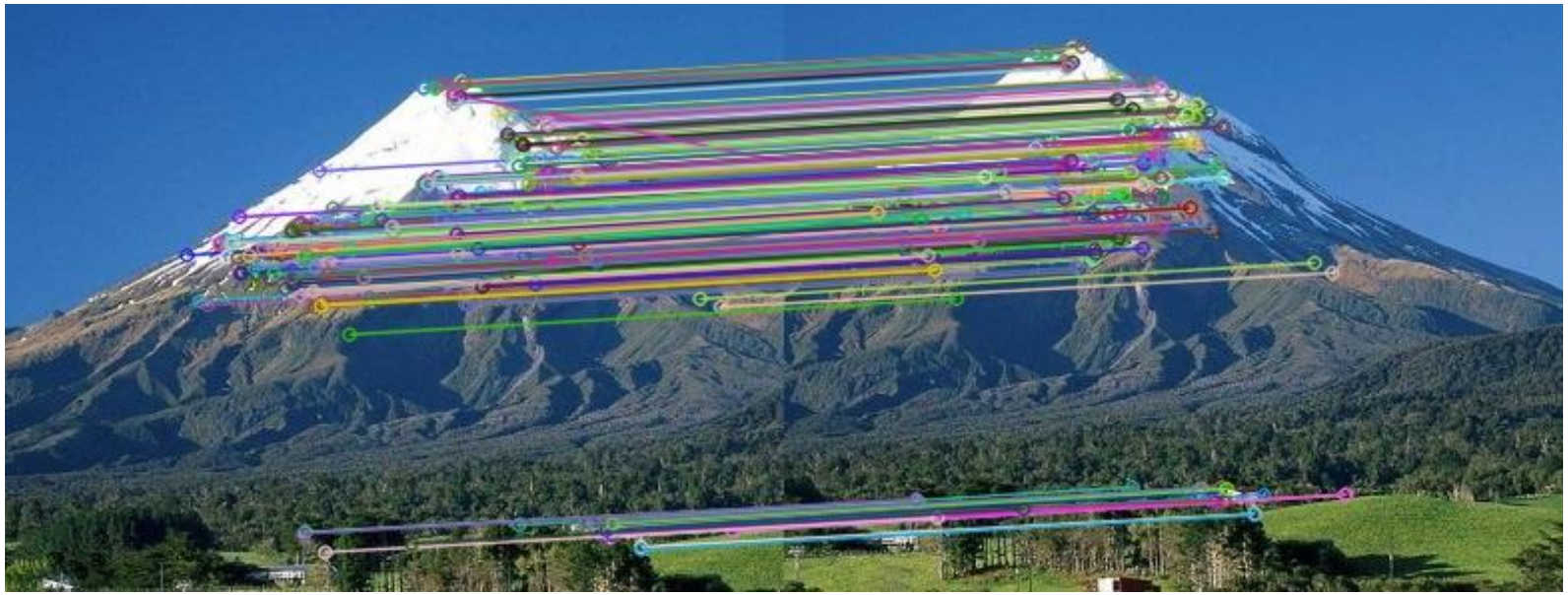
انطباق نقاط کلیدی

- پس از استخراج نقاط کلیدی از دو تصویر، نیاز است تا نقاط متناظر با یکدیگر مشخص شوند
- برای این منظور، ابتدا برای هر نقطه ویژگی یک توصیفگر محاسبه می‌شود
- سپس، دو به دوی توصیفگرها از دو تصویر مقایسه می‌شوند و مشابه‌ترین توصیفگرها به عنوان نقاط متناظر انتخاب می‌شوند
- برای جلوگیری از تناظریابی اشتباه، حد آستانه‌ای بر روی میزان مشابهت گذاشته می‌شود



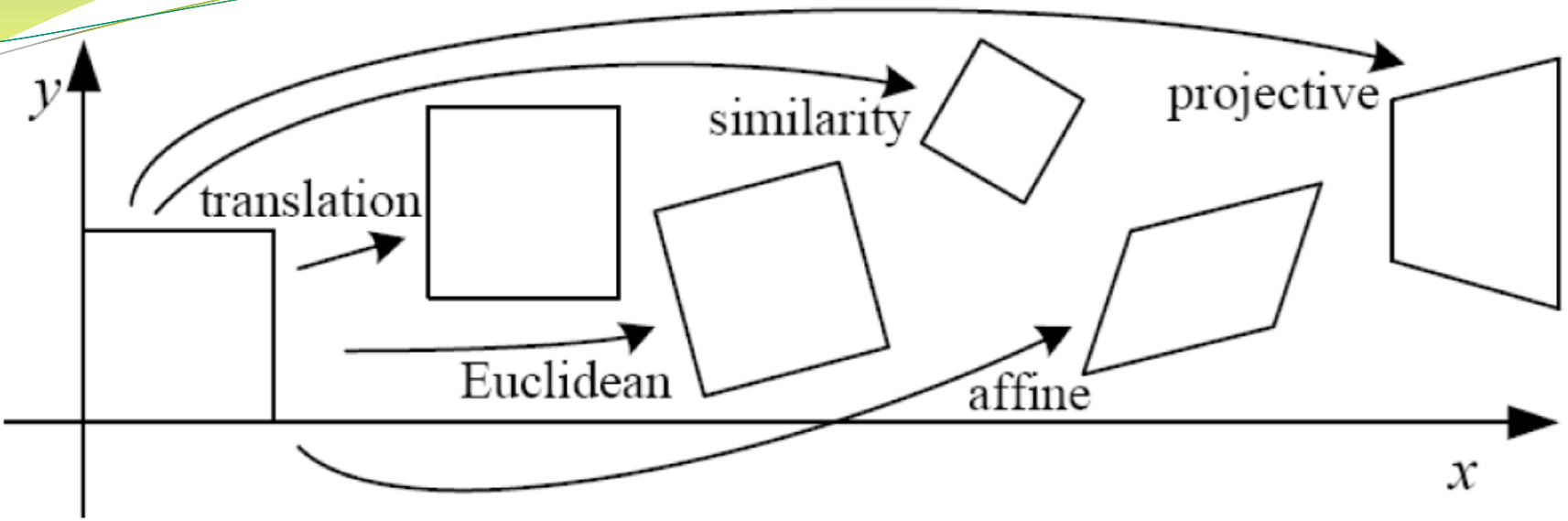
تابع تبدیل

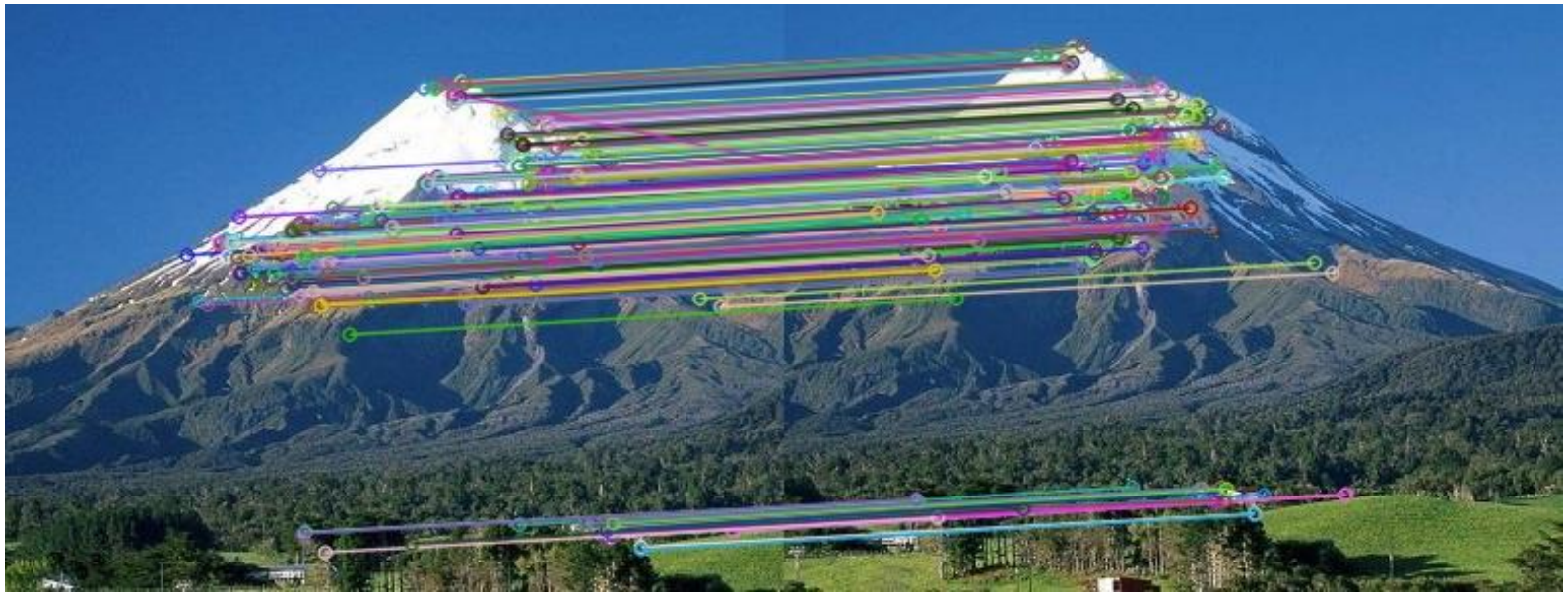
- پس از یافتن نقاط متناظر، باید تابع تبدیلی را بدست آورد که نقاط تصویر اول را به نقاط تصویر دوم نگاشت کنند
- برای این کار، ابتدا یک مدل برای تابع تبدیل انتخاب می‌شود و سپس پارامترهای آن بر اساس نقاط بدست آمده بهینه می‌شوند



$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = T\left(\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}\right)$$

تبدیل‌های هندسی



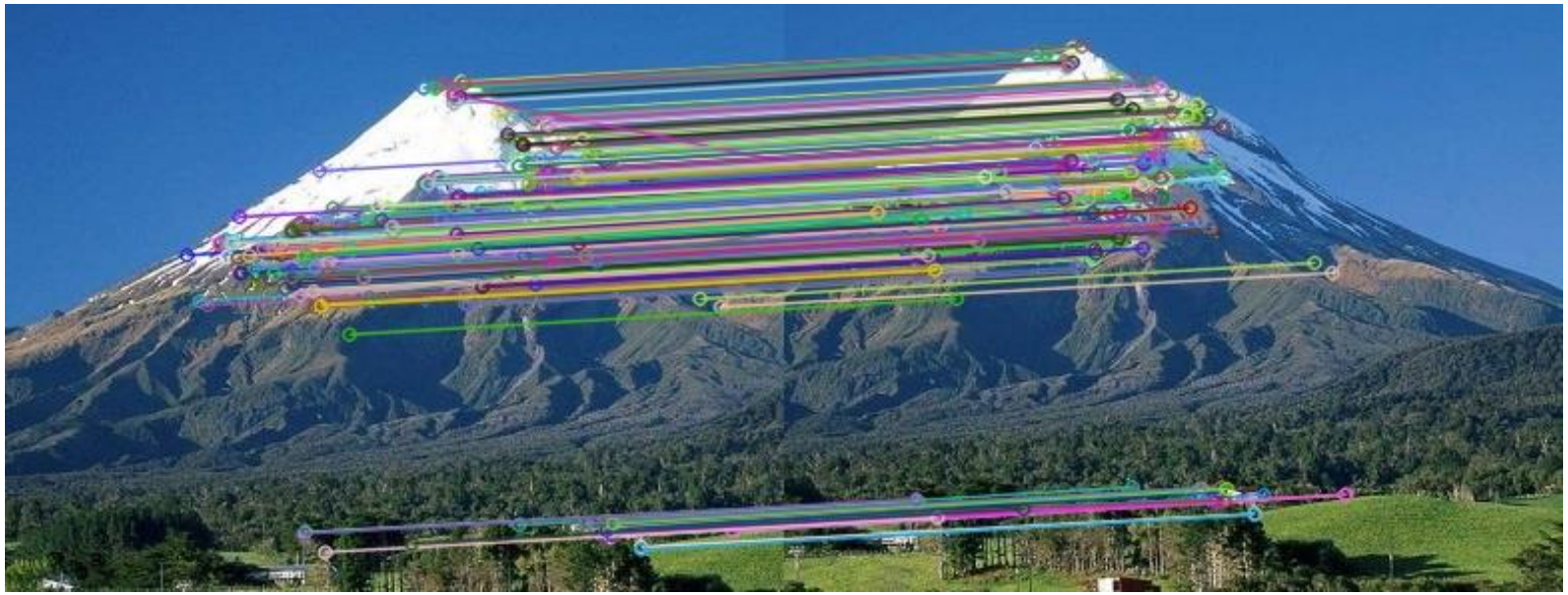


$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = T\left(\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}\right)$$

انتقال

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

- پارامترهای مدل (x_t, y_t) بر اساس نقاط متناظر محاسبه می‌شوند
- نیازمند تنها ۱ نقطه است!
- به دلیل وجود خطا در مکان‌یابی دقیق نقاط کلیدی، می‌توان با استفاده از تعداد بیشتری نقطه به پارامترهای دقیق‌تری دست یافت



$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = T\left(\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}\right)$$

حداقل مربعات خطا

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

- تابع هزینه

$$cost = \sum (x_2^n - x_1^n - t_x)^2 + \left(y_2^n - y_1^n - t_y\right)^2$$

- بهینه‌سازی
  - محاسبه مشتق

$$\frac{d}{dt_x} cost = -2 \sum (x_2^n - x_1^n - t_x) = 0$$

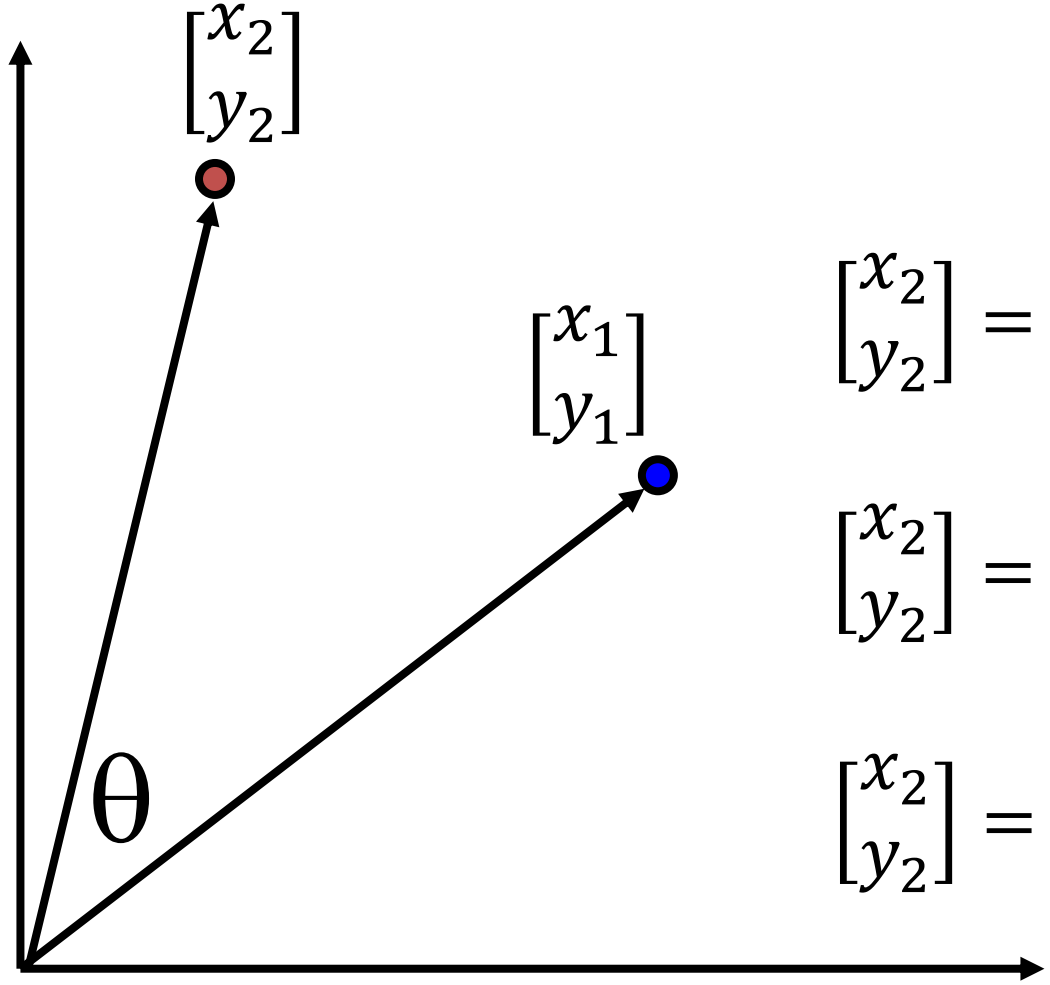
$$\Rightarrow t_x = \frac{1}{N} \sum (x_2^n - x_1^n) \qquad t_y = \frac{1}{N} \sum (y_2^n - y_1^n)$$

داده‌های پرت

- روش حداقل مربعات خطا حساس به داده‌های پرت است
- روش RANSAC برای بدست آوردن تابع تبدیل مقاوم نسبت به داده‌های پرت استفاده می‌شود

تبدیل Rigid

- این تبدیل تنها شامل چرخش و انتقال است



$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \cos\theta - y_1 \sin\theta \\ x_1 \sin\theta + y_1 \cos\theta \end{bmatrix}$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

تبدیل Rigid

- این تبدیل تنها شامل چرخش و انتقال است

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \underbrace{\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}}_{R} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \underbrace{\begin{bmatrix} t_x \\ t_y \end{bmatrix}}_{T}$$

- تبدیل Rigid تنها ۳ پارامتر دارد که توسط ۲ نقطه قابل محاسبه هستند
- البته باید خطای اندازه‌گیری و داده‌های پرت را لحاظ کرد

تبدیل شباهت

- این تبدیل شامل چرخش، انتقال و مقیاس است

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = a \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

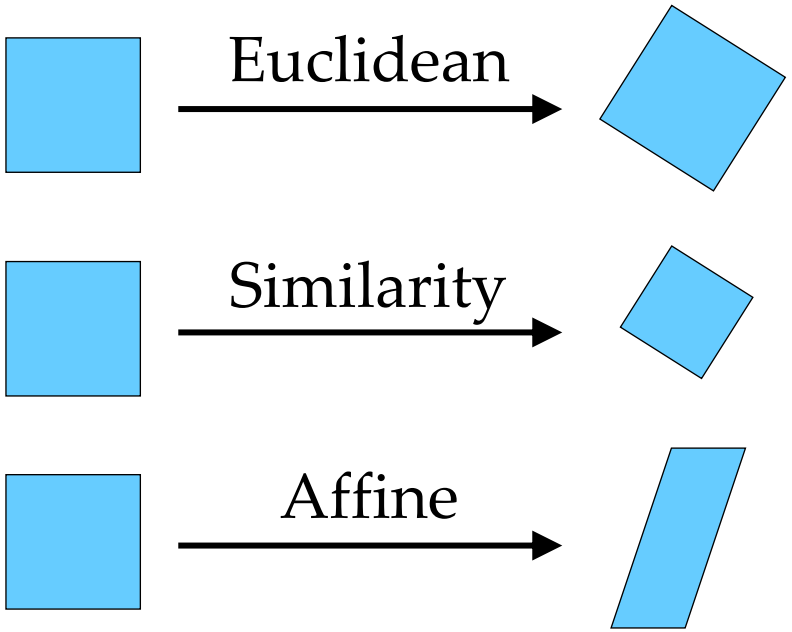
$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a\cos\theta & -a\sin\theta & t_x \\ a\sin\theta & a\cos\theta & t_y \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

- ۴ درجه آزادی و حداقل ۲ نقطه!

تبدیل Affine

- این تبدیل شامل چرخش، انتقال، مقیاس و کجی است

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

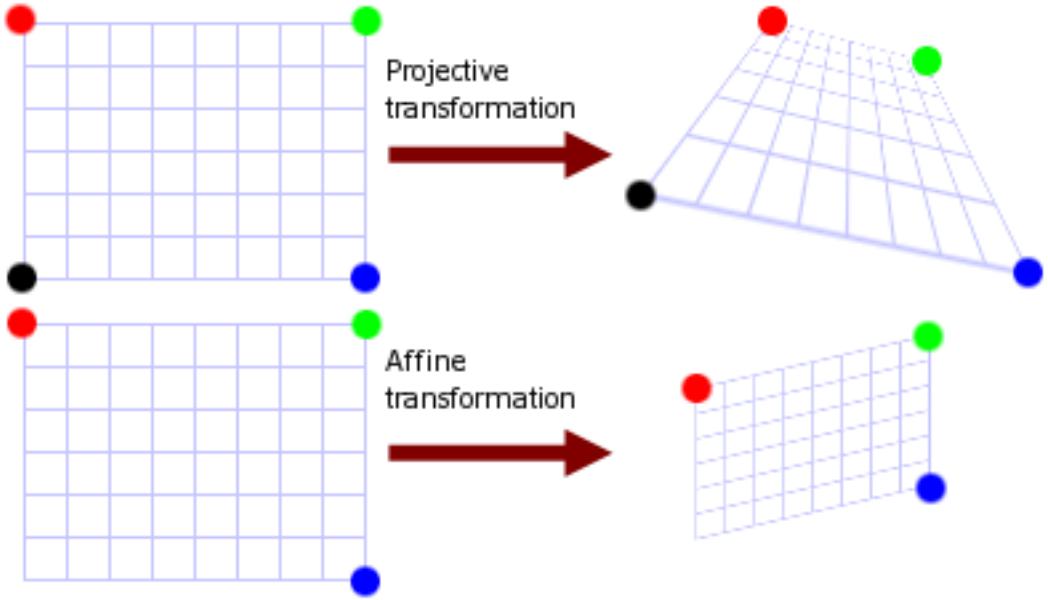


- ۶ درجه آزادی و حداقل ۳ نقطه!
- خط به خط نگاشت می‌شود
- خطوط موازی، موازی باقی می‌مانند
- نسبت‌ها روی یک خط حفظ می‌شود

تبدیل تصویری

- تبدیل‌های قبل نمی‌توانند تغییر عمق پیکسل‌ها را مدل کنند

$$s_2 \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$



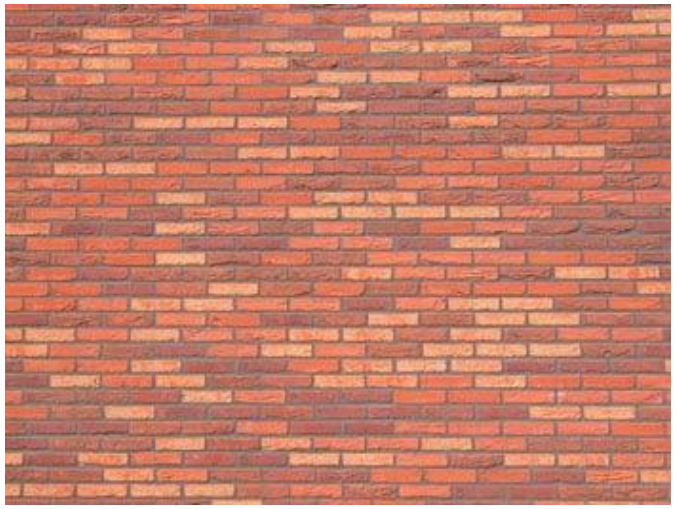
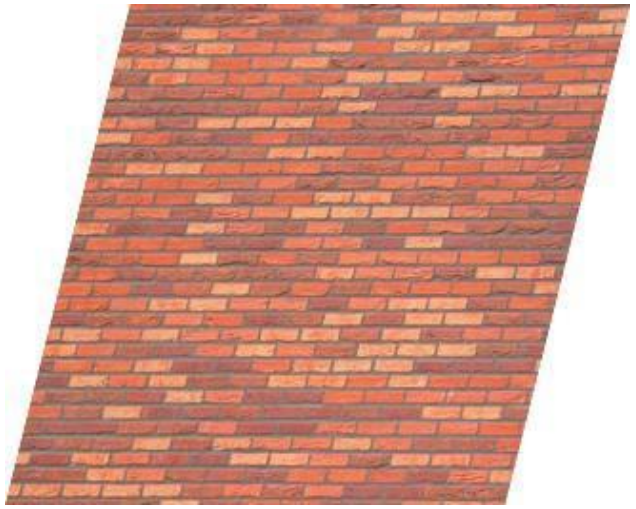
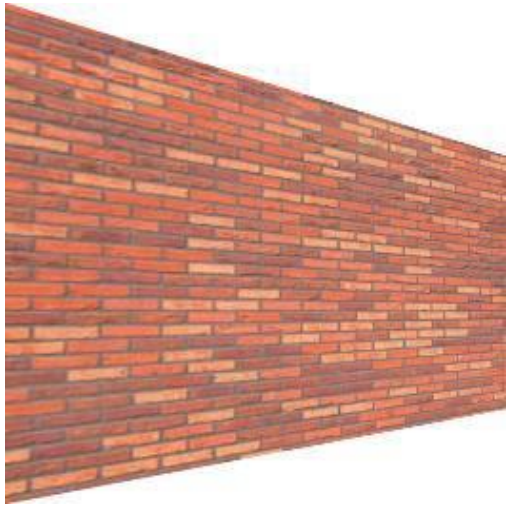
$$x_2 = \frac{h_{11}x_1 + h_{12}y_1 + h_{13}}{h_{31}x_1 + h_{32}y_1 + h_{33}}$$

$$y_2 = \frac{h_{21}x_1 + h_{22}y_1 + h_{23}}{h_{31}x_1 + h_{32}y_1 + h_{33}}$$

تبدیل تصویری

- تبدیل تصویری، تبدیل Affineای است که نسبت به موقعیت پیکسل در ضریب متفاوتی ضرب می‌شود
  - خط به خط نگاشت می‌شود
  - خطوط موازی لزوما موازی نمی‌مانند
  - نسبت‌ها لزوما حفظ نمی‌شود
  - ۸ درجه آزادی دارد و حداقل به ۴ نقطه نیاز دارد

$$s_2 \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

توابع OpenCV

```
mat = cv2.getPerspectiveTransform(src_points, dst_points[, solveMethod])

// src_points:    Coordinates of quadrangle vertices in the source image
// dst_points:    Coordinates of the corresponding quadrangle vertices in the destination image
// mat:           Perspective transform from four pairs of the corresponding points
```

```
dst_points = cv2.perspectiveTransform(src_points, mat)

// src_points:      Input two-channel or three-channel floating-point array; each element is a 2D/3D vector to be transformed
// mat:             3x3 or 4x4 floating-point transformation matrix
// dst_points:      Output array of the same size and type as src
```

$$\text{dst}(x,y) = \text{src}\left(\frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}\right)$$

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

// src_points:      Input image
// mat:             3x3 floating-point transformation matrix
// dsize:           Size of the output image
// flags:           Combination of interpolation methods and the optional flag WARP_INVERSE_MAP
// borderMode:      Pixel extrapolation method
// borderValue:     value used in case of a constant border; by default, it equals 0
// dst:             Output image that has the size dsize and the same type as src .
```

توابع OpenCV

```
mat, mask = cv2.findHomography(src_points, dst_points[, method[, ransacReprojThreshold[, maxIters[, confidence]]]])

// src_points:              Coordinates of the points in the original plane
// dst_points:              Coordinates of the points in the target plane
// method:                  Method used to compute a homography matrix (least squares, RANSAC, LMEDS, RHO)
// ransacReprojThreshold:   Maximum allowed reprojection error to treat a point pair as an inlier
// maxIters:                The maximum number of RANSAC iterations
// confidence:              Confidence level, between 0 and 1
// mask:                    Optional output mask set by a robust method (RANSAC or LMEDS)
// mat:                     Estimated perspective transform  between two planes
```

Function	Use
cv::transform()	Affine transform a list of points
cv::warpAffine()	Affine transform a whole image
cv::getAffineTransform()	Calculate affine matrix from points
cv::getRotationMatrix2D()	Calculate affine matrix to achieve rotation
cv::perspectiveTransform()	Perspective transform a list of points
cv::warpPerspective()	Perspective transform a whole image
cv::getPerspectiveTransform()	Fill in perspective transform matrix parameters