مبانی بینایی کامپیوتر

مدرس: محمدرضا محمدی

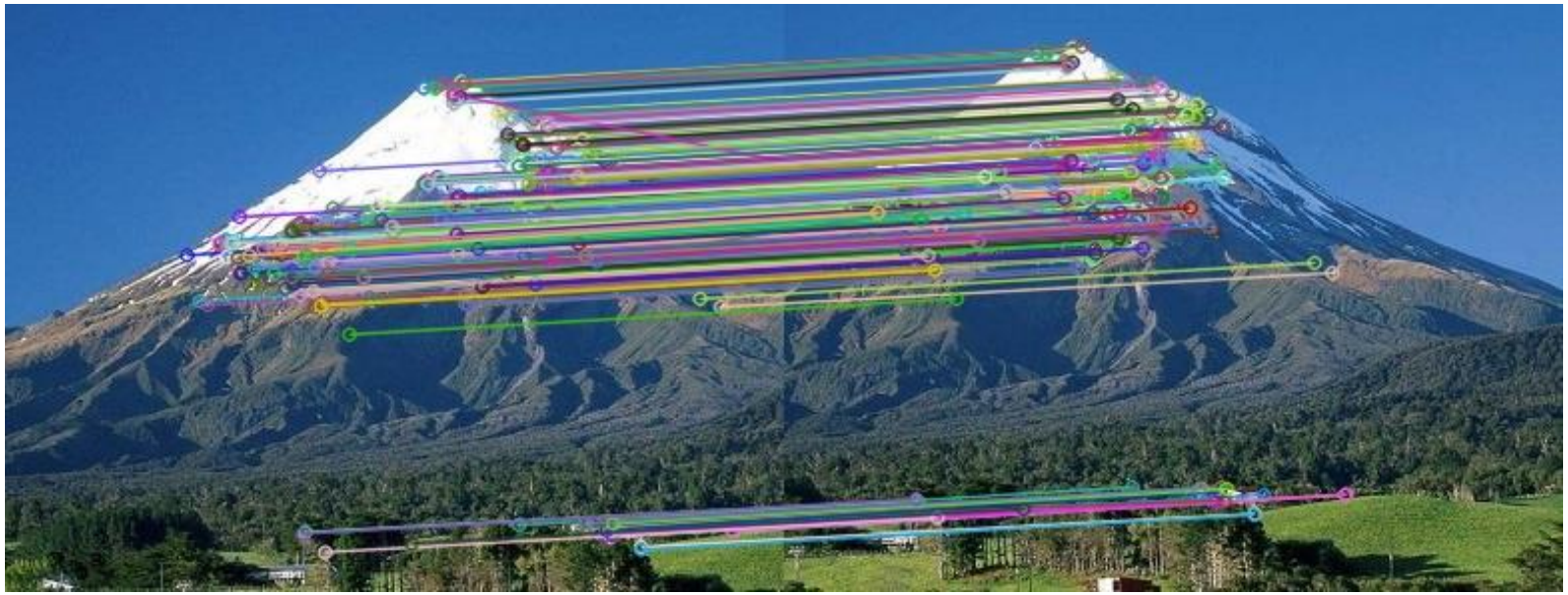
بهار ۱۴۰۲

تناظر و هم‌ترازی تصاویر

Correspondence and Image Alignment

تابع تبدیل

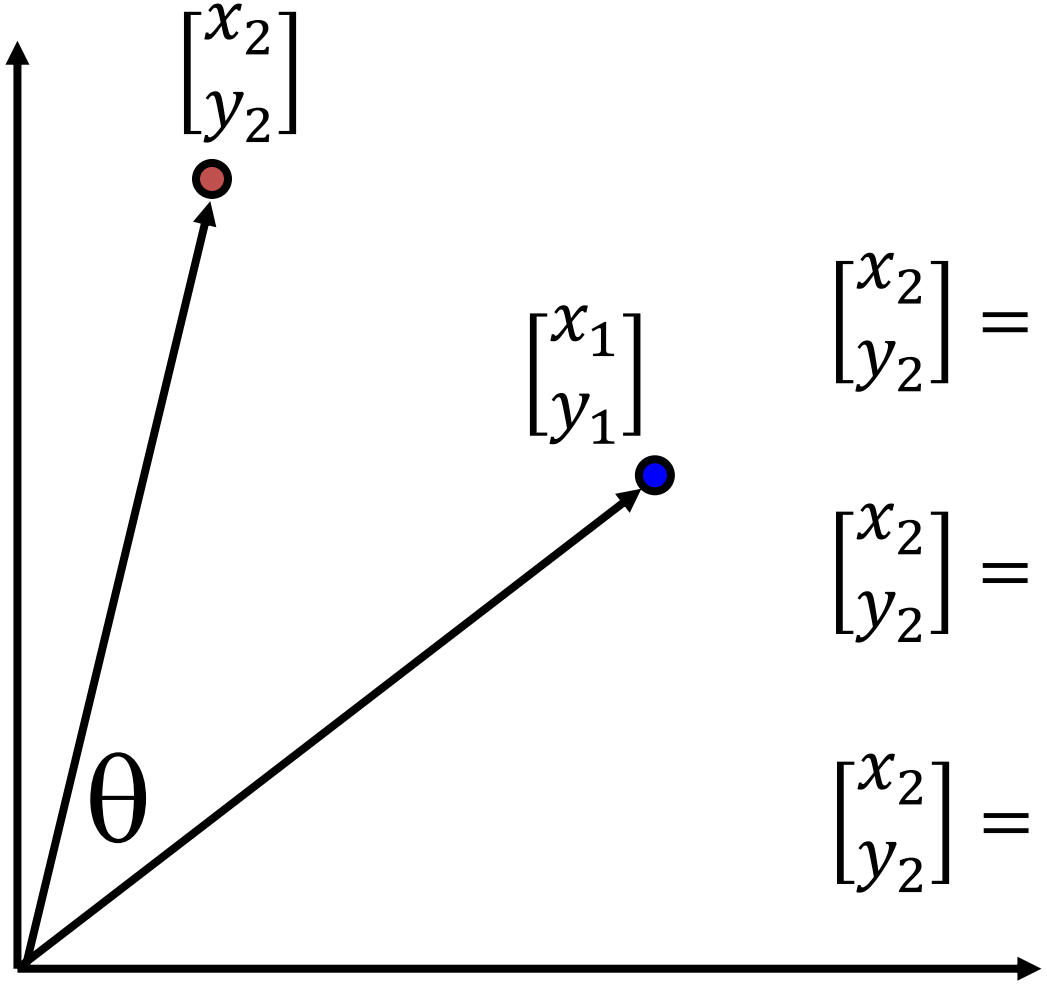
- پس از یافتن نقاط متناظر، باید تابع تبدیلی را بدست آورد که نقاط تصویر اول را به نقاط تصویر دوم نگاشت کنند
- برای این کار، ابتدا یک مدل برای تابع تبدیل انتخاب می‌شود و سپس پارامترهای آن بر اساس نقاط بدست آمده بهینه می‌شوند



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

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

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

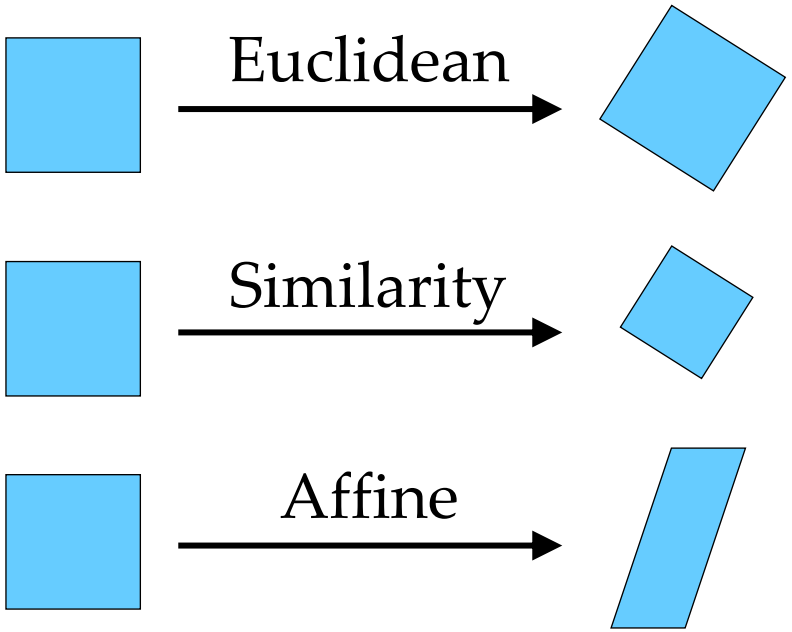
$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a\cos\theta & -a\sin\theta & t_x \\ a\sin\theta & a\cos\theta & t_y \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

- ۴ درجه آزادی و حداقل ۲ نقطه!

تبدیل Affine

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

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

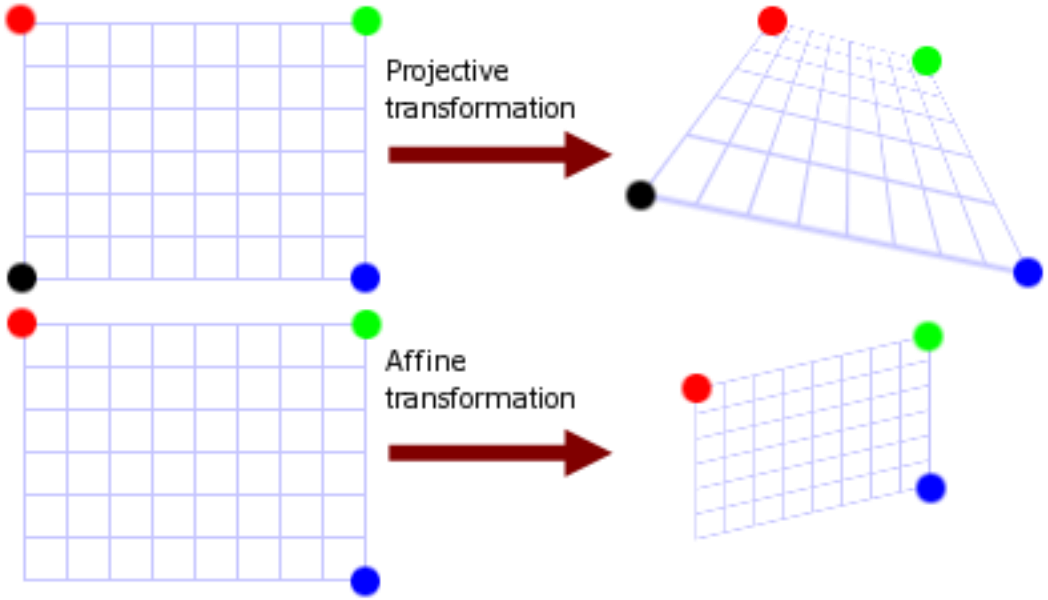


- ۶ درجه آزادی و حداقل ۳ نقطه!
- خط به خط نگاشت می‌شود
- خطوط موازی، موازی باقی می‌مانند
- نسبت‌ها روی یک خط حفظ می‌شود

تبدیل تصویری

- تبدیل‌های قبل نمی‌توانند تغییر عمق پیکسل‌ها را مدل کنند

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



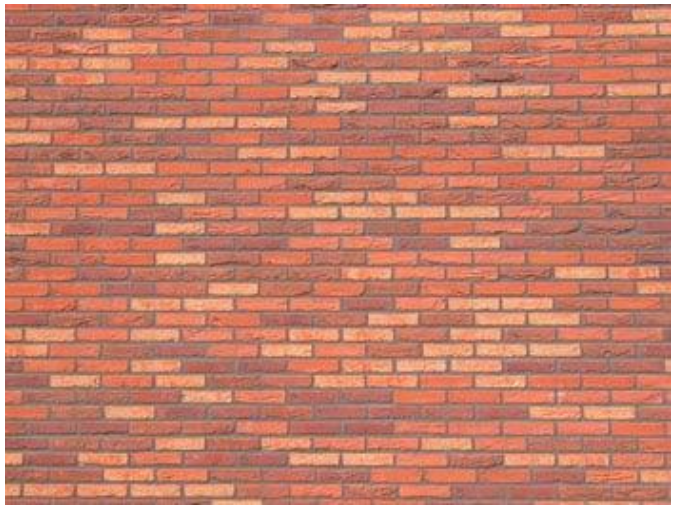
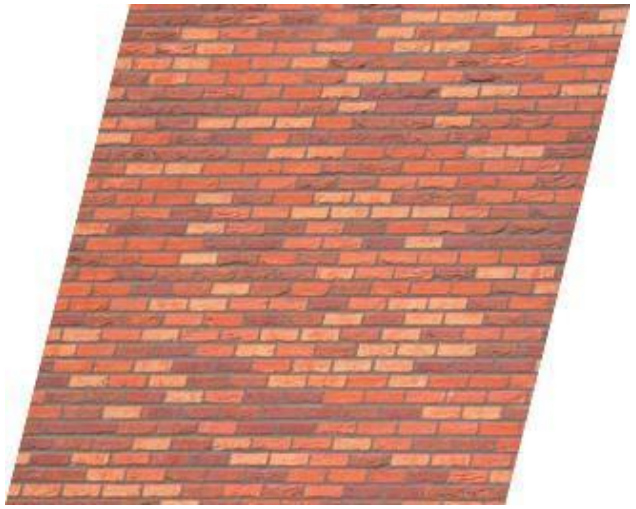
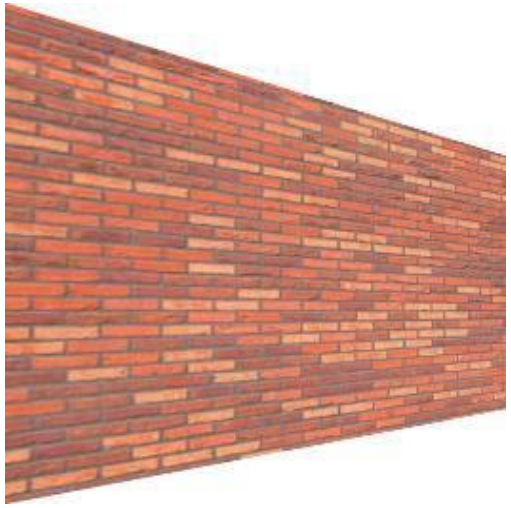
$$x_2 = \frac{h_{11}x_1 + h_{12}y_1 + h_{13}}{h_{31}x_1 + h_{32}y_1 + h_{33}}$$

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

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
```

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

```
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 .
```

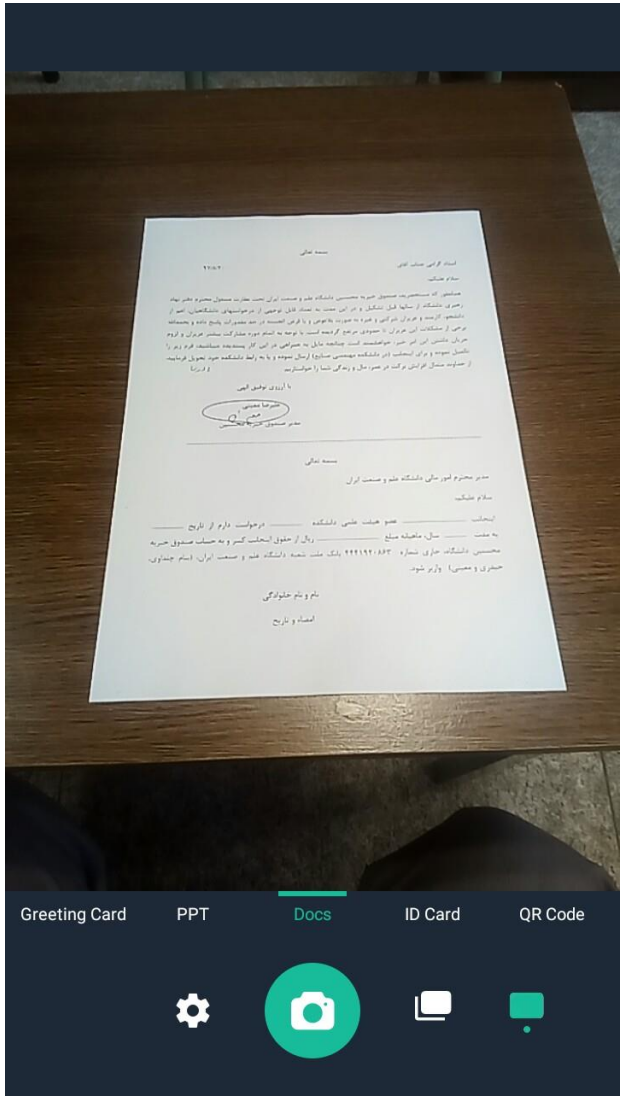
توابع OpenCV

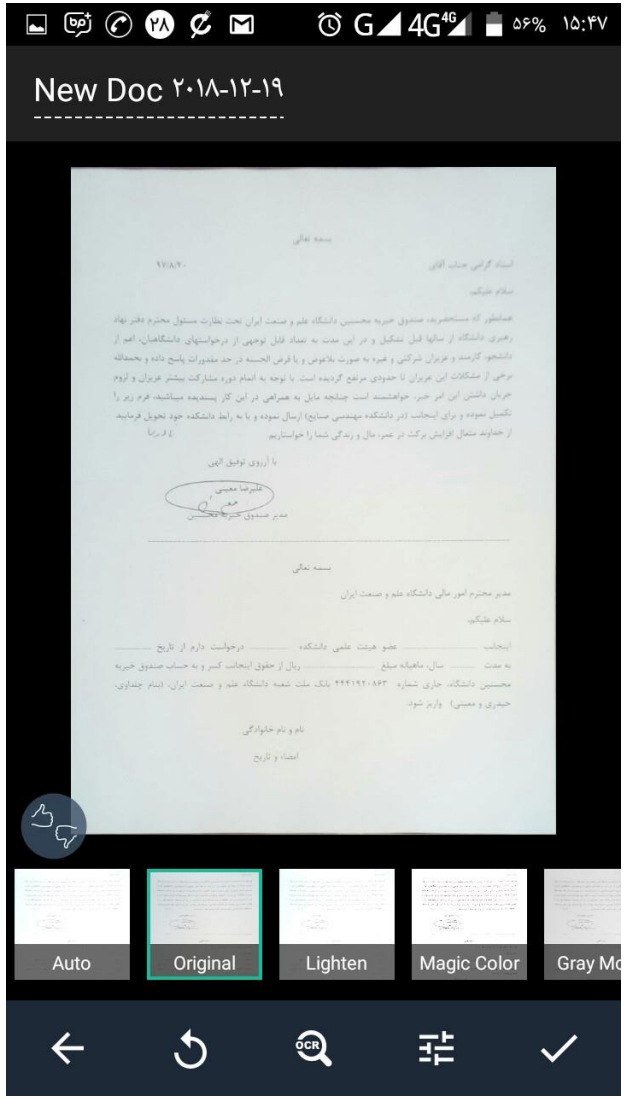
```
mat, mask = cv2.findHomography(src_points, dst_points[, method[, ransacReprojThreshold[, maxIters[, confidence]]]])

// 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

CamScanner


Greeting Card
PPT
Docs
ID Card
QR Code


New Doc ۲۰۱۸-۱۲-۱۹
Auto
Original
Lighten
Magic Color
Gray Mo

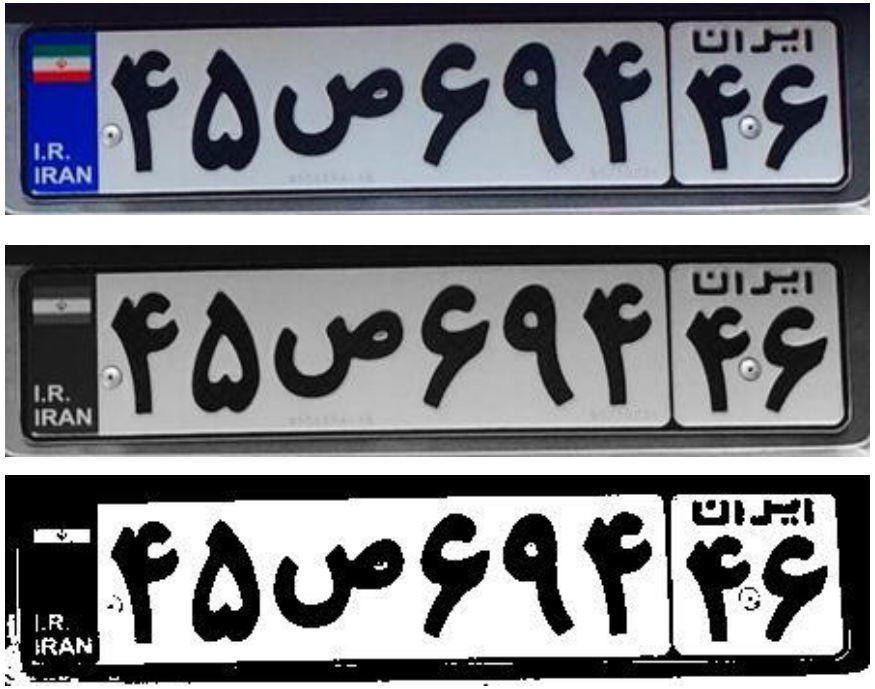
ناحیه‌بندی تصویر

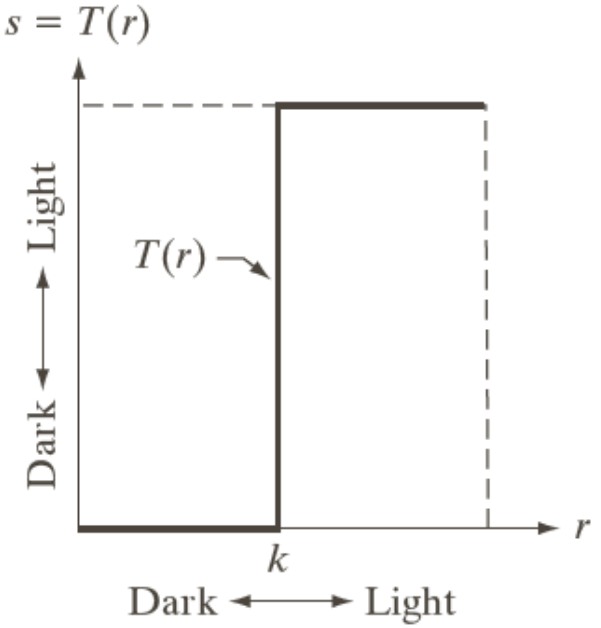
Image Segmentation

ناحیه‌بندی تصویر

آستانه‌گذاری سطح خاکستری

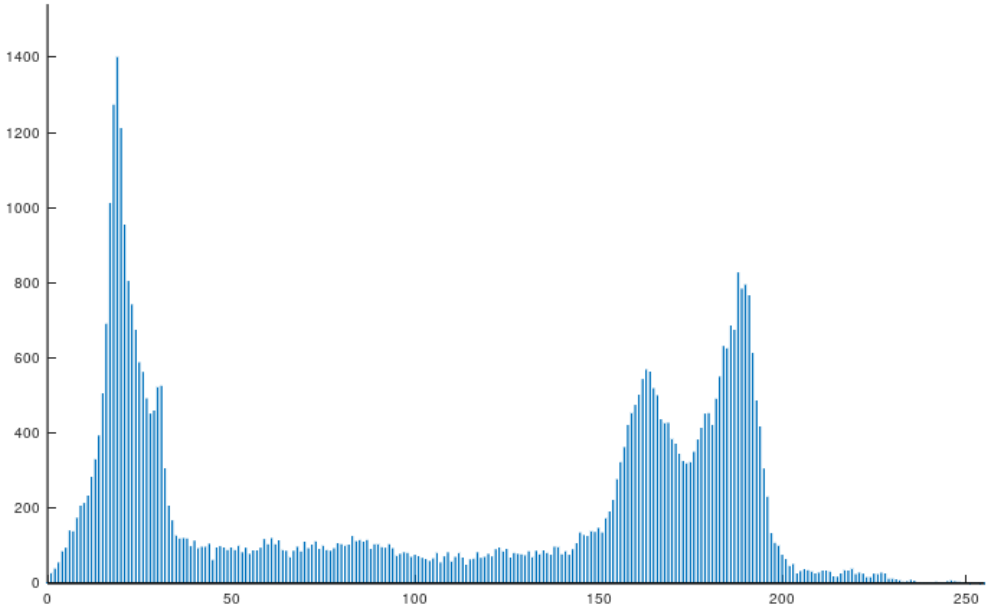
- ساده‌ترین راه برای استخراج ناحیه از تصویر استفاده از مقادیر سطح خاکستری است
- پس از این عملگر نقطه‌ای، هر ناحیه به هم پیوسته یک ناحیه است

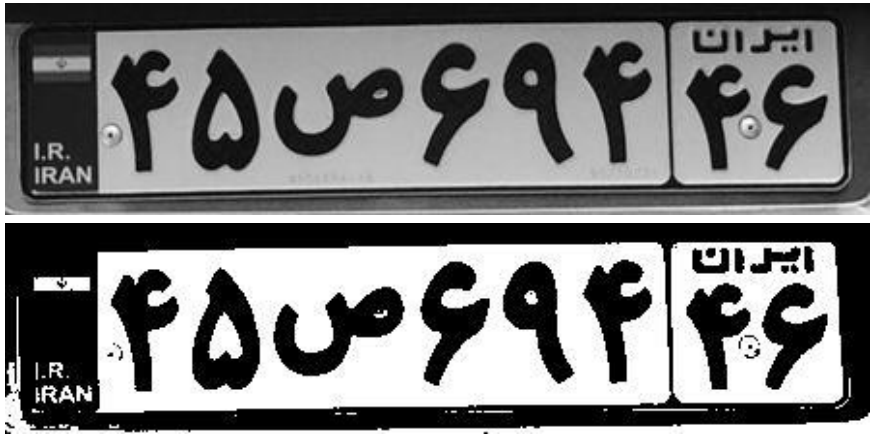




تعیین سطح آستانه

- سطح آستانه بهینه چه عددی است؟
- می‌توان با استفاده از دانش پیشین از یک عدد ثابت استفاده کرد
- می‌توان از مشخصه‌های آماری مانند میانگین یا میانه سطوح خاکستری استفاده کرد
- می‌توان از استفاده از هیستوگرام استفاده کرد





الگوریتم Otsu

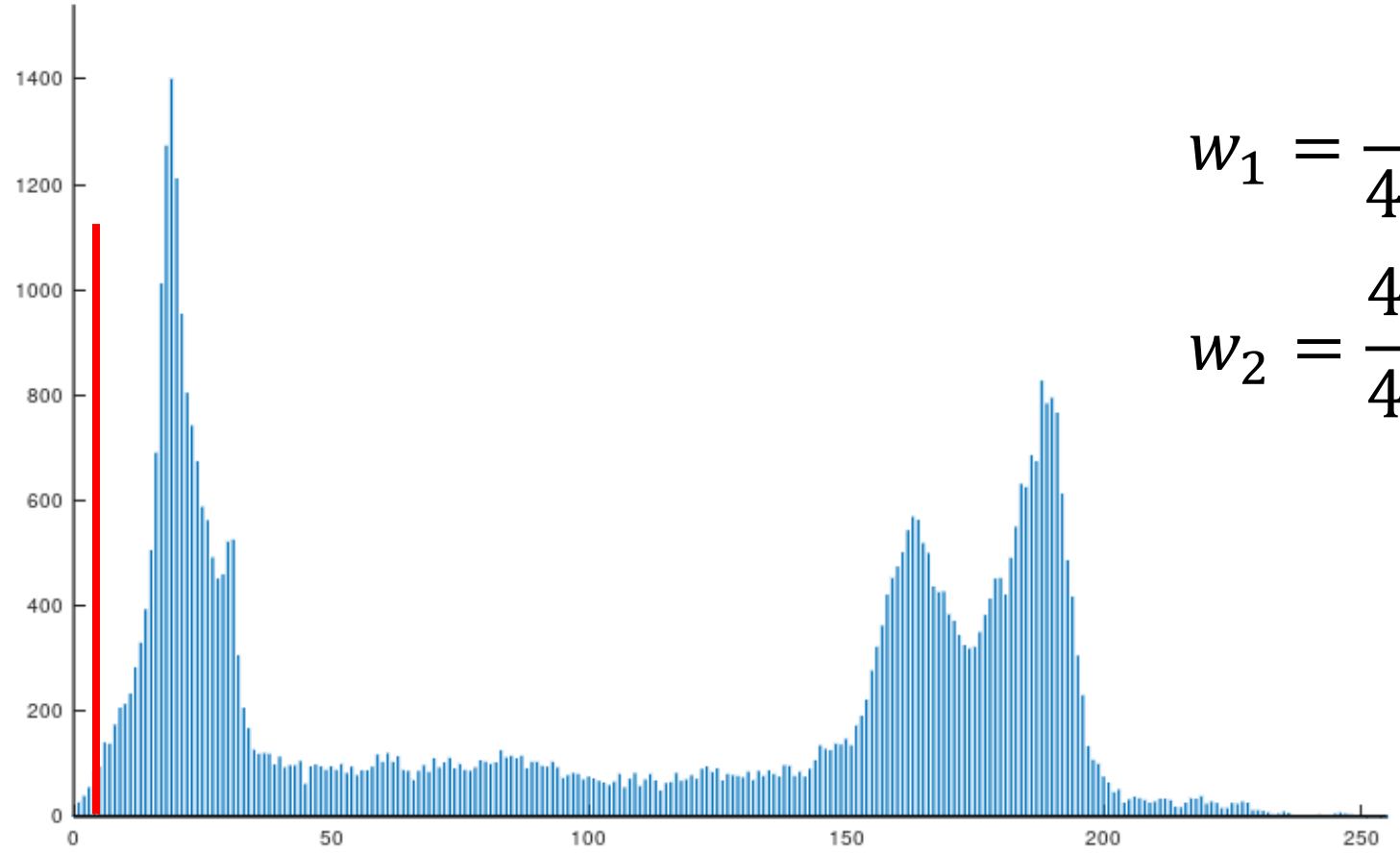
- یک الگوریتم تعیین سطح مقدار آستانه بر حسب مشخصه‌های آماری است
- خلاصه الگوریتم این است که سطح آستانه‌ای را انتخاب کنیم که واریانس بین پیکسل‌های هر کلاس کمینه شود

$$\sigma_w^2 = w_1\sigma_1^2 + w_2\sigma_2^2$$

- w_i تعداد پیکسل‌های کلاس i، و σ_i^2 واریانس پیکسل‌های آن کلاس است

الگوریتم Otsu

- برای یک تصویر ۸ بیتی سطح آستانه یکی از ۲۵۵ مقدار است



$$w_1 = \frac{302}{48832} = 0.006$$

$$w_2 = \frac{48530}{48832} = 0.994$$

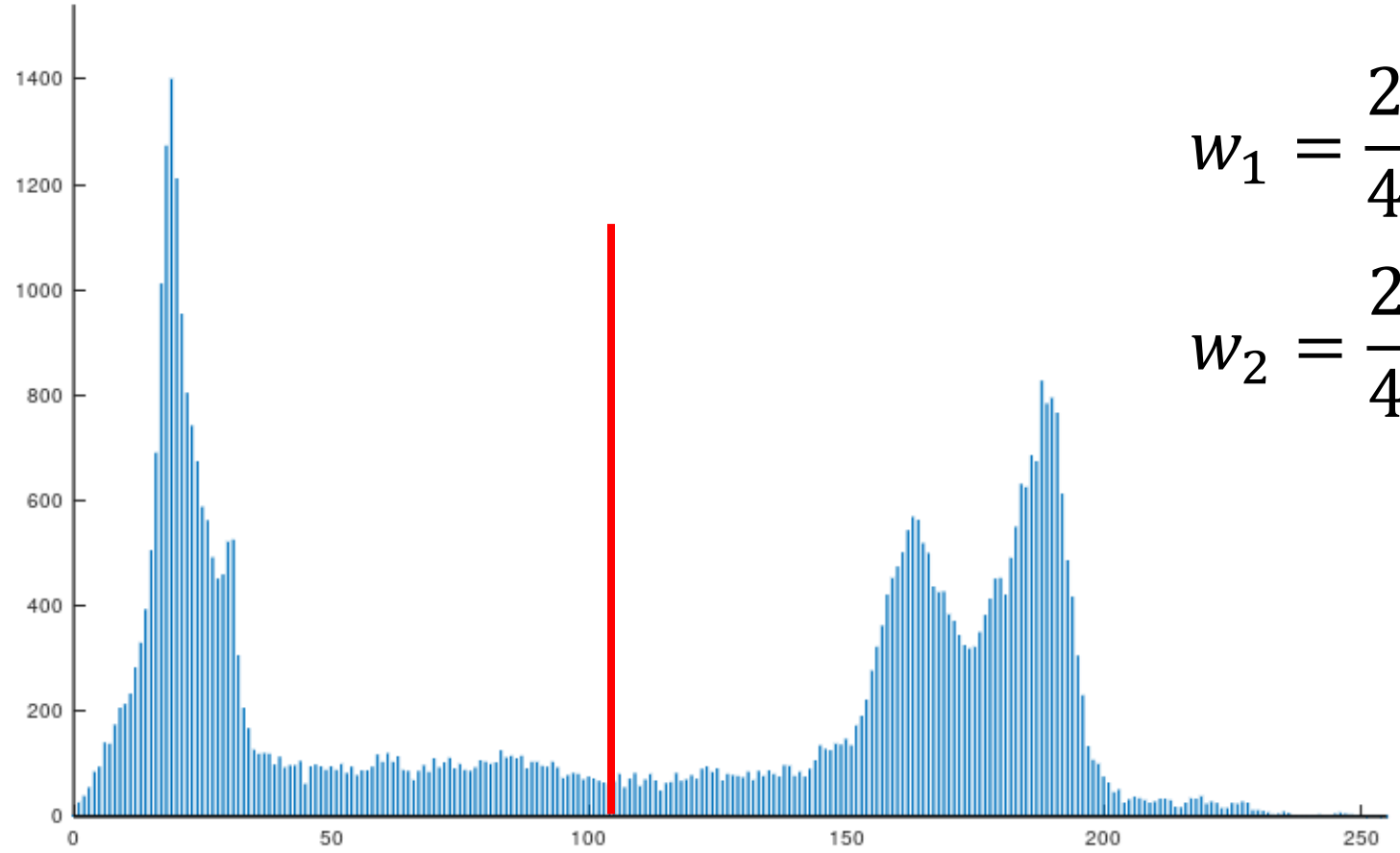
$$\sigma_1^2 = 1.75$$

$$\sigma_2^2 = 5154.8$$

$$\sigma_w^2 = 5123.0$$

الگوریتم Otsu

- برای یک تصویر ۸ بیتی سطح آستانه یکی از ۲۵۵ مقدار است



$$w_1 = \frac{23033}{48832} = 0.472$$

$$w_2 = \frac{25799}{48832} = 0.528$$

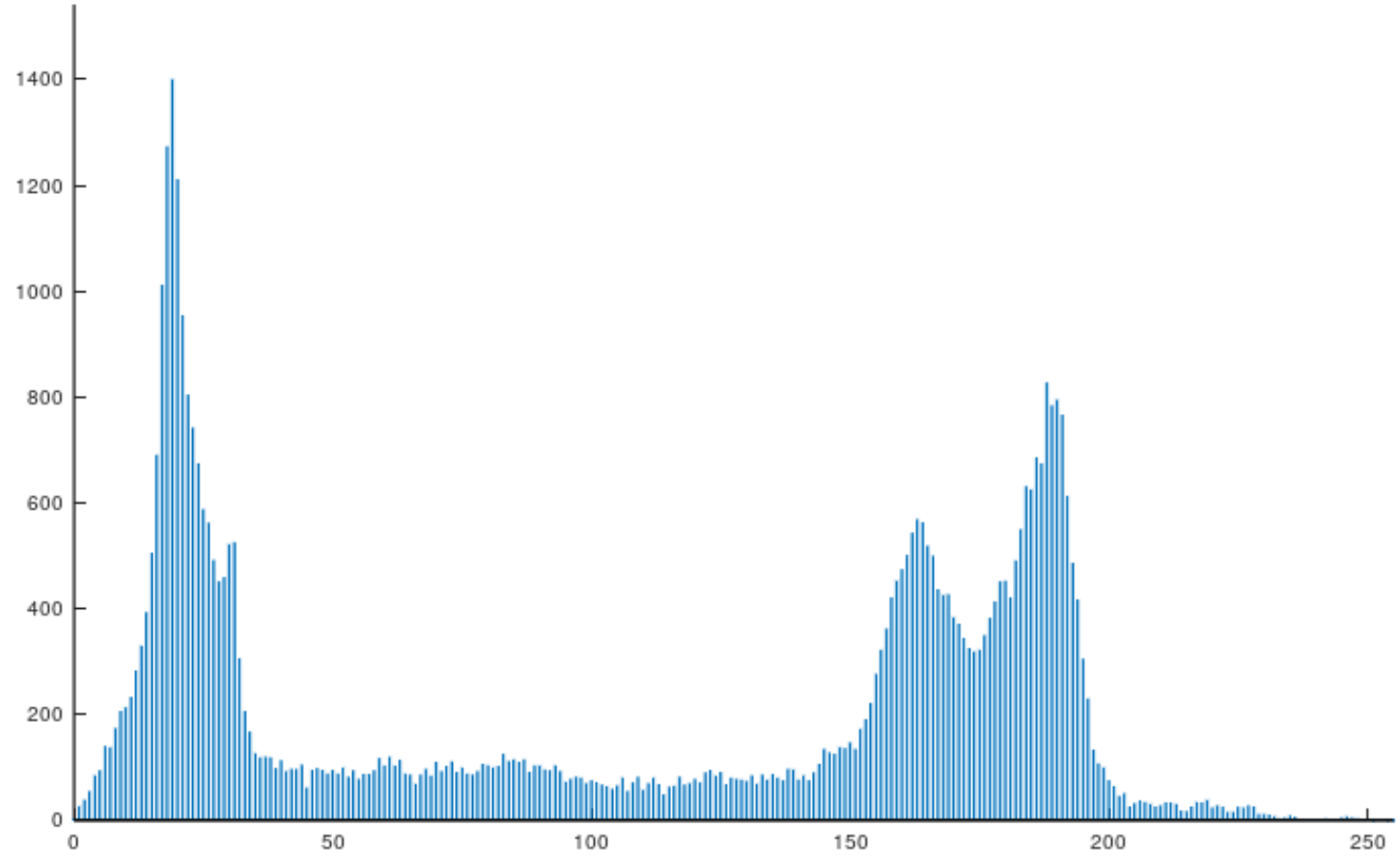
$$\sigma_1^2 = 703.2$$

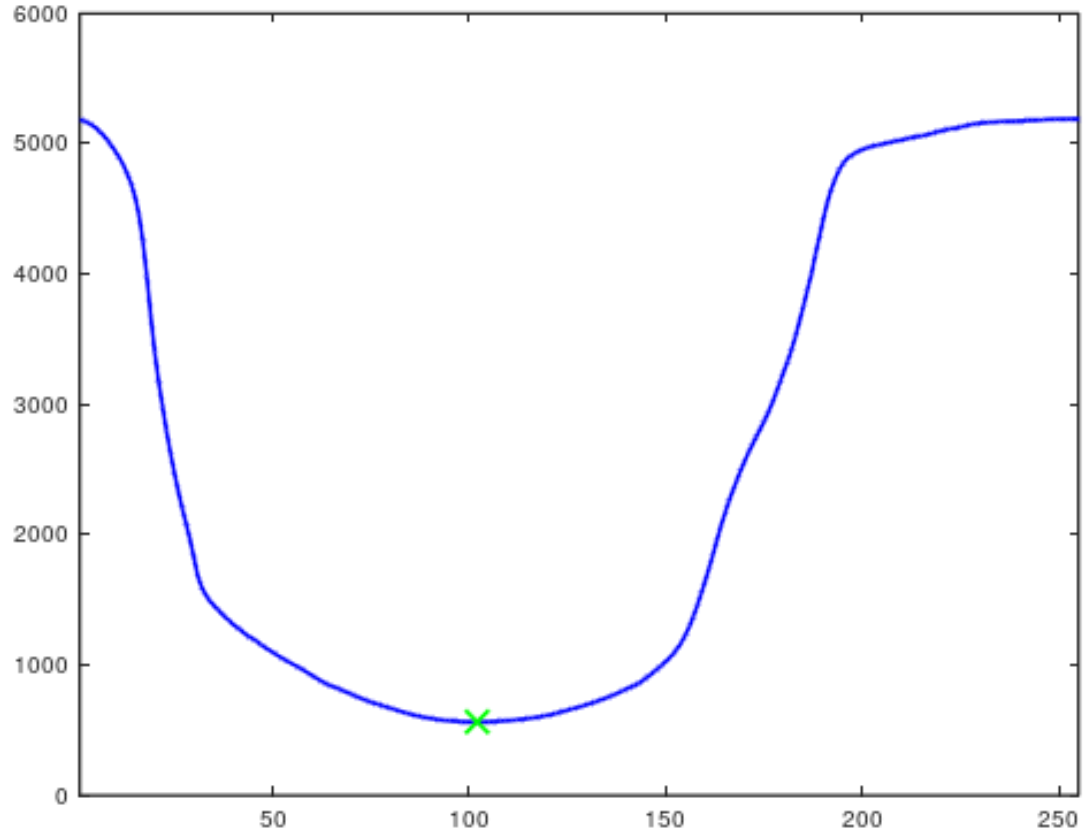
$$\sigma_2^2 = 456.9$$

$$\sigma_w^2 = 573.1$$

الگوریتم Otsu

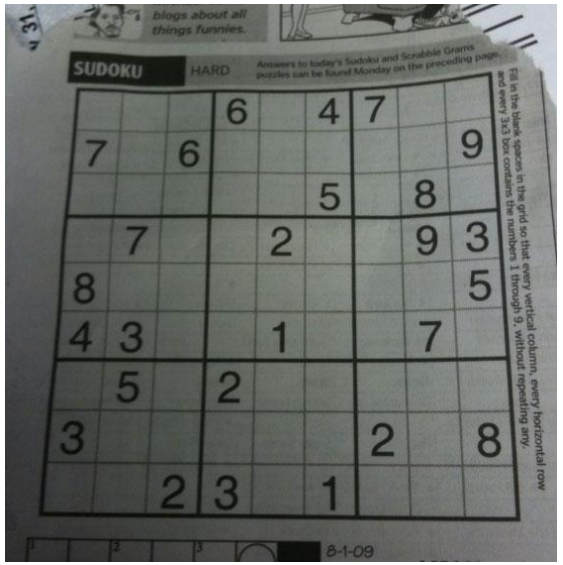
- برای یک تصویر ۸ بیتی سطح آستانه یکی از ۲۵۵ مقدار است





الگوریتم Otsu

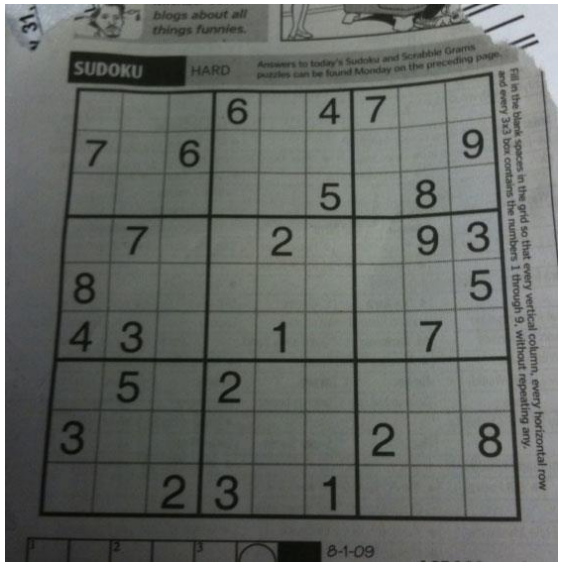
1		8			6	9	2	
	2		4	9		1		
	6						4	5
		3		7				
	9					2		3
					5			9
9							8	
	5		1				6	4
		1		5				

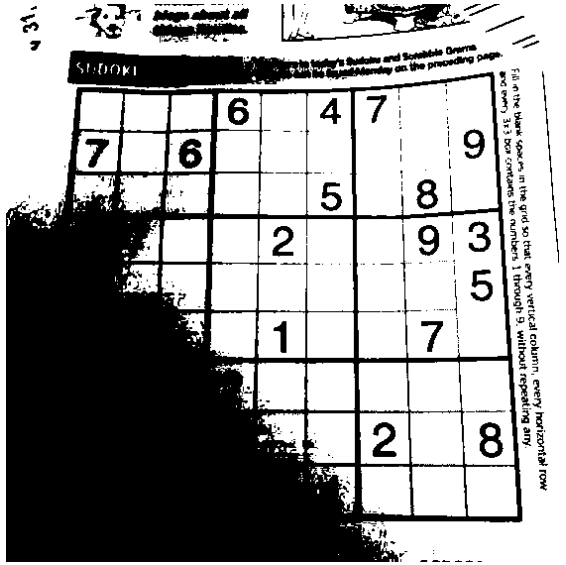


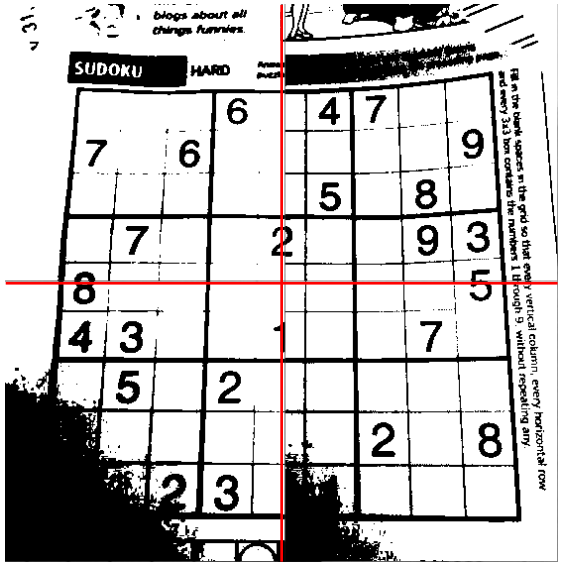
1		8			6	9	2	
	2		4	9		1		
	6						4	5
		3		7				
	9					2		3
					5			9
9							8	
	5		1				6	4
		1		5				

آستانه‌گذاری وفقی

- به منظور رفع چالش قبل، مناسب است تا برای هر ناحیه از تصویر یک آستانه متناسب تعریف شود
- در حالت حدی می‌توان برای هر پیکسل یک آستانه تعریف کرد
- البته این محاسبات پیچیده برای هر پیکسل هزینه‌بر است
- می‌توان میانگین پیکسل‌های اطراف هر ناحیه را به عنوان معیاری برای مقدار آستانه محاسبه کرد







آستانه‌گذاری وفقی

```
dst = cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C)

// src:                 Source 8-bit single-channel image
// maxValue:            Non-zero value assigned to the pixels for which the condition is satisfied
// adaptiveMethod:      Adaptive thresholding algorithm to use (MEAN or GAUSSIAN)
// thresholdType:       Thresholding type that must be either THRESH_BINARY or THRESH_BINARY_INV
// blockSize:           Size of a pixel neighborhood that is used to calculate a threshold value
// C:                   Constant subtracted from the mean or weighted mean
// dst:                 Destination image of the same size and the same type as src
```