计算机视觉
Computer Vision

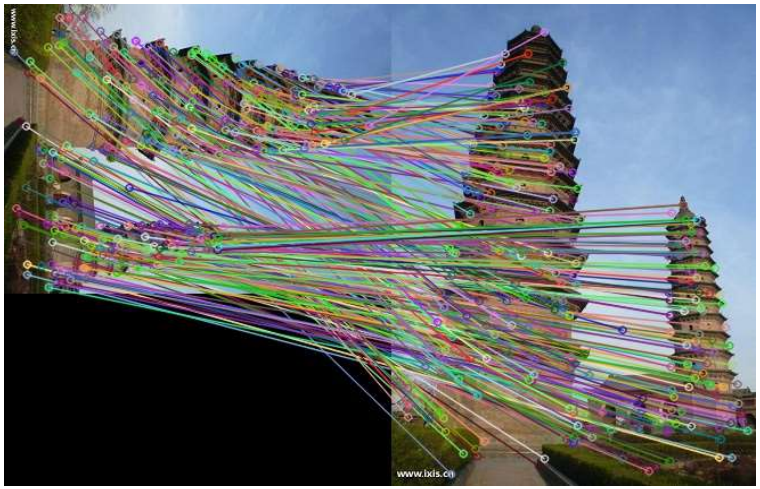
-- Matching 3

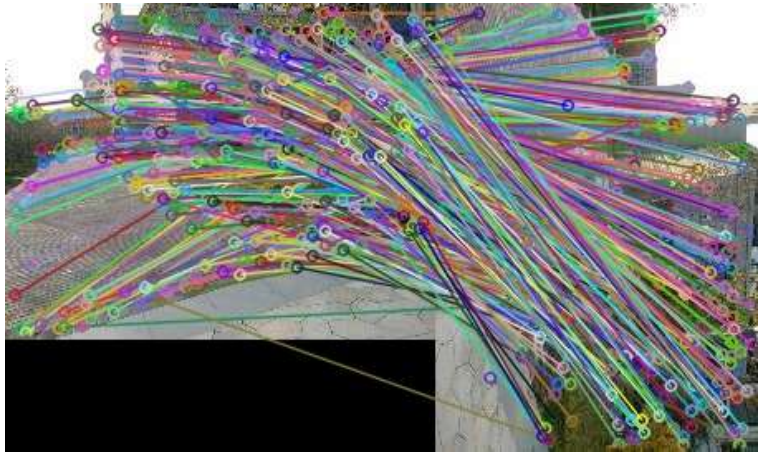
钟 凡
zhongfan@sdu.edu.cn

图像匹配

特征检测 → 特征匹配 → 运动估计

特征检测+匹配

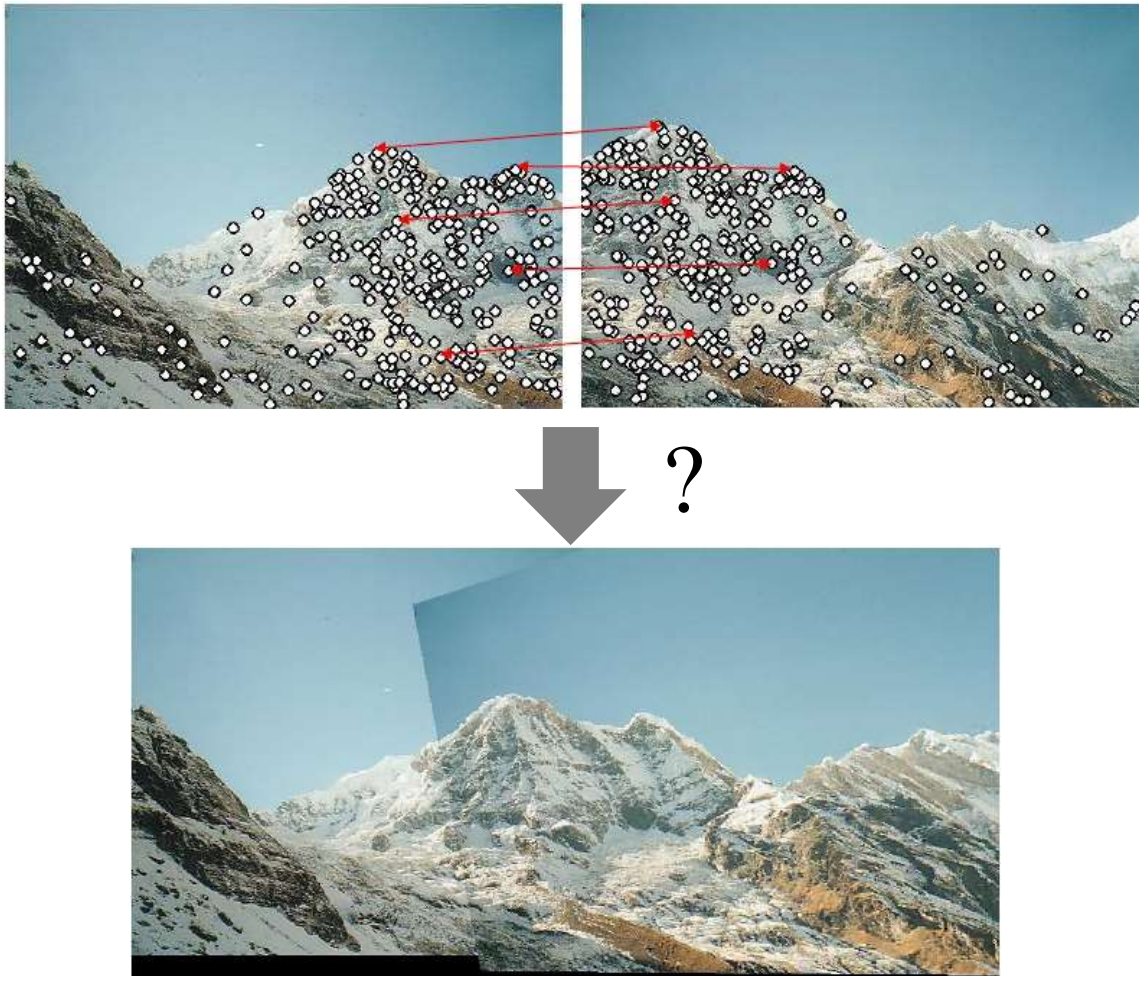




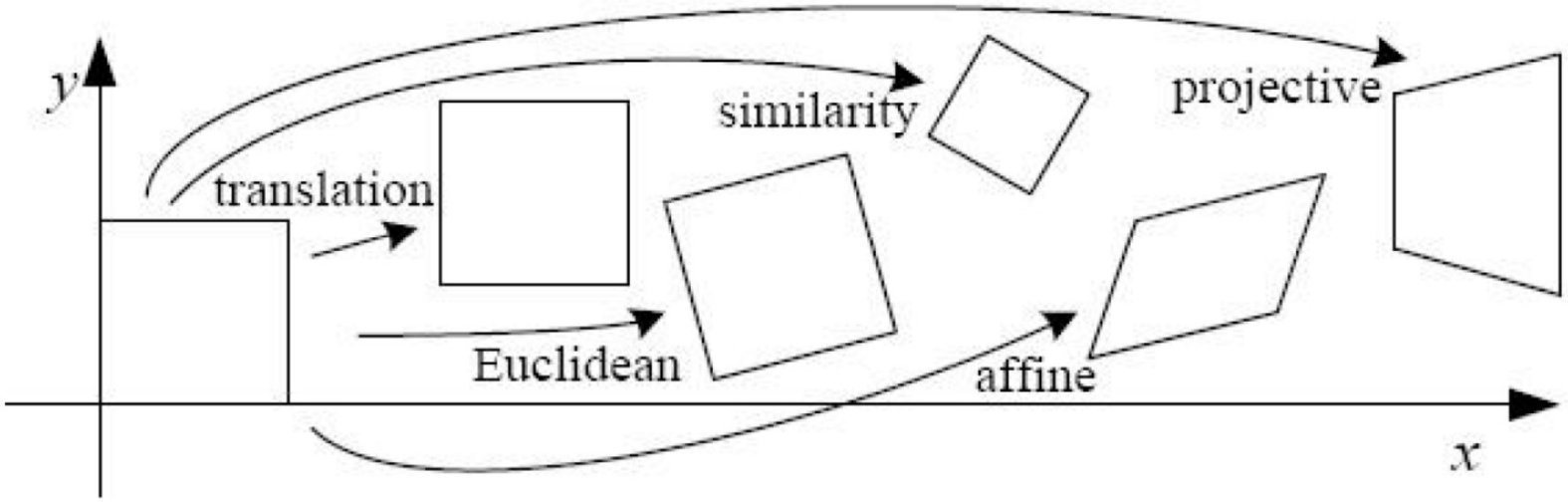
运动估计（图像配准/Alignment）

忽略外点(Outlier)

- 假设特征点匹配都基本正确



参数化运动模型



Homogeneous Coordinates & Transform Matrix

homogeneous coordinates:

(齐次坐标)

$$[x, y] = [x, y, 1] = [\omega x, \omega y, \omega]$$

2D Affine Transform:

（2维仿射变换）

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = A_{2\times 3} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Leftrightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D Perspective Transform:

（2维透视变换）

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = A_{3\times 3} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Leftrightarrow \begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

仿射变换

- **Linear Transform:**

$$f(a+b) = f(a) + f(b)$$
$$f(ka) = kf(a)$$

- **Affine = Linear + Translation** $f(a)+t$

e.g 2D仿射变换： $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{pmatrix} u \\ v \end{pmatrix}$

把平行直线映射为平行直线

Transform Matrix

Translation (平移):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scale (缩放):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation (旋转):

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

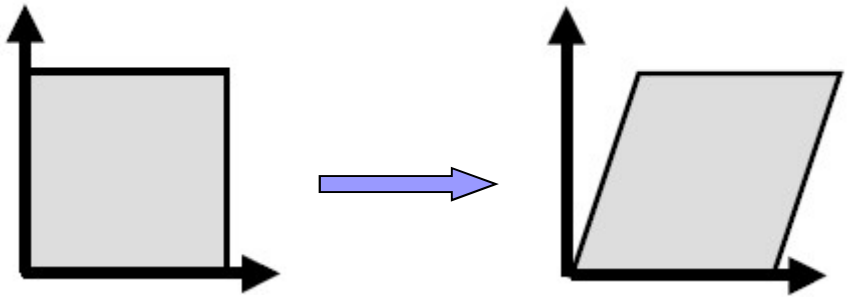
Transform Matrix

X切变（Shear):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x + sy \\ y \end{bmatrix} = \begin{pmatrix} 1 & s & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Y切变:

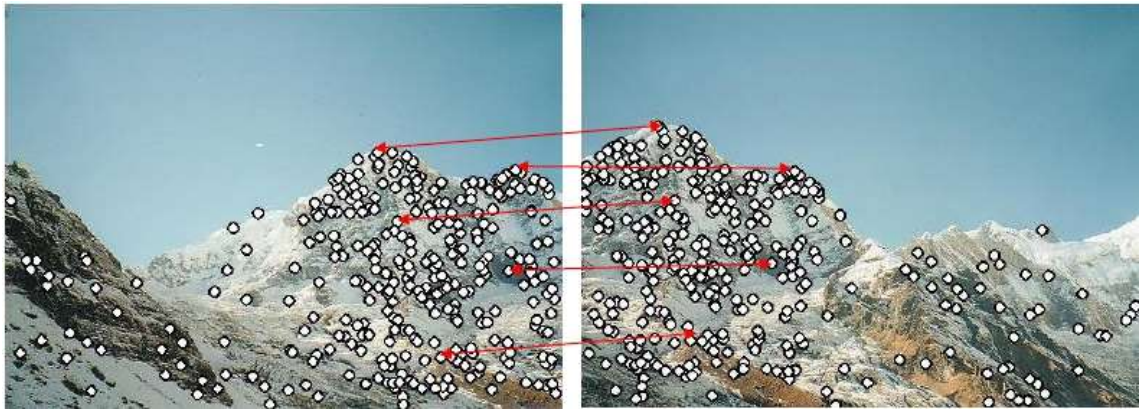
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ sx + y \end{bmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ s & 1 & 0 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



如果只有平移….

- 如何估计最优平移参数 (dx, dy) ?

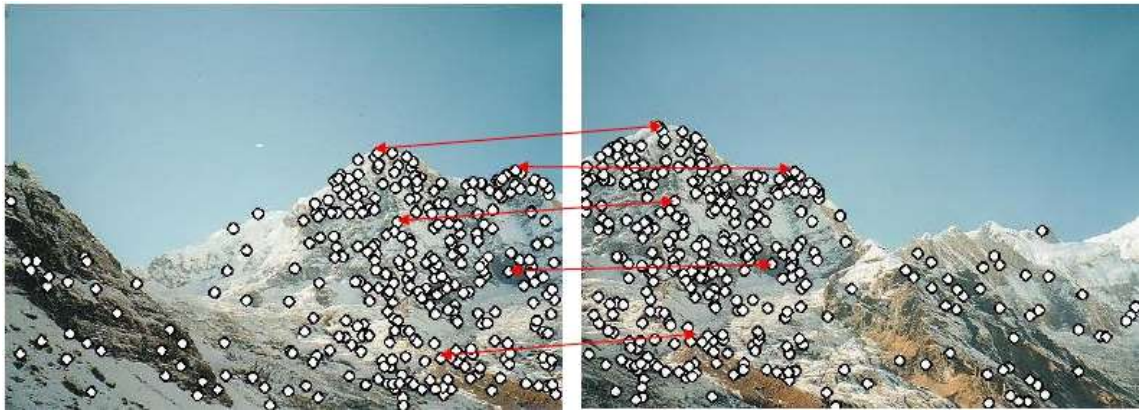
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



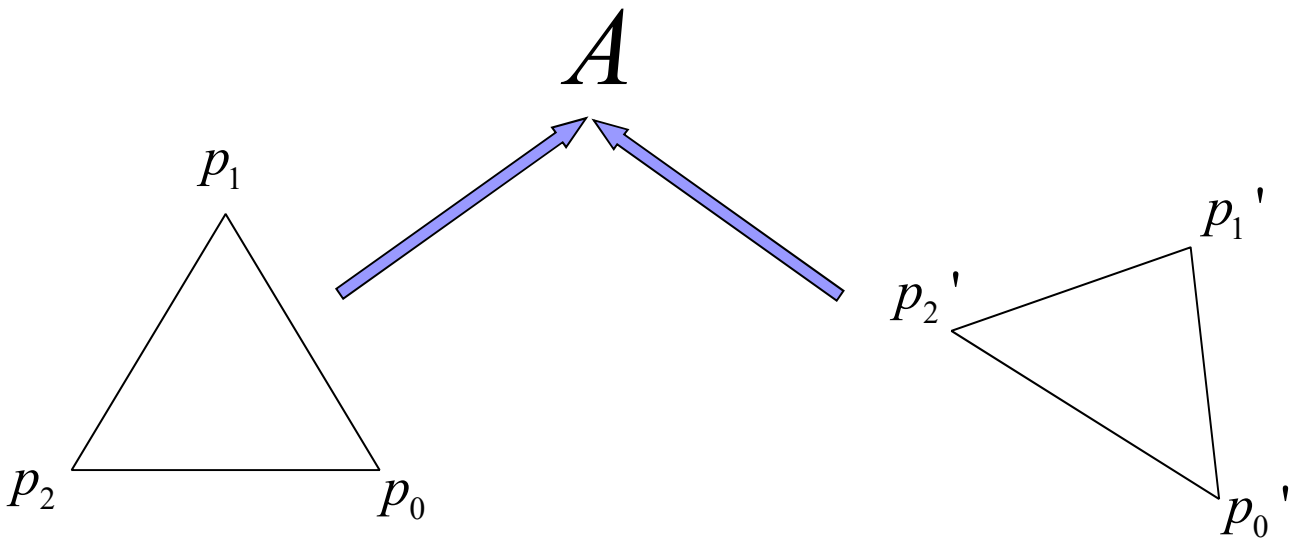
仿射变换

- 如何估计变换参数 (a,b,c,d,e,f) ?

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



- 估计两副图像之间的仿射变换?



- **不共线的3个平面点对决定一个二维仿射变换**

$$\begin{bmatrix} u_i' \\ v_i' \end{bmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \longrightarrow \begin{cases} u_0 a + v_0 b + c = u_0' \\ u_0 d + v_0 e + f = v_0' \\ u_1 a + v_1 b + c = u_1' \\ u_1 d + v_1 e + f = v_1' \\ u_2 a + v_2 b + c = u_2' \\ u_2 d + v_2 e + f = v_2' \end{cases}$$

$$p_i = [u_i, v_i]$$

- **>3 个点对?**

$$A = \arg\min_{A} \sum_{i} \| Ap_i - p_i' \|^2$$

特殊仿射变换

- 相似变换（Similar)
  - 只包含平移、旋转和等比缩放
  - 保持物体的形状

- 刚性变换（Rigid)
  - 只包含平移和旋转
  - 保持物体的形状和尺寸

特殊仿射变换

- 相似变换（Similar)
  - 只包含平移、旋转和等比缩放
  - 保持物体的形状

- 刚性变换（Rigid)
  - 只包含平移和旋转
  - 保持物体的形状和尺寸

参数化形式与求解???

相似变换

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} a & -b & t_x \\ b & a & t_y \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

刚性变换

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

$$q' = Rq + t$$

$$(R, \mathrm{t}) = \arg\min \sum_i \| Rq_i + t - q'_i \|^2$$

刚性变换

- **方法1：非线性最小二乘**

$$
\begin{aligned}
E_{\mathrm{NLS}}(\Delta \boldsymbol{p}) &= \sum_i \|\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p}+\Delta \boldsymbol{p}) - \boldsymbol{x}_i'\|^2 \\
&\approx \sum_i \|\boldsymbol{J}(\boldsymbol{x}_i; \boldsymbol{p})\Delta \boldsymbol{p} - \boldsymbol{r}_i\|^2 \\
&= \Delta \boldsymbol{p}^T \left[\sum_i \boldsymbol{J}^T \boldsymbol{J}\right] \Delta \boldsymbol{p} - 2\Delta \boldsymbol{p}^T \left[\sum_i \boldsymbol{J}^T \boldsymbol{r}_i\right] + \sum_i \|\boldsymbol{r}_i\|^2 \\
&= \Delta \boldsymbol{p}^T \boldsymbol{A} \Delta \boldsymbol{p} - 2\Delta \boldsymbol{p}^T \boldsymbol{b} + c,
\end{aligned}
$$

$$A\Delta p = b \qquad \text{或者} \qquad (A + \lambda diag(A))\Delta p = b$$

f(x;p)=Rx+t，其中p是[R, t]矩阵的参数化向量： $\mathrm{p} = (t_x, t_y, \theta)$

刚性变换的雅可比矩阵： $\begin{pmatrix} 1 & 0 & -\sin\theta x - \cos\theta y \\ 0 & 1 & \cos\theta x - \sin\theta y \end{pmatrix}$

$$\sum_{i=1}^{n} w_i \left\| (R\mathbf{p}_i + \mathbf{t}) - \mathbf{q}_i \right\|^2 .$$

1. Compute the weighted centroids of both point sets:

$$\bar{\mathbf{p}} = \frac{\sum_{i=1}^{n} w_i \mathbf{p}_i}{\sum_{i=1}^{n} w_i}, \quad \bar{\mathbf{q}} = \frac{\sum_{i=1}^{n} w_i \mathbf{q}_i}{\sum_{i=1}^{n} w_i} .$$

2. Compute the centered vectors

$$\mathbf{x}_i := \mathbf{p}_i - \bar{\mathbf{p}}, \quad \mathbf{y}_i := \mathbf{q}_i - \bar{\mathbf{q}}, \qquad i = 1, 2, \ldots, n.$$

3. Compute the $d \times d$ covariance matrix

$$S = XWY^{\mathsf{T}},$$

where X and Y are the $d \times n$ matrices that have $\mathbf{x}_i$ and $\mathbf{y}_i$ as their columns, respectively, and $W = \operatorname{diag}(w_1, w_2, \ldots, w_n)$.

4. Compute the singular value decomposition $S = U\Sigma V^{\mathsf{T}}$. The rotation we are looking for is then

$$R = V \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \\ & & & & \det(VU^{\mathsf{T}}) \end{pmatrix} U^{\mathsf{T}}.$$

5. Compute the optimal translation as

$$\mathbf{t} = \bar{\mathbf{q}} - R\bar{\mathbf{p}}.$$

仿射变换

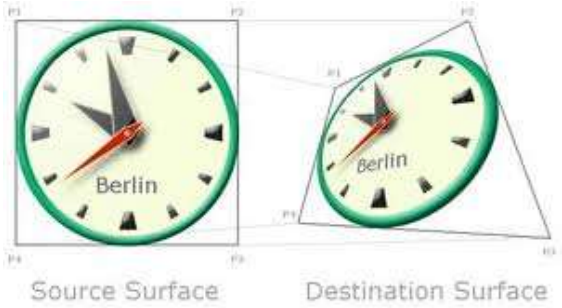
- 线性：
  - 平移
  - 相似
  - 一般仿射变换
- 非线性：
  - 旋转
  - 刚性

透视变换 (Perspective Transform & Homography)

- 平面到平面的保持直线性的映射

A projectivity is an invertible mapping from points in $\mathbb{P}^2$ (that is homogeneous 3-vectors) to points in $\mathbb{P}^2$ that maps lines to lines. More precisely,

Definition 2.9. A *projectivity* is an invertible mapping h from $\mathbb{P}^2$ to itself such that three points $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ lie on the same line if and only if $h(\mathbf{x}_1)$, $h(\mathbf{x}_2)$ and $h(\mathbf{x}_3)$ do.

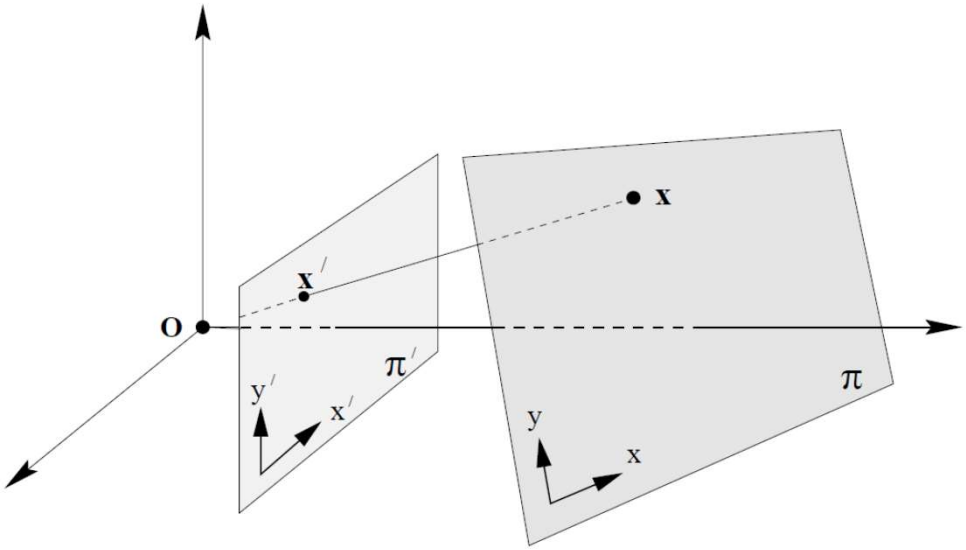


- 任意一个3x3可逆矩阵都是透视变换
- 任意透视变换都可表示为3x3可逆矩阵

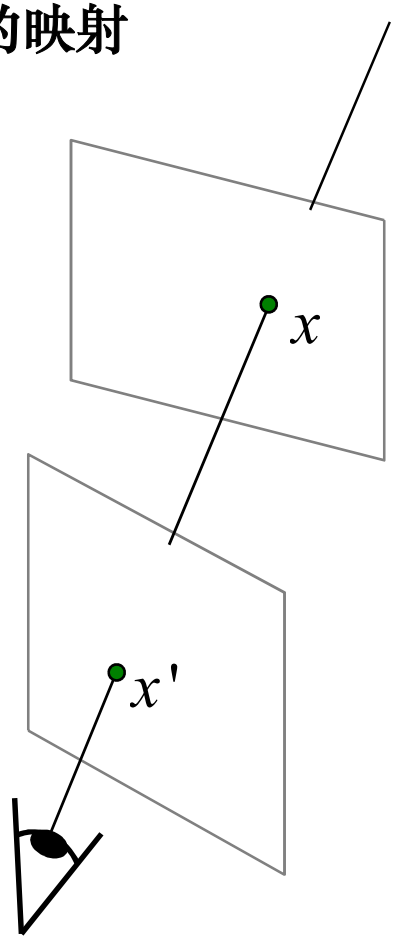
Theorem 2.10. *A mapping $h : \mathbb{P}^2 \to \mathbb{P}^2$ is a projectivity if and only if there exists a non-singular 3×3 matrix $\mathtt{H}$ such that for any point in $\mathbb{P}^2$ represented by a vector $\mathbf{x}$ it is true that $h(\mathbf{x}) = \mathtt{H}\mathbf{x}$.*

透视变换 (Perspective Transform & Homography)

- 中心投影对应的平面映射是透视变换
  - 场景（三维）中任意平面到图像平面的映射
  - 场景中同一平面在不同视点下图像之间的对应点的映射
  - 旋转相机在不同角度得到的图像之间的映射



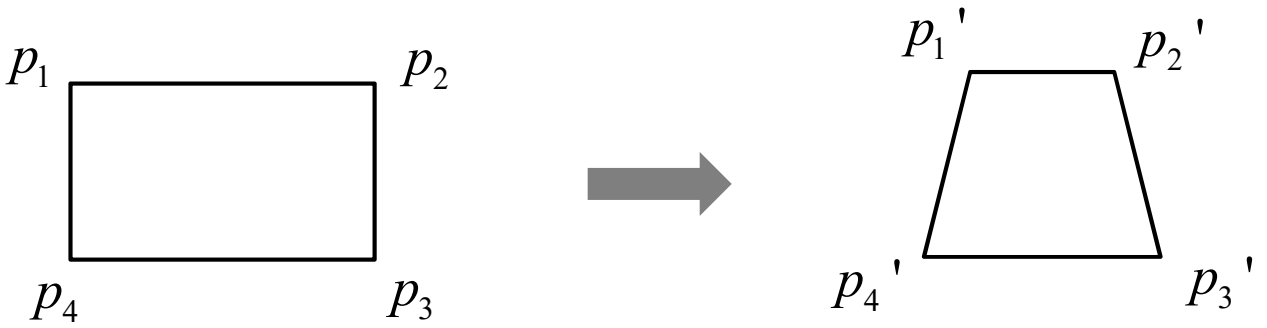
中心投影



透视变换估计

- **8个参数，最少需要4个点对**

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



透视变换估计

- **方法1：非线性最小二乘**

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{pmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = \frac{(1+h_{00})x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1}$$

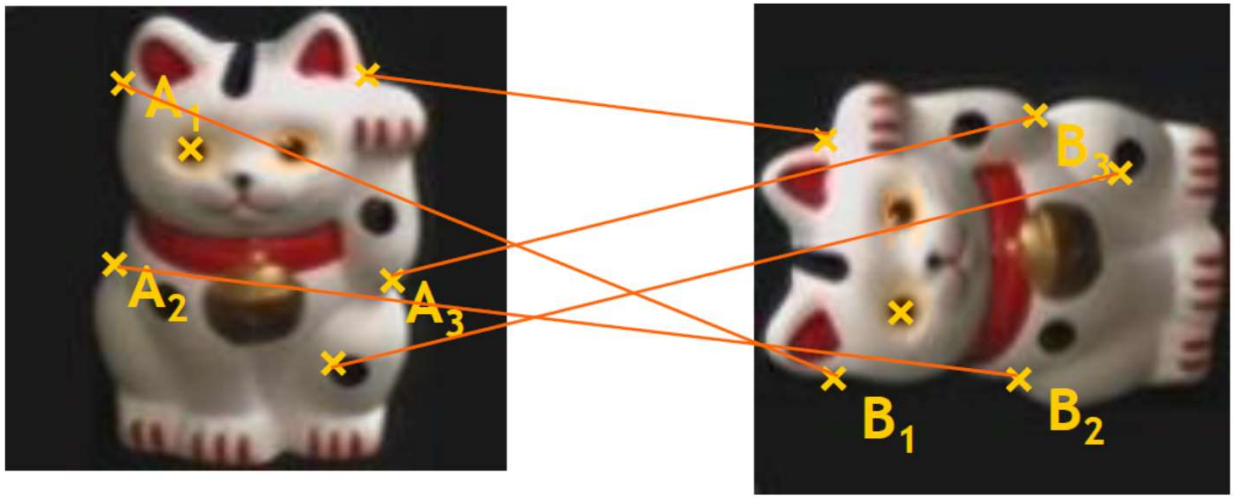
$$y' = \frac{h_{10}x + (1+h_{11})y + h_{12}}{h_{20}x + h_{21}y + 1}$$

$$\boldsymbol{J} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}} = \frac{1}{D} \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix}$$

$$D = h_{20}x + h_{21}y + 1$$

透视变换估计

- 方法2：直接线性变换（DLT，Direct Linear Transform)



Homogenous coordinates　　Image coordinates

$$\mathbf{x}_{A_1} \leftrightarrow \mathbf{x}_{B_1}$$
$$\mathbf{x}_{A_2} \leftrightarrow \mathbf{x}_{B_2}$$
$$\mathbf{x}_{A_3} \leftrightarrow \mathbf{x}_{B_3}$$
$$\vdots$$

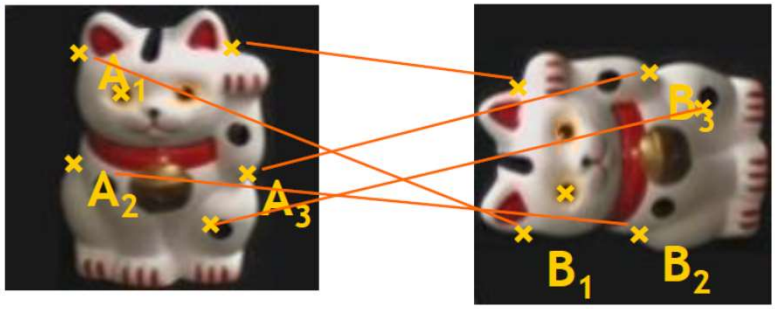
$$x_{A_1} = \frac{h_{11}x_{B_1} + h_{12}y_{B_1} + h_{13}}{h_{31}x_{B_1} + h_{32}y_{B_1} + 1} \qquad y_{A_1} = \frac{h_{21}x_{B_1} + h_{22}y_{B_1} + h_{23}}{h_{31}x_{B_1} + h_{32}y_{B_1} + 1}$$

$$x_{A_1}h_{31}x_{B_1} + x_{A_1}h_{32}y_{B_1} + x_{A_1} = h_{11}x_{B_1} + h_{12}y_{B_1} + h_{13}$$

透视变换估计

- 方法2：直接线性变换（DLT，Direct Linear Transform)

$$h_{11}x_{B_1}+h_{12}y_{B_1}+h_{13}-x_{A_1}h_{31}x_{B_1}-x_{A_1}h_{32}y_{B_1}-x_{A_1}=0$$
$$h_{21}x_{B_1}+h_{22}y_{B_1}+h_{23}-y_{A_1}h_{31}x_{B_1}-y_{A_1}h_{32}y_{B_1}-y_{A_1}=0$$



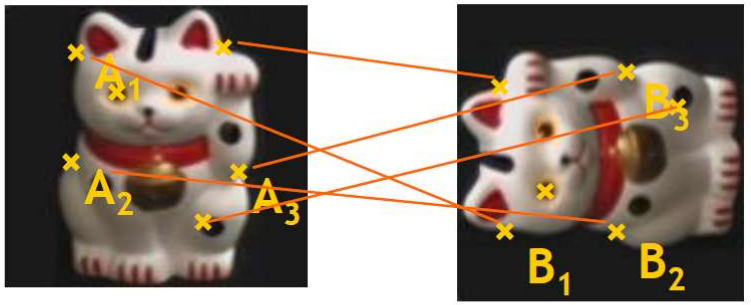
$$\mathbf{x}_{A_1}\leftrightarrow\mathbf{x}_{B_1}$$
$$\mathbf{x}_{A_2}\leftrightarrow\mathbf{x}_{B_2}$$
$$\mathbf{x}_{A_3}\leftrightarrow\mathbf{x}_{B_3}$$
$$\vdots$$

$$\begin{bmatrix} x_{B_1} & y_{B_1} & 1 & 0 & 0 & 0 & -x_{A_1}x_{B_1} & -x_{A_1}y_{B_1} & -x_{A_1} \\ 0 & 0 & 0 & x_{B_1} & y_{B_1} & 1 & -y_{A_1}x_{B_1} & -y_{A_1}y_{B_1} & -y_{A_1} \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \end{bmatrix} \cdot \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ . \end{bmatrix}$$

$$Ah=0$$

透视变换估计

- 方法2：直接线性变换（DLT，Direct Linear Transform)



- Solution:
  - Null-space vector of A
  - Corresponds to smallest singular vector

$$\mathbf{x}_{A_1} \leftrightarrow \mathbf{x}_{B_1}$$
$$\mathbf{x}_{A_2} \leftrightarrow \mathbf{x}_{B_2}$$
$$\mathbf{x}_{A_3} \leftrightarrow \mathbf{x}_{B_3}$$
$$\vdots$$

SVD

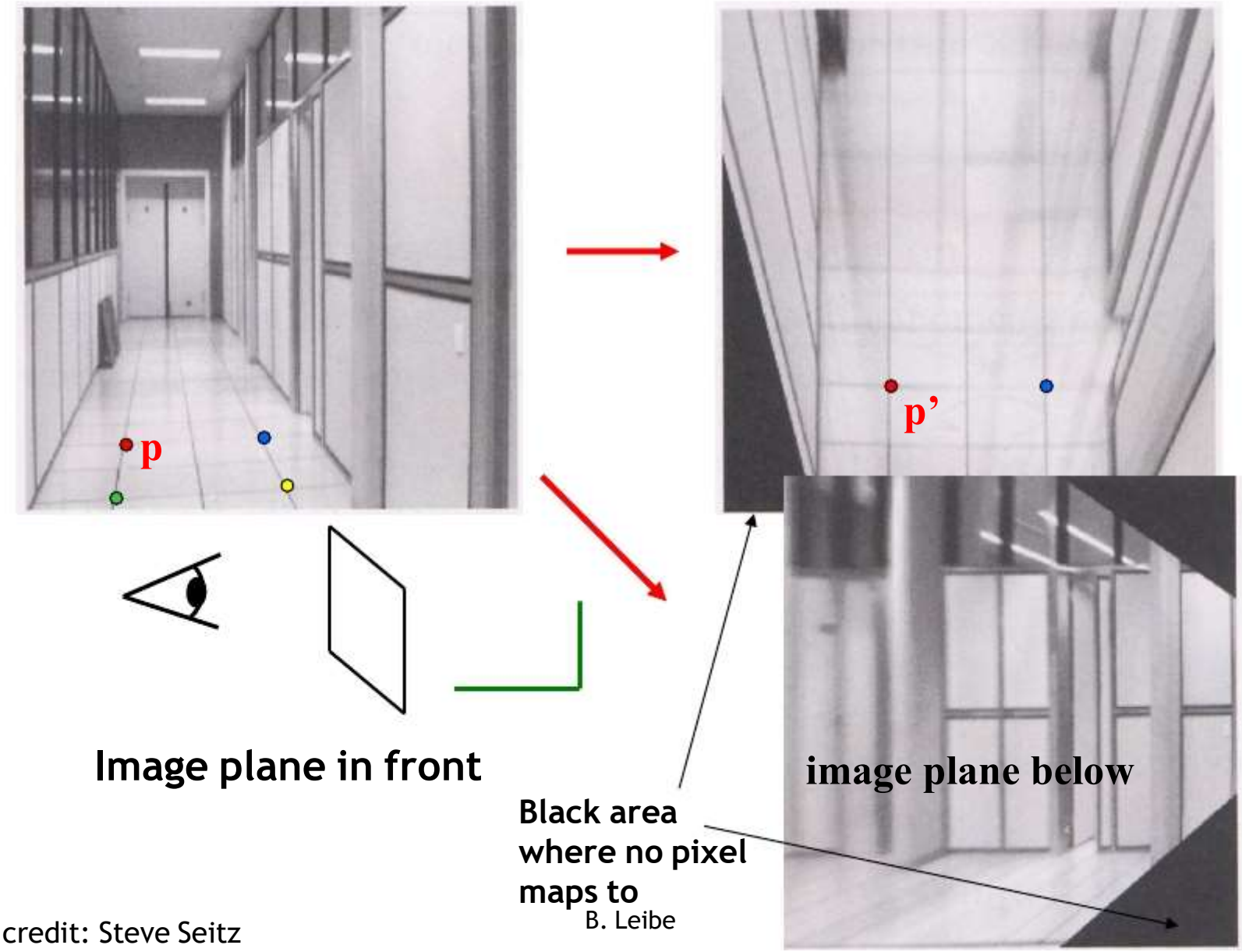
$$Ah = 0$$

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T = \mathbf{U}\begin{bmatrix} d_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & d_{99} \end{bmatrix}\begin{bmatrix} v_{11} & \cdots & v_{19} \\ \vdots & \ddots & \vdots \\ v_{91} & \cdots & v_{99} \end{bmatrix}^T$$

$$\mathbf{h} = \frac{[v_{19}, \cdots, v_{99}]}{v_{99}}$$

Minimizes least square error

Image Warping with Homographies



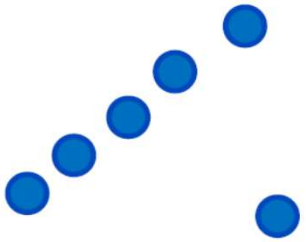
Slide credit: Steve Seitz

考虑外点(Outlier)

- 实际的特征点匹配都可能包含大量的错误

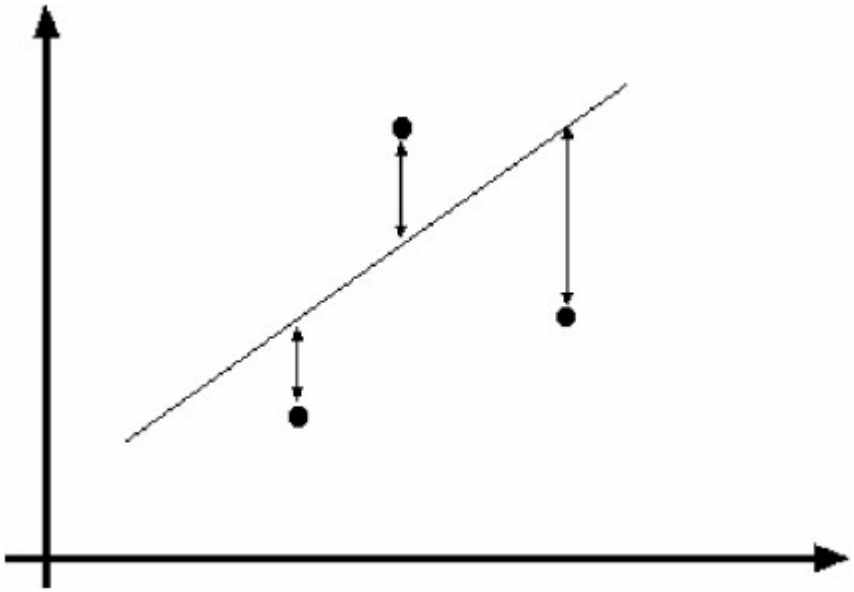
考虑外点(Outlier)

- 实际的特征点匹配都可能包含大量的错误



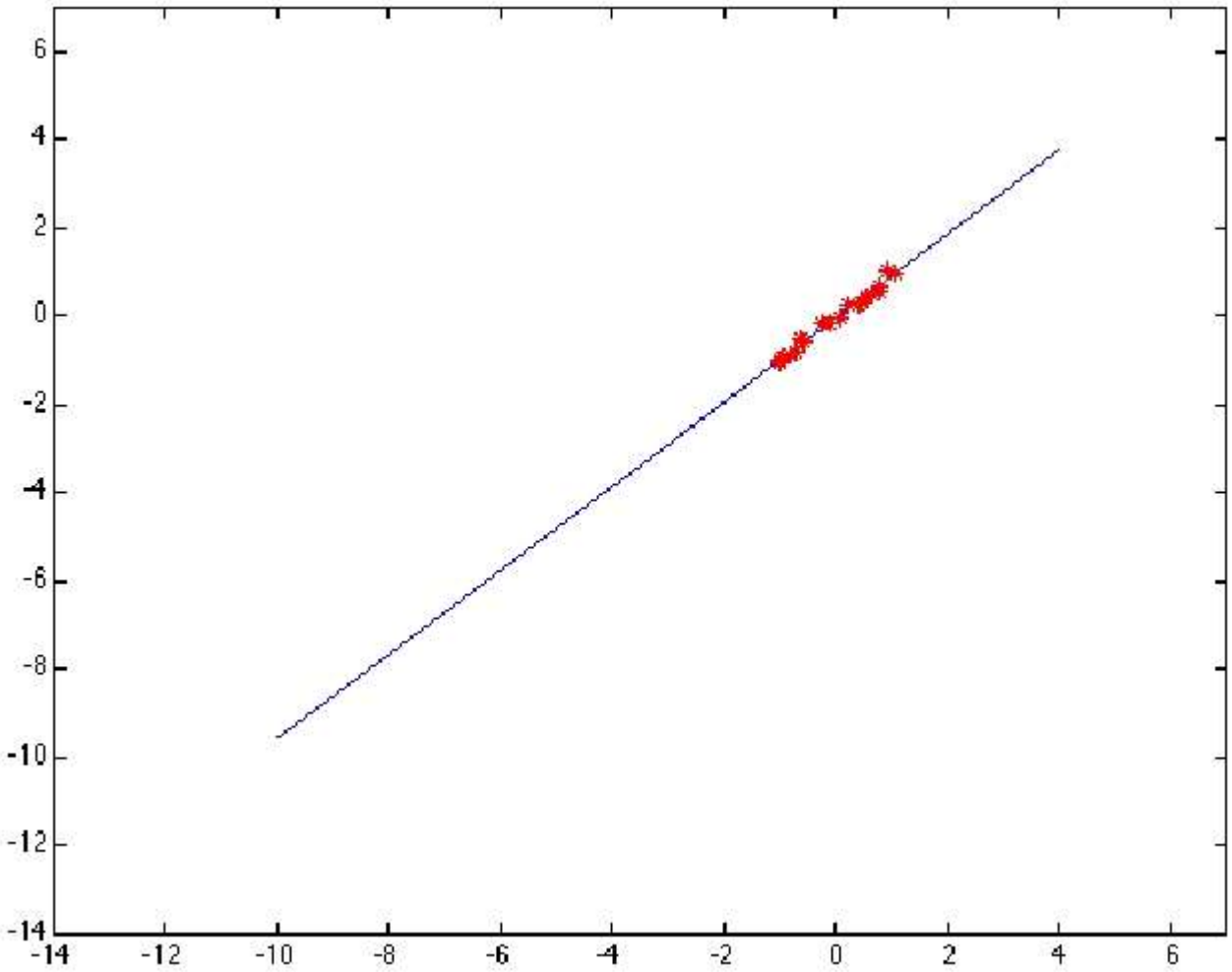
Example: Least-Squares Line Fitting

- Assuming all the points that belong to a particular line are known



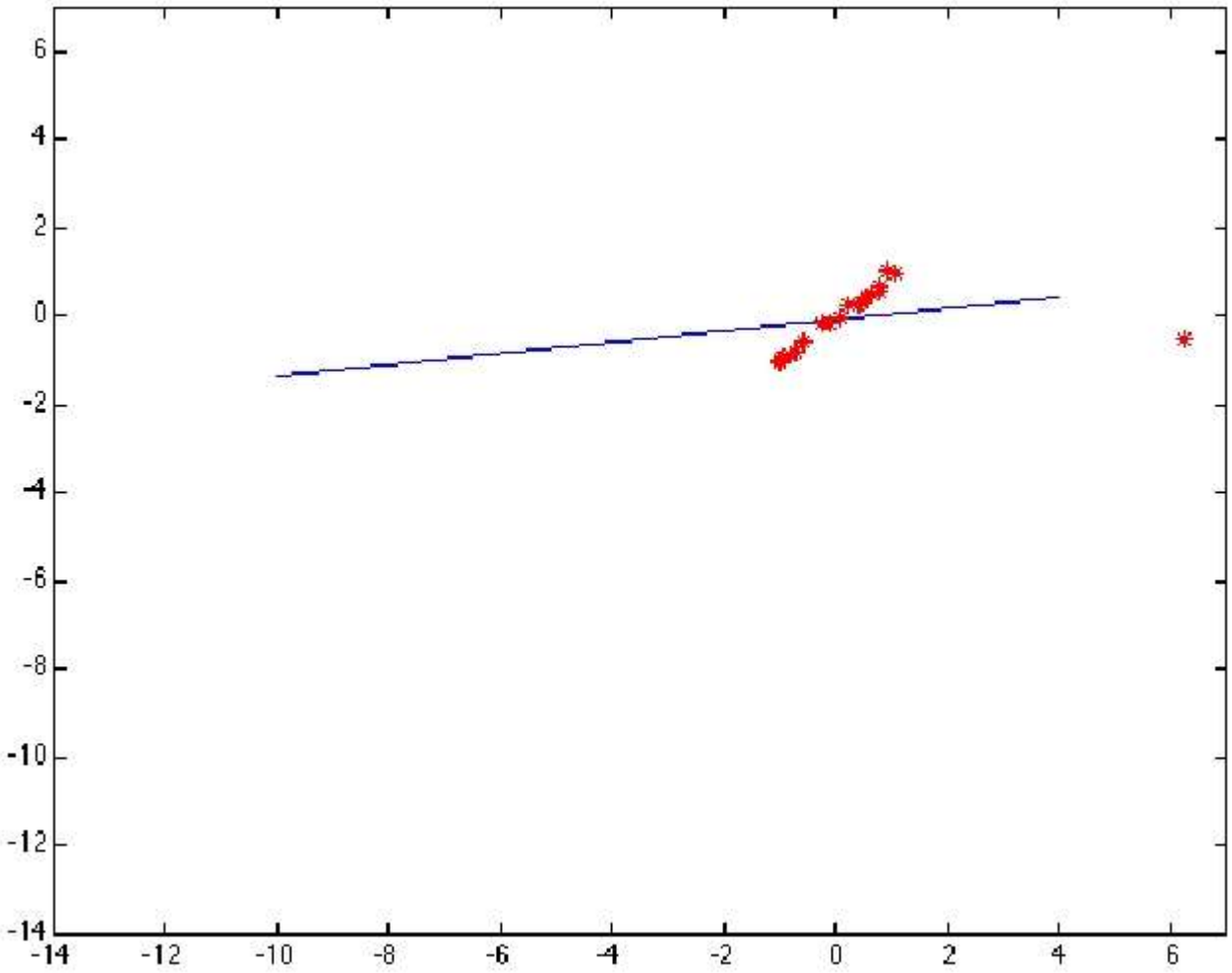
Source: Forsyth & Ponce

Outliers Affect Least-Squares Fit



Source: Forsyth & Ponce

Outliers Affect Least-Squares Fit



Source: Forsyth & Ponce

Strategy 1: RANSAC [Fischler81]

- RANdom SAmple Consensus
- Approach: we want to avoid the impact of outliers, so let's look for "inliers", and use only those.
- Intuition: if an outlier is chosen to compute the current fit, then the resulting line won't have much support from rest of the points.

Slide credit: Kristen Grauman

RANSAC

~~RANSAC loop~~:

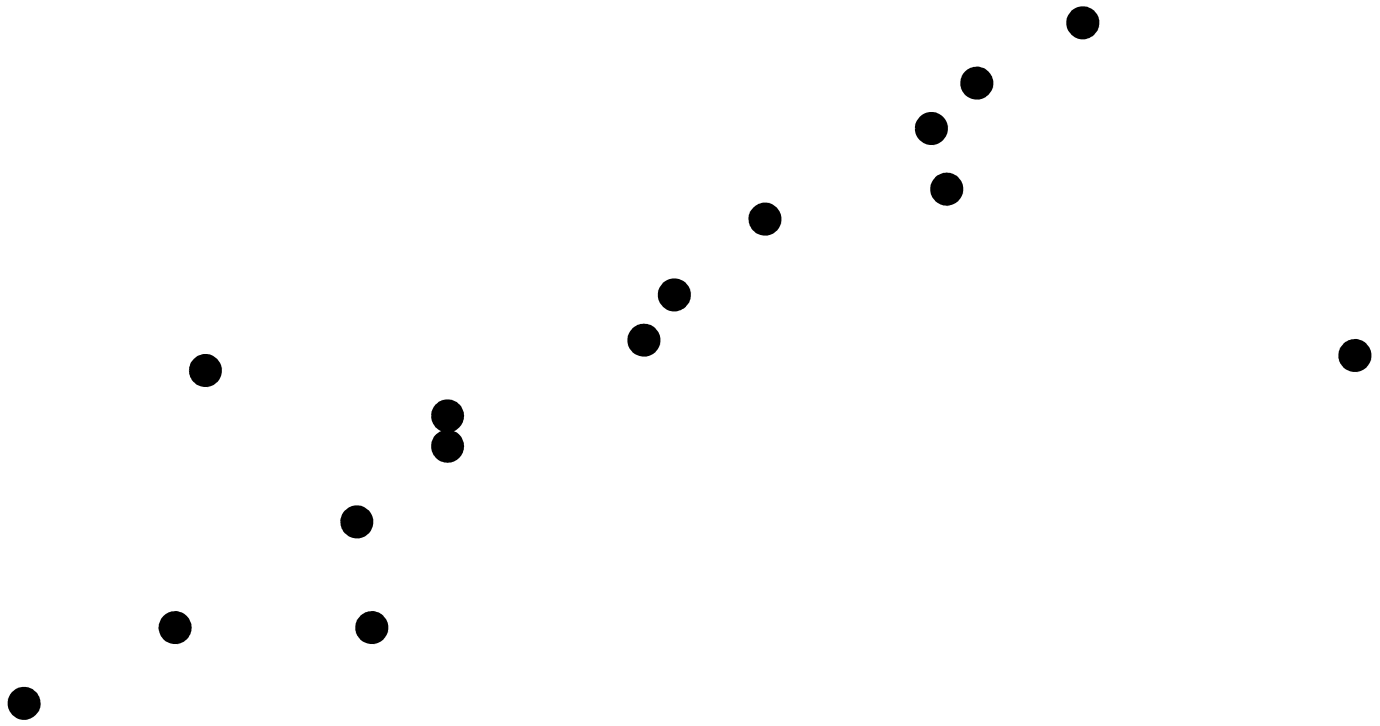
1. Randomly select a *seed group* of points on which to base transformation estimate (e.g., a group of matches)
2. Compute transformation from seed group
3. Find *inliers* to this transformation
4. If the number of inliers is sufficiently large, re-compute least-squares estimate of transformation on all of the inliers

- Keep the transformation with the largest number of inliers

Slide credit: Kristen Grauman

RANSAC Line Fitting Example

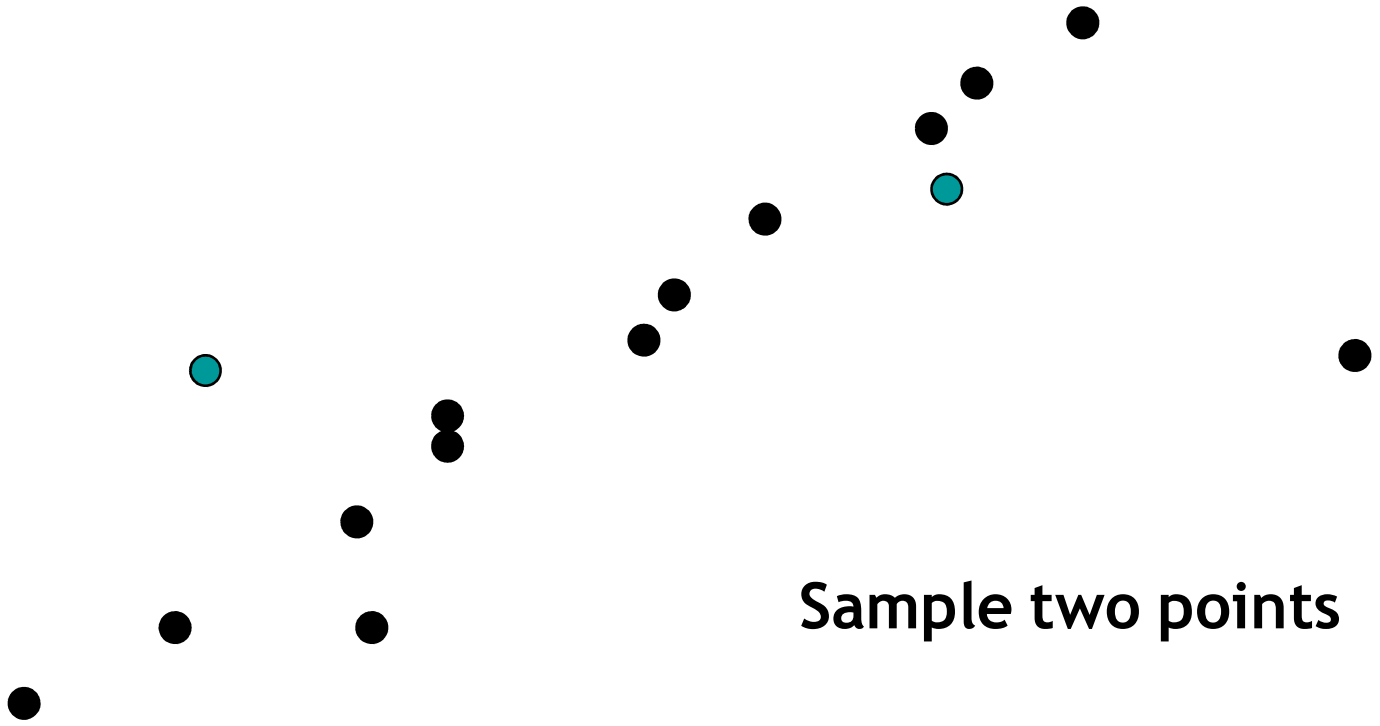
- **Task: Estimate the best line**
  - ***How many points do we need to estimate the line?***



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

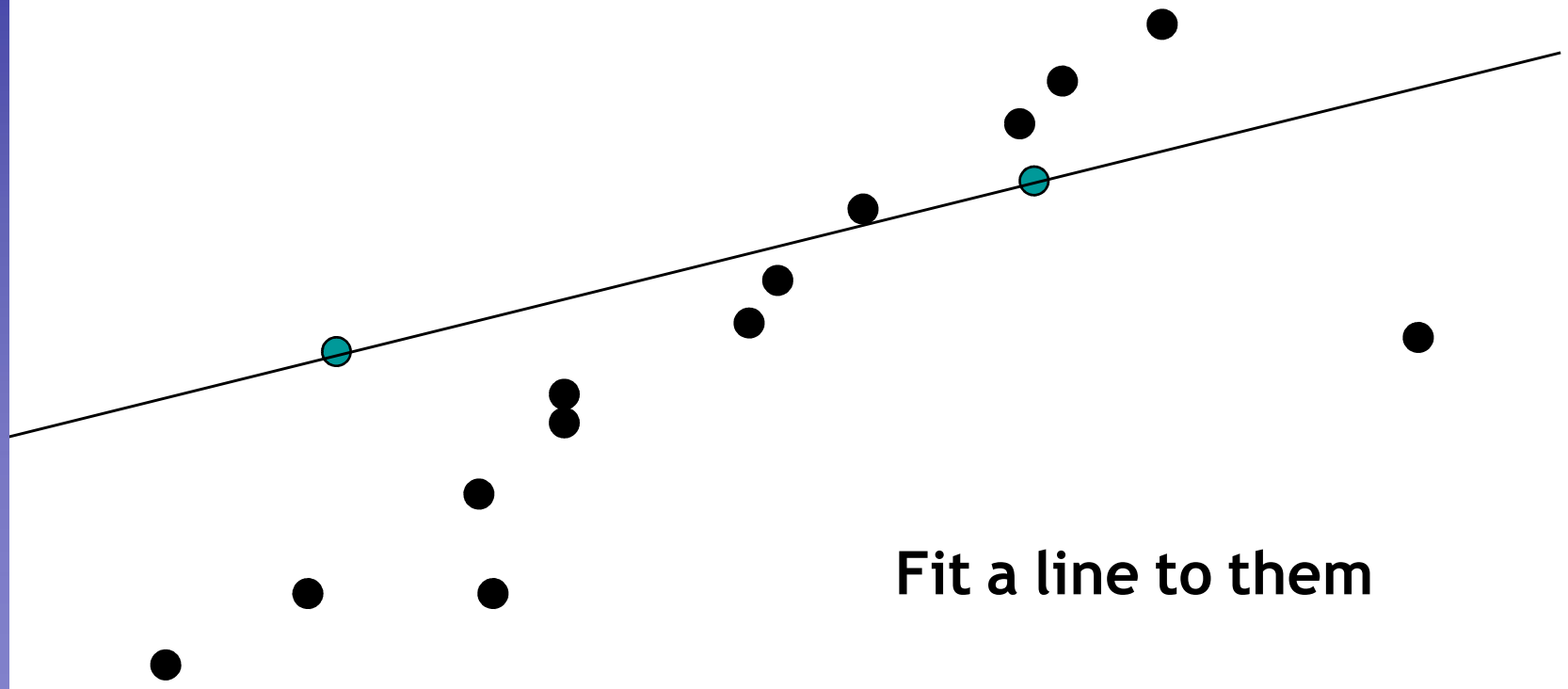
- Task: Estimate the best line



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

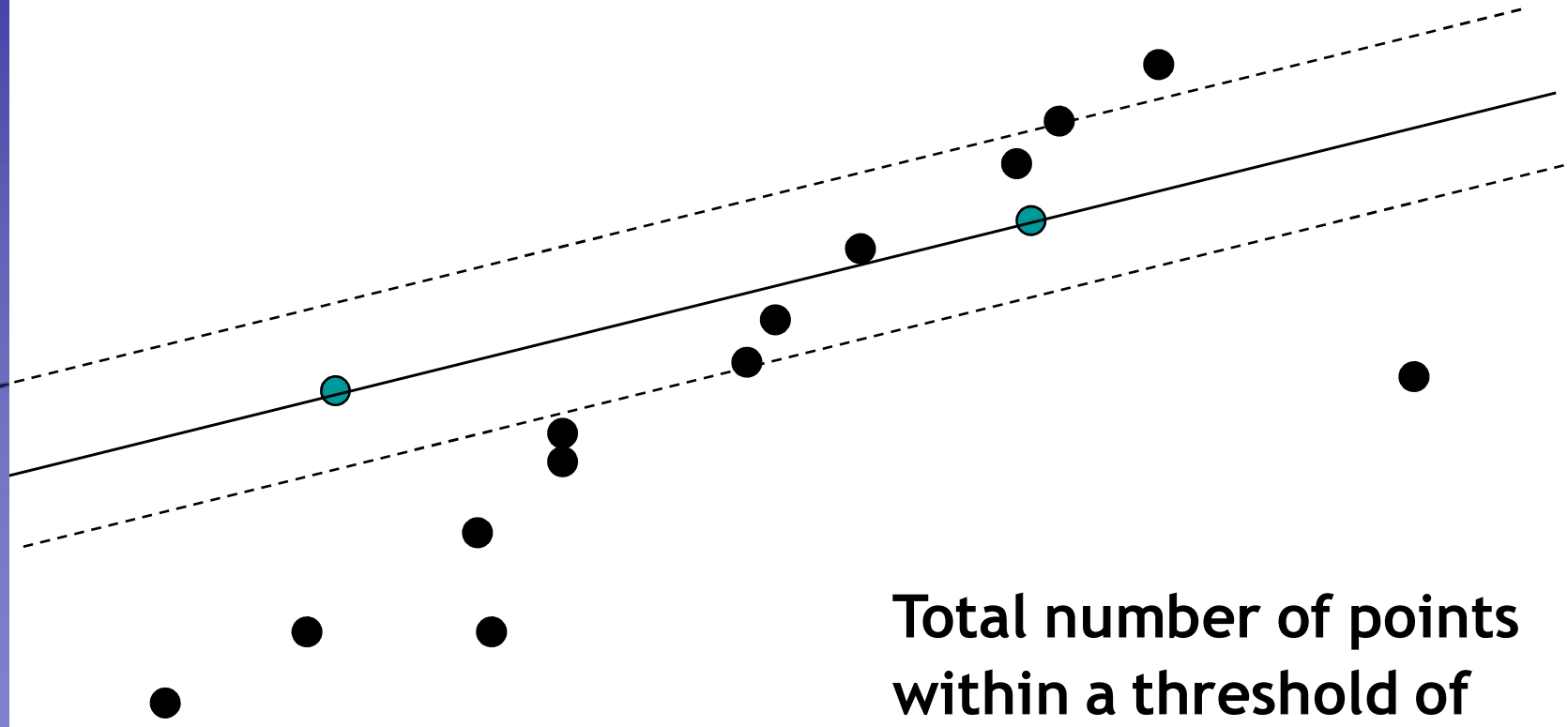
- Task: Estimate the best line



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

- Task: Estimate the best line

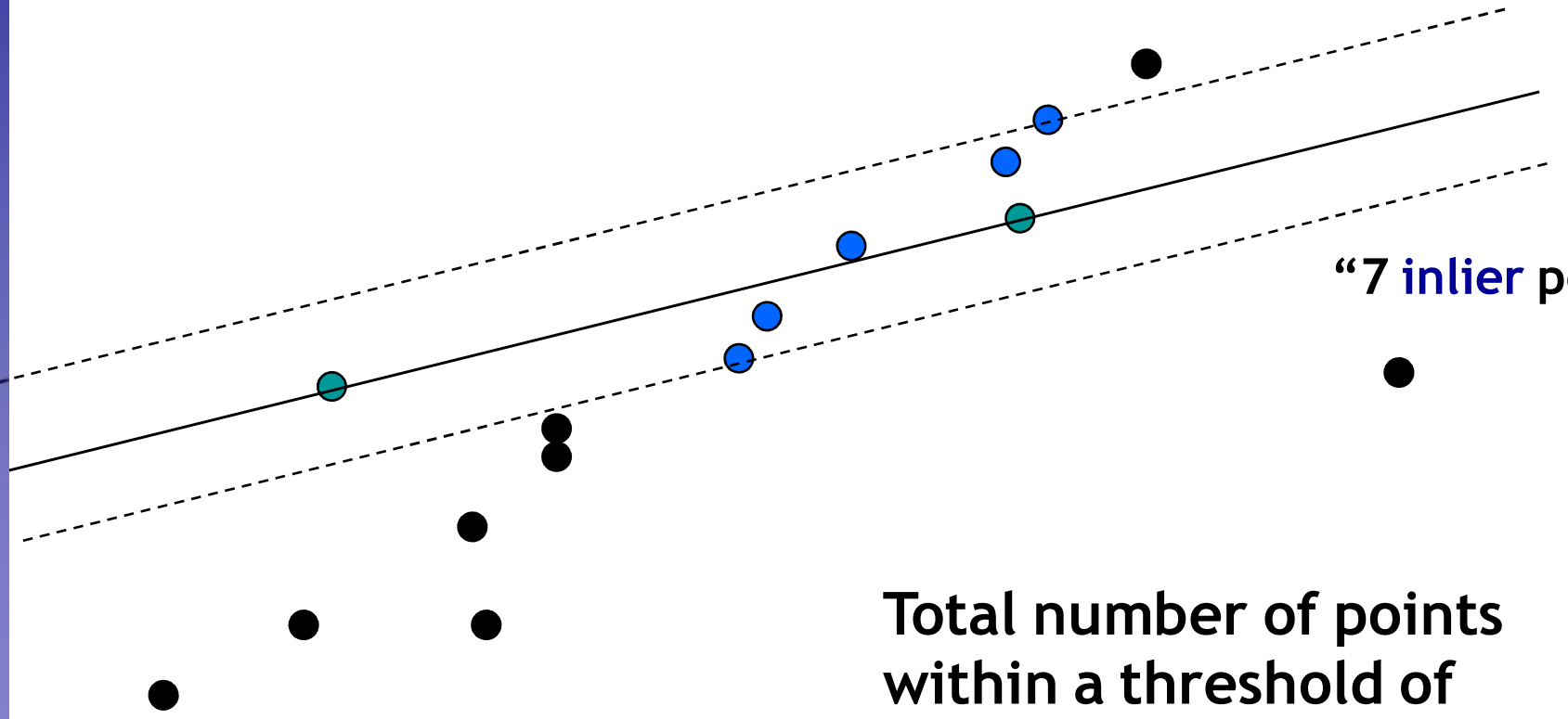


Total number of points within a threshold of line.

Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

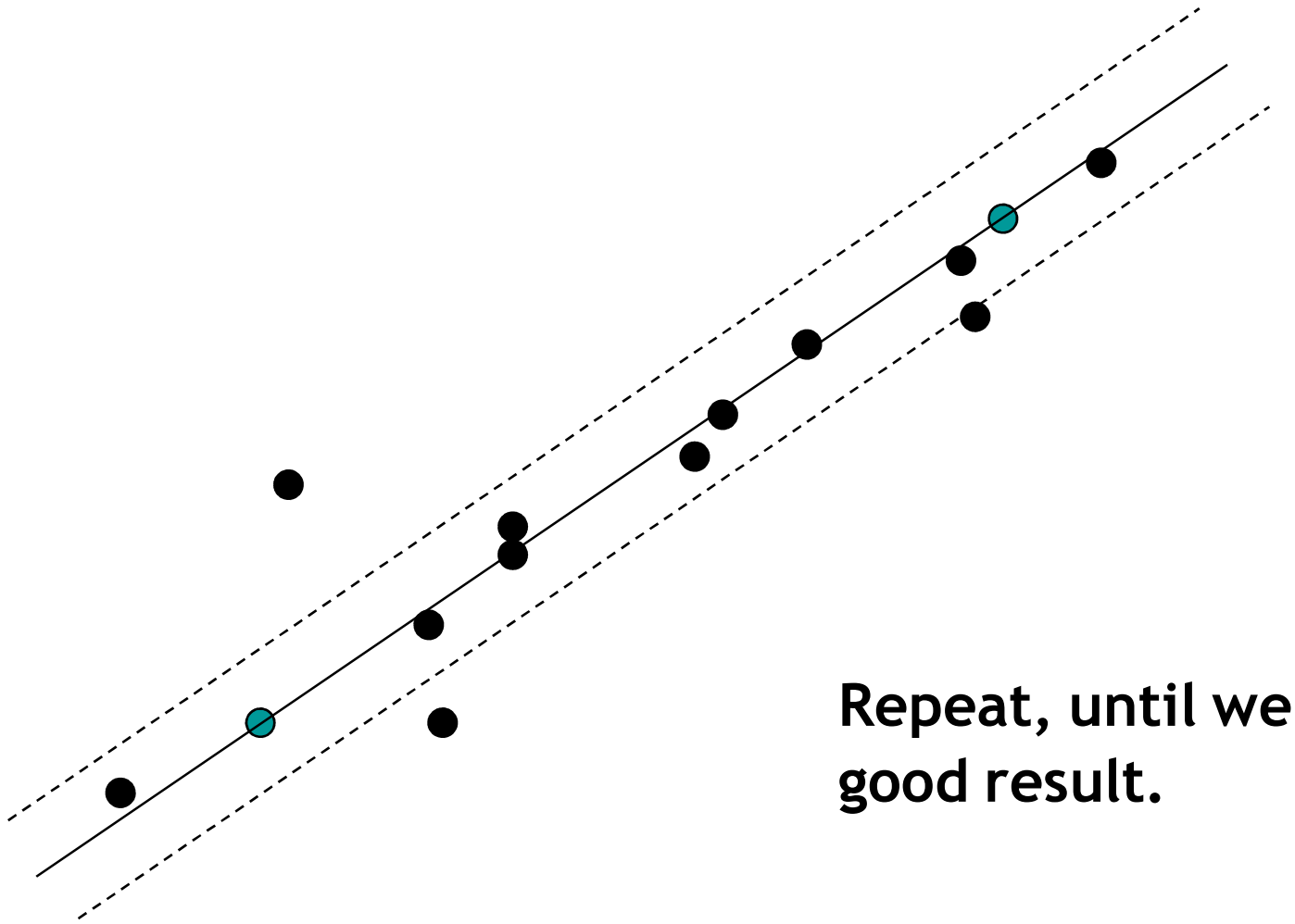
- Task: Estimate the best line



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

- Task: Estimate the best line

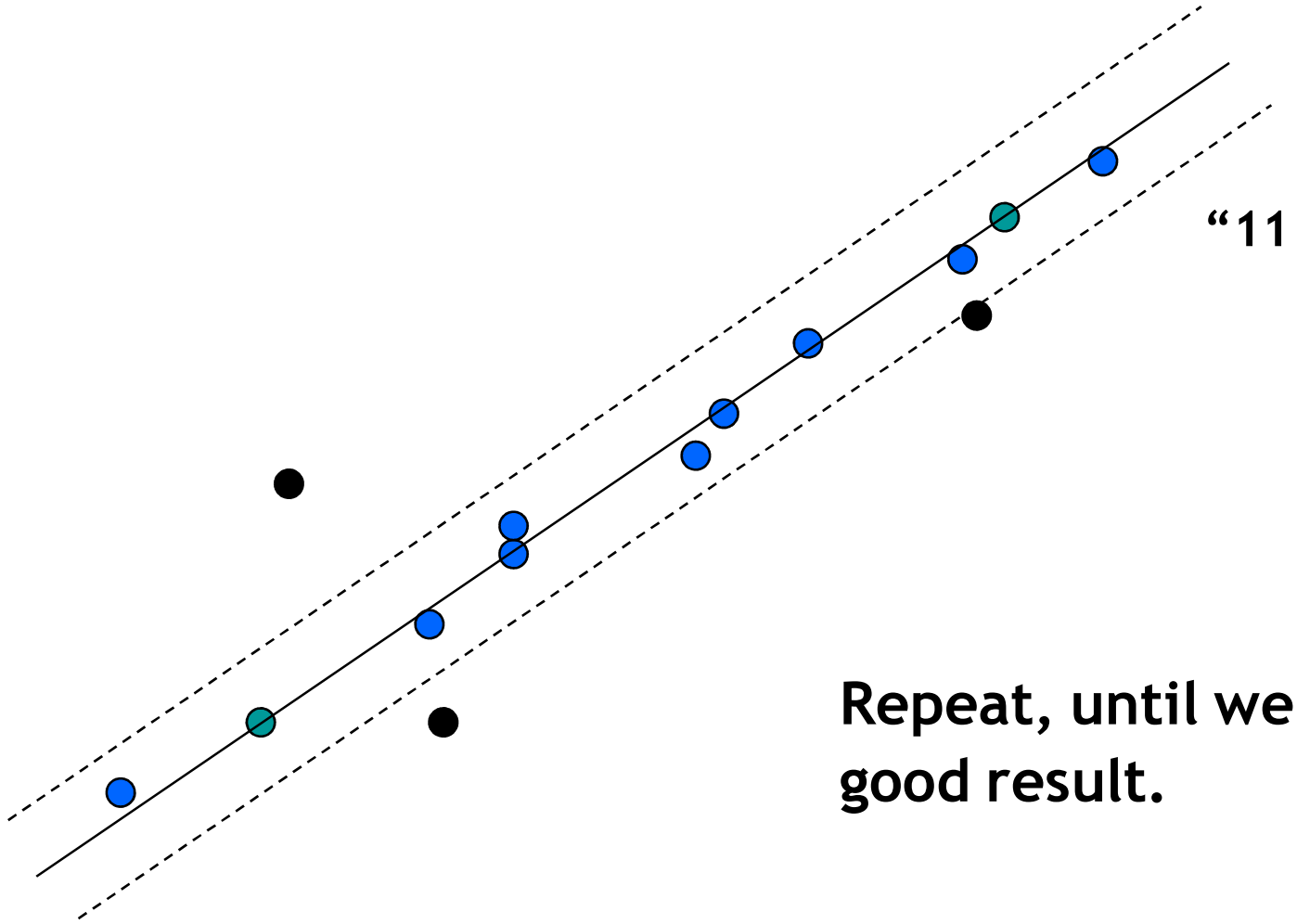


Repeat, until we get a good result.

Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

- Task: Estimate the best line



"11 inlier points"

Repeat, until we get a good result.

Slide credit: Jinxiang Chai

RANSAC: How many samples?

- How many samples are needed?
  - Suppose w is fraction of inliers (points from line).
  - n points needed to define hypothesis (2 for lines)
  - k samples chosen.
- Prob. that a single sample of n points is correct: w^n
- Prob. that all k samples fail is: p= $(1-w^n)^k$

$\Rightarrow$ Choose k high enough to keep this below desired failure rate.

Slide credit: David Lowe

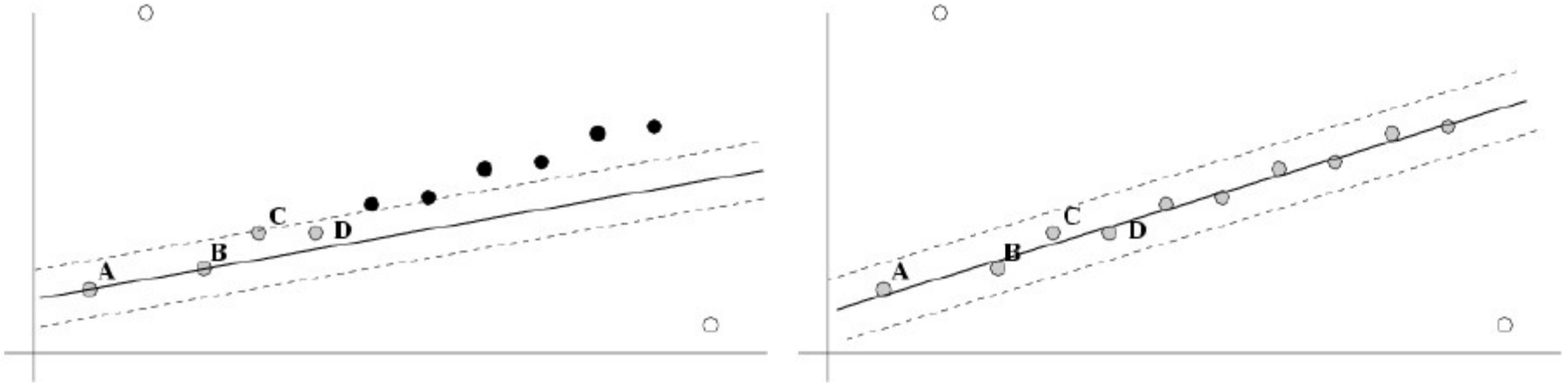
RANSAC: Computed k (p=0.01)

Sample size n	Proportion of outliers						
	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Slide credit: David Lowe

After RANSAC

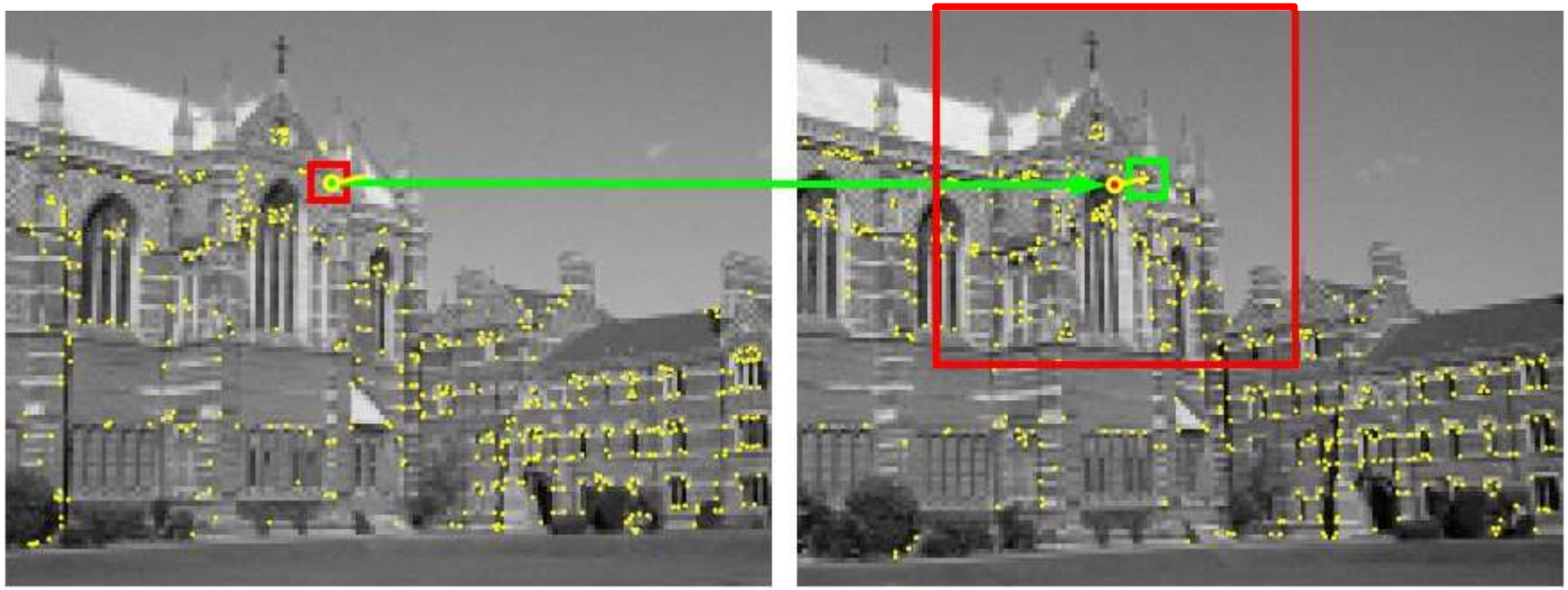
- RANSAC divides data into inliers and outliers and yields estimate computed from minimal set of inliers.
- Improve this initial estimate with estimation over all inliers (e.g. with standard least-squares minimization).
- But this may change inliers, so alternate fitting with re-classification as inlier/outlier.



Slide credit: David Lowe

Example: Finding Feature Matches

- Find best stereo match within a square search window (here 300 pixels^2)
- Global transformation model: epipolar geometry



Images from Hartley & Zisserman

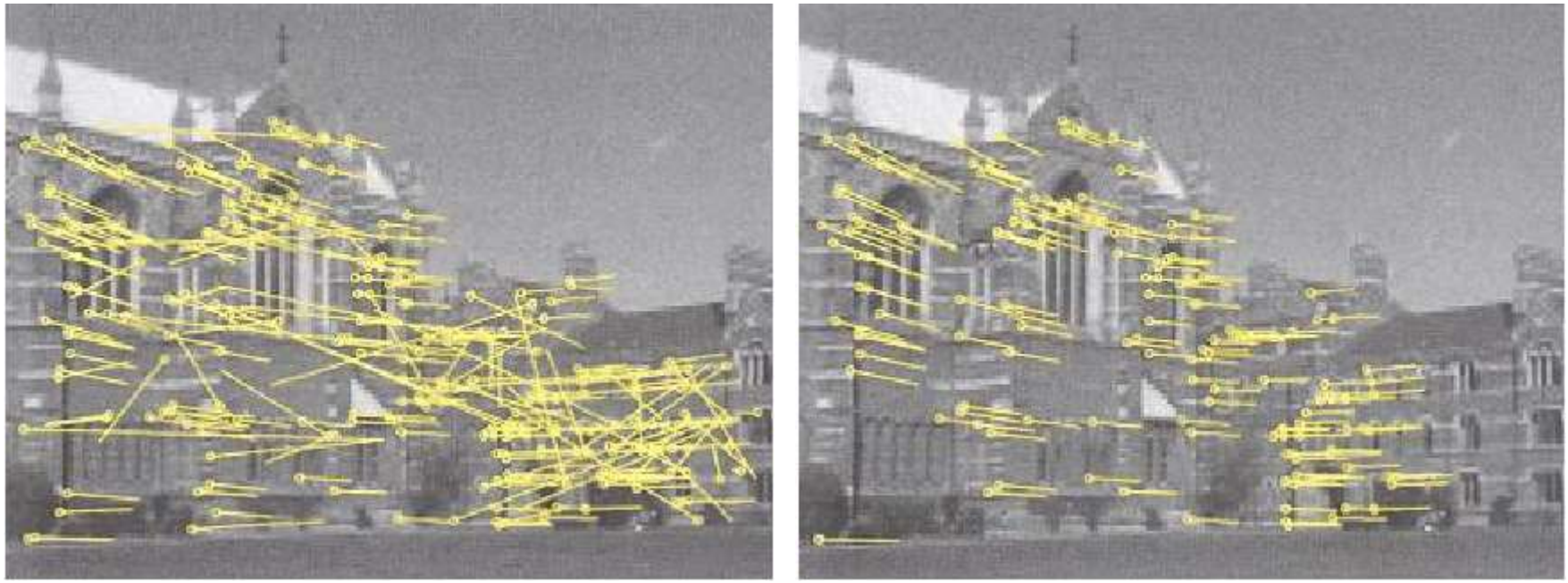
Slide credit: David Lowe

Example: Finding Feature Matches

- Find best stereo match within a square search window (here 300 pixels$_2$)
- Global transformation model: epipolar geometry

before RANSAC

after RANSAC



Images from Hartley & Zisserman

Slide credit: David Lowe

Problem with RANSAC

- In many practical situations, the percentage of outliers (incorrect putative matches) is often very high (90% or above).
- The outlier ratio is unknown.

Slide credit: Svetlana Lazebnik

OpenCV相关函数

§ findHomography() [1/2]

```
Mat cv::findHomography ( InputArray   srcPoints,
                         InputArray   dstPoints,
                         int          method = 0,
                                      ransacReprojThreshold =
                         double       3,
                         OutputArray  mask = noArray(),
                         const int    maxIters = 2000,
                         const double confidence = 0.995
                        )
```

Python:

```
retval, mask = cv.findHomography( srcPoints, dstPoints[, method[, ransacReprojThreshold[, mask[, maxIters[, confidence]]]]] )
```

Finds a perspective transformation between two planes.

Parameters

srcPoints	Coordinates of the points in the original plane, a matrix of the type CV_32FC2 or vector<Point2f> .
dstPoints	Coordinates of the points in the target plane, a matrix of the type CV_32FC2 or a vector<Point2f> .
method	Method used to compute a homography matrix. The following methods are possible:

- **0** - a regular method using all the points, i.e., the least squares method
- **RANSAC** - RANSAC-based robust method
- **LMEDS** - Least-Median robust method
- **RHO** - PROSAC-based robust method

§ estimateAffine2D()

```
cv::Mat cv::estimateAffine2D ( InputArray  from,
                               InputArray  to,
                               OutputArray inliers = noArray(),
                               int         method = RANSAC,
                               double      ransacReprojThreshold = 3,
                               size_t      maxIters = 2000,
                               double      confidence = 0.99,
                               size_t      refineIters = 10
                             )
```

Python:

```
retval, inliers = cv.estimateAffine2D( from, to[, inliers[, method[, ransacReprojThreshold[, maxIters[, confidence[, refineIters]]]]]] )
```

Computes an optimal affine transformation between two 2D point sets.

It computes

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Parameters

from	First input 2D point set containing (X, Y).
to	Second input 2D point set containing (x, y).
inliers	Output vector indicating which points are inliers (1-inlier, 0-outlier).
method	Robust method used to compute transformation. The following methods are possible: • **cv::RANSAC** - RANSAC-based robust method • **cv::LMEDS** - Least-Median robust method RANSAC is the default method.
ransacReprojThreshold	Maximum reprojection error in the RANSAC algorithm to consider a point as an inlier. Applies only to RANSAC.
maxIters	The maximum number of robust method iterations.
confidence	Confidence level, between 0 and 1, for the estimated transformation. Anything between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation.
refineIters	Maximum number of iterations of refining algorithm (Levenberg-Marquardt). Passing 0 will disable refining, so the output matrix will be output of robust method.

§ estimateAffinePartial2D()

```
cv::Mat cv::estimateAffinePartial2D ( InputArray  from,
                                      InputArray  to,
                                      OutputArray inliers = noArray(),
                                      int         method = RANSAC,
                                      double      ransacReprojThreshold = 3,
                                      size_t      maxIters = 2000,
                                      double      confidence = 0.99,
                                      size_t      refineIters = 10
                                    )
```

Python:

```
retval, inliers = cv.estimateAffinePartial2D( from, to[, inliers[, method[, ransacReprojThreshold[, maxIters[, confidence[, refineIters]]]]]] )
```

估计相似变换

$$\begin{bmatrix} \cos(\theta)\cdot s & -\sin(\theta)\cdot s & t_x \\ \sin(\theta)\cdot s & \cos(\theta)\cdot s & t_y \end{bmatrix}$$

Computes an optimal limited affine transformation with 4 degrees of freedom between two 2D point sets.

Parameters

from	First input 2D point set.
to	Second input 2D point set.
inliers	Output vector indicating which points are inliers.
method	Robust method used to compute transformation. The following methods are possible: • **cv::RANSAC** - RANSAC-based robust method • **cv::LMEDS** - Least-Median robust method RANSAC is the default method.
ransacReprojThreshold	Maximum reprojection error in the RANSAC algorithm to consider a point as an inlier. Applies only to RANSAC.
maxIters	The maximum number of robust method iterations.
confidence	Confidence level, between 0 and 1, for the estimated transformation. Anything between 0.95 and 0.99 is usually good enough. Values too close to 1 can slow down the estimation significantly. Values lower than 0.8-0.9 can result in an incorrectly estimated transformation.
refineIters	Maximum number of iterations of refining algorithm (Levenberg-Marquardt). Passing 0 will disable refining, so the output matrix will be output of robust method.

§ estimateRigidTransform()

```
Mat cv::estimateRigidTransform ( InputArray  src,
                                 InputArray  dst,
                                 bool        fullAffine
                               )
```

不是刚性变换!

Computes an optimal affine transformation between two 2D point sets.

Parameters

src	First input 2D point set stored in std::vector or **Mat**, or an image stored in **Mat**.
dst	Second input 2D point set of the same size and the same type as A, or another image.
fullAffine	If true, the function finds an optimal affine transformation with no additional restrictions (6 degrees of freedom). Otherwise, the class of transformations to choose from is limited to combinations of translation, rotation, and uniform scaling (4 degrees of freedom).

§ getAffineTransform() [1/2]

Mat cv::getAffineTransform (const **Point2f** src[],
const **Point2f** dst[]
)

Python:

retval = cv.getAffineTransform(src, dst)

Calculates an affine transform from three pairs of the corresponding points.

The function calculates the 2×3 matrix of an affine transform so that:

$$\begin{bmatrix} x'_i \\ y'_i \end{bmatrix} = \texttt{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

where

$$dst(i) = (x'_i, y'_i), src(i) = (x_i, y_i), i = 0, 1, 2$$

§ getPerspectiveTransform() [1/2]

```
Mat cv::getPerspectiveTransform ( InputArray  src,
                                  InputArray  dst,
                                  int         solveMethod = DECOMP_LU
                                )
```

Python:

```
retval = cv.getPerspectiveTransform( src, dst[, solveMethod] )
```

Calculates a perspective transform from four pairs of the corresponding points.

The function calculates the 3×3 matrix of a perspective transform so that:

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = \texttt{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

where

$$dst(i) = (x'_i, y'_i), src(i) = (x_i, y_i), i = 0, 1, 2, 3$$

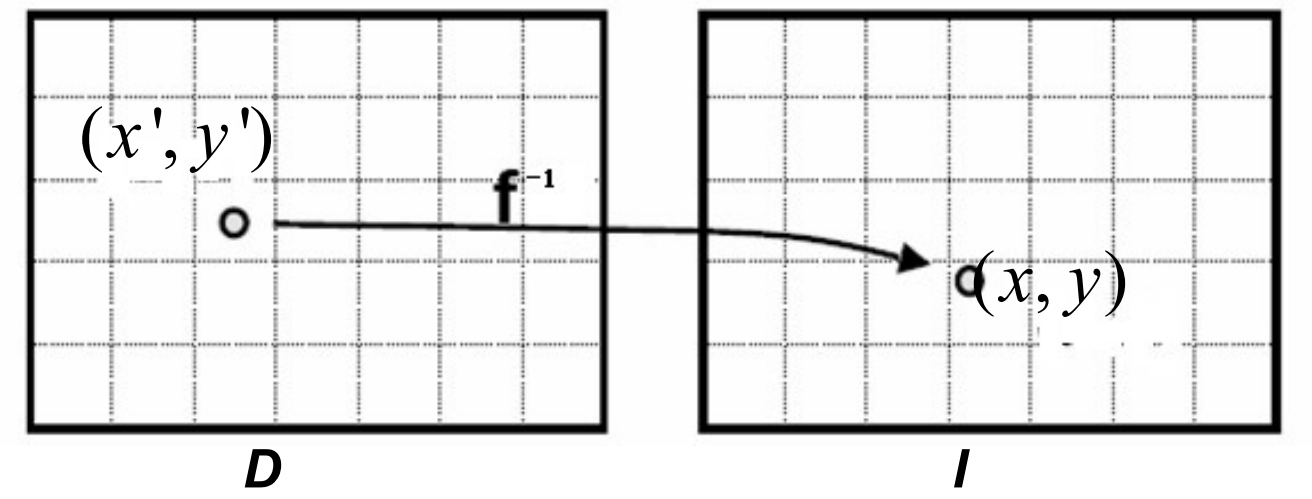
Parameters

src	Coordinates of quadrangle vertices in the source image.
dst	Coordinates of the corresponding quadrangle vertices in the destination image.
solveMethod	method passed to **cv::solve** (**DecompTypes**)

基于变换矩阵对齐图像

- **Image Warp (变形/卷绕)**

$$D = f(I)$$



§ warpPerspective()

```
void cv::warpPerspective ( InputArray     src,
                           OutputArray    dst,
                           InputArray     M,
                           Size           dsize,
                           int            flags = INTER_LINEAR,
                           int            borderMode = BORDER_CONSTANT,
                           const Scalar & borderValue = Scalar()
                         )
```

Python:

```
dst = cv.warpPerspective( src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]] )
```

Applies a perspective transformation to an image.

The function warpPerspective transforms the source image using the specified matrix:

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

§ warpAffine()

```
void cv::warpAffine ( InputArray     src,
                      OutputArray    dst,
                      InputArray     M,
                      Size           dsize,
                      int            flags = INTER_LINEAR,
                      int            borderMode =
                                     BORDER_CONSTANT,
                      const Scalar & borderValue = Scalar()
                    )
```

Python:

```
dst = cv.warpAffine( src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]] )
```

Applies an affine transformation to an image.

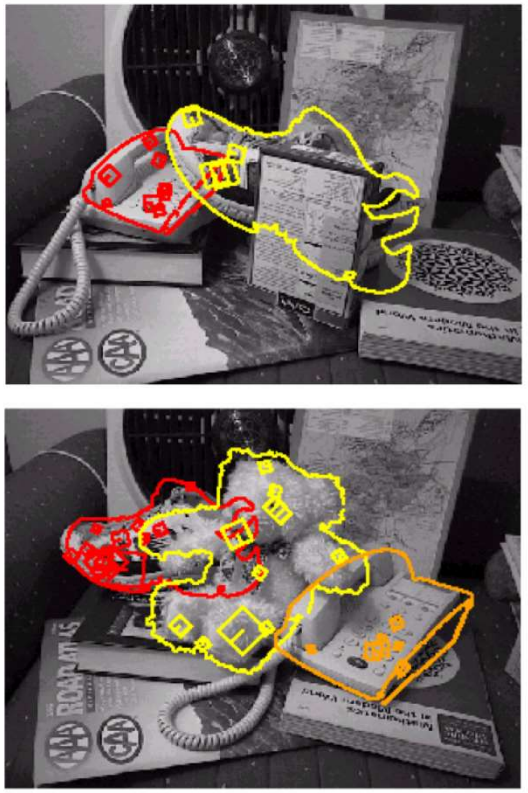
The function warpAffine transforms the source image using the specified matrix:

$$\texttt{dst}(x, y) = \texttt{src}(\texttt{M}_{11}x + \texttt{M}_{12}y + \texttt{M}_{13}, \texttt{M}_{21}x + \texttt{M}_{22}y + \texttt{M}_{23})$$

基于匹配的识别

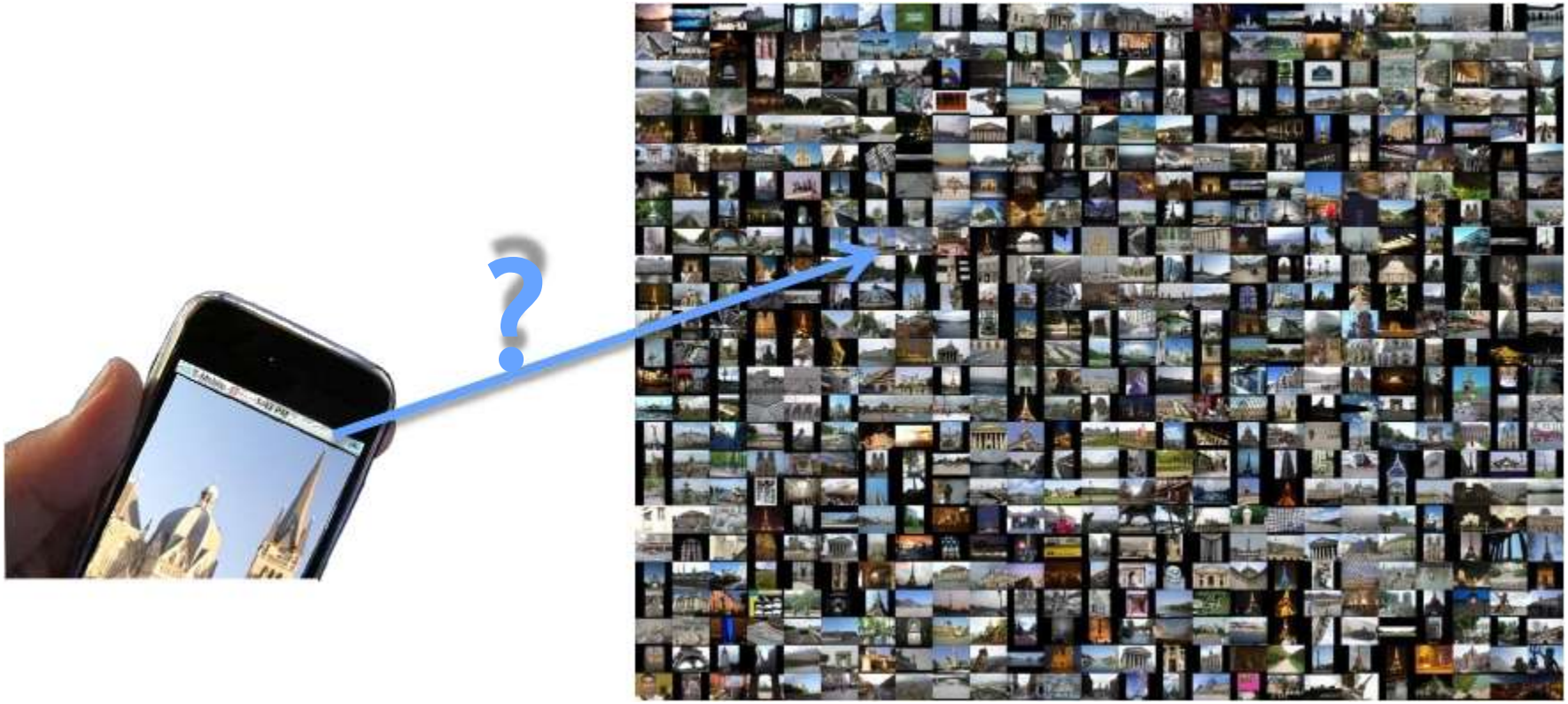
Background subtract for model boundaries

Objects recognized



Recognition in spite of occlusion

Large-Scale Image Matching Problem



Database with thousands (millions) of images

- How can we perform this matching step efficiently?

Indexing Local Features

- Each patch / region has a descriptor, which is a point in some high-dimensional feature space (e.g., SIFT)

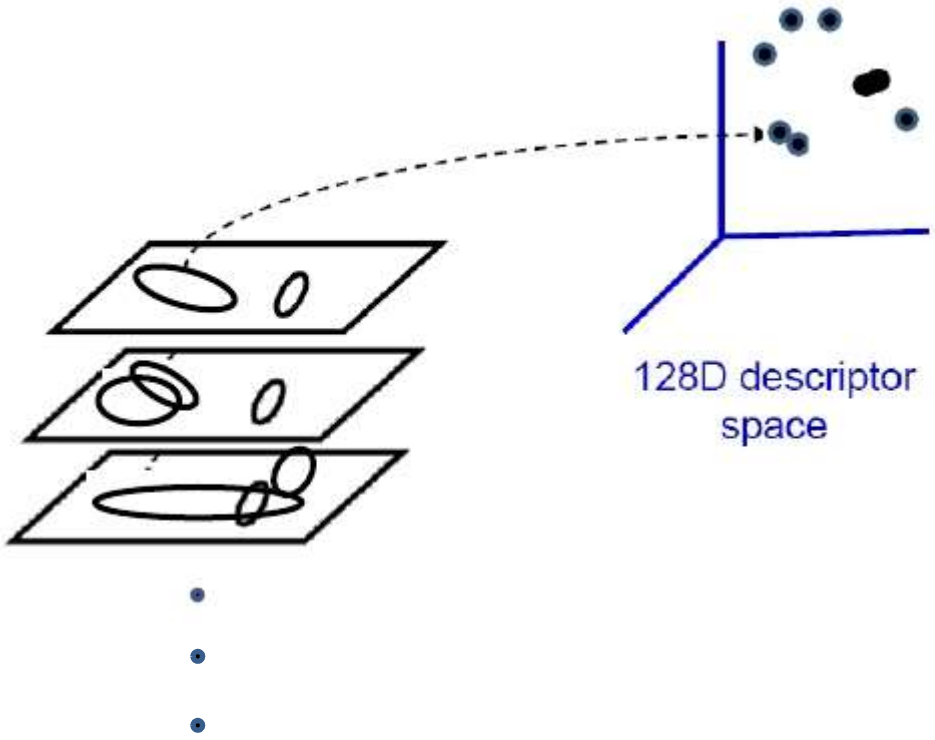
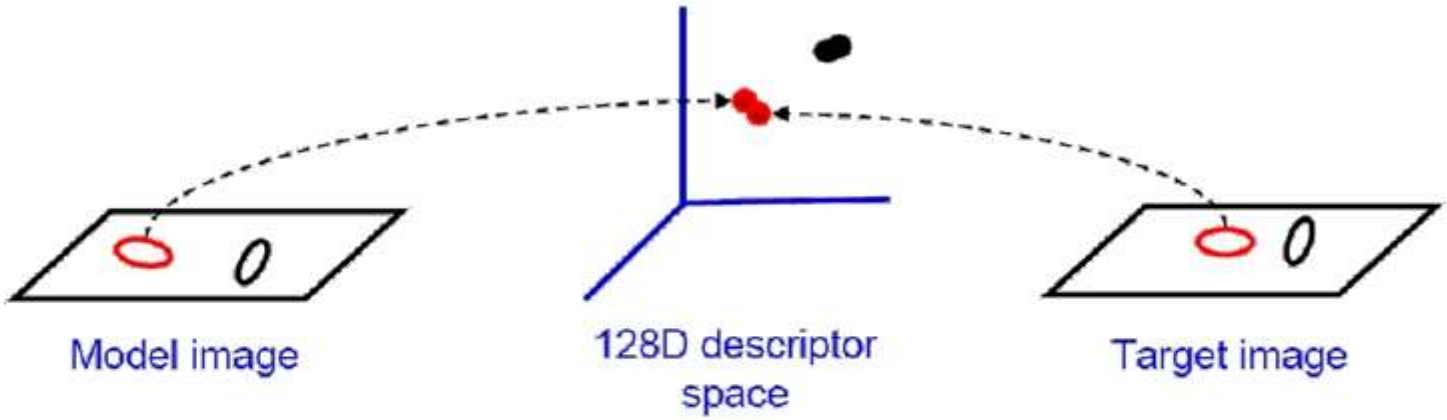


Figure credit: A. Zisserman

Indexing Local Features

- When we see close points in feature space, we have similar descriptors, which indicates similar local content.



- This is of interest for many applications
  - E.g. Image matching,
  - E.g. Retrieving images of similar objects,
  - E.g. Object recognition, categorization, 3d Reconstruction,…

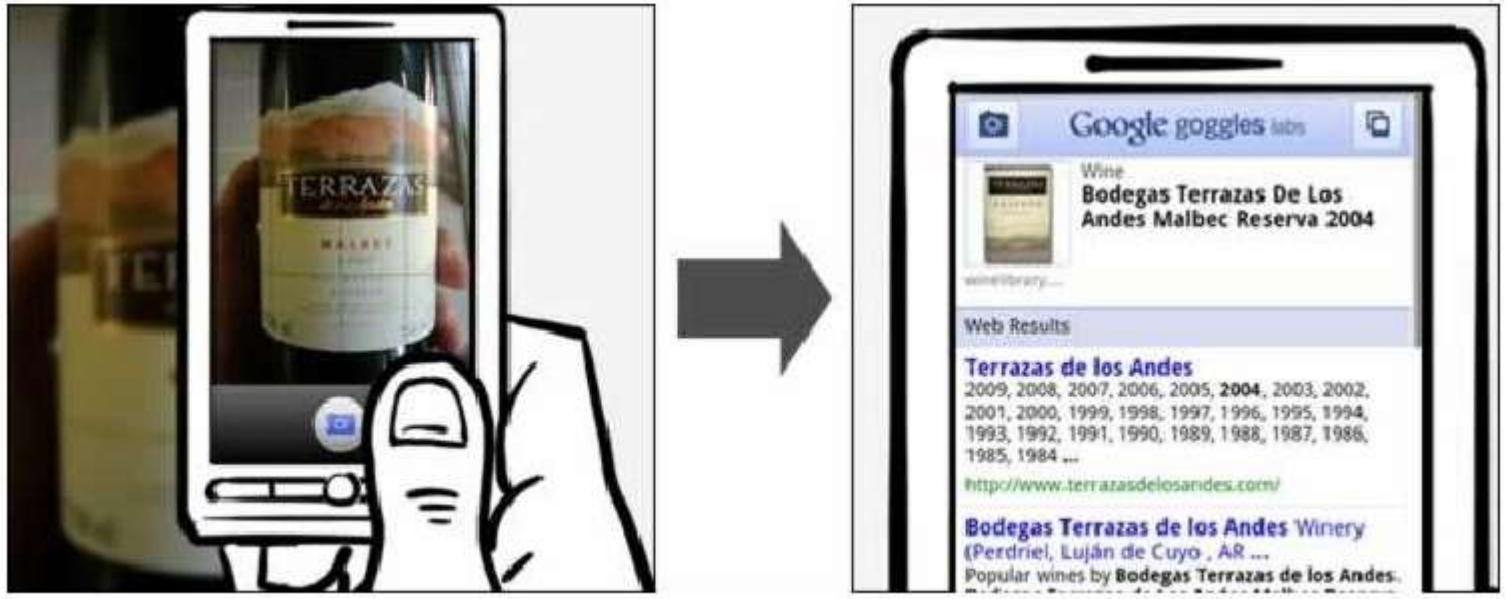
Figure credit: A. Zisserman

Application: Mobile Visual Search

Google goggles labs

Google Goggles in Action

Click the icons below to see the different ways Google Goggles can be used.



- Take photos of objects as queries for visual search

Video Google System

1. Collect all words within query region
2. Inverted file index to find relevant frames
3. Compare word counts
4. Spatial verification

Sivic & Zisserman, ICCV 2003

- Demo online at :
  http://www.robots.ox.ac.uk/~vgg/research/vgoogle/index.html

Query region

Retrieved frames

Slide credit: Kristen Grauman

Collecting Words Within a Query Region

- **Example: Friends**

Query region:

pull out only the SIFT descriptors whose positions are within the polygon

Slide credit: Kristen Grauman

Example Results

Query

raw nn 1sim=0.56697

raw nn 2sim=0.56163

raw nn 5sim=0.54917

Slide credit: Kristen Grauman

More Results

Query

Retrieved shots

Slide credit: Kristen Grauman

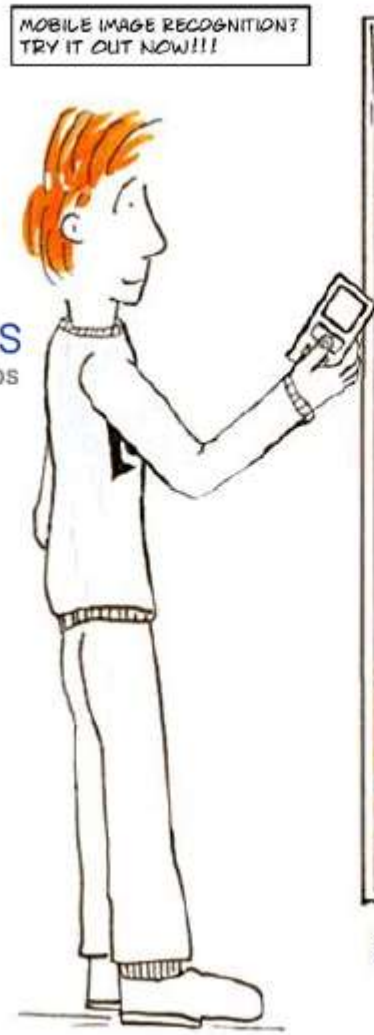
Applications: Specific Object Recognition

- **Commercial services coming out:**

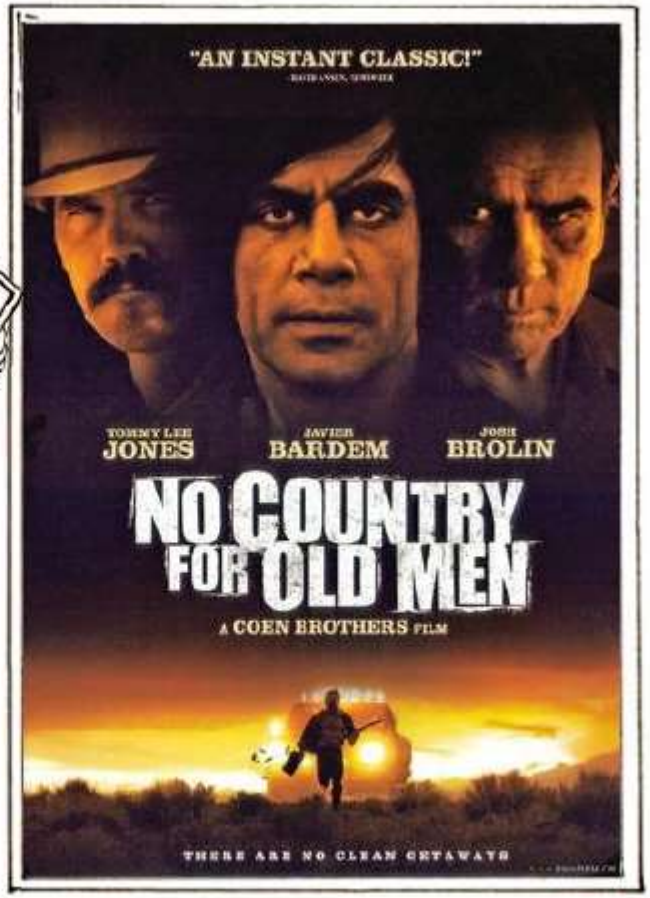
kooaba

Google goggles labs

amazon



kooaba



Works well for mostly planar objects:

- Movie posters,
- Book covers,
- CD/DVD covers,
- Video games,
- …

(~20M images indexed)

Source: http://www.kooaba

Applications: Fast Image Registration