

Assegnamento 2

Corso di Visione Artificiale
A.A. 2019/2020

Regole

- L'assegnamento vale **3** punti se consegnato entro
13 gennaio 2020
- L'assegnamento vale **1** punto se consegnato entro fine gennaio
- Consegnare: mail a **bertozzi@ce.unipr.it**
 - Subject: A2 <n. matricola>

Regole

- **Funzioni OpenCv **NON** consentite:**

`cv::cornerHarris e altri detector di keypoint`

`cv::findHomography(points1, points0, CV_RANSAC)`

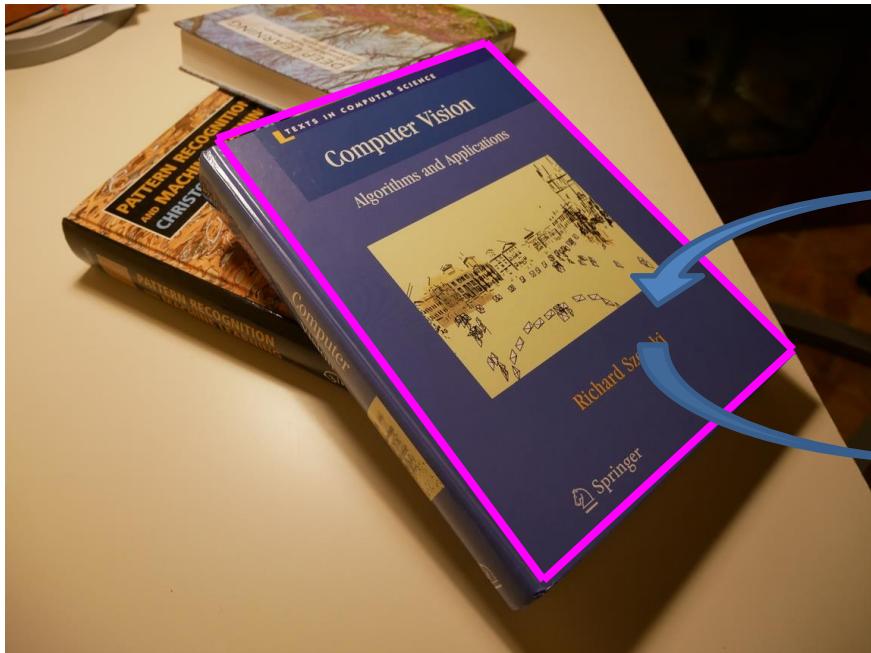


- **Potete invece utilizzare:**

`cv::findHomography(points1, points0, 0)`

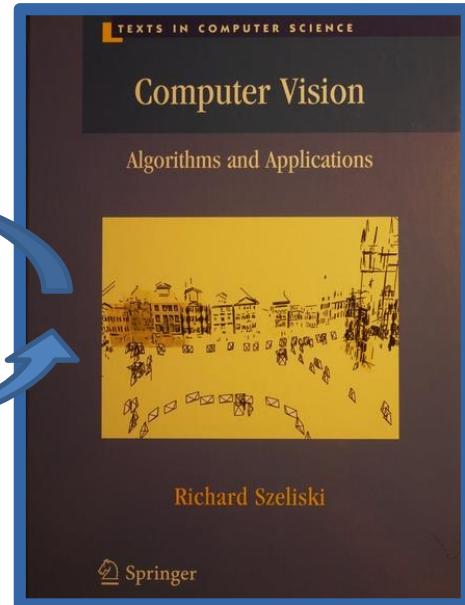


Find Homography



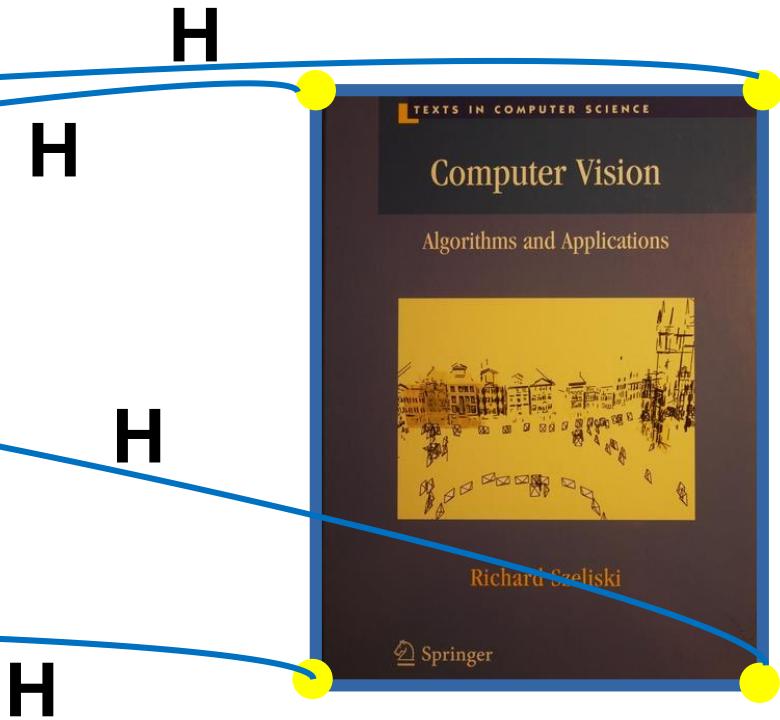
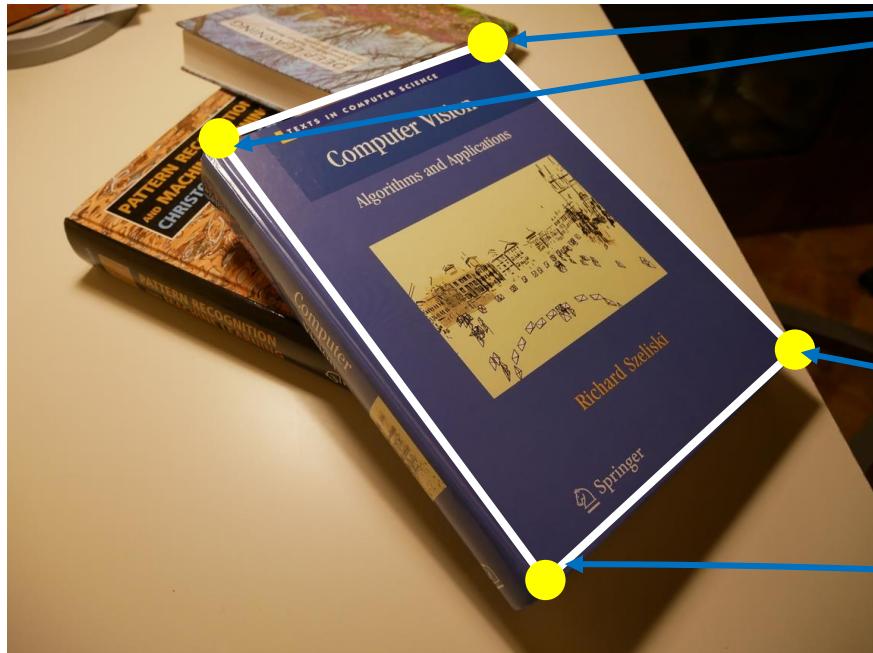
H

H^{-1}



- Due immagini osservano la stessa superficie *planare*, da punti di vista diversi
- Esiste una omografia H tra le due superfici

Find Homography



- Applicazione: data la copertina target a destra, individuarne la posizione della stessa copertina nell'immagine di input a sinistra

Find Homography

1. Calcolare i corner di Harris per entrambe le immagini
2. Calcolare i descrittori dei corner trovati (OpenCV)
3. Associare i descrittori tra le due immagini (OpenCV)
4. Calcolare la migliore trasformazione omografica con RANSAC e SVD
5. Disegnare i bordi della copertina sull'immagine di input (già fornito)

Parte 1

- Scrivere una funzione che calcoli i **corner di Harris**

```
void myHarrisCornerDetector(const cv::Mat image,  
std::vector<cv::KeyPoint> & keypoints, float alpha,  
float harrisTh)
```

- [in] image: immagine di ingresso, singolo canale uint8
- [in] alpha: parametro per il calcolo della response
- [in] harrisTh: minima response per avere un corner
- [out] keypoints: lista dei corner individuati, espressi come (riga,colonna)

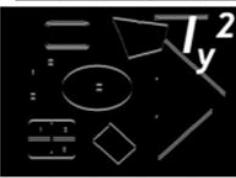
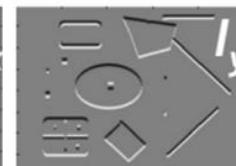
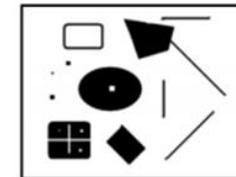
Parte 1

- Compute second moment matrix
(autocorrelation matrix)

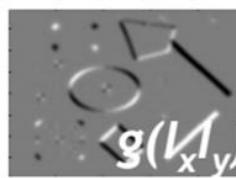
$$M(\sigma_I, \sigma_D) = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

2. Square of derivatives

1. Image derivatives



3. Gaussian filter $g(\sigma)$



4. Cornerness function - two strong eigenvalues

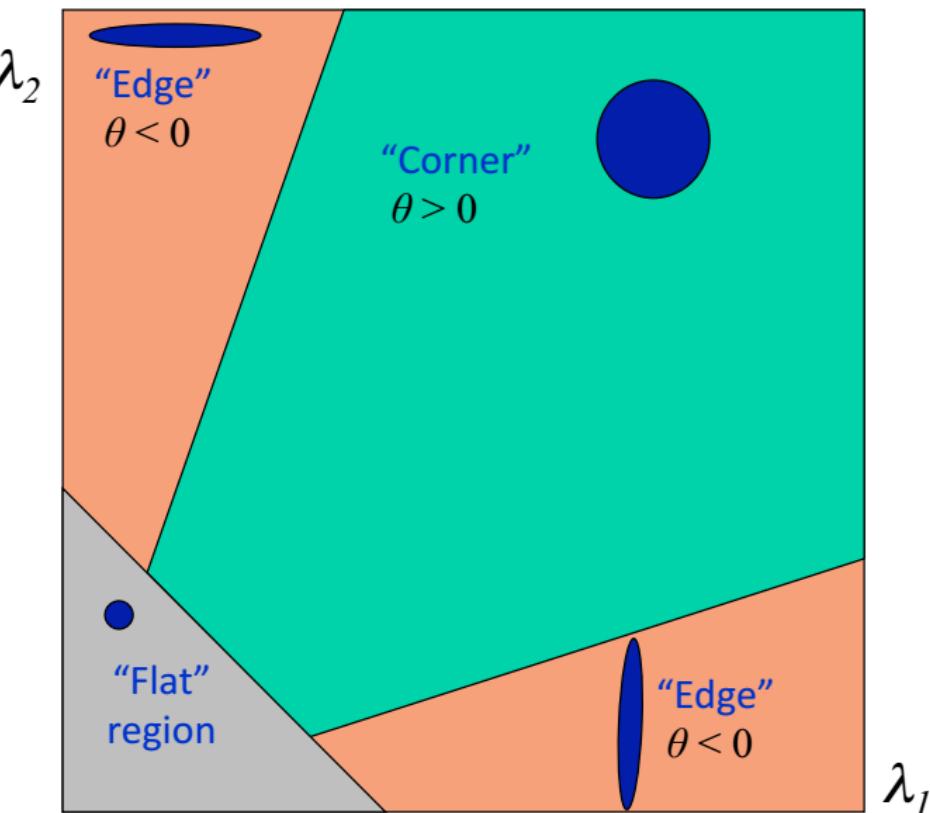
$$\begin{aligned} \theta &= \det[M(\sigma_I, \sigma_D)] - \alpha[\text{trace}(M(\sigma_I, \sigma_D))]^2 \\ &= g(I_x^2)g(I_y^2) - [g(I_x I_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2 \end{aligned}$$

5. Perform non-maximum suppression



Parte 1

$$\theta = \det(M) - \alpha \text{trace}(M)^2 = \lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$



Slide credit: Kristen Grauman

- Fast approximation
 - Avoid computing the eigenvalues
 - α : constant (0.04 to 0.06)

Parte 1

- Non-maximum suppression semplificata:
 1. Eliminare tutti i punti $\theta \leq harrisTh$
 2. Eliminare tutti i punti $\theta > harrisTh$ che **NON** sono massimi locali rispetto al loro vicinato 3x3 o 5x5
 3. Quello che resta sono i keypoint

Parte 1

- Utilizzare le funzioni convoluzione e gradiente viste nell'assegnamento 1 per calcolare le componenti I_x I_y I_{xy} e filtro di smooth g
- Suggerimento: visualizzare *tutti* i passaggi intermedi: I_x I_y $g(I_{xy})$ $g(I^2_x)$ $g(I^2_y)$ θ

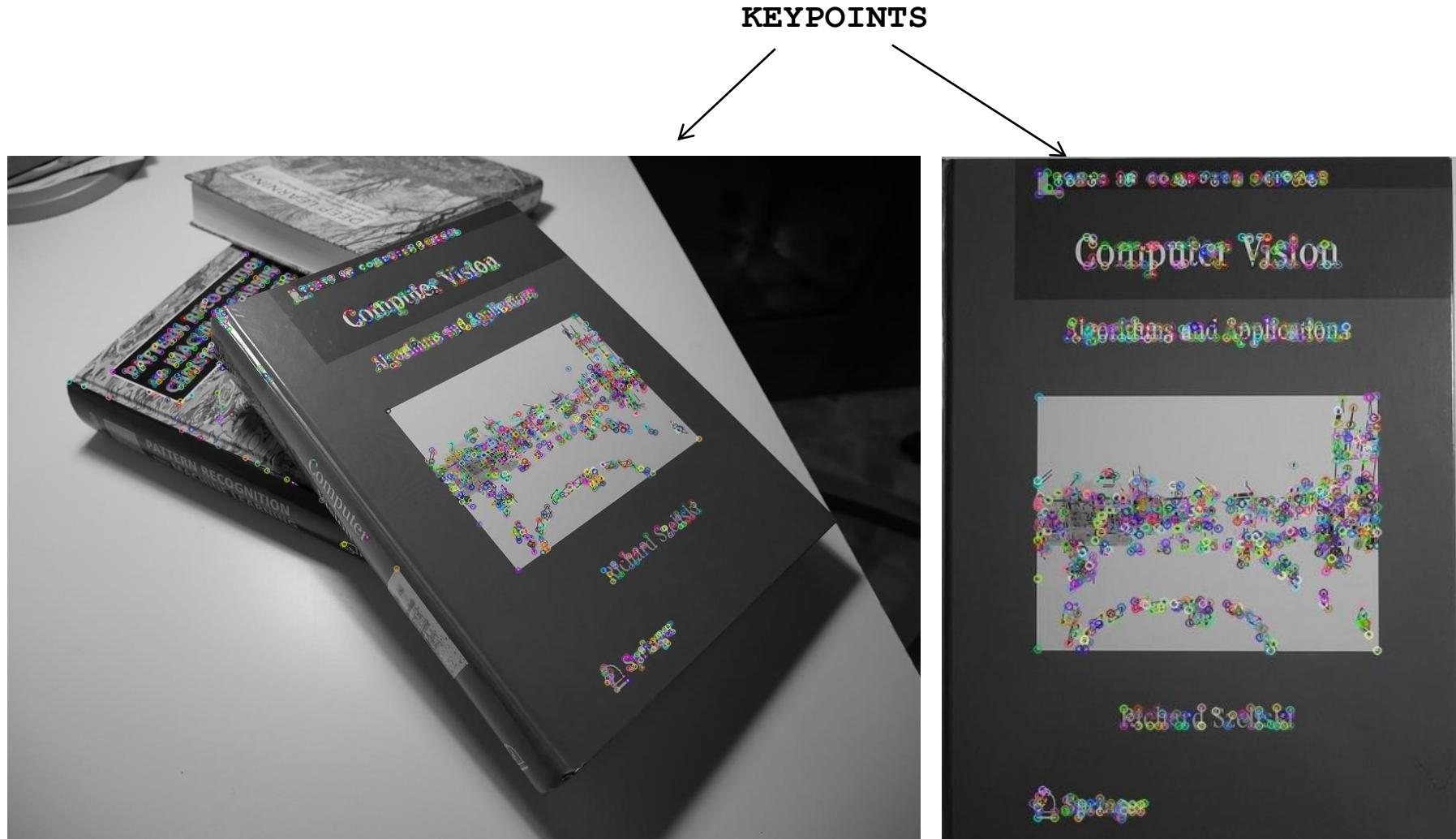
Parte 1

- Per visualizzare la response θ potreste utilizzare:

```
cv::Mat adjMap;
cv::Mat falseColorsMap;
double minr,maxr;

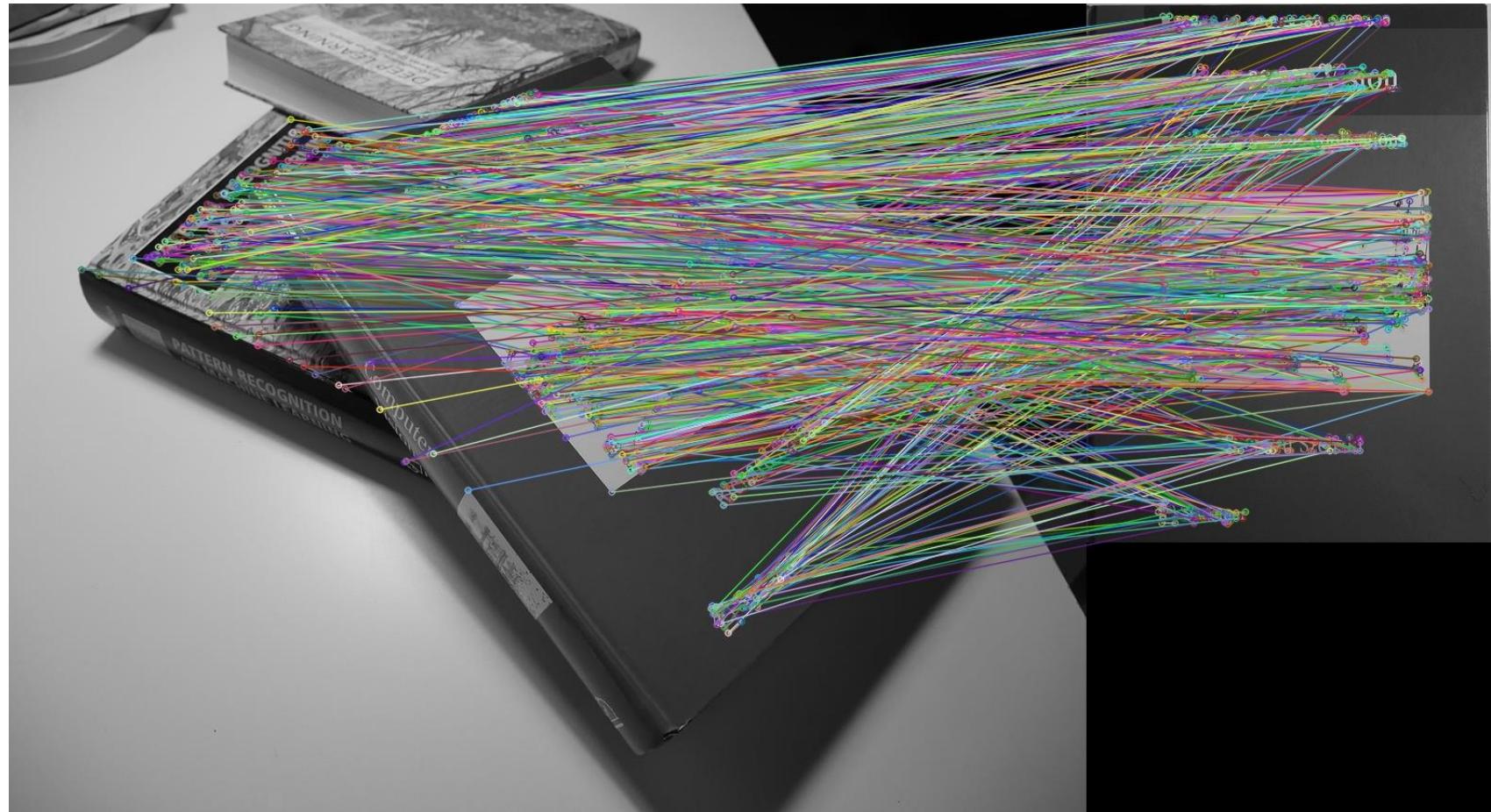
cv::minMaxLoc(response1, &minr, &maxr);
cv::convertScaleAbs(response1, adjMap, 255 / (maxr-minr));
cv::applyColorMap(adjMap, falseColorsMap,
cv::COLORMAP_RAINBOW);
cv::namedWindow("response1", cv::WINDOW_NORMAL);
cv::imshow("response1", falseColorsMap);
```

Parte 1



Parte 2

points0[i] ←→ points1[i]



Parte 2

- Scrivere una funzione che calcoli la migliore omografia possibile a partire dai match ottenuti.
- Mitigare l'effetto dei match errati con RANSAC

```
void findHomographyRansac(const std::vector<cv::Point2f>
& points1, const std::vector<cv::Point2f> & points0, int N,
float epsilon, int sample_size, cv::Mat & H,
std::vector<cv::Point2f> & inliers_best0,
std::vector<cv::Point2f> & inliers_best1)
```

- [in] points1 e point0: lista di corner che sono stati associati tra le due immagini: $\text{points0}[i] \leftrightarrow \text{point1}[i]$
- [in] N: numero di iterazioni di RANSAC
- [in] epsilon: errore massimo di un *inlier*
- [in] sample_size: dimensione dei campioni di RANSAC
- [out] H: omografia
- [out] inliers_best0, inliers_best1: lista dei corner che risultano essere inliers rispetto ad H

RANSAC

1. Selezionare *sample_size=4* match **a caso** tra quelli (p_i^0, p_i^1) in input:

sample0[0] = points0[random0]

sample1[0] = points1[random0]

...

sample0[3] = points0[random3]

sample1[3] = points1[random3]

2. **Inizialmente (!),** calcolare H corrispondente tramite la funzione:

```
H = cv::findHomography(cv::Mat(sample1),  
                      cv::Mat(sample0), 0)
```

RANSAC

3. Contare gli *inliers*: quanti, tra tutti i match (p_i^0, p_i^1) forniti in input, soddisfano la trasformazione H a meno di un piccolo errore

$$\|p_i^0, H p_i^1\| < \varepsilon$$

Utilizzare la norma euclidea righe,colonne tra il punto p_i^0 e $H p_i^1$

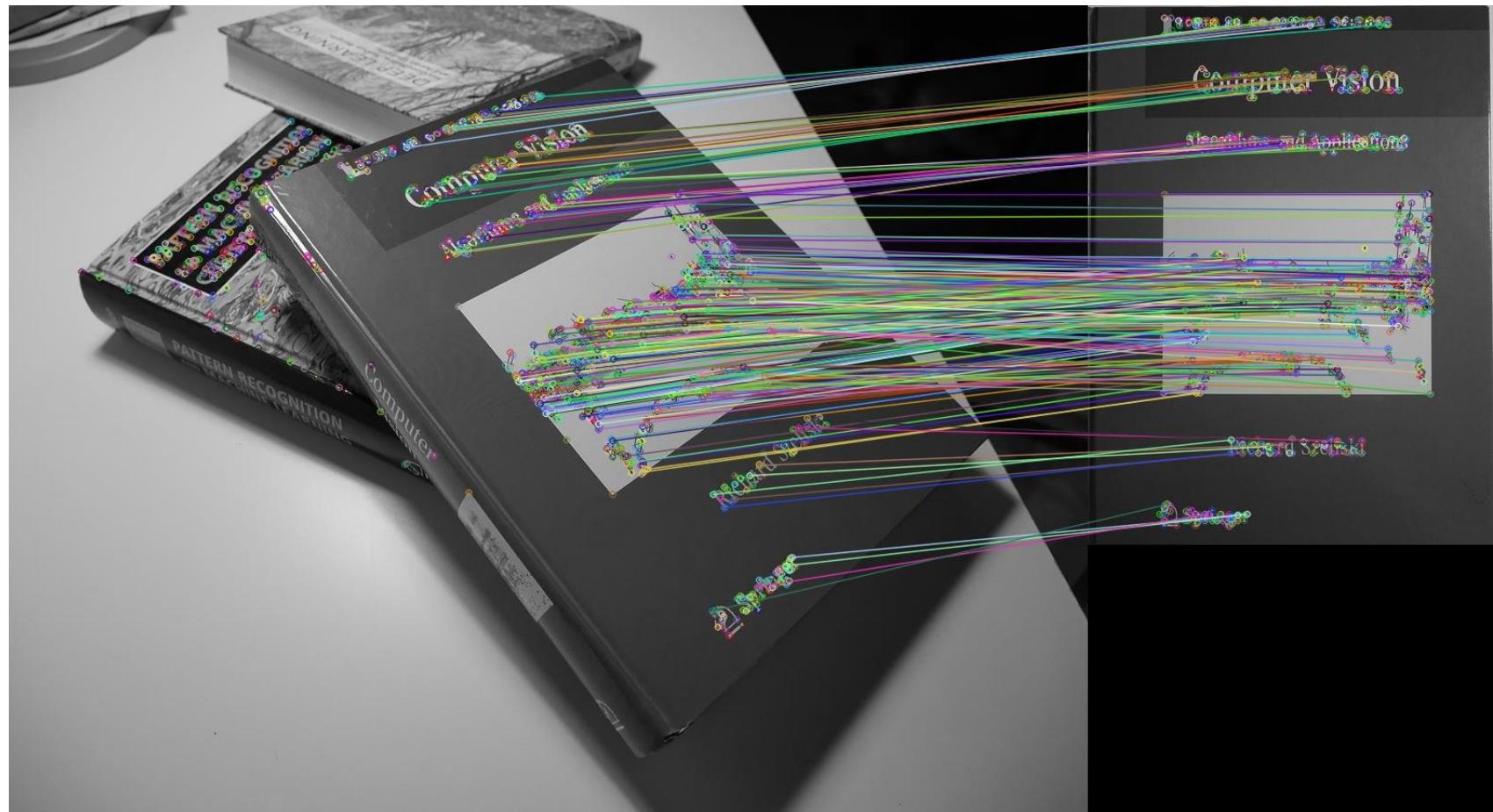
Attenzione ad utilizzare le coordinate omogenee per $H p_i^1$, per poi tornare in eucleede

4. Tornare al punto 1 e ripetere per N volte

- Al termine, ricalcolare H utilizzando i match *del set di inliers più numeroso.*

Parte 2

INLIERS



Parte 2

- Il codice di esempio viene fornito con una chiamata a `findHomography` di OpenCv che fa uso di RANSAC
- In questo modo e' possibile capire immediatamente se i *corner di Harris* sono implementati correttamente senza avere gia' il proprio RANSAC pronto
- Una volta che si e' sicuri della propria implementazione di Harris, commentare la chiamata OpenCv ed utilizzare la propria `myFindHomographyRansac`

Parte 3

- Una volta che il vostro RANSAC funziona, implementate il calcolo della matrice H tramite decomposizione SVD:

```
void myFindHomographySVD(const std::vector<cv::Point2f>
& points1, const std::vector<cv::Point2f> & points0,
cv::Mat & H)
```

- [in] points1 e point0: lista di esattamente **4** corner associati tra le due immagini: $\text{points0}[i] \leftrightarrow \text{point1}[i]$
- [out] H: omografia risultante

Parte 3

- Per calcolare H da un set di n corrispondenze (\bar{x}_1, \bar{x}_2) , costruire la matrice A e risolvere il corrispondente sistema omogeneo:

$$A \cdot h = 0 \quad h = \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix}$$

Parte 3

- Per calcolare H da un set di n corrispondenze (\bar{x}_1, \bar{x}_2) , costruire la matrice A e risolvere il corrispondente sistema omogeneo:

$$A = \begin{bmatrix} -x_1^{(1)} & -y_1^{(1)} & -1 & 0 & 0 & 0 & x_1 x_2'^{(1)} & y_1 x_2'^{(1)} & x_2'^{(1)} \\ 0 & 0 & 0 & -x_1^{(1)} & -y_1^{(1)} & -1 & x_1 y_2'^{(1)} & y_1 y_2'^{(1)} & y_2'^{(1)} \\ \vdots & \vdots \\ -x_1^{(n)} & -y_1^{(n)} & -1 & 0 & 0 & 0 & x_1 x_2'^{(n)} & y_1 x_2'^{(n)} & x_2'^{(n)} \\ 0 & 0 & 0 & -x_1^{(n)} & -y_1^{(n)} & -1 & x_1 y_2'^{(n)} & y_1 y_2'^{(n)} & y_2'^{(n)} \end{bmatrix}$$

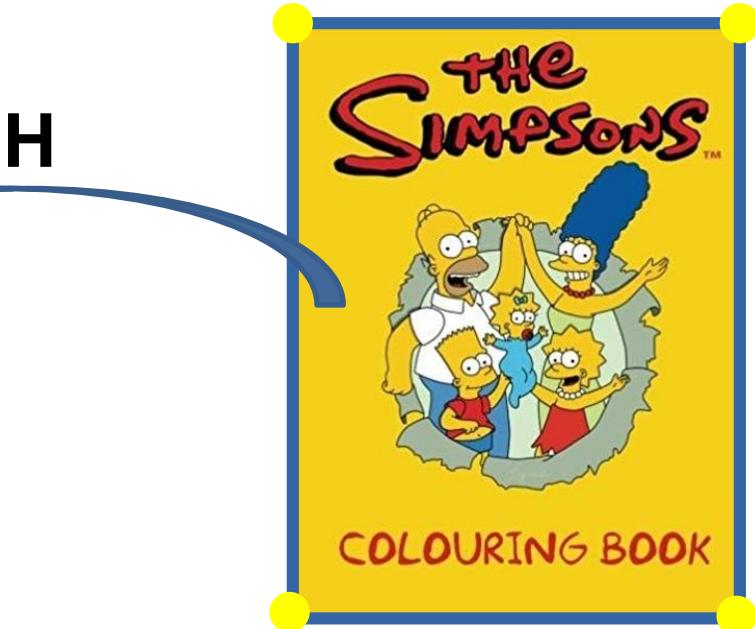
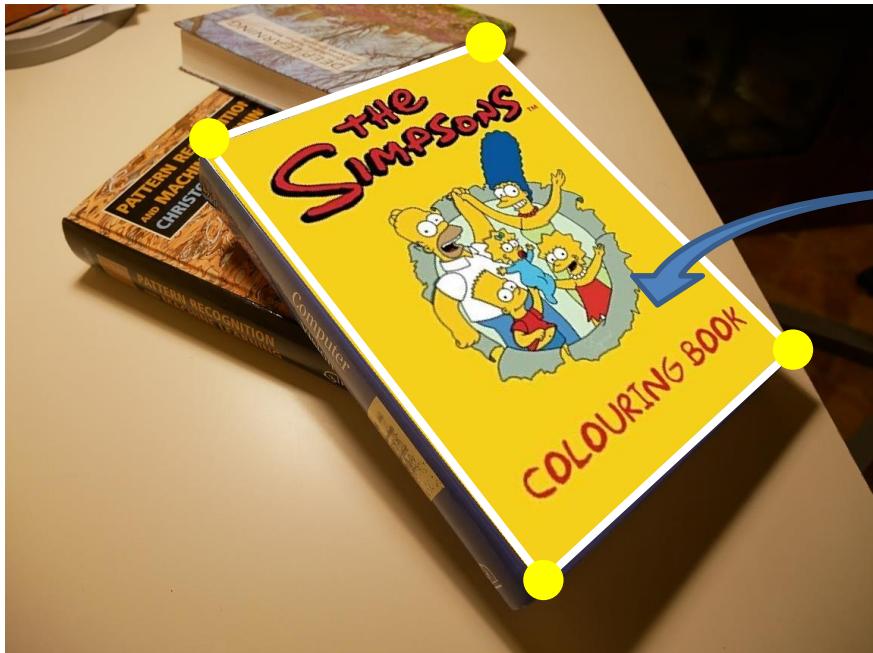
Parte 3

- OpenCv (cosi' come Eigen) fornisce gia' un risolutore SVD:

```
sv::SVD::compute(const Mat & A,  
                  Mat & D, Mat & U, Mat & Vt)
```

- La soluzione va poi normalizzata in modo che l'elemento h_{33} sia pari ad 1

Parte 3



- Usare H per applicare una diversa copertina all'immagine di input

Codice di esempio

- Il file A2_2020.zip contiene le immagini di input e lo scheletro della soluzione in cui dovete inserire il codice
- Esecuzione:

```
./simple ../images/inputX.jpg ../input/book.jpg ../input/cover2.jpg
```

Codice di esempio

Procedere in questo modo:

1. Sostituire i corner di Harris

goodFeaturesToTrack di OpenCV (righe 144-159) con la propria myHarrisCornerDetector e verificare che tutto continui a funzionare

2. Sostituire la findHomography RANSAC di OpenCV (righe 254-257) con il vostro loop RANSAC myFindHomographyRansac, e verificare che tutto continui a funzionare.

Codice di esempio

3. All'interno del vostro loop RANSAC

myFindHomographyRansac **inizialmente** potete **utilizzare** findHomography(sample1,
sample0, 0) per ottenere l'omografia a partire da **un dato set di match**.

4. Quando tutto funziona, sostituire anche questa findHomography **con la vostra** myFindHomographySVD

Codice di esempio

Le soglie sono molto critiche e, nel caso di Harris, dipendenti dalla particolare implementazione.

E' ragionevole ottenere 800-1000 keypoint per immagine.