Assegnamento 2

Corso di Visione Artificiale

A.A. 2018/2019

Regole

- L'assegnamento vale **3** punti se consegnato entro

8 Gennaio 2019

- L'assegnamento vale **1** punto se consegnato dopo

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)
```

Image Stitching

Image Stitching

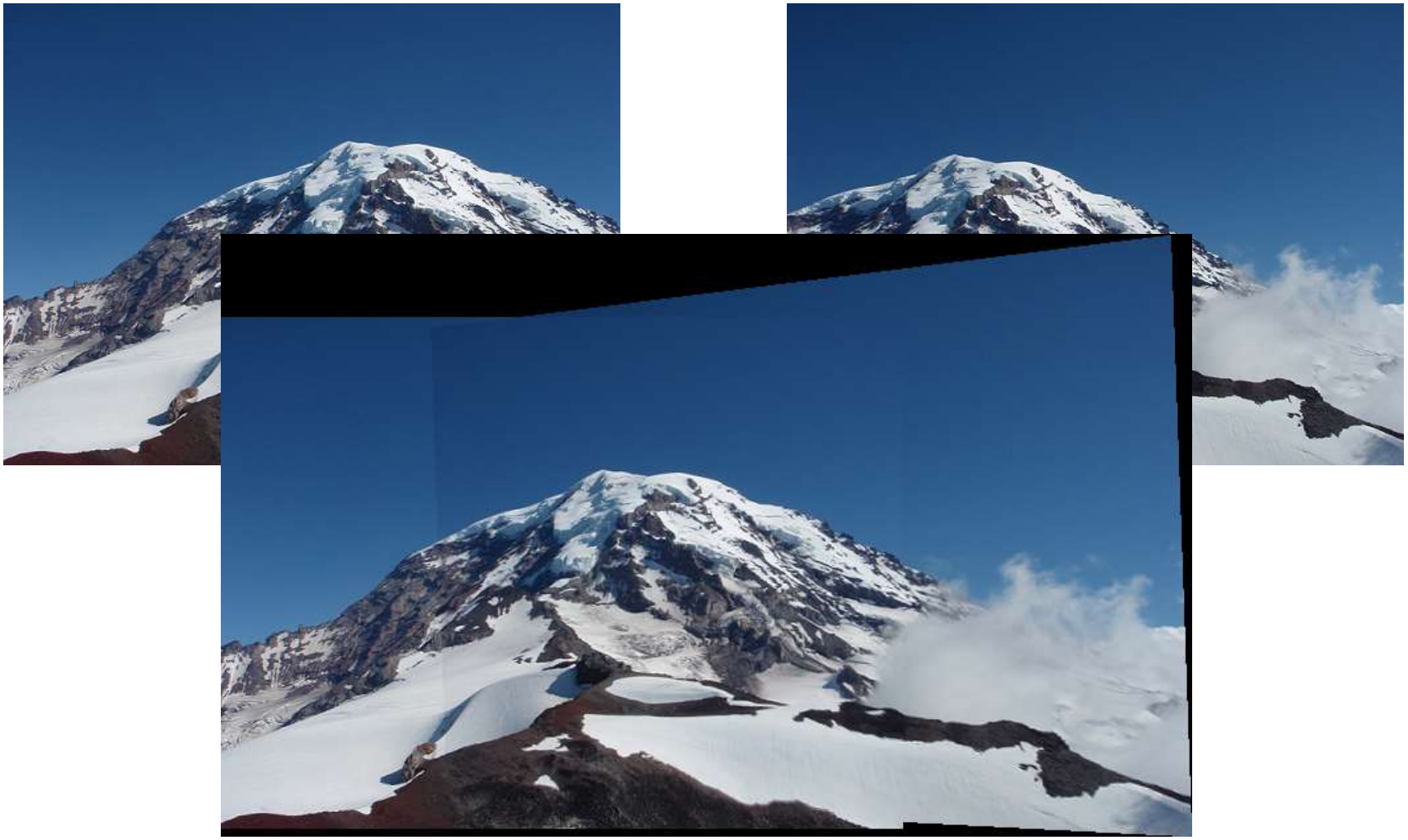


Image Stitching

1. Due immagini della stessa scena, punti di vista diversi
2. Calcolare i corner di Harris per entrambe le immagini
3. Calcolare i descrittori dei corner trovati
4. Associare i descrittori tra le due immagini
5. Calcolare la migliore trasformazione omografica con RANSAC
6. Riproiettare le immagini nel piano destinazione 1 e mescolarle (blend)

Parte 1

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

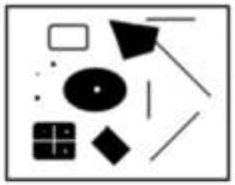
```
void harrisCornerDetector(const cv::Mat
image, std::vector<cv::KeyPoint> &
keypoints, float alpha, float harrisTh)
```

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

Parte 1

- Compute second moment matrix (autocorrelation matrix)

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



1. Image derivatives

2. Square of derivatives

3. Gaussian filter $g(\sigma_I)$

4. Cornerness function - two strong eigenvalues

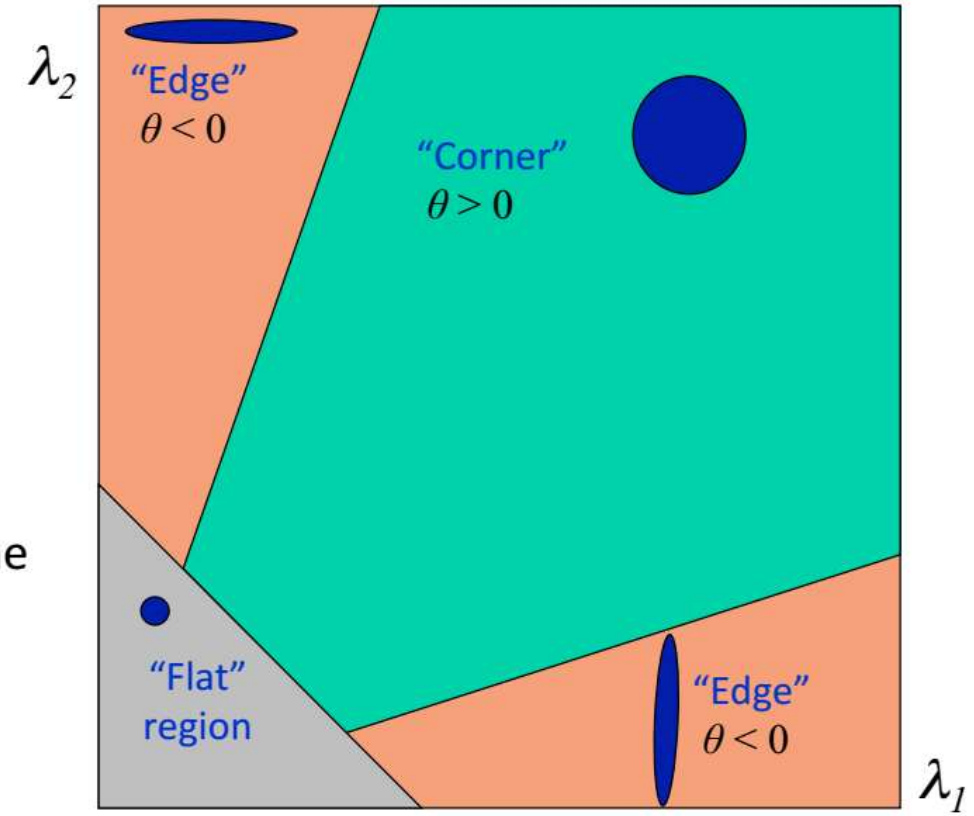
$$\theta = \det[M(\sigma_I,\sigma_D)] - \alpha[\text{trace}(M(\sigma_I,\sigma_D))]^2$$
$$= g(I_x^2)g(I_y^2) - [g(I_x I_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2$$

5. Perform non-maximum suppression

Slide credit: Krystian Mikolajczyk

Parte 1

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



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

Slide credit: Kristen Grauman

Parte 1

- Non-maximu 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 $g(I_{xy})$ $g({I^2}_x)$ $\mathrm{g}({I^2}_y)$ θ

Parte 1

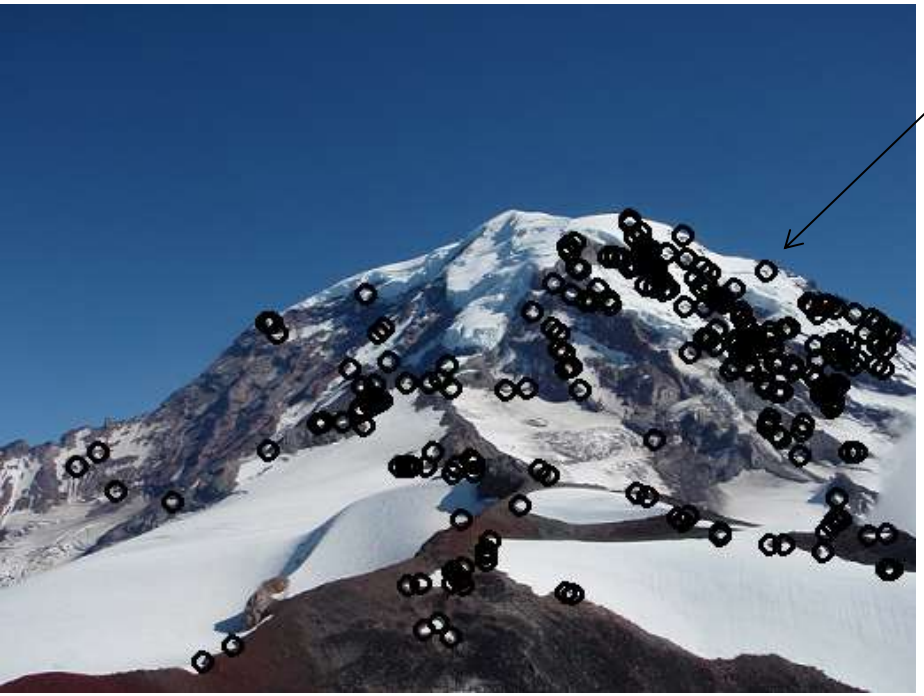
- Per visualizzare la response θ potreste utilizzare:

```
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

keypoints



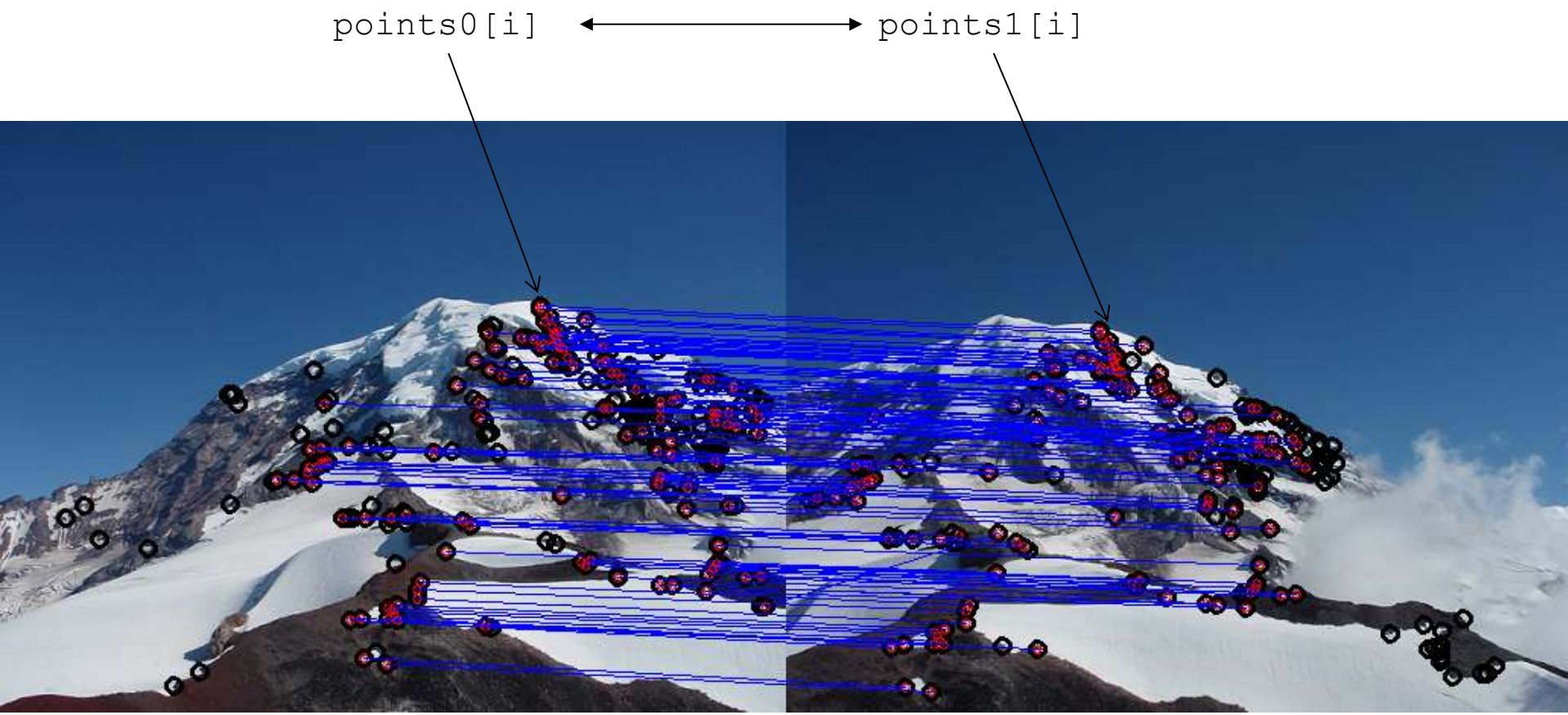
Parte 2

- 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)
```

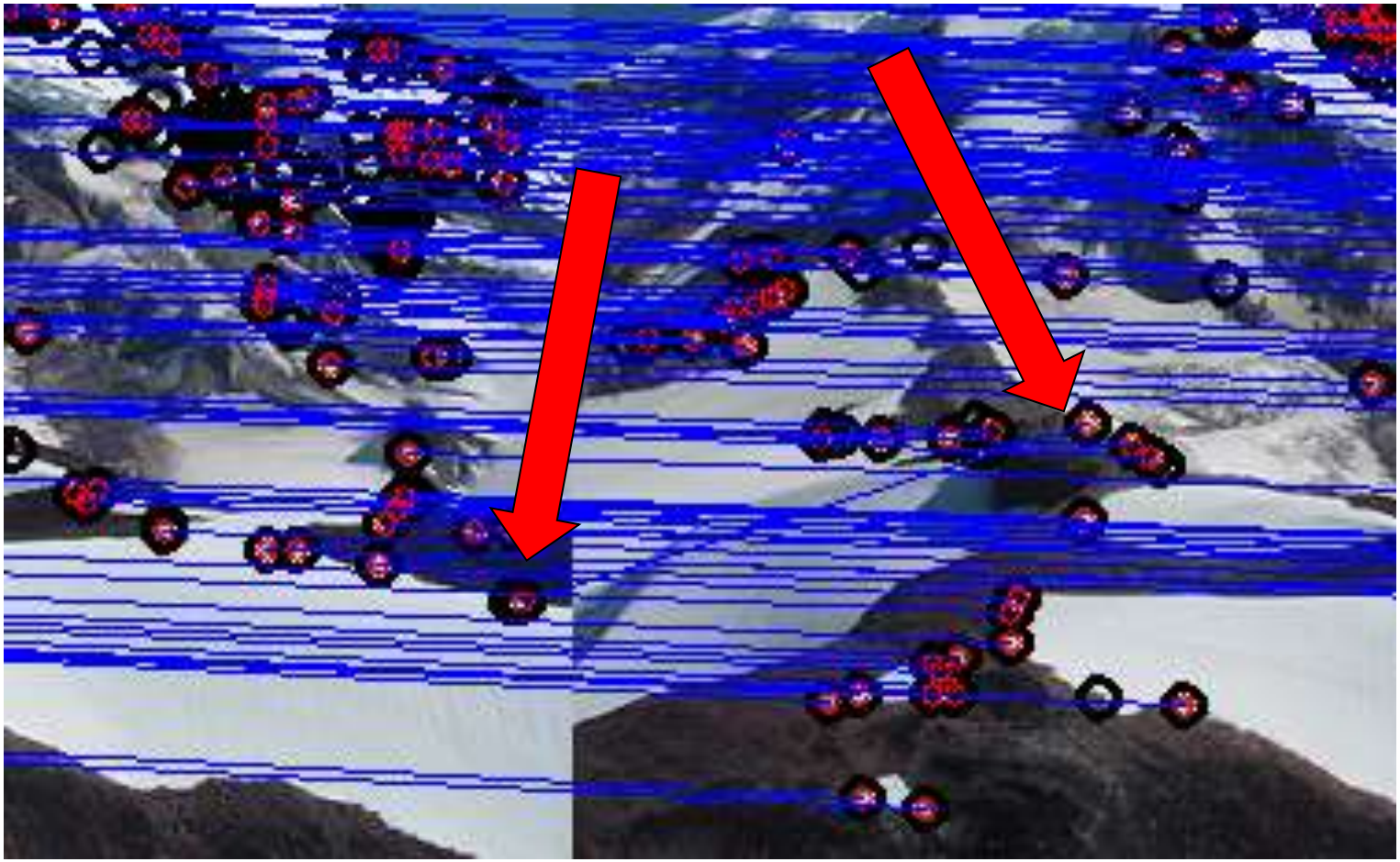
  - [in] points1 e point0: lista di corner che sono stati associati tra le due immagini: points0[i] <-> point1[i]
  - [in] N: numero di iterazioni di RANSAC
  - [in] epsilon: errore massimo di un *inlier*
  - [in] sample_size: dimensione dei campioni di RANSAC
  - [out] H: omografia
  - [out] inliers_best0,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

Parte 2



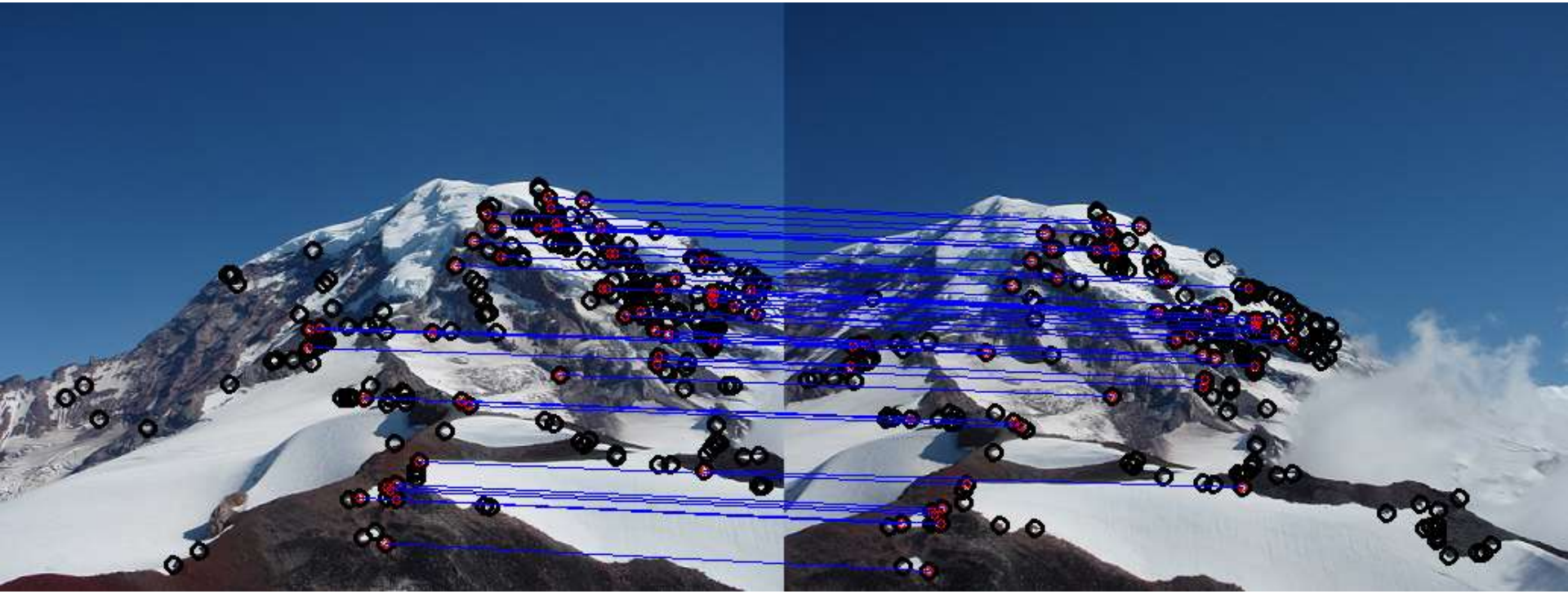
Parte 2

WORNG MATCH!



Parte 2

INLIER MATCHES



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. 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

$$\|p_i{}^0, \boldsymbol{H}\, p_i{}^1\| < \varepsilon$$

Utilizzare la norma euclidea righe,colonne tra il punto $p_i{}^0$ e $\boldsymbol{H}\, p_i{}^1$
Attenzione ad utilizzare le coordinate omogenee per $\boldsymbol{H}\, p_i{}^1$, per poi tornare in euclidee

4. Tornare al punto 1 e ripetere per *N* volte

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

```
H = cv::findHomography( cv::Mat(inliers_best1),
                        cv::Mat(inliers_best0), 0)
```

Parte 2

- Il codice di esempio viene fornito con una chiamata a `findHomography` di OpenCv (riga 239) 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 `findHomographyRansac`

Esercizio 2

Codice di esempio

- Il file A2.zip contiene lo scheletro della soluzione in cui dovete inserire il codice e le immagini da unire

- Esecuzione:

```
./simple -i ../Rainer%d.png
```