Markerless Visual AR: demo Python e OpenCV

Corso di Smart City e Tecnologie Mobili

Università di Bologna

Dipartimento di Informatica — Scienza e Ingegneria

Dario Maio

Obiettivi della lezione e indice degli argomenti

- ✓ Obiettivi:
  - realizzare un dimostrativo della tecnica markerless Visual AR col linguaggio Python (eseguibile in Jupyter notebook) facendo ricorso alla libreria OpenCV.

- ✓ N.B. Il codice è stato ripreso, apportando qualche adattamento, da quello riportato nel tutorial a cura di Juan Gallostra Acín. Anche alcune descrizioni sono frutto di una traduzione di sezioni del suddetto tutorial.

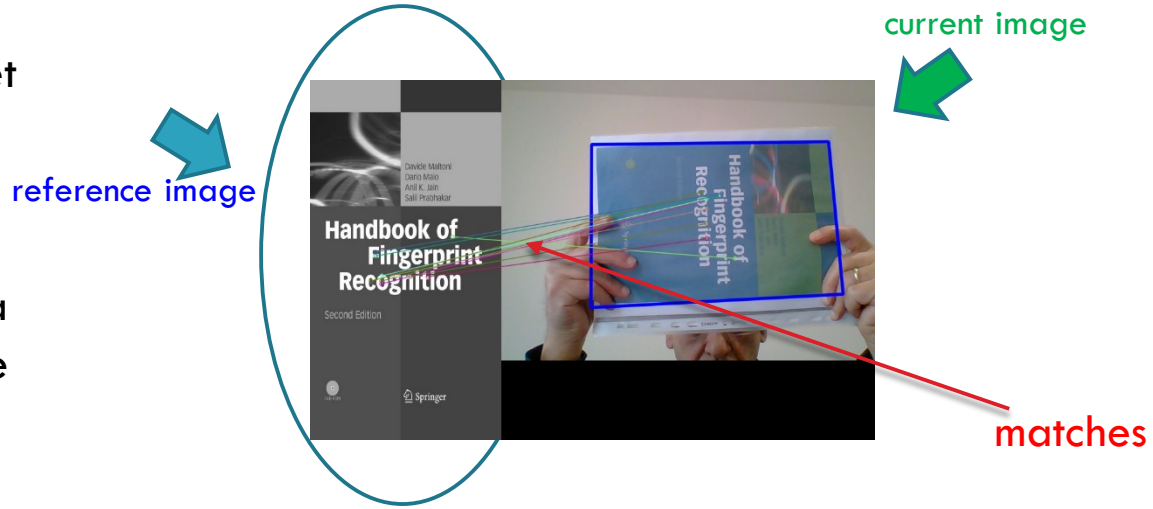
Markerless Visual AR: i passi fondamentali

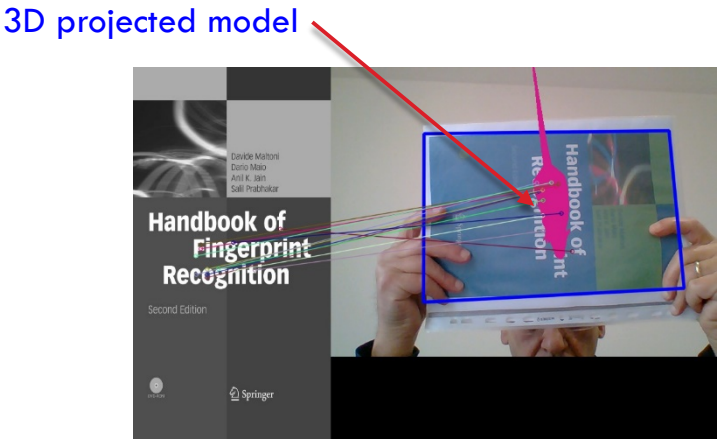
1. Riconoscimento dell'immagine target di riferimento nel frame catturato.

2. Derivazione dell'omografia tra target di riferimento e immagine corrente.

3. Derivazione della matrice di proiezione.

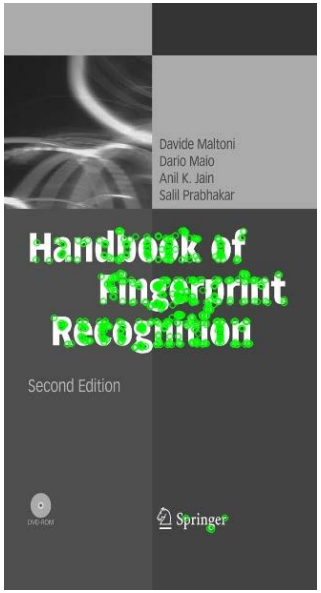
4. Proiezione di un modello 3D sull'immagine corrente.



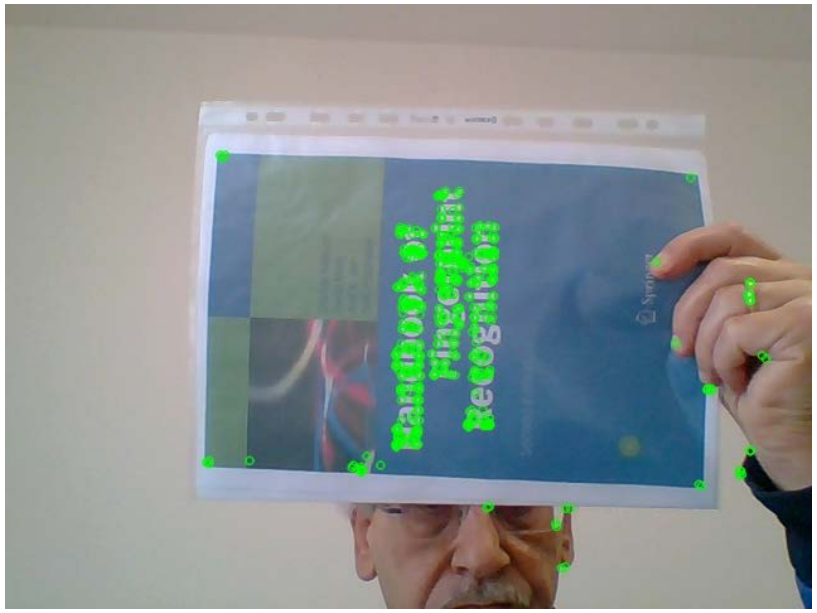


Riconoscimento dell'immagine di riferimento.

- Si utilizza feature detection, feature description and feature matching.
- Per l'estrazione dei keypoints e la costruzione dei relativi descrittori si fa ricorso a ORB (Oriented FAST and Rotated BRIEF).



reference image:
detected keypoints



current image:
detected keypoints

L'estrazione dei keypoints avviene una sola volta

L'estrazione dei keypoints si effettua su ogni frame

OpenCV: ORB detector

- La libreria OpenCV consente di estrarre facilmente le feature da un'immagine e costruire i relativi descrittori.
- Esempio per l'immagine target di riferimento:

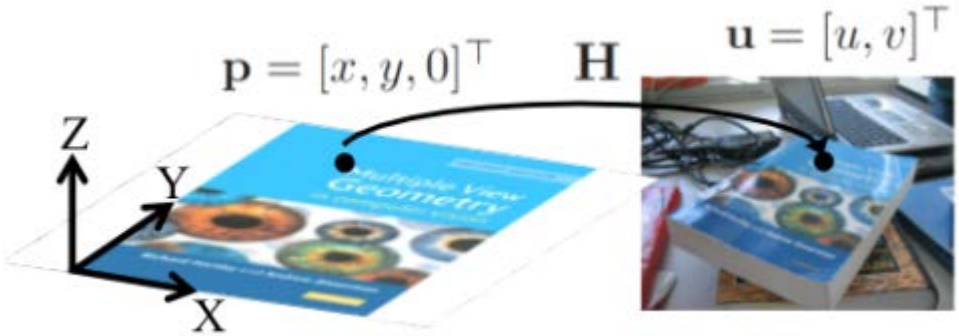
```
# create ORB keypoint detector
orb = cv2.ORB_create()
# load the reference image
dir_name = '../'
reference_image = cv2.imread(os.path.join(dir_name, 'reference/model.jpg'), 0)
# Compute keypoints of the reference image and related descriptors
kp_reference_image, des_reference_image = orb.detectAndCompute(reference_image, None)
# draw only keypoints location, not size and orientation
img1 = cv2.drawKeypoints(reference_image, kp_reference_image, reference_image, color=(0,255,0), flags=0)
# show keypoints over the image
……..
```

Feature matching

- Per il matching si fa ricorso alla distanza di Hamming poiché i descrittori sono stringhe binarie. Esistono metodi più sofisticati.
- Le corrispondenze sono ordinate secondo la distanza di Hamming; solo se sono state trovate un certo numero di corrispondenze MIN_MATCHES valide, allora si considera effettuato il riconoscimento dell'immagine di riferimento nel frame corrente.
- Poiché si deve operare in real time sarebbe più opportuno adottare una tecnica di tracking sfruttando per il frame corrente le informazioni già estratte nel precedente frame, riducendo pertanto la complessità computazionale del riconoscimento.
- Un'altra considerazione: più è facile trovare l'immagine di riferimento, più affidabile sarà il riconoscimento. L'immagine di riferimento che viene utilizzata potrebbe non essere l'opzione migliore, ma aiuta a comprendere il processo.

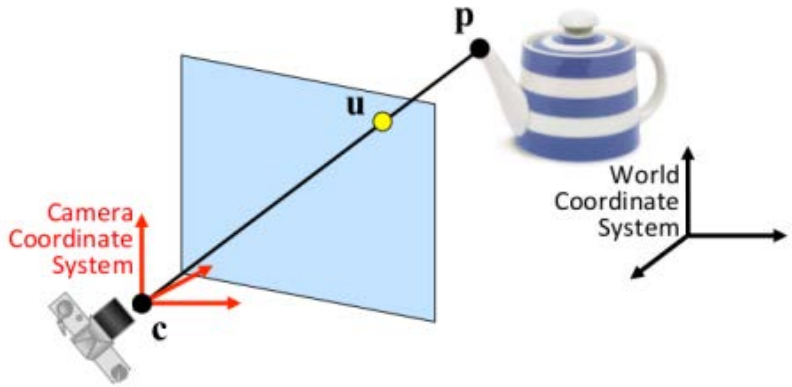
Stima dell'omografia (1)

- Una volta rilevata l'immagine di riferimento nel corrente frame, si può procedere alla stima della trasformazione geometrica $\mathbf{H}$ che mappa punti dell'immagine di riferimento sul piano dell'immagine catturata nel frame.



- Si assume il modello pinhole camera.
- Un punto $\mathbf{p}$ in coordinate mondo 3D è mappato nel punto $\mathbf{u}$ in coordinate 2D del piano immagine.

Si tratta di un'omografia planare: preso un piano e due sue differenti proiezioni 2D, l'omografia lega le coordinate dei punti nei due sistemi di riferimento



Stima dell'omografia (2)

- Nota la matrice $\mathbf{K}$ di calibrazione della camera (detta anche intrinsic camera matrix), assumendo il modello pinhole, il punto $\mathbf{u}$ in coordinate 2D del piano immagine, espresse tramite le coordinate camera, è così derivabile:

$$\begin{bmatrix} u \cdot k \\ v \cdot k \\ k \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^{\text{cam}} \\ y^{\text{cam}} \\ z^{\text{cam}} \end{bmatrix}$$

$\mathbf{K}$ Calibration matrix

(u_0, v_0) Projection of the optical center

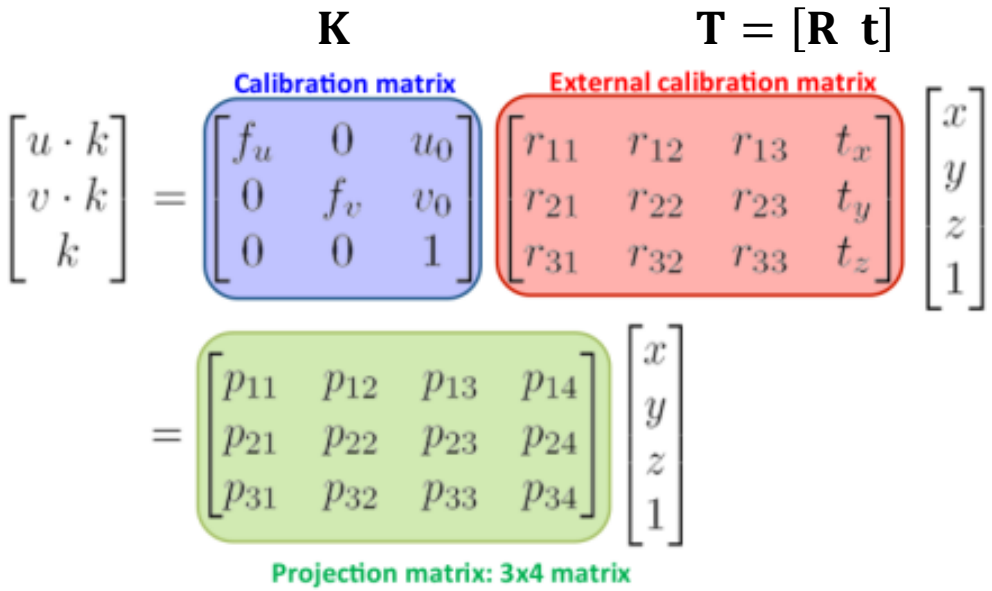
f_u, f_v Focal lengths

k fattore di scala

- Questa relazione ci fa comprendere come si forma l'immagine. Tuttavia, si conoscono solo le coordinate del punto $\mathbf{p}$ nel sistema di coordinate mondo e non nel sistema di coordinate camera, pertanto si deve aggiungere un'altra trasformazione che mappa da coordinate mondo al sistema di coordinate camera.
- La trasformazione suddetta è rappresentata dalla matrice di rototraslazione $\mathbf{T}$, detta extrinsic camera matrix o external calibration matrix, che consente di derivare la posa della camera.

Stima dell'omografia (3)

- Dunque la relazione che lega un punto in coordinate mondo a un punto proiettato in 2D in coordinate immagine è:



- Tenendo conto che nel caso in esame l'immagine di riferimento è planare, e dunque $z = 0$, si può operare una semplificazione della relazione sopra riportata.

Stima dell'omografia (4)

- La relazione semplificata è:

vettori colonna di $\mathbf{R}$

$$\begin{bmatrix} ku \\ kv \\ k \end{bmatrix} = \mathbf{K}[\mathbf{R}\ \ \mathbf{t}] \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \mathbf{K}[\mathbf{R_1}\ \mathbf{R_2}\ \mathbf{R_3}\ \ \mathbf{t}] \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \mathbf{K}[\mathbf{R_1}\ \mathbf{R_2}\ \mathbf{t}] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} ku \\ kv \\ k \end{bmatrix} = \overbrace{\begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ h_{3,1} & h_{3,2} & h_{3,3} \end{bmatrix}}^{\mathbf{H}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Si fa ricorso, in questo esempio, al metodo RANSAC per la stima dei coefficienti della matrice omografica **H**.

OpenCV: Matching e stima dell'omografia

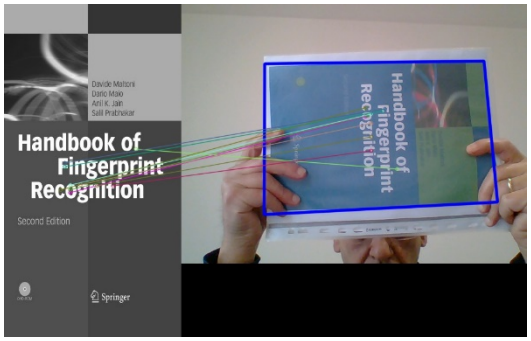
N.B. Si assume di avere già inizializzato un BFMatcher basato sulla distanza di Hamming
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

```
# find the keypoints of the frame
    kp_frame, des_frame = orb.detectAndCompute(frame, None)
    # match frame descriptors with reference image descriptors
    matches = bf.match(des_reference_image, des_frame)
    # sort them in the order of their distance,  the lower the distance, the better the match
    matches = sorted(matches, key=lambda x: x.distance)
    # compute Homography if enough matches are found
    if len(matches) > MIN_MATCHES:
        # differentiate between source points and destination points
        src_pts = np.float32([kp_reference_image[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
        dst_pts = np.float32([kp_frame[m.trainIdx].pt for m in matches]).reshape(-1, 1, 2)
        # compute Homography
        homography, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
     else: print ("Not enough matches found - %d/%d" % (len(matches), MIN_MATCHES))
```

Disegno di una cornice sull'immagine

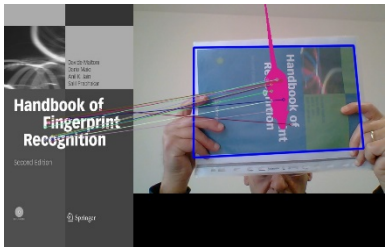
- Se si vuole disegnare un rettangolo che incornicia l'immagine target rilevata nel frame corrente, una volta stimata l'omografia, è sufficiente procedere al calcolo delle coordinate dei quattro corner sia per l'immagine di riferimento sia per quella corrispondente rilevata nel frame.

```
# Draw a rectangle that marks the found model in the captured frame
h, w = reference_image.shape
pts = np.float32([[0, 0], [0, h - 1], [w - 1, h - 1], [w - 1, 0]]).reshape(-1, 1, 2)
# project corners into frame
dst = cv2.perspectiveTransform(pts, homography)
# connect them with lines
frame = cv2.polylines(frame, [np.int32(dst)], True, 255, 3, cv2.LINE_AA)
# show result
cv2.imshow('frame', frame)
```



Rendering di un modello 3D

- Al fine di effettuare il rendering di un modello 3D sul piano dell'immagine di riferimento rilevata nel frame corrente, non è sufficiente la matrice $\mathbf{H}$ prima derivata. Ciò perché la trasformazione geometrica ottenuta non ha preso in considerazione la coordinata z.
- In sintesi, $\mathbf{H} = \mathbf{K}[\mathbf{R_1}\,\mathbf{R_2}\,\mathbf{t}]$ è sufficiente solo per proiettare sul piano dell'immagine di riferimento rilevata nel frame oggetti virtuali come punti, linee, poligoni che si immaginano giacere sul piano dell'immagine di riferimento nel mondo reale.
- Si deve derivare, per ogni frame, la matrice $\mathbf{P}$ di proiezione 3D-2D in grado di proiettare ogni punto 3D del modello (appoggiato virtualmente sull'immagine di riferimento nel mondo reale) in un corrispondente punto 2D sul piano dell'immagine di riferimento catturata nel frame corrente.
- Ciò è possibile conoscendo la matrice di calibrazione $\mathbf{K}$ e la matrice omografica $\mathbf{H}$.



Stima della matrice di proiezione (2)

- Premoltiplicando $\boldsymbol{H}$ per l'inversa della matrice di calibrazione interna $\boldsymbol{K}$, si ottiene:

$$\boldsymbol{G = K^{-1}H = K^{-1}K[R_1\, R_2\, t] = [R_1\, R_2\, t] = [G_1\, G_2\, G_3]}$$

- Poiché la matrice di calibrazione esterna $[\mathbf{R}'_1\, \mathbf{R}'_2\, \mathbf{R}'_3\ \ \mathbf{t}]$è una trasformazione omogenea che mappa punti di due differenti sistemi di riferimento, allora $[\boldsymbol{R'_1\, R'_2\, R'_3}]$ è una base ortonormale, e pertanto si ha:

$$\boldsymbol{R'_3 = R'_1 \times R'_2}$$

avendo indicato con × l'operatore prodotto vettoriale.

- Avendo ottenuto $\boldsymbol{G_1 = R_1, G_2 = R_2}$, $\boldsymbol{G_3 = t}$ dalla stima di $\boldsymbol{K}$ e $\boldsymbol{H}$, non è garantito che $[\boldsymbol{R_1 = G_1\ \ R_2 = G_2\ R_3 = G_1 \times G_2}]$ sia una base ortonormale; infatti $\boldsymbol{G_1}$ e $\boldsymbol{G_2}$ rappresentano solo delle stime dei veri vettori $\boldsymbol{R'_1, R'_2}$. $\boldsymbol{G}$ è definita a meno di un fattore di scala, $\boldsymbol{R_1}$ e $\boldsymbol{R_2}$ non sono perfettamente ortonormali.

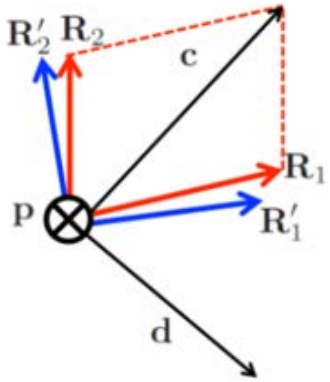
Derivazione della matrice di proiezione (3)

- Vi sono diversi modi per stimare $\boldsymbol{R'_1, R'_2}$ e conseguentemente ricavare $\boldsymbol{R'_3}$ come il prodotto vettoriale di $\boldsymbol{R'_1}$ e $\boldsymbol{R'_2}$: $\boldsymbol{R'_3 = R'_1 \times R'_2}$.
- Si può procedere, ad esempio, tenendo conto che l'angolo fra i vettori stimati $\boldsymbol{G_1, G_2}$ è circa $\frac{\pi}{2}$. Inoltre il modulo di ognuno di questi vettori è prossimo a 1. Si può costruire una base ortonormale a partire da $\boldsymbol{G_1, G_2}$ come illustrato dall'algoritmo in figura (dispense Prof. Vincent Lepetit).

$$l = \sqrt{\|\boldsymbol{G_1}\|\|\boldsymbol{G_2}\|} \quad \boldsymbol{R_1} = \frac{\boldsymbol{G_1}}{l} \quad \boldsymbol{R_2} = \frac{\boldsymbol{G_2}}{l} \quad \boxed{\boldsymbol{t} = \frac{\boldsymbol{G_3}}{l}}$$

$$\boldsymbol{c = R_1 + R_2} \quad \boldsymbol{p = R_1 \times R_2} \quad \boldsymbol{d = c \times p}$$

$$\boldsymbol{R'_1} = \frac{1}{\sqrt{2}}\left(\frac{\boldsymbol{c}}{\|\boldsymbol{c}\|} + \frac{\boldsymbol{d}}{\|\boldsymbol{d}\|}\right) \quad \boldsymbol{R'_2} = \frac{1}{\sqrt{2}}\left(\frac{\boldsymbol{c}}{\|\boldsymbol{c}\|} - \frac{\boldsymbol{d}}{\|\boldsymbol{d}\|}\right) \quad \boldsymbol{R'_3 = R'_1 \times R'_2}$$



$$\boxed{\boldsymbol{R} = [\boldsymbol{R'_1}\ \boldsymbol{R'_2}\ \boldsymbol{R'_3}\ ]}$$

$$\boxed{\boldsymbol{M} = \boldsymbol{K}[\boldsymbol{R'_1}\ \boldsymbol{R'_2}\ \boldsymbol{R'_3}\ \boldsymbol{t}]}$$

- Si applica di norma un raffinamento tramite ottimizzazione non lineare minimizzando l'errore di riproiezione (ciò non è stato nel codice Python sviluppato). Si ricorda che $\boldsymbol{M}$ deve essere calcolata a ogni passo di cattura di un frame.

Matrice di proiezione: codice Python

```python
def projection_matrix(camera_parameters, homography):
    """
     From the camera calibration matrix and the estimated homography
     compute the 3D projection matrix
     """
    # Compute rotation along the x and y axis as well as the translation
    homography = homography * (-1)
    rot_and_transl = np.dot(np.linalg.inv(camera_parameters), homography)
    col_1 = rot_and_transl[:, 0]
    col_2 = rot_and_transl[:, 1]
    col_3 = rot_and_transl[:, 2]
    # normalise vectors
    l = math.sqrt(np.linalg.norm(col_1, 2) * np.linalg.norm(col_2, 2))
    rot_1 = col_1 / l
    rot_2 = col_2 / l
    translation = col_3 / l
    # compute the orthonormal basis
    c = rot_1 + rot_2
    p = np.cross(rot_1, rot_2)
    d = np.cross(c, p)
    rot_1 = np.dot(c / np.linalg.norm(c, 2) + d / np.linalg.norm(d, 2), 1 / math.sqrt(2))
    rot_2 = np.dot(c / np.linalg.norm(c, 2) - d / np.linalg.norm(d, 2), 1 / math.sqrt(2))
    rot_3 = np.cross(rot_1, rot_2)
    # finally, compute the 3D projection matrix from the model to the current frame
    projection = np.stack((rot_1, rot_2, rot_3, translation)).T
    return np.dot(camera_parameters, projection)
```

N.B. La matrice $\mathbf{K}$ può essere stimata euristicamente come effettuato in questo esempio. In generale si procede applicando un metodo di calibrazione.

Rendering di un modello 3D

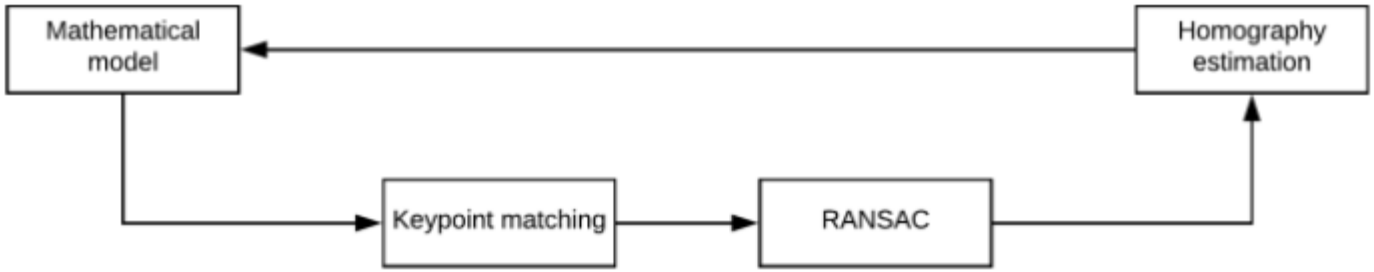
```python
def render(img, obj, projection, model, color=False):
    vertices = obj.vertices
    scale_matrix = np.eye(3) * 3
    h, w = model.shape

    for face in obj.faces:
        face_vertices = face[0]
        points = np.array([vertices[vertex - 1] for vertex in face_vertices])
        points = np.dot(points, scale_matrix)
        # render model in the middle of the reference surface. To do so,
        # model points must be displaced
        points = np.array([[p[0] + w / 2, p[1] + h / 2, p[2]] for p in points])
        dst = cv2.perspectiveTransform(points.reshape(-1, 1, 3), projection)
        imgpts = np.int32(dst)
        if color is False:
            cv2.fillConvexPoly(img, imgpts, (137, 27, 211))
        else:
            color = hex_to_rgb(face[-1])
            color = color[::-1] # reverse
            cv2.fillConvexPoly(img, imgpts, color)

    return img
```

N.B. Si suppone di aver caricato un modello 3D (denominato obj) tramite un opportuno loader. Si deve tener conto della scala e del sistema di coordinate del modello stesso. Si rimanda al tutorial per ulteriori dettagli.

Sommario

- Nella realtà aumentata markerless un oggetto è rilevato grazie alle feature che possiede.
- L'intero processo della derivazione della matrice di proiezione (per ogni frame acquisito) può essere riassunto nella seguente figura.

$$\mathbf{P} = \mathbf{K}[\mathbf{R}'_1\ \mathbf{R}'_2\ \mathbf{R}'_3\ \mathbf{t}] \qquad \mathbf{H} = \mathbf{K}[\mathbf{R}_1\ \mathbf{R}_2\ \mathbf{t}]$$



- La matrice $\mathbf{P}$ consente di proiettare sul piano immagine un modello 3D.