UNIVERSITÀ DEGLI STUDI
DI MILANO

Elaborazione delle Immagini

Raffaella Lanzarotti

FLUSSO OTTICO

Computer Vision System Toolbox

- Altro toolbox di MATLAB che implementa strumenti delle Computer Vision
- Adotta la programmazione ad oggetti
- Es. di strumenti disponibili:
  - feature detection, description, and matching
  - object detection and matching
  - motion estimation
  - video processing

Es: feature detection

- Harris Corner:

```
points = detectHarrisFeatures(I)
```

- SURF: Versione efficiente dei SIFT:

```
points = detectSURFFeatures(I)
```

Es: Feature descriptors

- HOG:

```
features = extractHOGFeatures(I)
```

- LBP:

```
features = extractLBPFeatures(I)
```

- SURF:

```
features = extractFeatures(I, 'Method','SURF')
```

Es: find matching

```
points1 = detectHarrisFeatures(I1);
points2 = detectHarrisFeatures(I2);

[feats1, valid1] = extractFeatures(I1, points1);
[feats2, valid2] = extractFeatures(I2, points2);

% MATCHING:
indexPairs = matchFeatures(feats1, feats2);

 matched1 = valid1(indexPairs(:, 1), :);
 matched2 = valid2(indexPairs(:, 2), :);

figure;
showMatchedFeatures(I1, I2, matched1, matched2);
```

FLUSSO OTTICO

Primo step: aprire file video

```
OBJ = VideoReader(FILENANE);
```

- Costruisce un oggetto per la lettura di multimedia
- Metodi associati:
  - `readFrame` - Legge il prossimo frame disponibile
  - `hasFrame` - Determina se ci sono ancora frame disponibili per la lettura

Proprietà dell'oggetto

- `Name` - Nome del file da leggere.
- `Path` - Percorso del file
- `Duration` - Lunghezza del file in secondi
- `CurrentTime` - Posizione del prossimo frame da leggere (espressi in secondi)
- `Height` - Altezza dei frame (in pixel)
- `Width` - Larghezza dei frame (in pixel)
- `BitsPerPixel` - Bits per pixel del video
- `VideoFormat` - Video format come è rappresentato in MATLAB
- `FrameRate` - Frame rate del video (in frame al secondo)

Flusso Ottico (HS)

`obj = opticalFlowHS`

- restituisce un oggetto per la stima del flusso ottico con il metodo proposto da Horn-Schunck.
- il flusso ottico è espresso in termini di direzione e velocità dal frame precedente a quello corrente
- metodi:
  - `estimateFlow` - stima del flusso ottico
  - `reset` - Reset dello stato interno dell'oggetto
- Proprietà:
  - `'Smoothness'` : scalare che definisce il peso da dare al termine di smoothness. Default: 1
  - `'MaxIteration'`. Default: 10
  - `'VelocityDifference':` soglia da usare come criterio di terminazione (scalare non negativo). Default: 0

Flusso Ottico (LK)

```
obj = opticalFlowLK
```

- restituisce un oggetto per la stima del flusso ottico con il metodo proposto da Lucas Kanade.
- il flusso ottico è espresso in termini di direzione e velocità dal frame precedente a quello corrente
- metodi:
  - `estimateFlow` - stima del flusso ottico
  - `reset` - Reset dello stato interno dell'oggetto
- Proprietà:
  - `'NoiseThreshold'` : soglia per la riduzione del rumore. più alta è meno il calcolo è sensibile a piccoli movimenti Default: 0.0039

Point Tracking, KLT

- Dati dei punti su un primo frame, vogliamo “inseguirli”.
- Problema: quali punti?
  - potremmo scegliere dei buoni punti secondo varie euristiche (Harris, SIFT, …) ma non garantirebbero una buona tracciabilità.

- IDEA di Kanade-Lucas-Tomasi:

una feature è buona se si può tracciare bene.

Ricordiamo Lucas Kanade

$$\begin{bmatrix} \sum f_{xi}^2 & \sum f_{xi} f_{yi} \\ \sum f_{xi} f_{yi} & \sum f_{yi}^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum f_{xi} f_{ti} \\ -\sum f_{xi} f_{ti} \end{bmatrix}$$

$A^T A$

- Qual è la condizione di buon condizionamento?
  - A^TA deve essere <u>invertibile</u>
  - A^TA non troppo piccola
    - autovalori (λ_1, λ_2) non troppo piccoli
  - A^TA ben condizionata

    λ_1/λ_2 non troppo grande, dove $\lambda_1 > \lambda_2$

Point Tracking, KLT, cont.

- un punto è buono se

$$\min(\lambda_1, \lambda_2) > \lambda$$

- Si sposta il problema: quale valore di soglia scegliere?

- in linea teorica, disponendo della camera di acquisizione, si misurano gli autovalori su una regione uniforme (coglie il rumore della camera) —> lower bound λ_{low}
- misura gli autovalori in regioni con alta tessitura—> upper bound λ_{high}
- fissa la soglia a metà strada

$$\lambda = 0.5 * \lambda_{low} + 0.5 * \lambda_{high}$$

Point Tracking, MATLAB

```
H = vision.PointTracker
```

- Crea un oggetto, H, che traccia un insieme di punti usando l'algoritmo Kanade-Lucas-Tomasi feature tracking

- Richiede un'inizializzazione del processo, per specificare la posizione iniziale dei punti e il frame iniziale:

```
initialize(H, POINTS, I)
```

Point Tracking, cont.

- Il tracking frame per frame si ottiene con il metodo step

```
[POINTS,POINT_VALID,SCORES] = step(H, I)
```

- `POINTS`: array MX2 di coord [x y] corrispondenti alle nuove coordinate
- `POINT_VALID`: array Mx1, di valori booleani indicanti se il punto trovato è affidabile o meno
- `SCORES`: array Mx1, di valori reali tra 0 e 1, calcolato come LSE tra i blocchi intorno al punto tracciato, nei frame in esame

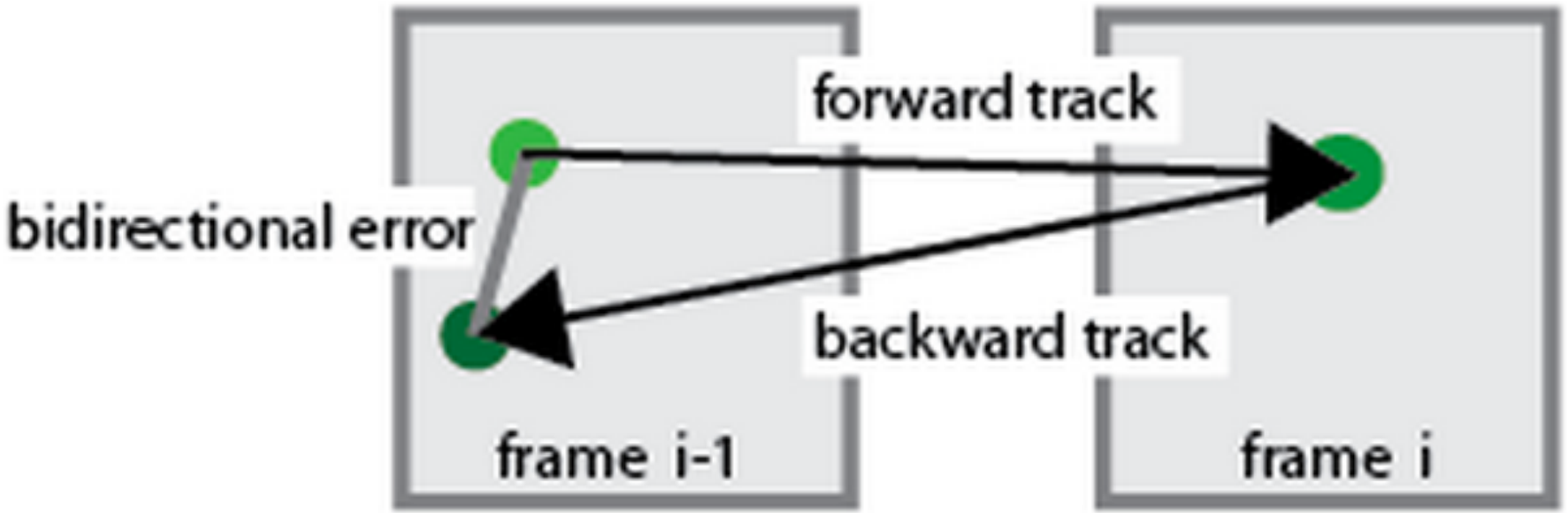
- E' anche possibile resettare i punti e usare il metodo setPoints

```
setPoints(H, POINTS)
```

Proprietà del Point Tracker

- `BlockSize` - dimensione della finestra di integrazione intorno a ciascun punto
- `NumPyramidLevels` - numero di livelli nella piramide
- `MaxIterations` - num. max di iterazioni
- `MaxBidirectionalError` - Soglia massima dell'errore forward-backward.

- `MaxBidirectionalError`,
  ideale nel range 0-3



Default: `inf`

Esercizio: Frame Interpolation usando il flusso ottico

- Dati due frame I_0 e I_1 di un video al tempo t=0 e t=1, vogliamo:

- generare frame intermedi, corrispondenti a un tempo 0<t<1 interpolando i due frame dati
- partire dalla funzione:

`FrameInterpolation.m`

Interpolazione

A. Interpolazione lineare: cross-fading

$$I_t(x,y) = (1-t)I_0(x,y) + tI_1(x,y)$$

- Implementa questo nella funzione:
  `ComputeCrossFadeFrame.m`
- per vedere il risultato, scommenta le righe 65-68 in "`FrameInterolation.m`"

Interpolazione, cont

B. Interpolazione forward warping

- l'interpolazione tiene conto del flusso ottico:

$$I_t(x + tu_0(x, y), y + tv_0(x, y)) = I_0(x, y)$$

- Implementa questo nella funzione:

`ComputeForwardWarpingFrame.m`

- per vedere il risultato, scommenta le righe 71-74 in "`FrameInterolation.m`"

NOTA: Introduce artefatti!

Iterpolazione, cont

C. Interpolazione backward warping:

$$I_t(x, y) = I_0(x - tu_t(x, y), y - tv_t(x, y))$$

problema: non conosciamo u_t, v_t

stima:

$$x' = x + tu_0(x, y), \quad y' = y + tv_0(x, y)$$

$$\forall (x'', y''), \text{t.c. } x'' \in \{floor(x'), ceil(x')\}, \quad y'' \in \{...\}$$

$$u_t(x'', y'') = u_0(x, y)$$

$$v_t(x'', y'') = v_0(x, y)$$

Iterpolazione, cont

- OSSERVAZIONE: in caso più punti (x,y) risultino convergere su uno stesso punto (x'',y''), scegliamo il massimo.

- Completa `ComputeFlowWarpFrame.m` (compreso `WarpFlow`)
- per vedere il risultato, scommenta le righe 77-80 in "`FrameInterolation.m`"

APPROFONDIMENTI...

Detection and motion-based tracking (Approfondimento)

- ipotesi: oggetti in movimento con camera ferma
- due passi:
  1. individuare oggetti in movimento in ogni frame
  2. associare tra loro le localizzazioni corrispondenti allo stesso oggetto e acquisite al variare tempo

Detection and motion-based tracking (approfondimento)

1. Detection:
   background subtraction + morfologia
2. Associazione: stima del movimento con filtri di Kalman: predice la posizione dell'oggetto nel prossimo frame, e determina la probabilità che una detection al tempo *t+1* corrisponda a un'altra al tempo *t*.

`multiObjectTracking()`

Face Tracking (Approfondimento)

`visionfacetracking()`

1. Detection di un volto in un frame via Viola-Jones detection
   `vision.CascadeObjectDetector`

2. Characterization: identificare caratteristiche del volto invarianti nel video e ben riconoscibili. Es Hue della pelle

3. Tracking della feature: avendo scelto come feature il colore, si può usare
   `vision.HistogramBasedTracker`

Face Tracking, detection

- Detection:

```
% Create a cascade detector object.
faceDetector = vision.CascadeObjectDetector();

% Read a video frame and run the detector.
videoFileReader =
vision.VideoFileReader('visionface.avi');
videoFrame      = step(videoFileReader);
bbox            = step(faceDetector, videoFrame);

% Draw the returned bounding box around the detected face.
videoOut =
insertObjectAnnotation(videoFrame,'rectangle',bbox,'Face'
);
figure, imshow(videoOut), title('Detected face');
```

Face Tracking, characterization

- Characterization:

```
% Get the skin tone information by
extracting the Hue from the video frame

% converted to the HSV color space.
[hueChannel,~,~] = rgb2hsv(videoFrame);


% Display the Hue Channel data and draw the
% bounding %box around the face.
figure, imshow(hueChannel);
title('Hue channel data');
rectangle('Position',bbox(1,:),'LineWidth',
2,'EdgeColor',[1 1 0]);
```

Nose detection and characterization

```
% Detect the nose within the face region.
The nose provides a more accurate measure of the skin tone
because it does no contain any background pixels.
noseDetector =
vision.CascadeObjectDetector('Nose',
'UseROI', true);

noseBBox = step(noseDetector, videoFrame,
bbox(1,:));
```

Nose tracking

```
% Create a tracker object.
tracker = vision.HistogramBasedTracker;

% Initialize the tracker histogram using the Hue channel pixels
from the nose.
initializeObject(tracker, hueChannel, noseBBox(1,:));

% Create a video player object for displaying video frames.
videoInfo    = info(videoFileReader);
videoPlayer  = vision.VideoPlayer('Position',[300 300
videoInfo.VideoSize+30]);

%…
```

```
% Track the face over successive video frames until finish.
while ~isDone(videoFileReader)

    % Extract the next video frame
    videoFrame = step(videoFileReader);

    % RGB -> HSV
    [hueChannel,~,~] = rgb2hsv(videoFrame);

    % Track using the Hue channel data
    bbox = step(tracker, hueChannel);

    % Insert a bounding box around the object being tracked
    videoOut =
insertObjectAnnotation(videoFrame,'rectangle',bbox,'Face');

% Display the annotated video frame using the video player object
    step(videoPlayer, videoOut);

end

% Release resources
release(videoFileReader),  release(videoPlayer);
```