Analisi di Immagini e Video: Homework 5

Consegna: Venerdì, 6 giugno, 2014

Prof. Pietro Zanuttigh

Tommaso Martini - 1081580

Introduzione

In questo homework ci è richiesto di implementare la tecnica di rilevazione del moto introdotta da Lucas e Kanade nel 1981. Di seguito vengono riportate delle brevi spiegazioni riguardo l'implementazione in Matlab dell'algoritmo; si rimanda ai commenti nel codice per una illustrazione più esauriente.

Soluzione

Per prima cosa il video su cui vogliamo operare viene caricato in memoria. Per motivi di semplicità computazionale è possibile scegliere un numero arbitrario di frame da leggere, in modo da non dover processare l'intera sequenza di immagini. Una volta importato il video in una matrice quadri-dimensionale, entriamo nel

ciclo principale del programma, che scorre ogni frame; in realtà escludiamo il primo e l'ultimo frame, in modo da non avere problemi di frame mancanti quando, in seguito, avremo bisogno dell'immagine precedente e successiva per eseguire la derivata temporale al frame corrente.

Per semplicità implementativa non lavoreremo con tutte e tre le componenti di colore, ma ci porteremo nello spazio YCbCr ed useremo solo la matrice bidimensionale Y, cioè della luminanza, poiché questa contiene gran parte dell'informazione percepibile dall'occhio umano. Estraiamo la luminanza anche per i frame precedente e successivo.

Del frame corrente calcoliamo le features più significative usando la funzione `corner()` di Matlab, che di default implementa l'algoritmo di Harris e calcoliamo il gradiente tramite il comando `gradient()`, che restituisce una colonna per il gradiente lungo l'asse X e una per il gradiente lungo l'asse Y.

Ci interessa rilevare il moto solo rispetto ai punti caratteristici dell'immagine, perciò eseguiamo un ulteriore ciclo annidato che prende in considerazione solo i punti classificati come features dal comando `corner()`, che in uscita dà una colonna di coordinate (x, y). Dato che nella nostra implementazione di Lucas-Kanade dovremo lavorare con una finestra quadrata attorno al pixel considerato, preveniamo possibili problemi ignorando i punti caratteristici troppo vicini al bordo dell'immagine, cioè quei punti che avrebbero una finestra uscente dal frame stesso. Questa, ovviamente, è una semplificazione tanto rude quanto drastica, ma per gli scopi di questo esercizio trascuriamo l'accuratezza e la completezza dell'algoritmo, tralasciando i casi più problematici.

Nel caso in cui sia possibile individuare una finestra intera attorno al pixel corrente, invece, la scorriamo interamente e costruiamo la matrice A e il vettore $\mathbf{b}$, definiti come segue:

$$A = \begin{bmatrix} \frac{\partial}{\partial x} I(p_1) & \frac{\partial}{\partial y} I(p_1) \\ \frac{\partial}{\partial x} I(p_2) & \frac{\partial}{\partial y} I(p_2) \\ \vdots & \vdots \\ \frac{\partial}{\partial x} I(p_M) & \frac{\partial}{\partial y} I(p_M) \end{bmatrix} \quad p_1, p_2, \ldots, p_M \in W \tag{1}$$

$$\mathbf{b} = \begin{bmatrix} \frac{\partial}{\partial t} I(p_1) \\ \frac{\partial}{\partial t} I(p_2) \\ \vdots \\ \frac{\partial}{\partial t} I(p_M) \end{bmatrix} \quad p_1, p_2, \ldots, p_M \in W \tag{2}$$

dove I è l'immagine e W è la finestra di M pixel attorno al pixel considerato.

Per risolvere il sistema:

$$A \begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{b} \tag{3}$$

secondo i minimi quadrati, ci serviamo del metodo proposto da Lucas e Kanade e valutiamo, invece, il seguente sistema:

$$A^T A \mathbf{d} = A^T \mathbf{b} \tag{4}$$

Come viene richiesto dal compito, a questo punto eseguiamo tre controlli sulla matrice $A^T A$ per individuare casi anomali:

1. controlliamo la singolarità della matrice 2×2 $A^T A$. Per fare ciò non usiamo la funzione `det()` di Matlab per controllare che il gradiente non sia nullo perché, operando questa per via numerica, può restituire facilmente risultati inconsistenti ed errati. Controlliamo, invece, che il rango della matrice (comando Matlab `rank()`) non sia minore del numero di righe o colonne, vale a dire 2. Se il rango è minore di 2 il programma viene bloccato e viene segnalata l'anomalia all'utente;

2. controlliamo che gli autovalori non siano troppo piccoli, sintomo di grande sensibilità al rumore: dopo aver calcolato gli autovalori di $A^T A$ tramite la funzione `eig()`,

li riordiniamo in ordine derescente e verifichiamo che il minore dei due, vale a dire λ_2, non scenda sotto una certa soglia, che possiamo scegliere arbitrariamente, ma che è stata impostata di default a 0.001; anche in questo caso, qualora si riscontrasse questo problema, il programma verrebbe fermato, comunicando all'utente l'evenienza;

3. controlliamo che il rapporto tra gli autovalori λ_1/λ_2 non superi una certa soglia, di default impostata a 100. Se la soglia viene superata, ancora una volta lo si comunica all'utente e si arresta il programma.

Una volta eseguiti tutti e tre i controlli, con l'istruzione `d = (A' * A)\(A' * b)` viene risolto il sistema e vengono calcolate le componenti dei vettori spostamento stimati del pixel centrale della finestra, vale a dire del pixel "feature" considerato.

Visualizzazione del moto stimato

Per visualizzare il moto stimato si usa una semplice tecnica: si crea un'immagine Matlab `figure`, che viene aggiornata con il frame corrente ad ogni passo del ciclo più esterno, sui frame del video. Per ogni corner valido identificato, di cui si è calcolato il vettore spostamento stimato, si disegna sopra il frame una freccia con origine sul corner e direzione e modulo identificati dal vettore spostamento (funzione `quiver()`). Per una migliore visualizzazione si è prima moltiplicato per un valore arbitrariamente grande il vettore spostamento, nel nostro caso per 10, così da poter vedere vettori più grandi e più facilmente apprezzabili.

Ricaricando ad ogni passo del ciclo il frame corrente e ridisegnandovi sopra i vettori individuati si ottiene una sorta di video in cui le frecce si muovono indicando la direzione di spostamento stimata.