UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Progetto di Visione Computazionale

Docente

Prof. Francesco Isgrò

Studenti

Iacuaniello Luigi (n97/234)
Panzuto Gianluca (n97/217)

Anno Accademico 2016-17

INDICE

1 - Introduzione

In questo progetto del corso di Visione Computazionale, abbiamo implementato in “Matlab” un’interfaccia grafica dove è possibile effettuare le tecniche di computational vision su immagini studiate a lezione.
Dato in input un’immagine (che può essere sia tra quelle di default di Matlab, sia scelta dall’utente), è possibile aggiungere rumori e filtri, effettuare line&corner detection, ridurre l’immagine tramite SVD e visualizzare l’istogramma dell’immagine.
Le tecniche di computational vision applicate sono implementate da zero e non sono quelle di default in matlab.
Nei capitoli successivi verranno spiegati nel dettaglio queste tecniche, come viene utilizzato il software e i test effettuati con i relativi commenti ai risultati ottenuti.

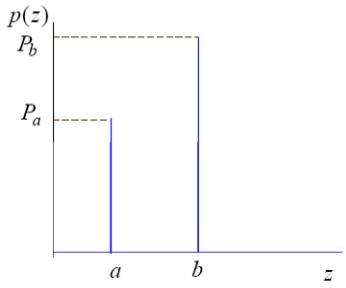
2 - Tecniche di Computational Vision utilizzate

In questo capitolo tratteremo delle tecniche di computational Vision studiate durante il corso. In questo progetto sono stati utilizzati algoritmi per aggiungere rumore all'immagine, per aggiungere filtri, effettuare line&corner detection ed effettuare la SVD.

2.1 – Rumori

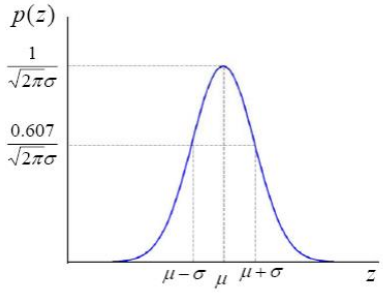
- Salt&Pepper noise: detto anche rumore impulsivo. Sostanzialmente un rumore di questo tipo è tale che esiste una probabilità che un pixel risulti essere o bianco o nero (da cui il nome). Abbiamo implementato una semplice funzione matlab che dato un intero p come input genera un numero random intero compreso tra 0 e p. Se il numero p in input è proprio uguale a quello sorteggiato (da qui il rispetto della definizione della probabilità) allora il pixel correntemente selezionato verrà settato a 0 o a 255 in base ad un sorteggio;

$$p(z) = \begin{cases} P_a & \text{per } z = a \\ P_b & \text{per } z = b \\ 0 & \text{altrimenti} \end{cases}$$



- Gaussian noise: Il rumore gaussiano è tale che un pixel è sporcato con una quantità che segue una distribuzione gaussiana. Formalmente:

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-\mu)^2/2\sigma^2}$$



2.2 – Filtri

- Gaussian Filter: Si crea una griglia in cui vengono calcolati i valori della funzione di Gauss di cui si assume sia la distribuzione dell'errore di cui sono affetti i pixel, e con un prodotto di convoluzione tra la maschera corrente e la griglia di Gauss si procede al calcolo della nuova matrice dell'immagine shiftando la maschera sull'immagine originale;

- Median Filter: L'idea è quella di definire una maschera e mentre si scorre la maschera sulla matrice dei pixel, si calcola il valore mediano dell'intorno definito dalla maschera e lo si sostituisce ai valori dei pixel originari. Per evitare dannose sovrascritture si usa una matrice di appoggio M inizializzata a I che è la matrice dell'immagine di input;

- Adaptive Median Filter: Il filtro mediano adattativo preserva meglio i contorni dell'immagine affetta da rumore impulsivo verificando se il mediano trovato nella maschera sia affetto da rumore impulsivo. Se lo è allarga la maschera. Se non lo è, verifica se il mediano preserva l'informazione e verifica se il valore originale è un impulso. Se non lo è lascia il valore del pixel così com'è, altrimenti si usa il valore mediano nella maschera;

- Sharp Filter: Per evidenziare i contorni all'immagine di partenza, si somma l'immagine trattata con una maschera che ha i valori del *laplaciano* che si ottiene in questo modo:

$$\nabla^2 I(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\delta^2 I}{\delta^2 y}$$

dove:

$$\frac{\partial^2 I}{\partial x^2} = I(x+1,y) - 2I(x,y) + I(x-1,y) \text{ e } \frac{\delta^2 I}{\delta y^2} = I(x,y+1) - 2I(x,y) + I(x,y-1)$$

da cui

$$\nabla^2 I(x,y) = I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1) - 4I(x,y)$$

Pertanto applicare il laplaciano ad un'immagine I significa fare il prodotto di convoluzione della maschera del laplaciano sull'immagine I. La maschera del laplaciano assume pertanto la forma

$$L = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

- Unsharp Filter: In pratica si procede ad applicare un effetto blur all'immagine originale a cui abbiamo applicato il laplaciano, questo lo sottraiamo all'immagine originale per poi sommare il risultato all'immagine originale.

2.3 – Line&Corner detection

- Corner Detection (Harris): L'obiettivo primario per l'identificazione dei corner è quello di identificare preliminarmente in un'immagine i contorni. I contorni possono essere assimilati a quell'insieme di pixel in cui esiste lungo la direzione x o lungo una direzione y per esempio, una variazione di intensità. Ovviamente se i contorni sono curve per esempio più o meno parallele alla direzione x si ha che la variazione lungo quella direzione sarà veramente piccola. Bisogna quindi mettere in evidenza quei pixel che si trovano in quella zona di confine lungo la quale esistono importanti variazioni di intensità. L'idea è quella di utilizzare quindi un funzionale che data la matrice dell'immagine originale segnali queste variazioni. Il passo successivo visto che un corner non è altro che un insieme di punti dove esiste una forte variazione di intensità lungo la curva a cui appartengono, basterà settare un valore soglia oltre il quale quel pixel (o meglio quell'insieme di pixel) verrà segnalato come corner. Il funzionale utilizzato per evidenziare i punti che appartengono a curve più o meno parallele alla direzione x o alla direzione y è il seguente:

$$E_{u,v} = A(x,y)u^2 + 2C(x,y)uv + B(x,y)v^2$$

dove

$$A(x,y) = \sum_{i,j=-a}^{a} I_x(x+i, y+j)^2 \text{ e } B(x,y) = \sum_{i,j=-a}^{a} I_y(x+i, y+j)^2 \text{ mentre}$$

$$C(x,y) = \sum_{i,j=-a}^{a} I_x(x+i, y+j) I_y(x+i, y+j)$$

I_x e I_y sono le variazioni di intensità dei pixel lungo rispettivamente la direzione x e y;

- Kanade-Tomasi (corner): Si calcola per ogni pixel dell'immagine la seguente matrice M:

$$M = \begin{pmatrix} A(x,y) & C(x,y) \\ C(x,y) & B(x,y) \end{pmatrix}$$

  di cui si calcola gli autovalori.
  Se l'autovalore minimo è maggiore di una certa tolleranza, quel pixel I(x,y) viene marcato come corner. Questo metodo richiede più risorse computazionali visto che per ogni pixel bisogna calcolare i gradienti e diagonalizzare la matrice M;

- Canny (line): Si effettua un edge enhancement, applicando un filtro gaussiano. Successivamente effettuo una non-maxima suppression ed infine applico una sogliatura, scegliendo quale pixel evidenziare come edge.

- Hough Trasformation (line): Si trasforma il problema della rilevazione di linee in un problema di intersezione in cui dato una retta y = mx + n , il punto p = (x , y) corrisponde alla retta n = (-x) m + y nello spazio dei parametri. Nello spazio dei parametri una retta è identificata da punti definiti dall'intersezione di n curve. Quindi se alcuni pixel appartengono ad una retta nello spazio tradizionale, ciò significa che nello spazio dei parametri ci sarà un intorno di accumulazione di punti. Pertanto un problema di rilevazione di rette può essere trasformato in un problema di rivelazione di punti di aggregazione. Usiamo la rappresentazione polare anziché quella cartesiana per il semplice motivo che nella prima i parametri che la definiscono sono limitati e definiscono quindi una griglia. Ricordando quindi che la rappresentazione polare di una retta è

$$\rho = x\cos\theta + y\sin\theta$$

  dove

$$\rho \in [0, \sqrt{M^2 + N^2}] \text{ e } \theta \in [0, \pi]$$

  M e N sono le dimensioni della matrice dell'immagine.

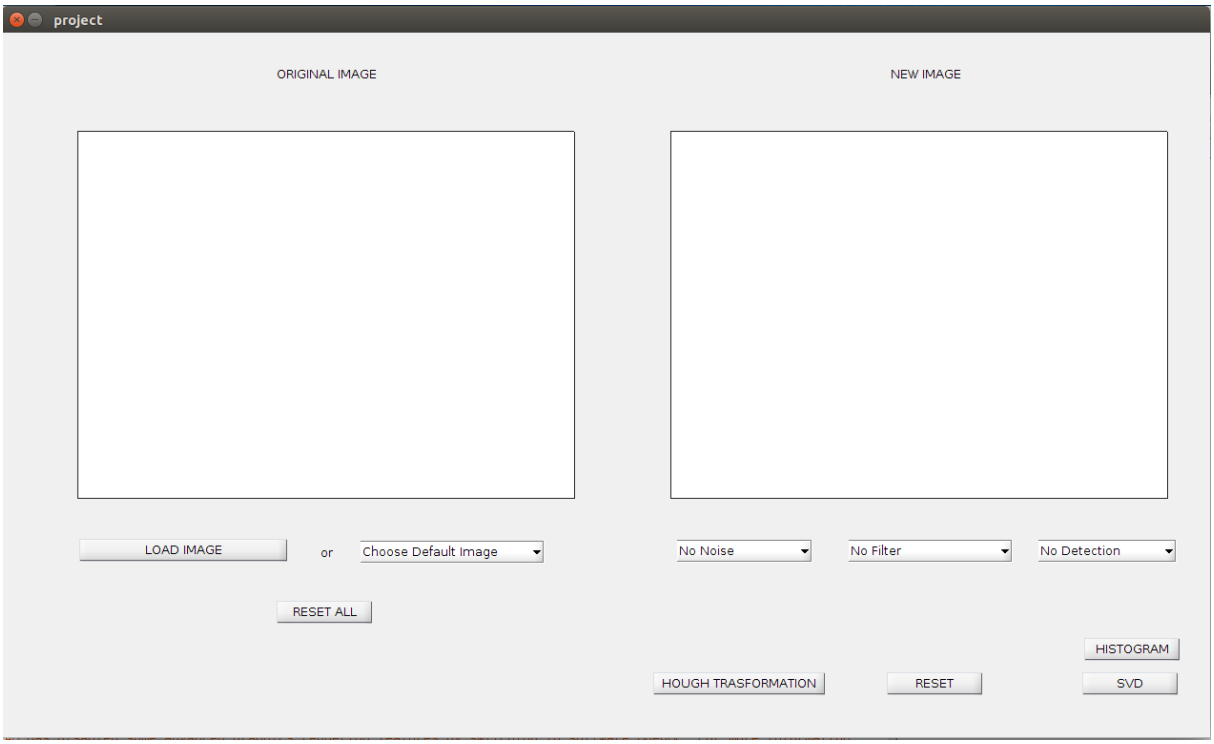
2.4 – SVD

SVD sta per *Singular Value Decomposition*.
Qualsiasi matrice *m x n* può essere decomposta nel prodotto di tre matrici U, D e V tali che le colonne della matrice U di dimensione *m x n* siano vettori unitari mutuamente ortogonali come le colonne della matrice *n x n* V. La matrice D di dimensione *m x n* è diagonale, e i suoi elementi diagonali chiamati valori singolari sono tutti maggiori o uguali a zero.
Un'interessante applicazione di questa decomposizione è quella che permette di fare una compressione di un'immagine andando a scartare tutti i suoi valori singolari non significativi perché vicini allo zero macchina. L'idea è quella di andare a decomporre la matrice dell'immagine data in input in un prodotto di matrici del tipo di cui sopra e di analizzare la matrice D.
La matrice D viene ridotta ad una nuova matrice che ha i valori diagonali significativi andando a scartare le colonne delle matrici U e V a cui sono asssociati gli elementi diagonali esclusi. Si ottengono così tre matrici ridotte di U, V e D di dimensione inferiore. Per recuperare l'immagine si procede ad un nuovo prodotto delle ridotte di U, V e D.
Nel codice implementato si può notare che di tutta la matrice U, S e V prendiamo solo le prime due righe e colonne corrispondenti ai valori singolari di D significativi. Ne sono solo due poiché l'immagine data in input è un'immagine sintetica in bianco e nero. Procedendo al prodotto delle ridotte di U, D e V, e procedendo alla visualizzazione dell'immagine non si notano perdite di informazione nell'immagine risultante.

3 - Utilizzo del software

In questo capitolo spiegheremo come funziona il software implementato, facendo vedere come è fatta l'interfaccia grafica e le sue funzionalità.

Una volta che si manda in esecuzione il file "project.m", apparirà la seguente interfaccia grafica:



Da come si può notare, ci sono due axis: nel primo axis (quello a sinistra) apparirà l'immagine originale, mentre nel secondo apparirà l'immagine modificata dai filtri e/o rumori etc.
Analizziamo la parte sinistra dell'interfaccia grafica.
L'immagine in input può essere sia caricata dall'utente (tramite il pulsante "LOAD IMAGE"), sia scelta tra alcune immagini di default che matlab mette a disposizione (vedere il menù a tendina a fianco al pulsante "LOAD IMAGE").
È anche possibile resettare il tutto, tramite il pulsante "RESET ALL" (diverso dal pulsante "RESET" che spiegheremo a breve).
Analizziamo adesso la parte destra dell'interfaccia grafica.

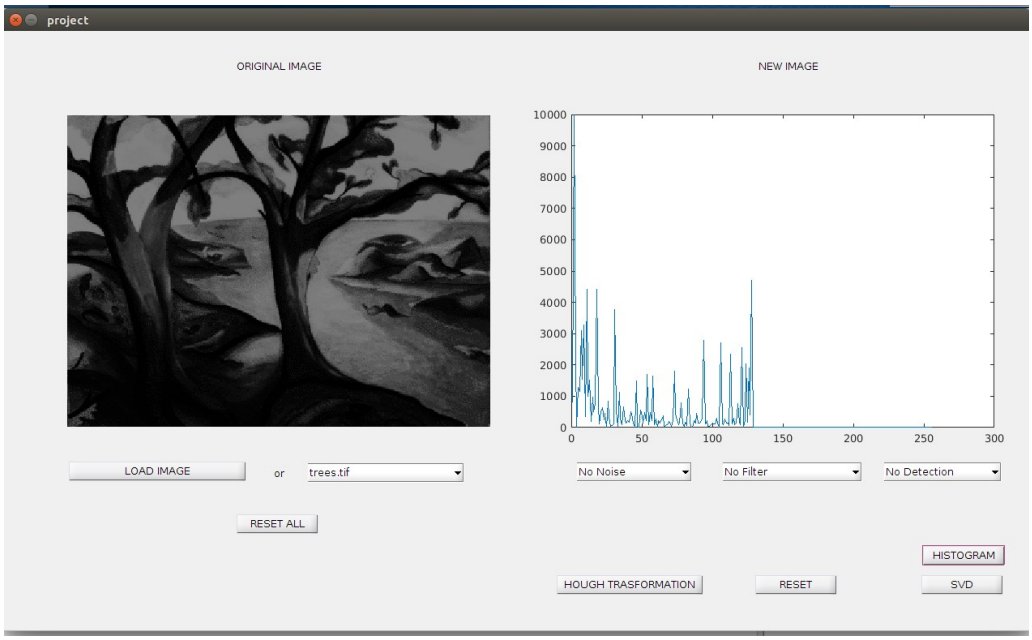
Abbiamo 3 menù a tendina e 4 pulsanti. Il primo menù a tendina contiene i rumori da applicare all'immagine (SaltPepper e Gaussiano).
Il secondo menù a tendina contiene i filtri da applicare all'immagine (gaussiano, sharp, unsharp, mediano e mediano adattativo).
Il terzo menù a tendina contiene gli algoritmi di detection (Canny, Harris e Tomasi).
Analizziamo infine i pulsanti.
Il pulsante "RESET" pulisce solo l'axes a destra (quindi rimuove tutti i filtri, rumori etc.).
Il pulsante "SVD" applica l'algoritmo di Singular Value Decomposition dell'immagine.
Il pulsante "HISTOGRAM" visualizza l'istogramma dell'immagine.
Il pulsante "HOUGH TRASFORMATION" applica la trasformata di Hough dell'immagine.
Attenzione: se non si fa il reset dell'axes e si applicano consecutivamente due operazioni di modifica/filtraggio/detection, esse si sovrappongono (esempio: se applico il rumore SaltPepper e poi dopo applico l'algoritmo di Harris, quest'ultimo viene applicato sull'immagine affetta dal rumore SaltPepper).
Di seguito un esempio di applicazione di Harris su un immagine di default.

4 - Test

In questo capitolo andremo ad analizzare alcuni test che sono stati effettuati nell'interfaccia grafica.
Andremo ad analizzare alcune particolarità degli istogrammi delle scale di grigio, sui pregi e difetti dei filtri e delle feature detection.

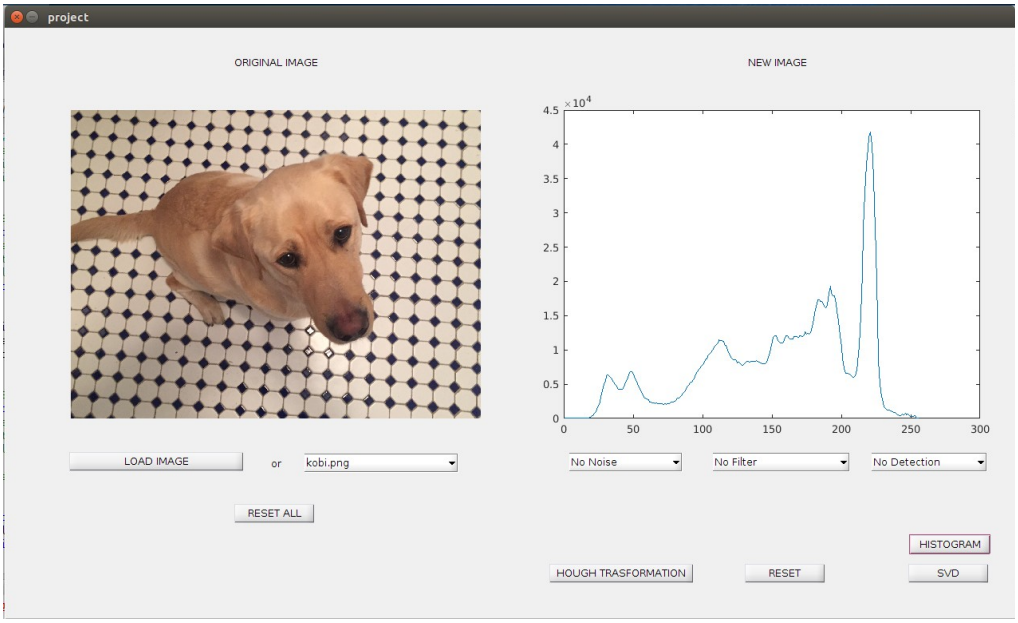
4.1 – Test su Istogrammi

Qui andremo ad analizzare alcuni casi particolari dei vari istogrammi delle scale di grigio.
Il primo test lo abbiamo fatto su una immagine di default ("trees.tif):



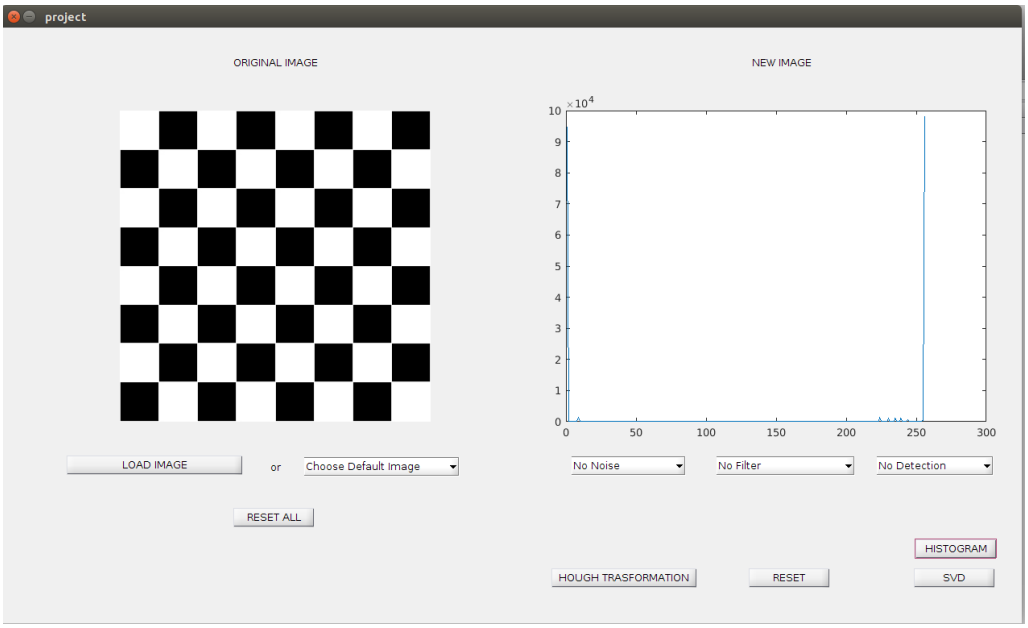
Da come possiamo notare nel grafico sulla destra, i valori alti dell'istogramma tendono a stare sulla sinistra. Questo ci dice che l'immagine è abbastanza scura. Inoltre possiamo notare che l'istogramma è "ammassato". Questo ci dice che l'immagine scelta ha poco contrasto.

Passiamo al test successivo, fatto su un’altra immagine di default (“kobi.png”).



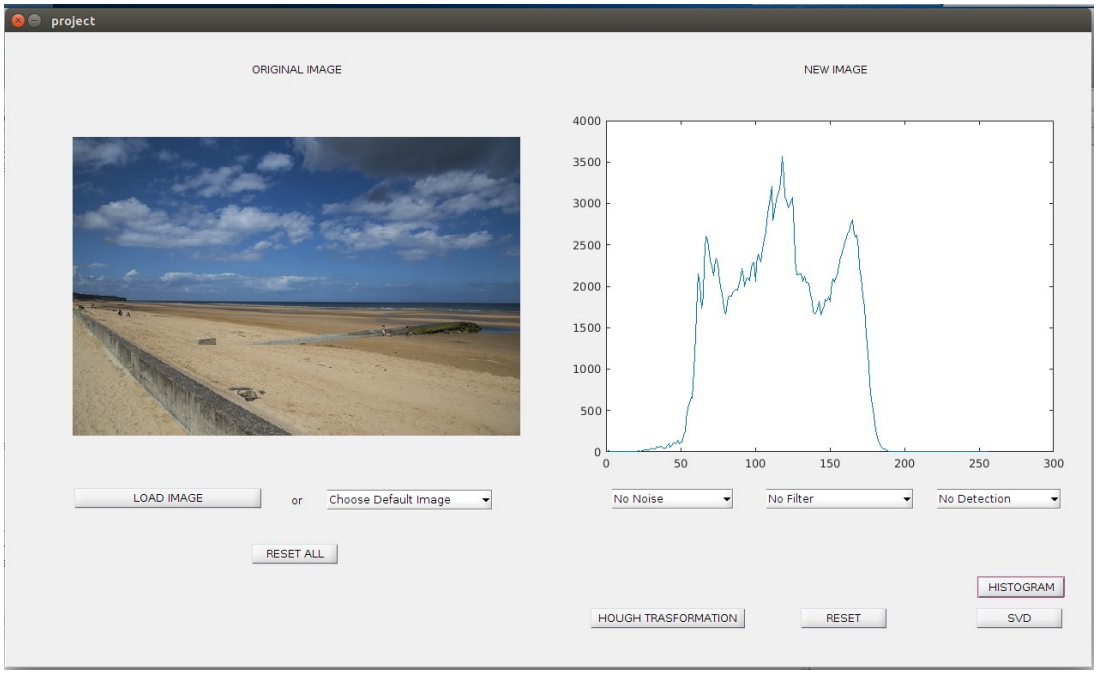
Qui notiamo subito che l’istogramma è più “sparso”. Questo perché l’immagine è molto più contrastata rispetto all’immagine del test precedente.

Passiamo ora al terzo test, fatto su un’immagine scelta dall’utente (in particolare “chess.jpg”).



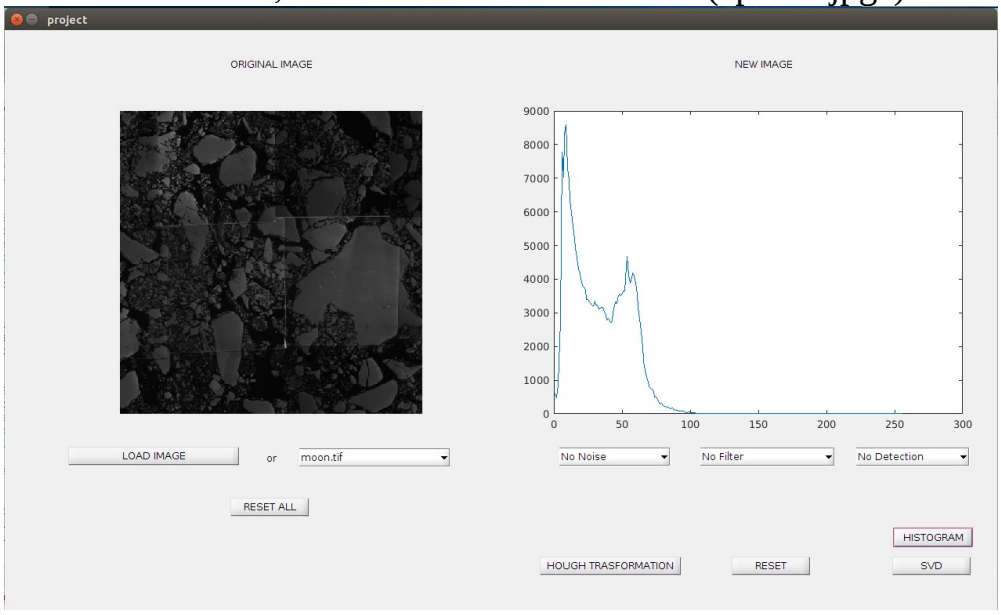
In questo caso, l’immagine scelta è formata solo da 2 colori: bianco e nero. Pertanto l’istogramma avrà i picchi solo ai due estremi. Ovviamente l’immagine risulta ben contrastata, causata dal fatto che l’istogramma risulta “sparso”.

Passiamo al penultimo test, utilizzando un’immagine scelta dall’utente (“beach.jpg”).



La prima cosa che si nota è che l’istogramma risulta “ammassato” a causa del poco contrasto dell’immagine.

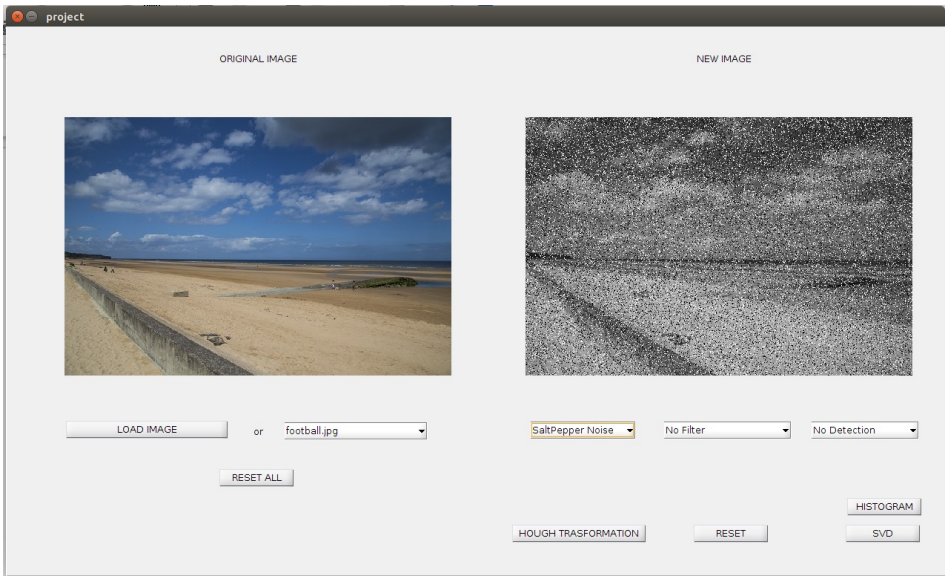
Passiamo ora all’ultimo test, su un file scelto dall’utente (“prova.jpg”).



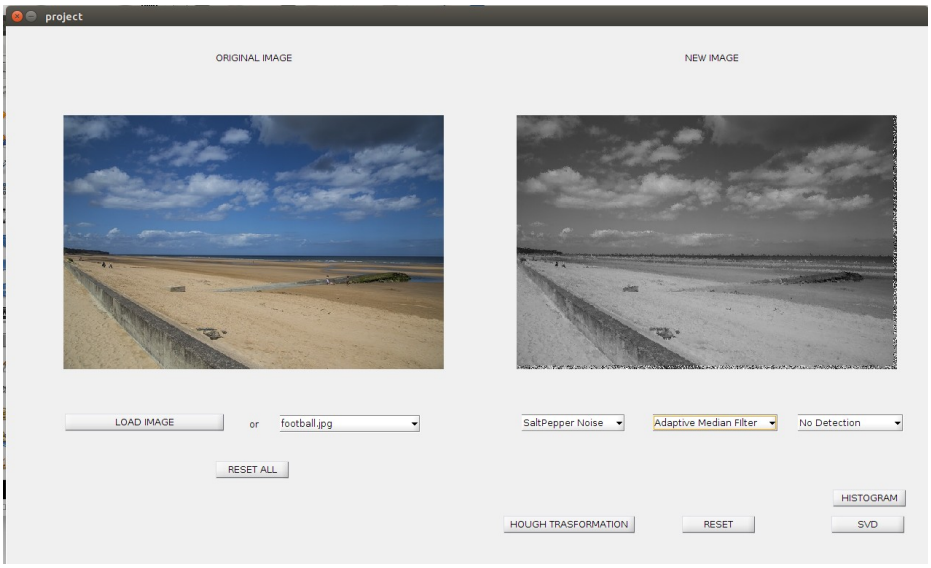
Banalmente anche qui abbiamo un’immagine poco contrastata (istogramma ammassato) e scura (istogramma tendente sulla sinistra).

4.2 – Test su filtri e feature detection

Il primo test è quello del filtro mediano adattativo. Come citato nel capitolo 2, il filtro mediano adattativo ha la particolarità di togliere il più possibile il rumore SaltPepper. Abbiamo voluto verificare tramite l’immagine “beach.jpg”.

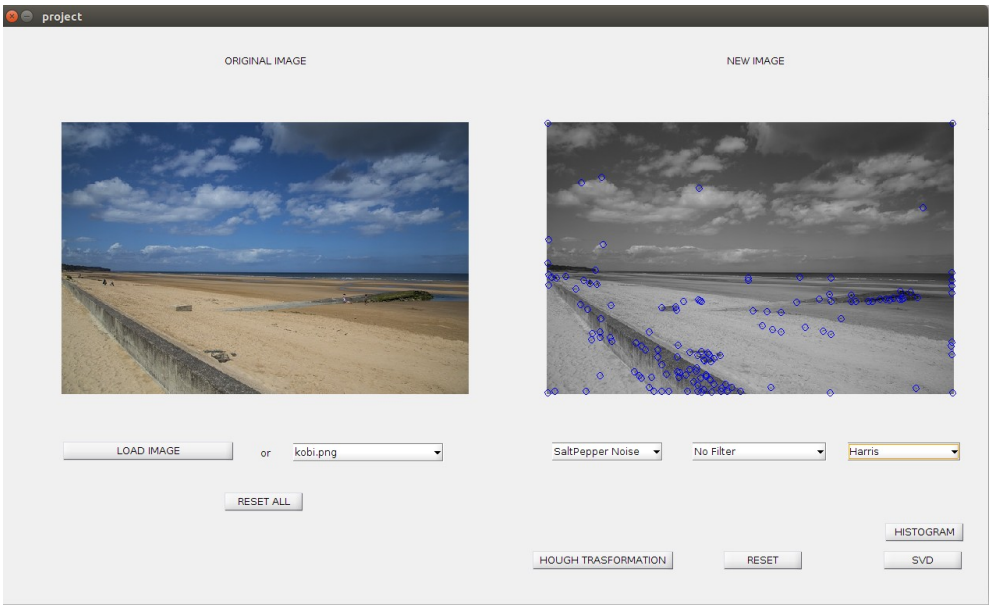


Qui è stato applicato un rumore SaltPepper con probabilità 1/5.
Successivamente abbiamo applicato il filtro mediano adattativo e il risultato è stato il seguente:

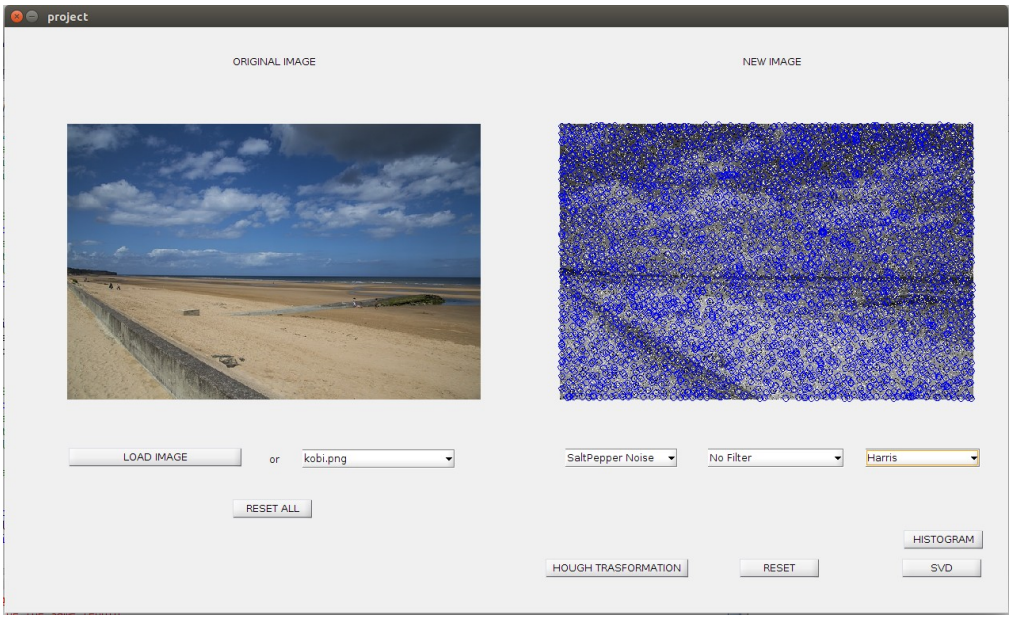


Praticamente il rumore SaltPepper è scomparso (a parte qualcosa nei lati dell'immagine).

Il secondo test consiste nel verificare il corretto funzionamento dell'algoritmo di Harris (corner detection), con e senza rumore.
Analizziamo prima di tutto il caso senza rumore tramite l'immagine "beach.jpg".

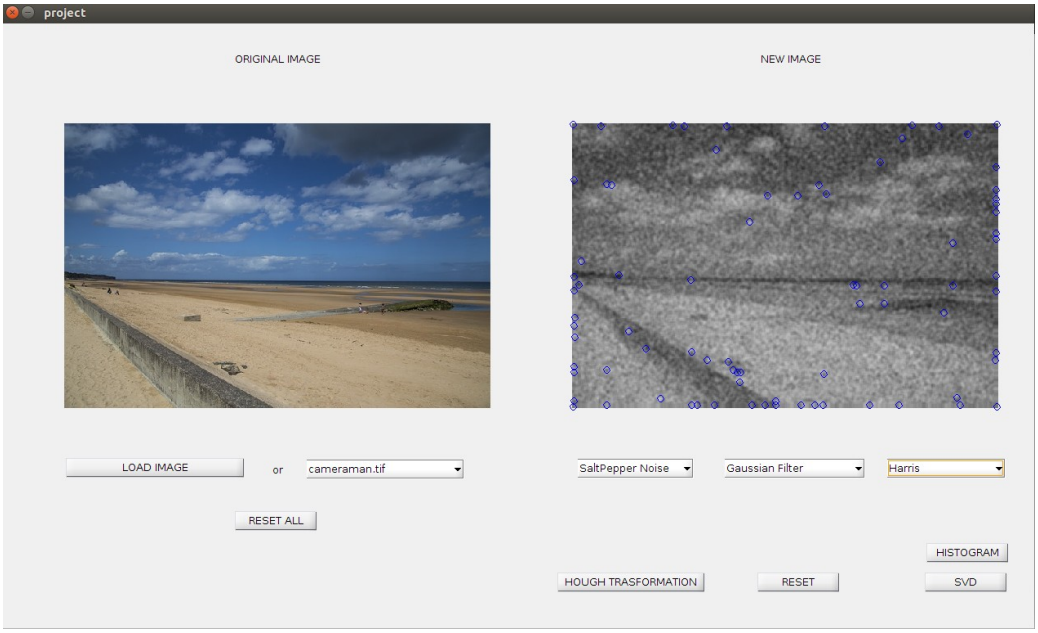


L'algoritmo riesce a catturare i corner dell'immagine correttamente. Vediamo adesso nel caso di immagine con rumore SaltPepper (1/5 probabilità).



Qui l'algoritmo segnerà tutti corner i punti causati dal rumore SaltPepper. Da qui si capisce come l'algoritmo di Harris sia fortemente condizionato dal rumore.

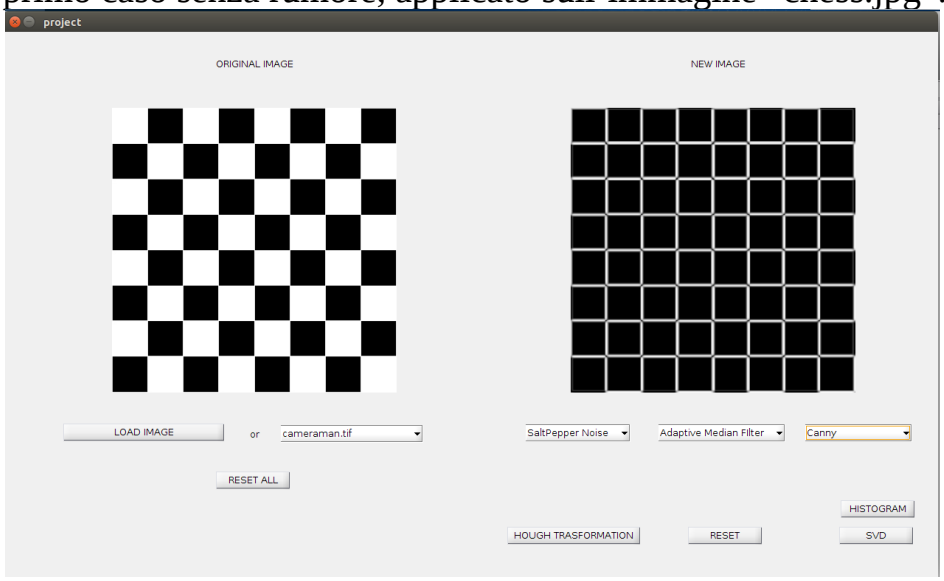
Adesso applichiamo l’algoritmo di Harris su un’immagine con rumore SaltPepper, ma applicando un filtro gaussiano.



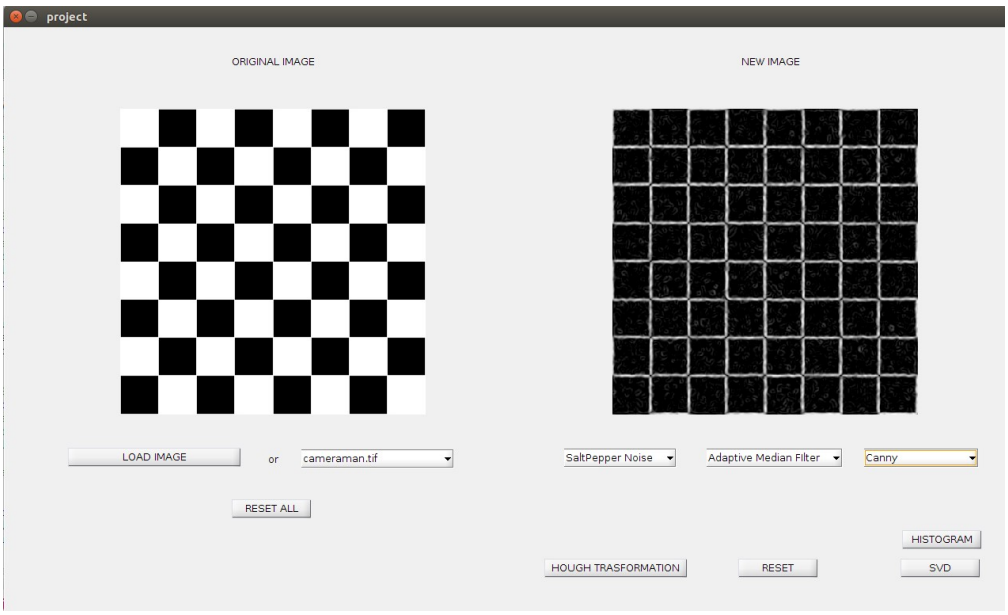
Qui invece il risultato non è ottimo, ma sicuramente migliore rispetto a prima. Questo perché il filtro gaussiano ha in parte ridotto il rumore saltPepper.

Il terzo test consiste nell’applicazione dell’algoritmo di Canny (sia nei casi con rumore, sia nei casi senza rumore).

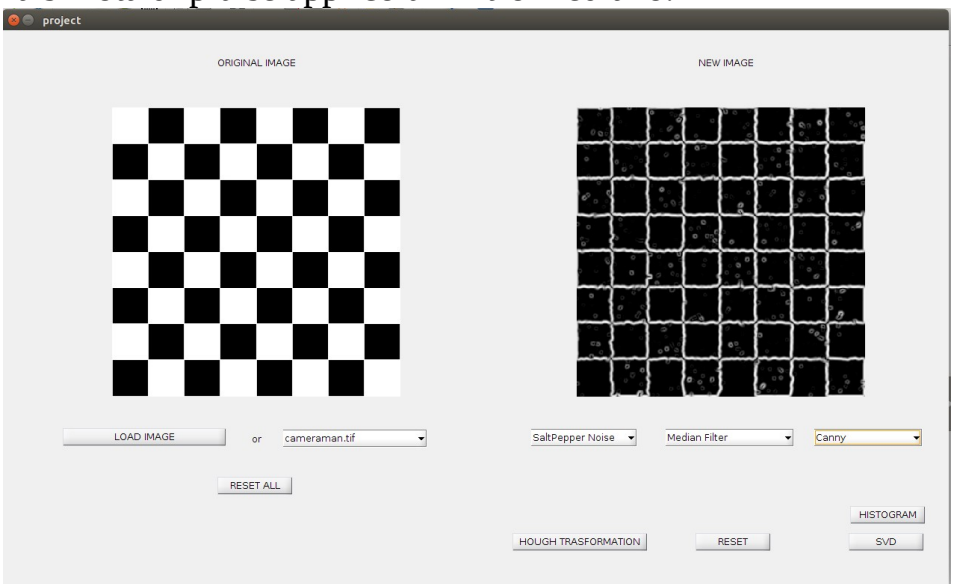
Vediamo il primo caso senza rumore, applicato sull’immagine “chess.jpg”.



Qui vediamo come l’algoritmo di Canny riesce a prendere correttamente tutte le linee della scacchiera. Vediamo adesso se applico un rumore SaltPepper all’immagine.

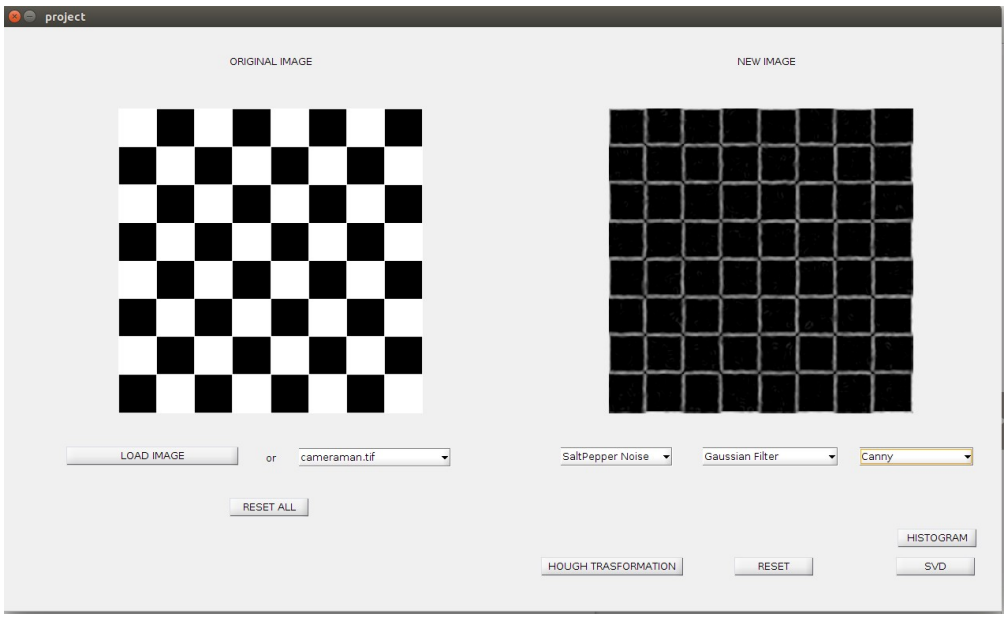


Qui subito vediamo che riesce correttamente a prendere le linee della scacchiera, ma vediamo dei piccoli segni bianchi sui quadrati neri che sono causati dal rumore.
La differenza si nota di più se applico un filtro mediano.



Qui il rumore si nota di più rispetto a prima e si può notare che le linee che si è tirato fuori canny non sono proprio dritte, ma sono affette anche loro da rumore.

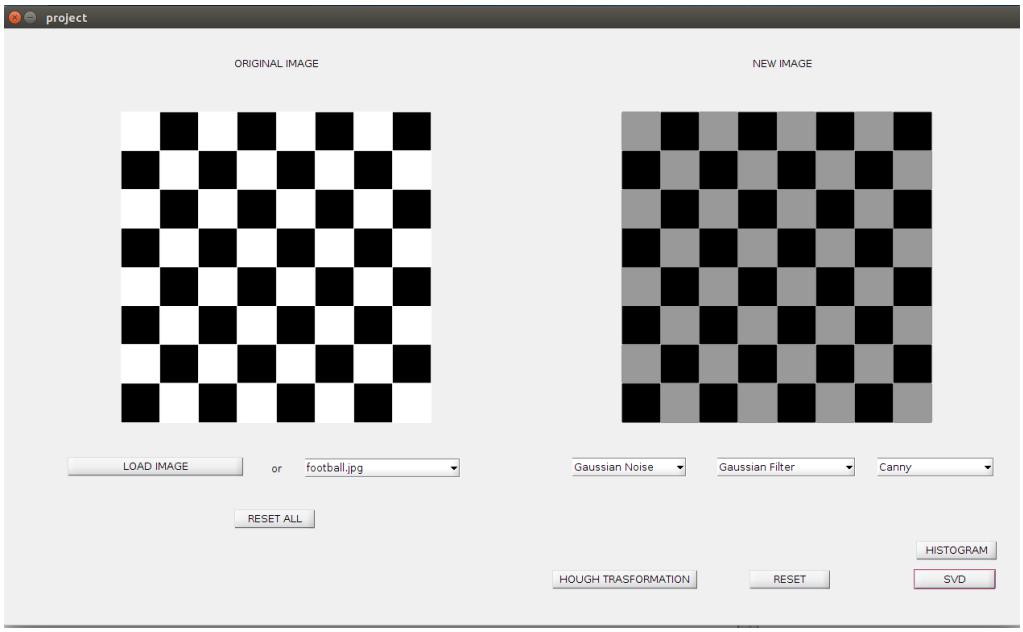
Vediamo cosa succede su un’immagine con rumore SaltPepper e filtro gaussiano.



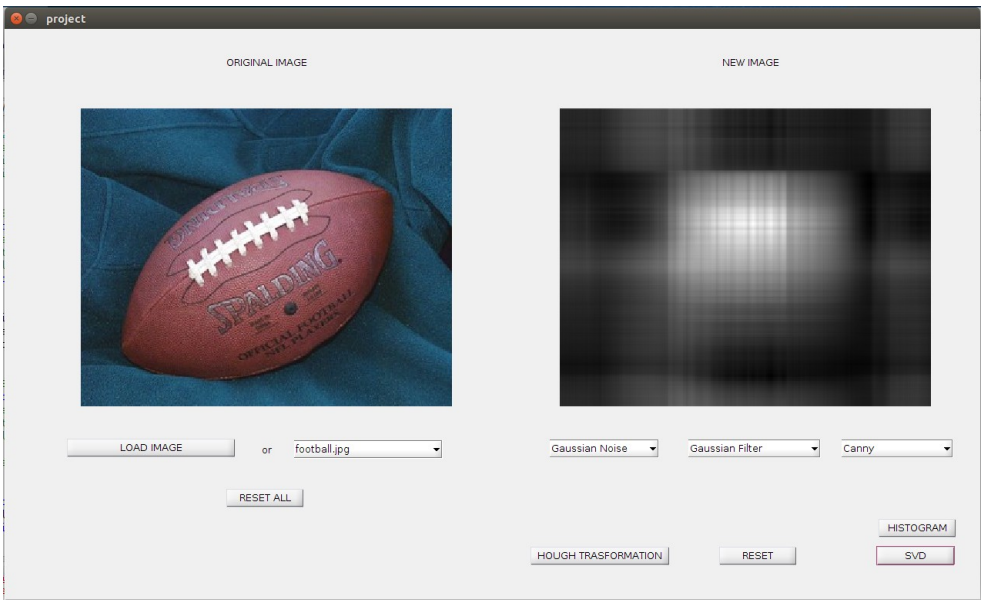
A parte qualche altro piccolissimo rumore sulle linee e qualche piccolissimo rumore sulle parti nere, l'algoritmo funziona correttamente.

Il quarto test consiste di applicare alle immagini SVD (Singular Value Decomposition).

Abbiamo preso due immagini: chess.jpg e football.jpg
Analizziamo la prima immagine:



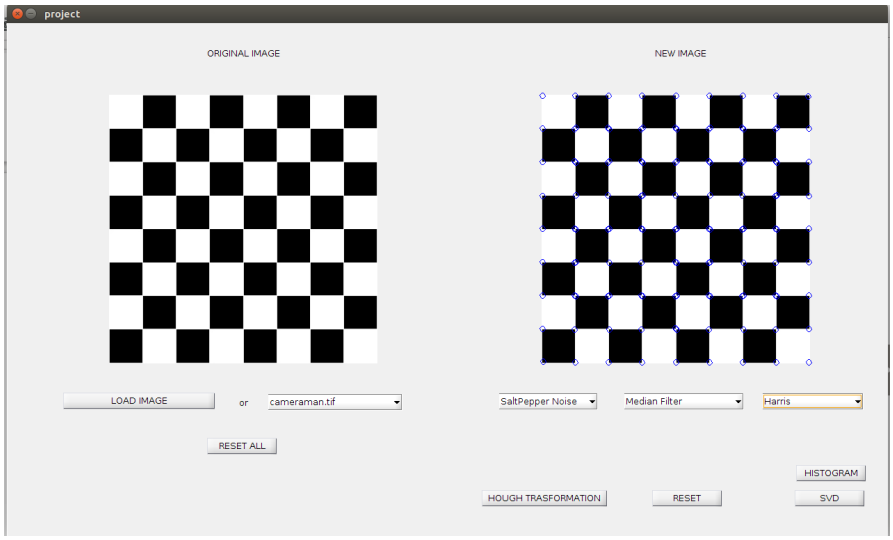
Subito notiamo che l'immagine di output è uguale a quella di input, al più di un cambio di colore dal bianco al grigio.
Vediamo adesso la seconda immagine:



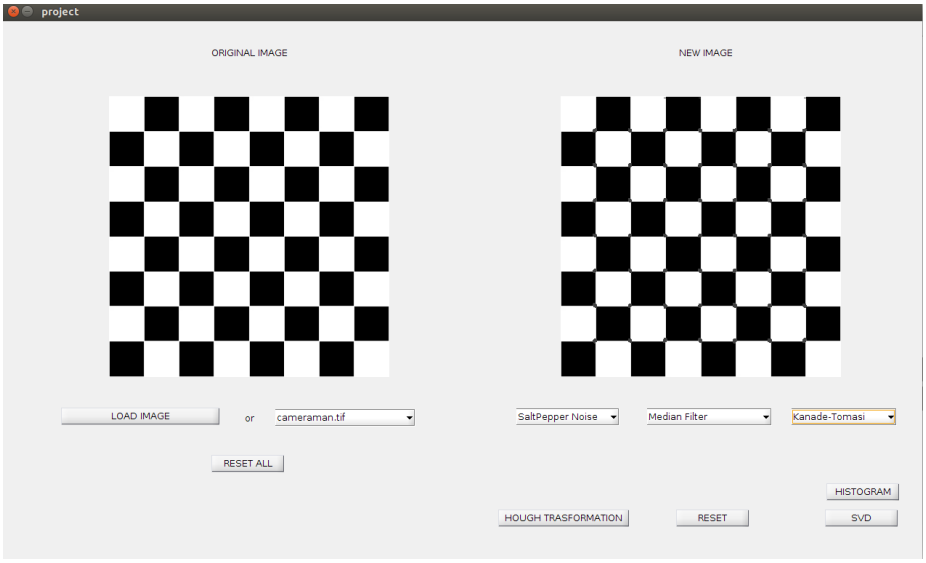
Qui si nota subito come SVD decompone l'immagine originale. Si può notare la forma che, bene o male, rimane la stessa, ma i colori e altro vengono completamente alterati.

Il quinto e ultimo test mette in evidenza la differenza tra il corner detection di Harris e quello di Tomasi.

Vediamo l'applicazione dell'algoritmo di Harris.



Vediamo successivamente l'applicazione dell'algoritmo di Tomasi.



Subito notiamo che l'algoritmo di Tomasi è più preciso di quello di Harris, visto che Harris prende in considerazione sia i corner ai lati dell'immagine, sia quelli all'interno dell'immagine (li segna due volte, visto che il punto che unisce due quadrati neri li considera due corner). Mentre Tomasi prende in considerazione solo i corner interni all'immagine e li prende una volta soltanto.

4.3 – Commenti sui risultati

In base ai risultati ottenuti in questi test, possiamo concludere che il progetto realizzato rispetta la teoria della computational vision citata al capitolo 2.
Questo vale sia per i filtri e per le feature detection studiate (line e corner), sia per le proprietà dell'istogramma studiate.

5 - Codice sorgente

canny.m

```
function [Ibw] = canny(x)

    Nx1=10;Sigmax1=2;Nx2=10;Sigmax2=2;Theta1=pi/2;
    Ny1=10;Sigmay1=2;Ny2=10;Sigmay2=2;Theta2=0;

    % Calcolo il gradiente su x
    filterx=d2dgauss(Nx1,Sigmax1,Nx2,Sigmax2,Theta1);
    Ix= conv2(double(x),double(filterx),'same');

    % Calcolo il gradiente su y
    filtery=d2dgauss(Ny1,Sigmay1,Ny2,Sigmay2,Theta2);
    Iy=conv2(double(x),double(filtery),'same');

    % Calcolo la norma del gradiente
    NVI=sqrt(Ix.*Ix+Iy.*Iy);

    % Applico la sogliatura
    I_max=max(max(NVI));
    I_min=min(min(NVI));
    level=0.1*(I_max-I_min)+I_min;
    Ibw=max(NVI,level.*ones(size(NVI)));

end


function h = d2dgauss(n1,sigma1,n2,sigma2,theta)
    r=[cos(theta) -sin(theta);
       sin(theta)  cos(theta)];
    for i = 1 : n2
        for j = 1 : n1
            u = r * [j-(n1+1)/2 i-(n2+1)/2]';
            h(i,j) = gauss(u(1),sigma1)*dgauss(u(2),sigma2);
        end
    end
    h = h / sqrt(sum(sum(abs(h).*abs(h))));
end


function y = dgauss(x,std)
    y = -x * gauss(x,std) / std^2;
end

function y = gauss(x,std)
    y = exp(-x^2/(2*std^2)) / (std*sqrt(2*pi));
end
```

fl_gauss.m

```
function [M] = fl_gauss(I,s)
    t = s
    [num_righe, num_col]=size(I);
    sigma=round((2*s+1)/5);
    %I=imnoise(I, 'gaussian');
    %
    for i=1:num_righe
        for j=1:num_col
            %aggiungi rumore
            I(i,j)=I(i,j)+floor(45*randn(1,1));
        end
    end
    M=I;
    H=gauss_kernel(s+1,sigma);
    h_min=min(H(:));
    H=round(H/h_min);
    den=sum(H(:));
    H=double(H/den);
    for i=1:num_righe-s
        for j=1:num_col-t
            %maschera del kernel (s+1)*(t+1)
            T=I(i:i+s,j:j+t);
            T=double(T);
            A=conv2(H,T,'valid');
            M(i:i+s,j:j+t)=A;
            % pause
        end
    end
end

function K = gauss_kernel(N,std)
    % Filtro Gaussiano
    % - N = Dimensione (NxN) del filtro
    % - std = deviazione standard
    % - K = kernel gaussiano
    dim = (N-1)/2;
    % preparazione griglia
    [x,y] = meshgrid(-dim:dim,-dim:dim);
    arg = -(x.*x + y.*y)/(2*std^2);
    % funzione di Gauss
    K = exp(arg);
    K=double(K);
end
```

fl_mediano.m

```
function [M] = fl_mediano(I,s,t)
   [num_righe, num_col]=size(I);
   I= saltpepper(I,2);
   M=I;
   for i=1:num_righe-s
      for j=1:num_col-t
         %maschera del kernel (s+1)*(t+1)
         T=I(i:i+s,j:j+t);
         %calcolo mediano in un insieme che Ã¨ ordinato
         T_ord=sort(T(:));
         i_mediano=round(length(T_ord)/2);
         %filtro mediano
         mediano=T_ord(i_mediano);
         % mediano=median(T_ord(:));
         M(i:i+s,j:j+t)=mediano;
      end
   end
end
```

---

fl_mediano_ad2.m

```
function [M] = fl_mediano_ad2(I)
   I=saltpepper(I,5);
   [num_righe num_col]=size(I);
   M=I;
   smax=8;
   for i=1:num_righe-8
      for j=1:num_col-8
         raggio=2;
         while (raggio<=smax)
            window=I(i:i+raggio,j:j+raggio);
            MM=window;
            window_ord=sort(window(:));
            Zmax=max(window_ord(:));
            Zmin=min(window_ord(:));
            i_mediano=round(length(window_ord)/2);
            Zmediano=window_ord(i_mediano);
            if(Zmediano>Zmin && Zmediano<Zmax)
               [mw nw]=size(window);
               for rr=1:mw
                  for cc=1:nw
                     if(window(rr,cc)<=Zmin || window(rr,cc)>=Zmax)
                        MM(rr,cc)=Zmediano;
                     end
                  end
               end
               M(i:i+raggio,j:j+raggio)=MM;
               break;
            end
            raggio=raggio+1;
         end
      end
   end
```

```
end
```

---

gaussianNoise.m

```
function [I] = gaussianNoise(I)
    [num_righe, num_col]=size(I);
    for i=1:num_righe
        for j=1:num_col
            I(i,j)=I(i,j)+floor(25*randn(1,1));
        end
    end
end
```

Harris.m

```
function [I,cols,rows] = Harris(I,s)

    I = I(:,:,1);

    radius = 1;
    order = 2*radius + 1;
    threshold = 1000;

    [dx,dy] = meshgrid(-1:1, -1:1);

    Ix = conv2(double(I),dx,'same');
    Iy = conv2(double(I),dy,'same');

    dim = max(1,fix(6+s));
    m = dim;
    n = dim;

    [h1,h2] = meshgrid(-(m-1)/2 : (m-1)/2, -(n-1)/2 : (n-1)/2);

    hg = exp(-(h1.^2+h2.^2)/(2+s^2));

    [a,b] = size(hg);

    sum = 0;
    for i=1:a
        for j=1:b
            sum = sum+hg(i,j);
        end
    end
    g = hg ./sum;

    Ix2 = conv2(double(Ix.^2),g,'same');
    Iy2 = conv2(double(Iy.^2),g,'same');
    Ixy = conv2(double(Ix.*Iy),g,'same');
```

```
    R = (Ix2.*Iy2 - Ixy.^2) ./(Ix2+Iy2 + eps);
    mx = ordfilt2(R, order^2,ones(order));
    harris_points = (R == mx) & (R > threshold);
    [rows,cols] = find(harris_points);

end
```

---

houghTransform.m

```
function [rho,theta,houghSpace] = houghTransform(theImage,thetaSampleFrequency)

    %Definisco lo spazio hough
    theImage = flipud(theImage);
    [width,height] = size(theImage);

    rhoLimit = norm([width height]);
    rho = (-rhoLimit:1:rhoLimit);
    theta = (0:thetaSampleFrequency:pi);

    numThetas = numel(theta);
    houghSpace = zeros(numel(rho),numThetas);

    %trovo i pixel classificati come edge
    [xIndicies,yIndicies] = find(theImage);

    %Alloco spazio per l'array accumulator
    numEdgePixels = numel(xIndicies);
    accumulator = zeros(numEdgePixels,numThetas);

    %Per aumentare la velocitÃ , alloco spazio per il sine e cosine.
    cosine = (0:width-1)'*cos(theta);
    sine = (0:height-1)'*sin(theta);

    accumulator((1:numEdgePixels),:) = cosine(xIndicies,:) + sine(yIndicies,:);

    %Ciclo sui theta e ottengo i valori rho
    for i = (1:numThetas)
        houghSpace(:,i) = hist(accumulator(:,i),rho);
    end
    pcolor(theta,rho,houghSpace);
    shading flat;
    xlabel('Theta (radians)');
    ylabel('Rho (pixels)');
    colormap('gray');

end
```

project.m

```
function varargout = project(varargin)
% PROJECT MATLAB code for project.fig
%      PROJECT, by itself, creates a new PROJECT or raises the existing
%      singleton*.
%
%      H = PROJECT returns the handle to a new PROJECT or the handle to
%      the existing singleton*.
%
%      PROJECT('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PROJECT.M with the given input arguments.
%
%      PROJECT('Property','Value',...) creates a new PROJECT or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before project_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to project_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help project

% Last Modified by GUIDE v2.5 22-Jul-2017 15:33:25

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @project_OpeningFcn, ...
                   'gui_OutputFcn',  @project_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before project is made visible.
function project_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to project (see VARARGIN)
clc;
% Choose default command line output for project
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes project wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = project_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    global im im2
    [path,user_cance] = imgetfile();
    if user_cance
        msgbox(sprintf('ERROR'),'Error','Error');
        return;
    end
    [im,map] = imread(path);
    axes(handles.axes1);
    imshow(im);
    if(ndims(im) == 3)
        im = rgb2gray(im);
    end
    im2 = im;

% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns listbox1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from listbox1


% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function popupmenu1_Callback(hObject, eventdata, handles)
    global im2
```

```
    val = get(hObject,'Value');
    switch val
        case 3
            im2 = saltpepper(im2,5);
        case 2
            im2 = gaussianNoise(im2);
        case 1
            im2 = im2
        otherwise
    end
        cla(handles.axes2);

    axes(handles.axes2);
    imshow(im2);
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserv - to be defined in a future version of MATLAB
% handles    structur
% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu2 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu2

    global im2
    val = get(hObject,'Value');
    switch val
        case 1
            im2 = im2
        case 2
            im2 = fl_gauss(im2,5);
        case 3
```

```
            im2 = sharp(im2);
        case 4
            im2 = unsharp(im2);
        case 5
            im2 = fl_mediano(im2,3,3);
        case 6
            im2 = fl_mediano_ad2(im2);
        otherwise

    end
        cla(handles.axes2);

    axes(handles.axes2);
    imshow(im2);
% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    global im2
        cla(handles.axes2);

    im2 = svdReduction(im2);
    axes(handles.axes2);
    imshow(im2,[]);

% --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu3 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu3

    global im2 map
    val = get(hObject,'Value');
        cla(handles.axes2);

    switch val
```

```
        case 1
            im2 = im2;
        case 2
            [im2,cols,rows] = Harris(im2,3);
            axes(handles.axes2);
            imshow(im2), hold on,
            plot(cols,rows,'bO');
            hold off;
        case 3
            im2 = tomasi(im2,3);
            axes(handles.axes2);
            imshow(im2,[]);
        case 4
            im2 = canny(im2);
            axes(handles.axes2);
            imshow(im2,[]);
        otherwise
            im2 = im2;
    end

% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1


% --- Executes during object creation, after setting all properties.
function axes2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes2


% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    global im im2
    cla(handles.axes2);
    cla(handles.axes1);
    im2 = im;
%     axes(handles.axes2);
%     imshow(im);
%     im2 = im;


% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    global im
        cla(handles.axes2);

    h=imhist(im);
    axes(handles.axes2);
    plot((1:256),h)


% --- Executes on selection change in popupmenu4.
function popupmenu4_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu4 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu4

    global im im2
    val = get(hObject,'Value');
    switch val
        case 1

        case 2
            [im,map] = imread('cameraman.tif');
        case 3
            [im,map] = imread('football.jpg');
        case 4
            [im,map] = imread('kobi.png');
        case 5
            [im,map] = imread('moon.tif');
        case 6
            [im,map] = imread('text.png');
        case 7
            [im,map] = imread('trees.tif');
        case 8
            [im,map] = imread('yellowlily.jpg');
        otherwise

    end
    axes(handles.axes1);
    imshow(im);
    if(ndims(im) == 3)
        im = rgb2gray(im);
```

```
    end
        im2 = im;
% --- Executes during object creation, after setting all properties.
function popupmenu4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    global im im2
    cla(handles.axes2);
    axes(handles.axes2);
    imshow(im);
    im2 = im;


% --- Executes on button press in pushbutton13.
function pushbutton13_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    global im im2
    im2 = canny(im2);
    axes(handles.axes2); imshow(im2,[]);
    figure(1);
    [R,T,H] = houghTransform(im2,0.01);
    imshow(H,[],'XData',T,'YData',R,...
        'InitialMagnification','fit');
    xlabel('\theta'), ylabel('\rho');
    clear error;
    axis on, axis normal;
```

saltpepper.m

```
function [I] = saltpepper(I,p)
   [num_righe, num_col]=size(I);
   for i=1:num_righe
      for j=1:num_col
         %aggiungi rumore salt and pepper
         s=randi(p);
         if s==p
            if randi(2)==1
               I(i,j)=0;
            else
               I(i,j)=255;
            end
         end
      end
   end
end
```

---

sharp.m

```
function[M,laplAppl] = sharp(I)
   [num_righe, num_col]=size(I);
   s=2;
   t=s;
   I=double(I);
   L=[0,-1,0;-1,4,-1;0,-1,0;];
   L=double(L);
   M=zeros(num_righe,num_col);
   laplAppl=M;
   for i=1:num_righe-s
      for j=1:num_col-t
         laplAppl(i:i+s,j:j+t)=conv2(L,I(i:i+s,j:j+t),'valid');
         M(i:i+s,j:j+t)=I(i:i+s,j:j+t)+conv2(L,I(i:i+s,j:j+t),'valid');
      end
   end
end
```

---

svdReduction.m

```
function [I_temp] = svdReduction(I)
   I=double(I);
   [U,S,V]=svd(I);
   U_ridotta=U(:,1:2);
   S_ridotta=S(1:2,1:2);
   V_ridotta=V(:,1:2);
   I_temp=U_ridotta*S_ridotta*V_ridotta';
   dim=size(I_temp);
end
```

tomasi.m

```
function [I] = tomasi(I,s)
    if(ndims(I) == 3)
        I = rgb2gray(I);
    end

    [num_righe, num_col]=size(I);
    I=double(I);

    I_x=zeros(num_righe,num_col);
    I_y=I_x;
    A=I_y;
    B=A;
    C=B;

    %calcolo I_x e I_y per ogni pixel
    for i=s:num_righe-s
        for j=s:num_col-s
        % maschera del kernel (s+1)*(t+1)
        I_x(i,j)=I(i+1,j)-I(i-1,j);
        I_y(i,j)=I(i,j+1)-I(i,j-1);
        % T=I(i:i+s,j:j+t);
        % M(i:i+s,j:j+t)=A;
        end
    end

    %calcolo A(x,y)
    for i=s:num_righe-s
        for j=s:num_col-s
        % maschera del kernel (s+1)*(t+1)
            for ii=i:i+s
                for jj=j:j+s
                    A(i,j)=A(i,j)+I_x(ii,jj)^2;
                end
            end

        end
    end

    %calcolo B(x,y)
    for i=s:num_righe-s
        for j=s:num_col-s
            % maschera del kernel (s+1)*(t+1)
            for ii=i:i+s
                for jj=j:j+s
                    B(i,j)=B(i,j)+I_y(ii,jj)^2;
                end
            end

        end
    end

    %calcolo C(x,y)

    for i=s:num_righe-s
        for j=s:num_col-s
```

```
            % maschera del kernel (s+1)*(t+1)
            C(i,j)=C(i,j)+A(i,j)*B(i,j);

        end
    end

    k=10^7;
    hold on
    for rr=1:num_righe
        for cc=1:num_col
            M=[A(rr,cc),C(rr,cc);
            C(rr,cc),B(rr,cc);];
            lambda=eig(M);
            ind_lambda_div_0=find(lambda>0);
            lambda_new=zeros(length(ind_lambda_div_0)-1);
            for kk=1:length(ind_lambda_div_0)
                lambda_new(kk)=lambda(ind_lambda_div_0(kk));
            end
            if isempty(ind_lambda_div_0)
                lambda_min=0;
            else
                lambda_min=min(lambda_new);
            end
            if lambda_min>=k
                I(rr,cc)=64;
            end
        end
    end


end
```

---

unsharp.m

```
function [ fin ] = unsharp(I)
    s=2;
    t=s;
    [num_righe, num_col]=size(I);
    M=I;
    L=[1,2,1;1,-8,1;1,2,1;];
    for i=1:num_righe-s
        for j=1:num_col-t
            %maschera del kernel (s+1)*(t+1)
            T=I(i:i+s,j:j+t);
            [m,n]=size(T);
            %filtro media per fare il blur
            media=sum(T(:))/(m*n);
            M(i:i+s,j:j+t)=conv2(L,media,'valid');
        end
    end
    sottr=(I-M);
    fin=sottr+I;
end
```

6 - Bibliografia

- Lucidi del corso di “Visione Computazionale” del prof. Francesco Isgrò;
- Gonzalez – Woods. *Digital Image Processing* 3°edizione;
- Nixon – Aguado. *Feature Extraction & Image Processing* 2° edizione;
- Szeliski. *Computer Vision;*