UNIVERSITA' DEGLI STUDI DI BRESCIA

Facoltà di Ingegneria
Corso di laurea in Ingegneria Elettronica

PROGETTO D'ESAME per il corso di
Robotica - A.A. 1999/2000

IMAGE HISTOGRAM PROCESSING: EQUALIZATION, STRETCHING, NON-LINEAR EQUALIZATION, GAMMA FUNCTION

Pialorsi Paolo - matricola 27126

Indice degli argomenti

Indice delle figure

Introduzione

Spesso le immagini acquisite al calcolatore sono in stato tale da avere una distribuzione della gamma cromatica non uniforme e tale quindi da risultare troppo chiare o troppo scure in termini di luminosità. Questi difetti di acquisizione sono facilmente osservabili esaminando gli istogrammi delle immagini.

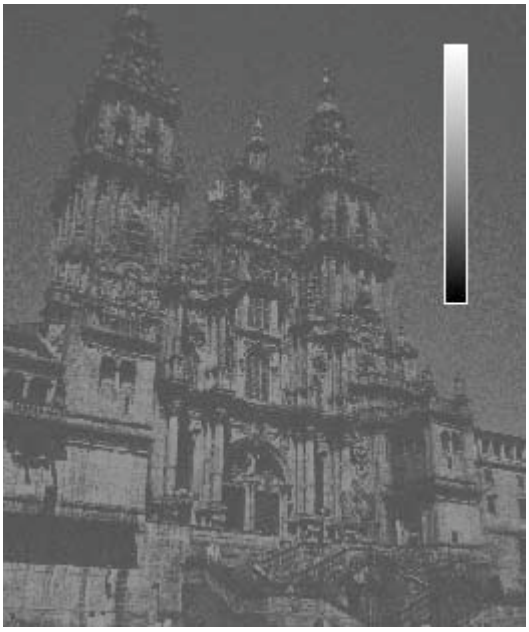


Figura 1: Immagine con istogramma compresso.

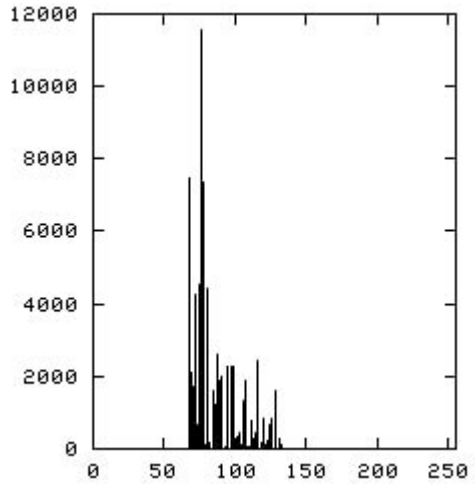


Figura 2: Istogramma dell'immagine (notare la compressione del grafico nell'intervallo 75-125).

Per ovviare a questo tipo di inconvenienti - specialmente quando si trattano immagini da elaborare ed interpretare nell'ambito della computer vision e della robotica - ci si appoggia a funzioni matematiche appositamente elaborate.
Scopo di questo elaborato è lo studio e l'applicazione di alcune tecniche di correzione delle immagini al fine di ricavare delle indicazioni utili per la scelta della giusta metodologia di intervento.

Le funzioni analizzate ed approfondite a tale scopo sono le seguenti:

- histogram stretching
- histogram linear equalization
- histogram non-linear equalization
- histogram gamma-function equalization

L'elaborato nello specifico è stato così svolto:

- ricerca delle informazioni e della documentazione disponibile sull'argomento
- ricerca e valutazione degli algoritmi disponibili per svolgere i compiti scelti
- elaborazione di un programma che eseguisse gli algoritmi scelti

- verifica delle funzionalità confrontando i risultati ottenuti dal programma autonomamente sviluppato con quelli forniti da altri software di elaborazione delle immagini
- stesura della presente relazione

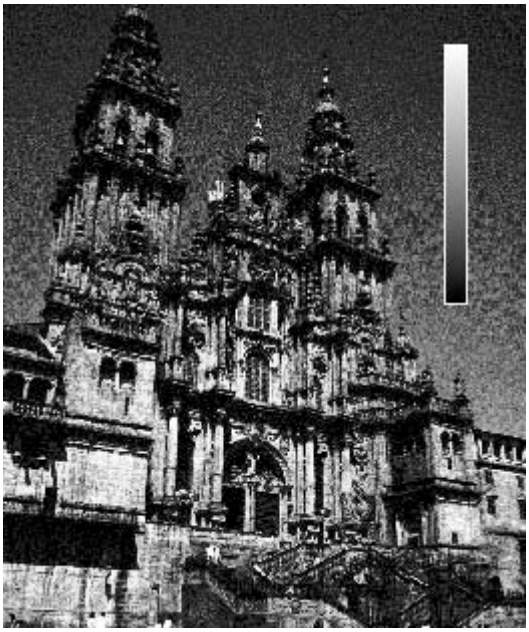


Figura 3: Immagine con istogramma rielaborato tramite histogram-stretching.

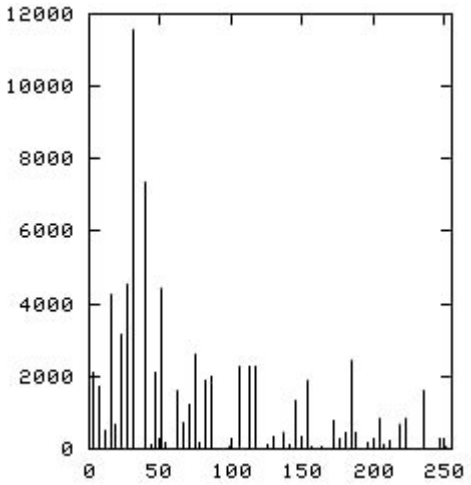


Figura 4: Istogramma dell'immagine elaborata (l'istogramma si estende sull'intero asse X delle scale di grigio).

Istogramma di un'immagine

Data un'immagine si definisce istogramma la funzione che restituisce la frequenza di occorrenze di ogni colore o scala di grigio dell'immagine stessa. I colori o le scale di grigio sono quantizzati da 0 a *n*. Nel corso dell'elaborato ci si è concentrati sullo studio delle immagini in **256 scale di grigio** (da 0 a 255). Quindi il valore dell'istogramma in un particolare punto *p* della scala di grigio, detto *h(p)* è dato dal numero di punti dell'immagine che presentano quel colore fratto il numero complessivo di punti presenti nell'immagine.

$$h(p) = [\text{Numero occorrenze di } p] / [\text{Numero Pixel}]$$

L' istogramma è una funzione utile in moltissime situazioni differenti. Nel caso di questo elaborato sarà utilizzata per rilocare la distribuzione dei colori nelle immagini ed ottenere quindi delle visualizzazioni migliori delle stesse.
La funzione istogramma è rappresentabile su di un grafico bidimensionale come il seguente:

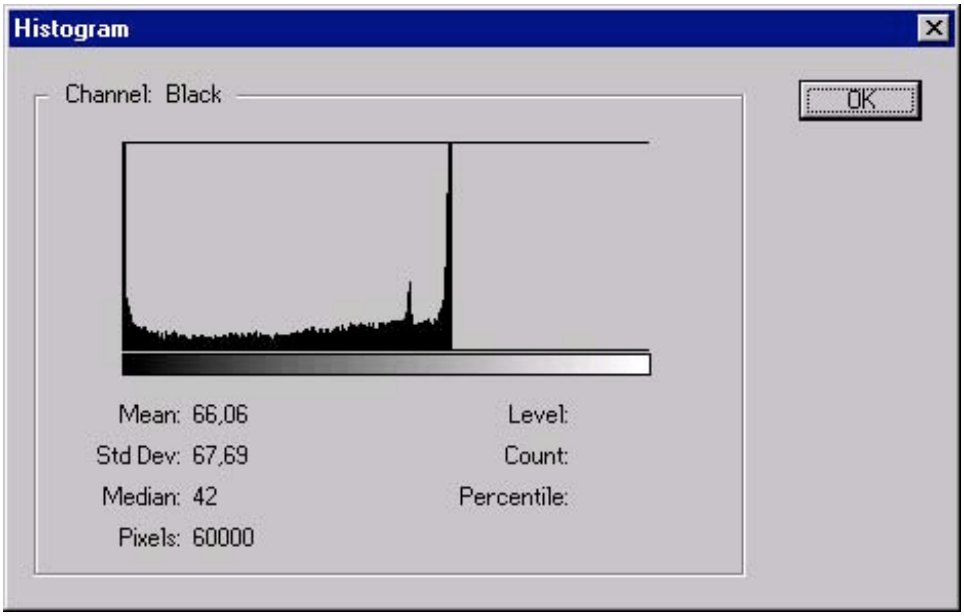


Figura 5: Istogramma di una immagine in scale di grigio.

Si riporta di seguito l'algoritmo utilizzato per determinare i punti della funzione istogramma di un'immagine in scale di grigio. Come si nota dal codice sorgente riportato l'algoritmo scorre tutti i punti dell'immagine e per ciascuno di essi va ad incrementare all'interno di un array di 256 scale di grigio il valore della variabile che conta le occorrenze del colore rappresentato. Infine tutti i contatori di occorrenza di un colore vengono divisi per il numero totale di pixel dell'immagine.

```
void Histogram(struct Image *ImgInput, float HIST[])
{
int X, Y, I, J;
long int IHIST[256], SUM;
for(I=0;I<=255;I++) IHIST[I] = 0;
SUM=0;
for(Y=0; Y<ImgInput->Rows; Y++)
      {
            for (X=0; X<ImgInput->Cols; X++)
            {
            J = *(ImgInput->Data+X+(long)Y*ImgInput->Cols);
            IHIST[J] = IHIST[J] + 1;
            SUM = SUM +1;
            }
      }
for (I=0;I<=255;I++)
      {
      HIST[I] = (float)IHIST[I]/(float)SUM;
      }
}
```

Histogram Stretching

Per histogram stretching si intende la redistribuzione uniforme dell'istogramma di un'immagine in modo tale da far sì che l'immagine occupi l'intera gamma di colori a disposizione (da 0 a 255 nel nostro caso).
L'algoritmo che ne risulta è di seguito riportato.

```
void StretchHistogram(struct Image *ImgInput, struct Image *ImgOutput)
{
//Variabili di appoggio
int X, Y, I;
int valLeft, valRight, valStretched;
float valScaleFactor, valMedium;
float HIST[256];

//Leggo l'istogramma della funzione
Histogram(ImgInput, HIST);

//Leggo il limite sinistro
valLeft = 0;
for(I=0; I<=255; I++)
      if (HIST[I] > 0)
            {
            valLeft = I;
            break;
            }

//Leggo il limite destro
valRight = 255;
for(I=255; I>0; I--)
      if (HIST[I] > 0)
            {
            valRight = I;
            break;
            }

//Calcolo il valore mediano
valMedium = (float)(valRight - valLeft);
valScaleFactor = (255 / valMedium);

//Elaboro l'immagine
for(Y=0; Y<ImgInput->Rows; Y++)
      {
      for(X=0; X<ImgInput->Cols; X++)
            {
            valStretched = (int) *(ImgInput->Data+X+(long)Y*ImgInput->Cols);
            valStretched = (int) (valScaleFactor * (valStretched - valLeft));
            if (valStretched < 0) valStretched = 0;
            if (valStretched > 255) valStretched = 255;

            //Salvo il valore nell'immagine di output
            *(ImgOutput->Data+X+(long)Y*ImgInput->Cols) = valStretched;
            }
      }
}
```

L’algoritmo calcola il colore minimo (sinistro) e massimo (destro) utilizzati dall’immagine all’interno del grafico dell’istogramma, ne calcola il valore mediano e quindi ricava un fattore di scala per il quale saranno moltiplicati tutti i punti dell’immagine.

L’algoritmo è stato applicato a diverse immagini denotando un rilevante miglioramento della qualità. Come modello per l’intera relazione verrà considerata l’immagine seguente.

Figura 6: Immagine di pessima qualità usata come modello nell’arco dell’intero elaborato.

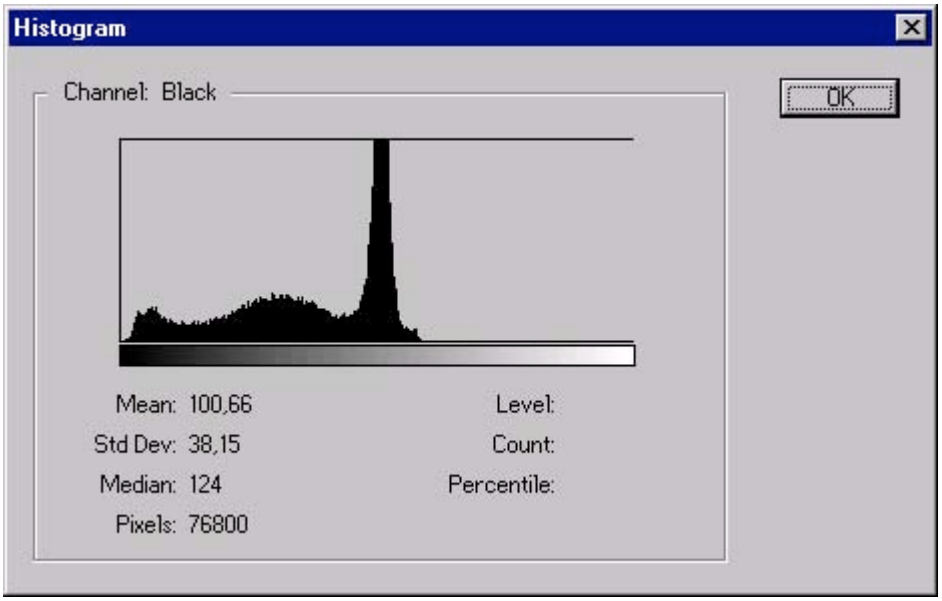


Figura 7: Istogramma dell’immagine usata come modello nell’arco dell’intero elaborato.

L’immagine elaborata con l’algoritmo di histogram stretching si presenta come segue.

Figura 8: Immagine modello rielaborata con algoritmo histogram stretching.

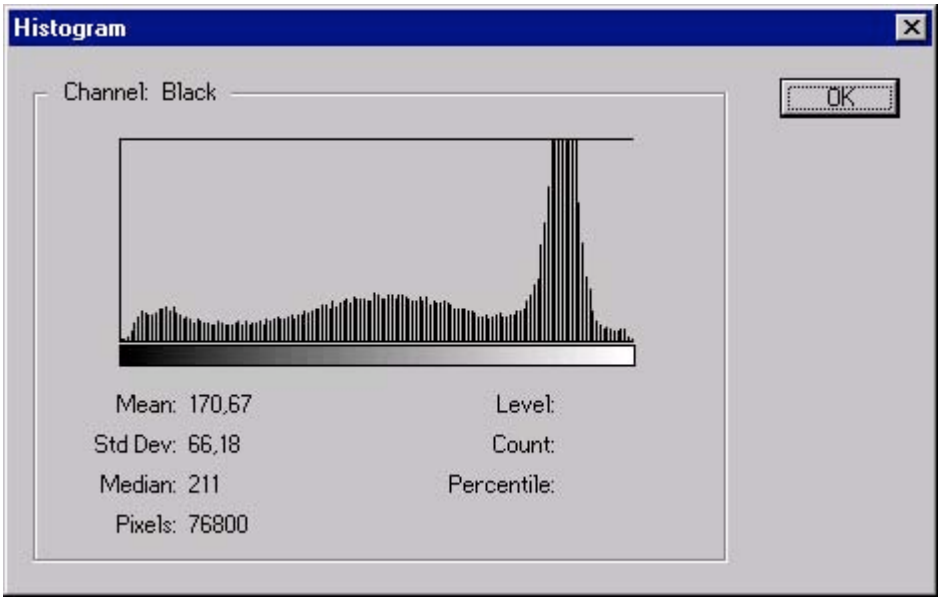


Figura 9: Istogramma dell’immagine modello dopo l’applicazione dell’algoritmo histogram stretching.

È interessante notare come il grafico della funzione istogramma dell’immagine sia variato, pur senza perdere informazioni relative all’immagine ma anzi guadagnando della nuova informazione. Osservando il grafico dell’istogramma è altresì evidente da cosa derivi il nome dell’algoritmo applicato, infatti il grafico è stato “stirato” lungo l’asse delle X (cioè delle scale di grigio).
È importante svolgere alcune considerazioni emerse dall’utilizzo di questo algoritmo: questa tecnica di elaborazione risulta molto utile nel caso sia applicata ad immagini con un istogramma molto compresso in uno specifico intorno di colori. Immagini con una pessima distribuzione del colore ma con una occupazione pressoché completa

della scala di grigi purtroppo risentono di questo e quindi la loro rielaborazione rimane scadente come qualità.

Histogram Equalization lineare

L'equalizzazione lineare di un'immagine si ottiene in modo simile allo stretching della stessa. L'idea alla base dell'algoritmo è quella di far sì che tutti i colori *p* abbiano un valore *h(p)* il più possibile uniforme. La funzione lineare da applicare è la seguente:

$$g_i = (M - 1) / n_t * \sum_{j=0 \rightarrow i} (n_i)$$

dove n_t é il numero totale di punti che costituiscono l'immagine, n_i è il numero di punti che hanno la scala di grigio *i* ed *M* è il numero totale di scale di grigio (per noi 256).
Al fine di ottenere questo risultato l'algoritmo che è stato applicato è il seguente.

```
void EqualizeHistogram(struct Image *ImgInput, struct Image *ImgOutput)
{
int X, Y, I, J;
int HISTEQ[256];
float HIST[256], SUM;
Histogram(ImgInput, HIST);
for(I=0; I<=255; I++)
      {
      SUM = 0.0;
      for(J=0; J<=I; J++)
            SUM += HIST[J];
      HISTEQ[I] = (int)(255*SUM+.5);
      }
for(Y=0; Y<ImgInput->Rows; Y++)
      {
      for(X=0; X<ImgInput->Cols; X++)
            {
            *(ImgOutput->Data+X+(long)Y*ImgInput->Cols) =
                  HISTEQ[*(ImgInput->Data+X+(long)Y*ImgInput->Cols)];
            }
      }
}
```

Il risultato è mostrato in Figura 10 e l'istogramma del risultato in Figura 11.
Si può notare dall'istogramma che – a parte i colori evidentemente e marcatamente troppo forti in termini di presenza – i colori sono stati ridistribuiti uniformemente in termini di presenza nell'immagine.

Figura 10: Immagine modello rielaborata con algoritmo histogram equalization.

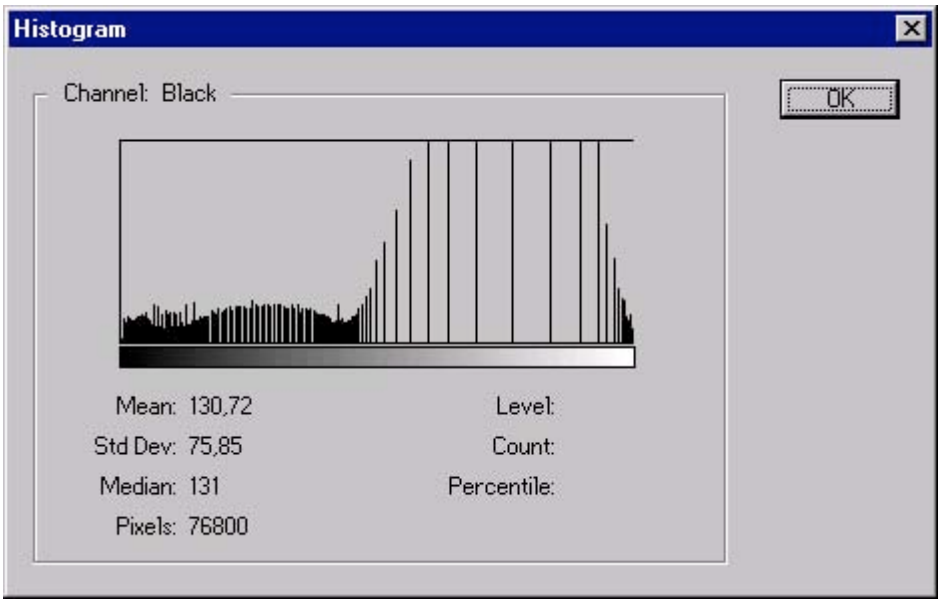


Figura 11: Istogramma dell'immagine modello dopo l'applicazione dell'algoritmo histogram equalization.

Questa tecnica – contrariamente all'histogram stretching fornisce buoni risultati in qualsiasi condizione. Una sovrapposizione di effetti dovuta all'applicazione di histogram stretching e histogram equalization alla stessa immagine ha dimostrato che ciò non porta vantaggi significativi.
Si sottolinea ancora inoltre il fatto che l'histogram equalization, contrariamente all'histogram stretching non richiede una funzione istogramma di ingresso molto compressa.

Histogram Equalization non lineare

L'algoritmo alla base di questo tipo di conversione è di per sé banale. Si tratta di un'associazione uno a molti (lookup table) tra una singola scala di grigio di un punto ed il nuovo colore grigio che tale immagine deve assumere e si può basare su diverse tipologie di funzione.
Tra le varie funzioni non lineari che possono dare origine ad una lookup table la migliore tra quelle sperimentate è risultata essere la funzione logaritmica diretta. La funzione logaritmica inversa invece lavora molto bene su immagini molto chiare da scurire.

L'algoritmo che è stato applicato è il seguente.

```
void ShapeHistogram(struct Image *ImgInput, struct Image *ImgOutput, int LUT)
{

int X, Y;
for(Y=0; Y<ImgInput->Rows; Y++)
      for(X=0; X<ImgInput->Cols; X++)
            {
            if (LUT == 1)
                        *(ImgOutput->Data + X + (long)Y * ImgOutput->Cols) =
LOG_LOOKUP[*(ImgInput->Data + X + (long)Y * ImgInput->Cols)];
            else if (LUT == 2)
                        //Prendo il nuovo valore e sposto il puntatore della Lo-
op-Up Table
                        *(ImgOutput->Data + X + (long)Y * ImgOutput->Cols) =
INV_LOG_LOOKUP[*(ImgInput->Data + X + (long)Y * ImgInput->Cols)];
      }
}
```

L'algoritmo alla base della trasformazione scorre tutti i colori dell'immagine e quando individua un colore gestito vi scrive il risultato della funzione di equalizzazione non lineare.
Il risultato di questa forma di elaborazione non si è dimostrato tra i migliori dopo le prove ed i test effettuati.
In particolare si nota una sempre forte presenza di toni di colore chiari.
Il risultato dell'algoritmo applicato all'immagine di partenza fornisce il seguente risultato.

Figura 12: Immagine modello rielaborata con algoritmo histogram equalization.

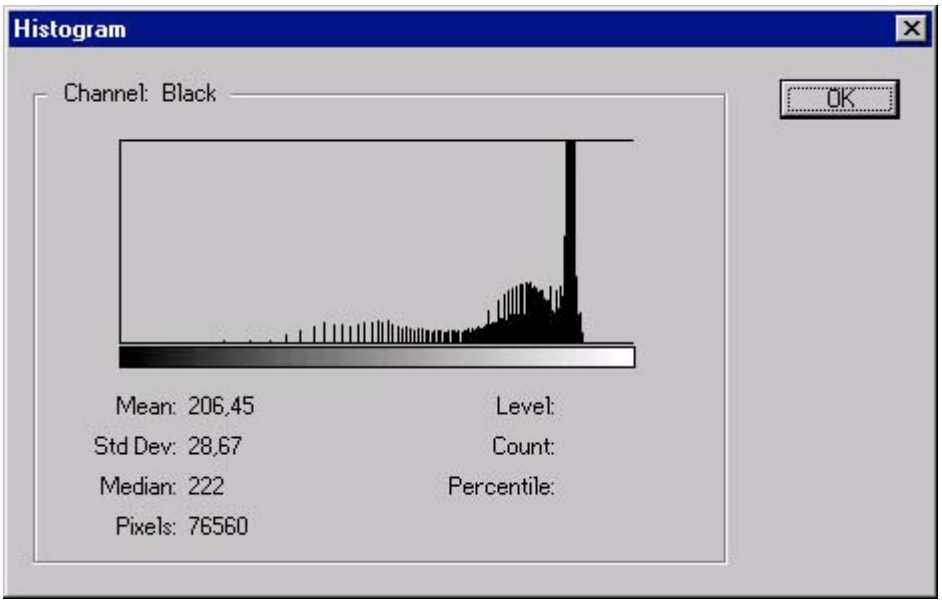


Figura 13: Istogramma dell'immagine modello dopo l'applicazione dell'algoritmo histogram equalization.

Come si può vedere l'algoritmo in questo caso non ha ridistribuito i valori delle scale di grigio, ma li ha semplicemente spostati verso la parte più chiara dell'immagine. Per questo stesso motivo l'histogram equalization non lineare basata su lookup table logaritmiche inverse lavora bene su immagini molto chiare (in quanto le sposta nella zona dei neri) ma fornisce un risultato pessimo se applicato ad immagini con una bassa presenza di toni di grigio chiari.

Gamma Function

Questo ultimo algoritmo sperimentato è una particolare forma di equalizzazione non lineare basata sulla gamma function. Questa funzione trasforma l'intensità di luminosità lineare di un'immagine in un segnale non lineare.
Nel settore dell'elaborazione delle immagini il simbolo gamma _ rappresenta un parametro numerico che descrive la non linearità dell'intesità di riproduzione dell'immagine. Per intensità si intende l'energia trasferita per unità di area.
L'intensità di luce generata da un dispositivo fisico (TV, CRT, LCD, ecc.) generalmente non è rappresentabile con una funzione lineare dei valori in ingresso. Per esempio un visore CRT ha una funzione di risposta al segnale in ingresso pari ad una esponenziale con esponente 2.5 . Il valore 2.5 si definisce _. Questa non linearità deve essere compensata al fine di avere buoni risultati di visualizzazione.
La gamma correction è una tecnica per compensare le diverse non-linearità di un'immagine, generalmente dovute a come i monitor convertono il voltaggio RGB in intesità di luminosità.

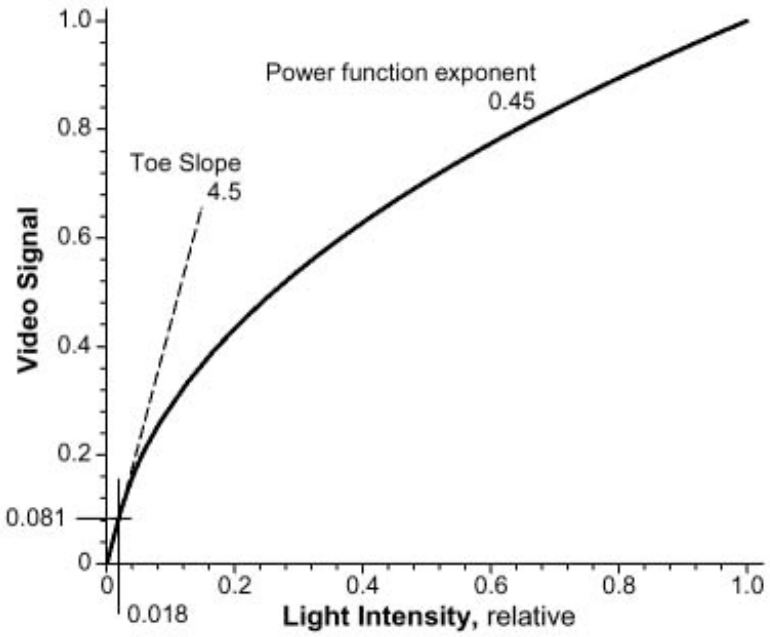


Figura 14: Grafico della funzione esponenziale usata nella gamma function

Con il nome gamma function si definisce la funzione che rappresenta il valore numerico dell'esponenziale della funzione potenza di risposta al voltaggio elevato alla _.
Questa tecnica di correzione delle immagini nasce dalla necessità di avere immagini chiare e leggibili su tutti i dispositivi di visualizzazione presenti sul mercato (TV, teCRT, LCD, ecc.) indipendentemente dalla fonte di origine delle informazioni.
La gamma correction permette tramite appositi valori del parametro gamma di ottenere risultati soddisfacenti in termini di qualità di visualizzazione su diversi apparati.

Tale funzione però per la sua natura si rende utile anche nell'ambito della nostra ricerca in quanto ci permette, agendo sul parametro _, di ottenere diverse visualizzazioni delle immagini di partenza permettendoci di intervenire sul contrasto delle stesse mostrando così eventuali informazioni significative altrimenti nascoste.

L'algoritmo che è stato applicato è il seguente.

```
void GammaHistogram(struct Image *ImgInput, struct Image *ImgOutput)
{
int i;
int X, Y;

int GAMMA_LOOKUP[256];
double gamma;

/* Genero una lookup table per la
gamma correction con esponente gamma = 2.2 */
gamma = 2.2;
GAMMA_LOOKUP[0] = 0;

for (i = 0;  i <= 255;  i++)
      GAMMA_LOOKUP[i] = (int) (pow(i/255.0, 1/gamma) * 255 + 0.5);

/* Applico la Gamma LUT all'immagine */
for(Y=0; Y<ImgInput->Rows; Y++)
      for(X=0; X<ImgInput->Cols; X++)
            *(ImgOutput->Data   +   X   +   (long)Y   *   ImgOutput->Cols)   =
GAMMA_LOOKUP[*(ImgInput->Data + X + (long)Y * ImgInput->Cols)];
}
```

L'algoritmo di trasformazione si basa, come per l'equalizzazione non lineare, su una lookup table dalla quale attinge il nuovo colore da attribuire a ciascun pixel dell'immagine originale.
Nell'applicativo sviluppato si è utilizzato un valore di gamma pari a 2.2 in quanto dalle ricerche svolte esso è risultato essere il valore che su sistemi PC fornisce il miglio risultato in termini di qualità dell'immagine.

Nella pagina seguente è riportato il risultato dell'elaborazione sull'immagine di partenza.

Figura 14: Immagine modello rielaborata con l'algoritmo gamma function.

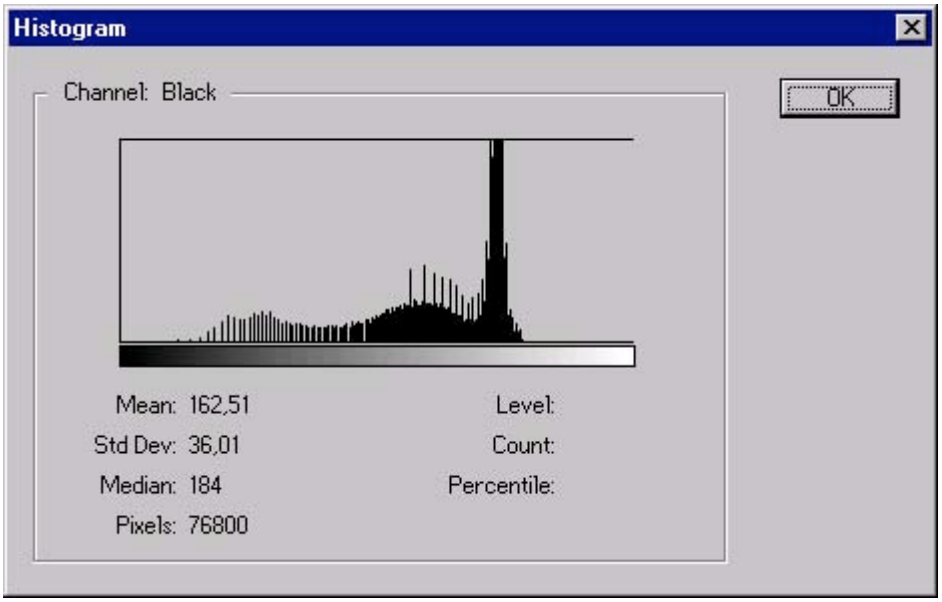


Figura 15: Istogramma dell'immagine modello dopo l'applicazione dell'algoritmo gamma function.

Considerazioni finali

Le prove svolte dimostrano che l'algoritmo migliore tra quelli esaminati risulta essere quello di histogram stretching, laddove l'immagine presenti un istogramma compresso. In tutti i casi in cui invece l'istogramma risulti non compresso può convenire utilizzare la funzione di histogram equalization.
Gli algoritmi non lineari sono di difficile applicazione in programmi di correzione automatica delle immagini in quanto sarebbe necessario individuare delle tecniche di selezione della lookup table (per l'equalizzazione non lineare) piuttosto che dell'esponente della gamma function al fine di ottenere i migliori risultati possibili.

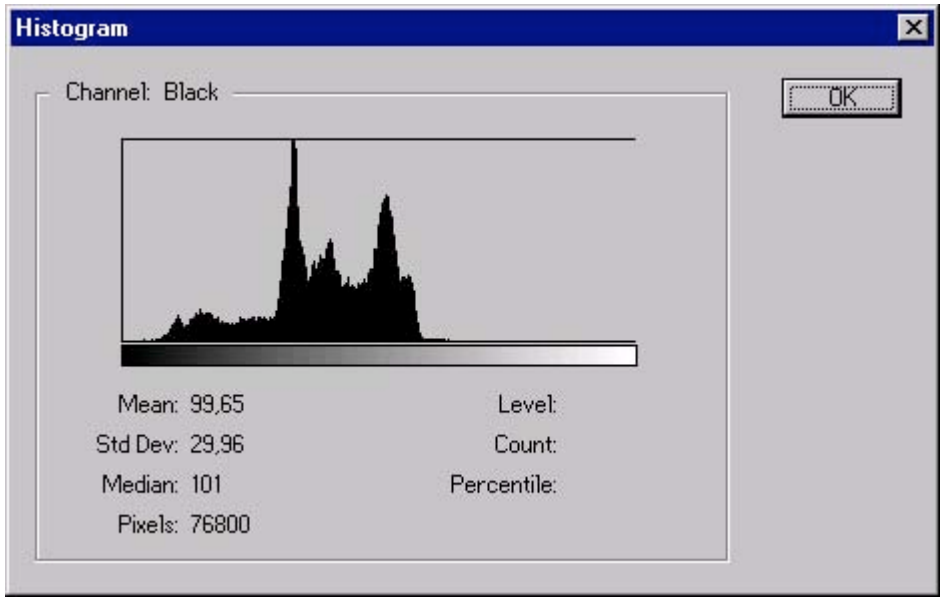
Nella Appendice 1 si riportano come ulteriori esempi alcune altre immagini elaborate con il software di prova prodotto.

Il software è documentato nella Appendice 2 del presente elaborato.

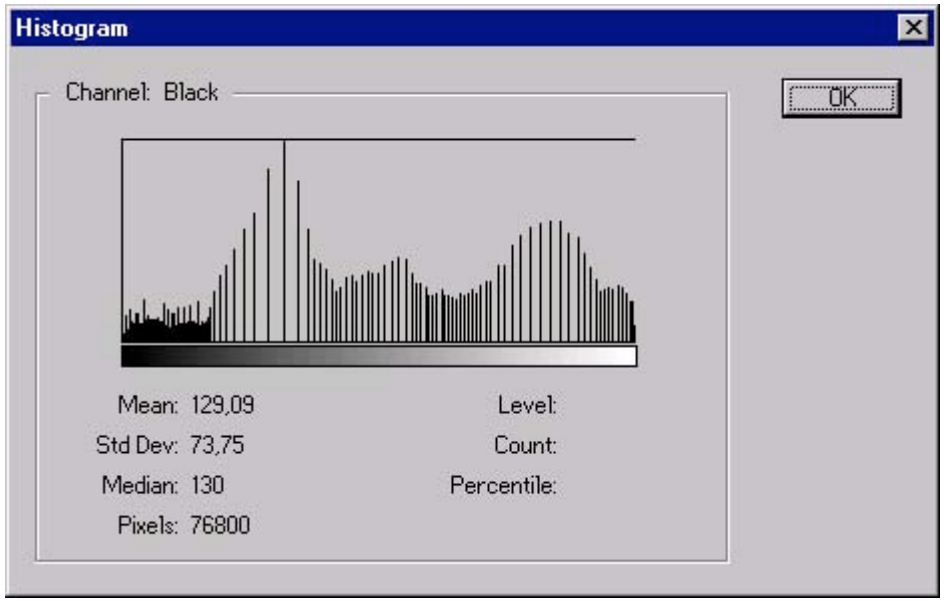
Appendice 1

Altre Immagini di Esempio

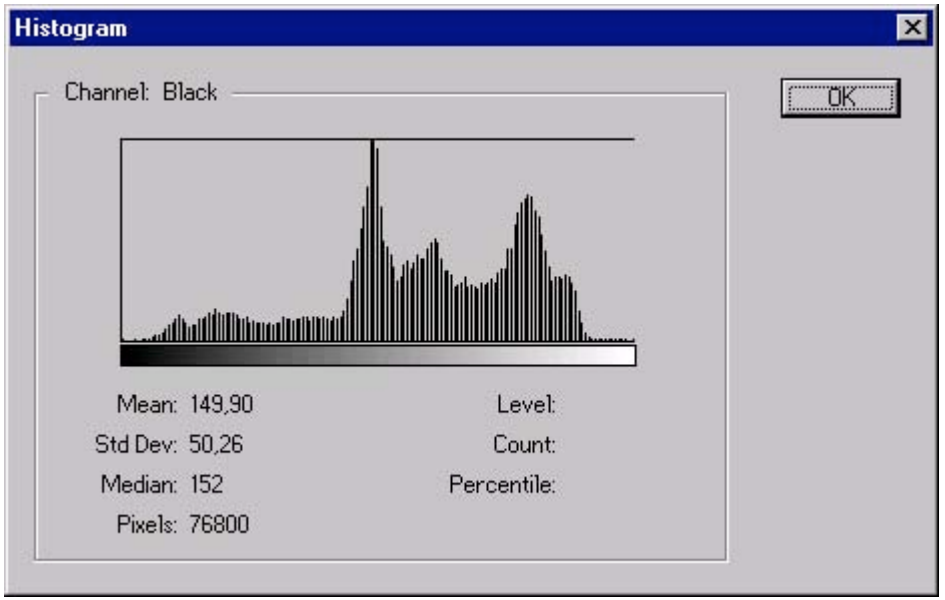
Immagine Originale



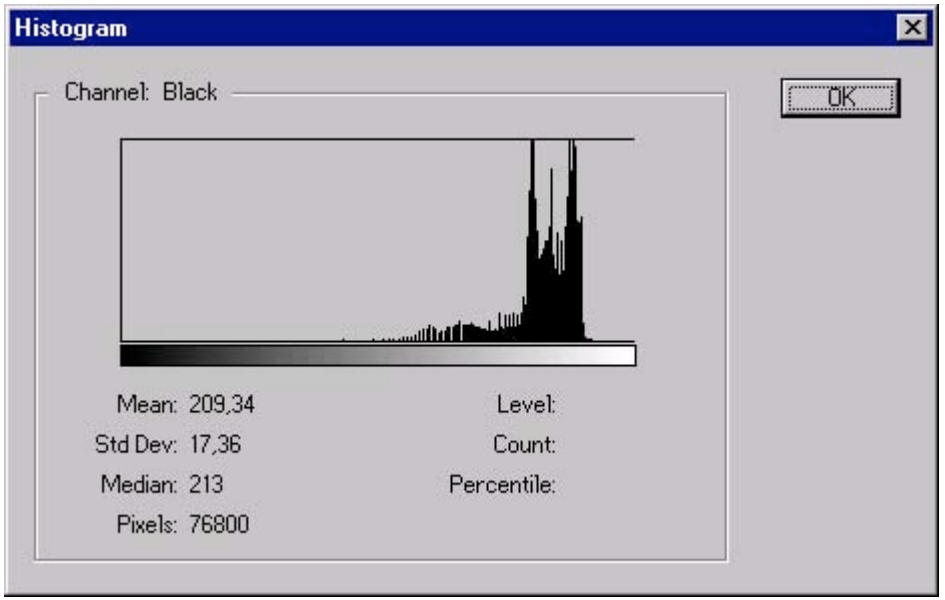
Histogram Stretching



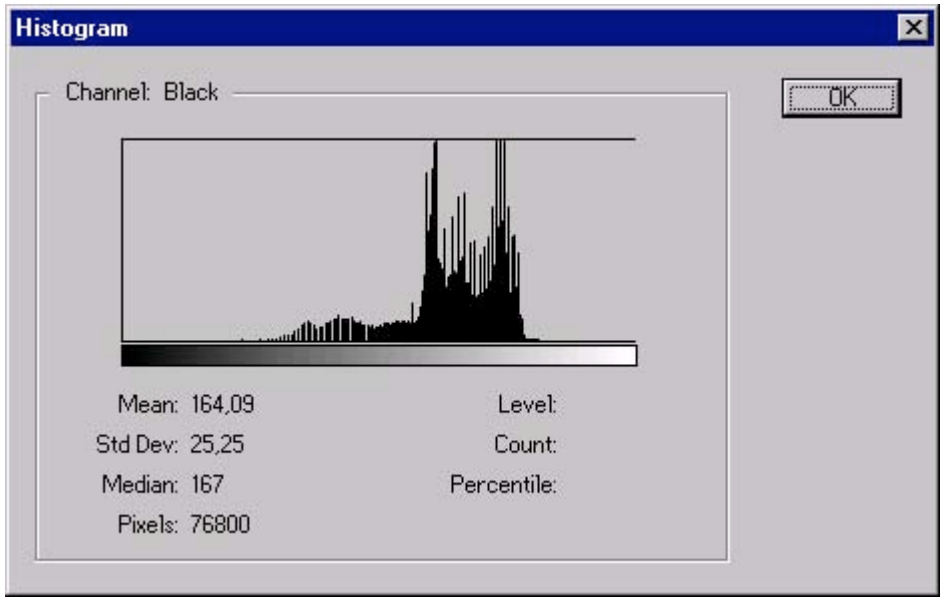
Linear Histogram Equalization



Non-Linear Histogram Equalization



Gamma Function



Appendice 2

Manuale utente del Software

ImageHFilter 1.0

L'applicazione è realizzata come un applicativo di elaborazione immagini che non interagisce graficamente con l'utente ma che accetta da riga di comando le informazioni necessarie ad operare.
È sufficiente avviare il software (HistogramFilters.exe) per avere informazioni sulle opzioni disponbili a riga di comando.

```
Elaboratore testuale di immagini in scale di grigio.

Realizzato da Paolo Pialorsi (Mat. 027126 - paolo@pialorsi.com)
nell'ambito dell'esame di Robotica presso la Facolta'
di Ingegneria dell'Universita' degli Studi di Brescia.

Docente: Cassinis Dott. Riccardo
Assistente: Rizzi Dott. Alessandro
Anno Accademico 1999/2000

Riga di comando:

ImageHFilter.EXE [FileInPut] [FileOutPut] [Filter]

Dove:
[FileInput]  => file di immagine in input (deve essere un file PPM)
[FileOutPut] => file di immagine in output LinearHFilter (è un file PPM)
[Filter] => tipo di filtro da applicare
            1 - Histogram Stretching lineare
            2 - Equalizzatore lineare di istogramma
            3 - Equalizzatore non lineare generico di istogramma LUT Logaritmica
            4 - Equalizzatore non lineare generico di istogramma LUT Log-Inversa
            5 - Equalizzatore non lineare di istogramma LUT Gamma Function (gamma 2.2)
```

Quindi per esempio se si vuole applicare il filtro di histogram stretching all'immagine C:\PROVA.PPM occorre eseguire il programma nel modo seguente:

```
ImageHFilter.EXE C:\PROVA.PPM  C:\PROVAOUT.PPM  1
```

Se invece si vuole applicare il filtro di histogram equalization con Gamma function all'immagine C:\PROVA.PPM occorre eseguire il programma nel modo seguente:

```
ImageHFilter.EXE C:\PROVA.PPM C:\PROVAOUT.PPM 5
```

Il codice sorgente dell'applicativo è prelevabile alla URL:

http://www.pialorsi.it/paolo/IHP/software.ZIP

L'applicazione è stata sviluppata utilizzando il tool di sviluppo Microsoft Visual C++ 6.0 . Si è scelto di utilizzare questo strumento per realizzare un'applicazione console Win32, quindi eseguibile da un qualsiasi ambiente Microsoft Windows all'interno di una shell con interfaccia a caratteri. Il software si appoggia comunque soltanto al linguaggio C senza fare uso di tecniche di programmazione a oggetti specifiche di Microsoft Visual C++ 6.0 (MFC, ATL) e può quindi essere facilmente portato in altri ambienti (Unix, Linux, ecc.).

Bibliografia

- The Pocket HandBook of Image Processing – Nykler & Weeks – Ed. P.T.F.
- Abstract: A visibility matching tone reproduction – Operator for high dynamic range scenes – Ward Larson, Rushmeier & Oiatko
- Frequently asked questions about Gamma – Poyton – Ed. Python
- DSP Benchmarks - http://www.eecg.toronto.edu/~wongca/Work/
- 65K Mathematical programming, optimization and variational techniques - http://www.math.niu.edu/~rusin/known-math/index/65KXX.html
- GRASS GIS source code page - ftp://ftp.digital.com.au/pub/grass421/main_source.html
- Histogram Equaization - http://palgong.kyungpook.ac.kr/~galce/chap3_7.htm
- Histogram Equalization Functions - http://medx.sensor.com/www/equalize.html
- Lab Histogram Stretching Contrast Enhancement - http://sevilleta.unm.edu/~bmilne/khoros/html-dip/c4/s4/node2.html
- Early Processing –
- Bynari image processing –
- Poynton Color Tecniques - http://www.inforamp.net/~poynton/notes/links/color-links.html
- The rehabilitation of gamma - www.inforamp.net/~poynton/papers/IST_SPIE_9801/index.html
- Gamma-Lookup by Adam M. Costello - amc@cs.berkeley.edu