Elaborato di Robotica

Algoritmi di costanza cromatica come ausilio alla segmentazione di immagini

Docente: Prof. Riccardo Cassinis
Autore: Roberto Finazzi
Matr. 022276

Data: 31/05/2000

INDICE

1. Obiettivo

Lo scopo di questo elaborato è quello di implementare e testare alcuni algoritmi di costanza cromatica, detti di *color constancy*, utili per togliere da una immagine eventuali dominanti cromatiche dovute all'illuminazione della scena che, se non trattate, renderebbero il risultato della segmentazione dipendente dalla particolare illuminante utilizzata restringendo la dinamica dell'immagine.
Tutti gli algoritmi proposti sono caratterizzati da una buona velocità di esecuzione e questo li rende particolarmente adatti al pre-processing real time dell'immagine RGB acquisita da telecamera nel caso di applicazioni di robotica autonoma.
Sono stati testati l'algoritmo di G.Buchsbaum [Buck], detto *grey world*, quello basato sulla normalizzazione del bianco [Funt], detto *white patch*, e quello di G.D.Finlayson [Fin], detto *gamut match*.
Tutti questi verranno descritti in dettaglio nelle prossime sezioni.
Per finire è stata valutata la possibilità di applicare in cascata tali filtri. In particolare è stata testata la combinazione che si è rivelata la più promettente, consistente nell'applicazione successiva del filtro *white patch* seguito dal filtro *grey world*.
Per tutti i summenzionati algoritmi sono stati realizzati altrettanti programmi, scritti in linguaggio C e operanti in ambiente Linux, in grado di leggere un'immagine in formato TIFF e generare in uscita la relativa immagine filtrata.

2. Algoritmo di *grey world*

2.1 Descrizione

Questo algoritmo, dovuto a G.Buchsbaum [Buck], si basa sull'assunzione che effettuando la media dei colori di una scena si ottenga sempre il colore grigio e che ogni deviazione dal grigio sia dovuta esclusivamente al particolare colore dell'illuminante utilizzata.
Come è risaputo il grigio nello spazio RGB è caratterizzato dal fatto di avere le tre componenti (rossa, verde e blu) uguali.
Quindi l'algoritmo è semplice e si compone dei seguenti passi:

- Scansione dell'immagine e calcolo della media delle componenti RGB per tutti i pixel.
- Calcolo del valore medio di grigio che dovrà avere l'immagine filtrata. In prima battuta viene calcolato come media delle medie delle tre componenti RGB calcolate al punto precedente, poi si riscala in modo che non si abbiano overflow sulla scala tonale delle tre componenti.
- Calcolo dei coefficienti di correzione da applicare a ciascuna componente cromatica di ciascun pixel affinchè si ottenga il grigio medio calcolato in precedenza. Detto *grey* il grigio voluto, *media_red, media_green* e *media_blue* le medie RGB dell'immagine di partenza, *Rin, Gin* e *Bin* le tre componenti di un generico pixel dell'immagine in ingresso, *Rout, Gout* e *Bout* le tre componenti del corrispondente pixel dell'immagine d'uscita, si ha:

  *Rout = Rin * grey / media_red*
  *Gout = Gin * grey / media_green*
  *Bout = Bin * grey / media_blue*

- Generazione del file in uscita dell'immagine filtrata in formato TIFF.

2.2 Uso del programma grey_world_filtro

La sintassi è la seguente:

./grey_world_filtro nome_file_input.tif nome_file_output.tif [grigio]

Il parametro opzionale "*grigio*" permette di impostare direttamente il valore di grigio medio voluto per l'immagine di uscita. Se non specificato verrà calcolato automaticamente come media delle componenti RGB dell'immagine nel suo complesso.
Il tempo impiegato per l'elaborazione di una immagine da 320 x 210 pixel su un PC K6@300Mhz, ammettendo di operare su strutture dati in memoria statica, è di circa 2 ms.

3. Algoritmo di *white patch*

3.1 Descrizione

Questo algoritmo [Funt] scala indipendentemente ciascun canale dell'immagine in ingresso (R, G e B) per il pixel di valore massimo trovato in ciascun canale. Questo equivale a stimare il colore dell'illuminante come il colore dato dalla tripletta <*Rmax, Gmax, Bmax*> che sono, rispettivamente, i valori massimi trovati per le componenti R, G e B.
L'algoritmo si compone dei seguenti passi:

- Scansione dell'immagine per trovare il massimo delle componenti per ciascun canale (che chiameremo *Rmax, Gmax* e *Bmax*).
- Calcolo dei coefficienti di correzione da applicare a ciascuna componente cromatica di ciascun pixel in modo tale che il bianco nell'immagine d'uscita corrisponda all'ipotetico pixel di componenti <*Rmax, Gmax, Bmax*> nell'immagine in ingresso. Dette *Rin, Gin* e *Bin* le tre componenti di un generico pixel dell'immagine in ingresso, *Rout, Gout* e *Bout* le tre componenti del corrispondente pixel dell'immagine d'uscita, e *WHITE* la componente del colore bianco per i canali RGB (che corrisponde al valore numerico 255 se l'immagine ha una profondità di colore di 8 bit), si ha:

  *Rout = Rin * WHITE / Rmax*
  *Gout = Gin * WHITE / Gmax*
  *Bout = Bin * WHITE / Bmax*

- Generazione del file in uscita dell'immagine filtrata in formato TIFF.

3.2 Uso del programma white_patch_filtro

La sintassi è la seguente:

./white_patch_filtro nome_file_input.tif nome_file_output.tif

Il tempo impiegato per l'elaborazione di una immagine da 320 x 210 pixel su un PC K6@300Mhz, ammettendo di operare su strutture dati in memoria statica, anche in questo caso è di circa 2 ms.

4. Algoritmo di *gamut match*

4.1 Descrizione

Questo algoritmo è stato proposto da G.D.Finlayson [Fin] che, a sua volta, prende spunto dal più complesso algoritmo *CRULE* proposto da D.Forsyth [Fors]; per tale motivo in alcuni testi questo algoritmo viene chiamato *Simple CRULE o S-CRULE*.
Alla base di entrambi questi algoritmi vi è la nozione di *gamut canonico*. Questo è definito come l'insieme convesso dei vettori RGB rappresentativi della riflettanza spettrale di tutte le superfici reali sottoposte all'illuminante canonico. In parole più semplici se immagino di tracciare nello spazio tridimensionale RGB tutti i punti dati dai vettori che definiscono la riflettanza di tutte le possibili superfici di tutti i possibili colori sottoposte all'illuminante canonico allora ottengo una nuvola di punti descrivibile come un solido convesso; questo è il gamut canonico.
E' chiaro che una immagine reale conterrà solamente un sottinsieme di queste superfici e, in generale, sarà illuminata da un altro illuminante sconosciuto a priori. Tutte quelle trasformazioni

che portano il gamut di questa immagine reale all'interno del gamut canonico sono possibili soluzioni del problema di costanza cromatica. In generale deve valere che:

$\forall \underline{p} \in G(I), D * \underline{p} \in G(C)$

cioè la matrice 3x3 D effettua una trasformazione che porta ciascun vettore RGB (indicato con $\underline{p}$) del gamut dell'immagine reale a mapparsi nel gamut dell'immagine canonica.
Si può dimostrare che per risolvere il problema della costanza cromatica è sufficiente considerare una matrice D di trasformazione diagonale.
L'algoritmo CRULE originale eseguiva proprio la ricerca di questa matrice D usando complessi e lunghi algoritmi di ricerca euristici, quindi ogni vettore $\underline{p}$ appartenente all'immagine di partenza veniva trasformato nel corrispondente vettore $\underline{q}$ tramite la seguente trasformazione:

$\underline{q} = D * \underline{p}$

E' facile intuire che il processo di ricerca di questa matrice D, anche se diagonale, dovendo operare su uno spazio tridimensionale risulta lungo e laborioso rendendo il tutto poco adatto ai nostri scopi. Tuttavia, partendo da questa idea originale, G.D. Finlayson ha introdotto alcune semplificazioni che hanno reso questo metodo sufficientemente veloce per poter essere usato anche per l'elaborazione in tempo reale delle immagini.
La semplificazione cruciale che è stata introdotta è quella di abbandonare lo spazio RGB e di operare in uno spazio bidimensionale (che viene chiamato spazio rg) in grado di conservare l'informazione di cromaticità che è l'unica informazione di interesse nel caso si eseguano delle segmentazioni sull'immagine.
Per far questo è stata scelta la seguente trasformazione:

$\underline{p} = < R, G, B > \rightarrow < R/B, G/B > = \underline{q}$

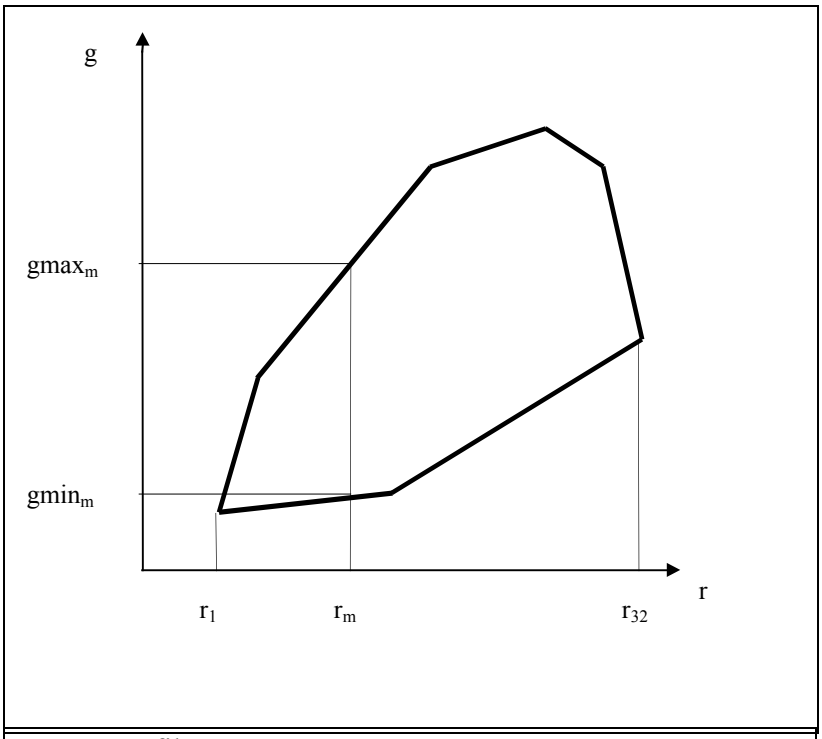
In questo caso viene persa l'informazione di intensità del vettore RGB $\underline{p}$ ma viene mantenuta l'informazione di orientamento.
Con questa operazione il gamut canonico si trasforma diventando una superficie convessa bidimensionale ma il concetto di fondo non cambia. In questo caso la matrice D da ricercare sarà una matrice diagonale 2x2 e il tutto risulta molto più semplice.
Riassumendo le operazioni da compiere per questa trasformazione sono le seguenti:

1) Calcolare il gamut canonico attraverso una immagine campione sottoposta all'illuminante standard e traslata nello spazio bidimensionale rg.
2) Portare l'immagine da filtrare nello spazio bidimensionale rg.
3) Calcolare il gamut di questa.
4) Ricercare la matrice diagonale 2x2, D, tale che il gamut dell'immagine da filtrare diventi il più simile possibile al gamut canonico.
5) Trasformare l'immagine di partenza per mezzo della matrice D.

4.1.1 Calcolo del gamut canonico

Il calcolo del gamut canonico è una operazione lunga ma che può essere svolta una sola volta visto che il risultato ottenuto è valido per qualsiasi immagine che poi verrà filtrata.
Per questo si è deciso di creare un programma a parte in grado di calcolare il gamut e di fornire una sintetica descrizione della sua forma attraverso un file che poi verrà letto e usato dal programma di filtro vero e proprio.
Come illuminante canonico si è optato per il tipo D65 (daylight 6500K definito dal CIE).

Il programma in grado di calcolare il gamut è show_gamut che, data in ingresso un'immagine in formato TIFF, è in grado di generare una nuova immagine in cui viene mostrata la costellazione di punti che compongono il gamut e un file che contiene la descrizione del perimetro del gamut come mostrato di seguito.



Contenuto file *gamut.per*

$r_1 gmax_1\ gmin_1$
....
$r_m\ gmax_m\ gmin_m$
....
$r_{32}\ gmax_{32}\ gmin_{32}$

In pratica in questo file vi è la descrizione del perimetro del gamut data per mezzo delle coordinate nello spazio rg di 64 suoi punti.

4.1.2 Algoritmo del filtro basato sul gamut match

L'algoritmo si compone dei seguenti passi:

- Calcolo della descrizione del perimetro del gamut dell'immagine d'ingresso con la stessa procedura descritta in precedenza per il gamut canonico.
- Impongo un valore iniziale alla matrice diagonale D (d_{11}=1.0 e d_{22}=1.0).
- Calcolo del valore finale della matrice D per mezzo di un algoritmo di ricerca euristica di tipo simulated annealing. Questo algoritmo usa una funzione in grado di calcolare la distanza dall'obiettivo e, variando con incrementi casuali di entità decrescente con il tempo gli elementi d_{11} e d_{22} della diagonale di D, è in grado di minimizzare questa funzione distanza. Così facendo, al termine dei cicli di ricerca, la matrice D sarà tale da minimizzare la differenza fra il gamut canonico e il gamut dell'immagine da filtrare. La funzione che misura la distanza che ho scelto merita un piccolo approfondimento. Questa funzione, come è ovvio, ha in ingresso gli elementi della diagonale principale di D e da questi calcola la nuova posizione del gamut dell'immagine da filtrare facendo per ogni punto del gamut: $r' = r * d_{11};\ g' = g * d_{22};$ con r' e g' le nuove coordinate del punto del perimetro. Poi per ciascun punto del perimetro del gamut canonico si

calcola la distanza con il punto più vicino (distanza geometrica) del gamut dell'immagine da filtrare e viceversa. Tutte queste distanze minime vengono sommate e il risultato è dato in uscita alla funzione come distanza dall'obiettivo. Come è ovvio il caso limite si ha con distanza uguale a 0 nel caso di perfetta sovrapposizione dei due gamut quindi l'algoritmo di ricerca, minimizzando questa funzione, tende a ricercare la condizione di perfetta sovrapposizione.

- La matrice D trovata dalla ricerca euristica viene usata per trasformare l'immagine in ingresso. Per ciascun vettore RGB p dell'immagine in ingresso, il corrispondente vettore q dell'immagine filtrata è calcolato come:

  $Rq = Rp * d_{11}$ (perché sarebbe $Rp / Bp * Bp * d_{11}$)
  $Gq = Gp * d_{22}$
  $Bq = Bp$ → questa assegnazione non è rigorosa ma è una scelta che è stata fatta per ovviare alla perdita di informazioni data dal passaggio dallo spazio tridimensionare RGB a quello bidimensionale rg. Tuttavia per poter generare in uscita una immagine in formato TIFF è indispensabile disporre anche della componente B, quindi si è optato per questa soluzione. Un eventuale algoritmo di segmentazione che opera nello spazio rg, effettuando la solita trasformazione R/B e G/B, è in grado di usare l'immagine filtrata nello spazio di cromaticità così come appare al termine della trasformazione operata dal filtro, senza alcun errore.

- Generazione del file in uscita dell'immagine filtrata in formato TIFF.

4.2 Uso del programma gamut_match_filtro

Come già anticipato, per l'uso di tale filtro è necessario creare preventivamente il file che contiene la descrizione dei punti notevoli del perimetro del gamut canonico. Per far questo è necessario avere a disposizione una immagine con illuminante canonico (D65) che contenga un numero significativo di patch colore (potrebbe andar bene una colorchart di Macbeth). La sintassi è la seguente:

./show_gamut nome_file_immagine_D65.tif nome_file_output.tif [scala]

Nel file di uscita, che avrà le stesse dimensioni in pixel dell'immagine in ingresso, verrà mostrata una costellazione di punti che rappresenta il gamut trovato nello spazio rg. Per mezzo del parametro opzionale "*scala*" è possibile inserire la scala di rappresentazione per questo gamut, ingrandendola o rimpicciolendola a piacere. Questo deve essere un numero reale e, per default, viene imposto uguale a 100.
Oltre a questo viene anche generato il file "*gamut.per*" che contiene la famosa descrizione dei 64 punti notevoli del perimetro del gamut. Per default il file che viene usato dal programma filtro per la descrizione del perimetro è il file "D65_gamut.per" quindi è necessario copiarlo:

cp gamut.per D65_gamut.per

A questo punto è possibile eseguire il filtraggio vero e proprio di una immagine:

./gamut_match_filtro nome_file_input.tif nome_file_output.tif

Durante l'esecuzione vengono mostrate a video alcune informazioni molto significative. In particolare viene mostrata la distanza iniziale e finale, calcolata dalla funzione che misura la distanza dall'obiettivo, usata nella ricerca euristica. Inoltre vengono anche stampate le distanze intermedie calcolate durante lo svolgersi della ricerca. In generale il matching dei gamut ha avuto un buon successo se la distanza finale è scesa sotto quota 30.

Il tempo impiegato per l'elaborazione di una immagine da 320 x 210 pixel su un PC K6@300Mhz, ammettendo di operare su strutture dati in memoria statica e disabilitando la stampa di messaggi sul video, è di circa 700 ms. Questo tempo, decisamente più elevato rispetto ai risultati ottenuti dagli altri filtraggi, può essere ridotto (anche sotto i 200 ms) diminuendo il numero di cicli dell'algoritmo di ricerca. Naturalmente questo aumento di efficienza va a scapito dell'efficacia e della precisione della soluzione proposta.

5. Algoritmo di *white + grey*

5.1 Descrizione

Come già anticipato, sono state affettuate alcune prove di filtraggio applicando alle immagini più filtri in cascata e si è valutata l'efficienza e l'efficacia delle varie soluzioni ottenute.
La combinazione di filtri che, alla luce delle prove svolte, si è rivelata la più promettente è l'applicazione successiva del filtro *white patch* seguito dal filtro *grey world* che, per brevità, chiameremo *white+grey*.
Per le prove si è utilizzato uno script di shell in grado di lanciare in sequenza sulla stessa immagine i programmi di filtraggio *white patch* e *grey world*.
Tuttavia l'efficienza di tale soluzione non è quanto di meglio si possa ottenere. Infatti, in totale, vi sono quattro cicli di scansione per tutti i pixel dell'immagine mentre realizzando un'apposita soluzione integrata è possibile ridurre questi cicli a tre. In quest'ultima ipotesi l'algoritmo diventerà il seguente:

- Scansione dell'immagine per trovare il massimo delle componenti per ciascun canale (che chiameremo *Rmax, Gmax* e *Bmax*).
- Calcolo dei coefficienti di correzione da applicare a ciascuna componente cromatica di ciascun pixel in modo tale che il bianco nell'immagine d'uscita corrisponda all'ipotetico pixel di componenti <*Rmax, Gmax, Bmax*> nell'immagine in ingresso. Dette *Rin, Gin* e *Bin* le tre componenti di un generico pixel dell'immagine in ingresso, *Rout, Gout* e *Bout* le tre componenti del corrispondente pixel dell'immagine d'uscita, e *WHITE* la componente del colore bianco per i canali RGB (che corrisponde al valore numerico 255 se l'immagine ha una profondità di colore di 8 bit), si ha:

  *Rout = Rin * WHITE / Rmax*
  *Gout = Gin * WHITE / Gmax*
  *Bout = Bin * WHITE / Bmax*

  In questo stesso ciclo, man mano che si procede nella scansione dei pixel, viene calcolata anche la media delle componenti RGB per tutti i pixel.
- Calcolo del valore medio di grigio che dovrà avere l'immagine filtrata. In prima battuta viene calcolato come media delle medie delle tre componenti RGB calcolate al punto precedente, poi si riscala in modo che non si abbiano overflow sulla scala tonale delle tre componenti.
- Calcolo dei coefficienti di correzione da applicare a ciascuna componente cromatica di ciascun pixel affinchè si ottenga il grigio medio calcolato in precedenza. Detto *grey* il grigio voluto, *media_red, media_green* e *media_blue* le medie RGB dell'immagine di partenza, *Rin, Gin* e *Bin* le tre componenti di un generico pixel dell'immagine in ingresso, *Rout, Gout* e *Bout* le tre componenti del corrispondente pixel dell'immagine d'uscita, si ha:

  *Rout = Rin * grey / media_red*
  *Gout = Gin * grey / media_green*
  *Bout = Bin * grey / media_blue*

- Generazione del file in uscita dell'immagine filtrata in formato TIFF.

5.2 Uso del programma White+Grey

La sintassi è la seguente:

./WhitePatch+GreyWorld nome_file_input.tif nome_file_output.tif

Considerando di realizzare un programma integrato che realizza l'algoritmo proposto al paragrafo precedente, è verosimile supporre che il tempo impiegato per l'elaborazione di una immagine da 320 x 210 pixel su un PC K6@300Mhz, ammettendo di operare su strutture dati in memoria statica, possa essere di circa 3 ms.

6. Test

6.1 Descrizione

Per i test degli algoritmi proposti sono state utilizzate tre diverse serie di immagini ritraenti lo stesso soggetto ma sottoposte a diversi illuminanti. Per ciascuna serie si è scelta come immagine di riferimento quella con illuminante di tipo daylight 65 e tutte le altre immagini della stessa serie sono state confrontate con questa per mezzo di alcuni indici di riferimento. Tutti gli indici usati misurano distanza fra i colori di patch corrispondenti fra due diverse immagini secondo particolari criteri. Sono state confrontate le serie originali con le serie ottenute usando i vari filtri per individuare i miglioramenti ottenuti nella diminuzione della distanza fra colori di soggetti uguali sotto illuminanti diverse. Come indici sono stati considerati il De94 (che è la distanza percettiva fra due colori nello spazio CIE Lab) il Drg (che è la distanza geometrica fra 2 colori nel piano di cromaticità rg) e il Dlab (che è la distanza geometrica nello spazio CIE Lab).
Negli allegati si possono vedere un campione per ciascuna serie di immagini con indicate le varie patch scelte per i test. A tal proposito, per facilitare il lavoro, è stato realizzato un apposito programma in grado di calcolare la media delle componenti RGB calcolata su una particolare patch dell'immagine. La sintassi è la seguente:

./media_RGB_patch nome_file_immagine.tif

Alla partenza verrà richiesto di specificare la posizione in x e y dell'angolo in alto a sinistra del patch (pixel) e della dimensione del lato. Poi verrà fornita la media delle componenti RGB calcolate per tale patch. Il processo può proseguire finchè non si inserisce almeno un dato non numerico.
I dati così ottenuti sono stati inseriti in fogli elettronici dove sono stati eseguiti i calcoli degli indici. In allegato sono state proposte le tabelle riepilogative dei test con alcuni grafici che permettono di valutare i risultati delle prove ad una prima occhiata.
In particolare vi sono 3 fogli, uno per ciascuna serie, ciascuno con le 3 tabelle per gli indici considerati. Ciascuna tabella riporta sulle colonne i risultati ottenuti per ciascun filtraggio applicato alle varie immagini. Per brevità e per chiarezza non è stato riportato il valore dell'indice per ciascuna patch ma solo la sommatoria complessiva del valore dell'indice per tutte le patch di una immagine.
In allegato vi sono anche le stampe dei fogli elettronici usati per i calcoli dove è possibile visionare i dati disaggregati per ciascuna patch dell'immagine.

6.2 Risultati e commenti

Come si può facilmente osservare alle tabelle riepilogative, tutti i metodi proposti portano ad un generale miglioramento dell'immagine diminuendo la dipendenza dall'illuminante usato. Si può vedere che la distanza fra i colori rilevati diminuisce notevolmente sulle immagini filtrate e questo

si traduce in un miglioramento netto del risultato della segmentazione (che in generale è sempre basata su algoritmi che sfruttano soglie sulla distanza fra colori).
E' da notare che per quanto riguarda il filtraggio basato sul *gamut match* l'unico indice significativo è il Drg dato che, come spiegato in precedenza, nel passaggio allo spazio di cromaticità rg viene persa l'informazione di intensità e non è più possibile ritornare con precisione allo spazio RGB (quindi nemmeno allo spazio CIE Lab).
Il *gamut match*, oltre a una scarsa efficienza computazionale, mostra una non sempre buona qualità dei risultati dato che, quasi sempre, ottiene risultati peggiori rispetto agli altri metodi.
Per quanto riguarda il filtraggio di tipo *white patch* si sono ottenuti discreti risultati ma, nel caso di immagine sovraesposte, rischia di essere poco o per nulla efficace. Infatti se i 3 canali RGB raggiungono la saturazione (cioè il canale raggiunge il valore massimo sulla scala tonale) questo filtro risulta perfettamente inefficace; e questa è una notevole limitazione.
Entrambi i metodi di *grey world* e *white+grey* hanno dato buoni risultati in tutte le situazioni ottenendo quasi sempre i migliori risultati in assoluto. Questo fatto, unito alla loro ottima efficienza computazionale, suggerisce un loro uso come blocchi di filtro per il pre-processing di immagini in tempo reale in applicazioni di robotica autonoma; l'overhead è così basso e i risultati sono talmente buoni che converrebbe sempre utilizzarli.
L'unica differenza fra i due metodi è che *white+grey* sembra fornire immagini visivamente più nitide dato che esegue una preventiva espansione della dinamica dell'immagine (per mezzo della parte *white patch*). E' possibile che la frammentazione delle immagini pretrattate da questo filtro riesca a ottenere risultati migliori rispetto a quelle di *grey world* soprattutto su landmark con colori molto simili fra loro.

A. Installazione e compilazione

Tutti i programmi di filtro realizzati operano in ambiente Linux e sfruttano la libreria *LIB_TIFF* come ausilio al trattamento delle immagini in formato Tiff.
Per prima cosa occorre creare una directory dove posizionare i sorgenti:

mkdir col_constancy
cd col_constancy

Poi si devono copiare tutti i sorgenti da dischetto:

mcopy a:*

A questo punto si deve esplodere la libreria LIB_TIFF:

tar xmvzf lib_tiff.tgz

Per finire si fa la compilazione di tutti i sorgenti:

make

B. Bibliografia

[Buch] – G.Buchsbaum, *A spacial processor model for object color perception*, J.Franklin inst., 310 (1), pag. 1-26, (1980).
[Funt] – B.Funt e V.Cardei, *Committee-based color constancy*, J.Opt.Soc.Am. A, 11(11), pp.3011-3020, 1994.
[Fin] – G.D.Finlayson, *Color in perspective*, IEEE Transaction on pattern analysis and machine intelligence, Vol.18 No.10, October 1996.
[Fors] – D.Forsyth, *A novel algorithm for color constancy*, Intelligence journal of computer vision, vol.5 pp.5-36, 1990.

C. Allegati

C.1 Listati dei programmi

C.2 Immagini di test

C.3 Risultati: tabelle riepilogative

C.4 Risultati: fogli elettronici