UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Elettronica per l'Automazione

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica
(Prof. Riccardo Cassinis)

Ricerca di Marmot mediante marker passivo

Elaborato di esame di: **Marco Arrigoni**
Valerio Parola

Consegnato il:
23 marzo 2002

Sommario

Scopo di questo lavoro è la scelta di un adeguato marker passivo da associare al robot Marmot e lo sviluppo di un opportuno algoritmo per il monitoraggio del marker all'interno di un ambiente noto mediante l'utilizzo di una webcam fissa, determinando prima di tutto la presenza o meno del robot nella regione d'interesse e in caso affermativo di stabilirne la posizione e l'orientamento.

1. Introduzione

Nel laboratorio di Robotica Avanzata (LRA) è stato individuato uno spazio all'interno del quale può muoversi il robot Marmot. Questa regione deve essere monitorata da una delle webcam disponibili presso lo stesso laboratorio.

Il lavoro propone una tecnica per determinare la posizione e l'orientamento del robot Marmot, quando è presente nella regione d'interesse, sfruttando le immagini fornite dalla webcam e un marker opportunamente scelto da associare al robot.

Si è quindi dovuto procedere, all'installazione nel laboratorio di una webcam, alla scelta di un opportuno marker da associare al robot Marmot, allo sviluppo di un algoritmo di ricerca per il ritrovamento del marker all'interno dell'immagine fornita dalla webcam, estraendone le informazioni di posizione e orientamento.

2. Il problema affrontato

Come sopra specificato la regione da monitorare è uno spazio all'interno del laboratorio di Robotica Avanzata (LRA) e le risorse usate per eseguire questo lavoro sono una webcam e un PC del laboratorio.

Il problema affrontato può essere riassunto in questi punti:

- Installazione della webcam;
- Scelta del marker;
- Sviluppo di un algoritmo per il ritrovamento del marker.

Ognuno di questi punti è suddiviso in sottoproblemi esaminati nei seguenti sottoparagrafi.

2.1. Installazione della webcam

Il laboratorio di Robotica Avanzata dispone di tre webcam, una di tipo parallela collegata ad un PC, un'altra sempre di tipo parallelo collegata ad un Macintosh e di una più recente di tipo USB.

Dovendo lavorare su PC si esclude la webcam collegata al Macintosh, rimangono così quella di tipo parallelo e quella di tipo USB.

La webcam parallela è una Connectix Color Quickcam mentre quella USB è una Philips Vesta ToUCam.

La Connectix Color Quickcam è una videocamera digitale in grado di registrare immagini e filmati a colori (a 24bit) e la risoluzione massima raggiungibile è di 640X480. Questa webcam si collega al PC mediante la porta parallela e tramite un connettore passante per la porta tastiera per l'alimentazione.

La Philips Vesta ToUCam è una videocamera digitale che si connette alla porta USB, è basata su un sensore CCD e un formato a palette RGB24 che garantisce una buona qualità dell'immagine e una risoluzione massima di 640x480, inoltre è in grado di fornire una sorgente video fino a 30 immagni al secondo.

Una volta individuata la webcam da usare occorre scegliere la posizione all'interno dell'LRA per la sua collocazione ed infine di installarla.

La webcam parallela era collegata ad un PC e in particolare a "nofive" su cui è stato installato un software per la cattura di immagini funzionante con questa webcam.

2.2. Scelta del marker

Uno dei metodi per il monitoraggio di un robot propone di associare al robot un marker [2], ovvero un oggetto di forma, dimensioni e colori noti, in questo modo il riconoscimento e la localizzazione di quest'ultimo all'interno dell'immagine fornita dalla webcam (o più in generale da una telecamera) risulta più semplice.

In letteratura esistono diversi tipi di marker: passivi, attivi, bidimensionali e tridimensionali, etc. In questo lavoro si è limitata la ricerca ai marker di tipo passivo e bidimensionale.

Scegliere un marker passivo bidimensionale significa decidere il numero, la forma, le dimensioni e i colori degli oggetti costituenti il marker e questa scelta deve essere fatta in dipendenza con l'algoritmo di ricerca.

2.3. Algoritmo di ricerca

Come precedentemente scritto l'algoritmo di ricerca dipende dal tipo di marker usato e vista la libertà di scelta del marker le strade possibili sono molteplici.

Quando si guarda l'immagine fornita dalla webcam due sono i principali problemi che ne emergono: la distorsione introdotta dalla telecamera e la dipendenza della forma degli oggetti dal loro orientamento nello spazio.

Quindi i problemi da risolvere sono due: il problema della distorsione e come semplificare l'algoritmo e/o il marker al fine di velocizzare la procedura di ritrovamento del marker stesso.

Nel prossimo capitolo si parlerà ampiamente di come sono stati risolti questi problemi.

3. La soluzione adottata

In questo capitolo si presenta la soluzione del problema nelle sue singole parti a partire da quanto scritto nei capitoli precedenti.

3.1. Installazione della webcam

Analizzando i pro e i contro delle due webcam a disposizione si possono trarre le seguenti conclusioni: i vantaggi della Connectix Color Quickcam sono un'ampia disponibilità di driver e la facilità di configurazione sulla piattaforma Linux nonché una buona quantità di software per il suo utilizzo, mentre i suoi punti deboli sono sicuramente la sua lentezza e una qualità non eccezionale delle immagini.

I punti di forza della Philips Vesta ToUCam sono la velocità di acquisizione e la buona qualità d'immagine, mentre i punti deboli sono la scarsa reperibilità dei driver e la sua configurazione in ambiente Linux.

La risoluzione scelta per le immagini catturate dalla webcam è di 320x240, in questo modo si hanno dimensioni ridotte dei file su cui lavorare e una discreta qualità dell'immagine.

Utilizzando entrambe le webcam alla risoluzione scelta (320x240) si è visto che la differenza di qualità tra le immagini fornite dalla Connectix Color Quickcam e dalla Philips Vesta ToUCam è minima. Inoltre considerando la necessità di costruire una prolunga di 10m per il collegamento tra webcam e PC il quale avrebbe comunque influito sulla velocità, si è deciso di usare la Connectix Color Quickcam.

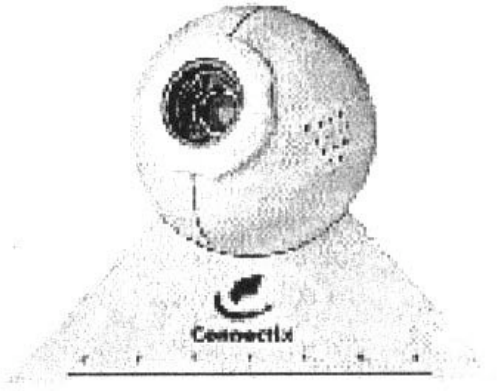


Figura 1: La Connectix Color Quickcam

L'area destinata al monitoraggio del robot Marmot è delimitata da un portone, un mobile, un tavolo da lavoro e una finestra. Tali elementi hanno limitato la scelta della posizione della webcam a sopra la finestra oppure sul soffitto del laboratorio.

Tra le due possibili scelte si è preferito installare la webcam sul soffitto del laboratorio. Con questa soluzione la webcam risulta più vicina al PC (cavo di prolunga più corto) ed inoltre questa è la posizione dove la distorsione introdotta dalla webcam risulta minima.

Come sopra specificato è stato necessario costruire un cavo di prolunga per poter collegare la webcam al PC. Anche in questo caso le possibilità di realizzazione erano due:

- un cavo parallelo, che però avrebbe necessitato di un ulteriore cavo per l'alimentazione della webcam (si ricorda che la Connectix Color Quickcam ha un connettore apposito per prendere l'alimentazione dalla porta tastiera)
- si poteva tagliare il cavo della webcam e inserire una prolunga.

La soluzione adottata è stata la seconda. Il cavo della webcam è composto da otto fili più una calza e di conseguenza è stato creato una prolunga specifica.

Costruita la prolunga si è provato che il collegamento funzionasse e si è quindi proceduto all'installazione sul soffitto del laboratorio. La webcam è stata attaccata mediante una staffa ad una conduttura metallica passante.

3.2. Scelta del marker

La scelta del marker detta le caratteristiche dell'algoritmo di ricerca, quindi una scelta accurata permette di semplificare il lavoro di implementazione software.

Riassumiamo i problemi da risolvere:

- Deformazione del marker nell'immagine (deformazione prospettica e deformazione introdotta dalla webcam);
- Impatto della forma del marker sull'algoritmo di ricerca;

Il primo problema si risolve in parte utilizzando un algoritmo di raddrizzamento dell'immagine, ma anche adottando un marker che permetta di essere deformato senza perdere troppa informazione sulla sua forma. Il marker dovrebbe essere il più simmetrico possibile, in modo da risultare pressoché uguale sia quando è ruotato che quando è distante dalla webcam. L'unica forma geometrica che rispetta le limitazioni imposte è il cerchio, non si hanno problemi di rotazione, e deformato assomiglia al più ad un ellisse.

Il cerchio diventa così un ottimo candidato per diventare un elemento del marker, ma non porta alcuna informazione sulla rotazione. Diventa necessario affiancare a questa figura un'altra figura (marker secondario) che definisca l'orientamento di tutto il marker. La seconda figura può essere inserita all'interno o all'esterno del marker principale che d'ora in poi sarà il nostro cerchio. L'inserimento del marker secondario nel marker principale (esempio fig. A), mostra come l'area utile del marker principale venga diminuita dell'area del marker secondario e come si introduca il problema della rotazione. Il marker in fig. A introduce la difficoltà di fare una correlazione tra il marker e un'immagine di riferimento che non risenta della rotazione, esempio una correlazione tra il marker e un cerchio pieno, la correlazione non potrà mai risultare massima vista l'introduzione del marker secondario. Si potrebbe allora pensare di usare una figura di riferimento identica al marker e realizzare una serie di correlazioni tra il marker e le possibili rotazioni della figura di riferimento, ma questo sistema introduce un peso computazionale per la macchina non indifferente.

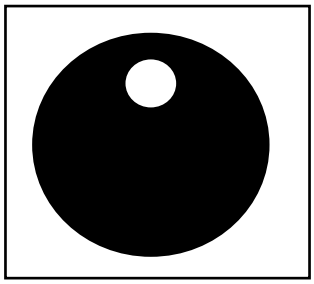


Fig. A

La strada da seguire per facilitare la correlazione tra marker e figura di riferimento è di adottare un marker secondario esterno al marker principale in modo da risolvere il problema scindendolo in due parti distinte e facilmente implementabili.

La Fig. B mostra il marker adottato per la soluzione del nostro problema.

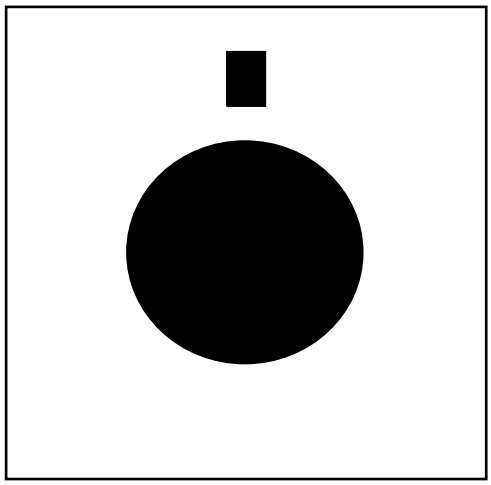


Fig. B

Il marker principale è un cerchio di raggio noto, il marker secondario è un rettangolo di base e altezza note e distanza prefissata dal centro del marker principale.

La ricerca del marker viene divisa in due ricerche distinte, si effettua una ricerca iniziale del marker principale nell'immagine e si individuano i candidati a marker attraverso una correlazione con una figura di riferimento. Per ogni candidato che abbia un grado di correlazione superiore a una percentuale prefissata, si analizza l'intorno del marker principale per determinare l'esistenza del marker secondario.

L'algoritmo di ricerca verrà commentato meglio nel paragrafo in cui si presenta il codice e l'algoritmo di ricerca.

3.3. Conversione del formato immagine

Il software utilizzato per catturare le immagini fornisce un'immagine in formato jpeg che è un formato compresso. Ovviamente non è possibile lavora direttamente su questo tipo di formato, occorre quindi convertire l'immagine in un formato più gestibile. Il formato scelto è il tipo raw che di fatto è una matrice di punti, dove ogni punto è rappresentato dai bit necessari per definire la profondità del colore (256 colori sono 8bit = 1Byte).

Non è possibile ottenere direttamente il formato raw dal formato compresso jpeg, occorre prima trasformare l'immagine in un formato non compresso per esempio il

bitmap (bmp). Facendo uso delle funzioni della libreria `libjpeg-6b` ed in particolare si è usato il comando `djpeg` che permette di convertire il jpeg in altri formati:

```
djpeg [switches] [jpegfile]  >imagefile
```

Dove gli `switches` più importanti sono:

`-colors N`	produce in uscita un'immagine con al più N colori.
`-quantize N`	simile a `-colors N`.
`-fast`	produce un'uscita veloce ma di bassa qualità.
`-grayscale`	forza l'uscita a scala di grigio anche se il `jpeg` è a colori.
`-scale M/N`	scala l'uscita di un fattore `M/N`.
`-bmp`	seleziona l'uscita nel formato `BMP` (Windows).
`-gif`	seleziona l'uscita nel formato `GIF`.
`-os2`	seleziona l'uscita nel formato `BMP (OS/2).`
`-pnm`	seleziona l'uscita nel formato `PBMPLUS (PPM/PGM).`
`-rle`	seleziona l'uscita nel formato `RLE`.
`-targa`	seleziona l'uscita nel formato `Targa`.

Per maggiori dettagli si rimanda al manuale della libreria in questione.

Il comando completo usato in questo lavoro è:

```
djpeg -grayscale -bmp sorgente.jpg > destinazione.bmp
```

In questo modo si ottiene un'immagine a 256 livelli di grigio nel formato non compresso bmp.

L'algoritmo sviluppato utilizza un'immagine a due livelli di colore, di conseguenza si è scelto di convertire l'immagine jpeg direttamente in una a 256 livelli di grigio, in questo modo l'algoritmo di decodifica è più veloce. Inoltre un'immagine a 256 livelli di grigio associa ad ogni pixel esattamente un byte e questo, come si vedrà nei capitoli successivi, porta a delle semplificazioni.

Pur non entrando in dettagli, si può dire che un file bmp è composto da un header e da una matrice di punti, nel caso in questione l'immagine è una 320x240 pixel ed essendo ogni pixel esattamente un byte la dimensione della matrice di punti nel file bmp è di 76800 byte. Quest'ultima parte non essendo compressa è analoga ad una matrice di punti in formato raw.

Per ottenere definitivamente il formato raw si è usato il comando tail (comando utente Linux) che di default restituisce le ultime dieci righe di un file e tramite varie opzioni è possibile ottenere come output (in questo caso in un nuovo file che sarà poi l'immagine in formato raw) gli ultimi n byte di un file (l'immagine in formato bmp).

```
tail [opzioni][file]
```

Dove le opzioni più importanti di questo comando sono:

`-n[k]` Inizia a stampare dall'n-esimo elemento dalla fine del file e k specifica il tipo di elemento da contare (l righe, c caratteri e b blocchi);

`-k` Come –n ma utilizza il conteggio di default (10);

`+n[k]` Come –n[k] ma inizia dall'n-esimo elemento dall'inizio del file;

`+k` Come –k ma conta dall'inizio del file;

`-c` *num*`{bkm}, --bytes` *num*`{bkm}`

Stampa gli ultimi *num* byte. É possibile specificare una diversa dimensione di blocco mediante i caratteri b (512byte), k (1 kilobyte) e m (1 megabyte);

`-n` *num*`, --lines` *num*

Stampa le ultime num righe;

`-q, --quiet, --silent`

Sopprime le intestazioni dei nomi di file.

Per togliere l'header del formato bmp dall'immagine si è usato il comando tail nel seguente modo:

```
tail -bytes 76800 sorgente.bmp > destinazione.raw
```

Si fa notare che l'immagine che si ha è a 256 livelli di grigio il che significa un byte per pixel e che la risoluzione è di 320x240 e quindi il numero di byte che ci interessa è esattamente 1*320*240=76800 byte.

Nota: il formato bmp memorizza i pixel a partire dall'ultimo pixel dell'immagine (ovvero quello in basso a destra) e quindi l'immagine in formato raw restituita dal tail risulta ribaltata e capovolta, il problema si può risolvere durante la fase di caricamento dell'immagine.

3.4. Ricerca del marker

Prima di parlare dell'algoritmo vero e proprio di ricerca occorre risolvere il problema dovuto alla distorsione introdotta dalla webcam. Anche in questo caso sono molteplici le strade possibili: si può lavorare direttamente sull'immagine fornita dalla webcam ed introducendo una distorsione sul marker da trovare, oppure si può togliere la distorsione dall'immagine applicando un opportuno algoritmo oppure si può salvare in un vettore un certo numero di marker visto dalla webcam e in posizioni note e poi si cerca il marker per settori dove ad ogni settore si deve usare un marker specifico precedentemente salvato nel vettore.

La soluzione adottata in questo lavoro è stata quella di creare un opportuno algoritmo di raddrizzamento da applicare all'immagine fornita dalla webcam e quindi lavorando su di essa utilizzando un'immagine del marker.

Ora si vedrà in dettaglio l'algoritmo di raddrizzamento e l'algoritmo di ritrovamento.

3.4.1. Algoritmo di raddrizzamento

L'algoritmo di raddrizzamento applicato all'immagine fornita dalla webcam restituisce una nuova immagine priva di distorsione.

Scopo di questo lavoro è ottenere una posizione del robot in una stanza e quindi non è richiesta una precisione elevata, di conseguenza si è deciso di adottare un algoritmo di approssimazione di tipo lineare.

Occorre quindi affrontare il problema della modellizzazione e taratura della telecamera. Solitamente l'approccio a questo problema è di tipo matematico: si cerca di ricondurre il funzionamento del dispositivo ad alcune equazioni che tengono conto di tutti i processi che intervengono nell'acquisizione di un'immagine; conoscendo la struttura del modello matematico attraverso un processo di taratura si arriva a determinare i valori dei coefficienti di tale modello.

3.4.1.1 Parametri del modello

Un modello è una rappresentazione matematica del comportamento di un dispositivo reale. Nel caso di una telecamera il modello deve tener conto di due aspetti: la relazione tra telecamera e mondo esterno ed il funzionamento interno del dispositivo. Questi due aspetti portano a due gruppi di parametri: estrinseci ed intrinseci. I parametri estrinseci indicano la posizione della telecamera rispetto ad un sistema di riferimento esterno, mentre i parametri intrinseci dipendono dalle caratteristiche interne della telecamera quali il tipo di lente, la focale etc. [1]

Non si darà qui una trattazione completa dell'argomento ma si riassume la parte significativa per lo scopo di questo lavoro.

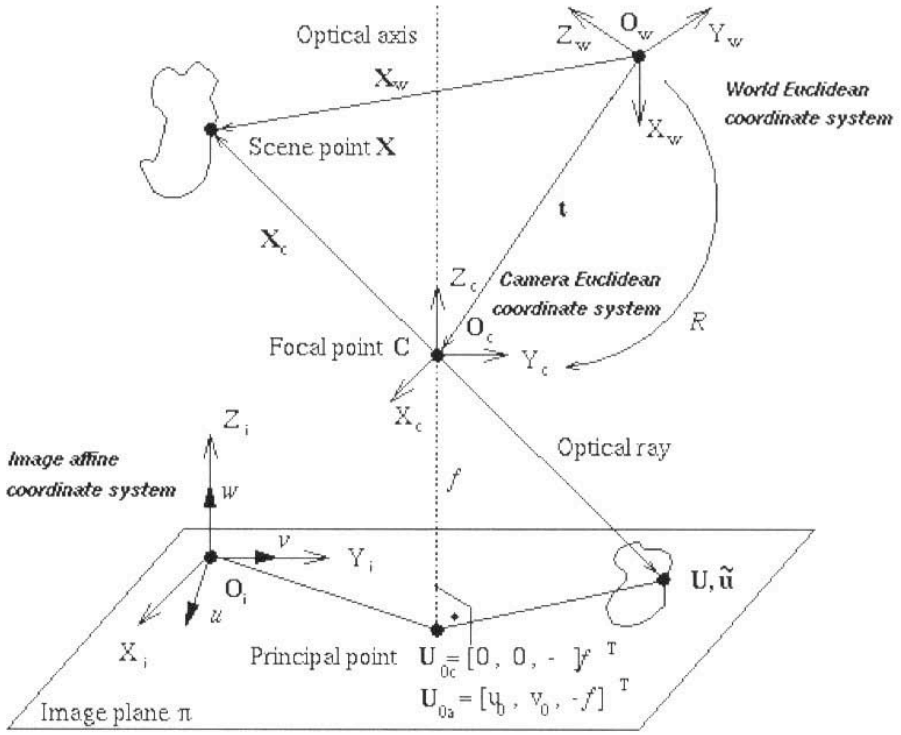


Figura 2: Prospettiva lineare di una telecamera

Prima di tutto si ricorda che in questo tipo di trattazioni le coordinate sono espresse in coordinate omogenee.

Di solito per questo tipo di problemi si dichiarano quattro differenti sistemi di coordinate: un sistema di coordinate euclideo riferito al mondo reale, un sistema di coordinate euclideo riferito alla telecamera, un sistema di coordinate euclideo riferito al piano dell'immagine e un quarto sistema di coordinate ancora sul piano dell'immagine ma non euclideo. I vari sistemi di riferimento sono rappresentati in Figura 2.

Per capire come si ottiene la forma finale a cui si vuole raggiungere si fa riferimento a tutti i sistemi di riferimento anche se allo scopo di questo lavoro servono solo il primo e l'ultimo.

Per riferire un generico punto dalle coordinate riferite al mondo reale a quelle riferite alla telecamere si usa una trasformazione del tipo:

$$X_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = R\left(X_w - t\right)$$

dove X_w sono le coordinate di un punto nel sistema di riferimento del mondo reale e X_c sono le coordinate dello stesso punto ma riferite al sistema di coordinate della telecamera. R è la matrice di rotazione (specifica come sono orientati gli assi da un sistema di riferimento all'altro) e t è un vettore che specifica la traslazione dell'origine di un sistema rispetto all'altro.

In generale la forma matriciale a cui si arriva è del tipo:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & -u_0 \\ 0 & c & -v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -fx_c/z_c \\ -fy_c/z_c \\ 1 \end{bmatrix}$$

Dove u, v sono le coordinate nel sistema di riferimento dell'immagine, u_0 v_0 sono le coordinate del punto intersezione dell'asse ottico con il piano dell'immagine riferite al sistema di coordinate relativo all'immagine, mentre i parametri a, b, c sono parametri intrinseci della telecamera.

Volendo riferire tutto al sistema di coordinate relative al mondo risulta la seguente espressione matriciale:

$$\begin{bmatrix} uz_c \\ vz_c \\ z_c \end{bmatrix} = \begin{bmatrix} -fa & -fb & -u_0 \\ 0 & -fc & -v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = KR\left(X_w - t\right)$$

Questa forma matriciale si può esprimere anche come:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Dove M è una matrice 3x4 chiamata matrice di proiezione e include tutti i parametri, intrinseci ed estrinseci.

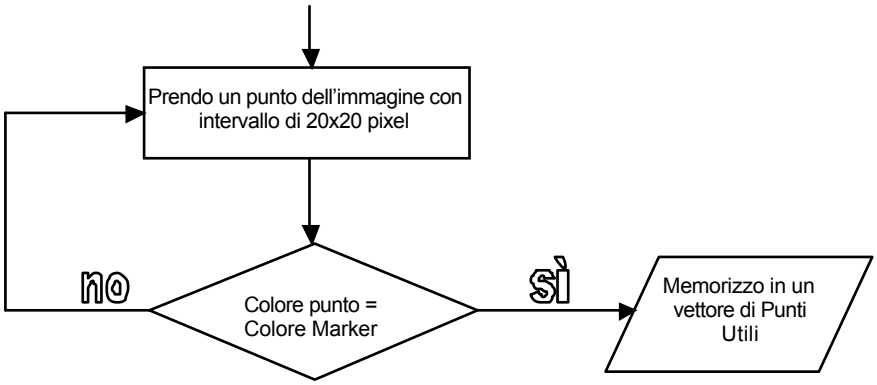
Trovare i coefficienti della matrice di proiezione significa tarare il sistema. Esistono vari metodi per estrapolare i coefficienti della matrice di proiezione, in questo caso il metodo utilizzato per la taratura è quello relativo ad una scena nota ed è analizzato nel capitolo 4.3.2 relativo alla taratura.

Una volta trovati i coefficienti della matrice di proiezione si è in grado di stabilire dove un punto nello spazio inquadrato dalla telecamera va a finire nel piano immagine. Ovviamente passando da un ambiente a tre dimensioni ad uno bidimensionale non ci sarà corrispondenza biunivoca tra i punti del piano immagine e lo spazio tridimensionale inquadrato dalla telecamera bensì ad ogni punto del piano immagine corrispondono più punti nello spazio tridimensionale. Questo porta ad avere un sistema non invertibile e

questo lavoro necessita proprio della relazione inversa. Ai fini di questo lavoro, però, il robot si muove su un piano e quindi scelto un opportuno sistema di coordinate con gli assi x e y sul piano di movimento del robot e la z uscente da questo si ha che per ogni posizione del robot la z è una coordinata costante. In questo modo riusciamo ad avere un sistema lineare e invertibile in quanto la matrice di proiezione diventa una 3x3 e quindi è sufficiente calcolarsi la matrice inversa.

3.4.2. Algoritmo di ricerca

Per capire l'algoritmo di ricerca si è pensato di realizzare un diagramma di flusso che ne esemplifichi il funzionamento. Come prima cosa campiono l'immagine in cui trovare il marker ed estraggo un vettore di punti che potrebbero appartenere ad un marker.



Il vettore di punti utili mantiene le coordinate del punto che rispecchia il colore del marker, il campionamento deve essere più piccolo della dimensione del marker principale per essere sicuri di recuperare almeno un punto interno. Il campionamento permette di non dover verificare e fare una correlazione di tutti i punti dell'immagine, ma di controllare soltanto un set limitato di possibilità, diminuendo il carico computazionale.

Il secondo passo consiste nel determinare l'ipotetico centro del cerchio cioè del marker principale partendo dal presupposto che il punto utile sia interno al cerchio. Da un punto di vista geometrico si può determinare il centro di un cerchio partendo dalle coordinate xp, yp di un punto interno e determinando le distanze dx1, dx2 sull'asse delle scisse e rispettivamente dy1, dy2 sulle ordinate, queste quattro misure rappresentano le distanze dal punto alla circonferenza del cerchio (fig. C). Le coordinate del centro si calcolano con la seguente formula:

$$xcentro = xp - dx1 + (dx1 + dx2) / 2 \qquad ycentro = yp - dy1 + (dy1 + dy2) / 2$$

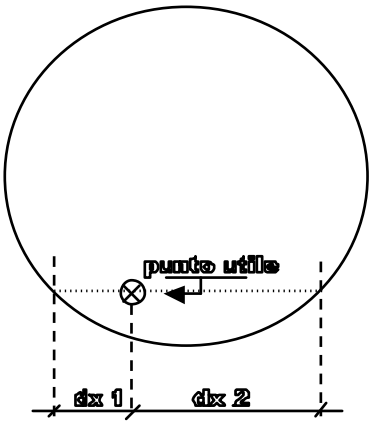
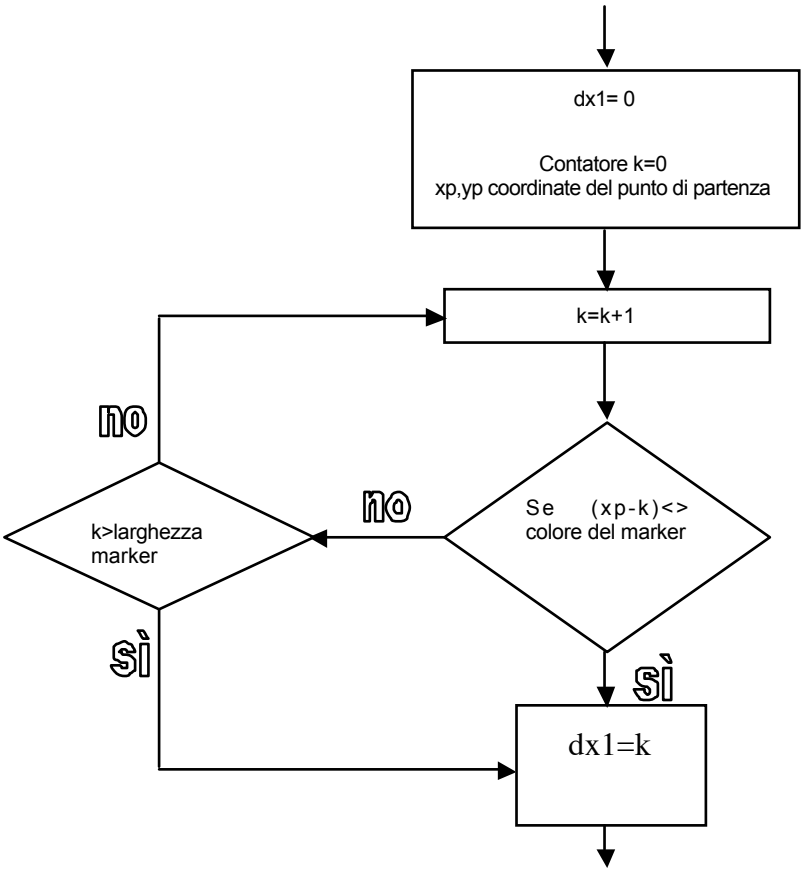


Fig. C

Di seguito viene riportato lo schema del procedimento per la determinazione delle misure dx1, dx2, dy1, dy2, il grafico mostra l'algoritmo riferito ad una sola distanza.



Una volta determinato il centro del presunto marker effettuo una correlazione pixel a pixel tra l'ipotetico marker e un'immagine di riferimento. Durante la correlazione si contano i punti uguali tra marker e figura di riferimento e si crea un parametro di affidabilità uguale al numero di punti uguali diviso il numero di punti controllati (l'area in pixel della figura di riferimento). Se il grado di affidabilità è superiore ad un certo valore allora posso ipotizzare di aver trovato il marker principale, ma sarò sicuro di aver trovato il marker solo dopo aver controllato l'esistenza del marker secondario.

La ricerca del marker secondario è più problematica del primo. Si sarebbe dovuto effettuare una correlazione tra il marker secondario e una figura di riferimento, in tutte le sue possibili rotazioni e posizioni intorno al marker principale, tale ricerca avrebbe allungato i tempi di utilizzo del computer e avrebbe aggravato sull'efficienza dell'algoritmo. Si è pensato allora di adottare un altro criterio di ricerca con un grado di velocità maggiore, ma variabile consentendo di garantire la robustezza dell'algoritmo di ricerca. L'algoritmo di ricerca del marker secondario, crea un vettore di punti prendendo i punti su una circonferenza, rispetto al centro del marker principale, di raggio noto. Tale raggio può variare dal raggio minimo al raggio massimo di distanza tra un punto interno del marker secondario e il centro del marker primario. Una volta presi i punti si cerca un punto nel vettore che potrebbe appartenere al marker secondario e poì si analizza l'intono di tale punto con quello che dovrebbe esserci.

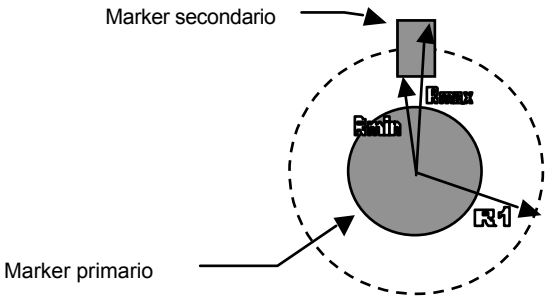


Fig. D

Qualsiasi sia la rotazione del marker secondario, il vettore dei punti sulla circonferenza di raggio R1, contiene un insieme di punti del marker secondario di pari larghezza. Questo fatto garantisce di trovare il marker con estrema facilità analizzando un vettore di punti e senza effettuare un analisi più approfondita. Per aumentare la robustezza dell'algoritmo sarebbe sufficiente effettuare più prove con raggi che vanno da Rmin a Rmax, se tali prove danno esiti simili allora il marker secondario è trovato e si può determinare l'angolo di rotazione.

0	0	0	0	0	0	255	255	255	255	255	255	0	0	0	0	0	0

Esempio del Vettore contenente i punti sulla circonferenza di raggio R1

Il vettore contiene dei valori pari a 255 che indicano la presenza di un punto del marker secondario. L'algoritmo parte dalla posizione zero del vettore e inizia a cercare il valore 255, trovato un valore richiama la procedura di correlazione per verificare se si tratta del marker secondario, se tale procedura restituisce un valore di affidabilità adeguatamente alto allora considero di aver trovato il marker.

Per calcolare la rotazione risentendo al minimo del problema della distorsione dell'immagine e delle approssimazioni effettuate, si considera un vettore di 314 campi, a ognuno si considera assegnato un angolo di 2 radianti. L'angolo non sarà dato dalla posizione del centro del marker secondario nel vettore, ma consideriamo l'inizio del marker e la fine all'interno del vettore e ne facciamo una media.

Considerando il fatto che un vettore di 314 posti è grande per le dimensioni delle immagini considerate, è facile che nell'andare a recuperare i punti sulla circonferenza di raggio R1 si faccia un sovracampionamento considerando dei pixel a stesse coordinate più volte, tali valori nell'array vengono considerati doppi e quindi si assegna un valore negativo. Altro problema affrontato è la possibilità che il marker secondario finisca fuori l'area visibile e quindi non sia presente. In una situazione del genere il programma dovrà comunque scrivere che ha trovato il marker principale, ma non è sicuro che sia il marker effettivo perché non ha rilevato la presenza del marker secondario.

4. Modalità operative

Ora si specifica tutto quello che serve e come procedere per il corretto funzionamento ed utilizzo.

4.1. Componenti necessari

Questo lavoro per poter funzionare necessita sia di componenti hardware che di componenti software elencati nei sottoparagrafi successivi.

4.1.1. Componenti hardware

I componenti hardware necessari per il corretto utilizzo di questo lavoro sono i seguenti:

- PC
- Connectix Color Quickcam
- Prolunga per il collegamento della webcam
- Marker

Questo lavoro è stato svolto utilizzando il PC "nofive" del laboratorio di Robotica Avanzata al quale è stato installato la Connectix Color Quickcam usufruendo della prolunga realizzata per questa webcam.

Ovviamente per l'utilizzo corretto del software sviluppato occorre il marker di cui si è discusso nei capitoli precedenti.

4.1.2. Componenti software

Tutti i componenti software utilizzati in questo lavoro necessitano di una piattaforma Linux. L'elenco del software utilizzato è il seguente e nei prossimi sottoparagrafi saranno spiegati in dettaglio:

- Cqcam;
- Script primo_via;
- Script secondo_via;
- Lancio;
- El_Robot.

4.1.2.1 Cqcam

Cqcam è un pacchetto sviluppato da Patrick Reynolds (reynolds@cs.duke.edu) che supporta le webcam Color Quickcam e Color Quickcam 2 prodotte da Connectix (il caso in questione) e da Logitech.

Queste webcam sono ormai tutte fuori produzione e oggi la maggior parte dei produttori le hanno sostituite con delle più efficienti USB mentre per quanto riguarda il software Cqcam non sono stati previsti degli sviluppi.

Il pacchetto Cqcam contiene sia i driver per il funzionamento della Connectix Color Quickcam che il software necessario per la cattura e salvataggio di immagini.

La cattura di un'immagine avviene con il comando:

```
cqcam > file_destinazione.jpg
```

Il pacchetto contiene anche altri programmi interessanti tra i quali si cita "Xqcam" e "Gtkcam" che permettono la visione in ambiente X del video catturato dalla webcam.

4.1.2.2 Script primo_via

Questo script, che viene riportato per esteso nel paragrafo 8.1, lancia Cqcam per settare la webcam con le opzione (-a+ –r). La prima opzione permette il settaggio automatico dei parametri della webcam (come contrasto, luminosità, etc.) al fine di ottenere l'immagine migliore. La seconda opzione salva i parametri della webcam in un opportuno file, in modo da ricaricarli ad un successivo avvio della webcam senza richiamare l'operazione di settaggio automatico, che è abbastanza lenta.

4.1.2.3 Script secondo_via

Questo script, riportato nel paragrafo 8.2, viene lanciato in ciclo infinito dal programma "Lancio" e il suo compito è quello di catturare l'immagine mediante Cqcam e di salvarla in un file, poi converte l'immagine in forma di matrice di punti dove ogni punto rappresenta il livello di grigio del pixel (formato raw). Dopo aver convertito l'immagine viene lanciato il programma "El_Robot", passandogli come parametri il nome del file contenente l'immagine e il marker. "El_Robot" ha il compito di trovare il robot e stabilirne posizione e orientamento.

Si fa notare che quando viene lanciato Cqcam si utilizza l'opzione –a- così che non c'e più la fase di autosettaggio e si usano i valori salvati precedentemente durante lo script primo_via. Questo perché la fase di autosettaggio richiede molto tempo e così l'acquisizione dell'immagine risulta più veloce.

4.1.2.4 Lancio

Lo scopo di questo software è quello di mandare in esecuzione il sistema sviluppato. il programma lancia prima lo script "primo_via" per l'autosettaggio della webcam e poi, in un ciclo infinito, viene lanciato lo script "secondo_via". Per uscire da questo programma e quindi da tutto il sistema sviluppato è sufficiente premere un tasto sulla keyboard.

4.1.2.5 El_Robot

Questo è il programma principale, scopo di questo lavoro. El_Robot è un eseguibile lanciato dallo script "secondo_via" e come parametri di ingresso vuole l'immagine in formato raw e il marker in formato raw. L'uscita di questo programma sono le coordinate di posizione e l'orientamento del robot se trovato.

I particolari dell'algoritmo implementato da questo software sono ampiamente discussi nei paragrafi precedenti, comunque, giusto per riassumere, i passi principali sono: raddrizzamento dell'immagine, ricerca del marker e ricerca del secondo marker per stabilire l'orientamento del robot.

4.2. Modalità di installazione

Per quanto riguarda l'installazione hardware l'unica cosa di rilievo è l'installazione della webcam Connectix Color Quickcam, va collegata alla porta parallela del PC e ad un connettore da inserire al posto della tastiera da cui prende l'alimentazione. Ovviamente lo spinotto della tastiera va inserito nel rimante connettore della webcam.

Per quanto riguarda i componenti software prima di tutto si deve installare il pacchetto contenente Cqcam, quindi vanno copiati i due script creati e i due eseguibili in una cartella di lavoro.

Va inoltre osservato che lo script di esecuzione fa uso di un comando utente di Linux e un comando della libreria libjpeg-6b.

4.2.1. Installazione di Cqcam

Come indicato all'inizio di questo capitolo quando si scompatta il file di installazione viene creata una directory per Cqcam. A questo punto mediante il comando

```
cd cqcam-0.91
```

ci si porta all'interno della directory riservata a Cqcam. Ora per installare correttamente il software occorre digitare

```
./configure
```

questo comando ha lo scopo di creare un file di installazione chiamato makefile e ora bisogna lanciare questo digitando semplicemente

```
make
```

al prompt di shell. In questo modo viene compilato Cqcam. Per concludere l'installazione si deve digitare

```
make install
```

A questo punto l'installazione è completata e nella directory `/usr/local/bin` si dovrebbero trovare i seguenti file:

`./cqcam-0.91/cli/cqcam`	(software per grabbare le immagini da linea di comando)
`./cqcam-0.91/xfe/xcqcam`	(interfaccia grafica di Cqcam)
`./cqcam-0.91/gtkfe/gtkcam`	(interfaccia grafica per GTK+)
`./cqcam-0.91webcam/webcam`	(Utility per lo streaming)

La procedura di installazione dovrebbe inoltre copiare nella home un file di configurazione di nome `./cqcrc`. Qualora questo file non si trovasse nella home Cqcam non funzionerebbe quindi per risolvere il problema è sufficiente copiare manualmente questo file.

Per il funzionamento di Cqcam è necessario aver installato la libreria per la gestione dei jpeg. Normalmente questa libreria (libjpeg-6b) è disponibile su tutte le distribuzioni, comunque per chi non l'avesse può facilmente scaricarla all'indirizzo: `ftp://ftp.uu.net/graphics/jpeg`.

4.2.2. Installazione del software sviluppato

Il software è composto essenzialmente da quattro pezzi: due script e due eseguibili.

Per quanto riguarda l'installazione occorre creare una cartella in cui copiare i quattro file, dopodiché bisogna rieditare i due script e modificare la variabile che specifica il percorso di lavoro con il percorso della cartella appena creata.

Si ricorda che per lanciare il programma che si è sviluppato occorre digitare al prompt dei comandi la seguente stringa:

```
./Lancio
```

Mentre per concludere l'esecuzione del programma è sufficiente premere un qualsiasi tasto.

4.3. Modalità di taratura

Per tarare l'intero sistema sviluppato occorre prima di tutto settare i parametri del software per la cattura delle immagini e quindi si passa alla calibrazione del modello utilizzato per la rimappatura.

4.3.1. Settaggio di Cqcam

Per quanto riguarda questo software si posso modificare diversi parametri, si può vederne l'elenco intero digitando al prompt di shell

```
Cqcam –h
```

Dovrebbe comparire una schermata del tipo:

```
-32[+|-]                Turn 32-bpp mode on or off
                        (off=24bpp)
-a[+|-]                 Use/suppress brightness and color
                        balance auto-adjustment
-b val                  Set brightness
-B val                  Set black level
-c val                  Set contrast
-d val                  Specify (or skip) camera-detection
-E val                  Set software blue level
-G val                  Set software green level
-h                      View this brief help screen
-H val                  Set hue (blue level)
-j                      Write output in JPEG format
-l val                  Set left column
-m delay,name[,count]   Movie mode: capture count frames,
                        one every val second to name-X.ppm
-P val                  Set port to attempt (val must be in
                        hex format)
-q val                  Set JPEG quality, for use with –j
                        (default=50)
-r                      Remember (store) the brightness in
                        .cqcrc
-R val                  Set software red level
-s val                  Set scale factor (decimation)
-S val                  Set saturation
-t val                  Set top row
```

```
-u                      Force a unidirectional port mode
-V                      Print version information and exit
-w val                  Set white level
-x val                  Set width
-y val                  Set height
```

Si considera ora solo i parametri utilizzati da questo lavoro, per gli altri si rimanda all'help del software.

Sono due i parametri utilizzati in questo lavoro: `-a` e `-j`. Il primo serve per fare l'autosettaggio dei parametri ovvero quando si lancia il software prima di salvare l'immagine Cqcam setta automaticamente i parametri secondo una sua modalità di taratura. Il secondo, invece, serve per salvare l'immagine in formato jpeg con una qualità pari a 50 (è possibile variare la qualità usando il parametro `-q val`).

Come meglio spiegato nel paragrafo relativo allo script primo_via, Cqcam viene chiamato la prima volta mediante il comando

```
Cqcam -a+ -r -j > file_destinazione.jpg
```

In modo che ci sia l'autosettaggio della webcam e il salvataggio dei vari parametri, mentre nello script secondo_via viene usato nel seguente modo:

```
Cqcam -a- -j > file_destinazione.jpg
```

Questo perché si è visto che l'autosettaggio impiega alcuni secondi e visto che le condizioni non variano nei successivi lanci del software non ha senso aspettare tutte le volte l'autosettaggio e il programma risulta così più veloce.

4.3.2. Calibrazione del modello

Come specificato nel capitolo 3 per passare dal spazio tridimensionale visto dalla telecamera ad uno spazio bidimensiale, che è poi l'immagine prodotta dalla telecamera, è necessario risolvere un sistema ove i coefficienti dipendono dai parametri estrinseci ed intrinsici dei sistemi di coordinate utilizzati e dalla telecamera.

In questo lavoro si utilizzato un'approssimazione di tipo lineare arrivando ad un sistema del tipo:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Come scritto nel capitolo 3 la matrice M di proiezione non è invertibile, ma considerando che si lavora a z costante questo sistema può essere scritto come:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & \hat{m}_{13} \\ m_{21} & m_{22} & \hat{m}_{23} \\ m_{31} & m_{32} & \hat{m}_{33} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ 1 \end{bmatrix}$$

ove i coefficienti rappresentati con il cappuccio sono del tipo:

$$\hat{m}_{ij} = m_{ij} + m_{ij+1} z_w$$

In questo modo la matrice di proiezione è invertibile.

Scopo della taratura è trovare i nove coefficienti della matrice di proiezione. Se, come nel nostro caso, la scena è nota allora il problema da risolvere è relativamente facile, infatti è sufficiente stabilire delle coppie di punti corrispondenti e quindi risolvere il sistema lineare. Per quanto riguarda il caso in questione si sono scelti tre punti sul piano di movimento del robot e quindi si è misurato i corrispondenti nell'immagine fornita dalla webcam. Per la risoluzione di tale sistema si è utilizzato Matlab, ma non è complicato eseguire i conti anche "a mano".

Se si desiderasse una precisione migliore del modello occorrerebbe trovare più coppie di punti e risolvere il sistema per esempio ai minimi quadrati.

Si ricorda inoltre che questo metodo approssima la lente della telecamera a lente sottile e questo porta a non considerare la distorsione radiale e il possibile decentramento del punto principale dall'asse ottico. Questi problemi si possono ridurre utilizzando appositi modelli.

4.4. Avvertenze

Questo lavoro ha portato alla creazione di un marker passivo come specificato nei capitoli precedenti il quale dovrebbe essere posto sopra il robot Marmot. In realtà tutto il sistema è stato sviluppato senza il robot Marmot e cioè appoggiando semplicemente il marker sul pavimento. Di conseguenza volendo usare il software sviluppato con il marker su Marmot si deve ritarare il sistema considerando che il marker si trova ad una quota diversa.

Per fare ciò è sufficiente risolvere il sistema lineare descritto nel paragrafo 4.3.2 considerando che il marker è ad una certa quota.

Avendo utilizzato un modello lineare nel raddrizzamento è ovvio che si è introdotto un errore dovuto alle varie approssimazioni che questo modello introduce. In questo lavoro sono stati presi tre punti e ovviamente la precisione nell'intorno di questi punti è molto buona mentre in lontananza di questi la precisione scende. Comunque nel caso in questione l'errore massimo che si è ottenuto è di circa quattro centimetri. Nel software sviluppato non si è tenuto conto di questo errore o meglio l'indice di affidabilità che viene indicato è relativo solo al grado di assomiglianza tra il marker caricato e il marker trovato in figura e quindi a questo andrebbe aggiunto l'errore dovuto al raddrizzamento. Si potrebbe aggiungere l'errore di raddrizzamento semplicemente dividendo il piano di movimento del robot in quadrati e per ogni quadrato associare un errore (errore massimo misurato) e quindi una volta trovato il marker si può aggiungere al grado di affidabilità un errore di posizione che è esattamente l'errore associato al quadrato in cui il marker viene trovato.

Per quanto riguarda la Connectix Color Quickcam è possibile che all'accensione del calcolatore non venga rilevata dal sistema operativo. Questo è dovuto all'autoprobe di Linux. Il motivo di questo fatto è che la webcam in questione quando alimentata passa continuamente dallo stato command-mode al picture-mode.

Quando la webcam è nello stato di picture-mode testa la quantità di luce sull'obiettivo e copia l'immagine in una sua memoria dinamica e quindi passa nello stato command-mode. In questo stato la webcam attende un comando dal calcolatore (per esempio una richiesta di cattura dell'immagine). Quando l'immagine nella memoria dinamica inizia a "scaricarsi" lo stato della webcam passa automaticamente nello stato picture-mode. La durata di questo stato è variabile poiché dipende dalla quantità di luce che colpisce l'obiettivo e quando questo stato dura poco, l'autoprobe di Linux non è in grado di capire che la webcam è installata.

5. Conclusioni

L'elaborato iniziato con l'installazione di una webcam nel laboratorio di Robotica Avanzata, è proseguito fino allo sviluppo di algoritmi che permettessero, tramite le immagini fornite dalla webcam, non solo di localizzare il robot Marmot ma anche di estrapolarne informazioni su posizione ed orientamento, grazie alla scelta di un marker in grado di ridurre i problemi dovuti all'orientamento e ad un lavoro di raddrizzamento dell'immagine. Inoltre i risultati ottenuti in termini di velocità computazionale e precisione si possono ritenere più che buoni.

6. Sviluppi futuri

Diversi sono i possibili sviluppi futuri su questo lavoro: sicuramente ci sono molte vie non esplorate per quanto riguarda il raddrizzamento dell'immagine e la stessa cosa si può dire sia sulla scelta del marker che di algoritmi per il suo ritrovamento.

In letteratura si trovano diversi algoritmi per un migliore raddrizzamento dell'immagine, algoritmi che usano approssimazioni migliori introducendo delle non linearità e considerando modelli più complessi che tengano conto anche di effetti di bordo e altro ancora. Inoltre ci sono algoritmi di autocalibrazione.

Per quanto riguarda il marker si ricorda che in questo lavoro si è utilizzato un marker di tipo passivo e bidimensionale e quindi anche sulla scelta del marker ci sono possibilità di sviluppo.

Il programma svoltosi prevede l'analisi di immagini trasformate in bianco e nero, e non consente nessuna elasticità di analisi sui colori. La scelta di lavorare in queste condizioni migliora le prestazioni degli algoritmi che risultano meno complessi e più veloci, ma dall'altra parte riducono la robustezza del programma stesso. Un miglioramento quindi ovvio sarebbe l'introduzione di un'analisi con toni di grigio o addirittura usando discriminazioni diverse con i colori.

L'analisi del marker secondario è stata effettuata solo per un vettore realizzato su un raggio di circonferenza fissa, anche questa soluzione è stata adottata per velocizzare l'algoritmo, ma l'analisi risulta facilmente soggetta ad errore, un'altro miglioramento al progetto sarebbe quello di irrobustire in generale l'algoritmo di ricerca del marker secondario, richiamando la procedura con raggi intorno a quello della circonferenza principale.

Un ulteriore miglioramento potrebbe essere fatto nella ricerca del marker principale, l'algoritmo sviluppato non fa una discriminazione sui punti del marker di riferimento e li valuta tutti con lo stesso peso e importanza, non tenendo in considerazione che sul contorno del marker di solito si accumula più errore dovuto al fatto che nell'immagine si trova l'approssimazione di un cerchi che tende ad avere dei contorni non perfetti.

Nel caso preso in esame da questo elaborato la telecamera è situata in una posizione ottimale, questo favorisce sicuramente il progettista nel creare gli algoritmi per la ricerca del marker, ma vincola di molto la portabilità dell'intero progetto, un ulteriore sviluppo potrebbe garantire di usare telecamere con diversi orientamenti tenendo in considerazione gradi di affidabilità che non risentano di un eccessivo effetto prospettico.

Non da ultimo c'è la possibilità di estendere questo lavoro all'utilizzo di più telecamere il che permetterebbe di lavorare in un ambiente a tre dimensioni.

7. Esempio di funzionamento

Di seguito si forniscono due esempi di posizionamento del marker nella regione catturata dalla webcam installata nel laboratorio di Robotica Avanzata e l'uscita a video del software sviluppato in corrispondenza dei due casi.

Primo caso.

L'immagine fornita dalla webcam è mostrata in Figura 3

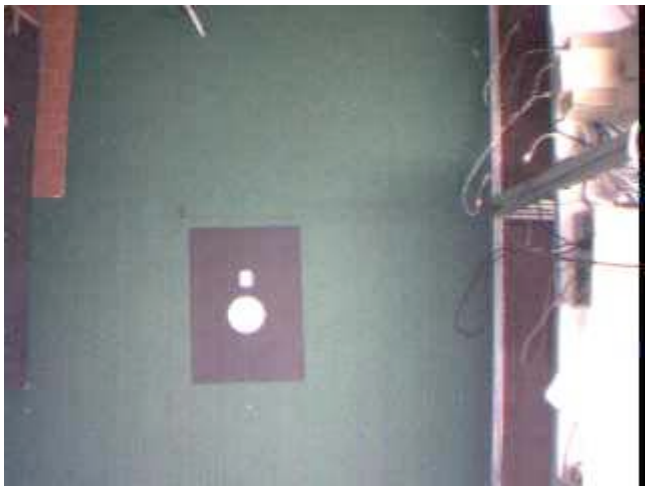


Figura 3: immagine fornita dalla webcam

L'uscita fornita dal software sviluppato è il seguente:

```
[arrigoni@nofive Robotica2]$ ./lancio
autosetting e salvataggio dei parametri

MARKER TROVATO!!
Coordinate x,y: 1174mm,1508mm (Affidabilità: 92.19%)
Rotazione del Marker: 91.10° (Affidabilità: 100.00%)

MARKER TROVATO!!
Coordinate x,y: 1174mm,1508mm (Affidabilità: 92.01%)
Rotazione del Marker: 91.10° (Affidabilità: 100.00%)

MARKER TROVATO!!
Coordinate x,y: 1174mm,1508mm (Affidabilità: 91.84%)
Rotazione del Marker: 91.10° (Affidabilità: 100.00%)
```

Secondo caso.

In Figura 4 è rappresentata l'immagine proveniente dalla webcam.

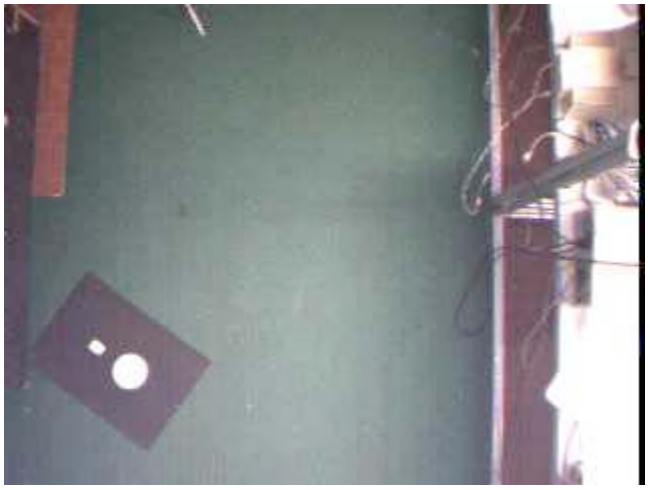


Figura 4: immagine fornita dalla webcam

L'uscita corrispondente a questo secondo caso è:

```
[arrigoni@nofive Robotica2]$ ./lancio
autosetting e salvataggio dei parametri

MARKER TROVATO!!
Coordinate x,y: 639mm,1803mm (Affidabilità: 93.58%)
Rotazione del Marker: 143.24° (Affidabilità: 94.44%)

MARKER TROVATO!!
Coordinate x,y: 639mm,1803mm (Affidabilità: 93.06%)
Rotazione del Marker: 145.54° (Affidabilità: 100.00%)

MARKER TROVATO!!
Coordinate x,y: 639mm,1803mm (Affidabilità: 93.58%)
Rotazione del Marker: 143.24° (Affidabilità: 94.44%)

MARKER TROVATO!!
Coordinate x,y: 639mm,1803mm (Affidabilità: 93.40%)
Rotazione del Marker: 145.54° (Affidabilità: 100.00%)
```

8. Codice

In questo capitolo si riporta tutto il codice creato per questo lavoro e cioè i due script per il lancio del programma e i due codici scritti in C.

8.1. Script primo_via

```
#! /bin/bash

#script di primo lancio per l'autosettaggio

CQCAMDIR="/usr/local/bin"
LAVORODIR="/home/herbie/arrigoni/robotica"

echo aggiustaggio
```

```
$CQCAMDIR/cqcam -a+ -r -j > $LAVORODIR/foto1.jpg 2 > /dev/null
sleep 2

#fine dello script
```

Come già specificato in precedenza questo script viene lanciato all'inizio del programma e ha come scopo l'autosettaggio della webcam e di salvare i parametri in un file così che nei lanci successivi la webcam userà questi parametri senza perdere tempo per il settaggio.

8.2. Script secondo_via

```
#! /bin/bash

#script di lancio del programma dopo l'autosettaggio

CQCAMDIR="/usr/local/bin"
LAVORODIR="/home/herbie/arrigoni/robotica"

$CQCAMDIR/cqcam -a- -j > $LAVORODIR/foto1.jpg 2 > /dev/null
sleep 2
djpeg -grayscale -bmp $LAVORODIR/foto1.jpg > $LAVORODIR/foto1.bmp
tail --bytes 76800 $LAVORODIR/foto1.bmp > $LAVORODIR/foto1.raw
$LAVORODIR/./El_robot2 foto1.raw mark.raw

#fine dello script
```

Scopo di questo script è di catturare l'immagine e salvarla in un file mediante il software "Cqcam", quindi di convertire questa immagine nel formato (raw) che è quello che serve al programma "El_robot" lanciato in seguito.

8.3. Lancio.c

```
#include<unistd.h>
#include<termios.h>

char buf[16];
int fd, n;

main()
{
    struct termios new;
    struct termios old;
    fd = 0;

    if (tcgetattr(fd, &old) == -1)
      perror("tcgetattr");
    if (tcgetattr(fd, &new) == -1)
      perror("tcgetattr");

    new.c_lflag &= ~ (ICANON | ISIG);
    new.c_lflag |= (ECHO | ECHOCTL);
    new.c_iflag = 0;
    new.c_cc[VMIN] = 0;
    new.c_cc[VTIME] = 0;

    if (tcsetattr(fd, TCSAFLUSH, &new) == -1)
       perror("tcgetattr");
```

```
    system ("./first_go.sh");

    while(1) {
       n = read(fd, buf, 1);
       if (n == 1)
          break;
       else system("./second_go.sh");
    }

    if (tcsetattr(fd, 0, &old) == -1)
       perror("tcsetattr");
    exit(0);

}
```

Questo codice è il codice del programma “lancio” e ha il compito prima di tutto di lanciare lo script primo_via per l’autosettaggio della webcam e quindi di mandare in esecuzione infinita lo script secondo_via. Per uscire da questo ciclo infinito è sufficiente premere un tasto.

8.4. El_Robot.c

```
/*
ELABORATO DI ROBOTICA
PROGRAMMA DI RICERCA DI UN MARKER PASSIVO
ALL'INTERNO DI UN'IMMAGINE PRESA DA WEBCAM
DOCENTE: CASSINIS RICCARDO
AUTORI: ARRIGONI MARCO
        PAROLA VALERIO

*/


// Librerie da includere nel programma
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

// Definizione delle costanti
#define xfoto 320             // Larghezza immagine
#define yfoto 240             // Altezza immagine
#define xcampo 320            // Larghezza immagine in bianco e nera
#define ycampo 240            // Altezza immagine in bianco e nera
#define soglia 200            // Livello di grigio oltre il quale si
traforma in bianco e nero
#define xmarker 24            // Larghezza Marker
#define ymarker 24            // altezza Marker
// mm = mini marker o marker secondario, utile per determinare
rotazione
#define raggio_mm 20          // Raggio della circonferenza del
marker secondario
#define larg_mm 18            // Larghezza del vettore da controllare
del marker secondario
#define bordo_nero_mm 6       // Bordo nero del vettore sia a destra
che sinistra
#define tolleranza 0.2        // tolleranza della correlazione mini
marker
```

```
#define scala 9.5416667       // Scala di conversione per la
linearizzazione

// Prototipi Funzioni

// Inizializza la matrice campo tutta nera
void init_campo(unsigned char campo[][ycampo]);
// Converte l'immagine in bianco e nero usando la soglia per
discriminare il colore
void bianconero(unsigned char foto[][yfoto]);
// Raddrizza l'immagine e il risultato lo colloca in campo
void mappatura(unsigned char foto[][yfoto], unsigned char
campo[][ycampo]);
// Carica l'immagine di riferimento del marker principale
void carica_mark(char nome_file[80],unsigned char mark[][ymarker]);
// Carica l'immagine in cui controllare la presenza del marker
void carica_foto(char nome_file[80],unsigned char foto[][yfoto]);
// Salva i risultati della linearizzazione (solo per verifica)
void salva_campo(unsigned char campo[][ycampo]);
// Salva l'immagine caricata (solo per verifica)
void salva_foto(unsigned char foto[][yfoto]);
// Procedura che cerca il marker principale
void trova_marker(unsigned char campo[][ycampo]);
// Effettua una correlazione tra marker nell'immagine e marker di
riferimento
double correla(int x, int y, unsigned char campo[][ycampo]);
// Cerca il marker secondario
int rotazione(int x, int y, unsigned char campo[][ycampo]);
// effettua una correlazione tra il vettore della circonferenza e il
marker secodario di riferimento
double correla_mm(int k, int circ[],double *angolo);


// Costanti della matrice di linearizzazione
const double M[3][3] = { 0.97291950587 ,0.0279157691124,0.1325009  ,
                        -0.03424627423 ,0.991471403    ,8.846133556,
                         0.00000000000 ,0.000000000    ,1.000000000
};


// Marker secondario di riferimento
const  int  MM[larg_mm] =
{0,0,0,0,0,0,255,255,255,255,255,255,0,0,0,0,0,0};

// Variabile globale Marker di riferimento
unsigned char  MARK[xmarker][ymarker];

void main(int argc, char *argv[])
{
  int i,j;
  unsigned char foto[xfoto][yfoto];
  unsigned char campo[xcampo][ycampo];
  if (argc<3)
      printf("\nERRORE PARAMETRI NON PRESENTI\nes. El_robot
nome_foto.raw nome_marker.raw\nfoto: immagine a 320x240\nmarker:
immagine 24x24\n\n");
  else
      {
      carica_foto(argv[1],foto);
      carica_mark(argv[2],MARK);
```

```
      init_campo(campo);            // Inizializza la matrice campo
con tutti zeri
      bianconero(foto);             // Trasforma l'immagine in bianco
e nero
      mappatura(foto, campo);       // Raddrizza l'immagine
      trova_marker(campo);          // Trova le coordinate del marker
e la rotazione con relativi gradi di affidabilità
      }
}



void init_campo(unsigned char campo[][ycampo])
// inizializza il vettore campo a zero
{
   int i,j;
   for(i=0;i<xcampo;i++)
     for(j=0;j<ycampo;j++)
      campo[i][j]=0;
}

void bianconero(unsigned char foto[][yfoto])
// Utilizza la soglia per trasformare la foto in un'immagine a 2
colori 0=Nero 255=Bianco
{
  int i,j;
  for(j=0;j<yfoto;j++)
     for(i=0;i<xfoto;i++)
        {
        if (foto[i][j]<soglia) foto[i][j]=0;
        else foto[i][j]=255;
        }
}

void mappatura (unsigned char foto[][yfoto], unsigned char
campo[][ycampo])
// Raddrizza la foto e il risultato lo mette nella matrice campo
{
   int u,v,X,Y;
   double s,t,z,w,fx,fy;
   for(v=0;v<yfoto;v++)
   {
     for(u=0;u<xfoto;u++)
     {
        if(foto[u][v]==255)
          {
          // Applicazione delle formule per il raddrizzamento
          z=M[0][0]*(u+1)+M[0][1]*(v+1)+M[0][2];
          w=M[1][0]*(u+1)+M[1][1]*(v+1)+M[1][2];
          fx=modf(z, &s);
          fy=modf(w, &t);
          X=(int)s;
          if(fx>=0.5) X++;
          Y=(int)t;
          if(fy>=0.5) Y++;
          // printf("X,Y,u e v valgono %d %d %d %d\n", X,Y,u,v); //
stampa i punti validi x Debug
          // Rimappo da immagine a campo
          if (X>0  && X<xcampo && Y>0 && Y<ycampo)
                   campo[X][Y]=255;
```

```
            //  else
            //      printf("La matrice di Rotazione Va oltre i
limiti di campo %d %d \n",X,Y );
            // stampa i punti fuori campo
          }
     }
   }
}

void carica_mark(char nome_file[80],unsigned char mark[][ymarker])
// Carica il marker da file RAW a 256 toni di grigio
{
 FILE *fp;
 int i,j;

 if((fp=fopen(nome_file,"rb"))==NULL) {
     printf("Errore di apertura del file MARKER!\n");
     exit(1);
     }
  for (j=0;j<ymarker;j++)
     for (i=0;i<xmarker;i++)
        fread(&mark[i][j], sizeof(char), 1,fp);
  fclose(fp);
}

void carica_foto(char nome_file[80],unsigned char foto[][yfoto])
// Carica il file con l'immagine RAW a 256 toni di grigio
{
 FILE *fp;
 int i,j;

 if((fp=fopen(nome_file,"rb"))==NULL) {
     printf("Errore di apertura del file FOTO!\n");
     exit(1);
     }
  for (j=0;j<yfoto;j++)
     for (i=0;i<xfoto;i++)
        fread(&foto[i][yfoto-j-1], sizeof(char), 1,fp);
  fclose(fp);
}

void salva_campo(unsigned char campo[][ycampo])
// Salva la matrice campo, questa funzione serve solo per una
eventuale verifica del raddrizzamento
{
 FILE *fp;
 int i,j;
 if((fp=fopen("bin-map.raw","wb"))==NULL) {
     printf("errore di salvataggio file!!!!\n");
     exit(1);
     }
  for (j=0;j<240;j++)
     for (i=0;i<320;i++)
        fwrite(&campo[i][j], sizeof(char), 1,fp);

  fclose(fp);
}

void salva_foto(unsigned char foto[][yfoto])
```

```
// Salva la matrice foto, questa funzione serve solo per verifica del
corretto caricamento della foto
{
 FILE *fp;
 if((fp=fopen("foto_new.raw","wb"))==NULL) {
     printf("errore di salvataggio file!!!!\n");
     exit(1);
     }
  fwrite(foto, 1, 76800,fp);
  fclose(fp);
}

void trova_marker(unsigned char campo[][ycampo])
// Permette di determinare il marker principale nella matrice campo e
richiama la procedura
// di determinazione della rotazione del marker analizzando il mini
marker
{
   int punti_utili_x[768];          // coordinate x punti utili
   int punti_utili_y[768];          // coordinate y punti utili
   int k=0;                         // contatore punti utili max 768
   int x1,x2,y1,y2;                 // contatori punti bianchi
intorno ai punti utili
                                    // x1->,x2<-,y1 ^,y2 v
   int x1v,x2v,y1v,y2v;             // Variabili di controllo fine
dei
                                    // punti bianchi
   int step=int(xmarker/2);         // Step di ricerca dei punti
utili
   int xcentro,ycentro;             // cooordinate del centro del
marker
   double aff;                      // Affidabilità dei risultati
   int j,i;                         // Contatori per i cicli
   int goal=0;                      // Se vale 1 ho trovato il marker

   if (step<10) step=10;            // Step minimo 10 pixel per
trovare i punti utili

   for(j=0;j<ycampo;j=j+step)       // scansiono l'immagine con step
xmarker/2
      for(i=0;i<xcampo;i=i+step)    // se trovo un 255 (bianco)
memorizzo le x,y
         if (campo[i][j]==255)
           {
            punti_utili_x[k]=i;
            punti_utili_y[k]=j;
            k++;
           }

   // printf("Punti utili %d!!!\n",k); // Stampa il numero di punti
utili fase di debug

   // Per ogni punto utile trovo le distanze dal punto all'ipotetica
circonferenza del marker
   for (j=0;j<k;j++)
      {
      x1v=0;
      x2v=0;
      y1v=0;
      y2v=0;
```

```
      x1=0;
      x2=0;
      y1=0;
      y2=0;
      for(i=1;i<xmarker;i++) // conta i bianchi adiacenti
         {
          if (((punti_utili_x[j]+i)<xcampo) &&
(campo[punti_utili_x[j]+i][punti_utili_y[j]]==255) && (x1v==0)) x1++;
else x1v=1;
          if (((punti_utili_x[j]-i)>=0) && (campo[punti_utili_x[j]-
i][punti_utili_y[j]]==255) && (x2v==0)) x2++; else x2v=1;
          if (((punti_utili_y[j]-i)>=0) &&
(campo[punti_utili_x[j]][punti_utili_y[j]-i]==255) && (y1v==0)) y1++;
else y1v=1;
          if (((punti_utili_y[j]+i)<ycampo) &&
(campo[punti_utili_x[j]][punti_utili_y[j]+i]==255) && (y2v==0)) y2++;
else y2v=1;
         }

      // Calcolo le coordinate del centro del cerchio
      xcentro=punti_utili_x[j]-x2+(int)((x1+x2)/2);
      ycentro=punti_utili_y[j]-y1+(int)((y1+y2)/2);

      // printf("Punto X,Y %d,%d - XCENTRO,YCENTRO %d,%d - x1,y1
x2,y2 (%d,%d) (%d,%d)\n",punti_utili_x[j],
punti_utili_y[j],xcentro,ycentro,x1,y1,x2,y2); // Stampa le
coordinate dei punti utili

      aff=correla((xcentro-(int)(xmarker/2)),(ycentro-
(int)(ymarker/2)),campo);

      // se la correlazione tra marker e immagine è >85% esco dal
ciclo
      if (aff>0.85)
        {
        printf("\nMARKER TROVATO!!\n");
        printf("Coordinate x,y: %.0fmm,%.0fmm
",((double)xcentro*scala),((double)ycentro*scala));
        printf("(Affidabilità: %.2f%%)\n",(aff*100));
        if  (rotazione(xcentro,ycentro,campo)==1)
         {
          goal++;
          break;
         }
        }
      }
   if (goal==0) printf("MARKER NON TROVATO!!!\n");
}

double correla(int x,int y, unsigned char campo[][ycampo])
// Controlla la corrispondenza tra il marker nel campo e quello nella
matrice MARK
// x,y sono le coordinate dell'angolo alto a dx dell'immagine del
marker
// La correlazione è un'analisi punto a punto
{
   int j,i,k=0;
   double indice;
   for(j=0;j<xmarker;j++)
      for(i=0;i<ymarker;i++)
```

```
         if (campo[x+i][y+j]==MARK[i][j]) k++;
   indice= ((double)k)/(xmarker*ymarker);

   // printf("x: %d, y: %d, indice: %f\n",x,y,indice); //fase di
debug

   return indice;
}

int rotazione(int x,int y, unsigned char campo[][ycampo])
// Determina la rotazione del Marker cercando intorno al marker il
mini marker
{
   int circ[314];             // Vettore dei punti sulla
circonferenza intorno al marker principale
   int x1,y1;                 // Coordinate intere per posizione dei
pixel sulla circonferenza
   int x1_old=-1,y1_old=-1;   // Coordinate vecchie della
circonferenza da cui creo un vettore
   double aff,rad;            // Affidabilità, radianti
   int goal=0;                // se vale 1 ho trovato il marker
secondario
   int j;                     // Contatore per cicli

   // Prendo 314 punti sulla circonferenza escludendo di riprendere
lo stessoo punto
   // più volte
   for(j=0;j<314;j++)
      {
      // Coordinate della circonferenza intorno al centro del marker
principale
      x1=x+(int)(cos((double)(j)/50)*raggio_mm);
      y1=y-(int)(sin((double)(j)/50)*raggio_mm);
      // Prendo il punto una sola volta
      if ((x1!=x1_old) || (y1!=y1_old))
         {
          if ((x1>=0) && (x1<xcampo) && (y1>=0) && (y1<ycampo))
                circ[j]=campo[x1][y1];
          else circ[j]=-2;  // -2 indica che il punto è fuori matrice
campo
          x1_old=x1;
          y1_old=y1;
         }
      else circ[j]=-1;   // -1 indica che il valore corrisponde alle
coordinate del punto precedente

      // rad=(double)j/50;  // serve per il debug
      //printf("circ[%d] = %d - x1,y1 = %d,%d -  rad:
%.4f\n",j,circ[j],x1,y1,rad); // serve per debug
      }

   // analizzo la matrice circ trovo il mini marker o marker
secondario
   // cerco una riga bianca con due bande nere di dimensioni da me
ipotizzate
   for(j=0;j<314;j++)
     if (circ[j]==255)
        {
        aff=correla_mm(j,circ,&rad);
        if (aff>=(1-tolleranza))
```

```
           {
            printf("Rotazione del Marker: %.2f° ",rad);
            printf("(Affidabilità: %.2f%%)\n\n",(aff*100));
            goal++;
            break;
           }
         }
   if (goal==0) printf("Marker di rotazione non trovato!!!\n\n",aff);
   return(goal);
}

double correla_mm(int k, int circ[],double *angolo)
// Controllo la somiglianza tra il vettore circ e MM (mini marker)
// cont1 conta i -1 nel vettore circ e corregge il controllo
// salto permette di passare dal primo all'ultimo campo del vettore
{
  int j,salto;
  int cont1;
  int errore;
  double aff;
  int inizio,fine;

      salto=0;
      inizio=0;
      fine=0;
      cont1=0;
      errore=0;
      j=0;
      while (j<larg_mm-bordo_nero_mm)
         {
         if ((k+j+cont1)>=314) salto=-314;
         if (circ[k+j+cont1+salto]!=-1)
            {
            if (circ[k+j+cont1+salto]!=MM[j+bordo_nero_mm]) errore++;
            j++;
            }
         else cont1++;
         }
      fine=k+j+cont1;

      salto=0;
      cont1=0;
      j=0;
      while (j<bordo_nero_mm)
         {
         if ((k-j-cont1)<0) salto=314;
         if (circ[k-j-cont1-1+salto]!=-1)
            {
            if (circ[k-j-cont1-1+salto]!=MM[bordo_nero_mm-j-1])
errore++;
            j++;
            // printf("circ: %d  MM: %d\n",circ[k-j-cont1-
1],MM[bordo_nero_mm-j-1]); // fase di debug
            }
         else cont1++;
         }
      inizio=k-j-cont1;

  *angolo = ((((double)fine/50)-
((double)inizio/50))/2+(double)inizio/50)*(180/3.1415);
```

```
  aff=(larg_mm-(double)errore)/larg_mm;
  // printf("errore: %d fine: %d inizio: %d aff:
%f\n",errore,inizio,fine,aff); // fase di debug
  return (aff);
}
```

Bibliografia

[1] Trucco, E.: “Introductory Techniques for 3-D Computer Vision”, *Prentice Hall*, Canada, 1998.

[2] Nielsen, E.: “Automatic Guidance of Vehicles using Vision and Projective Invariant Marking”, *Automatica* Vol. 24, No.2, 1988.

Indice

SOMMARIO 1

1. INTRODUZIONE 1

2. IL PROBLEMA AFFRONTATO 1
2.1. Installazione della webcam 1
2.2. Scelta del marker 2
2.3. Algoritmo di ricerca 2

3. LA SOLUZIONE ADOTTATA 2
3.1. Installazione della webcam 3
3.2. Scelta del marker 4
3.3. Conversione del formato immagine 5
3.4. Ricerca del marker 7
3.4.1. Algoritmo di raddrizzamento 7
3.4.2. Algoritmo di ricerca 10

4. MODALITÀ OPERATIVE 13
4.1. Componenti necessari 14
4.1.1. Componenti hardware 14
4.1.2. Componenti software 14
4.2. Modalità di installazione 15
4.2.1. Installazione di Cqcam 16
4.2.2. Installazione del software sviluppato 16
4.3. Modalità di taratura 17
4.3.1. Settaggio di Cqcam 17
4.3.2. Calibrazione del modello 18
4.4. Avvertenze 19

5. CONCLUSIONI 20

6. SVILUPPI FUTURI 20

7. ESEMPIO DI FUNZIONAMENTO 21

8. CODICE 22
8.1. Script primo_via 22
8.2. Script secondo_via 23
8.3. Lancio.c 23
8.4. El_Robot.c 24

BIBLIOGRAFIA 32

INDICE 33