POLLICINO

Relazione del progetto di robotica di
Bonassi Daniele (matricola 25046) e
Spinoni Carlo (matricola 24986)
Anno accademico 97/98

Indice

Introduzione

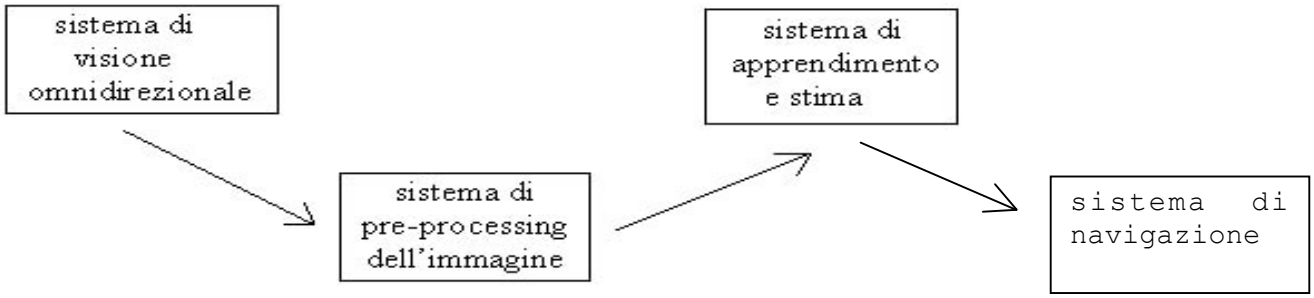
Il progetto POLLICINO vuole verificare la possibilità di autolocalizzare e quindi pianificare la navigazione autonoma di un robot. Per realizzare tutto ciò ci si avvale di una rete neurale e di una minitelecamera. Prima della navigazione vera e propria, la rete neurale è addestrata tramite una serie di immagini prese nell'ambiente operativo (non condizionato). Grazie a ciò, la rete è in grado di riconoscere elementi naturali presenti nell'intorno della posizione del robot e di associare ad ogni immagine presa dalla telecamera una coppia di coordinate X,Y. L'utilizzo di una rete neurale implica una certa robustezza nel sistema. Questa robustezza si traduce in una stima accurata della posizione del robot anche quando siano introdotte nell'ambiente operativo delle perturbazioni volute o accidentali che modifichino moderatamente il campo di visione della telecamera.
POLLICINO acquisisce immagini degli oggetti presenti attorno ad esso senza apprenderli né riconoscerli esplicitamente. Il sistema non si costruisce alcuna mappa dell'ambiente in cui si trova, il suo compito è semplicemente associare le caratteristiche dell'ambiente con la posizione attuale del robot. Le caratteristiche dell'ambiente sono di tipo visuale e omnidirezionale e possono essere le più varie : aperture nelle pareti quali porte o finestre, sedie o tavoli, armadi, zoccoli delle pareti, ecc. L'omnidirezionalità delle immagini è ottenuta tramite uno specchio conico che riflette l'immagine dell'ambiente circostante. La rete, una volta addestrata, viene utilizzata nella fase operativa o di navigazione vera e propria.
Sul robot viene fissata una quickcam ed un cono riflettente con lo scopo di catturare le immagini. Queste immagini devono essere trasformate in un vettore opportuno e poi passate come ingresso ad una rete neurale che ha lo scopo di restituire al processo chiamante le coordinate X,Y stimate. In base a queste coordinate e alla posizione dell'obiettivo che si desidera raggiungere, si riesce ad impostare una navigazione.
Prima della navigazione vera e propria, viene eseguita una fase di addestramento della rete neurale. L'addestramento è necessario per fornire alle connessioni dei pesi opportuni che riflettano in qualche modo le caratteristiche delle immagini prese dalla telecamera.
Per procedere all'addestramento, viene scelto un ambiente operativo, viene fissata una griglia di punti in cui verranno prese le immagini e quindi viene creata tutta una serie di directory e file di lavoro che serviranno per configurare correttamente il simulatore di rete neurale.
Una volta impostato il sistema, si può procedere all'addestramento vero e proprio della rete neurale tramite un tool presente nel simulatore. L'uscita di questo tool è una rete neurale addestrata per l'ambiente operativo scelto, ovvero una rete in cui ogni connessione ha un peso di valore ben definito. Questa rete può essere memorizzata su file e poi utilizzata per la navigazione. A questo punto la fase di addestramento è terminata e si può passare alla navigazione vera e propria.
Brevemente, le fasi principali che il sistema è chiamato ad espletare durante una navigazione sono :

1. inserimento delle coordinate della posizione iniziale
2. acquisizione di una immagine e preprocessing
3. stima della posizione tramite la rete neurale
4. inserimento delle coordinate della posizione successiva
5. spostamento del robot nella posizione successiva in base ai valori di dx e dy
6. torna al passo 2 fino a che non si è raggiunta la posizione finale

i valori dx e dy rappresentano il passo del robot in un sistema di coordinate rettangolare. In particolare :

dx = X_posizione_successiva - X_posizione_stimata_attuale
dy = Y_posizione_successiva - Y_posizione_stimata_attuale

Uno schema a blocchi del sistema che evidenzia le funzionalità principali è invece illustrato in figura :



Sono ora descritti brevemente i quattro blocchi funzionali schematizzati sopra (per una descrizione più dettagliata, si faccia riferimento al paragrafo *descrizione dell'ambiente operativo*):

1. Il sistema di visione omnidirezionale consiste in uno specchio conico, una quickcam e un sostegno regolabile per la messa a fuoco. Il loro montaggio, fatto tramite una squadretta di metallo, può essere visto nella figura che rappresenta l'hardware del sistema.
2. Il sistema di pre-processing delle immagini è un insieme di script di shell che accetta come ingresso l'output della quickcam e lo elabora trasformandolo in un vettore di 360 elementi per ogni componente cromatica : R, G, B. Il risultato viene in seguito normalizzato tra -1 e +1 per fornire degli ingressi adeguati alla rete neurale e salvato su file. Quest'ultimo deve essere fornito come ingresso alla rete neurale e perciò deve avere uno schema ben definito a seconda che l'immagine serva come pattern di addestramento o come ingresso vero e proprio di una rete già addestrata.
3. Il sistema di apprendimento e di stima della posizione è costituito da una rete neurale dalla struttura predefinita (file .net) e da un simulatore di rete neurale (SNNS). In particolare, del simulatore, ci interessa il tool che addestra una rete in background ed il tool che converte le reti in funzioni C inseribili agevolmente in un programma.
4. Il sistema di navigazione è un programma (linguaggio C) in cui è codificata la strategia di navigazione, le istruzioni per collegarsi a Saphira e fornire comandi al robot, i calcoli degli errori tra posizione stimata e posizione attuale.

Descrizione degli obiettivi

Lo scopo delle esperienze fatte è quello di verificare se l'errore cumulativo, cumulativo teorico e puntuale (o locale), si mantengono limitati, crescono oppure diminuiscono in valore assoluto durante i vari passi di una navigazione. Questo, comunque, può non precludere una corretta navigazione se il robot riesce a compensarli tra un passo e l'altro. Statisticamente è infatti previsto che gli errori compiuti ad ogni passo non abbiano sempre la stessa direzione e verso; è quindi prevista una compensazione in un certo numero di passi di modo che si riesca sempre ad arrivare alla stima della posizione del passo successivo.

Il sistema è affetto principalmente da due fonti di potenziali errori :

- Errori di odometria causati dal pattinamento delle ruote possono portare il robot in posizioni troppo fuori asse e dalla notevole inerzia del sistema che si è rivelato abbastanza pesante.
- Perturbazioni dell'ambiente come la presenza di persone, la variabilità dell'illuminazione tra una prova e l'altra, ecc possono influire (anche se marginalmente) sulle risposte della rete neurale.

Uno scopo secondario è quindi quello di rendere minima l'influenza di questi errori sul risultato finale.

Descrizione dell'ambiente operativo

Questo capitolo della relazione si occupa di descrivere approfonditamente la parte hardware e software del sistema POLLICINO.

Descrizione del software

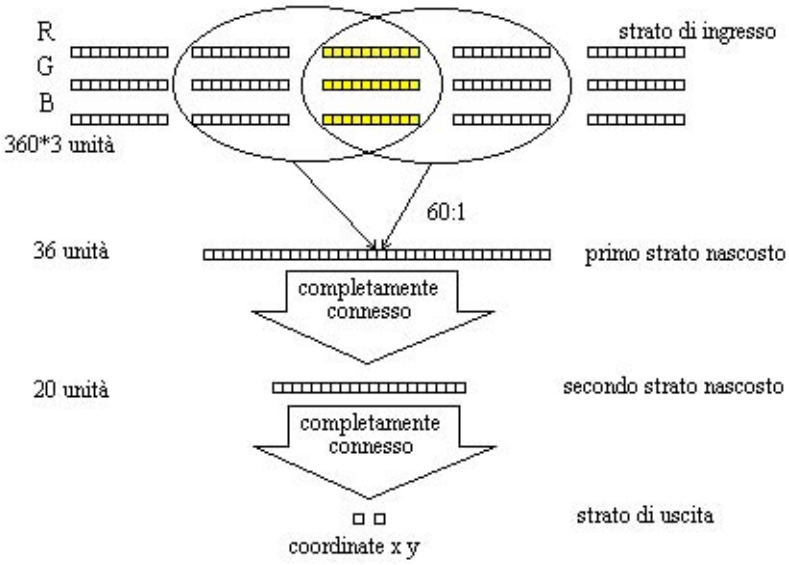
Il software utilizzato si compone di :

- script della shell di Linux
- programmi scritti in linguaggio C
- driver per la quickcam
- SNNS e tools collegati (*snnsbat, snns2c*)
- Rete neurale (*std3bp*)
- Ambiente di lavoro del robot: Saphira

In generale gli script servono per la creazione delle directory di lavoro, per la lettura, modifica e scrittura dei file delle immagini, per creare i file di log e di uscita, per invocare altri programmi (come appunto SNNS) con gli opportuni parametri d'inizializzazione e file di ingresso formattati a dovere.
I programmi scritti in linguaggio C servono a stabilire il piano di navigazione del robot, calcolare gli errori commessi dalla rete neurale, interagire con l'utente e, soprattutto, con Saphira per impartire i comandi al robot.
Il software della quickcam serve per catturare le immagini e tarare i parametri in modo da avere una qualità accettabile.
SNNS (Stuttgart Neural Network Simulator, realizzato all'università di Stuttgart all'Institute for parallel and distributed high performance system) è un ambiente di lavoro molto ricco e sofisticato per lavorare con reti neurali e simularne il comportamento. Oltre alle più comuni features (creazione, simulazione dell'apprendimento ed esecuzione delle reti neurali), SNNS offre una serie di tools che si sono rivelati molto utili nel corso del progetto:

- *Snnsbat* è un tool che permette di addestrare una rete in background ovvero senza dover aprire l'IDE grafico. I parametri che gli servono per lavorare sono letti in un opportuno file (*batch.cfb*)
- *Snns2c* è un tool che consente la conversione della rete neurale addestrata in una funzione C integrabile in altri programmi tramite una semplice direttiva di #include.

La rete neurale STD3BP è stata progettata per lavorare su immagini bidimensionali a colori e fornire due uscite (le coordinate X e Y). Di seguito è riportata la sua struttura:



Saphira è l'IDE tramite il quale impartire ordini al robot. Una volta connessi (tramite seriale) con il robot, Saphira esporta una serie di funzioni che sono perfettamente integrabili in comuni programmi C e che consentono di dare al robot istruzioni riguardanti la direzione di movimento, la velocità di traslazione, la velocità di rotazione, l'angolo di rotazione e così via.
Procediamo ora alla descrizione dettagliata degli script e dei programmi C utilizzati per la realizzazione del sistema (addestramento e navigazione).

Messa_a_punto_cam

Questo script serve durante le fasi preliminari degli esperimenti. La quickcam è stata usata, oltre che per il progetto Pollicino, anche da altri gruppi. Per questa ragione, ogni volta che si accedeva al laboratorio di robotica occorreva montare nuovamente la quickcam sul suo supporto in modo che fosse orientata in maniera coassiale con il cono. Oltre a ciò era spesso necessario regolare il fuoco per ottenere immagini più nitide e regolare i parametri in modo da non avere immagini troppo chiare o troppo scure. Per svolgere le azioni sopra citate sono necessarie numerose prove e molto tempo.
Per velocizzare il lavoro si è scritto un breve e semplice script che contiene un ciclo infinito durante il quale l'utente deve decidere i valori dei parametri , il cui range è indicato tra parentesi, e poi vedere a video il risultato ed, eventualmente, modificarli al prossimo ciclo.
I parametri sono :

- Dimensione lungo l'asse x dell'immagine
- Dimensione lungo l'asse y dell'immagine
- Offset rispetto all'asse x
- Offset rispetto all'asse y
- Luminosità (brightness)
- Contrasto (contrast)
- Livello di bianco
- Black_offset
- Saturazione (saturation)
- Hue

- Livello di nero

Non è necessario impostare tutti i parametri. Ai parametri per i quali non si è specificato alcun valore particolare viene assegnato un valore di default letto nel file *qcam.conf* (directory */usr/local/lib*).
Alla fine dello script è lanciato il comando *qcread* con tutti gli opportuni parametri.
Nella versione allegata alla relazione, sono forniti due esempi di *qcread* che spesso danno immagini soddisfacenti.

Attendi.c

Il file *attendi.c* è un semplice programma in linguaggio C che esegue una pausa pari al numero di secondi indicato in argv[1] (primo parametro della riga di comando).
Si è preferito scrivere un breve programma C data la difficoltà nel trovare un opportuno comando della shell di linux che facesse altrettanto.

Shift.c

Questo file serve per tentare di rendere la rete neurale meno dipendente dai possibili errori nell'orientamento del robot. Si è riscontrato che anche piccoli errori nell'orientamento della quickcam rispetto alla direzione di riferimento possono portare a grossolani errori di valutazione della posizione. Per ridurre questo problema si è adottato il seguente stratagemma: per ogni immagine presa sono generati 10 vettori risultato. Tutti questi vettori sono dati in ingresso alla rete neurale con la stessa posizione X,Y. I 10 vettori non sono prodotti direttamente dall'immagine della quickcam. Tutte le immagini della quickcam sono trasformate in vettori e inserite, unitamente alla coppia di coordinate X,Y relativa, nel file *learn_piccolo.pat*. Questo file è aperto e letto da *shift.c*. *Shift.c* apre e scrive il file *learn.pat* copiandovi anche l'intestazione. Per ogni vettore di *learn_piccolo.pat* sono generati altri nove vettori (per un totale di 10) dati dallo shift delle componenti del vettore originario.
Questo metodo ci ha permesso di evitare di prendere un numero eccessivo di immagini di addestramento.

Stimaxy.c

È un programma a linea di comando che serve a stimare la posizione X,Y della quickcam. È stato realizzato nelle prime fasi del progetto per testare l'affidabilità della rete neurale ed è quindi completamente slegato dalle problematiche di movimentazione del robot.
Il programma viene invocato assieme al pattern che si vuole testare.
Dopo una prima parte dedicata al controllo della correttezza formale dei parametri passati, viene assegnato il pattern d'ingresso alla rete neurale (tramite un ciclo for avente come limite la variabile networkREC.NoOfInput). Segue l'invocazione della funzione network() con i propri parametri inizializzati. Dopo questa invocazione, la variabile netOutput contiene i valori normalizzati tra -1 e +1 della posizione stimata. Per ottenere i valori effettivi, si richiama la funzione *denormalizza()* la quale, facendo riferimento al file *scale.def* ricostruisce i valori della posizione stimata.
Il file *stimaxy.c* è il primo esempio di utilizzo delle funzionalità di SNNS e dei risultati degli script di addestramento della rete neurale.
Ricordiamo che, a questo punto del lavoro, l'addestramento della rete neurale deve essere completato e devono esistere tutti i file di supporto alla navigazione (tra i quali *scale.def* il quale contiene le dimensioni fisiche dell'area di lavoro).
Il tool di SNNS chiamato *SNNS2C* permette di trasformare la rete neurale dal formato interno di SNNS ad una funzione in linguaggio standard C. Questa funzione, unitamente alla dichiarazione

della variabile networkREC (la cui definizione è riportata sotto), può essere inserita in ogni programma C.

```
static struct {
                  int NoOfInput;    /* Number of Input Units  */
                  int NoOfOutput;   /* Number of Output Units */
                  int(* propFunc)(float *, float*, int);
                  } networkREC = {1080,2,network};
```

Agc

Lo script *agc* serve per normalizzare i valori letti e trasformati dalla quickcam in numeri compresi tra -1 e +1 che possano essere interpretati come ingressi dalla rete neurale. Per realizzare i necessari confronti e le opportune operazioni matematiche si è utilizzato il linguaggio awk che presenta una sintassi C-like. Lo script è invocato, dopo aver preso un'immagine, tramite il comando :

agc -1 +1 $nome_imm > $nome_i_rgb

I primi due valori rappresentano il range di ingresso alla rete neurale (-1 +1); il terzo parametro è il file da cui estrarre i dati da normalizzare; il quarto parametro è il file verso il quale indirizzare i valori normalizzati. Sarà quest'ultimo file ad essere letto dalla rete neurale.

Train_net

È lo script principale della fase di addestramento della rete neurale. Si compone essenzialmente di due parti: una prima parte per la creazione delle directory/sottodirectory di lavoro e la cattura/elaborazione delle immagini, ed una seconda parte di invocazione dei comandi e compilazione dei file per la successiva parte di navigazione.
I file di navigazione vanno ricompilati dopo ogni fase di addestramento perché la navigazione è basata su una rete neurale convertita in una funzione in linguaggio C. In particolare, la funzione C che rappresenta la rete neurale conterrà degli opportuni pesi che dipendono dall'addestramento. Rifare l'addestramento vuol dire creare una funzione C diversa da quella precedente (contenente pesi diversi) e quindi il programma di navigazione dovrà linkare una funzione diversa e ciò fa nascere l'esigenza di ricompilarlo ogni volta.
Train_net inizia acquisendo il nome della griglia di lavoro (variabile 'nomegriglia'). Successivamente vengono create le sottodirectory di lavoro invocando lo script *genenv*, sono impostati i permessi, creata una directory (*/nets*), vi vengono copiati i files della rete neurale non addestrata ed il tool di SNNS chiamato *snns2c*.
Segue un ciclo while in cui si ripetono le seguenti operazioni fino a che non si sono acquisite sufficienti immagini :

- Acquisizione immagine (*testall*)
- Normalizzazione dell'immagine
- Inserimento delle coordinate X,Y (in centimetri)
- Copia del file normalizzato in */data* e */maps*
- Inserimento dei dati nome_immagine e coordinate X,Y nel file *all.map* (directory */maps*)
- Rimozione del file dell'immagine dalla directory attuale

Ora viene creata la directory */test0* sotto la directory /test; questa directory contiene tutti i risultati dell'addestramento della rete neurale.
Viene poi invocato lo script *generamap_modificato* per la creazione del file *learn.map* che contiene, in sequenza, tutti i nomi dei vettori delle immagini e le loro coordinate. A questo punto, tutti i file

*.*map* e *.*def* sono copiati nella directory finale /*test0* e viene richiamato *genpat_modificato* che crea il file *learn.pat.* Quest'ultimo file contiene i vettori veri e propri assieme alle coordinate relative (e per questo motivo può risultare molto voluminoso, specie dopo la seguente fase di espansione). Tramite il programma *shi.exe* (eseguibile del file *shift.c*), il file *learn.pat* viene espanso per migliorare l'affidabilità della rete neurale e dell'intero sistema di navigazione.
Tornati nella directory di partenza (/*home/polli/bin*/), viene richiamato *regenera1test_modificato*.
Il tool *snnsbat* viene ricopiato nella /*test0* e viene invocato passandogli gli opportuni parametri : batch.cfb (file di configurazione per la modalità batch) e *oo.log* (file di uscita). A questo punto occorre attendere qualche minuto affinchè la rete neurale si addestri.
Se lo si desidera, è possibile consultare il file oo.log per vedere l'esito dell'addestramento.
Ottenuta la rete neurale addestrata, la si converte in un file (con sintassi C) chiamato network.c.
Tutti i file utili alla navigazione sono copiati nella directory */test0 : stimaxy.c, kaba.c, scale.def, testall, qcread, attendi.c, quick, agc* e tutti i makefile.
Infine, vengono compilati i file *kaba.c, stimaxy.c* e *network.c*. Arrivati a questo punto abbiamo tutti gli eseguibili per navigare nell'ambiente selezionato e non si deve fare altro che invocarli assieme a Saphira.

Genenv

Questo script crea tutte le directory di lavoro come sottodirectory di una radice la quale è fornita da linea di comando. In particolare la radice è un nome fornito dall'utente per identificare l'esperimento al quale sta lavorando. L'estensione '.env' (enviroment) viene aggiunta automaticamente e serve ad evidenziare quelle directory che identificano prove di lavoro dalle directory comuni.
Dopo una prima fase in cui si verifica se lo script è stato lanciato con i giusti parametri (funzione checkparam), vengono create le directory */data, /test, /maps*.
La prima contiene tutte le immagini, la directory /test contiene le sottodirectory dei test effettuati (es /test0, /test1, /testn) sullo stesso ambiente di lavoro e con la stessa rete neurale addestrata. Infine la directory /maps contiene vari files *(*.map, *.def*) che serviranno alla navigazione (la mappa dell'ambiente di navigazione consiste nell'insieme di coppie immagine-coordinate, i file di definizione indicano quali immagini vanno usate per l'addestramento della rete, quali sono le dimensioni dell'area di lavoro ecc.).

Genpat_modificato

Questo script, letto il file *learn.map*, costruisce il file *learn.pat*. Dopo gli opportuni controlli sui parametri attuali con i quali è richiamato, viene richiamata la funzione *genscale* che genera i valori di normalizzazione e poi viene chiamato *mksnnspat* che, tramite i valori contenuti nelle variabili $datatpath e $scale, è in grado di scrivere il file *learn.pat*.
Tutte le righe commentate si riferiscono ai comandi relativi alla fase di test della rete neurale. Tale fase, in quanto inessenziale, è stata omessa.

Kaba.c

Il programma *kaba.c* (il nome è stato scelto senza alcun motivo particolare) è il programma di navigazione vero e proprio; si differenzia da *stimaxy.c* per il fatto che, oltre a stimare la posizione, invia al robot tutti i comandi necessari a farlo muovere per raggiungere la sua destinazione.
Fondamentalmente, *kaba.c* esegue i seguenti compiti:

- Interfacciamento con l'ambiente Saphira
- Acquisizione/elaborazione delle immagini richiamando gli opportuni programmi
- Stima della posizione e calcolo delle distanze

- Movimentazione del robot in base ai valori calcolati al punto precedente

Così come *stimaxy.c*, anche *kaba.c* integra la funzione *network.c* (risultato della fase di addestramento) per poter stimare correttamente la posizione attuale. Se non si fosse potuta realizzare e integrare la funzione *network()* in *kaba.c*, la stima della posizione sarebbe dipesa da SNNS e avrebbe comportato la necessità di mantenere in memoria e attivo un pacchetto applicativo di dimensioni notevoli. Inoltre si sarebbero dovuti risolvere i problemi di interfacciamento tra SNNS e *kaba.c*. Le uniche variabili globali utilizzate sono X_stimato e Y_stimato. *Kaba.c* parte interfacciandosi a Saphira (lanciato Saphira, l'utente deve connettersi 'manualmente' con il robot). Segue il ciclo principale composto da un do-while che ha come condizione di terminazione il fatto che l'utente digiti qualcosa di diverso da 's' o 'S' alla richiesta di continuare il ciclo. Terminato il ciclo ci si disconnette da Saphira.
Le istruzioni contenute nel ciclo effettuano in sequenza le seguenti operazioni :

1. Acquisisci le coordinate dell'obiettivo da raggiungere con il passo attuale
2. Acquisisci l'immagine nel file IMM_ACQ.r_g_b (il file è sempre quello per ogni immagine campionata)
3. Stima delle coordinate fatta tramite la funzione ritorna_coordinate() e inserite nelle variabili globali X_stimato, Y_stimato
4. Echo a video di alcuni valori (X_stimato, Y_stimato, X_obiettivo, Y_obiettivo)
5. Spostamento in avanti/indietro di (Y_obiettivo - Y_stimato)* 10 millimetri del robot
6. Attesa di qualche secondo
7. Rotazione di 90 gradi del robot
8. Attesa di qualche secondo
9. Spostamento in avanti/indietro di (X_obiettivo - X_stimato)* 10 millimetri del robot
10. Attesa di qualche secondo
11. Rotazione di 90 gradi
12. Attesa di qualche secondo

Le funzioni *ritorna_coordinate()* e *denormalizza()* sono simili i tutto e per tutto alle analoghe funzioni trovate in *stimaxy()*.

Generamap_modificato

Una volta creato il file *all.map*, è possibile generare automaticamente le mappe relative alle immagini di learning. Ciò è reso possibile dallo script *generamap_modificato* invocato con il comando:

generamap_modificato $nomegriglia.env learn.map.

Vengono ora creati i file *nomegriglia.env/maps/learn.def*
nomegriglia.env/maps/learn.map

Il file *learn.map* ha la stessa struttura del file *all.map* e contiene nelle sue righe i file di learning. Qui sotto è riportato un esempio.

```
im1.r_g_b 000 000
im10.r_g_b 040 080
im11.r_g_b 040 120
im12.r_g_b 040 160
im13.r_g_b 040 200
im14.r_g_b 040 240
im15.r_g_b 080 000
im16.r_g_b 080 040
```

```
im17.r_g_b 080 080
im17.r_g_b 080 080
im18.r_g_b 080 120
im19.r_g_b 080 160
im2.r_g_b 000 040
im20.r_g_b 080 200
im21.r_g_b 080 240
im22.r_g_b 120 000
…
```

Il file *learn.def* contiene in riga l'elenco dei file di learning. Qui sotto ne è riportato un esempio.

```
/home/polli/bin/prov_finale.env/maps/learn.map
```

Vediamo più in dettaglio lo script. Il main si apre con un controllo sui parametri in ingresso che devono essere due. Il primo, la directory di riferimento *nomegriglia.env*, deve essere preesistente, così come la directory */maps* ed il file *all.map*.
I file di learning possono essere passati allo script o tramite inserimento da utente (per mettere a punto lo script o per debuggare in fase di testing il sistema), oppure possono essere estratti direttamente dal file *all.map* tramite la funzione *asknames* che restituisce nella variabile names i nomi dei suddetti files. Questa variabile viene poi passata alla funzione built, la quale genera i due files *learn.map* e *learn.def*.
Ritorna inoltre a video il numero di file di learning che verranno quindi considerati successivamente.

Testall

La prima parte dello script richiama un piccolo programmino in linguaggio C che introduce un ritardo di 6 secondi.

Attendi 6

Successivamente si chiama il programma *qcread* di acquisizione delle immagini con gli opportuni parametri di regolazione delle immagini catturate dalla quickcam. Per maggiori informazioni sui parametri si rimanda alla documentazione di qcread. La messa a punto della quickcam avviene tramite lo script *messa_a_punto_cam*.

Il risultato del comando *qcread* viene inviato al programma C di nome *quick*, il quale effettua la trasformazione cartesiano-polare dell'immagine acquisita.

Regenera1test_modificato

Tramite lo script *regenera1test_modificato* richiamato dalla *train_net* con la riga di comando

regenera1test_modificato $nomegriglia.env test0

vengono richiesti all'utente i parametri del test, il nome della rete da utilizzare e le mappe di learning da considerare. Sono prodotti i file

nomegriglia.env/test/test0/comment
nomegriglia.env/test/test0/param.def
nomegriglia.env/test/test0/net.def
nomegriglia.env/test/test0/batch.cfb

Comment contiene i nomi delle mappe di learning e di test ed il commento eventualmente inserito dall'utente. Un esempio del file comment è riportato qui sotto.

 # Test created Fri Oct 23 18:25:24 CEST 1998
 # learn map: /home/polli/bin/prov_finale.env/maps/learn.map
 # network : /home/polli/bin/nets/std3bp.net

Param.def contiene i parametri della rete neurale che sono:
Max batch iteration
Iteration step for result
Max random initialization
Tolerance for convergence
che vengono inseriti dall'utente se diversi da quelli di default.
Un esempio di questo file è

500 500 0.1 0.01

Learn.def contiene il nome della mappa di learning inserita dall'utente; per default è learn.map.

Net.def contiene il nome della rete neurale da utilizzare per *snnsbat*. Nel nostro caso la rete è std3bp.net.

Batch.cfb contiene la configurazione di default della rete e ne riassume i parametri assieme a quelli inseriti dall'utente. Sarà il file di ingresso alla rete *snnsbat*.

Vediamo ora in dettaglio lo sviluppo dello script.
Il main si apre con il solito controllo sui parametri in ingresso; devono inoltre esistere le sottodirectory richieste.
Si esegue ora la *build*, la quale a sua volta richiama la *askparams*. Askparams chiede all'utente l'inserimento dei parametri della rete. Se non viene modificato niente vengono utilizzati i parametri di default. Subito dopo viene chiamata la funzione *askcomment* che crea il file comment con gli eventuali commenti dell'utente. Dopo la creazione o la modifica dei file *learn.def*, *param.def*, *test.def*, avviene il passo fondamentale dello script: la creazione del file *batch.cfb* tramite la funzione *makebatch*.
Un esempio di file *batch.cfb* è mostrato qui sotto.

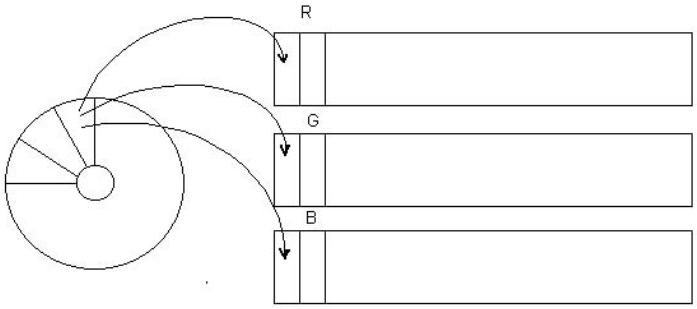
```
#
Type: SNNSBATCH_2
#
# The following keyword-value combinations may be supplied in any order.
# If a key is given twice, the second appearence is taken.
# Keys that are not required for a special run may be omitted.
# If a key is ommited but required, a default value is assumed.
# The lines may be separated with comments.
#
# Please note the mandatory file type specification at the begining and
# the colon following the key.
#                                              Defaults:
#
#     NoOfLearnParam:      <int>                  0
#     LearnParam:          [ <float> ]*           0.0
#     NoOfInitParam:       <int>                  0
#     InitParam:           [ <float> ]*           0.0
```

```
#     NetworkFile:         <string>             ""
#     TrainedNetworkFile:  <string>             ""
#     LearnPatternFile:    <string>             ""
#     TestPatternFile:     <string>             ""
#     ResultFile:          <string>             ""
#     InitFunction:        <string>             ""
#     MaxLearnCycles:      <int>                0
#     MaxErrorToStop:      <float>              0.0
#     Shuffle:             [ YES | NO ]         NO
#     ResultMinMaxPattern: <int><int>           0 0
#     ResultIncludeInput:  [ YES | NO ]         NO
#     ResultIncludeOutput: [ YES | NO ]         YES
#     PerformActions:
#
##############################################################################
#
#### default configuration
#
NetworkFile: /home/polli/bin/nets/std3bp.net
#
InitFunction: Randomize_Weights
NoOfInitParam: 2
InitParam: -0.1 0.1
#
LearnPatternFile: /home/polli/bin/prov_finale.env/test/test0/learn.pat
NoOfLearnParam: 1
LearnParam: 0.2
MaxLearnCycles: 1000
MaxErrorToStop: 0.01
Shuffle: YES
TestPatternFile: /home/polli/bin/prov_finale.env/test/test0/learn.pat
ResultFile: 00001000.res
ResultMinMaxPattern: 1 35
ResultIncludeInput: NO
ResultIncludeOutput: YES
TrainedNetworkFile:  /home/polli/bin/nets/network.net
#
#### end of default configuration
#
#### start of cmds ####
#
PerformActions:
NetworkFile: <OLD>
LearnPatternFile: /home/polli/bin/prov_finale.env/test/test0/learn.pat
NoOfLearnParam: <OLD>
LearnParam: <OLD>
MaxLearnCycles: <OLD>
MaxErrorToStop: <OLD>
Shuffle: <OLD>
TestPatternFile: <OLD>
TrainedNetworkFile: <OLD>
ResultFile: /home/polli/bin/last.res
ResultMinMaxPattern: <OLD>
ResultIncludeInput: <OLD>
ResultIncludeOutput: <OLD>
#
```

E’ inoltre possibile visualizzare il file a video.

Quick

Questo programma scritto in linguaggio "C" ha il compito di estrarre dal frame acquisito dalla telecamera 360 componenti angolari per canale cromatico, ed avviene essenzialmente attraverso una trasformazione cartesiano – polare.
Si ricava quindi dall'immagine una corona circolare mascherando la parte centrale e laterale, lasciando quindi solo la parte significativa del frame.
La fase successiva consiste in una mappatura della corona circolare in una matrice per ciascuna componente cromatica.



Per ogni colore sono ora ottenuti 360 elementi, compresi tra 0 e 255, rappresentativi dei 360 settori circolari in cui è stato suddiviso il frame iniziale.

Descrizione dell'hardware

La parte hardware del progetto consiste in:

- un robot Pioneer1 (della ActiveMedia Incorporated)
- una quickcam della Connetix
- un computer portatile
- una serie di coni a diversa levigatura
- un sostegno regolabile per fissare i coni

Il sistema finale prevede il robot per la navigazione fisica; il portatile, per l'elaborazione dei dati e l'interazione con l'utente, viene appoggiato sul dorso del robot; quickcam, sostegno e cono vengono raggruppati in un'unica struttura fissata a sua volta sul robot e collegata al portatile tramite seriale.
Per una comprensione intuitiva del collegamento dei vari componenti tra loro, si rimanda alle appendici.

Il robot Pioneer1 scelto è stato Tobor. La pinza montata sul davanti fornisce un riferimento (anche se approssimativo) per orientare correttamente il robot e quindi agevolare il lavoro della rete neurale.

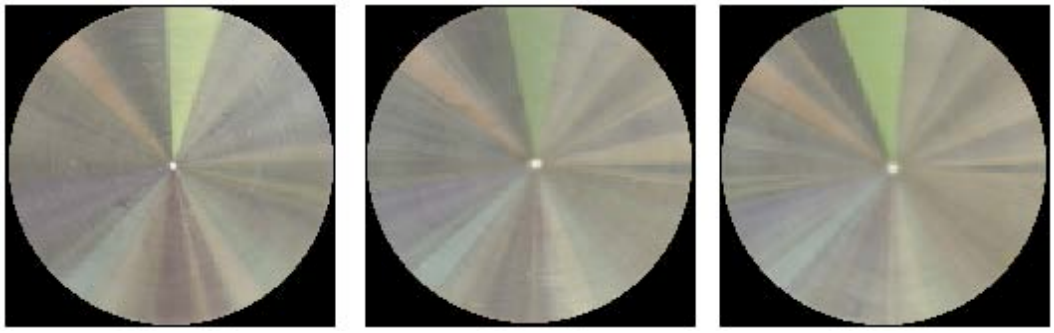
Si è preferito utilizzare una quickcam invece della telecamera in dotazione al robot per i seguenti motivi :

1. la quickcam è orientabile con più facilità
2. ci serviva una telecamera che potesse essere posta a diverse altezze in modo da regolare la dimensione dell'immagine catturata
3. la telecamera in dotazione a tobor era in una posizione troppo aderente al dorso del robot; l'immagine presa in queste condizioni consisteva quasi interamente dallo schermo del portatile (appoggiato anch'esso su tobor) e risultava quasi totalmente nera.

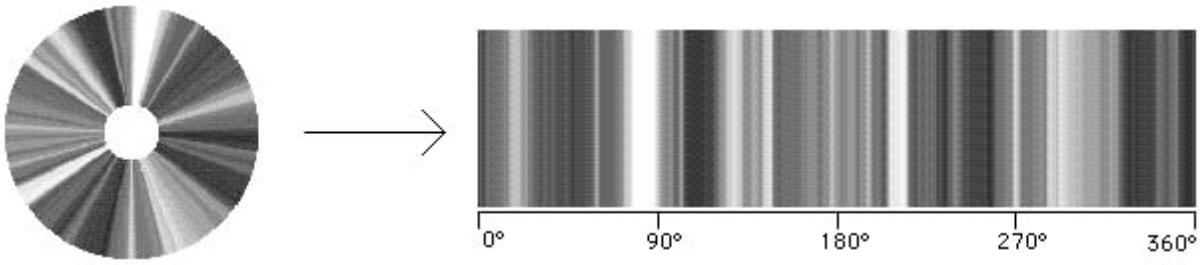
L'utilizzo del portatile era obbligatorio data la mole di software richiesta e la necessità di interagire con l'utente in fase di addestramento della rete neurale. Utilizzare un normale desktop collegato tramite cavo seriale avrebbe reso tutto il sistema meno autonomo; oltre a ciò il cavo seriale avrebbe potuto creare intralcio al robot. L'utilizzo dei radio-modem sarebbe stato possibile se solo non si fosse creato il problema di come alimentare la quickcam. La quickcam richiede infatti una alimentazione che può giungergli da una porta seriale, oppure bypassando la tastiera di un PC.
Il portatile Frost ha una porta libera che è stata utilizzata per alimentare la quickcam.

I coni utilizzati hanno la seguente struttura:

La parte centrale del cono non ha rilevanza pratica quando si tratta di riconoscere i colori e forme. Per questo motivo viene mascherata (ignorata) dall'algoritmo che trasforma l'immagine dalle coordinate polari alle coordinate rettangolari che ci servono. Oltre alla parte centrale vengono ignorati anche i quattro angoli dell'immagine. Per evitare che la quickcam riceva troppa luce (infatti è puntata verso l'alto e quindi verso fonti luminose), si è montata una sagoma di cartone nero sopra il cono in modo che la luce diretta non la 'accechi'.
Sotto sono riportati degli esempi di immagini prese con la quickcam:



Queste immagini vengono trasformate in modo che ad ogni settore angolare corrisponda una striscia nell'immagine risultato. La trasformazione che porta da un sistema di coordinate all'altro è fatta tramite formule trigonometriche e, il suo risultato pratico, è osservabile nella seguente figura:



L'immagine a destra riportata qui sopra viene divisa in 360 parti corrispondenti ognuna ad un grado. Di ogni 'striscia' vengono considerate le tre componenti cromatiche R,G,B e viene fatta la media lungo tutti i pixel della stessa striscia. Il risultato finale è un vettore di 1080 elementi, 360 elementi per ognuna delle componenti cromatiche.

I coni vengono montati su un supporto che fornisca un minimo di flessibilità in fase di montaggio. Questo supporto è costituito da una squadretta di acciaio ad angolo retto fissata al robot tramite viti. La squadretta monta un'asta regolabile in altezza. Sulla sommità dell'asta è fissato un aggancio per montare i coni. Sempre sulla squadretta è montato anche un sostegno per la quickcam. Uno schema completo è illustrato nelle appendici.

Descrizione degli esperimenti e risultati

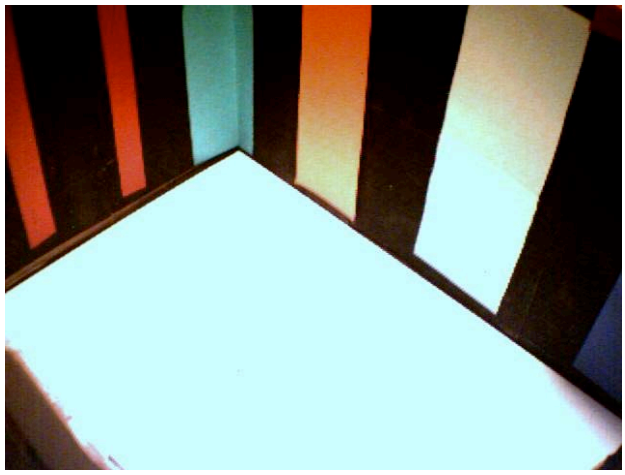
Sostanzialmente gli esperimenti si sono effettuati in due ambienti distinti :

- Ambiente chiuso e fortemente condizionato
- Ambiente aperto e debolmente condizionato

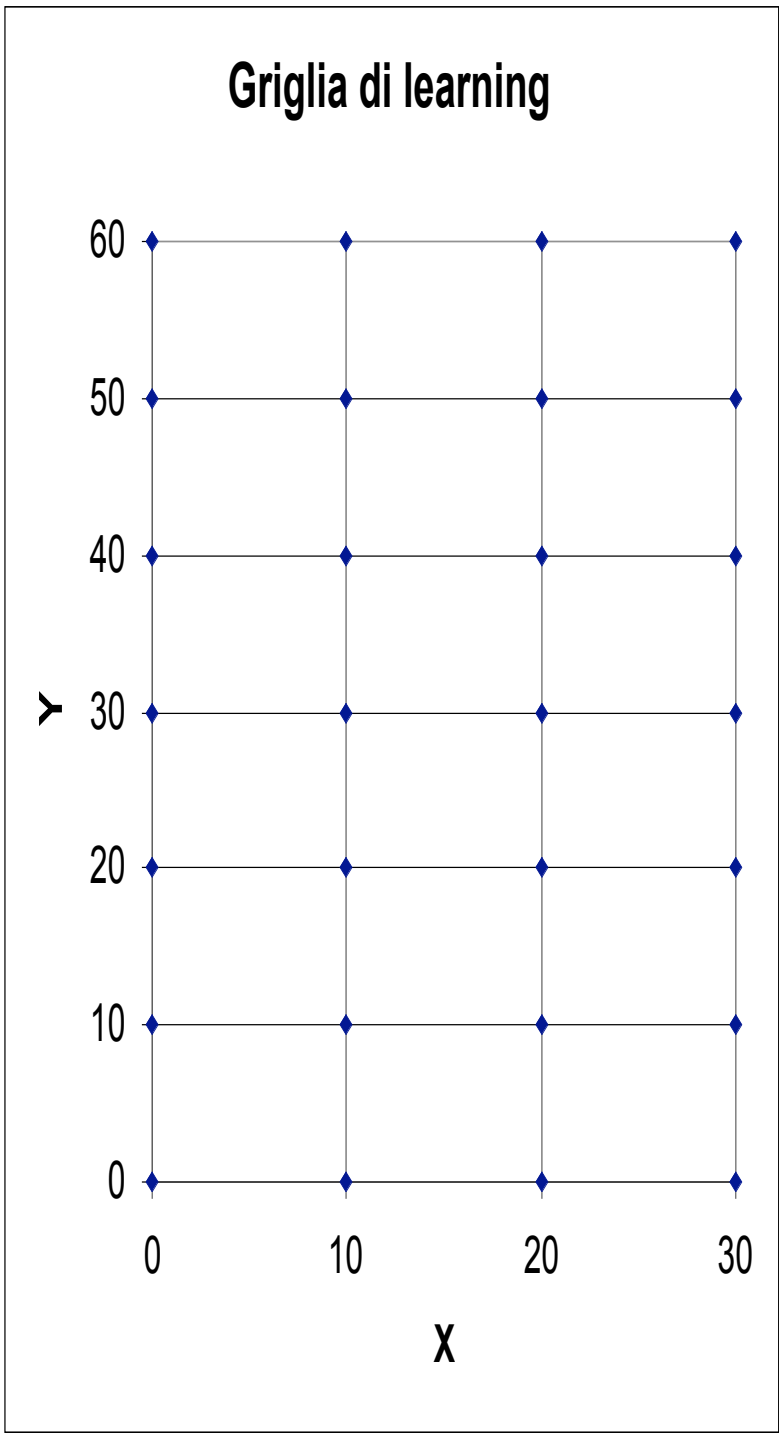
In entrambe gli esperimenti, si sono prese misure sia su una griglia a maglie fitte che su griglia a maglie larghe (meno punti di misura).
L'ambiente chiuso e fortemente condizionato consiste di un parallelepipedo sul cui fondo è stata disegnata una griglia, e sulle cui pareti laterali sono state applicate bande colorate per aiutare l'addestramento della rete.
L'ambiente aperto consiste in una stanza sul cui pavimento è stata disegnata una griglia di punti per addestrare il robot e per avere punti di riferimento.

Le prime prove effettuate hanno come obiettivo principale la verifica della qualità delle immagini catturate dalla quickcam e del corretto funzionamento della rete neurale.
Per mettere a punto la telecamera si è infatti utilizzato lo script *messa_a_punto_cam*, tramite il quale è possibile centrare l'immagine del cono e modificare la qualità dell'immagine catturata impostandone ad esempio la luminosità, il contrasto, ecc, come indicato dal file di documentazione allegato al programma *qcread*. Regolando man mano la lente, e riproducendo ripetutamente le immagini a video, si procede alla messa a fuoco dell'immagine ed alla regolazione del sostegno del cono fino ad una sua visualizzazione limpida e chiara.
Al fine della verifica del buon funzionamento dell'intero sistema, le prime prove sono state eseguite in un ambiente condizionato appositamente costruito utilizzando una scatola di cartone con delle strisce colorate ai lati.

Figure della scatola



Questo ambiente, per le particolari scelte dei colori e della luce presente al suo interno, dovrebbe costituire infatti la condizione ottimale di operatività del sistema Pollicino.
La prima prova consiste nell'addestrare la rete neurale con 28 punti disposti come nella figura sottostante.

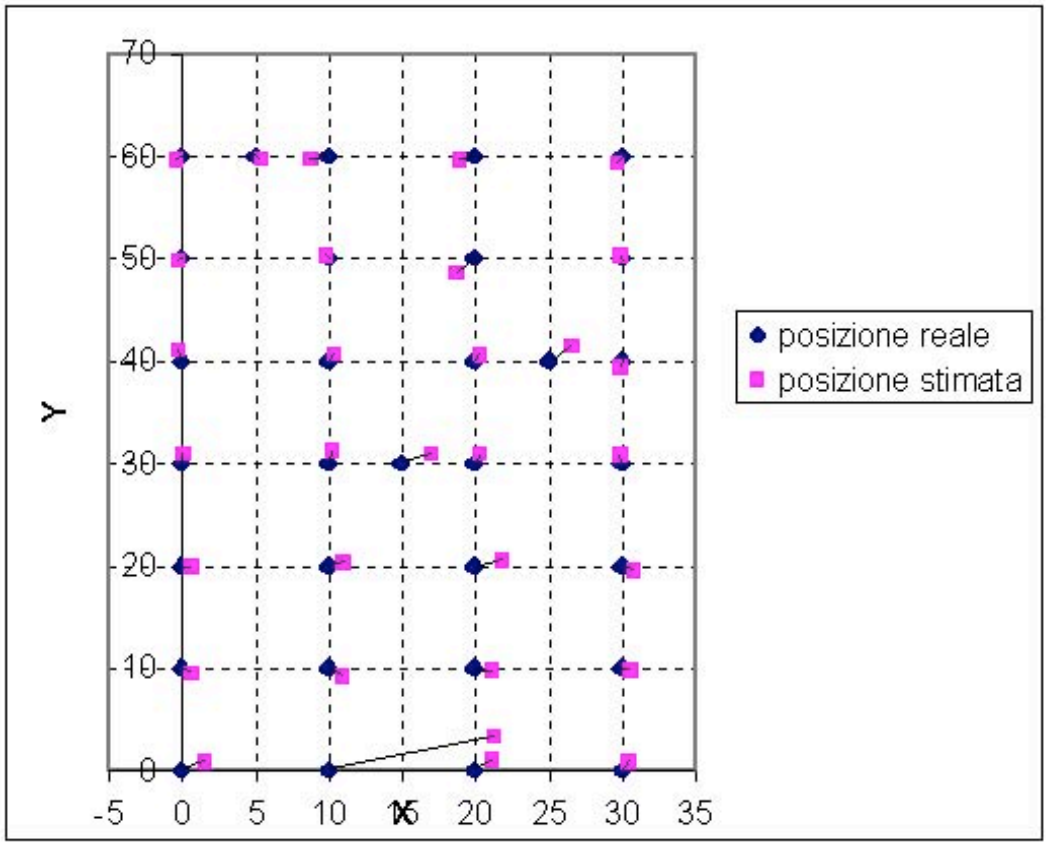


Successivamente, terminata la fase di acquisizione delle immagini e della fase di training della rete attraverso il programma *train_net*, al fine di verificarne un buon comportamento, si procede con l'esecuzione delle stime. *stimaxy* è la funzione che permette infatti alla rete di stimare la posizione in cui si trova in quel momento la quickcam. Si procede così alla stima delle posizioni nei punti della griglia di training e di altri tre punti interni.
I risultati ottenuti sono nella tabella sottostante in cui è riportato anche l'errore di stima. Sotto è anche riportato il grafico delle posizioni stimate in funzione della posizione reale della quickcam.
Tutte le grandezze si intendono in centimetri.

Tabella prova alta densità

X_stimata	Y_stimata	X_reale	Y_reale	Errore di stima	Media errore di stima in percentuale
1,66	0,63	0	0	1,78	14,134
21,37	3,22	10	0	11,82	
21,35	0,92	20	0	1,63	
30,54	0,7	30	0	0,88	
0,76	9,37	0	10	0,99	
11,05	9,08	10	10	1,40	
21,32	9,66	20	10	1,36	
30,64	9,69	30	10	0,71	
0,76	19,75	0	20	0,80	
11,08	20,23	10	20	1,10	
21,95	20,33	20	20	1,98	
30,94	19,45	30	20	1,09	
0,15	30,81	0	30	0,82	
10,39	31,06	10	30	1,13	
20,43	30,82	20	30	0,93	
30,06	30,69	30	30	0,69	
-0,19	40,98	0	40	1,00	
10,42	40,6	10	40	0,73	
20,45	40,38	20	40	0,59	
30,02	39,17	30	40	0,83	
-0,16	49,64	0	50	0,39	
9,88	50,25	10	50	0,28	
18,82	48,5	20	50	1,91	
29,99	50,17	30	50	0,17	
-0,29	59,37	0	60	0,69	
8,93	59,55	10	60	1,16	
19,12	59,39	20	60	1,07	
29,86	59,16	30	60	0,85	
17,05	30,71	15	30	2,17	
26,71	41,28	25	40	2,14	
5,53	59,51	5	60	0,72	

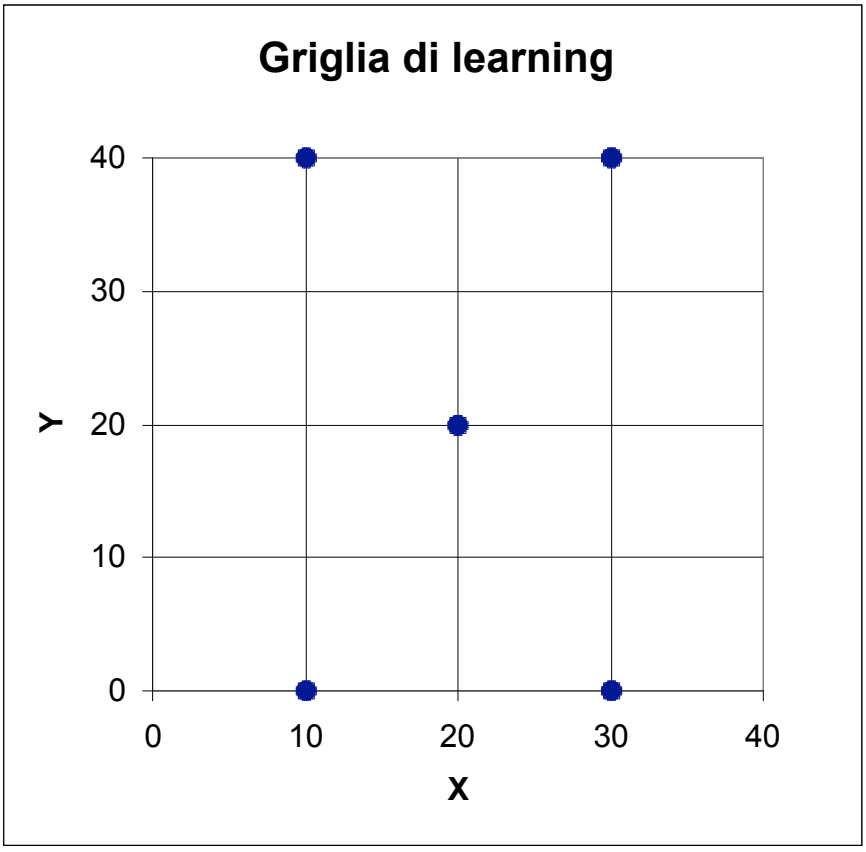
Grafico risultati prova alta densità



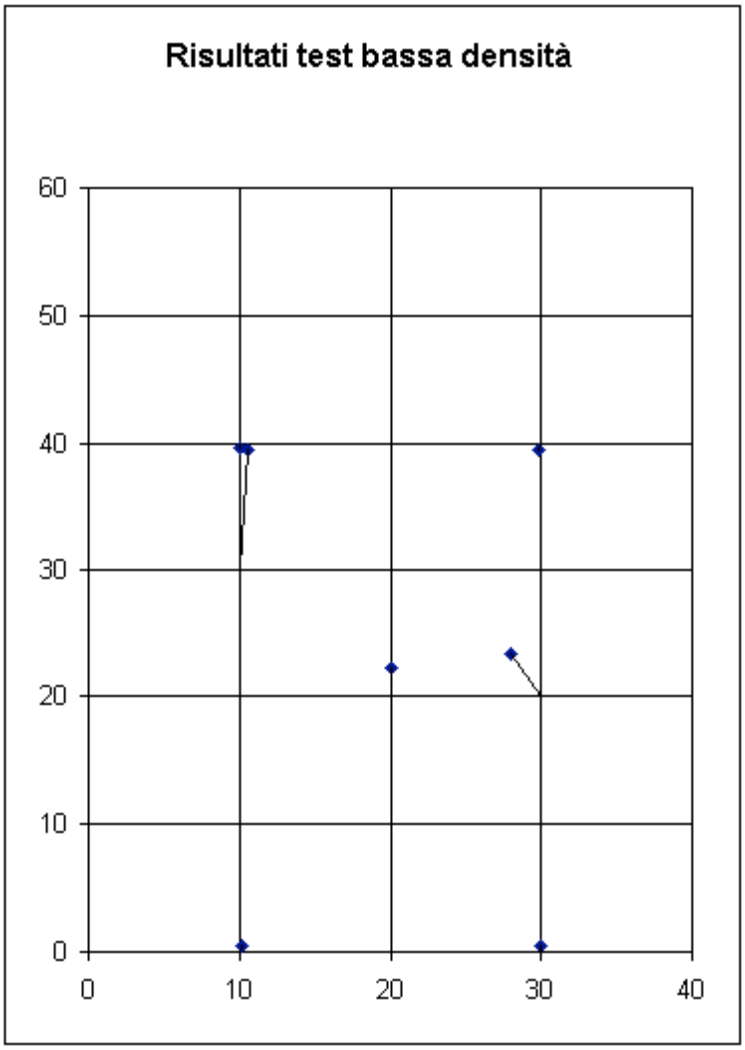
Dagli errori trovati, si nota come la stima sia precisa, con un errore percentuale pari al 14 % circa rispetto alla distanza di 10 cm tra i campioni di learn.

La seconda prova consiste nell'eseguire una stima utilizzando un insieme minore di immagini di learn al fine di provare la robustezza del sistema verificandone poi l'errore di stima percentuale.
La griglia di riferimento per le 5 immagini di learn è mostrata qui sotto assieme alla tabella dei risultati ottenuti per la stima in 7 punti.

Grafico griglia di learn bassa densità



X_stimata	Y_stimata	X_reale	Y_reale	Errore di stima	Errore medio percentuale
10,23	0,5	10	0	0,55	11,13
30,03	0,49	30	0	0,49	
20,08	22,17	20	20	2,17	
10,02	39,67	10	40	0,33	
29,85	39,48	30	40	0,54	
27,98	23,32	30	20	3,89	
10,62	39,44	10	30	9,46	



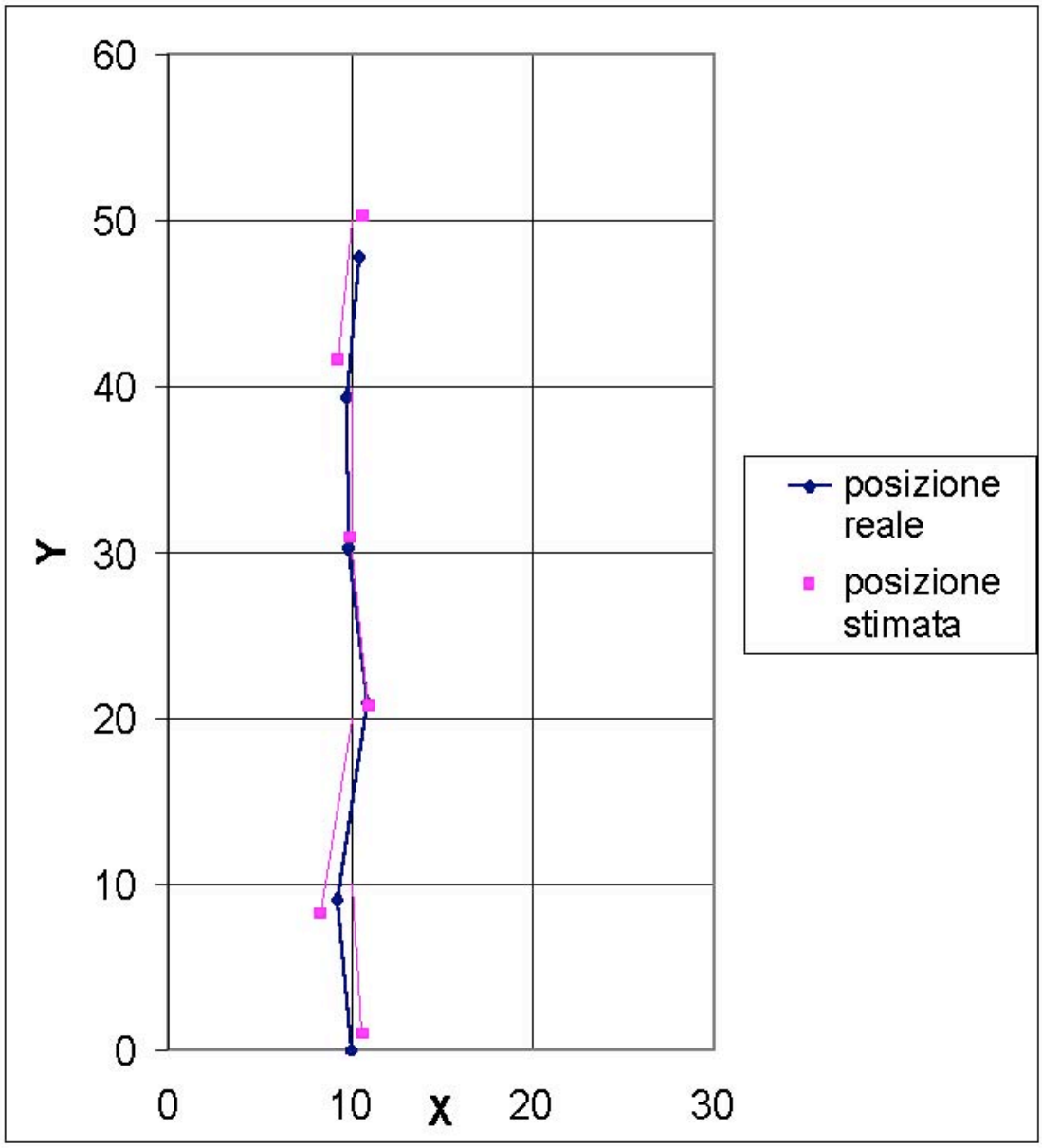
L'errore percentuale è circa 11 %; valore prossimo a quello calcolato per il test ad alta densità.
Sia per il primo che per il secondo test, osservando i grafici delle posizioni stimate e le tabelle, si nota come le stime siano a volte superiori ed a volte inferiori ai valori reali. Questa osservazione è molto importante perché sta alla base del principio di compensazione, secondo il quale gli errori cumulativi si compensano, dando origine ad una navigazione pressoché esatta come verrà verificato nella prova successiva.

Quest'ultima consiste in una prova di navigazione. La quickcam viene posizionata in un punto iniziale e, dopo averne stimata la posizione, la si pone nella posizione voluta con uno spostamento dato dalla differenza tra la posizione stimata e la posizione di destinazione per quel passo. Il processo viene ripetuto passo dopo passo fino al raggiungimento della posizione finale. Ad ogni passo vengono misurate la posizione stimata, la posizione reale, e la posizione finale teorica e reale. Viene riportata la tabella con i risultati relativi alla navigazione che è stata realizzata utilizzando alcuni punti della griglia di learning come punti di passo.

Tabella risultati navigazione

		X_stimato	Y_stimato	X_teorico	Y_teorico	X_reale	Y_reale
passo1		10,72	0,93	10	0	10	0
passo2		8,35	8,18	10	10	9,28	9,07
passo3		11,05	20,7	10	20	10,93	20,89
passo4		10,05	30,86	10	30	9,88	30,19
passo5		9,31	41,57	10	40	9,83	39,33
passo6		10,68	50,27	10	50	10,52	47,76

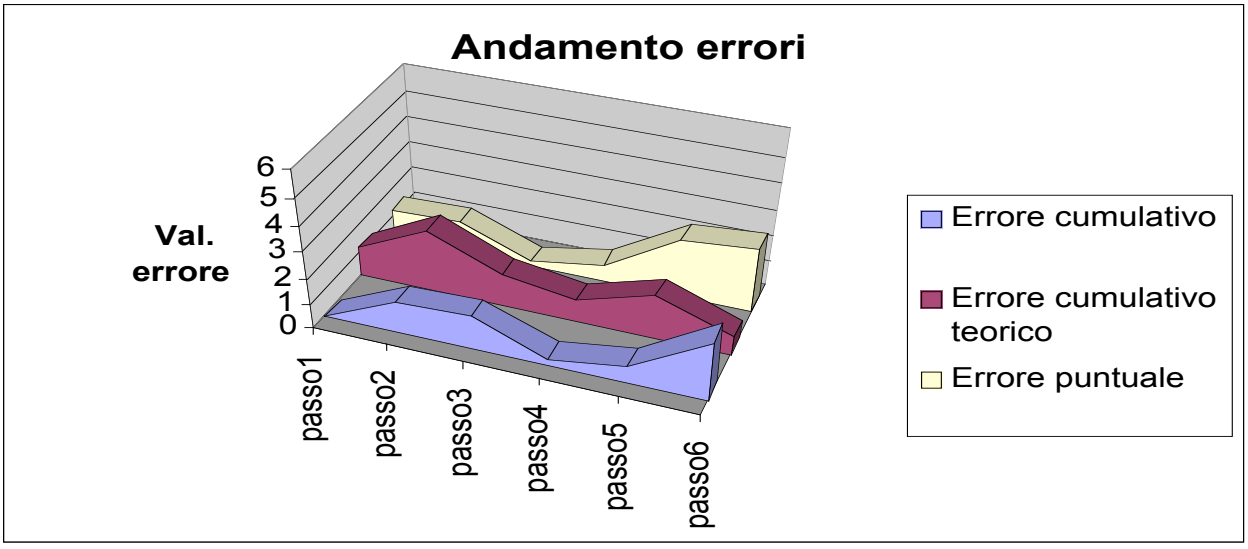
Qui sotto è riportata la figura delle traiettorie teoriche e reali della navigazione.



Con questi risultati è possibile stimare tre tipi di errori: l'errore cumulativo, l'errore cumulativo teorico e l'errore puntuale.
L'errore puntuale (o locale) è dato dalla differenza tra il punto stimato e quello reale; l'errore cumulativo è dato dalla distanza tra il punto di riferimento della griglia e la posizione reale; infine l'errore cumulativo teorico dato dalla distanza tra il punto della griglia di riferimento e la posizione stimata.

	Errore cumulativo	Errore cumulativo teorico	Errore puntuale
passo1		1,18	1,18
passo2	1,18	2,46	1,29
passo3	1,29	1,26	0,22
passo4	0,22	0,86	0,69
passo5	0,69	1,71	2,30
passo6	2,30	0,73	2,52

L'andamento degli errori è riportato qui sotto.

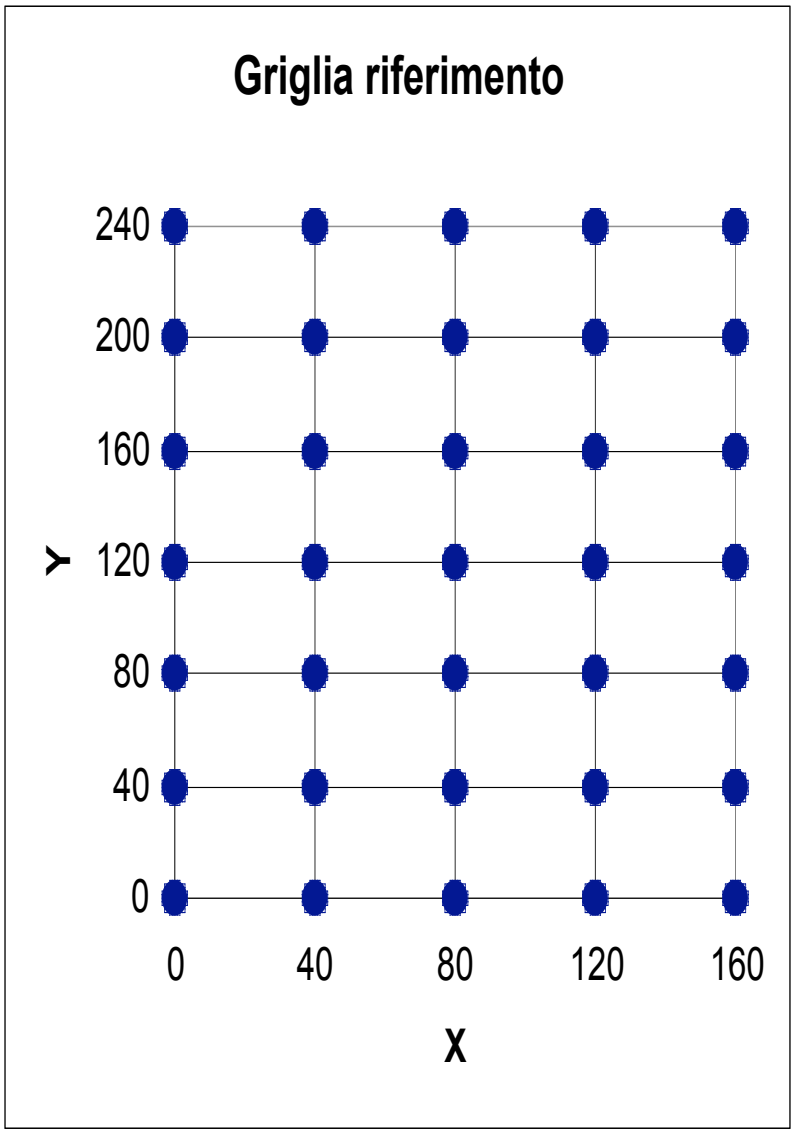


Dall'analisi dei risultati si può concludere che il software del sistema Pollicino, ed in particolare la rete neurale, sono funzionali in condizioni ottimali.

Si procede ora alla fase operativa di prova del sistema complessivo su robot. Per effettuare la serie di prove successive si è utilizzato lo studio 35, rappresentato nella figura sottostante.

Figura studio 35

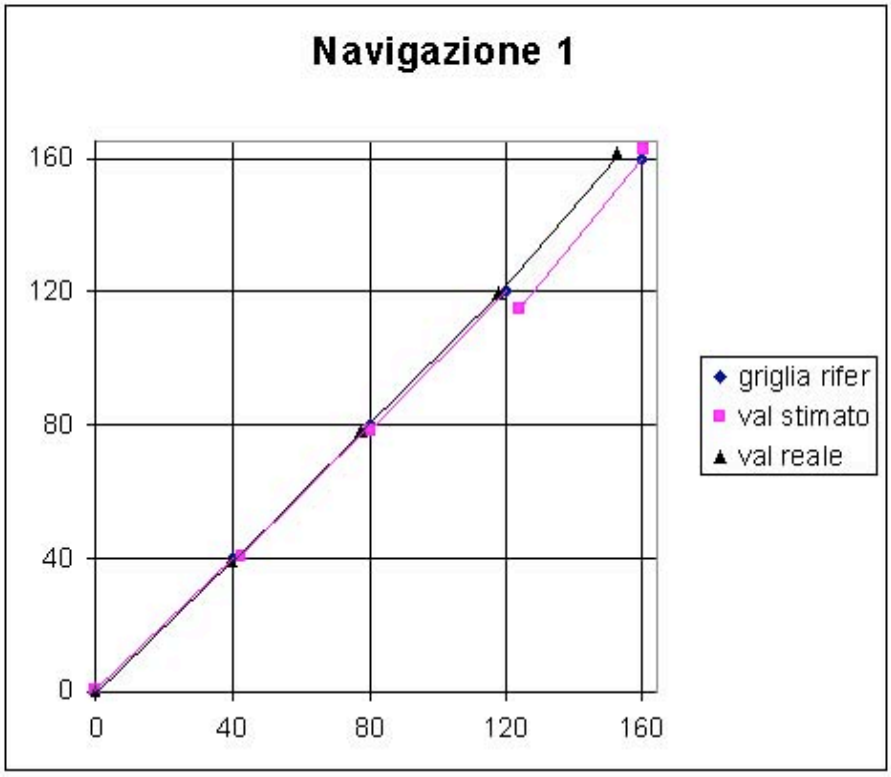
La griglia utilizzata come riferimento è stata tracciata sul pavimento ed è composta da 6x4 punti disposti secondo la figura riportata sotto. Tutte le misure si intendono in cm.



La fase di training della rete è stata effettuata con la ormai nota funzione *train_net*.
Le prove che seguono sono quattro navigazioni effettuate con la quickcam a bordo del robot. Le funzioni per il movimento del robot, della stima della posizione e della gestione delle traiettorie sono effettuate dal programma *kaba* che ci permette di inserire man mano le coordinate che Pollicino dovrà raggiungere ad ogni passo. Dopo l'inserimento delle coordinate, Pollicino stimerà la sua posizione attuale e si sposterà raggiungendo l'obiettivo in base alla sua posizione stimata. Ad obiettivo raggiunto si fermerà comunicando la posizione stimata di partenza lasciando il tempo all'operatore di misurare la sua posizione reale.
All'interno della funzione *kaba* è previsto un ritardo per permettere all'operatore di allontanarsi dopo l'inserimento delle coordinate, onde evitare di perturbare troppo l'ambiente memorizzato.

A passo terminato, è prevista la risistemazione del robot in senso rotatorio per rimetterlo in direzione ortogonale alla griglia di riferimento. Per far ciò si utilizza una penna laser opportunamente sistemata sulle pinze del robot. Si gira infatti il robot fino a renderlo parallelo alla direzione della griglia aiutandosi con il raggio laser, emesso dalla penna, che incide contro il muro.

Sotto è mostrata la figura rappresentante la navigazione 1.

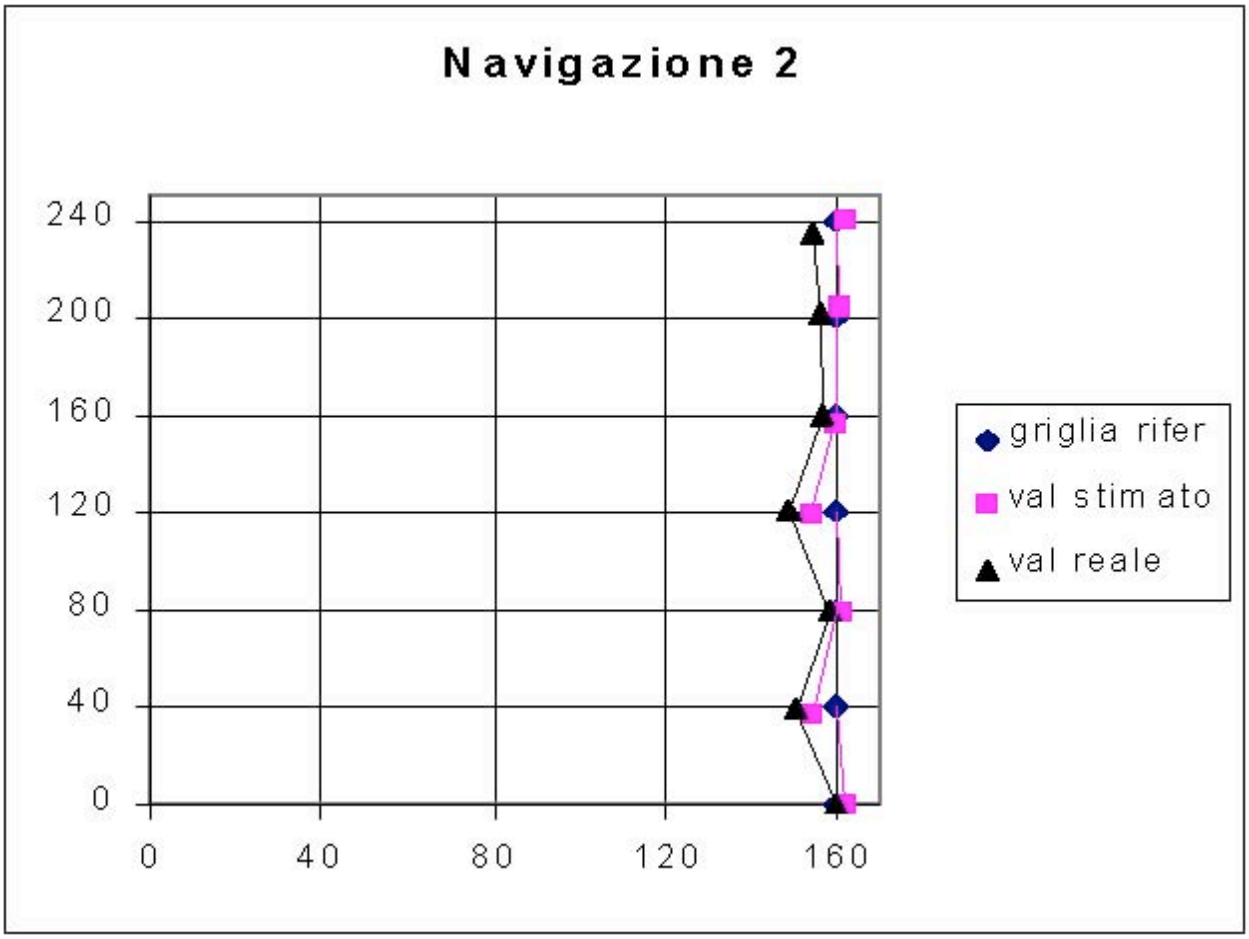


I risultati della prima navigazione sono riportati nella tabella sotto, assieme alla tabella degli errori.

	X_griglia	Y_griglia	X_stimato	Y_stimato	X_real	Y_real
passo1	0	0	-0,02	0,36	0	0
passo2	40	40	42,38	40,19	39,1	39,7
passo3	80	80	80,18	78,39	77,1	78,4
passo4	120	120	124,08	114,99	117,7	119,8
passo5	160	160	160,28	162,75	152,3	161,8

	Errore cumulativo	Errore cumulativo teorico	Errore puntuale	Lunghezza passo
passo1	0	0,36	0,36	55,72
passo2	0,95	2,39	3,32	54,24
passo3	3,31	1,62	3,08	57,98
passo4	2,31	6,46	7,99	54,42
passo5	7,91	2,76	8,04	

La seconda navigazione è stata effettuata lungo il lato destro della griglia come è mostrato in figura. Dalla figura è anche possibile distinguere la traiettoria reale e quella teorica del robot.



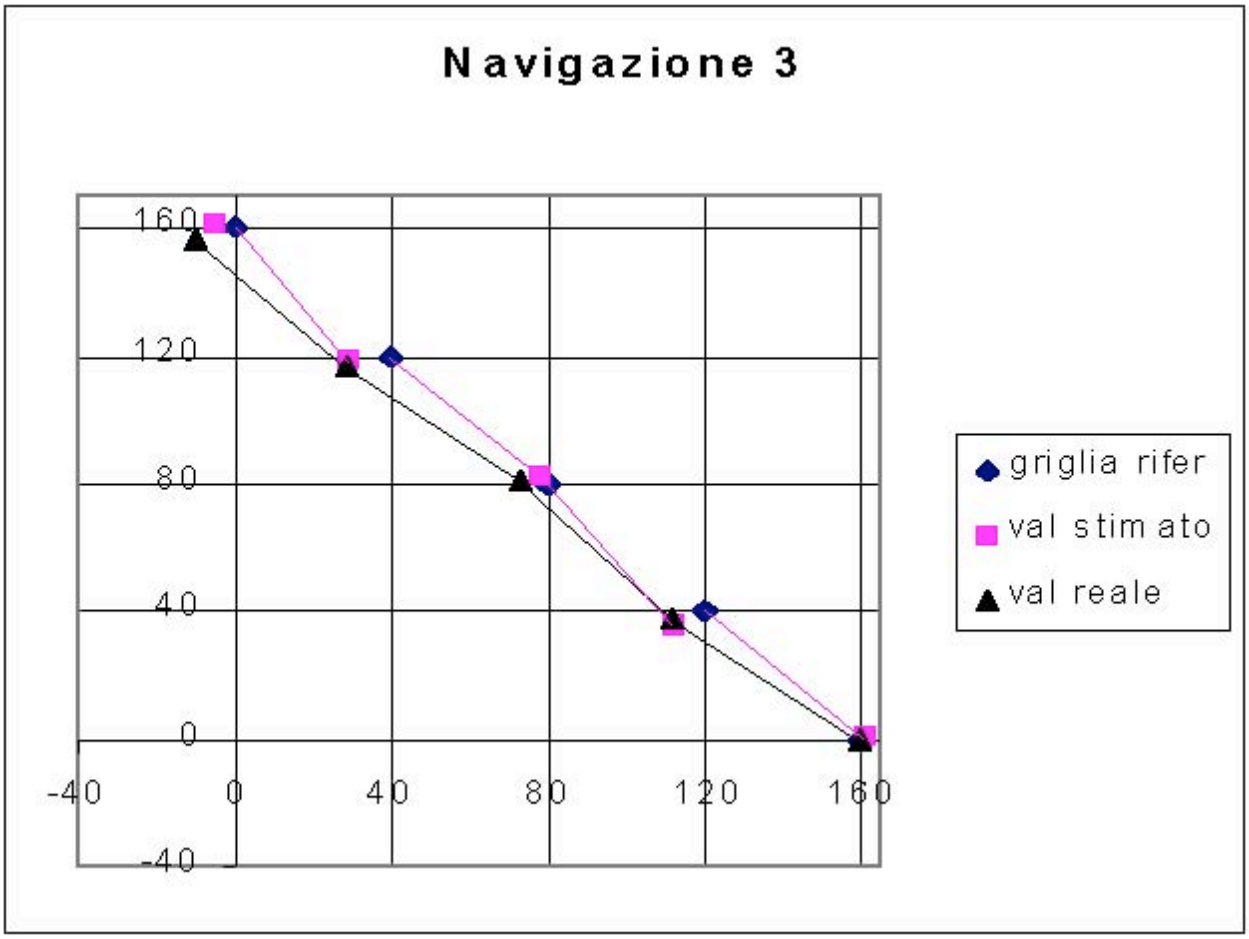
Qui sotto sono riportate le tabelle con le posizioni e gli errori di ogni passo.

Tabelle seconda navigazione

	X_griglia	Y_griglia	X_stimato	Y_stimato	X_real	Y_real
passo1	160	0	162,22	0,33	160	0
passo2	160	40	154,14	36,3	150,6	39,4
passo3	160	80	161,65	78,37	158	80,1
passo4	160	120	154,64	119,63	148,6	120
passo5	160	160	160,16	156,34	157	159,3
passo6	160	200	160,98	204,72	155,9	201,4
passo7	160	240	162,59	239,59	154,2	234,6

	Errore cumulativo	Errore cumulativo teorico	Errore puntuale	Lunghezza passo
passo1	0	2,24	2,24	40,50
passo2	9,41	6,93	4,70	41,37
passo3	2,00	2,32	4,04	40,99
passo4	11,4	5,37	6,05	40,19
passo5	3,08	3,66	4,33	42,11
passo6	4,33	4,82	6,07	33,24
passo7	7,92	2,62	9,76	

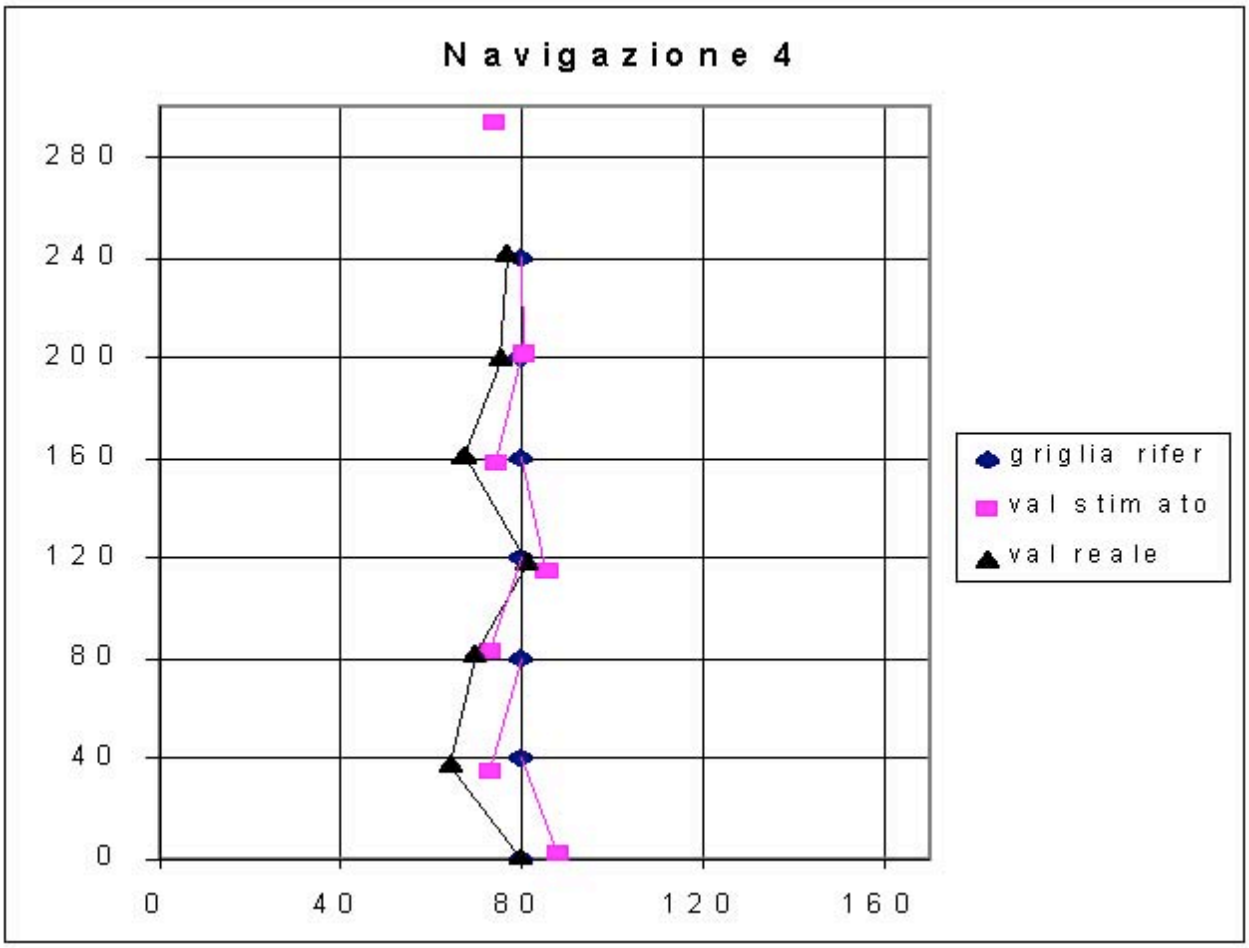
Segue ora la terza navigazione. Sotto sono mostrati la traiettoria e le tabelle rispettivamente delle posizioni e degli errori.



	X_griglia	Y_griglia	X_stimato	Y_stimato	X_real	Y_real
passo1	160	0	161,43	0,32	160	0
passo2	120	40	112,7	35,12	111,6	37,7
passo3	80	80	77,96	81,78	73,5	81,2
passo4	40	120	29,73	118,52	29	117
passo5	0	160	-4,87	161,17	-10	156

	Errore cumulativo	Errore cumulativo teorico	Errore puntuale	Lunghezza passo
passo1	0	1,47	1,47	61,35
passo2	8,71	8,78	2,80	57,83
passo3	6,61	2,71	4,50	57,11
passo4	11,40	10,38	1,69	55,15
passo5	10,77	5,01	7,28	

Infine, la quarta navigazione.



	X_griglia	Y_griglia	X_stimato	Y_stimato	X_real	Y_real
passo1	80	0	87,61	2,24	80	0
passo2	80	40	72,91	34,34	64	37,9
passo3	80	80	73,06	82,97	69,7	81,4
passo4	80	120	85,58	114,57	81,4	118,9
passo5	80	160	74,46	158,14	67,5	161
passo6	80	200	80,81	200,82	75,2	200,7
passo7	80	240	74,28	293,78	76,5	241,4

	Errore cumulativo	Errore cumulativo teorico	Errore puntuale	Lunghezza passo
passo1	0	7,93	7,93	41,14
passo2	16,14	9,07	9,59	43,87
passo3	10,39	7,55	3,71	39,28
passo4	1,78	7,79	6,02	44,34
passo5	12,54	5,84	7,52	40,44
passo6	4,85	1,15	5,61	40,72

passo7	3,77	54,08	52,43	

Per ogni tipo di errore, viene riportata una tabella riassuntiva indicante la media e la deviazione standard per ogni passo. Di seguito è presentato anche il rispettivo grafico. Tutte le grandezze sono espresse in cm.

ERRORE CUMULATIVO

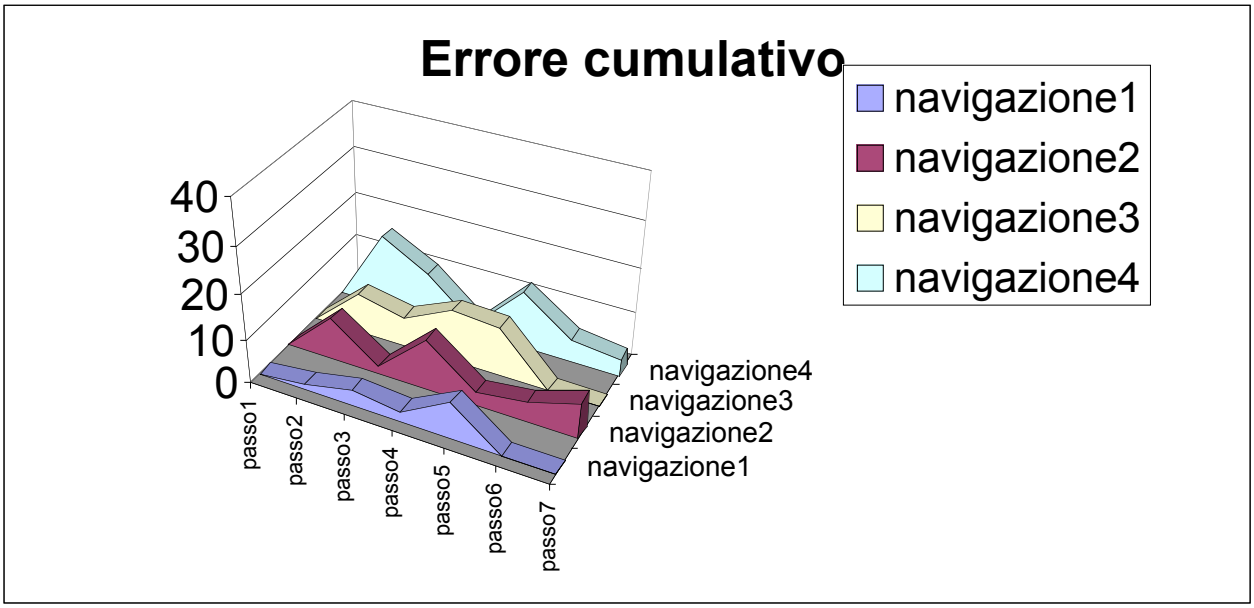
	passo1	passo2	passo3	passo4	passo5	passo6	passo7	dev standard	media
navigazione1	0	0,95	3,31	2,31	7,91			3,07	2,90
navigazione2	0	9,42	2,00	11,40	3,08	4,33	7,92	4,20	5,45
navigazione3	0	8,71	6,61	11,40	10,77			4,59	7,50
navigazione4	0	16,14	10,39	1,78	12,54	4,85	3,77	6,01	7,07

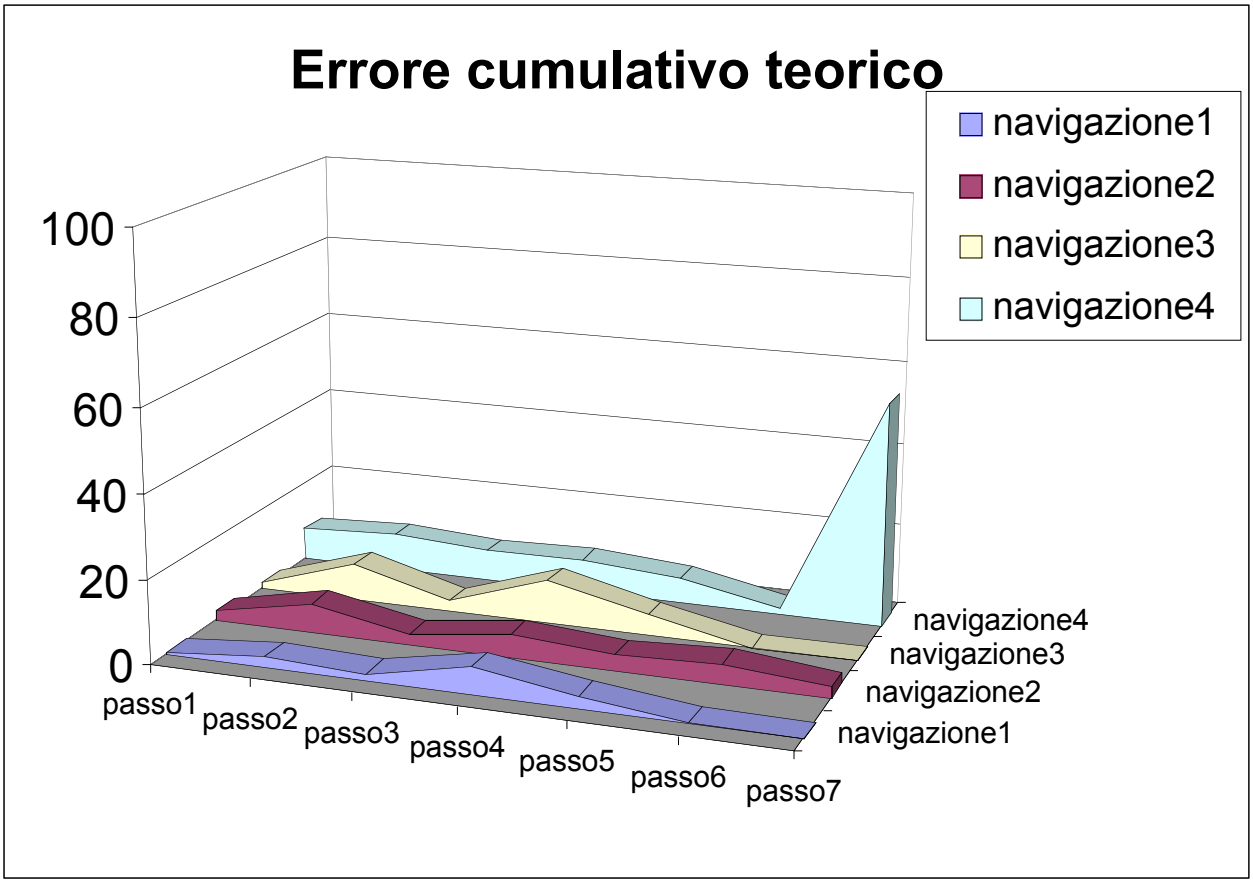
ERRORE CUMULATIVO TEORICO

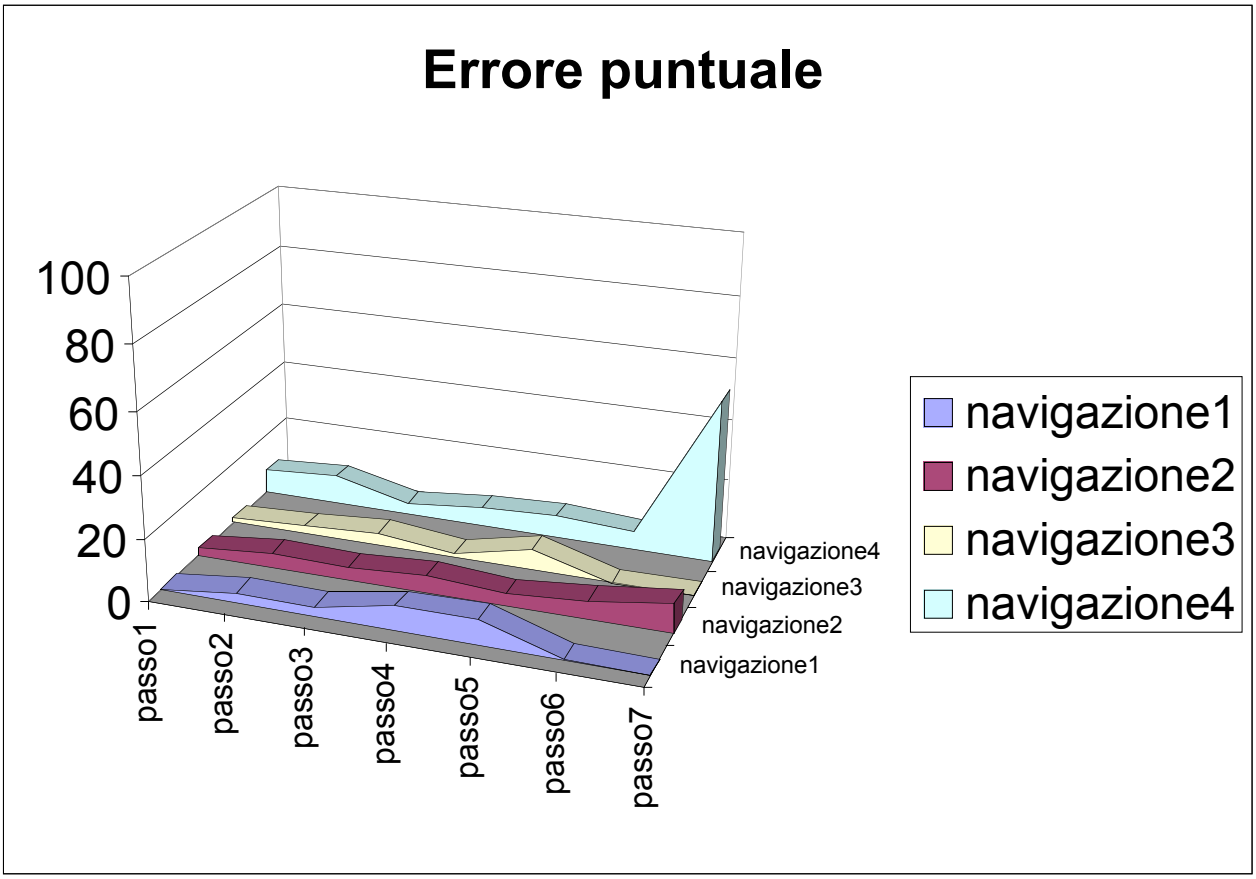
	passo1	passo2	passo3	passo4	passo5	passo6	passo7	dev standard	media
navigazione1	0,36	2,39	1,62	6,46	2,76			2,29	2,72
navigazione2	2,24	6,93	2,32	5,37	3,66	4,82	2,62	1,78	4,00
navigazione3	1,47	8,78	2,71	10,38	5,01			3,83	5,67
navigazione4	7,93	9,07	7,55	7,79	5,84	1,15	54,08	18,15	13,35

ERRORE PUNTUALE

	passo1	passo2	passo3	passo4	passo5	passo6	Passo7	dev standard	media
navigazione1	0,36	3,32	3,08	7,99	8,04			3,36	4,56
navigazione2	2,24	4,71	4,04	6,05	4,33	6,07	9,76	2,36	5,31
navigazione3	1,47	2,80	4,50	1,69	7,28			2,41	3,55
navigazione4	7,93	9,59	3,71	6,02	7,52	5,61	52,43	17,37	13,26


Errore cumulativo
navigazione1
navigazione2
navigazione3
navigazione4
40
30
20
10
0
passo1
passo2
passo3
passo4
passo5
passo6
passo7
navigazione4
navigazione3
navigazione2
navigazione1


Errore cumulativo teorico
navigazione1
navigazione2
navigazione3
navigazione4
100
80
60
40
20
0
passo1
passo2
passo3
passo4
passo5
passo6
passo7
navigazione4
navigazione3
navigazione2
navigazione1



Conclusioni

I risultati ottenuti evidenziano un buon andamento dell'errore cumulativo che si mantiene pressoché costante lungo tutta la navigazione.
Come ci si poteva aspettare, gli errori commessi in ambiente condizionato sono molto bassi e le stime sono abbastanza precise: errore di circa il 10% sul passo eseguito durante la navigazione.
In ambiente non condizionato (studio 35), in presenza di pochi riferimenti colorati nella stanza, e con pareti di colore castano chiaro, gli errori di stima aumentano, come si può osservare dagli errori di passo nella navigazione. Una delle cause degli errori del sistema Pollicino è da attribuirsi alla scarsa sensibilità e precisione dei movimenti del robot. Infatti il sistema Pollicino da noi implementato si è rivelato piuttosto pesante dal punto di vista dell'inerzia ai movimenti e ciò si ripercuote in errori nella fase di rotazione del robot. La rotazione virtuale delle immagini durante la fase di learn (eseguita con il programma *shift*) ha in parte abbassato la sensibilità della rete ad errori angolari dati da una visione dell'ambiente ruotata di un certo angolo. Ciò si è rivelato molto efficace in quanto, in assenza di questa modifica, prove da noi effettuate dimostrano che sono sufficienti angoli di rotazione di soli 5 gradi per avere errori inaccettabili nella stima della posizione.

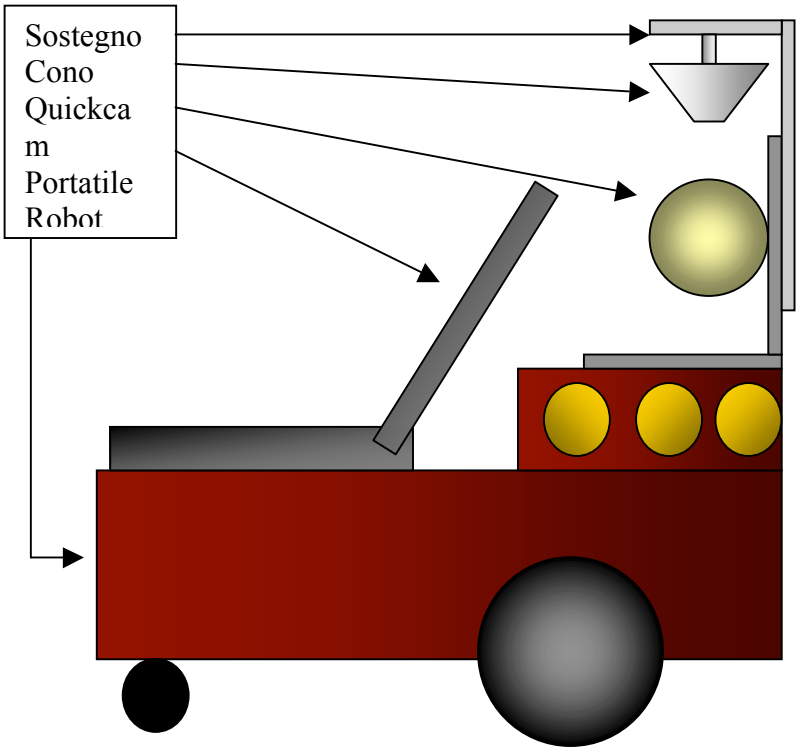
Seguono ora considerazioni personali e soggettive basate sulla pratica e sulla nostra esperienza personale per migliorare le prestazioni del sistema Pollicino. Un problema da risolvere è certamente

il peso del sistema portatile + telecamera + supporto metallico, il quale preclude movimenti agili e precisi del robot. Sarebbe inoltre opportuno non avere ruote pivottanti.
Un altro accorgimento potrebbe essere quello di utilizzare un sostegno trasparente per la quickcam; si risolverebbe così il problema della inevitabile ed onnipresente riga nera in ogni immagine presa.
Un ambiente ideale per la navigazione di Pollicino dovrebbe avere molti ostacoli colorati e vicini alla quickcam, in modo da aumentare il contenuto informativo delle immagini prese, soprattutto per le immagini di learning. Ambienti con queste caratteristiche sono soprattutto studi piccoli e con colori vivaci, non certo lo studio 35 in cui abbiamo fatto le navigazioni.
Un elevato numero di immagini di learning certamente aumenta la sensibilità e la precisione della stima delle posizioni. A nostro avviso sarebbe utile implementare un software per l'apprendimento automatico delle immagini di learning tramite navigazioni ad hoc opportunamente pilotate durante le quali sia possibile la cattura di un elevato numero di immagini. Il posizionamento manuale del robot in fase di learning si è rivelato faticoso e lungo.
Le navigazioni effettuate nella stanza (ambiente poco condizionato), mostrano che gli errori dovuti alla rete neurale nella stima della posizione e dai motori/ruote del robot nel portarsi nella posizione dell'obiettivo si mantengono abbastanza bassi. In particolare si è notato che passando dalla griglia a bassa densità di punti a quella ad alta densità, questi errori non cambiano significativamente.
Probabilmente, aumentando il numero di cicli di addestramento della rete neurale si otterrebbero risultati più precisi per quel che riguarda le stime.

Appendici

a.1) fotografia del robot e schema dell'hardware utilizzato
a.2) kaba.c
a.3) train_net
a.4) Agc
a.5) Attendi.c
a.6) Field
a.7) Genenv
a.8) Genpat_modificato
a.9) Generamap_modificato
a.10) Regenera1test_modificato
a.11) Messa_a_punto_cam
a.12) Testall
a.13) Shift.c
a.14) Stimaxy.c
a.15) link internet

a.1) fotografie del robot e schema dell'hardware utilizzato



a.2) file kaba.c per la navigazione del robot

```
/* programma per la stima della posizione del robot. I principali passi eseguiti
sono :
   1-   acquisizione delle coordinate dell'obiettivo da raggiungere
   2-   acquisizione dell'immagine della posizione corrente
   3-   estrazione delle coordinate attuali stimate dalla posizione corrente
tramite la rete neurale
   4-   calcolo della distanza tra la posizione stimata e l'obiettivo
   5-   movimento del robot in base alla posizione stimata
   6-   ripeti dal passo 2
*/

#include "stdio.h"
#include "stdlib.h"
#include "math.h"
#include "errno.h"
#include "saphira.h"
#include "string.h"

#define ROTATEVEL  4            /* velocita' di rotazione */
#define VELOCITY  300           /* velocita' di traslazione del robot */
#define ERR_MAX 5  /* massimo errore tollerato nel raggiungere l'obiettivo*/
#define DELAY 5     /* durata in secondi dell'attesa */

void denormalizza (float xin, float yin );
int network(float *in, float *out, int init);
void ritorna_coordinate ( char nome_file[]);

/*sezione variabili globali*/
float X_stimato = 0.0;
float Y_stimato = 0.0;

int main (void)
{

  extern int errno;
  int val_sys;
  char carat;

  float X_obiettivo = 0.0;
  float Y_obiettivo = 0.0;

/*************************************************************************/

  sfStartup(1);          /* start up the Saphira window, and then keep going */
  sfMessage("Connect to robot to start this demo");
  while (!sfIsConnected) sfPause(100); /* wait until we're connected */
  sfMessage("Waiting a little...");
  sfPause(5000);        /* wait for connection to settle */


/************************* ciclo principale ********************************/

do {

/********** acquisisce le coordinate dell'obiettivo da raggiungere***********/
  printf (" Inserisci la posizione dell'obiettivo \n");
```

```
  printf (" X ( espresso in cm ) ---> ");
  scanf ("%f" ,&X_obiettivo);
  printf (" Y ( espresso in cm ) ---> ");
  scanf ("%f" ,&Y_obiettivo);

  val_sys=system("testall | agc -1 +1 > IMM_ACQ.r_g_b");

  /******************* acquisisce l'immagine da quick **********************/

  ritorna_coordinate( "IMM_ACQ.r_g_b" );
  printf(" X_stimato=%f Y_stimato=%f X_obiettivo=%f \n
Y_obiettivo=%f\n",X_stimato,Y_stimato,X_obiettivo, Y_obiettivo);

  /************ blocco di istruzioni per muovere il robot di Dx, DY **********/

  sfSetMaxVelocity(VELOCITY);      /* setta velocita' di traslazione */
  sfSetPosition((int)((Y_obiettivo - Y_stimato)*10));
       /* avanza di Y_obiettivo - Y_stimato millimetri */
  sfSMessage("Avanza di DY = %d", (int)((Y_obiettivo-Y_stimato)*10));
  while ( !sfDonePosition(5))
                /* aspetta che sia arrivato alla posizione voluta */
       sfPause(100);
  sfSetHeading(-90);/*gira di 90 gradi */
  while ( !sfDoneHeading (5))
  sfPause(100);   /* aspetta che sia girato */
  sfSetPosition((int)((X_obiettivo - X_stimato)*10));
  /* avanza di X_obiettivo-X_stimato millimetri */
  sfSMessage("Avanza di DX = %d", (int)((X_obiettivo - X_stimato)*10) );
  while ( !sfDonePosition(5))
      sfPause(100);   /* aspetta che sia arrivato alla posizione voluta */
  sfSetDHeading(90);           /* gira di -90 gradi e torna dritto */
  while ( !sfDoneHeading (5))
       sfPause(100);   /* aspetta che sia girato */

  sfPause(3000);
  printf("ho restituito le coordinate\n");
  printf (" X_stimato = %f \n Y_stimato = %f \n", X_stimato, Y_stimato );
  printf ("continua ?(S/N)\n");
  scanf ("%c", &carat);
/********************** premi 's' o 'S' per continuare ********************/
}while((carat == 's') || (carat == 'S'));
sfDisconnectFromRobot();      /* we're gone... */
}


/* funzione che restituisce le coordinate della posizione stimata X,Y
rispetto alla griglia definita come ambiente operativo */

void ritorna_coordinate ( char nome_file[] )
{
FILE *fp;
int n_unit_input;
char stringa[8];
int test_end;
static struct {
                  int NoOfInput;    /* Number of Input Units  */
                  int NoOfOutput;   /* Number of Output Units */
                  int(* propFunc)(float *, float*, int);
              } networkREC = {1080,2,network};
```

```
float *netInput; float *netOutput; /* array di input e di output della rete
STD3BP */

fp = fopen ( nome_file, "r");

if ( fp == NULL)
            {
            printf (" non trovo il file %s\n", nome_file);
            exit(0);
            }

/*
   assegna uno spazio in memoria sufficiente a contenere gli array di input e di
output che vengono letti dal file IMM_ACQ
*/

netInput = malloc ( (float) (networkREC.NoOfInput * sizeof (float) ) );
netOutput = malloc ( (float) (networkREC.NoOfOutput * sizeof ( float) ) );

for ( n_unit_input = 0; n_unit_input < networkREC.NoOfInput; n_unit_input++ )
  {
    test_end = fscanf ( fp, "%s", stringa);     /* legge stringhe di caratteri */
    netInput[n_unit_input]=(float) atof(stringa);/*converte stringhe in float */
    if ( test_end == -1 )           /* controlla la corretta lettura del file */
         printf ("file non letto e non assegnato correttamente a netInput\n");
  }

network ( netInput, netOutput, 0);

/* denormalizzazione dell'output per renderlo compatibile con le dimensioni
della griglia */

denormalizza(netOutput[0],netOutput[1]);

/* chiusura del file e rilascio degli spazi di memoria */

fclose(fp);
free(netInput);
free(netOutput);

}


/* funzione che accetta come parametri di ingresso due valori compresi tra 0  e
1 e restituisce due valori nelle variabili globali X_stimato, Y_stimato*/

void denormalizza ( float xin, float yin )
{

      FILE *ppf;
      char stringa_letta[4][11];          /* puntatore al file scale.def */

      float scalex;
      float scaley;
      float xoffs;
      float yoffs;
      float X_reale;
      float Y_reale;
      int i, test;
```

```
      ppf = fopen("scale.def","r");

      if ( ppf == NULL)                   /* verifica esistenza file*/
            {
            printf ("\nNon trovo il file scale.def\n");
            exit(0);
            }

      for ( i=0 ; i <= 3; i++){
         test = fscanf(ppf, "%s", stringa_letta[i]);/*legge stringhe di
caratteri */
            if ( test == -1 )  /* controlla la corretta lettura del file */
            printf("file non letto e non assegnato correttamente a netInput\n");
            }

      fclose(ppf);
      scalex = (float) atof(stringa_letta[0]);  /* converte stringhe in float */
      scaley = (float) atof(stringa_letta[1]);  /* converte stringhe in float */
      xoffs = (float) atof(stringa_letta[2]);   /* converte stringhe in float */
      yoffs = (float) atof(stringa_letta[3]);   /* converte stringhe in float */
      X_stimato = scalex * xin + xoffs;
      Y_stimato = scaley * yin + yoffs;

 }
```

a.3) train_net

```
#!/bin/sh

echo -n "Inserisci il nome della griglia ( senza alcuna estensione ): "
read nomegriglia
#crea le sottodirectory
genenv $nomegriglia.env
#setta gli accessi
chmod -R 777 $nomegriglia.env
mkdir nets
cp std3bp.net ./nets/
cp snns2c ./nets/

#ciclo while per la cattura delle immagini#
echo
echo -e "######################################################################\n"
echo -e "  ciclo per l'immissione delle immagini e delle loro coordinate   \n"
echo -e "######################################################################\n"
echo
ancora=si
while test "$ancora" != n
do
      echo -n "Inserisci il nome dell'immagine  (senza .r_g_b): "
      read nome_imm
      testall > $nome_imm
        rm *.pgm
      #aggiunta dell'estensione .r_g_b
      nome_i_rgb="$nome_imm.r_g_b"
      #procedo alla normalizzazione
      agc -1 +1 $nome_imm > $nome_i_rgb
      echo -n "Inserisci la coordinata X (in cm con tre cifre nel formato xxx):
"
      read X
```

```
      echo -n "Inserisci la coordinata Y (in cm con tre cifre nel formato xxx):
"
      read Y
      #copia il file normalizzato in /data
      cp $nome_i_rgb ./$nomegriglia.env/data/
      #copia il file normalizzato in /maps
      cp $nome_i_rgb ./$nomegriglia.env/maps/
      #inserisco i dati in all.map
      cd $nomegriglia.env/maps/
      echo -n "$nome_i_rgb " >> all.map
      echo -n "$X " >> all.map
      echo "$Y " >> all.map
      cd ..
      cd ..
      #cancello i file delle immagini dalla dir attuale
      rm $nome_imm
      rm $nome_i_rgb


      echo -n "Devo catturare altre immagini?(s/n)"
      read ancora
      echo
done

#crea la sottodirectory test0
cd $nomegriglia.env/test/
mkdir test0
cd ../..
# fino a qui ho tutte le directory, tutte le immagini gia' normalizzate
#e mi trovo nella /bin

generamap_modificato $nomegriglia.env learn.map
#sposta i file *.map e *.def nella /test0
cp ./$nomegriglia.env/maps/learn.* ./$nomegriglia.env/test/test0/
genpat_modificato $nomegriglia.env test0

cp ./shi.exe ./$nomegriglia.env/test/test0/
cd ./$nomegriglia.env/test/test0
cp learn.pat learn_piccolo.pat
shi.exe learn_piccolo.pat learn.pat
cd /home/polli/bin/


regenera1test_modificato $nomegriglia.env test0
cp snnsbat $nomegriglia.env/test/test0
cd $nomegriglia.env/test/test0
snnsbat batch.cfb oo.log
echo "Vuoi vedere il risultato del learning ? (s/n)"
read risp
#il file di output si chiama oo.log

if test "$risp" = s; then cat oo.log |more ; fi
cd ../../..

#ora sono nella /bin e devo fare snns2c

#vado ora nella nets
cd nets
snns2c network.net network.c

cp /home/polli/bin/stimaxy.c ./
```

```
cp /home/polli/bin/make_stimaxy ./
cp /home/polli/bin/make_kab ./
cp /home/polli/bin/kaba.c ./
cp /home/polli/bin/$nomegriglia.env/test/test0/scale.def ./
cp /home/polli/bin/testall /home/polli/bin/nets/
cp /home/polli/bin/qcread /home/polli/bin/nets/
cp /home/polli/bin/attendi /home/polli/bin/nets/
cp /home/polli/bin/quick /home/polli/bin/nets/
cp /home/polli/bin/agc /home/polli/bin/nets/

gcc -c network.c

make -f make_stimaxy stimaxy
make -f make_kab kaba

cp ./stimaxy ../$nomegriglia.env/test/test0
cp ./kaba ../$nomegriglia.env/test/test0
```

a.4) agc

```
#! /bin/sh
if [ $# -lt 2 -o $# -gt 3 ]
then
      echo "usage: agc minval maxval [ signalfile ]\n" 1>&2
      exit 1
fi
awk '
      { if (ymin == "") {
            ymin = $1; ymax = $1
        } else {
            if ($1 < ymin) ymin = $1;
            if ($1 > ymax) ymax = $1;
        }
        signal[NR] = $1;
      }
END   { range = (ymax - ymin);
        scale = (maxv - minv) / range;
        for (i=1; i<=NR; i++)
            printf("%f\n", minv + (scale * (signal[i]-ymin)) );
      } ' minv="$1" maxv="$2" $3
```

a.5) attendi.c

```
// il programma attendi.c esegue una pausa di un numero di secondi pari a
argv[1], percio' va
// chiamato con un argomento che sia un numero intero di secondi
#include "stdio.h"

int main ( int argc, char *argv[])
{
sleep( atoi(argv[1]));
}
```

a.6) field

```
#! /bin/sh

if [ $# -lt 1 -o $# -gt 2 ]
then
      echo "usage: field numfield [file|-]" 1>&2
      exit 1
fi
n="$1"
if [ "$2" = "" ]
then
      f="-"
else
      f="$2"
fi
awk '
BEGIN { FS=",[ \t]*|[ \t]+" }
      {
            print $n
      }
' n="$n" $f
```

a.7) genenv

```
#! /bin/sh
#
#


#
# checkparam $*
#

checkparam()
{
      if [ $# -gt 2 ]
      then
            return 1
      fi
      if [ $# -lt 1 ]
      then
            return 1
      fi
      env=`basename $1 '.env' `
        if [ -d $env.env ]
      then
            echo -e "\nOverwrite existing $env.env directory (y/n) ? \c"
            read answer
            echo ""
            if [ "$answer" != "y" ]
            then
                  return 1
            fi
      fi
      if [ "$2" != "" ]
      then
            if [ ! -d "$2" ]
            then
                  echo -e "can't find original map directory $2\n"
```

```
                return 1
        fi
    fi
    return 0
}


#
# main
#
#


if checkparam $*
then
    env=`basename $1 '.env'`
    mkdir $env.env
    mkdir $env.env/data
    mkdir $env.env/test
    mkdir $env.env/maps
    if [ "$2" != "" ]
    then
        cp $2/*.map $2/*.def $env.env/maps
    fi
else
    echo -e "usage: `basename $0` nom.env [mapdirectoryname]\n"
    exit 1
fi
exit 0
```

a.8) genpat_modificato

```
#! /bin/sh

#
# checkparam envname testname
#

checkparam()
{
    if [ "$1" -a "$2" ]
    then
        env=`basename $1 '.env'`
        test=`basename $2`
        testenv="$env.env/test/$test"
        if [ ! -d $testenv ]
        then
            return 1
        fi
        if [ ! -f $testenv/learn.map ]
        then
            echo "can't find file $testenv/learn.map "
            return 1
        fi
#       if [ ! -f $testenv/test.map ]
#       then
#           echo "can't find file $testenv/test.map "
#           return 1
#       fi
```

```
      else
            return 1
      fi
      return 0
}

#
# check if file(s) exists
#

fileexists()
{
      for f
      do
            if [ ! -f $f ] ; then return 1 ; fi
      done
      return 0
}


#
# mksnnspat datapath scalex scaley ofsx ofsy
# reads input lines as "filename x y \n"...
#
#mksnnspat()
#
#     while read name x y
#     do
#           awk '
#           {     printf("%1.5f", $1);
#                 if (NR % 10 == 0)
#                       printf("\n");
#                 else
#                       printf(" ");
#           }
#           END {
#                 printf("%1.5f %1.5f\n", (x - ofsx)/scalex, (y - ofsy)/scaley);
#           } ' x="$x" y="$y" scalex="$2" scaley="$3" ofsx="$4" ofsy="$5"
$datapath/$name
#     done
#     return 0

mksnnspat()
{
      while read name x y
      do
            echo '#I'
            cat $datapath/$name
            echo '#0'
            #echo "x=$x y=$y scalex=$2 scy=$3 ofx=$4 ofy=$5"
            date | awk '{
 printf("%1.5f %1.5f\n", (x - ofsx)/scalex , (y - ofsy)/scaley);
 } '  x="$x" y="$y" scalex="$2" scaley="$3" ofsx="$4" ofsy="$5"
      done
      return 0
}

#
# genscale :
# reads a real map in stdin and write to stdout "scalex scaley offsetx offsety"
# xreal = scalex * xnetwork + xoffset           xmin <= x <= xmax
```

```
# xnetwork = (xreal - xoffset) / scalex          -vmax <= xnetwork <= +vmax
#

genscale()
{
      awk ' { if (xmin == "") {
                  xmin=$2;
                  ymin=$3;
                  xmax=$2;
                  ymax=$3;
                  vmax=1;
              } else {
                  if ($2 < xmin) { xmin = $2 }
                  if ($3 < ymin) { ymin = $3 }
                  if ($2 > xmax) { xmax = $2 }
                  if ($3 > ymax) { ymax = $3 }
              }
            }
      END   {
              xoffs = (xmin + xmax) / 2;
              yoffs = (ymin + ymax) / 2;
              scalex = (xmax - xmin) / (vmax * 2);
              scaley = (ymax - ymin) / (vmax * 2);
              if (scalex > scaley) { scaley=scalex } else {scalex=scaley};
              printf("%f %f %f %f\n",scalex,scaley,xoffs,yoffs);
      } ' -
#echo "valore di scalex = $scalex, valore di scaley = $scaley, valore di
#xoffs = $xoffs, valore di yoffs = $yoffs"
}


#
# genpat envname testname
#

if checkparam $*
then
      learnmap="$testenv/learn.map"
#     testmap="$testenv/test.map"
      learnpat="$testenv/learn.pat"
#     testpat="$testenv/test.pat"
      datapath="$env.env/data"
#echo "sto entrando in genscale .."
      scale=`cat $learnmap | genscale | tee $testenv/scale.def`
#echo "sono uscito da genscale .."
      fnam=`tail -1 $learnmap | field 1`
#echo "ho controllato field1"
      numinps="`cat $datapath/$fnam | wc -l`"
#echo "ho fatto il cat"
      learncount=`cat $learnmap | wc -l`

#     testcount=`cat $testmap | wc -l`
#echo "valore di $testcount"
      echo -e "genpat: generating $learnpat (" $learncount ") ...\t\c"
      mkhead $learncount $numinps 2 > $learnpat
#echo "entro in mksnnspat"

#echo "$datapath $scale"

      cat  $learnmap | mksnnspat $datapath $scale >> $learnpat
      echo "done."
```

```
#      echo -e "genpat: generating $testpat  (" $testcount ") ...\t\c"
#      mkhead $testcount $numinps 2 > $testpat
#      echo " mksnnspat"
#      cat $testmap | mksnnspat $datapath $scale >> $testpat
#      echo "done."
else
      echo -e "usage: `basename $0` environment.env testname \n"
      exit 1
fi
```

a.9) generamap_modificato

```
#! /bin/sh

#
# checkparam "$*"
#

checkparam()
{
        if [ $# -lt 2 ]
        then
                return 1
        fi
        env=`basename $1 '.env' `
        map=`basename $2 '.map' `
        mapfile="$env.env/maps/$map.map"
        mapdef="$env.env/maps/$map.def"
        if [ ! -d $env.env ]
        then
                echo "can't find $env.env directory"
                return 1
        fi
        if [ ! -d $env.env/maps ]
        then
                echo "can't find $env.env/maps directory"
                return 1
        fi
        if [ ! -f $env.env/maps/all.map ]
        then
                echo "can't find $env.env/maps/all.map file"
                return 1
        fi
        if [ "`wc -l $env.env/maps/all.map`" = "0" ]
        then
                echo "file $env.env/maps/all.map is empty"
                return 1
        fi
        return 0
}


expandnames()
{
        (cd $env.env/data; ls $* 2>/dev/null)
        return 0
}

expandnames()
```

```
{
        (cd $env.env/data; ls $* 2>/dev/null)
        return 0
}

asknames()
{
        echo -e  "\nList of files in $env.env/maps/all.map :\n"
        vectors=`cat $env.env/maps/all.map | cut -f1 -d" "`
        for v in $vectors
        do
                echo -e "$v\t\c"
        done
        echo ""
        #echo -e "\nNames to be added to $mapfile, separated by spaces :\n ? \c"
        answer=$vectors
        echo ""
        if [ ! "$answer" ]
        then    echo "File $mapfile not saved!"
                return 1
        fi
        names=`expandnames $answer | sort | uniq`
        return 0
}

#
# build destmapname
#

build()
{
        count=0
        echo -e "\c" > $mapdef
        for f in $names
        do

            echo -e "$f\t\c" >> $mapdef
                grep "^$f[ \t]" $env.env/maps/all.map  >> $mapfile
                count=`expr $count + 1`
        done
        echo "" >> $mapdef
}

#
# main
#
#


if checkparam $*
then
        if [ $# -gt 2 ]
        then
                shift 2
                answer="$*"
                names=`expandnames $answer | sort | uniq`
                 if [ "$names" ]
                then
                        build
                        echo "$count files in $mapfile"
                fi
```

```
        else
                asknames && build > $mapfile
                echo "$count files in $mapfile"
        fi
else
        echo "usage: `basename $0` nom.env nom.map [ filespec ... ]"
        exit 1
fi
exit 0
```

a.10) regenera1test_modificato

```
#! /bin/sh

#
# checkparam envname testname
#

checkparam()
{
      if [ $# -ne 2 ]
      then
            return 1
      fi

      env=`basename $1 '.env'`
        test=`basename $2`


        testenv="$env.env/test/$test"
      maps="$env.env/maps"

        if [ ! -d $env.env/test ]
      then
            echo "can't find $env.env/test directory "
            return 1
      fi
      if [ ! -d $testenv ]
      then
            echo "Can't regenerate nonexisting $testenv directory "
            echo "use generatest instead."
            return 1
      fi
      learnmap=/home/polli/bin/$maps/learn.map
#     testmap=`cat $testenv/test.def`
      network=/home/polli/bin/nets
        return 0
}

#
# check if file(s) exists
#

fileexists()
{
        for f
        do
                if [ ! -f $f ] ; then return 1 ; fi
```

```
        done
        return 0
}


#
#
#


askcomment()
{
      if [ -f $testenv/comment ]
      then
            echo "**** Old Comment ****"
            cat $testenv/comment
      fi
       echo "# Test created `date`" > $testenv/comment
       echo "# learn map: $learnmap " >> $testenv/comment
#       echo "# test map : $testmap " >> $testenv/comment
       echo "# network  : $network " >> $testenv/comment
      if [ -f $testenv/comment ]
      then
            echo -e "Do you want to enter or change the comment ? (y/n) \c"
            read answer
      else
            answer="y"
      fi
      echo ""
      if [ "$answer" = "y" ]
      then
             echo -e  "\nEnter a description for $testenv \n(type ENTER 2 times
to terminate) \n ? \c"
             while read answer
             do
                  if [ "$answer" ]
                  then
                        echo "# comment  : $answer" >> $testenv/comment
                  else
                        break
                  fi
             done
      fi
       return 0
}

#
# $1=prompt $2=nomvar $3 =default
#

ask()
{
      nomvar=$2
      echo -e "$1\t($3)\t? \c";
      read answer
      if [ "$answer" ]
      then
            value=$answer
      else
            value=$3
      fi
```

```
      eval $nomvar=$value
}
#! /bin/sh
#
#

#
# $1 = file par
#

askparams()
{
      read maxit stepit kick tolerance < $testenv/param.def
      ask "Network to use             " network $network/std3bp.net
      ask "Learn pattern map to use   " learnmap $learnmap learn.map
#     ask "Test  pattern map to use   " testmap $learnmap learn.map
      ask "Max batch iteration        " maxit $maxit 500
      ask "Iteration step for results " stepit $stepit 500
      ask "Max random initialization  " kick $kick 0.1
      ask "Tolerance for convergence  " tolerance $tolerance 0.01
      ask "Number of hidden units     " hidden 25
      ask "Seed for random generator  " seed 7
      ask "Sigmoid sharpness          " sharpness 0.5
#     echo "kick" $kick >> $1
#     echo "hidden" $hidden > $1
#     echo "seed" $seed >> $1
#     echo "sharpness" $sharpness >> $1
#     echo "tolerance" $tolerance >> $1
#     echo "maxit" $maxit >> $1
#     echo "stepit" $stepit >> $1
}

#
# fnumber 12 --> "00000012"
#

fnumber()
{
      echo "$1" | awk '{ printf("%08d\n", ($1 + 0)) }'
}



makebatch()
{


env2=`pwd`
test2="$env2/$env.env/test/$test"

#
i=$stepit
echo "#"
echo "Type: SNNSBATCH_2"
echo "#"
echo "# The following keyword-value combinations may be supplied in any order."
echo "# If a key is given twice, the second appearence is taken.  "
echo "# Keys that are not required for a special run may be omitted.  "
echo "# If a key is ommited but required, a default value is assumed.  "
echo "# The lines may be separated with comments.  "
echo "# "
```

```
echo "# Please note the mandatory file type specification at the begining and "
echo "# the colon following the key.  "
echo "#                                              Defaults: "
echo "# "
echo "#     NoOfLearnParam:      <int>                0"
echo "#     LearnParam:          [ <float> ]*         0.0"
echo "#     NoOfInitParam:       <int>                0"
echo "#     InitParam:           [ <float> ]*         0.0"
echo "#     NetworkFile:         <string>             \"\""
echo "#     TrainedNetworkFile:  <string>             \"\""
echo "#     LearnPatternFile:    <string>             \"\""
echo "#     TestPatternFile:     <string>             \"\""
echo "#     ResultFile:          <string>             \"\""
echo "#     InitFunction:        <string>             \"\""
echo "#     MaxLearnCycles:      <int>                0"
echo "#     MaxErrorToStop:      <float>              0.0"
echo "#     Shuffle:             [ YES | NO ]         NO"
echo "#     ResultMinMaxPattern: <int><int>           0 0"
echo "#     ResultIncludeInput:  [ YES | NO ]         NO"
echo "#     ResultIncludeOutput: [ YES | NO ]         YES"
echo "#     PerformActions:"
echo "#"
echo "##############################################################################"
echo "#"
echo "#### default configuration"
echo "#"
echo "NetworkFile: $network"
echo "#"
echo "InitFunction: Randomize_Weights "
echo "NoOfInitParam: 2"
echo "InitParam: -$kick $kick"
echo "#"
echo "LearnPatternFile: $test2/learn.pat"
echo "NoOfLearnParam: 1"
echo "LearnParam: 0.2"
echo "MaxLearnCycles: $stepit"
echo "MaxErrorToStop: $tolerance"
echo "Shuffle: YES"
echo "TestPatternFile: $test2/learn.pat"
echo "ResultFile: `fnumber $i`.res"
echo "ResultMinMaxPattern: 1 `echo $testpatcount`"
echo "ResultIncludeInput: NO"
echo "ResultIncludeOutput: YES"
echo "TrainedNetworkFile:  $env2/nets/network.net"
echo "#"
echo "#### end of default configuration"
echo "#"
echo "#### start of cmds ####"
while [ $i -le $maxit ]
do
      next=`expr $i + $stepit`
      echo "#"
      echo "PerformActions:"
      echo "NetworkFile: <OLD>"
      echo "LearnPatternFile: $test2/learn.pat"
      echo "NoOfLearnParam: <OLD>"
      echo "LearnParam: <OLD>"
      echo "MaxLearnCycles: <OLD>"
      echo "MaxErrorToStop: <OLD>"
      echo "Shuffle: <OLD>"
      echo "TestPatternFile: <OLD>"
```

```
      echo "TrainedNetworkFile: <OLD>"
      if [ $i -ge $maxit ]
      then
            echo "ResultFile: $env2/last.res"
      else
            echo "ResultFile: `fnumber $next`.res"
      fi
      echo "ResultMinMaxPattern: <OLD>"
      echo "ResultIncludeInput: <OLD>"
      echo "ResultIncludeOutput: <OLD>"
      i=$next
done
echo "#"
}




#
# build
#

build()
{
      b="$testenv/batch.cfb"
      echo -e "\nEnter new parameters for $testenv :\n"
      askparams
      askcomment
      echo ""
      echo -e "\nRegenerating $testenv/*.def...\c"
      echo $learnmap > $testenv/learn.def
        echo $network > $testenv/net.def
      echo "$maxit $stepit $kick $tolerance" > $testenv/param.def
# ho tolto da riga sotto $maps/$testmap
      testpatcount=`cat $learnmap | sort | uniq | wc -l`
#     echo $testmap > $testenv/test.def
#     echo "done."
      echo -e "Building $b...\c"
      makebatch > $b
      echo "done."
        echo -e "\nEdit the batch command file $b ? (y/n) \c"
        read answer
        if [ "$answer" = "y" ]
        then
                echo -e "\nEditing the batch command for test $b ...\c"
                textedit $b &
                echo -e "done.\n"
      else
            echo -e "\n"
        fi

}

#
# main
#
#

if checkparam $*
then
      build
```

```
else
        echo "usage: `basename $0` nom.env nomtest"
      exit 1
fi
exit 0
```

a.11) messa_a_punto_cam

```
#!/bin/sh

#---------------script per la messa a fuoco della quickcam--------------------

# Lo script esegue un ciclo per richiedere tutti i parametri piu' importanti
# della quickcam e successivamente lancia il programma qcread con i parametri
# appena impostati.

echo -n " ****    script per la messa a fuoco della quickcam   ****"
echo
ripeti_test=si
while test "$ripeti_test" != n
do
      #echo -n "inserisci il valore di x : "
      #read xcoord
      #echo -n "inserisci il valore di y : "
      #read ycoord
      #echo -n "inserisci l'offset x : "
      #read xoffs
      #echo -n "inserisci l'offset di y : "
      #read yoffs
      #echo -n " Inserisci il valore di brightness ( 0-255 ) : "
      #read brightness
      #echo -n " Inserisci il valore di contrast ( 0-255 ) : "
      #read contrast
      #echo -n " Inserisci il valore di white level ( 0-255 ) : "
      #read white_level
      #echo -n " Inserisci il valore di black offset ( 0-255 ) : "
      #read black_offset
      #echo -n " Inserisci il valore di saturation ( 0-255 ) : "
      #read saturation
      #echo -n " Inserisci il valore di hue ( 0-255 ) : "
      #read hue
      #echo -n " Inserisci il valore di black level ( 0-255 ) : "
      #read black_level

      #lancio il programma qcread con i parametri appena impostati
        #./qcread -b $brightness --contrast $contrast --saturation $saturation |
xv -&
      #./qcread -b 200 --contrast 255 --saturation 150 -x $xcoord -y $ycoord -l
$xoffs -t $yoffs | xv -&
      ./qcread -b 190 --contrast 255 --saturation 150 -x 240 -y 240 -l 50 -t 0 |
xv -&
      echo -n " Ripeti il test ? ( s/n ) "
      read ripeti_test
done
```

a.12) attendi.c

```
#!/bin/sh

#echo -e " attendi ...\n"
./attendi 6
#echo -e " ... ho finito !\n"
qcread -x 240 -y 240 -s 1 -l 50  -t 0 -b 190 --contrast 255 --saturation 150 |
quick
```

a.13) shift.c

```
/* prova: legge e calcola max per net */

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <math.h>

#define REAL float
#define DIM  2160
#define DIM2 1080

int main (int argc, char *argv[])
{
  FILE *f1,*f2;
  REAL t,tt,n,th,x[DIM];
  REAL max,min,xx,yy;
  int i,j,l,imax,imin,segn[3];
  char line[500];


  printf (" *** INIZIO *** \n");
  f2 = fopen (argv[2],"w");
  fclose (f2);
  f1 = fopen (argv[1],"r");
  f2 = fopen (argv[2],"a");
  if (f1 && f2) {

    fprintf (f2,"SNNS pattern definition file V1.4\n");
    fprintf (f2,"generated at Mon Oct 19 09:22:29 1998\n");
    fprintf (f2,"\n");
    fprintf (f2,"\n");
    fprintf (f2,"No. of patterns     : 90\n");
    fprintf (f2,"No. of input units  : 1080\n");
    fprintf (f2,"No. of output units : 2\n");
    fprintf (f2,"\n");
    fprintf (f2,"\n");

    for (j=0; j<9; j++) {
      printf ("\nleggo immagine %i ... ",j);
      fgets (line,500,f1);
      //puts (line);
      while (!(isdigit(line[0]))){
        fgets (line,500,f1);
        //puts (line);
      }
      for (i=0; i<DIM2; i++) {
        sscanf(line,"%f \n",&t);
        // fscanf (f1,"%f \n",&t);
```

```
    for (l=0; l<2; l++)
      x[i+1080*l] = t;
    fgets (line,500,f1);
    //puts (line);
  }
  while (!(isdigit(line[0]))){
    fgets (line,500,f1);
    //puts (line);
  }
  fgets (line,500,f1);
  //puts (line);
  sscanf (line,"%f %f \n",&xx,&yy);

  /*
  imin = 0;
  min  = +1000.0;
  imax = 0;
  max  = -1000.0;
  for (i=0; i<360; i++) {
    t=(x[i]+x[i+360]+x[i+720]);
    if (t>max) {
      max  = t;
      imax = i;
    }
    if (t<min) {
      min  = t;
      imin = i;
    }
  }
  printf("\nrisultati %i %f %i %f %i",j,max,imax,min,imin);
  */

  printf("\nscrivo immagine %i ... ",j);


  /*
  for (l=0; l<10; l++) {
    printf (".");
    fprintf (f2,"#%i\n",j);
    for (i=0; i<DIM2; i++)
      fprintf (f2,"%f \n",x[i+l*108]);
    //t = ((REAL)(i)   - 4.5) / 4.5;
    //t = ((REAL)(imin)- 180) / 180;
    //tt= ((REAL)(imax)- 180) / 180;
    fprintf (f2,"%f %f \n",xx,yy);
  }
  */
  for (l=0; l<5; l++) {
    printf (".");
    fprintf (f2,"#%i\n",j);
    for (i=0; i<DIM2; i++)
      fprintf (f2,"%f \n",x[i+l*3]);
    //t = ((REAL)(i)   - 4.5) / 4.5;
    //t = ((REAL)(imin)- 180) / 180;
    //tt= ((REAL)(imax)- 180) / 180;
    fprintf (f2,"%f %f \n",xx,yy);
  }   f
  for (l=1; l<6; l++) {
    printf (".");
    fprintf (f2,"#%i\n",j);
    for (i=0; i<DIM2; i++)
```

```
          fprintf (f2,"%f \n",x[DIM2+i-l*3]);
        //t = ((REAL)(i)   - 4.5) / 4.5;
        //t = ((REAL)(imin)- 180) / 180;
        //tt= ((REAL)(imax)- 180) / 180;
        fprintf (f2,"%f %f \n",xx,yy);
      }

    }
    fclose (f1);
    fclose (f2);
    printf ("\n *** FINE *** \n");
  }
  return 0;

}
```

a.14) stimaxy.c

```
/*

Programma per la stima della posizione (x,y) del robot in base al risultato
della rete neurale STD3BP addestrata con SNNS e trasformata in funzione c dal
tool snns2c.

Utilizzo : stimaxy <nome_pattern_input>

*/
#include "stdlib.h"
#include "stdio.h"

int network(float *in, float *out, int init);

void denormalizza (float xin, float yin);


int main ( int argc, char *argv[] )
{
  /*
     dichiarazione delle variabili
  */

       static struct {
                  int NoOfInput;    /* Number of Input Units  */
                  int NoOfOutput;   /* Number of Output Units */
                  int(* propFunc)(float *, float*, int);
                  } networkREC = {1080,2,network};

      float *netInput; float *netOutput; /* array di input e di output della
rete STD3BP */

      FILE *fp; /* puntatore al file <nome_pattern_input> */

      char stringa[8];

      int n_unit_input; /* variabile di ciclo */

      int test_end; /* variabile di fine file */

      fp = fopen ( argv[1], "r");
```

```
      if ( argc > 2 || argc == 1)
            {
            printf (" Utilizzo : stimaxy <nome_pattern_input>\n");
            exit(0);
            }

      if ( fp == NULL)
            {
            printf (" non trovo il file %s\n", argv[1]);
            exit(0);
            }

/* assegna uno spazio in memoria sufficiente a contenere gli array di input e di
output
   che vengono letti dal file <nome_pattern_input>
*/

      netInput = malloc ( (int) (networkREC.NoOfInput * sizeof (int) ) );
      netOutput = malloc ( (int) (networkREC.NoOfOutput * sizeof ( float) ) );

      for ( n_unit_input = 0; n_unit_input < networkREC.NoOfInput;
n_unit_input++ )
            {
                  test_end = fscanf ( fp, "%s", stringa);               /* legge
stringhe di caratteri */
                  netInput[n_unit_input] = (float) atof(stringa);       /*
converte stringhe in float */

                                                                  /*    printf ("%f\n",
netInput[n_unit_input]);  */
                  if ( test_end == -1 )                           /* controlla la
corretta lettura del file */
                        printf ("file non letto e non assegnato correttamente a
netInput\n");
            }


      network ( netInput, netOutput, 0);

      printf ("La prima uscita normalizzata dalla rete e' = %f\nLa seconda
uscita e'= %f\n", netOutput[0], netOutput[1] );
      fclose(fp);
      denormalizza(netOutput[0], netOutput[1]);
      free(netInput);
      free(netOutput);

}


/* denormalizza le coordinate della posizione ricavata utilizzando il contenuto
di
scale.def "scalex, scaley, xoffs, yoffs" */

void denormalizza (float xin, float yin)
{

      FILE *ppf;
      char stringa_letta[4][11];          /* puntatore al file scale.def */

      float scalex;
```

```
      float scaley;
      float xoffs;
      float yoffs;
      float X_reale;
      float Y_reale;
      int i, test;


      ppf = fopen("scale.def","r");

      if ( ppf == NULL)                    /* verifica esistenza file*/
            {
            printf ("\nNon trovo il file scale.def\n");
            exit(0);
            }

      for ( i=0 ; i <= 3; i++)
            {
            test = fscanf(ppf, "%s", stringa_letta[i]);     /* legge stringhe di
caratteri */
            if ( test == -1 )                          /* controlla la corretta
lettura del file */
                  printf ("file non letto e non assegnato correttamente a
netInput\n");
            }

      fclose(ppf);
      scalex = (float) atof(stringa_letta[0]);        /* converte stringhe in
float */
      scaley = (float) atof(stringa_letta[1]);        /* converte stringhe in
float */
      xoffs = (float) atof(stringa_letta[2]);   /* converte stringhe in float */

      yoffs = (float) atof(stringa_letta[3]);   /* converte stringhe in float */

      X_reale = scalex * xin + xoffs;
      Y_reale = scaley * yin + yoffs;

      printf("\nLa posizione stimata e' %f %f\n", X_reale, Y_reale);
 }
```

a.15) link internet

Per avere informazioni e aggiornamenti riguardo al robot Pioneer1 e all'ambiente Saphira :

http://robots.activemedia.com

Per avere una copia gratuita del simulatore SNNS nell'ultima versione, collegarsi a :

ftp://ftp.informatik.uni-stuttgart.de/pub/SNNS

Per scaricare l'ultima release dei driver per quickcam e imparare ad utilizzarla al meglio, il link è :

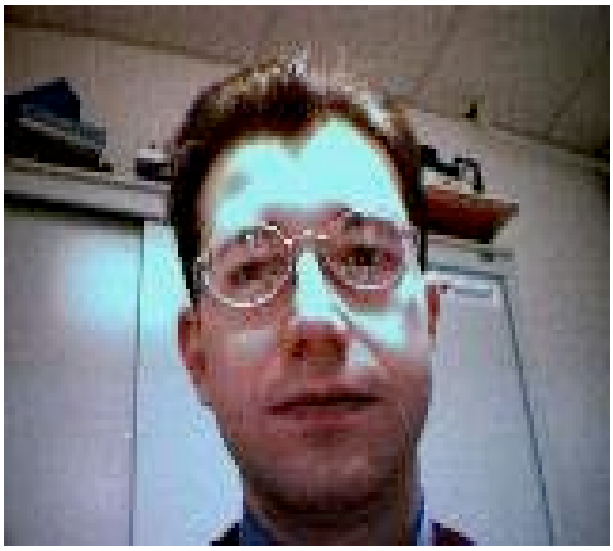
ftp://ftp.cs.unm.edu/pub/chris/quickcam

Il compilatore gcc utilizzato è stato scaricato all'indirizzo :

ftp://prep.ai.mit.edu/pub/gnu

Il sistema operativo utilizzato è Linux (versione 2.1.43)

Autori della relazione



Daniele Bonassi

Matricola: 025046
Data di nascita: 05/10/1974
Residente a Palazzolo S/O (BS)
Telefono: 0307300808
Hobby: appassionato di hardware

Carlo Spinoni

Matricola: 024986
Data di nascita: 02/09/1974
Residente a Pontoglio (BS)
Telefono 0307376260
Hobby: appassionato di software