Appendice A

Calcolo dell'angolo di inclinazione.

Dato un oggetto C' si ponga l'origine del sistema di riferimento x, y coincidente con il centro di massa $(\bar{x}, \bar{y})$ di C' (vedere figura A.1). Si identifichi un generico punto appartenente alla retta u' indicante l'asse di inerzia tramite[1]

$$\begin{cases} x_0 = S\cos\theta \\ y_0 = S\sin\theta \end{cases} \tag{A.1}$$

dove S è una coordinata retilinea che identifica con i suoi valori tutti i punti (x_0, y_0) della retta. Riprendendo la 3.55

$$I = \iint r^2 b(x,y)dxdy \tag{A.2}$$

possiamo scrivere

$$r^2 \quad = \quad (x - x_0)^2 + (y - y_0)^2 \tag{A.3}$$

sostituendo la A.1 si ottiene dopo alcuni passaggi

$$r^2 = x^2 + y^2 + S^2 - 2xS\cos\theta - 2yS\sin\theta \tag{A.4}$$

Derivando rispetto ad S ed annullando la derivata, si ricava per un generico punto (x, y), il punto (x_0, y_0) sulla retta u' che minimizza la distanza r, per cui

[1]Notare che l'asse di inerzia passa per $(\bar{x}, \bar{y})$, questa è una proprietà fisica del centro di massa che permette di semplificare i calcoli prendendo quest'ultimo come origine del sistema di riferimento.

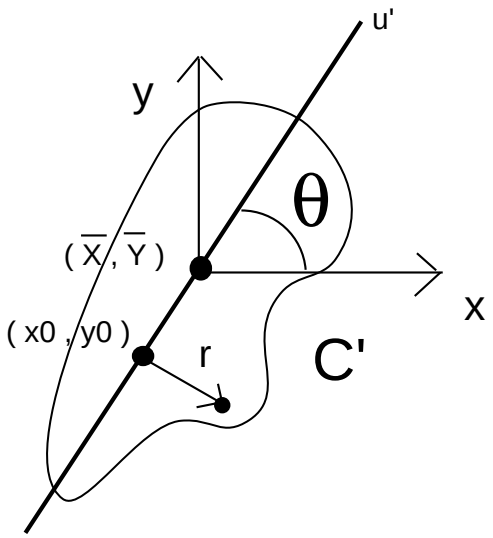

Figura A.1: Asse di inerzia di un oggetto.

$$\frac{\partial(r^2)}{\partial S} = 2x + 2y + 2S - 2x\cos\theta - 2y\sin\theta = 0 \tag{A.5}$$

É possibile calcolare la coordinata retilinea S che verifica la A.5

$$S = x\cos\theta + y\sin\theta \tag{A.6}$$

sostituendo A.6 nella A.3 ricaviamo l'espressione indicante r^2

$$r^2 = (x - (x\cos\theta + y\sin\theta)\cos\theta)^2 + (y - (x\cos\theta + y\sin\theta)\sin\theta)^2 \tag{A.7}$$

sviluppando e raccogliendo opportunamente risulta

$$\begin{aligned} r^2 &= x^2(1+\cos^4\theta) + y^2(1+\sin^4\theta) - 2xy\cos\theta\sin\theta + \\ &\quad + x^2(\sin^2\theta\cos^2\theta - 2\cos^2\theta) + y^2(\sin^2\theta\cos^2\theta - 2\sin^2\theta) \\ &\ldots \\ r^2 &= x^2\sin^2\theta + y^2\cos^2\theta - 2xy\sin\theta\cos\theta \end{aligned} \tag{A.8}$$

L'espressione appena ricavata di r^2, sostituita nella A.2, permette di ricavare il valore dell'integrale in funzione dell'angolo θ, ricordando le proprietà della *funzione caratteristica*, risulta

$$I = \iint_{C'} (x^2 \sin^2\theta + y^2 \cos^2\theta - 2xy \sin\theta\cos\theta) dxdy \tag{A.9}$$

che può essere scritto come

$$I = a \sin^2\theta + c \cos^2\theta - b \sin\theta\cos\theta \tag{A.10}$$

dove

$$a = \iint_{C'} x^2 dxdy \qquad c = \iint_{C'} y^2 dxdy \qquad b = \iint_{C'} 2xy dxdy \tag{A.11}$$

Cambiando opportunamente la forma della A.10 si ottiene

$$I = a + \cos^2\theta(c - a) - b \sin\theta\cos\theta \tag{A.12}$$

Ora non rimane altro che ricavare l'angolo θ che minimizza la A.12, quindi derivando

$$\frac{\partial(I)}{\partial\theta} = -2\cos\theta\sin\theta(c - a) - b\cos^2\theta + b\sin^2\theta \tag{A.13}$$

annullando la A.13 ed utilizzando le seguenti formule trigonometriche

$$\begin{cases} \sin 2\alpha = 2\sin\alpha\cos\alpha \\ \cos 2\alpha = \cos^2\alpha - \sin^2\alpha \end{cases} \tag{A.14}$$

finalmente si ricava

$$\tan 2\theta = \tfrac{b}{a-c} \quad \Rightarrow \quad \theta = \tfrac{1}{2}\arctan\tfrac{b}{a-c} \tag{A.15}$$

formula direttamente utilizzata dal codice C++, per ricavare l'inclinazione di un oggetto. I tre coefficienti a, b, c sono ricavati tramite il calcolo delle rispettive sommatorie al posto degli integrali.

Appendice B

Controllo con raccordo di velocità

Il programma *"Goa1"* è in grado di generare profili di velocità raccordati da tratti lineari, questo permette di non sollecitare in maniera impulsiva l'organo terminale del robot. Di seguito si presentano le relazioni matematiche direttamente implementate nel programma.

Definiamo, in riferimento alla figura B.1, le seguenti grandezze

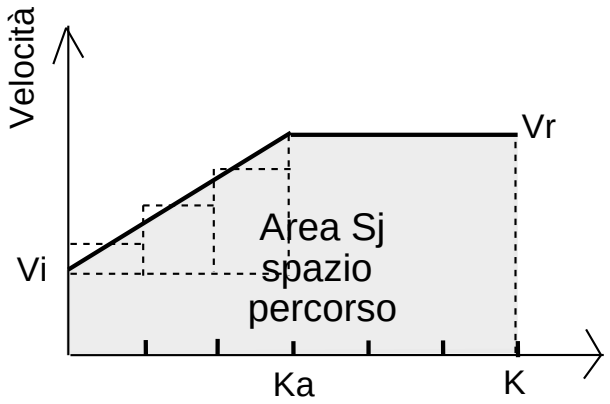

Figura B.1: Schema di un intervallo di controllo.

1. K: il numero di time-slice (ognuna di durata $T_S = 32ms$) di cui é composto un intervallo di controllo, tempo in cui il robot deve compiere lo spostamento S_j.

2. S_j: lo spazio che deve compiere il robot nel tempo determinato dalle K time-slice.

3. K_a: il numero di time-slice dedicate al raccordo lineare tra due istanti j e $j+1$ con velocità rispettive V_j e V_{j+1}.

4. Vr_j: velocità di regime raggiunta nell'istante di controllo j.

5. Vi_j: velocità iniziale coincidente con la velocità tenuta dall'organo terminale del braccio alla fine del precedente intervallo di controllo (per cui $Vi_j \equiv Vr_{j-1}$).

Di seguito, trattando unicamente un solo intervallo j di controllo, non si indicherà più il pedice j snellendo così la notazione.

L'area del profilo di velocità indicato in figura B.1 indica lo spazio che il robot Puma 260 deve percorrere in K time-slice questo é un vincolo che deve essere sempre rispettato.

Lo spazio percorso durante l'intervallo di accelerazione (Time-Slice $< K_a$) deve essere rappresentato attraverso un incremento costate ΔV per time-slice, cioè

$$(Vr - Vi) \cdot \frac{K_a T_S}{2} = \sum_{j=1}^{K_a} j \cdot \Delta V \cdot T_S \tag{B.1}$$

da cui ricaviamo l'incremento che deve essere apportato in fase di accelerazione

$$\Delta V = (Vr - Vi) \cdot \frac{K_a T_S}{2 \cdot \sum_{j=1}^{K_a} j} \tag{B.2}$$

a questo punto basta ricalcolare l'area complessiva (cioè lo spazio totale percorso durante le K time-slice) imponendo che il suo valore sia S

$$Vi \cdot K_a \cdot T_S + \sum_{i=1}^{K_a} i \cdot \Delta V \cdot T_S + Vr(K - K_a) \cdot T_S = S \tag{B.3}$$

dopo qualche semplificazione e raccoglimento é possibile ottenere il valore di regime Vr velocità che il sistema assume dopo i K_a istanti di accelerazione.

$$Vr = \frac{2S - Vi \cdot K_a \cdot T_S}{(2K - K_a) \cdot T_S} \tag{B.4}$$

La B.2 e la B.4 sono direttamente implementate all'interno del programma VAL II denominato *Goa1*[1].

Notare che ponendo $K_a = 0$ si ricava $Vr = S/K$ che coincide con la velocità di regime assunta con l'utilizzo del programma *"Goa"*[2].

Di seguito vengono presentati i profili di velocità e posizione del Puma 260 al variare del numero di intervalli K_a di accelerazione (decelerazione). I dati da cui sono stati ricavati i grafici sono $K = 10$ e $S = 40mm$ con una velocità iniziale di $Vi = -15\frac{mm}{T_S}$ e K_a che varia da 1 a 9.

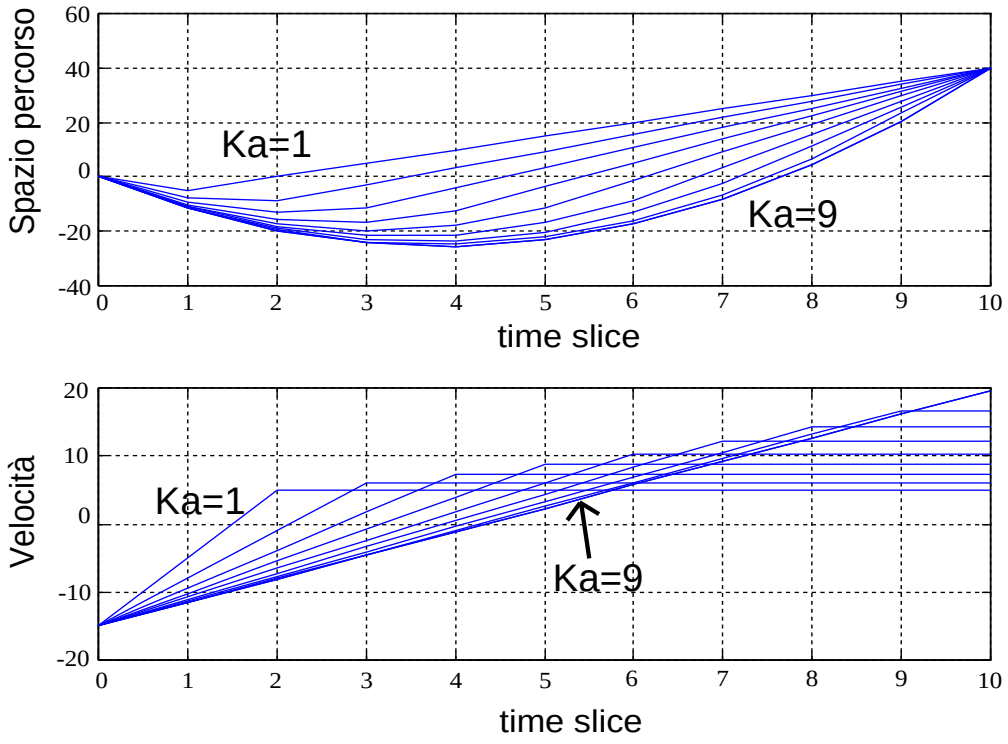

Figura B.2: Spazio percorso con il controllo del profilo di velocità.

Per piccoli valori di K_a il sistema risponde in maniera pronta ma impulsiva invertendo immediatamente la direzione di movimento invece all'aumentare di K_a l'inversione del movimento arriva in modo graduale, questo crea una sovraelongazione sempre più pronunciata nello spazio percorso.

Notare che per K_a elevati dovendo compiere in K time-slice lo stesso spazio

[1] In realtà l'implementazione software, tenendo conto che T_S é un tempo costante, è basata su variazioni dello spazio invece che sulle velocità.

[2] Vedere l'appendice D.

S la velocità finale sarà più elevata (scotto da pagare per avere accelerazioni meno pronunciate).

Appendice C

Installazione del software

L'ambiente di sviluppo utilizzato per la scrittura dei codici è stato **Microsoft VISUAL C 6.0**.

Di seguito viene riportata la lista completa dei file[1] che costituiscono il corpo dei progetti di entrambe le applicazioni sviluppate. Eseguire l'applicazione *"Pick and place"* oppure l'applicazione *"Eye in hand"* dipende solo dall'inclusione nel progetto rispettivamente dei file `OPENLOOP.CPP` oppure `EYEHAND.CPP`, sono appunto questi file che contengono la procedura `main()` del progetto.

```
   nomefile                    commento
-------------------------------------------------------------
BUFFER   CPP    Lib. dell'oggetto Buffer comandi per lib. R1
BUFFER   H
CONTROL  CPP    libreria PID, preditori e calibrazione.
CONTROL  H
IMEL     CPP    libreria di visione artificiale.
IMEL     H
MATRICE  CPP    Libreria dell'oggetto matrice.
MATRICE  H
MIL      LIB    Libreria frame grabber Matrox
MILMET2  LIB    Libreria frame grabber Matrox
MY_COM   CPP    Procedure di com. seriale per oggetto Robot.
MY_COM   H
ROB_CTRL CPP    Lib. R1 di implementazione oggetto Robot
ROB_CTRL H
```

[1] Per i codici completi vedere le varie appendici.

```
ROBOT    CPP    Lib. R2 di intefaccia oggetto Robot.
ROBOT    H
VECTOR   H      Implementazione Oggetto Vettore per lib. R1
OPENLOOP CPP    Main applicazione ad anello aperto.
EYEHAND  CPP    Main applicazione ad anello chiuso.
```

Definire i file `mil.lib` e `milmet2.lib` come librerie di default del progetto nel seguente modo: selezionare l'opzione "Settings" del progetto, scorrere fino alla cartella "Link" ed aggiungere in "Project Option"

```
/defaultlib:mil.lib

/defaultlib:milmet2.lib
```

Appendice D

Codici VAL II

Di seguito vengono riportati i codici dei programmi eseguiti dal controllore UNIVAL nelle due applicazioni sviluppate.

D.1 Applicazione *"Pick and Place"*

File nominato: *Task_95.v2*
Processo di nome:*pick*

```
.PROGRAM pick()
        ; Programma VAL II dell'applicazione pick and place.
        exit = 0      ; variabile di terminazione processo.
        attesa = 0
        distance = 30
        rot_delta = 0 ; angolo di rotazione dell'oggetto
        WHILE exit == 0 DO
            MOVES a
            BREAK
            SET temp = HERE
            PROMPT "insert position:", px, py, pz, rot_obj, rot_delta
            IF ((px == 0) OR (py == 0) OR (pz == 0)) THEN
                exit = 1
            END
            IF exit == 0 THEN
                roto = RO(HERE)
                rota = RA(HERE)
```

```
 18                rott = RT(HERE)
 19                SET place = TRANS(px, py, pz, roto, rota, rott)
 20                SET rotation = TRANS(0, 0, 0, 90, -90, rot_obj)
 21                APPROS place:rotation, distance
 22                MOVES place:rotation
 23                BREAK
 24                MOVES place:rotation:TRANS(0, 0, 0, 90, -90, rot_delta)
 25                BREAK
 26                PROMPT "attesa:", attesa
 27                IF attesa <> 0 THEN
 28                    exit = 1
 29                END
 30            END
 31            DEPARTS distance
 32        END
 33 .END
```

```
      Locazione di partenza del Puma ad ogni ciclo di lavoro.
*****************************************************************

          X/Jt1     Y/Jt2     Z/Jt3     O/Jt4     A/Jt5     T/Jt6
 a       187.197   121.570    93.395   121.722    88.564   118.660
```

D.2 Applicazione *"Eye in Hand"*

Vengono di seguito riportati il *robot control program* di nome *"Goa"* ed il *process control program* operante in modalità alter mode nelle due versioni rispettivamente senza e con il raccordo continuo di velocità.
File Nominato:*o_col_5.v2*
Processi di nome :*Goa* e *Path*.

```
 1 Program goa();
 2         dx = 0
 3         dy = 0
 4         dz = 0
 5         cn = 0
 6         exit = 0
```

```
 7          PCEND 3
 8          MOVE a
 9          BREAK
10          PROMPT "Cal1:", tmp
11          MOVE a:TRANS(0, mv, 0, 90, -90, 0)
12          BREAK
13          PROMPT "Cal2:", tmp
14          MOVE a:TRANS(0, 0, mv, 90, -90, 0)
15          BREAK
16          PROMPT "Cal3:", tmp
17          MOVE a
18          BREAK
19          PCEXECUTE path (), -1, 1, 3
20          WAIT
21          ALTER (-1, 17)
22          WHILE exit == 0 DO
23              MOVES a
24              PROMPT ".", px, py, pz
25              IF ((ABS(px) > mx) OR (ABS(py) > my) OR (ABS(pz) > mz)) THEN
26                  NOALTER
27                  PCABORT 3
28                  exit = 1
29                  END
30              dx = px/k
31              dy = py/k
32              dz = pz/k
33              cn = 0
34              END
35          NOALTER
36          PCABORT 3
```

```
 1 .PROGRAM path()
 2          IF cn < k THEN
 3              ALTOUT 0, dx, dy, dz, 0, 0, 0
 4          ELSE
 5              ALTOUT 0, 0, 0, 0, 0, 0, 0
 6          END
```

```
   7         cn = cn+1
   8         WAIT
   9 .END

     Variabili di ambiente non definite dai programmi
*****************************************************************
 k                10  ; Numero di Time slice per effettuare lo
                      ;   lo spostamento inviato da seriale.
 ka                2  ; Istanti di accelerazione (usato solo
                      ;   nel robot control program goa1)
 mx               50  ; limiti di sicurezza sulle velocita'
 my               50
 mz               10
 mv               30  ; ampiezza movimento per calibrazione EyeInHand

      Locazione di partenza delle applicazioni eye in hand.
*****************************************************************
       X/Jt1     Y/Jt2     Z/Jt3     O/Jt4     A/Jt5     T/Jt6
 a    169.720   221.581   144.422  -177.965   -2.027   179.867
```

D.3 Raccordo continuo di velocità

Le locazioni e le variabili di ambiente precedentemente definite vengono utilizzate anche dai programmi *Goa1* e *Path* presenti nel file *Accel_5.v2*.

```
   1 .PROGRAM goa1()
   2         dx = 0
   3         dy = 0
   4         dz = 0
   5         dax = 0
   6         day = 0
   7         daz = 0
   8         holdx = 0
   9         holdy = 0
  10         holdz = 0
  11         srx = 0
```

```
12          sry = 0
13          srz = 0
14          incx = 0
15          incy = 0
16          incz = 0
17          cn = 0
18          exit = 0 ; variabile di terminazione processo.
19          sum = 0
20          FOR c = 1 TO ka
21              sum = sum+c
22          END
23          PCEND 3
24          MOVE a
25          BREAK
26          PROMPT "Cal1:", tmp
27          MOVE a:TRANS(0, mv, 0, 90, -90, 0)
28          BREAK
29          PROMPT "Cal2:", tmp
30          MOVE a:TRANS(0, 0, mv, 90, -90, 0)
31          BREAK
32          PROMPT "Cal3:", tmp
33          MOVE a
34          BREAK
35          PCEXECUTE path (), -1, 1, 3
36          WAIT
37          ALTER (-1, 17)
38          WHILE exit == 0 DO
39              MOVES a
40              PROMPT ".", px, py, pz
41              IF ((ABS(px) > mx) OR (ABS(py) > my) OR (ABS(pz) > mz)) THEN
42                  NOALTER
43                  PCABORT 3
44                  exit = 1
45              END
46              holdx = srx
47              holdy = sry
48              holdz = srz
```

```
49              srx = (2*px-holdx*ka)/(2*k-ka)
50              sry = (2*py-holdy*ka)/(2*k-ka)
51              srz = (2*pz-holdz*ka)/(2*k-ka)
52              dax = (srx-holdx)*(ka/(2*sum))
53              day = (sry-holdy)*(ka/(2*sum))
54              daz = (srz-holdz)*(ka/(2*sum))
55              incx = holdx ;
56              incy = holdy ;
57              incz = holdz ;
58              dx = px/k
59              dy = py/k
60              dz = pz/k
61              cn = 0
62          END
63          NOALTER
64          PCABORT 3
65  .END
```

```
 1  .PROGRAM path()
 2          IF cn < ka THEN
 3              incx = incx+dax ;
 4              incy = incy+day ;
 5              incz = incz+daz ;
 6              ALTOUT 0, incx, incy, incz, 0, 0, 0
 7          ELSE
 8              IF cn < k THEN
 9                  ALTOUT 0, srx, sry, srz, 0, 0, 0
10              ELSE
11                  ALTOUT 0, 0, 0, 0, 0, 0, 0
12              END
13          END
14          cn = cn+1
15          WAIT
16  .END
```

Appendice E

Libreria IMEL

E.1 File IMEL.h

```
#ifndef IMEL_VER_001
#define IMEL_VER_001       // versione libreria....

#define PIXEL            int             // tipo di dato coordinate pixel.
#define TIPO_PIXEL        unsigned char   // tipo di dato rappresentazione pixel.
#define CALCOLO            double            // tipo di dato utilizzato per i calcoli.


/* ------------------------------------------------------------------------*/
/* ***********   DEFINIZIONE possibili ERRORI della CLASSE IMEL    ********** */
/* ------------------------------------------------------------------------*/

#define IMEL_NOT_INIT                -1       // classe non inizializzata
#define IMEL_LABEL_OUT_OF_RANGE_X    -2       // confini del Child fuori dal Parent
#define IMEL_LABEL_OUT_OF_RANGE_Y   -3
#define EXTERN_IMAGE_NOT_VALID      -4     // puntatore immagine passata=NULL
#define IMEL_OUT_OF_DIM_MAX         -5     // immagine Parent troppo grande.
#define IMEL_NOT_LABEL              -6     // immagine non ancora etichettata.
#define IMEL_ERROR_MEMORY           -7     // errore allocazione momoria.


// ------------------------------------------------------------------------
// **********  Definizione dei modi di utilizzo funzione ImelObjectInfo() *******
// ------------------------------------------------------------------------

#define IMEL_MODE_NEXT   -1   // ritorna l'elem. corrente e si sposta al prossimo.
#define IMEL_MODE_START  -2      // si sposta al primo elemento lista oggetti.
#define IMEL_MODE_END    -3      // si sposta all'ultimo elemento lista oggetti.


#define MAX_DIM_IMAGE  (764*577)    // massima lunghezza vettore immagine
#define PIXEL_ON   0              // valore indicante un pixel acceso
```

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include <math.h>


// ------------------------------------------------------------
// Struttura Dati di una singola figura
// ------------------------------------------------------------
struct FIGURE
{
    int label;               // etichetta (numero) identificativo della figura.
    PIXEL north;            // ordinata pixel piu' alto
    PIXEL south;            // ordinata pixel pi' basso
    PIXEL east;                // ascissa pixel piu' a destra
    PIXEL west;                // ascissa pixel piu' a sinistra

    CALCOLO area;            // area della figura
    CALCOLO cgx;            // ascissa centro geometrico
    CALCOLO cgy;            // ordinata centro geometrico

    CALCOLO a;                // momenti del 2 ordine per il calcolo
    CALCOLO b;                // dell'asse di rotazione....
    CALCOLO c;

    CALCOLO teta;            // pendenza asse di rotazione.

    struct FIGURE *next;    // puntatore al prossimo elemento della lista figure.
};



/*  ****************************************************************
    Dichiarazione della Classe "Image ELaboration".
    Estrae dall'immagine "image" passata attraverso ImelInit,
    una lista di strutture dati di tipo FIGURE.
    Tale lista  accessibile attraverso ImelObjectInfo() ed ogni suo
    elemento contiene tutti i dati di un oggetto rilevato in "image".
    ****************************************************************/

class Imel
{
public:

    int ImelInit(TIPO_PIXEL* image,    // Inizializza l'oggetto IMEL
                         int dim_x,
                         int dim_y);


    // ------------------------------------------------------------------
    // Etichetta, calcola gli estremi e il centro di massa degli oggetti
    // riconosciuti nella immagine figlio (child) di "image", con inizio alle
```

```
    // coordinate ofx ofy di larghezza size_x e sizey con definizione
    // defx e defy (quando defx=1 e defy=1 ho definizione massima!!!)
    // Ritorna poi il numero di oggetti trovati.
    // ------------------------------------------------------------------
    int ImelLabeling(TIPO_PIXEL* image_ext,
                     int size_x,        // Applica l'algoritmo di labeling...
                     int size_y,      // Dimensione  x,y immagine child
                     int ofx,        // Offset in x da cui parte l'image chile
                     int ofy,        // Offset in y da cui parte l'image child
                     int defx,       // Definizione etichettamento in x
                     int defy);        // Definizione etichettamento in y

    bool ImelFree();    // rilascia tutte le risorse occupate dall'oggetto.

    FIGURE *ImelObjectInfo(int modo); // ritorna un stuttura dati di un oggetto.

    int ImelOrientation(FIGURE*target);  // calcola l'orientamento di *target.

    int ImelFilter(int thresold); // filtra gli oggetti di area < thresold.

    int Tracking(int Kp,int Ka,int Kt); // funzione di aggancio oggetto.
};

#endif
```

E.2 File IMEL.c

```
#include "Imel.h"

#define INFINITY   999999999999999

int  ImelCount();            // Conteggio elementi etichettati
void ImelFreeTarget();        // Dealloca la lista degli oggetti etichettati
void ImelCenterTarget();    // Calcolo dei campi della struttura FIGURE.

// ------------------------------------------------------------------
// ***** COMMENTO SELLE PRESTAZIONI
// Variabili globali a livello di modulo.
// Non implemento le variabili globali come Public della classe
// perch mi portano ad una considerevole diminuzione delle velocit
// (anche del 30%) questo per mi impedisce di poter creare pi
// istanze della classe IMEL.
// ------------------------------------------------------------------

FIGURE  *first,         // testa della lista di oggetti trovati
        *last,            // coda della lista
        *current;        // "Cursore" di lettura tramite IMELOBJECTINFO.

int* ImelLabel;            // puntatore all'immagine etichette
TIPO_PIXEL* ImelImage;  // puntatore all'immagine esterna al modulo
```

```
int number_target;      // numero degli oggetti rilevati
int ImelParentx;        // dimensioni immagine padre lungo x e y
int ImelParenty;
int ImelImageSize;      // dimensione pixel immagine

int defx;                // Passo di campionamento in x e y
int defy;                // ..quando defx=defy=1 ho la definizione massima
int ofx;                // Offset x,y immagine child campionata dentro
int ofy;                // ....l'immagine Parent.
int size_x;               // Dimensioni x,y immagine figlio (child)
int size_y;
                        // ****** variabili di ottimizzazione *********
int passoy=0;            // Indica l'inizio della riga corrente
int last_x,last_y;        // indicano l'ultimo pixel in basso a destra
int limx;
int limy;
int riga;                // Largezza riga immagine (coincide con imelparentx)
int valx;                // Variabili temporanee...
int valy;                // ...atte unicamente ad evitare di calcolare...
int valxx;                // ...dei prodotti inutilmente.
int valyy;
int valtt;

bool switch_init=false;      // indica se l'oggetto  stato inizializzato.
bool switch_label=false;     // indica se l'oggetto  stato etichettato.

// ************************************************************************
// *******  COMMENTO SULLE PRESTAZIONI
// In questo modulo lo stile procedurale  abbandonato per motivi
// di efficienza, per cui avremo meno procedure possibili e quelle
// che verranno implementate saranno lunghe e meno leggibili. Questo
// permettara' per di guadagnare anche un 15% in termini di velocit.
//
// ************************************************************************

// -------------------------------------------------------------
// Alloca tutta la memoria necessaria all'utilizzo delle funzioni
// ed inizializza l'oggetto.
// -------------------------------------------------------------
int Imel::ImelInit(TIPO_PIXEL* image_ext,int dim_x, int dim_y)
{
    if (image_ext==NULL) return EXTERN_IMAGE_NOT_VALID;
    if (dim_x*dim_y>MAX_DIM_IMAGE) return IMEL_OUT_OF_DIM_MAX;
    ImelImage=image_ext;

    first=NULL;
    last=NULL;
    current=NULL;
    ImelImageSize=dim_x*dim_y;
    ImelParentx=dim_x;
    ImelParenty=dim_y;
    number_target=0;
    ImelLabel=(int*)malloc(ImelImageSize*sizeof(int));
```

```
    if (ImelLabel==NULL) return IMEL_ERROR_MEMORY;
    switch_init=true;
    return true;
}

// ------------------------------------------------------------
// Metodo che rilascia  tutte le risorse acquisite.
// ------------------------------------------------------------
bool Imel::ImelFree()
{
    if (first!=NULL) ImelFreeTarget();
    free(ImelLabel);
    switch_init=false;   // resetto gli interruttori di
    printf("Imel: Risorse liberate.\n");
    return(true);
}

// ------------------------------------------------------------------
// rilascio la memoria occupata dalla struttura dati delle figure
// ------------------------------------------------------------------
void ImelFreeTarget()
{
    FIGURE *slider;
    FIGURE *temp;
    slider=first;
    last=first;      // azzero tutti i riferimenti della lista...
    current=first;
    while (slider!=NULL)
    {
        temp=slider->next;
        free(slider);
        slider=temp;
    }
    first=NULL;
    last=NULL;
    current=NULL;
    switch_label=false;     // ....utilizzo delle funzioni.
}

// ------------------------------------------------------------
// Funzione che etichetta l'immagine child dentro l'immagine Parent
// e ritorna il numero di oggetti trovati.
// ------------------------------------------------------------
int Imel::ImelLabeling(TIPO_PIXEL* image_ext,
                int child_size_x,    // dimensione  x immagine child
                int child_size_y,  // dimensione  y immagine child
                int child_ofx,        // offset in x da cui parte l'image chile
                int child_ofy,        // offset in y da cui parte l'image child
                int child_defx,    // definizione etichettamento in x
                int child_defy)    // definizione etichettamento in y
{

    if (image_ext==NULL) return EXTERN_IMAGE_NOT_VALID;
```

```
    if (first!=NULL) ImelFreeTarget();  // svuoto una eventuale elabor. precedente.
    if (!switch_init) return IMEL_NOT_INIT;
    if (child_size_x+child_ofx>ImelParentx) return IMEL_LABEL_OUT_OF_RANGE_X;
    if (child_size_y+child_ofy>ImelParenty) return IMEL_LABEL_OUT_OF_RANGE_Y;
    ImelImage=image_ext;
    size_x=child_size_x-1;
    size_y=child_size_y-1;
    ofx=child_ofx;
    ofy=child_ofy;
    defx=child_defx;
    defy=child_defy;

    int i,j;
    int min;            // contiene il valore minimo tra i 2 pixel confinanti
    bool change;        // switch di rilevazione cambiamento in immagine
    int n_label=100;    // numero prima etichetta

    riga=ImelParentx;
    limx=ofx+size_x-1;  // limiti x,y dell'immagine
    limy=ofy+size_y-1;

    for(i=ofy;i<=limy;i+=defy)
    {
        passoy=i*ImelParentx;
        for(j=ofx;j<=limx;j+=defx)
            if (*(ImelImage+passoy+j) == PIXEL_ON)
            {
//            printf("0");
              *(ImelLabel+passoy+j)=n_label;
              n_label++;

            }
            else
            {
                *(ImelLabel+passoy+j)=0;
//                printf("-");
            }
//        printf("\n");
    }

do
{
    change=false;
    for(i=ofy;i<=limy;i+=defy)
    {
        passoy=i*ImelParentx;
        valtt=i*riga;
        valxx=valtt-defx;
        valyy=valtt-riga*defy;

        for(j=ofx;j<=limx;j+=defx)
            if (*(ImelLabel+passoy+j)!=0)
            {
```

```
                // Inizio calcolo della connettivit.....
                // 2-neightbors  top-down
                // ---------------------------------------------
                    valx=j+valxx;
                    valy=j+valyy;
                    min =*(ImelLabel+j+valtt);
                    if (i>defy-1) if((min>(*(ImelLabel+valy))) &&
                          (*(ImelLabel+valy)!=0)) min=*(ImelLabel+valy);

                    if (j>defx-1) if((min>(*(ImelLabel+valx))) &&
                          (*(ImelLabel+valx)!=0)) min=*(ImelLabel+valx);

              if (min!= *(ImelLabel+passoy+j)) change=true;
              *(ImelLabel+passoy+j)=min;
                // ---------------------------------------------
            }
    }
    last_x=j-defx;
    last_y=i-defy;

    for(i=last_y;i>=ofy;i-=defy)
    {
        passoy=i*ImelParentx;
        valtt=i*riga;
        valxx=valtt+defx;
        valyy=valtt+riga*defy;

        for(j=last_x;j>=ofx;j-=defx)
            if (*(ImelLabel+passoy+j)!=0)
            {
                // calcolo della connettivit di tipo 2
                // 2-neghtbors Down Top
                // ---------------------------------------------
                    valx=j+valxx;
                    valy=j+valyy;

                    min =*(ImelLabel+j+valtt);
                    if (i<(ofy+size_y-defy)) if((min>(*(ImelLabel+valy))) &&
                        (*(ImelLabel+valy)!=0)) min=*(ImelLabel+valy);

                    if (j<(ofx+size_x-defx)) if((min>(*(ImelLabel+valx))) &&
                        (*(ImelLabel+valx)!=0)) min=*(ImelLabel+valx);

              if (min!= *(ImelLabel+passoy+j)) change=true;
              *(ImelLabel+passoy+j)=min;
               // ---------------------------------------------
            }
    }

} while (change);// fine DO


number_target=ImelCount();
```

```
switch_label=true;
return number_target;
}

// ------------------------------------------------------------------
// Conto le etichette rimaste,che corrispondono al numero di oggetti
// rilevati, associo poi ad ogni oggetto una struttura dati FIGURE.
// ------------------------------------------------------------------
int ImelCount()
{
    int pezzi=0;
    int attuale=0;
    int i,j;
    int passoy;
    int co=1;
//    int limx;
//    int limy;
    FIGURE *slider;
    FIGURE *nuovo;

    limx=ofx+size_x-1;
    limy=ofy+size_y-1;
    for(i=ofy;i<=limy;i+=defy)
    {

        passoy=i*ImelParentx;

        for(j=ofx;j<=limx;j+=defx)
        {
            if (attuale<*(ImelLabel+passoy+j))
            {
                attuale=*(ImelLabel+passoy+j);
              // ---------------------------------------------
              // aggiungo un ogggetto alla lista...
                nuovo=(FIGURE*)malloc(sizeof(FIGURE));
                nuovo->label=attuale;
                nuovo->next=NULL;
                nuovo->area=1;
                nuovo->cgx=j;
                nuovo->cgy=i;
                nuovo->north=i;     // inizializzo gli estremi...
                nuovo->south=i;
                nuovo->east=j;
                nuovo->west=j;
                nuovo->a=0;            // inizializzo i parametri dell'integrale
                nuovo->b=0;            // per il calcolo dell'asse di inerzia.
                nuovo->c=0;
                nuovo->teta=0;

                if (first==NULL)
                {
                    first=nuovo;
                    last=nuovo;
```

```
                    current=nuovo;    // pongo il cursore in testa alla lista
                }
                else        // aggiungo in coda l'ultimo elemento.
                {
                last->next=nuovo;
                last=nuovo;
                }
              // ---------------------------------------------
              pezzi++;
            }
            else
                if (*(ImelLabel+passoy+j)>0)
                {
                    // ----------------------------------------
                    // Aggiorno i dati di un oggetto della lista
                        slider=first;
                        while(*(ImelLabel+passoy+j)!=slider->label)
                                                        slider=slider->next;
                        slider->area+=1;
                        slider->cgx+=j;
                        slider->cgy+=i;
                        if (i<slider->north) slider->north=i;
                        if (i>slider->south) slider->south=i;
                        if (j<slider->west) slider->west=j;
                        if (j>slider->east) slider->east=j;

                    // ----------------------------------------
                }
        }
    }
    ImelCenterTarget();
    return pezzi;
}

// ------------------------------------------------------------
// Calcolo il centro e l'area di ogni oggetto appartenente alla
// lista di oggetti etichettati.
// ------------------------------------------------------------
void ImelCenterTarget()
{
    FIGURE *slider;
    slider=first;
    while (slider!=NULL)
    {
        slider->cgx=slider->cgx/(slider->area);
        slider->cgy=slider->cgy/(slider->area);

        // stimo l'area in funzione della definizione di campionamento
        slider->area=slider->area*defx*defy;

        // aggiorno i confini al rettangolo che li contiene
        // necessario per quando si utilizza la stima (defx>1 e defy>1)
        if (slider->north>defy) slider->north-=defy;
```

```
                        else slider->north=0;
        if (slider->south<limy-defy) slider->south+=defy;
                        else slider->south=limy;
        if (slider->east<limx-defx) slider->east+=defx;
                        else slider->east=limx;
        if (slider->west>defx) slider->west-=defx;
                        else slider->west=0;
        slider=slider->next;
    }
}

// ------------------------------------------------------------
// Calcolo parametri a,b c momento di inerzia del 2 ordine
// e del'angolo teta di inclinazione dell'oggetto.
// ------------------------------------------------------------
int Imel::ImelOrientation(FIGURE *target)
{
    if (!switch_label) return IMEL_NOT_INIT;

    int i,j;
    CALCOLO rx;     // componente lungo gli assi xy della distanza tra il punto
    CALCOLO ry;     // della figura e il centro di massa.

    riga=ImelParentx;
    limx=ofx+size_x-defx;
    limy=ofy+size_y-defy;

    for(i=ofy;i<=limy;i+=defy)
    {
        passoy=i*ImelParentx;
        for(j=ofx;j<=limx;j+=defx)
            if (target->label ==*(ImelLabel+passoy+j))
            {
                rx=j-target->cgx;
                ry=i-target->cgy;
                target->a+=rx*rx;
                target->b+=2*rx*ry;
                target->c+=ry*ry;
            }
    }

    if ((target->a==target->c) && (target->b==0)) target->teta=-1;
    else
        if ((target->a==target->c) && (target->b>0))
            target->teta=atan(INFINITY)/2;
        else
            if ((target->a==target->c) && (target->b<0))
                target->teta=-atan(INFINITY)/2;
            else
                target->teta=(atan2(target->b,(target->a-target->c)))/2;
    return(1);
}
```

```
//-------------------------------------------------------------
// ritorna il puntatore ad una struttura dati FIGURE contenente i
// dati relativi all'oggetto specificato dal parametro 'modo'
// -------------------------------------------------------------
FIGURE * Imel::ImelObjectInfo(int modo)
{
    if (!switch_init) return NULL;    // imel not init
    if (!switch_label) return NULL;      // imel not label

    int cont=0;
    FIGURE *prec;
    prec=current;
    if (modo==IMEL_MODE_NEXT)
    {
        if (current!=NULL)    current=current->next;
        else return NULL;

    }
    if (modo==IMEL_MODE_START)  // si mette solamente in testa alla lista
    {
        current=first;
        return(current);
    }
    if (modo==IMEL_MODE_END)  // si mette solamente in coda alla lista
    {
        current=last;
        return(current);
    }
    if (modo>0) {
                // l'utente ha specificato il numero preciso dell'elemento.
                current=first;
                while( (current!=NULL) && cont<modo)
                {
                    prec=current;
                    current=current->next;
                    cont++;
                }
                if (prec==NULL) return NULL;
               }
    return prec;
}

// ---------------------------------------------------------------
// Elimina dalla lista di strutture FIGURE, quegli oggetti che hanno
// area inferiore a thresold
// ---------------------------------------------------------------
int Imel::ImelFilter(int thresold)
{
    FIGURE *slider;
    FIGURE *temp;
    FIGURE *prec;
    slider=first;
    prec=first;
```

```
    if (!switch_label) return IMEL_NOT_LABEL;
    if (!switch_init) return IMEL_NOT_INIT;

    while (slider !=NULL)
    {
        if (slider->area<thresold)
        {
            number_target-=1;
            temp=slider;
            if (slider==last) last=prec;
            if (slider==first)
            {
                prec=first->next;
                first=first->next;
            }
            else  prec->next=slider->next;

            slider=slider->next;
            free(temp);
        }
        else
        {
            prec=slider;
            slider=slider->next;
        }
    }
    current=first;  // pongo il cursore in prima posizione
    return(number_target);
}

// ------------------------------------------------------------------
// procedura di inseguimento oggetto
// ritorna il numero dell'oggetto della lista oggetti
// che minimizza la funzione costo, se ritorna 0 non ci sono oggetti!!
// ------------------------------------------------------------------
int Imel::Tracking(int Kp,int Ka,int Kt)
{
    double CostoMinimo=1000000;
    double Ca=0;            // costi parziali di Area,inclinazione(theta)
    double Ct=0;            // e posizione
    double Cp=0;
    double costo=100000;
    static CALCOLO Ta=0;  // dati dell'oggetto da seguire
    static CALCOLO Tt=0;
    static CALCOLO Tx=0;
    static CALCOLO Ty=0;
    int contobj=0;        // contatore oggetti
    int goal=0;           // indica il numero dell'oggetto indicante il target
    FIGURE *t;

    ImelObjectInfo(IMEL_MODE_START);
    //printf("\n\nPosizione obbiettivo %.2f %.2f\n",Tx,Ty);
    do
```

```
        {
            t=ImelObjectInfo(IMEL_MODE_NEXT);
            if (t!=NULL)
            {
                contobj++;
                if (Kt!=0)
                {
                    ImelOrientation(t);
                    Ct=fabs(t->teta-Tt);
                }

                Ca=fabs(t->area-Ta);
                Cp=sqrt(pow((t->cgx-Tx),2)+pow((t->cgy-Ty),2));
                //printf("ogg %i Loc %u Pos=%.2f\n",contobj,t,Cp);
                costo=Kp*Cp+Kt*Ct+Ka*Ca;
                if (costo<CostoMinimo)
                {
                    //printf("Target ogg=%i Costo %.2f\n",contobj,costo);
                    goal=contobj;
                    CostoMinimo=costo;
                }
            }
        }while(t!=NULL);
    t=ImelObjectInfo(goal);
    //printf("Minimo ogg %i loc =%u \n",goal,t);
    if(t!=NULL)
    {
        Ta=t->area;
        Tt=t->teta;
        Tx=t->cgx;
        Ty=t->cgy;
    }
    ImelObjectInfo(IMEL_MODE_START);
    return goal;
}
```

Appendice F

Codici Applicazioni

F.1 Codici di controllo

Di seguito si riportano i *codici di controllo* di entrambe le applicazioni sviluppate (vedere il capitolo 5) insieme a tutte le librerie utilizzate.

F.1.1 File Eyehand.c

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <conio.h>
#include <mil.h>
#include "matrice.h"            // header libreria matrici
#include "imel.h"                // header libreria di labeling
// ---------------------------------------------------------------
#include "Robot.h"                // header libreria robot
bool public_ender=false;        // variabile globale di terminazione thread
ROBOT Puma;                        // oggetto di controllo robot
// ---------------------------------------------------------------
#include "rob_ctrl.h"
#include "control.h"            // header libreria delle funzioni di controllo.

#define pi         3.141592654
#define FILE_DCF   "Cvm10an3.dcf"    // file di settaggio telecamera
#define SCALA          1                // scala
#define IMAGE_X_DIM   (763 /SCALA)     // dimensioni buffer Image allocati dalla Mil
#define IMAGE_Y_DIM   (576 /SCALA)
#define THRESOLD      200             // Soglia Look_up Table della MIL

void MatroxInit();      // inizializza il frame-grabber
void MatroxShutdown();  // rilascia le risorse allocate per il frame-grabber
void setup();            // settaggio buffer della console
void processamento();    // routine di processamento immagine
```

```
void clean();
bool Calibrazione(Matrix* B);

MIL_ID MilApplication,     /* Application identifier.  */
          MilSystem,       /* System identifier.       */
          MilDisplay,      /* Display identifier.      */
          MilDigitizer,    /* Digitizer identifier.    */
          MilImage,        /* Image buffer identifier. */
          Look_Table;      /* identificatore look table */

// ----------------------------------------------------------------------
HANDLE StdOut;
HANDLE Console;
COORD cur;
// ----------------------------------------------------------------------
TIPO_PIXEL *buffer;            // puntatore al buffer immagine
Imel image;                 // oggetto IMmagine ELaborata
FIGURE *t;                    // puntatore a risultato etichettamento
// ----------------------------------------------------------------------

int scala;             // indica la scala di acquisizione del frame-grabber
int image_size_x;     // dimensioni x,y immagine etichettata
int image_size_y;

void main(void)
{
    setup();
    StdOut=GetStdHandle(STD_OUTPUT_HANDLE);
    MatroxInit();
    processamento();
    printf("Premi un tasto...\n");
    while (!kbhit()) ;
    printf("Concludo.");
    MatroxShutdown();
}

// ********************************************************************
// --------------------------------------------------------------------
// Programma principale
// --------------------------------------------------------------------
// ********************************************************************
void processamento()
{
    int pitch;
    int car=0;            // carattere di comando ricevuto da tastiera

    // le dimensioni a cui riferirsi sono quelle del DIGITIZER
    int dimx,dimy;

    MdigInquire(MilDigitizer,M_SOURCE_SIZE_X,&dimx);
    MdigInquire(MilDigitizer,M_SOURCE_SIZE_Y,&dimy);

    // Ricavo il puntatore al buffer dell'immagine MILIMAGE
```

```
// ricavo poi le dimensioni x in pitch del buffer
buffer=(unsigned char*)MbufInquire(MilImage,M_HOST_ADDRESS,M_NULL);
pitch =(int)MbufInquire(MilImage,M_PITCH,M_NULL);

image_size_x=pitch;
image_size_y=dimy/scala;

MdigControl(MilDigitizer,M_GRAB_SCALE,(double)1/scala);
image.ImelInit(buffer,pitch,dimy);
printf("Dimensione x :%i \n",dimx);
printf("Dimensione y :%i \n",dimy);
printf("Fattore di riduzione :%i \n",scala);
printf("Dim ridotta x :%i \n",image_size_x);
printf("Dim ridotta y :%i \n",image_size_y);
printf("Init Camera Ok.\nPremi un tasto.....\n");

MdigGrabContinuous(MilDigitizer, MilImage);


getch();
MdigLut(MilDigitizer,Look_Table);

int row=0;
int col=0;
int obj;
int cont=0;
int c1=0;
char str_command[80];
bool target=false;
int ris;
int ve=0;     // numero vettore


printf("Inizializzo Puma \n");

Puma.Initialize();

Puma.Asincronous();
Puma.SendCommand("pcend");
Puma.Prompt(PROMPT_DOT);
Puma.SendCommand("pcend");
Puma.Prompt(PROMPT_DOT);
Puma.SendCommand("exec goa");

printf("\nPremi un tasto per iniziare la calibrazione...\n");
getch();
Puma.Sincronous();

// matrice cambiamento di base
// possibile ricavarla con una calibrazione su un oggetto statico.
// con K prog puma=10  e base>0.13....
// k=0.085 e Pcr=1.5 sec calcolato a manazza
```

```
// indici manuali utilizzati per ricavare un tuning del PID_MIO
double Pcr=2.4;            //   periodo critico
double Kcr=0.08 ;          //   guadagno critico

TIPO Base[4]={.13,0,0,-.12};   // passaggio di coordinate (calibrazione)
                               // i valori messi sono plausibili....
                               // ma e' meglio calibrare....
TIPO Identity[9]={1,0,0,0,1,0,0,0,1};   // valori matrice identita'
TIPO Prop[4]={0.6*Kcr,0,0,0.6*Kcr};
TIPO Integ[4]={0.5*Pcr,0,0,0.5*Pcr};
TIPO Deriv[4]={2*Pcr,0,0,2*Pcr};
TIPO Centro[2]={image_size_x/2,
                image_size_y/2};     // coordinate centro

TIPO end_effector[9]={0,0,1,1,0,0,0,1,0};

Matrix C(2,1);   // Posizione di riferimento nello schermo
Matrix P(2,1);   // Vettore posizione target
Matrix E(2,1);   // vettore errore posizione target in pixel
Matrix Er(2,1);  // vettore errore nello spazio robot al tempo del controllo.
Matrix Esr(2,1); // errore nello spazio del robot (in millimetri)
Matrix Mov(2,1); // stima della posizione dell'oggetto.

Matrix Outs(2,1);  // Stima uscita errore sistema puma.
Matrix Out1s(2,1); // Stima uscita errore sistema puma.

Matrix Pos(2,1);   // stima posizione oggetto.
Matrix Vel(2,1);   // Stima velocit dell'oggetto filtrata
Matrix Vel0(2,1);  // Stima velocit dell'oggetto derivata
Matrix Acc(2,1);   // Stima accelerazione oggetto
Matrix Ti(2,2);    // Parametro integrale del PID non lineare

Matrix Temp(2,1);  // vettore temporaneo per operazioni varie

Matrix B(2,2);     // matrice di passaggio coordinate
Matrix Kp(2,2);    // matrice guadagno proporzionale
Matrix Ki(2,2);    // matrice guadagno integrale
Matrix Kd(2,2);    // matrice guadagno derivativo
Matrix Delta(2,2); // matrice indice di trasformazione PIDNL

Matrix Tool(3,3);  // Trasformazione rappresentante la Telecamera.
                   // cioe' passaggio dall'END-effector all'obbietivo
                   // della telecamera.
Matrix Etool(3,1); // segnale di correzione mandato al Puma

// assegnamento matrici
C=Centro;
B=Base;
Kp=Prop;
Ki=Integ;
Kd=Deriv;
Tool=end_effector;   // sist di rif. telecamera rispeto all'end-effector
Tool.display("\nTOOL TELECAMERA");
```

```
    int cm=0;               // indica il comando da inviare
    double time=0;            // tempo trascorso tra invio di due comandi
    double time_tot=0;        // tempo trascorso da inizio simulazione
    double time_start=0;    // tempo iterazione precedente programma principale
    double time_prec=0;     // Istante di invio comando precedente
    double time_send=0;
    double time_control=0;  // Tempo di cotrollo imposto

    double freq(10);
                            // variabili di regolazione uscita dati per MATLAB
    double ecom1=0;         // errore asse x all'istante di invio comando
    double ecom2=0;         // errore asse y all'istante di invio comando
    double ecom=0;            // abilita la stampa tra due comandi inviati al PUMA

    int TrackObj;

    Calibrazione(&B);

    FILE *datastream;       // File uscita leggibile da Matlab
    datastream=fopen("c:/home/damiano/anello/matlab/Fileout.m","w");
    fprintf(datastream,"out=[");
    clean();
    MappTimer(M_TIMER_RESET,M_NULL);
        do
        {

            obj=image.ImelLabeling(buffer,image_size_x,image_size_y,0,0,8,8 );
            //printf("Immagine processate :%i \n",cont);
            cont=cont+1;
            c1=image.ImelFilter(500);
//            printf("Sono rimasti dopo il filtraggio :%i\n",c1);
            TrackObj=image.Tracking(1,0,0);   // individuo il target
            t=image.ImelObjectInfo(TrackObj); // ne estraggo il puntatore

                if (t!=NULL)
                {
                    image.ImelOrientation(t); // calcolo l'orientamento


                    if (t->area>2500)  // ne controllo l'area.
                    {
                        target=true;   // questo e' il blocco fondamentale
                        P(1,1)=t->cgx;
                        P(2,1)=t->cgy;
                        E=C-P;             // errore in pixel
                        Esr=B*E;           // ricavo l'errore nello spazio robot

                        // printf("X=%.1f Y=%.1f Area=%.1f teta=%.3f \n",t->cgx,
                        //        t->cgy,t->area,t->teta*180/pi);
                        MappTimer(M_TIMER_READ,&time_tot);

                        // -------------- Controllori ------------------
```

```
//Er=0.4*Esr;  // proporzionale
//PID_MIO(time_tot,&B,&Kp,&Ki,&Kd,&E,&Er);
//PD_TIME(time_tot,&B,0.5,.17,10,&E,&Er);
//PID_TIME(time_tot,&B,0.4,4,.17,10,&E,&Er);
PIDNL_TIME(time_tot,&B,0.3,1.5,.1,10,&E,&Er,&Delta);
// ---------------------------------------------

// ----------------  Preditore  ------------------
//Er=1*Esr;  // semplice proporzionale
//PREDICTOR2(time_tot,&Er,&Out1s,2);   // predizione futura
//PREDICTOR_PUMA(time_tot,&Er,&Out1s,2);
//Er=Er-Out1s;                          // aggiornamento predizione
// ********  eventuali controlli dopo il preditore
//PD_ROBOT(time_tot,0.5,.17,10,&Er,&Er);
//PID_ROBOT(time_tot,0.5,1,.12,10,&Er,&Temp);Er=Temp;
//Er=Er+.02*Vel;  // eventuale controllo di velocit.
// -------------- fine blocco preditore ----------


// -------------Stima della velocita' target ------------

PREDICTOR_PUMA(time_tot,&Er,&Outs,0);// stima pos.Puma
Mov=Esr+Outs;                         // stima pos.oggetto
STIMAVEL(time_tot,&Mov,&Vel);         // stima vel. oggetto
Delta(1,1)=FNCT_NL(Vel(1,1));         // delta asse x
Delta(2,2)=FNCT_NL(Vel(2,1));         // delta asse y
// ---------------------------------------------


// Calcola come inviare le uscite in funzione
// del TOOL (posizione telecamera rispetto
// all'End-effector) rappresentato dalla matrice Tool.
Etool(1,1)=Er(1,1);
Etool(2,1)=Er(2,1);
Etool(3,1)=0;
Etool=Tool*Etool; // conversione Tool effettuata

sprintf(str_command,"%.2f,%.2f,%.2f",Etool(1,1),
                                    Etool(2,1),Etool(3,1));


if (cm==0)
{
    if ((ris=Puma.Wait("."))!=ROBOT_CMD_BLOCK)
    {
        MappTimer(M_TIMER_READ,&time_send);
        cm=1;
        }

}
if (cm==1 /*&& time_tot-time_control>0.2*/)
{
    time_control=time_tot;
```

```
                            if ((ris=Puma.SendCommand(str_command))!=ROBOT_CMD_BLOCK)
                            {
                                Esr=B*E;
                                ecom1=Esr(1,1);
                                ecom2=Esr(2,1);
                                cm=0;
                                ecom=1;
                                MappTimer(M_TIMER_READ,&time_tot);
                                time=time_tot-time_prec;
                                time_prec=time_tot;
                                //MODELLO(time_tot,&Er,&Outs,1);
                                PREDICTOR_PUMA(time_tot,&Er,&Outs,1);
                                //PREDICTOR_PUMA(time_tot,&Er,&Out1s,2);

                            }
                        }

                    }


                }

        MappTimer(M_TIMER_READ,&time_tot);
        if (ecom==1)
        {
            printf("\nTEMPO DI ESECUZIONE %.2f \n",time);
            ecom=0;
        }
        // stampa dati sul file di output ogni 33ms (circa)
        if (time_tot-time_start>0.033)
        {

            // Misure di uscita per la visualizzazione in Matlab.
            fprintf(datastream,
"%.2f %.2f %.2f %.2f %.2f %.2f %.2f  %.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f \n",
                                time_tot,Esr(1,1),Esr(2,1),ecom1,ecom2,Outs(1,1),
                                Outs(2,1),Esr(1,1)+Outs(1,1),Esr(2,1)+Outs(2,1),
                                Pos(1,1),Pos(2,1),Vel(1,1),Vel(2,1),Acc(1,1),Acc(2,1),
                                Ti(1,1),Ti(2,2));

            time_start=time_tot-(time_tot-time_start-0.033);
        }
        if (kbhit()) car=getch();
        Sleep(0);
        }while(car!=32);

    printf("Terminato Processate %i immagini. \n",cont);
    fprintf(datastream,"];");
    fclose(datastream);
    public_ender=true;
    while (public_ender==false) ;
    image.ImelFree();
}
```

```
// ----------------------------------------------------------------
// Setto la console di visualizzazione stringhe.......
// ----------------------------------------------------------------
void setup()
{
    Console = CreateConsoleScreenBuffer(GENERIC_READ | GENERIC_WRITE,
                                FILE_SHARE_WRITE | FILE_SHARE_READ,
                                0,
                                CONSOLE_TEXTMODE_BUFFER,NULL);
    COORD max;
    SMALL_RECT rect;

    Console=GetStdHandle(STD_OUTPUT_HANDLE);
    max.X=100;
    max.Y=50;
    rect.Top=1;
    rect.Left=1;
    rect.Right=100;
    rect.Bottom=50;

    SetConsoleScreenBufferSize(Console,max);
    SetConsoleActiveScreenBuffer(Console);
    SetConsoleWindowInfo(Console,true,&rect);
    SetConsoleTitle("Prova di visione");
}

// ------------------------------------------------------------------------------------
// Procedura di pulizia shermo
// ------------------------------------------------------------------------------------
void clean()
{
            cur.X=0;
            cur.Y=0;
            SetConsoleCursorPosition(StdOut,cur);
            int c;
            for (c=1;c<30;c++)
            {
            printf("                                                                \n");
            }
            cur.X=0;
            cur.Y=0;
            SetConsoleCursorPosition(StdOut,cur);
}

// ------------------------------------------------------------------------
// Inizializzazione dispositivi (virtuali) della libreria Mil
// ------------------------------------------------------------------------
void MatroxInit()
{
    // Alloco tutti i dispositivi e strutture necessarie.....
    MappAlloc(M_DEFAULT,&MilApplication);
    MsysAlloc(M_DEF_SYSTEM_TYPE,M_DEF_SYSTEM_NUM,M_SETUP,&MilSystem);
```

```
    // Indico il file dcf da cui recuperare i dati di setup telecamera
    MdigAlloc(MilSystem,M_DEF_DIGITIZER_NUM,FILE_DCF,M_DEF_DIGITIZER_INIT ,&MilDigitizer);
    MdispAlloc(MilSystem,M_DEFAULT,M_DEF_DISPLAY_FORMAT,M_DEFAULT,&MilDisplay);

    // alloco il buffer immagine
    MbufAlloc2d(MilSystem,
                (long)IMAGE_X_DIM,
                (long)IMAGE_Y_DIM,
                8L+M_UNSIGNED,
                M_IMAGE+M_GRAB+M_DISP+M_PROC,&MilImage);

    // alloco la memoria per la look table
   MbufAlloc1d(    MilSystem,
                256,
                8+M_UNSIGNED,
                M_LUT,
                &Look_Table);

    scala=SCALA;      // indica la risuzione di scala


    // definisco la funzione a soglia della Lut table
    MgenLutRamp(Look_Table,THRESOLD,255,255,255);  // LUT
    // Modalit asincrona.....
    MdigControl(MilDigitizer,M_GRAB_MODE,M_ASYNCHRONOUS);

    // Formatto finestra del display
    // Seleziono  il display da visualizzare
    MdispSelect(MilDisplay,MilImage);
    MdispControl(MilDisplay,M_WINDOW_MINBUTTON,M_DISABLE);
    MdispControl(MilDisplay,M_WINDOW_MAXBUTTON,M_DISABLE);
    MdispControl(MilDisplay,M_WINDOW_INITIAL_POSITION_X,700);
    MdispControl(MilDisplay,M_WINDOW_INITIAL_POSITION_Y,300);
    MdispControl(MilDisplay,M_WINDOW_RESIZE,M_ENABLE);
    MdispControl(MilDisplay,M_WINDOW_TITLE_NAME,(long)"Camera View");
    MdispControl(MilDisplay,M_WINDOW_MENU_BAR_CHANGE,M_DISABLE);
    MdispControl(MilDisplay,M_WINDOW_SCROLLBAR,M_DISABLE);
    MdispZoom(MilDisplay,(long)scala,(long)scala);

    //regolo la scala del digitizer
    MdigControl(MilDigitizer,M_GRAB_SCALE,(double)1/scala);

    // regolazione precisione timer di sincronizzazione
    double time_precision=1/100;
    MappTimer(M_TIMER_RESOLUTION,&time_precision);
}

// ----------------------------------------------------------------------
// Procedura di rilascio risorse allocate dalla libreria MIL
// Bisogna rilasciare tutti i disp. usati in ordine inverso...
// ----------------------------------------------------------------------
void MatroxShutdown()
```

```
{
   MbufFree(Look_Table);
   MbufFree(MilImage);
   MdispFree(MilDisplay);
   MdigFree(MilDigitizer);
   MsysFree(MilSystem);
   MappFree(MilApplication);
}


// --------------------------------------------------------------------
// Calibrazione Automatica permette ricavare la matrice di trasformazione
// dallo spazio dell'immagine  a quello del robot (piano di lavoro)
// Questo in piu' permette di rendere il controllore (ad esempio PID)
// indipendente dal cambiamento di base (cioe' rotazione Telecamera rispetto
// all'end-effector)
// --------------------------------------------------------------------
bool Calibrazione(Matrix* B)
{
    int ris;
    int obj;
    Matrix p1(2,1);
    Matrix p2(2,1);
    Matrix p3(2,1);

    Matrix R(3,3);
    Matrix C(3,3);
    Matrix K(3,3);

    int mv=30;  // movimento in millimetri utilizzato per la calibrazione
                // deve essere uguale al corrispettivo coefficiente
                // nel programma sul Controller UNIVAL
    Puma.Sincronous();

    Sleep(2000);
    while((ris=Puma.Wait("Cal1:"))!=ROBOT_OK);
    image.ImelLabeling(buffer,image_size_x,image_size_y,0,0,1,1 );
    obj=image.ImelFilter(500);
    if (obj==1)
    {
        t=image.ImelObjectInfo(IMEL_MODE_NEXT);
        if (t!=NULL)
        {
            // punto di riferimento
            p1(1,1)=t->cgx;
            p1(2,1)=t->cgy;
        }
    }

    while((ris=Puma.SendCommand(" "))!=ROBOT_OK);
    while((ris=Puma.Wait("Cal2:"))!=ROBOT_OK);
    Sleep(3000);
    image.ImelLabeling(buffer,image_size_x,image_size_y,0,0,1,1 );
```

```
obj=image.ImelFilter(500);
if (obj==1)
{
    t=image.ImelObjectInfo(IMEL_MODE_NEXT);
    if (t!=NULL)
    {
        // traslazione in X (Tool robot)
        p2(1,1)=t->cgx;
        p2(2,1)=t->cgy;
    }
}

while((ris=Puma.SendCommand(" "))!=ROBOT_OK);
while((ris=Puma.Wait("Cal3:"))!=ROBOT_OK);
Sleep(3000);
image.ImelLabeling(buffer,image_size_x,image_size_y,0,0,1,1 );
obj=image.ImelFilter(500);
if (obj==1)
{
    t=image.ImelObjectInfo(IMEL_MODE_NEXT);
    if (t!=NULL)
    {
        // traslazione in Y ( tool robot)
        p3(1,1)=t->cgx;
        p3(2,1)=t->cgy;
    }
}

while((ris=Puma.SendCommand(" "))!=ROBOT_OK);
printf("\nFine Acquisizione riferimenti\n");

TIPO Rt[9]={mv,0,0,0,mv,0,1,1,1};
TIPO Ct[9]={0,0,0,0,0,0,1,1,1};
R=Rt;
C=Ct;

// Calcolo i differenziali di P3 e P2 rispetto a P1(riferimento)
C(1,2)=p3(1,1)-p1(1,1);
C(1,1)=p2(1,1)-p1(1,1);
C(2,2)=p3(2,1)-p1(2,1);
C(2,1)=p2(2,1)-p1(2,1);

C.display("\nC");
R.display("\nR");

K=R*INV(C);  // calcolo della matrice id trasf. da pixel a mm
K.display("Matrice K");

(*B)(1,1)=K(1,1);
(*B)(1,2)=K(1,2);
(*B)(2,1)=K(2,1);
(*B)(2,2)=K(2,2);
```

```
    (*B).display("B=");
    printf("Premi un tasto per continuare\n");
    getch();
    clean();
    return true;
}
```

F.1.2 File Openloop.c

Di seguito viene solo riportata la procedura `processamento()` del file `OPENLOOP.C` dato che tutto il resto coincide con il contenuto del file `EYEHAND.C`.

```
void processamento()
{
    int pitch;
    int car=0;            // carattere di comando ricevuto da tastiera

    int image_size_x;     // dimensioni x,y immagine etichettata
    int image_size_y;

    // le dimensioni a cui riferirsi sono quelle del DIGITIZER
    int dimx,dimy;

    MdigInquire(MilDigitizer,M_SOURCE_SIZE_X,&dimx);
    MdigInquire(MilDigitizer,M_SOURCE_SIZE_Y,&dimy);

    // Ricavo il puntatore al buffer dell'immagine MILIMAGE
    // ricavo poi le dimensioni x in pitch del buffer
    buffer=(unsigned char*)MbufInquire(MilImage,M_HOST_ADDRESS,M_NULL);
    pitch =(int)MbufInquire(MilImage,M_PITCH,M_NULL);

    image_size_x=pitch;
    image_size_y=dimy/scala;

    MdigControl(MilDigitizer,M_GRAB_SCALE,(double)1/scala);
    image.ImelInit(buffer,pitch,dimy);
    printf("Dimensione x :%i \n",dimx);
    printf("Dimensione y :%i \n",dimy);
    printf("Fattore di riduzione :%i \n",scala);
    printf("Dim ridotta x :%i \n",image_size_x);
    printf("Dim ridotta y :%i \n",image_size_y);

    MdigGrabContinuous(MilDigitizer, MilImage);

    int row=0;
    int col=0;
    int obj;
    int cont=0;
    int c1=0;
    char str_command[80];
    bool target=false;
```

```
    int ris;
    int ve=0;     // numero vettore

    CALCOLO angle=0;    // angolo di rotazione oggetto
    CALCOLO angle_rif=45; // angolo di riferimento
    CALCOLO angle_delta=0; // angolo di spostamento


    printf("Inizializzo Puma \n");

    Puma.Initialize();
    Puma.Asincronous();
    Puma.SendCommand("do departs(distance/2)");
    Puma.Prompt(PROMPT_DOT);
    Puma.SendCommand("do departs(distance/2)");
    Puma.Prompt(PROMPT_DOT);
    Puma.SendCommand("do move a");
    Puma.Prompt(PROMPT_DOT);
    Puma.SendCommand("exec goa");
    Puma.Wait("position");

    getch();

    // matrice piano orizzontale
    TIPO CRobot[9]={162.331,-52.138,-52.136,
                    222.655,221.386,397.318,
                    -13.108,-13.090,-11.306};

    // matrice piano inclinato
/*    TIPO CRobot[9]={161.994,-51.410,-48.230,
                    292.542,290.833,446.5,
                    -23.226,-22.851,73.2};*/

    TIPO CImage[9]={1,0,0,0,1,0,0,0,1};  // coordinate di partenza spazio immagine

    Matrix K(3,3);      // matrice di trasformazione da Image a Robot
    Matrix C(3,3);      // matrice riferimenti spazio Immagine
    Matrix R(3,3);      // matrice riferimenti spazio Robot
    Matrix Tc(3,1);   // vettore posizione Target nell'immagine
    Matrix Tr(3,1);   // vettore stima posizione Target nello spazio.
    Matrix Te(3,3);
    R[CRobot];
    C[CImage];

    // acquisisco i riferimenti...
        ve=0;
        obj=image.ImelLabeling(buffer,image_size_x,image_size_y,0,0,1,1);
        c1=image.ImelFilter(50);
        do
        {
            t=image.ImelObjectInfo(IMEL_MODE_NEXT);
            if (t!=NULL)
            {
```

```
            image.ImelOrientation(t);
            if (t->area<300
                )
            {
                printf("Riferimento %i trovato in x=%.2f y=%.2f \n",
                           ve+1,t->cgx,t->cgy);
                C(ve)=t->cgx;
                C(3*1+ve)=t->cgy;
                C(3*2+ve)=1;
                ve++;
            }
        }
    }while(t!=NULL);

if (ve== 3)
{
    printf("Riferimenti acquisiti.\n");
    printf("Premi un tasto.\n");
    getch();

    Te=precalib(R,C,1);   // preparazione di ua matrice per la calib.
    R.display("R=");
    C.display("C=");
    Te.display("Te");
    getch();


    K=R*Te.inverse();

    // Ordino i riferimenti.

    do    // ciclo do dei comandi !!!!!!!!!!!!!!
    {
    while (car!=32)
    {
        //cur.X=0;
        //cur.Y=0;
        //SetConsoleCursorPosition(StdOut,cur);
        if (cont%100==0) clean();
        obj=image.ImelLabeling(buffer,image_size_x,image_size_y,0,0,8,8);

        //printf("Immagine processate :%i\n",cont);
        cont=cont+1;
        //printf("Imel: Trovati %i oggetti\n",obj);
        //printf("Trovati %i riferimenti\n",ve);

        c1=image.ImelFilter(50);
        //printf("Sono rimasti dopo il filtraggio :%i\n",c1);
    //    printf("*****************************\n");

        target=false; // parto supponedno che il target non ci sia.
        do
        {
```

```
                t=image.ImelObjectInfo(IMEL_MODE_NEXT);
                if (t!=NULL)
                {
                    image.ImelOrientation(t);

                    printf("X=%.1f Y=%.1f Area=%.1f teta=%.3f \n",
                        t->cgx,t->cgy,t->area,t->teta*180/pi);

                    if (t->area>500)
                    {
                        target=true;
                        angle=0;
                        angle_delta=0;
                        //clean();
                        Tc(0)=t->cgx;
                        Tc(1)=t->cgy;
                        Tc(2)=1;
                        Tr=K*Tc;
                        angle=-(90-(t->teta*180/pi));

                        if (angle<-90) angle+=180;
                        angle_delta=angle_rif-angle;
                        if (angle_delta<-90) angle_delta+=180;
                        if (angle_delta>90) angle_delta-=180;
                        str_command[0]=0;
                        sprintf(str_command,"%.2f %.2f %.2f %.2f %.2f",
                            Tr(1,1),Tr(2,1),Tr(3,1),angle,angle_delta);
                        printf("Invio il comando: %s\n",str_command);
                        Puma.SendCommand(str_command);
                        Puma.Wait("attesa:");
                        Puma.SendCommand("0");
                        Puma.Wait("insert position:");
                        getch();
                        }
                }
            }while(t!=NULL);
        }
        }while(car!=32);
    printf("Terminato Processate %i immagini. \n",cont);
    }
    else printf("Riferimenti assenti....\n");
    Puma.Terminal();
    while (public_ender==false) ;
    image.ImelFree();
}
```

F.2 Codice dell'oggetto Robot

F.2.1 File Robot.h

```
#ifndef ROBOT_CLASS
#define ROBOT_CLASS
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include "buffer.h"
#include "vector.h"

#define COMMAND_MAX_CHAR    80      // massimo numero di caratteri per comando
#define PROMPT_DOT          "\n."   // prompt dot      di VAL II
#define PROMPT_MONITOR      "\n*"   // prompt monitor  di VAL II

// Modalit in cui si pu inviare un comando all 'oggetto ROBOT
#define ROBOT_CMD_SEND                  1    // Invia comando
#define ROBOT_CMD_WAIT                  2    // Aspetta stringa
#define ROBOT_CMD_TERMINAL        4    // Apri modalit emulatore terminale
#define ROBOT_CMD_SHUTDOWN        8
#define ROBOT_CMD_INPUT                16    // getch() asincrono di attesa
#define ROBOT_SLEEP                    32    // pausa nell'esecuzione comandi

/*  ************************************************************
    *  Definizioni delle costanti di ritorno dell'oggetto ROBOT      *
    ************************************************************/
typedef enum ROBOT_RESULT{

        ROBOT_NOT_INIT = -1,        // Errore buffer non inizializzato
        ROBOT_ERROR  = 0,           // Errore generico (dipende dall'operazione)
        ROBOT_OK = 1,               // operazione conclusa in maniera corretta
        ROBOT_CMD_BLOCK =2          // comando gia' in esecuzione
};

/*  ************************************************************
        Definizione struttura di un comando
    ************************************************************/
typedef struct Instruction
{
    char command[COMMAND_MAX_CHAR];
    int  opzioni;
}Instruction;

/*  ************************************************************
            Definizione classe robot
    ************************************************************/
class ROBOT
{
public:
        ROBOT_RESULT  Initialize();
        ROBOT_RESULT  SendCommand(char*cmd);
```

```
        ROBOT_RESULT  Wait(char *cmd);
        ROBOT_RESULT  Sleep(char *cmd);
        ROBOT_RESULT  Prompt(char *prompt);
        ROBOT_RESULT  Terminal();
        ROBOT_RESULT  ShutDown();
        ROBOT_RESULT  Error();
        ROBOT_RESULT  Script();
        ROBOT_RESULT  Pushkey(char*cmd);
        ROBOT_RESULT  Asincronous();
        ROBOT_RESULT  Sincronous();
private:
        bool ROBOT_INIT;
};
#endif
```

F.2.2 File Robot.cpp

```
/* ****************************************************************
   Grazie all'inclusione di Robot.h e rob_ctrl.h
   questo modulo ha accesso a tutte le funzionalit di :
   1) Esecuzione comandi
   2) Porta seriale
   3) Oggetti Buffer e Command.
*****************************************************************/

#include "Robot.h"
#include "Rob_ctrl.h"

extern BufferChar buffer;
extern vector <Instruction> Cmd;
extern bool cmd_sinc;
extern bool sincrono;

// ****************************************************************
//    Comando di inizializzazione  Thread,Strutture dati,Com.
// ****************************************************************

ROBOT_RESULT ROBOT::Initialize()
{
    if (ComInit())     printf("Init COM : OK\n");
                    else
                    {
                        printf("Errore inizializzazione COM\n");
                        return(ROBOT_NOT_INIT);
                    }
    RobotInit();
    ROBOT_INIT=true;
    return(ROBOT_OK);
}

// ****************************************************************
// Inserimento di un comando nella lista (VECTOR) comandi.
```

```
// ********************************************************************
ROBOT_RESULT ROBOT::SendCommand(char*cmd)
{
    if (!ROBOT_INIT) return(ROBOT_NOT_INIT);
    if (sincrono && cmd_sinc) return ROBOT_CMD_BLOCK;
    cmd_sinc=true;
    Instruction temp;
    strcpy(temp.command,cmd);
    temp.opzioni=ROBOT_CMD_SEND;
    Cmd.put(temp);
    return(ROBOT_OK);
}

// ********************************************************************
// Inserisce un comando (nella lista comandi) di attesa stringa.
// ********************************************************************
ROBOT_RESULT ROBOT::Wait(char*cmd)
{
    if (!ROBOT_INIT) return(ROBOT_NOT_INIT);
    if (sincrono && cmd_sinc) return ROBOT_CMD_BLOCK;
    cmd_sinc=true;
    Instruction temp;
    strcpy(temp.command,cmd);
    temp.opzioni=ROBOT_CMD_WAIT;
    Cmd.put(temp);
    return(ROBOT_OK);
}

// ********************************************************************
// Effettua una pausa
// ********************************************************************

ROBOT_RESULT ROBOT::Sleep(char* time)
{
    if (!ROBOT_INIT) return(ROBOT_NOT_INIT);
    if (sincrono && cmd_sinc) return ROBOT_CMD_BLOCK;
    cmd_sinc=true;
    Instruction temp;
    strcpy(temp.command,time);
    temp.opzioni=ROBOT_SLEEP;
    Cmd.put(temp);
    return(ROBOT_OK);
}

ROBOT_RESULT ROBOT::Prompt(char*prompt)
{
    if (!ROBOT_INIT) return(ROBOT_NOT_INIT);
    if (sincrono && cmd_sinc) return ROBOT_CMD_BLOCK;
    cmd_sinc=true;
    Instruction temp;
    strcpy(temp.command,prompt);
    temp.opzioni=ROBOT_CMD_WAIT;
    Cmd.put(temp);
```

```
    return(ROBOT_OK);
}

ROBOT_RESULT ROBOT::Pushkey(char*prompt)
{
    if (!ROBOT_INIT) return(ROBOT_NOT_INIT);
    if (sincrono && cmd_sinc) return ROBOT_CMD_BLOCK;
    cmd_sinc=true;
    Instruction temp;
    strcpy(temp.command,prompt);
    temp.opzioni=ROBOT_CMD_INPUT;
    Cmd.put(temp);
    return(ROBOT_OK);
}

// ********************************************************************
// Lancia asincronicamente l'emulatore terminale....
// Questo verra aperto solo quando tutti gli altri comandi prima di lui
// nella lista comandi sono stati eseguiti.
// ********************************************************************

ROBOT_RESULT ROBOT::Terminal()
{
    if (!ROBOT_INIT) return(ROBOT_NOT_INIT);
    if (sincrono && cmd_sinc) return ROBOT_CMD_BLOCK;
        cmd_sinc=true;
        Instruction temp;
        temp.command[0]=0;
        temp.opzioni=ROBOT_CMD_TERMINAL;
        Cmd.put(temp);
        return(ROBOT_OK);
}

// ********************************************************************
//
//    SHUTDOWN asincrono....invio un comando ROBOT_CMD_SHUTDOWN alla lista
//  comandi,quando questo verr eseguito chiuder tutte le strutture....
//  abilitando la variabile globale global_ender che mi conclude il main
//
// ********************************************************************
ROBOT_RESULT ROBOT::ShutDown()
{
    if (!ROBOT_INIT) return(ROBOT_NOT_INIT);
    if (sincrono && cmd_sinc) return ROBOT_CMD_BLOCK;
    cmd_sinc=true;
    Instruction temp;
    temp.opzioni=ROBOT_CMD_SHUTDOWN;
    Cmd.put(temp);
    return(ROBOT_OK);
}

ROBOT_RESULT ROBOT::Sincronous()
{
```

```
    sincrono=true;
    cmd_sinc=false;
    return(ROBOT_OK);
}

ROBOT_RESULT ROBOT::Asincronous()
{
    sincrono=false;
    return(ROBOT_OK);
}
```

F.3 Libreria Control

F.3.1 File Control.h

```
// ---------------------------------------------------------------------------
// libreria di implementazione regolatori lineari e non lineari piu' predittore
// ---------------------------------------------------------------------------

#ifndef CONTROL_VERSION
#define CONTROL VERSION 1.0
#ifndef MATRIX_VERSION
#include "matrice.h"
#endif

void PID_MIO(double time,Matrix* Base,Matrix* Kp,Matrix* Ki,Matrix*  Kd,Matrix* E,Matrix* Eprova);
void PD_TIME(double time,Matrix* B,TIPO Kp,TIPO Td,TIPO N,Matrix* E,Matrix* Exit);
void PID_TIME(double time,Matrix* B,TIPO K,TIPO Ti,TIPO Td,TIPO N,Matrix* E,Matrix* Exit);
void PD_ROBOT(double time,TIPO Kp,TIPO Td,TIPO N,Matrix* E,Matrix* Exit);
void PID_ROBOT(double time,TIPO K,TIPO Ti,TIPO Td,TIPO N,Matrix* E,Matrix* Exit);

void MODELLO(double time,Matrix* E,Matrix* Exit,int mode);

void PREDICTOR_PUMA(double time,Matrix* E,Matrix* Exit,int mode);
void PREDICTOR2(double time,Matrix* E,Matrix* Exit,int mode);

void STIMAVEL(double time,Matrix* Pos,Matrix* Vel);

void FILTRO_MEDIA(Matrix* E,Matrix* Exit);
void BUTTERWORTH(TIPO time,Matrix* E,Matrix* Exit);
void FILTRO_STATO(TIPO time,Matrix* E,Matrix* Pos,Matrix* Vel,Matrix* Acc);

TIPO FNCT_NL(TIPO x);
void PIDNL_TIME(double time,Matrix* B,TIPO Kp,TIPO Ti,TIPO Td,TIPO N,Matrix* E,Matrix* Exit,Matrix* Delta);

#endif
```

F.3.2 File Control.cpp

```
#include "control.h"
```

```
// ----------------------------------------------------------------
// Implementazione PID con guadagni differenziabili
// per ogni asse attraverso il passaggio di opportune matrici.
// ----------------------------------------------------------------
void PID_MIO(double time,Matrix* B,Matrix* Kp,Matrix* Ki,Matrix* Kd,Matrix* E,Matrix* Exit)
{
    static Matrix Eprec(2,1);
    static double time_prec=0;
    static Matrix integrale(2,1);
    Matrix derivata(2,1);

    // ------------------------------------------------------
    // codice da eliminare dopo le prove sovrascrive le variabi esterne

    TIPO Kcr=0.6;    // termine adimensionale
    TIPO Pcr=1.8;

    TIPO Prop[4]={Kcr,0,
                  0,Kcr};
    TIPO Integ[4]={.5*Pcr,0,
                   0,.5*Pcr};
    TIPO Deriv[4]={0.1*Pcr,0,
                   0,0.1*Pcr};

    (*Kp)=Prop;
    (*Ki)=Integ;
    (*Kd)=Deriv;
    (*Exit)=(*B)*(*Kp)*((*E)+(INV(*Ki)*integrale)+(*Kd)*derivata);
    Eprec=(*E);
    time_prec=time;
}

// ----------------------------------------------------------------
// Implementazione PD a tempo variabile
// ----------------------------------------------------------------
void PD_TIME(double time,Matrix* B,TIPO Kp,TIPO Td,TIPO N,Matrix* E,Matrix* Exit)
{
    static double time_pd_hold=0;
           Matrix Uk(2,1);
    static Matrix Uk0(2,1);
    static Matrix Ek0(2,1);
    Matrix Temp(2,1);
    TIPO Kbeta;
    TIPO Kalfa;
    TIPO E0;  // coef. errore passato
    TIPO U0;  // coef. uscita passata
    TIPO E1;  // coef. errore presente
    TIPO T;   // tempo di campionamento

    if (time-time_pd_hold>=0.07)
    {
        T=time-time_pd_hold;
```

```
        time_pd_hold=time;
        Kbeta=Td/(N*T+Td);
        Kalfa=N*Kbeta;

        U0=Kbeta;
        E1=Kp*(1+Kalfa);
        E0=-Kp*(Kbeta+Kalfa);
        Uk=E1*(*E)+U0*Uk0+E0*Ek0;
        (*Exit)=(*B)*Uk;
        Uk0=Uk; // valore precedente uscita
        Ek0=(*E);    // valore precedente errore
    }
}

// ------------------------------------------------------------------
// Implementazione PD a tempo variabile
// questo PD e' in coordinate robot ed e' stato implementato per
// essere utilizzato dopo un Eventuale Preditore.
// ------------------------------------------------------------------
void PD_ROBOT(double time,TIPO Kp,TIPO Td,TIPO N,Matrix* E,Matrix* Exit)
{
    static double time_pd_hold=0;
           Matrix Uk(2,1);
    static Matrix Uk0(2,1);
    static Matrix Ek0(2,1);
    Matrix Temp(2,1);

    TIPO Kbeta;
    TIPO Kalfa;
    TIPO E0;  // coef. errore passato
    TIPO U0;  // coef. uscita passata
    TIPO E1;  // coef. errore presente
    TIPO T;   // tempo di campionamento

    if (time-time_pd_hold>=0.13)
    {
        T=time-time_pd_hold;
        time_pd_hold=time;
        Kbeta=Td/(N*T+Td);
        Kalfa=N*Kbeta;

        U0=Kbeta;
        E1=Kp*(1+Kalfa);
        E0=-Kp*(Kbeta+Kalfa);

        // sono passati piu' di 30ms per cui elaboro l'immagine

        printf("\nTempo di controllo e' = %.2f\n",time);

        Uk=E1*(*E)+U0*Uk0+E0*Ek0;
        (*Exit)=Uk;
        Uk0=Uk; // valore precedente uscita
        Ek0=(*E);    // valore precedente errore
```

```
    }
}

// ---------------------------------------------------------------
// Implementazione PID a tempo variabile
// ---------------------------------------------------------------
void PID_TIME(double time,Matrix* B,TIPO K,TIPO Ti,TIPO Td,TIPO N,Matrix* E,Matrix* Exit)
{
    static double time_pd_hold=0;
           Matrix Uk0(2,1);
    static Matrix Uk1(2,1);
    static Matrix Uk2(2,1);
    static Matrix Ek1(2,1);
    static Matrix Ek2(2,1);
    Matrix Temp(2,1);

    TIPO Kbeta;    // costanti transitorie di calcolo.
    TIPO Kalfa;
    TIPO Kgamma;

    TIPO q0;  // coef. errore  presente
    TIPO q1;  // coef. errore di 1 istante passato
    TIPO q2;  // coef. errore di 2 istanti passati
    TIPO r2;  // coef. uscita di 2 istanti passati
    TIPO r1;  // coef. uscita di 1 istante passato.

    TIPO T;   // tempo di campionamento

    if (time-time_pd_hold>=0.130)
    {
        T=time-time_pd_hold;
        time_pd_hold=time;
        Kalfa=T/Ti;
        Kgamma=Td/(N*T+Td);
        Kbeta=N*Kgamma;

        q0=K*(1+Kbeta);
        q1=-K*(1+Kgamma-Kalfa+2*Kbeta);
        q2=K*(Kgamma-Kalfa*Kgamma+Kbeta);
        r1=Kgamma+1;
        r2=Kgamma;

        Uk0=r1*Uk1-r2*Uk2+q0*(*E)+q1*Ek1+q2*Ek2;
        (*Exit)=(*B)*Uk0;
        Uk2=Uk1;     // valore a due istanti precedenti uscita
        Ek2=Ek1;     // valore a due istanti precedenti errore(ingresso)
        Uk1=Uk0;     // valore all'istante precedente uscita
        Ek1=(*E);    // valore all'istante precedente errore(ingresso)
        }
}

// ---------------------------------------------------------------
// Implementazione PID a tempo variabile,
```

```
// questo PID e' in coordinate robot ed e' stato implementato per
// essere utilizzato dopo un Eventuale Preditore.
// ------------------------------------------------------------------
void PID_ROBOT(double time,TIPO K,TIPO Ti,TIPO Td,TIPO N,Matrix* E,Matrix* Exit)
{

    static double time_pd_hold=0;
           Matrix Uk0(2,1);
    static Matrix Uk1(2,1);
    static Matrix Uk2(2,1);
    static Matrix Ek1(2,1);
    static Matrix Ek2(2,1);
    Matrix Temp(2,1);

    TIPO Kbeta;    // costanti transitorie di calcolo.
    TIPO Kalfa;
    TIPO Kgamma;

    TIPO q0;  // coef. errore  presente
    TIPO q1;  // coef. errore di 1 istante passato
    TIPO q2;  // coef. errore di 2 istanti passati
    TIPO r2;  // coef. uscita di 2 istanti passati
    TIPO r1;  // coef. uscita di 1 istante passato.

    TIPO T;   // tempo di campionamento

    if (time-time_pd_hold>=0.13)
    {
        T=time-time_pd_hold;
        time_pd_hold=time;
        Kalfa=T/Ti;
        Kgamma=Td/(N*T+Td);
        Kbeta=N*Kgamma;

        q0=K*(1+Kbeta);
        q1=-K*(1+Kgamma-Kalfa+2*Kbeta);
        q2=K*(Kgamma-Kalfa*Kgamma+Kbeta);

        r1=Kgamma+1;
        r2=Kgamma;

        Uk0=r1*Uk1-r2*Uk2+q0*(*E)+q1*Ek1+q2*Ek2;
        (*Exit)=Uk0;
        Uk2=Uk1;     // valore a due istanti precedenti uscita
        Ek2=Ek1;     // valore a due istanti precedenti errore(ingresso)
        Uk1=Uk0;     // valore all'istante precedente uscita
        Ek1=(*E);    // valore all'istante precedente errore(ingresso)
        }
}

// ------------------------------------------------------------------
//  PUMA_PREDICTOR implementa un sistema dinamico lineare rappresentato
//  nello spazio degli stati del tipo
```

```
//  Xk+1=A*Xk+B*E ............. uno per ogni asse
// mode = 0  aggiorno lo stato e leggo l'uscita del sistema dinamico
// mode = 1  aggiorno l'ingresso del sistema dinamico
// mode = 2  genero una previsione dell'uscita (PREDITORE)
// ---------------------------------------------------------------------

void PREDICTOR_PUMA(double time,Matrix* E,Matrix* Exit,int mode)
{

    static double time_pd_hold=0;
    static double time_puma=0;
    int dim=10;                 // dimensione sopazioe degli stati
    static Matrix X(dim,1);     // stato del modello coordinata x
    static Matrix Y(dim,1);     // stato del modello coordinata y

    static Matrix Xm(dim,1);     // stato x in modalit di predizione
    static Matrix Ym(dim,1);     // stato y in modalit di predizione

    static Matrix A(dim,dim);   // matrice dell'evoluzione
    static Matrix Ap(dim,dim);  // matrice preditore
    static Matrix B(dim,1);     // matrice delgi ingressi
    static double K=10;         // attenuazione del guadagno Puma
    static TIPO Ux=0;            // ingressi assi x,y
    static TIPO Uy=0;
    static TIPO St=0.037;       // sample time (aggiustato)

    // --------  assegnamento matrice A -------------
    A(2,1)=1;A(3,2)=1;A(4,3)=1;A(5,4)=1;
    A(6,5)=1;A(7,6)=1;A(8,7)=1;A(9,8)=1;
    A(10,9)=1;
    A(dim,dim-1)=1/K;A(dim,dim)=1;
    // --------  assegnamento matrice Ap -------------
    Ap(dim,1)=1/K;Ap(dim,2)=1/K;Ap(dim,3)=1/K;Ap(dim,4)=1/K;
    Ap(dim,5)=1/K;Ap(dim,6)=1/K;Ap(dim,7)=1/K;Ap(dim,8)=1/K;
    Ap(dim,9)=1/K;Ap(dim,10)=1/K;
    Ap(dim,dim)=1;
    // --------- assegnamento matrice B --------------
    B(1,1)=1;

    if (mode==1)   // e' giunto un nuovo ingresso
    {
        Ux=(*E)(1,1);
        Uy=(*E)(2,1);
        X=A*X+Ux*B;
        Y=A*Y+Uy*B;
    }

    if (mode==2)
    {
        Xm=X;
        Ym=Y;

        Xm=Ap*X-X;   // calcolo solo il differenziale dell'evoluzione
```

```
        Ym=Ap*Y-Y;

        (*Exit)(1,1)=Xm(dim,1);   // prendo lo stato che coincide con l'uscita
        (*Exit)(2,1)=Ym(dim,1);
        return;
    }
    if ((time-time_puma>St) && (mode==0))
    {
        // blocco calcolo evoluzione sistema in real-time
        time_puma=time-(time-time_puma-St);
        X=A*X+Ux*B;
        Y=A*Y+Uy*B;
    }
(*Exit)(1,1)=X(dim,1);   // prendo lo stato che coincide con l'uscita
(*Exit)(2,1)=Y(dim,1);
}

// ---------------------------------------------------------------------
// Modello del Puma 260
// devo garantire che questa procedura sia chiamata almeno una volta
// ogni 33ms altrimenti la previsione e' sbagliata.
// mode = 0  aggiorno lo stato e leggo l'uscita del sistema dinamico
// mode = 1  aggiorno l'ingresso del sistema dinamico
// mode = 2  genero una previsione dell'uscita (PREDITORE)
// ---------------------------------------------------------------------

void PREDICTOR2(double time,Matrix* E,Matrix* Exit,int mode)
{
    static double time_hold=0;
    static double time_puma=0;
    static double time_sample=0;
    TIPO ST=0.033;  // mediamente ottengo un simple time di 33ms.
    TIPO Kr=10;     // istanti di ritardo
    TIPO K=10;      // 95= tempo controllo 3sec
    static Matrix S0(2,1);
    static Matrix S1(2,1);
    static Matrix S2(2,1);
    static Matrix S3(2,1);
    static Matrix S4(2,1);
    static Matrix S5(2,1);
    static Matrix S5a(2,1);
    static Matrix S6(2,1);
    static Matrix S7(2,1);
    static Matrix S8(2,1);
    static Matrix S9(2,1);

    // variabili di memorizzazione stato di partenza.
    Matrix Sm0(2,1);Matrix Sm1(2,1);Matrix Sm2(2,1);Matrix Sm3(2,1);
    Matrix Sm4(2,1);Matrix Sm5(2,1);Matrix Sm5a(2,1);Matrix Sm6(2,1);
    Matrix Sm7(2,1);Matrix Sm8(2,1);Matrix Sm9(2,1);

    Matrix Statem(2,1);Matrix Inputm(2,1);Matrix Outm(2,1);
```

```
static Matrix Input(2,1);
static Matrix State(2,1);
static Matrix Out(2,1);

if (mode==1)
{
    time_hold=time;  // e' giunto un nuovo ingresso.
    S0=*E;
}
if (mode==2)
{
    // memorizzo lo stato di partenza
    Statem=State;
    Sm0=S0;Sm1=S1;Sm2=S2;Sm3=S3;Sm4=S4;
    Sm5=S5;Sm5a=S5a;Sm6=S6;Sm7=S7;Sm8=S8;Sm9=S9;

    for (int c=0;c<10;c++)
    {
    Inputm=Sm9;
    Sm9=Sm8;  // aggiorno gli istanti di ritardo...
    Sm8=Sm7;
    Sm7=Sm6;
    Sm6=Sm5a; // ritardo di aggiustamento
    Sm5a=Sm5;
    Sm5=Sm4;
    Sm4=Sm3;
    Sm3=Sm2;
    Sm2=Sm1;
    Sm1=Sm0;

    Outm=Statem+(1/K)*Inputm; // aggiornamento uscita in alter mode
    Statem=Outm;

    }
    (*Exit)=Statem-State;  // ritorno solo il differenziale delle uscite
    return;
}
if ((time-time_puma>ST) && (mode==0))
{
    Input=S9;
    S9=S8;  // aggiorno gli istanti di ritardo...
    S8=S7;
    S7=S6;
    S6=S5a; // ritardo di aggiustamento
    S5a=S5;
    S5=S4;
    S4=S3;
    S3=S2;
    S2=S1;
    S1=S0;

    time_puma=time-(time-time_puma-ST);
    Out=State+(1/K)*Input; // aggiornamento uscita in alter mode
```

```
        State=Out;
    }
    (*Exit)=Out;
}

// ---------------------------------------------------------------------
// Modello del Puma 260
// devo garantire che questa procedura sia chiamata almeno una volta
// ogni 33ms altrimenti la previsione e' sbagliata.
// ---------------------------------------------------------------------
void MODELLO(double time,Matrix* E,Matrix* Exit,int mode)
{

    static double time_hold=0;
    static double time_puma=0;
    static double time_sample=0;

    TIPO ST=0.033;  // mediamente ottengo un simple time di 33ms.
    TIPO Kr=10;     // istanti di ritardo
    TIPO K=10;

    static Matrix S0(2,1);
    static Matrix S1(2,1);
    static Matrix S2(2,1);
    static Matrix S3(2,1);
    static Matrix S4(2,1);
    static Matrix S5(2,1);
    static Matrix S5a(2,1);
    static Matrix S6(2,1);
    static Matrix S7(2,1);
    static Matrix S8(2,1);
    static Matrix S9(2,1);

    static Matrix Input(2,1);
    static Matrix State(2,1);
    static Matrix Out(2,1);

    if (mode==1)
    {
        time_hold=time;  // e' giunto un nuovo ingresso.
        S0=*E;
    }

    if (time-time_puma>ST)
    {
        Input=S9;
        S9=S8;  // aggiorno gli istanti di ritardo...
        S8=S7;
        S7=S6;
        S6=S5a; // ritardo di aggiustamento
        S5a=S5;
        S5=S4;
        S4=S3;
```

```
        S3=S2;
        S2=S1;
        S1=S0;

        time_puma=time-(time-time_puma-ST);
        Out=State+(1/K)*Input; // aggiornamento uscita in alter mode
        State=Out;
    }
    //printf("State=%.3f Input=%.3f\n",State(1,1),Input(1,1));
    (*Exit)=Out;
}

// ------------------------------------------------------------------
// Stima della velocit attraverso la derivazione
// del vettore in ingresso Pos ed un filtraggio passabasso
// ------------------------------------------------------------------
void STIMAVEL(double time,Matrix* Pos,Matrix* Vel)
{
    static double time_prec;
    static Matrix X1(2,1); // velocit stimata
    static Matrix X0(2,1); // velocit passata
    static Matrix E0(2,1); // posizione passata
    static int start=0;
    Matrix Temp(2,1);

    TIPO T;
    TIPO Td=0.260; // periodo tipico
    TIPO N=5;      // per N basso filtro molto....
    TIPO Kalfa;
    TIPO Kbeta;

    if (start==0)
    {
        start=1;
        E0=(*Pos);
        X1=(*Pos);
        (*Vel)=X1;
    }
    if (start==1 && time-time_prec>=0.040)
    {
        T=time-time_prec;
        time_prec=time;
        Kbeta=Td/(N*T+Td);
        Kalfa=N*Kbeta;

        X1=Kbeta*X0+Kalfa*((*Pos)-E0);
        X0=X1;
        E0=(*Pos);
        (*Vel)=X1;
    }
}

// ------------------------------------------------------------------
```

```
// filtro passa basso ottenuto mediando 10 campioni
// -------------------------------------------------------------------
void FILTRO_MEDIA(Matrix* E,Matrix* Exit)
{
    TIPO ST=0.26;  // mediamente ottengo un simple time di
    static TIPO start=0;

    static Matrix S0(2,1);
    static Matrix S1(2,1);
    static Matrix S2(2,1);
    static Matrix S3(2,1);
    static Matrix S4(2,1);
    static Matrix S5(2,1);
    static Matrix S6(2,1);
    static Matrix S7(2,1);
    static Matrix S8(2,1);
    static Matrix S9(2,1);

    static Matrix U0(2,1);
    static Matrix U1(2,1);

    if (start==0)
    {
        start=1;
        S0=*E;
    }

    if (start==1)
    {

        S9=S8;S8=S7;S7=S6;S6=S0;S5=S4;S4=S3;S3=S2;S2=S1;S1=S0;
        U1=U0+(.2)*((*E)-S9);
        U0=U1;
        S0=(*E);
    }
    (*Exit)=U1;
}

// -------------------------------------------------------------------
// filtro a variabili di stato
// -------------------------------------------------------------------
void FILTRO_STATO(TIPO time,Matrix* E,Matrix* Pos,Matrix* Vel,Matrix* Acc)
{
    TIPO ST=0.033;
    static TIPO time_puma=0;
    static Matrix X(3,1); // stato del filtro coordinata X
    static Matrix Y(3,1); // stato del filtro coordinata Y
    static Matrix A(3,3);
    static Matrix B(3,1);
    static start= 0;
    TIPO dtime=0;
    TIPO k1=7;
    TIPO k2=8;
```

```
    dtime=time-time_puma;

    if (dtime>ST)
    {
    time_puma=time;
    // --------  assegnamento matrice A -------------
    A(1,2)=-1;A(1,3)=-1;
    A(2,1)=k1*dtime;A(2,2)=1;
    A(3,2)=k2*dtime;A(3,3)=1;
    // --------- assegnamento matrice B --------------
    B(1,1)=1;

    X=A*X+(*E)(1,1)*B;
    Y=A*Y+(*E)(2,1)*B;
    }
    // aggiornamento delle uscite del sistema
    (*Pos)(1,1)=X(3,1);
    (*Pos)(1,2)=Y(3,1);
    (*Vel)(1,1)=k2*X(2,1);
    (*Vel)(1,2)=k2*Y(2,1);
    (*Acc)(1,1)=k1*k2*X(1,1);
    (*Acc)(1,2)=k1*k2*Y(1,1);
}

// ---------------------------------------------------------------------
// Implementazione digitale del filtro di Butterworth
// Filtro con Ft=0.2 * (Fc/2), i coefficienti sono ricavati dalla
// funzione Butter(ordine,Ft) di Matlab.
// ---------------------------------------------------------------------
void BUTTERWORTH(TIPO time,Matrix* E,Matrix* Exit)
{
    static TIPO start=0;
    static Matrix E1(2,1);
    static Matrix E2(2,1);
    static Matrix E3(2,1);
    static Matrix U1(2,1);
    static Matrix U2(2,1);
    static Matrix U3(2,1);
    TIPO dec=0.0001;
    TIPO cu[4]={1.0000,-2.7645,2.5559,-0.7900};
    TIPO ce[4]={0.0001823,0.0005469,0.0005469,0.0001823};
    TIPO ST=0.033; // sample time blocco
    TIPO dtime;
    static TIPO time_puma=0;
    dtime=time-time_puma;
    (*Exit)=-cu[1]*U1-cu[2]*U2-cu[3]*U3+ce[0]*(*E)+ce[1]*E1+ce[2]*E2+ce[3]*E3;
    //(*Exit)=-cu[1]*U1-cu[2]*U2+ce[0]*(*E)+ce[1]*E1+ce[2]*E2; // 3 erdine

if (dtime>ST)   // aggiorno le uscite
    {
    time_puma=time;
    U3=U2;    U2=U1;
```

```
    U1=(*Exit);
    E3=E2;E2=E1;E1=(*E);
    }
}

// ---------------------------------------------------------------
// Funzione non lineare che restituisce il parametro Delta
// indice di trasformazione del PIDNL
// ---------------------------------------------------------------
TIPO FNCT_NL(TIPO x)
{
   TIPO div=40;
   TIPO n;
   TIPO ris;

   div=20;
   n=3; // potenza polinomio
   ris=(pow((x/div),(2*n)))/(pow((x/div),(2*n)+1));
   return ris;
}



// ---------------------------------------------------------------
// Implementazione PID a tempo variabile
// ---------------------------------------------------------------
void PIDNL_TIME(double time,Matrix* B,TIPO K,TIPO Ti,TIPO Td,TIPO N,Matrix* E,Matrix* Exit,Matrix* delta)
{
    static double time_pd_hold=0;
           Matrix Uk0(2,1);
    static Matrix Uk1(2,1);
    static Matrix Uk2(2,1);
    static Matrix Ek1(2,1);
    static Matrix Ek2(2,1);
    Matrix Temp(2,1);

    Matrix Kbeta(2,2);    // costanti transitorie di calcolo.
    Matrix Kalfa(2,2);
    Matrix Kgamma(2,2);

    Matrix q0(2,2);  // coef. errore  presente
    Matrix q1(2,2);  // coef. errore di 1 istante passato
    Matrix q2(2,2);  // coef. errore di 2 istanti passati
    Matrix r2(2,2);  // coef. uscita di 2 istanti passati
    Matrix r1(2,2);  // coef. uscita di 1 istante passato.

    Matrix I(2,2);   // Matrice identita'
    I(1,1)=1;
    I(2,2)=1;

    TIPO T;   // tempo di campionamento

    if (time-time_pd_hold>=0.130)
```

```
    {
        T=time-time_pd_hold;
        time_pd_hold=time;

        Kalfa=T*(1/Ti)*I;
        Kgamma=(Td/(N*T+Td))*I;
        Kbeta=(N*Kgamma)*I;

        q0=K*(I+Kbeta);
        q1=-K*((*delta)+Kgamma-(*delta)*Kalfa+Kbeta+(*delta)*Kbeta);
        q2=K*(*delta)*(Kgamma-Kalfa*Kgamma+Kbeta);

        r1=Kgamma+(*delta)*I;
        r2=Kgamma*(*delta);

        Uk0=r1*Uk1-r2*Uk2+q0*(*E)+q1*Ek1+q2*Ek2;
        (*Exit)=(*B)*Uk0;  // trasformo l'uscita nello spazio robot
        Uk2=Uk1;
        Ek2=Ek1;
        Uk1=Uk0;
        Ek1=(*E);
        }
}
```

F.4 Codici dell'oggetto Matrix

F.4.1 File Matrice.h

```
#ifndef MATRIX_VERSION
#define MATRIX_VERSION   2.00

#include <stdio.h> // necessaria per il metodo Display
#include <math.h>  // vecessaria per il metodo precalib
#include <stdlib.h> // per malloc e free

#define TIPO  double     // Tipo di dato trattato dall'oggetto matrice

#define MATRIX_OK             0
#define ERROR_MATRIX_DIM      1     // una matrice ha dimensioni errate.
#define ERROR_MATRIX_REF      2  // Riferimenti non trovati.....

class Matrix
{

    int MatrixCol;
    int MatrixRow;
    TIPO* mat;

public:
```

```
    int MatrixError;         // variabile di rilevazione errore

    Matrix(int row,int col); // definisco una matrice
    ~Matrix(){ delete[] mat;};

    double determinante();        // ritorna il determiante della matrice
    display(char *stringa);        // visualizza il contenuto della matrice

    friend Matrix precalib(Matrix& matrob,Matrix& matime,double fatYX);
    friend Matrix operator+(const Matrix&,const Matrix&);
    friend Matrix operator-(const Matrix&,const Matrix&);
    friend Matrix operator*(const Matrix&,const Matrix&);
    friend Matrix operator*(TIPO num,const Matrix& mat2);
    friend Matrix INV(Matrix &mat1);
    friend Matrix TRASP(Matrix &mat1);
    friend double DET(Matrix &mat1);

    TIPO&    operator ()(int x, int y);  // ritorna/assegna l'elemento x,y
    Matrix&    operator =(TIPO* dati);
    Matrix& operator =(Matrix&);

private:
    double CalcDet(TIPO*matr,int dim);
    Sot(TIPO* mat,TIPO *ridotto,int dim,int posx,int posy);

};

#endif
```

F.4.2 File Matrice.cpp

```
#include "matrice.h"

// ------------------------------------------------------------
// Costruttore oggetto matrice....
// inizializza le strutture dati dell'implementazione oggetto
// ------------------------------------------------------------
Matrix::Matrix(int dimy, int dimx)
{
    MatrixError=MATRIX_OK;
    if (dimx<0 || dimy<0)
    {
        MatrixError|= ERROR_MATRIX_DIM;
    }
    MatrixCol=dimx;
    MatrixRow=dimy;
    mat=new TIPO[dimy*dimx];
    for (int c=0;c<dimx*dimy;c++) mat[c]=0;
}

// ------------------------------------------------------
//    Operatore di assegnaento valori alla matrice
```

```
//  tramite un puntatore ad un vettore di elementi TIPO
//  Precedentemente definito.
// -------------------------------------------------------
Matrix& Matrix::operator =(TIPO* dati)
{
    for (int c=0;c<MatrixRow*MatrixCol;c++) mat[c]=*(dati+c);
    return *this;
}

// -------------------------------------------------------
//    Assegna il valore della matrice passata come paramentro
//  se la matrice M  pi piccola dell'oggetto matrice chiamente
//  viene  copiata in alto a sinistra!!!!
// -------------------------------------------------------
Matrix& Matrix::operator =(Matrix& m)
{
    if ((m.MatrixCol>MatrixCol) ||
        (m.MatrixRow>MatrixRow))
    {
        MatrixError|=ERROR_MATRIX_DIM;
        return *this;
    }
    for (int col=0;col<m.MatrixCol;col++)
        for (int row=0;row<m.MatrixRow;row++)
            mat[col+row*MatrixCol]=m.mat[col+row*m.MatrixCol];
    return *this;
}

// -------------------------------------------------------
//    Ritorna l-value dell'elemento (riga,col) dell'oggetto
// -------------------------------------------------------
TIPO& Matrix::operator ()(int riga,int col)
{
    return( mat[col-1+(riga-1)*MatrixCol]);
}

// -------------------------------------------------------
// Stampa il contenuto dell'oggetto
// -------------------------------------------------------
Matrix::display(char* stringa)
{
    printf("%s\n",stringa);
    int c=0;
    for (int row=0;row<MatrixRow;row++)
    {
        for (int col=0;col<MatrixCol;col++)
        {
                printf(" %.3f ",mat[c]);
                c++;
        }
    printf("\n");
    }
}
```

```
// -------------------------------------------------------
//    Funzione privata del calcolo del determinante.
// -------------------------------------------------------
double Matrix::CalcDet(TIPO*matr,int dim)
{
    TIPO result=0;
    TIPO *submatrix;
    if (dim==1) return (*matr);
    if (dim==2) return ((*matr)*(*(matr+3))-(*(matr+1))*(*(matr+2)));
    int row=1;
    TIPO temp=0;
    TIPO sub=0;

    submatrix=(TIPO*)malloc((dim-1)*(dim-1)*sizeof(TIPO));
        for(int col=1;col<=dim;col++)
        {
        Sot(mat,submatrix,dim,col,row);
        sub=CalcDet(submatrix,dim-1);
        temp=0;
        temp=(*(mat+col-1))*sub*pow(-1,(col+row));
        result=result+temp;        }
    free(submatrix);
    return result;
}

// -------------------------------------------------------------------
// procedura di estrazione sottomatrice con esclusione
// della linea posy e della colonna posx.
// -------------------------------------------------------------------
Matrix::Sot(TIPO* mat,TIPO *ridotto,int dim,int posx,int posy)
{
    int x=0;
    for (int row=1;row<=dim;row++)
        for(int col=1;col<=dim;col++)
        {
         if ((col!=posx) && (row!=posy))
         {
             *(ridotto+x)=*(mat+(col-1)+(row-1)*dim);
             x=x+1;
         }
        }
}

// -------------------------------------------------------------------
// ritorna un puntatore ad una matrice "somma di due matrici"
// caso in cui opero con un risultato parziale
// -------------------------------------------------------------------
Matrix operator+(const Matrix& mat1,const Matrix& mat2)
{
    int dim1x=mat1.MatrixCol;
    int dim1y=mat1.MatrixRow;
    int dim2x=mat2.MatrixCol;
```

```
    int dim2y=mat2.MatrixRow;

    // creo una matrice temporanea in cui metto il risultato
    Matrix *ris;
    ris= new Matrix(mat1.MatrixRow,mat1.MatrixCol);
    if ((dim1x!=dim2x) ||(dim1y!=dim2y))
    {
        (*ris).MatrixError|=ERROR_MATRIX_DIM;
        return (*ris);
    }
    for(int r1=1;r1<=dim1y;r1++)
    {
        for(int c1=1;c1<=dim1x;c1++)
        {
          (*ris).mat[(c1-1)+(r1-1)*(*ris).MatrixCol]=mat1.mat[(c1-1)+(r1-1)*mat1.MatrixCol]+
                                                mat2.mat[(c1-1)+(r1-1)*mat2.MatrixCol];
        }
    }
    return (*ris);
}

// ------------------------------------------------------------------
//  ritorna un puntatore ad una matrice "prodotto di due matrici"
// ------------------------------------------------------------------
Matrix operator *(const Matrix& mat1,const Matrix& mat2)
{
    int dim1x=mat1.MatrixCol;
    int dim1y=mat1.MatrixRow;
    int dim2x=mat2.MatrixCol;
    int dim2y=mat2.MatrixRow;

    TIPO temp;
    // creo una matrice temporanea in cui metto il risultato
    Matrix *ris;
    ris= new Matrix(mat1.MatrixRow,mat2.MatrixCol);
    if (dim1x!=dim2y)
    {
        (*ris).MatrixError|=ERROR_MATRIX_DIM;
        printf("Errrrrrrr*********************************");
        return *ris;
    }
    for(int r1=1;r1<=dim1y;r1++)
    {
        for(int c1=1;c1<=dim2x;c1++)
        {
            temp=0;
            for(int x=1;x<=dim1x;x++)
            {
            temp+=mat1.mat[(x-1)+(r1-1)*mat1.MatrixCol]*
                  mat2.mat[(c1-1)+(x-1)*mat2.MatrixCol];
            }
            (*ris).mat[(c1-1)+(r1-1)*(*ris).MatrixCol]=temp;
        }
```

```
    }
    return (*ris);
}

// ---------------------------------------------------------------------
// ritorna un puntatore ad una matrice "sottrazione di due matrici"
// caso in cui opero con un risultato parziale
// ---------------------------------------------------------------------
Matrix operator-(const Matrix& mat1,const Matrix& mat2)
{
    int dim1x=mat1.MatrixCol;
    int dim1y=mat1.MatrixRow;
    int dim2x=mat2.MatrixCol;
    int dim2y=mat2.MatrixRow;

    // creo una matrice temporanea in cui metto il risultato
    Matrix *ris;
    ris= new Matrix(mat1.MatrixRow,mat1.MatrixCol);
    if ((dim1x!=dim2x) || (dim1y!=dim2y))
    {
        (*ris).MatrixError|=ERROR_MATRIX_DIM;
        return (*ris);
    }
    for(int r1=1;r1<=dim1y;r1++)
    {
        for(int c1=1;c1<=dim1x;c1++)
        {
          (*ris).mat[(c1-1)+(r1-1)*(*ris).MatrixCol]=mat1.mat[(c1-1)+(r1-1)*mat1.MatrixCol]-
                                                mat2.mat[(c1-1)+(r1-1)*mat2.MatrixCol];
        }
    }
    return (*ris);
}

// ---------------------------------------------------------------------
//    Associa i vettori immagine a quelli dello spazio robot.
// ---------------------------------------------------------------------
Matrix precalib(Matrix& matrob,Matrix& matime,double fatYX)
{
    Matrix *ris;
    ris= new Matrix(3,3);

    double distRobot[3][2];  // distanze tra i punti nello spazio Robot e Image
    double rappRobot[3][2];  // rapporti delle ditanze normanlizzate
    double distImage[3][2];
    double rappImage[3][2];
    double minRobot;         // distanza minima
    double minImage;
    bool first_min=true;
    int c1,c0,c,cc=0;

    double k1=.96,k2=1.04;
```

```
// Creo un vettore di coppie delle distanze sia per Image che per Robot
// ogni punto infatti viene identificato dalla distanza
// dai rimanenti (2) punti.
for ( c0=0;c0<3;c0++)
{
 cc=0;
 for ( c1=0;c1<3;c1++)
 {
    if (c0!=c1)
    {
        distRobot[c0][cc]=sqrt(pow(matrob.mat[c1]-matrob.mat[c0],2)+
                           pow(matrob.mat[c1+3]-
                               matrob.mat[c0+3],2)+
                           pow(matrob.mat[c1+6]-
                               matrob.mat[c0+6],2));

        distImage[c0][cc]=sqrt(pow(matime.mat[c1]-matime.mat[c0],2)+
                           pow(matime.mat[c1+3]-
                               matime.mat[c0+3],2));
        //printf("distanza  ROBOT vet %i da vet%i= %.2f\n",c1+1,c0+1,distRobot[c0][cc]);
        //printf("distanza  IMAGE vet %i da vet%i= %.2f\n",c1+1,c0+1,distImage[c0][cc]);


        if (distRobot[c0][cc]<minRobot) minRobot=distRobot[c0][cc];
        if (distImage[c0][cc]<minImage) minImage=distImage[c0][cc];
        if (first_min)
        {
            first_min=false;
            minImage=distImage[c0][cc];
            minRobot=distRobot[c0][cc];

        }
    cc++;
    }

 }
}

// Normalizzo le distanze rispetto la distanza minima
// sia per i punti immagine che per i punti nello spazio reale.
for ( c=0;c<3;c++)
{
    rappRobot[c][0]=distRobot[c][0]/minRobot;
    rappRobot[c][1]=distRobot[c][1]/minRobot;
    rappImage[c][0]=distImage[c][0]/minImage;
    rappImage[c][1]=distImage[c][1]/minImage;
}

// ora trovo le coppie delle distanze normalizzate simili....
// entro una certa tolleranza indicata da k1 e k2....
c=0;
    for (c1=0;c1<3;c1++)
        for (c0=0;c0<3;c0++)
```

```
            {
                    if (((rappRobot[c1][0]>(k1*rappImage[c0][0]) &&
                         (rappRobot[c1][0]<(k2*rappImage[c0][0]))) &&
                         (rappRobot[c1][1]>(k1*rappImage[c0][1]) &&
                         (rappRobot[c1][1]<(k2*rappImage[c0][1])))) ||
                        ((rappRobot[c1][0]>(k1*rappImage[c0][1]) &&
                         (rappRobot[c1][0]<(k2*rappImage[c0][1]))) &&
                         (rappRobot[c1][1]>(k1*rappImage[c0][0]) &&
                         (rappRobot[c1][1]<(k2*rappImage[c0][0])))))
                        {
                        (*ris).mat[c1]=matime.mat[c0];
                        (*ris).mat[c1+3]=matime.mat[c0+3];
                        (*ris).mat[c1+6]=matime.mat[c0+6];
                        c++;
                    }
            }
    if (c!=3)    (*ris).MatrixError|=ERROR_MATRIX_REF;
    return (*ris);
}

// ------------------------------------------------------------------
//  ritorna un puntatore ad una matrice "prodotto scalare per matrice"
// ------------------------------------------------------------------
Matrix operator *(TIPO num,const Matrix& mat2)
{
    int dim2x=mat2.MatrixCol;
    int dim2y=mat2.MatrixRow;
    TIPO temp;

    // creo una matrice temporanea in cui metto il risultato
    Matrix *ris;
    ris= new Matrix(mat2.MatrixRow,mat2.MatrixCol);
        for(int cy=1;cy<=dim2y;cy++)
        {
            temp=0;
            for(int cx=1;cx<=dim2x;cx++)
            {
            temp=num*mat2.mat[(cx-1)+(cy-1)*mat2.MatrixCol];
            (*ris).mat[(cx-1)+(cy-1)*(*ris).MatrixCol]=temp;
            }
        }
    return (*ris);
}

// ------------------------------------------------------------------
// Calcolo inversa con utilizzo aggiunto classico.
// ------------------------------------------------------------------
Matrix INV(Matrix &mat1)
{
    TIPO* submatrix;

    // creo una matrice temporanea in cui metto il risultato
    Matrix *ris;
```

```
    ris= new Matrix(mat1.MatrixRow,mat1.MatrixCol);
    double deter;
    int dim=mat1.MatrixCol;
    if (mat1.MatrixError!=MATRIX_OK)
    if ((dim!=mat1.MatrixRow))
    {
        (*ris).MatrixError|=ERROR_MATRIX_DIM;
        return (*ris);
    }

    deter=mat1.CalcDet(mat1.mat,mat1.MatrixRow);

    for (int row=1;row<=dim;row++)
        for(int col=1;col<=dim;col++)
        {
            // creo la matrice trasposta dei cofattori
            submatrix=(TIPO*)malloc((dim-1)*(dim-1)*sizeof(TIPO));
            mat1.Sot(mat1.mat,submatrix,dim,col,row);
        (*ris).mat[(row-1)+(col-1)*dim]=pow(-1,col+row)*mat1.CalcDet(submatrix,dim-1)/deter;
        free(submatrix);
        }

    return (*ris);
}

// -------------------------------------------------------
//    Assegna la trasposta della matrice passata come paramentro
// -------------------------------------------------------
Matrix TRASP(Matrix &mat1)
{

// creo una matrice temporanea in cui metto il risultato
    Matrix *ris;
    int  colonne=mat1.MatrixCol;
    int     righe=mat1.MatrixRow;
    ris= new Matrix(righe,colonne);

    for (int col=0;col<colonne;col++)
        for (int row=0;row<righe;row++)
            (*ris).mat[col+row*colonne]=mat1.mat[row+col*righe];
return (*ris);
}

// -------------------------------------------------------
//    Ritorna il determinante della matrice
// -------------------------------------------------------
double DET(Matrix &mat1)
{
    if (mat1.MatrixRow!=mat1.MatrixCol) mat1.MatrixError|=ERROR_MATRIX_DIM;
    return(mat1.CalcDet(mat1.mat,mat1.MatrixRow));
}
```

Bibliografia

[1] A. Silberschatz P.B. Galvin *"Sistemi Operativi"*, Addison-Wesley -1998.

[2] J. Richter *"Programming Applications for Microsoft Windows"*, Microsoft Press - 2000.

[3] B. Stroustrup. *"Il Linguaggio C++"*, Addison-Wesley Italia Editoriale 1997.

[4] Unival Robot Controller Programming Manual, *"Users's Guide to VAL II"*. Dicembre 1993.

[5] Unival Robot Controller, *"19 Inch Rack Mount Equipment Manual"*, January 1994.

[6] Unival Robot Controller Arm Manual, *"Users's Guide to Robot PUMA 260"*. Giugno 1993.

[7] Matrox, *"MIL/ MIL-lite ver 6.0 Board-Specific Notes "*. Manual February 24, 1999.

[8] Matrox, *"MIL–lite ver 6.0 User Guide and Command Reference"*. Manual February 24, 1999.

[9] Rafael C. Gonzalez Richard E. Woods, *"Digital Image Processing"*,Addison-Wesley Publishing Company - 1993.

[10] R. Cipolla A. Gee, *"Computer Vision and Robotics"*, University of Cambridge Module i12. October 1998.

[11] H. Freeman, *"Machine Vision for Inspection and Measurement"*: *"Pose Estimation from Corresponding Point Data"* R. Haralick,H. Joo, C. Lee, X. Zhuang, V.Vaidya and M. Kim. University of Washington - 1989.

[12] M. Gozzi, *"Riconoscimento di Manufatti Industriali con Tecniche di Visione Artificiale."*. Universitá degli Studi di Ferrara– A.A. 1998-1999.

[13] R.L. Carceroni C, M.Brown, *"Numerical Methods for Model-Based Pose Recovery"*,the University of Rochester, New York August 1997.

[14] L. Quan Z.Lan *"Linear N-Point Camera Pose Determination"*, IEEE Transactions on Pattern Analysis and Machine Intelligense. July 1999.

[15] A.Ciuffoli, *"Manipolazione Robotica Mediante Elaborazione di Immagine"*. Universitá degli Studi di Ferrara – Dicembre 1998.

[16] D.Vigorelli, *"Esperimenti di Manipolazione Robotica Tramite Feedback Visivo"*. Universitá degli Studi di Ferrara – Dicembre 2000.

[17] J.Michael Brady, cap. 7 *"Intelligenza Artificiale e Robotica"* da *"Intelligenza Artificiale Principi-Strumenti-Applicazioni-Sviluppi"* a cura di M. Yazdani. Hoepli 1994.

[18] V. Tagliasco *"Mente e Corpo nei Robot"* da *"L'Automa Spirituale, Menti, Cervelli e Computer"* a cura di G. Giorello e P. Stratta, Laterza 1991.

[19] S. Hutchinson G. Hager P. Corke *"A Tutorial on Visual Servo Control"* IEEE Transactions on Robotics and Automation, vol 12, October 1996.

[20] C. Melchiorri dispense delle lezioni del corso di *"Robotica Industriale"*, Università di Bologna.

[21] Peter I. Corke Seth A. Hutchinson, *"Real-Time Vision, Tracking and Control"*. Proc. IEEE Int. Conf. Robotics and Automation 2000.

[22] P.I. Corke, *"Dynamics of Visual Control"* CSIRO Division of Manufacturing Technology, April 1994.

[23] M.Vincze, *"Dynamics and System Preformace of Visual Servoing"*. Proc. IEEE Int. Conf. Robotics and Automation 2000.

[24] Grégory Flandin Franqois Chaumette Eric Marchand, "*Eye-in-hand / Eye-to-hand Cooperation for Visual Servoing*". Proc. IEEE Int. Conf. Robotics and Automation 2000.

[25] G. Marro *"Controlli Automatici"*- Quarta edizione, Zanichelli 1992.

[26] C. Bonivento C. Melchiorri R. Zanassi *"Sistemi di Controllo Digitale"*. Progetto Leonardo-Bologna 1995.

[27] Claudio Melchiorri *"Traiettorie per Azionamenti Elettrici"*. Progetto Leonardo-Bologna.

[28] M. Tibaldi *"Progetto di Sistemi di Controllo"*. Pitagora Editrice Bologna 1995.

[29] B. Shahian M.Hassul *"Control System Design using Matlab"*. Prentice Hall 1993.

[30] *"Simulink Dynamic System Simulation for Matlab"*. The MathWorks Inc. January 1997.

[31] P. Blaha, P. Pivonka *"Intelligent Corrector for the System with Time Delay"*.