Ferrari Marco – Zucca Riccardo

Saracinescu

Dipartimento di Elettronica per l'Automazione
Università di Brescia
Via Branze 38 – I25123 Brescia

Capitolo 1

Introduzione

1.1 Cos'è Saracinescu

Saracinescu è un robot portiere realizzato nel laboratorio di Robotica Avanzata sotto la supervisione del professor Riccardo Cassinis e del dottor Alessandro Rizzi, per la Robocup '98.

Esso è stato realizzato a partire dal modello standard *Pioneer 1 Mobile Robot* prodotto dalla *ActiveMedia Inc* a cui è stato aggiunto un sistema di visione, un Personal Computer e potenziati i motori per le due ruote motrici.

1.2 Hardware

Per attivare Saracinescu vi sono due deviatori posti nella parte posteriore del robot; prima bisogna commutare il deviatore nero, per l'alimentazione del robot e poi l'altro argentato per quella del PC. L'ordine con cui si devono attivare le alimentazioni non possono essere assolutamente invertite.

Prima dell'accensione, se si vuole che la batteria non si esaurisca nell'arco di poche decine di minuti si può inserire nel connettore posto a fianco dei deviatori la presa d'uscita del carica batterie.

E' buona cosa non lasciare scaricare mai completamente la batteria all'interno di Saracinescu e fare attenzione nella sostituzione della medesima a non far toccare parti metalliche del robot alla scheda madre.

Un possibile problema che si può riscontrare se il PC è stato spento per molto tempo è l'esaurimento della batteria sulla scheda madre che deve essere sostituita.

1.3 Software

Per poter lavorare con Saracinescu ci si può collegare direttamente con la scheda madre avendo a disposizione: monitor, tastiera e mouse. Durante lo sviluppo del software, si sono riscontrati dei problemi nel caricamento del

ServerX, dovuto a conflitti creati dal mouse PS2; per ovviare a tali problemi si è proceduto disabilitando il mouse ed al caricamento automatico di una shell *xterm* in modo da poter caricare le applicazioni di taratura e visione. Nel caso si voglia eseguire il programma di movimentazione senza utilizzare il *ServerX* è possibile eseguire l'alias *ystart* che permette di indirizzare la grafica in Ram, senza quindi mostrare a video alcunchè; quindi si lancia l'eseguibile *saracinescu* ed al termine si possono ripristinare le condizioni normali lanciando *ystop*.

Un ulteriore modo di utilizzo di Saracinescu è il collegamento in rete; questa ultima soluzione è anche il modo migliore per provare i programmi di movimentazione, ma non permette il trasferimento della grafica sul PC con cui ci si collega e di conseguenza non si possono vedere le immagini catturate della telecamera.

Per lavorare con la rete si devono digitare i seguenti comandi una volta attivata la finestra *xterm*:

-)xhost +saracinescu

-)rsh saracinescu

-)Fare il login di saracinescu ("*saracinescu*" e passwork)

-)export DISPLAY=[nome-PC]:0.0

Il software è realizzato in linguaggio C per l'ambiente *Saphira* ed è separabile in due categorie: Taratura e Movimentazione.

Alla prima categoria rispondono i file che si trovano nella directory *camview-1.1* e alla seconda quelli della directory *saracinescu*.

Se si vuol far funzionare il robot, prima di farlo muovere bisogna tarare la visione, che permette a Saracinescu di distinguere gli oggetti del suo "mondo", cioè del campo di calcio. Per far questo si deve usare il programma *set_colori* che si trova nella directory */camview-1.1/taratura/* e si deve tarare le distanze con la tabella contenuta nel file di testo *Tab_dist* che assegna, in base al numero di pixel riconosciuti appartenenti ad un oggetto, la sua distanza.

Questa taratura è necessaria perché a seconda di dove si fa giocare il robot, i colori del campo possono cambiare anche di poco e Saracinescu potrebbe non riconoscere più gli oggetti. Inoltre persino nello stesso campo ma con illuminazioni differenti Saracinescu molto probabilmente perde la capacità di distinguere gli oggetti. Le distanze sono ancor più soggette ad errore e per la scarsa affidabilità sono state usate nel programma di movimentazione il meno possibile.

Il file che permette la movimentazione del robot è *saracinescu.c* che si trova nella directory */saracinescu/src/*, e per compilarlo bisogna lanciare il MakeFile nella directory */saracinescu/* "*make -f make_sarac saracinescu*".

Il programma *saracinescu.c* è realizzato seguendo un approccio di programmazione a *Tasks*; il quale consiste nell'esecuzione "parallela" di processi, che il sistema operativo del Pioneer esegue ogni 10ms e per questo il programmatore deve garantire che l'esecuzione di ogni singolo task abbia una durata inferiore a 10ms. Per osservare se non si è verificata questa esigenza di può osservare la percentuale di utilizzo della C.P.U., visualizzata nella finestra di Saphira, la quale non deve superare il 100%.

Capitolo 2
Taratura

Come è già stato detto nel paragrafo 1.3, prima di far muovere il robot, bisogna assicurarsi che veda correttamente tutti gli oggetti del campo che sono: la palla, i due pali e i due bordi del campo vicini ai pali.

Il programma *set_colori* visualizza in basso a sinistra il nome dell'oggetto che si sta tarando.

Per tarare un oggetto bisogna portare il quadrato al centro dello schermo sull'oggetto premendo con il mouse il tasto della direzione voluta e la barra di spazio per spostarlo. Si può anche cambiare la dimensione del quadrato (tasto **+xy** per aumentarla e tasto **–xy** per diminuirla). Portato il quadrato sull'oggetto si deve premere **CAP**; a questo punto l'oggetto si colora di punti neri, che rappresentano i pixel riconosciuti appartenenti all'oggetto. Si possono variare le componenti dei tre colori (red, green e blu) con i rispettivi tasti (**R+, R-, G+, G-, B+, B-**) per permettere un miglior riconoscimento. Finita la taratura di un oggetto si deve premere di nuovo **CAP** per poi passare all'oggetto successivo premendo **Oggetto**.

Terminata la taratura di tutti gli oggetti si deve premere **Uscita**.

Il programma *set_colori* permette di tarare anche un solo oggetto, mantenendo la taratura degli altri.

Per verificare la taratura e vedere cosa Saracinescu riconosce realmente in campo, si può usare il programma *camview* nella directory *camview-1.1/* ; questo programma colora i pixel riconosciuti come appartenenti ad un oggetto in modo differente a seconda dell'oggetto:

-) quadrati blu per i pixel riconosciuti come appartenenti alla palla

-) quadrato bianco per la posizione stimata della palla (baricentro del blob di punti)

-) punti verdi per i pixel riconosciuti come appartenenti al palo 1

-) quadrato verde per la posizione stimata del palo 1 (baricentro del blob di punti)

-) punti rossi per i pixel riconosciuti come appartenenti al palo 2

-) quadrato rosso per la posizione stimata del palo 2 (baricentro del blob di punti)

Terminata la taratura dei colori si devono tarare le distanze. Questa operazione si effettua posizionando la palla sulla retta dei 90° del robot, partendo dalla distanza di 40 cm (palla appoggiata alla paletta) fino a 250 cm con passo di 10 cm; quindi si effettuano 22 rilevamenti di distanza.
Ad ogni rilevamento, si pone la palla alla distanza voluta e si annota nella riga corrispondente nella tabella del file *Tab_dist*, la distanza in pixel fornita dal programma *camview*, che viene visualizzata nella shell di xterm da cui si è lanciato il programma. I primi 3 valori nella tabella sono posti a zero in quanto non si può mettere il pallone ad una distanza inferiore di 40 cm.

Capitolo 3
Movimentazione

La movimentazione è realizzata, come già detto nel paragrafo 1.3 dal programma *saracinescu.c*. All'esecuzione si attiva il processo *vai_in_porta* che permette a Saracinescu di posizionarsi al centro della porta e alla distanza voluta, attivando poi il processo *Para* che elimina *vai_in_porta*.

Il processo *vai_in_porta* prima si preoccupa di posizionare Saracinescu parallelo alla porta, richiamando più volte la procedura *StaiTH*, poi lo muove al centro della porta e qui calcola la distanza dalla porta (conoscendo l'angolo tra il robot e il palo destro e la lunghezza della porta). Quindi si avvicina alla porta di metà della distanza calcolata. A questo punto ripete i passaggi di prima: porsi parallelo alla porta, andare al centro, calcolarsi la distanza dalla porta; e infine si porta alla distanza voluta dalla porta.

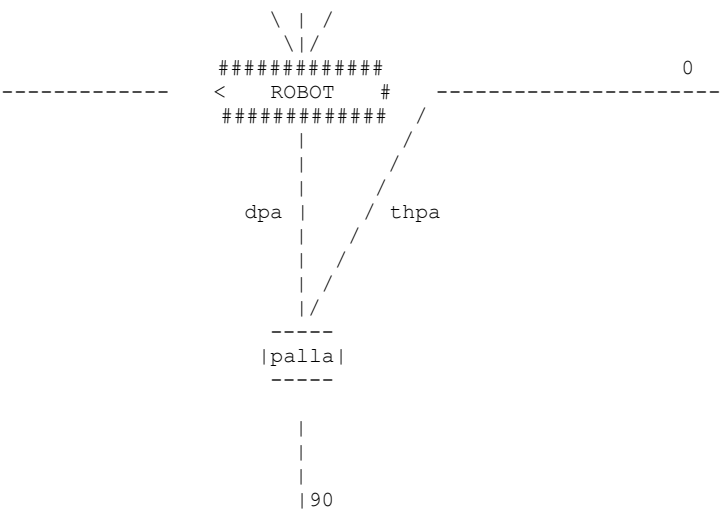
La movimentazione gestita dal processo *Vai_in_porta* è stata pensata in due parti quasi identiche al fine di migliorarne il posizionamento finale, perché più Saracinescu è distante dalla porta più le misure degli angoli e delle distanze sono imprecise.

Il processo *Para* si occupa del corretto posizionamento del robot, tra la palla e la porta. Al fine di eliminare sgradevoli oscillazioni che si verificano quando la palla è vicina al robot (l'angolo rispetto alla palla cambia molto più rapidamente a fronte di piccoli spostamenti della medesima), si sono utilizzate tre diverse curve di velocità, in funzione della distanza della palla dal robot. Inoltre si sono effettuati dei controlli sulla posizione di Saracinescu in modo che se supera il palo, indietreggia fino a riportarsi sul palo stesso.; se per qualsiasi motivo il robot non vede la palla, dopo un periodo di tempo prefissato, esso si posiziona al centro della porta.

L'ultima versione del programma *saracinescu.c* è riportata qui di seguito:

```
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

   Standalone client per il controllo di un robot in ambiente Saphira
            Programma per la movimentazione di SARACINESCU
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

#include <math.h>
#include <malloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include "saphira.h"
#include "saracinescu.h"
#include "grabber.h"

//define per task

#define END       100

//define per priorita' temporali

#define PRIORITA_01      1   // Indica che il task verra' fatto ogni ciclo di Saphira
#define PRIORITA_02      2   // Indica che il task verra' fatto ogni 2 cicli di Saphira
#define PRIORITA_10      10  // Indica che il task verra' fatto ogni 10 cicli di Saphira

extern unsigned char *dataL;

 int NumPali = 0;      // numero di pali che vede

//variabili globali per task

 const float Vmax = 800.0;

 int  non_vedo_palla=-1,        //conta quante volte non vede la palla (se la vede e' 0)
      non_vedo_palo1=-1,        //conta quante volte non vede il palo 1 (se lo vede e' 0)
      non_vedo_palo2=-1;        //conta quante volte non vede il palo 2 (se lo vede e' 0)

 typedef struct record_dati_cam {
     float  thpa;      //  <-- input dai dati della telecamera (angolo con la palla)
     float  thpo1;     //  <-- input dai dati della telecamera (angolo col palo 1)
     float  thpo2;     //  <-- input dai dati della telecamera (angolo col palo2)
     float  thpo;      //  angolo con la porta calcolato dalla procedura SetInfo
     float  th_papo;   //  angolo tra la palla e la porta calcolata dalla procedura
SetInfo
     float  dpo1;      //  <-- input dai dati della telecamera (distanza dal palo 1)
     float  dpo2;      //  <-- input dai dati della telecamera (distanza dal palo 2)
     float  dpo;       //  distanza dalla porta calcolata dalla procedura SetInfo
     float  dpa;       //  <-- input dai dati della telecamera
     float  a1;        //  angolo sul palo 1 del triangolo palo1-palo2-robot
     float  a2;        //  angolo sul palo 2 del triangolo palo1-palo2-robot
 } dati_cam;

 dati_cam Robot;            // Struttura per caricare i dati della telecamera




/********  SCHEMA DEI CAMPI DELLA STRUTTURA DATI_CAM

                        PALO 2                    PALO 1
                          |---------------------|
                          \)a2      |      a1(/
                           \        |dpo     /
                            \       |       /
                             \      |      /
                        dpo2  \     |     / dpo1
                               \    |    /
                                \   |   /
                                 \  |  /
```




```
**************************/

unsigned int conta_ciclo=0;     // var per memorizzare il tempo per le priorita'
int okCam = 0;

#define LARGHEZZA_PORTA   1560    // Larghezza della porta in mm
#define DIST_VOLUTA_PORTA 700     // Distanza posizione Saracinescu dalla porta
#define VELOCITA_SPOST 100        // Velocità degli spostamenti verso la porta
#define TOLLERANZA 3              // Angolo minimo di toll. per i posizionamenti


int alfa=0,
    goto_case=1,
    dist_palo1=0,
    dist_palo2=0,
    dist_porta=0,
    pausa=0,
    fase=1,
    ang_a1=0,
    ang_a2=0,
    verso_spostamento=0;

float  ang_palo1=0,
       ang_palo2=0;







// ----------------------------
//    LE FUNZIONI PER I TASK
// ----------------------------

float deg2rad (float deg)  // restituisce un angolo in radianti se gli e lo si passa in
gradi
{
  float rad;

  rad = deg * PI / 180;
  return rad;
}

float rad2deg (float rad) // restituisce un angolo in gradi se gli e lo si passa in
radianti
{
  float deg;
```

```
  deg = rad * 180 / PI;
  return deg;
}


// ++++++++++++++++++++++++++ SetInfo +++++++++++++++++++++++++++++++

void SetInfo (void) // riempie la struttura di Robot e calcola i dati che mancano
{
 float thpa,dpa,thpo1,thpo2,dpo1,dpo2;
 int PallaCE;
 static unsigned int alterna; // permette alternare la ricerca della palla con qualla
dei pali

 NumPali = 0;
 alterna++;
 if (alterna == 0) alterna++;
 if ( (alterna%2) == 0)
   {
    trova_porta (&thpo1,&thpo2,&dpo1,&dpo2); // procedura esterna
    if (thpo1 != -1)
      {
       Robot.thpo1 = rad2deg(thpo1);
       Robot.dpo1  = 10*dpo1;
       non_vedo_palo1 = 0;
      }
    else
      {
       non_vedo_palo1 ++;
      }
    if (thpo2 != -1)
      {
       Robot.thpo2 = rad2deg(thpo2);
       Robot.dpo2  = 10*dpo2;
       non_vedo_palo2 = 0;
      }
    else
      {
       non_vedo_palo2 ++;
      }
   }







 if ((alterna%2)== 0)
   {
    PallaCE = trova_palla(&thpa,&dpa);  //procedura esterna
    if (PallaCE == -1)
      {
       non_vedo_palla++;
      }
    else
     {
      non_vedo_palla = 0.0;
      Robot.thpa = rad2deg(thpa);
      Robot.dpa  = 10*dpa;
     }
   }


 if ( (non_vedo_palo2 <= 0) && (non_vedo_palo1 <=0))     // se vedo entrambi i pali
 {
  NumPali = 2;
```

```
  // calcolo angolo con la porta
  if (Robot.thpo2<Robot.thpo1)  Robot.thpo = Robot.thpo1 - Robot.thpo2;
  else Robot.thpo = 360 - Robot.thpo2 + Robot.thpo1;

  Robot.a1       = Robot.dpo2 * (180-Robot.thpo) / (Robot.dpo1 + Robot.dpo2) ;
  Robot.a2       = Robot.dpo1 * (180-Robot.thpo) / (Robot.dpo1 + Robot.dpo2);
  Robot.dpo      = 0.5 * (Robot.dpo1 * sin (deg2rad(Robot.a1)) + Robot.dpo2 * sin
(deg2rad(Robot.a2)) );

  if (Robot.dpo1 > Robot.dpo2)
    {
     if (Robot.thpo2<Robot.thpa) Robot.th_papo = 360 - Robot.thpa + Robot.thpo2 +
Robot.thpo / 2;
     else  Robot.th_papo = - Robot.thpa + Robot.thpo2 + Robot.thpo / 2;
    }
  else
    {
     if (Robot.thpo1<Robot.thpa) Robot.th_papo = 360 - Robot.thpa + Robot.thpo1 -
Robot.thpo / 2;
     else  Robot.th_papo = - Robot.thpa + Robot.thpo1 - Robot.thpo / 2;
    }
  } // end if vedo entrambi i pali
 else
  {
   NumPali = ((non_vedo_palo1 > 0) && (non_vedo_palo2 > 0) ) ?  0 : 1;  // se vedo 0
pali NumPali=0, se no NumPali=1
  }

 sfMessage ("============================================================");
 sfSMessage("Ang.: pa=%d , po1=%d , po2=%d , po=%d , a1=%d , a2=%d , papo=%d",

(int)Robot.thpa,(int)Robot.thpo1,(int)Robot.thpo2,(int)Robot.thpo,(int)Robot.a1,(int)Rob
ot.a2,(int)Robot.th_papo);
 sfSMessage("Dist.: pa=%d , po1=%d , po2=%d , po=%d , Dark[0]=%d , Dark[1]=%d ,
Dark[2]=%d",(int)Robot.dpa,(int)Robot.dpo1,(int)Robot.dpo2,(int)Robot.dpo,(int)non_vedo_
palla,(int)non_vedo_palo1,(int)non_vedo_palo2);
 sfMessage ("============================================================");


} //end SetInfo




// ++++++++++++++++++++++++++ Vel_Ins_Pal ++++++++++++

float Vel_Ins_Pal(float ang_palla) // calcola la velocita' in base all'angolo con la
palla
{
 float xsoglia = 12.5;      // Angolo dopo il quale vado a Vmax
 float velocita;
 float a1,a2;
 float xpend1 = 0.13;       // Pendenza della curva di velocità media

 a1 = deg2rad (Robot.a1);
 a2 = deg2rad (Robot.a2);

 if ( (Robot.dpa<800) && (Robot.thpa>60) && (Robot.thpa<150) )
    {   // abbassa le pendenze della curva di velocita' se la palla e' vicina
     xpend1=0.05;
    }
 else if  (Robot.dpa>1500 )
    {   // alza le pendenze della curva di velocita' se la palla e' lontana
     xpend1=0.30;
    }

// calcolo la velocita' in base alla curva ang_palla/velocita
  if (ang_palla<0)
```

```
      velocita =  -(Vmax/PI*atan(xpend1*(ang_palla+xsoglia)))- Vmax*atan(-
(xpend1)*xsoglia)/PI;
  else  velocita =  -(Vmax/PI*atan(xpend1*(ang_palla-xsoglia)))+ Vmax*atan(-
(xpend1)*xsoglia)/PI;

  return velocita;
}  // end Vel_Ins_Pal



//------------------------------
//      I nostri TASK
//------------------------------


// ++++++++++++++++++++++++++++ LeggiCam ++++++++++++++++++++++++++++++++

void LeggiCam (void) // attiva il collegamento con la telecamera e richiama SetInfo
{
 if (!sfIsConnected) process_state = END;

 if ( !okCam && sfIsConnected )
   {
    inizializza ();
    sfRobotCom2Bytes(sfCOMDIGOUT,0,255);   // abilitazione della paletta;
    sfSMessage("digoutput = %d",(int)sfRobot.digoutput);
    okCam ++;
   }

 if (conta_ciclo<10000) conta_ciclo++; else conta_ciclo = 2;
 if ((conta_ciclo%PRIORITA_02) == 0)
   {
    if (!grVideoWaitImage(400))
      {
      printf("No image, timed out!\n");
      }
    else
      {
       grVideoGetImages(&dataL,NULL);
      }

   SetInfo();            // Richiama la procedura SetInfo per acquisire i dati dalla
telecamera
  }

 switch(process_state)
  {
   case END:
    grVideoClose();
    process_state = sfSUCCESS;
   break;
  }
} // end LeggiCam



// ++++++++++++++++++++++++++ Para ++++++++++++++++++++++++++++++++++

void Para (void)         // muove Saracinescu davanti alla palla rimanendo nella porta
{
 static int AngPalo = 0;
 int offset,
     prod,
     velocita=0,
     alfa;               // permette di frenare prima il robot se sta andando veloce ed
e' sul palo

 sfSetMaxVelocity(800);
 if (!sfIsConnected) process_state = END;
```

```
 sfRemoveTask("vai_in_porta");

 if (non_vedo_palla>5)  // Se non vede la palla torna in centro alla porta
    {
     ang_a1=360-Robot.thpo1;
     ang_a2=Robot.thpo2-180;
     if (ang_a1>ang_a2)
       verso_spostamento=1;
     else verso_spostamento=-1;
     if((abs(ang_a1-ang_a2))<TOLLERANZA*3)
        sfSetVelocity(0);

     else   sfSetVelocity(VELOCITA_SPOST*2*verso_spostamento);

    }

else  // Se vede la palla
  {
   velocita=(int)sfRobot.tv;    // velocita attuale del robot
   if ((velocita>VELOCITA_SPOST*2) || (velocita<-VELOCITA_SPOST*2)) alfa=30;
   else alfa=0;

if ( ((conta_ciclo%PRIORITA_01)==0) && (Robot.thpo1>180) && (Robot.thpo2>180) )
  {  // se ha la porta dietro di se

   if (
       ( ((Robot.thpo1)<=(275+alfa)) && ( ((Robot.thpa)<90) || ((Robot.thpa)>270) ) ) ||
       ( ((265-alfa)<=(Robot.thpo2)) && ( ((Robot.thpa)>90) && ((Robot.thpa)<270) ) )
      )
     {
      if  ( ((Robot.thpa)<90) || ((Robot.thpa)>270) )       //se la palla e' a destra
        {
         if ( (Robot.thpo1<260) && (non_vedo_palo1==0) )    //se e' fuori dal palo torna
indietro
          {
            sfSetVelocity(VELOCITA_SPOST);
          }
         else
          {
            sfSetVelocity(0);
          }
        } // end if
      if    ( ((Robot.thpa)>90) && ((Robot.thpa)<270) )      // se la palla è a sinistra
        {
         if ( (Robot.thpo2>280) && (non_vedo_palo2==0) )    // se è fuori dal palo torna
indietro
          {
            sfSetVelocity(-VELOCITA_SPOST);
          }
         else
           {
            sfSetVelocity(0);
           }
        } //end if
     }
   else
     {
      if (Robot.thpa>180) {offset=270; prod=1;} else {offset=90; prod=-1;}
      velocita=(int)(Vel_Ins_Pal((Robot.thpa-offset)*prod)); // Richiama la procedura
per calcolare la velocità
      sfSetVelocity(velocita);
     }

  }    // end if conta_ciclo
 else  // se non ha la porta dietro di se permette l'avvio nel modo corretto (cioe' da
fermo di StaiTH
  {
  sfSetVelocity(0);
  }
```

```
 }

  switch(process_state)
    {
     case END:
         process_state = sfSUCCESS;
     break;
    }
}  // end Para



// ++++++++++++++++++++++++++ VAI_IN_PORTA ++++++++++++++++++++++++++

void vai_in_porta (void)
{ // se si e' allontanato dalla porta si riposiziona correttamente
 int verso_spostamento;
 float coseno;

 if (!sfIsConnected) process_state = END;

 sfSMessage(" process  %d",goto_case);

 switch(goto_case)
  {



  case 1:            // Se vede i pali si allinea con la porta

   if ((Robot.thpo2!=-1) && (Robot.thpo1!=-1))
     {

      sfRemoveTask("Para");

      pausa++;
      if (pausa<50) StaiTH();
         else {pausa=0;
        goto_case=2;}
     }
   break;

  case 2:        // Va in centro alla porta
     ang_a1=360-Robot.thpo1;
     ang_a2=Robot.thpo2-180;
     if (ang_a1>ang_a2)
       verso_spostamento=1;
     else verso_spostamento=-1;
   if (abs(ang_a1-ang_a2)<TOLLERANZA)
      { // e' in centro
       sfSetVelocity(0);
        dist_porta=(int)(((float)(LARGHEZZA_PORTA/2))*tan(deg2rad(ang_a1)));
       sfSetDHeading(-90);
       goto_case=3;
       break;
      }
     else      sfSetVelocity(VELOCITA_SPOST*verso_spostamento);
   break;

  case 3:
     if (sfDoneHeading(10)) goto_case=4;
    break;

  case 4:      // Va verso la porta

      sfSetMaxVelocity(VELOCITA_SPOST*4);
     if (-50000<dist_porta<50000)
       {
         if (fase==1) {sfSetPosition((int)((dist_porta-DIST_VOLUTA_PORTA)/2));
                 fase=2;}
```

```
                else {sfSetPosition((int)(dist_porta-DIST_VOLUTA_PORTA));
                  fase=1;}
          goto_case=5;
        }
      else goto_case=6;
      break;

  case 5:
      // controlla che si sia spostato
      if (sfDonePosition(2))
        {
         sfSetVelocity(0);
         sfSMessage("Fase %d",fase);
         if (fase==1) goto_case=6;
             else goto_case=1;
        }
      break;




  case 6:
     // ++++++++ SONO IN PORTA ++++++++++

       pausa++;
       if (pausa<50) StaiTH();
          else {pausa=0;
          fase=1;
           sfInitProcess(Para,"Para");}
          break;

  }  // end swith(goto_case)

 switch(process_state)
  {
   case END:
       process_state = sfSUCCESS;
       break;
  }

} // end vai_in_porta




// ++++++++++++++++++++++++++++++ StaiTH ++++++++++++++++++++++++++++++

void StaiTH (void) // cambia l'orientamento di Saracinescu parallelo alla porta
{

 float ang_tolleranza = 3;  // Massima tolleranza sull'angolo
 float ang_errato=0;         // angolo d'errore del robot rispetto all'asse parallelo
della porta

 if (!sfIsConnected) process_state = END;

 if ( ((conta_ciclo%PRIORITA_01)==0) && (NumPali>1) && ((sfRobot.tv)==0))
  {  //esegue la procedura se vede entrambi i pali e Saracinescu e' fermo
   if (Robot.thpo1>(180-Robot.a1))
     ang_errato = (int)(360 - Robot.a1 - Robot.thpo1);
   else ang_errato = (int)(-Robot.thpo1 - Robot.a1);

   if ((fabs(ang_errato)) > ang_tolleranza)
    {  // se l'angolo d'errore e maggiore della tolleranza riposiziona il robot
     sfSetDHeading ((ang_errato));
    }
  } // end if
```

```
 switch(process_state)
  {
   case END:
    process_state = sfSUCCESS;
   break;
  }

} // end StaiTH




//-------------------------------------------------------
//      FUNZIONI NECESSARIE AL COLLEGAMENTO CLIENT
//-------------------------------------------------------

int myKeyFn(int ch)    /* any user processing of keyboard msgs here */
{
 switch(ch)
   {
    case SPACEKEY: // permette di fermare il robot se si preme la barra spaziatrice
      sfSetVelocity(0);
      sfMessage("Stopped!");

   return 1;
   }
 return 0;                   //return 0 for default handling
}

void myStartupFn(void)
{
 sfSetDisplayState(sfDISPLAY, 0);

 Robot.thpa = -1;
 Robot.dpa = -1;
 Robot.thpo1 = -1;
 Robot.thpo2 = -1;
 Robot.thpo = -1;
 Robot.th_papo = -1;
 Robot.dpo1 = -1;
 Robot.dpo2 = -1;
 Robot.dpo = -1;
 Robot.a1 = -1;
 Robot.a2 = -1;

 sfConnectToRobot(sfTTYPORT,sfCOM1);

}

void myConnectFn(void)
{
 sfInitInterpretationProcs();

// dichiarazione dei Tasks
    sfInitProcess(LeggiCam,"LeggiCam");
    sfInitProcess(vai_in_porta,"vai_in_porta");
}

//*******************************  MAIN  *********************************

void main(int argc, char **argv)
{

 /* set up user button and key processing */
 sfKeyProcFn(myKeyFn);

 // register callbacks
 sfOnConnectFn(myConnectFn);
```

```
 sfOnStartupFn(myStartupFn);

 /* start up the Saphira micro-tasking OS, don't return */
 printf("starting...\n");
 sfStartup(0);
}
 // end MAIN
```