Università degli Studi di Brescia – Facoltà di Ingegneria

Sistema automatico per la consegna della posta nel Dipartimento di Elettronica della facoltà di Ingegneria.

Progetto sviluppato nell'ambito del corso di "Robotica"
Prof. Riccardo Cassinis

studenti : Copeta Mauro 025048 – Ragnoli Riccardo 022828

Sommario :

1 Struttura del programma.

La struttura del programma è quella definita da Saphira come un “Standalone Client”, ottenuta compilando codice C e linkandolo con le librerie di Saphira (rel. 6.1e). Il risultato della compilazione è un eseguibile.
Il programma gira su una piattaforma basata su Linux (Distribuzione Slackware 3.30 e kernel 2.0.30), è stato scritto utilizzando l’utility “joe”, ed è stato compilato utilizzando il compilatore “gcc” utilizzando un makefile (vedi Appendice); per il debug è stato utilizzato il programma “xxgdb”.

Il programma si lancia aprendo una shell di xterm all’interno di X-window e digitando “posta”seguito da ENTER al propt dei comanti.

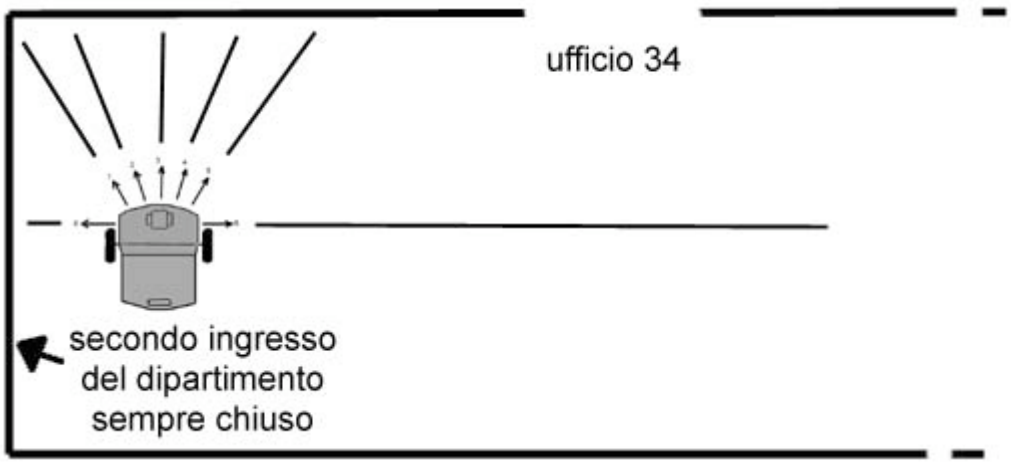
Una volta lanciato occorre posizionare le finestre con le consuete modalità; l’interfaccia utente è composta da due finestre: la prima in modo testo mostra il percorso che sarà seguito per raggiungere l’obiettivo; la seconda invece è l’interfaccia del programma Saphira, che mostra in real-time lo stato del robot.

Osservando la finestra di Saphira ci si può rendere conto di come il Robot stia interpretando il mondo che lo circonda è come lo stia mappando nella propria conoscenza interna.

Caratteristica fondamentale del programma a cui si deve il fatto che il comportamento dei robot sia quello desiderato è la correzione di errori che si accumulano durante il movimento reale; sulla base delle letture dei sonar infatti il sistema riposiziona la mappa interna in modo da far ricoincidere le letture dei sonar ai profili dei corridoi definiti nell’ambiente.

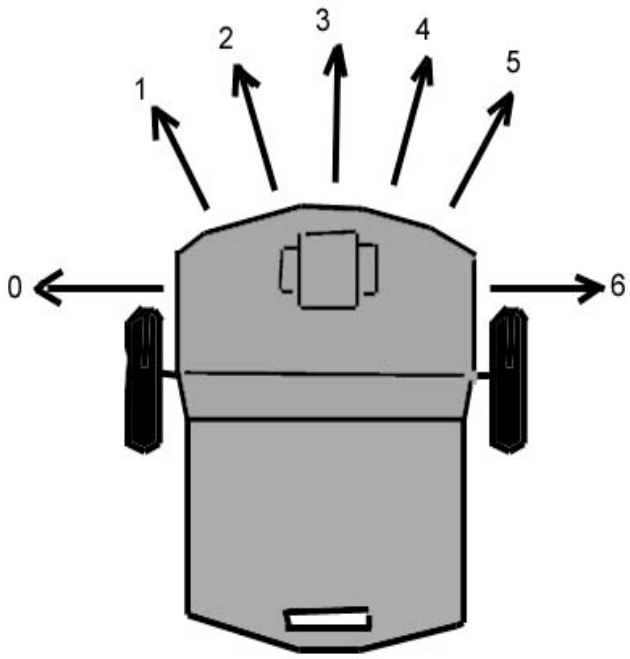
2 Strategia.

Quando il programma viene lanciato, il robot deve necessariamente trovarsi nella posizione indicata in figura:



Il robot in questa posizione infatti, può **acquisire** la propria posizione. Successivamente ruota di 90° verso destra ed è pronto a ricevere i comandi.

Inserito il numero d'ufficio di destinazione, viene verificato innanzitutto che l'ufficio esista, in caso positivo viene calcolato il percorso ottimo ed il robot inizia la movimentazione.
Se durante la movimentazione, il robot incontra un ostacolo, subito si arresta per evitare la collisione.
La presenza o meno di un ostacolo viene valutata attraverso l'uso di 3 sonar frontali, la cui soglia è stata diversificata:



Sonar	Soglia di arresto (mm)
2	800
3	500
4	800

In presenza di un ostacolo, prima di calcolare un percorso alternativo, il robot attente qualche secondo. Questo perché spesso gli ostacoli che si possono trovare di fronte al robot non sono fissi ma possono essere rimossi.

Per la movimentazione si è utilizzato il behavior "sfFollowCorridor" che consente al robot di seguire un corridoio, precedentemente definito come un artefatto.
Per definire il corridoio è necessario inserire la posizione nella mappa, la lunghezza, la larghezza e specificare un goal position che il robot cercherà di raggiungere.

Strategia.

2.1 Dettagli sulla movimentazione.

2.1.1 Corpo principale

Come si può notare nel main del programma, il behavior sfFollowCorridor viene lanciato con l'istruzione

```
sfStartBehavior(sfFollowCorridor,"Corridoio",0,1,0,corr[n],goal[n]);
```

così facendo, Saphira cerca di raggiungere il Goal ennesimo attraverso l'ennesimo corridoio. Il nome del Task creato è "*Corridoio*".
La fine del task viene continuamente ciclata attraverso l'uso della variabile

```
sfTaskFinished("Corridoio")
```

quando questa variabile diventa pari ad "1", (task concluso) occorre rimuovere il task "*Corridoio*" ed il goal appena raggiunto; a questo punto si fa ripartire il ciclo caricando il goal successivo e relativo corridoio.

Per una descrizione più dettagliata si rimanda al flowchart contenuto in Appendice in cui è descritto il main del programma.

2.1.2 Eccezione

In presenza di un ostacolo frontale il robot deve immediatamente arrestarsi e ciò viene fatto con il behavior sfStop. Occorre considerare però che anche in presenza di un ostacolo, il behavior sfFollowCorridor rimane comunque attivo in quanto non è possibile raggiungere il goal finale.

Per far convivere più behavior si è reso necessario considerare livelli opportuni di priorità.
Il task, sfFollowCorridor, è stato lanciato a priorità 1 (4° parametro di sfStartBehavior) consentendoci quindi di lanciare quando necessario sfStop a priorità 0 (quella massima).

```
sfStartBehavior(sfStop,"STOP",0,0,0);
```

Se non è presente nessun ostacolo frontale, il livello di priorità per sfStop è "2" consentendo al robot di continuare a cercare di raggiungere il goal.

```
sfStartBehavior(sfStop,"STOP",0,2,0);
```

2.2 Algoritmo per il calcolo del percorso.

Il calcolo del percorso del robot viene effettuato dall'algoritmo di Dijkstra dotato di euristica. Tale euristica viene calcolata pesando i vari percorsi di ispezione e concentrando la ricerca su quello di costo minimo.
Il costo di un percorso è dato dalla sommatoria dei pesi degli obbiettivi intermedi. Il peso di un obbiettivo è indice del tempo perso dal Robot.
L'area di movimentazione è specificata in un file testo chiamato "Dea1.map" (per il contenuto vedi Appendice) in cui per ogni goal si indica il peso associato.

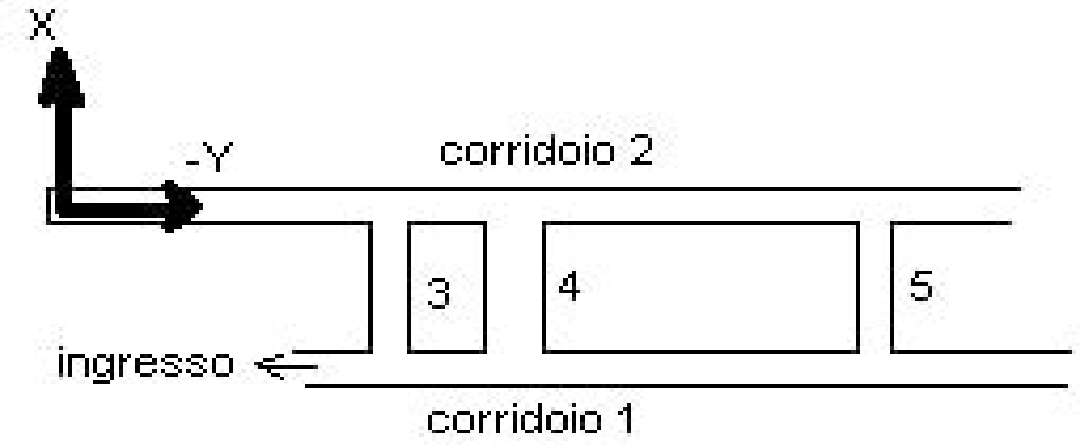
Il peso è un intero che varia da 0 a 9 se la zona è accessibile al Robot alttrimenti se non lo è, è pari a "#" (peso infinito).

Nel calcolo del percorso minimo, l'algoritmo specifico, potrà esplorare solo le "zone" caratterizzate da un numero.
Ad ognuno dei goal con peso diverso da "#", verranno poi associate coordinate del mondo reale; in questo modo ci si svincola dall'avere un passo di discretizzazione fisso.

I coefficienti utilizzati nel progetto variano da 0 a 2; questo è dovuto alla struttura piuttosto regolare della DEA ed alla pavimentazione liscia.

3 Sistema di riferimento

Il Dipartimento di Elettronica, è stato suddiviso in 5 corridoi.
In figura è mostrato il sistema di riferimento a cui è riferita la conoscenza del robot.



La struttura della DEA è stata suddivisa in 5 corridoi.
Di seguito indichiamo i parametri dei 5 corridoi richiesti per la definizione della mappa (vedere cap.4.1. per maggiori dettagli).

Num.	X_centro	Y_centro	Orient.	Larghezza	Lunghezza
1	-9900	-32500	0	2300	37400
2	0	-24500	0	2300	49500
3	-4600	-15300	-90	2100	7300
4	-4600	-28000	-90	2700	7300
5	-4600	-53200	-90	2300	7300

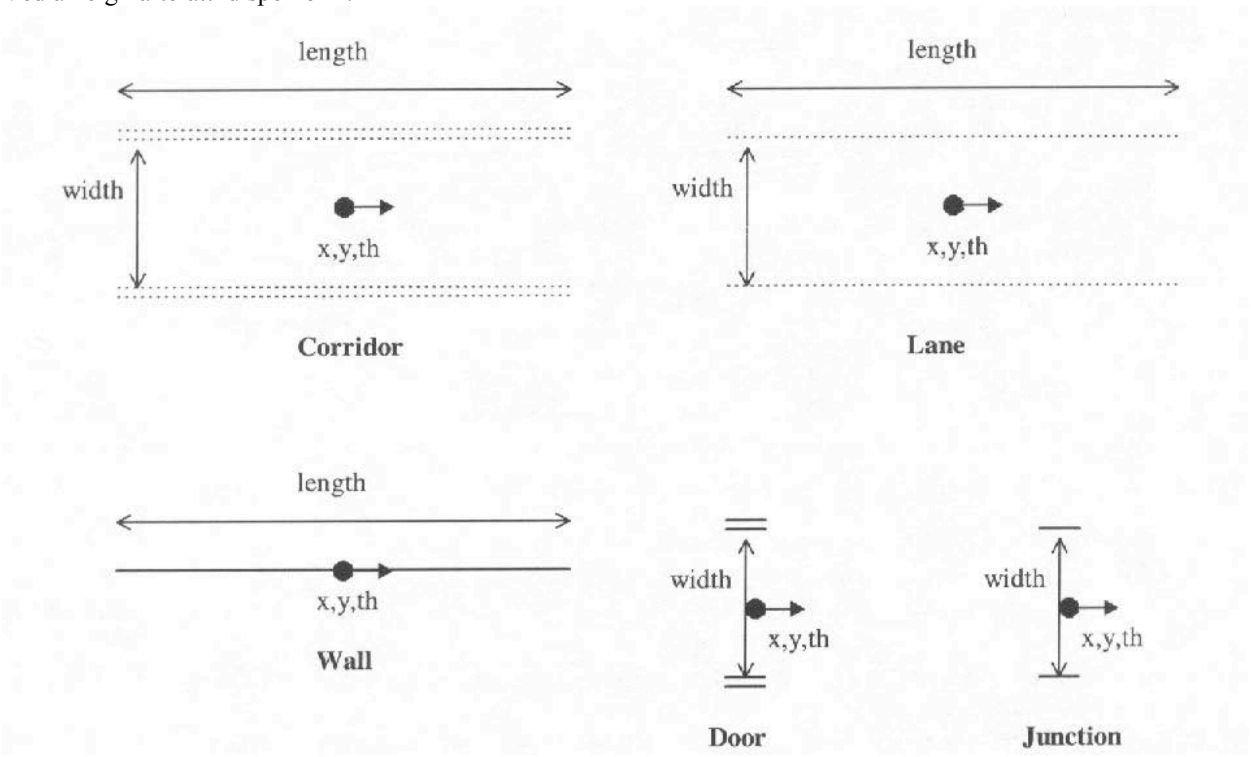
4 Commenti ai File

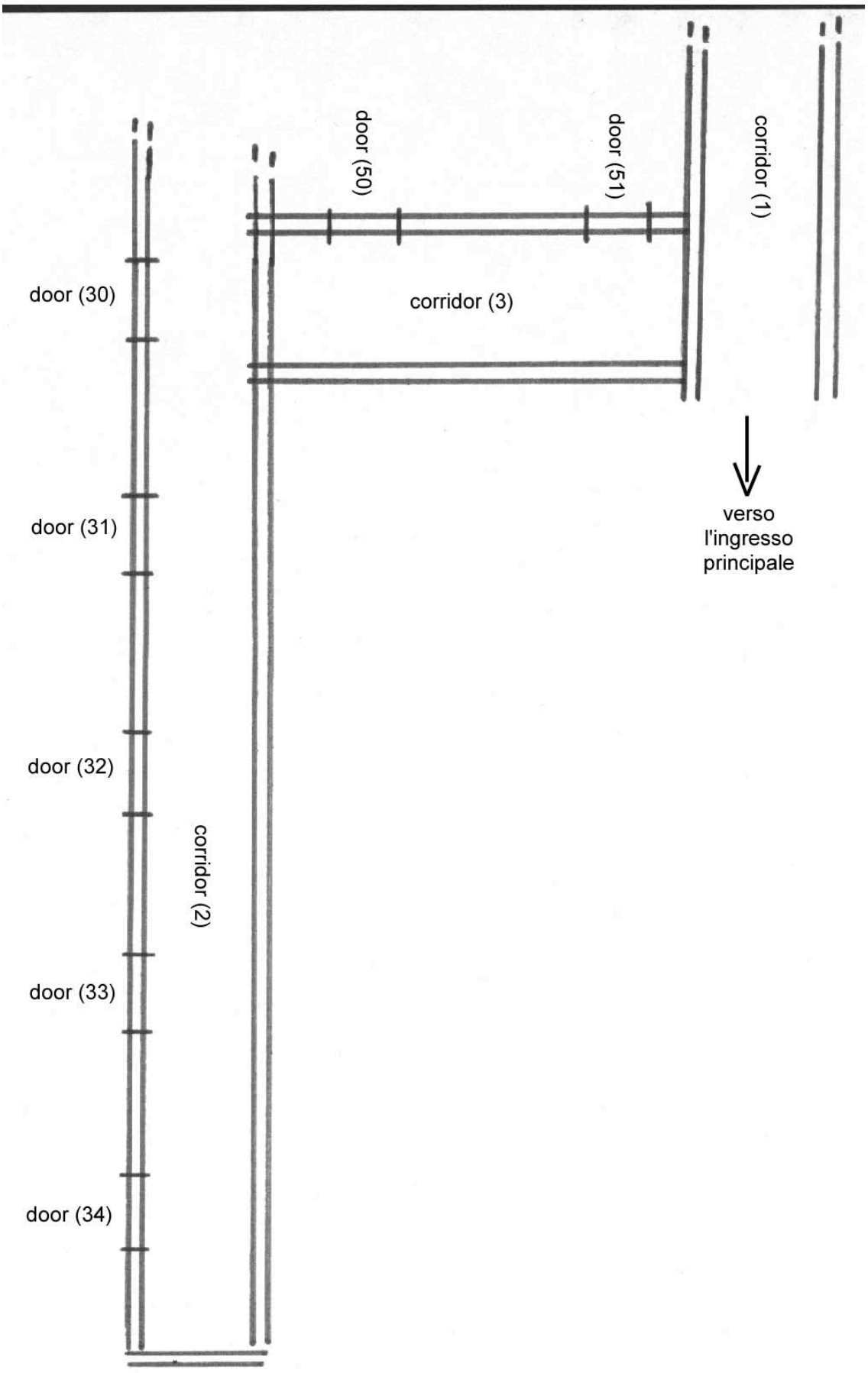
Di seguito sono riportati alcuni commenti relativi ai file utilizzati nel progetto. Per quanto riguarda il contenuto si rimanda in Appendice.

4.1 Commenti al file “DEA_S.map”

Questo file contiene la definizione del Dipartimento di Elettronica, con gli artefatti messi a disposizione da Saphira.

Vediamo gli artefatti disponibili :





Vediamo a scopo illustrativo una parte del Dipartimento, definita con gli artefatti.

4.2 Note sul file “Corridoi.txt”

Questo file specifica per ogni goal, i parametri del corridoio utilizzato per raggiungerlo e le coordinate reali del goal. L’informazione contenuta è ridondante in quanto sarebbe stato sufficiente indicare il numero di corridoio e cercare in parametri nella base di dati.

> N.B. : Abbiamo optato per questa soluzione in quanto era sicuramente la più semplice e naturale ed ha semplificato notevolmente l’algoritmo per la lettura dei parametri.

La necessità di indicare per ogni goal il relativo corridoio di appartenenza è dovuto all'uso del behavior "sfFollowCorridor".

I parametri contenuti nel file sono :

- Coordinata X dell’obbiettivo
- Coordinata Y dell’obbiettivo
- Numero corridoio
- Coordinata X del centro del corridoio
- Coordinate Y del centro del corridoio
- Orientamento del corridoio
- Larghezza del corridoio
- Lunghezza del corridoio

4.3 Commenti al file “DEA1.map”

Questo file costituisce una discretizzazione del mondo reale. Il passo di discretizzazione non è fisso; in questo modo, è possibile usare un passo piccolo nelle zone ricche di "Obiettivi", ed un passo maggiore nelle zone di poca rilevanza. Ad ognuno di questi goal saranno poi associate coordinate del mondo reale.

Nel nostro caso il file "DEA1.map" è:

```
################################################.
#1111111111112121112121111111111111111111111111#.
###############2#####2########################1##.
###############1#####1########################1##.
###############1#####1########################1##.
###############1#####1########################1##.
###############1#####1########################1##.
###############1#####1########################1##.
###############1#####1########################1##.
###############1#####2########################1##.
###############1111121211111111111111111111111##.
################################################.
```

Il calcolo del percorso minimo è dotato di euristica ed è basato sull'algoritmo di Dijkstra, ed è affidato al file “libreria.c” (vedere in Appendice).

4.4 Commenti al file “uffici3.txt”

Dato che l'unica informazione richiesta da robot per raggiungere l'obiettivo è il numero d'ufficio, occorre trovare quali sono le coordinate del goal nel mondo discretizzato da "DEA1.map".
Vediamo il contenuto del file:

```
;;ufficio xgoal ygoal xreali yreali
1 1 1 0 0
34 3 1 0 -1700
33 6 1 0 -5300
32 9 1 0 -8900
31 12 1 0 -12500
30 16 1 0 -16100
29 23 1 0 -23300
28 26 1 0 -26900
27 30 1 0 -30500
26 33 1 0 -34100
25 36 1 0 -37700
24 39 1 0 -41000
23 40 1 0 -45100
22 46 1 0 -48500
11 42 10 -9500 -46300
10 39 10 -9500 -42200
9 37 10 -9500 -38800
8 33 10 -9500 -35200
7 30 10 -9500 -31600
6 27 10 -9500 -28000
5 23 10 -9500 -24400
4 20 10 -9500 -21100
```

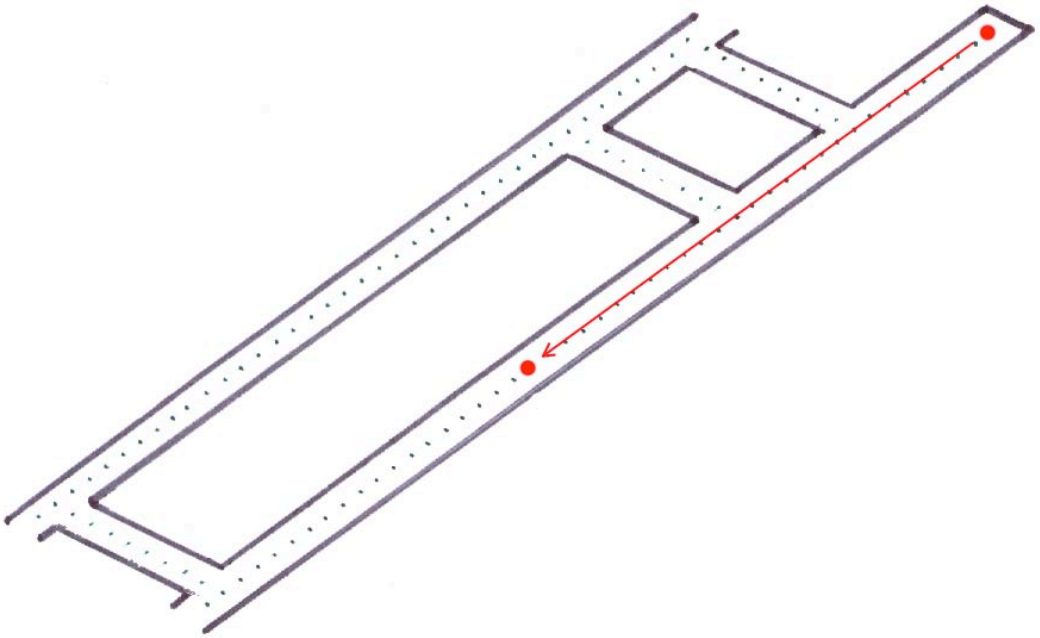
Ad esmpio l'ufficio numero 26 è rappresentato in DEA1.map dalla casella (33,1) e nel mondo reale corrisponde alle coordinate (0,-34100) riferite al sistema di riferimento definito nel cap.3.

5 ESEMPI

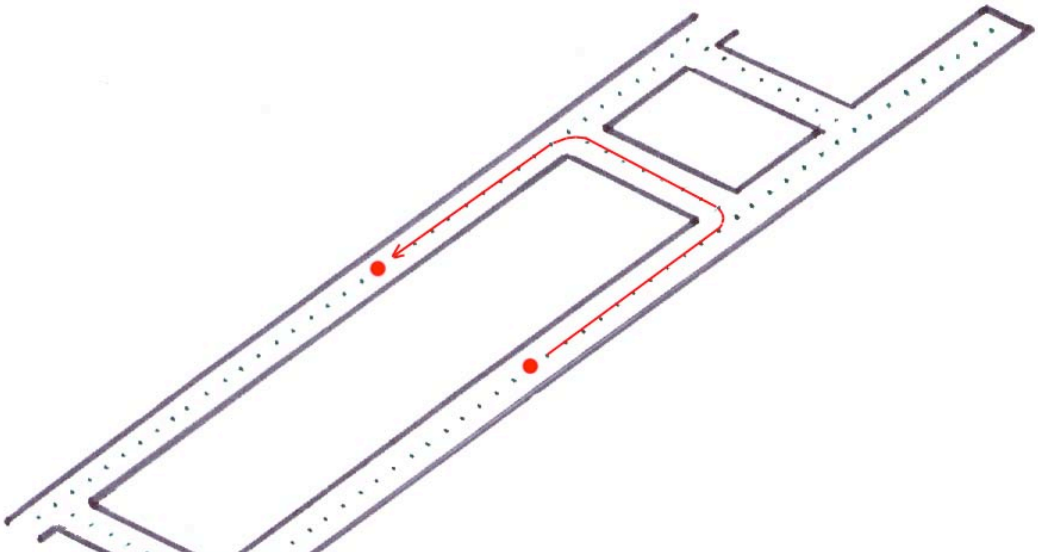
Vediamo ora qualche esempio atto ad illustrare il comportamento del robot.

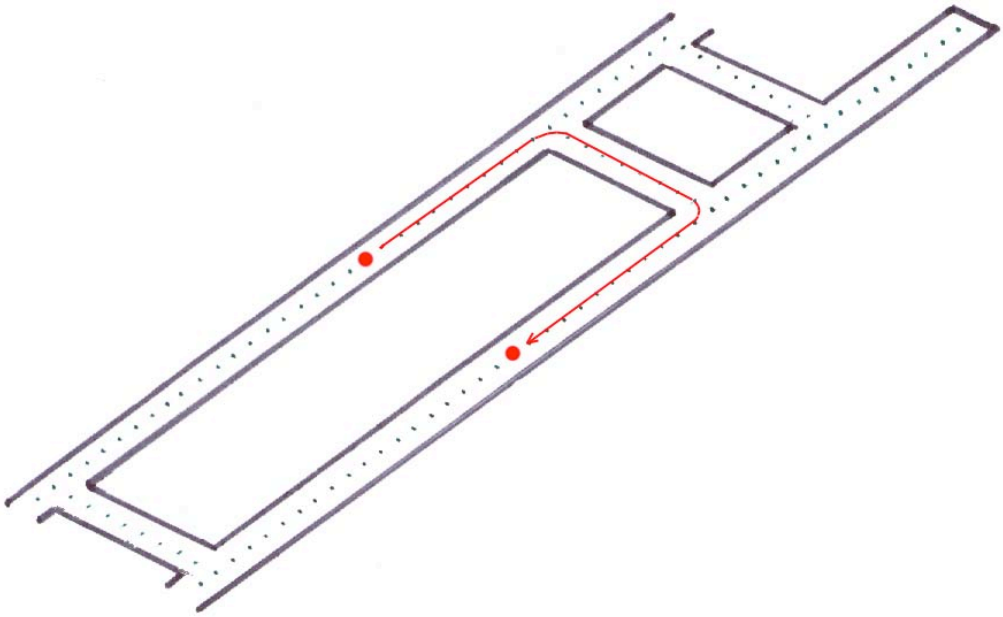
5.1 Esempio 1 : dalla posizione di riposo all'ufficio 26.

Dalla posizione di riposo digitiamo al prompt comandi il numero 26 seguito da Enter. Il robot pianifica il percorso ed inizia la movimentazione.



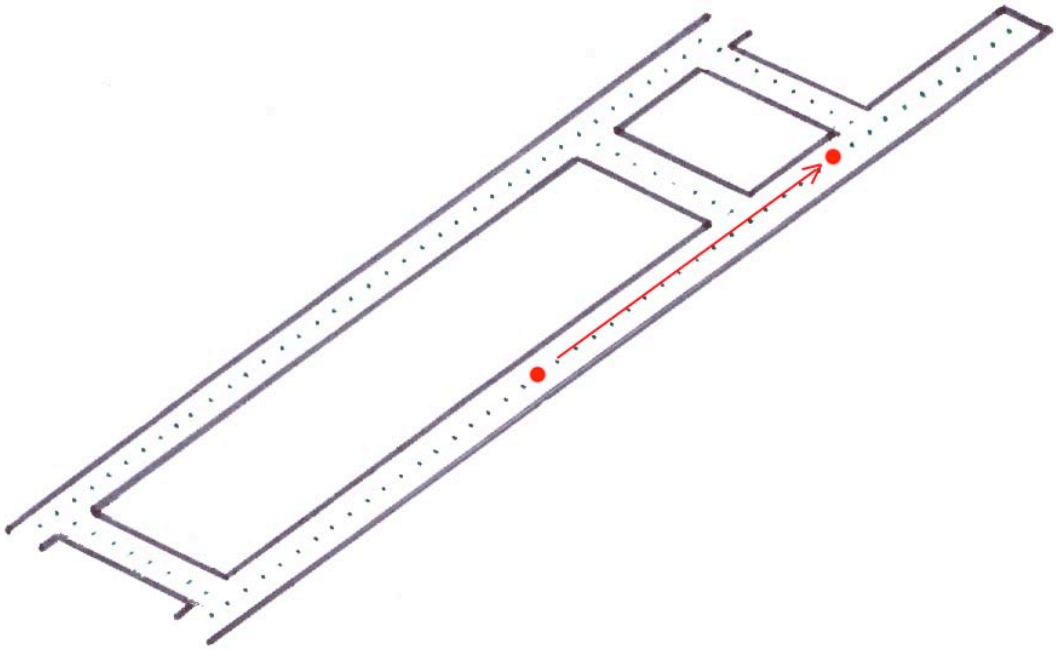
5.2 Esempio 2 : dall'ufficio 26 all'ufficio 8 e ritorno.



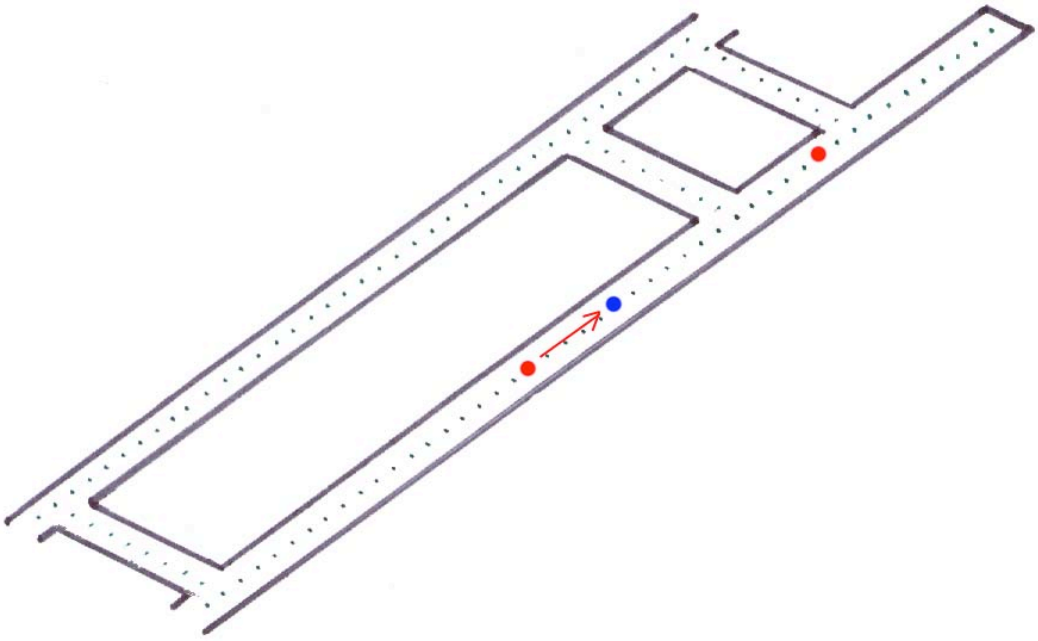


5.3 Esempio 3 : dall'ufficio 26 all'ufficio 30 con un ostacolo.

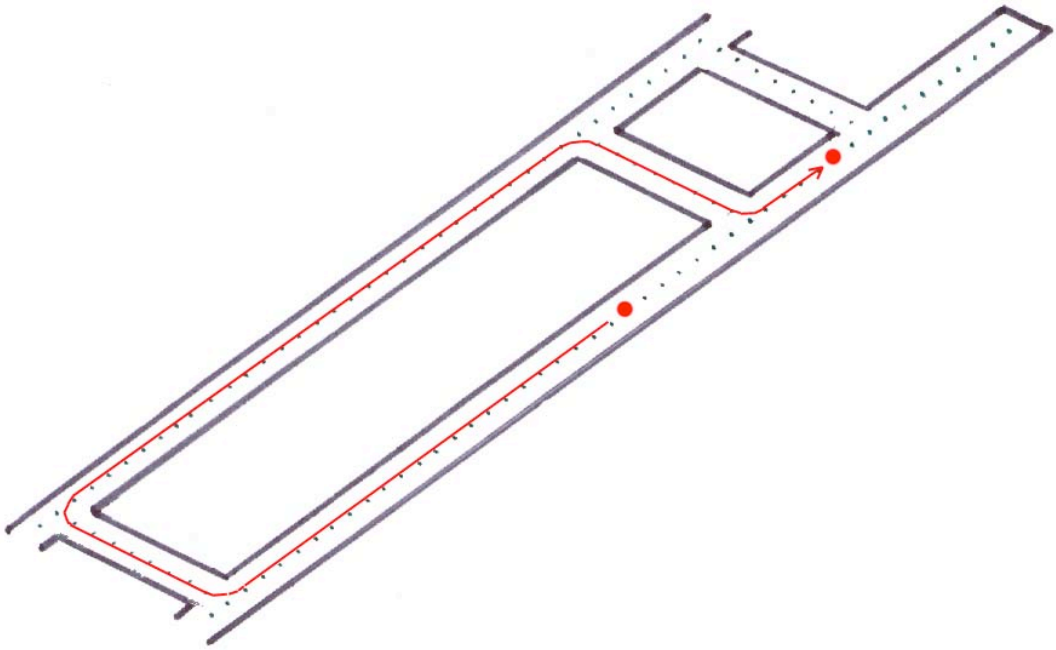
Inseriamo ora l'ufficio numero 30. Il movimento pianificato è quello indicato in figura.



Vediamo ora come si comporta il robot di fronte ad un ostacolo. Blocchiamo quindi il corridoio durante la movimentazione.



Il robot si arresta; attende qualche secondo che l'ostacolo venga rimosso e poi dato che l'ostacolo permane pianifica il percorso alternativo migliore.



6 APPENDICI

6.1 FILE di Programma

6.1.1 Makefile

```
#-----------------------------------------------------------------
#
#  Makefile for Posta
#  RAGNOLI : 022828
#  COPETA  : 025028
#  Ultima modifica: 1-10-98
#
#-----------------------------------------------------------------

CC = gcc
OFLAGS = -O -g -DUNIX -DLINUX

X11D = /usr/X11
BIND = ./
#Directory file bin

SRCD = ./
#Dir. dove ci deve essere il sorgente

OBJD = obj/
#Dir. dove mettera' gli obj

INCD = -I$(SAPHIRA)/handler/include/ -I$(X11D)include/ -
Iinclude/
LIBD = -L$(SAPHIRA)/handler/obj/ -L$(X11D)lib/ -L/lib/ -Llib/
COLBERT = $(SAPHIRA)/colbert/
OAALIB = $(SAPHIRA)/oaa/agents/lib/

LIBS = -ldl  -lm -lc -lXm -lXt -lX11 -lXext -lXpm

include $(SAPHIRA)/handler/include/os.h

CFLAGS = -g -D$(CONFIG) $(OFLAGS) $(INCD)

INCLUDE = -I$(INCD) -I$(X11D)include

#
# rules
#

all: $(BIND)posta
      touch all

$(OBJD)posta.o: $(SRCD)posta.c
      $(CC) -c $(CFLAGS) $(SRCD)posta.c $(INCLUDE) -o
$(OBJD)posta.o

OBJS = $(OBJD)posta.o

$(BIND)posta: $(OBJS)
      $(CC) $(OBJS) -o $(BIND)posta \
-L$(MOTIFD)lib $(LIBD) -lsf $(LIBS) -lc -lm
```

```
clean:
	-rm -f $(OBJD)* *~ $(SRCD)*~
	-rm -f $(BIND)posta $(BIND)*.pgm $(BIND)*.ppm
```

6.1.2 File “posta.c”

```
/*########################################
 *   Progetto Pellegrino
 *   RAGNOLI RICCARDO matr : 022828
 *   COPETA MAURO     matr : 025048
 *          Non cancellare queste linee !!!
 *########################################
 */

#include "libreria.c"
#include "saphira.h"

void myConnectFn(void);
void myStartupFn(void);
int myKeyFn(int ch);
int myButtonFn(int x, int y, int b, int m);


point *goal[102];       /* array dei puntatori ai goal da raggiungere */
point *corr[102];       /* array dei puntatori ai corridoi da seguire */
boolean Raggiunto;      /* variabile per il controllo del raggiungimento */
                        /* del goal prefissato */
int fine;               /* variabile di controllo per la fine del ciclo */

const
    Security_F=800;     /* sensibilità agli ostacoli del sonar frontale */
const
    Security=500;       /* sensibilità agli ostacoli dei sonar laterali */


/*************************************************************************
*
*
*void myStartupFn(void)
*
* Questa funzione dopo aver impostato lo stato della modalità di visualizzazione
* della finestra di SAPHIRA avvia in automatico la connessione con il robot
* reale .
*
*************************************************************************/


void myStartupFn(void)
{
  sfSetDisplayState(sfWAKE,TRUE);
  sfSetDisplayState(sfDISPLAY, 2);              /* set it to 5 Hz */
  sfRunEvaluator();                             /* do the evaluator */
  sfConnectToRobot(sfTTYPORT,sfCOM2);
}



/*************************************************************************
*
*
*int myKeyFn(int ch)
*
*
* Questa funzione viene utilizzata per associare ai tasti una funzione
* particolare nel nostro caso :
*   - premendo la barra spaziatrice il robot si arresta per un istante (  1 sec.
```

```
circa  )
*     per poi riprendere a seguire il percorso stabilito
*   - premendo invece la 'z'/'Z' si fuoriesce dal programma non prima però di
essersi
*      sconnessi dal robot .
*
**************************************************************************/


int myKeyFn(int ch)
{

  BEHCLOSURE b;

  switch(ch)
    {
    case SPACEKEY:
         sfBehaviorControl=0;
         sfSetVelocity(0);
         sfBehaviorControl=1;
         break;

    case 'Z':
    case 'z':
              fine=1;
              Raggiunto=1;
              sfDisconnectFromRobot();
              return 1;

    }
  return 0;                   /* return 0 if we don't process keys */
}


/*************************************************************************
*
*
*void myConnectFn(void)
*
*
* Questa funzione inizializza una serie di micro-tasks predefiniti di SAPHIRA
per il
* controllo del robot .
*
**************************************************************************/


void myConnectFn(void)        /* start those processes */
{

      /* inizializza i processi … */
sfInitRegistrationProcs();/* registrazione della posizione, utili */
                          /* per la navigazione del robot . */
   sfInitBasicProcs();    /* controllo delle comunicazioni di base */
                          /* del robot, della visualizzazione, dei sensori e
                          /* dei motori */
   sfInitInterpretationProcs();/* l'interpretazione delle letture dei sonar */
   sfInitControlProcs();      /*valutazione dei behaviors attivi */
   sfInitAwareProcs();
 }


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

```
*
*
* int myscanf(void)
*
*
* Questa funzione consente l'inserimento del numero dell'ufficio che il robot
deve
* raggiungere effettuando un controllo sul valore inserito affinché esso
corrisponda
* ad uno di quelli a disposizione ; in caso contrario stampa a video un elenco
degli
* uffici attualmente disponibili .
*
**********************************************************************/


int myscanf(void)
{

  int a;
  int ctrl;
  int k,s;


  ctrl=0;
  while(ctrl==0){
    printf("INSERISCI UFFICIO DI DESTINAZIONE :\n");
    scanf("%d",&a);
    if (a==0){
      ctrl=1;
      break;
     }
    for(s=0 ; s<=22 ; s++){
      if(a==Arr_uff[s][0]){
        ctrl=1;
        break;
       }
      else {
        ctrl=0;
      }
    }
    if(ctrl==0){
      printf("L'ufficio inserito non esiste \n ");
      printf("La scelta deve rientrare tra le seguenti possibilita':\n ");
      for(k=0 ;k<=20 ;k=k+2){
        printf("UFFICIO : %d --- UFFICIO : %d  \n
",Arr_uff[k][0],Arr_uff[k+1][0]);
      }
      printf("Premere '0' per disattivare il collegamento ed uscire \n");
      printf("ATTENZIONE il robot DEVE trovarsi nella POSIZIONE INIZIALE
(UFFICIO 1)\n");
     }
  }
  return a;

}



/**********************************************************************
*
*
* int main(int argc, char ** argv)
```

```
*
*
* Questa funzione è responsabile del controllo del robot durante l'esecuzione
del compito
* prefissato ed  è in  grado di gestire le  situazioni impreviste che  si
dovessero verificare
* come ad esempio la presenza di un ostacolo nel corso del tragitto .
*
**********************************************************************/


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

  int i,t,n,m,test,provv;
  int row,col;
  int X_provv,Y_provv;
  int Xn_2,Xn_1;


  Leggi_Uff();          /* caricamento da file delle informazioni */
  Leggi_Corr();         /* relative agli uffici e ai corridoi */
  abort_at_dest++;      /* setting dei flags utilizzati */
  debug_level = 10;
  use_heuristic++;
  test=Insert("DEA1.map");    /* caricamento della mappa da disco ,  */
  if (test==-1) return(0);    /* inserimento del primo ufficio e  */
                              /* determinazione del relativo percorso */
  for (n=lung_p-1; n>=0 ; n--)/* inversione della sequenza di punti individuati
*/
    Arr_int_f[lung_p-1-n][0]=Arr_int[n][0];
    Arr_int_f[lung_p-1-n][1]=Arr_int[n][1];
  }
  Carica_Corr(lung_p);        /* caricamento caratteristiche dei corridoi */
                              /* percorsi durante il tragitto individuato */
  sfButtonProcFn(myButtonFn); /* attivazione delle funzioni di gestione */
  sfKeyProcFn(myKeyFn);       /* della connessione/comunicazione */
  sfOnConnectFn(myConnectFn); /* con SAPHIRA
  sfOnStartupFn(myStartupFn);
  add_new_features=FALSE;
  sfAddEvalFn("myscanf",myscanf,sfINT,0);
  printf("\n starting...");
  printf("\n");
  sfStartup(1);               /* attivazione della connessione con il ROBOT */
  while(!sfIsConnected) sfPause(0); /* e ritorno del controllo dello stesso al
*/                                          /* programma chiamante */
  sfMessage("Il controllo del Robot l'ha il programma C chiamante.");
  sfSetDHeading(-90);
  sfPause(5000);
  if (sfLoadMapFile("/saphira/ver61/maps/DEA_S.map")==-1) {
/*caricamento della mappa del */
    sfMessage("NON E' STATA CARICATA LA MAPPA");
/* dipartimento in memoria */
   }
  fine=0;
  n=0;
  while (fine<1){     /* inizio della gestione della movimentazione */
    sfMessage("dentro il ciclo for");
      /* del ROBOT per il raggiungimento */
      /* dell'ufficio  desiderato */
    if ((n>lung_p-1)){  /* gestione dell'inserimento di un nuovo ufficio */
      sfRemPoint(goal[n]);          /* una volta raggiunto il precedente */
      sfStartBehavior(sfStop,"STOP",0,0,0);
```

```
      start_uff=end_uff;      /* l'ufficio iniziale diviene l'ufficio appena */
      Carica_Uff(start_uff,&startc,&startr);          /* raggiunto … */
      end_uff=myscanf();                  /* … mentre all'utente viene richiesto
                                          di */

                                          /* inserire il nuovo ufficio da
                                          raggiungere */
      if (end_uff==0){              /* nel caso in cui venga premuto lo zero */
        fine=1;                     /* si attiva la sconnessione dal ROBOT e */

        break;                      /* la conseguente uscita dal programma */
       }
      printf("L'UFFICIO PRESCELTO E': %d",end_uff);
      Carica_Uff(end_uff,&endc,&endr);
      Xn_2=Arr_int_f[n-2][1];
      Xn_1=Arr_int_f[n-1][1];
      for (row = 0; row < MAX_R; row++)
             /* azzeramento dell'array delle connessioni */
        for (col = 0; col < MAX_C; col++){      /* e di quello delle distanze */

          dist[row][col] = INFINITY;
          parent [row][col][0]= -1;
          parent [row][col][1]= -1;
         }
      pops = 0;
      pushes = 0;
      build_path();                 /* determinazione del percorso desiderato */
      limit_path();
      map_draw_path();
      for (m=lung_p ; m>=0 ; m--){
        Arr_int_f[lung_p-1-m][0]=Arr_int[m][0];
        Arr_int_f[lung_p-1-m][1]=Arr_int[m][1];
       }
      Carica_Corr(lung_p);
      n=1;

        /* determinazione della necessità di compiere o meno una */
              /* rotazione di 180° a seconda del percorso da seguire */

      if ((Xn_2>Xn_1)&&(Xn_1<Arr_int_f[n][1])){
        sfBehaviorControl=0;
        sfMessage("MI DEVO GIRARE!!!");
        sfSetDHeading(180);
        sfPause(3000);
        sfBehaviorControl=1;
       }
      if ((Xn_2<Xn_1)&&(Xn_1>Arr_int_f[n][1])){
        sfBehaviorControl=0;
        sfMessage("MI DEVO GIRARE!!!");
        sfSetDHeading(180);
        sfPause(3000);
        sfBehaviorControl=1;
       }
      sfStartBehavior(sfStop,"STOP",0,2,0);

     }

    goal[n]=sfCreateGlobalPoint(xgoal[n],ygoal[n],100);
    sfAddPoint(goal[n]);

corr[n]=sfCreateGlobalArtifact(sfCORRIDOR,nc[n],xc[n],yc[n],tc[n],wc[n],lc[n]);
    sfStartBehavior(sfFollowCorridor,"Corridoio",0,1,0,corr[n],goal[n]);
    sfSMessage("Obbiettivo %d",n+1);
```

```
    Raggiunto=0;
    while (Raggiunto==0) {    /* attesa del raggiungimento dell'i-esimo goal */
      if ((sfTaskFinished("Corridoio")==0)) {         /* del percorso
precedentemente definito */
        sfPause(0);
        sfMessage("attendo fine task");
       }
      if ((sfTaskFinished("Corridoio")==1))
      { /* rimozione dell'i-esimo goal appena raggiunto */
        sfPause(0);           /* e abilitazione al raggiungimento del goal */
        RemoveTask("Corridoio");/* successivo appartenente al percorso trovato
*/
        sfMessage("rimuove ");
        sfSMessage("Obbiettivo %d  raggiunto",n+1);
        sfRemPoint(goal[n]);
        Raggiunto=1;
        n++;
       }

      sfPause(500);

      if ((sfSonarRange(4)<Security)||
           (sfSonarRange(3)<Security_F)|| /* controllo dell'eventuale presenza */
           (sfSonarRange(2)<Security)) {  /* di ostacoli lungo il tragitto */
        sfStartBehavior(sfStop,"STOP",0,0,0);
        sfPause(8000);
        if ((sfSonarRange(4)<Security)||
/* se l'ostacolo permane per più di 8 sec. */
            (sfSonarRange(3)<Security_F)||
/* viene determinato un percorso alternativo */
            (sfSonarRange(2)<Security)) {/* quando ciò sia possibile */
          sfRemPoint(goal[n]);
          sfBehaviorControl=0;
          sfSetVelocity(0);
          sfSetDHeading(180);             /* inversione di 180° del ROBOT */
          sfPause(4000);
          X_provv=Arr_int_f[n][0];
          Y_provv=Arr_int_f[n][1];
          provv=map[Arr_int_f[n][0]][Arr_int_f[n][1]];
/* inserimento dell'ostacolo nella mappa */
          map[Arr_int_f[n][0]][Arr_int_f[n][1]]=INFINITY;
          for (row = 0; row < MAX_R; row++)
            for (col = 0; col < MAX_C; col++){
              dist[row][col] = INFINITY;
/* azzeramento dell'array delle connessioni */
              parent [row][col][0]= -1;   /* e di quello delle distanze */
              parent [row][col][1]= -1;
             }
          startr = Arr_int_f[n-1][0];
          startc = Arr_int_f[n-1][1];
          pops = 0;
          pushes = 0;
          build_path(); /* determinazione del nuovo percorso */
          limit_path();
          map_draw_path();
          map[X_provv][Y_provv]=provv;
/* rimozione dell'ostacolo rilevato della mappa */
          for (m=lung_p ; m>=0 ; m--){
            Arr_int_f[lung_p-1-m][0]=Arr_int[m][0];
            Arr_int_f[lung_p-1-m][1]=Arr_int[m][1];
           }
          Carica_Corr(lung_p);
          sfBehaviorControl=1;
```

```
          n=0;
         }
        sfStartBehavior(sfStop,"STOP",0,2,0);
       }
     }
  }


  sfBehaviorControl=0;
  sfSetDHeading(-90);
  sfPause(3000);
  sfMessage("Attendere sconnessione del Robot (3 secondi circa)");
  sfDisconnectFromRobot();
  printf("\n Fine del programma");
  printf("\n");
  sfPause(3000);
  return(0);
}
```

6.1.3 File “libreria.c”

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "SP.H"
#include "PQ.C"

static int use_heuristic = 0; /* flag per il setting dell'uso dell'euristica */
static int abort_at_dest = 0; /* flag per il setting del bloccaggio della  */
                              /* ricerca al primo percorso utile individuato */
int Arr_int[150][2],Arr_int_f[150][2];
/* array delle coordinate del percorso minimo */
float Arr_map[103][10];       /* array dei parametri dei corridoi e dei goal */
int Arr_uff[22][3];           /* array delle coordinate degli uffici */
int nc[102];
float xc[102],yc[102];        /* array dei parametri dei corridoi */
float lc[102],wc[102],tc[102];/* relativi al percorso individuato */
float xgoal[102],ygoal[102];  /* array delle coordinate dei goal */
int r_iniz,r_fin;
int wt,n_ufficio,x,y;
int lung_p;                   /* numero di punti del percorso individuato */


MAP map;
int dist[MAX_R][MAX_C];       /* array delle distanze */
int parent[MAX_R][MAX_C][2];  /* array delle connessioni tra i punti  del
percorso */
int startr,startc;            /* coordinate del punto iniziale */
int endr,endc;                /* coordinate del punto finale */
int start_uff,end_uff;        /* numero dell'ufficio iniziale e finale */
int rows_read, cols_read;     /* numero di righe e colonne della mappa adottata
int pops, pushes;
int debug_level = 0;


/*********************************************************************
*
*int map_read(char *fname)
*
* Questa funzione legge il file ( fname ), contenente la mappa relativa alla
zona in esame
* (il parametro fname viene passato alla funzione) e immagazzina le informazioni
nella
* struttura map[][] di tipo MAP. Dopo aver letto il file inizializza l'array
dist[][] ad un valore
* elevato ( 16354), e l'array  parent[][][] a -1 ( assenza di connessione con
altri punti ) .
* Il numero di righe della mappa viene salvato in row_read mentre il numero di
colonne
* in col_read . Questa funzione ritorna -1 in caso d'errore e 0 in caso di
successo. In
* caso d'errore lo stato delle variabili globali resta indefinito.
*********************************************************************/

int map_read(char * fname)
{

  FILE * file;
  int    row, col;
  char   line[MAX_C*2+1], * line_ptr;
```

```
  if ((file = fopen(fname, "r+")) == NULL) return(-1);
  cols_read = 0;
  for (row = 0; row < MAX_R && ! feof(file); row++) {
    if (fgets(line, MAX_C*2, file) == NULL) break;
    for (col = 0, line_ptr = line; ((col < MAX_C) && (*line_ptr)); col++,
line_ptr++)
      if (isdigit(*line_ptr))
        map[row][col] = *line_ptr - '0';
      else
        map[row][col] = INFINITY;
      if (col > cols_read) cols_read = col - 1;
    for (; col < MAX_C; col++) map[row][col] = INFINITY;
  }
  rows_read = row;
  for ( ; row < MAX_R; row++)
    for (col = 0; col < MAX_C; col++)
      map[row][col] = INFINITY;
  fclose(file);
  for (row = 0; row < MAX_R; row++)
    for (col = 0; col < MAX_C; col++) {
      dist[row][col] = INFINITY;
      parent[row][col][0] = -1;
      parent[row][col][1] = -1;
    }

  startr = startc = endr = endc = -1;
/* inizializzo il punto di partenza e quello */
/* d'arrivo in (-1,-1)*/
  return(0);

}


/*************************************************************************
*
*(void) map_draw_path()
*
*
* Questa funzione traccia a video il percorso minimo trovato sfruttando le
informazioni
* contenute nell'array parent[][] . Il punto di partenza viene rappresentato
attraverso il '$' , il
* punto d'arrivo attraverso il '^' , i punti intermedi che appartengono al
percorso utile tramite
* '+' ed i rimanenti come '.'
*************************************************************************/

void map_draw_path(void)
{

  int row, col;

  wt = 0;
  for (row = 0; row < rows_read; row++) {
    for (col = 0; col < cols_read; col++) {
      if (startr == row && startc == col)
        putc('^', stdout);
      else if (endr == row && endc == col)
        putc('$', stdout);
      else if (parent[row][col][0] >= 0 && parent[row][col][1] >= 0) {
        putc('+', stdout);
        wt += map[row][col];
```

```
      }
      else if (map[row][col] < INFINITY)
        putc('.', stdout);
      else
        putc('#', stdout);
    }
    printf("\n");
  }
}

#define drow(row,i) (row+delta[i][0])
#define dcol(col,i) (col+delta[i][1])


/* Array per l'abilitazione della traslazione in diagonale */
* A seconda dei valori impostati e' possibile scegliere tra la traslazione
* ad angolo retto e quella in diagonale
*/
int delta[8][2] = { -1, -1, -1, 1, 1, -1, 1, 1, -1, 0, 0, 1, 1, 0, 0, -1 };


#ifdef maxx
#undef maxx
#endif
#ifdef abs
#undef abs
#endif

#define maxx(a,b) ((a) > (b) ? (a) : (b))
#define abs(a)   ((a) < 0 ? -(a) : (a))


static int

h(int r1, int c1, int r2, int c2)
{
  if (use_heuristic) return(maxx(abs(r2 - r1), abs(c2 - c1)));
  return(0);
}


/*****************************************************************
*
*(void) build_path()
*
*
* Questa funzione implementa l'intero algoritmo .
*  Pone innanzitutto il punto di partenza nella coda
*  Mentre c'e' un elemento della coda di peso ridotto (chiamato (r,c)),
*    per ogni quadrato nell'intorno di (r,c) (che chiameremo (r',c')) che
soddisfi la seguente condizione :
*
*           dist[da (r,c)] + peso(r',c') < dist[da (r',c')]
*
* non facciamo altro che aggiornare l'array dist[][] tenendo conto del peso del
nuovo
* quadrato (r',c') * e successivamente inseriamo quest'ultimo nella coda .
*****************************************************************/

void build_path(void)
{

  PQUEUE * pq;
```

```
  int      row, col;
  int  i;

  pq = pqueue_new();
  if (startr < 0 || startr >= rows_read || startc < 0 || startc >= cols_read)
   return;
  dist[startr][startc] = 0;
  parent[startr][startc][0] = -1;
  parent[startr][startc][1] = -1;
  pqueue_insert(pq, startr, startc, 0 + h(startr,startc,endr,endc));
  pushes++;
  while (pqueue_popmin(pq, &row, &col) == 0) {
      /* DEBUG INFORMATION */
    pops++;
    if (debug_level > 0)
    /* See if we only wanted the one path */
    /* controllo se desidero il solo percorso minimo */
    if (abort_at_dest && row == endr && col == endc) {
     /* MI  FERMO */
      break;
    }
          /* PER OGNI quadrato adiacente a quello in esame che indichiamo con
(row,col)*/
    for (i = 0; i < 8; i++) {
      /* Controllo innanzitutto che esso appartenga alla mappa in esame */
      if (drow(row,i) < 0 || drow(row,i) >= rows_read ||
           dcol(col,i) < 0 || dcol(col,i) >= cols_read)
        continue;
      /* Controllo se ho trovato un percorso minore */
      if (dist[row][col] + map[drow(row,i)][dcol(col,i)] <
dist[drow(row,i)][dcol(col,i)]) {
        /* DEBUG INFORMATION */
        if (debug_level > 0) {
        }
        /* Se la risposta e' affermativa aggiungo questa scoperta a quelle fatte
nei passi precedenti */
        dist[drow(row,i)][dcol(col,i)] = map[drow(row,i)][dcol(col,i)] +
dist[row][col];
        parent[drow(row,i)][dcol(col,i)][0] = row;
        parent[drow(row,i)][dcol(col,i)][1] = col;
        /* Push the modified square onto the priority queue */
        /* inserisco il percorso trovato nella coda*/
        pqueue_insert(pq, drow(row,i), dcol(col,i),
dist[drow(row,i)][dcol(col,i)] + h(drow(row, i),
                                   dcol(col, i), endr, endc));
        pushes++;
      }
      else if (debug_level >= 5) {
       }
    }
  }
  return;

}


/*********************************************************************
*
* void limit_path(void)
*
*
* Questa funzione sulla base delle informazioni contenute negli array dist[][] e
parent[][]
```

```
* (il cui contenuto e' stato opportunamente determinato dalla funzione
buid path()),
* fornisce la sequenza di punti della mappa che definiscono il percorso minimo
cercato
* a partire dall'arrivo sino alla partenza; l'informazione viena memorizzata
nell'array Arr_int[][].
* A questo punto viene ripristinato il contenuto degli array dist[][] e
parent[][].
**************************************************************************/


void limit_path(void)
{

  int row, col;
  int tmp,i=1;



  if (endr < 0 || endr >= rows_read || endc < 0 || endc >= cols_read) return;
  row = endr;
  col = endc;
  printf("\n");
  Arr_int[0][0]=endr;
  Arr_int[0][1]=endc;
  do {
    dist[row][col] = -dist[row][col];
    tmp = parent[row][col][0];
    col = parent[row][col][1];
    row = tmp;
    Arr_int[i][0]=row;
    Arr_int[i][1]=col;
    i++;
  } while(row >= 0 && col >= 0);

  lung_p=i-1;
  printf("\n");
  for (row = 0; row < rows_read; row++)
    for (col = 0; col < cols_read; col++)
      if (dist[row][col] < 0)
        dist[row][col] = -dist[row][col];
      else
        parent[row][col][0] = -1;

  return;

}


/**************************************************************************
*
*void Leggi_Corr(void)
*
* Questa funzione salva nell'array Arr_map[][] tutte le informazioni relative
alle caratteristiche
* geometriche dei corridoi e dei goal associati alla mappa in esame, recuperati
dal file
* "CORRIDOI3.txt", presente in memoria.
**************************************************************************/


void Leggi_Corr(void)
{
```

```
  int i,j,q;
  int Xg_i,Yg_i,Nc_i;



  float Xc_i,Yc_i,Tc_i,Wc_i,Lc_i,Xgr_i,Ygr_i;
  FILE *INFORM;

  INFORM=fopen("CORRIDOI3.txt","r");
  if (INFORM==NULL) {
    printf("\nERRORE NELL'APERTURA DI CORRIDOI3.txt");
    return;
  }
  rewind(INFORM);
  q=0;
  while(!feof(INFORM)){
    fscanf(INFORM," %d %d %d %f  %f  %f  %f  %f  %f  %f ",

&Xg_i,&Yg_i,&Nc_i,&Xc_i,&Yc_i,&Tc_i,&Wc_i,&Lc_i,&Xgr_i,&Ygr_i);
    Arr_map[q][0]=Xg_i;
    Arr_map[q][1]=Yg_i;
    Arr_map[q][2]=Nc_i;
    Arr_map[q][3]=Xc_i;
    Arr_map[q][4]=Yc_i;
    Arr_map[q][5]=Tc_i;
    Arr_map[q][6]=Wc_i;
    Arr_map[q][7]=Lc_i;
    Arr_map[q][8]=Xgr_i;
    Arr_map[q][9]=Ygr_i;
    q++;
  }

  fclose(INFORM);
  q=0;

}




/**************************************************************************
*
*void Leggi_Uff(void)
*
* Questa funzione salva nell'array Arr_uff[][] le coordinate dei vari uffici
presenti nella mappa,
* recuperate  dal file "uffici3.txt", contenuto in memoria.
**************************************************************************/

void Leggi_Uff(void)
{

  int i,j,q;
  int xx,yy;
  float Xuff_Goal,Yuff_Goal;
  FILE *dati;
  int uff;

  dati=fopen("uffici3.txt","r");
  if (dati==NULL) {
    printf("\nERRORE NELL'APERTURA DI uffici3.txt");
    return;
```

```
  }
  rewind(dati);
  q=0;
  while(!feof(dati)){
    fscanf(dati,"%d %d %d %f %f",&uff,&xx,&yy,&Xuff_Goal,&Yuff_Goal);
    Arr_uff[q][0]=uff;
    Arr_uff[q][1]=xx;
    Arr_uff[q][2]=yy;
    q++;
  }
  q=0;
  fclose(dati);

}


/*************************************************************************
*
*void Carica_Corr(int Lengh_coda)
*
* Questa funzione recupera le informazioni geometriche dei corridoi attraversati
durante
* il tragitto che porta all'ufficio di destinazione; la stessa cosa viene
ripetuta per le
* coordinate dei goal da raggiungere.
* La funzione richiede il passaggio del numero di punti che compongono
* il percorso minimo.
*************************************************************************/

void Carica_Corr(int Lengh_coda)
{

  int i,q;




  for(i=0;i<=(Lengh_coda-1);i++){
    for(q=0 ; q<=102; q++){
      if ((Arr_int_f[i][0]==Arr_map[q][1])&&(Arr_int_f[i][1]==Arr_map[q][0])){
        nc[i]=Arr_map[q][2];
        xc[i]=Arr_map[q][3];
        yc[i]=Arr_map[q][4];
        tc[i]=Arr_map[q][5];
        wc[i]=Arr_map[q][6];
        lc[i]=Arr_map[q][7];
        xgoal[i]=Arr_map[q][8];
        ygoal[i]=Arr_map[q][9];
        break;
      }
    }
  }
}
/*************************************************************************
*
*void Carica_Uff()
*
* Questa funzione restituisce le coordinate dell'ufficio passatogli (ufficio).
*************************************************************************/


void Carica_Uff(int ufficio_s,int *row_coord,int *col_coord)
{
```

```
  int j=0;

  for(j=0 ; j<=21 ; j++){
    if (ufficio_s==Arr_uff[j][0]){
      *row_coord=Arr_uff[j][1];
      *col_coord=Arr_uff[j][2];
      break;
    }
  }

}


/*************************************************************************
*
*int Insert(char *fMappaUtiliz)
*
* Questa funzione gestisce l'inserimento del primo ufficio (quando il  robot si
trova nella
* posizione iniziale), accertandosi che il valore inserito sia tra quelli
disponibili; in caso
* contrario, dopo aver visualizzato l'elenco degli uffici presenti, richiede un
nuovo
* inserimento.
*************************************************************************/


int Insert(char *fMappaUtiliz)
{

  int rit,q,l;
  int ctrl;

  rit=map_read(fMappaUtiliz);
  if (rit==-1) return -1;
  ctrl=0;
  while(ctrl==0){
    printf("INSERISCI UFFICIO DI DESTINAZIONE (IL ROBOT E' NELL'ORIGINE): \n");
    if (scanf("%d", &end_uff) != 1) return(0);
    for(l=0 ; l<=22 ; l++){
      if (end_uff==Arr_uff[l][0]){
        ctrl=1;
        break;
      }
      else{
        ctrl=0;
      }
    }
    if(ctrl==0){
      printf("L'ufficio inserito non esiste \n");
      printf("La scelta deve rientrare tra le seguenti possibilita' :\n");
      for(q=0 ; q<=20 ; q=q+2){
        printf("UFFICIO : %d --- UFFICIO : %d \n
",Arr_uff[q][0],Arr_uff[q+1][0]);
      }
    }
  }
  startr=1;
  startc=1;
  Carica_Uff(end_uff,&endc,&endr);
  pops = 0;
  pushes = 0;
```

```
  build_path();
  limit_path();
  map_draw_path();

}
```

6.1.4 File “PQ.C”

```
/*********************************************************************
*
* Le funzioni seguenti vengono utilizzate per creare una coda di elementi con
priorità che
* verrà poi utilizzata per la ricerca del percorso minimo dal programma posta3.c
(libreria3.c );
* la caratteristica di questa coda è che essa mantiene un elenco degli stacks
ordinati
* in base alla loro priorità .Quando viene inserito un elemento si crea, se è
necessario, un
* nuovo stack il quale verrà poi inserito nella lista e riempito con i suoi
argomenti .
*
*********************************************************************/

#include <stdio.h>
#include <memory.h>
#include <stdlib.h>
#include <malloc.h>
#include "SP.H"

size_t max_memory = 0;
size_t current_memory = 0;


void *my_malloc(size_t size)
{

  void * ptr;

  if ((ptr = malloc(size)) == NULL) return(NULL);
  current_memory += size;
  if (current_memory > max_memory) max_memory = current_memory;
  return(ptr);

}


void my_free(void * ptr, size_t size)
{

  free(ptr);
  current_memory -= size;

}


PQUEUE *pqueue_new(void)
{

  PQUEUE * pq;

  if ((pq = (PQUEUE *) my_malloc(sizeof(PQUEUE))) == NULL) return (NULL);
  pq->count = 0;
  pq->priorities = NULL;
  return(pq);

}
```

```
int pqueue_insert(PQUEUE * pq, int row, int col, int wt)
{

  ELIST * e;
  PLIST * p, * pp;

  if (pq == NULL) return(-1);
  if ((e = (ELIST *) my_malloc(sizeof(ELIST))) == NULL) return(-1);
  e->row = row;
  e->col = col;
  e->next = NULL;
  if (pq->priorities == NULL ||                 /* controllo se la coda è vuota
oppure se  */
       pq->priorities->wt > wt) {               /* si tratta del primo elemento
della coda */
    if ((p = (PLIST *) my_malloc(sizeof(PLIST))) == NULL) return(-1);
    p->wt = wt;
    p->elements = e;
    p->next = pq->priorities;
    pq->priorities = p;
    pq->count++;
    return(0);
   }
  pq->count++;
  for (p = pq->priorities; p->next && p->next->wt <= wt; p = p->next)
      ;
  if (p->wt == wt) {
    e->next = p->elements;
    p->elements = e;
    return(0);
   }


 if ((pp = (PLIST *) my_malloc(sizeof(PLIST))) == NULL) return(-1);
 pp->wt = wt;
 pp->elements = e;
 pp->next = p->next;
 p->next = pp;
 pq->count++;
 return(0);

}

int pqueue_peekmin(PQUEUE * pq, int * row, int * col)
{

  if (pq == NULL || pq->priorities == NULL || pq->priorities->elements == NULL)
    return(-1);
  *row = pq->priorities->elements->row;
  *col = pq->priorities->elements->col;
  return(0);

}


int pqueue_popmin(PQUEUE * pq, int * row, int * col)
{

  ELIST * tmpe;
  PLIST * tmpp;

  if (pqueue_peekmin(pq, row, col) < 0) return(-1);
```

```
  pq->count--;
  tmpe = pq->priorities->elements;
  pq->priorities->elements = pq->priorities->elements->next;
  my_free(tmpe, sizeof(ELIST));
  if (pq->priorities->elements == NULL) {
    tmpp = pq->priorities;
    pq->priorities = pq->priorities->next;
    my_free(tmpp, sizeof(PLIST));
   }
  return (0);

}
```

6.1.5 File “SP.h”

```
#ifndef __SP_H__
#define __SP_H__

#define MAX_R           30
#define MAX_C           70
#define INFINITY  (1 << 14)

typedef int MAP[MAX_R][MAX_C];

typedef struct _ELIST {
     int      row, col;
     struct _ELIST * next;
} ELIST;

typedef struct _PLIST {
     int     wt;
     ELIST * elements;
     struct _PLIST * next;
} PLIST;

typedef struct {
     int    count;
     PLIST * priorities;
} PQUEUE;

int  map_read(char * fname);
void map_draw_path(void);
void build_path(void);
void limit_path(void);

PQUEUE * pqueue_new(void);
int pqueue_insert(PQUEUE * pq, int row, int col, int wt);
int pqueue_peekmin(PQUEUE * pq, int * row, int * col);
int pqueue_popmin(PQUEUE * pq, int * row, int * col);

#endif
```

6.2 FILE di Dati.

6.2.1 File “Dea1.map”

```
################################################.
#1111111111112121112121111111111111111111212#.
################2#####2######################2##.
################1#####1######################1##.
################1#####1######################1##.
################1#####1######################1##.
################1#####1######################1##.
################1#####1######################1##.
################1#####1######################1##.
################2#####2######################2##.
################121112121111111111111111121##.
################################################.
```

6.2.2 File “uffici3.txt”

```
1 1 1 0 0
34 3 1 0 -1700
33 6 1 0 -5300
32 9 1 0 -8900
31 12 1 0 -12500
30 16 1 0 -16100
29 23 1 0 -23300
28 26 1 0 -26900
27 30 1 0 -30500
26 33 1 0 -34100
25 36 1 0 -37700
24 39 1 0 -41000
23 40 1 0 -45100
22 46 1 0 -48500
11 42 10 -9500 -46300
10 39 10 -9500 -42200
9 37 10 -9500 -38800
8 33 10 -9500 -35200
7 30 10 -9500 -31600
6 27 10 -9500 -28000
5 23 10 -9500 -24400
4 20 10 -9500 -21100
```

6.2.3 File “Dea_s.map”

```
;; MAPPA della DEA
CORRIDOR (1) -9600, -32500, 90, 37400, 2300
CORRIDOR (2)  0, -24500, 90, 49500, 2300
CORRIDOR (3) -4750, -15400, 0, 7300, 2100
CORRIDOR (4) -4750, -21900, 0, 7300, 2700
CORRIDOR (5) -4750, -48200, 0, 7300, 2000
WALL (1) 0, 400, 0, 2300
WALL (2) -1134, 18000, 0, 32
WALL (3) -1134, 19000, 0, 32
WALL (4) -1118, 18500, 90, 100
;; UFFICI
DOOR (34) 1150, -1700, 180, 800
DOOR (33) 1150, -5300, 180, 800
```

```
DOOR (32) 1150, -8900, 180, 800
DOOR (31) 1150, -12500, 180, 800
DOOR (30) 1150, -16100, 180, 800
DOOR (29) 1150, -23300, 180, 800
DOOR (28) 1150, -26900, 180, 800
DOOR (27) 1150, -30500, 180, 800
DOOR (26) 1150, -34100, 180, 800
DOOR (25) 1150, -37700, 180, 800
DOOR (24) 1150, -41000, 180, 800
DOOR (23) 1150, -42300, 180, 800
DOOR (22) 1150, -48300, 180, 800
DOOR (12) -10650, -46300, 0, 800
DOOR (11) -10650, -42200, 0, 800
DOOR (10) -10650, -38800, 0, 800
DOOR (9) -10650, -35200, 0, 800
DOOR (8) -10650, -31600, 0, 800
DOOR (7) -10650, -28000, 0, 800
DOOR (6) -10650, -24400, 0, 800
DOOR (5) -10650, -21100, 0, 800
;; PORTE DEI BAGNI
DOOR (50) -1150, -25100, 0, 900
DOOR (51) -8300, -25100, 180, 900
DOOR (52) -1150, -38300, 0, 900
DOOR (53) -8300, -38300, 180, 900
DOOR (54) -6500, -16400, 90, 900
DOOR (55) -1150, -41700, 0, 900
DOOR (56) -8300, -41700, 180, 900
;; GIUNZIONI
JUNCTION (1) -1150, -15400, 0, 2100
JUNCTION (2) -1150, -21900, 0, 2700
JUNCTION (3) -1150, -48200, 0, 2000
JUNCTION (4) -8300, -15400, 0, 2100
JUNCTION (5) -8300, -21900, 0, 2700
JUNCTION (6) -8300, -48200, 0, 2000
```

6.2.4 File “corridoi.txt”

```
1 1 2 0 -24500 90 2300 49500 0 0
2 1 2 0 -24500 90 2300 49500 0 -1000
3 1 2 0 -24500 90 2300 49500 0 -1700
4 1 2 0 -24500 90 2300 49500 0 -3000
5 1 2 0 -24500 90 2300 49500 0 -4000
6 1 2 0 -24500 90 2300 49500 0 -5300
7 1 2 0 -24500 90 2300 49500 0 -6300
8 1 2 0 -24500 90 2300 49500 0 -7500
9 1 2 0 -24500 90 2300 49500 0 -8900
10 1 2 0 -24500 90 2300 49500 0 -10000
11 1 2 0 -24500 90 2300 49500 0 -11000
12 1 2 0 -24500 90 2300 49500 0 -12500
13 1 2 0 -24500 90 2300 49500 0 -13300
14 1 2 0 -24500 90 2300 49500 0 -14300
15 1 2 0 -24500 90 2300 49500 0 -15400
16 1 2 0 -24500 90 2300 49500 0 -16200
17 1 2 0 -24500 90 2300 49500 0 -17500
18 1 2 0 -24500 90 2300 49500 0 -18600
19 1 2 0 -24500 90 2300 49500 0 -19800
20 1 2 0 -24500 90 2300 49500 0 -20700
21 1 2 0 -24500 90 2300 49500 0 -21800
22 1 2 0 -24500 90 2300 49500 0 -23000
```

```
23 1 2 0 -24500 90 2300 49500 0 -23500
24 1 2 0 -24500 90 2300 49500 0 -24800
25 1 2 0 -24500 90 2300 49500 0 -25700
26 1 2 0 -24500 90 2300 49500 0 -26800
27 1 2 0 -24500 90 2300 49500 0 -27800
28 1 2 0 -24500 90 2300 49500 0 -28800
29 1 2 0 -24500 90 2300 49500 0 -29700
30 1 2 0 -24500 90 2300 49500 0 -30400
31 1 2 0 -24500 90 2300 49500 0 -31700
32 1 2 0 -24500 90 2300 49500 0 -32700
33 1 2 0 -24500 90 2300 49500 0 -33900
34 1 2 0 -24500 90 2300 49500 0 -35000
35 1 2 0 -24500 90 2300 49500 0 -36000
36 1 2 0 -24500 90 2300 49500 0 -37500
37 1 2 0 -24500 90 2300 49500 0 -38500
38 1 2 0 -24500 90 2300 49500 0 -40000
39 1 2 0 -24500 90 2300 49500 0 -40900
40 1 2 0 -24500 90 2300 49500 0 -42500
41 1 2 0 -24500 90 2300 49500 0 -43700
42 1 2 0 -24500 90 2300 49500 0 -44800
43 1 2 0 -24500 90 2300 49500 0 -45800
44 1 2 0 -24500 90 2300 49500 0 -46900
45 1 2 0 -24500 90 2300 49500 0 -47800
45 1 2 0 -24500 90 2300 49500 0 -48000
15 2 3 -4750 -15400 0 2100 7300 -850 -15400
15 3 3 -4750 -15400 0 2100 7300 -2200 -15400
15 4 3 -4750 -15400 0 2100 7300 -3200 -15400
15 5 3 -4750 -15400 0 2100 7300 -4100 -15400
15 6 3 -4750 -15400 0 2100 7300 -5300 -15400
15 7 3 -4750 -15400 0 2100 7300 -6300 -15400
15 8 3 -4750 -15400 0 2100 7300 -7300 -15400
15 9 3 -4750 -15400 0 2100 7300 -8500 -15400
21 2 4 -4750 -21900 0 2700 7300 -850 -21900
21 3 4 -4750 -21900 0 2700 7300 -2200 -21900
21 4 4 -4750 -21900 0 2700 7300 -3200 -21900
21 5 4 -4750 -21900 0 2700 7300 -4100 -21900
21 6 4 -4750 -21900 0 2700 7300 -5300 -21900
21 7 4 -4750 -21900 0 2700 7300 -6300 -21900
21 8 4 -4750 -21900 0 2700 7300 -7300 -21900
21 9 4 -4750 -21900 0 2700 7300 -8500 -21900
45 2 5 -4750 -48200 0 2000 7300 -850 -48200
45 3 5 -4750 -48200 0 2000 7300 -2200 -48200
45 4 5 -4750 -48200 0 2000 7300 -3200 -48200
45 5 5 -4750 -48200 0 2000 7300 -4100 -48200
45 6 5 -4750 -48200 0 2000 7300 -5300 -48200
45 7 5 -4750 -48200 0 2000 7300 -6300 -48200
45 8 5 -4750 -48200 0 2000 7300 -7300 -48200
45 9 5 -4750 -48200 0 2000 7300 -8500 -48200
15 10 1  -9600 -32500 90 2300 37400 -9600 -15400
16 10 1 -9600 -32500 90 2300 37400 -9600 -16100
17 10 1 -9600 -32500 90 2300 37400 -9600 -17500
18 10 1 -9600 -32500 90 2300 37400 -9600 -18500
19 10 1 -9600 -32500 90 2300 37400 -9600 -19700
20 10 1 -9600 -32500 90 2300 37400 -9600 -20800
21 10 1 -9600 -32500 90 2300 37400 -9600 -21900
22 10 1 -9600 -32500 90 2300 37400 -9600 -23000
23 10 1 -9600 -32500 90 2300 37400 -9600 -24400
24 10 1 -9600 -32500 90 2300 37400 -9600 -25400
25 10 1 -9600 -32500 90 2300 37400 -9600 -26400
26 10 1 -9600 -32500 90 2300 37400 -9600 -27500
27 10 1 -9600 -32500 90 2300 37400 -9600 -28000
```

```
28 10 1 -9600 -32500 90 2300 37400 -9600 -29000
29 10 1 -9600 -32500 90 2300 37400 -9600 -30400
30 10 1 -9600 -32500 90 2300 37400 -9600 -31700
31 10 1 -9600 -32500 90 2300 37400 -9600 -33000
32 10 1 -9600 -32500 90 2300 37400 -9600 -34000
33 10 1 -9600 -32500 90 2300 37400 -9600 -35300
34 10 1 -9600 -32500 90 2300 37400 -9600 -36200
35 10 1 -9600 -32500 90 2300 37400 -9600 -37200
36 10 1 -9600 -32500 90 2300 37400 -9600 -38800
37 10 1 -9600 -32500 90 2300 37400 -9600 -39500
38 10 1 -9600 -32500 90 2300 37400 -9600 -40600
39 10 1 -9600 -32500 90 2300 37400 -9600 -42100
40 10 1 -9600 -32500 90 2300 37400 -9600 -42900
41 10 1 -9600 -32500 90 2300 37400 -9600 -43900
42 10 1 -9600 -32500 90 2300 37400 -9600 -45000
43 10 1 -9600 -32500 90 2300 37400 -9600 -46200
44 10 1 -9600 -32500 90 2300 37400 -9600 -47100
45 10 1 -9600 -32500 90 2300 37400 -9600 -48200
```

7 Ringraziamenti.

Si ringrazia:
- l'**UFFICIO TECNICO** per aver fornito la mappa del Dipartimento di Elettronica per l'automazione.
- Il dott. **IVAN SERINA** per l'interessamento e l'aiuto fornito.