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

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica
(Prof. Riccardo Cassinis)

Sviluppo di algoritmi per la ricerca di fonti di gas

Elaborato di esame di: **Andrea Frata, Gabriele Bettoni**

Consegnato il: **2 Agosto 2003**

Sommario

Questo lavoro ha come scopo il ripristino dell'hardware e del software necessari all'implementazione di algoritmi per la ricerca di fonti di gas, utilizzando i robot della serie Pioneer 1 dell'Activ Media Robotics e sensori di vapori di solventi organici Figaro Tgs 822.

L'Hardware è stato ripristinato a partire dal lavoro dei progetti sniffer sviluppati presso il laboratorio di robotica avanzata della facoltà di ingegneria dell'università degli studi di Brescia, mentre il software è stato interamente riscritto per poter essere utilizzato nell'ambiente di sviluppo Saphira 8.2.

Le soluzioni adottate permettono all'utente di visualizzare in modo efficace le letture dei sensori e di sperimentare alcuni semplici algoritmi di ricerca di fonti di gas.

Introduzione

Il presente elaborato documenta i problemi riscontrati durante lo sviluppo del software ed il ripristino dell'hardware, contestualmente cerca di chiarire e motivare le scelte effettuate e di indicare all'utente finale in che modo possano essere applicate in un contesto più complesso.

1.1. Mancanza di documentazione dell'Hardware

Prima di descrivere in modo dettagliato il problema affrontato è doveroso premettere che l'hardware impiegato per lo sviluppo del progetto è stato ripristinato partendo da alcuni circuiti realizzati dagli studenti impegnati nel progetto sniffer 1 basato sul robot speedy del laboratorio di robotica avanzata. L'hardware in oggetto non è altro che un semplice circuito di condizionamento del segnale che permette di interfacciare i sensori ai robot della serie pioneer 1 dell' Activ Media.

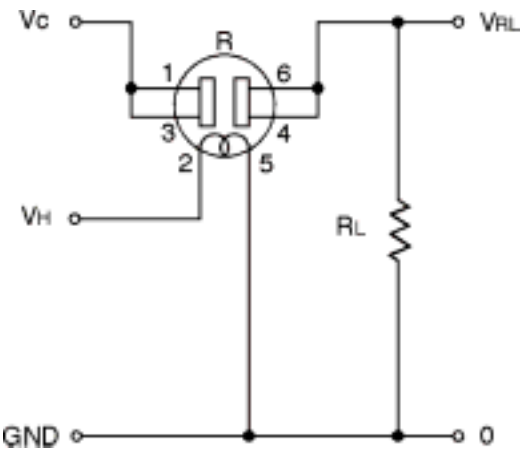
A causa della mancanza di documentazione relativa a questi circuiti abbiamo dovuto analizzare e smontare il sistema di condizionamento in modo da capire in che modo sviluppare il software.

A causa del tempo trascorso abbiamo notato che alcuni componenti risultavano essere dissaldati o mal cablati, per questo motivo abbiamo ricablato parte del circuito e ricostruito il contenitore.

Diamo ora una breve descrizione dei componenti elettronici e dei circuiti utilizzati. I sensori Figaro tgs 822 sono dotati di un elemento sensibile basato su una una membrana semiconduttiva (SnO_2) caratterizzata da bassa conducibilità in aria aperta. Tale conducibilità cresce in funzione della concentrazione di gas in aria, un semplice circuito elettrico permette di associare questa variazione di conducibilità ad un

segnale di tensione. Inoltre il sensore è dotato di un riscaldatore che condiziona il tempo necessario a raggiungere una condizione di funzionamento a regime. Il sensore misura in modo efficace la variazione di concentrazione di alcuni vapori di solventi organici ed altri gas combustibili, per questo motivo risulta essere un sensore general purpose.

Lo schema circuitale del sensore rappresentato a lato mostra le connessioni del sensore con l'elettronica di condizionamento, i piedini 2 e 5 alimentano il riscaldatore del sensore, mentre i piedini 1 e 6 sono connessi all'elemento sensibile *R*.



Il riscaldatore deve essere alimentato con una tensione $V_H = 5V$, e per ottenere un segnale d'uscita V_{RL} compatibile con il convertitore analogico digitale del 68HC11 integrato nei robot Pioneer 1, anche la tensione applicata all'elemento sensibile è $V_C = 5V$. Come si evince dal circuito rappresentato l'elemento sensibile e la resistenza R_L danno luogo ad un partitore di tensione che fa sì che la tensione d'uscita sia:

$$V_{RL} = \frac{R_L}{R_L + R} \cdot V_C$$

L'elettronica di condizionamento per poter essere messa in funzione a bordo del pioneer senza bisogno di batterie ausiliarie è stata progettata per poter funzionare con tensioni di alimentazione compatibili con le batterie del pioneer (11-14V). Per ottenere le tensioni di 5V necessarie al circuito sono stati impiegati due regolatori di tensione 7805 a 5V della ST microelectronics, si è reso necessario l'utilizzo di due regolatori poiché il circuito di condizionamento è stato progettato per condizionare fino a quattro segnali e a causa dell'elevata dissipazione di potenza dei riscaldatori un unico regolatore di tensione sarebbe risultato sottodimensionato.

Per assicurare una efficiente dissipazione del calore connesso all'impiego dei regolatori di tensione sono stati utilizzati due dissipatori passivi di alluminio, posti a contatto dei regolatori stessi.

Il segnale ottenuto in uscita da ciascun sensore utilizzando il partitore di tensione appena trattato risultava ancora poco adatto all'utilizzo diretto del convertitore analogico digitale a causa delle incertezze associate ai resistori ed ai regolatori di tensione utilizzati, per questo motivo il segnale di ciascun sensore viene amplificato mediante una configurazione ad amplificatore non invertente, dotato di trimmer in retroazione. Questa configurazione permette una taratura hardware del circuito di condizionamento che consente di ottenere segnali compatibili con l'ingresso 0-5V della porta analogica del 68HC11 integrato nel pioneer.

Come abbiamo già detto l'elettronica di condizionamento è stata progettata per garantire il funzionamento di al più quattro sensori Figaro Tgs 822, ma la porta analogica del pioneer è unica sui robot non dotati dell'experimenter module come il

robot speedy del laboratorio di robotica avanzata. Per questo motivo si è reso necessario integrare nel circuito di condizionamento anche un multiplexer.

Il multiplexer scelto è il CD4051 che risulta essere un Multiplexer/Demultiplexer per segnali analogici a otto porte, compatibile con le tensioni di alimentazione delle batterie del pioneer. Per comandare questo dispositivo vengono utilizzate le uscite digitali a disposizione nella general I/O port nel pioneer OD5, OD6, OD7.

Per garantire la connessione elettrica dell'elettronica di condizionamento con i robot della serie pioneer 1 dell' ActivMedia sono state integrati due connettori, una porta standard a 26 pin per le connessioni logiche con la general I/O port ed un connettore bipolare per l'alimentazione dell'intero circuito.

Per poter permettere future modifiche dell'hardware e per meglio comprendere il software sviluppato vengono riportate di seguito le connessioni elettriche del connettore a 26 pin del pioneer (per ulteriori informazioni vedi [7]) e dell'elettronica di condizionamento.

Pioneer General I/O Port			
Pin	Label	Pin	Label
1	A/D	14	Gnd
2	Gnd	15	ID3
3	IDT	16	Gnd
4	Gnd	17	OD7
5		18	Gnd
6	Vcc	19	OD6
7	ID7	20	Gnd
8	Vcc	21	OD5
9	ID6	22	Gnd
10	Gnd	23	OD4
11	ID5	24	Gnd
12	Gnd	25	OD3
13	ID4	26	Gnd

Elettronica di Condizionamento			
Pin	Label	Pin	Label
1	Out Mux	14	
2		15	
3		16	
4		17	In A Mux
5		18	
6		19	In B Mux
7		20	
8		21	In C Mux
9		22	
10		23	
11		24	
12		25	
13		26	

Come si può notare utilizzando in modo corretto le uscite digitali del pioneer OD7, OD6, OD5 è possibile comandare tutte le otto porte del multiplexer, ciò permette future espansioni del circuito di condizionamento in termini di maggior numero o tipo di sensori impiegati.

L'esperienza pregressa dei progetti per la ricerca di fonti di gas sviluppati presso il laboratorio di robotica avanzata di Brescia indica che le prestazioni dei sensori figaro tgs822 sono migliorate dalla presenza di sistemi di condizionamento dell'aria, in modo che le letture effettuate dai sensori possano essere considerate "direzionali". Per questo motivo ogni sensore è posto all'interno di un tubo di 40mm di diametro dotato di ventola di aspirazione.

Le ventole utilizzate sono comuni ventole a 12V a bronzine di diametro compatibile con le tubazioni utilizzate. Tali ventole gravano ulteriormente sulla dissipazione di potenza del pioneer, ciò implica una durata delle batterie del robot nettamente inferiore rispetto alle specifiche della casa in assenza di circuiti addizionali.

Per visualizzare la corretta gestione del consumo elettrico sono stati inseriti un interruttore ed un diodo luminoso per controllare lo stato di carica della batteria.

L'elettronica appena descritta è stata racchiusa in un contenitore plastico dalle dimensioni di 18x11,5x7cm, di cui di seguito riportiamo alcune immagini.

Vista Superiore

Vista Frontale (Connessioni Sensori)

Vista Posteriore (Connessione del Pioneer 1)

Il problema affrontato

Il problema richiedeva in primo luogo di poter gestire in maniera efficace i sensori figaro tgs 822, come se fossero direttamente connessi ad una porta digitale del pioneer, per far ciò si è reso necessario mettere a punto l'elettronica di condizionamento vista nel capitolo precedente, ma anche realizzare alcune funzioni software che potessero essere utilizzate per comandare le diverse porte del multiplexer, e la lettura della porta analogica integrata nel 68HC11 del pioneer 1. Una volta letti i valori misurati dai sensori si è cercato di sviluppare alcuni algoritmi che potessero guidare il robot nella ricerca di fonti di gas. Questa fase ha reso manifesti alcuni problemi legati direttamente all'applicazione dei sensori figaro ed ai solventi utilizzati per fare le prove di laboratorio. Abbiamo utilizzato fondamentalmente soluzioni al 20% di alcool etilico e di alcool alimentare, entrambi questi solventi manifestano elevata volatilità, il che porta ben presto alla saturazione dell'ambiente in cui vengono posti. In queste condizioni il problema della lettura dei sensori risulta essere non banale, poiché i sensori vengono ad essere poco direzionali, cioè tendono tutti a raggiungere valori simili.

Un secondo problema è nato a causa delle differenze dei diversi sensori impiegati, che in alcuni casi avevano comportamenti fra loro ben diversi, in primo luogo tutti manifestano una resistenza minima variabile in funzione del riscaldamento e dell'ambiente in cui si eseguono le prove ed in secondo luogo non hanno comportamenti lineari. Ciò comporta che la variazione delle letture in corrispondenza di concentrazioni elevate di gas è ben diversa dalle variazioni a concentrazioni basse.

I sensori utilizzati come si evince dalle specifiche tecniche del costruttore sono caratterizzati da tempi di risposta consistenti, questo genera dei problemi poiché quando il robot è in movimento la lettura effettuata in un determinato istante è proporzionale alla concentrazione di gas di un punto dello spazio che non coincide con la posizione attuale del robot, questo risulta in particolar modo evidente durante il cambiamento di direzione della traiettoria seguita dal robot.

Un altro problema da risolvere è connesso alla necessità di esplorazione di ambienti sconosciuti, ciò implica che per non incappare in ostacoli oltre che alle letture dei sensori figaro tgs 822 bisogna considerare anche le letture dei sonar. Inoltre si devono valutare le dimensioni del robot con i "nasi artificiali" costituiti dai sensori figaro e dal sistema di condizionamento dell'aria, che risultano essere notevolmente superiori sia in lunghezza che larghezza al semplice robot pioner, ciò implica che alcuni comportamenti sviluppati dall'Activ Media come la funzione Swerve (vedi [2]) non possono essere direttamente impiegati nell'applicazione richiesta.

Infine sulla scorta delle esperienze dei primi progetti sniffer veniva richiesto che il comportamento del robot non fosse determinato in modo deterministico, ma in modo più performante con una logica fuzzy utilizzando come variabili di ingresso sia le semplici letture dei sensori, ma anche la loro variazione nel tempo.

La soluzione adottata

La soluzione adottata consiste nello sviluppo di un sistema di controllo del robot basato su logica fuzzy; i problemi affrontati e risolti, come gia ampiamente discusso, sono stati:

- L'acquisizione da parte del calcolatore dei valori dei sensori collegati al robot mediante il comando diretto del multiplexer CD4051.
- L'implementazione dell'algoritmo fuzzy in grado di determinare la direzione di spostamento del robot in funzione dell'intensità di gas rilevata, costruito come "comportamento" per Saphira.
- L'implementazione di un semplice algoritmo in grado di diminuire la velocità del robot e l'angolo di sterzata in funzione dell'intensità di gas.
- L'implementazione di una funzione in grado di disegnare per ogni istante i valori dei sensori nella finestra grafica di Saphira.
- La modifica della funzione Swerve dell'Activ Media, in modo che l'angolo di deviazione del robot quando rileva un ostacolo dipenda anch'esso dalle letture dei sensori di gas.
- La sviluppo di una semplice funzione di taratura per eliminare eventuali offset dovuti ai sensori tgs822.
- La creazione dell'interfaccia verso Colbert.

Analizzeremo in dettaglio ognuno dei problemi citati e il modo in cui sono stati risolti, premettendo che i "nasi" utilizzati sono stati tre, uno orientato verso destra, uno verso sinistra e l'ultimo al centro (vedi immagini del capitolo 4).

1.2. Acquisizione dei valori dei sensori di gas

Come già descritto in precedenza, il robot utilizzato ha la possibilità di acquisire un solo ingresso analogico per volta. È stato necessario quindi l'impiego di un multiplexer per selezionare di volta in volta il sensore da leggere. Per comandare questo dispositivo vengono utilizzate le uscite digitali a disposizione nella general I/O port nel pioneer OD5, OD6, OD7. Per quanto riguarda il software il comando utilizzato è stato:

```
SfROBOT->com2Bytes(ArCommands::DIGOUT,0xFF,porta);
```

Tale comando invia agli output digitali del robot i due byte passati come parametri; il primo identifica la maschera dei bit delle porte di uscita digitali del robot, il secondo il numero della porta che si vuole leggere. Nel nostro caso la porta 0 corrispondeva al sensore di destra, la porta 1 al sensore di sinistra e la porta 2 al sensore centrale. Il comando che effettua la lettura del valore presente sull'ingresso analogico (pin 1 del pioneer general I/O port) è:

```
let=SfROBOT->getAnalog();
```

Tale comando restituisce un unsigned char corrispondente al valore di tensione presente sull'ingresso analogico (0-5V) normalizzato tra 0 e 254; il corrispondente valore di tensione (in volt) si può ottenere dalla formula:

$$V_{an} = \frac{let \times 5}{254}$$

Riportiamo qui il frammento di codice utilizzato per la lettura della porta analogica. Questa parte di codice viene eseguita da Saphira ogni 100ms. Per ogni ciclo leggiamo il valore di una delle porte e settiamo il multiplexer per la lettura della porta successiva. In questo modo riusciamo ad avere un intervallo di almeno 100ms per permettere al multiplexer di cambiare configurazione senza tuttavia interrompere l'esecuzione di Saphira. Nel primo ciclo leggiamo quindi il valore del sensore destro, nel secondo quello del sensore sinistro e nel terzo quello del sensore centrale. Eseguito il terzo ciclo la variabile di appoggio `porta` viene settata uguale a 0 e le letture si ripetono come sopra. Il valore che considereremo corretto sarà quello memorizzato in `tar1, tar2, tar3,` valori calcolati mediando su tre letture di ogni singola porta. Le variabili `off1, off2, off3` sono utilizzate poi per la taratura; le variabili `fuzz, grad` e `tempfuzz` utilizzate invece per la fuzzy logic e approfondite in seguito. Le funzioni `sprintf(…)` memorizzano nelle variabili globali `s1, s2, s3,` le stringhe che verranno poi mostrate a video da un'apposita funzione anch'essa discussa in seguito.

```
//Porta 0
     if (porta==0) {
     val1=val1+(SfROBOT->getAnalog())/3;
     if (ciclo1==3){
          sprintf(s1,"Sensore Dx: %d",val1-off1);
          ciclo1=1;
          tar1=val1;
          val1=0;
          }
     else ciclo1=ciclo1+1;}

//Porta 2
     if (porta==2) {
     val2=val2+(SfROBOT->getAnalog())/3;
     if (ciclo2==3){
          sprintf(s2,"Sensore C: %d",val2-off2);
          ciclo2=1;;
```

```
     tar2=val2;
     val2=0;
     }
else ciclo2=ciclo2+1;}

//Porta 1
if (porta==1) {
val3=val3+(SfROBOT->getAnalog())/3;
if (ciclo3==3){
     sprintf(s3,"Sensore Sx: %d",val3-off3);
     ciclo3=1;
     tar3=val3;
     val3=0;
     fuzz=(tar1-off1-(tar3-off3));
     grad=fuzz-tempfuzz;
     tempfuzz=fuzz;
     }
else ciclo3=ciclo3+1;}

if (porta<2) {porta=porta+1;}
 else porta=0;

SfROBOT->com2Bytes(ArCommands::DIGOUT,0xFF,porta);
```

1.3. Implementazione algoritmo fuzzy

Il concetto di fuzzy logic fu concepito da Lotfi Zadeh, professore all'università di Berkley California. Attraverso tale logica è possibile implementare , sia in hardware che in software, sistemi di controllo in grado di definire valori di uscita basati su ingressi vaghi, imprecisi o in parte mancanti. L'approccio utilizzato da un sistema fuzzy per risolvere un problema ricorda molto il modo in cui le persone prendono decisioni, ma in modo molto più veloce. I sistemi fuzzy sono sistemi di regole IF…THEN… che contengono termini linguistici per esprimere l'appartenenza di una variabile ad un certo fuzzy set. La differenza con i sistemi esperti sta nel fatto che in quelli fuzzy più regole possono essere contemporaneamente attive, inoltre gli input e output sono variabili numeriche e non simboli o termini. Nei sistemi fuzzy è possibile tramite relazioni di input output approssimare qualsiasi funzione, anche non lineare. Possono quindi essere usati per l'identificazione o per il controllo di sistemi, a questo proposito c'è da dire che come i sistemi esperti e le reti neurali, non richiedono un

modello matematico del sistema da descrivere o da controllare. L'implementazione di un algoritmo fuzzy avviene attraverso alcuni passi fondamentali:

- Definizione delle variabili di ingresso.
- Definizione della matrice delle regole (rule matrix).
- Definizione delle funzioni di membership per gli ingressi.
- "Defuzzificazione" delle variabili di uscita.

Nel seguito applicheremo ognuno di tali punti al nostro problema (per ulteriori informazioni in merito alla fuzzy logic vedi [8])

1.3.1. Definizione variabili di ingresso

Il modo in cui abbiamo deciso di affrontare il problema consiste nell'analizzare di volta in volta la differenza tra il valore del "naso" di destra e quello di sinistra, e come tale differenza varia nel tempo. Per fare ciò abbiamo definito due variabili; la prima, denominata `fuzz`, in cui viene memorizzato il valore della differenza di tali valori; la seconda, denominata `grad`, in cui viene memorizzata la differenza tra l'ultimo valore di `fuzz` e quello del ciclo precedente. Ciò è ottenuto attraverso le tre assegnazioni:

```
fuzz=(tar1-off1-(tar3-off3));
grad=fuzz-tempfuzz;
tempfuzz=fuzz;
```

Le variabile `off1` e `off3` servono per la taratura, la variabile `tempfuzz` memorizza il valore di `fuzz` che nel ciclo successivo permetterà di calcolare `grad`.

La logica fuzzy prescrive di utilizzare variabili linguistiche al posto di variabili numeriche. La variabile `fuzz` potrà quindi essere Neg_X, Zero_X o Pos_X; l'appartenenza a ciascuna di tali valori sarà determinata dalla funzione di membership analizzata in seguito. Stesso discorso per la variabile `grad` che potrà essere Pos_grad, Zero_grad o Neg_grad.

1.3.2. Definizione della matrice delle regole

La matrice delle regole determina il termine da applicare all'uscita quando gli ingressi assumono tutti i possibili valori. Tale matrice sarà poi convertita in un insieme di condizioni IF… and … THEN … .

I possibili termini per l'uscita sono Dx, quando si vuole che il robot giri a destra, Sx se si vuole che giri a sinistra e 0 per mantenere una traiettoria rettilinea.

La matrice delle regole nel nostro caso assume il seguente aspetto:

	Neg_X	Zero_X	Pos_X
Pos_grad	zero	Dx	Dx
Zero_grad	Sx	zero	Dx

Neg_grad	Sx	Sx	zero

Essa viene poi codificata nel seguente modo:

```
// Tabella delle regole

     if(Zero_x>0 && Zero_grad>0) {
          if (Zero_x<Zero_grad) out5=Zero_x;
               else out5=Zero_grad;}

     if(Neg_x>0 && Pos_grad>0) {
          if (Neg_x<Pos_grad) out1=Neg_x;
               else out1=Pos_grad;}

     if(Neg_x>0 && Zero_grad>0) {
          if (Neg_x<Zero_grad) out4=Neg_x;
               else out4=Zero_grad;}

     if(Neg_x>0 && Neg_grad>0) {
          if (Neg_x<Neg_grad) out7=Neg_x;
               else out7=Neg_grad;}

     if(Zero_x>0 && Pos_grad>0) {
          if (Zero_x<Pos_grad) out2=Zero_x;
               else out2=Pos_grad;}

     if(Zero_x>0 && Neg_grad>0) {
          if (Zero_x<Neg_grad) out8=Zero_x;
               else out8=Neg_grad;}

     if(Pos_x>0 && Pos_grad>0) {
          if (Pos_x<Pos_grad) out3=Pos_x;
               else out3=Pos_grad;}

     if(Pos_x>0 && Zero_grad>0) {
          if (Pos_x<Zero_grad) out6=Pos_x;
```

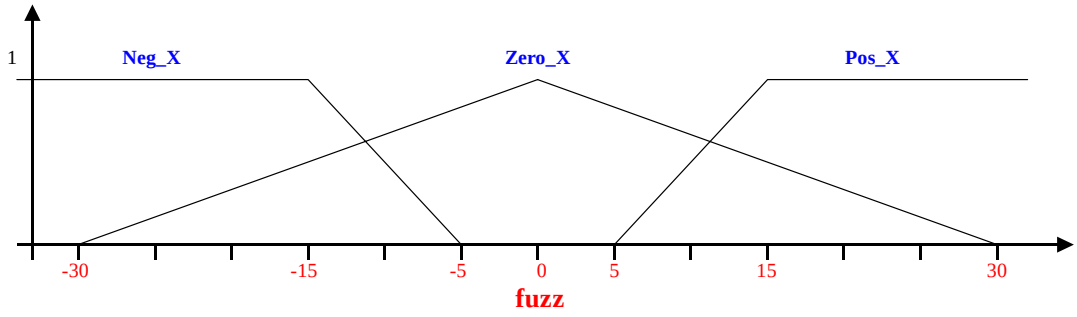
```
            else out6=Zero_grad;}

if(Pos_x>0 && Neg_grad>0) {
     if (Pos_x<Neg_grad) out9=Pos_x;
          else out9=Neg_grad;}
```

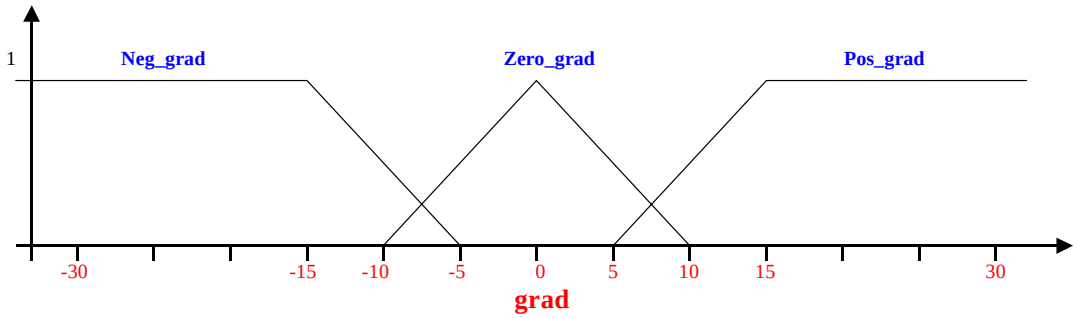
Le variabili `out1...out9` determinano il grado di applicabilità di ogni regola considerando il valore minimo tra i valori delle variabili di ingresso calcolati dalla funzione di membership, come sarà chiarito tra breve. I vari valori ottenuti vengono combinati con il metodo dei minimi quadrati per ottenere i valori di ogni termine di uscita.

```
Dx=sqrt((out2*out2)+(out3*out3)+(out6*out6));
zero=sqrt((out5*out5)+(out9*out9)+(out1*out1));
Sx=sqrt((out4*out4)+(out7*out7)+(out8*out8));
```

1.3.3. Definizione funzioni di membership

La definizione della funzione di membership ha lo scopo di assegnare a ciascuno dei possibili valori degli ingressi (Neg_X, Zero_X, Pos_X, Neg_grad, Zero_grad, Pos_grad) un determinato valore in funzione della variabile di ingresso (`fuzz`, `grad`). Nel nostro caso abbiamo utilizzato funzioni di membership triangolari, come si evince chiaramente dai grafici riportati:





Si noti che per determinati valori di `fuzz` e `grad` può succedere che agli ingressi vengano assegnati due valori, ognuno con un determinato "peso" stabilito dalle funzioni di membership. Questo comporta che in particolari condizioni più regole possano essere attive contemporaneamente, ed è per tale motivo che il valore dell'uscita è determinato combinando insieme (con il metodo dei minimi quadrati) tutti i "pesi" di ogni singola regola, come visto nel paragrafo precedente.

Come fatto prima, la funzioni di membership vengono tradotte nel seguente modo per poter essere incorporate nel codice:

```
// Determinazione del valore delle funzioni di membership

// Variabile fuzz
if (fuzz>=30) {Pos_x=1;
               Neg_x=0;
               Zero_x=0;};
if (fuzz<30 && fuzz>=15) {Pos_x=1;
                          Zero_x=1-fuzz/30;
                          Neg_x=0;};
if (fuzz<15 && fuzz>=5)  {Pos_x=(fuzz/10-0.5);
                          Zero_x=(1-fuzz/30);
                          Neg_x=0;};
if (fuzz<5 && fuzz>=0)   {Pos_x=0;
                          Zero_x=(1-fuzz/30);
                          Neg_x=0;};
if (fuzz<0 && fuzz>=-5)  {Pos_x=0;
                          Zero_x=(fuzz/30+1);
                          Neg_x=0;};
```

```
if (fuzz<-5 && fuzz>=-15)     {Pos_x=0;
                               Zero_x=(fuzz/30+1);
                               Neg_x=(-fuzz/10-0.5);};
if (fuzz<-15 && fuzz>=-30)    {Pos_x=0;
                               Zero_x=(fuzz/30+1);
                               Neg_x=1;};
if (fuzz<=-30) {Pos_x=0;
                Zero_x=0;
                Neg_x=1;};

//variabile grad
if (grad>=15)  {Pos_grad=1;
                Neg_grad=0;
                Zero_grad=0;};
if (grad<15 && grad>=10) {Pos_grad=(grad/10-0.5);
                          Zero_grad=0;
                          Neg_grad=0;};
if (grad<10 && grad>=5)  {Pos_grad=(grad/10-0.5);
                          Zero_grad=(1-grad/10);
                          Neg_grad = 0;};
if (grad<5 && grad>=0)   {Pos_grad=0;
                          Zero_grad=(1-grad/10);
                          Neg_grad=0;};
if (grad<0 && grad>=-5)  {Pos_grad=0;
                          Zero_grad=(grad/10+1);
                          Neg_grad=0;};
if (grad<-5 && grad>=-10)     {Pos_grad=0;
                               Zero_grad=(grad/10+1);
                               Neg_grad=(-grad/10-0.5);};
if (grad<-10 && grad>=-15)    {Pos_grad=0;
                               Zero_grad=0;
                               Neg_grad=(-grad/10-0.5);};
if (grad<=-15) {Pos_grad=0;
                Zero_grad=0;
                Neg_grad=1;};
```

1.3.4. "Defuzzificazione" della variabile d'uscita

La "defuzzificazione" della variabile di uscita avviene attraverso il calcolo del "centroide fuzzy" dell'area selezionata sul grafico dell'uscita dai valori di Dx, zero e Sx. Il valore finale da applicare è dato dalla formula:

$$forza\_ang = \frac{\left[\left(Dx \times \left(-100\right)\right) + \left(zero \times 0\right) + \left(Sx \times 100\right)\right]}{\left(Dx + zero + Sx\right)}$$

forza_ang rappresenta la percentuale del valore max dell'angolo da applicare. Il tutto tradotto in codice diviene:

```
// Definizione della variabile d'uscita
forza_ang=(Dx*(-100)+Sx*100)/(Dx+zero+Sx);

// Angolo di sterzata desiderato
angleTurn2=(int)(forza_ang*Turn/100);
```

Dove con `angleTurn2` si intende l'angolo che verrà effettivamente applicato al robot, mentre `Turn` è una variabile che determina l'angolo massimo di sterzata, determinata con modalità che vengono descritte nel capitolo seguente.

1.4. Algoritmo per la variazione della velocità e dell'angolo di sterzata in funzione dell'intensità del gas.

L'algoritmo che ci apprestiamo a descrivere è molto semplice e intuitivo, ma allo stesso tempo permette di migliorare sensibilmente il comportamento del robot. Esso è fondamentalmente un algoritmo "empirico", nel senso che i valori che compariranno sono stati determinati con prove dirette sul campo. Abbiamo considerato in questo caso le letture fornite dal sensore centrale (`tar2`) e le abbiamo suddivise in alcune fasce caratteristiche, in ognuna delle quali viene impostata una determinata velocità e un determinato angolo. Il codice utilizzato è il seguente:

```
// Algoritmo per la variazione della velocità e
// dell'angolo di sterzata in funzione dell'intensità di
// gas rilevata

if (tar2<=30) {speed=70;Turn=8;};
if (tar2<=80 && tar2>30) {speed=50;Turn=7;};
if (tar2<=120 && tar2>80) {speed=30;Turn=6;};
```

```
if (tar2<=165 && tar2>120) {speed=20;Turn=5;};
if (tar1>=175 || tar2>=175 || tar3>=175)
                                {speed=0;Turn=4;};
if (tar2>=170) {sfMessage("TROVATO!!!");speed=0;Turn=0;};
```

1.5. Funzione di disegno in Saphira

Anche in questo caso la funzione sviluppata è semplice; abbiamo utilizzato la funzione `draw(SfWin *w)` appartenente alla classe SfArtifact. Essa viene chiamata ad ogni ciclo e gestisce la finestra grafica di Saphira.

Anche in questo caso il codice utilizzato è:

```
// Funzione di disegno dell'interfaccia grafica

SFEXPORT void
SfDisAction::draw(SfWin *w)
{
char ang[100];
char grad1[100];

w->PenColor(sfColorRed);

w->Text(s3, SfROBOT->getX()-600, SfROBOT->getY()+1000);
w->Text(s2, SfROBOT->getX()-600, SfROBOT->getY()+750);
w->Text(s1, SfROBOT->getX()-600, SfROBOT->getY()+500);

sprintf(ang,"Potenza: %3.3f",forza_ang);
w->Text(ang,SfROBOT->getX() -600,SfROBOT->getY()-750 );

sprintf(grad1,"Gradiente: %f ",grad);
w->Text(grad1,SfROBOT->getX()-600,SfROBOT->getY()-1000);

}
```

la funzione `Text(char _str, double x, double y)` disegna la stringa `_str` nella posizione `x`, `y`.

1.6. La funzione Swerve2

Abbiamo apportato una piccola modifica al codice dell'Activ Media riguardo alla funzione Swerve. Nel codice originario infatti quando la funzione veniva chiamata come parametro si passava anche la direzione nella quale il robot avrebbe deviato in seguito alla rilevazione di un ostacolo. Ciò però rende tale algoritmo inutilizzabile per la nostra applicazione perché il robot devierebbe nella direzione passata come parametro anche se la sorgente di gas si trovasse dalla parte opposta. La modifica da noi apportata rende la direzione di deviazione della Swerve dipendente dall'angolo imposto dalla fuzzy logic prima studiata. Abbiamo inoltre variato il parametro `radius` in modo che la Swerve2 tenesse in considerazione anche la presenza dei "nasi" elettronici.

Il codice completo della Swerve2 è riportato di seguito:

```
// Funzione utilizzata per evitare gli ostacoli secondo
// la direzione con maggior intensità di gas; questa
// funzione è stata creata basandosi sulla swerve
// dell'activmedia



class SfSwerve2Action : public ArAction, public
SfArtifact
{
 public:

SFEXPORT SfSwerve2Action(int turn, int avoid, int
                         standoff);
SFEXPORT ~SfSwerve2Action() {};
//Definizione dell'azione
SFEXPORT virtual ArActionDesired *fire(ArActionDesired
                                  currentDesired);
// Interfaccia verso Colbert
static SfSwerve2Action *invoke(int turn, int avoid, int
                         standoff);
// Interfaccia grafica
SFEXPORT void draw(SfWin *w);

protected:          // arguments
```

```
  int turnDir;      // direction to turn
  int avoidDist;    // close contact distance to robot
  int standDist;    // first notice distance to robot

 private:
  ArActionDesired myDesired;//what the action wants to do
  int angleTurn;         //what we found for an angle
  int standOff;          //when we're near an obstacle
  int nearDist;          //full throttle on turning
  int frontDist;    //how close we actually are in front
};



SFEXPORT
SfSwerve2Action::SfSwerve2Action(int turn, int avoid, int
standoff)
  : ArAction("Swerve2")
{
  angleTurn=-5;     //impostiamo l'angolo iniziale
  turnDir = turn;
  avoidDist = avoid;
  standDist = standoff;
}



//
// Body of action
// First get sensor readings on the side
// Check closest approach, using two front readings
//

SFEXPORT ArActionDesired *
SfSwerve2Action::fire(ArActionDesired d
)
```

```
{
// reset the actionDesired (must be done)
myDesired.reset();

// set up parameters
int radius = (int)SfROBOT->getRobotRadius()+100;

// Definizione dell'angolo per evitare l'ostacolo
if (angleTurn2 > 0) angleTurn = +7;
if (angleTurn2 < 0) angleTurn = -7;

//how far away on first contact
standOff = radius + standDist;

//how close before full avoidance
nearDist = radius + avoidDist;

double x, y, strength;

// get sensor readings in front of robot
SfSonarDevice *sd = Sf::sonar(); // get the device
if (!sd)
    frontDist = 5000;         // large value, no return
else
    {
      ArPose rpose = SfROBOT->getPose();
      frontDist  =  sd->getCurrent()->getClosestBox(100,  -
radius-100,standOff + 200, radius+100,rpose,10000, NULL);
    }

if (frontDist < standOff) // we've found an obstacle
    {
    x = (double)(frontDist - nearDist);
    if (x < 0) x = 0;
    y = (double)(standOff - nearDist);
```

```
    // get stronger as we get closer
    strength = 1.0 - (x/y);
    myDesired.setHeading(SfROBOT->getTh() + angleTurn,1);
    myDesired.setVel(0.0, 1);
    }
  return &myDesired;
}

// drawing function, indicate robotsstate of behavior
graphically

SFEXPORT void
SfSwerve2Action::draw(SfWin *w)
{
  int radius = (int)SfROBOT->getRobotRadius()+100;
  int dd = frontDist;
  if (dd < 1000)
    w->PenColor(sfColorRed);
  else
    {
      w->PenColor(sfColorGreen);
      dd = standOff + 200;
    }

  w->Rectangle(100, -radius-100, dd, radius+100, NULL);

  w->PenColor(0);
}
```

1.7. Funzione di taratura dei sensori

La funzione di taratura serve per portare i valori dei sensori ad un punto iniziale uguale per tutti, in modo da eliminare eventuali offset presenti. Essa può essere chiamata direttamente dalla riga di comando di Saphira scrivendo semplicemente tara(). Tale funzione andrebbe usata rigorosamente in assenza di sorgenti di gas e dopo aver aspettato un tempo ragionevole per permettere al riscaldatore interno ai sensori di arrivare a regime. Tale funzione porta il valore di tutti i sensori a 10 nell'istante in cui è chiamata, memorizzando nelle variabili globali `off0`…`off3` la differenza tra il valore vero del sensore e il valore 30. La funzione di taratura può

essere richiamata più volte. Per tornare ad utilizzare i valori veri delle letture bisogna sospendere da Saphira l'Activity sniffer e successivamente riattivarla.

Il semplice codice della funzione tara() è il seguente:

```
void tara(void)
{
     off0=tar0-30;
     off1=tar1-30;
     off2=tar2-30;
     off3=tar3-30;
}
```

Le variabili `off0`…`off3` verranno poi utilizzate nel calcolo della variabile `fuzz.`

1.8. Interfaccia verso Colbert

Tutte le funzioni viste devono essere esportate in Saphira per poter poi essere utilizzate dalle action definite in Colbert.

Il codice che permette questo è il seguente:

```
//
// Interface to Colbert
//
// This static function returns a behavioral action
// object,with arguments that can be set from Colbert
//

SfSwerve2Action *
SfSwerve2Action::invoke(int turn, int avoid, int
standoff)
{
  return new SfSwerve2Action(turn, avoid, standoff);
}

SfDisAction *
SfDisAction::invoke(void)
{
  return new SfDisAction();
```

```
}

// define interface to Colbert here

SFEXPORT void
sfLoadInit (void)
{
sfAddEvalAction("fuzzy", (void *)SfDisAction::invoke, 0);
sfAddEvalFn("tara", (void *)tara,sfVOID, 0);
sfAddEvalAction("Swerve2",(void*)SfSwerve2Action::invoke,
3, sfINT, sfINT, sfINT);
}
```

In coda a questo capitolo riportiamo il codice del makefile utilizzato per compilare il programma e uno script in Colbert utilizzato per caricare da Saphira i "comportamenti" da noi definiti.

```
#
# Sniffer2
# makefile
#
# Questo file è stato realizzato da
#
# Andrea Frata           Matr. 37921
# Gabriele Bettoni       Matr. 37610
#
# come parte del progetto
#
# Sniffer2
#
# per il corso di
#
# Robotica
# Prof. Riccardo Cassinis
# Università degli studi di Brescia
# Facoltà di Ingegneria
# A.A. 2002/03
#
```

```
# Requisiti software necessari per la compilazione:
# + che siano stati installati i pacchetti:
#   - Saphira 8.x
#   - Aria 1.x
# + che siano definite le veriabili di ambiente
#   - SAPHIRA
#   - ARIA
#   relative alle directory in cui sono stati
#   installati i rispettivi pacchetti (vedere il
#   file readme.txt nella directory di installazione di
#   Saphira per maggiori informazioni)
# + la presenza del compilatore C++
#   - GNU gcc-2.95.3 per il sistema operativo Linux
#   - Microsoft Visual C++ 6.x o successivi per i
#     sistemi operativi Windows 9x/ME/2000.
#
#
############################################################

SRCD = ./
OBJD = ./
INCD = ../../ohandler/include/
BIND = ../../bin/
LIBD = ../../lib/
ARIAD = ../../../Aria/

# check which OS we have
include $(INCD)os.h

SHELL = /bin/sh

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

############################################################
all: $(LIBD)sniffer2.so
```

```
     touch all

$(OBJD)sniffer2.o: $(SRCD)sniffer2.cpp
$(CC) $(CFLAGS) -c $(SRCD)sniffer2.cpp $(INCLUDE) -o
$(OBJD)sniffer2.o


$(LIBD)sniffer2.so: $(OBJD)sniffer2.o
$(LD) $(SHARED) -o $(LIBD)sniffer2.so $(OBJD)sniffer2.o
```

Questa invece l'activity utilizzata:

```
loadlib sniffer2;
remove Sniffer;

act Sniffer()
{
  behaviors;        // turn on behavioral action
  remove Swerve2;   // just in case it's already started
  start Swerve2(sfRIGHT,500, 700) priority 1000 noblock;
  remove fuzzy;
  start fuzzy priority 15 noblock;
  waitfor 0;        // infinite wait
}

start Sniffer suspend;
```

Modalità operative

Le funzioni e le activity viste nel paragrafo precedente contenute nella libreria sniffer2 come del resto le librerie predefinite di Saphira ed Aria, hanno la possibilità di essere utilizzate sia su calcolatori provvisti dei sistemi operativi della famiglia Windows, sia su calcolatori con sistema operativo Linux.

Le funzioni da noi sviluppate però sono state testate e sviluppate in ambiente Linux ed in particolare utilizzando calcolatori dotati di distribuzioni Mandrake 9.0 e 9.1 e Red Hat 9.0. Qualora si volessero utilizzare le funzioni in ambiente Windows, il codice sorgente deve essere compilato utilizzando Microsoft Visual C++ v.6.x. Per

quanto concerne le modalità di compilazione rimandiamo al manuale dell'ambiente di sviluppo utilizzato.

1.9. Componenti necessari

La messa in esercizio del sistema descritto nel presente elaborato richiede l'utilizzo di un robot Pioneer 1 dell'Activ Media Robots, un calcolatore dotato di sistema operativo Linux e di porta seriale standard, un cavo di connessione seriale ed infine l'elettronica di condizionamento descritta nel capitolo 1.

1.10. Modalità di installazione

Riportiamo di seguito le operazione necessarie all'installazione dell'hardware e del software per verificare gli algoritmi sviluppati.

Per quanto concerne il software è sufficiente copiare lo shared object "sniffer2.so" che si trova della directory obj del cd-rom allegato alla presente relazione in /usr/local/Saphira/lib. In seguito è necessario che la libreria sniffer2 venga caricata in Colbert (Vedi [5]), con il comando "loadlib sniffer2". A questo punto è possibile richiamare le activity e le funzioni analizzate nel capitolo precedente direttamente da Saphira.

Per permettere all'utente di caricare la libreria e di far partire direttamente l'azione sniffer abbiamo realizzato l'activity file riportato nel capitolo precedente che può essere caricata direttamente da saphira attraverso il menu file: load activity file. .

Nel caso in cui si preferisca ricompilare il codice sorgente è sufficiente copiare in /usr/local/Saphira/tutor la directory sniffer2 con tutto il suo contenuto, eseguire il makefile in /sniffer2 e le librerie saranno automaticamente create nella directory /usr/local/Saphira/lib.

Le distribuzioni Linux più recenti installano di default il gcc versione 3.2.2 o successiva, questo compilatore permette di compilare i file sorgente, ma gli shared object così realizzati non possono essere caricati in Saphira 8.2. Per evitare questi problemi il compilatore come riportato nelle note di installazione di Saphira deve essere il gcc versione 2.95.3.

Per quanto concerne l'hardware in primo luogo bisogna svitare due viti d'assemblaggio del Pioneer 1 come indicato nella figura seguente.

Una volta rimosse le due viti indicate, si può procedere al fissaggio della struttura portante dei sensori Figaro Tgs822:

Per facilitare l'operazione d'assemblaggio i due supporti laterali della struttura rappresentata possono essere rimossi, in questo modo è più facile serrare in modo più preciso le viti del case del robot.

Ogni naso è fissato alla struttura mediante una vite ed un dado, per questo motivo prima della messa in esercizio del robot è utile serrare tutti i dadi di fissaggio, per evitare che questi si allentino con le vibrazioni dovute al movimento del robot.

1.11. Avvertenze

Durante lo sviluppo del software abbiamo rilevano un malfunzionamento della connessione fra robot e calcolatore di controllo quando si usano particolari configurazioni software e adattatori seriale - USB.

In particolare utilizzando un calcolatore portatile Gericom Overdose S con sistema operativo Linux mandrake 9.1 e RedHat 9.0 ed un adattatore seriale prolific USR2309, la lettura della porta analogica risultava essere sempre nulla, mentre le letture d'altre informazioni di stato del robot come la tensione della batteria venivano eseguite correttamente.

Conclusioni e sviluppi futuri

L'esperienza fatta mostra come gli algoritmi sviluppati possano essere impiegati con successo nella ricerca di fonti di gas in ambienti chiusi in cui non siano presenti consistenti correnti d'aria. A causa dell'impossibilità di utilizzare un calcolatore portatile per effettuare le prove non abbiamo potuto provare il sistema in ambienti più ampi aventi conformazioni più complesse del laboratorio di robotica avanzata. Per questo motivo è auspicabile in futuro il testing del sistema sviluppato in ambienti più ampi e con configurazioni di sensori anche più complesse, in primo luogo si potrebbe pensare all'aggiunta di alcuni sensori nella parte posteriore del robot o alla loro diversa disposizione nella parte frontale.

Per quanto concerne il software potrebbe essere interessante cercare di sviluppare un numero più elevato di algoritmi per la ricerca di fonti di gas e renderli tutti contemporaneamente attivi attribuendo a ciascuno un peso ed una priorità diversi per migliorare il comportamento del robot durante la fase di ricerca.

Bibliografia

[1] Konolige, K.: "Saphira Software Manual", Saphira Version 8 (Saphira/Aria integration), 2001

[2] Konolige, K.: "SAPHIRA TUTORIAL", Compiling, Loading and Debugging C++ Files: Unix Systems Sofware version 8.0 (Saphira/Aria), September 2001

[3] Doxygen 1.2.10: "Saphira Reference Manual 8.2.0", February 2003

[4] Doxygen 1.2.10: "Aria Reference Manual 1.2.0", February 2003

[5] Konolige, K.: "Colbert User Manual", Software version 8.0a (Saphira/Aria), September 2001

[6] Konolige, K.: "COLBERT: A Language for Reactive Control in Saphira", February 2003

[7] ActiveMEDIA ROBOTICS: "Pioneer Mobile Robot Operation Manual, Edition 2", January 1998

[8] Steven, D. Kaehner: "Fuzzy logic – an introduction", Seattle Robotics Society, 1998.

Indice