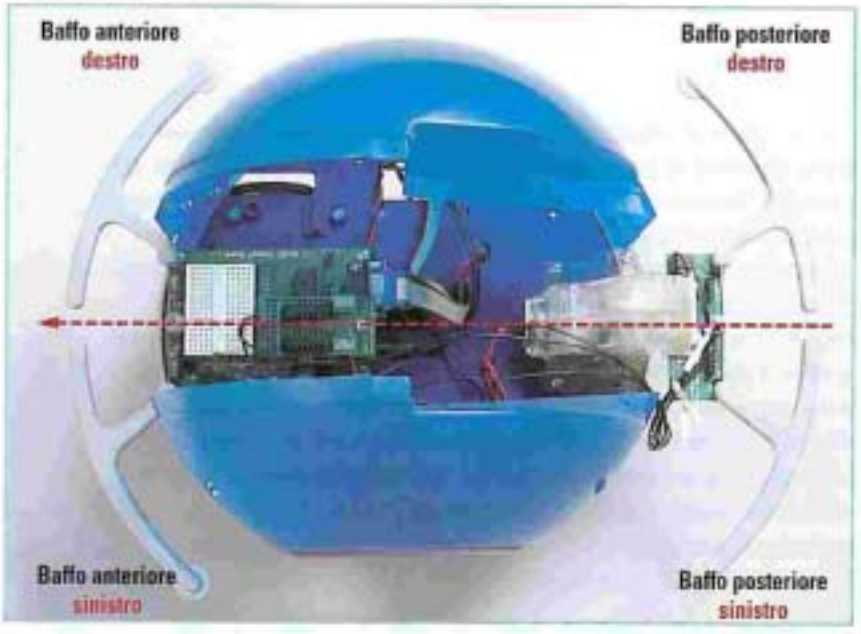
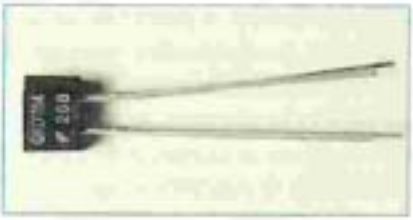
Sensibilità per la navigazione

Come il precedente, il sensore allegato a questo fascicolo andrà alloggiato in una delle sedi centrali del DeA Line Follower (illustrato a pag. 157). Tale dispositivo potenzierà le 'doti percettive' del tuo robot: un obiettivo perseguito anche grazie ad altri componenti a tua disposizione, come il cicalino e il circuito DeA Back Sensors (rispettivamente presentati a pag. 129 e 143). Quest'ultimo, in particolare, accoglie la seconda coppia di baffi (il cui montaggio è illustrato alle pagg. 149-152) che ti permetterà ora di dedicarti con maggiore attenzione ai programmi per la navigazione autonoma, lasciati in sospeso.

Il circuito DeA Back Sensors, montato sulla slitta verticale del manipolatore, è alternativo all'uso della pinza. Per quanto affascinante, quest'ultima è, tutto sommato, uno strumento poco 'robotico': in mancanza di sofisticati sensori visivi, infatti, la pinza è fortemente vincolata all'uso del telecomando. I sensori IR di cui dispone il robot, inoltre, non possono essere di grande aiuto nel determinare la posizione o l'assetto di un oggetto che debba essere manipolato. Sul fronte della navigazione, invece, la prestazione degli stessi sensori IR si rivela ottima per rilevare la presenza di un ostacolo.





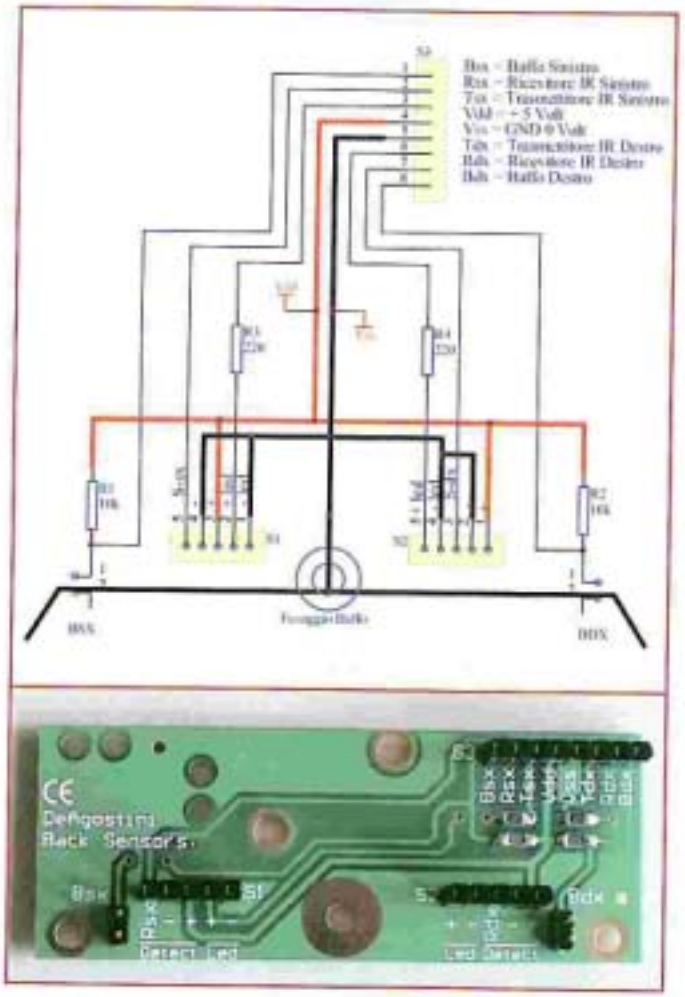
Sopra. Il secondo sensore monoblocco IR, allegato a questo fascicolo.

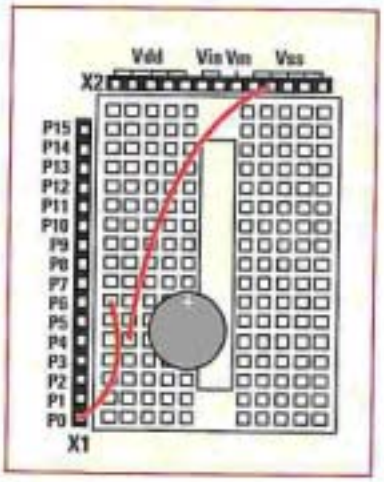
In basso a sinistra. Il robot con la doppia coppia di baffi (la freccia indica il verso convenzionale di avanzamento).

Questo significa che il tuo è essenzialmente un robot mobile, programmabile per la navigazione autonoma. Di fatto, poi, può disporre anche di un manipolatore, la cui gestione autonoma, però, è possibile solo in condizioni rigidamente prestabilite e fissate (come avviene per certi robot industriali inseriti in catena di montaggio) in cui, ad esempio, il robot sia in condizione di sapere con esattezza la posizione in cui trovarsi per poter chiudere la pinza e afferrare un oggetto. Ciò che, però, distingue un robot da un dispositivo programmabile e da un giocattolo telecomandato è l'autonomia dall'intervento umano, durante l'esecuzione di un compito. Vale dunque la pena concentrarsi sugli algoritmi di navigazione, limitando l'uso del telecomando alla gestione delle emergenze e dei casi difficili.

Le fasi di programmazione

A proposito della navigazione, il circuito DeA Back Sensors (a destra, lo schema e la foto) gioca un duplice ruolo: essa, infatti, permette di implementare nella parte posteriore del robot non solo il secondo paio di baffi, ma anche ulteriori sensori a infrarossi (nelle apposite sedi, analoghe a quelle della scheda madre), massimizzando così la copertura sensoriale perimetrale del robot. È inoltre importante notare che, fissando la scheda alla base della slitta verticale del manipolatore, è possibile variarne l'altezza utilizzando il corrispettivo motore collegato a X3 sulla scheda di controllo motori (e a P3 e P5). In questo modo, dunque, i sensori posteriori acquistano una maggiore importanza di quelli anteriori, in quanto permettono la scansione dello spazio operativo anche in altezza. Tale versatilità dei sensori posteriori, unita alle caratteristiche formali del robot (la cui base è pressoché circolare), comporta che, nei fatti, la distinzione tra una porzione anteriore e una posteriore del robot si attenui un po', riducendosi in fondo a una questione di punti di vista: per chiarezza, però, continueremo a mantenere la convenzione per cui i sensori presenti sul DeA Back Sensors saranno indicati come sensori posteriori. Va inoltre tenuto presente che la denominazione riportata sulla scheda (Bsx per il baffo sinistro e Bdx per il destro), dipende da tale convenzione: Bsx, ad esempio, è realmente sul lato sinistro del robot solo se esso avanza; invertita la marcia, lo stesso sensore sarà sul lato destro.





Prima di vedere come tali informazioni si traducono in termini di programmazione, è necessario introdurre un altro componente che sarà utilizzato nel programma, vale a dire il cicalino. Tecnicamente, si tratta di un piezo speaker, ossia di un emettitore acustico piezoelettrico in grado di tradurre un segnale elettrico (proveniente da un circuito) nel movimento meccanico che produce la vibrazione e, dunque, il suono. Per prima cosa, allora, dovrai realizzare sulla breadboard il circuito (a sinistra) che collega il cicalino a una porta e a massa. In particolare, il piedino più lungo del cicalino (ossia il polo positivo, contrassegnato con un + sul bordo del componente) deve essere connesso, tramite un cavetto, alla porta P0; l'altro piedino, invece, sarà collegato a Vss, sempre grazie a un cavetto. Si è deciso di utilizzare la porta P0, attualmente libera, in modo da poter gestire lo speaker indipendentemente da altri componenti. Il programma che segue, infatti, utilizzerà il cicalino come 'richiesta d'aiuto' del robot in caso di difficoltà di navigazione, nonché come segnalatore di un reset indesiderato, nel caso in cui il programma riparta da capo inavvertitamente.

Digita il listato a destra nell'area di editing, con l'accortezza di riportare esattamente la lunga istruzione di branch (per ragioni di spazio, infatti, la porzione tratteggiata in rosso nel listato è incompleta; quella completa, invece, è trascritta a fondo pagina). Noterai che le prime righe del listato, introdotte da un apice ('), non sono altro che un utile riassunto dello stato delle porte (libere o occupate dai rispettivi dispositivi), ininfluente ai fini della programmazione. Segue quindi la dichiarazione di quattro variabili, che vedremo in seguito nel dettaglio. Mandando in esecuzione il programma, lo speaker emetterà un primo suono, prodotto dalla routine Reset. Questo segnale acustico, apparentemente inutile all'avvio del programma, si rivela invece un prezioso 'campanello d'allarme' in fase di navigazione: se, infatti, il robot dovesse resettarsi automaticamente (per esempio, nel caso in cui ci fosse un eccessivo sfruttamento delle batterie e, dunque, un calo della tensione di alimentazione), il programma ripartirebbe da capo, a tua insaputa, invece di procedere nel suo corso normale. Grazie alla routine Reset, invece, sarai in grado di riconoscere il 'problema' per quello che è (un reset indesiderato) e non invece come un difetto di programmazione. Torniamo quindi al listato. Il suono viene emesso grazie all'istruzione freqout 0, 200, 3000, preceduta da output 0 che setta, una volta per tutte, la porta P0 come uscita. La sintassi di freqout prevede che all'istruzione (freqout) segua l'indicazione della porta (0), la durata (200) dell'impulso e la frequenza (3000) di quest'ultimo. Il programma, dunque, invia allo speaker (collegato a P0) un segnale impulsivo della durata di 0.2 secondi (200 millisecondi) e frequenza 3 KHz (3000 Hz). Per inciso, vale la pena ricordare che l'hertz (Hz) è l'unità di misura adottata per le frequenze ed è l'inverso del secondo (vale infatti l'identità 1 Hz=1/sec); inoltre, il suono è legato strettamente al valore di frequenza impostato come parametro: a valori di frequenza maggiori corrispondono infatti suoni più acuti e intensi. Nella subroutine Errore, ad esempio, le istruzioni freqout 0, 2000, 4000 e pause 100 producono il susseguirsi di un suono più acuto del precedente (4000>3000), per 2 secondi (2000 ms), seguito da una breve pausa, ottenendo così un effetto simile a una sirena: si tratta del 'campanello d'allarme' cui si accennava. Grazie al ciclo Errore... goto Errore, infatti, la routine pone il robot in *stand by* (cioè in attesa), finché (if) non sia stato premuto intenzionalmente il tasto di start/stop (il cui stato, in2, viene memorizzato nella variabile PBin). Questa stessa operazione deve essere fatta anche all'inizio del programma (grazie alla routine Start): soltanto dopo aver premuto il tasto start/stop (PBin=0), comincia il programma di navigazione vero e proprio (Main). La gestione dei baffi posteriori, identica a quella dei baffi anteriori, dipende dalla costruzione della variabile stato_baffi che, memorizzando in quattro bit (da bit0 a bit3) le porte relative alle due coppie di baffi (P10 e P11 per i baffi posteriori; P4 e P6 per quelli anteriori), realizza la fusione dei due sensori. Questa operazione, realizzata dalla routine RilevaBaffi, consente poi di usare l'istruzione branch per decidere il movimento da far eseguire al robot.

```
'{$STAMP BS2}
'*****************************************************************
'P0   piezo speaker
'P1   libera
'P2   Pulsante
'P3   Motore 2 M2A
'P4   Baffo Destro Anteriore (X4)
'P5   Motore 2 M2B
'P6   Baffo Sinistro Anterore (X5)
'P7   libera
'P8   libera
'P9   libera
'P10  Baffo Destro Posteriore (Bdx)
'P11  Baffo Sinistro Posteriore (Bsx)
'P12  Servomotore Sinistro
'P13  Servomotore Destro
'P14  libera
'P15  libera
'
PBin var in2

stato_baffi     var   nib
contatore       var   byte
direzione       var   bit



direzione=0

Reset:
      output 0
      freqout 0, 200, 3000

Start:
      If PBin= 0 then Main
      goto Start

Main:
      gosub RilevaBaffi
      branch stato_baffi,[Errore,Errore,Errore,Cambia_Direzione,Err
goto Main


RilevaBaffi:
      stato_baffi.bit0=in10
      stato_baffi.bit1=in11
      stato_baffi.bit2=in4
      stato_baffi.bit3=in6
      return


Errore:
      freqout 0, 2000, 4000
      pause 100
      If PBin= 0 then Main
      goto Errore


Avanti:
      if direzione=1 then Avanti_Inv
      for contatore = 0 to 10
      pulsout 12, 1000
      pulsout 13, 500
      pause 20
      next
      goto Main
Avanti_Inv:
      for contatore = 0 to 10
      pulsout 12, 500
      pulsout 13, 1000
      pause 20
      next
      goto Main


Destra:
      for contatore = 0 to 10
      pulsout 12, 1000
      pulsout 13, 1000
      pause 20
      next
      goto Main

Sinistra:
      for contatore = 0 to 10
      pulsout 12, 500
      pulsout 13, 500
      pause 20
      next
      goto Main


Cambia_Direzione:
      direzione=direzione+1
      goto Avanti
```

1 1

[Errore, Errore, Errore, Cambia_Direzione, Errore, Errore, Errore, Destra, Errore, Errore, Errore, Sinistra, Cambia_Direzione, Sinistra, Destra, Avanti]

LE FASI DI PROGRAMMAZIONE

Valore branch	Bsx anteriore	Bdx anteriore	Bsx posteriore	Bdx posteriore	Routine/azione
0	0	0	0	0	Errore
1	0	0	0	1	Errore
2	0	0	1	0	Errore
3	0	0	1	1	Cambia_Direzione
4	0	1	0	0	Errore
5	0	1	0	1	Errore
6	0	1	1	0	Errore
7	0	1	1	1	Destra
8	1	0	0	0	Errore
9	1	0	0	1	Errore
10	1	0	1	0	Errore
11	1	0	1	1	Sinistra
12	1	1	0	0	Cambia_Direzione
13	1	1	0	1	Sinistra
14	1	1	1	0	Destra
15	1	1	1	1	Avanti

LEGENDA TABELLA: 0 = baffo tocca; 1 = baffo non tocca

Le condizioni di branch (Errore, Cambia_Direzione, Sinistra ecc.) corrispondono alle possibili combinazioni dello stato (0 oppure 1) dei singoli baffi (rispettivamente, 0=tocca e 1=non tocca). La corrispondenza usata nel programma è illustrata nella tabella qui sopra. In particolare, i valori di branch sono 16 (numerati da 0 a 15) perché i bit di stato_baffi utilizzati per la fusione sensoriale sono 4 (2^4=16); tuttavia, ad alcuni valori di branch è stata assegnata la condizione di Errore: si tratta infatti di quelle configurazioni per cui il robot rileva un ostacolo (0) sia davanti che dietro di sé; non potendole gestire da solo, dunque, il robot innesca il 'campanello d'allarme' e attende la pressione del tasto di reset. Quindi, in seguito a un intervento umano che risolva la condizione di errore presentatasi (per esempio, spostando uno degli ostacoli), il robot riprende l'esecuzione del programma di navigazione. Tra le azioni previste, Cambia_Direzione merita una particolare attenzione. Alla variabile *direzione*, inizializzata a 0 all'inizio del programma, corrisponde il senso di avanzamento convenzionale del robot; *direzione*=1, invece, è quello 'invertito', per cui il robot presenta 'avanti' la scheda Back Sensors. La gestione del cambio di direzione (*direzione*=*direzione*+1) è resa possibile dalle caratteristiche della variabile stessa: dichiarata di tipo bit, essa può assumere solo due valori (1; 0), per cui esistono due sole relazioni possibili: se *direzione*=0, allora *direzione*+1=1; se *direzione*=1, allora *direzione*+1=0. In seguito al cambio di direzione, Cambia_Direzione richiama la subroutine Avanti che, in base al valore di *direzione* determina il movimento del robot in una direzione o nell'altra (Avanti_Inv). Come si diceva, infatti, data la doppia copertura dei baffi, si può intendere che il robot vada sempre e comunque avanti: presentando ora i baffi anteriori (Avanti) ora quelli posteriori (Avanti_Inv).

Riguardo alle subroutine di rotazione (Destra e Sinistra), si è deciso di usare la rotazione sul posto (data dalla rotazione di entrambe le ruote in senso inverso), invece che quella attorno a una ruota di perno: il centro di rotazione del robot, infatti, non è simmetrico rispetto al telaio (l'asse dei servomotori è spostato verso il fronte del robot), il che comporterebbe due tipi di rotazione molto diversi a seconda della direzione di avanzamento. La rotazione sul posto, invece, ottenuta comunque grazie all'istruzione pulsout (la cui sintassi è a pag. 114) minimizza questa diversità. Riguardo alla subroutine Errore, infine, noterai che contiene una pausa (pause 100) che separa l'emissione sonora (freqout) dalla rilevazione (if.. then) del tasto di reset (memorizzato in PBin). Per essere certi che il programma rilevi la pressione del tasto, possibile solo dopo la pausa tra un'emissione sonora e la successiva, sarà dunque necessario tener premuto il tasto di reset un po' più a lungo che non all'inizio del programma (ove il test su PBin era inserito nel ciclo continuo Start... goto Start). In conclusione, il programma, implementabile a piacimento, può darti un'idea delle possibilità che l'uso della scheda Back Sensors può offrirti. In questa sede, ad esempio, si è deciso di invertire semplicemente la direzione del robot in presenza di un ostacolo frontale; ciò non toglie che soluzioni diverse possano offrire una maggiore efficienza, soprattutto in situazioni che questo programma riconosce come errori. In presenza di due muri paralleli, ad esempio, questo programma non offre soluzioni e il robot continuerebbe a 'rimbalzare' da un muro all'altro; una buona soluzione, invece, sarebbe quella di introdurre una rotazione unita alla variazione di direzione. In generale, è comunque meglio creare più programmi specifici, in base alle precise condizioni in cui il robot dovrà muoversi.