Gare di velocità, ma con precisione

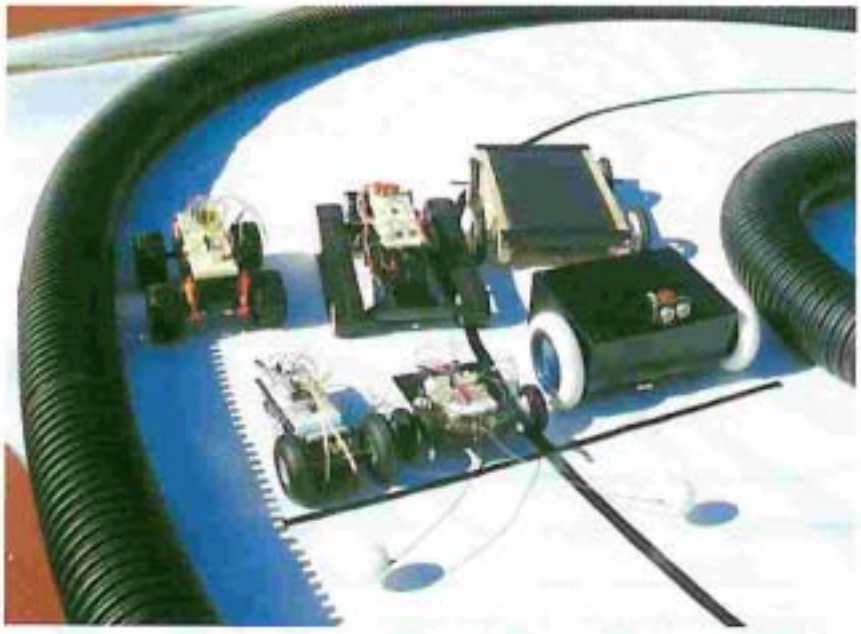
Il sensore a infrarossi allegato a questo fascicolo, in coppia con quello allegato al prossimo, verrà poi montato sul DeA Back Sensors, in modo tale da dotare il robot di una doppia copertura a infrarossi (anteriore e posteriore). Invece, per quanto riguarda il DeA Line Follower, dopo la fase di calibrazione effettuata con il programma di test di pagina 176, possiamo finalmente avviare il discorso della navigazione vera e propria. Solo grazie alla calibrazione del DeA Line Follower, infatti, è possibile ora sfruttare al meglio la sensibilità dei sensori rispetto al tipo di percorso, di superficie e di illuminazione presente.

Occorre inoltre ricordare che i LED dei sensori non sono l'unica fonte di radiazione infrarossa per il robot: sia la luce del sole sia quella artificiale, infatti, contengono una componente infrarossa che, seppure in minima parte, può alterare le prestazioni di questi sensori.

VALUTARE IL PERCORSO

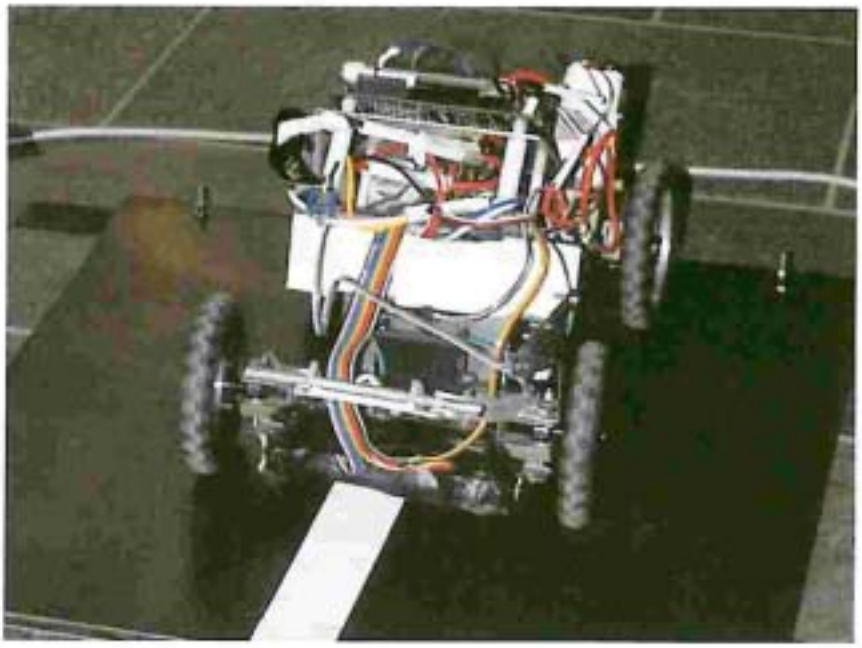
Prima di passare alla fase di programmazione del robot, vale la pena introdurre il tema delle gare tra robot mobili su percorsi tracciati. Tali competizioni, dette di *line following*, sono molto diffuse nella comunità robotica e prevedono vere e proprie manifestazioni, o *contests*, in cui robot costruiti con tecnologie anche molto diverse si sfidano, cercando di seguire la linea che individua il percorso quanto più velocemente possibile e con la massima precisione. Inoltre, spesso si utilizzano più circuiti, ciascuno dei quali progettato per testare una diversa capacità: ce ne saranno di appositamente pensati per premiare il robot più veloce; altri presenteranno invece tracciati molto complessi, per mettere in difficoltà i robot.

Sopra. I due componenti (il LED emettitore (1) e il ricevitore (2)) del primo sensore IR, che alloggeremo in seguito sulla scheda DeA Back Sensors.



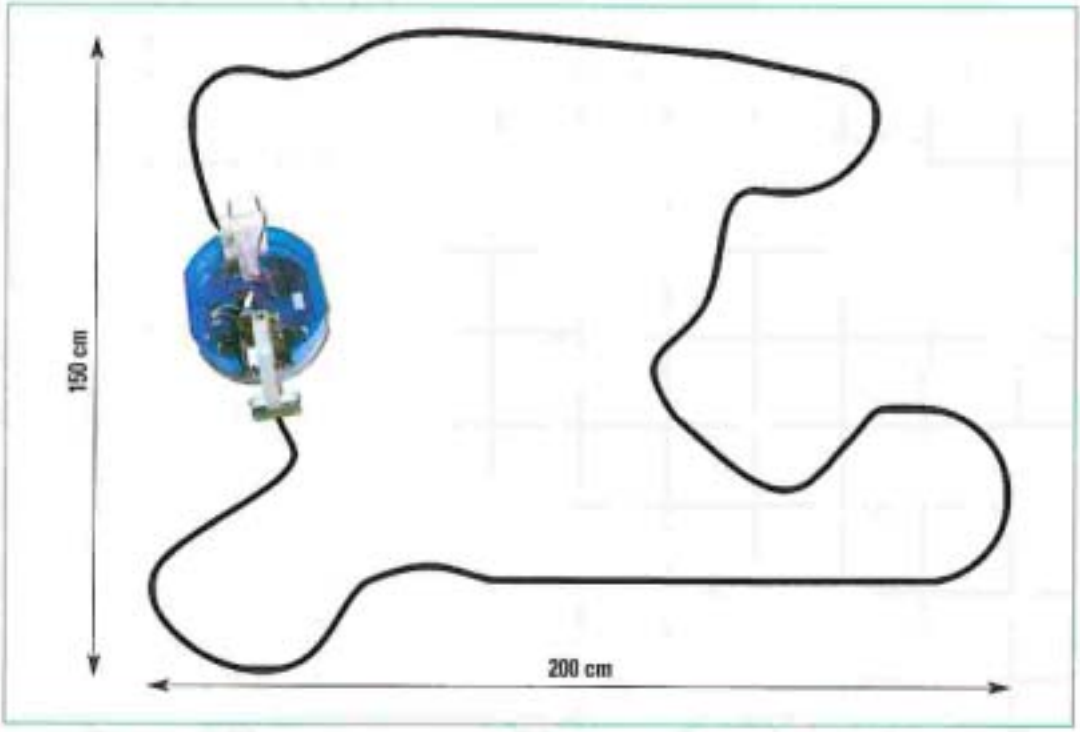
A sinistra. Alcuni concorrenti della gara di *line following* organizzata dall'università dell'Indiana (USA) su un circuito di 3x1.5 m.

Così, ad esempio, si potranno includere nel circuito ostacoli e saliscendi, ma anche diversivi che disturbino la lettura dei sensori (come flash di luce o variazioni cromatiche della superficie di sfondo), in modo da testare la tolleranza del robot agli errori. Oltre al tempo impiegato per completare il circuito, dunque, si potranno assegnare punteggi ad hoc e bonus intermedi per premiare il robot che ha seguito il tracciato con maggior precisione e superato meglio i passaggi più complessi, mentre ai robot usciti dal tracciato saranno assegnate opportune penalità.



CON IL NASTRO ISOLANTE

Vediamo allora come costruire un circuito. Il percorso di gara può essere realizzato su superfici diverse, quali un semplice foglio di cartone o il pavimento. Come linea puoi utilizzare del nastro adesivo, bianco oppure nero, tenendo presente che, per un corretto funzionamento dei sensori, è necessario che ci sia un sufficiente contrasto tra il colore della superficie e quello della linea. Inoltre, quest'ultima dovrà essere un po' più larga del singolo sensore (un nastro isolante da elettricista largo 18 mm andrà benissimo). Per ora ci serviremo di circuiti semplici (come quello qui sotto), privi di incroci, curve a gomito o spigoli.



In alto. Un robot mobile calciatore della Technical University danese: perché percepisca le linee del campo di calcio (ad esempio nelle competizioni di RoboCup), queste devono essere più larghe dei sensori e in contrasto con la superficie.

A sinistra. Un esempio di circuito chiuso (rapportato alle dimensioni del tuo robot): è relativamente semplice, giacché non presenta né incroci né curve o angoli stretti.

Le fasi di programmazione

Dopo aver tracciato il tuo percorso (liberamente, ma con caratteristiche analoghe a quelle del circuito presentato nella pagina precedente), puoi passare alla programmazione del robot e del DeA Line Follower che, per comodità, continueremo a chiamare DeA LF.

Digitato nell'area di editing il listato a destra, collega il robot al PC tramite il cavo seriale; manda quindi in esecuzione il programma premendo il pulsante Run della barra degli strumenti. Siccome in questo caso dovrai poi scollegare il robot e posizionarlo sul tracciato, il programma è provvisto di un ritardo in avvio (modificabile a piacere, come vedremo) che ti darà il tempo di compiere tali operazioni. Analizziamo ora il codice. Il programma si compone di una parte dichiarativa iniziale, di una fase di inizializzazione delle variabili e della routine principale, costituita dal ciclo infinito Main... goto Main. All'interno di questa routine principale si svolge l'algoritmo di navigazione del robot, basato sulla ripetizione continua di tre passi: anzitutto, la lettura e la memorizzazione delle informazioni provenienti dai sensori del DeA LF (attraverso la chiamata della subroutine Lettura_Sensori); quindi, l'elaborazione dell'informazione rilevata e la scelta del movimento da eseguire (attraverso la coppia di istruzioni lookdown e branch previste da DecidiAzione); infine, l'esecuzione del movimento opportuno (grazie al salto a una delle routine di movimento Avanti, GiraDestra o GiraSinistra). Entrando nel dettaglio della parte dichiarativa, riconoscerai facilmente alcune costanti e variabili, identiche a quelle del programma di test di pag. 176: in particolare, ritroviamo le costanti riguardanti lo stato dei LED dei sensori (LED_ON e LED_OFF), le tre costanti contenenti le informazioni necessarie per impostare il tipo di tracciato scelto (LINEA_BIANCA, LINEA_NERA e MODALITÀ) e quelle relative alle porte dei sensori utilizzati dal DeA LF (Sx1, LINE e Dx1). Le altre costanti presenti fanno invece riferimento alla gestione della navigazione e dei servomotori. Ritrovi infatti le consuete costanti relative alle porte dei servomotori (SERVO_DX e SERVO_SX) e alla loro velocità.

LFnavigazione1bo.bs2

```
'{$STAMP BS2}

'---------- Dichiarazione Costanti ----------------------

LED_ON          con         0
LED_OFF         con         1

LINEA_BIANCA    con         0
LINEA_NERA      con         1

MODALITA        con         LINEA_NERA

Sx1             con         6
LINE            con         4
Dx1             con         3

SERVO_DX        con         13          ' Porte relative ai servomotori
SERVO_SX        con         12

ANTIORARIO_DX   con         1000
ANTIORARIO_SX   con         1000
ORARIO_DX       con         500
ORARIO_SX       con         500

SERVO_STOP      con         750
VELOCITA100     con         100

'---------- Dichiarazione Variabili ----------------------

ledPos          var         nib
cont            var         nib
azione          var         nib

lfBits          var         nib

' ---------- Inizializzazione ----------------------------

pause 5000

outl = %01011000
dirl = %01011000
azione = 1

' ---------- Programma Principale ------------------------

Main:

      gosub Lettura_Sensori

DecidiAzione:
      lookdown lfBits, [%0001, %0010, %0100], azione
      branch azione, [GiraDestra, Avanti, GiraSinistra]


GiraDestra:
      pulsout SERVO_DX, SERVO_STOP
      pulsout SERVO_SX, SERVO_STOP + VELOCITA100
goto Main


Avanti:
      pulsout SERVO_DX, SERVO_STOP - VELOCITA100
      pulsout SERVO_SX, SERVO_STOP + VELOCITA100
goto Main

GiraSinistra:
      pulsout SERVO_DX, SERVO_STOP - VELOCITA100
      pulsout SERVO_SX, SERVO_STOP
goto Main

end

' ---------- Subroutine ----------------------------------

Lettura_Sensori:
      lfBits = 0
      for cont = 0 TO 2
            lookup cont, [Dx1, LINE, Sx1], ledPos
            outl.lowbit(ledPos) = LED_ON
            pause 1
            lfBits.lowbit(cont) = in9 ^ MODALITA
            outl.lowbit(ledPos) = LED_OFF
      next
return
```

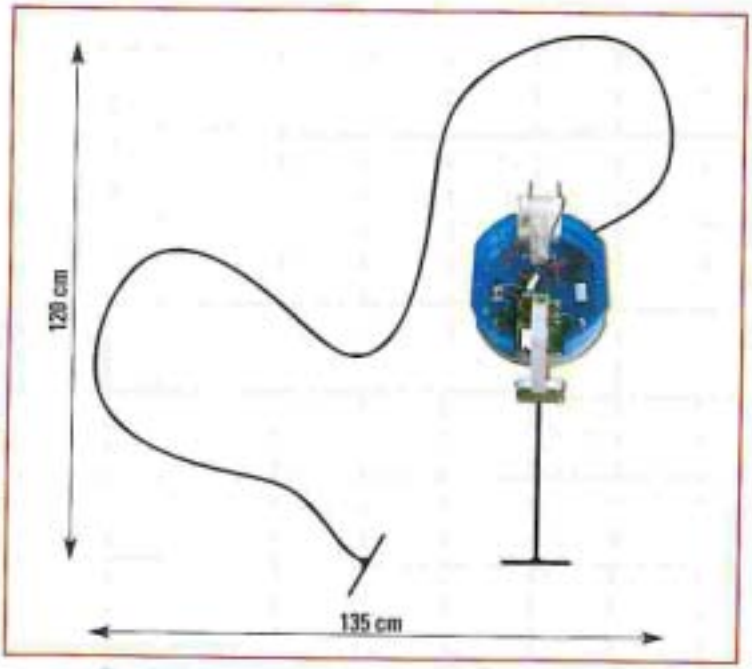
LE FASI DI PROGRAMMAZIONE

A questo proposito, i valori del listato sono quelli di default, da sostituire con i valori da te calibrati grazie alle informazioni delle pagg. 116 e 123-124. Seguono quindi altre due costanti (SERVO_STOP e VELOCITÀ100), sempre riferite alla velocità dei motori. Alla prima è attribuito il valore 750 (è l'intermedio tra 500 e 1000 che, come ricorderai, permette di mantenere fermi i servomotori): anch'esso di default, va sostituito con il valore che hai ricavato dalla calibrazione dei tuoi servomotori. La costante VELOCITÀ100, invece, si riferisce a un valore di velocità scelto empiricamente; come vedremo in seguito, potrai modificarlo a piacere, in base alle caratteristiche del tuo tracciato. Anche nella dichiarazione delle variabili ritroviamo tre variabili già incontrate per il programma di test (*ledPos*, *IfBits* e *cont*), cui si aggiunge una nuova variabile di tipo nib (*azione*) che verrà utilizzata per determinare il movimento da eseguire. Nella fase di inizializzazione viene per prima cosa imposta una pausa di 5 secondi (5000 ms) che, come già anticipato, ti permette di scollegare il cavo seriale dal robot e posizionare quest'ultimo sul tracciato: ovviamente puoi aumentare o diminuire il valore della pausa a seconda delle tue esigenze. Segue poi l'inizializzazione delle porte dei sensori attraverso le variabili speciali *dirl* e *outl*, esattamente come visto per il programma di test. Infine, la variabile *azione* viene inizializzata a 1: come vedremo fra poco, tale valore corrisponde al movimento in avanti del robot. All'interno del ciclo principale troviamo dapprima la chiamata alla subroutine Lettura_Sensori, che si occupa di leggere e memorizzare i valori logici forniti dai sensori: analizzandola, noterai che risulta pressochè identica a quella già vista per il programma di test, ad eccezione dell'istruzione di debug all'interno del ciclo for (allora necessaria per visualizzare nell'apposita finestra i valori rilevati, ora inutile). Come illustrato a pag. 178 (in alto a destra), i valori letti dai tre sensori vengono memorizzati nei tre bit meno significativi della variabile *IfBits*, mantenendo lo stesso ordine di rilevazione. Una volta effettuata la lettura dei sensori, dunque, la variabile *IfBits* conterrà tutte le informazioni necessarie perché il robot riconosca la propria posizione rispetto alla linea del tracciato e si muova di conseguenza. Lo scopo dell'algoritmo di navigazione, infatti, è quello di mantenere il robot il più possibile centrato sulla linea del tracciato, il che si ottiene facendo in modo che la linea sia sempre rilevata dal sensore centrale (Line). In caso contrario, cioè se la linea è rilevata da uno dei due sensori laterali (Sx1 o Dx1), l'algoritmo dovrà invece intervenire con opportune correzioni della traiettoria del robot. Vediamo allora come utilizzare a questo scopo le informazioni memorizzate, bit per bit, in *IfBits*. Osserva la tabella riportata a destra: sono presenti i tre casi citati (in cui, comunque, si è supposto che la linea del tracciato venisse rilevata da un solo sensore); a ciascun caso, inoltre, è associato un movimento del robot. Nel primo caso la linea viene rilevata dal sensore destro (Dx1) per cui la variabile *IfBits* ha valore %0001, il che significa che il robot potrà essere sia in curva sia su un rettilineo, ma comunque ancora sulla strada 'giusta' (non si è cioè girato a tal punto da allineare i sensori al tracciato): per non perdere il tracciato, il robot dovrà tendere a 'centrarsi' e il movimento correttivo necessario per farlo è una rotazione verso destra. Analogamente nel terzo caso, relativo al sensore sinistro (Sx1, con *IfBits* = %0100), il movimento correttivo necessario è la rotazione opposta, verso sinistra. Nel secondo caso, invece, la linea è rilevata dal sensore centrale (Line, con *IfBits* = %0010) e il robot può avanzare senza correzione alcuna. A livello di programmazione, tutto questo è stato 'tradotto' nella coppia di istruzioni lookdown e branch già utilizzata, ad esempio, per rimappare i codici del telecomando (a pag. 147): ora, con l'ausilio della variabile *azione* si associa a ciascuno dei casi presenti nella tabella il corrispondente movimento del robot.

I tre bit meno significativi di IfBits			Movimento abbinato
Bit2 (Sx1)	Bit1 (Line)	Bit0 (Dx1)	
0	0	1	Gira a Destra
0	1	0	Avanza
1	0	0	Gira a Sinistra

Ovviamente questi tre casi non esauriscono tutti quelli possibili: non sono stati considerati infatti i casi 'misti', che prevedono la rilevazione della linea da parte di più sensori in contemporanea. Per il momento abbiamo optato per tale esclusione (giustificata dalle caratteristiche di semplicità del circuito proposto) per non appesantire il codice di questo programma: circuiti senza incroci o spigoli, infatti, consentono di semplificare la gestione delle informazioni fornite dai sensori. Cosa succede, però, se il robot rileva un caso 'misto', non previsto dal programma? Di fatto queste rilevazioni vengono ignorate e il robot continua a eseguire il movimento imposto dall'ultima rilevazione valida. Supponiamo, ad esempio, che il robot si trovi all'interno di una curva, in posizione tale per cui *IfBits* memorizzi il valore %0110 (la linea è cioè rilevata sia dal sensore centrale sia dal laterale sinistro): la ricerca di lookdown all'interno della lista di tre valori, ovviamente, fallisce; la variabile *azione* (come detto a pag. 148) rimane invariata e il robot esegue nuovamente il movimento scelto in precedenza. In seguito, invece, vedremo alcuni esempi di come sia possibile (e utile) usare anche le informazioni relative ai casi 'misti'; per ora, invece, analizziamo le routine di movimento, prendendo ad esempio Avanti, che invia ai servomotori gli impulsi adatti per far avanzare il robot.

Anzitutto è necessario fare alcune osservazioni.
In primo luogo, noterai che le due istruzioni pulsout non sono comprese all'interno di un ciclo for, come succedeva invece in altri programmi visti finora (ad esempio, a pag. 124): questo significa che a ogni rilevazione dei sensori seguirà un singolo micromovimento dei motori e, quindi, l'immediato ritorno alla lettura dei sensori. Questo consente di verificare in tempo reale l'andamento del robot sul tracciato; con un ciclo for, invece, si sarebbe avuto un intervallo di tempo (per quanto breve) in cui il robot si sarebbe mosso senza controllare i riferimenti sul tracciato, rischiando così di uscirne irrimediabilmente. In secondo luogo, noterai anche l'assenza di pause al termine delle due istruzioni pulsout: in questo programma, infatti, non serve impostare una pausa, sia perché le due istruzioni non vengono eseguite all'interno di un ciclo for sia perché la subroutine Lettura_Sensori introduce tra l'esecuzione di due routine di navigazione un ritardo più che sufficiente per fungere esso stesso da pausa. Infine, puoi notare che si è scelto di non imporre ai servomotori la loro velocità massima (ottenibile con il semplice invio di un impulso di durata massima, cioè ORARIO_DX e ANTIORARIO_SX), bensì di imporre loro una velocità minore (ottenuta tramite impulsi la cui durata sia, a seconda dei casi, inferiore o superiore di 100 ms rispetto ai valori nominali). Data la natura del circuito, infatti, si impone un compromesso: va cioè imposta una velocità opportuna sia per sfruttare al meglio i rettilinei sia per mantenere la traiettoria in curva. Il parametro proposto (100) è ovviamente modificabile a tuo piacimento (nella dichiarazione), a seconda delle prestazioni effettive del tuo robot e delle caratteristiche del percorso da te tracciato: potrai cioè scegliere di imporre una velocità maggiore (diminuendo il parametro), per una migliore performance su rettilineo, a costo di minor precisione in curva; viceversa, potrai diminuire ulteriormente la velocità (aumentando il parametro) per dare al robot il tempo di effettuare un maggior numero di misure ed essere più preciso nelle eventuali correzioni della traiettoria. Per le altre due routine di navigazione (GiraDestra e GiraSinistra) valgono ovviamente le stesse considerazioni; in aggiunta, però, possiamo osservare (ad esempio in GiraDestra) che la rotazione (qui verso destra) viene effettuata mantenendo ferma (SERVO_STOP) una ruota (qui la destra) e facendo ruotare (SERVO_STOP+VELOCITÀ100) solo l'altra (qui la sinistra, in senso antiorario). Anche in questo caso, il valore di impulso utilizzato è inferiore (750+100=850) a quello massimo (1000): la rotazione è più lenta, ma più precisa. Si tratta cioè di un altro grado di libertà offerto dal programma, modificabile a tuo piacere. Testa quindi i diversi valori di velocità sul circuito, sui rettilinei e in curva, e massimizza le prestazioni del tuo robot.



Una diversa tipologia di circuito per robot mobili, utilizzata in molte gare di line following, è quella dei cosiddetti circuiti aperti, caratterizzati da due precise posizioni: di partenza e di arrivo (contrassegnate da una linea ortogonale al tragitto, simile a una T). Un esempio è illustrato qui sopra.

Lo scopo di una gara su circuito aperto è far sì che il robot, oltre a percorrere correttamente il tracciato, si arresti nella posizione di arrivo (la T da cui non è partito). Sempre grazie al nastro isolante, puoi riprodurre un circuito di questo tipo: ovviamente puoi inventare la geometria del percorso, a patto però di posizionare alle sue estremità due pezzi di nastro isolante lunghi almeno quanto la serie dei tre sensori del DeA LF (meglio se lunghi quanto l'intera scheda). Proviamo ora a costruire un programma adatto alla navigazione su questo tipo di circuito. La sua struttura sarà molto simile a quella del programma precedente, ma dovremo fare i conti con la gestione sia della partenza sia dell'arrivo: entrambi, infatti, rientrano in uno dei casi 'misti' di cui si diceva, ossia quello in cui la variabile *IfBits* memorizza il valore %0111. Per gestire solo la fase di arrivo, basterebbe associare il caso %0111 all'arresto immediato del robot; nel nostro caso, però, dovremo fare in modo che il robot non confonda ciò che accade all'arrivo con quanto accade alla partenza: seppur per breve tempo, infatti, anche alla partenza i sensori leggeranno il medesimo valore. Se dunque ci limitassimo ad associare il valore %0111 di *IfBits* soltanto all'arresto del robot, avremmo dei problemi a farlo partire.

LE FASI DI PROGRAMMAZIONE

Per risolvere l'ambiguità, dunque, sarà necessario costruire un algoritmo di decisione leggermente più complesso, che preveda l'utilizzo di una variabile di stato in cui memorizzare l'informazione riguardante l'avvenuta partenza o meno del robot. Ad esempio, potremo dichiarare una variabile di tipo bit da utilizzare come variabile booleana (abbinando cioè il valore 0 a 'falso' e il valore 1 a 'vero'): nel momento in cui il robot superasse il nastro di partenza, la variabile verrebbe settata a 1 ('vero'): il riconoscimento della condizione di arresto del robot, a questo punto, richiederebbe non una ma due informazioni, la lettura del valore %0111 in *IfBits* e il valore 1 associato alla variabile di stato. Puoi trovare un esempio di questo algoritmo nel programma proposto a destra. Il listato ripropone la struttura del programma precedente, con la chiamata della subroutine di lettura dei sensori cui però segue una più articolata sequenza di istruzioni, necessaria al riconoscimento delle condizioni relative alla partenza e all'arrivo del robot. Scendendo nel dettaglio, puoi notare la presenza di due nuove costanti (VERO e FALSO) che fanno riferimento ai valori 1 e 0. Inoltre, dichiarata la variabile di tipo bit *inCorsa* che conterrà l'informazione riguardante la partenza, essa viene inizializzata a FALSO. Analizziamo ora la routine principale. Dopo la lettura sensoriale effettuata con Lettura_Sensori, compare una prima istruzione if... then: se la linea del tracciato non si trova sotto il sensore centrale (IfBits <> %0010), si salta alla routine VerificaArrivo; altrimenti la variabile *inCorsa* viene settata a VERO. In questo secondo caso, infatti, appena il robot supera il nastro di partenza, i sensori rileveranno la linea al di sotto del sensore centrale e si potrà dare per superata la partenza (questo, però, solo a patto di aver allineato correttamente il robot sul nastro di partenza, altrimenti *inCorsa* verrà settata a VERO solo la prima volta che il robot rileverà la linea con il sensore centrale). Torniamo ora al primo caso, cioè a VerificaArrivo che si compone di altre due istruzioni di test. Se non viene rilevato alcun nastro disposto di traverso (IfBits <> %0111), siamo ancora lungo il tracciato e, quindi, si salta alla routine DecidiAzione che, con la coppia di istruzioni lookdown e branch, seleziona il movimento da compiere. In caso contrario, occorre controllare il valore di *inCorsa*: se è FALSO, significa che il nastro rilevato è quello di partenza, quindi si dovrà iniziare il percorso saltando alla routine Avanti; perché *inCorsa* sia VERO, invece, occorre che il nastro di partenza sia già stato superato, nel qual caso il nastro incontrato ora è quello di arrivo; il programma salta alla routine Arresto e il robot si ferma. Fine della gara.

```
'{$STAMP BS2}

'---------- Dichiarazione Costanti ----------

LED_ON          con         0
LED_OFF         con         1

LINEA_BIANCA    con         0
LINEA_NERA      con         1

MODALITA        con         LINEA_NERA

VERO            con         1
FALSO           con         0

Sx1             con         6
LINE            con         4
Dx1             con         3

SERVO_DX        con         13          ' Porte relative ai servomotori
SERVO_SX        con         12

ANTIORARIO_DX   con         1000
ANTIORARIO_SX   con         1000
ORARIO_DX       con         500
ORARIO_SX       con         500

SERVO_STOP      con         750
VELOCITA100     con         100

'---------- Dichiarazione Variabili ----------

inCorsa         var         bit

ledPos          var         nib
cont            var         nib
azione          var         nib

IfBits          var         nib

' ---------- Inizializzazione ----------

pause 5000

out1 = %01011000
dir1 = %01011000
azione = 1
inCorsa = FALSO

' ---------- Programma Principale ----------

Main:

      gosub Lettura_Sensori

      if (IfBits <> %0010) then VerificaArrivo
      inCorsa = VERO

VerificaArrivo:
      if (IfBits <> %0111) then DecidiAzione
      if (inCorsa = FALSO) then Avanti
      goto Arresto

DecidiAzione:
      lookdown IfBits, [%0001, %0010, %0100], azione
      branch azione, [GiraDestra, Avanti, GiraSinistra]


GiraDestra:
      pulsout SERVO_DX, SERVO_STOP
      pulsout SERVO_SX, SERVO_STOP + VELOCITA100
goto Main


Avanti:
      pulsout SERVO_DX, ORARIO_DX + VELOCITA100
      pulsout SERVO_SX, ANTIORARIO_SX - VELOCITA100
goto Main

GiraSinistra:
      pulsout SERVO_DX, SERVO_STOP - VELOCITA100
      pulsout SERVO_SX, SERVO_STOP
goto Main

Arresto:
end

' ---------- Subroutine ----------

Lettura_Sensori:
      IfBits = 0
      for cont = 0 TO 2
            lookup cont, [Dx1, LINE, Sx1], ledPos
            out1 lowbit(ledPos) = LED_ON
            pause 1
            IfBits.lowbit(cont) = in9 ^ MODALITA
            out1 lowbit(ledPos) = LED_OFF
      next
return
```