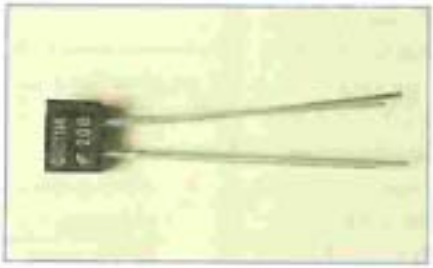
Sensori monoblocco a raggi infrarossi

Il sensore allegato a questo fascicolo è il primo di una serie di tre che, come vedremo, dovrà essere alloggiata nelle sedi centrali della scheda DeA Line Follower. A differenza dei sensori IR costituiti da un emettitore e un ricevitore (quelli montati sulla scheda madre), questi tre sono stati realizzati in un blocco unico, appositamente per il tuo robot. Un sensore monoblocco offre molteplici vantaggi: innanzitutto comporta una percezione migliore della soglia di passaggio dalla luce al buio; inoltre, il sensore potrà essere regolato su una banda di frequenze molto ristretta, risultando perciò meno soggetto ai disturbi luminosi ambientali. Maggiore è il numero di sensori a disposizione, ovviamente, migliore sarà la performance del robot: in presenza di una curva, ad esempio, un sensore solo potrebbe infatti "perdere di vista" la linea da seguire; grazie ai sensori laterali, invece, il robot potrà correggere la sua traiettoria e proseguire nell'inseguimento oltre la curva.



Nelle foto. Il sensore (in alto a sinistra) sarà alloggiato in una delle sedi centrali del DeA Line Follower (sopra, riquadrate).

Le fasi di programmazione

Come preannunciato, ci occuperemo ora di un programma che ti permetta di controllare i movimenti del manipolatore del robot attraverso il telecomando. A partire dal funzionamento del telecomando (affrontato a più riprese), si tratta ora di adattare quanto visto per i servomotori al caso dei motori a spazzola. Sarà dunque utile ricordare la differenza fondamentale nella gestione di questi due tipi di motori: i servomotori sono comandati attraverso una serie di impulsi impartiti con il comando PBASIC pulsout; i motori a spazzola, invece, hanno un comportamento di tipo acceso/spento, ottenuto tramite la variazione del valore delle porte logiche a cui sono connessi. Come vedremo, questo comporterà qualche modifica nella gestione del telecomando.

Con il programma illustrato a pag. 158 potrai comandare il manipolatore utilizzando i tasti del telecomando contrassegnati con P (a destra). In particolare, il tasto ◀▶ comanderà l'apertura dei becchi e il tasto ▶◀ la chiusura; i tasti ▲ e ▼, invece, saranno utilizzati per alzare e abbassare il carrello.

Sebbene sia possibile integrare anche la gestione dei servomotori, per semplicità verrà ora gestito solo questo insieme di tasti.

LE FASI DI PROGRAMMAZIONE

Analizziamo il codice. Puoi osservare che la struttura del programma è pressoché analoga a quella presentata in precedenza, a proposito dei servomotori (ad esempio, a pag. 148). Nel ciclo principale **main... goto main**, infatti, decodificato il segnale proveniente dal telecomando (tramite il salto alla subroutine **decodifica**) e in base alla posizione del valore corrispondente (individuata con **lookdown**), si salta (grazie al **branch**) alla routine di movimento (**Alza**, **Abbassa**, **Chiudi** o **Apri**) associata alla funzionalità richiesta. L'utilizzo di lookdown, in questo caso, è dovuto al fatto che i codici numerici dei tasti utilizzati (riassunti nelle tabelle a pag. 140 e 146) sono i numeri da **6** a **9**: per gestirli con un branch, dunque, è necessario rimapparli come abbiamo visto alle pagg. 147-148. Ed è proprio per questo che, nella dichiarazione iniziale, è presente la variabile ***posizione*** di tipo **nib** (è stata inoltre dichiarata una nuova variabile, ***impulso***, di tipo **word**, il cui uso sarà chiarito in seguito). Sono state inoltre dichiarate le **costanti** relative alle porte dei motori a spazzola: **M1a** (P14) e **M1b** (P15) relative al motore collegato a X2, **M2a** (P3) e **M2b** (P5) relative a quello collegato a X3. Prima del main, tali porte vengono **inizializzate** a zero (**low**), in modo da mantenere fermi i motori. Nel main, poi, viene richiamata la **subroutine di decodifica** del segnale trasmesso dal telecomando. In questo caso sono state utilizzate **cinque variabili** per la decodifica del codice numerico (***inputIR0... inputIR4***): in base alla tabella di pag. 146, sappiamo che il valore massimo da gestire è **9**, ossia il **codice DEC** abbinato al tasto ▼, che richiede un minimo di **4 cifre** per essere identificato correttamente; l'utilizzo di **una cifra in più** (cioè 5 in totale), però, ci permette di mantenere una maggiore tolleranza agli errori (ad esempio, la pressione di eventuali tasti con la stessa porzione di codice binario). Fatte salve queste scelte progettuali, comunque, la subroutine di decodifica rimane sostanzialmente inalterata. Decodificato il segnale, quindi, si procede con il **salto alla routine di movimento** associata al tasto premuto. In particolare, il valore di ***codicePulsante***, rimappato attraverso l'istruzione lookdown, assumerà un valore compreso tra **0** e **3** (memorizzato in ***posizione***), il che giustifica la preventiva inizializzazione di ***posizione*** a **4**, utile per neutralizzare eventuali errori. Per quanto riguarda le routine **Apri**, **Chiudi**, **Alza** e **Abbassa**, esse sono pressoché identiche, salvo la configurazione delle porte dei motori. Analizziamone dunque una sola, **Alza**, tenendo presente che, con le dovute varianti, il discorso vale anche per le altre tre. Per ottenere l'elevazione del carrello, la **routine Alza** dovrà ovviamente impostare la configurazione delle porte **P3** e **P5** del motore a spazzola **X3** secondo i dati della tabella a pag. 156. **A questo punto, occorre scegliere il modo con cui si intende usare i motori a spazzola**: una possibilità, infatti, è quella di gestirli con impulsi successivi, per cui la routine, avviato il motore X3 con la corretta configurazione delle porte, attende qualche istante (in cui si compie il breve movimento del carrello), spegne il motore e ritorna al main.

pinza.bs2

```
'{$STAMP BS2}
'---------------- Dichiarazione Variabili ----------------

posizione          var        nib

codicePulsante     var        byte

segnaleInizio      var        word
inputIR0           var        word
inputIR1           var        word
inputIR2           var        word
inputIR3           var        word
inputIR4           var        word
impulso            var        word

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

M1a                con        14      ' porte logiche dei motori
M1b                con        15
M2a                con        3
M2b                con        5

IR_SX              con        8
BASSO              con        0

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

' inizialmente i motori sono fermi
low   M1a:  low   M1b
low   M2a:  low   M2b

main:

      gosub decodifica

      posizione = 4
      lookdown codicePulsante, [6, 7, 8, 9], posizione
      branch posizione, [Apri, Alza, Chiudi, Abbassa]

goto main
'---------------- Routine di movimento della pinza ----------------

Alza:
      low M2a:    high  M2b
      pause 100
      pulsin IR_SX, BASSO, impulso
      if impulso > 200 then alza
goto arresto


Abbassa:
      high  M2a:  low  M2b:
      pause 100
      pulsin IR_SX, BASSO, impulso
      if impulso > 200 then abbassa
goto arresto

Chiudi:
      low   M1a:  high  M1b:
      pause 100
      pulsin IR_SX, BASSO, impulso
      if impulso > 200 then chiudi
goto arresto


Apri:
      high  M1a:  low  M1b:
      pause 100
      pulsin IR_SX, BASSO, impulso
      if impulso > 200 then apri
goto arresto


Arresto:
      low   M1a:  low   M1b
      low   M2a:  low   M2b
      pause 10
goto main


'----------- Subroutine decodifica impulsi IR -----------

decodifica:
   rileva:
      pulsin IR_SX, BASSO, segnaleInizio
      pulsin IR_SX, BASSO, inputIR0
      pulsin IR_SX, BASSO, inputIR1
      pulsin IR_SX, BASSO, inputIR2
      pulsin IR_SX, BASSO, inputIR3
      pulsin IR_SX, BASSO, inputIR4

   if segnaleInizio < 1000 then rileva

      codicePulsante.bit0 = inputIR0.bit9
      codicePulsante.bit1 = inputIR1.bit9
      codicePulsante.bit2 = inputIR2.bit9
      codicePulsante.bit3 = inputIR3.bit9
      codicePulsante.bit4 = inputIR4.bit9

return
```

Tale soluzione, però, può andare bene solo nel caso di una singola pressione del tasto (e, dunque, per brevi movimenti singoli). Non si rivela essere la soluzione ottimale, invece, nel caso più comune, in cui si voglia ottenere un movimento prolungato, grazie alla pressione continua di uno stesso tasto: alla lunga, infatti, può danneggiare il motore. Occorrerà, allora, intraprendere una strada diversa, che è quella presentata nel listato. In pratica, invece di spegnere il motore al termine della pausa necessaria al compimento del singolo movimento, si tratterà di verificare se il tasto del telecomando è ancora premuto o no: in caso affermativo, si manterrà acceso il motore, altrimenti verrà spento. Come vedremo, questo è possibile grazie all'introduzione di un'opportuna variabile (*impulso*). Torniamo dunque alla routine Alza, per vedere come ciò si traduca in termini di programmazione. Innanzitutto, le porte relative al motore X3 (cioè P3 e P5) devono essere configurate per produrre il movimento del carrello (P3=0, P5=1; ovvero low M2a, high M2b); quindi, introdotta una pausa di 100 ms (pause 100), si procede alla verifica dei segnali inviati. Tale verifica è svolta dalla coppia di istruzioni pulsin e if... then che attuano la soluzione di cui si diceva: in pratica, l'istruzione pulsin memorizza nella variabile *impulso* la durata di un eventuale passaggio del valore registrato dal ricevitore (IR_SX) da 1 a 0 (BASSO). Quindi if... then confronterà (>) tale durata con una soglia (200), per decidere se procedere o meno nel movimento. In particolare, se il valore di *impulso* verifica la condizione (ossia è >200), il programma torna a ripetere la stessa routine (Alza); altrimenti (se <200) salta a quella di Arresto. Come puoi notare, la verifica non comporta la decodifica completa del segnale, bensì soltanto un confronto generico. Il programma, di fatto, identifica la ricezione di un qualsiasi segnale con quello inviato dal tasto precedentemente premuto e decodificato, e lo può fare perché lavora molto più velocemente di te: nella pausa prevista (100 ms), infatti, possono accadere solo due cose: o continui a premere lo stesso tasto (e allora c'è un segnale e il movimento va ripetuto), oppure stai per cambiare tasto (senza però aver avuto il tempo di portare a termine il gesto). In tal caso, il segnale viene a mancare per un tempo sufficientemente lungo perché il programma lo interpreti come sua assenza (mancando il segnale, la condizione posta da if... then non è verificata e la routine rimanda all'arresto), il tutto prima che tu possa avere effettivamente premuto un altro tasto. Quando poi ciò avviene, il programma è pronto a ricominciare da capo, decodifica inclusa (Arresto, infatti, termina con goto main).

L'approccio scelto finora per programmare il movimento delle componenti mobili del manipolatore non risolve però un problema di origine meccanica: il cosiddetto fine corsa, ossia l'arresto dei becchi e del carrello, giunti agli estremi delle cremagliere.

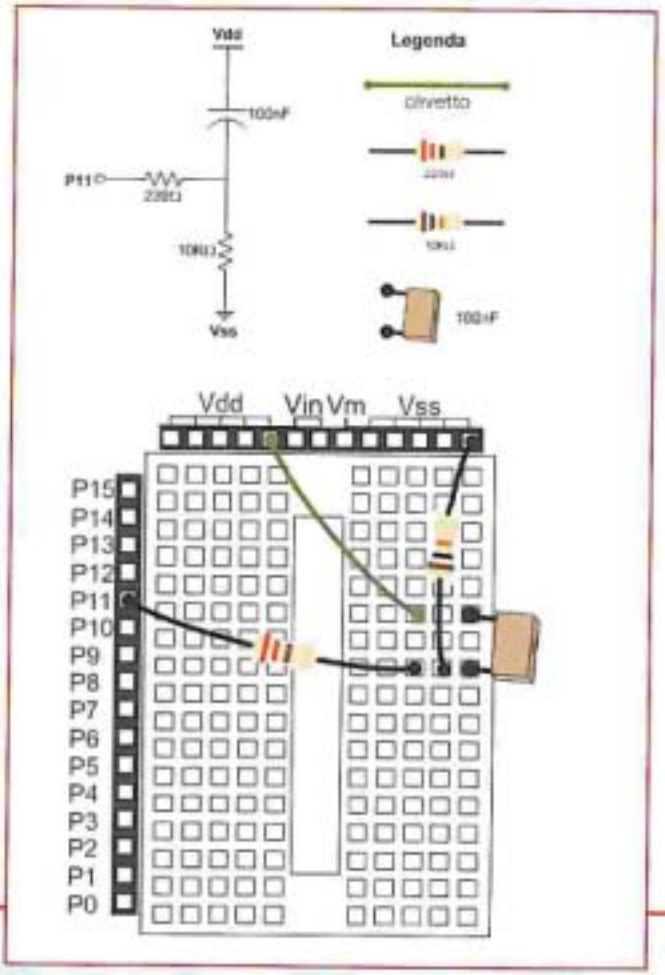
Il programma infatti si limita a imporre la durata del movimento, senza però verificare se tale movimento sia fisicamente possibile. Da un punto di vista hardware, invece, il movimento sia dei becchi sia del carrello è limitato: in particolare, il movimento di quest'ultimo è limitato sia in salita sia in discesa dalla lunghezza della cremagliera; analogamente, anche l'apertura dei becchi dipende dalla rispettiva cremagliera; la loro chiusura, invece, dipende dal punto di incontro dei becchi o di presa su un oggetto. Raggiunte queste posizioni limite (cioè il fine corsa), il persistere nel movimento (mantenendo premuto il tasto del telecomando) può comportare un danno ai motori che, trovando una resistenza molto forte, richiedono più energia e sforzano eccessivamente. Detto questo, il semplice prestare attenzione al problema, per quanto utile, può non essere sufficiente. Vediamo dunque come risolverlo a monte, con la programmazione, cercando un modo perché i motori, sottoposti a uno sforzo eccessivo, si arrestino automaticamente. Vogliamo cioè far in modo che sia il robot a registrare il fine corsa e a inibire un movimento che danneggerebbe i suoi motori.

Così, ad esempio, afferrato un oggetto con i becchi della pinza e consolidata la presa, il robot dovrà non solo spegnere il motore, ma anche impedire che risponda a successivi eventuali comandi di chiusura dei becchi, prima che siano stati di nuovo aperti. Tuttavia, prima di programmare i fine corsa che salvaguardino le parti meccaniche del robot è necessario indagare la natura fisica del problema. In condizioni ideali, cioè non sottoposto ad alcuno sforzo, il motore a spazzola è collegato alla tensione di alimentazione del robot ed è la corrente assorbita dalle batterie che produce la rotazione. Il prodotto tra la tensione (V) e la corrente assorbita (I) è la potenza (P) erogata dal motore (per cui P=V*I). In prima approssimazione, trascurando le perdite dovute a fattori come l'attrito, la potenza (P) erogata dal motore coincide con quella fornita dalle batterie. Quando il motore è sottoposto a uno sforzo (ad esempio per la messa in moto degli ingranaggi della pinza), invece, deve erogare una potenza maggiore. Dal momento che la tensione (V) delle batterie è una costante (nel nostro caso 1.5 V per ogni batteria, per un totale di 1.5*4=6 V), per produrre un valore di potenza superiore il motore non può fare altro che assorbire una maggiore quantità di corrente (I). Sebbene, dunque, siano sempre e comunque le batterie che forniscono tale potenza, esiste però un limite (Pmax), legato alle specifiche di costruzione delle batterie stesse. Superato tale valore massimo di potenza erogabile dalle batterie, si assiste al decadimento delle loro prestazioni.

LE FASI DI PROGRAMMAZIONE

Anche nel nostro caso, allora, per poter vincere le forze di inerzia delle posizioni critiche della pinza, il motore richiede alle batterie una potenza sempre maggiore, ossia un maggior assorbimento di corrente. Quando la potenza richiesta supera quella massima, le batterie continuano comunque a fornire tale potenza (Pmax), ma il motore assorbe sempre più corrente; di conseguenza, per compensare l'aumento di corrente (I), è necessario che la tensione di alimentazione (V) diminuisca. Dal momento che il prodotto P=V*I deve rimanere costante (perché P=Pmax), infatti, all'aumentare di (I) non può che diminuire (V). Nel momento di massimo sforzo del manipolatore (nelle posizioni critiche), dunque, si assiste a un abbassamento della tensione di alimentazione. Per poter sfruttare a nostro vantaggio questo principio fisico, basterà allora trovare un modo per monitorare il valore della tensione di alimentazione e usarlo come 'campanello d'allarme'. Ancora una volta ci viene in aiuto l'istruzione PBASIC rctime, incontrata a pag. 108. In quell'occasione, l'istruzione era stata utilizzata per stimare la costante di tempo τ di un circuito RC contenente un fotoresistore (ossia un componente a resistenza variabile). Vediamo ora cosa accade sostituendo il fotoresistore con un resistore (cioè un componente con valore di resistenza fisso e noto a priori), come mostrato nei circuiti a destra. Applicata a questo circuito, l'istruzione rctime restituirà sempre lo stesso valore: entrambi i parametri (R e C) che costituiscono la costante di tempo τ, infatti, sono fissi e noti. Tuttavia, come ricorderai, rctime misura il tempo di transizione dallo stato alto di tensione (Vdd) alla soglia della porta logica cui è collegato il circuito RC: dunque, se in questa configurazione assumiamo il valore di tensione Vdd come variabile, ecco che rctime tornerà a misurare valori diversi, più grandi se Vdd aumenta, più piccoli se Vdd diminuisce. Il valore misurato da rctime, relativo alle variazioni di Vdd, è esattamente l'indicatore che stavamo cercando: registrando le variazioni (negative) della tensione di alimentazione, infatti, quantifica lo sforzo a cui viene sottoposto il motore. Vediamo ora come tutto ciò possa tradursi a livello di hardware. Innanzitutto, è necessario implementare il circuito sulla breadboard: collega un capo del resistore da 220 Ω alla porta P11 e l'altro a un capo del condensatore da 100 nF; questo stesso capo del condensatore deve inoltre essere collegato, tramite un resistore da 10 kΩ, alla tensione Vss (cioè a massa); l'altro capo del condensatore, libero, va infine collegato tramite un cavetto alla tensione Vdd. Come puoi notare, la configurazione del circuito è analoga a quella realizzata per i fotoresistori. Grazie a questo circuito possiamo procedere all'implementazione software dei fine corsa. Digita nell'area di editing l'intero listato proposto in due porzioni separate alle pagine 161 e 162 (la fine dell'una non è che l'inizio dell'altra), collega il robot al PC con il cavo seriale e premi il pulsante RUN sulla barra degli strumenti. Questa volta, però, mantieni collegato il cavo seriale.

Come nel programma visto a pag. 158, premendo i tasti ◀▶ e ▶◀ si potranno aprire e chiudere i becchi della pinza; mentre con i tasti ▲ e ▼ si alzerà e abbasserà il carrello. Portando i becchi o il carrello nelle rispettive posizioni critiche, i motori si fermeranno e nella finestra di debug verrà visualizzato il corrispondente messaggio: così, alzando al massimo il carrello, sarà visualizzato "Carrello alzato". Inoltre, in tali condizioni, i motori potranno eseguire solo il movimento che li allontana dalla posizione critica: se, ad esempio, si continuasse a premere il tasto ▲ con il carrello completamente alzato, tale comando sarebbe ignorato e il motore rimarrebbe fermo; solo con la pressione del tasto ▼, infatti, la routine di salita sarà riabilitata. Analizziamo il codice, costruito a partire dal programma visto in precedenza, con l'aggiunta però delle parti relative alla gestione dei fine corsa (nella porzione a pag. 162). Riconoscerai, infatti, il ciclo principale e le varie routine di movimento (pur opportunamente modificate) e la subroutine di decodifica del segnale del telecomando. Consideriamo anzitutto la dichiarazione delle variabili. Oltre a quelle conosciute, ne sono state aggiunte di nuove, tra cui le quattro variabili *FC_aperto*, *FC_chiuso*, *FC_alzato* e *FC_abbassato*, di tipo bit, che saranno utilizzate per descrivere lo stato dei componenti della pinza (becchi o carrello), costituendo perciò l'implementazione dei fine corsa.



FC_alzato e *FC_abbassato*, infatti, si riferiscono alle posizioni critiche del carrello; mentre *FC_aperto* e *FC_chiuso* fanno riferimento ai becchi. *FC_alzato* = 1, per esempio, indicando che il carrello si trova nella posizione più alta, attiverà il fine corsa (vedremo come questo accada all'interno delle routine di movimento della pinza). La variabile *misuraRC*, di tipo word, servirà invece per misurare la costante di tempo e verificare lo sforzo cui il motore è sottoposto; infine *sogliaBecchi* e *sogliaCarrello* saranno utilizzate come valori di soglia.

Nella dichiarazione delle costanti, oltre a quelle relative alle porte dei motori (M1a/b e M2a/b) e del ricevitore (IR_SX), sono state dichiarate le costanti ALTO e BASSO, rispettivamente pari ai valori logici 1 e 0, e la costante PORTA_RC relativa a P11, la porta cui è collegato il circuito RC costruito in precedenza.

Nel programma principale i motori vengono inizializzati a 0 (low) e, prima del main, si salta alla routine valori_Fine_corsa che, eseguita una sola volta all'avvio del programma, misura e ottiene i valori di soglia per le costanti di tempo del circuito collegato alla porta P11. È infatti necessario, anzitutto, conoscere il valore relativo al movimento dei motori in condizioni normali, cioè non in posizioni critiche, che sarà quindi l'indicatore di riferimento del valore di tensione. Questo valore dipende dallo stato delle batterie e dalla potenza richiesta dal singolo motore per muovere le parti meccaniche a esso correlate: sarà dunque necessario effettuare due misure separate, una per ciascun motore (in quanto strutturalmente diversi, infatti, il carrello e i becchi richiedono potenze differenti). Tornando al codice, la routine di misurazione (*valori_Fine_corsa*) fa sì che il carrello si alzi per un brevissimo periodo di tempo, durante il quale viene effettuata la misura della costante di tempo di riferimento. In particolare dopo aver acceso il motore X3 in configurazione di salita, viene chiamata la subroutine RCT che, attraverso l'istruzione rctime, misura la costante di tempo, ne memorizza il valore in *misuraRC* e torna alla routine chiamante. Il valore di *misuraRC* viene poi memorizzato in *sogliaCarrello* e il motore viene spento. La stessa sequenza di operazioni viene eseguita per i becchi: in questo caso il motore X2 viene posto nella configurazione di chiusura della pinza e il valore *misuraRC* viene memorizzato in *sogliaBecchi*. All'avvio del programma, questa serie di operazioni produce una breve sequenza di movimenti del carrello e dei becchi. Per non falsare il risultato delle misure, però, è fondamentale che, inizialmente, né il carrello né i becchi siano in posizioni critiche. Al termine della routine di misurazione, si torna al main, che è rimasto immutato rispetto al programma precedente: alla decodifica del segnale IR del telecomando segue il salto a una delle routine di movimento, attraverso lookdown e branch. Le routine di movimento, invece, sono state modificate per permettere la gestione dei fine corsa; tuttavia, come in precedenza, hanno tutte la medesima struttura.

```
'{$STAMP BS2}

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

FC_aperto       var     bit
FC_chiuso       var     bit
FC_alzato       var     bit
FC_abbassato    var     bit

posizione       var     nib

codicePulsante  var     byte

segnaleInizio   var     word
inputIR0        var     word
inputIR1        var     word
inputIR2        var     word
inputIR3        var     word
inputIR4        var     word

misuraRC        var     word
sogliaBecchi    var     word
sogliaCarrello  var     word
impulso         var     word

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

M1a             con     14
M1b             con     15
M2a             con     3
M2b             con     5

IR_SX           con     8
ALTO            con     1
BASSO           con     0
PORTA_RC        con     11

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

' inizialmente i motori sono fermi
low   M1a   low   M1b
low   M2a   low   M2b

goto valori_Fine_corsa

main:

      gosub decodifica
      posizione = 4
      lookdown codicePulsante, [6, 7, 8, 9], posizione
      branch posizione, [Apri, Alza, Chiudi, Abbassa]

goto main

'---------- Routine di movimento della pinza ----------

Alza:
      FC_abbassato = 0
      if FC_Alzato = 1 then main
      low M2a    high  M2b
      gosub RCT
      if misuraRC < sogliaCarrello-2 then F_up

      pulsin IR_SX, BASSO, impulso
      if impulso > 200 then alza
goto arresto

Abbassa:
      FC_alzato = 0
      if FC_abbassato = 1 then main
      high  M2a   low  M2b
      gosub RCT
      if misuraRC < sogliaCarrello-2 then F_down

      pulsin IR_SX, BASSO, impulso
      if impulso >200 then abbassa
goto arresto

Chiudi:
      FC_aperto = 0
      if FC_chiuso = 1 then main
      low   M1a   high  M1b
      gosub RCT
      if misuraRC < sogliaBecchi-2 then F_closed

      pulsin IR_SX, BASSO, impulso
      if impulso >200 then chiudi
goto arresto

Apri:
      FC_chiuso = 0
      if FC_aperto = 1 then main
      high  M1a   low  M1b
      gosub RCT
      if misuraRC < sogliaBecchi-3 then F_open

      pulsin IR_SX, BASSO, impulso
      if impulso >200 then apri
goto arresto

Arresto:
      low   M1a   low   M1b
      low   M2a   low   M2b
      pause 10
goto main

'---------- Gestione Fine Corsa ----------
```

LE FASI DI PROGRAMMAZIONE

Ne analizzeremo, dunque, solo una: la routine Alza (a pag. 161). Anzitutto, settandone il valore a 0, c'è lo sblocco del fine corsa *FC_abbassato*, necessario per elevare il carrello, qualora fosse bloccato in posizione critica (*FC_abbassato* = 1). Segue il test (if... then) sull'altro fine corsa (*FC_alzato*): se il valore è 1 (fine corsa attivo), non è possibile alzare ulteriormente il carrello, pertanto si torna direttamente al main senza far eseguire movimenti al motore; viceversa, (valore 0: fine corsa disattivato) si procede con l'accensione del motore X2 secondo la configurazione delle porte per la salita. Ed è a questo punto che è necessario verificare lo sforzo eseguito dal motore durante il movimento: anche se il fine corsa è disattivato, infatti, il carrello potrebbe comunque essere in prossimità della posizione critica alta. Viene dunque chiamata la subroutine RCT che, misurato il valore della costante di tempo, lo memorizza in *misuraRC*: se il motore è sottoposto a sforzo, la tensione diminuisce e il valore misurato è inferiore a quello di riferimento iniziale (registrato con valori_Fine_corsa); verifica che si ottiene comparando (if... then) il valore di *misuraRC* con quello di *sogliaCarrello*. Noterai però che il test viene eseguito sul valore *sogliaCarrello*-2, perché offre un ulteriore margine (-2) rispetto a possibili errori di misura (come succedeva per la navigazione con i fotoresistori). Tale margine (-3 in altre routine) non è tassativo, bensì frutto di una prova empirica: dovrai quindi fare altrettanto, verificando se con questi margini la pinza si comporta in maniera corretta e, in caso contrario, aggiungendo o sottraendo qualche unità a seconda dell'esigenza. Inoltre, per avere un miglior riscontro durante le prove puoi aggiungere una riga di debug dopo la chiamata a RCT: visualizzando il valore di *misuraRC*, avrai un riscontro della variazione delle costanti di tempo. In generale, comunque, se il valore misurato è inferiore alla soglia, il motore è giunto nel punto critico e, quindi, occorre spegnerlo e attivare *FC_alzato*, saltando alla routine F_up. Questa, impostato il valore della variabile a 1 (fine corsa attivo), passa alla routine di arresto che termina tornando al main. A questo punto, pur premendo (o tenendo premuto) il tasto ▲, la condizione posta dal primo test (if... then) della routine Alza sarebbe verificata e i motori, fermi, non verrebbero sforzati. Nel caso in cui, invece, il valore di *misuraRC* fosse superiore o uguale alla soglia, il programma procederebbe come il precedente: verificando, con pulsin, l'eventuale emissione di nuovi impulsi e, in base a questi, procedendo nel movimento o arrestando i motori. Tutte le altre routine di gestione dei movimenti, con le opportune caratterizzazioni, hanno la stessa struttura.

Riassumendo: nella gestione dei fine corsa è stato necessario introdurre a livello hardware un nuovo circuito che monitorasse la tensione di alimentazione Vdd.

Nella gestione software, poi, campionati i valori di soglia in condizioni normali,

```
'---------- Gestione Fine Corsa ----------

' ---- Routine di stato dei fine corsa -------

F_up:
      FC_alzato =1
      debug "Carrello alzato", CR
goto arresto

F_down:
      FC_abbassato =1
      debug "Carrello abbassato", CR
goto arresto

F_open:
      FC_aperto=1
      debug "Bocchi aperti", CR
goto arresto

F_closed:
      FC_chiuso=1
      debug "Presa effettuata", CR
goto arresto

'------ Misurazione dei valori di soglia dei fine corsa ------

valori_Fine_corsa:

' Misura il valore di soglia per il carrello
      low M2a     high  M2b
      gosub RCT
      sogliaCarrello = misuraRC
      low M2a     low  M2b
      pause 100

' Misura il valore di soglia per la pinza
      low M1a     high  M1b
      gosub RCT
      sogliaBocchi = misuraRC
      low M1a     low  M1b
      misuraRC = 0

goto main

end
'------ Subroutine lettura Rctime ----------

RCT:
      high PORTA_RC
      pause 2
      rctime PORTA_RC, ALTO, misuraRC
      pause 100

return

'---------- Subroutine decodifica impulsi IR ----------

decodifica:
   rileva:
      pulsin IR_SX, BASSO, segnaleInizio
      pulsin IR_SX, BASSO, inputIR0
      pulsin IR_SX, BASSO, inputIR1
      pulsin IR_SX, BASSO, inputIR2
      pulsin IR_SX, BASSO, inputIR3
      pulsin IR_SX, BASSO, inputIR4

   if segnaleInizio < 1000 then rileva

      codicePulsante.bit0 = inputIR0.bit9
      codicePulsante.bit1 = inputIR1.bit9
      codicePulsante.bit2 = inputIR2.bit9
      codicePulsante.bit3 = inputIR3.bit9
      codicePulsante.bit4 = inputIR4.bit9

return
```

si è introdotta la gestione del fine corsa, per ogni movimento, rilevando e confrontando il valore di RC del circuito con quello di riferimento per poi settare in modo opportuno il valore delle variabili di stato. Una variante di progetto che è possibile introdurre nel programma appena visto è l'automatizzazione dell'operazione di chiusura della pinza, in modo che, premendo una sola volta il tasto ▶◀ del telecomando, la pinza si chiuda da sé fino a far presa sull'oggetto (o fino al fine corsa). Per farlo, basta modificare la routine Chiudi come illustrato sotto, ossia sostituendo le ultime tre righe del codice originale (cioè pulsin, if... then e goto arresto) con un'unica istruzione: goto Chiudi. Si realizza così un ciclo tra Chiudi e goto Chiudi, interrotto dal test di *misuraRC* (che imposta il fine corsa e spegne il motore) solo quando la pinza abbia di fatto raggiunto la posizione di presa.

```
Chiudi:
      FC_aperto = 0
      if FC_chiuso = 1 then main
      low   M1a:   high  M1b
      gosub RCT
      if misuraRC < sogliaBocchi-2 then F_closed
goto Chiudi
```