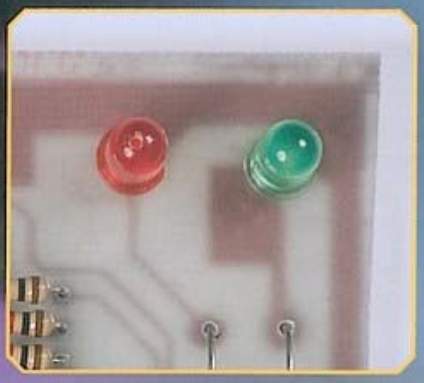
LE FUNZIONI BASE (seconda parte)

Continuiamo quanto abbiamo iniziato nel fascicolo precedente analizzando le funzioni di base di I/O utili per la scrittura del firmware di RZB-1p.

Nel Workshop 71 abbiamo iniziato a commentare la prima parte delle funzioni utili alla programmazione di RZB-1p, dedicandoci, nello specifico, alle **funzioni di sistema e di movimento.** Passiamo ora a studiare le procedure che ci permetteranno di interfacciarci con i dispositivi di I/O del robot.

LE FUNZIONI DI I/O ›››
Le **funzioni di I/O** (ossia di *input/output*) sono le procedure che possiamo utilizzare all'interno del nostro firmware per **gestire l'accensione dei LED** e **acquisire lo stato dei sensori.** Iniziamo ad analizzarle proprio partendo dai semplici meccanismi di controllo dei due dispositivi luminosi.



La scheda di controllo di RZB-1p è dotata di una coppia di LED che possiamo accendere utilizzando le funzioni base del firmware.

Il controllo dei LED

```
/* Definizione della funzione LED_Rosso(). Riceve come parametro i valori simbolici ACCESO e SPENTO e permette di impostare lo stato di accensione del LED rosso */

void LED_Rosso(int stato)
{
     if(stato == ACCESO || stato == SPENTO)
     {
          LED_ROSSO = stato;
     }
}

/* Definizione della funzione LED_Verde(). Riceve come parametro i valori simbolici ACCESO e SPENTO e permette di impostare lo stato di accensione del LED rosso */

void LED_Verde(int stato)
{
     if(stato == ACCESO || stato == SPENTO)
     {
          LED_VERDE = stato;
     }
}
```

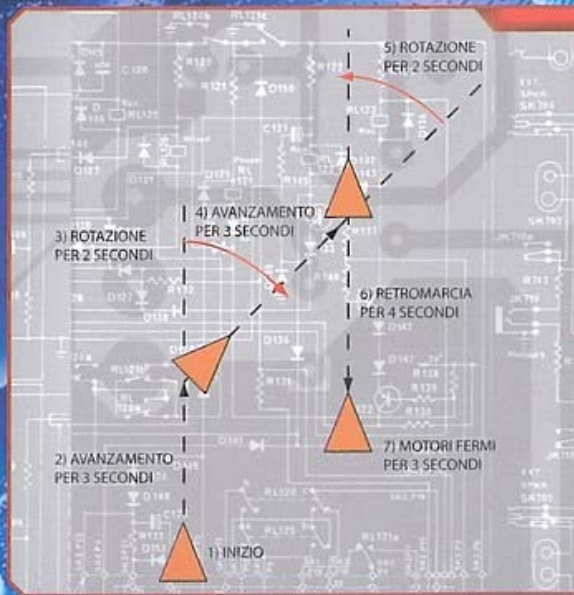
NOTA BENE›››
Il codice sorgente mostrato in queste pagine è parte del template di base che abbiamo introdotto con il Workshop 71. Non può essere, quindi, compilato a meno che non venga integrato con le definizioni simboliche e le funzioni mostrate in precedenza. Per conoscere nel dettaglio il significato specifico dei valori simbolici in uso in questo articolo, ti rimandiamo alle pagine 1 e 2 del fascicolo precedente, nelle quali compare l'elenco di direttive #define che dichiarano questi elementi di programma.

ACQUISIRE GLI INPUT ››› Passiamo ora a vedere **come gestire le linee di input digitale del sistema.** Nel fascicolo 70 abbiamo anticipato una lista di 5 funzioni (Ostacolo_a_Ovest, Ostacolo_a_NordOvest ecc.) che permettono di **rilevare la presenza di ostacoli nelle 5 direzioni di scansione dei sensori del robot.** Poiché tale nomenclatura è funzionale esclusivamente all'attuale configurazione del robot, ma non è generale, affiancheremo tali procedure ad altrettanti 'alias', ossia procedure aventi nomi differenti, ma con effetti praticamente analoghi. Il codice associato è contenuto in questa pagina e nella successiva.

Le funzioni di input

```
/* La funzione Ostacolo_a_Ovest restituisce SI (ossia 1) o NO (ossia 0) in base allo stato
logico del SENSORE_OVEST  (ricordiamo che i sensori producono un livello alto in presenza
di un ostacolo, basso altrimenti)*/

int Ostacolo_a_Ovest()
{
   return SENSORE_OVEST;
}

/* La funzione Ostacolo_a_NordOvest restituisce SI (ossia 1) o NO (ossia 0) in base allo
stato logico del SENSORE_NORDOVEST  (ricordiamo che i sensori producono un livello alto in
presenza di un ostacolo, basso altrimenti)*/

int Ostacolo_a_NordOvest()
{
   return SENSORE_NORDOVEST;
}


/* La funzione Ostacolo_a_Nord restituisce SI (ossia 1) o NO (ossia 0) in base allo stato
logico del SENSORE_NORD  (ricordiamo che i sensori producono un livello alto in presenza di
un ostacolo, basso altrimenti)*/

int Ostacolo_a_Nord()
{
   return SENSORE_NORD;
}


/* La funzione Ostacolo_a_NordEst restituisce SI (ossia 1) o NO (ossia 0) in base allo
stato logico del SENSORE_NORDEST  (ricordiamo che i sensori producono un livello alto in
presenza di un ostacolo, basso altrimenti)*/

int Ostacolo_a_NordEst()
{
   return SENSORE_NORDEST;
}


/* La funzione Ostacolo_a_Est restituisce SI (ossia 1) o NO (ossia 0) in base allo stato
logico del SENSORE_EST  (ricordiamo che i sensori producono un livello alto in presenza di
un ostacolo, basso altrimenti)*/

int Ostacolo_a_Est()
{
   return SENSORE_EST;
}


/* Gli alias generici */
/* La funzione StatoInput1 restituisce 0 o 1 in base allo stato logico della linea di input
```

```
numero 1 della scheda di controllo di RZB-1p */

int StatoInput1()
{
   return INPUT_1;
}


/* La funzione StatoInput1 restituisce 0 o 1 in base allo stato logico della linea di input
numero 2 della scheda di controllo di RZB-1p */

int StatoInput2()
{
   return INPUT_2;
}


/* La funzione StatoInput1 restituisce 0 o 1 in base allo stato logico della linea di input
numero 3 della scheda di controllo di RZB-1p */

int StatoInput3()
{
   return INPUT_3;
}

/* La funzione StatoInput1 restituisce 0 o 1 in base allo stato logico della linea di input
numero 4 della scheda di controllo di RZB-1p */

int StatoInput4()
{
   return INPUT_4;
}

/* La funzione StatoInput1 restituisce 0 o 1 in base allo stato logico della linea di input
numero 5 della scheda di controllo di RZB-1p */

int StatoInput5()
{
   return INPUT_5;
}
```



IL PRIMO ESPERIMENTO CON IL FIRMWARE DI RZB-1

Con la porzione di codice mostrata sopra abbiamo concluso la presentazione delle funzioni base di RZB-1p (nulla ti vieta, comunque di 'espandere' tale 'libreria' con nuove procedure personalizzate). Nel prossimo fascicolo affronteremo i primi esperimenti di programmazione del robot realizzando semplici firmware. In particolare, il primo programma che scriveremo sarà dedicato al test dei motori e sarà costituito da una sequenza di movimenti temporizzati (non verrà quindi usata la scheda sensori). Lo schema a lato mostra l'elenco di movimenti che implementeremo: avanzamento per 3 sec, rotazione a destra per 2 sec, avanzamento per 3 sec, rotazione a sinistra per 2 sec e indietreggiamento per 4 sec, spegnendo, infine, i motori.