```
1    /*
2    contiene tutte le routine per la  gestione dei sensori per le gare explorer
3    */
4
5    /*CmpCal ******************************************************************************
6    procedura calibrazione bussola
7    si posiziona sui quattro punti cardinali e scrive nel registro di calibrazione
8    */
9
10   void CmpCal (void)
11   {
12      BeepCount=6;
13      InitFlag = 0;     // esce dallo stato init
14
15      // ---- init movimento
16      FlagBumpers = 1;  // disattiva bumpers
17      FlagEye=0;        // sensori IR disabilitati
18      FlagLight=0;      // sensori luce disabilitati
19      FlagSound=0;      // sensori suono disabilitati
20      FlagGas=0;        // sensori gas disabilitati
21      TimerBumpers = 0; // bumpers disabilitati per 1Sec
22      Space2RunFlag = 0;// reset di qualsiasi routine di movimento
23      // fermo per 5 sec poi parte calibrazione
24           PathSeq [0] = 20;  // fermo 10 sec
25           PathSeq [1] = 23;  // fermo 5 secondi e calibra bussola
26           PathSeq [2] = 14;  // gira 90°
27           PathSeq [3] = 23;  // fermo 5 secondi e calibra bussola
28           PathSeq [4] = 14;  // gira 90°
29           PathSeq [5] = 23;  // fermo 5 secondi e calibra bussola
30           PathSeq [6] = 14;  // gira 90°
31           PathSeq [7] = 23;  // fermo 5 secondi e calibra bussola
32           PathSeq [8] = 14;  // gira 90°
33           PathSeq [9] = 20;  // fermo 10 sec
34           PathSeq [10] = 0;  // fine sequenza
35           PathSeqPointer = 0;// inizializza sequenza
36
37           En=1;         // avvia motori
38      TimerLcd=LcdCycle;      // tempo tra le visualizzazioni del display
39      LcdInitStatus=0;        // azzera contatore stati init LCD
40      LcdPutCharFlag=0;       // azzera trasmissione byte su LCD
41      LcdStringPtr=0xFF;      // disabilita LcdStringPut
42      LcdStatus=0;            // parte dal primo passo del ciclo
43      DisplayStatus=2;        // visualizza bussola
44
45      LedRossoOFF;            // led spenti
46      LedGialloOFF;
47      LedVerdeOFF;
48      LcdPutChar (0x01, 0, 4);// clear display
49
50
```

```
51    } // CmpCal
52    /*****************************************************************************/
53
54    /*CmpRead ********************************************************************
55    Legge il valore attuale della bussola tramite I2C
56    viene chiamata ogni x ms e solo se il buffer I2c e' vuoto
57    se il flag = 0 avvia la lettura e alza il flag
58    quando il buffer sara' di nuovo vuoto la lettura e' terminata
59    */
60
61    void CmpRead (void)
62    {
63       if (FlagCmpReg15)
64       {// e' nella fase di calibrazione, deve scrivere il valore 255 nel registro 15
65        // in  corrispondenza di ogni punto cardinale
66          I2c[CmpPtr].Flag.Tx =2;        // due byte da trasmettere
67          I2c[CmpPtr].TxBuff[0] = 15 ;  // scrive nel registro 15
68          I2c[CmpPtr].TxBuff[1] = 255 ; // il valore 255
69          FlagCmpReg15=0; // torna nella situazione normale
70       }
71       else
72       {
73          if (!FlagCmp)
74          {// la prima volta che viene chiamata avvia la lettura I2c
75           // scrive numero registro da leggere e poi lo legge
76             I2c[CmpPtr].Flag.Tx = 1;    // un byte da trasmettere
77             I2c[CmpPtr].TxBuff[0] = 1 ;// chiede il valore del registro 1
78             I2c[CmpPtr].Flag.Rx = 1;    // un byte da leggere
79             FlagCmp=1;                  // attende fine lettura
80          }
81          else
82          {// finita la lettura resetta flag e timer, verra' chiamata al prossimo ciclo
83             TimerCmp=CmpCycle;
84             FlagCmp=0;
85             CmpBearing=I2c[CmpPtr].RxBuff[0];   // legge il valore dal buffer
86          }
87       }
88
89
90    } // CmpRead
91    /*****************************************************************************/
92
93    /*TargetOk ********************************************************************
94    Segnala che ha raggiunto l'obiettivo Target:
95    1 = gas     LedRosso
96    2 = luce    LedVerde
97    3 = suono   LedGiallo
98
99       ferma n secondi
100      accende il led Target
```

```
101     suona il buzzer
102     disabilita i sensori per un periodo sufficiente a ripartire dalla sorgente
103     gira + o - 90°
104     riparte
105  */
106
107  void TargetOk (char Target)
108  {
109     /*
110        da notare che tutte le manovre non sono eseguite in sequenza
111        ma sono impostate per essere gestite in parallelo dal main
112     */
113        BeepCount = Target * 2;     // suona Target beep
114        TimerBeep = BeepTime;       // reset tempo beep
115        TimerStop = TimeMotStopTarget;    // pausa motori per N Sec
116        FlagStop = 1;                     // abilita routine Stop motori
117        FlagTargetLight=1;                // riabilita tutti i sensori
118        FlagTargetSound=1;                // eventualmente ancora disabilitati
119        FlagTargetGas=1;
120
121        // si sfrutta la routine Stop motori anche per la temporizzazione dei led
122        // verrannno spenti prima di riavviare i motori: si risparmia un timer
123
124        switch (Target)   // quale quale obiettivo ha raggiunto?
125           {
126              case 1:                 // gas
127                    LedRossoON;
128                    FlagTargetGas = 0;
129                    TimerSensor = TimeDisSensGas; // reset timer disabilitazione sensori
130
131                    // disabilita routine sensori gas per il tempo necessario
132                    // ad allontanarsi.
133                    break;
134
135              case 2:                 // luce
136                    LedVerdeON;
137                    FlagTargetLight = 0;
138                    TimerSensor = TimeDisSensLight;  // reset timer disabilitazione sensori
139                    // disabilita routine sensori luce per il tempo necessario
140                    // ad allontanarsi.
141                    break;
142
143              case 3:                 // suono
144                    LedGialloON;
145                    FlagTargetSound = 0;
146                    TimerSensor = TimeDisSensSound;  // reset timer disabilitazione sensori
147                    // disabilita routine sensori suono per il tempo necessario
148                    // ad allontanarsi.
149                    break;
150
```

```
151               default:
152                     LedVerdeOFF;   // a default spegne tutti i led
153                     LedRossoOFF;
154                     LedGialloOFF;
155                     break;
156               }
157
158         // cambia direzione
159               Space2RunFlag = 0; // reset di qualsiasi routine di movimento
160               PathSeq [0] = 2;   // indietro n cm
161               PathSeq [1] = 50;  // gira 90° o -90° randomicamente
162               PathSeq [2] = 255; // cammina avanti a velocita' costante
163               PathSeq [3] = 0;   // fine sequenza
164               PathSeqPointer = 0;// inizializza sequenza
165
166   } // TargetOk
167   /*****************************************************************************/
168
169   /*SegueLuce ******************************************************************
170   se la luce e' piu' forte a destra, gira a destra
171   se e' piu' forte a sinistra, gira a sinistra
172   altrimenti va dritto
173   */
174
175   void SegueLuce (void)
176   {
177         if ((FrLVal > FrCVal) && (FrLVal > FrRVal)) // la luce e' a sinistra
178            {  // gira a sinistra
179               FlagTargetLight = 0; // disabilita routine sensori luce fino a fine manovra
180               TimerSensor = TimeDisSensLight;// reset timer disabilitazione sensori
181               Space2RunFlag = 0;   // reset di qualsiasi routine di movimento
182               PathSeq [0] = 9;     // -10°
183               PathSeq [1] = 200;   // cammina avanti a velocita' ridotta
184               PathSeq [2] = 254;   // riabilita routine sensori luce
185               PathSeq [3] = 0;     // fine sequenza
186               PathSeqPointer = 0;  // inizializza sequenza
187            }
188         if ((FrRVal > FrCVal) && (FrRVal > FrLVal)) // la luce e' a destra
189            {  // gira a destra
190               FlagTargetLight = 0; // disabilita routine sensori luce fino a fine manovra
191               TimerSensor = TimeDisSensLight;// reset timer disabilitazione sensori
192               Space2RunFlag = 0;   // reset di qualsiasi routine di movimento
193               PathSeq [0] = 8;     // 10°
194               PathSeq [1] = 200;   // cammina avanti a velocita' ridotta
195               PathSeq [2] = 254;   // riabilita routine sensori luce
196               PathSeq [3] = 0;     // fine sequenza
197               PathSeqPointer = 0;  // inizializza sequenza
198            }
199         // se la luce non e' ne a destra ne a sinistra, allora prosegue dritto
200               DesSpeedR=lowSpeed; // ma approccia l'ostacolo a velocita' bassa
```

```
201             DesSpeedL=lowSpeed;
202
203 } // SegueLuce
204 /*****************************************************************************/
205
206
207 /*Light *********************************************************************
208 Controllo sensori di luce a fotoresistenze
209 Il valore letto e'  quello fornito da ReadAD()
210
211 if FrL>Light2 or FrC>Light2 or FrR>Light2; un sensore ha superato la seconda soglia il
212                                           robot e' vicino alla sorgente luminosa
213    if (distanza dall'ostacolo < TargetDist); se la distanza dalla luce rientra nei valori
214                                              imposti dal regolamento
215          segnala "obiettivo raggiunto"
216    altrimenti
217          gira verso la luce
218
219 altrimenti
220       if FrL>Light1 or FrC>Light1 or FrR>Light1; un sensore ha superato la prima soglia
221          if FrL>FrC and FrL>FrR
222             gira 10° a sinistra inseguendo la luce
223          if FrR>FrC and FrR>FrL
224             gira 10° a destra inseguendo la luce
225 */
226
227 void Light(void)
228 {
229
230 if ((FrLVal>=Light2)||(FrCVal>=Light2)||(FrRVal>=Light2))
231 // la sorgente di luce e' vicina
232 {
233    if (
234        ((EyeLlVal>=TargetDist) && (FrLVal>=Light2))||
235        ((EyeLfVal>=TargetDist) && (FrCVal>=Light2))||
236        ((EyeRfVal>=TargetDist) && (FrCVal>=Light2))||
237        ((EyeRlVal>=TargetDist) && (FrRVal>=Light2))
238       )
239    // se vede molta luce da un lato e dallo stesso lato rivela
240    // un ostacolo alla distanza stabilita
241    {
242       if (DebounceLight >= DebounceLightCount)
243       // solo dopo aver rivelato alcuni segnali positivi consecutivi
244       // conferma l'obiettivo, serve ad evitare falsi segnali
245       {
246          TargetOk(2);   // segnala che ha raggiunto una sorgente di luce
247          DebounceLight = 0; // resetta il contatore debounce
248       }
249       else
250       {
```

```
251           DebounceLight++;
252           DesSpeedR=lowSpeed; // rallenta in vicinanza dell'ostacolo
253           DesSpeedL=lowSpeed;
254
255        }
256     }
257     else
258     // e' ancora fuori dal cerchio di "bersaglio raggiunto", insegue la luce
259     {
260        DebounceLight = 0; // resetta il contatore debounce
261        SegueLuce();
262     }
263  }
264  else
265  // la luce non e' vicina
266  {
267     if ((FrLVal>=Light1)||(FrRVal>=Light1))
268     // però e' stata individuata dai sensori laterali
269     // se e' stata rivelata dal sensore centrale, prosegue dritto
270     {
271        DebounceLight = 0; // resetta il contatore debounce
272        SegueLuce();
273     }
274  }
275
276  FlagLight = 0; // la routine si auto-disabilita appena eseguita
277                 // sara' la routine ReadAD a sbloccarla appena terminata una nuova misura
278
279  } // Light
280  /*****************************************************************************/
281
282  /*Sound ********************************************************************
283  Controllo sensore di suono
284
285  il circuito per la rivelazione del suono miscela il segnale proveniente dai tre microfoni,
286  filtra i rumori ed invia il segnale utile al tone decoder.
287  Un debouncer SW sull'uscita (digitale) del tone decoder, filtra eventuali falsi allarmi
288
289  */
290
291  void Sound(void)
292  {
293     if (
294         ((EyeLlVal>=TargetDist) && (!SoundIn))||
295         ((EyeLfVal>=TargetDist) && (!SoundIn))||
296         ((EyeRfVal>=TargetDist) && (!SoundIn))||
297         ((EyeRlVal>=TargetDist) && (!SoundIn))
298        )
299     // ha rivelato il suono ed e' vicino all'obiettivo
300     {
```

```
301         if (DebounceSound >= DebounceSoundCount)
302         // solo dopo aver rivelato alcuni segnali positivi consecutivi
303         // conferma l'obiettivo, serve ad evitare falsi segnali
304         {
305            TargetOk(3);   // segnala che ha raggiunto una sorgente di suono
306            DebounceSound = 0; // resetta il contatore debounce
307         }
308         else
309         {
310            DebounceSound++;
311            DesSpeedR=lowSpeed; // rallenta in vicinanza dell'ostacolo
312            DesSpeedL=lowSpeed;
313         }
314      }
315      else
316      // non rivela suono oppure e' ancora fuori dal cerchio di "bersaglio raggiunto"
317      {
318         DebounceSound = 0; // resetta il contatore debounce
319      }
320
321   FlagSound = 0; // la routine si auto-disabilita appena eseguita
322                  // sara' la routine ReadAD a sbloccarla appena terminata una
323                  // nuova misura con la stessa temporizzazione della routine gas
324
325   } // Sound
326   /*****************************************************************************/
327
328
329   /*Gas ************************************************************************
330   Controllo sensori di gas
331   Il valore letto e'  quello fornito da ReadAD()
332
333   il circuito per la rivelazione del gas miscela il segnale proveniente dai due TGS822
334   */
335
336   void Gas(void)
337   {
338      if (GasVal > GasOn)
339      // ha rivelato una concentrazione di gas superiore alla soglia impostata
340      {
341         TargetOk(1);   // segnala che ha raggiunto una sorgente di gas
342      }
343
344   FlagGas = 0; // la routine si auto-disabilita appena eseguita
345                // sara' la routine ReadAD a sbloccarla appena terminata una nuova misura
346
347   } // Gas
348   /*****************************************************************************/
349
350
```

```
351  /*Eye ************************************************************************
352  Controllo sensori di prossimita' ad IR
353  Questa routine ha priorita' bassa rispetto alle altre routine di movimento
354  viene quindi eseguita solo se non ci sono manovre in atto, anche se innescate
355  dalla routine stessa
356     se Space2RunFlag = 1, e' ancora in esecuzione una singola routine di movimento
357     se PathSeq[PathSeqPointer] = 0 -> fine sequenza routines di movimento (EOL)
358     se FlagStop = 1 -> la routine Stop ha fermato i motori
359  Il valore letto e'  quello fornito da ReadAD()
360
361  sono prese in considerazione due soglie di allarme
362  sotto LowDist il bot comincia a rallentare dalla velocita' constSpeed fino a lowSpeed
363  sotto MinDist gira in senso opposto al lato dell'ostacolo rivelato
364  tra le due soglie la velocita' e' calcolata secondo la formula
365  varSpeed=(MinDist-EyeXVal)*(constSpeed-lowSpeed)/(MinDist-LowDist)+lowSpeed;
366
367
368  // ATTENZIONE: a tensione minore corrisponde distanza maggiore
369
370  ------------------------------------------------------------------------------------
371  lettura sensori  IR Sharp GP2D120
372  distanza dal paraurti
373
374  distanza    cm:25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02
375  lettura AD hex:10 11 13 14 16 18 1a 1e 20 24 26 2b 30 33 36 3b 3f 46 4e 5a 66 77 8d a2
376
377  sotto i 2cm la  tensione ricomincia a  scendere.
378  ------------------------------------------------------------------------------------
379  */
380
381  void Eye(void)
382  {
383  int varSpeed;  // velocita' di approccio, variabile in funzione della distanza dall'ostacolo
384  char EyeRVal;  // valore attuale del rivelatore IR destro
385  char EyeLVal;  // valore attuale del rivelatore IR sinistro
386  char GiraSX;   // angolo di cui girare
387  char GiraDX;   // angolo di cui girare
388
389  if (EyeRlVal > EyeRfVal)    // prende in considerazione solo il sensore destro piu' vicino all'ostacolo
390  {
391     EyeRVal = EyeRlVal;  // sensore laterale
392     GiraSX = 9;          // 10 gradi a sinistra
393  }
394  else
395  {
396     EyeRVal = EyeRfVal;  // sensore frontale
397     GiraSX = 13;            // 45 gradi a sinistra
398  }
399
400  if (EyeLlVal > EyeLfVal)    // prende in considerazione solo il sensore sinistro piu' vicino all'ostacolo
```

```
401  {
402     EyeLVal = EyeLlVal;  // sensore laterale
403     GiraDX = 8;          // 10 gradi a destra
404  }
405  else
406  {
407     EyeLVal = EyeLfVal;  // sensore frontale
408     GiraDX = 12;            // 45 gradi a destra
409  }
410
411
412  if ((EyeRVal < LowDist) && (EyeLVal < LowDist)) // ostacolo a distanza maggiore di LowDist
413  {
414      DesSpeedR=constSpeed; // a default cammina a velocita' costante
415      DesSpeedL=constSpeed; // a default cammina a velocita' costante
416  }
417
418  else  // ostacolo a distanza minore di LowDist da uno dei due lati
419
420  {
421     Space2RunFlag = 0; // reset di qualsiasi routine di movimento
422     if ((EyeRVal < MinDist) && (EyeLVal < MinDist)) // ostacolo a distanza compresa tra LowDist e MinDist
423     {
424        //calcola la velocita' di approccio in funzione della distanza dall'ostacolo
425        if (EyeRVal >= LowDist) varSpeed=(MinDist-EyeRVal)*(constSpeed-lowSpeed)/(MinDist-LowDist)+lowSpeed;
426        if (EyeLVal >= LowDist) varSpeed=(MinDist-EyeLVal)*(constSpeed-lowSpeed)/(MinDist-LowDist)+lowSpeed;
427
428        if (varSpeed > constSpeed) varSpeed=constSpeed;
429        if (varSpeed < lowSpeed  ) varSpeed=lowSpeed;
430
431        DesSpeedR=varSpeed; // cammina a velocita' di approccio
432        DesSpeedL=varSpeed; // cammina a velocita' di approccio
433     }
434
435     else  // ostacolo a distanza minore di MinDist da uno dei due lati
436
437     {
438        if (EyeRVal > MinDist)  //prossimita' a destra -> gira a sinistra
439        {
440              PathSeq [0] = 3;     // indietro 0,1mm
441              PathSeq [1] = GiraSX;// -10° se sensore laterale, -45° se frontale
442              PathSeq [2] = 255;   // cammina avanti a velocita' costante
443              PathSeq [3] = 0;     // fine sequenza
444              PathSeqPointer = 0;  // inizializza sequenza
445        }
446
447        if (EyeLVal > MinDist)  //prossimita' a sinistra -> gira a destra
448        {
449              PathSeq [0] = 3;     // indietro 0,1mm
450              PathSeq [1] = GiraDX;// 10° se sensore laterale, 45° se frontale
```

```
451             PathSeq [2] = 255;   // cammina avanti a velocita' costante
452             PathSeq [3] = 0;     // fine sequenza
453             PathSeqPointer = 0;  // inizializza sequenza
454         }
455     }
456 }
457
458 FlagEye = 0; // la routine si auto-disabilita appena eseguita
459              // sara' la routine ReadAD a sbloccarla appena terminata una nuova misura
460
461 } // Eye
462 /*****************************************************************************/
463
464 /*ReadAD ***********************************************************************
465 legge le porte A/D in sequenza e mette i valori letti nelle rispettive variabili
466 la lettura delle porte analogiche richiede alcune pause e l'attesa di un flag di
467 conferma. Particolare cura va quindi posta per effettuare questi ritardi senza
468 coinvolgere le temporizzazioni delle altre routine, anche in questo caso si fara'
469 ricorso a flag e variabili globali.
470
471 Quando e' passato il tempo di attesa Tacq, questa routine viene chiamata ad ogni
472 ciclo  di main.
473 La sequenza di operazioni per la lettura A/D e' scandita da un flag di stato:
474
475 TimerAD>timeout   FlagAD         ADGO     Stato
476   0                 x             x       Tacq o primo ciclo dopo boot
477   1                 0             0       Passato Tacq, si avvia conversione
478   1                 1             1       Conversione avviata, attesa ADGO
479   1                 1             0       Conversione effettuata, lettura valore,
480                                             reset timer e stato, imposta nuova lettura
481 */
482
483 void ReadAD(void)
484 {
485 // le porte da leggere sono: EyeRl, EyeLl, EyeRf, EyeLf, FrR, FrL, FrC
486
487 unsigned int i = 0; // contatore generico
488 #define MaskRead  0b00111100  // per leggere la porta in uso su ADCON0
489 #define MaskClear 0b11000011  // per resettare la porta in uso su ADCON0
490
491 if (!FlagAD)           // primo step: e' appena passato Tacq
492 {
493    ADCON0bits.GO_DONE = 1;     // avvio conversione
494    FlagAD = 1;         // passa allo  stato 2
495 }
496
497 else                   // secondo step: in attesa fine conversione
498 {
499    if (!ADCON0bits.GO_DONE)  // terzo step, quando GO DONE torna = 0 -> conversione finita
500                        // per inserire altre porte da leggere: aggiungere "case"
```

```
501     {
502           switch (ADCON0 & MaskRead)   // quale porta analogica sta usando?
503           {
504              case EyeRf:                // Sensore IR Frontale Destro
505                    EyeRfVal = ADRESH;   // solo il byte alto della lettura e' usato
506                    ADCON0 &= MaskClear; // clear bit relativi alla porta
507                    ADCON0 |= EyeLf;     // si imposta la successiva porta da leggere
508                    FlagEye = 1;// la misura e' stata effettuata, si abilita la routine Eye
509                                // al controllo degli ostacoli, questa verra' eseguita una
510                                // volta e poi rimarra' ferma fino alla prossima lettura valida
511                    break;
512
513              case EyeLf:                // Sensore IR Frontale Sinistro
514                    EyeLfVal = ADRESH;   // solo il byte alto della lettura e' usato
515                    ADCON0 &= MaskClear; // clear bit relativi alla porta
516                    ADCON0 |= EyeRl;     // si imposta la successiva porta da leggere
517                    FlagEye = 1;// la misura e' stata effettuata, si abilita la routine Eye
518                                // al controllo degli ostacoli, questa verra' eseguita una
519                                // volta e poi rimarra' ferma fino alla prossima lettura valida
520                    break;
521
522              case EyeRl:                // Sensore IR Laterale Destro
523                    EyeRlVal = ADRESH;   // solo il byte alto della lettura e' usato
524                    ADCON0 &= MaskClear; // clear bit relativi alla porta
525                    ADCON0 |= EyeLl;     // si imposta la successiva porta da leggere
526                    FlagEye = 1;// la misura e' stata effettuata, si abilita la routine Eye
527                                // al controllo degli ostacoli, questa verra' eseguita una
528                                // volta e poi rimarra' ferma fino alla prossima lettura valida
529                    break;
530
531              case EyeLl:                // Sensore IR Laterale Sinistro
532                    EyeLlVal = ADRESH;   // solo il byte alto della lettura e' usato
533                    ADCON0 &= MaskClear; // clear bit relativi alla porta
534                    ADCON0 |= FrR;       // si imposta la successiva porta da leggere
535                    FlagEye = 1;// la misura e' stata effettuata, si abilita la routine Eye
536                                // al controllo degli ostacoli, questa verra' eseguita una
537                                // volta e poi rimarra' ferma fino alla prossima lettura valida
538                    break;
539
540              case FrR:                  // FotoResistenza Destra
541                    FrRVal = ADRESH;     // solo il byte alto della lettura e' usato
542                    ADCON0 &= MaskClear; // clear bit relativi alla porta
543                    ADCON0 |= FrC;       // si imposta la successiva porta da leggere
544                    FlagLight = 1;// la misura e' stata effettuata, si abilita la routine Light
545                                  // alla ricerca della luce, questa verra' eseguita una
546                                  // volta e poi rimarra' ferma fino alla prossima lettura valida
547                    break;
548
549              case FrC:                  // FotoResistenza Centrale
550                    FrCVal = ADRESH;     // solo il byte alto della lettura e' usato
```

```
551                     ADCON0 &= MaskClear; // clear bit relativi alla porta
552                     ADCON0 |= FrL;       // si imposta la successiva porta da leggere
553                     FlagLight = 1;// la misura e' stata effettuata, si abilita la routine Light
554                                   // alla ricerca della luce, questa verra' eseguita una
555                                   // volta e poi rimarra' ferma fino alla prossima lettura valida
556                     break;
557
558               case FrL:                   // FotoResistenza Sinistra
559                     FrLVal = ADRESH;      // solo il byte alto della lettura e' usato
560                     ADCON0 &= MaskClear; // clear bit relativi alla porta
561                     ADCON0 |= GasIn;      // si imposta la successiva porta da leggere
562                     FlagLight = 1;// la misura e' stata effettuata, si abilita la routine Light
563                                   // alla ricerca della luce, questa verra' eseguita una
564                                   // volta e poi rimarra' ferma fino alla prossima lettura valida
565                     break;
566
567               case GasIn:                 // gas
568                     GasVal = ADRESH;      // solo il byte alto della lettura e' usato
569                     ADCON0 &= MaskClear; // clear bit relativi alla porta
570                     ADCON0 |= EyeRf;      // ricomincia il giro
571                     FlagGas = 1;  // la misura e' stata effettuata, si abilita la routine gas
572                                   // alla ricerca del gas, questa verra' eseguita una
573                                   // volta e poi rimarra' ferma fino alla prossima lettura valida
574                     FlagSound = 1;// con la stessa temporizzazione abilita anche la routine Sound
575                     break;
576
577               default:
578                                   // a default imposta EyeR
579                     ADCON0 &= MaskClear; // clear bit relativi alla porta
580                     ADCON0 |= EyeRf;      // si imposta la successiva porta da leggere
581
582                     break;
583             }
584             TimerAD = 0;      // resetta il timer
585             FlagAD = 0;       // e l'indicatore di stato
586                               // al prossimo giro del main  riaspettera'
587       }                       // N mS e poi ricomincera' dallo step 1
588
589   }
590
591   } //  ReadAD
592   /*****************************************************************************/
593
594   /*Bumpers  ******************************************************************
595   Controllo dei sensori di collisione
596   */
597
598   void Bumpers(void)
599   {
600
```

```
601  if (FlagBumpers) // routine disattiva
602  {
603     /* disabilita i sensori per 1 Sec (debouncer), il timer e' incrementato
604        dall'interrupt ogni mSec
605     */
606     if (TimerBumpers >= 1000)
607     {
608        FlagBumpers=0; // riabilita la routine
609     }
610  }
611  else
612  {
613      if (BumperDx == 0) // controlla bumper destro
614      {
615        FlagBumpers = 1;  // disattiva bumpers
616        TimerBumpers = 0; // per 1 Sec
617        Space2RunFlag = 0; // reset di qualsiasi routine di movimento
618
619        // evita l'ostacolo a SX
620        PathSeq [0] = 2;   // indietro n cm
621        PathSeq [1] = 13;  // gira 45°a SX
622        PathSeq [2] = 255; // cammina avanti a velocita' costante
623        PathSeq [3] = 0;   // fine sequenza
624        PathSeqPointer = 0;// inizializza sequenza
625      }
626
627      if (BumperSx == 0 && !FlagBumpers) // bumper sinistro ma non bumper destro
628      {
629        FlagBumpers = 1;  // disattiva bumpers
630        TimerBumpers = 0; // per 1Sec
631        Space2RunFlag = 0; // reset di qualsiasi routine di movimento
632
633        // evita l'ostacolo a DX
634        PathSeq [0] = 2;   // indietro n cm
635        PathSeq [1] = 12;  // gira 45°a DX
636        PathSeq [2] = 255; // cammina avanti a velocita' costante
637        PathSeq [3] = 0;   // fine sequenza
638        PathSeqPointer = 0;// inizializza sequenza
639      }
640  }
641  } // Bumpers
642  /*****************************************************************************/
643
```