```
1   /*
2   contiene tutte le routine per la gestione del movimento
3   */
4
5
6   /*Turn ********************************************************************************
7   routine per girare a destra o a sinistra di un angolo desiderato
8   la rotazione e' intorno al centro del bot senza avanzamento quindi le ruote girano una
9   in senso opposto all'altra, alla stessa velocita'.
10  Qui calcola e imposta i valori per girare dell'angolo desiderato.
11  L'interasse del bot e' di 140mm, la circonferenza lungo la quale girano le ruote e' quindi
12  di PI * 140 = 439,823mm. Per ruotare di 180° ogni ruota deve fare meta' circonferenza:
13  219.911micron (millesimi di millimetro), dividendo per 180 sappiamo che un grado
14  corrisponde a 1.222micron di spostamento di ogni ruota ((D * PI)/2/180).
15  Si considera:
16  0° la direzione di avanzamento
17  da +1 a +180° la rotazione in senso orario
18  da -1 a -180° la rotazione in senso antiorario
19
20  operazioni:
21  imposta: Space2RunXstart
22  calcola e imposta: Space2RunX
23  imposta  velocita': DesSpeedX
24  imposta: Space2RunFlag a 1
25  */
26
27  void Turn(long Gradi)
28  {
29  // fixed point base 2, la costante e' moltiplicata per 128, il risultato finale e' poi
30  // shiftato di 7 bit (diviso 128) per riportare il valore all'intero.
31
32  // Arco di circonferenza corrispondente ad un grado, in impulsi
33  #define unGradoR  1220/stepR*128 // ruota destra
34  #define unGradoL  1220/stepL*128 // ruota sinistra
35
36  // #define corrGradi  3
37  // se e' presente un errore sistematico assoluto, che quindi influisce in una percentuale
38  // maggiore sui valori di angolo piccoli, inserire qui il parametro di correzione
39
40  Space2RunRstart = EncoderRcount; // punto di partenza
41  Space2RunLstart = EncoderLcount;
42
43  if (Gradi > 0)
44  {
45     #ifdef corrGradi
46     if (Gradi > 20) Gradi = Gradi - corrGradi; // non  corregge per angoli molto piccoli
47     #endif
48
49     DesSpeedR = manSpeedRew; // rotazione oraria, motore destro gira indietro
50     DesSpeedL = manSpeedFwd; // ruota sinistra gira in avanti
```

```
51
52      Space2RunR = DivLongS128((unGradoR * Gradi));// spazio (in impulsi encoder) da percorrere
53      Space2RunL = DivLongS128((unGradoL * Gradi));
54    }
55    else
56    {
57      #ifdef corrGradi
58      if (Gradi < -20) Gradi = Gradi + corrGradi;// non  corregge per angoli molto piccoli
59      #endif
60
61      DesSpeedR = manSpeedFwd; // rotazione antioraria, motore destro gira avanti
62      DesSpeedL = manSpeedRew; // ruota sinistra gira indietro
63
64      Space2RunR = DivLongS128((unGradoR * Gradi));// spazio (in impulsi encoder) da percorrere
65      Space2RunL = DivLongS128((unGradoL * Gradi));
66    }
67    Space2RunFlag = 1; // inizia il controllo dello spazio da percorrere
68    } // Turn
69    /*****************************************************************************/
70
71    /*Walk ************************************************************************
72    Calcola e inizializza le variabili per Space2Run, per camminare dello spazio desiderato
73    operazioni:
74    imposta: Space2RunXstart
75    Space2RunX e' impostata dalla routine chiamante
76    imposta  velocita': DesSpeedX
77    imposta: Space2RunFlag a 1
78    */
79
80    void Walk(long Space)
81    {
82
83    Space2RunRstart = EncoderRcount; // punto di partenza
84    Space2RunLstart = EncoderLcount;
85
86    Space2RunR = Space / stepR; // da micron a conteggio encoder
87    Space2RunL = Space / stepL;
88
89    if (Space > 0) // avanti
90    {
91      DesSpeedR = manSpeedFwd; // durante le manovre va piu' piano
92      DesSpeedL = manSpeedFwd;
93    }
94    else // indietro
95    {
96      DesSpeedR = manSpeedRew; // durante le manovre va piu' piano
97      DesSpeedL = manSpeedRew;
98    }
99
100   Space2RunFlag = 1; // inizia il controllo dello spazio da percorrere
```

```
101
102 } // Walk
103 /*****************************************************************************/
104
105 /*Space2Run ******************************************************************
106 controlla continuamente lo spazio percorso per pilotare i motori fino al raggiungimento
107 della posizione desiderata
108
109 operazioni:
110 controlla se spazio percorso >= spazio desiderato:
111 if yes, ferma motore corrispondente
112 se entrambi motori fermi: Space2RunFlag a 0 (si autodisabilita)
113
114 */
115
116 void Space2Run (void)
117 {
118 /* qualche routine ha abilitato il controllo dello spazio percorso,
119    ha impostato la velocita' (DesSpeedX), lo spazio da percorrere (Space2RunX)
120    e la posizione di partenza (Space2RunXstart).
121    Questa routine sara' eseguita continuamente fino al raggiungimento della
122    posizione desiderata, dopodiche' fermera' i motori e disabilitera' il flag
123 */
124     if (abs(EncoderRcount - Space2RunRstart) >= abs(Space2RunR))
125     {
126       DesSpeedR = 0;
127     }
128
129     if (abs(EncoderLcount - Space2RunLstart) >= abs(Space2RunL))
130     {
131       DesSpeedL = 0;
132     }
133
134     if ((DesSpeedR == 0) && (DesSpeedL == 0))
135     {
136       // raggiunto il punto desiderato si ferma per 100 mSec
137       TimerStop = 100; // carica il contatore
138       FlagStop = 1;  // abilita routine Stop
139       Space2RunFlag = 0; // si autodisabilita
140     }
141 } // Space2Run
142 /*****************************************************************************/
143
144 /*Stop ***********************************************************************
145    Ferma i motori per X millisecondi (fino a 30 Sec)
146 */
147 void Stop(void)
148 {
149 DesSpeedR = 0;   // Motori fermi
150 DesSpeedL = 0;
```

```
151
152  if (TimerStop <= 0) //  il clock decrementa il contatore, quindi il valore caricato
153                      //  inizialmente in TimerStop e' = alla pausa in mSec
154     {
155        LedVerdeOFF;   // spegne i led accesi al raggiungimento di un obietivo
156        LedRossoOFF;    // i led rimano accesi mentre i motori sono fermi
157        LedGialloOFF;
158
159        FlagStop = 0; // disabilita la routine Stop, riabilita la routine Path
160     }
161
162  } // Stop
163  /*****************************************************************************/
164
165  /*Path ***********************************************************************
166     viene chiamata solo se (Space2RunFlag==0  & PathSeq[PathSeqPointer] != 0)
167     se Space2RunFlag = 1, sono ancora in esecuzione routine movimento e quindi non può
168     passare allo step successivo
169     se PathSeq[PathSeqPointer]= 0 la sequenza e' terminata e rilascia il controllo
170
171     routine che deve usare Path
172     -Imposta DesSpeedX = 0; // ferma motori
173     -imposta PathSeq con passi da fare in ordine (fine seq = 0)
174     -imposta PathSeqPointer = 0; // inizializza sequenza passi
175     -imposta Space2RunFlag = 0; // reset di qualsiasi routine di movimento
176
177     Se deve avanzare a velocita' costante si imposta semplicemente la velocita'
178
179  */
180
181   void Path (void)
182  {
183        switch (PathSeq[PathSeqPointer])
184        {
185           case 1:
186                       Walk(1000000); // 1.000.000 micron = 1 metro avanti
187                       break;
188           case 2:
189                       Walk(-30000); // -30.000 micron = 3 cm indietro
190                       break;
191           case 3:
192                       Walk(-100); // -100 micron indietro,per azzerare i contatori
193                       break;
194           case 8:
195                       Turn(10); // gira n gradi a Dx
196                       DeadCorner++; // contatore per routine angolo morto
197                       break;
198           case 9:
199                       Turn(-10);// gira n gradi a Sx
200                       DeadCorner++; // contatore per routine angolo morto
```

```
201                 break;
202         case 12:
203                 Turn(45);
204                 DeadCorner++; // contatore per routine angolo morto
205                 break;
206         case 13:
207                 Turn(-45);
208                 DeadCorner++; // contatore per routine angolo morto
209                 break;
210         case 14:
211                 Turn(90);
212                 DeadCorner++; // contatore per routine angolo morto
213                 break;
214         case 15:
215                 Turn(-90);
216                 DeadCorner++; // contatore per routine angolo morto
217                 break;
218         case 18:
219                 Turn(180);
220                 break;
221         case 19:
222                 TimerStop = 100; // pausa per  100 mSec
223                 FlagStop = 1;  // abilita routine Stop motori
224                 break;
225         case 20:
226                 TimerStop = 10000; // pausa per  10 Sec
227                 FlagStop = 1;  // abilita routine Stop motori
228                 break;
229         case 21:
230                 TimerStop = 1000; // pausa per  1 Sec
231                 FlagStop = 1;  // abilita routine Stop motori
232                 break;
233
234         case 22:
235                 TimerStop = 5000; // pausa per  5 Sec
236                 FlagStop = 1;  // abilita routine Stop motori
237                 break;
238
239         case 23:   // usato solo per la calibrazione della bussola
240                 FlagCmpReg15=1; // calibra per ogni punto cardinale
241                 TimerStop = 5000; // pausa per  5 Sec
242                 FlagStop = 1;  // abilita routine Stop motori
243                 break;
244
245         case 50:
246                 // gira in un verso o nell'altro pseudo-randomicamente
247                 // diminuisce la probabilita' di ripetere sempre gli stessi percorsi
248                 if (RandomBit)
249                  {
250                     Turn(90);
```

```
251                         }
252                       else
253                         {
254                           Turn(-90);
255                         }
256                       break;
257           case 51:
258                       // gira in un verso o nell'altro pseudo-randomicamente
259                       // diminuisce la probabilita' di ripetere sempre gli stessi percorsi
260                       if (RandomBit)
261                         {
262                           Turn(135);
263                         }
264                       else
265                         {
266                           Turn(-135);
267                         }
268                       break;
269           case 100:
270                       DesSpeedR=0; // velocita' 0 cm/s
271                       DesSpeedL=0; // sfrutta la funzione "brake" del metodo LAP
272                       break;
273           case 200:
274                       DesSpeedR=lowSpeed; // velocita' ridotta
275                       DesSpeedL=lowSpeed;
276                       break;
277           case 254:
278                       FlagTargetLight = 1;  // riabilita routine sensori luce fino a fine manovra
279                       break;
280
281           default:
282                       DesSpeedR=constSpeed; // a default cammina a velocita' costante
283                       DesSpeedL=constSpeed; // a default cammina a velocita' costante
284                       break;
285         }
286       PathSeqPointer++;  // predispone al prossimo passo
287 } // Path
288 /*****************************************************************************/
289
290 /*MotSpeed ********************************************************************
291 Controllo velocita' motori PID (Proportional + Integral + Derivative)
292 il ponte ad H e' usato in modalita' "Locked Anti Phase" (LAP):
293 CCPR2L collegato al motore R
294 CCPR1L collegato al motore L
295 */
296 void MotSpeed (void)
297 {
298 /*
299    PWM = 256 -> motore fermo
300    PWM = 512 -> velocita' massima FWD (circa  690 mm/sec)
```

```
301     PWM =   0 -> velocita' massima REW (circa -690 mm/sec)
302  per riportare i conti effettuati sulla velocita' in valori di PWM:
303  const divisore =((690-0)/(512-256)); rapporto tra range velocita' e range PWM = 2,70.
304  Per mantenere la precisione di due decimali nei calcoli intermedi senza usare la virgola mobile
305  la costante e' moltiplicata per 256, il risultato finale e' poi diviso per 256 (fixed point base 2)
306  Per velocizzare ancora di piu' i calcoli, invece che fare PWM / divisore
307  e' meglio fare PWM * (1/divisore) -> la moltiplicazione e' piu' veloce della divisione.
308  Quindi: 1/2,70*256 = 94,81 approssimato a 95
309  */
310  #define divisore  95
311
312  /*
313  Per ottenere 19KHz di PWM con 40MHz di clock il valore del prescaler e' 4 e il valore caricato
314  in PR2 e' 127. In questo modo si possono usare solamente 9 bit dei 10 disponibili.
315  Il valore di PWM al 100% corrisponde quindi con 512 e il 50% (motori fermi) a 256.
316  */
317
318  #define biasFwd  256 // (MinPWMFwd-(MinSpeed/divisore)), in pratica sono lo 0 del PWM FWD e REW, in questo caso
319  #define biasRew  256 // (MinPWMRew+(Minspeed/divisore)), non c'e' dead-band, la regolazione e' continua da 0 a 1023
320
321  #define intUpperLimit  10230  //limite superiore della correzione integrale
322  #define intLowerLimit  -10230 //limite inferiore della correzione integrale
323  #define devUpperLimit  10230  //limite superiore della correzione derivativa
324  #define devLowerLimit  -10230 //limite inferiore della correzione derivativa
325  #define speedUpperLimit  689 //limite superiore velocita'
326  #define speedLowerLimit -689 //limite inferiore velocita'
327
328  long PWMR;   // risultato del calcolo del PID, ancora e' in mm/sec
329  long PWML;
330
331  int ErroreR = 0;   //errore attuale
332  int ErroreL = 0;
333
334  int DevR   = 0;   //componente derivativa attuale
335  int DevL   = 0;   //componente derivativa attuale
336
337  /* questa routine e' eseguita solo su abilitazione della ISR, quest'ultima calcola la velocita' ogni x mSec
338     e solo dopo averla calcolata ne permette la correzione
339     il calcolo PID e' quindi eseguito piu' o meno con la stessa periodicita'
340  */
341
342     PidTick = 0;   // resetta il flag per il prossimo giro
343     ErroreR = (DesSpeedR - SpeedR);   //errore attuale = velocita' desiderata - velocita' attuale
344     ErroreL = (DesSpeedL - SpeedL);   //
345
346     /* Fattore I
347        esegue la sommatoria algebrica dell'errore, cioe' lo integra.
348        L'integrale dell'errore di velocita' e' lo spazio perso, o guadagnato, rispetto a quello voluto.
349        In questo modo anche se una ruota rallenta a causa di un ostacolo perdendo terreno rispetto all'altra
350        questo viene riguadagnato (entro certi limiti)
```

```
351     */
352     IntR =IntR + (ErroreR * ki);
353     if (IntR > intUpperLimit)   // la quantita' assoluta di correzione e' limitata per non saturare i contatori
354     {
355        IntR = intUpperLimit;
356     }
357     if (IntR < intLowerLimit)
358     {
359        IntR = intLowerLimit;
360     }
361
362     IntL = IntL + (ErroreL * ki);
363     if (IntL > intUpperLimit)
364     {
365        IntL = intUpperLimit;
366     }
367     if (IntL < intLowerLimit)
368     {
369        IntL = intLowerLimit;
370     }
371
372     /*
373     Fattore D
374     calcola la variazione dell'errore rispetto alla misura precedente,
375     essendo l'intervallo di tempo di misura costante, corrisponde con la derivata dell'errore di velocita'
376     e quindi con l'accellerazione
377     */
378     DevR = kd * (ErroreR - ErroreRprev);
379     ErroreRprev = ErroreR;
380     if (DevR > devUpperLimit)   // la quantita' assoluta di correzione e' limitata per non saturare i contatori
381     {
382        DevR = devUpperLimit;
383     }
384     if (DevR < devLowerLimit)
385     {
386        DevR = devLowerLimit;
387     }
388
389     DevL = kd * (ErroreL - ErroreLprev);
390     ErroreLprev = ErroreL;
391     if (DevL > devUpperLimit)
392     {
393        DevL = devUpperLimit;
394     }
395     if (DevL < devLowerLimit)
396     {
397        DevL = devLowerLimit;
398     }
399
400     /* Il risultato e' diviso 16 per sfruttare il decimale senza usare variabili float,
```

```
401         le costanti erano infatti moltiplicate per 16, in questo modo si usa la precisione
402         del decimale nei calcoli intermedi pur avendo un intero come risultato finale
403      */
404
405      PWMR = DivLongS16((long)(ErroreR * kp + IntR + DevR));    // P + I + D
406      PWML = DivLongS16((long)(ErroreL * kp + IntL + DevL));    //
407
408      /* converte la velocita' da mm/sec (signed int moltiplicato 256) a valore da impostare nei registri CCPRxL (char)
409         PWM da -690 a 690 -> CCPRxL:CCPxCON<5:4> da 0 a 512
410      */
411      if (PWMR > 0)    // Motore Destro in avanti
412      {
413         if (PWMR > speedUpperLimit)    // anche qui si limita per non sforare le variabili
414         {
415            PWMR = speedUpperLimit;
416         }
417         PWMR = DivLongS256(PWMR * divisore)  + biasFwd; //  diviso 256 per riportare alla giusta dimensione
418         SetDCPWM2(PWMR);
419      }
420      else   // indietro
421      {
422         if (PWMR < speedLowerLimit)
423         {
424            PWMR = speedLowerLimit;
425         }
426         PWMR = biasRew + DivLongS256(PWMR * divisore);
427         SetDCPWM2(PWMR);
428      }
429
430      if (PWML > 0)    // Motore Sinistro in avanti
431      {
432         if (PWML > speedUpperLimit)
433         {
434            PWML = speedUpperLimit;
435         }
436         PWML = DivLongS256(PWML * divisore) + biasFwd;
437         SetDCPWM1(PWML);
438      }
439      else   // indietro
440      {
441         if (PWML < speedLowerLimit)
442         {
443            PWML = speedLowerLimit;
444         }
445         PWML = biasRew + DivLongS256(PWML * divisore);
446         SetDCPWM1(PWML);
447      }
448
449
450   } //  MotSpeed
```

```
451  /*****************************************************************************/
452
```