

```

1  * This program is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General
2  * Public License as published by the Free Software Foundation, either version 3 of the License, or any later version.
3  * This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied
4  * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more
5  * details. You should have received a copy of the GNU Lesser General Public License along with this program.
6  * If not, see <http://www.gnu.org/licenses/> or <http://blog.robotica.eng.br/gnu-lesser-general-public-license/>.
7  *
8  *
9  * © Copyright 2021 Filippo Pardini - filippo@robotica.eng.br - http://blog.robotica.eng.br/
10 * *****
11 * Esta biblioteca implementa (para o Arduino) as funções para operação do "MD25 - 12v 2.8A dual H-bridge motor driver"
12 * da Robot Electronics, no modo I2C:
13 * http://www.robot-electronics.co.uk/htm/md25tech.htm
14 * http://www.robot-electronics.co.uk/htm/md25i2c.htm
15 *
16 * Atenção: a interface I2C do MD25 é de 5Vdc, se for utilizar com microcontrolador de I2C 3,3Vdc, é necessário utilizar*
17 * um conversor de nível. Em ambos os casos é aconselhável utilizar resistores pull-up de 1,8K no SDA e SCL.
18 *
19 * *****/
20
21 #ifndef MD25oppip_h
22 #define MD25oppip_h
23
24 #include "Wire.h" // I2C
25
26 // ***** Registros do MD25 *****
27
28 byte registro_speed1 = 0x00; // Registro de velocidade do motor 1 - Sped1
29 // Depending on what mode you are in, this register can affect the speed of one motor or both motors. If you are in
30 // mode 0 or 1 it will set the speed and direction of motor 1. The larger the number written to this register, the more
31 // power is applied to the motor. A mode of 2 or 3 will control the speed and direction of both motors
32 // (subject to effect of turn register).
33
34 // -----
35
36 byte registro_speed2 = 0x01; // Registro de velocidade do motor 2 - Speed2/Turn
37 // When in mode 0 or 1 this register operates the speed and direction of motor 2. When in mode 2 or 3 Speed2 becomes
38 // a Turn register, and any value in this register is combined with the contents of Speed1 to steer the device.
39
40 // -----
41
42 byte registro_enc1a = 0x02; // Registro do encoder do motor 1 - Enc1a - 1º byte
43 byte registro_enc1b = 0x03; // Registro do encoder do motor 1 - Enc1b - 2º byte
44 byte registro_enc1c = 0x04; // Registro do encoder do motor 1 - Enc1c - 3º byte
45 byte registro_enc1d = 0x05; // Registro do encoder do motor 1 - Enc1d - 4º byte
46 byte registro_enc2a = 0x06; // Registro do encoder do motor 2 - Enc2a - 1º byte
47 byte registro_enc2b = 0x07; // Registro do encoder do motor 1 - Enc2b - 2º byte
48 byte registro_enc2c = 0x08; // Registro do encoder do motor 1 - Enc2c - 3º byte
49 byte registro_enc2d = 0x09; // Registro do encoder do motor 1 - Enc2d - 4º byte
50 // Each motor has its encoder count stored in an array of four bytes, together the bytes form a signed 32 bit number,
51 // the encoder count is captured on a read of the highest byte (registers 2, 6) and the subsequent lower bytes will be
52 // held until another read of the highest byte takes place. The count is stored with the highest byte in the lowest
53 // numbered register. The registers can be zeroed at any time by writing 32 (0x20) to the command register.
54
55 // -----
56
57 byte registro_bateria_volts = 0x0A; // Registro da voltagem da bateria
58 // A reading of the voltage of the connected battery is available in this register. It reads as 10 times the voltage
59 // (121 for 12.1v).
60
61 // -----
62
63 byte registro_motor1_corrente = 0x0B; // Registro da corrente do motor 1
64 byte registro_motor2_corrente = 0x0C; // Registro da corrente do motor 2
65 // A guide reading of the average current through the motor is available in this register. It reads approx ten times
66 // the number of Amps (25 at 2.5A).
67
68 // -----
69
70 byte registro_software_versao = 0x0D; // Registro da versão do software
71 // This register contains the revision number of the software in the modules PIC16F873 controller.
72
73 // -----
74
75 byte registro_taxa_aceleracao = 0x0E; // Registro da taxa de aceleração
76 // If you require a controlled acceleration period for the attached motors to reach there ultimate speed, the MD25 has
77 // a register to provide this. It works by using a value into the acceleration register and incrementing the power
78 // by that value. Changing between the current speed of the motors and the new speed (from speed 1 and 2 registers).
79 // So if the motors were traveling at full speed in the forward direction (255) and were instructed to move at full
80 // speed in reverse (0), there would be 255 steps with an acceleration register value of 1, but 128 for a value of 2.
81 // The default acceleration value is 5, meaning the speed is changed from full forward to full reverse in 1.25 seconds.
82 // The register will accept values of 1 up to 10 which equates to a period of only 0.65 seconds to travel from full
83 // speed in one direction to full speed in the opposite direction.
84
85 // -----
86
87 byte registro_modos = 0x0F; // Registro do modo de operação

```

```

88 // The mode register selects which mode of operation and I2C data input type the user requires. The options being:
89 // 0 - (default setting) If a value of 0 is written to the mode register then the meaning of the speed registers is
90 // literal speeds in the range of 0 (full reverse) 128 (stop) 255 (full forward).
91 // 1 - Mode 1 is similar to mode 0, except that the speed registers are interpreted as signed values.
92 // The meaning of the speed registers is literal speeds in the range of -128 (full reverse) 0 (Stop)
93 // 127 (full forward).
94 // 2 - Writing a value of 2 to the mode register will make Speed1 control both motors speed, and Speed2 becomes
95 // the turn value. Data is in the range of 0 (full reverse) 128 (stop) 255 (full forward).
96 // 3 - Mode 3 is similar to mode 2, except that the speed registers are interpreted as signed values. Data is in the
97 // range of -128 (full reverse) 0 (stop) 127 (full forward).
98 // *****
99 // ESTA BIBLIOTECA USA SOMENTE O MODO 1 *****
100 // *****
101 //
102
103 byte registro_comando = 0x10; // Registro de comando
104 // 0x20 - Resets the encoder registers to zero
105 // 0x30 - Disables automatic speed regulation
106 // 0x31 - Enables automatic speed regulation (default)
107 // 0x32 - Disables 2 second timeout of motors (version 2 onwards only)
108 // 0x33 - Enables 2 second timeout of motors when no I2C comms (default) (version 2 onwards only)
109 // 0xA0 - 1st in sequence to change I2C address
110 // 0xAA - 2nd in sequence to change I2C address
111 // 0xA5 - 3rd in sequence to change I2C address
112 // To change the I2C address of the MD25 by writing a new address you must have only one module on the bus. Write the 3
113 // sequence commands in the correct order followed by the address. Example; to change the address of an MD25 currently
114 // at 0xB0 (the default shipped address) to 0xB4, write the following to address 0xB0; (0xA0, 0xAA, 0xA5, 0xB4 ).
115 // These commands must be sent in the correct sequence to change the I2C address, additionally, no other command may be
116 // issued in the middle of the sequence. The sequence must be sent to the command register at location 16, which means
117 // 4 separate write transactions on the I2C bus. Because of the way the MD25 works internally, there MUST be a delay of
118 // at least 5mS between the writing of each of these 4 transactions. When done, you should label the MD25 with its
119 // address, however if you do forget, just power it up without sending any commands. The MD25 will flash its address
120 // out on the green communication LED. One long flash followed by a number of shorter flashes indicating its address.
121 // Any command sent to the MD25 during this period will still be received and writing new speeds or a write to the
122 // command register will terminate the flashing.
123 // *****
124 // *****
125
126 #define stop_motor1 define_velocidade_motor1(stop) // Stop motor1
127 #define stop_motor2 define_velocidade_motor2(stop) // Stop motor1
128 #define stop_motores define_velocidade_motor1(stop);define_velocidade_motor2(stop) // Stop ambos os motores
129 #define encoder1 le_encoder(1) // Posição do encoder do motor 1
130 #define encoder2 le_encoder(2) // Posição do encoder do motor 2
131
132 byte endereco_i2c; // Endereço I2C do MD25
133 // 0x58,0x59,0x5A,0x5B,0x5C,0x5D,0x5E,0x5F
134 byte stop = 0; // Modo 1
135 byte modo = 1; // Modo de operação - modo 1
136 float perimetro_roda; // Perimetro das rodas em milímetros
137 int eixo; // Tamanho do eixo entre rodas em milímetros
138 bool const frente = true;
139 bool const re = false;
140 bool const esquerda = true;
141 bool const direita = false;
142 bool direcao; // Direção do movimento. true => frente; false => re
143
144 union // União para equivalência entre os 4 bytes do encoder
145 { // e um longo com sinal
146     byte encoder_bytes[4];
147     signed long encoder;
148 } valor;
149
150 //
151
152 void muda_endereco_i2c(byte novo_addr) // Muda endereço I2c do MD25
153 {
154     Wire.beginTransmission(endereco_i2c);
155     Wire.write(registro_comando);
156     Wire.write(0xA0);
157     delay(6);
158     Wire.write(0xAA);
159     delay(6);
160     Wire.write(0xA5);
161     delay(6);
162     Wire.write(novo_addr);
163     delay(6);
164     Wire.endTransmission();
165 }
166
167 //
168
169 void define_mod0_1(void) // Define o modo de operação 1
170 {
171     Wire.beginTransmission(endereco_i2c);
172     Wire.write(registro_mod0);
173     Wire.write(1);
174     Wire.endTransmission();

```

```

176 }
177 //
178
179 byte le_mod() // retorna o valor do modo
180 {
181     Wire.beginTransmission(endereco_i2c);
182     Wire.write(registro_mod);
183     Wire.endTransmission();
184
185     Wire.requestFrom(endereco_i2c,1);
186     while(Wire.available() < 1);
187     byte mod = Wire.read();
188     return mod;
189 }
190
191 //
192
193 void encoders_reset() // Zera o valor dos encoders
194 {
195     Wire.beginTransmission(endereco_i2c);
196     Wire.write(registro_comando);
197     Wire.write(0x20);
198     Wire.endTransmission();
199 }
200
201 //
202
203 void desabilita_regulacao_velocidade() // Desabilita regulaco automtica da velocidade
204 {
205     Wire.beginTransmission(endereco_i2c);
206     Wire.write(registro_comando);
207     Wire.write(0x30);
208     Wire.endTransmission();
209 }
210
211 //
212
213 void habilita_regulacao_velocidade() // Habilita regulaco automtica da velocidade (default)
214 {
215     Wire.beginTransmission(endereco_i2c);
216     Wire.write(registro_comando);
217     Wire.write(0x31);
218     Wire.endTransmission();
219 }
220
221 //
222
223 void desabilita_timeout() // Desabilita 2seg de timeout
224 {
225     Wire.beginTransmission(endereco_i2c);
226     Wire.write(registro_comando);
227     Wire.write(0x32);
228     Wire.endTransmission();
229 }
230
231 //
232
233 void habilita_timeout() // Habilita 2seg de timeout (default)
234 {
235     Wire.beginTransmission(endereco_i2c);
236     Wire.write(registro_comando);
237     Wire.write(0x33);
238     Wire.endTransmission();
239 }
240
241 //
242
243 void define_velocidade_motor1(signed char vel) // Aciona o motor1  velocidade vel (-128/0/127)
244 {
245     Wire.beginTransmission(endereco_i2c);
246     Wire.write(registro_speed1);
247     Wire.write(vel);
248     Wire.endTransmission();
249 }
250
251 //
252
253 signed char le_velocidade_motor1() // Retorna a velocidade do motor1
254 {
255     Wire.beginTransmission(endereco_i2c);
256     Wire.write(registro_speed1);
257     Wire.endTransmission();
258
259     Wire.requestFrom(endereco_i2c,1);
260     while(Wire.available() < 1);
261     signed char speed = Wire.read();

```

```

263 }
264 }
265 //
266
267 void define_velocidade_motor2(signed char vel) // Aciona o motor2 à velocidade vel (-128/0/127)
268 {
269     Wire.beginTransmission(endereco_i2c);
270     Wire.write(registro_speed2);
271     Wire.write(vel);
272     Wire.endTransmission();
273 }
274 //
275
276 signed char le_velocidade_motor2() // Retorna a velocidade do motor2
277 {
278     Wire.beginTransmission(endereco_i2c);
279     Wire.write(registro_speed2);
280     Wire.endTransmission();
281
282     Wire.requestFrom(endereco_i2c,1);
283     while(Wire.available() < 1);
284     signed char speed = Wire.read();
285     return speed;
286 }
287 //
288
289 void le_encoder(byte motor) // Retorna o valor do encoder do motor 1 ou 2
290 {
291     Wire.beginTransmission(endereco_i2c);
292     if (motor == 1) Wire.write(registro_encla);
293     else Wire.write(registro_enc2a);
294     Wire.endTransmission();
295
296     Wire.requestFrom(endereco_i2c,4);
297     while(Wire.available() < 4);
298     valor.encoder_bytes[3] = Wire.read();
299     valor.encoder_bytes[2] = Wire.read();
300     valor.encoder_bytes[1] = Wire.read();
301     valor.encoder_bytes[0] = Wire.read();
302     if (motor == 1) valor.encoder = -valor.encoder; // Acerto de sinal, depende de como estão montados
303     // os motores no chassiss
304     if ((motor == 1) && (valor.encoder > 0)) direcao = frente;
305     else direcao = re;
306 }
307 //
308
309 float le_voltagem_bateria() // Retorna a voltagem da bateria
310 {
311     Wire.beginTransmission(endereco_i2c);
312     Wire.write(registro_bateria_volts);
313     Wire.endTransmission();
314
315     Wire.requestFrom(endereco_i2c,1);
316     while(Wire.available() < 1);
317     float volts = (float)Wire.read()/10.0;
318     return volts;
319 }
320 //
321
322 float le_corrente_motor1() // Retorna a corrente do motor 1
323 {
324     Wire.beginTransmission(endereco_i2c);
325     Wire.write(registro_motor1_corrente);
326     Wire.endTransmission();
327
328     Wire.requestFrom(endereco_i2c,1);
329     while(Wire.available() < 1);
330     float current = (float)Wire.read()/10.0;
331     return current;
332 }
333 //
334
335 float le_corrente_motor2() // Retorna a corrente do motor 2
336 {
337     Wire.beginTransmission(endereco_i2c);
338     Wire.write(registro_motor2_corrente);
339     Wire.endTransmission();
340
341     Wire.requestFrom(endereco_i2c,1);
342     while(Wire.available() < 1);
343     float current = (float)Wire.read()/10.0;
344 }

```

```

350 }
351 }
352 //
353
354 byte le-versao_software() // Retorna a versão do software
355 {
356   Wire.beginTransmission(endereco_i2c);
357   Wire.write(registro_software-versao);
358   Wire.endTransmission();
359
360   Wire.requestFrom(endereco_i2c,1);
361   while(Wire.available() < 1);
362   byte software = Wire.read();
363   return software;
364 }
365
366 //
367
368 void define_taxa_aceleracao(byte taxa) // Define taxa de aceleração (default 5 - 1,275seg)
369 {
370   Wire.beginTransmission(endereco_i2c);
371   Wire.write(registro_taxa_aceleracao);
372   Wire.write(taxa);
373   Wire.endTransmission();
374 }
375
376 //
377
378 byte le_taxa_aceleracao() // Retorna a taxa de aceleracao
379 {
380   Wire.beginTransmission(endereco_i2c);
381   Wire.write(registro_taxa_aceleracao);
382   Wire.endTransmission();
383
384   Wire.requestFrom(endereco_i2c,1);
385   while(Wire.available() < 1);
386   byte rate = Wire.read();
387   return rate;
388 }
389
390 //
391
392 bool para_frente(signed char vel,byte ace) // Movimento para frente
393 {
394   // Só para modo 1 => -128/0/127
395   // vel => velocidade (1 => 127)
396   // ace => taxa de aceleração (1 a 10)
397
398   direcao = frente;
399   if ((vel < 1) || (vel > 127) || (ace < 1) || (ace > 10)) return false;
400   define_taxa_aceleracao(ace);
401   define_velocidade_motor2(vel);
402   define_velocidade_motor1(vel);
403   //encoder1;
404   //Serial.print("\nEncoder ");
405   //Serial.print(valor.encoder);
406 }
407
408 //
409
410 bool para_tras(signed char vel,byte ace) // Movimento para tras (ré)
411 {
412   // Só para modo 1 => -128/0/127
413   // vel => velocidade (-1 => -128)
414   // ace => taxa de aceleração (1 a 10)
415
416   direcao = re;
417   if ((vel > -1) || (ace < 1) || (ace > 10)) return false;
418   define_taxa_aceleracao(ace);
419   define_velocidade_motor2(vel);
420   define_velocidade_motor1(vel);
421   //encoder1;
422   //Serial.print("\nEncoder ");
423   //Serial.print(valor.encoder);
424 }
425
426 //
427
428 bool para_frente_distancia(long dis,signed char vel,byte ace) // Movimento para frente por uma distância
429 {
430   // Só para modo 1 => -128/0/127
431   // dis => distância a percorrer em milímetros
432   // vel => velocidade (1 => 127)
433   // ace => taxa de aceleração (1 a 10)
434
435   direcao = frente;

```

```

437 if ((vel < 1) || (vel > 127) || (ace < 1) || (ace > 10)) return false;
438 encoder1;
439 signed long encod = valor.encoder + round(float(dis) / perimetro_roda * 360.0);
440 define_taxa_aceleracao(ace);
441 //Serial.print("\n\nPARA FRENTE");
442 do
443 {
444   define_velocidade_motor2(vel);
445   define_velocidade_motor1(vel);
446   encoder1;
447   //Serial.print("\nEncoder ");
448   //Serial.print(valor.encoder);
449 }while(valor.encoder < encod);
450 stop_motores;
451 encoders_reset();
452 //Serial.print("\n_____");
453 return true;
454 }
455 // _____
456
457 bool para_tras_distancia(long dis,signed char vel,byte ace) // Movimento para tras por uma distância
458 {
459 // Só para modo 1 => -128/0/127
460 // dis => distância a percorrer em milímetros
461 // vel => velocidade (-1 => -128)
462 // ace => taxa de aceleração (1 a 10)
463
464 direcao = re;
465 if ((vel > -1) || (ace < 1) || (ace > 10)) return false;
466 encoder1;
467 signed long encod = valor.encoder - (round(float(dis) / perimetro_roda * 360.0));
468 define_taxa_aceleracao(ace);
469 //Serial.print("\n\nPARA TRAS");
470 do
471 {
472   define_velocidade_motor2(vel);
473   define_velocidade_motor1(vel);
474   encoder1;
475   //Serial.print("\nEncoder ");
476   //Serial.print(valor.encoder);
477 }while(valor.encoder > encod);
478 stop_motores;
479 encoders_reset();
480 //Serial.print("\n_____");
481 return true;
482 }
483 // _____
484
485 void gira(bool lad,int angulo,byte delta) // Movimento em curva
486 {
487 // Só para modo 1 => -128/0/127
488 // lad: true => esquerda; false => direita
489 // angulo: angulo para a direção final em graus
490 // delta: acrescimo na velocidade de uma das rodas para fazer a curva
491 if (direcao == frente) // Movimento para frente
492 {
493   if (lad == esquerda) // Movimento para a frente curva à esquerda
494   {
495     encoder2;
496     signed long giro = valor.encoder + round((float)angulo * PI * (float)eixo / 180.0);
497     signed char vc = le_velocidade_motor2() + delta;
498     //Serial.print("\nVelocidade giro ");
499     //Serial.print(vc);
500     do // Faz a curva
501     {
502       define_velocidade_motor2(vc);
503       encoder2;
504       //Serial.print("\nEncoder giro ");
505       //Serial.print(valor.encoder);
506     }while(valor.encoder < giro);
507     encoders_reset();
508     vc = le_velocidade_motor1();
509     define_velocidade_motor2(vc); // Volta ao movimento inicial
510   }
511   else // Movimento para a frente curva à direita
512   {
513     encoder1;
514     signed long giro = valor.encoder + round((float)angulo * PI * (float)eixo / 180.0);
515     signed char vc = le_velocidade_motor1() + delta;
516     //Serial.print("\nVelocidade giro ");
517     //Serial.print(vc);
518     do // Faz a curva
519     {
520       define_velocidade_motor1(vc);
521       encoder1;

```

```

523 //Serial.print("\nEncoder giro ");
524 //Serial.print(valor.encoder);
525 }while(valor.encoder < giro);
526 encoders_reset();
527 vc = le_velocidade_motor2();
528 define_velocidade_motor1(vc); // Volta ao movimento inicial
529 }
530 }
531 else // Movimento para tras
532 {
533 if (lad == esquerda) // Movimento para tras curva à esquerda
534 {
535 encoder1;
536 signed long giro = valor.encoder - round((float)angulo * PI * (float)eixo / 180.0);
537 signed char vc = le_velocidade_motor2() - delta;
538 //Serial.print("\nVelocidade giro ");
539 //Serial.print(vc);
540 do // Faz a curva
541 {
542 define_velocidade_motor2(vc);
543 encoder2;
544 //Serial.print("\nEncoder giro ");
545 //Serial.print(valor.encoder);
546 }while(valor.encoder > giro);
547 encoders_reset();
548 vc = le_velocidade_motor1();
549 define_velocidade_motor2(vc); // Volta ao movimento inicial
550 }
551 else // Movimento para tras curva à direita
552 {
553 encoder2;
554 signed long giro = valor.encoder - round((float)angulo * PI * (float)eixo / 180.0);
555 signed char vc = le_velocidade_motor1() - delta;
556 //Serial.print("\nVelocidade giro ");
557 //Serial.print(vc);
558 do // Faz a curva
559 {
560 define_velocidade_motor1(vc);
561 encoder1;
562 //Serial.print("\nEncoder giro ");
563 //Serial.print(valor.encoder);
564 }while(valor.encoder > giro);
565 encoders_reset();
566 vc = le_velocidade_motor2();
567 define_velocidade_motor1(vc); // Volta ao movimento inicial
568 }
569 }
570 }
571 }
572 // _____
573
574 void MD25modo1_begin(byte end,int eix,float per)
575 {
576 // end => endereço I2C do MD25
577 // eix => Tamanho do eixo entre centros das rodas em milímetros
578 // per => Perimetro das rodas em milímetros
579 endereco_i2c = end;
580 eixo = eix;
581 perimetro_roda = per;
582 define_mod0_1();
583 stop_motores;
584 encoders_reset();
585 direcao = frente;
586 encoder1;
587 delay(500);
588 }
589
590 // _____
591
592 #endif
593

```