```
/*****************************************************************************************************************
* This program is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General    *
* Public License as published by the Free Software Foundation, either version 3 of the License, or any later version.  *
* This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied   *
* warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more  *
* details. You should have received a copy of the GNU Lesser General Public License along with this program.          *
* If not, see <http://www.gnu.org/licenses/> or <http://blog.robotica.eng.br/gnu-lesser-general-public-license/>.     *
*                                                                                                                    *
* © Copyright 2021 Filippo Pardini - filippo@robotica.eng.br - http://blog.robotica.eng.br/                          *
* ****************************************************************************************************************** *
* Esta biblioteca implementa (para o Arduino) as funções para operação do "MD25 - 12v 2.8A dual H-bridge motor driver" *
* da Robot Electronics, no modo I2C:                                                                                 *
* http://www.robot-electronics.co.uk/htm/md25tech.htm                                                                *
* http://www.robot-electronics.co.uk/htm/md25i2c.htm                                                                 *
*                                                                                                                    *
* Atenção: a interface I2C do MD25 é de 5Vdc, se for utilizar com microcontrolador de I2C 3,3Vdc, é necessário utilizar*
* um conversor de nível. Em ambos os casos é aconselhavel utilizar resistores pull-up de 1,8K no SDA e SCL.         *
*                                                                                                                    *
*********************************************************************************************************************/

#ifndef MD25oppip_h
#define MD25oppip_h

#include "Wire.h"                                           // I2C

// ****************************************** Registros do MD25 ***********************************************

byte registro_speed1    = 0x00;                             // Registro de velocidade do motor 1 - Spped1
// Depending on what mode you are in, this register can affect the speed of one motor or both motors. If you are in
// mode 0 or 1 it will set the speed and direction of motor 1. The larger the number written to this register, the more
// power is applied to the motor. A mode of 2 or 3 will control the speed and direction of both motors
// (subject to effect of turn register).

//_____________________________________________________________________________________________________________

byte registro_speed2    = 0x01;                             // Registro de velocidade do motor 2 - Speed2/Turn
// When in mode 0 or 1 this register operates the speed and direction of motor 2. When in mode 2 or 3 Speed2 becomes
// a Turn register, and any value in this register is combined with the contents of Speed1 to steer the device.

//_____________________________________________________________________________________________________________

byte registro_enc1a = 0x02;                                 // Registro do encoder do motor 1 - Enc1a - 1º byte
byte registro_enc1b = 0x03;                                 // Registro do encoder do motor 1 - Enc1b - 2º byte
byte registro_enc1c = 0x04;                                 // Registro do encoder do motor 1 - Enc1c - 3º byte
byte registro_enc1d = 0x05;                                 // Registro do encoder do motor 1 - Enc1d - 4º byte
byte registro_enc2a = 0x06;                                 // Registro do encoder do motor 2 - Enc2a - 1º byte
byte registro_enc2b = 0x07;                                 // Registro do encoder do motor 1 - Enc2b - 2º byte
byte registro_enc2c = 0x08;                                 // Registro do encoder do motor 1 - Enc2c - 3º byte
byte registro_enc2d = 0x09;                                 // Registro do encoder do motor 1 - Enc2d - 4º byte
// Each motor has its encoder count stored in an array of four bytes, together the bytes form a signed 32 bit number,
// the encoder count is captured on a read of the highest byte (registers 2, 6) and the subsequent lower bytes will be
// held until another read of the highest byte takes place. The count is stored with the highest byte in the lowest
// numbered register. The registers can be zeroed at any time by writing 32 (0x20) to the command register.

//_____________________________________________________________________________________________________________

byte registro_bateria_volts   = 0x0A;                       // Registro da voltagem da bateria
// A reading of the voltage of the connected battery is available in this register. It reads as 10 times the voltage
// (121 for 12.1v).

//_____________________________________________________________________________________________________________

byte registro_motor1_corrente = 0x0B;                       // Registro da corrente do motor 1
byte registro_motor2_corrente = 0x0C;                       // Registro da corrente do motor 2
// A guide reading of the average current through the motor is available in this register. It reads approx ten times
// the number of Amps (25 at 2.5A).

//_____________________________________________________________________________________________________________

byte registro_software_versao= 0x0D;                        // Registro da versão do software
// This register contains the revision number of the software in the modules PIC16F873 controller.

//_____________________________________________________________________________________________________________

byte registro_taxa_aceleracao = 0x0E;                       // Registro da taxa de aceleração
// If you require a controlled acceleration period for the attached motors to reach there ultimate speed, the MD25 has
// a register to provide this. It works by using a value into the acceleration register and incrementing the power
// by that value. Changing between the current speed of the motors and the new speed (from speed 1 and 2 registers).
// So if the motors were traveling at full speed in the forward direction (255) and were instructed to move at full
// speed in reverse (0), there would be 255 steps with an acceleration register value of 1, but 128 for a value of 2.
// The default acceleration value is 5, meaning the speed is changed from full forward to full reverse in 1.25 seconds.
// The register will accept values of 1 up to 10 which equates to a period of only 0.65 seconds to travel from full
// speed in one direction to full speed in the opposite direction.

//_____________________________________________________________________________________________________________

byte registro_modo    = 0x0F;                               // Registro do modo de operação
```

```
// The mode register selects which mode of operation and I2C data input type the user requires. The options being:
// 0 - (default setting) If a value of 0 is written to the mode register then the meaning of the speed registers is
//     literal speeds in the range of 0 (full reverse)  128 (stop)   255 (full forward).
// 1 - Mode 1 is similar to mode 0, except that the speed registers are interpreted as signed values.
//     The meaning of the speed registers is literal speeds in the range of -128 (full reverse)   0 (Stop)
//     127 (full forward).
// 2 - Writing a value of  2 to the mode register will make Speed1 control both motors speed, and Speed2 becomes
//     the turn value. Data is in the range of 0 (full reverse)  128 (stop)  255 (full  forward).
// 3 - Mode 3 is similar to mode 2, except that the speed registers are interpreted as signed values. Data is in the
// range of -128  (full reverse)  0 (stop)   127 (full forward).
// ***************************************************************************************************************
// ESTA BIBLIOTECA USA SÓMENTE O MODO 1 ****************************************************************************
// ***************************************************************************************************************
//_______________________________________________________________________________________________________________

byte registro_comando   = 0x10;                                 // Registro de comando
// 0x20 - Resets the encoder registers to zero
// 0x30 - Disables automatic speed regulation
// 0x31 - Enables automatic speed regulation (default)
// 0x32 - Disables 2 second timeout of motors (version 2 onwards only)
// 0x33 - Enables 2 second timeout of motors when no I2C comms (default) (version 2 onwards only)
// 0xA0 - 1st in sequence to change I2C address
// 0xAA - 2nd in sequence to change I2C address
// 0xA5 - 3rd in sequence to change I2C address
// 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
// sequence commands in the correct order followed by the address. Example; to change the address of an MD25 currently
// at 0xB0 (the default shipped address) to 0xB4, write the following to address 0xB0; (0xA0, 0xAA, 0xA5, 0xB4 ).
// These commands must be sent in the correct sequence to change the I2C address, additionally, no other command may be
// issued in the middle of the sequence. The sequence must be sent to the command register at location 16, which means
// 4 separate write transactions on the I2C bus. Because of the way the MD25 works internally, there MUST be a delay of
// at least 5mS between the writing of each of these 4 transactions. When done, you should label the MD25 with its
// address, however if you do forget, just power it up without sending any commands. The MD25 will flash its address
// out on the green communication LED. One long flash followed by a number of shorter flashes indicating its address.
// Any command sent to the MD25 during this period will still be received and writing new speeds or a write to the
// command register will terminate the flashing.

// ***************************************************************************************************************

#define stop_motor1 define_velocidade_motor1(stop)              // Stop motor1
#define stop_motor2 define_velocidade_motor2(stop)              // Stop motor1
#define stop_motores define_velocidade_motor1(stop);define_velocidade_motor2(stop)  // Stop ambos os motores
#define encoder1 le_encoder(1)                                  // Posição do encoder do motor 1
#define encoder2 le_encoder(2)                                  // Posição do encoder do motor 2

byte endereco_i2c;                                              // Endereço I2C do MD25
                                                                // 0x58,0x59,0x5A,0x5B,0x5C,0x5D,0x5E,0x5F
byte stop = 0;                                                  // Modo 1
byte modo = 1;                                                  // Modo de operação - modo 1
float perimetro_roda;                                           // Perimetro das rodas em milimetros
int eixo;                                                       // Tamanho do eixo entre rodas em milimetros
bool const frente = true;
bool const re = false;
bool const esquerda = true;
bool const direita = false;
bool direcao;                                                   // Direção do movimento. true => frente; false => re

union                                                           // União para equivalência entre os 4 bytes do encoder
{                                                               // e um longo com sinal
  byte encoder_bytes[4];
  signed long encoder;
} valor;

//_______________________________________________________________________________________________________________

void muda_endereco_i2c(byte novo_addr)                          // Muda endereço I2c do MD25
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_comando);
  Wire.write(0xA0);
  delay(6);
  Wire.write(0xAA);
  delay(6);
  Wire.write(0xA5);
  delay(6);
  Wire.write(novo_addr);
  delay(6);
  Wire.endTransmission();
}

//_______________________________________________________________________________________________________________

void define_modo_1(void)                                        // Define o modo de operação 1
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_modo);
  Wire.write(1);
  Wire.endTransmission();
```

```
}

//_____________________________________________________________________________________________

byte le_modo()                                          // retorna o valor do modo
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_modo);
  Wire.endTransmission();

  Wire.requestFrom(endereco_i2c,1);
  while(Wire.available() < 1);
  byte mod = Wire.read();
  return mod;
}

//_____________________________________________________________________________________________

void encoders_reset()                                   // Zera o valor dos encoders
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_comando);
  Wire.write(0x20);
  Wire.endTransmission();
}

//_____________________________________________________________________________________________

void desabilita_regulacao_velocidade()                  // Desabilita regulação automática da velocidade
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_comando);
  Wire.write(0x30);
  Wire.endTransmission();
}

//_____________________________________________________________________________________________

void habilita_regulacao_velocidade()                    // Habilita regulação automática da velocidade (default)
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_comando);
  Wire.write(0x31);
  Wire.endTransmission();
}

//_____________________________________________________________________________________________

void desabilita_timeout()                               // Desabilita 2seg de timeout
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_comando);
  Wire.write(0x32);
  Wire.endTransmission();
}

//_____________________________________________________________________________________________

void habilita_timeout()                                 // Habilita 2seg de timeout (default)
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_comando);
  Wire.write(0x33);
  Wire.endTransmission();
}

//_____________________________________________________________________________________________

void define_velocidade_motor1(signed char vel)          // Aciona o motor1 à velocidade vel (-128/0/127)
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_speed1);
  Wire.write(vel);
  Wire.endTransmission();
}

//_____________________________________________________________________________________________

signed char le_velocidade_motor1()                      // Retorna a velocidade do motor1
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_speed1);
  Wire.endTransmission();

  Wire.requestFrom(endereco_i2c,1);
  while(Wire.available() < 1);
  signed char speed = Wire.read();
```

```
  return speed;
}

//_____________________________________________________________________________________________________________

void define_velocidade_motor2(signed char vel)                  // Aciona o motor2 à velocidade vel (-128/0/127)
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_speed2);
  Wire.write(vel);
  Wire.endTransmission();
}

//_____________________________________________________________________________________________________________

signed char le_velocidade_motor2()                              // Retorna a velocidade do motor2
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_speed2);
  Wire.endTransmission();

  Wire.requestFrom(endereco_i2c,1);
  while(Wire.available() < 1);
  signed char speed = Wire.read();
  return speed;
}

//_____________________________________________________________________________________________________________

void le_encoder(byte motor)                                     // Retorna o valor do encoder do motor 1 ou 2
{
  Wire.beginTransmission(endereco_i2c);
  if (motor == 1) Wire.write(registro_enc1a);
  else Wire.write(registro_enc2a);
  Wire.endTransmission();

  Wire.requestFrom(endereco_i2c,4);
  while(Wire.available() < 4);
  valor.encoder_bytes[3] = Wire.read();
  valor.encoder_bytes[2] = Wire.read();
  valor.encoder_bytes[1] = Wire.read();
  valor.encoder_bytes[0] = Wire.read();
  if (motor == 1) valor.encoder = -valor.encoder;               // Acerto de sinal, depende de como estão montados
                                                                // os motores no chassis
  if ((motor == 1) && (valor.encoder > 0)) direcao = frente;
  else direcao = re;
}

//_____________________________________________________________________________________________________________

float le_voltagem_bateria()                                     // Retorna a voltagem da bateria
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_bateria_volts);
  Wire.endTransmission();

  Wire.requestFrom(endereco_i2c,1);
  while(Wire.available() < 1);
  float volts = (float)Wire.read()/10.0;
  return volts;
}

//_____________________________________________________________________________________________________________

float le_corrente_motor1()                                      // Retorna a corrente do motor 1
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_motor1_corrente);
  Wire.endTransmission();

  Wire.requestFrom(endereco_i2c,1);
  while(Wire.available() < 1);
  float current = (float)Wire.read()/10.0;
  return current;
}

//_____________________________________________________________________________________________________________

float le_corrente_motor2()                                      // Retorna a corrente do motor 2
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_motor2_corrente);
  Wire.endTransmission();

  Wire.requestFrom(endereco_i2c,1);
  while(Wire.available() < 1);
  float current = (float)Wire.read()/10.0;
```

```
  return current;
}

//_____________________________________________________________________________________________________________

byte le_versao_software()                                   // Retorna a versão do software
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_software_versao);
  Wire.endTransmission();

  Wire.requestFrom(endereco_i2c,1);
  while(Wire.available() < 1);
  byte software = Wire.read();
  return software;
}

//_____________________________________________________________________________________________________________

void define_taxa_aceleracao(byte taxa)                      // Define taxa de aceleração (default 5 - 1,275seg)
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_taxa_aceleracao);
  Wire.write(taxa);
  Wire.endTransmission();
}

//_____________________________________________________________________________________________________________

byte le_taxa_aceleracao()                                   // Retorna a taxa de aceleracao
{
  Wire.beginTransmission(endereco_i2c);
  Wire.write(registro_taxa_aceleracao);
  Wire.endTransmission();

  Wire.requestFrom(endereco_i2c,1);
  while(Wire.available() < 1);
  byte rate = Wire.read();
  return rate;
}

//_____________________________________________________________________________________________________________

bool para_frente(signed char vel,byte ace)                  // Movimento para frente
{
  // Só para modo 1 => -128/0/127
  // vel => velocidade (1 => 127)
  // ace => taxa de aceleração (1 a 10)

  direcao = frente;
  if ((vel < 1) || (vel > 127) || (ace < 1) || (ace > 10)) return false;
  define_taxa_aceleracao(ace);
  define_velocidade_motor2(vel);
  define_velocidade_motor1(vel);
  //encoder1;
  //Serial.print("\nEncoder ");
  //Serial.print(valor.encoder);
}

//_____________________________________________________________________________________________________________

bool para_tras(signed char vel,byte ace)                    // Movimento para tras (ré)
{
  // Só para modo 1 => -128/0/127
  // vel => velocidade (-1 => -128)
  // ace => taxa de aceleração (1 a 10)

  direcao = re;
  if ((vel > -1) || (ace < 1) || (ace > 10)) return false;
  define_taxa_aceleracao(ace);
  define_velocidade_motor2(vel);
  define_velocidade_motor1(vel);
  //encoder1;
  //Serial.print("\nEncoder ");
  //Serial.print(valor.encoder);
}

//_____________________________________________________________________________________________________________

bool para_frente_distancia(long dis,signed char vel,byte ace)   // Movimento para frente por uma distância
{
  // Só para modo 1 => -128/0/127
  // dis => distância a percorrer em milimetros
  // vel => velocidade (1 => 127)
  // ace => taxa de aceleração (1 a 10)

  direcao = frente;
```

```
  if ((vel < 1) || (vel > 127) || (ace < 1) || (ace > 10)) return false;
  encoder1;
  signed long encod = valor.encoder + round(float(dis) / perimetro_roda * 360.0);
  define_taxa_aceleracao(ace);
  //Serial.print("\n\nPARA FRENTE");
  do
  {
    define_velocidade_motor2(vel);
    define_velocidade_motor1(vel);
    encoder1;
    //Serial.print("\nEncoder ");
    //Serial.print(valor.encoder);
  }while(valor.encoder < encod);
  stop_motores;
  encoders_reset();
  //Serial.print("\n_____________________________________________________");
  return true;
}

//_____________________________________________________________________________________________________________

bool para_tras_distancia(long dis,signed char vel,byte ace)     // Movimento para tras por uma distância
{
  // Só para modo 1 => -128/0/127
  // dis => distância a percorrer em milimetros
  // vel => velocidade (-1 => -128)
  // ace => taxa de aceleração (1 a 10)

  direcao = re;
  if ((vel > -1) || (ace < 1) || (ace > 10)) return false;
  encoder1;
  signed long encod = valor.encoder - (round(float(dis) / perimetro_roda * 360.0));
  define_taxa_aceleracao(ace);
  //Serial.print("\n\nPARA TRAS");
  do
  {
    define_velocidade_motor2(vel);
    define_velocidade_motor1(vel);
    encoder1;
    //Serial.print("\nEncoder ");
    //Serial.print(valor.encoder);
  }while(valor.encoder > encod);
  stop_motores;
  encoders_reset();
  //Serial.print("\n_____________________________________________________");
  return true;
}

//_____________________________________________________________________________________________________________

void gira(bool lad,int angulo,byte delta)                       // Movimento em curva
{
  // Só para modo 1 => -128/0/127
  // lad: true => esquerda; false => direita
  // angulo: angulo para a direção final em graus
  // delta: acrescimo na velocidade de uma das rodas para fazer a curva
  if (direcao == frente)                                        // Movimento para frente
  {
    if (lad == esquerda)                                        // Movimento para a frente curva à esquerda
    {
      encoder2;
      signed long giro = valor.encoder + round((float)angulo * PI * (float)eixo / 180.0);
      signed char vc = le_velocidade_motor2() + delta;
      //Serial.print("\nVelocidade giro ");
      //Serial.print(vc);
      do                                                        // Faz a curva
      {
        define_velocidade_motor2(vc);
        encoder2;
        //Serial.print("\nEncoder giro ");
        //Serial.print(valor.encoder);
      }while(valor.encoder < giro);
      encoders_reset();
      vc = le_velocidade_motor1();
      define_velocidade_motor2(vc);                             // Volta ao movimento inicial
    }
    else                                                        // Movimento para a frente curva à direita
    {
      encoder1;
      signed long giro = valor.encoder + round((float)angulo * PI * (float)eixo / 180.0);
      signed char vc = le_velocidade_motor1() + delta;
      //Serial.print("\nVelocidade giro ");
      //Serial.print(vc);
      do                                                        // Faz a curva
      {
        define_velocidade_motor1(vc);
        encoder1;
```

```
        //Serial.print("\nEncoder giro ");
        //Serial.print(valor.encoder);
      }while(valor.encoder < giro);
      encoders_reset();
      vc = le_velocidade_motor2();
      define_velocidade_motor1(vc);                          // Volta ao movimento inicial
    }
  }
  else                                                       // Movimento para tras
  {
    if (lad == esquerda)                                     // Movimento para tras curva à esquerda
    {
      encoder1;
      signed long giro = valor.encoder - round((float)angulo * PI * (float)eixo / 180.0);
      signed char vc = le_velocidade_motor2() - delta;
      //Serial.print("\nVelocidade giro ");
      //Serial.print(vc);
      do                                                     // Faz a curva
      {
        define_velocidade_motor2(vc);
        encoder2;
        //Serial.print("\nEncoder giro ");
        //Serial.print(valor.encoder);
      }while(valor.encoder > giro);
      encoders_reset();
      vc = le_velocidade_motor1();
      define_velocidade_motor2(vc);                          // Volta ao movimento inicial
    }
    else                                                     // Movimento para tras curva à direita
    {
      encoder2;
      signed long giro = valor.encoder - round((float)angulo * PI * (float)eixo / 180.0);
      signed char vc = le_velocidade_motor1() - delta;
      //Serial.print("\nVelocidade giro ");
      //Serial.print(vc);
      do                                                     // Faz a curva
      {
        define_velocidade_motor1(vc);
        encoder1;
        //Serial.print("\nEncoder giro ");
        //Serial.print(valor.encoder);
      }while(valor.encoder > giro);
      encoders_reset();
      vc = le_velocidade_motor2();
      define_velocidade_motor1(vc);                          // Volta ao movimento inicial
    }
  }
}

//_____________________________________________________________________________________________________

void MD25modo1_begin(byte end,int eix,float per)
{
  // end => endereço I2C do MD25
  // eix => Tamanho do eixo entre centros das rodas em milimetros
  // per => Perimetro das rodas em milimetros
  endereco_i2c = end;
  eixo = eix;
  perimetro_roda = per;
  define_modo_1();
  stop_motores;
  encoders_reset();
  direcao = frente;
  encoder1;
  delay(500);
}

//_____________________________________________________________________________________________________

#endif
```