UFRJ

Listagem dos programas de controle (*sketch*) para a placa Arduino utilizados no presente trabalho.

I - Programa de medida de concentração de CH_4 com saída de dados em cartão de memória.

No CD que acompanha a tese fornecemos os arquivos prontos para uso na Arduino Uno. A maior parte do programa é usada para escrever as medidas dos sensores no cartão de memória. A parte do programa que instrui a gravação no cartão de memória bem como as bibliotecas (sub-rotina) auxiliares **SD.h** e **RTClib.h** foram desenvolvidas pela fabricante da placa (*shield*), a Adafruit, e são disponibilizadas gratuitamente a partir do seu sítio (www.adafruit.com). A biblioteca **RTClib.h** controla o circuito integrado DS1307 (*real time clock*) que compõe o *shield* e é usado na atualização de data e hora nos registros gravados no cartão de memória. Algumas observações originais em inglês foram mantidas na listagem. É importante lembrar que a Adafruit fornece também vários textos explicativos sobre o *shield* e que são muito úteis e didáticos. A biblioteca **wire.h** é distribuída com a Arduino e usada para a comunicação com dispositivos I^2C.

O *shield* nas versões que utilizamos é fornecido desmontado. Será necessário soldar todos os componentes na placa de circuito impresso. Nesse caso, será necessário ter a mão um ferro de solda (30 W) e solda. Não é necessário muita experiência de soldagem, mas se você se sentir inseguro procure a ajuda de um técnico. Como o *shield datalogger* para a Arduino tem tido muita procura a Adfruit deve lançar uma nova versão atualizada, e com novas funcionalidades e pronta para uso.

```
// Tese de Sandro Monteiro da Costa - versão 2.0.0
// Um datalogger para Arduino e um sensor de metano

#include <SD.h>
#include <Wire.h>
#include "RTClib.h"


// intervalo de tempo (milisegundos) entre leituras
// use 10*LOG_INTERVAL para gravar um dado a cada 10 leituras

#define LOG_INTERVAL 20000 //define o intervalo entre o registro
                           //de duas medidas, neste caso 20000 ms, ou 20s.
                  //Altere esse número segundo suas necessidades.


#define SYNC_INTERVAL 1000 // milisegundos entre a chamada para
                           //limpar e para gravar dados no cartão.
```

```
uint32_t syncTime = 0; // time of last sync()

#define ECHO_TO_SERIAL   1 // echo data to serial port
#define WAIT_TO_START    0 // Wait for serial input in setup()

// pinos digitais conectados aos LEDs
#define redLEDpin 2
#define greenLEDpin 3

// pinos analogicos conectados aos sensores
#define metanPin 0           // analog 0

#define aref_voltage 5.0         // we tie 5.0V to ARef and
                            //measure it with a multimeter!

RTC_DS1307 RTC; // define the Real Time Clock object

//for the data logging shield, we use digital pin 10 for the SD
//cs line.

const int chipSelect = 10;

// the logging file
File logfile;

void error(char *str)
{
  Serial.print("error: ");
  Serial.println(str);

  // red LED indicates error
  digitalWrite(redLEDpin, HIGH);

  while(1);
}

void setup(void)
{
  Serial.begin(9600);
  Serial.println();

  // use debugging LEDs
  pinMode(redLEDpin, OUTPUT);
  pinMode(greenLEDpin, OUTPUT);

#if WAIT_TO_START
  Serial.println("Type any character to start");
  while (!Serial.available());
#endif //WAIT_TO_START

  // Inicializa o cartão SD
  Serial.print("Initializing SD card...");
  // make sure that the default chip select pin is set to
  // output, even if you don't use it:
  pinMode(10, OUTPUT);
```

```
  // VERIFICA SE O CARTÃO ESTÁ PRESENTE E PODE SER INICIALIZADO:
  if (!SD.begin(chipSelect)) {
    error("Card failed, or not present");
  }
  Serial.println("card initialized.");

// Cria um novo arquivo denominado LOGGER00.CSV. A terminação
//CSV (Comma-separated values) é reconhecido pelo EXCEL e
//facilmente transportado para ele.

  char filename[] = "LOGGER00.CSV";
  for (uint8_t i = 0; i < 100; i++) {
    filename[6] = i/10 + '0';
    filename[7] = i%10 + '0';
    if (! SD.exists(filename)) {
      // only open a new file if it doesn't exist
      logfile = SD.open(filename, FILE_WRITE);
      break;  // leave the loop!
    }
  }

  if (! logfile) {
    error("couldnt create file");
  }

  Serial.print("Logging to: ");
  Serial.println(filename);

  // connect to RTC
  Wire.begin();
  if (!RTC.begin()) {
    logfile.println("RTC failed");
#if ECHO_TO_SERIAL
    Serial.println("RTC failed");
#endif  //ECHO_TO_SERIAL
  }

  logfile.println("    data_tempo,         concCH4");  //títulos
//das colunas. A primeira coluna informa data e horário
//(data_tempo) e a segunda (concCH4)a DDP no sensor MQ-4.

#if ECHO_TO_SERIAL
  Serial.println("    data_tempo,         concCH4");
#endif //ECHO_TO_SERIAL
   }

void loop(void)
{
  DateTime now;

  // delay for the amount of time we want between readings
  delay((LOG_INTERVAL -1) - (millis() % LOG_INTERVAL));

  digitalWrite(greenLEDpin, HIGH);
```

```
  // log milliseconds since starting
  uint32_t m = millis();
  //logfile.print(m);           // milliseconds since start
  //logfile.print(", ");
#if ECHO_TO_SERIAL
 // Serial.print(m);         // milliseconds since start
 // Serial.print(", ");
#endif

  // fetch the time
  now = RTC.now();
  // log time
  //logfile.print(now.unixtime()); // seconds since 1/1/1970
  //logfile.print(", ");
  logfile.print('"');
  logfile.print(now.year(), DEC);
  logfile.print("/");
  logfile.print(now.month(), DEC);
  logfile.print("/");
  logfile.print(now.day(), DEC);
  logfile.print(" ");
  logfile.print(now.hour(), DEC);
  logfile.print(":");
  logfile.print(now.minute(), DEC);
  logfile.print(":");
  logfile.print(now.second(), DEC);
  logfile.print('"');

#if ECHO_TO_SERIAL
  //Serial.print(now.unixtime()); // seconds since 1/1/1970
  //Serial.print(", ");
  Serial.print('"');
  Serial.print(now.year(), DEC);
  Serial.print("/");
  Serial.print(now.month(), DEC);
  Serial.print("/");
  Serial.print(now.day(), DEC);
  Serial.print(" ");
  Serial.print(now.hour(), DEC);
  Serial.print(":");
  Serial.print(now.minute(), DEC);
  Serial.print(":");
  Serial.print(now.second(), DEC);
  Serial.print('"');

#endif //ECHO_TO_SERIAL

      int metaReading = analogRead(metanPin);
```

// Convertendo a leitura digital para analógica.

```
float conCHiv = metaReading * aref_voltage / 1024;

  logfile.print(", ");
  logfile.println(conCHiv);

#if ECHO_TO_SERIAL
  Serial.print(", ");
  Serial.println(conCHiv);

#endif //ECHO_TO_SERIAL

  digitalWrite(greenLEDpin, LOW);

// Now we write data to disk! Don't sync too often - requires
//2048 bytes of I/O to SD card which uses a bunch of power and
//takes time.

if ((millis() - syncTime) < SYNC_INTERVAL) return;
  syncTime = millis();

  // blink LED to show we are syncing data to the card &
  //updating FAT!

 digitalWrite(redLEDpin, HIGH);
 logfile.flush();
 digitalWrite(redLEDpin, LOW);

}
```

O arquivo de dados gravado (ASCII) no cartão de memória e denominado LOGGERxx.csv. XX são dígitos que vão se incrementando a partir de 00 (01, 02 ...). Você pode usar o nome que quiser alterando na linha correspondente. Você pode inspecioná-lo com o bloco de notas (notepad). Abaixo vemos um exemplo:

```
    Data_tempo,         conCH4
"2013/1/6 11:59:21", 0.96
"2013/1/6 11:59:41", 0.96
"2013/1/6 12:0:1", 0.98
"2013/1/6 12:0:21", 0.97
"2013/1/6 12:0:41", 1.00
"2013/1/6 12:1:1", 0.99
"2013/1/6 12:1:21", 0.99
"2013/1/6 12:1:41", 1.01
"2013/1/6 12:2:1", 1.00
"2013/1/6 12:2:21", 1.01
"2013/1/6 12:2:41", 1.01
"2013/1/6 12:3:1", 1.02
```

II – Programa (sketch) de controle de servomotor pela temperatura medida por uma resistência NTC.

Neste caso, o programa é muito simples por que os objetivos fixados são diretos: queremos mostrar como um servomotor pode ser posicionado em função da temperatura de um corpo de prova. Na aplicação que fizemos o corpo de prova gira solidário ao eixo do servomotor.

```
// Tese de Sandro Monteiro da Costa - versão 2.0
// Simulação de mecanismo de feedback.

#include <Servo.h>
int angulo = 0;
Servo servo1; // cria objeto servo

void setup() {
  servo1.attach(5);  // anexa o servo (físico) no pino 5 ao
                     //objeto servo1 (lógico).
}

void loop() {
  int angle = analogRead(0); //lê o valor da DDP no resistor NTC

  if (angle > angulo) {
   angulo = angle;
    angle = map(angle, 0, 1023, 0, 180); //mapeia os valores
                                          // de 0 a 180 graus.

     servo1.write(angle);  //escreve o ângulo para o servo.
  }

  delay(1000);  //espera 1000 ms para permitir que o servo
                // atinja a posição
}
```

Na instrução *map* indicada acima (iluminada em cinza) é estabelecida uma relação linear entre a DDP observada (variável *angle*) no sensor MQ-4, que tem como limites os valores [0,1023] (equivalente ao tensões [0 V, 5 V]), e os ângulos no intervalo [0^o, 180^o], para os servos de meia volta. Se as variações efetivas de DDP no sensor forem menores (isso depende da temperatura ambiente e da temperatura máxima atingida) é possível estreitar estes extremos de forma a conseguir uma variação angular plena (180^o) dentro da faixa de temperaturas atingidas. Exemplo: se as temperaturas atingidas numa experiência dada forem (em termos de volt) 1,8 V e 3,3 V podemos fazer:

$$1{,}8 / 5 \times 1023 \approx 368 \text{ e } 3{,}3 / 5 \times 1023 = 675$$

e escrever a instrução como:

```
angle = map(angle, 368, 675, 0, 180);
```

e com isso teremos um variação de 180º do eixo do servo na faixa de temperaturas efetivamente alcançadas.