Sistemas Embebidos para Computação Física e IoT

Volume 3:
Projetos e Conceitos Avançados

RUI POLICARPO DUARTE

Projeto de Sistemas Embebidos para Computação Física e IoT

Volume 3: Projetos e Conceitos Avançados

Rui Policarpo Duarte, PhD, Eng

24 de outubro de 2023

1ª Edição: outubro de 2023
ISBN: 978-989-33-5363-9

Imagem da capa: Foto da placa de desenvolvimento CoolRunner II por Rui Policarpo Duarte

Conteúdo

Lista de Figuras **v**

Lista de Tabelas **vii**

Lista de Programas Arduíno **ix**

Nomenclatura **xi**

1 Módulos **1**
1.1 Termómetro e Higrómetro 2
1.2 Memória EEPROM 6
1.3 SD Card 8
1.4 Rede Ethernet 16
1.5 WiFi 19
1.6 GSM/GPRS 21
1.7 Bluetooth 22
1.8 LORA 25
1.9 Leitor RFID 26
1.9.1 125 kHz - 7941E 26
1.9.2 13,56 MHz - MFRC522 31
1.10 Teclado Táctil Capacitivo 34
1.11 Teclado de Computador PS/2 36
1.12 Rato de Computador PS/2 42

2 Projetos **43**
2.1 Noções Práticas 44
2.2 Calculadora Básica 45
2.2.1 Componentes 45
2.2.2 Programa 45
2.3 Relógio-Despertador 47
2.3.1 Relógio de Tempo Real 48
2.4 Mini-Osciloscópio 54
2.5 CanSat 55
2.5.1 Estrutura 55

2.5.2 Circuito . . . 56
2.5.3 Programa . . . 56
2.5.4 Demonstração . . . 57
2.5.5 Outras Considerações . . . 57
2.6 Node IoT . . . 58
2.7 Pedal de Efeitos para Guitarra Elétrica . . . 59
2.8 Mini-Jogo . . . 61
2.9 Controlador de Estufa . . . 63
2.10 Robot . . . 64
2.10.1 Estrutura . . . 64
2.10.2 Circuito . . . 64
2.10.3 Programa . . . 64
2.10.4 Demonstração . . . 64
2.11 Desenvolvimento de um Shield Arduino . . . 64

3 Conceitos Avançados **65**
3.1 Miniaturização . . . 65
3.2 Engenharia Reversa . . . 66
3.3 Eficiência Energética . . . 67
3.4 Técnicas para Aumento de Desempenho . . . 70
3.5 Resposta em Tempo-Real . . . 74
3.6 Tolerância a Falhas . . . 75
3.6.1 Cão de Guarda (Watchdog) . . . 76
3.6.2 Redundância Temporal e Espacial . . . 77
3.6.3 Pontos de Verificação (Checkpointing) . . . 77
3.6.4 Consenso (Lockstep) . . . 78
3.6.5 Auto-Verificação . . . 78
3.6.6 Degradação Graciosa . . . 78
3.6.7 Falhas Bizantinas . . . 79
3.7 Paradigmas de Programação e Padrões Arquiteturais . . . 79
3.7.1 Programação Orientada a Objetos . . . 80
3.7.2 Model-View-Controller . . . 80
3.7.3 Blackboard . . . 82
3.7.4 Event-Driven . . . 83
3.7.5 Cliente-Servidor . . . 84
3.7.6 Mestre-Escravo . . . 85
3.7.7 Produtor-Consumidor . . . 86
3.8 Controlo . . . 86
3.8.1 Controlador ON-OFF . . . 87
3.8.2 Controlador PID . . . 89
3.9 Matemática e Métodos Numéricos . . . 92
3.9.1 Medidas Estatísticas . . . 93
3.9.2 Série de Taylor . . . 97

3.9.3 Interpolação Linear 98
3.9.4 Estimativa de Curva 99
3.9.5 Convolução 99
3.9.6 Séries Pseudo-Aleatórias 102

4 Exercícios **105**

Lista de Figuras

1.1 Diferentes modelos de módulos sensor de temperatura e humidade digital - SHT21, AM2320, DHT11, DHT22, AM2302 e AM2301. 2
1.2 Detalhes do interior do circuito do módulo sensor de temperatura e humidade digital DHT22. 2
1.3 Ligações entre os módulos SHT21 e DHT11, e o Arduíno Uno. 3
1.4 Micro-SDCard. 8
1.5 Shield SD Card (esquerda) e placas adaptadoras micro SD e SD Card. 10
1.6 Esquema shield SD Card. 10
1.7 Modulo ethernet. 16
1.8 ethernet-terminal. 17
1.9 Código HTML enviado pelo Arduíno e o seu aspeto quando traduzido por um navegador de internet. 17
1.10 Código HTML enviado pelo Arduíno e o seu aspeto quando traduzido por um navegador de internet. 19
1.11 Janela de diálogo para adicionar a placa ESP8266. 19
1.12 Demonstração da comunicação com um servidor via Internet usando a placa ESP14. 20
1.13 Modulo BT HC-6. 22
1.14 Esquema de ligações ao Arduíno Uno (esquerda) e Mega (direita). 23
1.15 Listagem de componentes BT emparelhados com o computador. 24
1.16 Modulo LORA. 25
1.17 Módulo RFID LF 7941E. 27
1.18 Circuito de demonstração do Módulo RFID LF 7941E. 27
1.19 Aparato usado na demonstração do Módulo RFID LF 7941E. . 28
1.20 Sinal série assíncrono gerado pelo módulo RFID 7941E, e sua descodificação, após a identificação de uma *tag* RFID LF. . . . 29
1.21 Aparato experimental com o módulo RFID LF 7941E. 30
1.22 Conteúdo do terminal de texto após a execução do programa de demonstração e deteção de duas *tags* RFID LF. 32
1.23 Modulo RFID MFRC522. 32
1.24 Modulo NFC. 32

1.25 Montagem do módulo NFC numa breadboard para leitura de um cartão de identificação. 32
1.26 Modulo NFC. 33
1.27 touch keypad 34
1.28 Esquema do touch keypad 35
1.29 Ligação do touch keypad 35
1.30 ps2 keyboard 36
1.31 Ligações das fichas de teclado PS/2 e USB (esquerda) e as ligações a realizar para ligar a um Arduíno Uno (direita). . . . 37
1.32 Diagrama de forma de onda dos sinais DATA e CLOCK gerados por um teclado PS/2. 37
1.33 Diagrama de forma de onda do sinal CLOCK gerado pelo teclado PS/2 e leitura do sinal DATA pelo Arduíno. 39
1.34 Correspondência entre os códigos de teclado PS/2 e as teclas . 39
1.35 Aparato usado no ensaio do teclado PS/2. 40
1.36 Aspeto do conteúdo do terminal de texto do Arduíno durante o ensaio ao teclado PS/2. 41

2.1 Módulos com relógio de tempo real DS1302. 48
2.2 Circuito de demonstração do circuito do relógio de tempo real (RTC). 48
2.3 Resultado da execução do Sketch de demonstração do DS1302. 49
2.4 Shield Multifunction assente num Arduíno Uno. 52
2.5 Esquema elétrico do Shield Multifunction. 53
2.6 Módulo amplificador de 3W estéreo. 53
2.7 Peças impressas com base nos ficheiros fornecidos pelo projeto Canduino. 55
2.8 Esquema elétrico do CanSat. 57
2.9 canduino. 57
2.10 Montagem do Node-IoT com um “Rich Shield” e um “Ethernet Shield”. 58
2.11 Pedal de guitarra elétrica baseado em Arduíno. 59
2.12 Diagrama do circuito do Shield para pedal de guitarra elétrica. 59
2.13 Aparato da ligação do Shield de pedal de efeitos de guitarra à guitarra, ao amplificador e ao computador. 60
2.14 Mini jogo arduino. 62

3.1 Rolo de fio para *wirewraping* à esquerda, e ferramenta para descarnar e enrolar o fio à direita. 67
3.2 Exemplo da utilização de um MOSFET para controlar um sensor de temperatura. 69
3.3 Redundância tripla com votação. 78
3.4 Esquema da relação de classes derivadas e modificadores de acesso. 81

3.5 model-view-controller. 82
3.6 Modelo Cliente-Servidor. 85
3.7 Modelo Mestre-Escravo. 86
3.8 Modelo Produtor-Consumidor. 86
3.9 Sistema de controlo genérico com malha fechada. 87
3.10 Diagrama de blocos de um controlador PID. 91
3.11 Gráfico produzido com os valores gerados aleatoriamente pelo Sketch randomVar. 104

Lista de Tabelas

1.1 Correspondência entre os pinos dos cartões SD e os modos de ligação 9
1.2 Configuração do ritmo de transmissão do módulo BT. 23
1.3 Estrutura da trama enviada pelo modulo RFID LF 7941E. . . 26

2.1 Exemplo de funcionamento da pilha de uma calculadora RPR. 45
2.2 Componentes escolhidos e respetivos critérios de selecção para o CanSat. 56

Lista de Programas

1.1 Listagem de demonstração do módulo de sensores de temperatura e humidade DHT11 4
1.2 Listagem do Sketch de demonstração do módulo DS3231 + AT24C32 6
1.3 Listagem SDCard-Info.ino 11
1.4 Listagem SDCard-fileWrite.ino 13
1.5 Teste de presença na rede através do comando ping 16
1.6 Listagem para adicionar uma imagem animada na página gerada pelo servidor Web Arduíno 17
1.7 Sketch de demonstração do módulo RFID LF 7941E 27
1.8 Listagem do Sketch PS2Keyboard.ino 40
2.1 Listagem RealTimeClock.ino 49
2.2 Sketch adc6ChDemo.ino para realizar um mini-osciloscópio de 6 canais com um Arduíno Uno 54
3.1 Listagem do comando para obter um Sketch traduzido em linguagem Assembly. 70
3.2 Exemplo de código Assembly 71
3.3 Demonstração da utilização de estruturas para aumentar o desempenho 73
3.4 resultado da avaliação da utilização de estruturas vs arrays independentes 74
3.5 varSwap.ino 74
3.6 Exemplo do modelo Blackboard 83
3.7 Exemplo da utilização de interrupções. 83
3.8 Exemplo da utilização de função de *callback*. 84
3.9 Sketch Controlador ON-OFF 88
3.10 Sketch de um Controlador PID. 91
3.11 corrCoeff.ino 95
3.12 sinTaylor.ino 97
3.13 Sketch de demonstração da função de convulução 100
3.14 randomVar.ino 103

Nomenclatura

Acronimos e Abreviaturas

ALU	Arithmetic Logic Unit
ASAP	As Soon As Possible
ASIC	Application Specific Integrated Circuit
CAD	Computer-Aided Design
CPU	Central processing Unit
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
FPGA	Field-Programable Gate Array
FSM	Finite State Machine
GPU	Graphics Processing Unit
HDL	Hardware Description Languages
I/O	Input/Output
LUT	Look-Up Table
MAC	Multiply Accumulate
MSb	Most Significant bit
NaN	Not a Number
RAM	Random Access Memory
ROM	Read Only Memory
VHDL	Very-high-speed integrated circuit Hardware Description Language

List of Symbols

$\mathrm{O}(n)$	Complexity order n

1 Módulos

Os módulos são sub-sistemas que integram vários componentes utilizados frequentemente. Os módulos são dedicados uma funcionalidade específica e são uma solução mais compacta, com melhor qualidade e barata do que a implementação discreta do módulo. Para simplificar a interligação dos módulos eles possuem um pequeno controlador digital que permite poupar o número de sinais a ligar ao sistema embebido.

1.1 Termómetro e Higrómetro

O módulo de medição de temperatura e humidade é uma solução bastante compacta que integra os sensores de humidade e temperatura, bem como o respetivo condicionamento de sinal e circuito de conversão analógico-digital.

Existem vários modelos de sensor, apresentando a mesma construção e princípio de funcionamento, variando as gamas de temperatura e humidade, resolução e precisão. A figura 1.1 mostra um sensor SHT21 e os detalhes sem a sua cobertura.

A figura 1.2 mostra diferentes modelos de sensores de temperatura e humidade, nomeadamente da esquerda para a direita, SHT21, DHT11, DHT22, AM2302B, AM2301 e AM2320. A figura 1.2 mostra os detalhes do interior do sensor, depois de removida a sua cobertura.

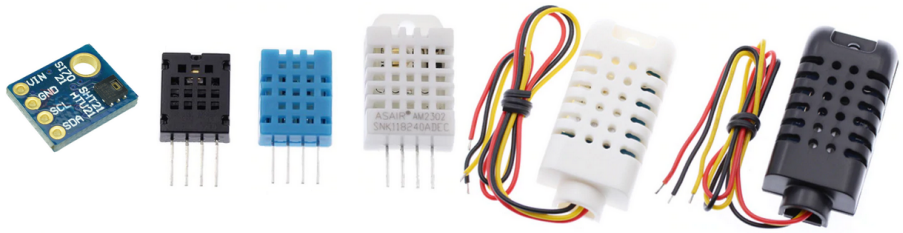

Figura 1.1: Diferentes modelos de módulos sensor de temperatura e humidade digital - SHT21, AM2320, DHT11, DHT22, AM2302 e AM2301.

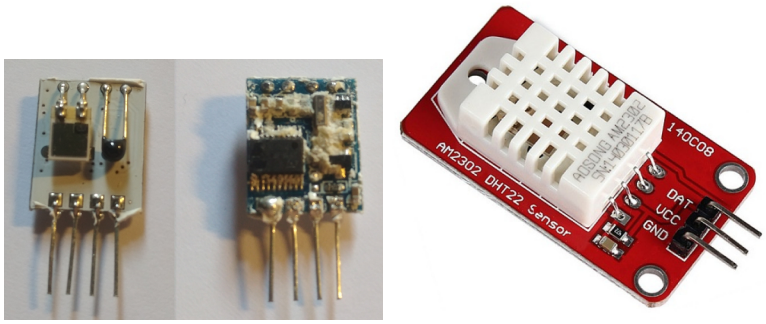

Figura 1.2: Detalhes do interior do circuito do módulo sensor de temperatura e humidade digital DHT22.

As caraterísticas principais do DHT11 são:

- Gama de medidas de humidade: 20-90% RH;
- Precisão (Accuracy) da medida de humidade: ±5% RH
- Gama de medidas de temperatura: 0-50 °C

- Precisão (Accuracy) da medida de humidade: ±5% RH
- Resolução: 16 bits
- Ritmo de amostragem máximo: 1 amostra/segundo;
- Máximo consumo de corrente: 2,5mA;
- Tensão de funcionamento: 3-5V;
- Protocolo 1-wire.

O DHT11 comunica com o sistema embebido através do protocolo 1-Wire. A comunicação dos valores de temperatura e humidade pelo sensor consiste no envio de 40 bits, com o seguinte formato: parte inteira da humidade relativa (8bits) + parte fracionária da humidade relativa (8bits) + parte inteira da temperatura (8bits) + parte fracionária da temperatura (8bits).

Single-bus data format is used for communication and synchronization between MCU and DHT11 sensor. One communication process is about 4ms. Data consists of decimal and integral parts. A complete data transmission is 40bit, and the sensor sends higher data bitfirst. Data format:8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum. If the data transmission is right,the check-sum should be the last 8bit of "8bit integral RH data+8bit decimal RHdata+8bit integral T data+8bit decimal T data".

A figura 1.3 ilustra as ligações a realizar para testar o DHT11 com um Arduíno Uno.

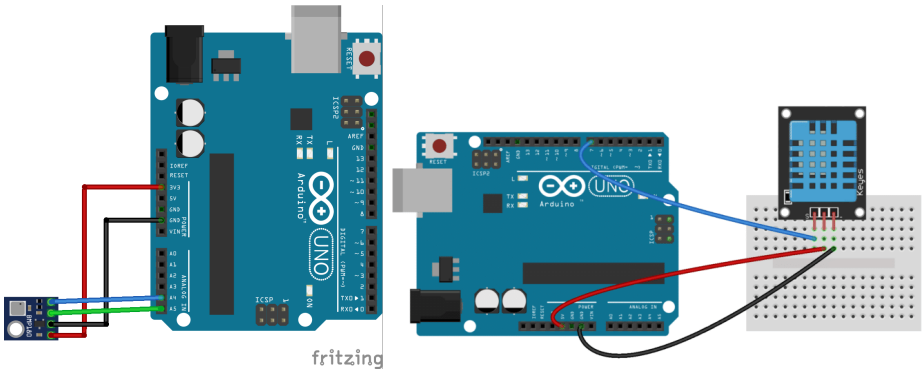

Figura 1.3: Ligações entre os módulos SHT21 e DHT11, e o Arduíno Uno.

A listagem 1.1 exemplifica o uso do DHT11.

Listagem 1.1: Listagem de demonstração do módulo de sensores de temperatura e humidade DHT11

```
// DHT11 Temperature and humidity sensor



#define DHT11PIN            8

#define DHTLIB_OK           0
#define DHTLIB_ERROR_CHECKSUM -1
#define DHTLIB_ERROR_TIMEOUT  -2

class DHT11
{
private:
  int pin;
  int humidity;
  int temperature;
public:
  DHT11()    { pin = DHT11PIN;    }
  int getData(int pin);
};




// Return values:
// DHTLIB_OK
// DHTLIB_ERROR_CHECKSUM
// DHTLIB_ERROR_TIMEOUT
int dht11::read(int pin)
{
  unsigned long t;
  unsigned int loopCnt

  uint8_t bits[5];
  uint8_t cnt = 7;
  uint8_t idx = 0;

  // EMPTY BUFFER
  memset(bits, 0, 5);

  // REQUEST SAMPLE
  pinMode(pin, OUTPUT);
  digitalWrite(pin, LOW);
  delay(18);
  digitalWrite(pin, HIGH);
  delayMicroseconds(40);
  pinMode(pin, INPUT_PULLUP);

  // ACKNOWLEDGE or TIMEOUT
  loopCnt = 10000;
  while(digitalRead(pin) == LOW)
    if (loopCnt-- == 0) return DHTLIB_ERROR_TIMEOUT;

  loopCnt = 10000;
  while(digitalRead(pin) == HIGH)
    if (loopCnt-- == 0) return DHTLIB_ERROR_TIMEOUT;

  // READ OUTPUT: 40 BITS => 5 BYTES or TIMEOUT
  for (int i=0; i<40; i++)
  {
    loopCnt = 10000;
    while(digitalRead(pin) == LOW)
      if (loopCnt-- == 0) return DHTLIB_ERROR_TIMEOUT;

    t = micros();

    loopCnt = 10000;
    while(digitalRead(pin) == HIGH)
      if (loopCnt-- == 0) return DHTLIB_ERROR_TIMEOUT;
```

```
    if ((micros() - t) > 40) bits[idx] |= (1 << cnt);
    if (cnt == 0)   // next byte?
    {
      cnt = 7;    // restart at MSB
      idx++;      // next byte!
    }
    else cnt--;
  }

  // WRITE TO RIGHT VARS
        // as bits[1] and bits[3] are allways zero they are omitted in formulas.
  humidity    = bits[0];
  temperature = bits[2];

  uint8_t sum = bits[0] + bits[2];

  if (bits[4] != sum) return DHTLIB_ERROR_CHECKSUM;
  return DHTLIB_OK;
}




dht11 DHT11;

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

}

void loop()
{
  Serial.println();

  int chk = DHT11.read(DHT11PIN);

  Serial.print("Humidity (%): ");
  Serial.println((float)DHT11.humidity, 2);

  Serial.print("Temperature (C): ");
  Serial.println((float)DHT11.temperature, 2);

  delay(2000);

}
```

1.2 Memória EEPROM

Os dados que não façam parte do Sketch Arduíno desaparecem após reset ou retirada a alimentação. As memórias Electrically Erasable Programmable Read-Only Memory (EEPROM) permitem guardar dados de forma não volátil. Isto significa que após guardar informação na memória caso o circuito seja desligado, os dados não se perdem e podem ser acedidos posteriormente. Este tipo de memórias é útil para guardar personalizações ou parâmetros do sistema, registo de dados remotamente com sistemas alimentados a bateria ou painel solar.

O módulo RTC DS3231 apresentado anteriormente tem na sua placa uma memória EEPROM AT24C32 a partilhar os pinos com o RTC. Na figura 2.1 à direita, a memória corresponde ao encapsulamento SOIC-8 (quadrado preto mais pequeno com 4 terminais de cada lado). Neste módulo a comunicação com a memória e com o RTC é distinguida pelo endereço de destino. O endereço I2C da memória AT24C32 é o 0x50.

Esta memória tem a capacidade de armazenar 32768 bits (32kb) ou 4096 palavras de 1 byte (4kB). A EEPROM AT24C64 tem o dobro da capacidade. Embora estas capacidades não pareçam muito, comparado com a quantidade de memória RAM disponível, se for usado 1 byte para registar a temperatura a cada hora, é possível registar a temperatura durante 4096 horas, ou seja, 170 dias ou cerca de 5 meses e meio. Para permitir registar dados durante muito tempo, esta memória foi especificada para gastar pouca energia (apx. 2uA) e funcionar com níveis de tensão baixos (1,8-5V).

A listagem 1.2 mostra o Sketch que permite demonstrar o uso do módulo com a EEPROM AT24C32 e o RTC DS3231.

Listagem 1.2: Listagem do Sketch de demonstração do módulo DS3231 + AT24C32

```
#include "DS3231_Simple.h"

DS3231_Simple Clock;

void setup() {


  Serial.begin(9600);
  Serial.println();

  Clock.begin();

  // Erase the contents of the EEPROM
  Clock.formatEEPROM();

  // First we will disable any existing alarms
  Clock.disableAlarms();

  // And now add the alarm to happen every second
  Clock.setAlarm(DS3231_Simple::ALARM_EVERY_SECOND);

  Serial.println(F("Logging value of analogRead(A1), enter any character to dump the log."));
```

```
}

void loop()
{
  if(Clock.checkAlarms())
  {
    // Time to log a data point
    Clock.writeLog(analogRead(A1));
    Serial.print('.');
  }

  if(Serial.available())
  {
    while(Serial.available()) Serial.read();
    dumpLog();
  }
}

void dumpLog()
{
  unsigned int loggedData;
  DateTime     loggedTime;

  // Note that reading a log entry also deletes the log entry
  // so you only get one-shot at reading it, if you want to do
  // something with it, do it before you discard it!
  unsigned int x = 0;
  while(Clock.readLog(loggedTime,loggedData))
  {
    if(x == 0)
    {
      Serial.println();
      Serial.println(F("Date,analogRead(A1)"));
    }

    x++;
    Clock.printTo(Serial,loggedTime);
    Serial.print(',');
    Serial.println(loggedData);
  }
  Serial.println();
  Serial.print(F("# Of Log Entries Found: "));
  Serial.println(x);
  Serial.println();
}
```

No Arduíno Uno o canal série é usado para descarregar o Sketch. Isto significa que o módulo Bluetooth (BT) deve ser desligado para evitar conflito.

1.3 SD Card

Os cartões de memória do tipo SD Card (*Secure Digital Card*) são não voláteis, permitem armazenar grandes volumes de dados (até centenas de GB) e são destinados a equipamentos portáteis como máquinas fotográficas, computadores, telemóveis, *tablets* e *drones*. Um SD card é caracterizado pela sua capacidade de armazenamento, velocidade de leitura e escrita.

A norma que define os SD Cards[1] estabeleceu 3 formatos físicos: SD, miniSD e microSD, sendo o microSD o predominante. Em termos de capacidade, e do sistema de ficheiros, eles encaixam-se nas seguintes familias: SDSC (de 1MB a 2GB com FAT12/16), SDHC (de 2GB a 32GB, com FAT32), SDXC (de 32GB a 2TB, com exFAT) e SDUC (de 2TB a 128TB, com exFAT).

Em termos de velocidade, existem SD Cards em 6 velocidades disponíveis: DS (base) a 12,5MB/s, High-Speed (HS) a 25MB/s, UHS-I a 104MB/s, UHS-II a 312MB/s, UHS-III 624MB/s, e SD Express a 985MB/s. Todas as velocidades são suportadas por todas as familias SD Card exceto a SDSC que suporta apenas até HS.

A figura 1.4 mostra o aspeto de um cartão de memória - SD (esquerda) microSD (centro) e a identificação dos pinos no verso dos SD Cards (direita).

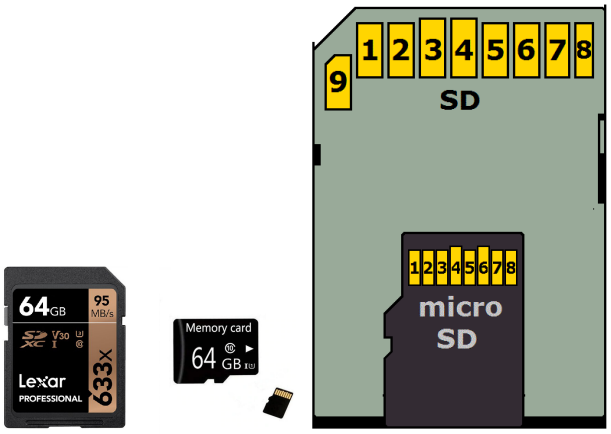

Figura 1.4: Micro-SDCard.

Em termos de funcionamento, os SD Cards diferenciam-se das memórias EEPROM pelo fato de organizarem a informação sob a forma de sistema de ficheiros. Isto permite para manter a compatibilidade com outros dispositivos, especialmente útil para transferir grandes volumes de informação. Os detalhes do sistema de ficheiros são normalmente ocultados ao utilizador, sendo geridos pelo sistema operativo. O sistema de ficheiros gere a memória do dispositivo de armazenamento de forma a organizar em ficheiros e pastas.

[1] em `https://www.sdcard.org/`

Pino SD / microSD	Nome SPI / One-bit	Descrição SPI / One-bit
1 / 2	nCS / CD	Selecão / Deteção
2 / 3	MOSI / CMD-RSP	Entrada / Comando-Resposta
3 / -	GND	Alimentação Negativo
4 / 4	VDD	Alimentação Positivo
5 / 5	CLK	Sincronismo
6 / 6	GND	Alimentação Negativo
7 / 7	MISO / DAT0	Saída / Dados
8 / 8	nIRQ	Interrupção
9 / 1	N/C	Não é usado

Tabela 1.1: Correspondência entre os pinos dos cartões SD e os modos de ligação

Um dispositivo de armazenamento de dados não volátil (disco rígido, SD Card, etc.) é conhecido como um volume. Um volume pode ter uma ou mais partições lógicas. Cada partição tem um sistema de ficheiros associado. Num disco rígido de um computador é possível ter várias partições, cada uma com um sistema de ficheiros diferente, no entanto num sistema embebido não existe essa necessidade e é suficiente ter apenas uma partição.

Os sistemas de ficheiros FAT12/16/32 têm o seu nome derivado de *File Allocation Table*, pois os ficheiros e diretórios são organizados numa tabela. O número corresponde ao bits usados para indexar a tabela. Nesta tabela ficam registados os *cluters* ocupados por cada ficheiro. Os *clusters* são as unidade de armazenamento de ficheiro. O nome FAT16 deriva do facto de suportar apenas 16 bits para indexar *clusters*, ou seja, existem apenas 65525 *clusters*. A FAT16 suporta dispositivos com capacidade de armazenamento até 2GB, o que leva ao uso de *clusters* de 32kB.

No caso do Arduíno, onde não existe um sistema operativo como o Linux ou o Windows, é necessário que o Sketch trate não só da interface física, mas como também de toda a parte lógica. A nível de programação, pretende-se usar uma biblioteca que ofereça as seguintes funcionalidades:

- Listar ficheiros num diretório;
- Criar e modificar um ficheiro;
- Eliminar um ficheiro;
- Criar e eliminar diretórios e sub-diretórios.

Os SD Cards suportam 4 modos de ligação: `SPI`, `One-bit`, `Four-bit` e `SD UHS-II`. Os modos `SPI` e `One-bit` devem ser suportados obrigatoriamente. A tabela 1.1 mostra a correspondência entre o pino e a sua funcionalidade.

Em termos de ligações ao Arduíno, o módulo faculta a adaptação de níveis de tensão, dos 5V do Arduíno para os 3,3V do cartão de memória.

Figura 1.5 mostra uma placa com suporte para SDCard para ligação a um Arduíno.

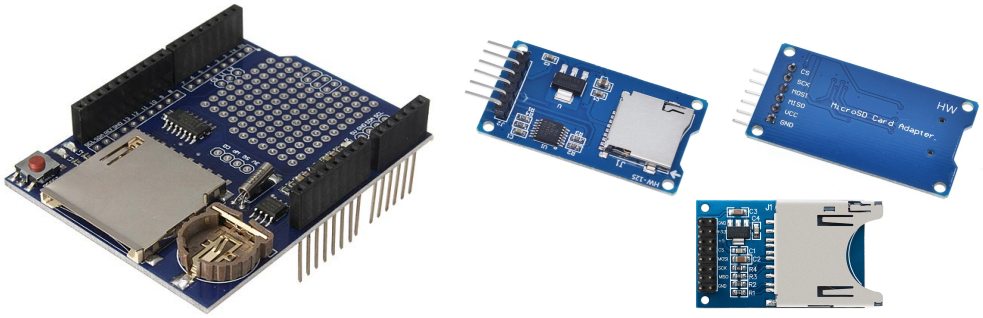

Figura 1.5: Shield SD Card (esquerda) e placas adaptadoras micro SD e SD Card.

Figura 1.6 mostra as ligações entre o cartão SD-Card e um Arduíno Uno.

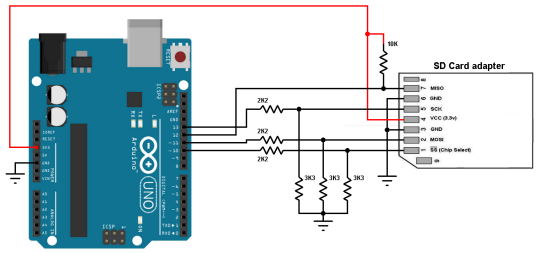

Figura 1.6: Esquema shield SD Card.

Para acelerar o desenvolvimento da demonstração do SD Card, usou-se a biblioteca SD[2], baseada na biblioteca sdfatlib[3]. Esta biblioteca suporta FAT16 e FAT32 e cartões até SDHC, e nomes no formato 8.3 - "FILE001.TXT" 8 caracteres para o nome e 3 caracteres para a extensão (ou tipo) do ficheiro. Se existirem nomes de ficheiros com mais de 8 carateres, eles são convertidos. Por exemplo, o ficheiro com o nome "novo ficheiro.txt" é convertido para "NOVOFI~1.TXT" A comunicação é feita através da interface SPI (pinos 4/10, 11, 12, e 13). O sinal de seleção varia para diferentes implementações e Shields, sendo ocasionalmente o pino 4 ou o 10.

Esta biblioteca tem vários exemplos de demonstração, recomendando-se o estudo dos exemplos `Card Info` e `Datalogger`. O primeiro exemplo permite saber se o Arduíno consegue aceder ao SD Card, ainda sem ler ou escrever ficheiros, e mostrar a informação acerca do cartão de memória. Antes de correr este exemplo é necessário que o esteja um cartão no suporte do cartão e que esteja formatado com FAT12 ou FAT16.

[2] `https://www.arduino.cc/en/Reference/SD`
[3] `http://code.google.com/p/sdfatlib/`

A listagem 1.3 contém um Sketch baseado no exemplo `Card Info` para testar o cartão de memória. A iniciação das acções do SD Card envolvem a chamada da função `card.init(velocidade SPI, pino de ativação do cartão)` que trata de iniciar a interface e detetar a presença de um cartão de memória. A função `volume.init(card)` lê a informação do volume do cartão, e verifica se ela contém uma partição do tipo FAT12 ou FAT16. Caso tenha uma partição válida, é lida a sua informação e é obtida a posição inicial da raíz da partição com a função `root.openRoot(volume);`. A função `root.ls(LS_R | LS_DATE | LS_SIZE);` é usada para listar os ficheiros, caso existam.

Listagem 1.3: Listagem SDCard-Info.ino

```
/*
  SD card test

  This example shows how use the utility libraries on which the'
  SD library is based in order to get info about your SD card.
  Very useful for testing a card when you're not sure whether its working or not.

  The circuit:
    SD card attached to SPI bus as follows:
 ** MOSI - pin 11 on Arduino Uno/Duemilanove/Diecimila
 ** MISO - pin 12 on Arduino Uno/Duemilanove/Diecimila
 ** CLK - pin 13 on Arduino Uno/Duemilanove/Diecimila
 ** CS - depends on your SD card shield or module.
       Pin 4 used here for consistency with other Arduino examples


  created  28 Mar 2011
  by Limor Fried
  modified 9 Apr 2012
  by Tom Igoe
*/
// include the SD library:
#include <SPI.h>
#include <SD.h>

// set up variables using the SD utility library functions:
Sd2Card card;
SdVolume volume;
SdFile root;

// change this to match your SD shield or module;
// Arduino Ethernet shield: pin 4
// Adafruit SD shields and modules: pin 10
// Sparkfun SD shield: pin 8
// MKRZero SD: SDCARD_SS_PIN
const int chipSelect = 4;

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(115200);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }


  Serial.print("\nInitializing SD card...");
#if 0
  // we'll use the initialization code from the utility libraries
  // since we're just testing if the card is working!
  if (!card.init(SPI_HALF_SPEED, chipSelect)) {
    Serial.println("initialization failed. Things to check:");
    Serial.println("* is a card inserted?");
    Serial.println("* is your wiring correct?");
    Serial.println("* did you change the chipSelect pin to match your shield or module?");
```

```
    while (1);
  } else {
    Serial.println("Wiring is correct and a card is present.");
  }

  // print the type of card
  Serial.println();
  Serial.print("Card type:         ");
  switch (card.type()) {
    case SD_CARD_TYPE_SD1:
      Serial.println("SD1");
      break;
    case SD_CARD_TYPE_SD2:
      Serial.println("SD2");
      break;
    case SD_CARD_TYPE_SDHC:
      Serial.println("SDHC");
      break;
    default:
      Serial.println("Unknown");
  }

  // Now we will try to open the 'volume'/'partition' - it should be FAT16 or FAT32
  if (!volume.init(card)) {
    Serial.println("Could not find FAT16/FAT32 partition.\nMake sure you've formatted the card");
    while (1);
  }

  Serial.print("Clusters:          ");
  Serial.println(volume.clusterCount());
  Serial.print("Blocks x Cluster:  ");
  Serial.println(volume.blocksPerCluster());

  Serial.print("Total Blocks:      ");
  Serial.println(volume.blocksPerCluster() * volume.clusterCount());
  Serial.println();

  // print the type and size of the first FAT-type volume
  uint32_t volumesize;
  Serial.print("Volume type is:    FAT");
  Serial.println(volume.fatType(), DEC);

  volumesize = volume.blocksPerCluster();    // clusters are collections of blocks
  volumesize *= volume.clusterCount();       // we'll have a lot of clusters
  volumesize /= 2;                           // SD card blocks are always 512 bytes (2 blocks are 1KB)
  Serial.print("Volume size (Kb):  ");
  Serial.println(volumesize);
  Serial.print("Volume size (Mb):  ");
  volumesize /= 1024;
  Serial.println(volumesize);
  Serial.print("Volume size (Gb):  ");
  Serial.println((float)volumesize / 1024.0);

  Serial.println("\nFiles found on the card (name, date and size in bytes): ");
  root.openRoot(volume);

  // list all files in the card with date and size
  root.ls(LS_R | LS_DATE | LS_SIZE);

#endif
  SD.begin(4);
  if (SD.exists("TESTE.TXT")) {
    Serial.println("TESTE.txt exists.");
  }
  SD.rmdir("ToT");
  SD.remove("TESTE.TXT");
}

void loop(void) {
}
```

Esta biblioteca oferece as seguintes funções, destinadas a manipular fi-

cheiros e diretórios:

- SD.begin() - iniciação da biblioteca e do SD Card
- SD.exists(nome) - testa se um ficheiro ou diretório existe com o nome passado como argumento. Devolve `true` se o ficheiro ou diretório existirem, `false` caso contrário;
- SD.mkdir(nome) - cria um diretório como nome passado por argumento. Devolve `true` se o diretório for criado corretamente, `false` caso contrário;
- SD.open(nome) - abre um ficheiro com o nome, para leitura ou escrita. Devolve `true` se o ficheiro for aberto corretamente, `false` caso contrário;
- SD.remove(nome) - remove o ficheiro. Devolve `true` se o ficheiro for removido corretamente, `false` caso contrário;
- SD.rmdir(nome) - remove o diretório. Devolve `true` se o diretório for removido corretamente, `false` caso contrário;

A listagem 1.4, baseada no exemplo `Datalogger` demonstra a escrita de um ficheiro num SD Card. Basicamente, as acções para escrever uma sequência de bytes após iniciar o SD Card são:

- Abrir o ficheiro em modo de escrita - `File dataFile = SD.open("datalog.txt", FILE_WRITE);`
- Escrever os dados no ficheiro - `dataFile.println(dataString);`
- Fechar o ficheiro - `dataFile.close();`

A abertura de um ficheiro pode ser de diferentes tipos, e é definido no segundo parâmetro da chamada da função `open()`:

- FILE_READ - Leitura do ficheiro.
- FILE_WRITE - Escrita com concatenação ao conteúdo anterior. Se o ficheiro não existir o ficheiro é criado.

Para se escrever num ficheiro sem manter o seu conteúdo anterior é necessário apagar o ficheiro.

Listagem 1.4: Listagem SDCard-fileWrite.ino

```
/*
  SD card test

  This example shows how use the utility libraries on which the'
  SD library is based in order to get info about your SD card.
  Very useful for testing a card when you're not sure whether its working or not.
```

```
  The circuit:
    SD card attached to SPI bus as follows:
 ** MOSI - pin 11 on Arduino Uno/Duemilanove/Diecimila
 ** MISO - pin 12 on Arduino Uno/Duemilanove/Diecimila
 ** CLK - pin 13 on Arduino Uno/Duemilanove/Diecimila
 ** CS - depends on your SD card shield or module.
       Pin 4 used here for consistency with other Arduino examples


  created  28 Mar 2011
  by Limor Fried
  modified 9 Apr 2012
  by Tom Igoe
*/
// include the SD library:
#include <SPI.h>
#include <SD.h>

// set up variables using the SD utility library functions:
Sd2Card card;
SdVolume volume;
SdFile root;

// change this to match your SD shield or module;
// Arduino Ethernet shield: pin 4
// Adafruit SD shields and modules: pin 10
// Sparkfun SD shield: pin 8
// MKRZero SD: SDCARD_SS_PIN
const int chipSelect = 4;

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(115200);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }


  Serial.print("\nInitializing SD card...");
#if 0
  // we'll use the initialization code from the utility libraries
  // since we're just testing if the card is working!
  if (!card.init(SPI_HALF_SPEED, chipSelect)) {
    Serial.println("initialization failed. Things to check:");
    Serial.println("* is a card inserted?");
    Serial.println("* is your wiring correct?");
    Serial.println("* did you change the chipSelect pin to match your shield or module?");
    while (1);
  } else {
    Serial.println("Wiring is correct and a card is present.");
  }

  // print the type of card
  Serial.println();
  Serial.print("Card type:         ");
  switch (card.type()) {
    case SD_CARD_TYPE_SD1:
      Serial.println("SD1");
      break;
    case SD_CARD_TYPE_SD2:
      Serial.println("SD2");
      break;
    case SD_CARD_TYPE_SDHC:
      Serial.println("SDHC");
      break;
    default:
      Serial.println("Unknown");
  }

  // Now we will try to open the 'volume'/'partition' - it should be FAT16 or FAT32
  if (!volume.init(card)) {
    Serial.println("Could not find FAT16/FAT32 partition.\nMake sure you've formatted the card");
```

```
    while (1);
  }

  Serial.print("Clusters:          ");
  Serial.println(volume.clusterCount());
  Serial.print("Blocks x Cluster:  ");
  Serial.println(volume.blocksPerCluster());

  Serial.print("Total Blocks:      ");
  Serial.println(volume.blocksPerCluster() * volume.clusterCount());
  Serial.println();

  // print the type and size of the first FAT-type volume
  uint32_t volumesize;
  Serial.print("Volume type is:    FAT");
  Serial.println(volume.fatType(), DEC);

  volumesize = volume.blocksPerCluster();     // clusters are collections of blocks
  volumesize *= volume.clusterCount();        // we'll have a lot of clusters
  volumesize /= 2;                            // SD card blocks are always 512 bytes (2 blocks are 1KB)
  Serial.print("Volume size (Kb):  ");
  Serial.println(volumesize);
  Serial.print("Volume size (Mb):  ");
  volumesize /= 1024;
  Serial.println(volumesize);
  Serial.print("Volume size (Gb):  ");
  Serial.println((float)volumesize / 1024.0);

  Serial.println("\nFiles found on the card (name, date and size in bytes): ");
  root.openRoot(volume);

  // list all files in the card with date and size
  root.ls(LS_R | LS_DATE | LS_SIZE);

#endif
  SD.begin(4);
  if (SD.exists("TESTE.TXT")) {
    Serial.println("TESTE.txt exists.");
  }
  SD.rmdir("ToT");
  SD.remove("TESTE.TXT");
}

void loop(void) {
}
```

1.4 Rede Ethernet

A rede Ethernet é uma forma A forma de ligar vários computadores em rede é através do protocolo Ethernet. Este tipo de redes suporta vários tipos de ligações

Ligação a rede de comunicações é fundamental para suportar a comunicação para sistemas embebidos para IoT.

conceitos:

1. cabo ethernet + endereço MAC
2. endereço IP
3. ligação TCP + UDP (porto)
4. HTTP
5. HTML / MQQT

Resolução de nomes

w5100 , 4 sockets em paralelo

A ligação com fio

Arduino Wiznet Ethernet shield.

Abrir o exemplo em File→Examples→Ethernet→WebServer

Este exemplo cria uma servidor de páginas web que mostra o valor das entradas analógicas. Este servidor tem o endereço IP 192.168.1.177, e recebe as ligações dos clientes no porto 80.

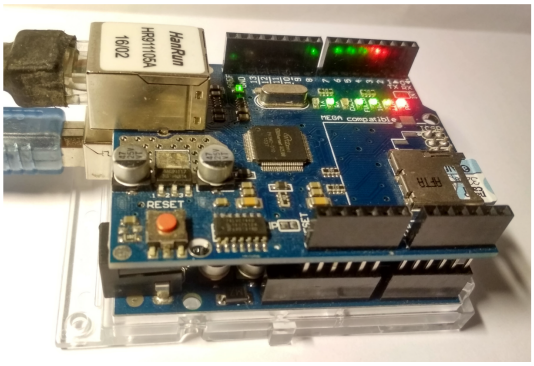

Figura 1.7: Modulo ethernet.

Listagem 1.5: Teste de presença na rede através do comando ping

```
C:\Users\R>ping 192.168.1.177

Pinging 192.168.1.177 with 32 bytes of data:
Reply from 192.168.1.177: bytes=32 time=3ms TTL=128
Reply from 192.168.1.177: bytes=32 time=1ms TTL=128
Reply from 192.168.1.177: bytes=32 time=1ms TTL=128
Reply from 192.168.1.177: bytes=32 time=1ms TTL=128

Ping statistics for 192.168.1.177:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 3ms, Average = 1ms
```

A figura 1.8 mostra o conteúdo do terminal após atender pedidos de ligação de um cliente () num computador ligado na rede do Arduíno.

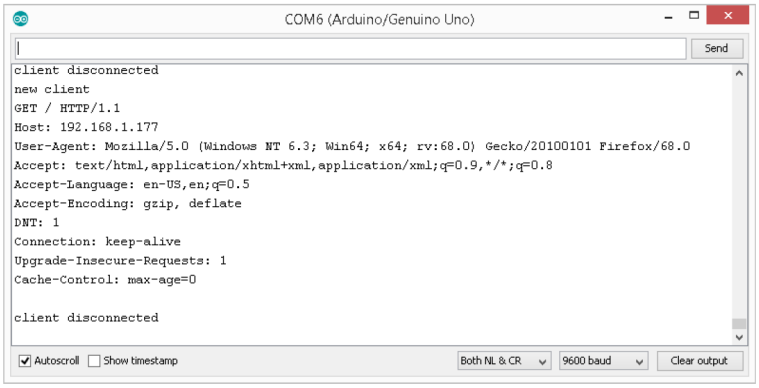

Figura 1.8: ethernet-terminal.

A figura 1.12 mostra dois navegadores de Internet abertos no endereço 192.168.1.177, onde um mostra a página HTML interpretada e o outro o código-fonte HTML.

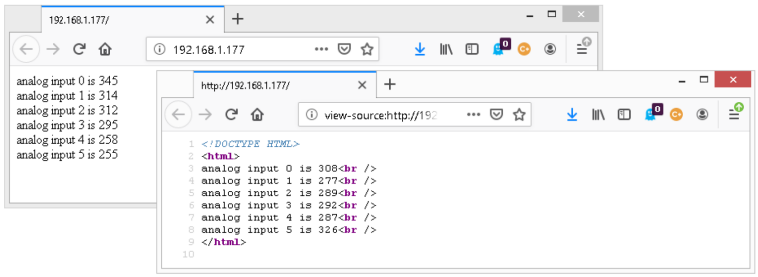

Figura 1.9: Código HTML enviado pelo Arduíno e o seu aspeto quando traduzido por um navegador de internet.

É possível programar qualquer conteúdo HTML desde que exista memória no Arduíno para guardar todo o programa. Como um Arduíno tem pouca memória disponível, ele não tem capacidade de guardar imagens na sua memória, no entanto é possível fazer uso do objetos noutras páginas Web ou usar serviços do tipo *Content Delivery Network* (CDNs) para alojar imagens e *scripts*, para que o código HTML no Arduíno apenas guarda o seu endereço. Por exemplo, para carregar um GIF animado poderia-se usar a listagem 1.6.

Listagem 1.6: Listagem para adicionar uma imagem animada na página gerada pelo servidor Web Arduíno

```
client.println("<p></p><img src=\"https://www.amazing-animations.com/"\
    "animations/media016.gif\" alt=\"bad tv\">");
client.println("</html>");
```

Uma alternativa a serviços online é o uso de um SD Card para armazenamento de figuras e scripts a carregar, permitindo também a possibilidade de registar dados de utilização. Neste caso o Sketch do servidor ao receber os pedidos (GET) do cliente tem de distinguir qual o objeto que está a ser pedido, a página principal `GET / HTTP/1.1` ou uma imagem `GET /ball.gif HTTP/1.1`.

Nota: o Ethernet-Shield escolhido para esta experiência é o W5100 da Sainsmart/Keyestudio. Este Shield inclui também a ligação ao um SD Card, no entanto ele não funciona tal como fornecido. O problema deste Ethernet-Shield é o circuito de adaptação da tensão de funcionamento dos 5 V para os 3,3 V do SD card, que foi bem feito. Observando o sinal no osciloscópio observa-se que o SD Card gera o sinal de saída (MISO) para a interface SPI, no entanto o nível 1 ou HIGH do sinal produzido pelo SD Card que deveria ser próximo de 5 V, fica em cerca de 2 V, o que não é suficiente pela ser discriminado como 1 pelo Arduíno.

A solução para este problema passa por intersectar o sinal de saída de dados (MISO), interrompendo a ligação da pista na PCB do Shield com um corte de X-acto ou bisturi e introduzir um adaptador de nível *ad-hoc*, ou então adquirir um Ethernet-Shield de outro fabricante.

1.5 WiFi

O comunicação sem fios é bastante usada para acesso à Internet, ou a uma rede privada, para publicar dados recolhidos de sensores. O fato dos Microcontrolador - *Micro-Controller Unit*s (MCUs) terem pouca capacidade de processamento mas vários sinais de E/S programáveis torna-os bastante atrativos para enviarem dados para a Internet (ou *Cloud*). Para

https://github.com/thinger-io/Arduino-Library

https://docs.thinger.io/quick-sart/devices/arduino

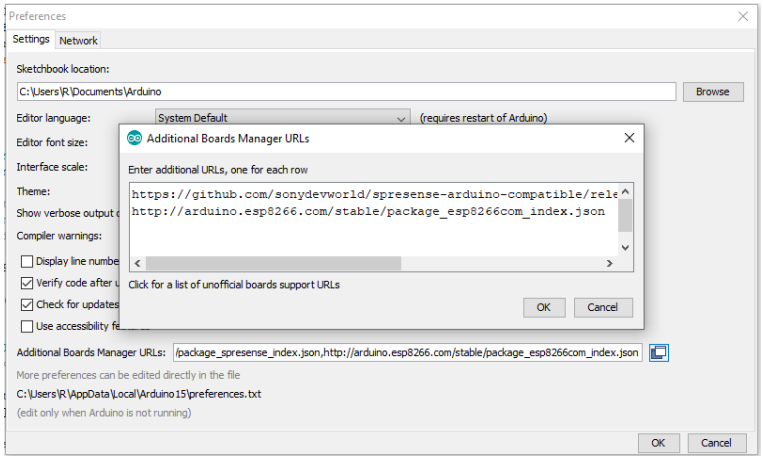

Figura 1.10: Código HTML enviado pelo Arduíno e o seu aspeto quando traduzido por um navegador de internet.

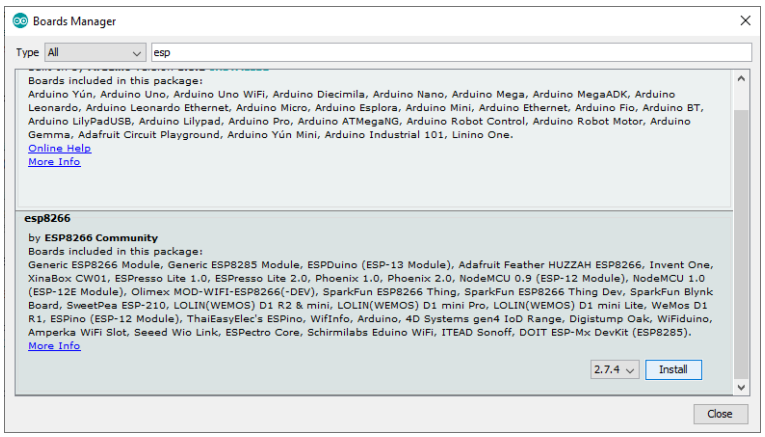

Figura 1.11: Janela de diálogo para adicionar a placa ESP8266.

Figura 1.12: Demonstração da comunicação com um servidor via Internet usando a placa ESP14.

1.6 GSM/GPRS

1.7 Bluetooth

A comunicação sem fios de custo alcance é uma funcionalidade bastante atrativa para sistemas embebidos. Com a disponibilidade da norma BT na maioria dos computadores, telemóveis e *tablets*, é possível comandar e receber informações de um sistema embebido sem ser necessário usar uma interface local ou uma ligação física a outro sistema.

Como os circuitos para comunicação sem fios são difíceis de realizar e os protocolos complexos, a melhor opção é usar um módulo que ofereça uma interface simples de ligar e programar. O módulo BT HC-6 trata de toda a interface rádio do protocolo BT, no entanto ele não é autónomo, é necessário sempre usar um sistema embebido para o controlar.

A figura 1.13 mostra o aspecto do módulo BT (esquerda) e a identificação dos pinos (direita). As principais caraterísticas deste módulo são:

- Protocolo Bluetooth v.2.0,
- Banda rádio: 2.40GHz-2.48GHz (com saltos de frequências para reduzir interferência),
- Tensão de funcionamento: 3,3-6V,
- Temperatura de funcionamento: -20^{o}C a +55^{o}C,
- Consumo de corrente: 40mA

A figura 1.14 mostra as ligações do módulo BT para um Arduíno Uno (esquerda) e Mega (direita). Como no Arduíno Uno a porta série é partilhada com o canal para programar o Arduíno, a solução passou por emular a porta serie em software, não fazendo uso da UART existente dentro do Arduíno. Esta solução permite ligar o modulo BT a outros pinos de E/S que não estejam a ser usados. O Arduíno Mega possui 4 portas série assíncronas, pelo que é possível ligar o módulo BT numa das outras 3 portas.

É necessário ter em atenção que embora o módulo seja alimentado a 5V, os sinais de comunicação série assíncrona funcionam a 3,3V! É necessário que os níveis de sinal entre o Arduíno e o HC-6 sejam os mesmos.

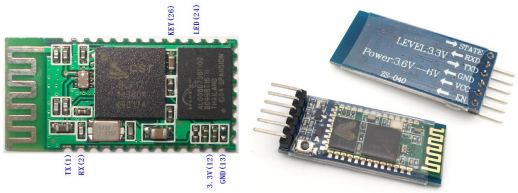

Figura 1.13: Modulo BT HC-6.

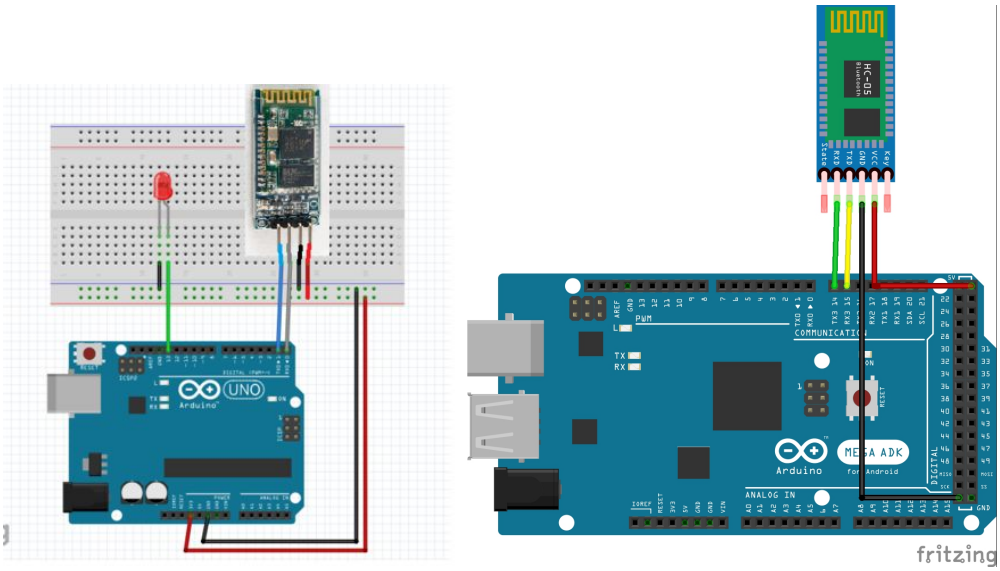

Figura 1.14: Esquema de ligações ao Arduíno Uno (esquerda) e Mega (direita).

Antes de introduzir o módulo numa aplicação, a sugestão é usar um Sketch para configurar o módulo e depois introduzi-lo no sistema completo. Isto é bastante útil especialmente para testar os comandos AT. O comando mais básico é simplesmente "AT", ao que o módulo responde "OK". O comando "AT+VERSION" permite identificar o módulo. Neste caso a resposta foi "OKlinvorV1.8". A tabela 1.2 mostra os comandos AT e o ritmo de transmissão.

Comando	Ritmo de Transmissão
AT+BAUD1	1200
AT+BAUD2	2400
AT+BAUD3	4800
AT+BAUD4	9600 (de fábrica)
AT+BAUD5	19200
AT+BAUD6	38400
AT+BAUD7	57600
AT+BAUD8	115200
AT+BAUD9	230400
AT+BAUDA	460800
AT+BAUDB	921600
AT+BAUDC	1382400

Tabela 1.2: Configuração do ritmo de transmissão do módulo BT.

Para alterar o nome do módulo no processo de descoberta usa-se o co-

mando `AT+NAMEnovo_nome`, ao que a resposta do módulo deverá ser: `OKnovo_nome`. Por exemplo, o comando `AT+NAMEbt-duino`, produzirá a resposta `OKbt-duino`. Para mudar o código de acesso para um novo valor com 4 dígitos (xxxx) usa-se o comando `AT+PINxxxx`. A resposta enviada pelo módulo é: `OKsetpin`. Por exemplo, o comando `AT+PIN1001`, produzirá a resposta `OKsetpin`.

Do lado do computador ou do telemóvel é necessário ativar a comunicação Bluetooth e procurar o nome do módulo pelos nomes "HC-06" e "SPP-CA". Aquando do emparelhamento é pedido o código de acesso. O código de fábrica é "1234". A figura 1.15 apresenta a lista de dispositivos BT emparelhados com o computador usado nos ensaios. Com o HC-6 emparelhado é criado um canal série no computador onde todos os dados enviados serão recebidos na porta série do Arduíno onde o módulo BT se encontra ligado. Este canal é bidirecional.

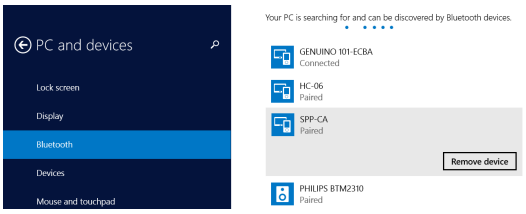

Figura 1.15: Listagem de componentes BT emparelhados com o computador.

1.8 LORA

Figura 1.16: Modulo LORA.

Campo	Cabeçalho	Dimensão	Tipo Tag	Número Série	XOR	Fim
Dimensão	1 byte	1 byte	1 byte	4 bytes	1 byte	1 byte
Valor	0x02	0x09	0x01-0x51			0x03

Tabela 1.3: Estrutura da trama enviada pelo modulo RFID LF 7941E.

1.9 Leitor RFID

As tecnologias *Radio Frequency IDentification* (RFID), NFC e MIFARE servem para comunicação com componentes ou sistemas na proximidade, e foram apresentadas na secção **??**. Esta secção exemplifica a utilização de um módulo RFID com um Arduíno. Das várias tecnologias disponíveis, foram escolhidas duas tecnologias RFID *Low Frequency* (125 kHz) e RFID *High Frequency* (13,56 MHz) RFID MIFARE por serem das mais usadas em sistemas de distribuição, segurança e controlo de acessos.

As *tags* RFID LF foram a primeira implementação RFID, e por isso são mais simples e apenas permitem ser lidas. As *tags* RFID HF permitem não só serem lidas como também escritas para incorporar informação adicional para além do seu identificador.

Para dotar um sistema embebido com esta tecnologia, existem vários controladores disponíveis, por exemplo: 7941E, RC552 ou PN532. Para demonstrar esta tecnologia usaram-se os módulos 7941E e MFRC522 da NXP, e que permitem comunicar nos 125 kHZ e 13,56 MHz, respetivamente.

1.9.1 125 kHz - 7941E

Este módulo RFID funciona na frequência de 125 kHz e envia uma sequência de bytes sempre que conseguir ler o identificador de uma *tag* RFID. Após obter o identificador (4 bytes), o módulo envia a informação sob a forma de uma trama com o formato ilustrado na tabela 1.3.

A interface com o módulo 7941E é feita através de uma interface série assíncrona a funcionar a 9600 baud, sendo possível usar também uma interface Wiegand (muito usado em sistemas de segurança). O módulo, na figura 1.19, funciona com tensões entre 3,3 V e 5 V e o MCU responsável pelo seu controlo é o STM8S003F3 (16 MHz).

No Sketch de demonstração do módulo, na listagem 1.7, o Arduíno aguarda a recepção de uma sequência de caracteres na porto série assíncrono e compara a sequência recebida com as autorizadas. Após a avaliação da comparação, o programa produz a autorização para as chaves registadas.

Nesta demonstração, como a *Universal Asynchronous Receiver/Transmitter* (UART) do Arduíno está dedicada à comunicação com o computador, foi necessário adicionar um mecanismo adicional para suportar a recepção do módulo RFID. Enquanto que a UART é um componente que se encontra

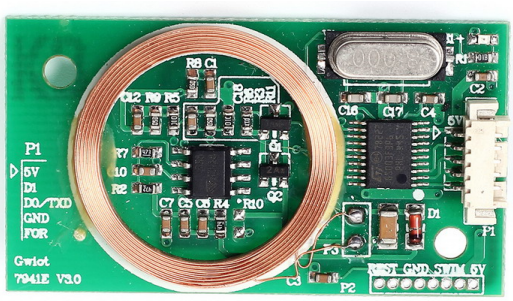

Figura 1.17: Módulo RFID LF 7941E.

embebido no próprio MCU, o facto do ATMega328P, existente no Arduíno Uno, apenas ter um porto faz com que seja necessário criar outro em software. Para além do leitor ligado ao Arduíno Uno, existe também um beeper ligado no pino 12. A figura 1.19 mostra o circuito com o módulo leitor de tags RFID e o *beeper*.

Figura 1.18: Circuito de demonstração do Módulo RFID LF 7941E.

Foi definida a classe RFID7941E que recebe como parâmetros os pinos da UART por software (objecto SwSerial) e tem os métodos públicos: `init()` para configurar o objecto `SwSerial`, `scanTags()` para verificar se existem carateres recebidos no objeto `SwSerial` e nesse caso prosseguir com a recolha dos bytes enviados pelo módulo RFID. Espera-se que o módulo envie o próximo byte dentro de um determinado limite. Caso esse tempo seja excedido é assumido que o identificador do cartão foi recebido. O método `isValid()` verifica o campo XOR da chave recebida, o método `isEquals()` compara o valor da chave recebida com outra chave e como valor de retorno indica se são iguais.

Foi criado uma estrutura adicional para ilustrar o uso de uma lista de controlo de acessos com o registo do nome dos utilizadores e chave associada.

A figura 1.20 mostra a aplicação do osciloscópio com o sinal da saída série assíncrona, e a descodificação do sinal, após a leitura de um cartão RFID LF. É possível identificar a sequência de bytes recebidos:

$0x02, 0x0A, 0x02, 0x01, 0x06, 0x48, 0x81, 0x51, 0x94, 0x03$.

A figura 1.21 mostra o aspeto do aparato experimental usado no ensaio do módulo RFID 7941E. O conteúdo da janela do terminal de texto encontra-se na figura 1.22.

Listagem 1.7: Sketch de demonstração do módulo RFID LF 7941E

```
#include <SoftwareSerial.h>

/* RFID 125 kHz module - 7941E
```

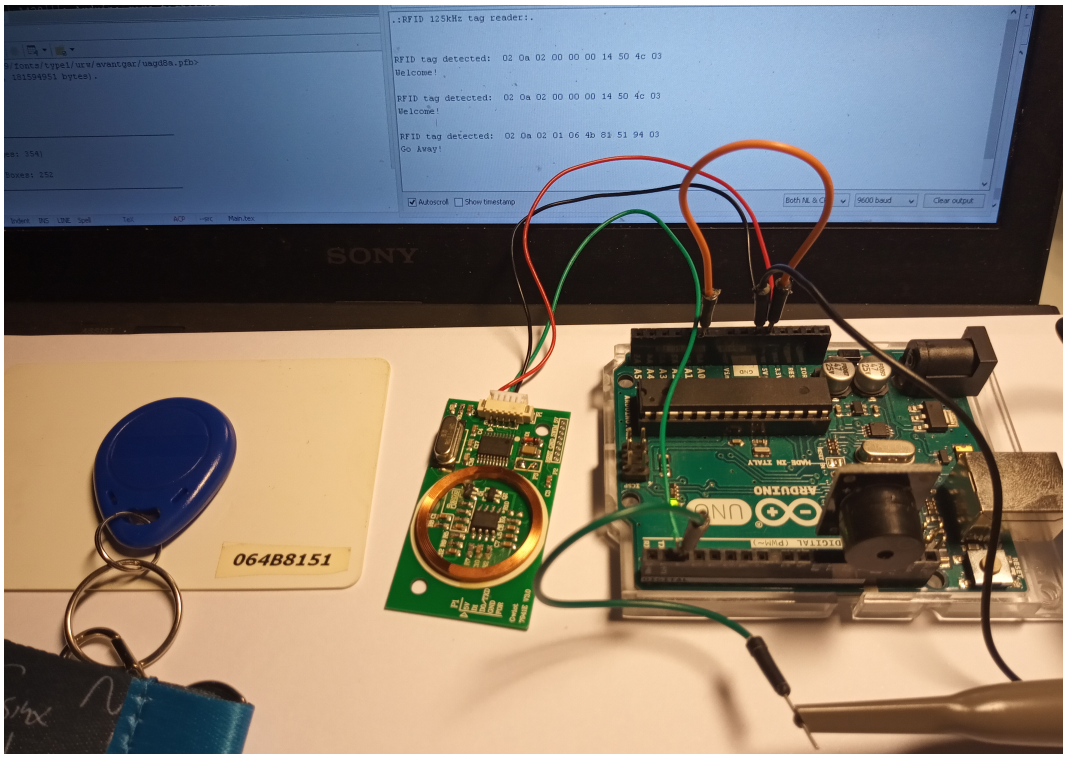

Figura 1.19: Aparato usado na demonstração do Módulo RFID LF 7941E.

```
 *
byte 1: start = 0x02
byte 2: length = 0x09
byte 3: card type =
    0x01 MIFARE 1K
    0x02 EM4100
    0x03 MIFARE 4K
    0x10 HID Card
    0x11 T5567
    0x20 2nd Card
    0x21 ISO14443B
    0x22 FELICA
    0x30 15693 Label
    0x50 CPU Card
    0x51 Sector Information

byte 4-8: tag ID
byte 9: XOR bytes 2-8
byte 10: final = 0x03
*/

#define RFID125_HEADER      0
#define RFID125_LENGTH      1
#define RFID125_TYPE        2
#define RFID125_ID_START    3
#define RFID125_ID_END      7
#define RFID125_CHECKSUM    8
#define RFID125_TAIL        9

#define RFID_TAG_LEN       10

#define TIMEOUT       400

#define BEEPER_PIN     12
```

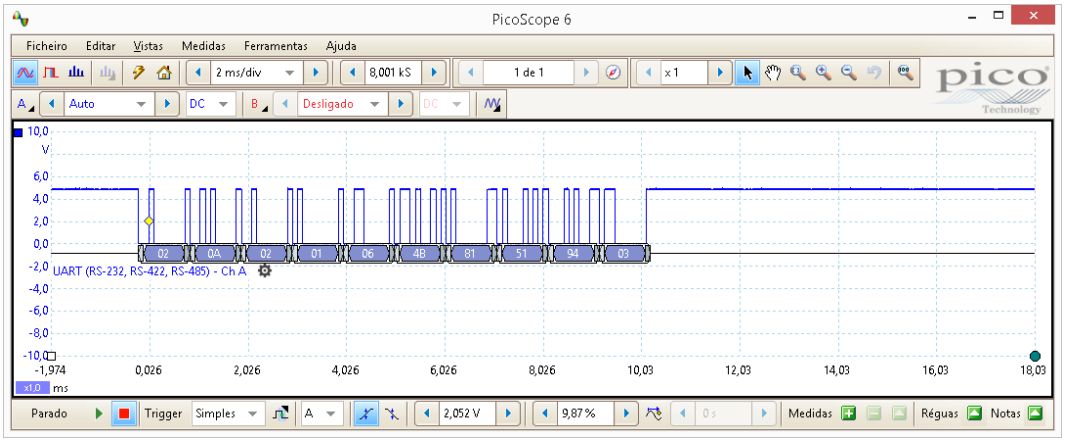

Figura 1.20: Sinal série assíncrono gerado pelo módulo RFID 7941E, e sua descodificação, após a identificação de uma *tag* RFID LF.

```
class RFID7941E {
  private:
    byte txPin, rxPin;
    SoftwareSerial* SwSerial;
    byte tagID[RFID_TAG_LEN];
    unsigned long lastScanTime;
    byte byteCount;
    bool newTag;

  public:
    RFID7941E(int rxPin, int txPin) { // constructor
      this->rxPin = rxPin;
      this->txPin = txPin;
      SwSerial = new SoftwareSerial(rxPin, txPin);
    }
    ~RFID7941E() { // destructor
      delete SwSerial;
    }
    bool init() {
      SwSerial->begin(9600); // Arduino-RFID reader
      while(!SwSerial);
    }
    bool scanTags() {
      if (SwSerial->available() > 0)    {
          if(millis() - lastScanTime > TIMEOUT) {
            // 100ms timeout = new card
            byteCount = 0;
            Serial.print("\nRFID tag detected: ");
          }
          tagID[byteCount] = SwSerial->read();
          byteCount++;
          if(byteCount > RFID_TAG_LEN) {
            return false; // something's wrong
          }
          lastScanTime = millis();
          newTag = true;
      }
      if((newTag == true) && (millis() - lastScanTime > TIMEOUT)) {
        newTag = false;
        return true;
      }
      return false;
    }
    void getLastTag(byte *tag) {
      byte i;
      for(i = 0; i < RFID_TAG_LEN; i++) {
        tag[i] = tagID[i];
```

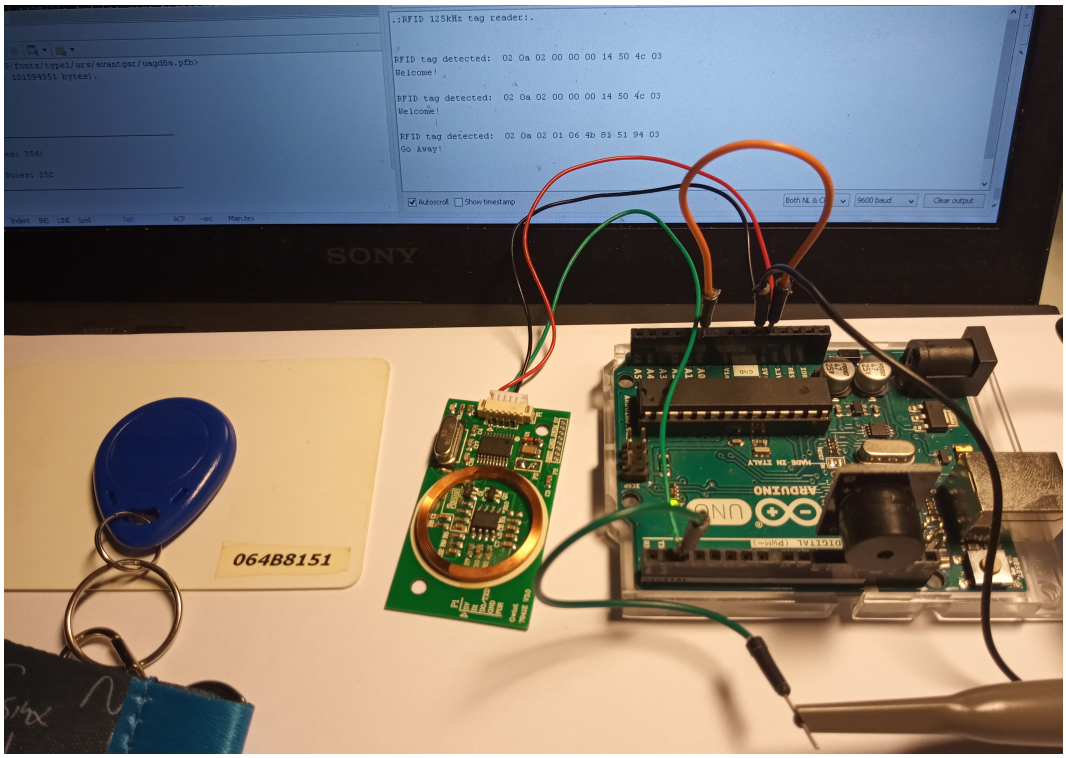

Figura 1.21: Aparato experimental com o módulo RFID LF 7941E.

```
    }
  }
  bool isValid() {
    int i;
    int x = 0;
    for(i = 1; i < 8; i++) {
      x ^= tagID[i];
    }
    if (x == tagID[RFID125_CHECKSUM]) {
      return true;
    } else {
      return false;
    }
  }
  bool isEquals(byte *tag) {
    byte i;
    for(i = 0; i < RFID_TAG_LEN; i++) {
      if(tag[i] != tagID[i]) return false;
    }
    return true;
  }
};


RFID7941E rfidTagReader(2, 3); // only 2 is connected

byte authID[RFID_TAG_LEN] = {0x02, 0x0a, 0x02, 0, 0, 0, 0x14, 0x50, 0x4c, 0x03 };

typedef struct {
  char *tagName;
  byte tagID[RFID_TAG_LEN];
} accRec_t  ;

accRec_t ar1 = { "Salustiano",  {0x02, 0x0a, 0x02, 0, 0, 0, 0x14, 0x50, 0x4c, 0x03 }};
```

```
accRec_t ar2 = { "Zulmira",     {0x02, 0x0a, 0x02, 0, 0, 0, 0x00, 0x00, 0x00, 0x03 }};
accRec_t ar3 = { "",            {0x00, 0x00, 0x00, 0, 0, 0, 0x00, 0x00, 0x00, 0x00 }};
accRec_t acl[] = { ar1, ar2, ar3 };


void beep() {
  tone(BEEPER_PIN, 1000); delay(100); noTone(12);
}

void beep_good() {
  tone(BEEPER_PIN, 500); delay(100); noTone(12);
  delay(50);
  tone(BEEPER_PIN, 2000); delay(100); noTone(12);
}

void beep_bad() {
  tone(BEEPER_PIN, 300); delay(1000); noTone(12);
}

void setup()
{
  Serial.begin(9600); // Arduino-PC
  pinMode(BEEPER_PIN, OUTPUT); // beeper
  rfidTagReader.init();
  beep();
  while(!Serial);
  Serial.println(".:RFID 125kHz tag reader:.\n");
}

bool newTagFound = false;
byte newTagID[RFID_TAG_LEN];
char i, aux[3]; // aux: 2 HEX chars + NULL

void loop()
{
  // ...
  if(rfidTagReader.scanTags() == true) {
    beep();
    newTagFound = true;
  }
  // ...
  if(newTagFound == true) {
    rfidTagReader.getLastTag(newTagID);
    for(i = 0; i < RFID_TAG_LEN; i++) {
      sprintf(aux, "%02x", newTagID[i]);
      Serial.print(aux);
    }
    Serial.println(" ");
    if((rfidTagReader.isValid() == true) && (rfidTagReader.isEquals(authID) )) {
      // alternative iterate over acl[] and find a match
      beep_good();
      Serial.println("Welcome!");
    }
    else {
      beep_bad();
      Serial.println("Go Away!");
    }
    newTagFound = false;
  }
}
```

1.9.2 13,56 MHz - MFRC522

Explicar a organização de memoria do cartão.

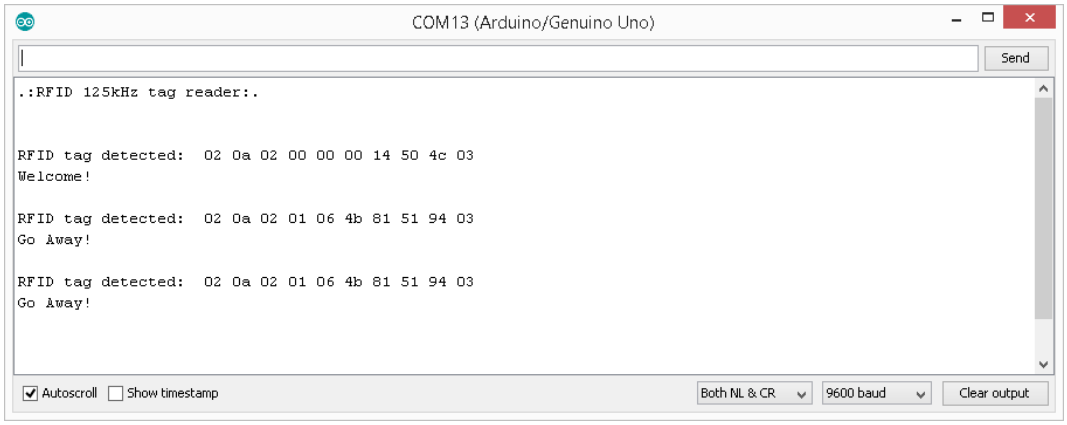

Figura 1.22: Conteúdo do terminal de texto após a execução do programa de demonstração e deteção de duas *tags* RFID LF.

Figura 1.23: Modulo RFID MFRC522.

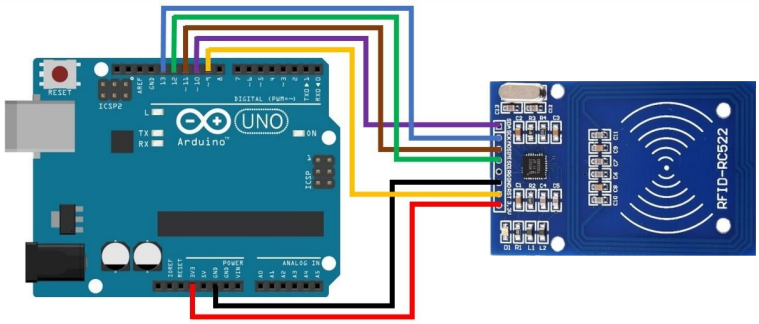

Figura 1.24: Modulo NFC.

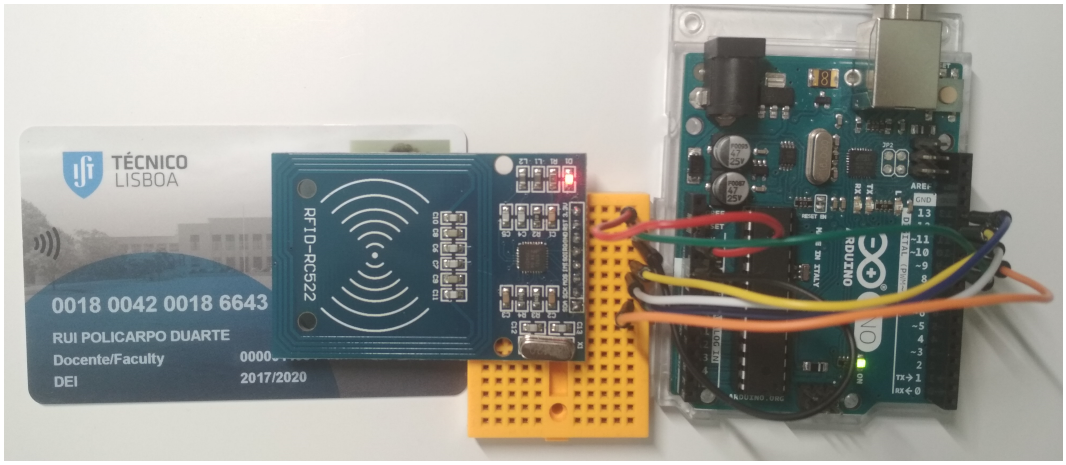

Figura 1.25: Montagem do módulo NFC numa breadboard para leitura de um cartão de identificação.

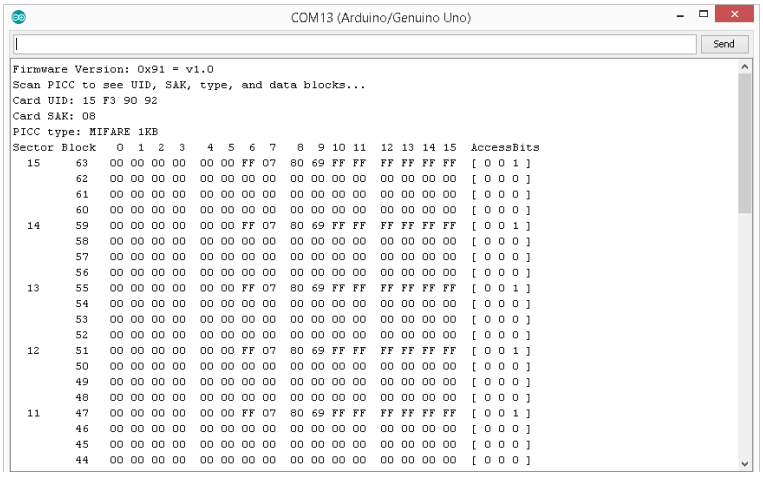

Figura 1.26: Modulo NFC.

1.10 Teclado Táctil Capacitivo

teclado que basta tocar sem pressionar.

Intro: Foto compoennte: Principio de funcionamento: Datasheet: TTP229 Ligações: Foto montagem e ensaio:

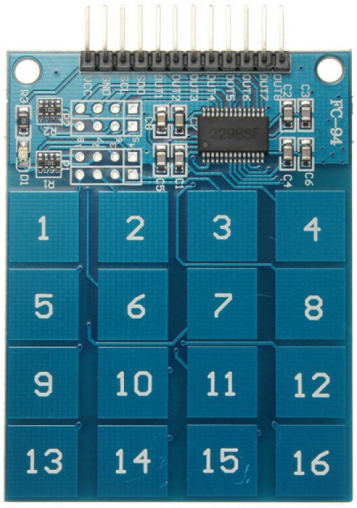

Figura 1.27: touch keypad

componentes: - 1x ttp229
datasheet url?

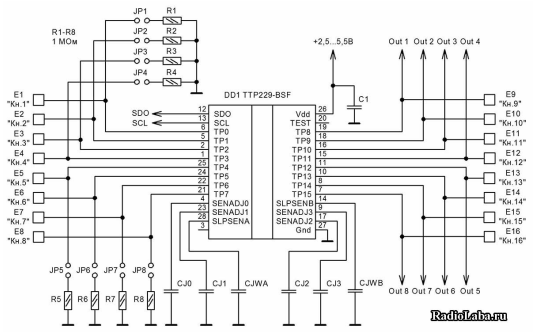

Figura 1.28: Esquema do touch keypad

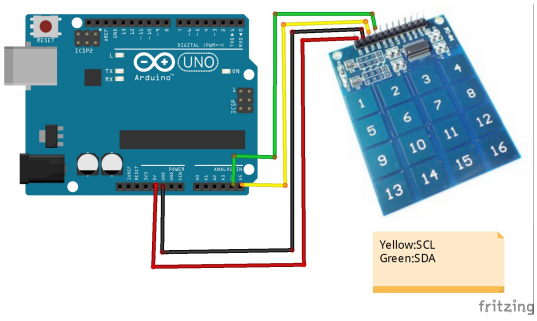

Figura 1.29: Ligação do touch keypad

1.11 Teclado de Computador PS/2

Os teclados de computador consistem numa matriz de teclas com os carateres usados frequentemente num idioma, muito semelhantes aos disponíveis nas antigas máquinas de escrever. Para reduzir o número de ligações às teclas e libertar o processador a fazer a sua avaliação, os teclados têm um controlador dedicado que dá indicação da tecla premida.

Os primeiros teclados de computador eram ligados via interface PS/2, e posteriormente passaram a ser ligados por ficha USB ou por RF (Bluetooth e 2,4 GHz). Por motivos de compatibilidade, alguns computadores ainda possuem ficha PS/2, ou admitem que teclados PS/2 sejam ligados a uma ficha USB mediante um adaptador. Existem também teclados USB que quando ligados a uma interface PS/2 cumprem com o protocolo antigo. Contudo nem todos os teclados USB funcionam com o protocolo PS/2. A figura 1.30 mostra o aspeto de um teclado de computador com ficha PS/2.

A vantagem de usar um teclado PS/2 num sistema embebido é a possibilidade de introduzir dados mais facilmente e ter menor custo de desenvolvimento de um teclado de matriz de botões de pressão. O teclado recebe comandos do sistema embebido e envia códigos de teclas premidas para o sistema embebido. A interface PS/2 é do tipo série síncrona, com um sinal de dados e outro relógio, o que é simples de ser programar num Arduíno para receber dados sobre as teclas premidas pelo utilizador. A geração do sinal de relógio e dados é feita pelo emissor, no entanto enquanto o sinal de relógio não estiver a VCC, o teclado assume que o computador está a enviar dados e não transmite.

Figura 1.30: ps2 keyboard

A interface física PS/2 do teclado é bidirecional e do tipo "coletor aberto" (*open colector/drain*) e não existe *pull-up* no teclado, ou seja o sistema embebido ou o computador deve ter o sinal normalmente a VCC (1) e o teclado faz apenas a ativação da ligação a GND para dar indicação de bit com valor 0. Os teclados funcionam a 5 V, pelo que se forem ligados a um sistema a funcionar a 3,3 V devem haver preocupação em respeitar os níveis máximos

de tensão de cada componente. A figura 1.31 mostra as ligações num teclado com ficha PS/2 e USB (esquerda) e as ligações a realizar para ligar a um Arduíno Uno (direita). Na ficha USB, VCC e GND ligam à alimentação do circuito ao passo que D+ liga a CLOCK (D2) e D- liga a DATA (D3).

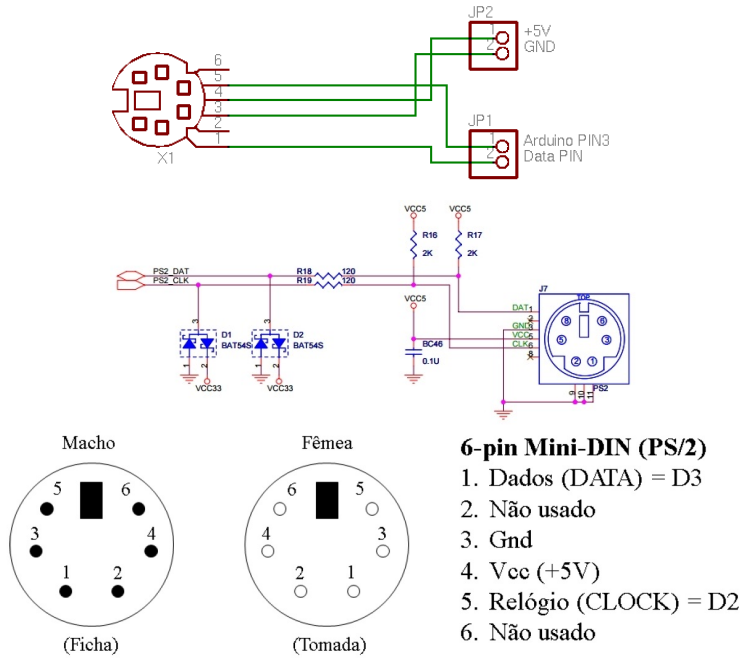

Figura 1.31: Ligações das fichas de teclado PS/2 e USB (esquerda) e as ligações a realizar para ligar a um Arduíno Uno (direita).

Os dados enviados pelos teclados podem variar, especialmente nos equipamentos mais antigos, sendo o mais habitual o envio de pacotes de 11 bits: 1 bit INÍCIO (0), 8 bits DATA, 1 bit PARIDADE (ímpar), e 1 bit de FIM (1). Quando uma tecla é premida o teclado pode enviar até 6 pacotes de 11 bits, sendo o mais habitual 2 pacotes.

A figura 1.32 exemplifica a captura dos sinais de dados (DATA) e relógio (CLOCK) enviados por teclado após o premir de uma tecla. No fundo do figura encontra-se a descodificação dos valores pelo osciloscópio (0xF0 0xF3).

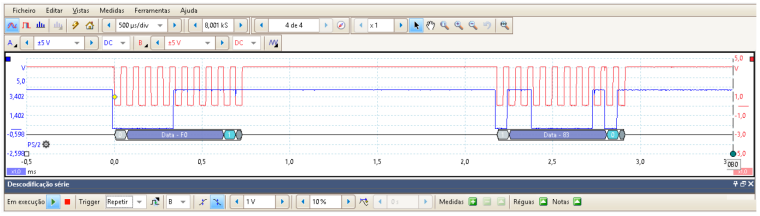

Figura 1.32: Diagrama de forma de onda dos sinais DATA e CLOCK gerados por um teclado PS/2.

Para descodificar os códigos enviados por um teclado PS/2 é necessário estar sensível às transições do sinal de relógio por ele gerado, pois é ele que indica quando existem bits no sinal de dados para serem lidos. Isto é conseguido à custa de uma rotina de interrupção, em que a cada interrupção é lido o nível do sinal de dados. Cada bit de dados recebido, é colocado numa variável do tipo byte (8 bits) na respetiva posição, começando com o bit menos significativo até ao mais significativo. A figura 1.34 mostra o sinal de relógio (CLOCK) e um sinal auxiliar produzido pelo Arduíno que fica ativo durante a leitura do sinal de entrada de dados (DATA). As leituras acontecem no meio da transição descendente (0-1) do sinal CLOCK. Os códigos enviados pelo teclado não possuem qualquer correspondência com os carateres ASCII das letras nas teclas.

Após a recepção dos códigos do teclado é necessário descodificá-los, e estabelecer a correspondência com os carateres do teclado, implementada com uma tabela (*Scan Code Set 2*). Um teclado envia o códigos de uma tecla premida. Quando a tecla é largada, o teclado envia o código 0xF0 seguido do código da tecla, e independentemente se uma tecla Shift foi premida ou não. Quando uma tecla premida não é libertada em 500 ms, o último código da tecla é enviado repetidamente até a tecla ser libertada. As teclas multimédia nos teclados recentes geram sequências de vários códigos estendidos, em que o primeiro código é 0xE0. A figura **??** mostra a correspondência entre um conjunto de códigos e as teclas mais elementares.

Alguns teclados possuem LEDs para indicar o seu estado, nomeadamente escrita em maiúsculas (CAPS LOCK) e alternância da utilização das teclas numéricas (Numpad) para introdução de dígitos ou cursor (NUM LOCK). A ativação dos LEDs é feita pelo computador ou Arduíno. Para mudar o estados dos LEDs o computador manipula os sinais de CLOCK e DATA, e envia o comando 0xED, ao qual o teclado devolve o valor 0xFA e só depois o Arduíno envia um byte para o teclado onde os bits menos significativos correspondem aos LEDs do teclado: bit 0 = Scroll Lock, bit 1 = Num Lock e bit 2 = Caps Lock, sendo os restantes bits ignorados. Por exemplo, o código para ativar o Num Lock e o Caps Lock é o 0x06.

Os teclados suportam ainda outros códigos tais como: 0xFF = reiniciar, 0xFE = reenviar a última tecla, 0xF3 = definir a taxa de repetição de códigos e 0xEE = pedido de eco, para o teclado devolver 0xEE.

A listagem **??** mostra um Sketch de demonstração da programação de um Arduíno Uno ligado a um teclado PS/2. Este Sketch inclui a biblioteca onde está definida a classe `PS2Keyboard`. Este classe tem como métodos públicos o construtor com parâmetros os identificadores dos pinos CLOCK e DATA, e o vetor com os carateres correspondentes às teclas do teclado. A biblioteca inclui o mapeamento das teclas de um teclado Português (`keyMapPT`) que figa guardado em memória de programa (`const PROGMEM char keyMapPT[]`). É possível alterar ou adicionar outros mapas de carateres de acordo com o

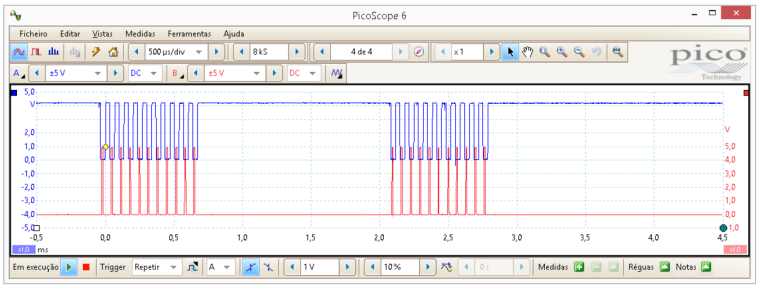

Figura 1.33: Diagrama de forma de onda do sinal CLOCK gerado pelo teclado PS/2 e leitura do sinal DATA pelo Arduíno.

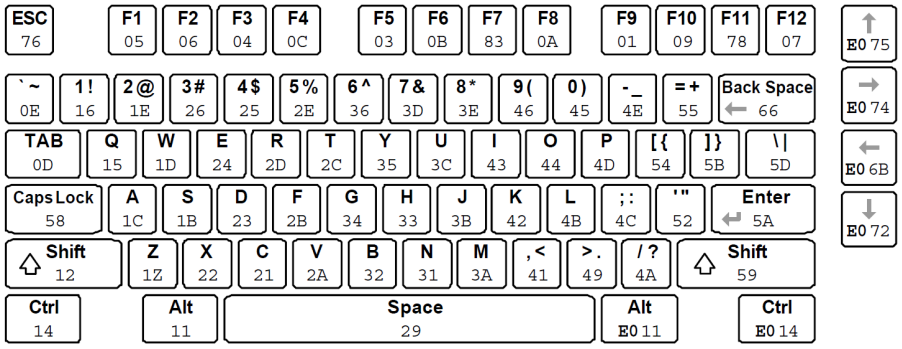

Figura 1.34: Correspondência entre os códigos de teclado PS/2 e as teclas .

teclado usado. A classe `PS2Keyboard` tem ainda os métodos públicos `kbhit()` para indicação de deteção de tecla premida e `getch()` para indicação do carater da tecla premida. O método `getcode()` fornece o código da última tecla enviada pelo teclado.

A nível físico da implementação, o Arduíno recebe os códigos enviados pelo teclado através da interface série síncrona, que não tendo nenhum circuito controlador integrado, é realizada em programação através de uma rotina de atendimento de interrupção (ISR). A ISR é despoletada pela variação de sinal de relógio gerada pelo teclado por cada bit enviado. Contudo, as ISRs podem pertencer à classe do objecto mas não podem pertencer a instâncias de objetos - têm de ser `static`, pelo que para que os dados lidos fiquem no objecto, a ISR deve instanciar o objeto através do seu ponteiro, guardado na variável global ao módulo `ps2inst_`, e que é iniciado no construtor.

Por cada bit de dados recebido, a função `getKey()` avalia se recebeu todos os 11 bits do código dentro do período limite. Só quando os 11 bits forem recebidos, embora apenas 8 contenham informação, é registada a alteração na variável `hit`. No Sketch de demonstração, o método `kbhit()` devolve o valor desta variável, o que tendo o valor 1 leva à execução do método `getcode()`. De salientar que este método apenas devolve o código enviado pelo teclado. A descodificação da tecla é realizada pelo método `getch()`, usando o ma-

peamento de teclado Português. De momento este método não suporta funcionalidades avançadas, nomeadamente SHIFT, CTRL, ALT e CAPS, uma vez que a interpretação das teclas depende fortemente da aplicação. Uma implementação mais completa por ser encontrada em `https://github.com/techpaul/PS2KeyAdvanced` ou `https://playground.arduino.cc/Main/PS2Keyboard/`.

A figura 1.35 mostra o aspeto do aparato usado nos ensaios realizados com o teclado PS/2. Para além das ligações a VCC, GND, CLOCK e DATA entre a ficha do teclado e as ligações ao Arduíno Uno na breadboard, vê-se também a ligação das duas pontas de prova do osciloscópio e da garra de GND. O condensador eletrolítico é usado para suprimir os flutuações de tensão que possam haver devido a picos de corrente e ao comprimento dos fios. A figura 1.36 mostra o terminal de texto Arduíno após terem sido premidas algumas teclas no teclado PS/2.

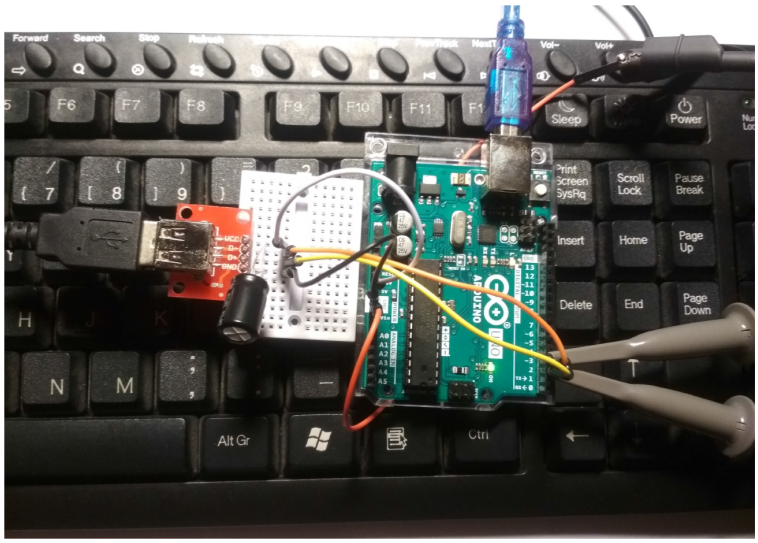

Figura 1.35: Aparato usado no ensaio do teclado PS/2.

Listagem 1.8: Listagem do Sketch PS2Keyboard.ino

```
#include "PS2Keyboard.h"

#define PS2_DATA_PIN     3
#define PS2_CLK_PIN      2

PS2Keyb keyb(PS2_CLK_PIN, PS2_DATA_PIN, keyMapPT);

void setup() {
  Serial.begin(115200); while(!Serial);
  Serial.println(".:PS/2 Keyboard:.");
}

void loop() {
  byte  c;
  char buf[80];
  // read the next key available
  if (keyb.kbhit())  {
    c = keyb.getch();
```

```
#if 1
    Serial.print(" #");
    c = keyb.getcode();
    Serial.print(c, HEX);
#else
    sprintf(buf, "%c", c);
    Serial.print(buf);
#endif
  }
}
```

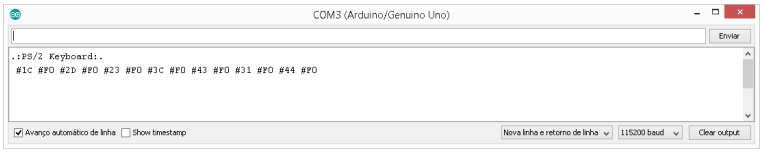

Figura 1.36: Aspeto do conteúdo do terminal de texto do Arduíno durante o ensaio ao teclado PS/2.

1.12 Rato de Computador PS/2

Antes da evolução da tecnologia para USB, também os ratos de computador eram ligados a uma ficha do tipo PS/2. Embora já não tão disponíveis, os ratos PS/2 são uma forma de introdução de coordenadas intuitiva e fácil de usar.

Tal como o teclado, quando o rato está imóvel os sinais de CLOCK e DATA estão a VCC, e quando ele é movido ele gera sinal de CLOCK. A cada movimento do rato ele envia 3 conjuntos de 11 bits: estado (estado dos botões e sentido), direção horizontal (X) e direção vertical (Y). Cada conjunto de 11 bit começa com um bit a 0, seguido de 8 bits de informação, um bit de paridade e um bit de paragem a 1. Quando o rato é movido para a esquerda ou para abaixo o sentido é considerado negativo, e positivo caso contrário. O sentido do deslocamento é dado nos bits de estado. A quantidade de movimento é proporcional à velocidade de deslocamento, saturando num valor máximo.

O bits de estado estão organizados da seguinte forma: L, R, 0, 1, XS, YS, XV, YV, P. L e R correspondem ao estado dos botões do rato esquerdo (L) e direito (R), XS e YS correspondem à indicação de saturação horizontal (XS) ou vertical (YS), e XV e YV indicam o sentido do deslocamento do rato para cada eixo.

2
Projetos

Neste capítulo são apresentados alguns projetos didáticos, alguns pensados de raiz e outros baseados em soluções existentes. A ideia é partir de um conjunto de requisitos, realizar o projeto aplicando uma abordagem modular, reutilizando alguns dos componentes apresentados nos capítulos anteriores e ir testando-os isoladamente até construir uma solução mais complexa, e facilitar a realização de um protótipo capaz de servir de demonstrador. Esta abordagem é conhecida como *Test-Driven Development* (TDD).

Os projetos apresentados que se baseiam em soluções existentes são analisados e discutidos para dar compreensão de como foi realizado, e poderá ser modificado. A grande vantagem de usar circuitos e módulos existentes nesta fase é a garantia de que esse circuito foi realizado e encontra-se disponível sob a forma de Shield Arduíno, e acompanha um Sketch de demonstração pronto a ser programado.

2.1 Noções Práticas

Recomenda-se que os seguintes aspetos sejam tomados em atenção para evitar erros que podem danificar o circuito ou causar perigo para quem estiver na proximidade. Na montagem de um componente com polaridade e com dois ou mais terminais é necessário identificar a função de cada terminal. Os métodos mais comuns de identificação de pinos em componentes eletrónicos são:

- O terminal negativo corresponde ao terminal mais curto em alguns componentes com dois terminais, ex. LED e condensador;
- O terminal negativo nos condensadores é indicado com "-", à excepção dos condensadores de tântalo que indicam o terminal positivo com "+";
- Nos componentes para montagem em superfície, ou Surface Mount Device (SMD), o terminal positivo é indicado por uma barra. A excepção acontece para os condensadores eletrolíticos, onde pode vir indicado o terminal positivo ou negativo.
- O primeiro terminal em encapsulamentos regulares com 2 ou mais filas de pinos, como, por exemplo, nos circuitos integrados, é identificado por um ponto. Em circuitos integrados com duas filas de pinos, o primeiro pino encontra-se no canto inferior esquerdo, seguindo a numeração para a direita, e continuando depois para cima e para a esquerda.
- Nas placas de circuito impresso, normalmente o primeiro pino tem um *pad* rectangular, enquanto os demais são ovais. Pode existir também um ponto na serigrafia ou um recorte no delimitador do componente.

Com o avanço na miniaturização, fabricam-se componentes eletrónicos cada vez mais pequenos, permitindo integrações que ocupam cada vez menos área. No entanto, a manipulação de tais componentes torna-se bastante difícil sendo executada em fábricas ou com o auxilio de lentes e outros instrumentos de precisão.

Quando não é possível colocar um componente numa *breadboard*, por ter um encapsulamento SMD pode-se usar uma pequena placa adaptadora (*breakout board*). As *breakout boards*, podem fazer a ligação de todos os pinos de um chip para poder ser ligado mais facilmente, ou apenas alguns dos pinos necessários para o testar. As ligações das breakout boards são feitas com o espaçamento de uma décima de polegada (100 mils), semelhante ao espaçamento nos pinos das *breadboards* para poderem ser encaixadas diretamente através de pinos cavaleiros (*headers*).

A alternativa é realizar uma placa de circuito impresso, Placa de Circuito Impresso - *Printed Circuit Board* (PCB), com um circuito altamente afinado para a aplicação a realizar, com *footprints* correspondentes à localização dos terminais dos componentes.

2.2 Calculadora Básica

Uma calculadora básica oferece operações de soma, substração, multiplicação e divisão. Neste caso o Arduíno é usado para emular esse funcionamento, recebendo a informação do utilizador num teclado de 16 teclas, e mostra os valores recebidos e calculados num mostrador alfanumérico. A calculadora deverá funcionar de acordo com a notação polaca invertida Reverse Polish Notation (RPN), onde os operados são inseridos numa pilha e são consumidos quando o operador é recebido, os dois operandos[1]. Se for recebido um operador e não houver operandos suficientes na pilha, é gerado um erro. A tabela 2.1 ilustra o funcionamento de uma calculadora RPR para a realização da operação 3×4. A pilha tem um funcionamento do tipo o primeiro a entrar é o último a sair - *First In, Last Out* (FILO), e a primeira posição (de entrada) é a zero.

3:	3:	3:
2:	2:	2:
1:	1: 3	1:
0: 3	0: 4	0: 12

Tabela 2.1: Exemplo de funcionamento da pilha de uma calculadora RPR.

Os desafios deste projeto passam pela recepção dos comandos do utilizador para compor um número com diversos dígitos e pela organização da afixação da informação que pode ser sob a forma de um número real.

Aspeto esperado do mostrador com a calculadora em funcionamento. 1:xxxxxxxxxxxx 0:xxxxxxxxxxxx

2.2.1 Componentes

- 1x Mostrador alfanumérico 16x2;
- 1x Teclado numérico de 16 teclas: 0-9, ., +, -, *, /, Enter;
- 1x Arduíno Uno;
- 1x Sensor de movimento para limpar os dígitos introduzidos.

2.2.2 Programa

- Qualquer acção é sempre despoletada por comandos do utilizador;
- O Sketch tem 3 estados de funcionamento: "RECEBE OPERANDO", "EXECUTA OPERAÇÃO", e "MOSTRA RESULTADO".

[1] O mesmo funcionamento era usado nas calculadoras da série 48 da HP.

- No estado "RECEBE OPERANDO": por cada digito premido pelo utilizador, o valor do operando é atualizado;
- No estado "RECEBE OPERANDO": quando utilizador prime a tecla de um comando o Sketch muda para o estado "EXECUTA OPERAÇÃO";
- No estado "EXECUTA OPERAÇÃO": é verificada a legalidade da operação e um resultado é calculado. Finda a operação, avança para o estado "MOSTRA RESULTADO";
- Se em qualquer instante, o sensor de movimento for ativo por mais de um segundo, os dígitos introduzidos são apagados.

Realização prática Para suportar a funcionalidade é necessário ligar um mostrador alfanumérico LCD 16x2 e um teclado 4x4. Este LCD pode ser ligado diretamente usando o modo 4 bits ou via I2C, tal como visto no capítulo 5, e o teclado ligado tal como descrito no capítulo 4, usando 8 sinais E/S.

2.3 Relógio-Despertador

Um relógio despertador tem a função de mostrar a hora atual, e ter pelo menos um alarme que toca sempre que a hora atual seja igual à hora do alarme. Para evitar acordar cedo ao fim-de-semana, o despertador pode ser programado para tocar apenas no dias úteis.

O desafio deste projeto está na ligação de todos os sinais E/S, nomeadamente os mostradores de 7 segmentos, e na programação da máquina de estados para suportar a funcionalidade desejada

Requisitos Componentes:

- 4x Mostradores de 7 segmentos para apresentar horas e minutos;
- 3x Botões de entrada para mudar o modo de funcionamento e acertar a hora atual e a hora do alarme (MODO, HORA, MINUTO);
- 1x RTC para manter a hora mesmo que o circuito fique momentaneamente sem alimentação;
- 1x LDR para medir a luminosidade ambiente;
- 1x
- 1x Pequena coluna, *beeper* ou *piezzo* para servir de alarme acústico;
- 1x Arduíno Uno.

Funcionalidade:

- Mostrar alternadamente as horas e a data atuais no mostrador de 7 segmentos a cada 2 segundos;
- Ao pressionar o botão "Modo" alterna entre os seguintes modos: NORMAL, ALARME, DIA, HORA, DATA;
- No modo NORMAL: ao pressionar o botão HORA ativa o alarme, e ao pressionar MINUTO desativa o alarme;
- No modo ALARME: os botões HORA e MINUTO permitem incrementar o valor das horas e minutos do alarme, respetivamente;
- No modo DIA: o botão MINUTO permitem alternar entre despertar todos os dias, os dias úteis, ou apenas o próximo dia;
- No modo HORA: os botões HORA e MINUTO permitem incrementar o valor das horas e minutos da hora atual, respetivamente;
- Sempre que o alarme esteja programado e a hora atual seja a mesma da hora do alarme, é emitido um sinal sonoro durante 2 minutos.

2.3.1 Relógio de Tempo Real

Existe necessidade de contar o tempo (horas e dias) de forma precisa em sistemas embebidos para medir o tempo ou registar um evento. Para aliviar o processador dessa tarefa, existem módulos com essa funcionalidade, designados por Relógio de Tempo Real, ou Real-Time Clock (RTC). O RTC gasta muito pouca energia e é capaz de contar o tempo de forma precisa e autónoma, mesmo quando o sistema embebido não se encontra em funcionamento.

A figura 2.1 mostra dois circuitos RTC instalados em PCBs, baseados no DS1302, à esquerda, e no DS3231, à direita. Ambos são fabricados pela Maxim e comunicam usado a interface I2C. O circuito do DS3231 contém também uma memória não volátil ligada em paralelo no barramento I2C. A figura 2.2 mostra as ligações entre o Arduíno Uno e os dois módulos RTC DS1302 e DS3231.

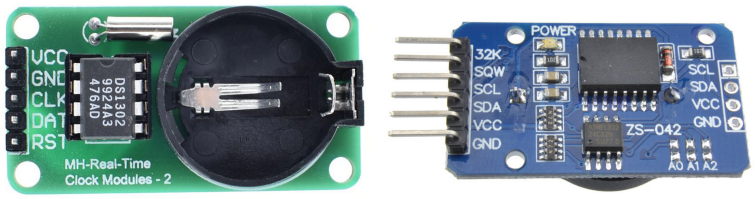

Figura 2.1: Módulos com relógio de tempo real DS1302.

Ligação via I2C para os endereços 0x80 para o DS1302, e 0x68 para o DS3231;

Datasheet: DS1302 + DS3231

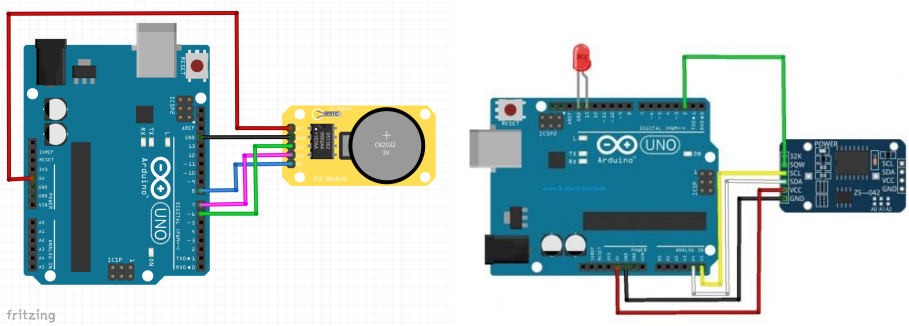

Figura 2.2: Circuito de demonstração do circuito do relógio de tempo real (RTC).

A figura 2.3 mostra o resultado da execução do Sketch apresentado na listagem 2.1 para demonstrar o DS1302. Para além de mostrar a hora e data atuais, este Sketch permite registar uma nova data e hora.

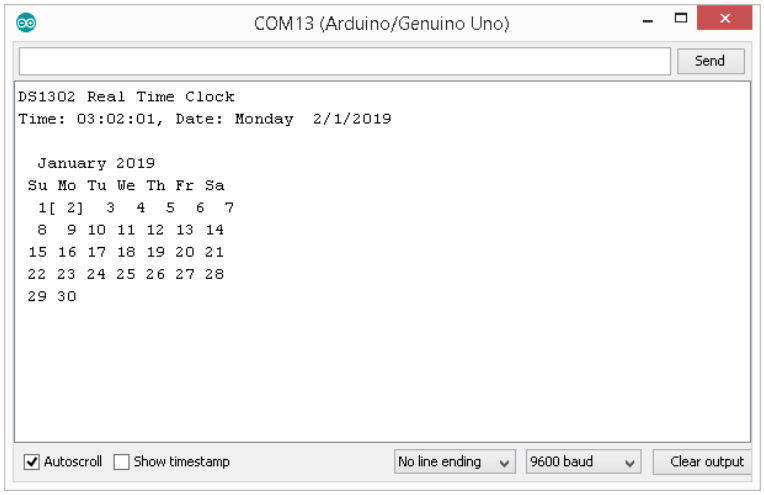

Figura 2.3: Resultado da execução do Sketch de demonstração do DS1302.

Listagem 2.1: Listagem RealTimeClock.ino

```
// DS1302 RTC

/*
// Set your own pins with these defines !
#define DS1302_SCLK_PIN   6    // Arduino pin for the Serial Clock
#define DS1302_IO_PIN     7    // Arduino pin for the Data I/O
#define DS1302_CE_PIN     8    // Arduino pin for the Chip Enable

*/

#include "ds1302.h"

DS1302 rtc  = DS1302();

char *dayOfWeek[7] = { "Sunday", "Monday ", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday" };
char *nameOfMonth[13]= { " ", "January", "February", "March", "April", "May", "June", "July", "August", "September",
char buffer[80];      // the code uses 70 characters.



void cal()
{

static const int days_in_month[2][13] = {
  {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
  {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
};

  int i;
  int day = rtc.Date10*10+ rtc.Date;
  int month = rtc.Month10*10 + rtc.Month;
  Serial.println("");
  sprintf(buffer, "  %s %d", nameOfMonth[month], ( 2000 +  rtc.Year10 *10 + rtc.Year));
  Serial.println(buffer);
  Serial.println(" Su Mo Tu We Th Fr Sa");
  for(i = 1; i < days_in_month[0][month]; i++) {
    if(i == day) Serial.print("[");
    else Serial.print(" ");
    sprintf(buffer, "%2d", i);
    Serial.print(buffer);
    if(i == day) Serial.print("]");
    if(i % 7 == 0) Serial.println("");
  }
  Serial.println("");
```

```
}



void setup()
{

  Serial.begin(9600);
  Serial.println(F("DS1302 Real Time Clock"));


  // Start by clearing the Write Protect bit
  // Otherwise the clock data cannot be written
  // The whole register is written,
  // but the WP-bit is the only bit in that register.
  rtc.wr(DS1302_ENABLE, 0);

  // Disable Trickle Charger if you're not using rechargable batteries.
  rtc.wr(DS1302_TRICKLE, 0x00);

// Remove the next define,
// after the right date and time are set.
#define SET_DATE_TIME_JUST_ONCE
#ifdef SET_DATE_TIME_JUST_ONCE

  // Fill these variables with the date and time.
  int seconds, minutes, hours, dayofweek, dayofmonth, month, year;

  // Example for april 15, 2013, 10:08, monday is 2nd day of Week.
  // Set your own time and date in these variables.
  seconds    = 1;
  minutes    = 2;
  hours      = 3;
  dayofweek  = 1;  // Day of week, any day can be first, counts 1...7
  dayofmonth = 2; // Day of month, 1...31
  month      = 1;  // month 1...12
  year       = 2019;

  // Set a time and date
  // This also clears the CH (Clock Halt) bit,
  // to start the clock.

  // Fill the structure with zeros to make
  // any unused bits zero
  memset ((char *) &rtc, 0, sizeof(rtc));

  rtc.Seconds    = bin2bcd_l( seconds);
  rtc.Seconds10  = bin2bcd_h( seconds);
  rtc.CH         = 0;      // 1 for Clock Halt, 0 to run;
  rtc.Minutes    = bin2bcd_l( minutes);
  rtc.Minutes10  = bin2bcd_h( minutes);
  // To use the 12 hour format,
  // use it like these four lines:
  //    rtc.h12.Hour   = bin2bcd_l( hours);
  //    rtc.h12.Hour10 = bin2bcd_h( hours);
  //    rtc.h12.AMPM   = 0;      // AM = 0
  //    rtc.h12.hour_12_24 = 1; // 1 for 24 hour format
  rtc.h24.Hour   = bin2bcd_l( hours);
  rtc.h24.Hour10 = bin2bcd_h( hours);
  rtc.h24.hour_12_24 = 0; // 0 for 24 hour format
  rtc.Date       = bin2bcd_l( dayofmonth);
  rtc.Date10     = bin2bcd_h( dayofmonth);
  rtc.Month      = bin2bcd_l( month);
  rtc.Month10    = bin2bcd_h( month);
  rtc.Day        = dayofweek;
  rtc.Year       = bin2bcd_l( year - 2000);
  rtc.Year10     = bin2bcd_h( year - 2000);
  rtc.WP = 0;

  // Write all clock data at once (burst mode).
  rtc.clock_burst_wr( (uint8_t *) &rtc);
#endif
}
```

```
void loop()
{

  // Read all clock data at once (burst mode).
  rtc.clock_burst_rd( (uint8_t *) &rtc);

  sprintf( buffer, "Time: %02d:%02d:%02d, ", ( rtc.h24.Hour10 * 10 + rtc.h24.Hour), \
    ( rtc.Minutes10 * 10 + rtc.Minutes), ( rtc.Seconds10 * 10 + rtc.Seconds));
  Serial.print(buffer);

  sprintf(buffer, "Date: %s %d/%d/%d   ", dayOfWeek[rtc.Day], rtc.Date10*10+ rtc.Date, rtc.Month10*10+ rtc.Month, ( 2
rtc.Year10 *10 + rtc.Year));
  Serial.println(buffer);

  cal();
  delay(9000);
}
```

Realização prática Para ligar 4 mostradores de 7 segmentos a um Arduíno em paralelo seriam necessários 32 sinais de E/S. Para poupar o número de sinais necessários, os mostradores de 7 segmentos são multiplexados no tempo, ou seja, acende um de cada vez durante um breve intervalo de tempo em sequência para dar a ilusão de que estão todos ligados. Neste caso apenas são necessários 8+4 sinais de E/S. 8 sinais para os segmentos e ponto, e 4 sinais para controlar o mostrador ativo. Ainda assim são necessários 12 sinais E/S digitais. Uma alternativa é usar um circuito semelhante ao do LCD alfanumérico, onde os dados são serializados e enviado um bit de cada vez. Como os restantes periféricos como necessitam apenas de poucos sinais de E/S, podem ser ligados diretamente ao Arduíno Uno.

Existe um Shield Arduíno que já reune a maioria dos componentes necessários para este projeto chamado "Multi-Function Shield". A figura 2.10 mostra o Shield Multi-function encaixado nos conectores de expansão de um Arduíno Uno. Existem bibliotecas para usar este Shield, no entanto, neste caso, os componentes vão ser tratados individualmente para ser mais fácil fazer alterações ao circuito, nomeadamente substituição de componentes.

A figura 2.5 mostra o detalhe das ligações dos componentes no Shield Multifunction. No esquema é possível observar o detalhe de como os mostradores de 7 segmentos se encontram ligados a *shit-registers* do tipo 74HC595. Um *shit-registers* recebe um bit a cada transição do sinal de relógio e guarda-o num registo de 1 bit, e avança para o registo seguinte. O *shit-register*

Dada a complexidade do Sketch para realizar a funcionalidade desejada, foi adotado o modelo **MVC!** (**MVC!**), explicado no capitulo 2. O Sketch vai ter 3 funções principais a serem executadas ciclicamente dentro da função loop():

- model() - neste caso vai resumir-se a mudar os estados de funcionamento, de acordo coma entrada do utilizador, e verificar se na hora atual o alarme se encontra ativo.

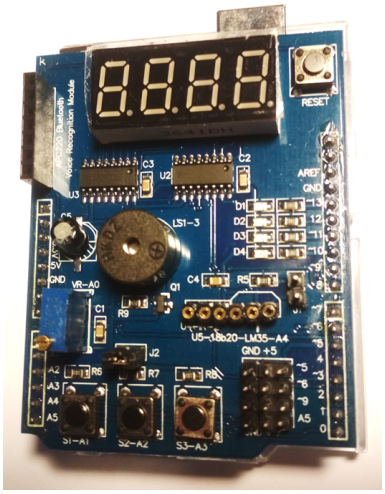

Figura 2.4: Shield Multifunction assente num Arduíno Uno.

- view() - mostrar informação atual e tratar da ativação do beeper;
- control() - identifica os comandos do utilizador através dos botões.

Como nota final, se após montar e testar o circuito sentir que o som do *beeper* é demasiado monótono ou pouco audível, pode-se usar um módulo receptor de rádio FM (Si4703 da Silicon Laboratories) juntamente com um módulo amplificador estéreo de 3W e um par de altifalantes. Ambos os módulos estão ilustrados na figura 2.6. Se ainda assim for difícil acordar, pode-se reutilizar parte do projecto da calculadora e combina-lo com o despertador, onde o alarme apenas deixa de tocar após o utilizador introduzir a resposta a uma sequência de operações aritméticas aleatórias.

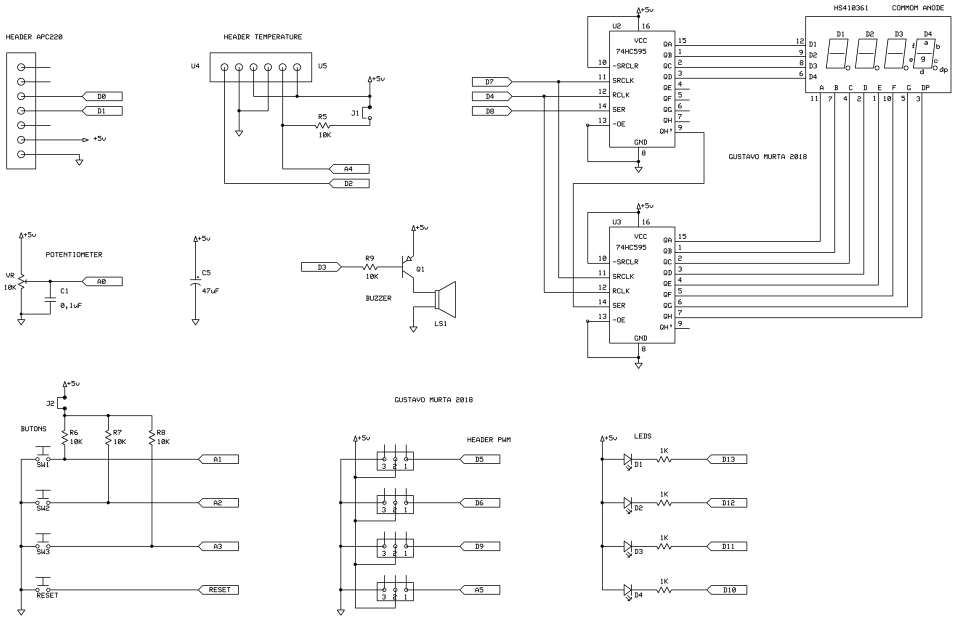

Figura 2.5: Esquema elétrico do Shield Multifunction.

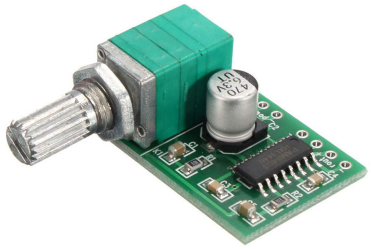

Figura 2.6: Módulo amplificador de 3W estéreo.

2.4 Mini-Osciloscópio

Um osciloscópio é um equipamento fundamental para realizar experiências com eletrónica, contudo o equipamento profissional é um investimento caro para quem está a começar. Este projeto faz uso das entradas analógicas do Arduíno e converte os valores de tensão nas suas entradas e envia um valor numérico proporcional para serem apresentados. Este projeto foi realizado com duas variantes: 1) para funcionar com o computador, e 2) com um mostrador LCD para funcionamento autónomo.

Computador

A listagem 2.2 mostra o Sketch `adc6ChDemo.ino` que ciclicamente lê o valor das 6 entradas analógicas de um Arduíno Uno e envia para a porto série o seu valor escalado e deslocado para que os seis sinais possam ser visualizados sem serem sobrepostos.

Listagem 2.2: Sketch adc6ChDemo.ino para realizar um mini-osciloscópio de 6 canais com um Arduíno Uno

```
#define AN_NUM_INPUTS   6

int i;
int adcPins[] = { A0, A1, A2, A3, A4, A6 };
int adcVals[AN_NUM_INPUTS];
int chScale = 25;
int chBias[] = { 0, 50, 100, 150, 200, 250};

void setup()
{
  Serial.begin(115200);
  while(!Serial);
}

void loop()
{
  for(i = 0; i < AN_NUM_INPUTS; i++) {
    adcVals[i] = analogRead(adcPins[i]);
    Serial.print(adcVals[i] / chScale + chBias[i]);
    Serial.print(",");
  }
  Serial.println(1023 / chScale + 300);
}
```

LCD

2.5 CanSat

O CanSat é uma iniciativa da Agência Espacial Europeia (ESA) para atrair o interesse dos jovens pelo espaço[2]. Esta inciativa junta equipas representantes de todos os países para competirem pela solução que seja mais interessante do ponto de vista cientifico e que tenha o melhor desempenho no lançamento.

O CanSat pretende ser uma aproximação a um satélite mas contido no volume e forma de uma lata de refrigerante 33 cl. O desafio passa por integrar todos os componentes do satélite nesse volume, incluindo computador de bordo, fonte de alimentação, sensores e elementos de comunicação. A competição inclui o lançamento dos satélites num foguete, para serem largados a uma altura de 1 km. Existem alternativas para o lançamento a partir de um balão, drone ou avioneta.

Um CanSat deve obedecer às seguintes especificações:

- Altura: 115 mm,
- Diâmetro: 66 mm,
- Peso máximo: 350 gr,
- Custo: <500 Euros,
- Alimentação: pilha, bateria ou painel solar,
- Autonomia: 4 h espera (stdby) e 170 s durante o voo,
- Velocidade de descida: 8-11 m/s.

Neste projecto procura-se demonstrar um CanSat simples capaz de registar o seu movimento, temperatura e humidade durante a descida, e transmitir as suas coordenadas GPS via LORA para poder ser recuperado após o lançamento.

2.5.1 Estrutura

Para cumprir com os requisitos mecânicos optou-se por construir o modelo 3D de CanSat existente em `https://cansat.eu`. Este modelo encontra-se pronto a ser enviado para uma impressora 3D (neste exemplo foi usada uma BeeCreative), e é capaz de acomodar um Arduino Uno, Mega ou Due. A figura 2.7 mostra as peças do CanSat após sairem da impressora 3D.

Figura 2.7: Peças impressas com base nos ficheiros fornecidos pelo projeto Canduino.

[2]

Função	Referência	Critério
Posição	?	Solução integrada que oferece Acelerómetro, Magnetómetro, e Giroscópio nos eixos X, Y e Z, e Altímetro
Temperatura e humidade	SHT21	Solução integrada e compacta capaz de medir temperatura e humidade dentro dos limites de operação
Registo de dados	módulo SD Card	Permite escrever num dispositivo e num formato que pode ser em qualquer computador
Computador de bordo	Arduíno Mega	Número de pinos de E/S
Receptor GPS	?	?
Módulo LORA	?	?
Sensor de luz	LDR	Enquanto o sensor estiver tapado o CanSat encontra-se adormecido, sendo ativado após ser exposto a luz solar

Tabela 2.2: Componentes escolhidos e respetivos critérios de selecção para o CanSat.

Após separadas as peças com um alicate de corte, é necessário limar vários pontos de contacto por terem excesso de material que impedem a junção das várias peças. Os varões roscados e as porcas de 3 mm permitem manter os painéis laterais fixados nos de topo e fundo para que o CanSat fique fechado e não se desmonte.

2.5.2 Circuito

Para cumprir com os requisitos a nível da funcionalidade do circuito foram escolhidos os componentes de acordo com os critérios apresentados na tabela 2.2. Outros componentes podem ser usados em substituição dos escolhidos se tiverem caraterísticas e funcionamento equivalentes. A solução de alimentação é apenas escolhida após a avaliação do consumo do sistema completo.

2.5.3 Programa

O programa de controlo do CanSat é relativamente simples pois apenas necessita de recolher e escrever a informação de vários sensores durante a descida.

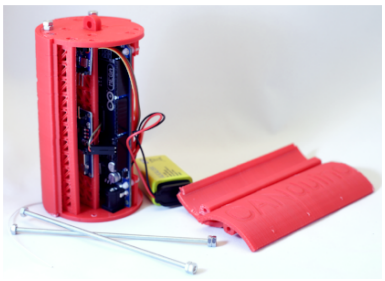

Figura 2.8: Esquema elétrico do CanSat.

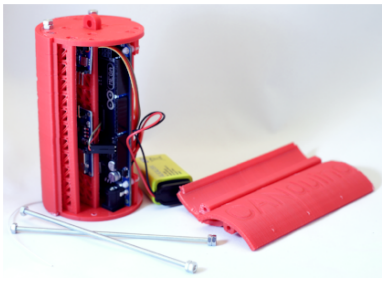

Figura 2.9: canduino.

2.5.4 Demonstração

2.5.5 Outras Considerações

Existem diversas páginas na Internet dedicadas aos CanSat.

`http://unisec.jp/library/i-cansat/manual_CanSat_textbook_eng_v5.pdf`

Por omissão os receptores GPS cumprem com a regulamentação CoCom (*Coordinating Committee for Multilateral Export Controls*), do Departamento de Comércio do EUA, onde o equipamento GPS deixa de funcionar em altitudes superiores a 18km ou quando atingem velocidades superiores a 1000 nós (apx 514 m/s).

2.6 Node IoT

- luz - proximidade - temperatura - humidade - comando remoto - beeper - toque - LORA ou Ethernet

2 botões de pressão: K1 = D8, K2 = D9 4 LEDs coloridos monocromáticos: Amarelo = D7, Azul = D6, Verde = D5 e Vermelho = D4; DHT11: D12; Receptor de infravermelhos: IR = D3; Beeper: BUZZER = D4;

A knob (potentiometer): A0 NTC thermistor: A1 LDR Light dependent resistor: A2 24C02 eeprom: A4 (SDA) + A5 (SCL) 4 mostradores de 7 segmentos via TM1637: D10 (CLK) + D11 (DATA) I2C connector UART connector D0 (RX) + D1 (TX)

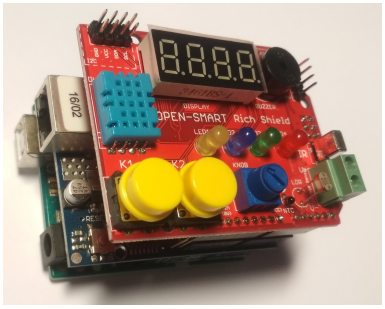

Figura 2.10: Montagem do Node-IoT com um "Rich Shield" e um "Ethernet Shield".

Uma alternativa à ligação Ethernet é usar um modulo LORA.

Demo com protothreads ??

2.7 Pedal de Efeitos para Guitarra Elétrica

Objectivo: Componentes: kit pedal - electrosmash pedal shield Código: Fotografia:

fonte: `https://www.electrosmash.com/pedalshield-uno`

A figura 2.11 mostra o aspeto da vista superior e inferior do Shield para fazer o pedal de efeitos de guitarra (esquerda) e o esquema de ligação do pedal ao computador, à guitarra e ao amplificador de som (direita).

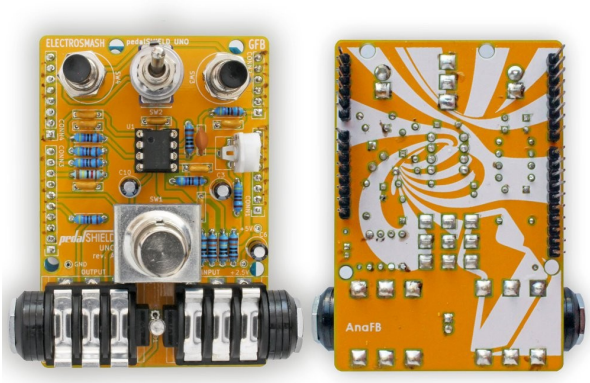

Figura 2.11: Pedal de guitarra elétrica baseado em Arduíno.

A figura 2.12 mostra o diagrama do circuito elétrico do Shield de guitarra. À esquerda, entre a guitarra e o Arduíno encontra-se o andar de amplificação do som da guitarra para poder ser adquirido pelo canal analógico A0. O circuito à esquerda está encarregue por criar e amplificar o sinal analógico para o amplificador à custa de dois sinais PWM (D9 e D10).

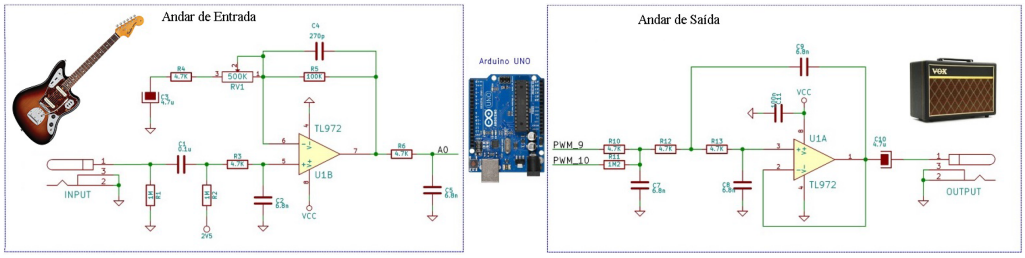

Figura 2.12: Diagrama do circuito do Shield para pedal de guitarra elétrica.

Figura 2.13: Aparato da ligação do Shield de pedal de efeitos de guitarra à guitarra, ao amplificador e ao computador.

2.8 Mini-Jogo

Objectivo: Componentes: lcd + joystick + push buttons breakout + beeper/piezo Código: Fotografia:

JoyStick da Funduino. Este shield tem disponíveis os seguintes componentes:

- Joystick de 2 eixos
- 2 botões de pressão grandes
- 2 botões de pressão pequenos
-

O shield possui ainda ligação a uma interface série assíncrona para ligar um módulo Bluetooth e outra para um módulo RF 2,4 GHz (nRF24L01), barramento I2C, por exemplo para ligar um acelerómetro, e um mostrador gráfico LCD idêntico ao que equipava os telemóveis 5110 da Nokia.

As ligações neste Shield são as seguintes:

- Botão A (Cima) = D2
- Botão B (Direita)= D3
- Botão C (Baixo) = D4
- Botão D (Esquerda) = D5
- Botão E (Começar) = D6
- Botão F (Selecionar) = D7
- Botão Joystick (Eixo Horizontal) = D8
- Eixo Horizontal Joystick (X) = A0
- Eixo Vertical Joystick (Y) = A1

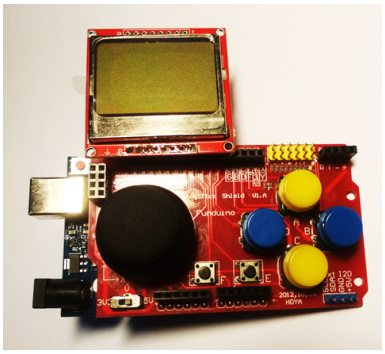

Figura 2.14: Mini jogo arduino.

2.9 Controlador de Estufa

Objectivo: Controlo e monitorização do ambiente da plantação (temperatura, humidade, e luminosidade) Componentes: base = arduino uno+eth, I2C c/ estações, RTC, LCD. estações= arduino nano, sensores analogicos para humidade, terra e ar, temperatura, ph solo, rele p/ luz artificial, + I2C c/ base Código: Fotografia:

Demo com protothreads ??

2.10 Robot

2.10.1 Estrutura

2.10.2 Circuito

- arduino
  - sensor shield
  - motores dc
  - sensor distancia

2.10.3 Programa

avança até encontrar um obstaculo, vira para onde tiver mais espaço, esquerda ou direita.

2.10.4 Demonstração

2.11 Desenvolvimento de um Shield Arduino

Nos circuitos digitais é habitual colocar um condensador de 100 nF o mais próximo dos pinos de alimentação para compensar as flutuações causadas pela transição de vários bits aquando da transição do sinal de relógio(*clock*).

3
Conceitos Avançados

O capítulo anterior introduziu os conceitos considerados fundamentais para entender e iniciar o estudo e desenvolvimento de sistemas embebidos com a plataforma Arduíno. Neste capítulo são apresentados tópicos que permitem desenvolver sistemas embebidos mais completos e complexos. As secções seguintes são dedicadas a conceitos de matemática e métodos numéricos, eficiência energética, tolerância a falhas, padrões arquiteturais em programação, processamento digital de sinal, e controlo.

3.1 Miniaturização

Em aplicações onde o volume e o peso disponíveis para o sistema embebido são limitados, tal como em satélites e dispositivos vestíveis, é necessário adotar estratégias que sejam minimizados, mas sem sacrificar a funcionalidade nem a qualidade do circuito. Antes de se partir para a construção de um circuito miniaturizado deve-se garantir que o circuito funciona corretamente num protótipo com dimensões maiores. Algumas das estratégias de miniaturização a adoptar são:

- Usar componentes para montagem em superfície (SMD) com *footprints* reduzidos, tais como 0603, 0402, ou até 0201, com o auxílio de uma boa lupa ou microscópio;

- Substituir partes do circuito por soluções modulares (circuitos integrados que suportam toda a funcionalidade), por exemplo modems *Global System for Mobile Communications* (GSM) e LORA;

- Soldar componentes nas duas camadas exteriores da PCB;
- Usar fios para *wirewrapping* ou envernizados (de bobinar) para fazer as ligações de sinais com pouca corrente e tensão, pois têm uma secção exterior menor do que os fios convencionais. A figura 3.1 mostra um rolo de fio para *wirewrapping* à esquerda, e a ferramenta para descarnar o fio à direita;
- Usar PCBs com quatro camadas ou mais para distribuir as pistas por mais camadas e reduzir a área da PCB;
- Separar o circuito por duas ou mais PCBs da mesma dimensão para realizar uma sanduíche suportada pelos comnectores laterais. À custa de um pequeno acréscimo em altura é possível reduzir a área da PCB;
- Usar vias internas (*blind and buried vias*) para estabelecer a ligação entre camadas internas da placa de circuito impresso sem impedir a utilização da mesma localização por outras camadas;
- Se o sistema for alimentado a baterias, usar a tecnologia que oferece a maior densidade de energia $[J/m^3]$, por exemplo, uma bateria Li-ion Molicel INR-18650-P28A fornece cerca de 2800mAh, tem 47 gr de massa e dimensões 18.5 $mm \times 65.2$ mm[1];
- Usar componentes e tecnologia que dissipe a menor energia possível. Um circuito miniaturizado é para ser alojado num espaço com pouca, ou nenhuma, ventilação, logo os componentes não devem contribuir para o aquecimento do circuito sob pena de entrarem em sobreaquecimento e degradarem-se e, por consequência, o circuito deixar de funcionar (corretamente);

Uma consequência inevitável da miniaturização do circuito é o aumento dos custos dos componentes (tecnologias mais compactas e mais recentes) e dos métodos de fabrico (PCB e soldadura exigem menor tolerância de fabrico).

3.2 Engenharia Reversa

A engenharia reversa consiste em obter um produto semelhante ao que se procura projectar, desmontá-lo, tentar perceber como funciona e tentar adaptar ou melhorar algumas das ideias no novo projecto.

Em sistemas embebidos, a primeira abordagem é identificar quais os *chips* (retângulos ou quadrados pretos com muitos terminais) que são usados e qual

[1] `http://www.molicel.com/hq/product/INR18650P28A-V1-80093.pdf`

Figura 3.1: Rolo de fio para *wirewraping* à esquerda, e ferramenta para descarnar e enrolar o fio à direita.

a sua funcionalidade, através do estabelecimento da referência gravada no corpo do componente e o seu *datasheet*. As ligações enter componentes pode ser descoberta através de um multímetro usado como detetor de continuidade. As ligações também permitem perceber quais os controladores usados dentro do MCU, se algum for.

A programação do MCU pode ser determinada com um osciloscópio ou analisador lógico. A vantagem do analisador lógico é conseguir ler mais canais digitais em paralelo (8-32) do que um osciloscópio (1-4). Sabendo o instante de tempo em que cada sinal está ativo é possível inferir quais os componentes usados em cada instante, e identificando os valores trocados entre os componentes é possível determinar qual a sua configuração, comandos e até algoritmos usados.

Alguns sistemas embebidos para dificultar que sejam analisados, e/ou alterados, eles usam mecanismos na sua instalação que permite saber se a sua caixa foi aberta ou não, como por exemplo deteção de nível de luminosidade num espaço que é suposto estar sempre às escuras, ou um sensor de movimento num equipamento instalado numa parede ou chão.

3.3 Eficiência Energética

A poupança de energia é importante para sistemas embebidos, especialmente para prolongar a sua autonomia quando são alimentados a pilha ou bateria. Existem estratégias para poupar energia a nível do circuito (hardware) e do programa (software).

A potência consumida por um sistema digital é dominada pela potência dinâmica (assume-se que existem sempre transições de 0/1 ou 1/0), e é derivada da potência consumida por um transistor a mudar de estado e que é dada por:

$$P = CV^2 f$$

, onde C é a capacidade parasita nos terminais do transistor, V a tensão de alimentação e f a frequência de mudança de estado num segundo.

A ideia mais simples para poupar energia é desligar todos os componentes que não estão a ser usados. Hoje em dia muitos componentes permitem entrar no modo de baixo consumo e serem acordados apenas quando são precisos. Também os MCUs possuem modos de funcionamento para poupar energia, em que o processador fica parcialmente parado e apenas retoma a atividade normal após receber uma indicação através de um sinal externo. Quando os MCUs não suportam esses modos, uma estratégia alternativa para reduzir o consumo passa pela redução da frequência do sinal de relógio. Se um MCU a 512 kHz conseguir realizar o mesmo processamento que outro a 16 MHz não há necessidade de desperdiçar energia, e se o sistema for alimentado a bateria a sua autonomia cresce cerca de 32×.

Vários componentes ligam aos sistemas embebidos a portos de E/S com *pull-ups*. O *pull-up* consiste numa resistência de polarização que coloca o porto a VCC (1), enquanto não existir nenhuma ativação em contrário. Quando um componente colocar esse sinal a GND (0) vai passar corrente de VCC para GND através da resistência de *pull-up* e que será dissipada. Para minimizar a energia dissipada na resistência de *pull-up* deve-se escolher um valor de resistência suficientemente elevado para que a corrente possa ser desprezada, mas não tão grande que o *pull-up* deixe de funcionar e torne o sinal sensível a interferências e ruído.

Se pensarmos numa bateria do tipo "botão" (ex. CR2032) que fornece 3 V e tem uma capacidade de fornecer 210 mAh, se um sistema embebido que funcione a 3 V e consuma 10 mA, vai funcionar durante 21 horas. Se existir um componente no sistema que consuma 40% da corrente total (4 mA), e ativando o modo de poupança de energia passe a consumir em média um quarto (1 mA), o sistema vai passar a funcionar durante 30 horas.

Muitas vezes os componentes não possuem modos de poupança de energia, de maneira que a alternativa é desligar o componente. Um componente que consuma 10 mA e que a cada milissegundo pode ser desligado durante cem microssegundos, permite uma poupança de energia de em média $(1000 - 900)/1000 = 10\%$.

Em sistema embebidos para aplicações Internet das Coisas - *Internet of Things* (IoT) e ciber-físicos, os módulos de comunicação com e sem fios são dos componentes que mais energia consomem. Para reduzir o consumo de energia desses sistemas, uma estratégia passa por comprimir os dados a enviar, ou a adoptar codificações de dados compactas para ocupar o canal de comunicação o menos possível. Em canais com o meio partilhados, tal como em RÃ¡dio FrequÃªncia (RF), longos períodos de transmissão significa maior probabilidade de haver outro sistema a comunicar e ocorrer colisão e por consequência nova transmissão. Desta forma, devem-se favorecer os débitos de transmissão mais elevados que os sistemas de processamento e comunicação

permitam.

Pode-se pensar em usar um pino de E/S como fonte de alimentação de um sub-circuito, ou componente, para o poder ligar e desligar. No entanto, é necessário ter em atenção que os pinos de E/S não fornecem a corrente necessária, ou a tensão estável e exata, para o componente funcionar corretamente.

Em termos práticos desligar um componente pode ser conseguido com um circuito condicionador do tipo dreno-aberto, colocando um MOSFET na alimentação do componente, com a Porta do MOSFET ligada a um pino de E/S. É necessário ter atenção da escolha do MOSFET, especialmente em relação ao valor de ativação e corrente máxima suportada pela carga.

A figura 3.2 exemplifica um circuito para comando de uma NTC. Apenas quando o sinal de E/S que controla a *gate* do MOSFET se encontra HIGH é que o MOSFET entra em modo de condução, e fecha o circuito da NTC e da resistência. Quando o sinal de E/S se encontra LOW, a tensão no nó entre a resistência R e a NTC encontra-se a VCC porque não existe corrente a percorrer as resistências.

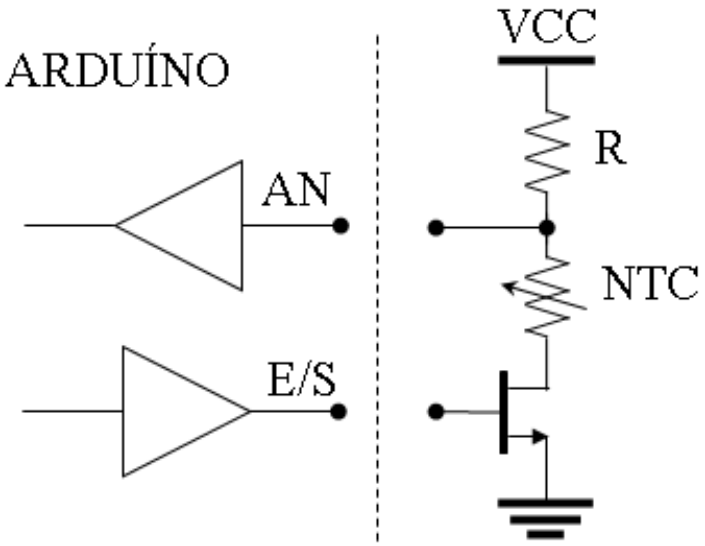

Figura 3.2: Exemplo da utilização de um MOSFET para controlar um sensor de temperatura.

Em programação, a poupança energética é conseguida através da redução de instruções executadas pelo *Central Processing Unit* (CPU), e colocando o CPU em modo de baixo consumo ou em suspensão, se possível. Algumas estratégias a nível da programação são:

- Trocar esperas ativas (*polling*) por resposta a eventos externos (*interrupt-triggered*).

- Se não for possível evitar esperas ativas, ao menos introduzir um intervalo de espera entre avaliações da condição de paragem, ex. `while(not ready) delay(100);`
- Usar macros e funções *inline* em vez de chamar funções. Isto faz com que o código da função fique replicado e gere um programa maior, no entanto permite para poupar a utilização da pilha habitual na chamada de funções.
- Se for mesmo necessário chamar funções, evitar passar argumentos (ex. usar ponteiros ou variáveis globais), para reduzir a utilização da pilha.
- Verificar conteúdo de registos que fazem a ativação de componentes e pinos de E/S. Reduzir ao mínimo essencial para o sistema funcionar. Usar os registos de controlo da frequência de relógio do CPU para selecionar frequências de relógio mais baixas. O programa vai demorar mais tempo a executar, mas vai gastar menos energia.
- Adaptar o programa para usar tipos de dados de menor dimensão, por exemplo `double`→`float` e `long`→`int`. Ao usar variáveis mais pequenas são usados menos acessos à memória e realizadas menos operações, o que se traduz numa poupança de energia e tempo de processamento.

3.4 Técnicas para Aumento de Desempenho

Alguns dos algoritmos usados em processamento digital de sinal requerem um tempo de processamento elevado, e não são adequados para dar resposta em tempo real. Nestes casos é necessário fazer uma avaliação prévia do algoritmo a usar (*profiling*) e escolher um sistema embebido adequado. Uma forma básica de conseguir esta avaliação é pelo tempo de execução num sistema conhecido ou identificar o número de instruções necessárias à execução do algoritmo e determinar qual a taxa de execução de instruções que o processador deve ter para serem executadas dentro de um determinado tempo limite, Eq. 3.1.

$$freq_{CPU} = \frac{num\,instr}{tempo\,exec} \tag{3.1}$$

O número de instruções executavas pelo processador pode ser obtido após a compilação do programa, inspecionando a listagem assembly (linguagem de "baixo nível" resultado da compilação do programa em C/C++ em instruções suportadas pelo Instruction Set do processador usado). A listagem Assembly pode ser obtida através do seguinte comando:

Listagem 3.1: Listagem do comando para obter um Sketch traduzido em linguagem Assembly.

```
C:\Program Files (x86)\Arduino\hardware\tools\avr\avr\bin>objdump.exe -S
C:\Users\R\AppData\Local\Temp\arduino_build_964891\push_button.ino.elf
```

A listagem 3.2 mostra parte do código *assembly*. A coluna à esquerda tem os valores dos endereços de memória de programa, em hexadecimal, as duas colunas seguintes têm a instrução a executar também em hexadecimal, e à sua direita a instrução em *assembly*.

Listagem 3.2: Exemplo de código Assembly

```
; (...)
  for (;;) {
    loop();
    if (serialEventRun) serialEventRun();
 1aa: c0 e0          ldi r28, 0x00 ; 0
 1ac: d0 e0          ldi r29, 0x00 ; 0
 1ae: 20 97          sbiw  r28, 0x00 ; 0
 1b0: f1 f3          breq  .-4       ; 0x1ae <main+0x8a>
 1b2: 0e 94 00 00    call  0 ; 0x0 <__vectors>
 1b6: fb cf          rjmp  .-10      ; 0x1ae <main+0x8a>
; (...)
```

Quando o desempenho do sistema é crítico para a aplicação adotam-se algumas técnicas para reduzir o tempo de processamento, nomeadamente:

- Realizar operações com inteiros em vez de valores reais (vírgula flutuante). As operações com valores em vírgula flutuante demoram mais tempo do que com valores inteiros. Por esse motivo, e para reduzir o tempo de processamento, evita-se usar vírgula flutuante em sistemas embebidos.

- Evitar o cálculo de funções matemáticas complexas. Por vezes, o cálculo de uma aproximação serve à aplicação, e reduz o tempo de processamento. Se tal não for possível, uma alternativa é o uso de tabelas. No arranque da aplicação o conteúdo da tabela é preenchido e sempre que seja necessário calcular um valor basta consultar a tabela na posição correspondente.

- Evitar instruções do tipo `if`, `for` e `while`. Ao introduzir condições de paragem no programa, obriga ao processador a ter de avaliar a condição de cada vez que o programa passa por ela. Os processadores hoje em dia encadeiam as instruções a executar para aumentar o desempenho, mas se houver uma condição na execução de um programa, o processador não sabe qual a instrução a executar sem avaliar a condição primeiro.

- Considerando duas variáveis, `a` e `b`, onde `a` funciona como *flag* que tem o valor 0 ou 1, e `b` conta o número de vezes que `a` é igual a um: `if (a == 1) b++;` uma alternativa que demora menos tempo a executar é: `b += a;`

- Evitar passar valores na chamada de funções para não perder tempo com o mecanismo de manipulação da pilha (*stack*). A alternativa é usar

ponteiros ou variáveis globais. Embora o uso de variáveis globais seja desaconselhado de uma forma geral, no caso dos sistemas embebidos onde o desempenho é crucial, esta prática pode permitir poupar tempo de programa.

- Adicionar o prefixo REGISTER a variáveis que sejam usadas repetidamente, nomeadamente iteradores de ciclos. Esta diretiva dá indicação ao compilador para usar registos do processador em vez de variáveis em memória. O resultado não é garantido, depende do programa e da arquitetura do CPU.

- Fazer uso dos canais *Direct Memory Access* (DMA), se existirem. Um controlador DMA permite a ligação direta entre um periférico e a memória, e a transferência de dados sem ter de passar pelo processador. Sem DMA, a leitura de uma sequência de valores de um sensor é feita da seguinte forma: 1) ler valor do sensor para uma variável temporária, 2) copiar valor da variável temporária para um vetor, 3) incrementar o contador de amostras. Com DMA, o controlador é configurado para copiar N amostras obtidas do sensor para um determinado endereço de memória, correspondente ao vetor para guardar as amostras. Na familia Arduíno apenas o MCU do Due tem DMA.

- Não usar a leitura de valores analógicos para interpretar eventos binários, por exemplo um botão de pressão. A leitura do porto analógico demora mais tempo do que um porto digital. Exceção caso não existam pinos digitais livres.

- Usar o maior débito possível do canal de comunicação, para perder menos tempo e energia com a transmissão.

- Evitar usar `print()`/`println()`. Se for necessário usar uma codificação que minimize o número de carateres, por exemplo hexadecimal para mostrar valores inteiros.

- Desligar os componentes que não estão a ser usados. Se um periférico só é usado 5% do tempo de execução do programa, não há motivo para desperdiçar os restantes 95%. Importa ter em mente que ligar e desligar um periférico só por si introduz um acréscimo de instruções.

- Usar estruturas (`struct`) para combinar variáveis de vetores a processar. O compilador coloca as variáveis em posições de memória de forma contígua, e os tempos de acesso à memória são menores para acessos a posições consecutivas. Um programa que em cada iteração use vários vetores vai ter de aceder a posições de memórias não consecutivos, o que introduz penalização no desempenho. Um vetor do tipo estruturas faz com que cada elemento do vetor tenha em posições consecutivas os

elementos que estariam em posições distantes de memória. A listagem 3.3 mostra o código so Sketch usado para avaliar o uso da `struct` num Arduíno Leonardo (ATmega32u4). A macro `USE_STRUCT` indica se a estrutura é usada ou não. A macro `ARRAY_SIZE` define o tamanho dos vetores. Dadas as limitações de memória no Arduíno só é possível realizar um ensaio de cada vez. A listagem 3.4 mostra o resultado. Em média, o uso da `struct` melhora o desempenho em 4% em relação ao uso de variáveis independentes.

Listagem 3.3: Demonstração da utilização de estruturas para aumentar o desempenho

```
// use USE_STRUCT and ARRAY_SIZE to change test conditions
#define USE_STRUCT  1
#define ARRAY_SIZE  200

typedef struct {
  byte x;
  int y;
  float z;
} s_t;

// need to declare a struct or individual arrays,
// no memory for both
#if USE_STRUCT
volatile s_t a[ARRAY_SIZE];
#else
volatile byte ax[ARRAY_SIZE];
volatile int ay[ARRAY_SIZE];
volatile float az[ARRAY_SIZE];
#endif
unsigned long t1, t2;

int i = 0;
volatile byte X = 0;
volatile int Y = 0;
volatile float Z = 0.0;

void setup() {
  Serial.begin(9600);
  while (!Serial);

  t1 = micros();
  for(i = 0; i < ARRAY_SIZE; i++) {
#if USE_STRUCT
    X += a[i].x;
    Y += a[i].y;
    Z += a[i].z;
#else
    X += ax[i];
    Y += ay[i];
    Z += az[i];
#endif
  }
  t2 = micros();
  Serial.print(ARRAY_SIZE, DEC);
  Serial.print(USE_STRUCT == 0 ? " element individual arrays: " :
                                 " element struct arrays: ");
  Serial.print(t2 - t1, DEC);
  Serial.println(" us");
}

void loop() {
  delay(1000);
}
```

Listagem 3.4: resultado da avaliação da utilização de estruturas vs arrays independentes

```
32 element individual arrays: 312 us
32 element struct arrays: 300 us
100 element individual arrays: 960 us
100 element struct arrays: 924 us
200 element individual arrays: 1916 us
200 element struct arrays: 1840 us
```

- Modificar ou substituir o algoritmo para executar menos instruções ou menos memória. A listagem 3.5 mostra um programa Arduíno para trocar os valores entre duas variáveis sem ser necessário ocupar o espaço de uma terceira variável (temporária). Comparado com a prática habitual, este método usa o mesmo número de instruções.

Listagem 3.5: varSwap.ino

```
unsigned int a = 0xCAFE; // Hexspeak
unsigned int b = 0xB01A;

void setup() {
  Serial.begin(9600);
  while (!Serial) ; // wait for serial port to connect.

  a = a xor b;
  b = b xor a;
  a = a xor b;
  Serial.println(a, HEX);
  Serial.println(b, HEX);
}

void loop() {
  delay(100);
}
```

3.5 Resposta em Tempo-Real

Nalgumas aplicações é importante que várias operações sejam executadas ao simultâneo, e (começarem dar) uma resposta imediata. Na programação de sistemas para ter este tipo de resposta é habitual organizar o programa em tarefas, que são executadas em simultâneo, ou de forma concorrente, por um núcleo (*Kernel*).

A ideia é dar a ilusão de que várias tarefas são executadas em paralelo, "partindo" cada tarefa em pequenas fatias (conjunto de poucas instruções), e executar uma fatia de cada tarefa de forma intervalada. Uma tarefa tem 3 estados possíveis: em execução no CPU, pronta para correr ou bloqueada à espera de um evento. Um núcleo pode ser baseado em eventos, ou por partilha de tempo, sendo a ativação de cada tarefa indicada pelo escalonador .

O escalonador do núcleo pode ser:

- Preemptivo - Aloca uma fatia de tempo a cada tarefa e interrompe-a para prosseguir para outra tarefa;

  - Escalonamento "à vez" (*Round-Robin*) - distribui de igual forma o tempo do processador por todas as tarefas em execução;
  - Escalonamento com secção crítica - tarefas com secção crítica não são interrompidas;
  - Escalonamento estático - cada tarefa tem o seu tempo de execução, mesmo que não existam outras para executar;

- Cooperativo, ou não-preemptivo - cada tarefa deixa de prosseguir com a sua tarefa para dar oportunidade a que outras tarefas que possam ser executadas.

- Prioritário - coloca em execução primeiro e dá mais tempo de execução a tarefas com prioridade maior;

- Tarefas mais breves primeiro (*Shortest-Remaining Processing Time*) - dá prioridade às tarefas que demoram menos tempo a executar.

Num sistema computacional o suporte para este tipo de execução é dado pelo sistema operativo, um programa que é carregado assim que o sistema inicia, e que gere todos os recursos *hardware* e *software*, bem como a execução de todas as aplicações no sistema. Os sistemas operativos mais comuns em computadores são o Linux e o Windows. No caso do Arduíno as aplicações são executadas diretamente da memória sobre o CPU, pelo que não existe suporte.

3.6 Tolerância a Falhas

A tolerância a falhas é a propriedade que os sistemas têm em continuar o seu funcionamento normal, mesmo na presença de falhas de alguns dos seus componentes. Em aplicações críticas, como o controlo de equipamentos que podem colocar em risco vidas humanas, é imprescindível que o sistema não se comporte de forma errada.

Um sistema embebido pode deixar de funcionar como pretendido por diversos motivos, contudo existem sistemas que, dada a sua importância no desempenho de uma função, ao deixarem de funcionar corretamente vão provocar consequências com um custo demasiado elevado. Considere-se por exemplo um sistema que gere um semáforo para veículos automóveis e outro para os peões na passadeira. É inadmissível que alguma vez os dois semáforos fiquem verde ao mesmo tempo.

As falhas num sistema podem ser provocadas por diversos motivos, tais como falhas no fornecimento de energia, excesso de temperatura, interferências eletromagnéticas ou descargas de radiação, defeitos de projeto e fabrico do sistema, componentes defeituosos ou degradados, ou o sistema foi projectado

para funcionar no limiar das condições do funcionamento normal. Existem alguns testes aquando do desenvolvimento e fabrico que permitem identificar algumas destas falhas, mas quando a complexidade do sistema aumenta é difícil garantir que todas foram observadas. Caso um sistema tenha requisitos de tolerância a falhas, devem ser avaliados os requisitos do sistema e introduzir mecanismos de recuperação de erros na especificação do projeto quanto possível para facilitar a verificação do seu funcionamento.

Num sistema embebido os mecanismos de tolerância a falhas podem ser realizados em circuito (hardware) ou programação (software). Para cada um deles existem vantagens e desvantagens.

Como mais tarde ou mais cedo todos os sistemas falham, eles são caraterizados, em relação a falhas pela seguintes métricas:

- *Mean Time To Failure* (MTTF) - Tempo médio até à próxima falha;
- *Failure Rate* (FR) - tempo médio entre falhas;

Um sistema embebido deve ser dimensionado de tal forma que tempo de médio para a ocorrência de falhas seja superior ao tempo de vida estimado.

3.6.1 Cão de Guarda (Watchdog)

"Cão de guarda" é a tradução do nome do mecanismo conhecido em inglês como *Watchdog*. Um *Watchdog* é um mecanismo que monitoriza uma determinada atividade como sendo indicadora de funcionamento correto. Existem diversos tipos de *Watchdog*, sendo os mais usados em sistemas embebidos os de monitorização da tensão de alimentação e *Heartbeat/Keepalive*. O primeiro tipo de *Watchdog* monitoriza a tensão de alimentação do sistema embebido. Se em algum instante o nível de tensão for inferior a um valor que ponha em causa do seu funcionamento correto, o sistema é reiniciado. O *Keepalive* é normalmente realizado com um mecanismo que deteta pulsos, ou mensagens, enviados periodicamente pelo sistema embebido. Caso o pulso, ou mensagem, não seja detetado, é assumido que o sistema deixou de ser funcionar corretamente, e deve ser reiniciado, por exemplo através da ativação do sinal de RESET. A realização mais simples é através de um temporizador que conta um período de tempo, após o qual, se não receber o pulso, desencadeia a reiniciação do MCU. Desta forma, o sistema embebido deve enviar periodicamente um sinal para reiniciar a contagem para evitar ser reiniciado. O *Keepalive* é também usado no contexto de redes de comunicação para determinar se a ligação continua ativa, e se o sistema do outro lado se encontra responsivo.

3.6.2 Redundância Temporal e Espacial

Quando um sistema embebido executa uma instrução, se existir alguma perturbação no sistema, o seu resultado pode ser errado/invalidado. Uma forma de contornar este problema consiste em várias execuções das mesmas instruções, e depois verificar qual a resposta mais comum através de um votador de maioria. Caso ocorra um erro espúrio numa das execuções, o esquema de votação irá selecionar a resposta certa a partir das restantes execuções corretas. Este tipo de redundância chama-se redundância temporal, pois executa várias réplicas das computações a proteger várias vezes ao longo do tempo. Este tipo de esquema tem de usar exatamente os mesmos dados e as mesmas instruções, sob pena de produzir respostas diferentes e não funcionar.

No caso em que um elemento do sistema se encontra degradado e nunca dá a resposta correta, o esquema de redundância temporal não é suficiente para mitigar o erro, pelo que é necessário recorrer a outro sistema que produza a mesma resposta. Idealmente, este sistema deve ser diferente do original e o programa escrito por outra pessoa para evitar que defeitos de fabrico e erros de programação sejam propagados.

A redundância espacial faz uso do mesmo princípio de votar sobre o resultado da execução de várias execuções da mesma operação, mas neste caso em vez de serem executadas várias vezes consecutivas, elas são calculadas simultaneamente em unidades distintas. Caso o dispositivo não suporte a execução de várias réplicas da mesma operação em simultâneo, devem ser usados outros sistemas para a sua implementação. Usar vários dispositivos tem a vantagem de manter o sistema em funcionamento mesmo que uma réplica fique avariada.

A redundância pode ser dupla, tripla ou até quíntupla. Se for dupla, apenas é possível sinalizar a existência de erro, pois não se sabe qual das duas unidades está errada. A redundância tripla é a usada mais frequentemente pois permite corrigir o erro através da identificação do resultado mais comum. É possível combinar a redundância temporal e espacial, em que cada unidade calcula o mesmo resultado N vezes, elege o mais comum, e por sua vez esse resultado vai ser comparado com o das restantes réplicas.

Como a redundância tripla implica repetir o mesmo sistema 3 vezes, existem propostas para alternativas mais simples, em que as réplicas têm precisão reduzida. Com isto consegue-se ter uma implementação diferente e que demora menos tempo a ser calculada. Neste caso importa que a aplicação permita a introdução de pequenos desvios nos cálculos. Este método não é bom para aplicações que envolvam transferências monetárias.

3.6.3 Pontos de Verificação (Checkpointing)

O *Checkpointing* consiste em guardar o estado do sistema num determinado instante da sua execução, e caso um erro seja detectado, o último estado

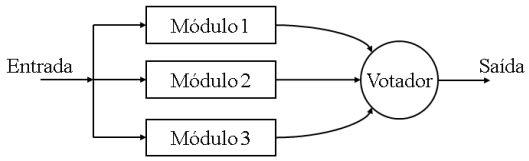

Figura 3.3: Redundância tripla com votação.

livre de erros é recuperado e volta-se a repetir a execução. A verificação da existência de erros pode ser através da utilização de réplicas ou de valores bem conhecidos para diferentes estados de execução da aplicação.

3.6.4 Consenso (Lockstep)

No *Lockstep* a mesma operação é executada em paralelo em sistemas diferentes, e os seus resultados comparados. Para implementar o *Lockstep*, os sistemas avançam para o mesmo estado, processam os mesmos dados, e produzem resultados. Se os resultados forem concordantes, ambos os sistemas avançam para o estado seguinte. Este método usa redundância espacial para detectar e corrigir erros automaticamente.

3.6.5 Auto-Verificação

A auto-verificação de um sistema passa pela sujeição do sistema à execução de um teste para o qual já se sabe qual o resultado esperado. Executando este teste periodicamente, é possível aferir se as unidades se encontram a dar a resposta certa. Por exemplo, para testar se as unidades de vírgula flutuante, que possuem circuitos mais complexos e mais propensas a erro, podem-se realizar várias operações aritméticas com valores existentes num vetor ou saídos de um gerador pseudo-aleatório com uma semente conhecida, e comparar com o resultado pré-calculado.

3.6.6 Degradação Graciosa

Num sistema em que não exista qualquer mecanismo de tolerância a falhas, a existência da menor falha pode fazer com que o sistema deixe de responder. A degradação graciosa é capacidade de ter o sistema em funcionamento, mas com menor funcionalidade, dependendo da gravidade da falha. Por exemplo, se a execução de uma função que envolve vários cálculos falhar, uma alternativa é a realização de um cálculo aproximado, ou reduzido. Por exemplo, se as contas em vírgula flutuante estão a dar resultados errados

3.6.7 Falhas Bizantinas

Uma falha bizantina é definida como ter um (um elemento do) sistema em execução de forma anormal, mas o mecanismo de deteção de erros não consegue identificar tal anomalia. Em sistemas embebidos é frequente ter um cenário onde dois sistemas embebidos autónomos recebem valores de medidas de vários sensores (temperatura, ângulo de ataque, tensão), mas ambos recebem valores diferentes para a mesma medida. Eles comunicam entre si o valor recebido, e conseguem perceber que algo está errado pois têm valores diferentes, mas não conseguem identificar o problema por muitas mensagens que troquem. A solução para este problema passa por ter o valor medido enviado para vários sistemas, e os sistemas comunicarem entre si quais os valores recebidos, dentro de uma janela de tempo. Findo o período de tempo para receber as respostas, cada sistema vai usar o valor mais comum ou um valor por omissão caso alguma leitura não tenha sido recebida. Desta forma os valores errados são eliminados.

3.7 Paradigmas de Programação e Padrões Arquiteturais

A forma como um programa é escrito, a linguagem de programação, e os mecanismos de programação utilizados podem limitar significativamente a facilidade de programação e o desempenho do programa. Habitualmente os sistemas embebidos são programados em C ou C++. Estas linguagens permitem "controlar" vários aspetos de programação e execução do programa, que outras linguagens como Python não permitem. No entanto, obriga o programador a prestar atenção a detalhes que poderão tornarem-se em fontes de erros se não forem devidamente considerados.

A engenharia de software trata do desenvolvimento de software, através de paradigmas de programação e de padrões arquiteturais. Os padrões arquiteturais são formas, ou fórmulas, genéricas de construir soluções em software para diversos tipos de problemas.

Como os sistemas embebidos não são realizados todos da mesma forma, nem partilham dos mesmos requisitos, esta secção apresenta alguns padrões por forma a auxiliar o projecto e desenvolvimento do software. Os mais relevantes para o contexto de sistemas embebidos são: *Model-View-Controller* (MVC), *Blackboard*, *Event-Driven*, Cliente-Servidor e Produtor-Consumidor. Não existe um padrão arquitetural que possa ser usado como solução ideal para todos os sistemas embebidos. Em alguns sistemas de média ou elevada complexidade pode ser necessário combinar vários padrões arquiteturais para resolver diferentes problemas dentro da mesma aplicação.

3.7.1 Programação Orientada a Objetos

O ambiente Arduíno suporta a programação em C++, o que significa que se pode fazer uso do paradigma da Programação Orientada por Objetos. Este paradigma permite abstrair o programa de tal forma que o programa é escrito com vista a interação de objetos, em vez de procedimentos e variáveis.

A vantagem da ProgramaÃ§Ã£o Orientada a Objetos - *Object-Oriented Programming* (POO) é a especificação dos objetos através de classes, ao passo que na programação procedimental, por exemplo em C, os dados são guardados em variáveis, e processados em funções. O comportamento de um objetos é definidos através de uma classe, e é o equivalente a uma estrutura de dados que contém variáveis e métodos, ou funções. Quando se declara uma variável de uma determinada classe, está-se a instanciar um objecto. Por exemplo, num programa para ler uma medida de um sensor, vamos definir uma classe chamada Sensor. Esta classe tem uma variável do tipo `int` chamada `valorLido`, e a função `int lerSensor()`.

Outra vantagem da POO é a possibilidade da definição de uma hierarquia de objetos, ou seja, permite que um objeto (filho/a) seja derivado de outros (pai/mãe), ou seja, criado à custa de outros. Prosseguindo com o exemplo do sensor, particularizando o tipo de sensor usado, pode-se ter um sensor analógico de temperatura, humidade, luminosidade, som, etc. Neste caso podem-se definir novas classes particularizadas para cada um destes sensores, herdando a funcionalidade da classe mãe. Desta forma a classe `SensorTemperatura`, deriva da classe Sensor e fornece um novo método int `converteTemp()` que converte o valor numérico do sensor num valor de temperatura em graus centígrados.

Em POO, é possível definir a abrangência dos atributos e métodos das classes, através de 4 identificadores: `public`, `friend`, `protected` e `private`. Os elementos públicos são vistos em qualquer âmbito do programa onde o objecto esteja instânciado, e são usados para interagir com o objecto. Os elementos privados são apenas vistos dentro da classe, nomeadamente para atributos e funções que não devem ser manipulados fora da classe. Por exemplo, o valor lido do sensor não deve ser manipulado, devendo ser escrito apenas pelo método responsável pela leitura do sensor, para garantir integridade dos valores lidos. A figura 3.4 mostra a relação entre os diferentes níveis de abrangência e os diferentes níveis de hierarquia de um objecto.

3.7.2 Model-View-Controller

Este padrão arquitetural foi pensado para o desenvolvimento de interfaces com o utilizador, sendo muito popular em aplicações para a Internet. Como a maioria dos sistemas embebidos requer alguma interação com o utilizador, e com o meio, este padrão pode ser usado com o mesmo princípio. O *Model-View-Controller* (MVC) baseia-se na passagem de informação entre

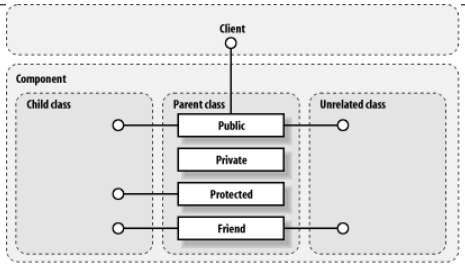

Figura 3.4: Esquema da relação de classes derivadas e modificadores de acesso.

três módulos, ou funções, executadas sequencialmente:

1. Modelo (*Model*) - é o coração do sistema, independente da apresentação e interface do sistema, ele incorpora todos os mecanismos necessários para fazer a aplicação funcionar e passa os dados de apresentação à Vista;

2. Vista (*View*) - funciona como "mostrador" do sistema, para passar informação ao utilizador. É responsável por todos os atuadores;

3. Controlador (*Controller*) - serve para receber os comandos do utilizador ou de sensores, validá-los e passá-los ao Modelo.

Em cada iteração da execução destas funções, os dados vão sendo tratados de acordo com a funcionalidade. Este padrão é interessante pois permite separar o programa responsável pela apresentação, controlo e modelo tendo as seguintes vantagens:

- delegar o desenvolvimento de cada uma das funções a equipas diferentes;

- facilitar a especificação do sistema;

- e facilitar a manutenção do programa.

. A figura 3.5 mostra a relação dos diferentes módulos MVC.

O modelo MVC adequa-se à programação para Arduíno. Por exemplo para realizar um termostato automático para um equipamento de ar condicionado, as funções setup() e loop() poderiam ter o seguinte conteúdo:

```
void setup() {
    configTempSensor();
    configDriverAC();
}
void loop() {
    controller();    // read temperature sensor and target temperature
    model();         // decide whether to make hot or cold
    view();          // show current temperature and activate heating or cooling elements
}
```

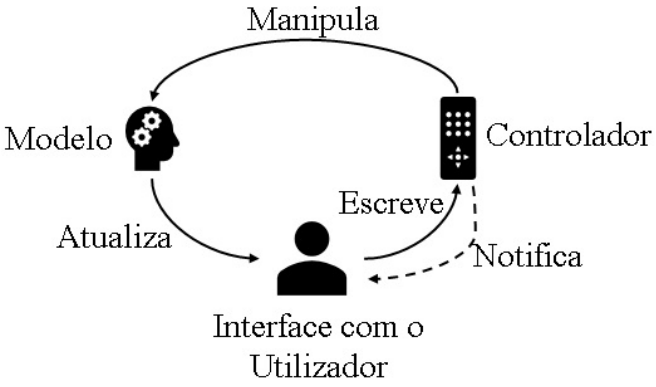

Figura 3.5: model-view-controller.

É importante ter em mente ao projectar uma aplicação para este modelo que nenhuma função pode ser bloqueante, ou seja, a maior ou menor rapidez vai definir a resposta do sistema. Se num sistema de controlo de temperatura não existe necessidade de um controlo muito apertado, permitindo oscilações nos valores de temperatura, já num sistema de condução autónoma, é imprescindível que os sensores sejam lidos o mais rápido possível, não podendo o sistema ficar à espera da resposta de componentes que demorem muito tempo a responder.

3.7.3 Blackboard

O modelo Blackboard, tal como nome sugere, funciona como um quadro onde são feitas anotações e leituras pelos vários componentes. Na prática, corresponde a ter um conjunto de variáveis e objetos partilhados, definidas como variáveis globais. Se por um lado as boas práticas de programação definem que declarar variáveis globais é uma prática de programação perigosa, por permitir que qualquer função altere os seus valores, por outro, se bem empregue, é vantajoso porque permite facilitar o acesso aos dados pelas diferentes funções sem ter necessidade de usar esquemas de comunicação ou passagem de argumentos de funções. É importante observar quem são os leitores e escritores das variáveis, e garantir que não existem escritas concorrentes, sob pena de se perderem dados ou haver funcionamento anómalo.

A passagem de dados como argumentos de funções é feita através da pilha (Stack). Num sistema embebido a pilha é um recurso bastante limitado, e que quando esgotado tende a gerar erros intermitentes difíceis de diagnosticar. Num sistema embebido para aquisição e processamento de imagem, com o Blackboard, usa-se uma variável global para conter toda a imagem a ser processada, seguindo a implementação ilustrado na listagem 3.6.

Listagem 3.6: Exemplo do modelo Blackboard

```
int image[8192];

void loop() {
    configCamera();
    configDisplay();
}

void loop() {
    getImage();
    processImage();
    displayImage();
}
```

3.7.4 Event-Driven

O modelo Event-Driven no âmbito de sistemas embebidos é usado para realizar programas que reagem a eventos externos ao MCU. Ou seja, em vez de executar as funções por um determinada sequência pre-definida, na maior parte do tempo o programa encontra-se parado à espera que "algo aconteça", como por exemplo mudança de estado de um pino associado a um botão de pressão. Para suportar tal funcionalidade, é necessário indicar no programa:

- Quais os eventos a que o sistema vai ficar sensível - máscara de interrupções;
- Associar uma função a realizar a cada evento, e a caraterização do evento - associar a rotina de atendimento a interrupções a um sinal de E/S digital;
- Ativar o atendimento de eventos - ativar tratamento de interrupções.

A listagem 3.8 ilustra a utilização de eventos externos para despoletar a execução de funções.

Listagem 3.7: Exemplo da utilização de interrupções.

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void isr() {
  state = !state;
}

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), isr, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
}
```

Num contexto do software, os eventos são suportados por funções de *CallBack*, que são associadas a instâncias de objetos e invocadas aquando

de uma determinada condição avaliada internamente, evitando adicionar a avaliação dessa condição explicitamente no corpo do programa.

Listagem 3.8: Exemplo da utilização de função de *callback*.

```
// demonstraÃ§Ã£o da funÃ§Ã£o callback

const int swPin =  3;       // input pin number
const int ledPin =  13;       // LED IO pin number
bool ledState = LOW;

class AlDemo {
private:
  void (*actionListener)(byte); // function to be executed upon activity
public:
  void scan() {
    if(digitalRead(swPin) == LOW) {
      ledState = !ledState;
      if(actionListener != NULL) {
        actionListener(ledState);
      }
    }
  }
  void setActionListener(void (*al)(byte)) {
    actionListener = al;
  }
};

AlDemo obj1;

// function external to the obj class
void actionPerformed(byte val) {
  if(val == HIGH) {
    digitalWrite(ledPin, HIGH);
  }
  else {
    digitalWrite(ledPin, LOW);
  }
}


void setup() {
  pinMode(swPin, INPUT_PULLUP);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);  while(!Serial);
  Serial.println("*** Action Listener Demo ***\n");
  obj1.setActionListener(actionPerformed);
}

void loop() {
  obj1.scan();
  delay(10);
}
```

A listagem ilustra a chamada de uma função (*callback*) externa por um objeto da classe `AlDemo`. Neste Sketch os objetos da classe possuem o atributo privado `void (*actionListener)(byte);`, onde o método void `setActionListener(void (*al)(byte))` regista a referência da função a ser chamada pelo objeto da classe usando `actionListener(ledState)`. função `actionPerformed(byte val)`

3.7.5 Cliente-Servidor

Em IoT os sistemas embebidos são usados principalmente para recolher informação de um ambiente remoto, fazer o pré-processamento dos dados re-

colhidos e de seguida enviar a informação resultante para um servidor que agrega e processa os dados contribuídos por todos. O modelo Cliente-Servidor para um nó IoT enviar mensagens de telemetria (com e sem garantia) para envio de dados recolhidos pelo nó, e comando-e-controlo para a instalação de configurações ou atualizações do software do nó. As principais vantagens do uso desta arquitetura são:

- Separação da funcionalidade dos vários elementos - um cliente não tem que conter o programa que corre remotamente no servidor, apenas tem de conhecer o seu protocolo de comunicação;
- Segurança/distribuição/gestão da informação numa entidade centralizada;
- Utilização de tecnologias mais baratas e acessíveis, em vez de investir em diversos sistemas complexos.

A maior desvantagem deste modelo prende-se com a indisponibilidade do servidor para atender pedidos dos clientes (sobrecarga de pedidos, falha no sistema, ou falta de energia elétrica). A ligação dos clientes ao servidor pode ser feita com recurso a diferentes tecnologias de telecomunicações com e sem fios, tais como: Ethernet, LORA, ou Sigfox. A figura 3.6 ilustra o esquema de ligações com vários clientes na Internet que se ligam ao mesmo servidor.

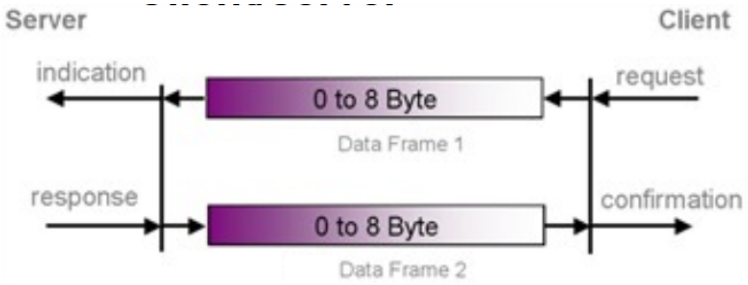

Figura 3.6: Modelo Cliente-Servidor.

3.7.6 Mestre-Escravo

Os sistemas embebidos têm barramentos que podem ser partilhados por vários periféricos. A coordenação do barramento é feita pelo mestre, que indica aos escravos quando devem comunicar e que dados devem processar. O mestre é o sistema embebido e os escravos os componentes periféricos. Exemplos de aplicação são os barramentos *Controller Area Network* (CAN), *Inter-Integrated Circuit* (I2C) e Serial Peripheral Interface (SPI). Este padrão arquitetural também pode ser usado numa aplicação IoT, onde o mestre coordena e concentra os envios de dados dos nós para um hub IoT. Os escravos

nunca tomam iniciativa de comunicar sem ordem do mestre. A figura 3.7 ilustra este modelo.

figura mestre-escravo

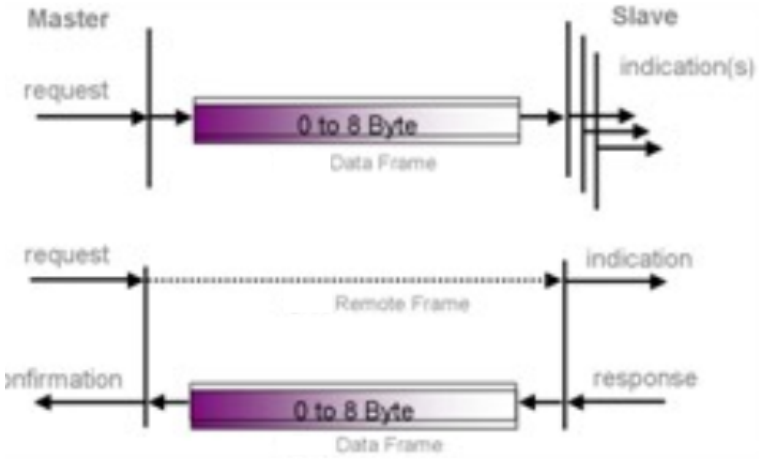

Figura 3.7: Modelo Mestre-Escravo.

3.7.7 Produtor-Consumidor

O modelo Produtor-Consumidor envolve um produtor a enviar um conjunto de dados a vários consumidores, ou um dos consumidores a pedir que o produtor lhe envie dados. A figura 3.8 ilustra este modelo.

figura Produtor-Consumidor

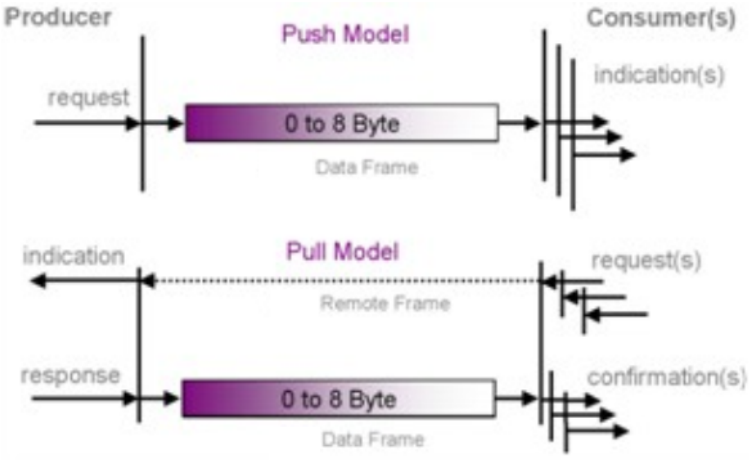

Figura 3.8: Modelo Produtor-Consumidor.

3.8 Controlo

O controlo é uma disciplina dedicada ao estudo e desenvolvimento de sistemas que controlam um determinado dispositivo ou sistema. Um controlador é um mecanismo que faz com que um sistema permaneça num determinado estado ou condição. Sistemas de controlo são por exemplo o termostato de um ar condicionado, ou um sistema de navegação autónomo. No primeiro cenário o interesse é manter a temperatura controlada num espaço fechado ao passo que o segundo é manter um veículo numa trajetória.

Existem diversos tipo de controladores, sendo o mais habitual o controlo em malha fechada, onde o controlador vai atuando sobre o sistema, ciclicamente avalia a diferença entre o valor desejado e o valor atual, e corrige a atuação até ele convirja a condição pretendida. Por exemplo, aquecer ou arrefecer a temperatura do ar condicionado para garantir que um espaço se mantenha nos 20^{o}C, independentemente da temperatura ambiente.

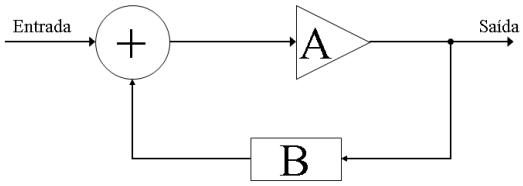

Figura 3.9: Sistema de controlo genérico com malha fechada.

3.8.1 Controlador ON-OFF

O controlador ON-OFF é o controlador mais básico. Define-se um limiar de ativação, e acima desse valor o sistema a controlar encontra-se ligado. O exemplo mais comum é o termostato de uma câmara frigorífica. Este controlador é indicado para o controlo de sistemas que apenas admitem uma entrada binária no seu controlo - em funcionamento ou não.

Tendo em conta que existe sempre ruído associado a uma medida de um sensor (neste caso o sensor de temperatura), perdas de temperatura derivadas do abrir a porta da câmara, e a temperatura do ar demorar a espalhar pela sala, vão fazer com que o sistema fique a oscilar no estado "liga-desliga" constante em torno do valor da temperatura limite, reduzindo drasticamente o tempo de vida do material.

Para evitar este efeito, define-se uma histerese, que é quantidade à volta do limiar onde controlador não atua. Considerando um sistema de controlo de temperatura do ar para 20^{o}C, e com uma histerese de 5^{o}C, ligado a um motor de geração de ar frio, vai manter o motor em funcionamento até que a temperatura chegue aos 20^{o}C, e só o volta a ligar quando a temperatura subir até aos 25^{o}C.

A listagem 3.9 corresponde ao Sketch de um controlador ON-OFF que recebe o valor da leitura de uma entrada analógica no pino A0 e atua sobre o LED na saída D13. O controlador tem por base a classe OnOffCtrl, para permitir que no mesmo programa possam ser instanciados vários controladores. Esta classe tem como atributo privado ponteiros para a entrada e saída do controlador (`bool *output, T *input`), bem como o valor limite (`bool *target`), o valor do limiar de atuação (`T hister`), e a lógica de atuação (`activation = HIGH/LOW`), e são passados ao objeto como parâmetros do construtor. O controlador começa com a saída desativada (`!activation`). O método updateCtrl() avalia a entrada e determina qual o estado da saída do controlador. O classe foi definida com um template para os valor de entrada, limite e para a histerese para ser usada com tipos de dados, por exemplo inteiros (byte, int, long) e reais (float, double).

Listagem 3.9: Sketch Controlador ON-OFF

```
/* demo ON-OFF Ctrl
 *  output: LED @pin 13
 *  input: potenciometer @A0
 */

#define CTRL_LOOP_TIME  10

template <class T>
class OnOffCtrl {
  private:
    bool activation;
    bool *output;
    T *input, *target, hister;
  public:
    OnOffCtrl(T *inputParam, T *targetParam, bool *outputParam, T histerParam, bool activationParam) {
      input = inputParam;
      target = targetParam;
      this->hister = histerParam;
      output = outputParam;
      activation = activationParam;
      *output = !activationParam; // start off
    }
    void updateHister(T histerParam) {
      hister = histerParam;
    }
    void updateCtrl() {
      if(((T)*input - (T)*target) > hister ) {
        *output = 1; //activation;
      }
      else if((T)*input < (T)*target ) {
        *output = 0; //!activation;
      }
      else {
        // do nothing, keep previous value
      }
    }
    bool status() {
      return *output;
    }
    void printInfo() {
      Serial.print("Input: "); Serial.print(*input, DEC);
      Serial.print("  Target: "); Serial.print(*target, DEC);
      Serial.print("  Histerese: "); Serial.print(hister, DEC);
      Serial.print("  Activation: "); Serial.print(activation, DEC);
      Serial.print("  Output: ");  Serial.println(*output, DEC);
    }
};

uint16_t input = 123, target = 300;
```

```
bool output, oldOutput;
char buf[40];
long oldTime;
OnOffCtrl <int16_t> ctrl(&input, &target, &output, 10, HIGH);

void setup() {
  Serial.begin(115200);
  while(!Serial);
  Serial.println(".:ON/OFF Controller Demo:.");
  ctrl.printInfo();
}

void loop() {
  if(millis() - oldTime > CTRL_LOOP_TIME) {
    input = analogRead(A0);
    ctrl.updateCtrl(); // use of pointers to avoid passing values
    digitalWrite(13, output);
    oldTime = millis();
  }
  if(oldOutput != output)
  { // update on changes
    ctrl.printInfo();
    oldOutput = output;
  }
  delay(100);
}
```

3.8.2 Controlador PID

O controlador Proporcional, Integral e Derivativo (PID) foi inicialmente desenvolvido para o sistema de navegação de barcos, como forma de compensar a força das correntes marítimas. O controlador PID tem o seu nome derivado das três componentes que o compõem: Proporcional, Integral e Derivativa. Cada componente contribui para a ativação do sistema com um determinado fator Kp/Ki/Kd, em relação à diferença entre o valor especificado e obtido (Erro). Kp é o fator proporcional à diferença entre os dois valores, Ki, é o fator do erro acumulado ao longo do tempo e Kd é a diferença "imediata" O valor de saída é a soma das três contribuições.

Para ilustrar o funcionamento do PID vamos considerar o mesmo sistema de controlo de temperatura de uma sala. É sabido que: a temperatura de um corpo não varia de forma instantânea, e que existem perdas de temperatura na natureza, e por consequência para aquecer o ar a 30ºC é necessário aquecê-lo com temperatura superior a 30ºC, e para arrefecer abaixo dos 20ºC é necessário arrefecer com uma temperatura inferior a 20ºC.

A componente Proporcional contribui para a potência aplicada à resistência de aquecimento do ar de forma proporcional com diferença entre a temperatura desejada e a temperatura atual. Se o ar estiver a 20ºC e a temperatura desejada for 30ºC, a resistência irá receber indicação para aquecer mais do que se o ar já estiver a 25ºC. Quando a temperatura do ar estiver próxima dos 30ºC a quantidade de aquecimento fornecido será muito menor. No entanto, a temperatura do ar jamais chegará aos 30ºC (teoricamente apenas no infinito) devido a perdas nos vários elementos do sistema: não linearidades dos componentes, perdas de temperatura, que-

das de tensão nos condutores, desvios no modelo, e tolerância no fabrico dos componentes. A componente proporcional é obtida diretamente do valor na entrada do controlador multiplicada pela constante proporcional Kp: $Output(n) = Kp.Error(n)$.

Para compensar o desvio da componente Proporcional do controlador, é adicionada a componente Integral. Esta componente contribui com o integral[2] do erro entre o valor desejado para o sistema e o seu valor atual, ou erro acumulado. Se com a componente Proporcional o sistema atingir os 27ºC, a componente integral irá acumular o desvio de 3ºC ao longo do tempo. Desta forma, a resistência de aquecimento irá receber indicação para aquecer mais que para os 30ºC (33, 36, 39...). À medida que a temperatura do ar se aproxima dos 30ºC, o erro acumulado será cada vez menor. Se a temperatura limite for excedida o erro terá um valor negativo e vai baixar a temperatura da resistência de aquecimento. A componente integral é obtida através do cálculo do integral do erro, multiplicado pela constante Ki: $Output(n) += Ki.Error(n)$. O cálculo do integral passa pela acumulação do valor do sinal de erro desde a iniciação do controlador ou durante as últimas N amostras para reduzir o impacto das condições iniciais na convergência do sistema.

A componente Derivativa serve "acelerar a convergência do sistema" para o valor desejado. A componente derivativa atua sobre o ritmo com que os valores na saída do sistema variam ao longo do tempo[3]. Se o ar estiver a ser aquecido apenas com a potência necessária para chegar à temperatura desejada (30ºC), vai levar mais tempo do que se for inicialmente, e apenas durante algum tempo, aquecido para a temperatura de 40 ou 50ºC. Isto é o que se faz instintivamente quando se liga o ar condicionado num dia de inverno (ou para arrefecer no verão). Neste exemplo isto vai fazer com que o ar chegue mais rapidamente a um valor próximo do desejado. No entanto é necessário ter em atenção que ao aquecer um corpo a uma temperatura elevada, todo o espaço aquecido vai manter a temperatura durante algum tempo e o ar poderá ultrapassar o valor limite (inércia térmica). O sistema ao medir um valor elevado irá depois provocar a variação contrária, e fazer com que a temperatura cresça rapidamente no sentido oposto, provocando um comportamento oscilatório. A componente Derivativa é obtida através do cálculo da derivada do erro, multiplicado pela constante Kd. O cálculo da derivada é a diferença entre do valor do sinal de erro atual subtraído do valor do mesmo sinal no instante anterior: $Output = Kd.(Error(n) - Error(n-1))$.

[2] O integral é uma operação matemática cujo resultado é a soma de todas as contribuições de uma função durante um período finito de tempo, semelhante ao cálculo da área da figura produzida pelo gráfico da função.

[3] A derivada é uma operação matemática que permite calcular a razão de crescimento de uma função em função de uma variável de entrada.

Num projecto com um controlador PID um dos objetivos é conseguir afina-lo para que seja o mais rápido possível a chegar ao valor pretendido, mas sem ter comportamento oscilatório. A figura 3.10 mostra o diagrama de blocos de um controlador PID, onde cada componente é indicada também qual a expressão matemática associada. Neste sistema a entrada é *r(t)*, *u(t)* é o sinal aplicado ao sistema a controlar e *y(t)* é a saída do sistema, e a resposta do controlador é dada pela soma das três componentes.

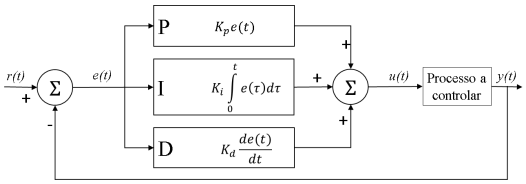

Figura 3.10: Diagrama de blocos de um controlador PID.

A listagem 3.10 mostra o Sketch para implementação de um controlador PID genérico.

Listagem 3.10: Sketch de um Controlador PID.

```
!! ATUALIZAR !!

/* demo PID Ctrl
 *  output: LED @pin 13
 *  input: potenciometer @A0
 */

#define CTRL_LOOP_TIME  10

template <class T> class OnOffCtrl {
  private:
    bool *output;
    T *input, *target;
    float ki, kp, kd;
  public:
    PIDCtrl(T *inputParam, T *targetParam, bool *outputParam, float ki, float kp, float kd) {
      input = inputParam;
      target = targetParam;
      this->hister = histerParam;
      output = outputParam;
      activation = activationParam;
      *output = !activationParam; // start off
    }
    void updateHister(T histerParam) {
      hister = histerParam;
    }
    void updateCtrl() {
      if(((T)*input - (T)*target) > hister ) {
        *output = 1; //activation;
      }
      else if((T)*input < (T)*target ) {
        *output = 0; //!activation;
      }
      else {
        // do nothing, keep previous value
      }
    }
    bool status() {
      return *output;
    }
    void printInfo() {
```

```
      Serial.print("Input: "); Serial.print(*input, DEC);
      Serial.print("  Target: "); Serial.print(*target, DEC);
      Serial.print("  Ki: "); Serial.print(Ki, DEC);
      Serial.print("  Kp: "); Serial.print(Kp, DEC);
      Serial.print("  Kd: "); Serial.print(Kd, DEC);
      Serial.print("  Output: ");  Serial.println(*output, DEC);
    }
};

uint16_t input = 123, target = 300; bool output, oldOutput; char buf[40];
long oldTime; OnOffCtrl <int16_t> ctrl(&input, &target, &output, 10, HIGH);

void setup() {
  Serial.begin(115200);
  while(!Serial);
  Serial.println(".:PID Controller Demo:.");
  ctrl.printInfo();
}

void loop() {
  if(millis() - oldTime > CTRL_LOOP_TIME) {
    input = analogRead(A0);
    ctrl.updateCtrl(); // use of pointers to avoid passing values
    digitalWrite(13, output);
    oldTime = millis();
  }
  if(oldOutput != output)
  { // update on changes
    ctrl.printInfo();
    oldOutput = output;
  }
  delay(100);
}
```

Para sistemas onde o tempo de resposta do sistema e do atuador sejam mais rápidos que o sistema a ser controlado, especialmente durante a fase de arranque, para evitar que saturem num dos extremos, ou produzam atrasos e oscilações, podem-se usar de funções paramétricas ou definir patamares (máximo e mínimo) para limitar o valor na saída do controlador PID.

Exemplos de aplicação de sistemas embebidos são o controlo de direção e velocidade de um veículo (carro, barco, drone). No caso do controlo da direção de um leme, um motor passo-a-passo é o atuador que varia o ângulo do eixo, e um potenciómetro nele acoplado serve como sensor do ângulo. No caso do controlo de velocidade, o atuador é um motor DC e o sensor é um gerador de pulsos acoplado ao eixo do motor.

3.9 Matemática e Métodos Numéricos

Quando um sistema embebido produz e processa muitos valores obtidos de sensores pode ser difícil tirar conclusões através da observação direta dos dados. Uma forma de extrair informação de grandes volumes de dados é através de medidas estatísticas, nomeadamente médias, variância e correlações. Estas medidas permitem dar uma indicação sobre os dados. Uma das áreas da computação em expansão é a "data mining", onde são desenvolvidos novos métodos para extrair informação de vastos conjuntos de dados, nomeadamente através de algoritmos de classificação, clustering, e regressão. De

seguida são revisitadas algumas medidas da estatística descritiva juntamente com outros métodos matemáticos relevantes no contexto de sistemas embebidos.

3.9.1 Medidas Estatísticas

As medidas estatísticas são uma ferramenta importante pois permitem estimar, ou inferir, informações sobre dados recolhidos. Elas são particularmente importantes no tratamento de grandes volumes de dados produzidos por sensores.

Média Aritmética

A média aritmética é a medida estatística mais simples. Ela fornece o valor médio de um conjunto de valores, dado pela soma de todos os valores a dividir pela sua quantidade. A média também é conhecida como valor esperado.

$$\overline{m} = \frac{1}{N} \sum_i a_i = \frac{a_1 + a_2 + \cdots + a_N}{N}$$

No contexto dos sistemas embebidos, a média aritmética é usada em processamento digital de sinal, e recolha de estatísticas para determinados períodos de tempo.

Média Ponderada

Enquanto que na média aritmética, todos os valores contribuem com o mesmo peso, ou importância, a média ponderada é usada para dar mais ou menos peso a algumas amostras. A média ponderada é calculada da seguinte forma:

$$\overline{mp} = \frac{1}{N} \sum_i w_i \times a_i = \frac{w_1 \times a_1 + w_2 \times a_2 + \cdots + w_N \times a_N}{N}$$

A média aritmética é usada em processamento digital de sinal, para reduzir a variação de um sinal devido a ruído.

Média Móvel

A média móvel, ou deslizante, corresponde à média aritmética das últimas N amostras de uma série de valores com K amostras, com $K > N$. Quando uma nova amostra é recebida, a mais antiga é descartada. Este conceito é aplicado no contexto de processamento de sinal, mercados financeiros e meteorologia. Exemplo, $N = 3$:

```
a = [1 4 1]
m = (1+4+1)/3 = 2
```

```
a = [1 4 1 4]
m = (4+1+4)/3 = 3

a = [1 4 1 4 -2]
m = (1+4-2)/3 = 1
```

Raiz da Média Quadrática

A raiz da média quadrática é calculada como a raiz quadrada da soma do quadrado dos valores das amostras a dividir pelo número de amostras:

$$a_{RMS} = \sqrt{\frac{1}{N}\left(a_1^2 + a_2^2 + \cdots + a_N^2\right)}$$

Esta medida é usada para calcular o valor médio de energia num circuito variante no tempo (sinusoide centrada em 0). O valor médio de potência é calculado usando a tensão RMS, que é dada pela tensão de pico a dividir por $\sqrt{2}$: $V_{RMS} = V_p/\sqrt{2}$:

$$P_{sin} = V_{RMS}^2/R$$

Média Geométrica

A média geométrica é usada para comparação de quantidades que não são relacionáveis (não existe figura de mérito, ou unidade de medida comum). A média geométrica de duas amostras (x, y) corresponde à sua norma, e indica a distância da origem do eixo ao centro de massa do plano definido por x e y. A média geométrica de N amostras é calculada como a raiz de ordem N, do produto dos N valores:

$$media_geomtrica = \sqrt[N]{a_1 \times a_2 \times \cdots \times a_N}$$

A média geométrica é usada em processamento de imagem, para filtragem de ruído.

Moda

A moda corresponde ao valor observado com mais frequência. Exemplo: Se forem observadas as seguintes medidas de um sensor $4, 5, 3, 6, 5, 7, 5$, a moda é o valor 5.

Mediana

A mediana é o ponto médio entre os valores mínimo e máximo observados num conjunto de dados, calculado como a média dos valores extremos:

$$mediana = \frac{a_{MIN} + a_{MAX}}{2}$$

Exemplo: Se forem observadas as seguintes medidas de um sensor $4, 5, 3, 6, 5, 7, 5$, a mediana é $(3 + 7)/2 = 5$.

Variância e Desvio Padrão

A variância permite medir quanto "espalhados" estão os valores do valor central de um conjunto de valores. Por definição, a variância é dada pelo quadrado da soma da diferença entre os valores das amostras de um vector e o seu valor esperado (ou média):

$$Var(X) = E[(X - \mu)^2]$$

$$Var(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

A variância corresponde ao quadrado do desvio padrão. O desvio padrão é obtido como a raiz quadrada da variância. Nalguns casos o desvio padrão é preferido por ter o resultado na mesma unidade dos dados observados. A variância e o desvio padrão são usados na prática para quantificar o desvio dos dados reais em relação a um modelo.

Nas aplicações típicas em sistemas embebidos não é possível saber *a priori* qual a média dos dados que estão a ser observados, pelo que a média usada é a do conjunto de valores amostrados.

Correlação

A correlação é a medida que indica a semelhança entre os valores de duas variáveis. Uma aplicação prática é na quantificação da semelhança entre um conjunto de dados esperado (modelo) e as amostras reais medidas. Por exemplo, observação de condições meteorológicas, ou identificação da semelhança entre (partes de) duas imagens. O valor do coeficiente de correlação indica quão relacionados estão dois vetores: 0 nada, 1 completamente. O valor do coeficiente de correlação das amostras de dois vetores é:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

A demonstração do cálculo da correlação com um Arduíno encontra-se no Sketch da listagem 3.11:

Listagem 3.11: corrCoeff.ino

```
#define NUM_SAMPLES  4

float X[] = {.15, .30, .45, .60};
float Y[] = {.10, .20, .30, .40};
float Z[] = {.60, .45, .30, .15};
float K[] = {.30, .85, .10, .45};


// function to compute correlation coefficient.
float corrCoeff(float x[], float y[], int n)
{
   int i,j;
   float mx, my, sx, sy, sxy, den;

   mx = 0.0;     my = 0.0;        sx = 0.0;
   sy = 0.0;     sxy = 0.0;

   // mean of x[] and y[]
   for (i = 0; i < n ; i++) {
      mx += x[i];
      my += y[i];
   }
   mx = mx / n;
   my /= n;

  // denominator
  for (i = 0; i < n; i++) {
    sx += (x[i] - mx) * (x[i] - mx);
    sy += (y[i] - my) * (y[i] - my);
  }
  den = sqrt(sx) * sqrt(sy);

  // correlation
  for (i = 0; i < n ; i++) {
        sxy += (x[i] - mx) * (y[i] - my);
  }
  return sxy / den;
}


void setup() {
  Serial.begin(9600);
  while (!Serial) ; // wait for serial port to connect.
}

void loop() {
  float corrXY, corrXZ, corrXK;
  corrXY = corrCoeff(X, Y, NUM_SAMPLES);
  corrXZ = corrCoeff(X, Z, NUM_SAMPLES);
  corrXK = corrCoeff(X, K, NUM_SAMPLES);
  Serial.print("\ncorrXY: "); Serial.print(corrXY);
  Serial.print("\ncorrXZ: "); Serial.print(corrXZ);
  Serial.print("\ncorrXK: "); Serial.print(corrXK);
  while(1) delay(100);
}
```

Para o exemplo dado, considerando os seguintes vetores, X, Y, Z, e K, o resultado da correlação entre cada par de vetores é:

- $corrXY = 1,00$: estes vetores têm o mesmo crescimento e as seus valores são idênticos através da mesma proporção, logo a correlação é máxima;

- $corrXZ = -1,00$: estes vetores têm uma evolução contrária entre eles, logo como a evolução tem o mesmo crescimento;

- $corrXK = -0,12$: os valores nestes dois vetores não aparentam ter uma relação entre eles, logo a sua correlação é perto de nula.

3.9.2 Série de Taylor

A Série de Taylor para representar uma função arbitrária através da soma de infinitos termos. É possível obter aproximações razoáveis, dependendo da quantidade de termos usados. De uma forma geral uma função é representada da seguinte forma:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

$n!$ é o factorial de n, e $f^{(n)}(a)$ a derivada de ordem n no ponto a.

Para facilitar a aplicação, existem funções matemáticas com termos tabelados, nomeadamente: exponencial, logaritmo, funções trigonométricas e hiperbólicas.

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots = \sum_{n=0}^{\inf} x^n$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

$$cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

$$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

$$tan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots$$

$$ln(x) = x - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \cdots$$

Várias aplicações admitem um pequeno erro nos cálculos, a série de Taylor é um método importante para substituição da implementação de funções por aproximações que demoram menos tempo a serem calculadas e requerem menos memória de programa. As funções trigonométricas podem ser calculadas em graus $[0 - 360]$ ou radianos $[0 - 2\pi]$. Qualquer ângulo em graus pode ser convertido para radianos com a seguinte razão: $rad = gra * 2\pi/360$.

Na listagem 3.12 mostra-se o conteúdo do *Sketch* para o cálculo da aproximação do seno em radianos:

Listagem 3.12: sinTaylor.ino

```
typedef float data_t;

data_t sinApx(data_t x)
{
  data_t apx = 0, power = x, fact = 1;
```

```
  for(int i = 1; i < 7; i++)
  {
    apx += power / fact;
    power *= -1 * x * x;
    fact *= (2 * i) * (2 * i + 1);
  }
  return apx;
}

void setup() {
  Serial.begin(9600);
  while (!Serial) ; // wait for serial port to connect.
}

void loop() {
  data_t sin1, sin2, sin3;
  sin1 = sinApx(0.0);
  sin2 = sinApx(3.14159 / 2);
  sin3 = sinApx(90.0 * 2 * 3.14159 / 360.0);
  Serial.print("\nsin(0): "); Serial.print(sin1);
  Serial.print("\nsin(pi/2): "); Serial.print(sin2);
  Serial.print("\nsin(90): "); Serial.print(sin3);
  while(1) delay(100);
}
```

O resultado enviado para o terminal série é: sin(0): 0.00, sin(pi/2): 1.00 e sin(90): 1.00.

3.9.3 Interpolação Linear

A interpolação serve para obter aproximações a novos pontos que se encontram entre dois pontos conhecidos. Esta técnica permite traçar curvas de valores com menos saltos, e mais suaves. A interpolação pode ser realizada através de diferentes métodos de regressão:

- linear: o novo ponto encontra-se sobre um segmento de reta entre os pontos vizinhos amostrados ($y = bias + scale * x$);
- polinomial: o novo ponto pertence a um polinómio de ordem N entre os pontos vizinhos amostrados. Quanto maior for o grau do polinómio mais fiel é a aproximação, mas mais tempo demora a ser calculado;
- paramétrica: uso de uma função paramétrica para calculo do novo ponto. A função paramétrica permite usar aproximações diferentes dependendo do valor do ponto pretendido. Por exemplo, uma sinusoide tem um andamento quase linear perto da origem, ao passo que se aproxima de uma parábola (quadrático) quando perto do máximo da função.

Na interpolação linear, o novo ponto y, na ordenada x é calculado com base no declive do segmento de reta dado pelos dois pontos vizinhos a,b:

$$y = y_a + (y_b - y_a)\,\frac{x - x_a}{x_b - x_a}$$

A interpolação polinomial é normalmente mais complicada de ser calculada, pelo que vale a pena recorrer aos métodos numéricos ou simplesmente usar uma ferramenta de computação numérica tal como o Octave, SciPi ou o SciLab.

Em processamento digital de sinal este método permite aumentar o número de amostras, como se o sistema embebido estivesse a fazer a amostragem a um ritmo superior (*upsampling*).

3.9.4 Estimativa de Curva

A estimativa de uma curva que mais fielmente descreva os dados amostrados é importante para descrever as amostras de dados através de um modelo, ou função matemática. Tendo um conjunto de dados com ruído (nuvem de pontos) o objectivo é encontrar uma função matemática que minimize o erro global dos pontos amostrados à reta. As funções polinomiais são uma das formas adotadas, onde se calculam os valores dos coeficientes das curvas para definir a curva:

- $1^{\underline{a}}$ ordem (linear): $y = a.x + b$
- $2^{\underline{a}}$ ordem (quadrática): $y = a.x^2 + b.x + c$
- $3^{\underline{a}}$ ordem (cúbica): $y = a.x^3 + b.x^2 + c.x + d$

Uma forma de aferir se uma curva descreve os pontos fielmente é através do Erro Médio Quadrático (do inglês *Mean Squared Error* - MSE), que é definido pela soma do quadrado das diferenças entre as amostras obtidas e os valores da função polinomial na mesma coordenada:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

3.9.5 Convolução

A operação de convolução é aplicada em processamento digital de sinais e equivale à produção de um sinal à custa de outros dois sinais, através do deslocamento de um deles ao longo do tempo. É como se observando um conjunto de valores a ser gerado por um sensor $x(n)$, eles fossem multiplicados e somados com outro sinal $h(n)$. O sinal $h(n)$ é tipicamente o sinal de convolução e tem uma duração finita e tende a ser pequeno (< 64).

Tendo o sinal $h(n)$ fixo num array, e o sinal $x(n)$ noutro vetor em paralelo com os valores iniciais nulos. À medida quando o sinal $x(n)$ começa a produzir valores a partir do sensor, o resultado da convolução vai ser o produto do valor da primeira amostra e o valor da primeira posição de $h(n)$. No instante seguinte, quando a segunda amostra for recebida do sensor, a

primeira amostra desloca-se para a segunda posição e a nova amostra ocupa a primeira posição. Neste caso o resultado da convolução vai ser o produto da primeira amostra com o valor da segunda posição de $h(n)$ somado do resultado do produto entre a segunda amostra de $x(n)$ e o primeiro valor de $h(n)$. Esta operação vai sendo repetida para todas as amostras de $x(n)$ à medida que vão passando por $h(n)$. Outra forma de descrever a operação de convolução é ter o sinal $x(n)$ com todas as amostras do sensor e ir passando $h(n)$ por ele, começando na primeira posição.

A convolução acima descrita é do tipo 1D (array), no entanto pode ser do tipo 2D. Neste caso $x(n)$ e $h(n)$ são matrizes, e cada valor produzido pela convolução é o resultado das contribuições de todas as posições de $h(n)$ para uma posição em $x(n)$. Em termos matemáticos, a convolução é definida da seguinte forma:

$$y[n] = x[n]h[n] = \sum_{i=-\inf}^{\inf} x[i].h[n-i] \tag{3.2}$$

A listagem 3.13 exemplifica a implementação da função para produzir a convolução entre dois sinais. Foi usado um template para o tipo dos sinais a usar para evitar de ter de replicar a função para diferentes tipos de dados.

Listagem 3.13: Sketch de demonstração da função de convulução

```
// demo convolucao 1D
#define MAX_SIZE  16
#define DEBUG 0

const uint16_t lenH = 5;
int intH[] = { 1, 1, 1, 1, 1 };
//float floatH[] = { 1.0, 1.0, 1.0, 1.0, 1.0 }; // integral
float floatH[] = { 1.0, 2.0, 3.0, 4.0, 5.0 };
//float floatH[] = { 5.0, 4.0, 3.0, 2.0, 1.0 };
//float floatH[] = { 0.2, 0.2, 0.2, 0.2, 0.2 }; // low-pass
//float floatH[] = { 1.0, -1.0, 0.0, 0.0, 0.0 }; // derivate

const uint16_t lenX = 5;
int intX[] = { 1, 1, 1, 1, 1 };
//float floatX[] = { 1.0, 1.0, 1.0, 1.0, 1.0 };
float floatX[] = { 1.0, 2.0, 3.0, 4.0, 5.0 };
//float floatX[] = { 5.0, 4.0, 3.0, 2.0, 1.0 };

const uint16_t lenY = lenH + lenX - 1;
float floatY[MAX_SIZE];
int intY[MAX_SIZE];

template <typename T>
void convul(T h[], T x[], T y[], int lenH, int lenX)
{
  uint8_t i, j;
  T acc;
  /* iterate over all positions for the new vector
   * and get its contributions */
   for(i = 0; i < lenH + lenX - 1; i++) {
#if DEBUG
    Serial.print("\n***\n > i: ");
    Serial.println(i, DEC);
#endif
      acc = 0.0;
      for(j = 0; j <= i; j++) {
        if(j >= lenX) continue;
        if(i - j < 0) continue;
```

```
        if(i - j > lenH) continue;
        acc += x[j] * h[i - j];
#if DEBUG
        Serial.print(">> j: ");
        Serial.print(j, DEC);
        Serial.print(" i-j: ");
        Serial.print(i - j, DEC);
        Serial.print(" x[j]: ");
        Serial.print(x[j], DEC);
        Serial.print(" h[i-j]: ");
        Serial.println(h[i - j], DEC);
#endif
      }
      y[i] = acc;
    }
}


template <typename T>
void printVec(T a, byte numPos)
{
  byte i;
  for(i = 0; i < numPos; i++) {
    Serial.print(a[i], DEC);      Serial.print(" ");
  }
  Serial.println("");

}

void setup() {
  Serial.begin(9600);
  while (!Serial) ; // wait for serial port to connect.

  Serial.print("float h[] = ");   printVec(floatH, lenH);
  Serial.print("float x[] = ");   printVec(floatX, lenX);

  convul<float>(floatH, floatX, floatY, lenH, lenX);
  convul<int>(intH, intX, intY, lenH, lenX);

  Serial.print("float y[] = ");   printVec(floatY, lenY);
  Serial.print("  int y[] = ");   printVec(intY, lenY);
}

void loop() { }
```

Em termos práticos a convolução é usada em filtragem. Existem ferramentas que permitem gerar núcleos de convolução ou filtragem (*kernels*) para produzirem uma determinada resposta, por exemplo num filtro passa-baixo $h(n) = 0,2\ 0,2\ 0,2\ 0,2\ 0,2$ (note-se que a soma é igual a 1), passa-alto $h(n) = -0,1\ -0,1\ -0,1\ 0,9\ -0,1\ -0,1\ -0,1$, andar de ganho $h(n) = 2,0\ 0,0\ 0,0\ 0,0\ 0,0\ 0,0$, ou gerador de eco $h(n) = 1,0\ 0,0\ 0,0\ 0,0\ 0,0\ 0,5\ 0,0$.

O uso de *kernels* de convolução permite ainda realizar operações como a derivada $h(n) = 1,0\ -1,0\ 0,0\ 0,0\ 0,0\ 0,0\ 0,0$ e o integral de um sinal $h(n) = 1,0\ 1,0\ 1,0\ 1,0\ 1,0\ 1,0$. No caso da derivada, é adicionada a contribuição da primeira amostra, para logo de seguida ser removida pela inversão do seu sinal, ficando apenas a contribuição do sinal em cada instante, ou seja o seu declive. Este exemplo pode ser ilustrado com a distância. Se o sinal de entrada forem amostras da distância percorrida, por exemplo, $x(n) = 2,06, 010, 016, 022, 028, 0$, no primeiro instante, partido do zero, determina-se que a derivada (diferença entre o valor atual menos o inicial a dividir pelo instante de tempo = velocidade) é 2. No segundo instante, o novo valor é

6, que multiplicado por 1 dá 6, mas a amostra inicial 2 é agora multiplicada por -1, dando o resultado 4, ou seja, a diferença entre os deslocamentos nos dois instantes. O mesmo acontece para as amostras seguintes.

3.9.6 Séries Pseudo-Aleatórias

Nalgumas aplicações pode ser necessário criar imprevisibilidade, tal como a movimentação de oponentes no espaço de um jogo, esquemas de cifra, ou no tempo de espera pela retransmissão de pacotes de rede. O Arduíno tem nas suas bibliotecas a função `random()` para gerar valores pseudo-aleatórios. Esta função pode receber como argumento um valor (máximo) ou dois valores (mínimo e máximo), e gera valores inteiros do tipo `long`.

O gerador diz-se pseudo-aleatório pois os valores não são verdadeiramente aleatórios. Eles são gerados através de uma expressão matemática, com base no valor anterior. Ou seja, obtém-se a mesma sequência de valores caso o valor inicial (semente) seja o mesmo. Gerar a mesma sequência de valores é interesse durante o desenvolvimento de uma aplicação para testar continuamente com valores conhecidos. No Arduíno a função `randomSeed(long seed)` permite definir o valor inicial. Em funcionamento normal a aleatoriedade da sequência pode ser obtida através da fixação do valor da semente proveniente de um relógio, do valor de tensão lido de um porto analógico não usado (ruído), do valor de um sensor como um acelerómetro, ou do intervalo de tempo decorrido entre premir duas vezes um botão de pressão pelo utilizador. Caso não seja indicado o valor da semente, o gerador assume que é 1.

O gerador aleatório existente no Arduíno (herdado da implementação ANSI C) é baseado no método de Schrage[4]. Para aplicações que necessitem de um gerador mais rápido, mas de menor qualidade, pode-se optar por um gerador do tipo linear congruencial (LC), ou registo de deslocamento de realimentação linear *Linear Feedback Shift-Register* (LFSR). Para aplicações que necessitem de um gerador de melhor qualidade, e que o tempo de geração não seja crítico, podem-se usar geradores como o Mersenne Twister. A qualidade de um gerador pseudo-aleatório depende:

- do número de valores gerados sem repetição, entre amostras consecutivas ou entre amostras de sequências diferentes;
- da correlação entre amostras ser nula;
- de ser capaz de produzir valores com uma distribuição uniforme.

O programa da listagem 3.14 demonstra a utilização da função `rand()` para a geração de valores pseudo-aleatórios entre 0 e 65535, com a semente

[4]Park and Miller, "Random number generators: good ones are hard to find", Communications of the ACM, vol. 31, no. 10, p. 1195, Oct. 1988.

0x1234 (4660). Neste Sketch existem várias possibilidades de geração do valor da semente do gerador pseudo-aleatório:

- Valor constante - sempre que o programa reinicia, é gerada a mesma sequência pseudo-aleatória. No caso de um jogo, basta observar o movimento dos adversários programados no jogo uma vez para saber onde vão estar da próxima vez!
- Valor de entrada analógica desligada - as medições têm sempre ruído associado, pelo que a variação de um bit irá originar uma sequência diferente, no entanto se o sinal estiver ligado a uma tensão constante a probabilidade de repetição aumenta;
- Valor de uma entrada analógica ligada a um sensor que produz dados com grande variabilidade, por exemplo, um acelerómetro;
- Valor derivado de uma acção do utilizador - este é o método que produz a maior variabilidade uma vez que é muito difícil alguém atuar um botão de pressão exatamente no mesmo instante de tempo duas vezes. O código apresentado está simplificado e considera que o utilizador não está a pressionar o botão aquando do arranque do programa (tempo próximo de 0).

A figura 3.11 mostra o gráfico com os valores gerados.

Listagem 3.14: randomVar.ino

```
#define BTN_PIN 2
long a;

void setup() {
  Serial.begin(9600);
  while (!Serial) ; // wait for serial port to connect.
  // 1. constant value
  randomSeed(0x1234); // generator seed value (deterministic)
  // 2. use analog input
  //randomSeed(analogRead(A5));
  // 3. wait for user action
  //pinMode(BTN_PIN, INPUT_PULLUP);
  //while(digitalRead(BTN_PIN) != LOW);
  //randomSeed(millis());
}

void loop() {
  a = random(65535);
  Serial.println(a);
  delay(50);
}
```

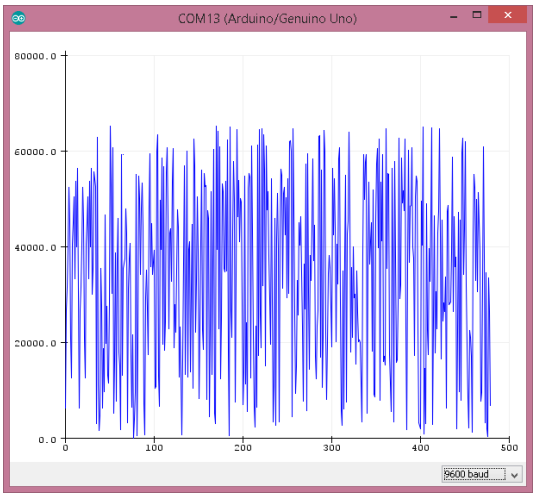

Figura 3.11: Gráfico produzido com os valores gerados aleatoriamente pelo Sketch randomVar.

4
Exercícios

Espera-se que este livro tenha contribuído para aumentar a compreensão do leitor sobre sistemas embebidos, ciber-físicos e IoT. Para terminar, o leitor é desafiado a aplicar os conhecimentos adquiridos e a resolver alguns exercícios.

Exercício 1

Considere uma porta serie assíncrona (UART) configurada a 115200 baud. Indique a sequência de bits enviadas no porto de transmissão (TX) do caracter ASCII 'A', e calcule o tempo de transmissão do caracter.

Exercício 2

Considere um valor lido de ADC de 10 bits para uma variável de 16 bits. Quais os valores mínimo e máximo da variável? Sabendo que o ADC funciona a 5V, indique qual o valor de tensão correspondente aos valores 1 e 123.

Exercício 3

Considere a representação virgula flutuante de acordo com a norma IEEE-754. Se pensar numa representação apenas com 16 bits (half-float), com 5 bits para o expoente e 10 bits para a mantissa. Indique quais os valores máximo e mínimo que consegue representar com esta codificação. Qual o valor mais pequeno que consegue representar?

Exercício 4

Considere o Sketch adcDemo.ino. Na função `mapf(..)`, experimente remover os parêntesis curvos à volta da expressão onde é aplicado o cast para float (float). Experimente aplicar o cast para float no termo `out_min`. Observe os resultados e explique o porquê.

Exercício 5

Considere o Sketch `adcDemo.ino`. Experimente configurar a porta de comunicação 9600 baud e a 115200 baud. Qual a diferença na frequência máxima do sinal amostrado? Quanto tempo é ganho?

Exercício 6

Determine a frequência máxima da onda quadrada que se pode observar no traçador gráfico do Arduíno Uno se a velocidade do canal série (UART) for 112200 baud?

Exercício 7

Crie uma biblioteca para a classe PushButton.

Exercício -

Assuma que tem 5 variáveis para guardar em memória. Sabendo que a memória livre começa em 0x1000, e que as variáveis ocupam 128, 32, 1024 e 500 bytes. Indique quais as gamas de endereços ocupados por cada variável.

Exercício -

Questão: Ao realizar o circuito de demonstração da camera OV7670, quais os passos necessários para adicionar suporte ao registo de ficheiros do tipo PGM num cartão Micro SD-Card ? (circuito e programa)