ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

INSTITUTO POLITÉCNICO DA GUARDA

MEDIÇÃO DA INTENSIDADE DO SOM ATRAVÉS DE SINAIS LUMINOSOS

PROJETO FINAL DA UNIDADE CURRICULAR

Curso	Mestrado Computação Móvel
Unidade Curricular	Sistemas Embebidos
Ano Letivo	2016/2017
Docente	Luís Figueiredo
Coordenador da área disciplinar	António Martins
Data	28-06-2017
Alunos	Anabela Tavares Nº1011109
	Rui Trigo Nº1011106

Índice

1. Introdução

Este relatório tem como objetivo documentar o projeto desenvolvido durante o semestre letivo nas aulas da unidade curricular Sistemas Embebidos.

A estrutura deste relatório encontra-se da seguinte forma: primeiramente os objetivos onde damos uma perspetiva do foi estipulado para a realização do projeto. Em seguida, uma pequena análise do problema onde particularizamos quais os desafios e metas que definimos logo após estudarmos o objetivo do projeto. Além disso, também nesta fase dizemos quais os problemas que surgiram ao longo do desenvolvimento do projeto.

Imediatamente depois, especificamos os passos que demos (algoritmos) e, a partir destes passos elaboramos o código necessário para que o projeto funcionasse. Por fim, mas não menos importante, demonstramos os resultados que obtivemos e, também uma conclusão sobre o projeto em si.

2. Objetivos

Neste projeto foi construído um sistema de medição de som captado por um microfone de um dispositivo Android, que envia os dados para um microcontrolador Arduino através de Bluetooth. O microcontrolador, consoante os dados recebidos, desencadeia o acender de LEDs ligados a este.

O sistema possui mecanismos de comunicação bidirecional, ou seja, nos dois sentidos. A comunicação é manipulada por botões ligados ao microcontrolador para que este, por sua vez, envie através de Bluetooth o modo desejado para o dispositivo Android. Este modo é alterado consoante o botão que é pressionado e faz com que seja aplicado um filtro nos valores enviados para o microcontrolador.

3. Análise do Problema

Após analisarmos e percebemos o objetivo do problema, percebemos que teríamos que ter algum cuidado da maneira como efetuávamos o algoritmo e o código. Depois de debatermos sobre alguns pontos, tais como, qual o sistema de comunicação a usar, como iríamos abordar a questão da comunicação bidirecional e, como iríamos fazer a comunicação Bluetooth no android, chegamos a um consenso para resolver alguns destes problemas.

Em relação ao sistema de comunicação decidimos usar o binário por ser mais eficiente a nível de transmissão de dados, isto é, se tivéssemos optado por um sistema de comunicação ASCII este levaria o dobro do tempo a transmitir os dados porque isso iria exigir dois carateres hexadecimais para representação de cada valor.

Já na parte da comunicação bidirecional optamos por colocar botões no Arduino. Estes botões quando pressionados enviam um comando através de Bluetooth para a aplicação Android. A aplicação através do comando vai filtrar o som capturado e, envia para o Arduino o valor do som, fazendo assim ligar os leds que correspondem ao comando "pedido".

Consoante fomos trabalhando no projeto, principalmente no Android, tivemos que apreender a trabalhar com algumas funções deste para que o sistema comunicasse com o Arduino, isto é, o Android SDK possui uma classe chamada *AsyncTask*, que permite a execução de tarefas assíncronas. Possui 4 funções, sendo apenas a segunda (*doInBackground()*) obrigatória na implementação de classes descendentes. Esta função cria *threads* adicionais para a execução de código com necessidades assíncronas, muitas vezes associadas à comunicação M2M (*Machine-To-Machine*).

Neste caso, o nosso programa tem 3 tarefas assíncronas:

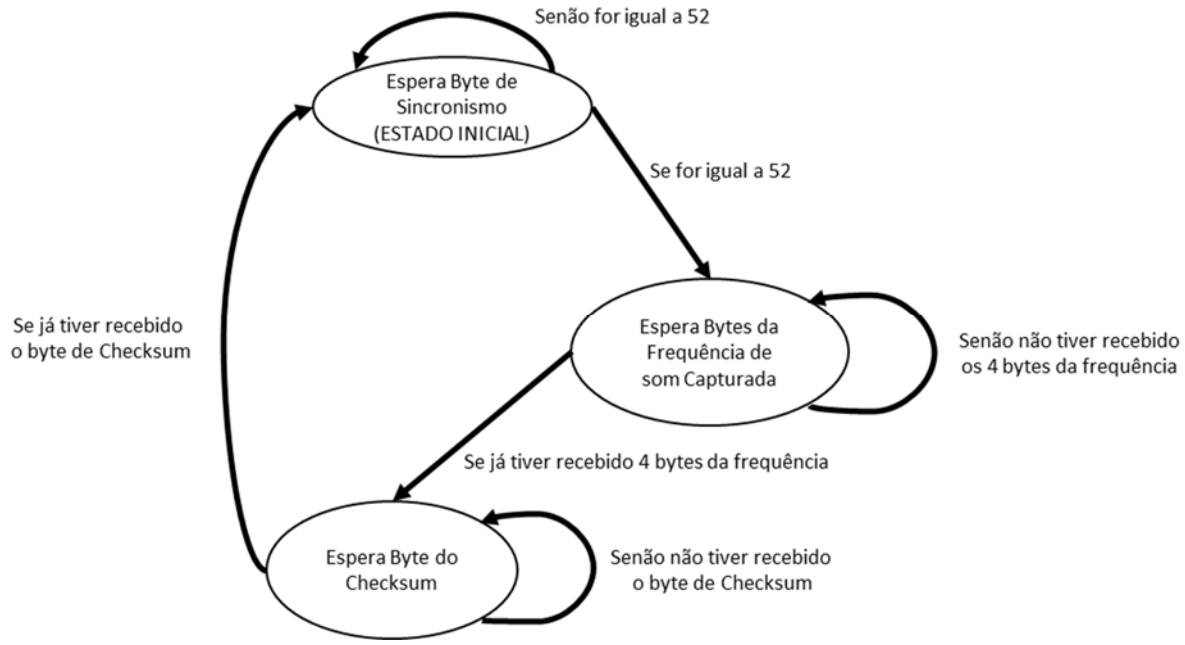
- RecordAudio: captura, quantifica e exibe no ecrã o som recebido;
- CospeValorPorBT: transmite o valor captado pelo microfone para o Arduino via *Bluetooth;*
- PuxaModoPorBT: lê o modo enviado pelo Arduino via Bluetooth.

3.1. Sistema de Comunicação

Byte de Sincronismo	*Bytes da Frequência do sJom capturado*				*Bytes de checksum*
1 byte	1 byte	1 byte	1 bytes	1 byte	1 byte

O dispositivo Android vai enviar por Bluetooth um pacote composto por um valor inteiro precedido de um byte de sincronização (52) e um *checksum* para garantir fiabilidade dos dados.

3.2. Máquina de Estados para confirmar dados enviados



4. Algoritmos

4.1. Algoritmos Arduino

4.1.1. Algoritmo *Setup*

1. Configurar a porta 2 a 9 como saída;
2. Configurar a porta 11, 12, 13 como entrada;
3. Configurar a porta série 0 e 1 com um baud rate de 9600 Hz.

4.1.2. Algoritmo *Loop*

1. Se a porta série 1 tiver dados para receber
   1.1. Ler os dados que estão na porta série;
   1.2. Chamar a função abre-latas para analisar os dados que estão na porta série.
2. Guardar o modo atual numa variável (modo);
3. Se o botão dos graves estiver pressionado
   3.1. Igualar o modo atual a 10;
4. Se o botão dos médios estiver pressionado
   4.1. Igualar o modo atual a 20;

5. Se o botão dos agudos estiver pressionado
   5.1. Igualar o modo atual a 30;
6. Senão estiver nenhum botão pressionado
   6.1. Igualar o modo atual a 0;
7. Se o modo for diferente do modo atual
   7.1. Escrever na porta serie 1 o byte de sincronização (52);
   7.2. Escrever na porta serie 1 o modo atual;

4.1.3. Algoritmo acendeLed

1. Ligar o led indicado pelo parâmetro de entrada.

4.1.4. Algoritmo apagarLed

1. Desligar o led indicado pelo parâmetro de entrada.

4.1.5. Algoritmo abreLatas

1. Caso estado for 0;
   1.1. Se o primeiro byte recebido for igual a 52
      1.1.1. Igualar o estado a 1;
      1.1.2. Igualar o contador a 0;
2. Caso o estado for 1
   2.1. Guardar os bytes recebidos;
   2.2. Incrementar o contador;
   2.3. Se o contador for igual a 4
      2.3.1. Guardar os bytes recebidos numa variável (valorRecebido);
   2.4. Se o contador for igual a 5
      2.4.1. Guardar o ultimo byte recebido;
      2.4.2. Para quando o contador for menor que 4
         2.4.2.1. Somar os bytes recebidos;
      2.4.3. Se o ultimo byte recebido for igual à soma dos bytes recebidos;
         2.4.3.1. Chamar a função blackout;

2.4.3.2. Chamar a função equalizer com o valorRecebido como parâmetro de entrada;

2.4.4. Igualar o estado a 0;

4.1.6. Algoritmo blackout

1. Para cada led

1.1. Chamar a função apagarLed;

4.1.7. Algoritmo equalizer

1. Se o valor recebido for maior ou igual que 0

1.1. Chamar a função acenderLed para o led verde 1;

1.2. Se o valor recebido for maior ou igual que o limite dos graves / 3

1.2.1.Chamar a função acenderLed para o led verde 2;

1.2.2.Se o valor recebido for maior ou igual que (limite dos graves / 3) * 2

1.2.2.1. Chamar a função acenderLed para o led verde 3;

1.2.2.2. Se o valor recebido for maior ou igual que o limite dos graves

1.2.2.2.1. Chamar a função acendeLed para o led amarelo 1;

1.2.2.2.2. Se o valor recebido for maior ou igual que o limite dos médios / 3

1.2.2.2.2.1. Chamar a função acendeLed para o led amarelo 2;

1.2.2.2.2.2. Se o valor recebido for maior ou igual que (limite dos médios / 3) * 2

1.2.2.2.2.2.1. Chamar a função acendeLed para o led amarelo 3;

1.2.2.2.2.2.2. Se o valor recebido for maior ou igual que o limite dos médios

1.2.2.2.2.2.2.1. Chamar a função acendeLed para o led vermelho 1;

1.2.2.2.2.2.2.2. Se o valor recebido for maior ou igual que o limite dos médios * 2

1.2.2.2.2.2.2.2.1. Chamar a função acendeLed para o led vermelho 2;

4.2. Algoritmos Android

4.2.1. Algoritmo CospeValorPorBT

Função *doInBackground()* com o valor quantificado do som como parâmetro de entrada.

1. Se modo for igual a 0
   1.1. Enviar valor para o Arduino
2. Senão se o modo for igual a 10
   2.1. Se o som for menor que o limite graves
      2.1.1. Enviar valor para o Arduino
3. Senão se o modo for igual a 20
   3.1. Se o som for maior que o limite dos Graves e menor que o limite dos médios
      3.1.1. Enviar valor para o Arduino
4. Senão se o modo for igual a 30
   4.1. Se o som for maior que o limite médios
      4.1.1. Enviar valor para o Arduino

4.2.2. Algoritmo PuxaModoPorBT

Função doInBackground()

1. Se a porta série 1 tiver dados para receber
   1.1. Guardar os bytes recebidos na variável *bytesTemp*;
2. Se a variável *bytesTemp* não estiver vazio
   2.1. Para quando o tamanho dos *byteTemp* for menor que o contador
      2.1.1. Caso o estado for 0
         2.1.1.1. Se o *bytesTemp* for igual ao byte de sincronização (52)
            2.1.1.1.1. Igualar o estado a 1;
            2.1.1.1.2. Igualar o contador a 0;
      2.1.2. Caso o estado for 1
         2.1.2.1. Guardar os *bytesTemp* na variável *bytesRecebidos*;
         2.1.2.2. Incrementar o contador;
         2.1.2.3. Se o contador for igual 2
            2.1.2.3.1. Igualar o estado a 0;
            2.1.2.3.2. Guardar os *bytesRecebidos* na variável *integer*;
            2.1.2.3.3. Devolver o valor da variável *integer*;

4.2.3. Algoritmo do intToByteArray

Função intToByteArray com parâmetro de entrada do valor que queremos enviar através da porta série.

1. Guardar o valor a enviar numa variável (bytesDoInteiro);
2. Colocar na variável *checksum* a soma dos quatro primeiros bytes da variável bytesDoInteiro;
3. Devolver a variável bytesDoInteiro;

4.2.4. Algoritmo do sendInt

Função sendInt com parâmetro de entrada do valor que queremos enviar através da porta série.

1. Guardar o valor a enviar numa variável (buffer);
2. Enviar através da porta série o valor do byte de sincronização (52);
3. Enviar o valor do variável *buffer*;
4. Enviar o valor da variável *checksum*;

5. Código

5.1. Código Arduino

```
byte inicio = 52;
int modo = 0, modoAtual = 1;
long tempoAtual, tempoInicio;
int botaoGraves = 13;//cabo branco
int botaoMedios = 12;//cabo laranja
int botaoAgudos = 11;//cabo roxo
int estadoBotaoGraves, estadoBotaoMedios, estadoBotaoAgudos;
int ledVerde1 = 8;
int ledVerde2 = 7;
int ledVerde3 = 6;
int ledAmarelo1 = 5;
int ledAmarelo2 = 4;
int ledAmarelo3 = 3;
int ledVermelho1 = 2;
int ledVermelho2 = 9;
long valorRecebido;
#define ESPERA52 0
#define ESPERADADOS 1
```

```
void setup() {
  // initialize both serial ports:
  Serial.begin(9600);

  Serial1.begin(9600);
  pinMode(botaoGraves, INPUT);
  pinMode(botaoMedios, INPUT);
  pinMode(botaoAgudos, INPUT);

  pinMode(ledVerde1, OUTPUT);
  pinMode(ledVerde2, OUTPUT);
  pinMode(ledVerde3, OUTPUT);
  pinMode(ledAmarelo1, OUTPUT);
  pinMode(ledAmarelo2, OUTPUT);
  pinMode(ledAmarelo3, OUTPUT);
  pinMode(ledVermelho1, OUTPUT);
  pinMode(ledVermelho2, OUTPUT);
}

void loop() {
  //   read from port 1, send to port 0:
  if (Serial1.available()) { ├ 1.
    int mByte = Serial1.read();  ├ 1.1
    abreLatas(mByte); ├ 1.2
  }

  estadoBotaoGraves = digitalRead(botaoGraves);
  estadoBotaoMedios = digitalRead(botaoMedios);
  estadoBotaoAgudos = digitalRead(botaoAgudos);

  modo = modoAtual; ├ 2.

  //se pressionado (HIGH)
  if (estadoBotaoGraves == HIGH) {   ├ 3
    modoAtual = 10;   ├    3.1
  } else if ( estadoBotaoMedios == HIGH) { ├ 4.
    modoAtual = 20; ├ 4.1
  } else if ( estadoBotaoAgudos == HIGH) {├ 5.
    modoAtual = 30; ├ 5.1
  } else {  ├ 6.
    modoAtual = 0;  ├ 6.1
  }

  if (modo != modoAtual) { ├ 7.
    Serial1.write(inicio); ├ 7.1
                                          ├
```

```
    Serial1.write((char*) &modoAtual, 2);        7.2
  }
}

void acenderLed(int varLed) {
  digitalWrite(varLed, HIGH);
}

void apagarLed(int varLed) {
  digitalWrite(varLed, LOW);
}

void abreLatas(int mByte) {
  byte bytesRecebidos[20];
  static int i = 0;
  static boolean estado = ESPERA52;

  switch (estado) {
    case ESPERA52: ├  1.
      if (mByte == inicio) { ├ 1.1
        estado = ESPERADADOS; ├  1.1.1
        i = 0; ├ 1.1.2
      }
      break;
    case ESPERADADOS:  ├  2.
      bytesRecebidos[i] = mByte; ├  2.1
      i++; ├ 2.2
      if (i == 4) { ├  2.3.
        valorRecebido = *((long*) (bytesRecebidos));  ├ 2.3.1
      } else if (i == 5) {  ├ 2.4
        byte checksum = bytesRecebidos[4]; ├ 2.4.1
        int soma = 0;
        for (int i = 0; i < 4; i++) {  ├ 2.4.2
          soma += bytesRecebidos[i];  ├ 2.4.2.1
        }
        if (soma == checksum) {  ├ 2.4.3
          blackout(); ├ 2.4.3.1
          equalizer(valorRecebido); ├ 2.4.3.2
        }
        estado = ESPERA52; ├  2.4.4
      }
      break;
  }

}
```

```
void blackout() {
  apagarLed(ledVerde1);
  apagarLed(ledVerde2);
  apagarLed(ledVerde3);
  apagarLed(ledAmarelo1);
  apagarLed(ledAmarelo2);
  apagarLed(ledAmarelo3);
  apagarLed(ledVermelho1);
  apagarLed(ledVermelho2);
}

void equalizer(int vr) {
  //todo ajustar
  int  limiteGraves = 150;
  int  limiteMedios = 250;

  if (vr >= 0) {
    acenderLed(ledVerde1);
    if (vr > limiteGraves / 3) {
      acenderLed(ledVerde2);
      if (vr > (limiteGraves / 3) * 2) {
        acenderLed(ledVerde3);
        if (vr > limiteGraves) {
          acenderLed(ledAmarelo1);
          if (vr > limiteMedios / 3) {
            acenderLed(ledAmarelo2);
            if (vr > (limiteMedios / 3) * 2) {
              acenderLed(ledAmarelo3);
              if (vr > limiteMedios) {
                acenderLed(ledVermelho1);
                if (vr > limiteMedios * 2) {
                  acenderLed(ledVermelho2);
                }
              }
            }
          }
        }
      }
    }
  }
}
```

5.2. Código Android (o mais importante)

```
public void sendInt(int dataToSend) {
            byte[] buffer = intToByteArray(dataToSend);  } 1.
            byte flag = 52;
            try {
                mmOutStream.write(flag);    } 2.
                mmOutStream.write(buffer);   } 3.
                mmOutStream.write(checksum);   } 4.
                //Log.d("chk", "" + checksum);
            } catch (IOException e) {
            }
        }

private byte[] intToByteArray(int val) {
            byte[] bytesDoInteiro = {
                    (byte) val,
                    (byte) (val >>> 8),
                    (byte) (val >>> 16),        1.
                    (byte) (val >>> 24)
            };
            checksum = 0;

            //byte em java cabem 128
            checksum += bytesDoInteiro[0] & 0xFF;    }   2.
            checksum += bytesDoInteiro[1];
            checksum += bytesDoInteiro[2];          2.
            checksum += bytesDoInteiro[3];


            return bytesDoInteiro;      }  3.
        }

private class CospeFreqPorBT extends AsyncTask<Integer, Void, Void> {
        @Override
        protected Void doInBackground(Integer... params) {
            int freq = params[0];
            int  limiteGraves = 150;
            int  limiteMedios = 250;
            if (modo == 0) {     }   1.
                connectedThread.sendInt(freq);   } 1.1
            } else if (modo == 10) { } 2
            if (freq < limiteGraves)  } 2.1
                connectedThread.sendInt(freq); } 2.11
            } else if (modo == 20) {  } 3.
             if (freq > limiteGraves && freq < limiteMedios)   } 3.1
               connectedThread.sendInt(freq);  } 3.1.1
            } else if (modo == 30) {  }  4
                if (freq > limiteMedios) } 4.1
               connectedThread.sendInt(freq);  } 4.1.1
            }
            return null;
       } }
    private class PuxaModoPorBT extends AsyncTask<Void, Void, Integer> {
```

```
        byte[] bytesTemp = new byte[100];
        @Override
        protected Integer doInBackground(Void... params) {
            try {
                if (mmInStream.available() > 0)   ├ 1.
                    mmInStream.read(bytesTemp);     ├ 1.1
            } catch (IOException e) {
                e.printStackTrace();
            }

            if (bytesTemp != null) {  ├ 2.
                for (int i = 0; i < bytesTemp.length; i++) {   ├ 2.1
                    switch (estado) {
                        case ESPERA52:  ├ 2.1.1.
                            if (bytesTemp[i] == 52) {   ├ 2.1.1.1
                                estado = RECEIVE_MESSAGE;  ├ 2.1.1.1.1
                                contador = 0;   ├ 2.1.1.1.2
                            }
                            break;
                        case RECEIVE_MESSAGE:   ├ 2.1.2
                            bytesRecebidos[contador++] = bytesTemp[i]; ├ 2.1.2.1;
2.1.2.2
                            if (contador == 2) {   ├ 2.1.2.3
                                estado = ESPERA52;   ├ 2.1.2.3.1
                                int integer = bytesRecebidos[0] +
(bytesRecebidos[1] << 8);  ├ 2.1.2.3.2
                                return Integer.valueOf(integer); ├ 2.1.2.3.3
                            }
                            break;
                    }
                }
            }
            return null;
        }
        @Override
        protected void onPostExecute(Integer integer) {
            if (integer != null) {
                tv_modo.setText("Modo: " + integer);
                modo = integer;
            }
        }
```

6. Resultados

O programa funciona corretamente e como previsto. Houve vários problemas no desenvolvimento do trabalho tais como:

- Problema: a receção de dados por parte do Arduino, ou seja, não estávamos a aplicar o sistema de comunicação proposto e por isso mesmo não conseguíamos receber dados certos.
    - Resolução: refazer o algoritmo e a máquina de estados para conseguirmos ter uma receção de dados completa.
- Problema: os botões mesmo que não fossem pressionados estavam a enviar dados constantemente. Assim sendo, estávamos a ocupar largura de banda e isso tornava o sistema mais lento.
    - Resolução: refazer o algoritmo e código dos botões para só quando pressionados ou largados enviar dados, com isso, desocupamos a largura de banda e ganhamos velocidade no sistema.
- Problema: quando uma *AsyncTask* está em funcionamento não deixa que outra funcione ao mesmo tempo, ou seja, não conseguíamos fazer duas tarefas ao mesmo tempo.
    - Resolução: para este problema ser resolvido colocamos as *AsyncTasks* a trabalhar em conjunto, isto é, através de uma função do Android (*executeOnExecutor*) conseguimos que as *AsyncTasks* sejam executadas em sequência e, assim já víamos resultados.

Após debate entre o grupo e o Docente chegámos à conclusão que a maneira como estávamos a planear desenvolver o projeto não era a mais correta. Sendo assim, optamos por organizar melhor o nosso algoritmo e consecutivamente o nosso código para a conclusão do projeto.

7. Conclusão

Concluímos que o programa foi um grande desafio e com bons resultados. Este projeto permitiu que ganhássemos conhecimento suficiente para explorar as potencialidades do Arduino bem como a comunicação *Machine-To-Machine* pela porta série.

Apesar de termos encontrados questões durante a execução do programa, conseguimos responder à maioria delas com simples discussões entre nós e/ou dúvidas tiradas ao professor.

Para finalizar, a realização deste trabalho correu dentro dos parâmetros previstos, pois conseguimos alcançar os objetivos que tínhamos definido. Sendo assim, estamos contentes com a escolha do projeto e satisfeitos com o resultado.