Sistemas Autónomos - Sensorização

Bruno Pereira A75135 | João Coelho A74859 | Luís Fernandes A74748 | Maria Ana de Brito A73580

28 de Abril de 2018

Resumo

O presente documento relata o processo de obtenção e tratamento de dados captados por um sensor, com a particularidade de esses dados terem sido usados para desencadear certas ações num *smartphone*, através da utilização das plataformas AdafruitIO e IFTTT.

1 Obtenção dos dados

A primeira fase deste trabalho consistiu na seleção e conexão de um sensor a um PC, para possibilitar a recolha de dados. A nível de hardware de sensorização foi utilizado um Arduino Mega, ao qual se ligou um detetor de chama (ESP8266) da forma que surge na figura abaixo (fio preto no GROUND, vermelho nos 5V e azul/verde no pino 7, sendo, neste último caso, meramente uma opção).

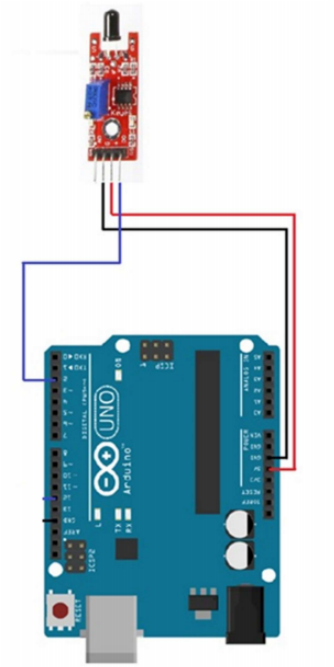

Figura 1: Esquema de conexão do sensor ESP8266 ao Arduino Mega.

Naturalmente, o próximo passo neste processo foi a ligação do hardware ao PC, não em termos físicos, uma vez que isso se fez apenas ligando um cabo USB, mas sim a conexão através do software, neste caso recorrendo ao ArduinoIDE, um ambiente de desenvolvimento próprio para este tipo de equipamento. Aqui, após se selecionar a porta de série onde o Arduino estava ligado, no menu *Tools*, e se instalar algumas bibliotecas - ArduinoHttpClient, Adafruit IO Arduino e Adafruit MQTT - correu-se o seguinte programa em C++, que por sua vez é executado no Arduino:

```
int FlamePin = 7; // This is for input pin
int Flame = HIGH; // HIGH when FLAME Exposed

void setup() {
  pinMode(FlamePin, INPUT);
  Serial.begin(9600);
}

void loop() {
  Flame = digitalRead(FlamePin);
  if (Flame == HIGH) {
    Serial.println("HIGH FLAME");
  }
  else {
    Serial.println("No flame");
  }
}
```

Acima, pode-se observar que a variável "FlamePin"define o pino onde se liga o fio azul/verde do hardware de sensorização escolhido para a experiência. Também é possível perceber-se quais são os dois valores de output deste programa:

- **No flame**, quando não é detetada chama;
- **HIGH FLAME**, aquando da deteção de chama.

Com este código em execução, foi já possível detetar e obter resultados lidos pelo sensor, observáveis no "Serial Monitor"do ArduinoIDE.

Estando estabelecida a conexão do hardware, quer fisicamente, quer a nível do software, o passo seguinte foi a criação de um programa Java que permitisse receber os valores a serem lidos pelo sensor. Para tal, foram necessários os seguintes *imports* no programa:

- import gnu.io.CommPortIdentifier
- import gnu.io.SerialPort
- import gnu.io.SerialPortEvent
- import gnu.io.SerialPortEventListener

Estas classes são parte da API RXTX do Java, biblioteca que permite enviar os dados do Arduino para o programa e cuja dependência se adicionou ao projeto recorrendo ao Maven - disponível em https://mvnrepository.com/artifact/org.rxtx/rxtx.

Continuando com a descrição do programa, a etapa seguinte foi a definição da conexão física, com a criação da variável **PORT_NAME** para definir a porta de série onde o Arduino se encontra ligado, seguida pela definição de uma função de inicialização, onde se associa à mesma porta todo o processo de conexão, como, por exemplo, o *listener* de eventos "SerialPort".

```
public void initialize() {
    CommPortIdentifier portId = null;
    Enumeration portEnum = CommPortIdentifier.getPortIdentifiers();

    while (portEnum.hasMoreElements()) {
        CommPortIdentifier currPortId = (CommPortIdentifier) portEnum.nextElement();
        if (currPortId.getName().equals(PORT_NAME)) {
            portId = currPortId;
            break;
        }
    }
    if (portId == null) {
        System.out.println("Could not find COM port.");
        return;
    }

    try {
        serialPort = (SerialPort) portId.open(this.getClass().getName(), TIME_OUT);
        serialPort.setSerialPortParams(DATA_RATE,
                SerialPort.DATABITS_8,
                SerialPort.STOPBITS_1,
                SerialPort.PARITY_NONE);
        // open the stream
        input = new BufferedReader(new InputStreamReader(serialPort.getInputStream()));
        serialPort.addEventListener(this);
        serialPort.notifyOnDataAvailable(true);
    } catch (Exception e) {
        System.err.println(e.toString());
    }
}
```

O evento "SerialPort"é lançado periodicamente pelo sensor, durante o seu ciclo de execução. É designada por "SerialPortEventListener"a interface que o programa JAVA implementa e que o coloca "à escuta"dos dados enviados, os quais são captados e "filtrados"numa outra função, que se designou por "serialEvent", que é executada quando um evento "SerialPort"é acionado. Esta função será alvo de análise na secção seguinte.

Estando este passo completo, tornou-se possível obter na aplicação JAVA os dados lidos pelo sensor, estabelecendo condições de passar à etapa seguinte deste trabalho, isto é, a interpretação dos dados e conexão a plataformas externas.

2 Publicação no AdafruitIO e ligação ao IFTTT

Após a criação de conta no AdafruitIO e IFTTT, criou-se um *feed* "command"no AdafruitIO e conectou-se o IFTTT a esta plataforma, seguindo o tutorial da aula. Depois deste processo, pensou-se na publicação dos dados lidos pelo sensor, tendo a escolha recaído numa arquitetura deliberativa, na medida em que só seriam submetidos dados dentro de certas condições.

De forma mais concreta, como os dados chegavam com grande frequência, provenientes da leitura do sensor, definiram-se dois valores de iterações entre as quais se enviaria, mediante os valores lidos pelo sensor, um 0 ou 1 ao AdafruitIO. Como foi usado um detetor de chamas na recolha dos dados, cujo *output* foi explicitado na secção anterior, estabeleceu-se a correspondência entre a deteção de chama ("HIGH FLAME") e o valor 1 e a ausência de chama ("No flame") e o valor 0. Assim, aquando da receção de dados do sensor, é avaliado se é necessário comunicar com o AdafruitIO ou não, estando isto dependente do valor de duas variáveis: "dangerAlert"e "safetyAlert". Estas variáveis funcionam como contadores decrescentes, que medem o perigo e a segurança, respetivamente. O valor de "dangerAlert"é iniciado a 10 e o de "safetyAlert"a 100. A explicação por detrás destes valores consiste no seguinte:

- Quando é detetada chama em 10 leituras consecutivas, face à possibilidade de o fogo ser uma ameaça no dado espaço, é de imediato comunicado à plataforma externa;

- Quando esta situação não acontece, ao fim de 100 leituras consecutivas do valor "No flame", é enviada uma notificação para reportar a segurança do espaço.

Em casos reais, eventualmente seria vantajoso a adição de uma terceira mensagem, que ao fim de algumas leituras em que nenhuma das mensagens anteriores fosse despoletada, enviaria uma mensagem com código diferente para o Adafruit - 2, por exemplo - notificando da ausência de perigo, mas com registos parciais de atividade de chama.

Em termos da ligação ao AdafruitIO, no caso de ameaça de chama, é enviado um 1 ao feed "command", desencadeando a ação correspondente. Naturalmente, na segunda situação, o valor passado à plataforma é 0, desencadeando uma ação diferente. Neste caso, as ações são idênticas, isto é, uma notificação recebida no telemóvel com a *app* do IFTTT instalada, porém a mensagem passada na mensagem é diferente. Em termos de código, estas ações são realizadas na função "serialEvent", cujo excerto surge abaixo:

```
public synchronized void serialEvent(SerialPortEvent oEvent) {
  if (oEvent.getEventType() == SerialPortEvent.DATA_AVAILABLE) {
        try {
                String inputLine = null;
                if (input.ready()) {
                   inputLine = input.readLine();
                   if (inputLine.equals("No flame")) {
                      this.safetyAlert--;
                      if (this.dangerAlert != 10)
                         this.dangerAlert = 10;
                       content = "0";
                    }
                    else {
                      this.dangerAlert--;
                      if (this.safetyAlert != 100)
                          this.safetyAlert = 100;
                      content = "1";
                    }
                    if (this.dangerAlert == 0 || this.safetyAlert == 0) {
                        System.out.println("Publishing message: "+content);
                        MqttMessage message = new MqttMessage(content.getBytes());
                        message.setQos(qos);
                        sampleClient.publish(topic, message);
                        System.out.println("Message published");
                        System.out.println(inputLine);
                        this.dangerAlert = 10;
                        this.safetyAlert = 100;
                    }
              }
        } catch (Exception e) { ... }
  }
}
```

Relativamente ao código anterior e ao envio das mensagens para o AdafruitIO, é relevante ter em conta que foi necessário importar as classes relativas ao protocolo de mensagens MQTT, o que também implicou a adição de um dependência ao projeto através do Maven, esta disponível em https://repo.eclipse.org/content/repositories/paho-releases. Após isto, foi também necessário configurar a ligação à plataforma, definindo o *username* e a *password* da conta no AdafruitIO.

Explicado o funcionamento do programa, resta agora exibir os resultados das conexões às diferentes plataformas. Na imagem imediatamente abaixo vê-se a prova da chegada dos valores 0 e 1 ao AdafruitIO. Após esta primeira imagem surge a representação das *Applets* definidas no IFTTT, ativas quando ao *feed* "command"do AdafruitIO chegam os valores 0 ou 1. Por último, mostra-se um comprovativo de como a ação foi desencadeada, surgindo no relatório de atividade que uma notificação foi enviada para a aplicação do IFTTT.

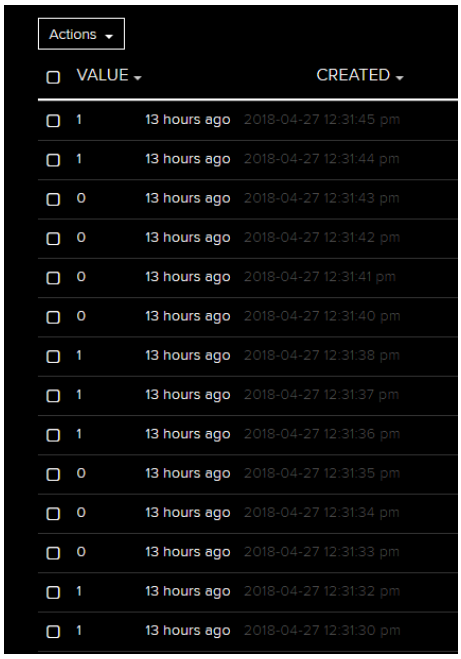

Figura 2: Dados enviados para o AdafruitIO.

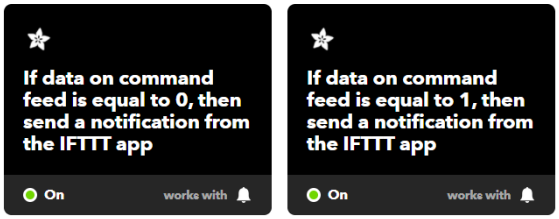

Figura 3: Applets no IFTTT.

Figura 4: Mostra da chegada da notificação à aplicação.