UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHARIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Telemetria sem fio com armazenamento e exportação dos dados de unidade de sensoreamento via aplicativo Android

Autor: Gabriel Camoese Salla

Orientador: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos

2015

Gabriel Camoese Salla

Telemetria sem fio com armazenamento e exportação dos dados de unidade de sensoreamento via aplicativo Android

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica

ORIENTADOR: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos
2015

Salla, Gabriel Camoese
S168t Telemetria sem fio com armazenamento e exportação dos dados de unidade de sensoreamento via aplicativo Android / Gabriel Camoese Salla; orientador Evandro Luís Linhari Rodrigues. São Carlos, 2015.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2015.

1. Android. 2. Telemetria. 3. Sistemas embarcados. 4. Comunicação sem fio. I. Título.

FOLHA DE APROVAÇÃO

Nome: Gabriel Camoese Salla

Título: "Telemetria sem fio com armazenamento e exportação dos dados de unidade de sensoreamento via aplicativo Android"

Trabalho de Conclusão de Curso defendido e aprovado em 26/11/2015,

com NOTA 8,7 (Oito, sete), pela Comissão Julgadora:

Prof. Associado Evandro Luis Linhari Rodrigues - (Orientador - SEL/EESC/USP)

Prof. Dr. José Roberto Boffino de Almeida Monteiro - (SEL/EESC/USP)

Mestre André Luís Martins - (Doutorando - SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Dr. José Carlos de Melo Vieira Júnior

Dedicatória

Dedico este trabalho à minha família pelo apoio e pela oportunidade de obter uma educação de qualidade.

Gabriel Camoese Salla.

Agradecimentos

Agradeço à minha família pelo suporte, aos meus amigos pelo auxílio e conhecimentos compartilhados, ao professor Evandro pela orientação e aos professores que se dedicam aos alunos e ao ensino.

Gabriel Camoese Salla.

Resumo

Estre trabalho contém o desenvolvimento de um aplicativo para dispositivos portáteis com sistema operacional Android que, através da comunicação via bluetooth e Wi-Fi com uma unidade de sensoreamento, obtenha leituras de um número de sensores, permitindo também o armazenamento e exportação destes dados. Para realizar testes e ilustrar o funcionamento do aplicativo também foram desenvolvidas unidades de sensoriamento com variações do protocolo de comunicação e tipos de dados. Para o desenvolvimento do aplicativo foi utilizada a IDE (*Integrated Development Environment* - Ambiente de Desenvolvimento Integrado) Android Studio juntamente com a SDK (*Software Development Kit* - Kit de Desenvolvimento de Software) do Android, com a linguagem de programação Java. Para as unidades de sensoreamento, os microcontroladores utilizados foram AT89LP4052 e ATMEGA328P (kit Arduino Uno), enquanto que os módulos de comunicação foram o ESP8266 (comunicação via Wi-Fi, utilizando o *firmware* NodeMCU para programação em linguagem Lua) e RN42 (comunicação via bluetooth). Os resultados obtidos mostram que os objetivos propostos foram atingidos com sucesso.

Palavras-Chave: Android, Telemetria, Sistemas embarcados, Comunicação sem fio.

Abstract

This document contains the development of an application for Android handheld systems that, by wireless communication via bluetooth and Wi-Fi with a sensing unit, acquire readings from a number of sensors, allowing storing and exporting these data. To help with testing and to illustrate the application's operation, some sensing units were developed varying the communication protocol and the types of data to be acquired. The application was developed using Android Studio IDE (Integrated Development Environment), along with the Android SDK (Software Development Kit), with Java programming language. For the sensing units, the selected microcontrollers are AT89LP4052 and ATMEGA328P (belonging to an Arduino development board), while the communication modules were ESP8266 (Wi-Fi communication using NodeMCU firmware with Lua programming language) and RN42 (bluetooth communication). The results show that the proposed objectives were archieved successfully.

Keywords: Android, Telemetry, Embedded systems, Wireless communication.

Lista de figuras

Lista de tabelas

Siglas

CPU	*Central Processing Unit* - Unidade central de processamento
I2C	*Inter-Integrated Circuit*
IDE	*Integrated Development Environment* - Ambiente de Desenvolvimento Integrado
IP	*Internet Protocol* - Protocolo de Internet
MAC	*Media Access Control*
NFC	*Near Field Communication* - Comunicação por Campo de Proximidade
OS	*Operating System* - Sistema Operacional
RAM	*Random Access Memory* - Memória de Acesso Aleatório
SDK	*Software Development Kit* - Kit de Desenvolvimento de Software
SPI	*Serial Peripheral Interface*
SQL	*Structured Query Language* - Linguagem de Consulta Estruturada
SSID	*Service Set Identifier*
TCP	*Transmission Control Protocol* - Protocolo de Controle de Transmissão
UART	*Universal Asynchronous Receiver/Transmitter*
UI	*User Interface* - Interface do Usuário
UDP	*User Datagram Protocol*
USB	*Universal Serial Bus* - Barramento Serial Universal

Conteúdo

Capítulo 1

Introdução

Microcontroladores são amplamente utilizados em equipamentos eletrônicos em praticamente qualquer aplicação, seja esta industrial, comercial ou pessoal. Este componente é, muitas vezes, responsável pelo controle de funções e operações e algumas vezes denominado o "cérebro" destes equipamentos. Atualmente, até os microcontroladores mais simples são capazes de processar mais de 20 milhões de instruções por segundo. Além disso, é importante que certos dispositivos sejam capazes de realizar medições e comunicação de informações de interesse para um usuário em uma central.

A demanda por equipamentos sem fio tem aumentado nos últimos anos com o desenvolvimento e barateamento de dispositivos portáteis capazes de se comunicar a distância. Atualmente, quase todas as pessoas possuem um celular com pelo menos um tipo de comunicação sem fio a curta distância, sendo as mais comuns *bluetooth*, Wi-Fi e NFC. As duas primeiras são capazes de alcançar, com facilidade, distâncias maiores que dez metros, enquanto que a terceira depende dos dispositivos estarem bem próximos entre si.

Com a popularização desta funcionalidade a telemetria sem fio começou a crescer, visto as vantagens que possui. A leitura remota de sensores facilita e agiliza a obtenção de vários dados a partir de um único ponto, sem a necessidade de conexão de dutos de dados entre os sensores e a central. Além disso, a integração com sistemas computadorizados permite que estas informações sejam armazenadas sem dificuldades para uso futuro.

Existem dezenas de módulos com conexão Wi-Fi e/ou *bluetooth* presentes no mercado, em várias faixas de preço. Estes módulos podem ser facilmente conectados a um microcontrolador permitindo que dispositivos se comuniquem entre si, abrindo portas para inúmeras novas aplicações.

Nota-se então a possibilidade de integração entre unidades de sensoreamento e disposi-

tivos portáteis como celulares e *tablets*, com sistema operacional Android, para a realização de telemetria remota. Além da maioria destes dispositivos possuírem pelo menos um dos protocolos de comunicação citados anteriormente, a vasta documentação permite o desenvolvimento de aplicativos sem demasiadas complicações.

1.1 Objetivos

O foco principal do projeto é o desenvolvimento de aplicativo para um dispositivo portátil com sistema operacional Android para realizar a telemetria de unidade de sensoreamento de forma simplificada para o usuário. Além disso, o aplicativo deve possibilitar o armazenamento e exportação destes dados na forma de planilhas para utilização em outras plataformas. Com a finalidade de testar e ilustrar o funcionamento do aplicativo, direcionou-se também a atenção para a criação de unidades de sensoreamento com alterações no protocolo de comunicação e nos tipos de dados a serem manipulados.

Capítulo 2

Embasamento Teórico

2.1 Meios de comunicação sem fio

Hoje é fácil reconhecer que a maior parte dos dispositivos portáteis possui mais de um meio de comunicação sem fio, sendo os mais comuns Wi-Fi, Bluetooth e NFC (*Near Field Communication*). A seguir serão analisadas as principais características de cada um destes meios.

2.1.1 Wi-Fi

O propósito do padrão IEEE 802.11 é disponibilizar uma conexão sem fio para dispositivos de forma rápida e simples, atingindo facilmente uma velocidade de troca de dados maior que 100kb/s [1].

A transmissão consome cerca de 100mA a 350mA, mas um alcance relativamente grande [1], sendo uma melhor opção quando a comunicação por uma maior distância é necessária.

Além disso, este padrão permite a criação de redes, descartando a necessidade de que os dispositivos estejam conectados diretamente entre si.

Os três principais protocolos de comunicação utilizados são TCP/IP e UDP/IP.

2.1.1.1 Protocolos TCP/IP [2]

A partir do endereço de uma máquina em uma rede com protocolo IP (*Internet Protocol*), é possível conectar-se a ela utilizando o protocolo TCP (*Transmission Control Protocol*) para realizar a troca de dados. Este protocolo permite a conexão entre dispositivos diferentes e é amplamente utilizado por praticamente todos os dispositivos conectados à Internet atualmente.

2.1.1.2 Protocolo UDP/IP [2]

Ao contrário do protocolos TCP/IP, o protocolo UDP/IP (*User Datagram Protocol*) não é orientado a conexão. Seu propósito principal é enviar pacote de dados (datagrama) sem manter a conexão ou necessidade de confirmação de recebimento. Por outro lado, permite que um pacote seja enviado para vários dispositivos ao mesmo tempo utilizando o endereço de *broadcast* da rede.

2.1.2 *Bluetooth*

Como um meio de comunicação sem fio barato e para curtas distâncias *bluetooth* é uma boa opção para sistemas embarcados alimentados com baterias [1]. Com um consumo de energia próximo a 35mA [1], pode atingir velocidades relativamente altas chegando facilmente a 40kb/s.

Os dispositivos realizam buscas por outros dispositivos próximos e, após o emparelhamento destes, podem conectar-se e trocar dados.

Bluetooth é bastante viável quando a conexão pode ser feita diretamente entre ambos os dispositivos, sem a necessidade de interconexões.

2.1.3 NFC (*Near Field Communication*)

Sendo uma tecnologia nova e, portanto, não tão difundida em dispositivos móveis quanto *bluetooth* e Wi-Fi, NFC tem como o propósito a troca de poucos dados entre dispositivos extremamente próximos, normalmente a menos de 10 centímetros [3].

A conexão é simples, necessitando somente da aproximação de ambos os dispositivos. Por não possuir grande alcance e velocidade de comunicação, seu consumo de energia é extremamente baixo.

2.2 Módulo de comunicação Wi-Fi - ESP8266

O módulo ESP8266, figura 2.1, é um dispositivo extremamente compacto porém muito versátil. Disponível em várias versões diferentes, foi desenvolvido para operar basicamente como uma ponte UART/Wi-Fi, possibilitando a comunicação sem fio entre dispositivos [4].

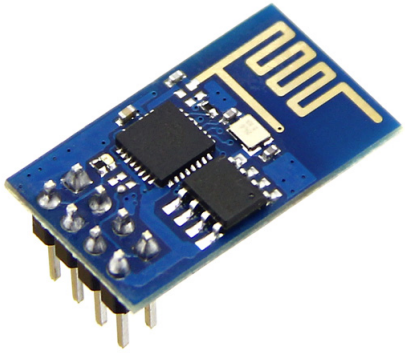

Figura 2.1: Módulo ESP8266, modelo 1

O *firmware* padrão presente no dispositivo está programado para responder via serial a comandos no padrão AT [4], onde é possível realizar as operações desejadas como busca por redes, conexão e troca de dados. Porém, esta aplicação exige que o microcontrolador controle o módulo ao invés deste ser autônomo.

Outra opção de *firmware* disponível é o NodeMCU que possui um interpretador de linguagem Lua embutido [5]. A linguagem Lua oferece mais opções ao desenvolvedor como a programação de funções a serem executadas quando um determinado evento ocorre (ao receber uma mensagem, por exemplo), possibilitando ainda a criação de *scripts* a serem salvos no próprio dispositivo, tornando-o mais independente. Em contrapartida, este *firmware* exige que mais memória esteja disponível, podendo gerar certas limitações.

Por já conter as pilhas TCP/IP e UDP/IP inclusas no *firmware*, o usuário preocupa-se somente com os comandos básicos de conexão e troca de dados sendo uma ótima ferramenta para dispositivos embarcados.

Uma terceira opção é a programação do *firmware* do módulo utilizando a IDE Arduino juntamente com uma biblioteca, tornando-o de certa forma compatível com a linguagem [6]. Esta opção apresenta a vantagem de que o código sendo executado é mais eficiente, utilizará menos memória e permitirá a criação e utilização de classes com mais abrangência, porém possuindo uma solução não tão direta quanto a opção do NodeMCU.

Com relação ao consumo de energia, o site [7] exibe uma tabela com algumas medições realizadas. É possível notar que um sistema alimentado por bateria pode tornar-se deficiente se este consumo observado for relevante em relação à capacidade dessas.

2.3 Banco de dados - SQLite

SQLite [8] é uma biblioteca que implementa um banco de dados de forma simples e rápida, sem a necessidade de configurações ou de um servidor. Por ser gratuito, compacto e com ampla compatibilidade entre sistemas, é o banco de dados mais utilizado no mundo. Como é rápido, mesmo quando operando em ambientes com pouca memória, apresenta-se como uma opção atraente para sistemas embarcados.

2.4 Android

Atualmente, milhões de dispositivos móveis estão em circulação mundialmente, sendo a maioria com OS Android [9], adquirido pela Google em 2005, sendo agora a principal desenvolvedora.

Por ser um sistema de código aberto e com a extensa documentação existente é possível desenvolver aplicativos com certa facilidade utilizando a linguagem de programação Java. Além disso, a Google disponibiliza também ferramentas para auxiliar o desenvolvimento como simuladores para computador [3].

2.4.1 Aplicativos

Um aplicativo para o sistema operacional Android é desenvolvido a partir de combinações de componentes [10]. Dos componentes existentes, serão analisados apenas os dois mais relevantes ao projeto, que são atividades e serviços. O primeiro apresenta uma interface ao usuário, enquanto o segundo realiza um trabalho no plano de fundo [10].

Um aplicativo sempre contém, também, um *App Manifest*, que possui informações essenciais do aplicativo em questão, incluindo as atividades e serviços presentes e permissões de acesso necessárias.

2.4.1.1 Manifesto (*App Manifest*)

O manifesto é responsável por informar ao sistema operacional o conteúdo de um aplicativo, as permissões necessárias e o que é capaz de manipular. Contém uma lista das classes de atividades e serviços além da capacidade de cada um deles de interpretar mensagens de *intent* [11].

Uma atividade é sempre iniciada através de uma mensagem *intent* que a descreve, po-

dendo ser tanto uma atividade específica quanto uma ação genérica, possibilitando assim o acesso a outros aplicativos [12]. Quando a mensagem *intent* apenas descreve a ação, o sistema exibe ao usuário uma lista com os aplicativos capazes de executar tal função.

Estas declarações permitem que o sistema operacional saiba quais são os componentes presentes em um aplicativo e quais as condições para que eles sejam iniciados [11].

2.4.1.2 Atividades

Uma atividade é, basicamente, uma interface para o usuário. Um aplicativo é composto por um conjunto de atividades interligadas umas às outras [12]. Cada atividade pode ter vários componentes sendo eles textos, campos de edição de texto, botões, listas, dentre outros [13]. Cada um destes elementos será brevemente descrito a seguir.

- Texto - Utilizado para exibir um texto na tela;

- Campo de edição de texto - Utilizado para que o usuário possa inserir um texto no aplicativo;

- Botão - Principal meio de interação com o usuário. A função de cada botão é programada pelo desenvolvedor;

- Lista - Exibe uma lista de itens;

É importante ter consciência do comportamento das atividades durante o desenvolvimento de um aplicativo. Quando uma nova atividade é iniciada, a execução da atividade anterior é suspensa. Quando o foco retorna a uma atividade suspensa, sua execução é retomada [12]. A figura 2.2 ilustra o ciclo de vida de uma atividade.

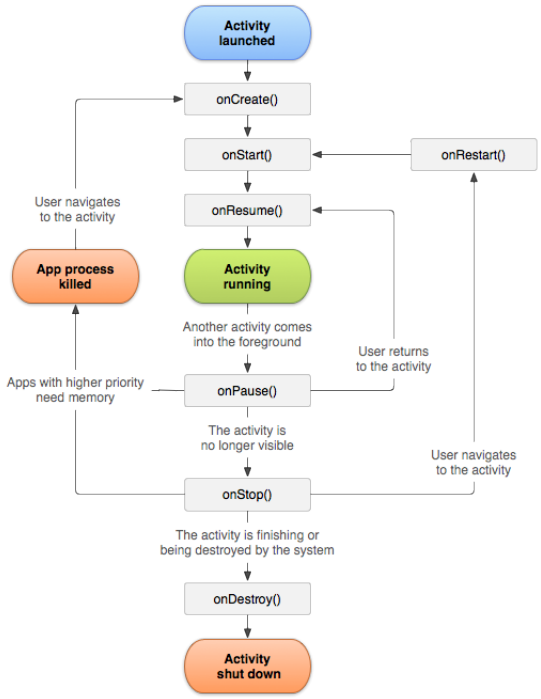

Figura 2.2: Ciclo de vida de uma atividade [12]

2.4.1.3 Serviços

Um serviço é utilizado quando alguma tarefa deve ser executada independentemente do estado atual de uma atividade. Um serviço pode ser encarregado, por exemplo, de realizar um *download*, tocar uma música, aguardar por um evento do sistema, etc [14]. Em grande parte das vezes, mesmo que a atividade que iniciou o serviço seja finalizada, este continuará ativo até finalizar a tarefa, ou indefinidamente.

Sempre há somente um processo de um serviço específico, sendo que se uma atividade tenta iniciar um serviço que já está em execução, este não será reiniciado. Atividades podem "ligar-se" a serviços permitindo a execução de métodos públicos deste, como parte da atividade [14].

O ciclo de vida de um serviço é ilustrado na figura 2.3. O diagrama à esquerda representa um serviço iniciado para a realização de uma tarefa (sem a atividade ligar-se a ele) e o diagrama a direita, quando uma atividade ligou-se ao serviço [14].

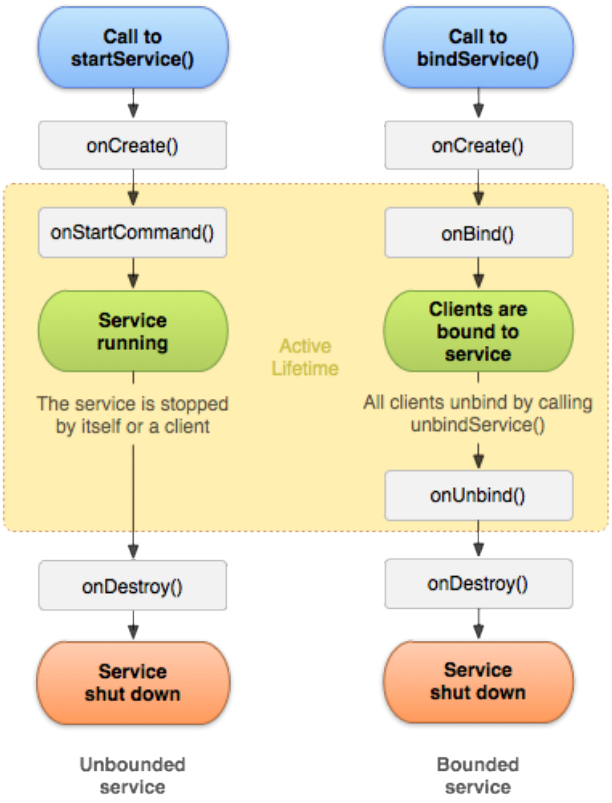

Figura 2.3: Ciclo de vida de um serviço [14]

2.4.1.4 Processos (*Threads*)

Processos são utilizados quando o processo principal de uma atividade não pode ser interrompido para realizar uma tarefa trabalhosa ou que toma tempo. Sendo assim, é possível executar esta tarefa em um processo secundário, de modo que o principal não seja afetado [15].

Enquanto um processo está aguardando por uma conexão via *bluetooth*, por exemplo, ele não pode realizar outra tarefa ao mesmo tempo. Sendo assim, um outro processo é indispensável [16].

Outro ponto importante é que o único processo que possui a permissão para alterar componentes de uma atividade (alterações na UI) é o processo principal. Como processos secundários são assíncronos, o sistema operacional não permite essas alterações e, por isso, devem executar um método pré-especificado no processo principal para realizar tal ação [15]. Esse procedimento é ilustrado na figura 2.4

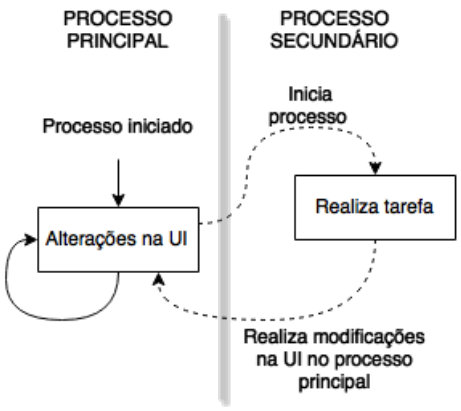

Figura 2.4: Processos principal e secundário

2.4.1.5 *Broadcast Receivers*

Broadcast Receivers são responsáveis por receberem mensagens de *intent* em que foram registrados. Provindos tanto do sistema operacional quanto de outros aplicativos, estes *intents* podem ser de tipos variados, desde eventos de conexão ou rede até estado da bateria, notificações, etc [17].

São utilizados como "interrupções", executando algum procedimento quando um evento pré-programado ocorre.

Aplicativos podem definir novas mensagens de *intent* para fins de comunicação entre suas atividades e serviços. O desenvolvedor deve garantir que estas mensagens são diferentes das existentes do sistema operacional e de outros aplicativos, visando evitar que uma mensagem dispare eventos de outra aplicação de forma não intencional.

2.4.1.6 *Bluetooth*

O sistema operacional facilita a utilização de comunicação via *bluetooth*. Os procedimentos a serem realizados, desde a busca por dispositivos próximos até a conexão, são simples e bem detalhados na documentação.

A recomendação para realizar a conexão com um dispositivo é a utilização de um *socket* em um segundo processo, visando não bloquear o processo principal. Quando a conexão for bem sucedida, outro processo toma o lugar para mantê-la ativa enquanto aguarda por mensagens [18].

2.4.1.7 Wi-Fi

Dado um endereço de IP e uma porta para ser realizada a conexão, o princípio pode ser o mesmo utilizado para comunicação via *bluetooth*, realizadas as devidas modificações.

A documentação do sistema operacional para conexão via Wi-Fi descreve apenas a utilização do padrão Wi-Fi Direct [19]. Este padrão prevê a conexão sem a necessidade de um ponto de acesso, facilitando também a busca de dispositivos compatíveis na rede [19].

Porém, por ser relativamente novo, grande parte dos dispositivos de comunicação via Wi-Fi (com exceção de aparelhos Android) não são compatíveis com este padrão, mesmo sendo muito útil e aplicável para comunicação entre dispositivos móveis com Android [20].

Capítulo 3

Material e Métodos

3.1 Definições iniciais

Antes de iniciar o desenvolvimento do aplicativo e das unidades de sensoreamento, certas características foram definidas a fim de evitar conflitos futuros. A seguir serão listados os padrões que foram escolhidos.

3.1.1 Meios de comunicação

Dentre os meios de comunicação sem fio disponíveis analisados na seção 2.1 foram selecionados dois para serem utilizados no projeto, sendo estes Wi-Fi e bluetooth, justificando-se pelas características velocidade e alcance de comunicação.

Em ambos os meios a unidade de sensoreamento se comportará como um servidor de dados (leituras dos sensores) enquanto que o dispositivo portátil (operando como cliente) se conectará a este servidor a fim de obter as leituras. A figura 3.1 ilustra superficialmente como será a relação entre as partes.

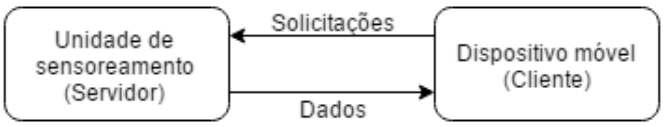

Figura 3.1: Princípio de comunicação entre unidade de sensoreamento e dispositivo portátil

Sendo assim, a unidade de sensoreamento deverá, sempre, esperar por uma conexão do cliente. Ao conectar-se, o dispositivo móvel realiza solicitações e a unidade responde de forma imperceptível ao usuário.

3.1.2 Protocolo de comunicação

O protocolo de comunicação foi criado visando evitar problemas de sincronismo que podem ocorrer devido a latência e quebra das mensagens em vários pacotes durante a troca de dados. As unidades e o aplicativo deverão receber as mensagens e concatená-las de forma a identificar quando uma mensagem está completa e decodificá-la. Sendo assim, seguem os padrões do protocolo do formato das mensagens.

- Toda mensagem deve ser iniciada e finalizada com o símbolo **$**;
- A mensagem será separada em partes, cada uma iniciada com **#**;
- A primeira parte categoriza a mensagem;
- As partes seguintes (nem sempre presentes), contém informações sobre os sensores;
- Cada sensor será identificado como um **item**;
- Cada informação de um item possuirá dois identificadores seguidos pelo dado em questão
  - Identificador do índice do item;
  - Identificador do tipo de informação;
  - Informação;
- Cada item possui 3 informações
  - Nome do item;
  - Descrição;
  - Valor de leitura;

3.1.3 Mensagens

Existirão quatro tipos de mensagens diferentes sendo duas utilizadas pelo servidor (interpretadas pelo cliente) e as outras duas utilizadas pelo cliente (interpretadas pelo servidor). A tabela 3.1 ilustra as mensagens existentes e serão definidas as regras a serem utilizadas em cada uma.

Tabela 3.1: Tabela ilustrando as mensagens existentes

Mensagens do cliente	Resposta esperada do servidor
Solicitação de lista de itens	Lista de itens
Solicitação valores dos itens	Lista de valores

As mensagens de solicitação serão compostas somente pela parte de categorização enquanto que as respostas terão conteúdo adicional. A seguir, o protocolo das mensagens de resposta é descrito.

- O menor índice de um item deve ser **0 (zero)**;
- O índice de número zero deve sempre estar presente;
- Os índices devem ser inteiros em ordem crescente com incremento obrigatório de **1 (um)**;
- Os itens, bem como suas informações, não precisam estar em ordem na mensagem;

Em resumo, a mensagem com a lista de um número N de itens do servidor deve possuir os índices de zero a $N-1$ que podem estar distribuídos em qualquer ordem. Nenhum dos índices intermediários poderá ser omitido e os três campos de cada item (nome, descrição e valor) devem estar presentes, podendo também estar em qualquer ordem. A mensagem com a lista de valores segue a mesma regra descrita, porém contém somente o campo "valor" de cada item.

3.1.4 Leituras

Por questões de padronização, fixou-se que os valores mínimos e máximos das leituras dos itens são de -100 a 100, sendo que as unidades deverão adaptar seus dados para este intervalo ou para um intervalo contido neste.

Seguindo o protocolo descrito, o cliente será capaz de identificar os nomes, descrições e valores dos itens presentes nas mensagens enviadas pelo servidor, enquanto que este identifica apenas o tipo de solicitação e envia a resposta adequada. Definido o protocolo foi possível iniciar o desenvolvimento do aplicativo e das unidades de sensoreamento.

3.2 Unidade de sensoreamento

Uma unidade de sensoreamento é composta basicamente por um microcontrolador, um módulo de comunicação sem fio e um número N de sensores. Os sensores e o módulo são conectados ao microcontrolador e este, por sua vez, será responsável pela interpretação das solicitações e pelas respostas a serem enviadas.

As figuras 3.2 e 3.3 mostram a organização de uma unidade de sensoreamento e seu método de operação, respectivamente.

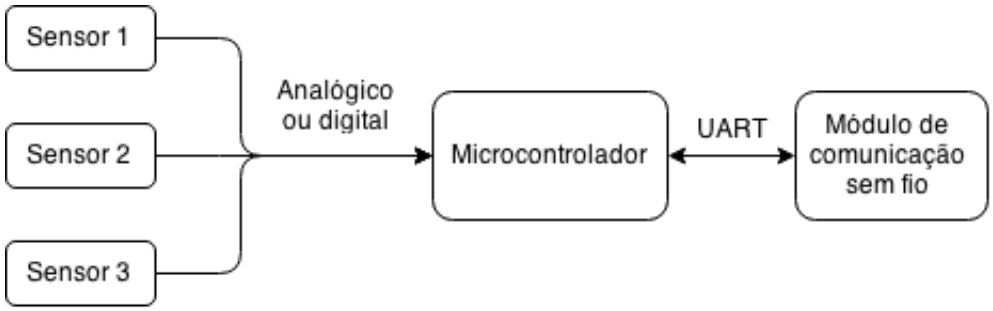

Figura 3.2: Organização de uma unidade de sensoreamento

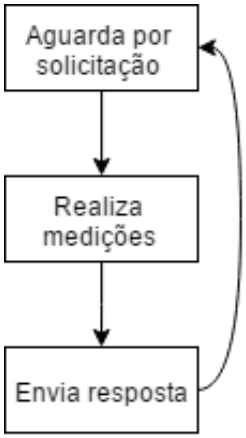

Figura 3.3: Método de operação de uma unidade

Foi criada uma biblioteca em C para auxílio no desenvolvimento de uma unidade de sensoreamento. Mais informações sobre a biblioteca pode ser encontrada no Apêndice A.

3.2.1 Módulos de comunicação

Os módulos são responsáveis por realizar a ponte de comunicação sem fio entre a unidade de sensoreamento e o dispositivo portátil, como ilustrado na figura 3.4.

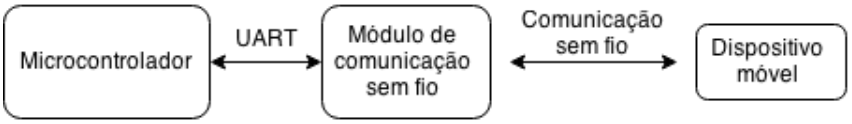

Figura 3.4: Ponte de comunicação

3.2.1.1 Módulo Wi-Fi - ESP8266

Este módulo, configurado utilizando a linguagem de programação Lua, é responsável por enviar todo o conteúdo recebido pela comunicação UART através da conexão TCP/IP estabelecida, e vice-versa, permitindo a troca de mensagem entre ambos os dispositivos.

Primeiramente, o módulo tenta conectar-se a rede Wi-Fi em que está configurado. Caso não obtenha sucesso, uma rede própria será criada. O usuário pode, a qualquer momento que estiver conectado na mesma rede que o módulo, reconfigurá-lo para uma nova rede utilizando uma mensagem no formato "WIFI:SSID-PSW", onde SSID e PSW são o nome e senha da rede, respectivamente. O diagrama de blocos da figura 3.5 mostra este método de operação.

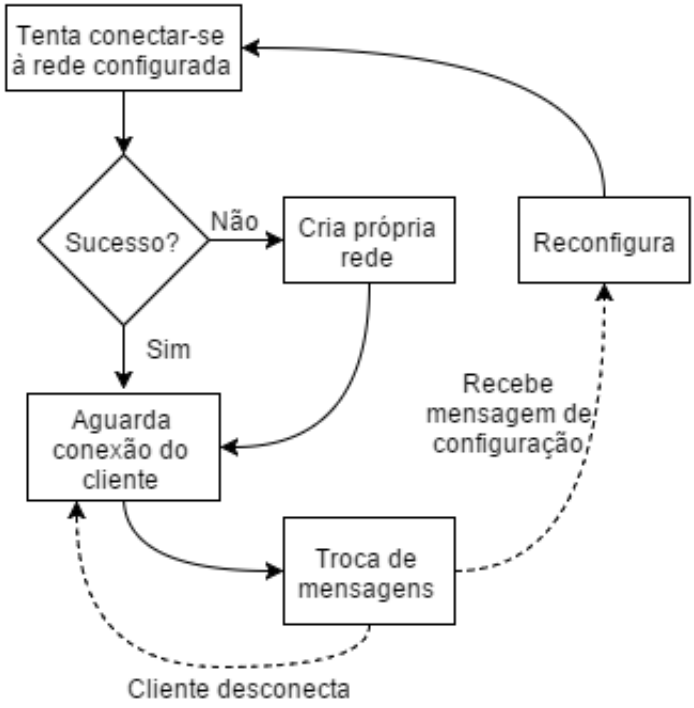

Figura 3.5: Diagrama de blocos da operação do ESP8266

No momento que o módulo cria sua própria rede, seu endereço IP é, automaticamente, "192.168.4.1".

Espera-se também que o módulo seja "rastreável", isto é, encontrar o IP do dispositivo

na rede, para que o cliente possa realizar a conexão. Para isso, o módulo responderá a mensagens *broadcast* UDP, com o endereço em questão, em uma porta pré-definida. A figura 3.6 exemplifica um dispositivo requisitando o endereço de IP do módulo, dado que ambos estão conectados à mesma rede Wi-Fi.

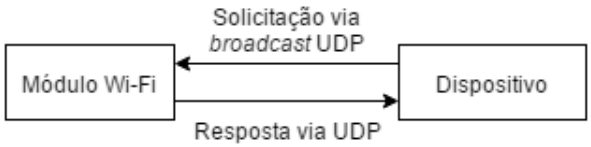

Figura 3.6: Processo de rastreamento do módulo em uma rede

3.2.1.2 Módulo *Bluetooth* - RN42

O módulo RN42, figura 3.7, é responsável por aguardar pela conexão do dispositivo móvel e, quando esta estiver ativa, enviar os dados recebidos via *bluetooth* para a interface UART e vice-versa.

Figura 3.7: Módulo RN42

O usuário pode também buscar pelo módulo utilizando o dispositivo móvel a fim de realizar o emparelhamento, que é necessário antes de iniciar a conexão.

3.2.2 Sensores

Como o propósito do trabalho é a criação de um método de leitura remota de sensores, não focou-se a atenção a sensores específicos já que cabe apenas à unidade de sensoreamento realizar as leituras e comunicar ao dispositivo móvel. Sendo assim, o usuário do aplicativo não necessita saber como uma unidade realiza as leituras, direcionando esta responsabilidade ao desenvolvedor da unidade. Portanto, qualquer sensor utilizado no projeto visa somente exemplificar aplicações e não foram considerados relevantes.

3.2.3 Unidade - AT89LP4052

Compatível com o conjunto de instruções da arquitetura MCS®51, o microcontrolador AT89LP4052 é capaz de processar até 20 milhões de instruções por segundo já que realiza a leitura de um byte por ciclo de *clock*. Além disso, possui até quinze pinos de entrada ou saída, *timers*, SPI, UART e um comparador analógico [21].

É um microcontrolador extremamente simples, contendo 4K bytes de ROM e 256 bytes de RAM, podendo operar entre 2,4V a 5,5V com um consumo de energia reduzido (cerca de 5,5mA quando ativo, atingindo apenas 5μA em modo *power-down*), tornando-o uma ferramenta interessante em aplicações onde energia é um recurso escasso.

A figura 3.8 mostra a placa criada operando a 16MHz. Esta placa contém cinco entradas digitais conectadas a botões que, quando pressionados, colocam nível lógico alto na porta correspondente, simulando a existência de sensores. Além disso, a placa contém terminais de alimentação, um regulador de tensão de 3,3V , um cristal e um botão de *reset*, além de outros componentes necessários para o funcionamento.

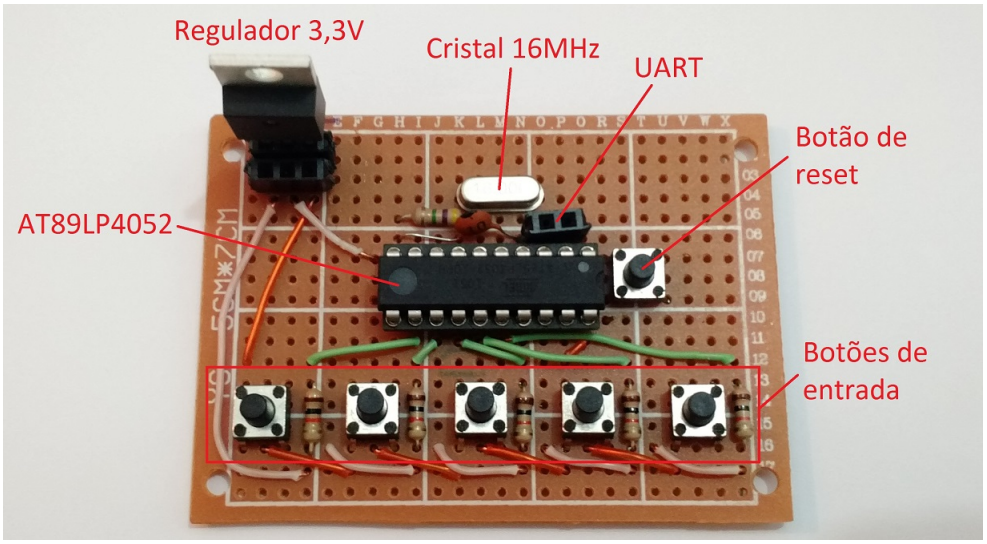

Figura 3.8: Unidade de sensoreamento com AT89LP4052

No *software* presente no microcontrolador, cada item está configurado como mostrado na tabela 3.2, sendo que os valores serão **0** caso a respectiva porta esteja em nível lógico baixo (botão solto) e **1**, caso contrário (botão pressionado).

Tabela 3.2: Itens da unidade com AT89LP4052

Índice do item	Nome	Descrição
00	Dig. Input 1	Boolean
01	Dig. Input 2	Boolean
02	Dig. Input 3	Boolean
03	Dig. Input 4	Boolean
04	Dig. Input 5	Boolean

Para este microcontrolador foi inviável utilizar a biblioteca criada pois a estrutura necessária requer bastante memória de dados. Para solucionar este problema, as informações dos sensores foram armazenadas na memória de programa e as funções foram adaptadas.

3.2.4 Unidade - Arduino Uno

A placa Arduino Uno é composta por um microcontrolador ATMEGA328P de arquitetura AVR, possuindo comunicação UART, SPI e I2C, além de portas digitais e analógicas. O microcontrolador é programado via USB de forma simples a partir de um computador, utilizando as linguagens de programação C ou C++. Além disso, possui até quinze pinos de entrada ou saída, *timers*, SPI, *clock* de 16MHz e sua IDE (*Integrated Development Environment*) própria [22].

A figura 3.9 mostra a placa Arduino Uno à esquerda e a IDE à direita.

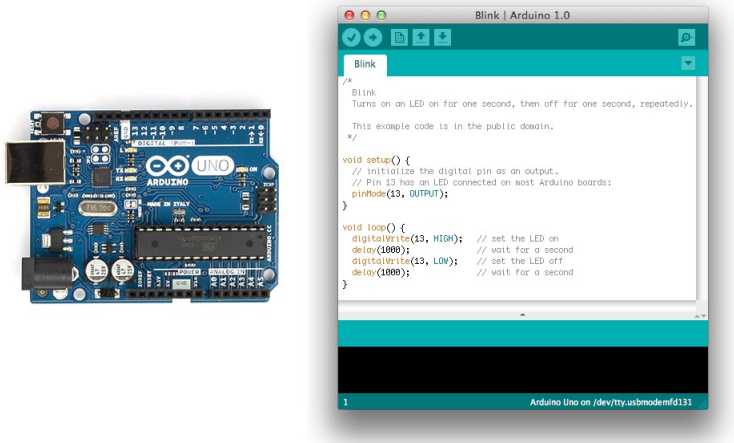

Figura 3.9: À esquerda a placa Arduino Uno e à direita a IDE.

Para facilitar o entendimento do método de operação de uma unidade, foi selecionado um acelerômetro de três eixos (ADXL345) para operar como um sensor, conectado ao arduino

via SPI. O usuário pode então realizar leituras da aceleração dos três eixos disponíveis. A unidade é mostrada na figura 3.10.

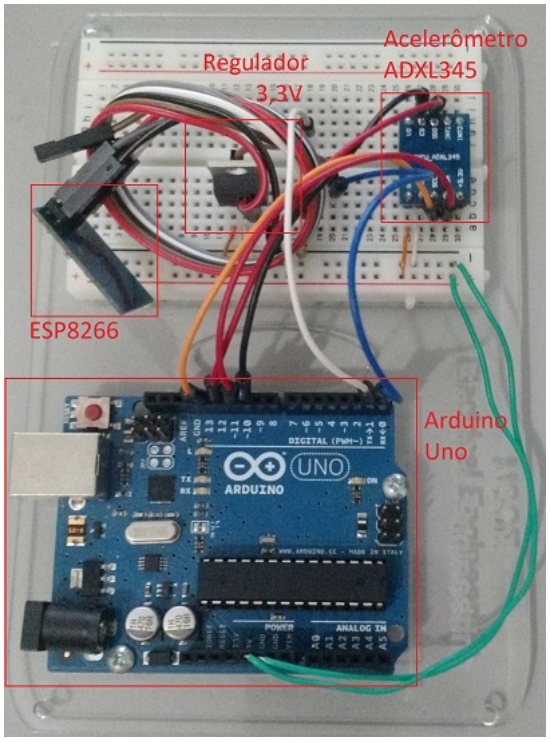

Figura 3.10: Unidade de sensoreamento com Arduino e ADXL345

Para o *software*, foi utilizada a biblioteca desenvolvida visando simplificar a programação. As informações dos sensores, salvos na estrutura, estão listados na tabela 3.3.

Nome	Descrição
Accelerometer X-axis	-100 to 100, 50 equals to 1g
Accelerometer Y-axis	-100 to 100, 50 equals to 1g
Accelerometer Z-axis	-100 to 100, 50 equals to 1g

Tabela 3.3: Itens da unidade com Arduino

3.3 Aplicativo

Para a criação do aplicativo foi selecionada a IDE Android Studio, que já possui o Android SDK embutido, utilizando a linguagem de programação Java. Uma segunda opção seria a utilização da IDE Eclipse (desenvolvida pela Oracle) instalando o Android SDK manualmente.

As figuras 3.11 e 3.12 mostram, superficialmente, como o aplicativo foi estruturado e, em

seguida, serão descritas as funcionalidades esperadas e os métodos utilizados.

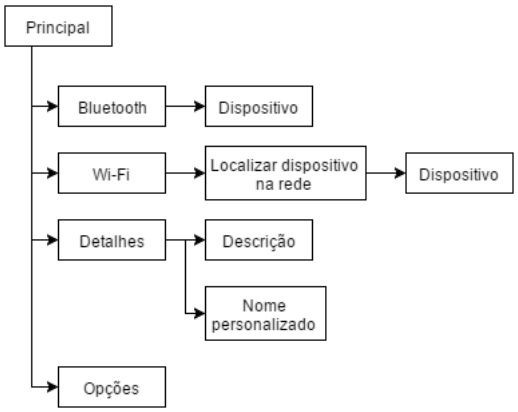

Figura 3.11: Funcionalidades do aplicativo

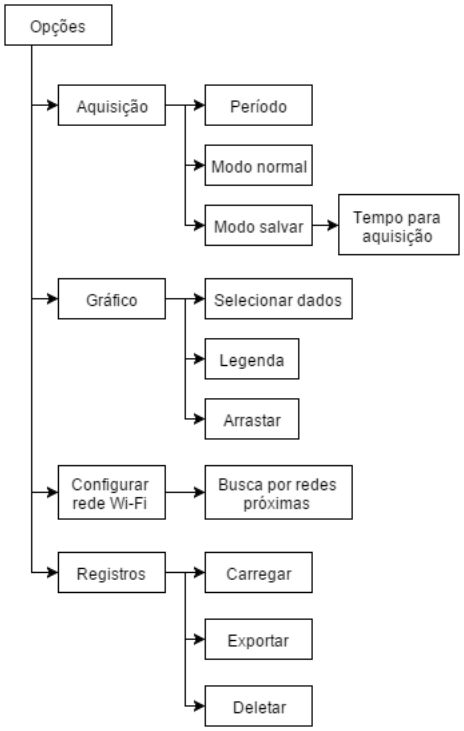

Figura 3.12: Bloco ’Opções’ expandido

3.3.1 Atividade - Principal

Na atividade inicial do aplicativo são apresentadas ao usuário informações sobre a conexão, leituras dos sensores e botões para acesso às funções. A princípio o aplicativo não está conectado a uma unidade de sensoreamento.

Foram criados dois botões para conexão (Wi-Fi e *bluetooth*) que iniciam respectivas atividades para seleção dos dispositivos, um botão para desligar as conexões existentes e um botão

para um menu de opções.

Estas funções serão descritas nas seções 3.3.3, 3.3.4 e 3.3.5, a seguir.

Além disso, o usuário pode acessar os detalhes de qualquer um dos itens em uma nova atividade, selecionando-o na lista.

3.3.2 Atividade - Detalhes

Esta atividade exibe ao usuário o nome do item selecionado e sua descrição. Além disso, permite a adição de um nome personalizado, caso deseje. Quando o nome de um item é alterado o nome original não é modificado, mas o novo será o utilizado na exibição.

3.3.3 Atividade - Conexão Wi-Fi

Ao iniciar, a atividade verifica se o dispositivo móvel está conectado a uma rede sem fio. Em caso negativo é exibida uma notificação ao usuário e, caso contrário, é requisitado o IP do servidor e porta onde a conexão está sendo aguardada. Além disso, esta atividade inclui um botão para busca na rede e uma lista com os dispositivos encontrados.

Quando o usuário preencher os campos IP e porta e confirmar, um serviço tentará realizar a conexão. Caso haja sucesso, servidor e cliente podem trocar mensagens sendo que o cliente enviará automaticamente uma mensagem requisitando a lista de itens.

Vale ressaltar que, sempre que o usuário retornar a esta atividade, os dados inseridos anteriormente serão recarregados nos respectivos campos de texto.

Mais informações sobre o serviço responsável pela conexão será descrito na seção 3.3.10.

3.3.4 Atividade - Conexão *bluetooth*

A princípio, esta atividade verifica se a comunicação via *bluetooth* do dispositivo móvel está habilitada e, em caso negativo, notifica o usuário. Caso contrário, é exibida uma lista de dispositivos *bluetooth* na forma "Nome - MAC *Address*", que já foram emparelhados. O usuário pode então selecionar um dispositivo da lista e um serviço tentará realizar a conexão. Caso haja sucesso, a troca de mensagens é habilitada e o cliente envia automaticamente uma mensagem requisitando a lista de itens.

Mais informações sobre o serviço responsável pela conexão será descrito na seção 3.3.11.

3.3.5 Atividade - Opções

Esta atividade exibe diversos botões ao usuário, sendo estes "Aquisição", "Gráfico", "Configuração Wi-Fi" e "Registros". Cada um destes botões inicia uma atividade que será descrita nas seções 3.3.6, 3.3.7, 3.3.8, 3.3.9, respectivamente, a seguir.

3.3.6 Atividade - Aquisição

Esta atividade só pode ser iniciada se uma conexão já está ativa e apresenta uma interface com o estado atual e opções da aquisição. A aquisição é feita por um serviço que será descrito na seção 3.3.12, porém suas configurações são realizadas pelo usuário na atividade em questão.

É exibido ao usuário se a aquisição está sendo realizada ou não e o período que foi configurado. O usuário pode iniciar e parar a aquisição de dados e definir seu período, além da possibilidade de habilitar o armazenamento dos dados adquiridos. Aquisições armazenadas serão exibidas na atividade "Registros" descrita em 3.3.9.

3.3.7 Atividade - Gráfico

Esta atividade contém um gráfico e uma lista de itens que foi obtida do servidor. Quando o usuário seleciona um ou mais (limitado a seis) itens na lista, o gráfico exibe as respectivas curvas de variação no tempo. O eixo vertical do gráfico representa os valores e o eixo horizontal o tempo decorrido.

O eixo dos valores é um eixo crescente para cima onde os limites se ajustam automaticamente para os valores existentes.

O eixo do tempo é normalizado pelo período de aquisição e, por isso, cada unidade corresponde a um período de aquisição decorrido. O zero então representa a primeira leitura realizada e, qualquer instante a direita dele representa as leituras seguintes. A figura 3.13 ilustra este eixo para um tempo de aquisição de 500 milissegundos.

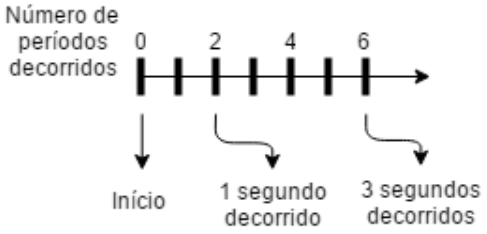

Figura 3.13: Eixo do tempo normalizado para um período de 500ms

O eixo do tempo ajusta seus limites de acordo com o tempo de aquisição (T_{aq}) decorrido. Caso o tempo decorrido seja menor que $100T_{aq}$, o eixo se expande progressivamente. Ao atingir $100T_{aq}$, o eixo cessa sua expansão e o gráfico é deslocado para a esquerda a cada novo período. A figura 3.14 mostra como ocorre este deslocamento a partir de um instante de tempo maior que $100T_{aq}$.

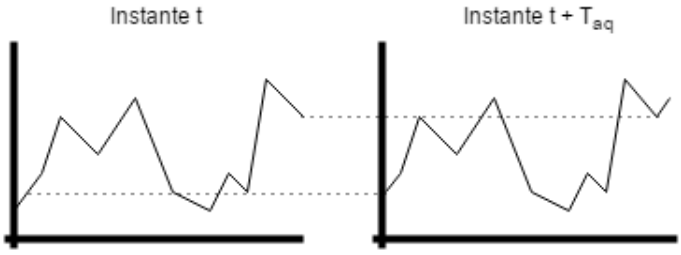

Figura 3.14: Deslocamento do gráfico

3.3.8 Atividade - Configuração Wi-Fi

Esta atividade é utilizada para realizar a configuração de rede Wi-Fi da unidade de sensoreamento e só estará disponível se o aplicativo não estiver conectado via *bluetooth*. Como especificado na seção 3.2.1.1 que trata do módulo ESP8266, caso o módulo não consiga conectar-se à rede configurada ele criará uma própria. O usuário pode, porém, reconfigurar o módulo de modo a tentar a conexão com outra rede. A interface apresenta quatro campos de texto para a inserção da nova configuração:

- IP - Endereço do módulo na rede. Se deixado em branco, será utilizado o padrão **192.168.4.1** que é o endereço do módulo na rede criada por ele;
- Porta - Porta a ser utilizada na comunicação;
- SSID - Nome da rede na qual o módulo deve tentar conectar-se;

- Senha - Senha da rede. Deixar em branco indica que a rede não possui senha;

O usuário pode também buscar por redes Wi-Fi ao alcance do dispositivo móvel a fim de dispensar o preenchimento do campo SSID.

Ao confirmar, através da conexão realizada pelo serviço Wi-Fi, o aplicativo envia a mensagem de configuração. Caso a conexão ainda não tenha sido iniciada, o aplicativo tenta realizá-la com o endereço de IP e porta especificadas, e, se bem sucedida, a mensagem de configuração será enviada.

3.3.9 Atividade - Registros

Esta atividade exibe ao usuário uma lista de aquisições que foram armazenadas. Ao selecionar uma, são exibidos os detalhes deste registro e as opções para "Carregar", "Exportar" ou "Deletar".

Ao selecionar "Carregar", a atividade do gráfico é iniciada com todos os dados armazenados. Selecionando "Exportar" fará com que o aplicativo crie uma planilha de dados no formato XLS (padrão do *software* Microsoft Excel) com todas as informações e solicita ao usuário qual aplicativo ele quer utilizar para exportação.

3.3.10 Serviço - Conexão Wi-Fi

Este serviço é responsável por realizar e manter a conexão Wi-Fi utilizando o protocolo TCP/IP via *sockets*. Este processo aguarda por mensagens de *intent* para realizar certas tarefas, especificadas a seguir.

- Iniciar conexão, modo normal - quando recebida, inicia um processo paralelo que tenta realizar a conexão com o endereço e porta selecionados. Quando bem sucedido, é iniciado então outro processo para manter a conexão ativa, enquanto permite a troca de mensagens entre as partes;
- Iniciar conexão, modo configuração da rede Wi-Fi - mesmo processo que o item anterior, porém ao conectar-se, automaticamente envia a mensagem de configuração da nova rede. Se a conexão já estiver ativa, somente envia a mensagem;
- Enviar mensagem - Envia uma mensagem através da conexão;
- Finalizar serviço - Utilizada para parar os serviços ativos quando o aplicativo é finalizado;

Além disso, o serviço também envia mensagens de *intent* para as atividades, descritas a seguir.

- Conexão bem sucedida - Conexão foi bem sucedida e está ativa;

- Servidor desconectado - A conexão entre cliente e servidor foi desfeita;

- Mensagem recebida - Uma mensagem foi recebida e está pronta para ser lida;

3.3.11 Serviço - Conexão *bluetooth*

Assim como no serviço de conexão Wi-Fi, este serviço aguarda por mensagens de *intent* para realizar as ações, sendo a principal diferença entre eles, o protocolo de comunicação. Além disso, por motivos óbvios, este serviço não aguarda pela mensagem de *intent* para configuração da rede Wi-Fi. Do restante, ambos os serviços operam da mesma maneira.

3.3.12 Serviço - Aquisição

Este serviço é responsável por controlar a aquisição dos dados e armazenar as informações dos sensores enquanto o aplicativo está em execução. Diferentemente dos outros serviços, este não aguarda por mensagens de *intent* para iniciar ou finalizar a aquisição.

Como este serviço é responsável por armazenar os dados da lista de itens, as atividades devem ligar-se a ele para obterem acesso a estas informações. Sendo assim, ao invés de utilizar mensagens de *intent*, as atividades executam métodos públicos do serviço em questão para iniciar e finalizar a aquisição, dentre outras ações.

Quando a aquisição é iniciada, uma solicitação é enviada à unidade de sensoreamento a cada período configurado pelo usuário. Ao mesmo tempo, a última leitura recebida pelo aplicativo é armazenada de forma a ignorar problemas de latência na comunicação. A figura 3.15 mostra como o serviço trata essas possíveis latências, onde os pulsos representam a ocorrência dos respectivos eventos.

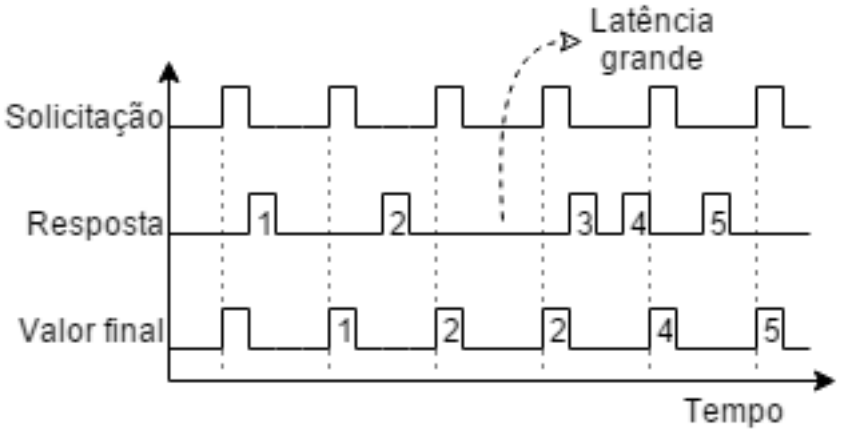

Figura 3.15: Linha do tempo da aquisição

Como pode ser visto, após a latência na comunicação ter sido grande, o valor final da leitura não foi alterado e se manteve igual a 2. Logo em seguida, duas mensagens de resposta são recebidas antes de completar um novo período e, por isso, somente a última leitura de valor igual a 4 foi registrada.

A cada período de amostragem, o serviço também envia uma mensagem de *intent* para informar às atividades que um período decorreu, mesmo que os valores não tenham sidos alterados.

Se o usuário optou por armazenar a aquisição, os dados das leituras são registrados no banco de dados a cada período.

Capítulo 4

Resultados e Discussões

4.1 Protocolo de comunicação e mensagens

Como citado anteriormente, as mensagens serão categorizadas a partir do seu significado. A tabela 4.1 mostra estas categorias e o trecho de mensagem correspondente.

Tabela 4.1: Categorias de uma mensagem

Categoria	Trecho de mensagem
Solicitação de lista de itens	**#req_list**
Solicitação valores dos itens	**#req_val**
Resposta com lista de itens	**#it_list**
Resposta com valores dos itens	**#it_val**

4.1.1 Mensagens de solicitação

As mensagens de solicitação contém somente a parte de categorização da mensagem, sendo, portanto, `$#req_list$` e `$#req_val$` para requisição da lista de itens e lista de valores, respectivamente.

4.1.2 Mensagens de resposta

A mensagem com a lista de itens deve conter, além da categorização, as informações (nome, descrição e valor) de cada sensor da unidade, sendo que cada trecho deve possuir dois identificadores (índice e categoria), seguidos da informação em si. Sendo assim, a categoria para uma informação do tipo "**nome**" será "**n**", do tipo "**descrição**" será "**d**" e do tipo "**valor**" será "**v**". Com isso, a tabela 4.2 mostra como são os trechos da mensagem para um item de

índice **zero**, com nome **Sensor 1**, descrição **-100 a 100** e valor **25**.

Tabela 4.2: Informações de um item

Informação	Trecho de mensagem
Nome do item	**#00nSensor1**
Descrição do item	**#00d-100 a 100**
Valor do item	**#00v25**

Como pode ser visto, o trecho "**00**" representa o índice do item, seguido pelo tipo da informação e a própria informação. Como especificado, estes trechos não necessitam estar em ordem na mensagem. Além disso, caso a unidade possua mais de um sensor, é obrigatório que os índices seguintes sejam **1, 2, 3, ...,** $N-1$ onde N é o número total de sensores.

Para exemplificar, dada uma unidade com **dois** sensores digitais e **um** sensor analógico, como ilustrados na tabela 4.3, será apresentada uma mensagem com a lista de itens.

Tabela 4.3: Exemplo de uma unidade

Índice do item	Nome	Descrição	Valor
00	Proximidade	Digital - Ativado quando um objeto se aproxima	0
01	Luminosidade	Digital - Ativado quando o ambiente está claro	1
02	Tensão	Analógico - Nível de tensão. 0=0V, 100=5V	26

Uma possível mensagem com a lista de itens para esta unidade seria:

```
$#it_list#00nProximidade#00dDigital - Ativado quando um objeto se aproxima
  #00v0#01nLuminosidade#01dDigital - Ativado quando o ambiente está claro
    #01v1#02nTensão#02dAnalógico - Nível de tensão.  0=0V, 100=5V#02v26$
```

É interessante citar que uma mensagem desta mesma unidade com a lista de valores, não possuiria as informações "nome" e "descrição", somente os campos de valores atualizados, além da categoria alterar-se para "**#it_val**".

4.2 Aplicativo

4.2.1 Início

Ao iniciar o aplicativo é exibida ao usuário a atividade da figura 4.1 onde é informado o estado da conexão e a lista de itens, além de botões de controle.

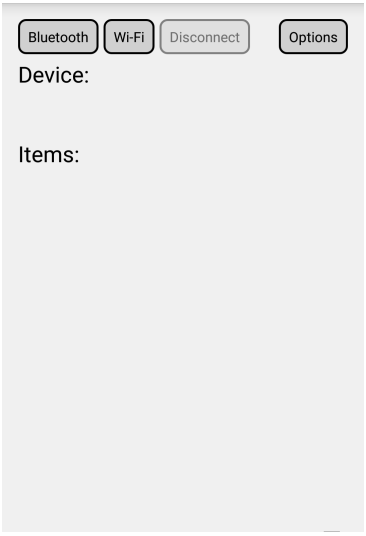

Figura 4.1: Início do aplicativo

4.2.2 Conexão Wi-Fi

Para realizar uma conexão via Wi-Fi, o usuário deve pressionar o botão "Wi-Fi" e a atividade da figura 4.2 será iniciada. Nesta atividade, é requisitado o IP e a porta do servidor para a realização da conexão e o usuário pode realizar uma busca na rede por dispositivos pressionando "Scan". A figura 4.3 mostra um exemplo de busca na rede local.

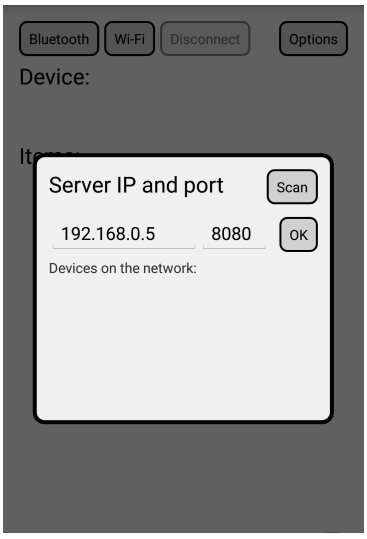

Figura 4.2: Conexão Wi-Fi

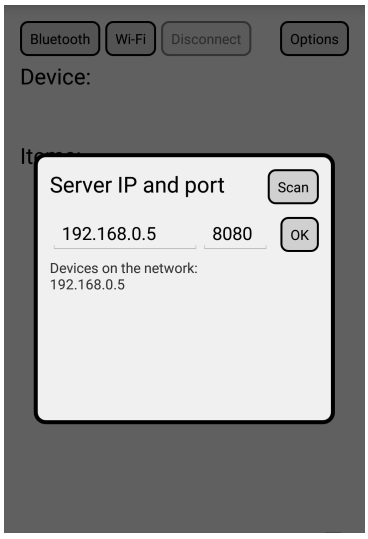

Figura 4.3: Busca na rede

Após o usuário inserir os dados e pressionar "Ok", o aplicativo retorna para a atividade inicial e um serviço tentará realizar a conexão com o servidor. Se obtiver sucesso, o estado da conexão será alterado para a cor verde e o aplicativo automaticamente envia a requisição da

lista de itens. Quando o servidor responder a mensagem com esta lista, as informações serão exibidas ao usuário como na figura 4.4.

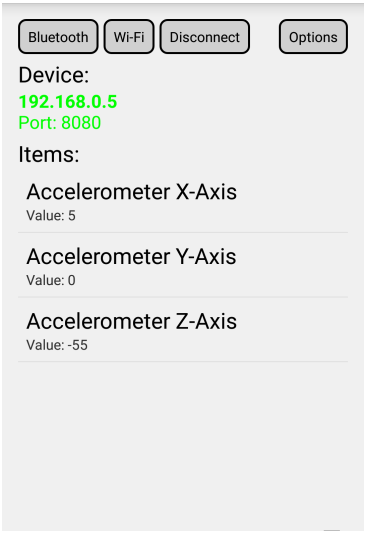

Figura 4.4: Lista de itens recebida

4.2.3 Conexão *bluetooth*

Como outra opção, para realizar uma conexão via *bluetooth*, o usuário deve pressionar o botão "Bluetooth" na atividade inicial, e a atividade da figura 4.5 será iniciada. Em seguida o usuário pode escolher na lista de dispositivos já emparelhados com qual ele deseja conectar-se. Ao realizar esta seleção, o aplicativo realiza os mesmos procedimentos descritos para conexão Wi-Fi.

Figura 4.5: Conexão *bluetooth*

4.2.4 Detalhes de um item

Após uma conexão bem sucedida, o usuário pode realizar a aquisição de dados. Caso seja de interesse, é possível acessar os detalhes de cada um dos itens da lista selecionando-os. Para exemplificar, será alterado o nome de exibição do item 1 da lista para "Eixo X", como pode ser visto na figura 4.6.

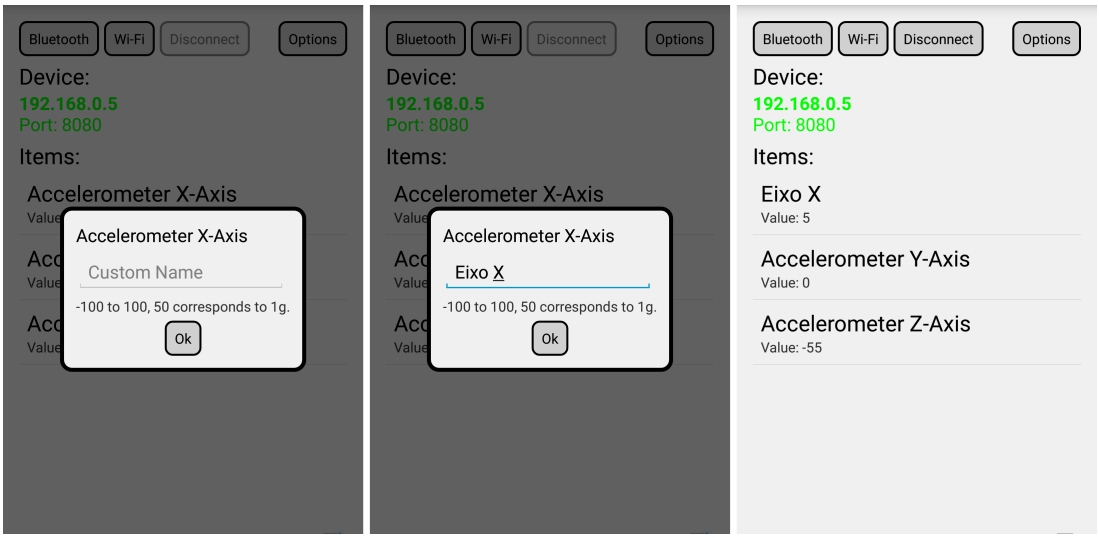

Figura 4.6: Detalhes de um item e alteração do nome

É importante notar que o nome que foi alterado é somente o de exibição. O nome original voltará a ser exibido caso o campo "Custom name" seja deixado em branco.

4.2.5 Aquisição

Para iniciar a aquisição, o usuário deve selecionar "Options" e, em seguida, "Acquisition" como na figura 4.7.

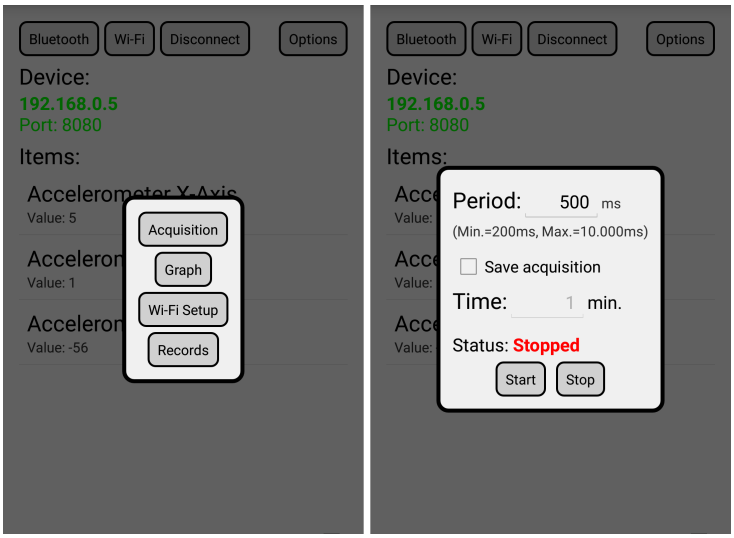

Figura 4.7: Configurar aquisição

Nesta atividade, o usuário pode verificar o estado atual, configurar o período, iniciar e parar o processo de aquisição, além de habilitar o armazenamento dos dados. Se o armazenamento dos dados for habilitado, o usuário tem a opção de selecionar por quanto tempo a aquisição deve ser realizada. Quando a aquisição for iniciada ou interrompida, uma mensagem será exibida ao usuário e o estado será atualizado na tela. Isso pode ser observado na figura 4.8.

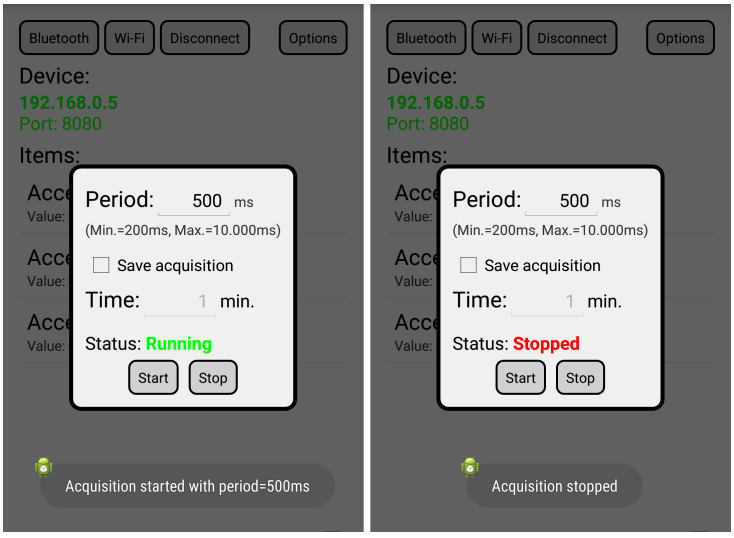

Figura 4.8: Início e parada da aquisição

Com a aquisição em andamento, o usuário pode observar a atualização dos valores na atividade inicial como na figura 4.9.

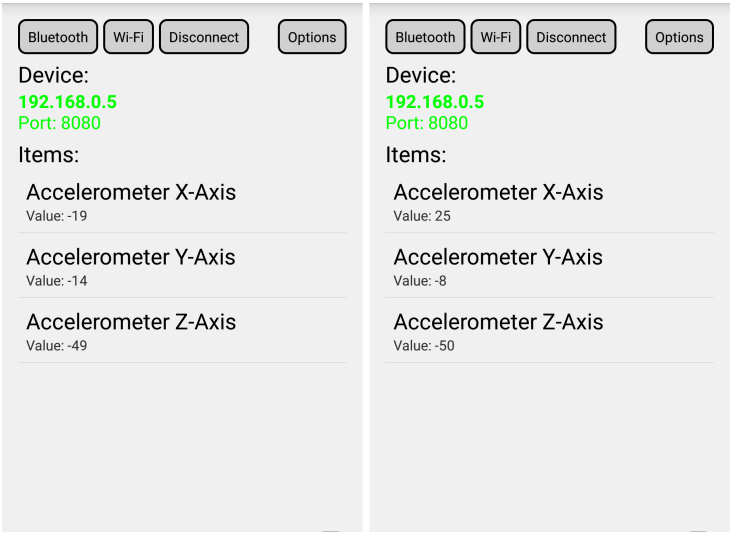

Figura 4.9: Aquisição em andamento

4.2.6 Gráfico

É possível também ver a variação das leituras em um gráfico selecionando "Options" e "Graph". Nesta atividade, é exibido um gráfico onde o usuário pode selecionar quais itens devem ser exibidos, como na figura 4.10.

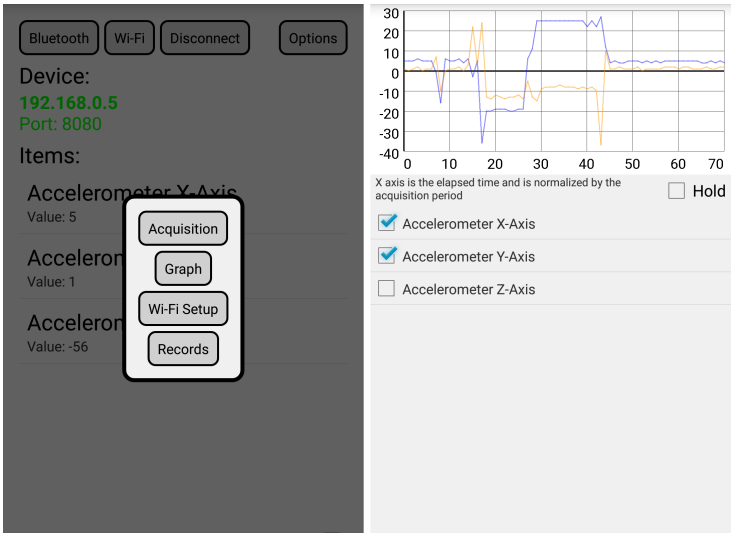

Figura 4.10: Gráfico

É possível também ativar e desativar a legenda do gráfico pressionando sobre ele, como na figura 4.11, além de permitir "arrastá-lo" para os lados direito e esquerdo quando o número de leituras ultrapassar 100, possibilitando a visualização dos dados anteriormente obtidos.

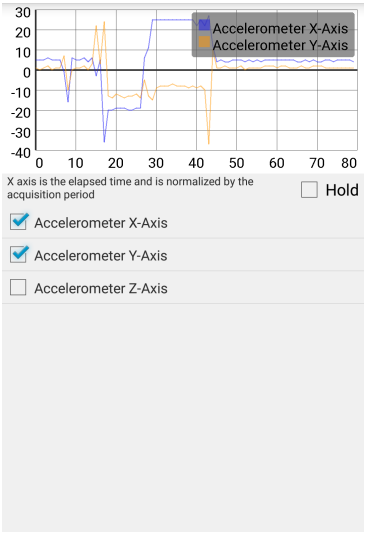

Figura 4.11: Legenda do gráfico

Um aspecto do gráfico interessante de ser citado é que, se os itens selecionados tiverem valores digitais (zero ou um), o gráfico adaptará seus limites verticais para exibir os dados de forma mais agradável. A figura 4.12 mostra um exemplo de alguns itens deste tipo sendo exibidos.

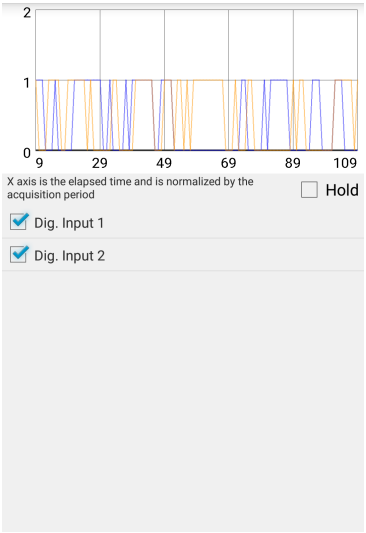

Figura 4.12: Gráfico exibindo somente valores digitais

Além disso, esta atividade conta com a opção "**Hold**" que ao invés de deslocar o gráfico a cada novo período, o mesmo intervalo será exibido. Isso garante que novos dados não interferirão na visualização mas ainda sejam incluídos à direita dos existentes.

4.2.7 Configuração de rede Wi-Fi

Outra funcionalidade é a realização da configuração de rede Wi-Fi de uma unidade de sensoreamento. Acessando o menu "Options" e "Wi-Fi Setup", é requisitado ao usuário o endereço de IP da unidade, a porta para comunicação e os dados da rede sem fio desejada. É importante citar que se já estiver conectado a uma unidade, o campo IP e porta serão automaticamente preenchidos e não será possível realizar alterações. Caso não esteja conectado, deixar o campo IP ou porta vazios fará com que sejam utilizados "192.168.4.1" e "8080", que são o endereço e porta padrão do módulo Wi-Fi na rede criada por ele. A figura 4.13 mostra essa atividade.

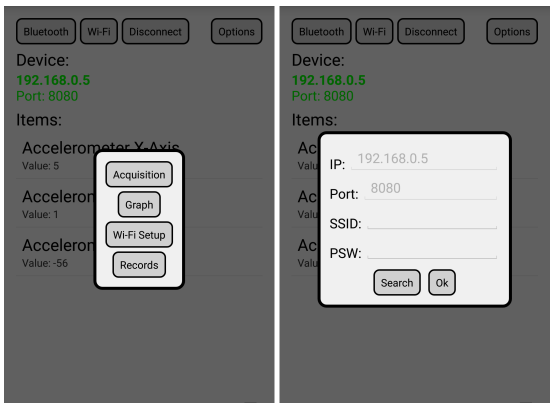

Figura 4.13: Configurar rede Wi-Fi da unidade

É possível também, para agilizar o processo de preenchimento do campo "SSID", buscar por redes ao alcance do dispositivo móvel através do botão "Scan", que listará as redes disponíveis e preencherá automaticamente o campo "SSID" com a rede selecionada, como mostrado na figura 4.14.

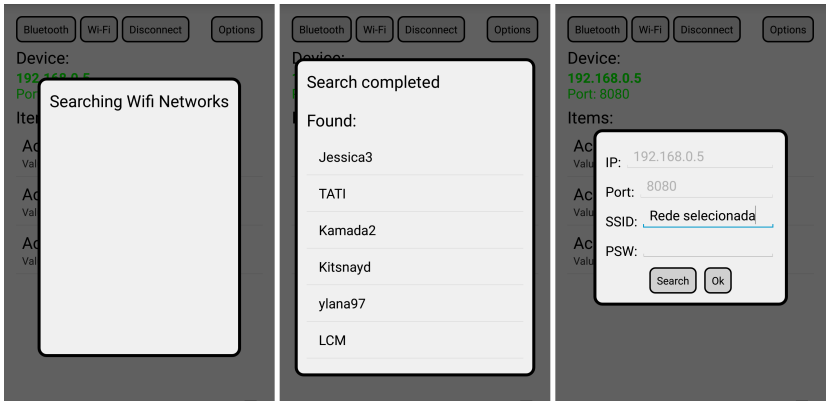

Figura 4.14: Busca por redes ao alcance

Ao pressionar o botão "Ok" nesta atividade, o aplicativo automaticamente tenta realizar a conexão com o endereço especificado e envia a mensagem de configuração de rede se obtiver sucesso.

4.2.8 Registros

Por fim, o usuário também pode consultar aquisições que foram armazenadas no dispositivo. Acessando o menu "Options" e "Records", uma lista de registros salvos serão exibidos, organizados por ordem cronológica, como demonstrado na figura 4.15.

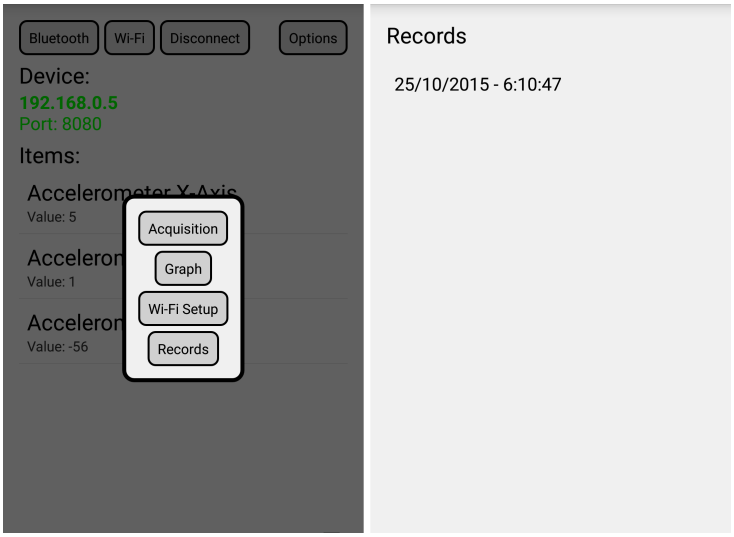

Figura 4.15: Aquisições salvas

Ao selecionar um dos registros, uma outra atividade exibirá detalhes básicos daquela aquisição como número de itens, período de aquisição, número de leituras e datas de início e término. Além disso, o usuário pode carregar a aquisição para visualização dos valores obtidos, exportar os dados e deletar tal registro. A figura 4.16 mostra a atividade contendo os detalhes de uma aquisição realizada.

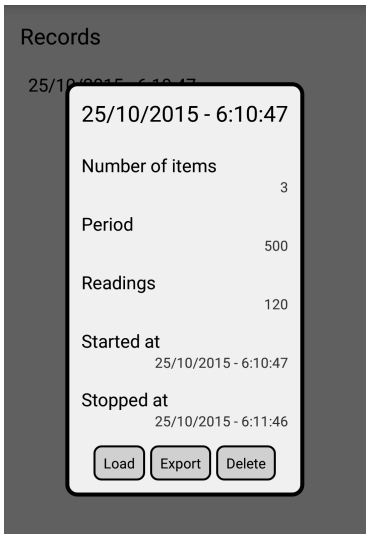

Figura 4.16: Detalhes de uma aquisição

Ao selecionar "Export", o aplicativo cria uma planilha com os dados no formato XLS e requisita ao sistema operacional por um aplicativo a ser utilizado para exportação onde, dentre as opções disponíveis, estão aplicativos de e-mail. As figuras 4.17 e 4.18 mostram, respectivamente, a solicitação do aplicativo e a escolha do Gmail, visando a exportação via e-mail. Nota-se que o arquivo a ser exportado já consta como anexo após a seleção do aplicativo.

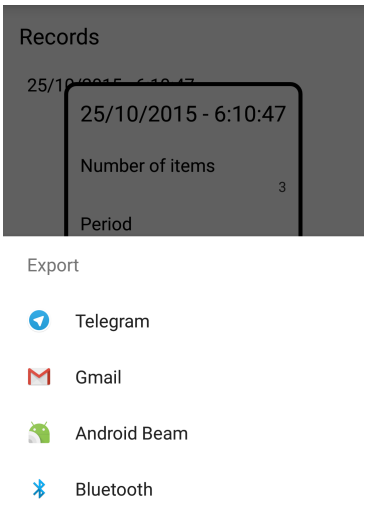

Figura 4.17: Solicitação do aplicativo

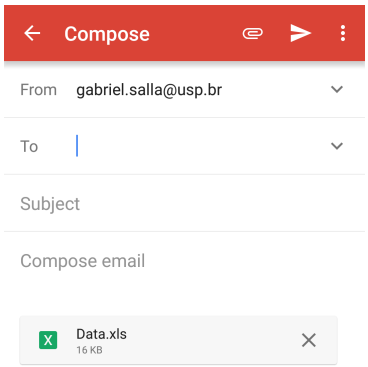

Figura 4.18: Aplicativo Gmail

O usuário pode escolher qualquer aplicativo para exportação, desde que este seja compatível com envio de arquivos. O envio via e-mail é apenas uma das possibilidades e foi utilizado somente para exemplificação.

As figuras 4.19 e 4.20 mostram o arquivo que foi exportado. Ele é composto por 2 pla-

nilhas sendo a primeira com as informações básicas da aquisição e a segunda com os dados obtidos das aquisições. É importante citar que a figura 4.20 mostra somente parte dos dados obtidos e que o número de linhas existentes corresponderá ao número de leituras realizadas.

	A	B	C	D
1	Number of items:	2		
2	Period:	500		
3	Readings:	240		
4	Started at:	11/9/2015 - 2:26:1		
5	Stopped at:	11/9/2015 - 2:28:1		
6				
7	Items:			
8	Number	Name	Custom Name	Description
9	00	SeekBar 1		Integer 0 to 100
10	01	SeekBar 2		Integer 0 to 100
11				
12	*Time is normalized by the acquisition period			
13				

Figura 4.19: Planilha com os detalhes da aquisição

	A	B	C
1	Time	SeekBar 1	SeekBar 2
2	0	79	21
3	1	78	21
4	2	77	21
5	3	78	20
6	4	79	19
7	5	79	19
8	6	80	19
9	7	80	19
10	8	81	20
11	9	82	19
12	10	83	19
13	11	84	19
14	12	84	20
15	13	83	21

Figura 4.20: Planilha com as leituras obtidas

4.3 Resultados práticos

Os resultados a seguir foram obtidos na prática, utilizando um celular com CPU quad-core de 2,5GHZ com 2GB de RAM para medição de valores durante a utilização do aplicativo.

4.3.1 Desempenho

Utilizando o próprio programa Android Studio é possível obter informações de uso de CPU e memória RAM do dispositivo móvel durante a execução do aplicativo. A figura 4.21 mostra dois gráficos (superior para uso de CPU e inferior para uso de memória RAM) com os recursos utilizados para uma aquisição com período de 200ms via Wi-Fi, salvando os dados no banco de dados.

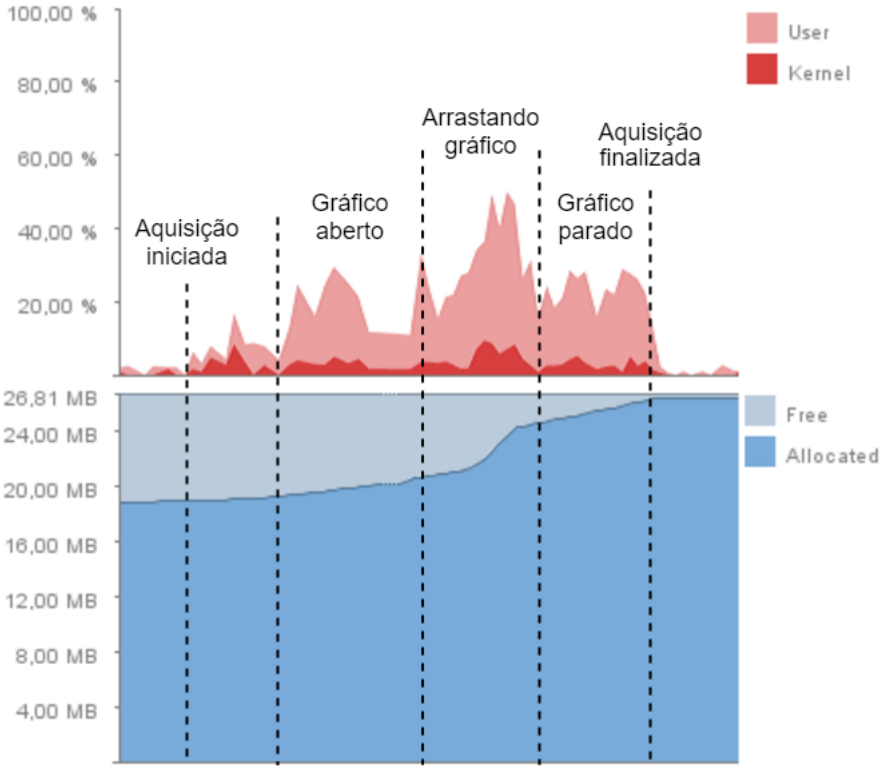

Figura 4.21: Recursos utilizados durante a execução do aplicativo

Com relação a utilização de CPU é possível notar que há um consumo elevado durante tarefas de alto processamento como visualização do gráfico e "arrastando" o gráfico. Mesmo assim, em tarefas que requeriam pouco processamento, o uso foi extremamente baixo.

Nota-se também que o sistema operacional reservou uma quantidade de memória RAM que foi sendo alocada conforme a execução do aplicativo e aquisição de dados. Um resultado inesperado é que esta memória não foi desalocada logo após a finalização do aplicativo e perdurou por mais de alguns minutos estável. Na documentação do sistema que pode ser consultada no site [23] contém a seguinte citação:

> Quando o usuário inicia um aplicativo, um processo é criado para ele, porém

> quando o usuário deixa o aplicativo, tal processo não é encerrado. O sistema mantém o processo armazenado em cache para que, se o usuário retornar para o aplicativo depois, o processo é reusado para uma troca mais rápida entre aplicativos.

Além disso, a mesma página contém a seguinte citação:

> Se o sistema ficar com pouca memória disponível ele pode finalizar processos armazenados em cache mais recentes, podendo também levar em consideração quais processos utilizam memória mais intensivamente.

Com isso é possível concluir que este resultado é absolutamente normal já que o sistema não ficou sobrecarregado e por isso não finalizou o processo armazenado em cache, mas irá se for necessário.

4.3.2 Uso de memória interna

A memória utilizada é composta em sua maior parte pelas aquisições armazenadas no dispositivo, já que são poucas as informações salvas pelo aplicativo (configurações utilizadas pelo usuário que são recarregadas quando uma atividade é inciada). A figura 4.22 mostra o uso de memória pelo aplicativo antes e depois de uma aquisição, respectivamente.

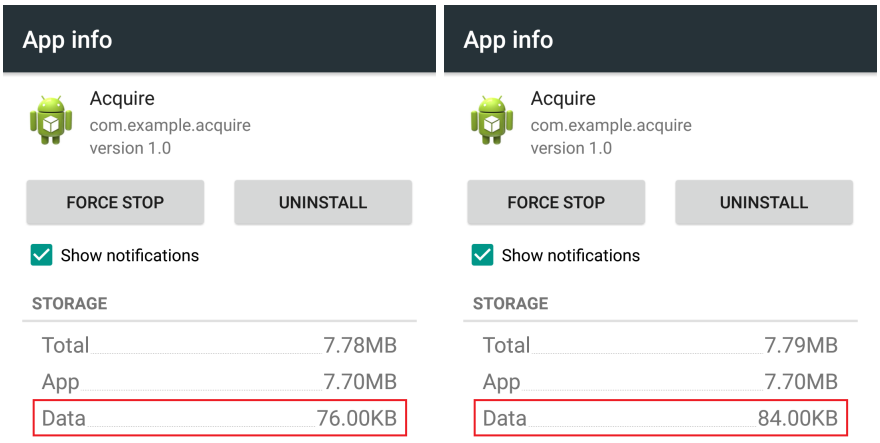

Figura 4.22: Uso de memória antes e depois da aquisição

A aquisição foi realizada para 2 itens com período de 500ms durante 1 minuto, resultando em 120 registros por item. Nota-se que o banco de dados utilizou cerca de 8KB para esta aquisição. Sendo assim, estima-se que 1MB de memória possibilita armazenar cerca de 30 mil leituras totais, podendo dividir-se entre vários itens. O fator limitante para registros é,

então, a memória disponível no dispositivo que hoje em dia atinge facilmente a ordem de gigabytes, deixando de ser uma barreira.

A planilha com os dados para exportação da aquisição descrita ocupou menos de 15KB na memória do dispositivo, não prejudicando o usuário com arquivos demasiadamente grandes.

Além disso, a memória necessária para a instalação do aplicativo no dispositivo é de 7,7MB, que é um tamanho considerado pequeno em função da disponibilidade de memória em celulares atuais.

4.3.3 Latência

A latência na troca de dados varia, principalmente, de acordo com o ambiente em que os dispositivos estão operando. Como descrito anteriormente, o aplicativo trata essas latências de forma a registrar somente a última mensagem obtida a cada período de aquisição.

O período de aquisição deve ser maior ou igual a latência da comunicação, visto que, se for menor, dados serão perdidos pois o sistema passará a receber mais de uma mensagem entre os períodos e não receberá em outros. Como o aplicativo foi programado para ignorar múltiplas mensagens durante o mesmo período (aceitando somente a mais recente) elas não afetarão as medições.

4.3.3.1 Comunicação via Wi-Fi

Durante uma comunicação via Wi-Fi, os principais fatores que interferem na latência são intensidade de tráfego na rede, interferências eletromagnéticas e de outras redes no local e distância entre as partes. A tabela 4.4 mostra os valores mínimo, máximo e médio de latência (em milissegundos) obtidos durante uma aquisição utilizando comunicação Wi-Fi em uma rede residencial.

Tabela 4.4: Latência da comunicação via Wi-Fi (em milissegundos)

Mínimo	Máximo	Médio
25	170	70,88

4.3.3.2 Comunicação via *bluetooth*

Pelo fato de não existir mais de um dispositivo conectado ao mesmo tempo, o único fator relevante para a latência são interferências eletromagnéticas e a distância entre os disposi-

tivos. A tabela 4.5 mostra os valores mínimo, máximo e médio de latência (em milissegundos) obtidos durante uma aquisição utilizando comunicação *bluetooth*.

Tabela 4.5: Latência da comunicação via *bluetooth* (em milissegundos)

Mínimo	Máximo	Médio
41	134	91,12

É possível notar que a latência via Wi-Fi varia em um intervalo maior que via *bluetooth*, porém possui um valor médio menor.

Capítulo 5

Conclusões

O Android SDK comprovou-se uma ótima ferramenta para desenvolvimento de aplicativos para Android, contendo uma vasta documentação, necessitando o conhecimento da linguagem de programação Java. O aplicativo desenvolvido em combinação com o protocolo elaborado possibilitou a aquisição de leituras de forma simples, permitindo ao usuário obter as informações dos sensores, abstraindo as fontes.

A utilização de um banco de dados em conjunto com o aplicativo desenvolvido permite o armazenamento das aquisições realizadas, garantindo que estes dados possam ser acessados posteriormente. Além disso, a possibilidade de exportação abre portas para a utilização das aquisições em outras plataformas, como computadores, descartando a necessidade de processamento imediato das leituras.

A criação de unidades de sensoreamento é simples do ponto de vista da compatibilidade com o protocolo, necessitando que o desenvolvedor apenas programe as mensagens existentes. Os tipos de sensores e dados medidos podem variar sem carecer da alteração do protocolo, atingindo inúmeras opções e aplicações. Além disso, a biblioteca desenvolvida facilita este processo por realizar certos procedimentos automaticamente.

A inviabilidade da biblioteca no desenvolvimento de uma unidade quando o microcontrolador empregado possui pouca memória de dados disponível obriga o desenvolvedor criar os próprios procedimentos podendo, porém, utilizar a biblioteca como diretriz neste processo.

O módulo de comunicação Wi-Fi (ESP8266) proporcionou a automatização de uma unidade com relação ao gerenciamento da rede em que está conectado ou da rede própria quando utilizando o NodeMCU. Com a linguagem de programação Lua e a criação de *scripts*, é possível programar o modo de operação do módulo, gerando mais possibilidades para o desenvolvimento. Em contrapartida, este *firmware* consome muita memória, podendo deixar a desejar

em casos onde pretende-se utilizar códigos mais extensos.

A possibilidade de programar diretamente o *firmware* do módulo Wi-Fi pela IDE Arduino torna-se uma opção atraente quando deseja-se mais funcionalidades no modo de operação do módulo. Por ser mais eficiente deve ser levada em conta para projetos futuros, caso a memória se torne escassa quando utilizando o NodeMCU.

O projeto está diretamente relacionado com o conceito de *Internet of things* (internet das coisas), que consiste na obtenção e exportação de dados a partir de unidades de sensoreamento remotas. Uma vez que o aplicativo e as unidades foram planejadas visando o mesmo propósito, o protocolo de comunicação e a abstração dos sensores foram fatores fundamentais, servindo como contribuição para este processo.

Trabalhos futuros

Listam-se como trabalhos futuros o desenvolvimento de um aplicativo para celular ou programa de computador capaz de realizar aquisições de várias unidades simultaneamente, armazenamento dos dados em um banco de dados online e possibilidade de atuar sobre as unidades ao invés de somente realizar leituras.

A necessidade de realizar a aquisição de várias unidades simultaneamente é desejável em casos onde vários dados são obtidos de forma dispersa, por várias unidades diferentes.

Armazenamento de dados na "nuvem" possibilita o acesso mais rápido das informações de vários pontos e por sistemas distantes fisicamente.

A opção de atuar possibilitará ao usuário, além de realizar leituras dos sensores, atuar sobre mecanismos presentes na unidade como lâmpadas, motores, etc. Essa implementação necessita de reformulação do protocolo desenvolvido que prevê somente requisições de leituras dos sensores.

Acrescenta-se ainda a possibilidade de programação de uma unidade de sensoreamento para realizar uma aquisição de forma *offline*, armazenando as informações em uma memória interna não volátil. Esta opção é atraente em casos que deseja-se uma aquisição por um longo tempo, dispensando a necessidade de um dispositivo móvel conectado durante este período.

Referências Bibliográficas

[1] Erina Ferro and Francesco Potortì. Bluetooth and Wi-Fi Wireless Protocols: A Survey and a Comparison. 2005.

[2] Philip M. Miller. *TCP/IP The Ultimate Protocol Guide*. Brown Walker Press, 2009.

[3] Doug Serfass, Kenji Yoshigoe. Wireless Sensor Networks Using Android Virtual Devices and Near Field Communication Peer-To-Peer Emulation. 2012.

[4] Apresentando o Módulo ESP8266. http://andrecurvello.com.br/apresentando-o-modulo-esp8266/, Acesso em: 27 de outubro de 2015.

[5] NodeMCU. http://nodemcu.com/, Acesso em: 27 de outubro de 2015.

[6] ESP8266 - Easiest way to program so far (Using Arduino IDE). http://www.whatimade.today/esp8266-easiest-way-to-program-so-far/, Acesso em: 27 de outubro de 2015.

[7] ESP8266 Serial WIFI Module. http://wiki.iteadstudio.com/ESP8266_Serial_WIFI_Module, Acesso em: 27 de outubro de 2015.

[8] About SQLite. https://www.sqlite.org/about.html, Acesso em: 27 de outubro de 2015.

[9] Fattoh AI-Qershi, Muhammad AI-Qurishi, Sk Md Mizanur Rahman, and Atif AI-Amri. Android vs. iOS: The Security Battle. 2014.

[10] Android - Introduction. http://developer.android.com/guide/index.html, Acesso em: 27 de outubro de 2015.

[11] Android - App Manifest. http://developer.android.com/guide/topics/manifest/manifest-intro.html, Acesso em: 27 de outubro de 2015.

[12] Android - Activities. http://developer.android.com/guide/components/activities.html, Acesso em: 27 de outubro de 2015.

[13] Android - User Interface. http://developer.android.com/guide/topics/ui/index.html, Acesso em: 27 de outubro de 2015.

[14] Android - Services. http://developer.android.com/guide/components/services.html, Acesso em: 27 de outubro de 2015.

[15] Android - Processes and Threads. http://developer.android.com/guide/components/processes-and-threads.html, Acesso em: 27 de outubro de 2015.

[16] Android - Class ServerSocket - Accept(). http://developer.android.com/reference/java/net/ServerSocket.html#accept(), Acesso em: 27 de outubro de 2015.

[17] Android - Class BroadcastReceiver. http://developer.android.com/reference/android/content/BroadcastReceiver.html, Acesso em: 27 de outubro de 2015.

[18] Android - Bluetooth. http://developer.android.com/guide/topics/connectivity/bluetooth.html, Acesso em: 27 de outubro de 2015.

[19] Android - Wi-Fi Peer-to-Peer. http://developer.android.com/guide/topics/connectivity/wifip2p.html, Acesso em: 27 de outubro de 2015.

[20] Jingyun Feng, Zhi Liu, Yusheng Ji. Wireless Channel Loss Analysis - A Case Study Using WiFi-Direct. 2014.

[21] *AT89LP4052 - Datasheet*, Acesso em: 27 de outubro de 2015.

[22] Yusuf Abdullahi Badamasi. The Working Principle Of An Arduino. 2014.

[23] Managing Your App's Memory. http://developer.android.com/intl/pt-br/training/articles/memory.html, Acesso em: 27 de outubro de 2015.

Apêndice A

Biblioteca para auxílio no desenvolvimento de uma unidade de sensoreamento

A biblioteca desenvolvida em C visa auxiliar o desenvolvedor na criação de uma unidade de sensoreamento. Ela realiza todos os procedimentos necessários para o usuário sendo que este só necessita realizar pequenas configurações.

Para utilizar a biblioteca, adicione-a ao projeto:

```
#include "SensorLibrary.h"
```

Por tratar-se de uma biblioteca para uma unidade genérica, não há como padronizar um método de envio de dados. Sendo assim, é necessário que o usuário informe à biblioteca uma função para realizar este procedimento. Isto é feito no início do programa através da função **setupLib** que tem como parâmetro o endereço de uma função a ser utilizada para envio de uma mensagem através da comunicação.

Como exemplo, se o programa principal tem a seguinte função:

```
void sendString(char *str){
   //Send the message some way
   ...
}
```

No início do programa deve-se fazer, então:

```
setupLib(&sendString);
```

Deste modo, a biblioteca está configurada para utilizar a função **sendString** para enviar uma mensagem. É importante citar que esta função deve ser sempre no formato "`void function(char *)`".

Em seguida, o usuário deve configurar os sensores presentes, utilizando um vetor de estruturas do tipo 'sensor'. Declarada na biblioteca, a estrutura possui 3 campos (*name*, *description* e *value*) e é utilizada uma posição no vetor para cada sensor. Para um acelerômetro de 3 eixos, por exemplo, utiliza-se um vetor de 3 posições.

```
struct sensor accelerometer[3]; //Using 3-axis accelerometer
```

O usuário pode alterar no arquivo **SensorLibrary.h** da biblioteca o tamanho de cada uma das *strings* caso seja necessário um tamanho maior ou economizar memória.

```
//Sizes of the strings, change if necessary
#define sizeName 30
#define sizeDescription 100
#define sizeValue 5
```

Para configurar os sensores, em cada posição do vetor, deve-se escrever as informações correspondentes (nome, descrição e valor). A biblioteca possui duas funções simples que podem ser utilizadas para tais processos (**printString** e **printInt**). É importante deixar claro que os tamanhos a serem utilizados devem ser maiores que as *strings* em pelo menos 1 unidade, pois estas são sempre finalizadas com caractere nulo.

```
//X axis
printString(accelerometer[0].name, "Accelerometer X-Axis");
printString(accelerometer[0].description, "-100 to 100, 50 equals to 1g.");
printInt(accelerometer[0].value, readX());
//Y axis
printString(accelerometer[1].name, "Accelerometer Y-Axis");
printString(accelerometer[1].description, "-100 to 100, 50 equals to 1g.");
printInt(accelerometer[1].value, readY());
//Z axis
printString(accelerometer[2].name, "Accelerometer Z-Axis");
printString(accelerometer[2].description, "-100 to 100, 50 equals to 1g.");
printInt(accelerometer[2].value, readZ());
```

A unidade deve sempre esperar por uma mensagem completa, isto é, iniciada e finalizada pelo caractere '**$**'. O apêndice B contém uma função que pode ser utilizada para este propósito. A cada caractere recebido, deve-se executar esta função que irá verificar automaticamente estes requisitos, assim como tratar outros problemas que podem ocorrer.

Quando uma mensagem está completa, ela deve ser decodificada e respondida. Sendo assim, os valores devem ser atualizados no vetor dos sensores e a função **decodeMessage**,

definida na biblioteca, deve ser executada. Esta função recebe como parâmetros a mensagem recebida, o vetor de sensores e a quantidade de sensores.

```
//Update the values
printInt(accelerometer[0].value, readX());
printInt(accelerometer[1].value, readY());
printInt(accelerometer[2].value, readZ());

//Decode the message
decodeMessage(messageReceived, accelerometer, 3);
```

É extremamente importante que as mensagens enviadas como parâmetro para esta função sigam rigorosamente o protocolo especificado neste trabalho. Por isso, não devem haver caracteres a mais antes do primeiro **'$'** e após o último **'$'**, com exceção de caracteres nulos neste último caso.

Apêndice B

Função para auxílio na recepção das mensagens

```
//Defining the length limit of the message
#define limit 16

//Global Variables
char messageReceived[limit], *ptr;
//Status of the reception
char status = 0;

//Check the received character
void charReceived(char received){
   int i;

   //If status = 0 -> it's waiting for the first '$'
   if(status == 0){
      //Received '$'
      if(received == '$'){
         status = 1;
         //Clear the string
         for(i = 1; i < limit; i++)
            messageReceived[i] = 0;
         messageReceived[0] = '$';
         //Point to the second character
         ptr = messageReceived + 1;
      }
   }
   //Is receiving the message
   else{
      *ptr = received;
      //Status = 1 -> checks the second character
      if(status == 1){
         //If it's a '#'
         if(*ptr == '#'){
            //The message is valid (second character is a '#')
            status = 2;
            ptr++;
```

```
            }
            else{
               //If it's a '$'
               if(*ptr == '$'){
                  //Restart the verification as the message is starting again
                  messageReceived[0] = '$';
                  messageReceived[1] = 0;
               }
               //Otherwise, discard everything and start again
               else{
                  status = 0;
               }
            }
         }
         //If is receiving the message (first character already is a '$' and the
         //second is '#')
         else{
            //If the message is too big (might cause problem), discard everything
            //and start again
            if(ptr == messageReceived + limit){
               status = 0;
               return;
            }
            //When the message is complete
            if(*ptr == '$'){
               //Update values in the struct before decoding the message
               ...
               //Decode the message
               decodeMessage(messageReceived, (struct sensor *) , short int);
               //Start again
               status = 0;
            }
            ptr++;
         }
      }
   }
```

Apêndice C

Códigos e bibliotecas

Códigos e bibliotecas desenvolvidos podem ser obtidos em:
`http://143.107.235.37/gabriel/`