UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

GUILHERME AUGUSTO DE MATOS SILVA

DESENVOLVIMENTO DE INTERFACE COM USUÁRIO PARA *BIOFEEDBACK* DE UMA MULETA INSTRUMENTADA

São Carlos
2017

GUILHERME AUGUSTO DE MATOS SILVA

DESENVOLVIMENTO DE INTERFACE COM O USUÁRIO PARA *BIOFEEDBACK* DE UMA MULETA INSTRUMENTADA

Monografia apresentada ao Curso de Engenharia Elétrica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Alberto Cliquet Júnior

São Carlos

2017

Matos Silva, Guilherme Augusto de
M586d Desenvolvimento de interface com usuário para biofeedback de uma muleta instrumentada / Guilherme Augusto de Matos Silva; orientador Alberto Cliquet Júnior. São Carlos, 2017.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2017.

1. Android. 2. Bluetooth. 3. Muleta. 4. Biofeedback. 5. Arduino. I. Título.

FOLHA DE APROVAÇÃO

Nome: Guilherme Augusto de Matos Silva

Título: "Desenvolvimento de interface com usuário para biofeedback de uma muleta instrumentalizada"

Trabalho de Conclusão de Curso defendido e aprovado em 24 / 11 / 2017,

com NOTA 9,0 (nove, zero), pela Comissão Julgadora:

Prof. Titular Alberto Cliquet Júnior - Orientador - SEL/EESC/USP

Dr. Renato Varoto - Pós-doutorado/ UNICAMP

Mestre Renata Manzano Maria - UNICAMP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

AGRADECIMENTOS

Agradeço aos meus pais, Paulo e Zélia, por todo esforço, sacrifício e paciência durante todo o período de graduação.

Agradeço ao meu irmão Luis Otávio, pelo apoio e orientação durante o desenvolvimento deste projeto.

Agradeço ao meu primo Lucas, pelo apoio e momentos de descontração.

Agradeço ao professor Dr. Alberto Cliquet e ao Dr. Renato Varoto pelo apoio, orientação e por tudo que aprendi durante o desenvolvimento deste projeto.

RESUMO

SILVA, G. A. M. **Desenvolvimento de interface com o usuário para *biofeedback* de uma muleta instrumentada**. 2017. Monografia (Trabalho de Conclusão de Engenharia Elétrica com Ênfase em Eletrônica) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2017.

Este projeto visou o desenvolvimento de um aplicativo *Android* que funcionasse como interface com o usuário para o envio de dados pelo usuário e recebimento dos dados obtidos por uma muleta instrumentada e seu circuito de amplificação. Os dados são recebidos por meio de comunicação *Bluetooth* estabelecida entre o aplicativo e um módulo conectado ao *Arduino*, que recebe os dados do circuito. O aplicativo, além de exibir os dados, alerta o usuário, por meio de uma notificação com alerta vibratório e sonoro, quando a força exercida sobre a muleta ultrapassar o limite de 20% de seu peso. Estes dados são armazenados em um arquivo, sempre que o usuário solicitar, ou quando a conexão for desfeita. Além disto, o usuário pode ler os dados do arquivo, sempre que solicitar. Os resultados mostraram que a conexão *Bluetooth* apresentou estabilidade, sem perdas significativas de dados. As medidas feitas pelo sistema eletrônico desenvolvido se distanciaram de no máximo de 5% com relação ao dinamômetro utilizado para a medição de força sobre a muleta, o que permite concluir que o trabalho atingiu os objetivos propostos.

Palavras-chave: *Android*. *Bluetooth*. Muleta. *Biofeedback*. *Arduino*

ABSTRACT

MATOS, G. A. M. **Development of user interface for biofeedback of an instrumented crutch**. 2017. 198 f. Monograph – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2017.

This project aimed the development of an Android application that worked as an user interface for sending data by the user and receiving data obtained by an instrumented crutch and it's amplification circuit. Data are received through Bluetooth connection established between the application and a module connected to Arduino, what receives circuit data. The application, besides displaying data, warns the user, through a notification with vibrating and audible alert, when the force exerted on the crutch exceeds the limit of 20% of user's weight. This data are stored in a file, whener the user requests, or when there is a disconnection. In addtion, the user can read the data from the file whenever resquested. The results showed that the Bluetooth connection was stable, with no significant loss of data.The measurements made by the developed electronic system distanced themselves from a maximum of 5% in relation to the dynamometer used for the measurement of force on the crutch, which allows to conclude that the work reached the proposed objectives.

Keywords: Android. Bluetooth. Crutch. Biofeedback. Arduino.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	–	*Application Programming Interface*
IDE	–	*Integrated Development Environment*
LED	–	*Light Emmiting Diode*
MAC	–	*Media Access Control*
RFCOMM	–	*Radio Frequency Communication*
SG	–	*Strain-gauge*
UART	–	*Universal Asynchronus Receiver/Transmitter*
UUID	–	*Universally Unique Identifier*

SUMÁRIO

1 INTRODUÇÃO

Ao sofrer uma fratura, ou quando é necessária a realização de cirurgia nos membros inferiores, o paciente passa por um processo de reabilitação, que envolve o controle da força aplicada sobre a perna fraturada. Uma forma de se fazer esse controle é através da utilização de auxiliadores de locomoção.

Auxiliadores de locomoção são prescritos para compensar problemas clínicos e são utilizados por diversas razões, como, por exemplo, para diminuir o excesso de peso nas extremidades inferiores, para corrigir desequilíbrio, para reduzir a fadiga ou para aliviar a dor resultante da carga em estruturas danificadas. Também auxiliam na produção de força, usando a parte superior para compensar a parte inferior do paciente [1]. Um dos principais auxiliadores de locomoção é a muleta do tipo *Lofstrand*.

Para que a força aplicada sobre os membros inferiores pudesse ser medida, foi desenvolvida uma muleta instrumentada, segundo [2], e um sistema de *biofeedback*, segundo [3].

De acordo com [3], a perna fraturada não deve estar submetida a uma carga superior a 20% do peso do usuário.

Uma forma de alertar o usuário que o limite de força foi excedido é por meio do sistema de *biofeedback,* utilizando-se uma comunicação sem fio e o dispositivo móvel do próprio usuário.

1.1 Dispositivos móveis

Como o uso de dispositivos móveis, *smartphones* e *tablets* principalmente, já é parte integrante da vida cotidiana de grande parte da população, o desenvolvimento de soluções implementáveis para estes dispositivos se torna cada vez mais acessível, devido à crescente disponibilidade de tecnologias.

O sistema operacional mais popular entre estes dispositivos é o *Android*, que é baseado em software livre e permite de forma fácil, acesso aos recursos dos dispositivos, por meio de aplicações criadas por um usuário.

A área da eletrônica embarcada também está em constante evolução, sendo maior o acesso a soluções de *hardware* que permitem fazer uso de tecnologias mais recentes de forma

mais fácil. Devido ao fato destes dispositivos terem se tornado tão comuns, torna-se interessante se beneficiar deles para desenvolver aplicações para usos clínicos, capazes de receberem dados de equipamentos eletrônicos, utilizados na medicina, por meio de conexões sem fio, facilitando o aviso ao usuário quando o dado recebido representar algum risco a ele, e também o armazenamento desses dados para análises posteriores.

1.2 Objetivo

O objetivo deste projeto é modificar o sistema de *biofeedback*, desenvolvido de acordo com [3], para a muleta, ou seja, um sistema que seja capaz de receber os dados da muleta e avisar ao usuário que a força sobre a sua perna foi excedida, além de armazenar em um arquivo os dados obtidos. Este sistema é baseado em uma aplicação que se comunica com um microcontrolador por meio da tecnologia *Bluetooth*, e também funciona como a interface com o usuário.

No sistema desenvolvido de acordo com [3], a interface com o usuário era feita por meio de um teclado numérico e um *display* de sete segmentos, e o aviso, era feito por um sinal sonoro.

1.3 Organização da Monografia

Neste primeiro capítulo foi discutido a motivação do desenvolvimento da muleta e do *biofeedback* por meio de um aplicativo para *Android*, e o objetivo do projeto.

No segundo capítulo é discutido a fundamentação teórica da construção da muleta, do circuito de amplificação e da utilização da eletrônica embarcada para que seja feita a comunicação com o celular, além de ser discutido conceitos utilizados para o desenvolvimento do aplicativo e da conexão *Bluetooth*.

O terceiro capítulo explica a utilização dos componentes do circuito e a construção dos códigos do programa do *Arduino* e do aplicativo.

No quarto capítulo são apresentados os resultados obtidos, comparando-se os dados do circuito com a medição feita no dinamômetro.

No quinto capítulo é feita a conclusão, validando os objetivos do projeto.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Muleta

A Figura 1 mostra o auxiliador de locomoção utilizado neste projeto, a muleta do tipo *Lofstrand*.

Figura 1 - Muleta do tipo *Lofstrand*

Fonte – Adaptado de www.dfarmacia.com/farma/ctl_servlet?_f=37&id=13155634. Acesso em 30 Nov. 2017

Como a carga aplicada na muleta está relacionada com a condição em que os membros inferiores se encontram é válido desenvolver uma muleta que possa medir e enviar os dados da força aplicada. Como a grandeza a ser medida é a força, diversos tipos de transdutores podem ser utilizados, como os *strain-gauges* [2].

A muleta possui, na configuração em ponte de *Wheatstone*, quatro *strain-gauges*. Sua utilização se deve ao fato da possibilidade de serem colocados diretamente sobre o material que recebe o esforço, além de serem insensíveis à temperatura, quando ligados em ponte completa, segundo [2].

“A configuração em ponte será tal que somente a força axial na muleta seja adquirida, excluindo outros tipos de esforços” [2].

O sinal de saída da ponte é baixo, da ordem de milivolts, necessitando assim de um circuito amplificador, o que também auxilia na redução de ruídos. O circuito será discutido no *Capítulo 3*.

2.2 *Arduino*

De acordo com [4], o *Arduino* é uma plataforma de prototipagem eletrônica aberta (*open-source*), baseada em *hardware* e *software* de fácil uso. As placas *Arduino* são capazes de receber na sua entrada sinais analógicos e digitais, e transmiti-los para outros dispositivos. O microcontrolador presente na placa pode ser programado utilizando a linguagem *Arduino*, baseada em *Wiring* e pela sua IDE (*Integrated Development Environment*), baseada no ambiente *Processing*.

Um dos meios de comunicação disponível é via serial pela UART (*Universal Asynchronus Receiver/Transmitter*), que com o auxílio de uma biblioteca presente na linguagem de programação do *Arduino*, pode também ser utilizada para estabelecer comunicação com o módulo *Bluetooth*.

A Figura 2 mostra o modelo de placa *Arduino* Uno, que possui um microcontrolador ATmega328P, fabricado pela *Atmel*.

Figura 2 - *Arduino* Uno

Fonte – Adaptado de https://makezine.com/2011/12/01/just-arrived-in-the-maker-shed-the-new-arduino-uno-revision-3/. Acesso em 30 Nov. 2017.

2.3 Módulo *Bluetooth*

O módulo utilizado para a comunicação entre o *Arduino* e o aplicativo foi o HC-05. De acordo com [5]

> Este módulo oferece uma forma fácil de comunicação com um projeto *Arduino*, suportando tanto o modo mestre como escravo. Possui regulador de tensão, podendo ser alimentado com tensões de 3,3V a 5V, bem como um LED (*Light Emitting Diode*) que indica se o módulo está pareado com outro dispositivo.

Suas especificações são:

- Senha padrão de pareamento: 1234;
- Protocolo Bluetooth: v2.0+EDR;
- Firmware: Linvor 1.8;
- Frequência: 2,4GHz Banda ISM;
- Modulação: GFSK;
- Emissão de energia: <=4dBm, Classe 2;
- Sensibilidade: <=84dBm com 0,1% BER;
- Velocidade Assíncrono: 2,1Mbps(Max)/160Kbps;
- Velocidade Síncrono: 1Mbps/1Mbps;
- Tensão: 3,3V (2,7V-4,2V);
- Temperatura: -40 ~ +105ºC;
- Alcance: 10m.

Para se conectar ao *Arduino* é necessário que os pinos de alimentação do módulo sejam conectados aos pinos disponíveis para alimentação do *Arduino*. O pino de transmissão T_x, seja conectado diretamente a um dos pinos digitais do *Arduino* e o pino de recepção R_x, seja conectado a um circuito divisor de tensão e a saída deste circuito, conectada a um dos pinos digitais do *Arduino*. O divisor de tensão é necessário no pino de recepção, pois o pino do módulo aceita tensões de no máximo 3,3V, enquanto que o pino do *Arduino* pode enviar sinais com tensões de 5V.

A Figura 3 mostra o módulo *Bluetooth*.

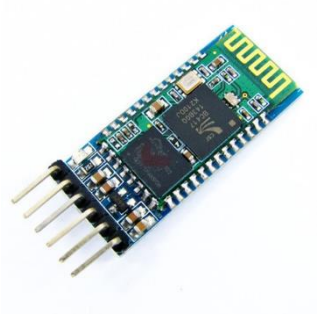

Figura 3 - Módulo *Bluetooth* HC-05

Fonte – Adaptado de https://www.filipeflop.com/produto/modulo-bluetooth-rs232-hc-05/. Acesso em 30 Nov. 2017.

2.4 *Android*

Android é um sistema operacional de código aberto, baseado em *Linux*, atualmente desenvolvido pela empresa *Google Inc*, que utiliza a linguagem *Java*, orientada a objetos.

De acordo com [6], "o *Android* chegou a marca de dois bilhões de usuários ativos por mês, sendo o sistema móvel mais utilizado no mundo", o que justifica sua escolha para este projeto.

Uma ferramenta para o desenvolvimento de aplicativos para *Android* é o *Android Studio*, utilizado para o desenvolvimento do aplicativo neste projeto, que segundo [7], "oferece ferramentas mais rápidas para a criação de aplicativos e possui recursos como edição de código nível global, depuração, ferramentas de desempenho e sistema flexível de compilação".

Alguns conceitos importantes para a criação do aplicativo para este projeto, são o de atividade, serviço e serviço vinculado.

De acordo com [8]

> A Atividade, ou *Activity*, é um componente de aplicativo que fornece uma tela com a qual os usuários podem interagir para fazer algo. Normalmente, uma atividade em um aplicativo é especificada como "principal", que é a apresentada ao usuário ao iniciar o aplicativo pela primeira vez. Cada atividade pode, então, iniciar outra atividade para executar diferentes ações.

Alguns métodos precisam ser implementados para a execução da atividade:

onCreate: “o sistema o chama ao criar a atividade. Na implementação, é preciso inicializar os componentes essenciais da atividade. E, fundamentalmente, é quando se deve definir o layout da interface do usuário da atividade” [8].

onStart: “chamado logo antes de a atividade se tornar visível ao usuário” [8].

onResume: “chamado logo antes de a atividade iniciar a interação com o usuário” [8].

onStop: “chamado quando a atividade não está mais visível ao usuário. Isso pode acontecer porque ela está sendo destruída ou porque outra atividade (uma existente ou uma nova) foi retomada e está cobrindo-a” [8].

onDestroy: “chamado antes de a atividade ser destruída” [8].

De acordo com [9]

> Um Serviço, ou Service, é um componente do aplicativo que pode realizar operações longas e não fornece uma interface do usuário. Outro componente do aplicativo pode iniciar o serviço e ele continuará em execução em segundo plano mesmo que o usuário alterne para outro aplicativo.

Um método que necessita ser criado é *onStarCommand*, que segundo [9]

> O sistema chama esse método quando outro componente, como uma atividade, solicita que o serviço seja iniciado. Quando esse método é executado, o serviço é iniciado e pode ser executado em segundo plano indefinidamente.

Segundo [10]

> Um serviço vinculado é o servidor em uma interface cliente-servidor. Um serviço vinculado permite que componentes (como atividade) sejam vinculados ao serviço, enviem solicitações, recebam respostas e até estabeleçam comunicação entre processos (IPC). Um serviço vinculado geralmente existe somente enquanto serve a outro

> componente do aplicativo e não é executado em segundo plano indefinidamente.

Um método necessário para este caso é o *onBind*. “O sistema chama esse método quando outro componente quer se vincular ao serviço” [10].

2.5 Conexão *Bluetooth*

De acordo com [11]

> Para que uma conexão seja estabelecida, é necessário implementar o lado do servidor, que abre um soquete de servidor e ouça conexões, e o lado do cliente, que inicia a conexão. O servidor e o cliente serão considerados conectados entre si quando cada um deles tiver um soquete conectado no mesmo canal RFCOMM (*Radio Frequency Communication*). Nesse momento, cada dispositivo pode obter *streams* de entrada e saída e a transferência de dados pode começar.
>
> Quando um dispositivo é pareado, as informações básicas sobre esse dispositivo (como nome, classe e endereço MAC (*Media Access Control*) do dispositivo) são salvas e podem ser lidas usando as *Bluetooth* API’s (*Application Programming Interface*). Com o endereço MAC conhecido de um dispositivo remoto, é possível iniciar uma conexão com ele a qualquer momento sem executar a descoberta (desde que o dispositivo esteja dentro do alcance). Estar pareado significa que os dois dispositivos estão cientes da existência um do outro, têm um link-chave compartilhado que pode ser usado para autenticação e podem estabelecer uma conexão criptografada entre si. Estar conectado significa que os dispositivos compartilham no momento um canal RFCOMM e podem transmitir dados entre si. A *Android Bluetooth* API atual exige que os dispositivos estejam pareados antes de estabelecer uma conexão RFCOMM. (O pareamento é executado automaticamente na inicialização de uma conexão criptografada com as API’s do *Bluetooth*).

3 DESCRIÇÃO DO PROJETO E DESENVOLVIMENTO

3.1 Célula de carga

Para a medição da carga, foi construída, na haste da muleta, uma célula de carga com quatro *strain-gauges* (SG), transformando-a em um transdutor de força. Os SGs foram colados sobre o material da haste e ligados na configuração de ponte de *Wheatstone*, segundo [2].

3.2 Circuito

O sinal de saída da ponte de *Wheatstone* é muito baixo e sujeito a ruídos, sendo assim, utilizou-se um circuito amplificador com elevada rejeição ao ruído, constituído de um amplificador de instrumentação e um amplificador operacional. Além disto, a alimentação da ponte é feita utilizando-se um circuito de referência precisa e são utilizados dois reguladores de tensão para a estabilização da alimentação dos amplificadores. O circuito possui um filtro passa baixas, ajustado para a frequência de corte de 5 Hz, nas entradas do amplificador de instrumentação, para atenuar ruídos sobre o sistema.

O circuito desenvolvido de acordo com [3] possuía o amplificador de instrumentação, o amplificador na configuração inversora na sua saída, o filtro na sua entrada, e o circuito de referência precisa para alimentação da ponte. Neste projeto foram adicionados os reguladores de tensão.

O diagrama do circuito pode ser observado na Figura 4.

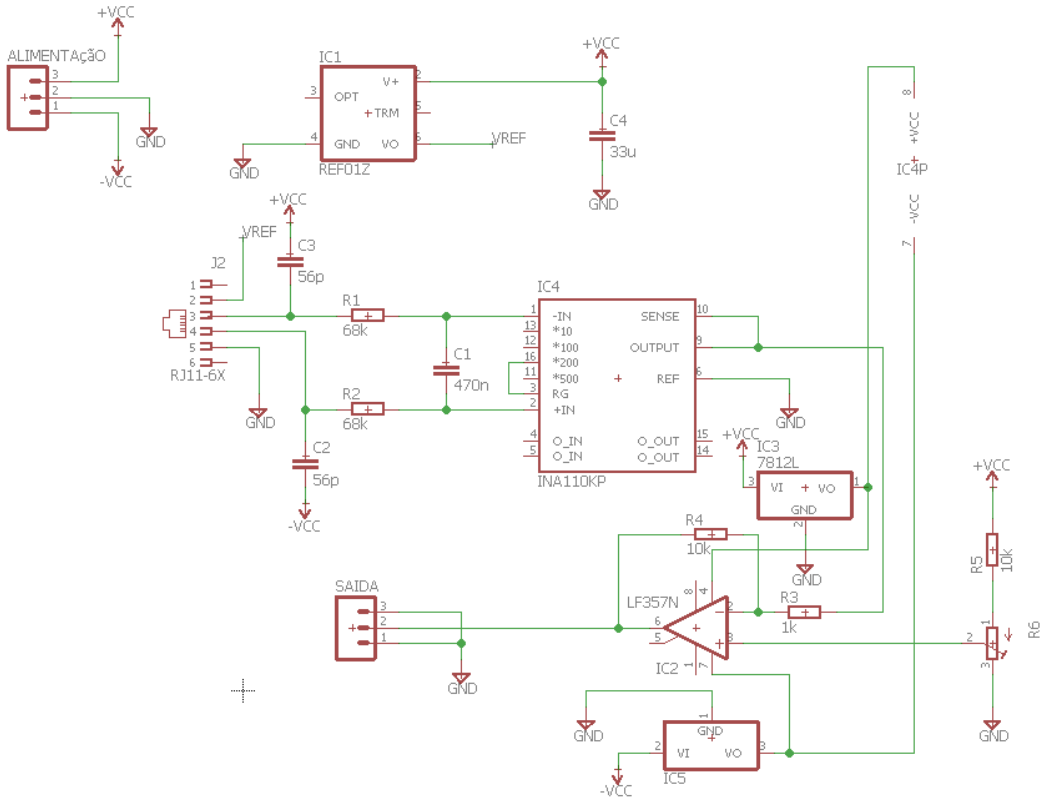

Figura 4 - Circuito utilizado para amplificação do sinal vindo dos SGs

Fonte – Autoria própria

Para associar-se corretamente a força sobre a muleta e a saída do circuito, utilizou-se um dinamômetro para medir a carga aplicada à haste e um amperímetro para a medição da tensão na saída do circuito. Utilizando-se pesos padrões, para que a força sobre a haste se mantivesse constante, foram feitos três conjuntos de medidas independentes, presentes no Apêndice A – Conjuntos de valores para a obtenção da curva. Com o auxílio do programa MATLAB, foi feita uma média aritmética entre os três valores de tensão e uma média aritmética dos três valores de força.

Os valores das médias estão na Tabela 1.

Tabela 1 - Valores médios calculados para a obtenção da curva

Força (N)	**Tensão (V)**
9,0000	0,5767
16,3333	0,6220
24,0000	0,6550
28,3333	0,6817

Força (N)	Tensão (V)
33,0000	0,7087
38,3333	0,7363
43,0000	0,7637
48,0000	0,7897
52,6667	0,8150
57,6667	0,8430
62,3333	0,8703
67,0000	0,8960
72,0000	0,9213
77,0000	0,9483
82,0000	0,9747
86,6667	1,0007
91,6667	1,0283
96,6667	1,0537
101,3333	1,0797
106,0000	1,1057
110,6667	1,1313
115,6667	1,1573
120,6667	1,1867
125,3333	1,2100
130,6667	1,2377
135,0000	1,2633
140,0000	1,2860
152,3333	1,3447
166,6667	1,4247
181,0000	1,5023
195,6667	1,5797
207,0000	1,6580
223,6667	1,7353
238,0000	1,8123
252,0000	1,8893
266,0000	1,9647

Força (N)	Tensão (V)
278,6667	2,0403
294,0000	2,1157
307,6667	2,1897
321,0000	2,2630
334,6667	2,3353
347,6667	2,4070
361,3333	2,4800
374,6667	2,5500
387,6667	2,6213
398,6667	2,6823
411,0000	2,7513
424,0000	2,8180

A partir destes dados, foi obtida a reta, descrita pela equação 1, que melhor representava os pontos medidos.

$$F = 185{,}4336V - 98{,}5793 \quad (1)$$

Sendo *V* a tensão medida na saída do circuito e *F* a força medida pelo dinamômetro.

Da equação 1 é possível obter a inclinação da reta, 185,4336, e via *software* é obtido o valor de tensão de correção, correspondente ao ponto onde a força vale zero. Este valor de correção posiciona o início da reta para o ponto onde a força vale zero.

A Figura 5 mostra a reta obtida.

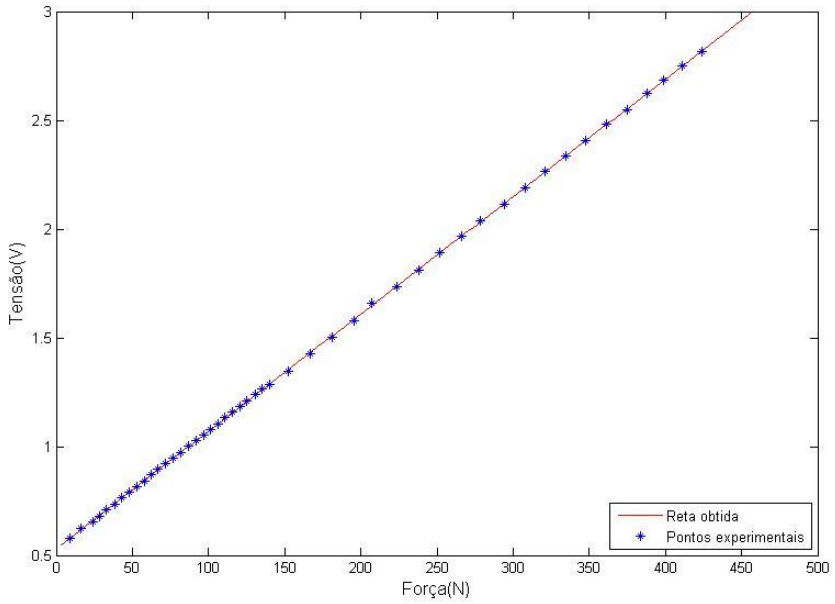

Figura 5 - Reta obtida a partir dos dados da Tabela 1

Fonte – Autoria própria

3.3 *Arduino*

O sinal de saída do circuito é conectado a uma das entradas analógicas do *Arduino* e o módulo *Bluetooth* é conectado a dois pinos digitais do *Arduino*, um para receber e outro para enviar os dados para o aplicativo.

O código do *Arduino* contém uma biblioteca para o envio e recebimento dos dados via *Bluetooth*, muito semelhante a uma conexão serial UART, de forma que as funções utilizadas são as mesmas. No início do programa são inicializadas as variáveis que serão utilizadas, a massa enviada pelo aplicativo, a tensão recebida pelo *Arduino*, os valores de força, correção e peso, calculados pelo programa, e as variáveis utilizadas para a contagem do tempo.

No código são definidos os pinos do *Arduino* que serão utilizados pelo módulo *Bluetooth* para o envio e recebimento dos dados. Utilizando-se a biblioteca da conexão serial, é definido o nome da conexão, que será utilizado quando as funções da conexão serial foram utilizadas.

O primeiro passo do programa é zerar todas as posições do vetor onde os valores de tensão recebidos pelo *Arduino* serão armazenados.

O segundo passo é esperar pelo recebimento dos dados, neste caso, da massa do usuário, que será enviada pelo aplicativo. Após esta etapa, é necessário que nenhuma força seja aplicada a muleta para que o programa possa calcular o ponto de zero força e obter o valor da correção que será aplicado as medidas feitas posteriormente. Para este valor de correção é feita uma média entre setenta valores obtidos pelo circuito, de maneira que este valor seja o mais exato possível.

Depois de calculado o valor da correção, o *Arduino* recebe do circuito os valores de tensão, por meio de uma de suas portas analógicas. Para que se possa obter um valor mais preciso, foi utilizado a média de setenta valores de tensão e foi calculado o tempo gasto para que este conjunto de valores seja obtido.

Utilizando-se o valor da massa enviada pelo usuário, é calculado o seu peso, pela equação 2.

$$P = mg \tag{2}$$

Sendo *g*, a constante da aceleração da gravidade, com valor aproximado de 9,8 m/s^2, e *m* a massa do usuário. Quando o valor de força calculado pelo *Arduino* exceder 20% deste peso, ele envia uma mensagem de alerta para o aplicativo, juntamente com os dados.

Os valores enviados são:

- Iteração, para conferência se não foi perdido nenhum dado durante a transmissão;
- Massa do usuário, para conferência do correto envio do dado pelo usuário;
- Força, calculada pelo *Arduino*;
- Tempo gasto para obtenção de setenta valores de força.

Todos os valores são enviados entre chaves, para que a leitura possa ser feita corretamente pelo aplicativo.

O programa contém funções de espera, de forma que haja sincronismo entre envio dos dados pelo módulo e o recebimento pelo aplicativo, ocorrendo perdas menos significativas de dados.

O envio dos dados é feito por funções de escrita, utilizadas da mesma maneira quando há conexão serial, *println*, quando há necessidade de passar para a linha seguinte após o envio, e *print*, quando não há necessidade de passar para a linha seguinte.

Ao final de cada envio de conjunto de dados, o *Arduino* verifica se algum dado foi enviado pelo aplicativo, quando o *Arduino* receber a mensagem "Sair", o *Arduino* para de enviar os dados e espera uma nova conexão, para receber novamente um valor de massa, e então reiniciar o envio dos dados.

É necessário que o programa seja carregado, que os pinos do módulo *Bluetooth* e a saída do circuito amplificador sejam corretamente conectados aos pinos do *Arduino*.

O código completo do *Arduino* está no Apêndice B – Código do Arduino.

A Figura 6 mostra o fluxograma do programa desenvolvido para o *Arduino*.

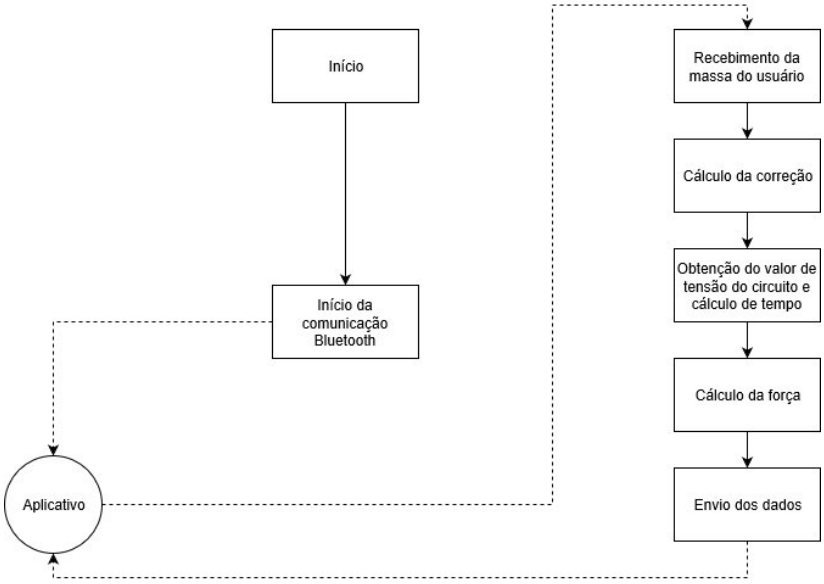

Figura 6 - Fluxograma do programa desenvolvido para o *Arduino*

Fonte – Autoria própria

3.4 Aplicativo

Quando o aplicativo for iniciado, se o *Bluetooth* não estiver ativado, é solicitado ao usuário a sua ativação. Caso o usuário não o ative, o aplicativo é encerrado. A Figura 7 mostra a tela de solicitação da ativação.

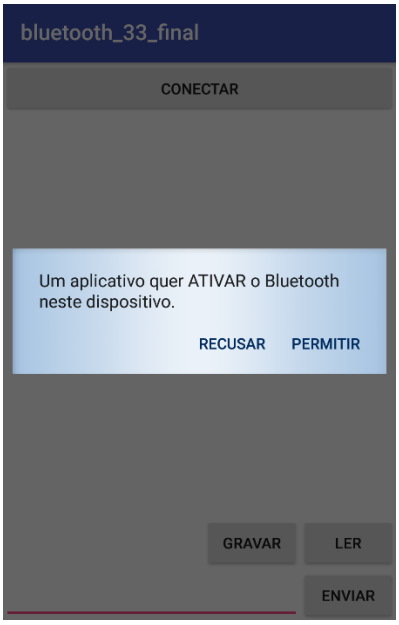

Figura 7 - Solicitação de ativação do *Bluetooth*

Fonte – Autoria própria

É também solicitada a permissão para leitura e escrita de arquivos, para que sempre que for necessário, o aplicativo poder gerar ou ler um arquivo. A Figura 8 mostra a tela de solicitação da permissão de escrita e leitura de arquivos.

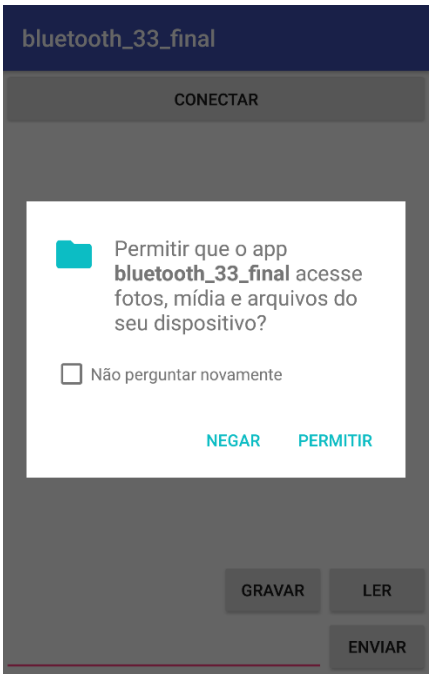

Figura 8 – Solicitação da permissão de escrita e leitura de arquivos

Fonte – Autoria própria

O aplicativo contém em sua tela inicial quatro botões, um para que se possa procurar pelo módulo *Bluetooth*, outro para gravar em um arquivo os dados que serão recebidos, outro para ler estes dados e um quarto botão para que o usuário envie os dados necessários para o *Arduino*. A tela inicial do aplicativo é mostrada na Figura 9.

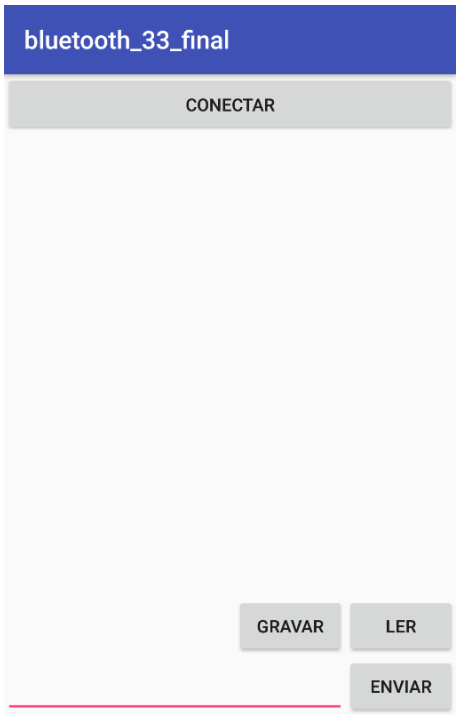

Figura 9 - Tela inicial do aplicativo

Fonte – Autoria própria

Além dos botões, o aplicativo possui uma caixa de texto para o envio dos dados, e duas outras caixas de texto, uma menor, onde o valor pelo usuário enviado será mostrado, e quando a conexão tiver sido estabelecida, mostrará um texto solicitando que o usuário envie sua massa em quilogramas, e uma segunda caixa de texto maior, que exibirá os dados recebidos pelo aplicativo, enviados pelo *Arduino*.

Acionando-se o botão com o texto “Conectar”, uma tela será exibida contendo os dispositivos pareados com o celular e um botão de busca, para que novos dispositivos com o *Bluetooth* ativado sejam procurados pelo aplicativo. Quando o módulo *Bluetooth* for encontrado, esteja ele pareado ou entre os dispositivos encontrados, o nome HC-05 será mostrado juntamente com o seu endereço MAC. Basta selecioná-lo para que a conexão seja iniciada. Caso o módulo ainda não esteja pareado, será solicitada a senha de pareamento.

A Figura 10 mostra a tela exibida com os dispositivos pareados, quando o módulo for encontrado e a solicitação da senha de pareamento.

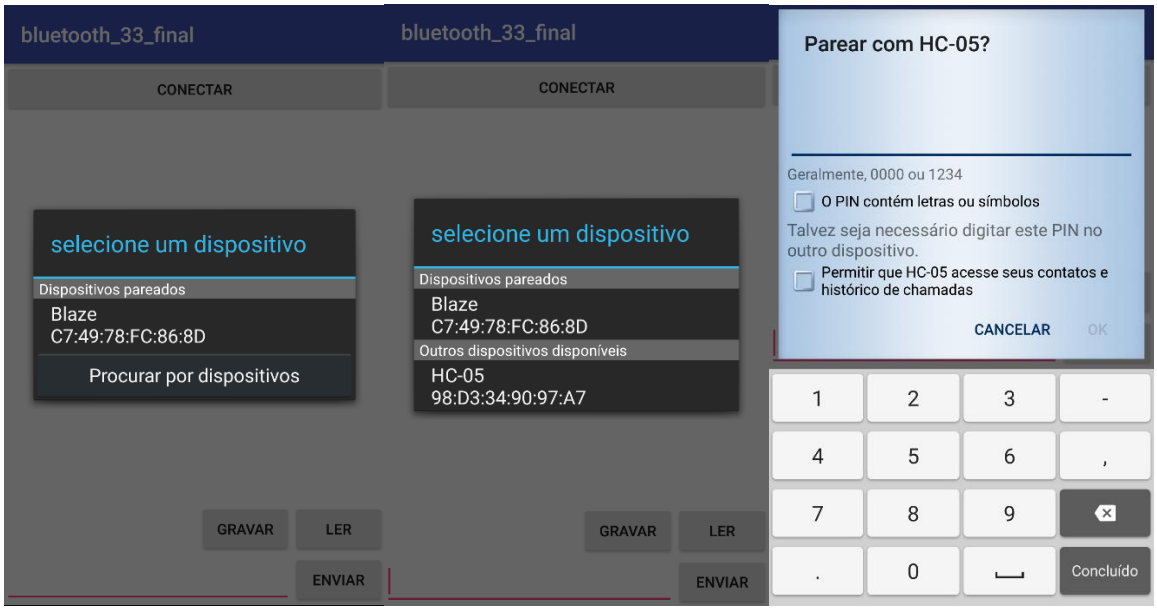

Figura 10 - Telas exibidas ao usuário quando o botão “Conectar” for acionado

Fonte – Autoria própria

Duas mensagens serão exibidas, a primeira, “Conectando...”, enquanto a conexão estiver sendo estabelecida, e “Conectado”, quando a conexão estiver pronta. A Figura 11 mostra como as duas mensagens são exibidas ao usuário.

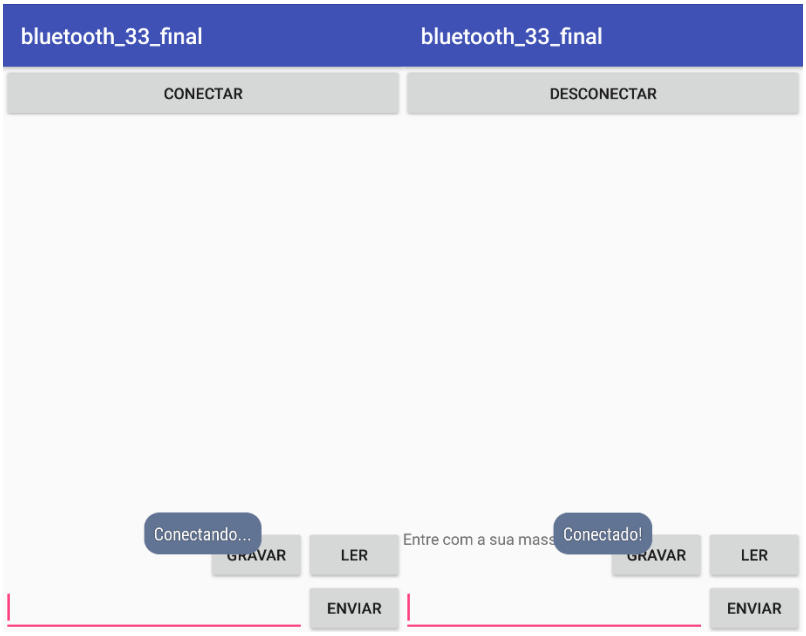

Figura 11 - Mensagens exibidas ao usuário no momento da conexão e com a conexão estabelecida

Fonte – Autoria própria

Despois de estabelecida a conexão, o texto do botão “Conectar” é alterado para “Desconectar”, e na caixa de texto dos dados enviados é mostrada uma mensagem solicitando o envio da massa do usuário. A Figura 12 mostra a tela do aplicativo nesta situação.

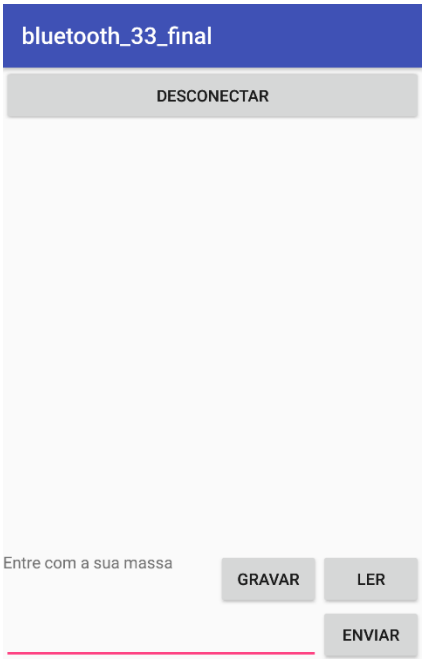

Figura 12 - Tela exibida ao usuário quando for estabelecida a conexão

Fonte – Autoria própria

Após o envio da massa pelo usuário, que deve conter apenas números, com os decimais separados por ponto, os dados começam a ser enviados pelo *Arduino* e recebidos pelo aplicativo. Um exemplo de como os dados são exibidos está na Figura 13.

Figura 13 - Exemplo de exibição dos valores recebidos pelo aplicativo

Fonte – Autoria própria

O *Arduino* coleta setenta valores de tensão, calcula a média destes valores e os transforma em força, segundo o valor da inclinação da reta descrita pela equação 1. Outro valor calculado é o do tempo gasto para que sejam recebidos os setenta valores de tensão, utilizado para que possa ser feito um gráfico que relacione a força obtida, com o tempo gasto. A quantidade de iterações, a massa do usuário, a força e o tempo são os valores enviados pelo *Arduino*.

A partir da massa (*kg*), o *Arduino* calcula o peso (*N*) do usuário. Quando a força enviada pelo *Arduino* exceder 20% do peso calculado, uma mensagem de alerta é enviada pelo *Arduino*, e quando for recebida pelo aplicativo, envia uma notificação ao sistema, seguido de um alerta vibratório e de um toque de notificação padrão do celular, para que o usuário perceba o alerta. A Figura 14 mostra como a notificação é exibida ao usuário, no canto superior esquerdo, com um pequeno símbolo do *Android* e quando a aba de notificações é aberta pelo usuário, o texto apresentado.

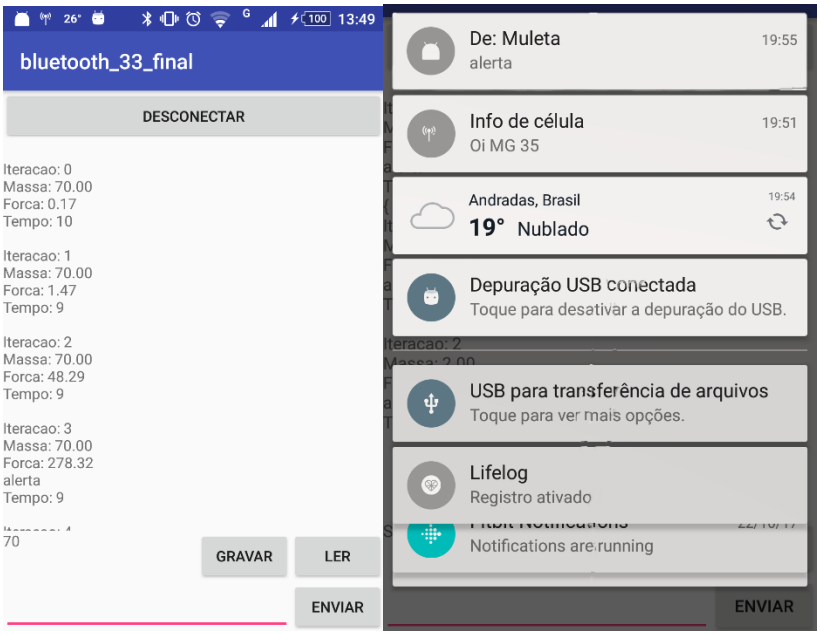

Figura 14 - Notificação exibida ao usuário

Fonte – Autoria própria

Se o usuário selecionar a notificação, o aplicativo é aberto na tela inicial e os valores obtidos desde o início da conexão atual, são exibidos.

Sempre que for necessário que os dados obtidos sejam gravados, basta acionar o botão “Gravar”, que os dados exibidos na tela serão gravados em um arquivo dentro da pasta *Downloads*, na memória interna do celular, de modo que ele possa ser acessado externamente, conectando o celular a um computador, ou por meio de outro aplicativo que gerencie arquivos. Uma mensagem com o diretório e o nome do arquivo gravado é exibida ao usuário. A Figura 15 mostra o acionamento do botão “Gravar”.

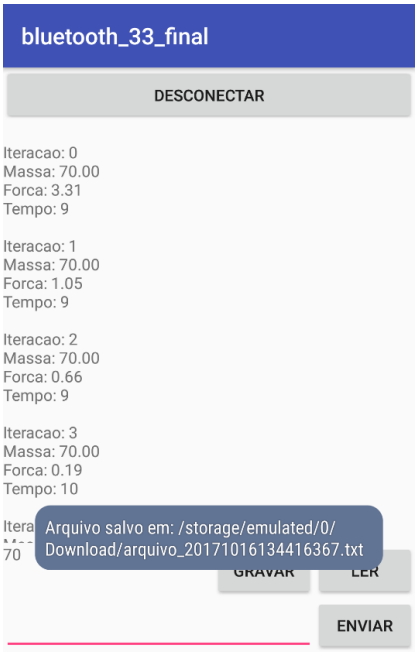

Figura 15 - Acionamento do botão “Gravar”

Fonte – Autoria própria

O botão “Ler” abre uma outra tela para exibir os dados contidos no último arquivo editado dentro da pasta *Downloads*. Uma mensagem com o diretório e o nome do arquivo lido é exibida ao usuário. A Figura 16 mostra o acionamento do botão “Ler”.

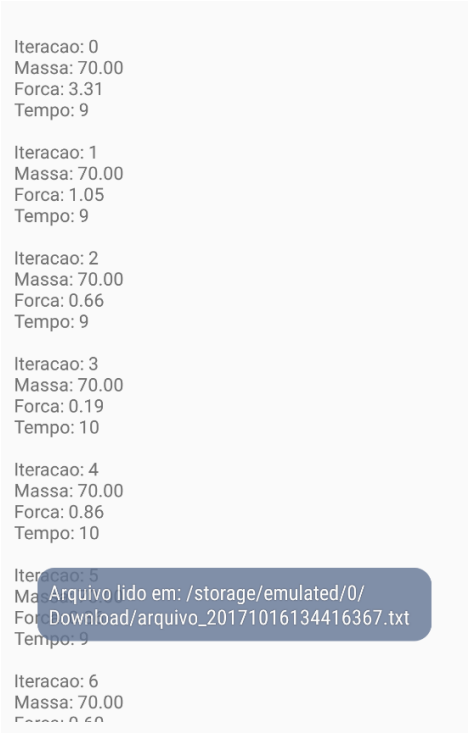

Figura 16 - Acionamento do botão "Ler"

Fonte – Autoria própria

Quando não houver mais necessidade da transmissão, basta pressionar o botão com o texto “Desconectar”, assim a conexão será encerrada e um arquivo com os dados exibidos na tela será gerado. A Figura 17 mostra o acionamento do botão “Desconectar”.

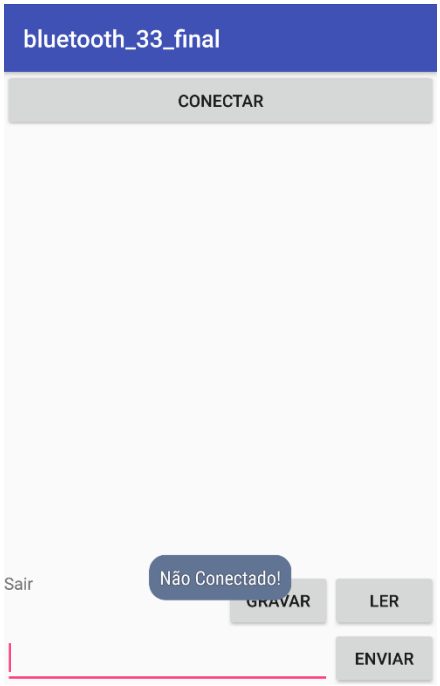

Figura 17 - Acionamento do botão "Desconectar"

Fonte – Autoria própria

3.4.1 Diagramas do aplicativo

Nas Figura 19, Figura 20, Figura 21 e Figura 22 foram utilizadas, para as classes e para o *Arduino* foi utilizado o bloco quadrado, para os métodos, o bloco retângulo com cantos arredondados, para as estruturas de decisão, o losango, para os botões, o triângulo e para o usuário, o círculo.

O diagrama da Figura 18 mostra o fluxograma simplificado do funcionamento do aplicativo.

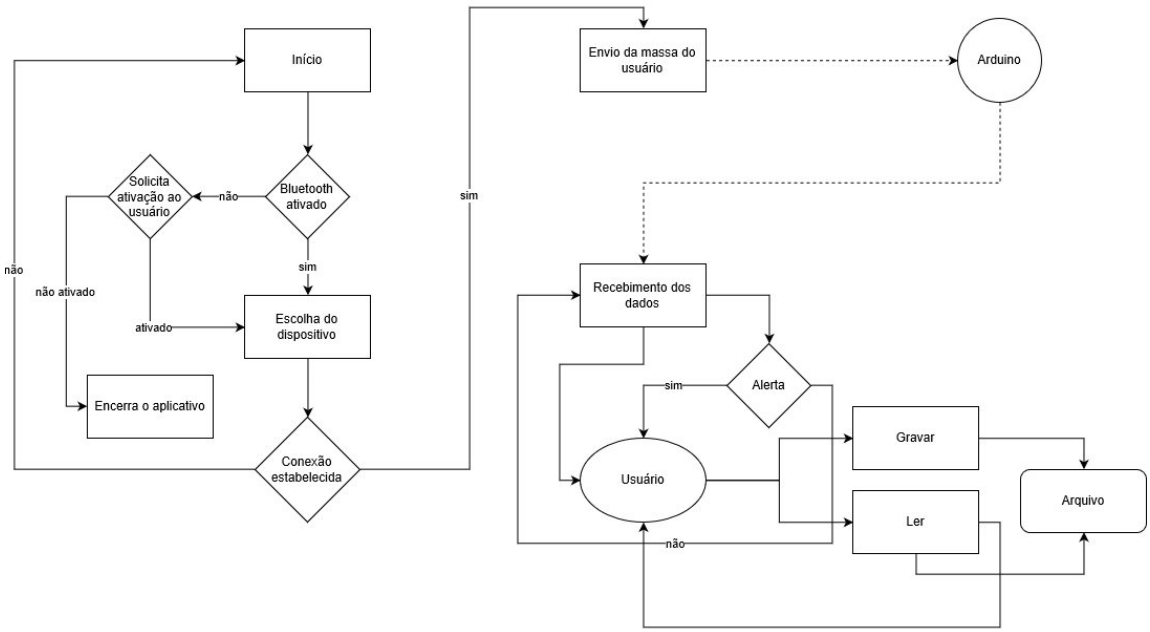

Figura 18 - Fluxograma do aplicativo

Fonte – Autoria própria

Na Figura 19 está o diagrama de fluxo detalhado do aplicativo, entre o instante em que ele é aberto, até o momento em que é estabelecida ou não a conexão com o módulo *Bluetooth*.

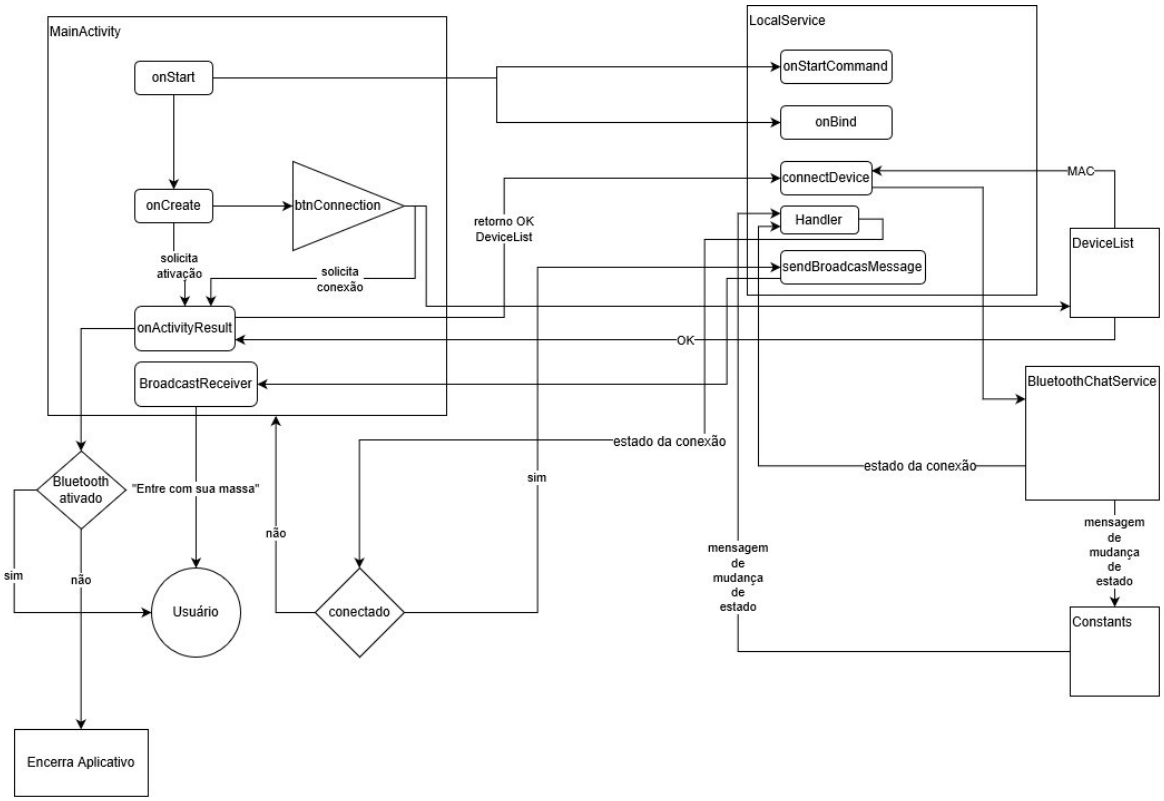

Figura 19 - Fluxo do aplicativo entre o seu início até o estabelecimento ou não de uma conexão

Fonte – Autoria própria

Na Figura 20 está o diagrama detalhado após o estabelecimento da conexão e envio dos dados pelo usuário.

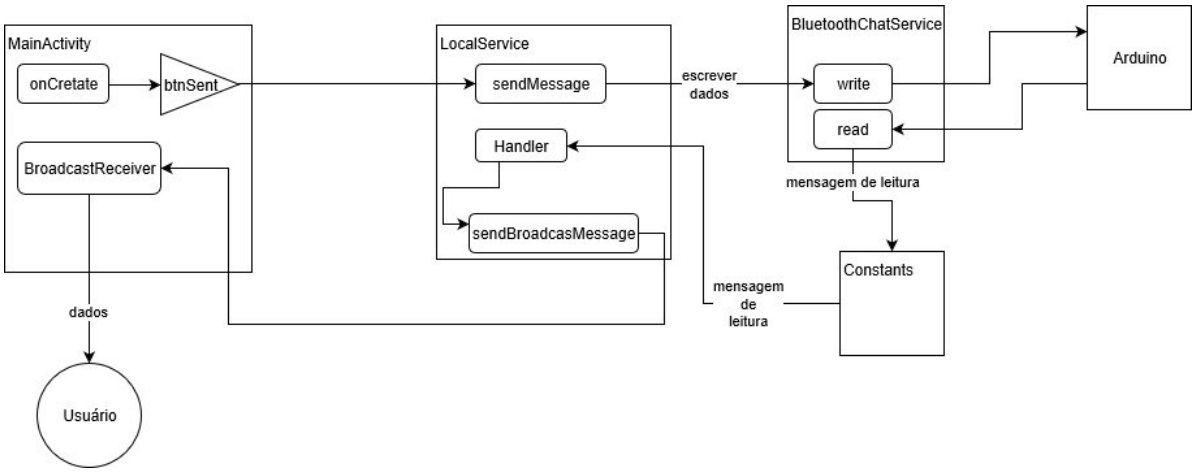

Figura 20 - Fluxo do aplicativo após o estabelecimento da conexão e envio dos dados pelo usuário

Fonte – Autoria própria

O diagrama detalhado da Figura 21 trata de quando for solicitada a desconexão com o módulo, ou algum dos botões de “Gravar” ou “Ler” for acionado.

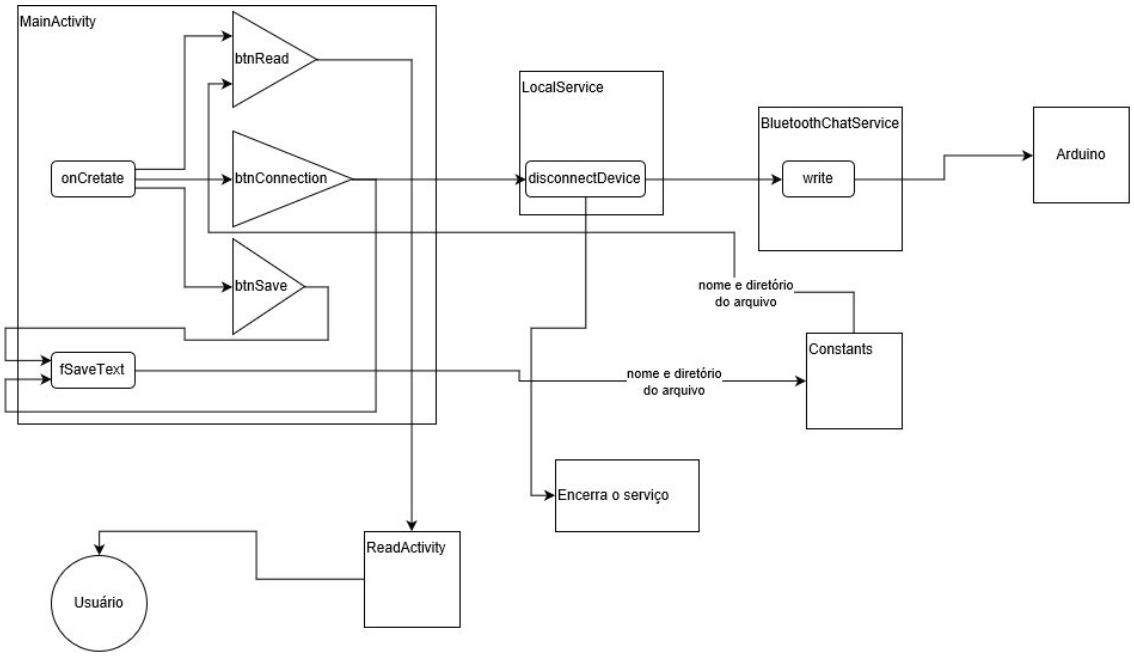

Figura 21 - Fluxo do aplicativo quando for solicitada a desconexão, ou algum dos botões de “Gravar” ou “Ler” for acionado

Fonte – Autoria própria

E a Figura 22 mostra o fluxo detalhado quando o aplicativo retorna do segundo para o primeiro plano.

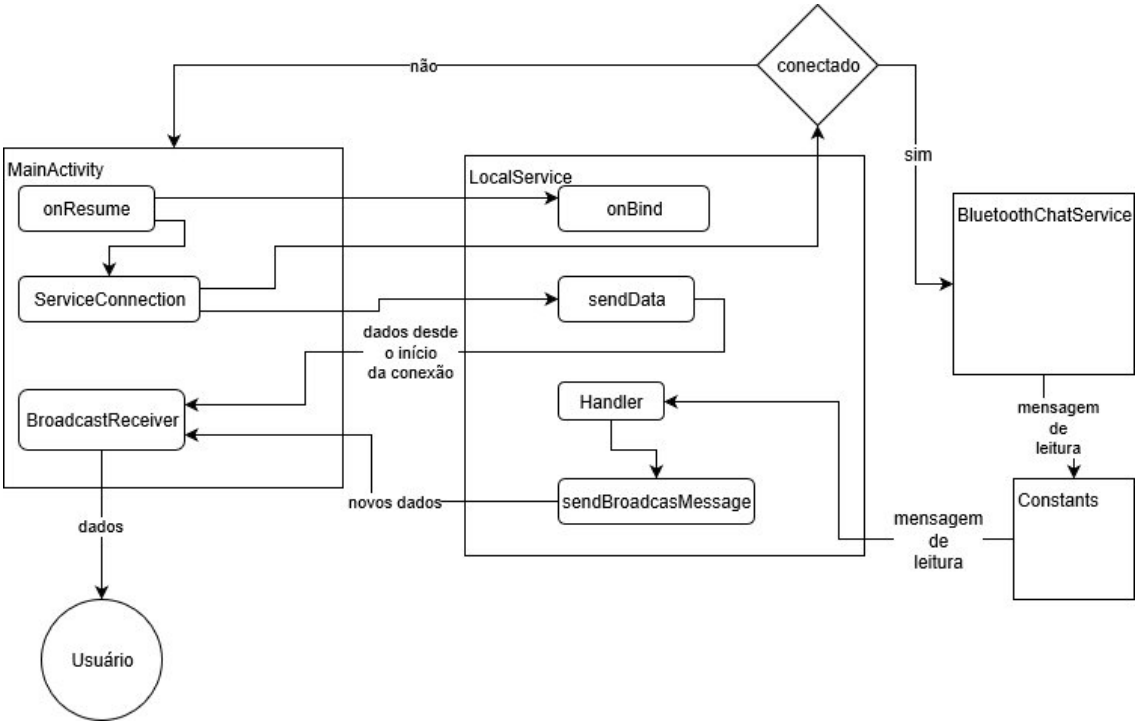

Figura 22 - Fluxo quando o aplicativo retorna do segundo para o primeiro plano

Fonte – Autoria própria

3.4.2 Arquivos de configuração

No arquivo *Manifest* foram adicionadas as atividades *DeviceList*, *ReadActivity* e *LocalService*. Foi definido a orientação retrato para as atividades que possuem *layout* e também foram adicionadas as permissões necessárias de *Bluetooth*, localização, escrita em arquivo e de vibração, que são utilizadas pelo aplicativo.

No arquivo *strings*, foi definido o nome da classe *ReadActivity*, além do nome do aplicativo, definido automaticamente quando um novo projeto é criado.

Uma classe, chamada *Constants* foi criada para armazenar valores que não serão alterados dentro do aplicativo, mas que podem ter valores associados a eles, para facilitar a comunicação entre as classes.

3.4.3 Classes

O código completo do aplicativo encontra-se disponível em [12].

3.4.3.1 *MainActivity*

Na classe principal, denominada *MainActivity*, inicializada quando o aplicativo é aberto, são definidos valores quando há solicitação de ativação do *Bluetooth*, quando há solicitação de conexão e uma variável para definir o nome do arquivo quando for solicitada a sua criação. São definidas também uma variável para saber se há conexão, outra para saber se o serviço foi registrado e outra para instanciar o serviço.

No método chamado ao iniciar esta classe, denominado *onStart*, checa-se a permissão de modificação de arquivos, pedindo ao usuário que aceite, para que os arquivos possam ser gerados pelo aplicativo. Ele também inicializa o serviço, permitindo que ele possa ser executado indefinidamente, e também vincula o serviço, permitindo que os dados possam ser enviados e recebidos do serviço para esta atividade.

O método quando a atividade se tornar visível ao usuário, quando ela for iniciada, ou quando ela voltar do segundo para o primeiro plano, denominado *onResume*, ele faz uma verificação se o serviço já foi registrado, se não foi, ele registra o mensageiro do serviço e vincula novamente a atividade ao serviço.

Quando a atividade não estiver mais visível, chamando *onStop*, o método desfaz o registro do mensageiro.

E quando a atividade estiver para ser destruída, o método *onDestroy* é chamado, e se houver alguma conexão, o serviço é desvinculado da atividade.

Quando a atividade for criada pela primeira vez, utilizando o método *onCreate*, ele correlaciona os objetos presentes no *layout* com as variáveis que serão utilizadas no código. Ele também define o adaptador *Bluetooth* do celular como padrão, e verifica se há este adaptador no celular, caso não haja, uma mensagem de erro é exibida ao usuário, se houver, é feita a verificação se o *Bluetooth* está ativado. Caso o *Bluetooth* não esteja, é solicitado ao usuário para ativá-lo, caso não seja, o aplicativo é encerrado.

Neste método também é definida a função do botão de conexão, inicialmente com o texto “Conectar”. Quando ele for acionado, é feita uma verificação se há alguma conexão, se não houver, é feita uma requisição ao sistema para abrir a classe *DeviceList*; e se houver algum dado na caixa de texto de mensagens recebidas, ele será apagado. Mas caso haja alguma conexão, seu texto estará definido como “Desconectar”, então, é mostrado a mensagem “Sair” na caixa de texto de mensagens enviadas, os dados que estiverem na caixa de texto de dados recebidos serão salvos em um arquivo. Em seguida, o método de desconexão, presente no serviço será chamado e o texto do botão será alterado para “Conectar”.

Também é definido a função do botão “Enviar”. Quando ele for pressionado, o que foi digitado pelo usuário é enviado ao *Arduino* e também será mostrado na caixa de texto de mensagens enviadas, além de apagar os dados recebidos que estiveram na tela e o dado digitado pelo usuário.

Outro método é definido para o retorno da solicitação de ativação do *Bluetooth* e solicitação de conexão. Se a ativação for aceita, uma mensagem é exibida, caso ela não seja aceita, o aplicativo será encerrado. Se a solicitação de conexão for aceita, ou seja, o retorno da atividade *DeviceList* for de sucesso, é chamado o método *connectDevice* do serviço para que seja feita a tentativa de conexão, o mensageiro é registrado e se a conexão for estabelecida, o texto do botão de conexão é alterado para “Desconectar”.

Quando o serviço for vinculado, é necessário criar um método que obtém a instância do serviço, além disto, ele verifica se há conexão, para alterar o texto do botão de conexão, se for necessário. Se houver algum dado armazenado no serviço para ser mostrado na tela do aplicativo, ele é obtido por meio do método *sendData* presente no serviço.

Outro método implementado é o que salva os dados recebidos em um arquivo. Quando o botão “Gravar” for acionado, ou quando uma desconexão for feita, todo o texto que estiver na caixa de texto de dados recebidos é salvo em um arquivo com o nome “arquivo”, adicionando-se a data, com hora, minuto, segundo e milissegundo dentro da pasta *Downloads*, para que este possa ser acessado externamente, tanto por outro aplicativo, como conectando-se o celular a um computador.

Quando for necessária uma verificação do arquivo gerado, o botão “Ler” pode ser acionado. Utilizando o método *getLatestFilefromDir*, que retorna o último arquivo editado dentro da pasta *Downloads*, este método armazena o diretório e o nome do arquivo em uma

variável, associada a uma *string* presente em *Constants*, instanciando assim a atividade *ReadActivity* com o diretório e o nome do arquivo.

Sempre que o serviço enviar algum dado para atividade principal, este será recebido em um método, que primeiramente verifica se a mensagem “conectado” foi enviada pelo serviço, para que o texto do botão de conexão seja alterado para “Desconectar”, o conteúdo da caixa de texto de dados recebidos seja apagado e seja mostrado na caixa de texto de dados enviados, uma mensagem de solicitação da massa do usuário. Caso o dado recebido não contenha a mensagem “conectado”, o dado é mostrado na caixa de texto de dados recebidos.

O método *getLatestFilefromDir* somente retorna o nome do arquivo editado mais recentemente dentro da pasta *Downloads*.

3.4.3.2 *LocalService*

Para que o usuário possa se conectar com o módulo, mas possa deixar que o aplicativo receba dados em segundo plano, foi criado um serviço, chamado *LocalService*.

Nesta classe são definidas variáveis para vincular o serviço, para fazer a comunicação com a classe *BluetoothChatService*: uma para verificação se há conexão e outras para armazenar os dados enviados pelo *Arduino*.

Por ser usado um serviço vinculado, foi criado um método para retornar a instância do serviço, para que uma atividade pudesse usar os métodos públicos desta classe, e um método que fornece uma interface que os clientes usam para se comunicar com o serviço.

Quando uma atividade solicita o início do serviço, é necessário criar um método para que ele possa ser executado em segundo plano, denominado *onStartCommand*. Neste método é chamado outro método que inicializa a classe *BluetoothChatService* e define como adaptador *Bluetooth* o padrão do celular.

Um método público foi criado, denominado *connectDevice*, para que a conexão com o módulo seja feita. Ele utiliza o MAC retornado pela classe *DeviceList* e utilizando um método público da classe *BluetoothChatService*, tenta se conectar com o módulo.

Outro método é o da desconexão, *disconnectDevice*, que envia ao Arduino uma mensagem de “Sair”, para que ele pare de enviar dados para a classe *BluetoothChatService*, altere a variável de conexão para não conectado e para o próprio serviço.

Para que a atividade vinculada ao serviço possa saber quando o módulo está conectado, foi criado um método que somente retorna a variável de controle da conexão.

Para que um dado possa ser enviado do aplicativo para o módulo, foi criado um método, *sendMessage*, que primeiramente apaga qualquer dado armazenado de outra conexão anterior, verifica se há um dispositivo conectado, verifica se há mensagem para ser enviada e então usando o método de escrita do *BluetoothChatService*, envia o dado.

Os dados recebidos pelo serviço precisam ser enviados à classe principal, para serem mostrados ao usuário. Um método foi criado, denominado *sendBroadcastMessage*, que associa os dados desejados a uma *string* estática, e esses dados podem ser acessados por meio do mensageiro da classe *MainActivity*, utilizando a *string* associada.

No serviço também é definido um mensageiro, *Handler*, para que o serviço receba as informações da classe *BluetoothChatService*. Ele verifica se houve alteração no valor associado a alguma constante em *Constants*.

Caso haja alteração no estado da conexão, ele então verifica no *BluetoothChatService* qual o estado atual. Se está conectando ou sem nenhuma conexão, somente uma mensagem é exibida na tela, mas se a conexão foi estabelecida, a variável definida no serviço para o controle da conexão é alterada para conectado e uma mensagem de “conectado” é enviado para a classe principal, utilizando-se o método *sendBroadcastMessage*.

Caso haja alteração na constante associada à mensagem lida, ele verifica a mensagem recebida, armazenando em uma *stringbuilder*, chamada *bluetoothData*. Procura no conteúdo da *stringbuilder* pela abertura e fechamento das chaves. O conteúdo entre chaves é armazenado em uma *string* e enviado à classe principal, mas também é armazenado em outra *stringbuilder*, chamada *serviceData*. Se houver uma mensagem de alerta no dado recebido, o serviço envia uma notificação ao sistema, acompanhada de um alerta vibratório e do toque de notificação padrão do aparelho, para que o usuário seja avisado. Os dados armazenados em *bluetoothData* são apagados ao final desta verificação, para que um novo dado recebido seja analisado separadamente. Contudo, este dado permanece armazenado em *serviceData*, para que sempre que o usuário voltar o aplicativo do segundo para o primeiro plano, todos os valores

armazenados desde o início da conexão atual sejam enviados à classe principal e exibidos na tela.

3.4.3.3 *BluetoothChatService*

A classe *BluetoothChatService* foi retirada de [13], retirando-se do código alguns métodos não necessários para este projeto.

Esta classe, chamada de *BluetoothChatService*, trata da tentativa e da conexão, propriamente dita. Nela são definidas variáveis como a UUID (*Universally Unique Identifier*) usada na conexão com o módulo e os estados da conexão, quando nada está sendo feito, quando está sendo feita a procura, quando há tentativa de conexão e quando é estabelecida a conexão.

Quando a classe *LocalService* receber da classe *DeviceList* o MAC do dispositivo, o serviço instancia a classe *BluetoothChatService*, utilizando o MAC e associando um mensageiro, além de chamar o método público de conexão, para que as *threads* sejam iniciadas e a conexão seja estabelecida.

Assim como na classe das listas, aqui também é definido o adaptador *Bluetooth* do celular como padrão e como estado inicial de conexão, o estado quando nada está sendo feito.

Dentro desta classe, é criado um método público para instanciar esta classe e outro para retornar o estado atual da conexão.

Nesta classe é criada a thread de conexão. Utilizando-se o MAC do dispositivo obtido por meio da atividade *DeviceList* e um UUID, é inicializado um soquete que se conectará ao dispositivo selecionado. Se o UUID do celular corresponder ao do dispositivo, a conexão é aceita e o canal RFCOMM é compartilhado. Nesta *thread* o estado de conexão é alterado para quando está tentando se conectar, e ao final, o método que inicia a thread de gerenciamento de conexão é chamado.

Após estabelecida a conexão, é criada a thread para gerenciar a conexão. Nesta *thread*, o estado de conexão é alterado para conectado e por meio do soquete criado anteriormente, é feito o envio e recebimento de dados, por meio de funções de leitura e escrita, sempre fazendo a transformação dos dados para bytes, e utilizando um mensageiro, chamado de *Handler*, para a transmissão. Por meio do mensageiro, os métodos presentes nesta *thread* associam um dado

à um valor inteiro presente em *Constants*. Quando outra classe quiser obter o dado, basta acessar a referência correta em *Constants*.

Há também um método público que inicia a thread de conexão, cancelando anteriormente qualquer outra *thread* de tentativa de conexão, ou qualquer thread de conexão já estabelecida. Outro método existente é o que inicia a thread de gerenciamento de conexão que estivesse em execução, associando também o nome do dispositivo a um inteiro presente em *Constants*. Estes métodos também executam o método de atualização do estado da conexão, alterando-o para conectando ou conectado, respectivamente.

Outros métodos utilizados são para quando a tentativa de conexão falha, ou quando ela for perdida. Eles mudam o estado da conexão para quando não há nada sendo feito, reiniciam a classe, e a diferença entre as duas é a mensagem exibida ao usuário.

Além de um método que atualiza o estado de conexão, utilizando o mensageiro e associando a um inteiro em *Constants* o novo estado da conexão.

3.4.3.4 *DeviceList*

No código do aplicativo existe uma classe que trata dos dispositivos que podem se conectar com o celular, chamada de *DeviceList*. Esta classe foi retirada de [13], alterando-se somente o idioma das mensagens exibidas ao usuário.

Esta classe possui dois campos de texto, somente para nomear as listas de dispositivos pareados e a lista de dispositivos encontrados. Possui um botão para buscar dispositivos e as duas listas propriamente ditas.

Esta classe usa o adaptador *Bluetooth* do celular como padrão e por meio dele, lista os dispositivos já pareados, obtendo o nome e o MAC de cada um. Caso o módulo não esteja pareado, é necessário fazer uma busca por ele, acionando-se o botão de procura presente no *layout* da lista. Quando for acionado, é feita uma verificação se foi dada permissão pelo usuário para a utilização do *Bluetooth*. Quando a permissão for concedida, uma procura por dispositivos com o *Bluetooth* acionado é feita, listando-os da mesma maneira, caso o dispositivo encontrado já esteja na lista de pareados, ele não é listado novamente.

Quando algum dispositivo de qualquer uma das listas for selecionado, as duas procuras são canceladas e o MAC do dispositivo selecionado é armazenado em uma variável pública, que pode ser recuperada por qualquer outra classe. O resultado desta operação é definido para concluído com sucesso e então esta classe é encerrada.

3.4.3.5 *ReadActivity*

Esta classe mostra os dados contidos no arquivo, chamada de *ReadActivity*. Ela possui somente um campo de texto.

Sempre que um arquivo for gerado, o seu diretório e seu nome são associados a uma variável presente em *Constants*, que é acessada quando esta classe for chamada. Ela lê o arquivo solicitado e mostra no seu campo de texto todo o conteúdo do arquivo.

4 RESULTADOS

4.1 *Hardware*

Foram realizados testes em bancada a fim de verificar a eficiência do circuito e o programa projetados. Os dados de força obtidos foram comparados com os exibidos no dinamômetro. O valor obtido no dinamômetro foi lido após a espera da sua estabilização. Foi feito um teste assumindo que a massa do usuário seja de 70 kg. A Tabela 2 mostra os valores obtidos pelo aplicativo e os valores lidos no dinamômetro, além do erro relativo, estudado em estatística, calculado pela equação 3.

$$erro = \frac{força - força\ dinamômetro}{força\ dinamômetro} 100\% \quad (3)$$

Tabela 2 - Valores obtidos no teste feito para um usuário com massa de 70 kg

Iteração	Massa (kg)	Força (N)	Alerta	Tempo (ms)	Força dinamômetro (N)	Erro (%)
0	70,00	0,81	Não	9	0	-
1	70,00	1,66	Não	9	0	-
2	70,00	1,50	Não	9	0	-
3	70,00	0,64	Não	9	0	-
4	70,00	1,43	Não	10	0	-
5	70,00	1,87	Não	9	0	-
6	70,00	43,62	Não	9	45	3,06667
7	70,00	46,24	Não	9	45	2,75555
8	70,00	47,17	Não	9	45	4,82222
9	70,00	46,44	Não	10	45	3,20000
10	70,00	46,32	Não	10	45	2,93333
11	70,00	46,44	Não	9	45	3,20000
12	70,00	45,81	Não	9	45	1,80000
13	70,00	46,58	Não	9	45	3,51111

Iteração	Massa (kg)	Força (N)	Alerta	Tempo (ms)	Força dinamômetro (N)	Erro (%)
14	70,00	47,41	Não	9	45	5,35555
15	70,00	46,26	Não	9	45	2,80000
16	70,00	45,91	Não	9	45	2,02222
17	70,00	67,23	Não	10	67	0,34328
18	70,00	67,47	Não	9	67	0,70149
19	70,00	68,89	Não	9	67	2,82090
20	70,00	67,29	Não	9	67	0,43284
21	70,00	67,14	Não	9	67	0,20895
22	70,00	68,29	Não	9	67	1,92537
23	70,00	67,11	Não	9	67	0,16418
24	70,00	68,34	Não	10	67	2,00000
25	70,00	67,94	Não	10	67	1,40298
26	70,00	67,79	Não	9	67	1,17910
27	70,00	67,93	Não	9	67	1,38806
28	70,00	93,80	Não	9	92	1,95652
29	70,00	94,02	Não	9	92	2,19565
30	70,00	92,52	Não	9	92	0,56522
31	70,00	93,58	Não	9	92	1,71739
32	70,00	93,33	Não	10	92	1,44565
33	70,00	92,26	Não	9	92	0,28261
34	70,00	94,02	Não	9	92	2,19565
35	70,00	93,89	Não	9	92	2,05435
36	70,00	95,02	Não	9	92	3,28261
37	70,00	118,96	Não	9	115	3,44348
38	70,00	119,77	Não	9	115	4,14783
39	70,00	120,05	Não	9	115	4,39130
40	70,00	120,34	Não	10	115	4,64348
41	70,00	120,29	Não	9	115	4,60000
42	70,00	120,12	Não	9	115	4,45217
43	70,00	120,22	Não	9	115	4,53913
44	70,00	120,19	Não	8	115	4,51304

Iteração	Massa (kg)	Força (N)	Alerta	Tempo (ms)	Força dinamômetro (N)	Erro (%)
45	70,00	119,98	Não	8	115	4,33043
46	70,00	145,91	Sim	11	143	2,03496
47	70,00	144,82	Sim	9	143	1,27273
48	70,00	144,76	Sim	10	143	1,23077
49	70,00	144,74	Sim	9	143	1,21678
50	70,00	145,38	Sim	9	143	1,66434
51	70,00	144,84	Sim	9	143	1,28671
52	70,00	145,79	Sim	9	143	1,95105
53	70,00	165,58	Sim	10	169	2,02367
54	70,00	172,78	Sim	10	169	2,23669
55	70,00	172,20	Sim	9	169	1,89349
56	70,00	171,68	Sim	10	169	1,58580
57	70,00	172,12	Sim	10	169	1,84615
58	70,00	171,11	Sim	9	169	1,24852
59	70,00	170,62	Sim	9	169	0,95858
60	70,00	171,72	Sim	9	169	1,60947
61	70,00	171,66	Sim	10	169	1,57396
62	70,00	196,83	Sim	10	193	1,98446
63	70,00	194,91	Sim	9	193	0,98964
64	70,00	196,56	Sim	10	193	1,84456
65	70,00	197,45	Sim	10	193	2,30570
66	70,00	199,08	Sim	9	193	3,15026
67	70,00	200,01	Sim	10	193	3,15026
68	70,00	198,45	Sim	9	193	3,63212
69	70,00	199,40	Sim	10	193	3,31606
70	70,00	191,76	Sim	10	193	0,64249
71	70,00	265,00	Sim	9	263	0,76046
72	70,00	264,96	Sim	10	263	0,74535
73	70,00	265,70	Sim	10	263	1,02662
74	70,00	263,37	Sim	9	263	0,14068
75	70,00	266,12	Sim	10	263	1,18631

Iteração	Massa (kg)	Força (N)	Alerta	Tempo (ms)	Força dinamômetro (N)	Erro (%)
76	70,00	265,08	Sim	9	263	0,79087
77	70,00	266,45	Sim	9	263	1,31179
78	70,00	265,62	Sim	10	263	0,99620
79	70,00	266,11	Sim	9	263	1,18251
80	70,00	266,28	Sim	10	263	1,24715

Foi possível observar um erro relativo máximo de aproximadamente 5,3%, para valores de força de 47 N e um erro quase constante de aproximadamente 4,6% para valores de força de 115 N. Estes erros são considerados baixos, além do valor apresentado no aplicativo ser maior do que o medido pelo dinamômetro, o que neste caso, faria com que o alerta fosse enviado para uma força menor do que os 20%.

O tempo para a medição dos setenta valores de tensão se manteve entre 8 e 11 ms, o que facilita a criação e análise de um gráfico que relacione o tempo com a força. Este tempo pode ser aumentado, bastando que o número de valores de tensão utilizados para o cálculo da média também seja aumentado. Para este projeto, visou-se um valor de 10 ms.

4.2 *Software*

O software desenvolvido para o *Arduino* não apresentou problemas, assim como o aplicativo. A comunicação *Bluetooth* operou como desejado, sem perdas de dados, mostrando-se bastante eficiente.

Analisando o código no *Android Studio* foi possível observar que o aplicativo recebe um conjunto de valores a cada 1 segundo, e que a opção pelo envio dos dados entre chaves se mostrou eficiente para que o sincronismo entre o envio de dados pelo *Arduino* e o recebimento pelo aplicativo funcionasse de maneira satisfatória.

O sistema de alerta também funcionou adequadamente, mostrando-se eficiente como aviso ao usuário.

A opção pelo envio da notificação se deu pelo fato da possibilidade de integração com *smartbands*, ou *smartwatches*. Existe a possibilidade desta implementação, bastando que o aplicativo seja selecionado como apto a utilizar a *smartband* ou *smartwatch* para notificar o usuário. Considerando que os valores são recebidos com intervalos curtos de tempo, caso seja feita esta integração, recomenda-se o uso de *smartbands* ou *smartwatches* capazes de receber avisos com esta taxa de tempo.

5 CONCLUSÃO

O trabalho proposto consistiu na mudança do sistema de *biofeedback* desenvolvido segundo [3], para a muleta instrumentada, desenvolvida segundo [2], utilizando-se da capacidade de processamento, da memória e da tela dos *smartphones*.

Empregando o protocolo de comunicação *Bluetooth*, presente na grande maioria dos aparelhos atuais, foi possível desenvolver uma solução que apresenta robustez, atenda às necessidades de muitos usuários e não apresente altos custos.

A parte de *hardware* utilizada, possui um circuito baseado no circuito desenvolvido segundo [3], enquanto que o microcontrolador empregado, o ATmega328P, foi o responsável pelo recebimento e processamento dos dados vindos do circuito, por apresentar facilidade de prototipagem e de programação. O módulo, conectado ao microcontrolador, foi o responsável pela comunicação com o aplicativo, utilizando-se de um protocolo de comunicação sem fio. Todo este sistema, contendo, circuito, microcontrolador e módulo, demonstra uma solução simples e barata, além de possuir caráter móvel, o que facilita a sua implementação.

Por meio de conceitos existentes na programação para *Android*, foi possível desenvolver um aplicativo que além de atender as necessidades do projeto, também levasse em consideração a maneira como os *smartphones* são utilizados no dia a dia.

Para analisar os dados obtidos pelo hardware, foi utilizado um dinamômetro conectado diretamente a haste onde os *strain-gauges* estão colocados.

Observa-se um erro relativo máximo de 5,3%. No entanto, as variações entre uma medida e outra obtidas pelo sistema não são relevantes, o que demonstra que uma melhor calibração da reta que relaciona força e tensão pode ser feita para que os erros sejam menores e menos variantes.

A análise do *software* foi feita a partir dos *logs* obtidos no *Android Studio*, verificando se o dispositivo selecionado correspondia ao desejado, se a conexão foi estabelecida, se os valores estavam sendo enviados corretamente e se os valores recebidos estavam coerentes. Os dados recebidos também podem ser analisados pelo arquivo gerado. Esta análise mostrou que a conexão é eficiente, não apresentando perdas significativas dos valores enviados pelo *Arduino*

e que são recebidos pelo aplicativo, além de apresentar os valores aos usuários em intervalos de tempo relativamente pequenos.

O projeto funcionou da forma esperada e demonstrou ser uma boa opção para a utilização deste sistema como *biofeedback* para a muleta.

Durante o desenvolvimento do projeto foram enfrentados problemas que ajudaram na formação acadêmica, além da busca de novos conhecimentos, como por exemplo, a programação para *Android*, utilizando-se a linguagem *Java*, que não foi estudada durante a graduação. Enquanto que algumas disciplinas auxiliaram no desenvolvimento do código utilizado no *Arduino*.

REFERÊNCIAS

[1] Melis, E. H., Torres-Moreno, R., Barbeau, H., Lemaire, E. D., 1999. Analysis of assited-gait characteristics in persons with incomplete spinal cord injury. Spinal Cord, 37, pp. 430-439

[2] Leite, F. I. L., Cliquet, A. Desenvolvimento de uma bengala instrumentalizada para fins de acompanhamento clínico. Em: 8º Congresso Brasileiro de Engenharia Biomédica, São José dos Campos, 2002

[3] Varoto, R., Sato, A. M. R., Lins, C., Cliquet, A. Can Simple Electronic Instrumentation Asociated with Basic Training Help Users of Assistive Devices? In: Biodevices, 2014. Universidade de São Paulo e Universidade de Campinas. Epub ahead of print, 2013.

[4] What is Arduino? Disponível em: <https://www.arduino.cc/en/Guide/Introduction> Acesso em 30 Out. 2017

[5] Módulo Bluetooth RS232 HC-05. Disponível em: <https://www.filipeflop.com/produto/modulo-bluetooth-rs232-hc-05/> Acesso em 30 Out. 2017

[6] Android ultrapassa marca de 2 bi de dispositivos ativos por mês. Disponível em: <https://exame.abril.com.br/tecnologia/android-ultrapassa-marca-de-2-bi-de-dispositivos-ativos-por-mes/> Acesso em 30 Out. 2017

[7] Android Studio. Disponível em: <https://developer.android.com/studio/index.html> Acesso em 30 Out. 2017

[8] Atividades. Disponível em: <https://developer.android.com/guide/components/activities.html?hl=pt-br> Acesso em 30 Out. 2017

[9] Serviços. Disponível em: <https://developer.android.com/guide/components/services.html?hl=pt-br> Acesso em 30 Out. 2017.

[10] Serviços Vinculados. Disponível em: <https://developer.android.com/guide/components/bound-services.html?hl=pt-br> Acesso em 30 Out. 2017

[11] Bluetooth. Disponível em: <https://developer.android.com/guide/topics/connectivity/bluetooth.html> Acesso em 30 Out. 2017

[12] App Muleta. Disponível em: < https://github.com/gamatossilva/app_muleta> Acesso em 01 Dez. 2017

[13] Android BluetoothChat Sample. Disponível em: <https://github.com/googlesamples/android-BluetoothChat/#readme> Acesso em 30 Out. 2017

Apêndice A – Conjuntos de valores para a obtenção da curva

Tabela 3 - Primeiro conjunto de valores para a obtenção da curva

Força (N)	Tensão (V)
7	0,566
16	0,611
22	0,643
26	0,672
31	0,693
36	0,721
41	0,749
46	0,776
50	0,800
56	0,830
60	0,855
65	0,884
70	0,905
75	0,934
80	0,961
85	0,988
90	1,015
95	1,040
99	1,068
104	1,094
109	1,119
114	1,147
118	1,171
123	1,193
128	1,219
133	1,246
138	1,269
153	1,337
168	1,417

Força (N)	Tensão (V)
182	1,494
197	1,572
211	1,649
225	1,728
240	1,806
254	1,883
268	1,958
282	2,035
296	2,112
310	2,184
323	2,257
337	2,329
350	2,400
364	2,474
377	2,545
390	2,616
403	2,688
415	2,758
425	2,809

Tabela 4 - Segundo conjunto de valores para a obtenção da curva

Força (N)	**Tensão (V)**
10	0,557
15	0,604
25	0,636
30	0,664
35	0,699
40	0,723
44	0,747
49	0,774
54	0,801
58	0,828
63	0,856
68	0,882
73	0,909
78	0,936
83	0,963
87	0,987
92	1,015
97	1,040
102	1,066
107	1,066
111	1,116
116	1,141
121	1,171
126	1,195
132	1,229
136	1,256
141	1,276
152	1,330
166	1,410
181	1,488
195	1,566

Força (N)	Tensão (V)
205	1,645
223	1,722
237	1,799
251	1,876
265	1,951
279	2,025
293	2,098
306	2,174
320	2,248
333	2,321
346	2,392
360	2,465
373	2,536
386	2,606
393	2,648
406	2,718
422	2,801

Tabela 5 - Terceiro conjunto de valores para a obtenção da curva

Força (N)	**Tensão (V)**
10	0,607
18	0,651
25	0,686
29	0,709
33	0,734
39	0,765
44	0,795
49	0,819
54	0,844
59	0,871
64	0,900
68	0,922
73	0,950
78	0,975
83	1,000
88	1,027
93	1,055
98	1,081
103	1,105
107	1,132
112	1,159
117	1,184
123	1,218
127	1,242
132	1,265
136	1,288
141	1,313
152	1,367
166	1,447
180	1,525
195	1,601

Força (N)	Tensão (V)
205	1,680
223	1,756
237	1,832
251	1,909
265	1,985
275	2,061
293	2,137
307	2,211
320	2,284
334	2,356
347	2,429
360	2,501
374	2,569
387	2,642
400	2,711
412	2,778
425	2,844

Apêndice B – Código do *Arduino*

```
#include <SoftwareSerial.h>
# define cont 70

// Inicialização de variáveis
int i = 0, j = 0;
float massa = 0.0, forca_media = 0.0, correcao = 0.0, peso = 0.0, auxiliar = 0.0,
tensao_media = 0.0, tensao[cont];
String comando = "";

// Variáveis para contagem do tempo
unsigned long m1, m2, tempo;

// Pinos para o recebimento e envio de dados via Bluetooth
SoftwareSerial bluetooth(10, 11);

// Inicialização da comunicação
void setup() {
  Serial.begin(9600);
  bluetooth.begin(9600);

}

void loop() {

  // Zera os valores do vetor que que receberá as tensões enviadas pelo circuito
  for (j = 0; j < cont; j++){
    tensao[j] = 0.0;
    }

  // Espera para receber algum dado do aplicativo
  if(bluetooth.available()){
    while(bluetooth.available()){

      // Transforma a "string" recebida em "float"
      massa = bluetooth.parseFloat();

      // Faz o cálculo do peso
      peso = massa * 9.8;
    }

    // Verifica se a massa é diferente de zero
    if (massa > 0.00){

      // Calcula o valor da correção, para que a reta sempre comece em (0,0)
      for (i = 0; i < 70; i++) {
        int sensorValue = analogRead(A1);
        correcao = correcao + sensorValue * (5.0 / 1023.0);
```

```
    }

    // Faz a média de 70 valores para um valor mais preciso da correção
    correcao = correcao / cont;
    delay(1000);
    j = 0;

    // Envia os dados até receber a mensagem "Sair"
    while(true){

      delay(1000);

      // Começa o envio dos dados com um abre chave para que possa ser exibido
corretamente
      bluetooth.println("{");

      // Variável utilizada apenas para conferência se houve dados não enviados
      bluetooth.print("Iteracao: ");
      bluetooth.println(j);

      // Inicia contagem do tempo
      m1 = millis();

      // Recebe 70 valores de tensão para uma medida mais precisa e depois faz a média
      for (i = 0; i < cont; i++) {
        int sensorValue = analogRead(A1);
        tensao[i] = sensorValue * (5.0 / 1023.0);
        tensao_media = tensao_media + tensao[i];
      }

      // Termina a contagem do tempo
      m2 = millis();

      // Calcula o tempo gasto para aquisição das tensões
      tempo = m2 - m1;
      tensao_media = tensao_media/cont;

      // Faz a correção para o ponto (0,0) da reta
      auxiliar = tensao_media - correcao;
      tensao_media = fabs(auxiliar);

      // Converte a tensão em força, através da reta obtida pelos pesos padrões
      forca_media = 185.4336 * tensao_media;

      // Envia o valor da massa, que o usuário enviou, apenas para conferência
      bluetooth.print("Massa: ");
      bluetooth.println(massa);

      // Envia o valor da força calculada
      bluetooth.print("Forca: ");
```

```
        bluetooth.println(forca_media);

        // Caso o valor da força exceda 20% do peso do usuário, envia uma mensagem de alerta
para o aplicativo
        if(forca_media >= 0.2 * peso){
          bluetooth.println("alerta");
          }

        //Envia o tempo gasto na medição das tensões
        bluetooth.print("Tempo: ");
        bluetooth.println(tempo);

        // Termina o envio, fechando a chave
        bluetooth.println("}");

        // Lê o valor enviado pelo aplicativo, se receber a mensagem "Sair", para de enviar os
dados
        char caracter = bluetooth.read();
        comando += caracter;
        delay(1);

        if(comando.indexOf("Sair") >= 0){
          bluetooth.println("sair");
          comando = "";
          break;
        }
        j++;
      }
    }
  }
}
```