Universidade Estadual do Maranhão - UEMA
Centro de Ciências Tecnológicas - CCT
Curso de Engenharia de Computação

Rafael Ribeiro Bento e Silva

Sistema embarcado integrado à aplicativo Android para uso em bicicletas

São Luís
2018

Universidade Estadual do Maranhão - UEMA
Centro de Ciências Tecnológicas - CCT
Curso de Engenharia de Computação

Rafael Ribeiro Bento e Silva

Sistema embarcado integrado à aplicativo Android para uso em bicicletas

Monografia apresentada ao Curso de Graduação em Engenharia de Computação da Universidade Estadual do Maranhão como requisito para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Cícero Costa Quarto

São Luís
2018

Silva, Rafael Ribeiro Bento e.
Sistema embarcado integrado à aplicativo Android para uso em bicicletas/Rafael Ribeiro Bento e Silva. – São Luís, 2018
61 f.

Monografia (Graduação) – Curso de Engenharia de Computação, Universidade estadual do Maranhão, 2018.

Orientador: Prof. Dr. Cícero Costa Quarto.

1. Aplicativo Android. 2. Bicicleta. 3. Bluetooth.
4. Sistema embarcado. I. Título.
CDU 004.451:796.61

Rafael Ribeiro Bento e Silva

Sistema embarcado integrado à aplicativo Android para uso em bicicletas

Monografia apresentada ao Curso de Graduação em Engenharia de Computação da Universidade Estadual do Maranhão como requisito para obtenção do grau de Bacharel em Engenharia de Computação.

Trabalho aprovado. São Luís, 28 de junho de 2018:

Prof. Dr. Cícero Costa Quarto
Orientador

Prof. Me. Alfredo Costa Oliveira Junior
Avaliador

Prof. Me. Wesley Batista Dominices de Araújo
Avaliador

São Luís
2018

Dedico este trabalho aos meus pais Lindioneza e José, e a namorada, Thamires, que me apoiaram durante o desenvolvimento deste trabalho.

Agradecimentos

Agradeço aos meus pais pelo suporte e incentivo que me deram durante o curso e antes dele, e também a namorada que me apoiou para concluir este trabalho. Ao meu orientador Cícero Costa Quarto, agradeço pela ajuda na realização deste trabalho e aos meus colegas de turma por serem pessoas amigáveis, que tendem a ajudar uns aos outros, principalmente com relação às "coisas do curso".

"O grande objetivo da educação não é o saber, mas a ação."
(Herbert Spencer)

Resumo

O projeto desenvolvido para este trabalho consiste de um sistema embarcado e um aplicativo Android, se comunicando através da tecnologia Bluetooth Low Energy, permitindo, assim, a exploração de aspectos não existentes no Bluetooth clássico. Entre as ferramentas envolvidas tem-se a linguagem de programação Kotlin, e o sistema embarcado Arduino Nano, este último sendo um dos elementos do próprio sistema embarcado do projeto. A comunicação das duas partes principais do projeto (do sistema embarcado com o aplicativo) tem como objetivo auxiliar o ciclista no acompanhamento do seu desempenho em atividades com a bicicleta. No presente trabalho é abordado também os principais tópicos teóricos subjacentes das tecnologias empregadas, trazendo-os de maneira suscinta, para uma compreensão facilitada da aplicação. Para compôr a parte de resultados, opta-se por fazer testes, que, mesmo não sendo exaustivos em quantidade, apontam para uma direção sólida de resultados que o sistema obtém, comparando-o com outros aplicativos conhecidos de ciclismo.

Palavras-chave: Android. Bicicleta. Bluetooth. Sistema Embarcado.

Abstract

The project developed for this work consists of an embedded system and an Android application, communicating through Bluetooth Low Energy technology, thus allowing the exploration of aspects that do not exist in classic Bluetooth. Among the tools involved are the Kotlin programming language, and the Arduino Nano embedded system, the latter being one of the elements of the project's own embedded system. Communication of the two main parts of the project (from the embedded system with the application) aims to assist the cyclist in monitoring his performance in bicycle activities. In this paper we also discuss the main theoretical topics underlying the technologies employed, bringing them in a succinct way, for a better understanding of the application. To compose the results part, we choose to do tests, which, even though they are not exhaustive in quantity, point to a solid direction of results that the system obtains, comparing it with other known cycling applications.

Keywords: Android. Bicycle. Bluetooth. Embedded System.

Lista de ilustrações

Lista de tabelas

Lista de códigos

Lista de abreviaturas e siglas

ADC	Analog-to-digital converter (conversor analógico para digital)
ATT	Attribute Protocol
BLE	Bluetooth Low Energy
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GPS	Global Positioning System (Sistema de Posicionamento Global)
LCD	Liquid crystal display (display de cristal líquido)
ONU	Organização das Nações Unidas
PWR	Power LED
SIG	Special Interest Group
SO	Sistema Operacional
TCC	Trabalho de Conclusão de Curso
UI	User Interface (Interface do Usuário)
UUID	Universally Unique Identifier

Sumário

1 Construção do Objeto de Pesquisa

1.1 Contextualização

Tem havido recentemente incentivos governamentais em vários lugares do mundo para suportar o uso de bicicletas em ambientes urbanos, incentivando a utilização desse meio de transporte, substituindo outros meios como carro e moto para uma parcela da população. Vários diferentes tipos de programas almejam aumentar o seu acesso, tanto através de incetivos a sua aquisição, ou através de programas que permitem o seu uso temporário, conhecidos como programas de compartilhamento de bicicleta, que em conjunto com a infraestrutura viabilizou o seu uso como transporte e viu este aumentar em grandes cidades como Barcelona e Paris (PUCHER; DILL; HANDY, 2010).

A utlização de bicicletas no Brasil é maior nas pequenas e médias cidades, tendo os horários entre 6h e 7h e das 16h às 19h dos dias úteis como os horários de maior uso (BRASIL, 2007). As grandes cidades tendem a enfrentar problemas externos às bicicletas que dificultam o seu uso como tráfego urbano intenso de veículos e falta de infra-estrutura.

Na tabela 1 se faz um comparativo de algumas informações sobre a produção de bicicletas e motocicletas no Brasil, se destacando que o Brasil é o quarto maior produtor mundial de bicicletas, segundo dados da Abraciclo:

Tabela 1 – Fechamento de 2017 para a fabricação nacional de bicicletas e motos.

Bicicletas	Motocicletas
Frota nacional: mais de 70 milhões de unidades Produção anual: 2,5 milhões de unidades 4º maior produtor mundial	Frota nacional: mais de 26 milhões de unidades Produção anual: mais de 880 mil unidades 8º maior produtor mundial

Fonte – (ABRACICLO, 2018)

Nota – Excluídas as bicicletas infantis, classificadas como brinquedos.

Uma das preocupações recentes no âmbito social é com relação a preservar o meio ambiente para a geração atual e futuras. A Organização das Nações Unidas (ONU) elegeu a bicicleta como o transporte ecologicamente mais sustentável do planeta (BRASIL, 2007).

Dentre as características favoráveis para o seu uso, destaca-se o **baixo custo de aquisição e manutenção**, sendo a bicicleta muito utilizada no Brasil entre a população com baixa renda nas periferias das grandes cidades e cidades de pequeno e médio porte; **Rapidez**, sendo a mesma três vezes mais rápida que com relação à uma pessoa caminhando e dependendo do tráfego, e em distâncias de menos de 5 km, chega a ser mais rápida que o carro (BRASIL, 2007).

Há vários modos de uso da bicicleta, não apenas para lazer, incluindo também modos de uso como:

- Uso para deslocar-se para o trabalho;
- Uso dpara deslocar-se para a escola;
- Uso para competições;
- Uso no transporte de mercadorias;
- Uso no transporte de correspondência;
- Uso no transporte eventual de produtos e correspondências pessoais;
- Uso como veículo propulsor de baú;
- Uso como veículo de transporte de pessoas além do condutor.

Destaca-se aqui o uso da bicicleta para deslocar-se para estudar, sendo este o segundo maior motivo para o uso da mesma, tanto no Brasil, como em todo o mundo (BRASIL, 2007).

Sensores com a tecnologia de bluetooth se tornaram uma melhor alternativa a outros métodos de coleta de dados de tempo de viagem que tem sido tradicionalmente mais caros e invasivos a privacidade (MEI; WANG; CHEN, 2012). Algumas das formas de mensurar o tempo de viagem, além do uso de sistema embarcado com bluetooth, incluem: GPS, dispositivos mecânicos ligados ao odômetro (usado extensivamente em carros) e computadores de bordo.

Da definição encontrada em (GIL, 2002), "pesquisa é um procedimento racional e sistemático que tem como objetivo proporcionar respostas aos problemas que são propostos". Mais adiante nesta mesma obra, Gil (2002) elenca duas categorias de razões pelo qual as pesquisas são feitas, que são as pesquisas de ordem intelectual e pesquisas de ordem prática. O presente trabalho procura explorar de maneira prática o problema do monitoramento de dados do ciclista para suas atividades, buscando maior efetividade no uso de equipamentos mais adaptados às características da bicicleta, sendo almejado equipamentos de menor tamanho e baixo consumo de energia, para que permita um uso prolongado dos mesmos por ciclistas em atividades diárias.

Este trabalho busca contribuir como uma adição na procura por alternativas de controle do desempenho do ciclista através de dispositivos e tecnologias com aspectos que, quando combinados, trazem melhorias no registro das atividades do ciclista com relação ao consumo de energia e possível adaptação em um produto comercial, dado o uso de componentes menores que os seus equivalentes tradicionais, como os que empregam o Arduino Uno ou MSP430. Outro enfoque é para seu uso *offline*, ou seja, sem conexão com a

internet, por isso a escolha dos sensores ao invés de utilizar APIs de geo-localização como a API Google Maps.

1.2 Motivação

Ciclismo é saudável e impacta nos níveis de atividade física, taxa de obesidade, saúde cardiovascular e morbidade (PUCHER; DILL; HANDY, 2010). Muitas agências governamentais e organizações de saúde pública tem advogado por mais ciclismo como uma maneira de, além de melhorar a saúde individual, contribuir para a população no geral, reduzindo a poluição do ar, emissão de carbono, barulho e perigos do tráfego urbano (PUCHER; DILL; HANDY, 2010).

Motivado por fatores como a grande oferta de smartphones atualmente e da importância do ciclismo como uma alternativa de transporte, neste trabalho de conclusão de curso se busca desenvolver um sistema embarcado para ser usado com bicicleta sem precisar de conexão com Internet ou GPS ativo, que produz como resultados e mostra na interface de um aplicativo Android (desenvolvido para este trabalho) informações que podem ser úteis para o controle próprio da atividade do ciclista, como velocidade média, comprimento do percurso percorrido, e tempo de viagem. Se estabelece para este trabalho algumas características almejadas que o mesmo atinja quando concluído, que são: uso de componentes de tamanho reduzido; baixo consumo de energia; e facilidade de replicação deste projeto tendo as informações contidas neste trabalho.

1.3 Objetivos

1.3.1 Objetivo geral

O objetivo geral deste TCC consiste em desenvolver um sistema de pequeno porte e de baixo consumo de energia, que auxilie ciclistas em obter estatísticas do uso da bicicleta com precisão satisfatória.

1.3.2 Objetivos específicos

A partir do objetivo geral, tem-se os seguintes objetivos específicos:

- Implementar os componentes do projeto (aplicativo Android e sistema embarcado acoplado a bicicleta);
- Estabelecer comunicação entre os componentes através da tecnologia BLE (bluetooth low energy);

- Comparar a precisão dos valores obtidos com o sistema deste TCC com outros aplicativos que utilizam outros métodos para obter dados do ciclista.

1.4 A estrutura organizacional do trabalho

Além do Capítulo 1, já descrito, o trabalho se encontra organizado em mais quatro capítulos. O capítulo 2 aborda a fundamentação teórica dos maiores tópicos que o trabalho abrange, como o SO Android, a linguagem de programação Kotlin, e a tecnologia BLE; Na parte final do capítulo é dedicado uma seção a descrever sobre a bicicleta, que é um tópico que difere dos anteriores por se tratar de uma tecnologia mecânica, e na última seção é analisado três trabalhos correlatos deste, e traçado paralelos com os mesmos. O capítulo 3 refere-se a uma apresentação do projeto e seus componentes, explicando o funcionamento das partes isoladamente e em conjunto, como um sistema, com propósito definido. No mesmo capítulo é explicado os componentes de hardware utilizados e o desenvolvimento do aplivativo Android. O capítulo 4 apresenta os resultados do trabalho. O capítulo 5 conclui com sugestões de melhorias no sistema desenvolvido para este trabalho, e apontando para trabalhos futuros a partir deste.

2 Fundamentação Teórica

2.1 Android

Android é um sistema operacional para dispositivos móveis conhecidos como smartphones, onde sua arquitetura se apresenta em camadas de *software*, que do nível mais baixo são mais próximas do *hardware* do smartphone, enquanto as camadas mais altas estão mais ligadas às aplicações em si, como mostra a figura 1:

Figura 1 – Arquitetura do sistema operacional Android

fonte: (LEE, 2012)

As aplicações Android são desenvolvidas utilizando-se de "blocos de construção" de aplicações chamados **componentes**. Em Android, há quatro tipos de componentes:

- Activities: Uma *activity* proporciona a janela pelo qual o aplicativo desenha sua UI. Geralmente, uma activity implementa uma tela em um aplicativo (DEVELOPERS, 2018a).
- Services: Um *Service* é um componente do aplicativo que pode realizar operações longas e não fornece uma interface do usuário. Outro componente do aplicativo pode iniciar um serviço e ele continuará em execução em segundo plano, mesmo que o usuário alterne para outro aplicativo (DEVELOPERS, 2018c).

- Broadcast receivers: Um *broadcast receiver* é um componente que pode responder à uma mensagem de broadcast enviada por um cliente (MACLEAN; KOMATINENI; ALLEN, 2015).

- Content providers: Provedores de conteúdo (Content Providers) gerenciam o acesso a um conjunto estruturado de dados. Eles encapsulam os dados e fornecem mecanismos para definir a segurança dos dados. Provedores de conteúdo são a interface padrão que conecta dados em um processo com código em execução em outro processo (DEVELOPERS, 2018b).

2.2 Kotlin

Kotlin é uma liguagem de programação que foi desenvolvida inicialmente por um time de programadores da JetBrains[1], que foi anunciada pela mesma em 2011 e tornada open source no começo de 2012 sob a licença Apache 2.

Na conferência Google I/O de 2017, a Google anunciou que a linguagem Kotlin passa a ser oficialmente suportada para o desenvolvimento de aplicações Android, junto com as que já contam com suporte oficial que são java e C++. No apândice A são explicados através de exemplos a sintaxe da linguagem Kotlin em alguns tópicos de interesse para o entendimento do código do aplicativo deste trabalho.

2.3 Bluetooth Low Energy

O BLE (Bluetooth Low Energy) foi primeiramente introduzido em 2010 e especificado pelo Bluetooth SIG (Bluetooth Special Interest Group) com o propósito de ser uma nova especificação do Bluetooth que atendesse a demanda da indústria por uma tecnologia que fosse construída desde seu início pensada em ser empregada em dispositivos com restrições de consumo de energia, e não um paliativo sobre a tecnologia do Bluetooth clásssico. O BLE foi introduzido com a especificação 4.0 do Bluetooth que agora passou a especificar tanto a versão clássica e a nova versão chamada também de Bluetooth Smart.

Abaixo lista-se os conceitos essenciais, junto com uma breve descrição de cada para o entendimento de como foi utilizado a tecnologia BLE no aplicativo desenvolvido para este trabalho:

- GATT: Generic Attribute Profile (GATT) define uma estrutura de dados hierárquica (figura 2) que é exposta aos dispositivos BLE conectados. GATT é construído no topo do Attribute Protocol (ATT) (SIG, 2018c).

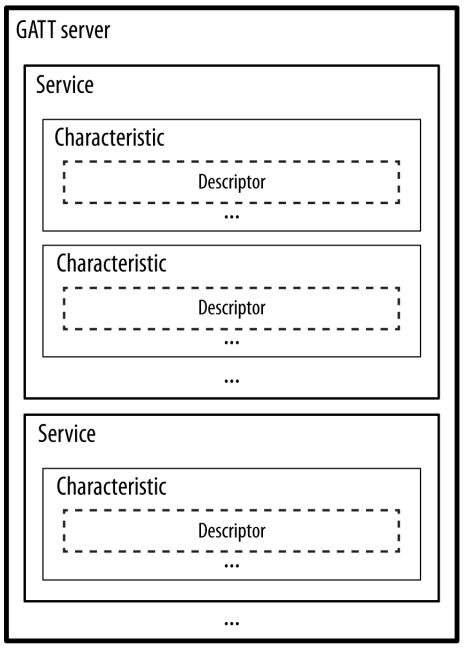
[1] JetBrains é uma companhia de software fundada no ano 2000, que tem como foco geral de mercado o desenvolvimento de ferramentas e tecnologias para uso por programadores.

- Services: *Services* são conjuntos de *characteristics* e relacionamentos com outros *services* que encapsulam o comportamento de parte de um dispositivo (SIG, 2018d).
- Characteristics: *Characteristics* são tipos de atributos definidos que contêm um único valor lógico (SIG, 2018a).
- Descriptor: *Descriptors* são atributos definidos que descrevem um valor *characteristic* (SIG, 2018b).
- UUID: Um UUID é um número de 128 bits (32 hexas) que é garantido (ou tem grande chance) de ser globalmente único. UUIDs são usados em diversos protocolos e aplicações além do Bluetooth; Seu formato, uso e geração são especificados pelo ISO/IEC 9834-8:2005 (TOWNSEND et al., 2014). Para o uso com BLE, os 24 hexas menos significativos dos UUIDs são padrões e se repetem entre as classes de atributos. Logo um BLE UUID é como o mostrado abaixo:

  ```
  XXXXXXXX-0000-1000-8000-00805f9b34fb
  ```

  Onde os "X" acima devem ser substituídos pelos valores hexa do UUID específico.

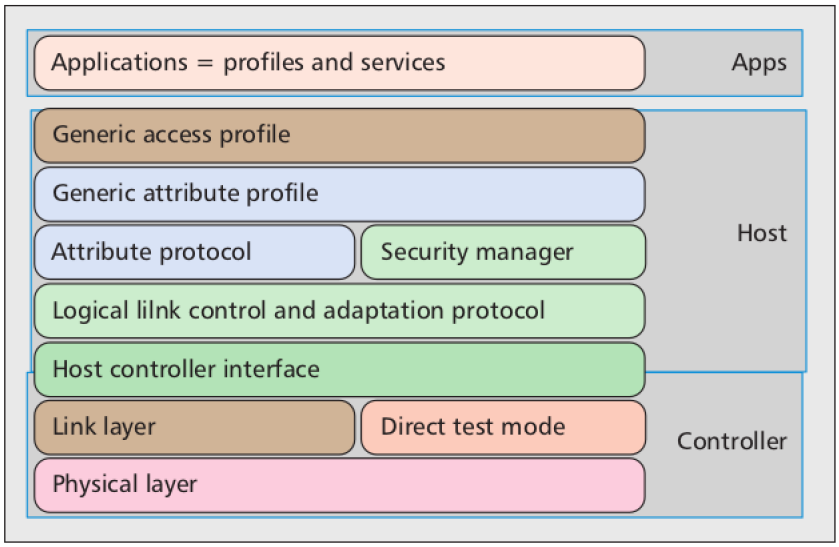
Figura 2 – Hierarquia de dados GATT



Fonte: (TOWNSEND et al., 2014)

A pilha de protocolos BLE é exibida na figura 3, e é visto na figura a divisão que ocorre dos protocolos dada a hierarquia de camadas na pilha de protocolos e funções que exercem.

Figura 3 – Pilha de protocolos BLE



Fonte: (CHANG, 2014)

Há uma diferença entre papéis e seus significados quando trata-se de papéis GATT e papéis GAP, e um dispositivo pode mesclar papéis de ambos os tipos sem nenhuma restrição. Um cliente GATT envia requisições à um servidor GATT e aguarda por respostas. Já um servidor GATT recebe requisições de um cliente e envia a resposta para o mesmo cliente. No caso dos papéis GAP, tem-se quatro tipos de papéis:

- Broadcaster: Um dispositivo com papel de Broadcaster envia periodicamente pacotes advertising com dados.
- Observer: Um Observer é otimizado para únicamente fazer a leitura dos dados dos pacotes advertising oriúndos dos Broadcasters.
- Central: Um dispositivo com papel central pode se conectar a um ou mais periféricos.
- Peripheral: Um periférico se conecta à apenas um Central.

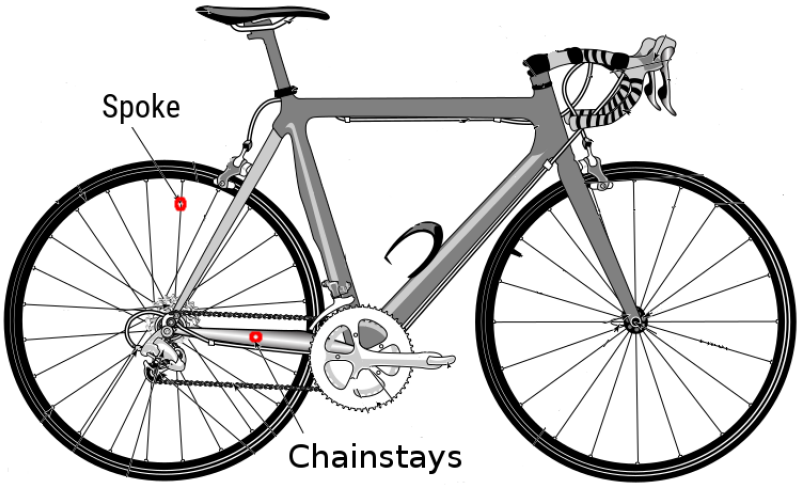
Cada dispositivo pode operar em um ou mais papéis GAP ao mesmo tempo, e a especificação não impõe restrições dessa ordem (TOWNSEND et al., 2014).

2.4 Bicicletas

Bicicletas são máquinas mecanizadas (BAILEY; GATES, 2009) que precisam de reparos e manutenção ao longo do tempo, sendo parte deles podendo ser feitos em casa/-garagem e outros podendo ser feitos normalmente apenas em lugares especializados, por precisarem de ferramentas de maior custo e complexidade de uso. Na figura 4 temos

as partes de uma bicicleta de estrada com seus respectivos nomes apontados, e a seguir uma descrição suscinta das partes da bicicleta no qual os componentes do projeto deste trabalho são fixadas.

Figura 4 – Partes de uma bicicleta de estrada



Fonte: (BAILEY; GATES, 2009)

- Chainstays: Tubo inferior traseiro do quadro da bicicleta. Nesta parte são fixados a maior parte dos componentes do projeto como o Arduino Nano, módulo HM-10 e bateria de 9 V, além dos componentes auxiliares.
- Spoke: Raio de roda da bicicleta. Em um dos raios da roda traseira da bicicleta é fixado um imã de tal modo que ele passe rente ao reed switch que está fixado no chainstays.

2.5 Trabalhos correlatos

Nesta seção, são apresentados de maneira suscinta trabalhos correlatos, pautando-se nas semelhanças e diferenças. São trazidos três trabalhos para apresentação e comparação.

Em seu trabalho, Oliveira e Afonso (2017) apresentam uma plataforma para monitorar vários parâmetros durante a atividade de ciclismo. Toda informação, ao ser coletada, é georeferenciada com a localização por GPS, diferindo deste trabalho que não se utiliza de GPS. Os parâmetros são obtidos (1) através de sensores do próprio smartphone onde o aplicativo está instalado, (2) através de sensores espalhados pelo corpo do ciclista, e (3) através de sensores colocados na bicicleta. O aplicativo permite visualizar, armazenar, e compartilhar mapas de rotas com amigos. Com isso, o presente trabalho difere deste

também com relação a variedade de uso de sensores, pois o mesmo não utiliza-se de sensores embutidos no smartphone nem de sensores fixados no ciclista.

Já Miskon et al. (2014) expõe um dispositivo de ciclismo *fitness* atrelado à uma bicicleta, assumindo a função de bicicleta estacionária, que gamifica o usuário com o intuito de incentivar a atividade no mesmo ambiente, seja interno ou externo. Esse dispositivo usa sensores infra-vermelho em conjunto com Arduino Fio (diferindo do presente trabalho que utiliza Arduino Nano) para mensurar e calcular a velocidade da bicicleta, e outras informações, enviando essas informações para um monitor através de módulos X-Bee Pro que viabilizam a comunição sem fio do sistema. O presente trabalho difere mais significativamente desse trabalho correlato descrito do que do primeiro com relação à sua funcionalidade e dispositivos empregados, apesar de ambos apresentarem numerosas diferenças (tabela 2).

Em Silva, Oliveira e Vanzin (2014), tem-se a descrição de aspectos técnicos e de projeto de um sistema para controle da atividade do ciclista, em que a sua interface é uma tela de LCD 16x2, onde ao lado desta há botões para trocar a informação exibida e para zerar todos os valores da atividade corrente. Um aspecto a ser destacado é a função de diagnosticar se o sistema precisa de calibragem de pneus, de acordo com a duração da atividade.

O presente trabalho difere dos trabalhos correlatos descritos por se utilizar exclusivamente da tecnologia BLE para a comunicação entre o sistema embarcado e a UI (aplicação Android), e por empregar componentes de menor tamanho, como o Arduino Nano.

Tabela 2 – Comparativo do presente trabalho e dos trabalhos correlatos

Sistema	Comunicação principal	Sistema embarcado da família Arduino	Interface	Tipo de bicileta
BicikloDroid	BLE	Arduino Nano	Aplicação Android	Convencional
Sistema por Oliveira e Afonso	BLE e Bluetooth clássico	Não informado se utiliza	Aplicação Android	Convencional
Sistema por Miskon et al.	Módulos XBee Pro (rádio)	Arduino Fio	*Laptop*/Projetor	Estacionária
MyBike por Silva, Oliveira e Vanzin	Direta por fios	Arduino Uno	*Display* de LCD 16x2	Convencional

Fonte – Oliveira e Afonso (2017), Miskon et al. (2014), e Silva, Oliveira e Vanzin (2014)

3 Arquitetura do Projeto

O projeto para este trabalho consiste de um sistema embarcado acoplado à uma bicileta e uma aplicação Android nomeada BicikloDroid.

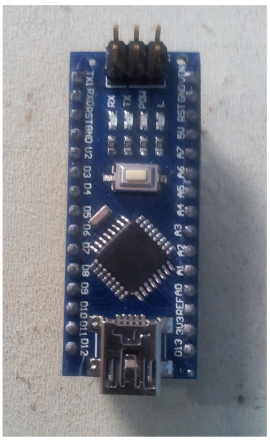
3.1 Sistema embarcado

O sistema embarcado é constituído dos seguintes componentes:

- Arduino Nano: O Arduino Nano é um sistema embarcado que contém no mesmo o microcontrolador ATmega328P. Este sistema embarcado é muito similar ao mais popular Arduino Uno, tendo o diferencial de ser menor que este e mais algumas diferenças importantes, como o número de portas analógicas do Nano sendo superior (NANO, 2009).

  Abaixo uma figura do Arduino Nano utilizado neste trabalho, em sua versão 3.0:

Figura 5 – Arduino Nano 3.0 utilizado



Fonte: O autor

A seguir é apresentado uma tabela com algumas informações técnicas referentes ao Arduino Nano:

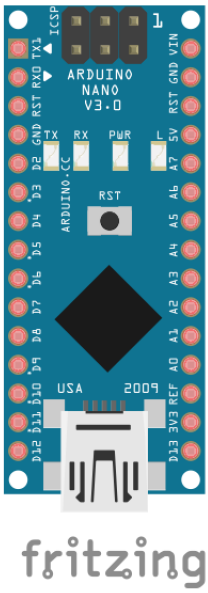
Tabela 3 – Características principais do Arduino Nano

atributo	valor
CPU	Microcontrolador Atmel Atmega 328P
Tensão mínima de alimentação	$3.3V$
Tensão máxima de alimentação	$20V$
Tensão ideal para alimentação	$7V - 12V$
Pinos digitais I/O	14 pinos, 6 destes também servindo para saídas PWM
Pinos de entrada analógica	8 entradas com 10 bits de resolução
Corrente DC por pino I/O	$40mA$
Memória flash	$16Kb$, sendo que $2Kb$ são utilizados pelo bootloader
SRAM	$2Kb$
EEPROM	$1Kb$
Velocidade de clock	$16MHz$

Fonte – (ARAÚJO, 2014)

Na figura 6 tem-se o esquema de pinagem do Arduino Nano e na tabela 4 informações referentes a cada pino na figura.

Figura 6 – Esquema de pinagem do Arduino Nano



Fonte: o autor (gerado com o software Fritzing)

Tabela 4 – Descrição e tipo dos pinos do Arduino Nano

Pinos	Tipo	Descrição
TX1, RX0, D2-D13	Entrada/saída	Portas digitais entrada/saída de números 0 à 13
RST (2 pinos)	Entrada	Reset
GND (2 pinos)	PWR	Ground
3V3	Saída	Saída +3,3V (do FTDI)
REF	Entrada	Referência ADC
A0-A7	Entrada	Canais de entrada analógica de 0 à 7
5V	Entrada ou saída	Saída +5 V (do regulador na própria placa) ou Entrada +5 V (de uma fonte de alimentação externa)
VIN	PWR	Tensão de alimentação

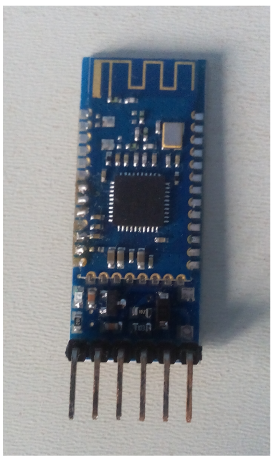
Fonte – Adaptado de (NANO, 2009)

- Módulo Bluetooth 4.0 BLE HM-10 CC2541: O HM-10 é um pequeno módulo V4.0 BLE (Bluetooth Low Energy), baseado no chip CC2540 ou CC2541 (JTC, 2014). Há duas versões do HM-10:
  - HM-10C
  - HM-10S

  Operacionalmente os dois são iguais e a substituição de um por outro neste trabalho não leva a qualquer mudança no projeto.

  O módulo HM-10 (figura 7) utilizado neste trabalho tem uma estrutura acoplada que expõe conexões em pinos macho que facilitam sua utilização em protoboards.

Figura 7 – Módulo HM-10



Fonte: o autor

Na tabela 5 elenca-se esses pinos com seus respectivos rótulos e descrições, para facilitar na visualização do esquema posteriormente e servindo como consulta.

Tabela 5 – Pinagem do módulo HM-10 na estrutura ZS-040

Pinos	Descrição
STATE	Status da conexão. ALTO quando conectado, e BAIXO quando não conectado
RXD	Receptor UART serial
TXD	Transmissor UART serial
GND	Terra comum
VCC	Energia de alimentação de 3,6V à 6V
EN	Opera em estado ALTO, caso transmute para BAIXO desabilita o módulo

Fonte – Adaptado de Currey (2018)

Conectando o módulo no PC, pode-se utilizar o Arduino IDE para enviar e receber comandos através da interface UART do módulo com o monitor serial da IDE. A seguir apresentamos alguns dos comandos AT que o módulo aceita e sua utlização.

O comando AT retorna OK caso o módulo não esteja conectado à um dispositivo remoto, e caso conectado o módulo se desconecta.

```
>AT
Ok
```

O comando seguinte retorna o endereço MAC do módulo.

```
>AT+ADDR?
OK+ADDR:C8FD1900F91F
```

O comando AT+ADVI? retorna o parâmetro do intervalo do *advertising* atual do módulo. Há um comando auxiliar para modificar o valor do parâmetro atual.

```
>AT+ADVI?
OK+Get:0
```

O comando AT+ADTY? retorna o tipo de advertising. Também há um comando auxiliar para modificar o valor do parâmetro atual.

```
>AT+ADTY?
OK+Get:0
```

AT+CHAR? retorna os números que indicam a characteristic configurada no módulo. Ao desenvolver uma aplicação em Android que utilize o módulo, a characteristic utilizada no código fonte deve coincidir com a characteristic especificada no módulo, caso contrário, ao utilizar certas funções de comunicação entre o módulo e a aplicação, a aplicação finaliza inadvertidamente.

```
>AT+CHAR?
OK+Get:0xFFE1
```

Este comando retorna o nome do módulo bluetooth, este nome que será visto ao tentar parear o módulo ou escanear procurando por dispositivos dentro de uma aplicação.

```
>AT+NAME?
OK+NAME:HMSoft
```

O próximo comando retorna o código PIN presente no módulo. Este código pode ser modificado através do comando AT+PIN[*valor*], sendo o *valor* um número entre 000000 e 999999. O código PIN presente no módulo por padrão é 000000.

```
>AT+PIN?
OK+Get:000000
```

No comando a seguir o retorno do comando será o papel GATT tipificado no dispositivo. Quando retorna 0, significa que o dispositivo está configurado com papel servidor, e quando retorna 1 significa q o papel identificado no dispositivo é de cliente.

```
>AT+ROLE?
OK+Get:0www.jnhuamao.cn
```

Já este comando retorna a versão do módulo. Dada a versão do módulo, ao consultar a página do fabricante pode-se obter informações sobre se algum recurso está presente ou faltando no módulo daquela versão.

```
>AT+VERS?
HMSoft V602
```

- Reed Switch: Reed switch é um interruptor formado por duas chapas metálicas, e é operado por um campo magnético aplicado neste dispositivo (EVANS, 2011). Quando se aproxima um imã à este dispositivo, as chapas se fecham fazendo assim com que essa mudança de estado possa ser captada por possíveis componentes conetados a este, no caso deste projeto o Arduino Nano. Na figura 8 tem-se a imagem real do tipo de reed switch utilizado.

Figura 8 – Reed switch usado no projeto

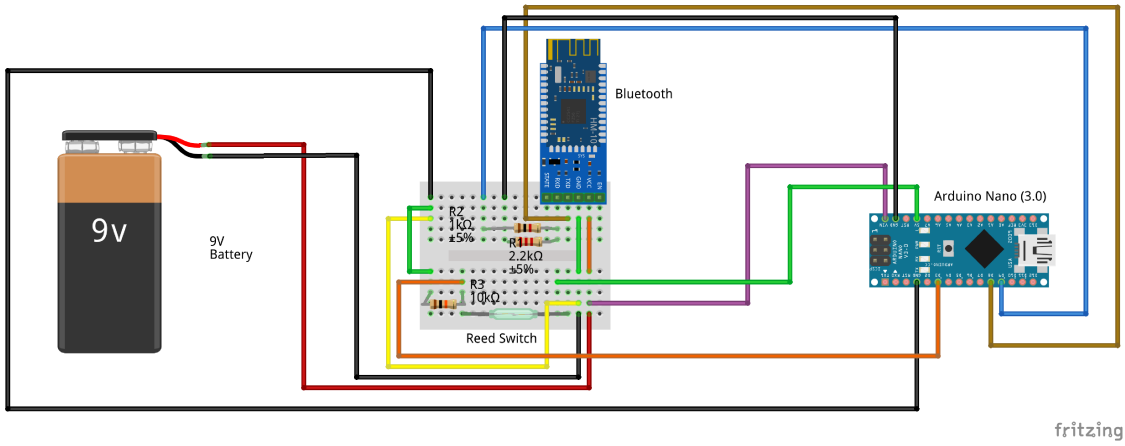
Fonte: O autor

- Imã
- Fios
- Resistores
- Mini-protoboard

No aro da bicileta é preso um imã que a cada revolução de roda passará rente ao reed switch do sistema embarcado, fazendo que esta passagem seja captada pelo Arduino Nano através da programação gravada neste. Com a detecção dessa revolução, o Arduino Nano envia uma flag para o módulo na sua entrada RX, e este módulo, estando no papel de servidor, enviará ao cliente (no caso, a aplicação Android esperando por respostas do servidor conectado), esta flag será tratada pelo aplicativo como mais uma revolução da roda.

A seguir apresenta-se a modelagem do sistema embarcado:

Figura 9 – Modelagem do sistema embarcado do projeto



Fonte: O autor (gerado com o *software* Fritzing)

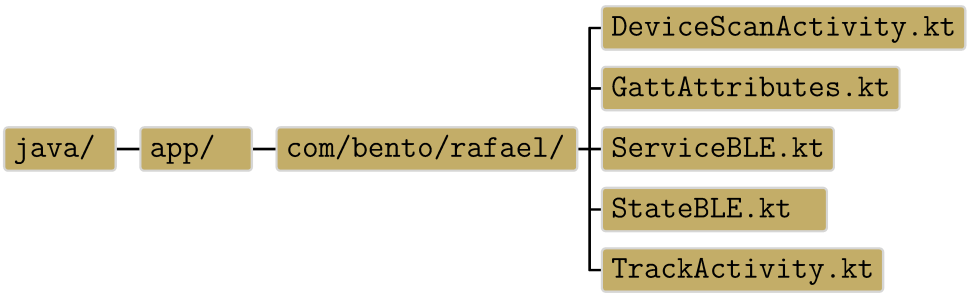
3.2 Aplicação Android

3.2.1 Diagramas UML

Para o desenvolvimento do aplicativo Android deste trabalho é utilizado para testes e debug um smartphone positivo selfie com a versão do Android 5.0.2. Optou-se por utilizar nesse aplicativo a linguagem de programação Kotlin, uma das linguagens de programação suportadas oficialmente pela Google para o desenvolvimento de aplicações Android atualmente. Ao criar um novo projeto com a IDE Android Studio, tem-se na raiz desse projeto os seguintes diretórios e arquivos como mostrado na figura 10.

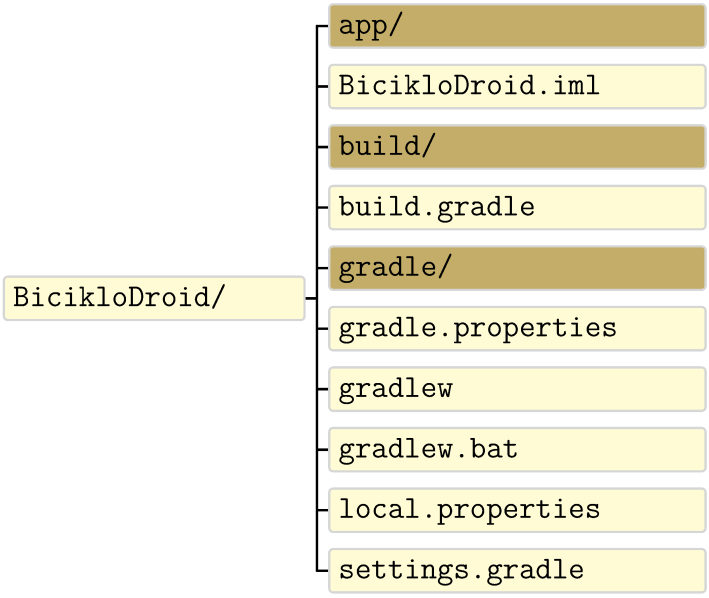
Os arquivos Kotlin que foram desenvolvidos para a parte do aplicativo se encontram no subdiretório do diretório app (indicado na figura 10) em *src/main/java/com/bento/rafael*, listados na figura 11:

Figura 11 – Arquivos kotlin localizados em subdiretório



Fonte: o autor

Figura 10 – Árvore de diretórios/arquivos do aplicativo em Android



Fonte: o autor

A seguir é explicado a função de cada arquivo na aplicação:

- DeviceScanActivity.kt: Esse arquivo contém a definição da classe **DeviceScanActivity** e das classes internas **ViewHolder** e **LeDeviceListAdapter**. A classe DeviceScanActivity é uma Activity de entrada da aplicação que é responsável por escanear e mostrar os dispositivos BLE disponíveis.

Figura 12 – Diagrama de classe da classe DeviceScanActivity

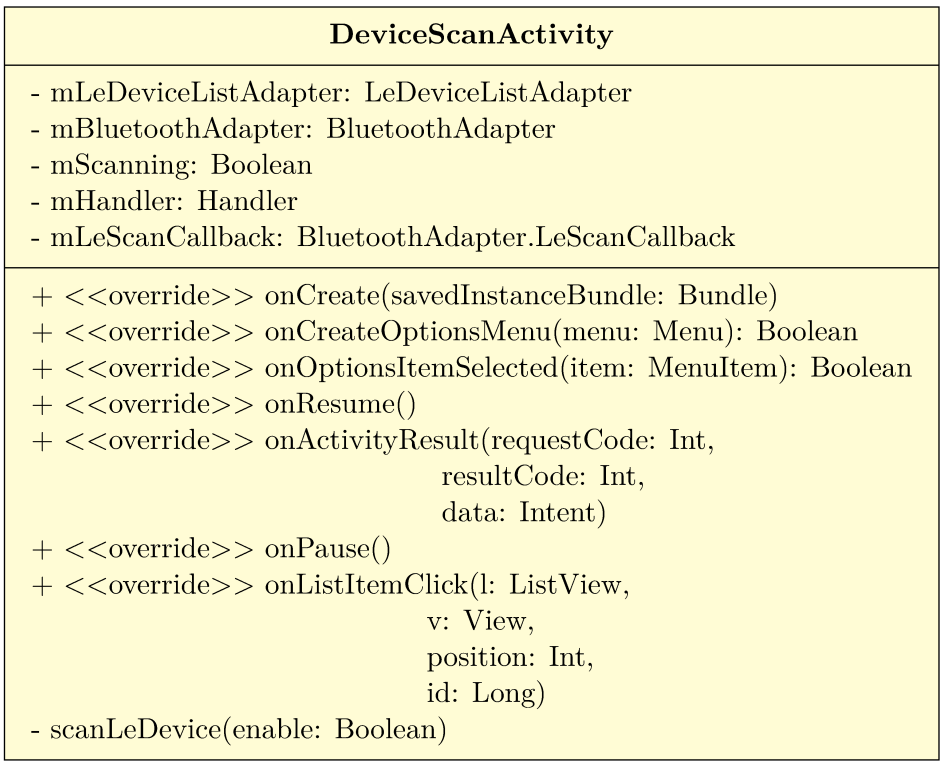


Figura 13 – Diagrama de classe da classe interna ViewHolder

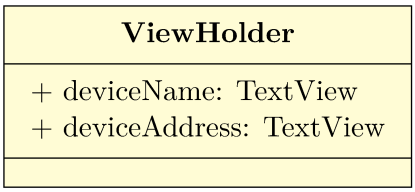
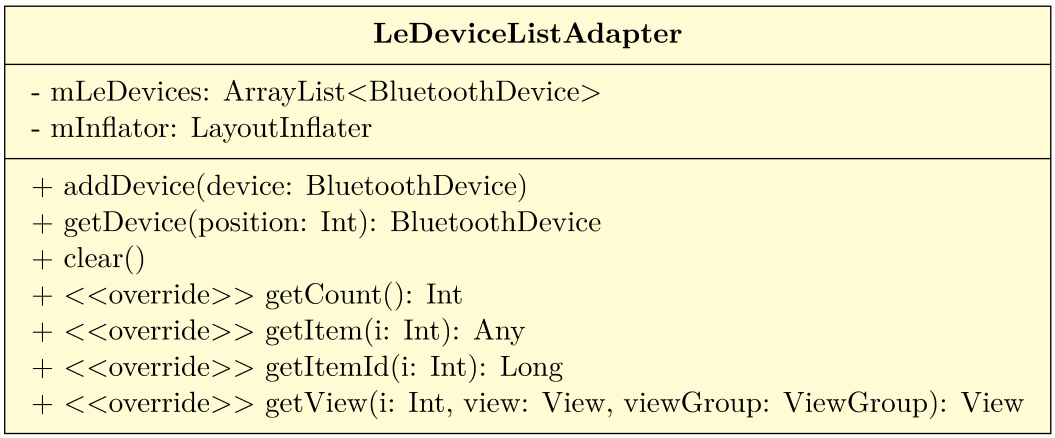


Figura 14 – Diagrama de classe da classe interna LeDeviceListAdapter



- GattAttributes.kt: Esse arquivo contém o objeto **GattAttributes**. O objeto inclui um pequeno conjunto de atributos GATT para uso da aplicação.

- ServiceBLE.kt: Nesse arquivo há a definição da classe **ServiceBLE** e da classe interna **LocalBinder**. A classe ServiceBLE é um *Service* para gerenciar a conexão e transmissão de dados com um *Server* GATT hospedado em um dispositivo BLE dado.

Figura 15 – Diagrama de classe da classe ServiceBLE

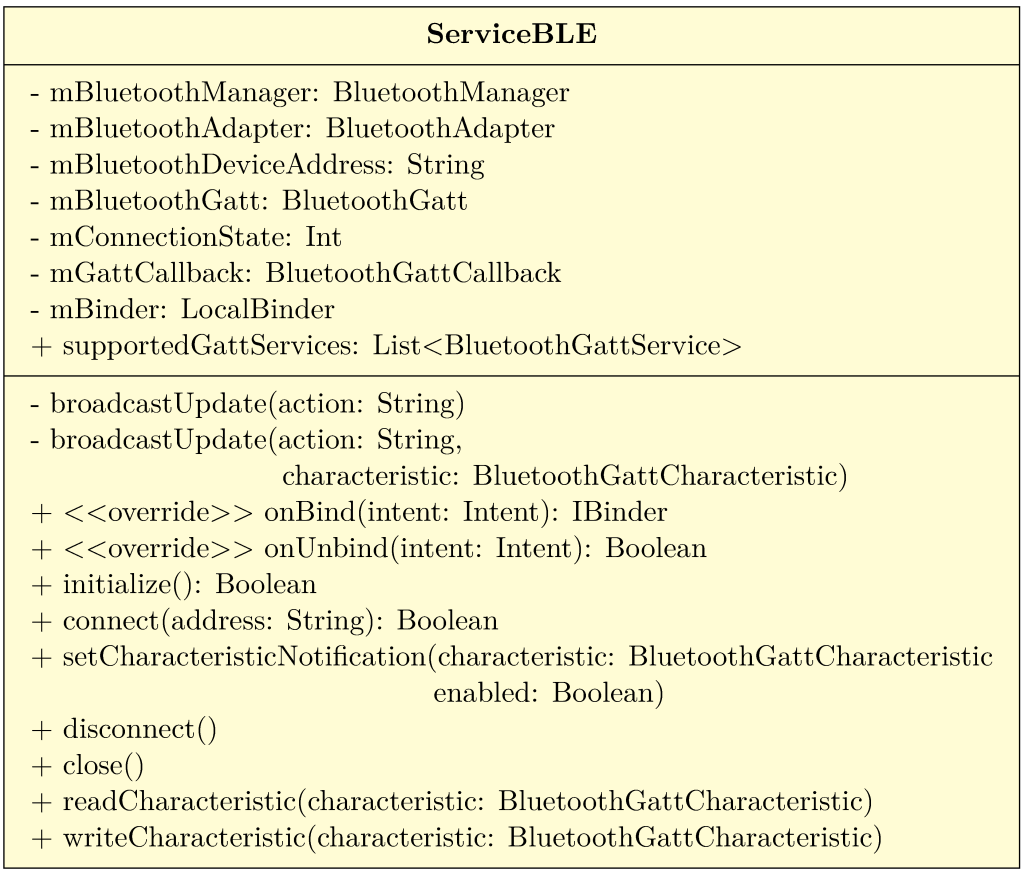
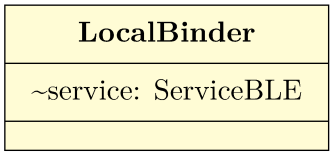


Figura 16 – Diagrama de classe da classe interna LocalBinder



- StateBLE.kt: Nesse arquivo é definida a classe **StateBLE**. A classe StateBLE é uma classe BroadcastReceiver que responde a alterações no estado da conexão BLE.

Figura 17 – Diagrama de classe da classe StateBLE

StateBLE
+ <<override>> onReceive(context: Context, intent: Intent)

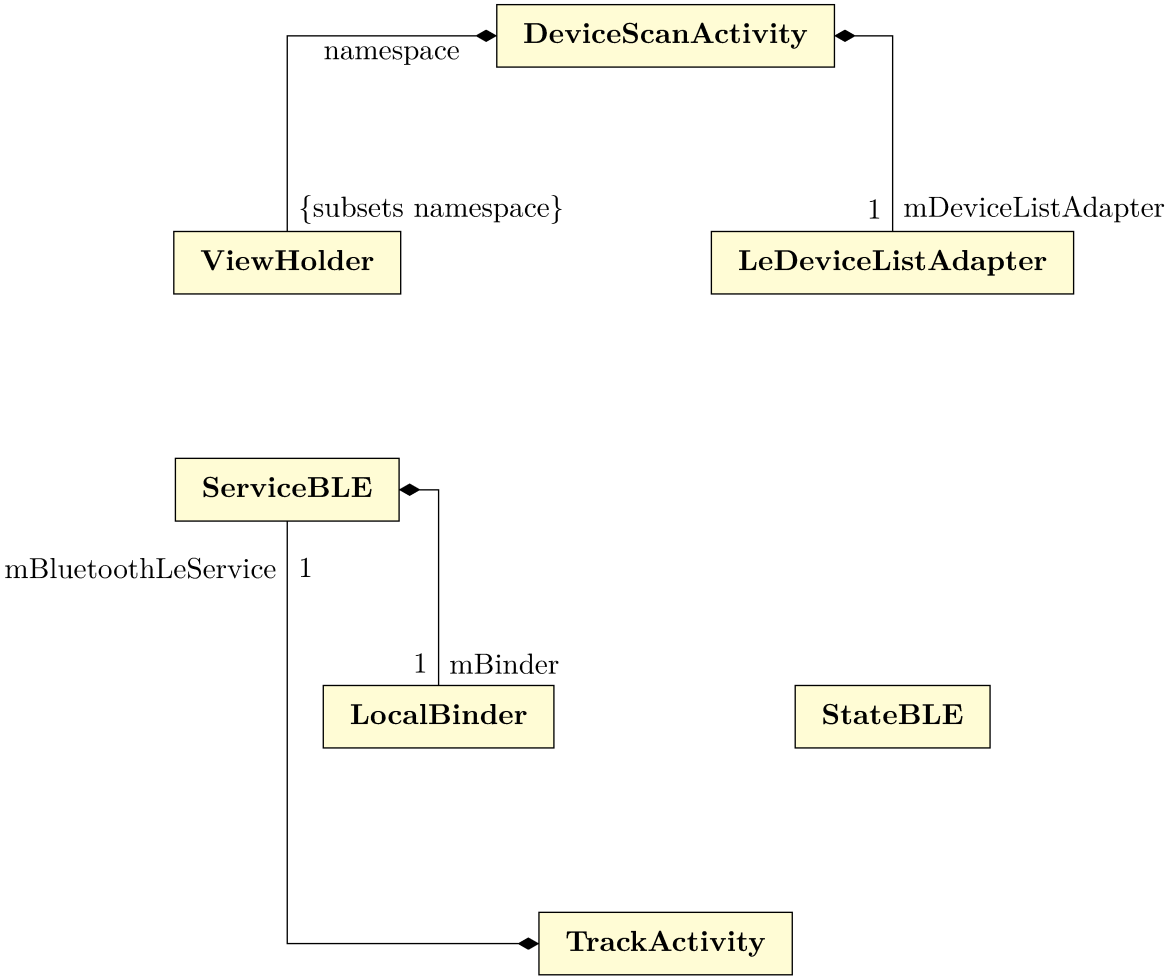
- TrackActivity.kt: Esse arquivo contém a definição da classe **TrackActivity**. A classe TrackActivity é a Activity principal da atividade do ciclista, onde mostra na UI as informações da atividade de maneira dinâmica.

Figura 18 – Diagrama de classe da classe TrackActivity

TrackActivity
- mWheelSizeSpinner: Spinner - mChronometer: Chronometer - mDistance: TextView - mAverageSpeed: TextView - mButtonTrack: Button - mDeviceName: String - mDeviceAddress: String - mBluetoothLeService: ServiceBLE - mConnected: Boolean - mActiveStartTrack: Boolean - characteristicTX: BluetoothGattCharacteristic - characteristicRX: BluetoothGattCharacteristic - mWheelRevolutions: Long - mDialog: Dialog - mServiceConnection: ServiceConnection - mGattUpdateReceiver: BroadcastReceiver
- clearUI() - displayData(data: String) + startTrack(v: View) - getSecondsFromDurationString(value: String): Int - wheelSizeToKm(wheelSizeText: String): Float - displayGattServices(gattServices: List<BluetoothGattService>) + <<override>> onCreate(savedInstanceBundle: Bundle) - showPopup() - makeChange() + <<override>> onResume() + <<override>> onPause() + <<override>> onDestroy() + <<override>> onCreateOptionsMenu(menu: Menu): Boolean + <<override>> onOptionsItemSelected(item: MenuItem): Boolean

Na figura 19 visualiza-se as classes em um diagrama de classes com os atributos e métodos ocultados (as classes da figura podem ser vistas com seus atributos e métodos nas figuras 12 à 18) para facilitar o entedimento global da aplicação:

Figura 19 – Diagrama de classes da aplicação Android



3.2.2 Funcionalidades da aplicação

A aplicação Android que compõe o projeto faz uma listagem na tela inicial dos dispositivos BLE permitidos, como na figura 20, e o usuário deve clicar no item da lista correspondente ao módulo que está acoplado à bicicleta, para parear com este dispositivo e manter a comunicação para a próxima tela, que é a tela de atividade ou TrackActivity (figura 21).

Figura 20 – Tela para escanear por dispositivos bluetooth

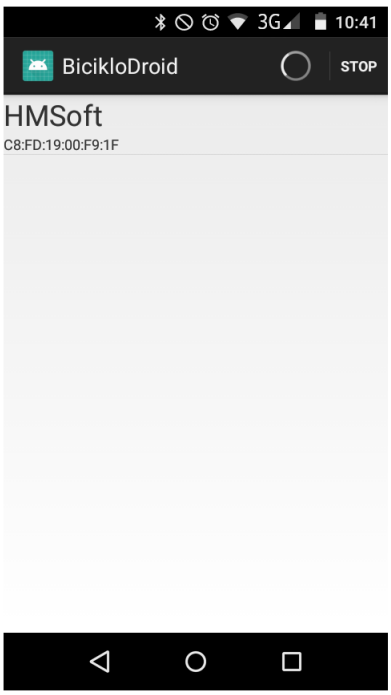
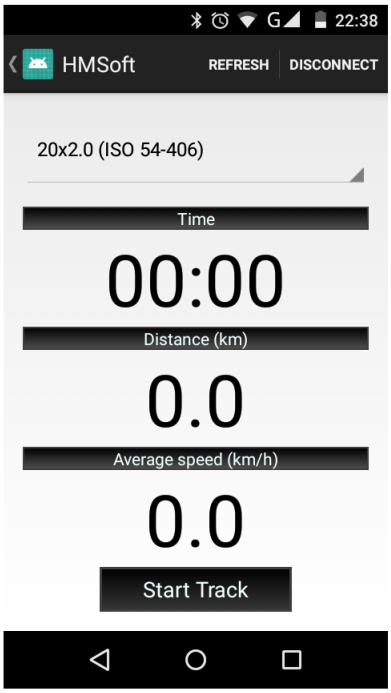


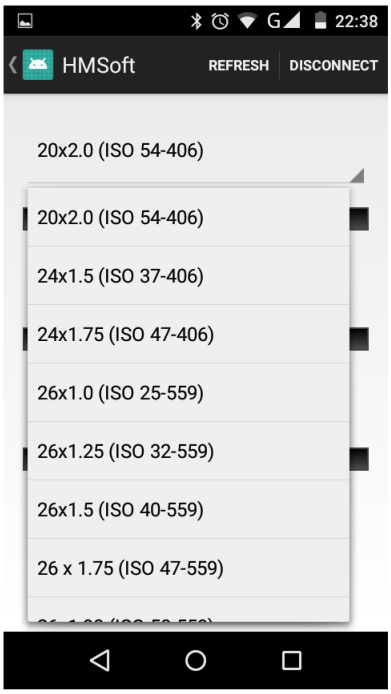
Figura 21 – Esperando começar a registrar a atividade



A tela da atividade contém 3 áreas de informações: (1) área do tempo corrente da atividade, (2) área da distância percorrida até o momento desta atividade, (3) área da velocidade média da atividade. Antes de clicar em começar a atividade, acima dessas

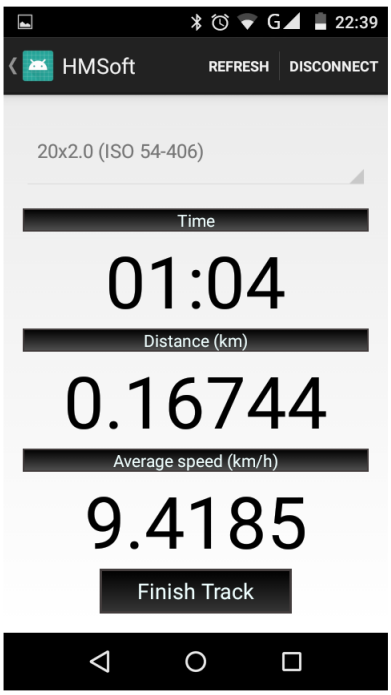
áreas de informações, o ciclista pode escolher o aro da roda em um spinner contendo os tamanhos de rodas mais comuns com base em polegadas (figura 22).

Figura 22 – Escolhendo o tamanho da roda da bicicleta



Ao clicar no botão rotulado “Start Track”, a atividade começa e os dados são atualizados na tela a cada segundo, e não por cada revolução de roda completada, pois se assim fosse, caso o ciclista se mantivesse parado, a velocidade média não estaria concordante com as outras informações exibidas na interface (figura 23).

Figura 23 – Tela de atividade do aplicativo em execução



O ciclista, ao clicar no botão rotulado no instante como “Finish Track”, lhe é apresentado na interface um pop-up, contendo informações da atividade recém-finalizada, além de outros dados como uma foto, nome e endreço, sendo padrões apenas para propósito de apresentação (figura 24).

Figura 24 – Pop-up após finalizar a atividade

4 Resultados

A fixação do sistema embarcado na bicicleta é visto na figura 25. Os componentes do sistema estão firmados ao chainstays da bicileta pelo uso de abraçadeiras de nylon.

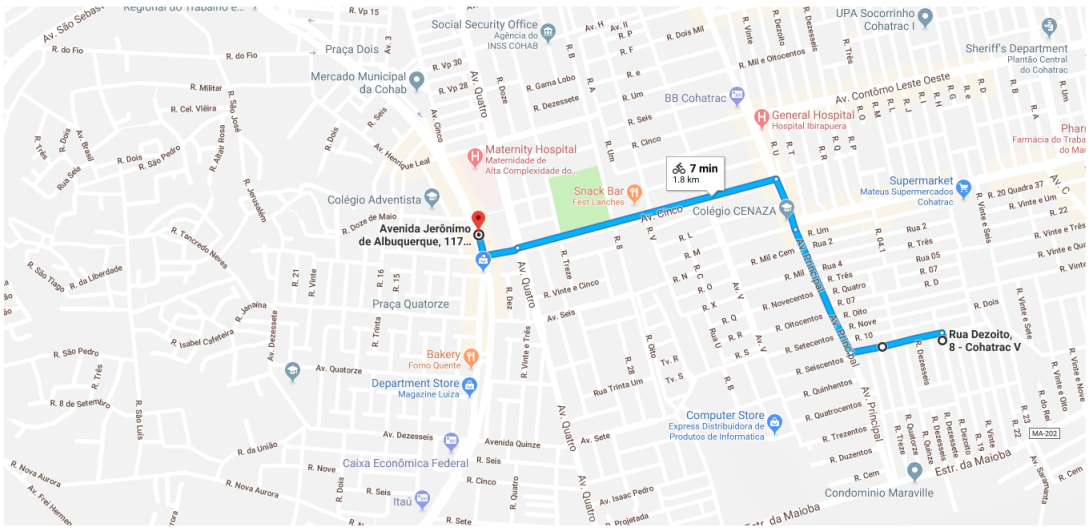
Figura 25 – Sistema embarcado acoplado a uma bicicleta

Fonte: O autor

Para verificar se os resultados que o aplicativo fornece para o ciclista são satisfatórios, primeiramente utilizou-se do Google Maps[1] para obter uma distância aproximada de um trajeto passível de ser seguido com o próprio aplicativo deste trabalho pelo seu autor, e o caminho escolhido para os testes é mostrado na fígura 26.

[1] Google Maps é um serviço web de mapas da empresa Google

Figura 26 – Trajeto para testes do aplicativo



Fonte: O autor (gerado através do Google Maps)

Como informado no site do Google Maps ao traçar este trajeto, tanto o trajeto marcado (que está destacado de azul na figura 26) como os trajetos alternativos destacados de cinza são predominantemente planos.

Para um primeiro teste de verificação da proximidade da distância, foi utilizado o próprio aplicativo desenvolvido, e com os dados da atividade finalizada sendo percorrido o trajeto na figura 26, confronta-se com a distância total do trajeto detalhado na tabela 6.

Tabela 6 – Trajeto detalhado

Direção	Passo	Nome	Distância
↑	1	Siga para o norte na rua dezoito em direção a rua onze	27 m
↰	2	Vire à esquerda para rua onze	280 m
↱	3	Vire à direita na avenida principal	400 m
↑	4	Continue na avenida Contorno Leste	160 m
↰	5	Vire à esquerda na avenida cinco	800 m
↑	6	Continue na avenida dezessete	100 m
↱	7	Vire à direita na avenida jerônimo de albuquerque	66 m

Nota – Informações do trajeto obtidas pelo Google Maps

Com a atividade finalizada do trajeto predefinido, as informações sumarizadas da mesma são apresentadas em um pop-up, como revelado na figura 27:

Figura 27 – Pop-up de finalização da atividade do primeiro teste

Verifica-se neste primeiro teste que a distância obtida com a atividade pelo aplicativo (1,9044 km) condiz com os dados da distância do mesmo trajeto da atividade traçado através do Google Maps (1,833 km), pois a diferença (inferior a 100 metros) é aceitável, dado que o Google Maps utiliza-se de algoritmos para mensurar o trajeto de bicicleta, diferenciando-se do aplicativo, que calcula a trajetória apenas por sensores, então é esperada a diferença na precisão. Não há garantia de acurácia pelo Google Maps em relação às ferramentas de medição de distância, e também não provê qualquer informação relacionada à taxa de erros.

Para o próximo teste, a comparação é feita com relação a outros aplicativos que realizam o controle da atividade de ciclistas por métodos diferentes do usado neste trabalho, não empregando o uso de um sistema embarcado. O primeiro aplicativo se chama “Road Bike” (Runtatic), que utiliza apenas GPS (sem acelerômetro) para o controle das atividades (PUREWAL, 2012). O segundo aplicativo é o “Strava”, que utiliza tanto GPS quanto acelerômetro. O teste foi realizado com o mesmo smartphone com um certo número de atividades para cada aplicativo, e os resultados são apresentados nas tabelas 7 à 9. Além dos testes para comparação, é feito a mensuração da energia da bateria antes e depois de cada atividade com o BicikloDroid, e os valores obtidos encontram-se na tabela 30.

Tabela 7 – Atividades de teste com aplicativo BicikloDroid

Atividade	Duração	Distância	Velocidade média
Atividade 01	08:30	1,898189 km	13,39898 km/h
Atividade 02	08:34	1,902329 km	13,32371 km/h
Atividade 03	07:58	1,89612 km	14,28040 km/h
Atividade 04	08:46	1,90647 km	13,04808 km/h
Atividade 05	08:17	1,894049 km	13,71947 km/h
Atividade 06	08:24	1,902329 km	13,58807 km/h
Atividade 07	08:09	1,898189 km	13,97440 km/h
Atividade 08	07:57	1,898189 km	14,32596 km/h
Atividade 09	07:46	1,902329 km	14,69611 km/h
Atividade 10	09:25	1,908539 km	12,16060 km/h

Fonte – O autor

Tabela 8 – Atividades de teste com aplicativo Road Bike

Atividade	Duração	Distância	Velocidade média
Atividade 01	00:07:46	1,71 km	13,2 km/h
Atividade 02	00:09:19	1,76 km	11,3 km/h
Atividade 03	00:07:40	1,70 km	13,3 km/h
Atividade 04	00:07:49	1,75 km	13,4 km/h
Atividade 05	00:08:11	1,70 km	12,5 km/h
Atividade 06	00:08:45	1,74 km	11,9 km/h
Atividade 07	00:08:43	1,76 km	12,1 km/h
Atividade 08	00:08:12	1,77 km	12,9 km/h
Atividade 09	00:09:56	1,69 km	10,2 km/h
Atividade 10	00:07:26	1,72 km	13,9 km/h

Fonte – O autor

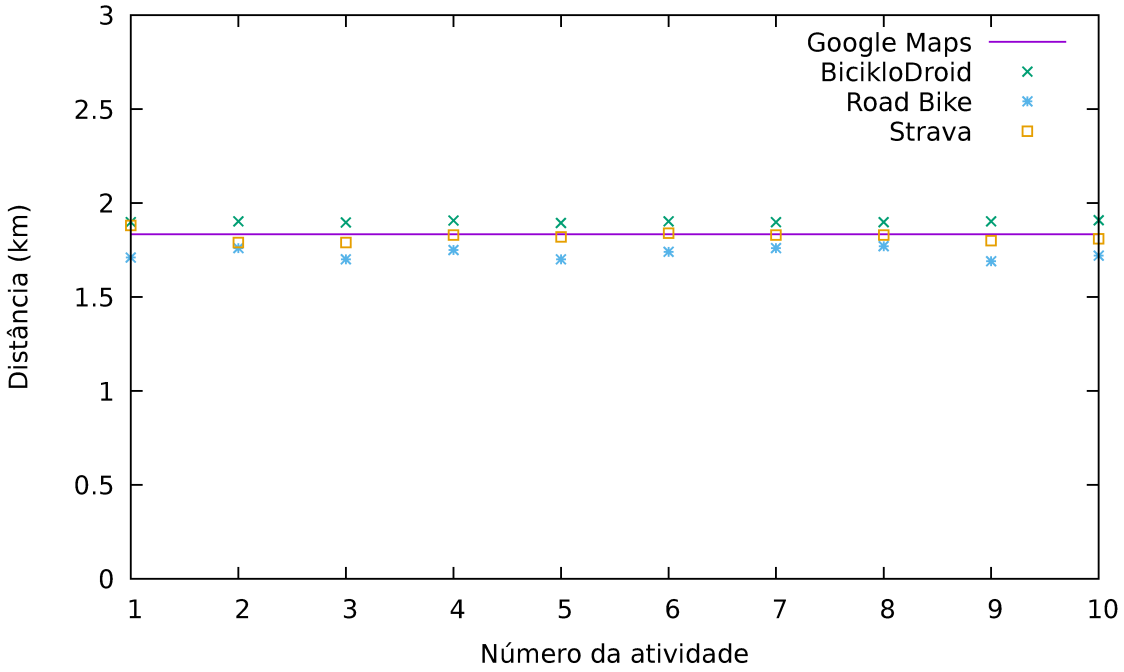
Tabela 9 – Atividades de teste com aplicativo Strava

Atividade	Duração	Distância	Velocidade média
Atividade 01	0:07:50	1,88 km	14,5 km/h
Atividade 02	0:09:44	1,79 km	11,1 km/h
Atividade 03	0:09:06	1,79 km	11,8 km/h
Atividade 04	0:08:37	1,83 km	12,8 km/h
Atividade 05	0:08:52	1,82 km	12,4 km/h
Atividade 06	0:08:05	1,84 km	13,6 km/h
Atividade 07	0:06:49	1,83 km	16,1 km/h
Atividade 08	0:07:10	1,83 km	15,3 km/h
Atividade 09	0:06:48	1,80 km	15,9 km/h
Atividade 10	0:07:01	1,81 km	15,5 km/h

Fonte – O autor

As informações das atividades podem ser visualizadas de maneira gráfica na figura 28, onde a distância da trajetória com maior precisão fornecida pelo Google Maps é utilizada (obtida através da soma total do trajeto na tabela 6 como 1,833 km) na forma de uma linha reta, e os testes com os aplicativos e o sistema deste TCC são representados por pontos no gráfico, com ícones que os diferenciam. Pelo gráfico, percebe-se que as distâncias se mantém próximas a linha da distância do trajeto com Google Maps, tendendo a uma variação menor no Strava e uma elevação na distância com o BicikloDroid, esperada pela questão de o mesmo contabilizar todo o trajeto, até mesmo quando desvios e zigue-zagues são feitos.

Figura 28 – Comparação de distâncias obtidas com Google Maps, BicikloDroid, Road Bike e Strava para o mesmo trajeto



Fonte: O autor

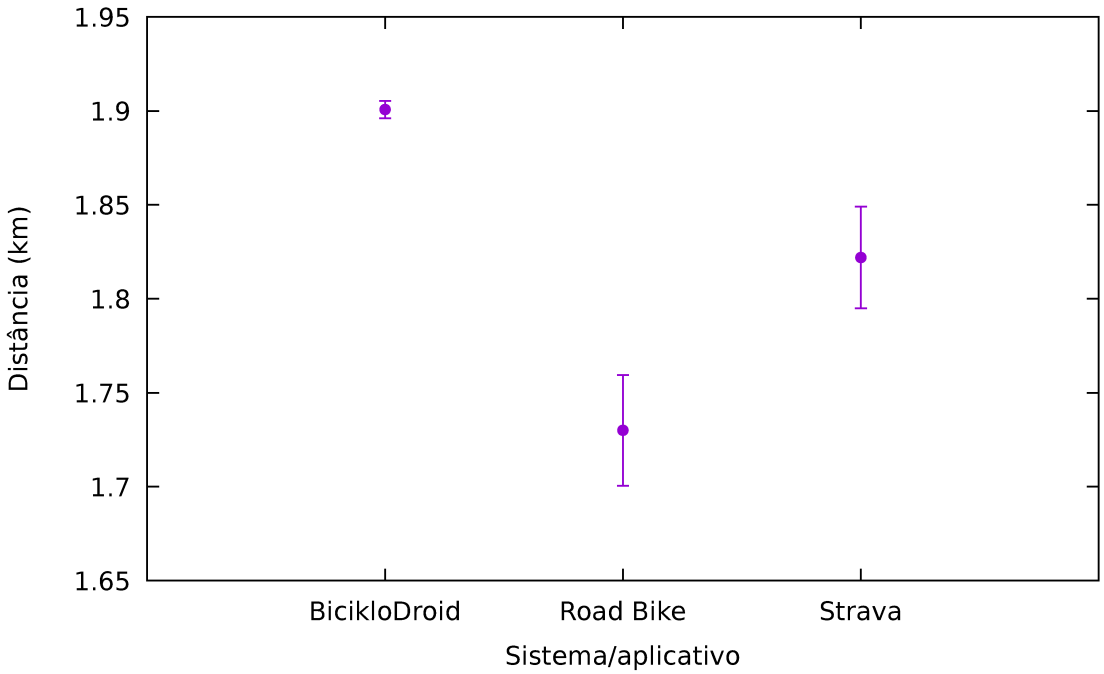
Com os dados das distâncias nas tabelas 7 à 9, obtém-se os dados estatísticos da média e desvio padrão, apresentados na tabela 10, e em gráfico de barras de erro na figura 29.

Tabela 10 – Média e desvio padrão para BicikloDroid, Road Bike e Strava

Sistema/Aplicativo	Média	Desvio padrão
BicikloDroid	1,9007 km	0,0045561 km
Road Bike	1,7300 km	0,029439 km
Strava	1,8220 km	0,026998 km

Fonte – O autor

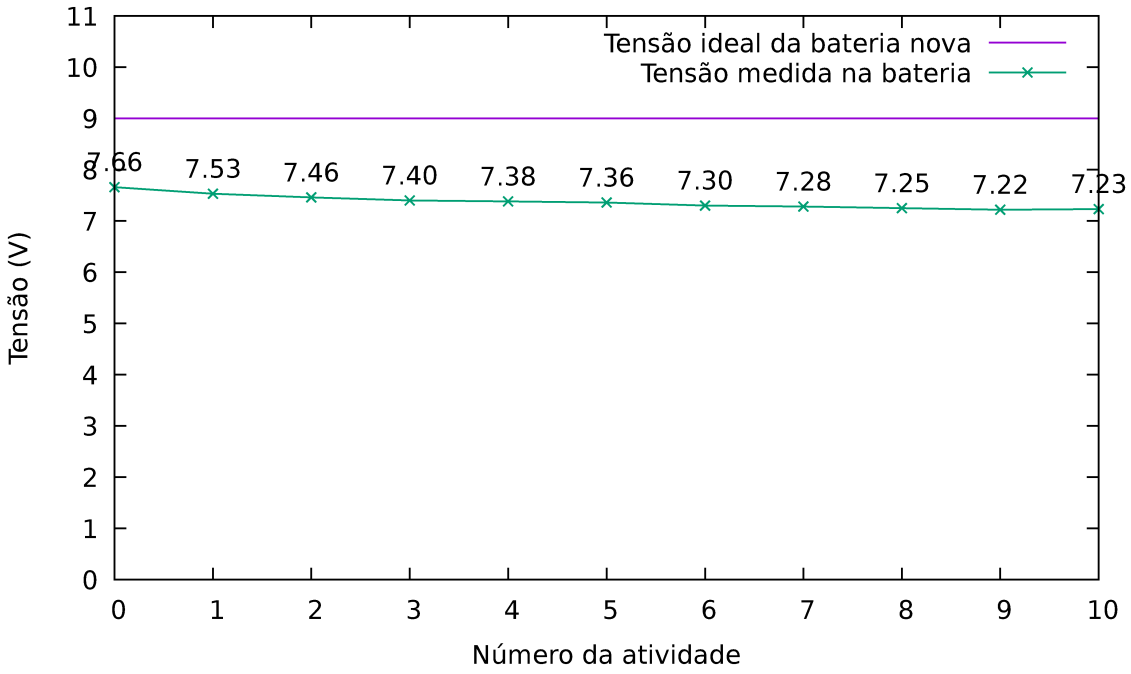
Figura 29 – Gráfico de barras de erro para BicikloDroid, Road Bike e Strava com erro sendo o desvio padrão



Fonte: O autor

Sabendo-se das mensurações da tensão na bateria durante as atividades, contidas na figura 30, sintetiza-se que a redução de tensão na bateria para todas as dez atividades foi de $0,43V$, em um tempo total de 1 hora, 23 minutos e 46 segundos de atividades.

Figura 30 – Tensão na bateria durante as atividades



Fonte: O autor

5 Conclusões e Trabalhos Futuros

O uso de biciletas como meio de transporte impacta na saúde das pessoas e causa danos mínimos ao ambiente. O advento de tecnologias que auxiliem o ser humano nas mais diversas tarefas também alcançou esse veículo, onde a atividade do ciclismo pode ser acompanhada por dispositivos e softwares dedicados e pensados nesta atividade. Com este trabalho busca-se explorar uma alternativa de controle da atividade do ciclista.

O projeto testado, em comparação à aplicativos com a mesma funcionalidade, mostra-se suficientemente satisfatório tratando-se das informações da qual se propõe a mostrar no aplicativo. A informação mais sensível utilizada para calcular os valores na UI é a distância, dado que a velocidade média depende da distância, e a duração é calculada de maneira automática através do componente chronometer do Android. Pelos testes feitos, as comparações com a distância fornecida pelo Google Maps, e as distâncias obtidas com o aplicativo desenvolvido para este trabalho (BicikloDroid) e com outros dois aplicativos (Road Bike e Strava) mostram uma aproximidade das distâncias, que é o que se pretendia verificar, dado que a distância não será exata pela questão do mecanismo de controle de atividades diferir entre os aplicativos e também diferir com relação ao mecanismo de cálculo de trajeto do Google Maps. O uso de componentes menores foi possível e não aumentou a complexidade lógica da aplicação. Não foi possível estimar o consumo de energia do sistema embarcado de maneira precisa, mas alguns apontamentos foram feitos com relação ao uso da bateria.

A aplicação desenvolvida pode ser melhorada, abarcando características não implementadas, como identificar a inatividade do ciclista e pausar a atividade enquanto permanecer nesse estado, com isso otimizando o uso de energia. O acabamento do sistema embarcado no chainstays pode ser trabalhado, até mesmo através de um case em impressora 3D. Os testes podem ser substituídos por outros mais precisos e abrangentes, usando além de aplicativos dedicados ao controle da atividade do ciclista, também dispositivos em conjunto com aplicativos. A estimativa do consumo de bateria também pode ser parte integrante dos testes, utilizando-se de equipamentos apropriados para esse tipo de medição, sendo acoplados ao sistema durante as atividades.

Referências

ABRACICLO. Após estabilidade no ano passado, produção de bicicletas deve aumentar. 2018. Disponível em: <http://www.abraciclo.com.br/2018/1146-apos-estabilidade-no-ano-passado-producao-de-bicicletas-deve-aumentar>. Acesso em: 21 jan. 2018.

ARAÚJO, T. Arduino nano 3.0 - conheça este pequeno e poderoso membro da família arduino. *Fazedores*, Rio de Janeiro, nov. 2014. Disponível em: <http://blog.fazedores.com/arduino-nano-3-0/>. Acesso em: 10 jan. 2018.

BAILEY, D.; GATES, K. *Bike Repair and Maintenance For Dummies*. [S.l.]: John Wiley & Sons, 2009.

BRASIL, P. B. D. M. P. B. B. *Caderno de referência para elaboração de Plano de Mobilidade por Bicicleta nas Cidades*. Brasília: [s.n.], 2007.

CHANG, K.-H. Bluetooth: a viable solution for iot?[industry perspectives]. *IEEE Wireless Communications*, IEEE, v. 21, n. 6, p. 6–7, 2014.

CURREY, M. *HM-10 Bluetooth 4 BLE Modules*. 2018. Disponível em: <https://github.com/abntex/limarka>. Acesso em: 09 jun. 2018.

DEVELOPERS, A. *Activities*: Introduction to activities. 2018. Disponível em: <https://developer.android.com/guide/components/activities/intro-activities>. Acesso em: 13 mai. 2018.

DEVELOPERS, A. *Content Providers*: Overview. 2018. Disponível em: <https://developer.android.com/guide/topics/providers/content-providers>. Acesso em: 08 jun. 2018.

DEVELOPERS, A. *Services*: Overview. 2018. Disponível em: <https://developer.android.com/guide/components/services>. Acesso em: 08 jun. 2018.

EVANS, B. *Beginning Arduino Programming*. [S.l.]: Apress, 2011.

GIL, A. C. *Como Elaborar Projetos de Pesquisa*. 4. ed. São Paulo: Atlas, 2002.

JNHUAMAO TECHNOLOGY COMPANY. *Bluetooth 4.0 BLE module Datasheet*. D-4020, Qilu soft zone Jinan city,Shandong, China, 2014. 41 p.

KOTLINLANG.ORG. *Kotlin Language Documentation*. [S.l.], 2017.

LEE, W.-M. *Beginning android 4 application Development*. Indianapolis, IN 46256: John Wiley & Sons, 2012.

MACLEAN, D.; KOMATINENI, S.; ALLEN, G. *Pro Android 5*. [S.l.]: Apress, 2015.

MEI, Z.; WANG, D.; CHEN, J. Investigation with bluetooth sensors of bicycle travel time estimation on a short corridor. *International Journal of Distributed Sensor Networks*, SAGE Publications Sage UK: London, England, v. 8, n. 1, p. 7, 2012.

MISKON, M. T. et al. Fitness cycling device with graphical user interface based on ieee 802.15. 4 transceiver for real time monitoring. *Journal of Applied Environmental and Biological Sciences*, v. 4, n. 12, p. 108–114, 2014.

NANO, P. A. *Arduino Nano (V3.0)*: User manual. [S.l.], 2009.

OLIVEIRA, D. S.; AFONSO, J. A. A smartphone-based multi-sensor wireless platform for cycling performance monitoring. In: WORLD SCIENTIFIC. *IAENG TRANSACTIONS ON ENGINEERING SCIENCES: Special Issue for the International Association of Engineers Conferences 2015*. [S.l.], 2017. p. 322–335.

PUCHER, J.; DILL, J.; HANDY, S. Infrastructure, programs, and policies to increase bicycling: an international review. *Preventive medicine*, Elsevier, v. 50, p. S106–S125, 2010.

PUREWAL, S. J. *Review: Runtastic's mobile apps make tracking a workout easier*. 2012. Disponível em: <https://www.pcworld.com/article/2017159/review-runtastics-mobile-apps-make-tracking-a-workout-easier.html>. Acesso em: 11 jun. 2018.

SIG, I. B. *GATT Characteristics | Bluetooth Technology Website*. 2018. Disponível em: <https://www.bluetooth.com/specifications/gatt/characteristics>. Acesso em: 09 jun. 2018.

SIG, I. B. *GATT Descriptors | Bluetooth Technology Website*. 2018. Disponível em: <https://www.bluetooth.com/specifications/gatt/descriptors>. Acesso em: 09 jun. 2018.

SIG, I. B. *GATT Overview | Bluetooth Technology Website*. 2018. Disponível em: <https://www.bluetooth.com/specifications/gatt/generic-attributes-overview>. Acesso em: 08 jun. 2018.

SIG, I. B. *GATT Services | Bluetooth Technology Website*. 2018. Disponível em: <https://www.bluetooth.com/specifications/gatt/services>. Acesso em: 09 jun. 2018.

SILVA, O.; OLIVEIRA, V.; VANZIN, M. Mybike: Um computador de bordo para biciletas de baixo custo. Manaus, 2014.

TOWNSEND, K. et al. *Getting started with Bluetooth low energy: tools and techniques for low-power networking*. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., 2014.

APÊNDICE A – Revisão direcionada da linguagem de programação Kotlin

A.1 Obtendo dados de entrada do usuário

Através do uso da função **readLine**, é possível ler uma linha de entrada do stream padrão, tendo em essência a mesma funcionalidade do **cin** em C++ e do **scanf** em C, fazendo uma analogia para os familiarizados com essas linguagens.

Código A.1 – GetUserInput.kt

```
fun main(args: Array<String>){
  print("Qual é o seu nome? ")
  val name = readLine()
  println("Olá " + name)
}
```

Abaixo um exemplo de execução do programa A.1:

```
Qual é o seu nome? Rafael
Olá Rafael
```

A.2 Funções

A seguir é explicado como definir funções em Kotlin através de alguns exemplos práticos. A função a seguir é definida com o nome soma, com dois parâmetros do tipo Int, e que tem como retorno um valor Int.

```
fun soma(a: Int, b: Int): Int {
  return a + b
}
```

Pode-se reparar que a sintaxe da linguagem em alguns aspectos é semelhante a sintaxe de um pseudo-código, com a declaração do tipo vindo depois do nome da variável.

A próxima função em funcionalidade é equivalente a função anterior, mas sua sintaxe difere pois seu retorno é implicito, inferido pelo retorno da função, sendo utilizado o operador = neste caso.

```
fun soma(a: Int, b: Int) = return a + b
```

Também é possível ter o retorno implícito como mostrado abaixo:

```
fun soma(a: Int, b: Int) = a + b
```

Toda função em Kotlin retorna um objeto, ainda que este objeto não contenha nenhum valor. Quando uma função não deve retornar nenhum valor/objeto (que seja utilizado pela aplicação), se retorna implicitamente um objeto **Unit** ou pode-se retornar explicitamente:

```
fun printSoma(a: Int, b: Int): Unit {
  println("Soma de $a e $b é ${a + b}")
}
```

```
fun printSoma(a: Int, b: Int) {
  println("Soma de $a e $b é ${a + b}")
}
```

A.3 Variáveis

Para usar váriaveis em Kotlin, declaram-se com as palavras-chave **val** ou **var**, sendo o primeiro a declaração de uma variável imutável, ou seja, uma variável que após ser atribuido um valor a esta, este valor armazenado pela variável não poderá ser alterado durante a execução do programa. Já com **var**, a variável é mutável e pode ser modificada durante o programa. A seguir um programa que mostra como pode-se utilizá-las.

Código A.2 – ValeVarUso.kt

```
fun main(args: Array<String>){

  // usando vaviáveis mutáveis

  val a: Int = 1
  val b = 2
  val c: Int
  c = 3

  // usando variáveis imutáveis

  var x = 5
  x += 1
}
```

A.4 Comentários

Comentários em Koltin podem ser de uma linha, ou de múltiplas linhas.

```
// Isto é um comentário
val = 5 // outro comentário de linha após uma declaração

/* Isto é um comentário de
   múltiplas linhas */
```

A.5 Interpolação de string

Expressões template servem para concatenar valores e expressões armazenadas em outros lugares no programa dentro da string.

```
val nome = "José"
val idade = 52
println("${nome} completará ${idade + 1} anos em alguns dias.")
```

A.6 Expressões condicionais

Exemplo de uso de estrutura **if** em Kotlin:

```
val maria = 42
val joao = 33

fun idadeMaxima(a: Int, b: Int): Int {
  if(a > b){
    return a
  } else {
    return b
  }
}

println("A maior idade entre João e Maria é ${idadeMaxima(joao,
   ↪ maria)}")
```

Também tem-se a estrutura condicional **when**, com um exemplo mostrado abaixo, que usa a palavra-chave opcional **else**, que é alcançada quando todas as outras cláusulas não forem verdadeiras:

```
val idade = 8

when(idade) {
  0,1,2,3,4 -> println("Pré-escola")
```

```
  5 -> println("Jardim de infância")
  in 6..17 -> {
    val serie = idade - 5
    println("Série $serie")
  }
  else -> println("Faculdade")
}
```

A.7 Valores nulificáveis

Em Kotlin, pode-se especificar que uma variável ou retorno de função receba algum valor do seu tipo especificado como também receba **null** (caso de uma variável) ou retorne do tipo especificado ou nulo (no caso de funções), mas só é possível com a utilização do uso de interrogação no final do tipo da variável ou retorno conforme mostra-se no exemplo a seguir:

Código A.3 – ValoresNulificaveis.kt

```
fun main(args: Array<String>){
  var variavelNulificavel: String? = null

  // comment: Exception in thread "main"
  // println(variavelNulificavel!!)

  var variavelNulificavel2 = retornaNulo()

  if(variavelNulificavel2 != null){
    println(variavelNulificavel2.length)
  }

  // comment: Exception in thread "main"
  // var variavelNulificavel3 = variavelNulificavel2!!.length

  var variavelNulificavel4: String = retornaNulo() ?: "Qualquer
    ↪ nome"
}

  fun retornaNulo(): String? = null
```

A.8 Estruturas de repetição

Kotlin contém as estruturas de repetição **for**, **while**, e **do while**.

Código A.4 – EstruturasRepeticao.kt

```
fun main(args: Array<String>)
{
  for(i in 0..9)
    print("${i + 1}")

  var x = 0

  while(x < 10)
  {
    print("${x + 1}")
    x += 1
  }

  var j = 0

  do
  {
    print("${j + 1}")
    j += 1
  }while(j < 10)
}
```

Saída do código A.4:

```
12345678910123456789101234567891O
```

A.9 Ranges

Código A.5 – RangesExemplos.kt

```
fun main(args: Array<String>)
{
  val zeroAteDez = 0..10
  val alfabeto = "a".."z"
  val dezAteUm = 10.downTo(1)
  val doisAte10 = 2.rangeTo(10)
  val rangeComPasso3 = zeroAteDez.step(3)
```

```
  for(numero in zeroAteDez)
    print("${numero} ")

  println("r em alfabeto: ${"r" in alfabeto}")
  for(x in rangeComPasso3) println("rangeComPasso3: $x")
  for(x in dezAteUm.reversed()) println("dezAteUm invertido: $x")
}
```

```
0 1 2 3 4 5 6 7 8 9 10 r em alfabeto: true
rangeComPasso3: 0
rangeComPasso3: 3
rangeComPasso3: 6
rangeComPasso3: 9
dezAteUm invertido: 1
dezAteUm invertido: 2
dezAteUm invertido: 3
dezAteUm invertido: 4
dezAteUm invertido: 5
dezAteUm invertido: 6
dezAteUm invertido: 7
dezAteUm invertido: 8
dezAteUm invertido: 9
dezAteUm invertido: 10
```

A.10 Coleções

Na linguagem Kotlin, com relação a coleções, se faz distinção entre coleções mutáveis e coleções imutáveis (KOTLINLANG.ORG, 2017). Pode-se declarar **List**, **Set** e **Map** tanto mutáveis como imutáveis, e consegue-se especificar uma visualização apenas de leitura de uma coleção mutável, conforme o código mostrado a seguir (adaptado de kotlinlang.org (2017)):

Código A.6 – ColecoesExemplos.kt

```
fun main(args: Array<String>){
  val numeros: MutableList<Int> = mutableListOf(1, 2, 3, 4)
  val viewApenasDeLeitura: List<Int> = numeros
  println(numeros)
  numeros.add(5)

  // comment: --- error: unresolved reference: clear ---
```

```
  // viewApenasDeLeitura.clear()

  println(viewApenasDeLeitura)
}
```

Saída do código A.6:

```
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
```

A.11 Classes e herança

A declaração de uma classe em Kotlin usa a palavra chave **class** e essa declaração consiste do nome da classe, do cabeçalho especificando outras informações como parâmetros de tipo e herança, e também consiste de um corpo de classe, que é limitado por abre e fecha chaves. Ambos cabeçalho e corpo da classe são opcionais (KOTLINLANG.ORG, 2017).

```
class NameClass {

}
```

Em Kotlin, uma classe pode ser daclarada com um construtor primário e vários construtores secundários. A seguir, a declaração de uma classe com um construtor primário:

```
class Person constructor(firstName: String){

}
```

Quando o construtor não tem modificadores de visibilidade e anotações, a palavra-chave **constructor** pode ser omitida:

```
class Person(firstName: String) {

}
```

A inicialização de um construtor primário acontece pelo uso de um bloco com a palavra-chave **init**:

Código A.7 – InicializandoConstrutorPrimario.kt

```
fun main(args: Array<String>) {
  val jose = Person("José")
}
```

```
class Person(firstName: String) {

  init {
    val customerKey = firstName
    println("Cliente inicializado com nome: ${customerKey}")
  }
}
```

Saída do código em A.7:

```
Cliente inicializado com nome: José
```

Note que no código acima declara-se um atributo chamado customerKey e inicializa-se com o parâmetro passado para o construtor. Há um modo mais conciso de fazer essa inicialização de atributos:

```
class Person(val customerKey: String) {

  init {
    println("Customer initialized with value ${customerKey}")
  }
}
```

A palavra-chave **open** deve ser usada para que uma classe possa ser a classe base de outra classe, ou seja, por padrão as classes são como as classes que contém **final** em Java, não podendo ter classes derivadas desta.

Código A.8 – ClassesHeranca.kt

```
fun main(args: Array<String>) {
  val bowser = Animal("Bowser", 20.0, 13.5)
  bowser.getInfo()

  val spot = Dog("Spot", 20.0, 14.5, "Paul Smith")
  spot.getInfo()
}

open class Animal(val name: String, var height: Double, var
   ↪ weight: Double){
  init {
    require(height > 0){"Altura deve ser maior que zero"}
    require(weight > 0){"Peso deve ser maior que zero"}
  }

  open fun getInfo(): Unit {
```

```
    println("$name tem altura $height e peso $weight")
  }
}

class Dog(name: String, height: Double, weight: Double, var owner
   ↪ : String): Animal(name, height, weight) {
  override fun getInfo(): Unit {
    println("$name tem altura $height, peso $weight e pertence a
       ↪ $owner")
  }
}
```

```
Bowser tem altura 20.0 e peso 13.5
Spot tem altura 20.0, peso 14.5 e pertence a Paul Smith
```

A.12 Interfaces

Interfaces em Kotlin podem conter tanto métodos abstratos como métodos com implementação, atributos abstratos, e se não abstratos, que implementem modificadores de acesso. Não é necessário o uso da palavra-chave **abstract** nos métodos sem corpo na interface, pois por padrão são reconhecidas como **abstract** na interface.

Código A.9 – InterfacesExemplo.kt

```
fun main(args: Array<String>) {
  val d = D()
  d.foo()

  val c = C()
  c.foo()
}

interface A {
  fun foo() { print("A") }
  fun bar()
}

interface B {
  fun foo() { print("B") }
  fun bar() { print("bar") }
}
```

```
class C : A {
  override fun bar() { print("bar") }
}

class D : A, B {
  override fun foo() {
    super<A>.foo()
    super<B>.foo()
  }

  override fun bar() {
    super<B>.bar()
  }
}
```

No código A.9, a classe C deve implementar a função bar que é abstract em A, enquanto a classe D que implementa as interfaces A e B deve sobrecarregar ambas as funções, bar e foo, mesmo que a função foo esteja implementada em ambas as interfaces.

A.13 Modificadores de visibilidade

Há quatro modificadores de visibilidade em Kotlin: **public**, **private**, **protected** e **internal** (KOTLINLANG.ORG, 2017). Estes modificadores podem ser aplicados em contextos diferentes, mas para este trabalho é de maior interesse seu contexto com relação a membros de classe. A tabela 11 sumariza suas informações.

Tabela 11 – Modificadores de visibilidade para membros de classe.

Modificador	Descrição
public	Visível para qualquer cliente que veja a classe declarada
protected	Visível dentro da própria classe e subclasses
private	Visível apenas dentro da classe
internal	Vísivel para qualquer cliente dentro do mesmo módulo

Fonte – Adaptado de kotlinlang.org (2017)

A.14 Classes enum

Classes enum são declaradas com as palavras chaves **enum class** antes do nome da classe enum:

```
enum class Clima {
```

```
    ENSOLARADO, CHUVOSO, VENTOSO
}
```