Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Sistema embarcado em conjunto com aplicativo Android para monitoramento do desempenho e percurso do ciclista

Charles Cardoso de Oliveira

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Engenharia de Computação

Orientador
Prof. Dr. Eduardo Peixoto Fernandes da Silva

Brasília
2015

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Engenharia de Computação

Coordenador: Prof. Dr. Ricardo Zelenovsky

Banca examinadora composta por:

Prof. Dr. Eduardo Peixoto Fernandes da Silva (Orientador) — ENE/UnB
Prof. Dr. Ricardo Zelenovsky — ENE/UnB
Prof. Me. José Edil Guimarães de Medeiros — ENE/UnB

CIP — Catalogação Internacional na Publicação

Oliveira, Charles Cardoso de.

Sistema embarcado em conjunto com aplicativo Android para monitoramento do desempenho e percurso do ciclista / Charles Cardoso de Oliveira. Brasília : UnB, 2015.
131 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2015.

1. bicicleta, 2. sistema embarcado, 3. microcontrolador, 4. aplicativo Android, 5. bluetooth

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Sistema embarcado em conjunto com aplicativo Android para monitoramento do desempenho e percurso do ciclista

Charles Cardoso de Oliveira

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Engenharia de Computação

Prof. Dr. Eduardo Peixoto Fernandes da Silva (Orientador)
ENE/UnB

Prof. Dr. Ricardo Zelenovsky
ENE/UnB

Prof. Me. José Edil Guimarães de Medeiros
ENE/UnB

Prof. Dr. Ricardo Zelenovsky
Coordenador do Bacharelado em Engenharia de Computação

Brasília, 09 de dezembro de 2015

Dedicatória

Dedico esse trabalho a minha família que sempre me apoiou durante todo o período em que estive nesta instituição.

Agradecimentos

Agradeço a minha família por sempre ter me apoiado nas minhas escolhas.

Agradeço ao orientador Prof. Dr. Eduardo Peixoto Fernandes da Silva que passou dois semestres me auxiliando desde a implementação do projeto até a fase de testes. Também ao Prof. Me. José Edil Guimarães de Medeiros e Prof. Dr. Ricardo Zelenovsky que mesmo indiretamente contribuiram para que esse projeto fosse finalizado. Além de todos os outros professores que são responsáveis por grande parte do conhecimento que adquiri durante esse curso.

Também agradeço aos meus colegas de turma que tornaram a vida acadêmica mais interessante, sendo também responsáveis por contribuir com o aprendizado através de longas discussões sobre engenharia.

Resumo

O projeto consiste em um sistema para auxiliar o ciclista durante o seu percurso. Através desse sistema, o usuário poderá ver seu trajeto no mapa, cadência e velocidade da bicicleta, tempo de percurso, entre outras medidas.

O trabalho proposto pode ser dividido em duas partes: sistema embarcado e aplicativo Android. O primeiro é um circuito elétrico utilizando um microcontrolador MSP430 e sensores *reed switch*, usados para medir a velocidade e cadência de uma bicicleta, enquanto o último, é um software para smartphone que além de mostrar esses dados, implementa novas funcionalidades, como: mapas, GPS e persistência de estatísticas. A comunicação entre os dois é feita através de bluetooth, utilizando o módulo HC-05, sendo que este abstrai a complexidade do protocolo, transformando a interface do microcontrolador em comunicação serial.

Os testes foram divididos em etapas. A primeira etapa foi feita isolando cada unidade, e apenas na etapa final um teste prático que empregava todos os componentes do projeto. Os resultados foram satisfatórios, sendo que o teste prático demonstrou que o sistema proposto está muito próximo de um produto popular de mercado.

Palavras-chave: bicicleta, sistema embarcado, microcontrolador, aplicativo Android, bluetooth

Abstract

This work proposes and implements a system to aid cyclists, showing his current and average speed and cadence, as well as track time and the area map.

The proposed project can be divided in two parts: an embedded system and an Android application. The former is an electric circuit using a MSP430 microcontroller and reed switch sensors, used to measure the speed and cadence of the bicycle, while the latter is a software for smartphone which shows these data and implements new features, such as maps, GPS and statistics persistence. The two parts communicate via bluetooth, using an off-the-shelf HC-05 module.

The tests were divided in stages. Each unit that composes the system was tested on its own, isolating it from all other components. Then, the system as a whole was tested. The results were satisfactory, and the practical test showed that the proposed system performance is close to a popular market product.

Keywords: bicycle, embedded system, microcontroller, Android application, bluetooth

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O ciclismo vem ganhando espaço no cotidiano das pessoas, sendo não apenas uma opção de lazer, mas também um importante meio de transporte diário. Segundo a pesquisa Origem e Destino do metrô, aplicada na Região Metropolitana de São Paulo, o uso desse tipo de deslocamento aumentou 18% entre 1997 e 2008.[1]

Além disso, nas cidades ao redor do mundo, nota-se um crescimento popular dos sistemas de compartilhamento de bicicletas, que são estações onde as pessoas podem retirar as bicicletas, e após o uso, devolvê-las na mesma ou outra estação. Esse tipo de meio de transporte tem sido visto como uma opção barata, eficiente e saudável. Há, aproximadamente, 450 desses sistemas funcionando no mundo inteiro.[2]

Portanto, o uso de bicicletas está ganhando mais espaço como alternativa de transporte, e dentro desse cenário, soluções para auxiliar o ciclista durante o seu percurso, seja através de mapas e GPS ou até mesmo mostrando sua velocidade instantânea, possui potencial para ser um importante item ao ciclismo.

No mercado já existem diversas soluções para monitorar parâmetros do ciclista durante o seu trajeto. Por exemplo:

- Aplicativos: são softwares, geralmente para *smartphone*, que utilizam GPS para mostrar localização, trajeto do ciclista e até mesmo uma velocidade instantânea aproximada. **Minhas trilhas**[3] e **Strava**[4] são exemplos desse tipo de aplicativo. **Vantagens**: baixo custo e sem necessidade de *hardware* adicional. **Desvantagens**: A aproximação do GPS pode não ser suficiente para ciclistas mais exigentes e não é possível medir a cadência da bicicleta.

- Computadores de bordo: trata-se de sistemas específicos, cujo *hardware* e *software* foi projetado apenas para o auxílio do ciclista. A **Garmin**[5] possui uma grande variedade desse tipo de solução. **Vantagens**: são mais precisos e por ser um *hardware* específico para essa atividade terá atributos como, tela menos reflexiva, por exemplo. **Desvantagens**: Custo maior e um equipamento a mais para ser transportado.

O objetivo deste projeto é incorporar as vantagens dos dois exemplos citados, isto é, a precisão do computador de bordo que possui um *hardware* específico para calcular certas medidas e a praticidade e baixo custo do *smartphone*. Com isso em mente, tem-se um sistema embarcado que calcula o tempo de revolução da roda e do pedal da bicicleta (necessários para a medida da cadência e velocidade) em conjunto com um aplicativo Android – *software* para *smartphone* que será toda a interface do usuário com o sistema.

1.1 O sistema proposto

Um sistema embarcado é uma solução baseada em um microprocessador, cujo objetivo é gerenciar uma função ou um conjunto delas. Sendo que essas funções podem ser gerenciadas em simples eletrodomésticos ou até mesmo em um conjunto de máquinas industriais.[6] Tipicamente, um sistema embarcado é composto por:

- Um microcontrolador para prover a inteligência;
- Circuitos que fazem interface com a aplicação principal;
- *Software* de tempo real;
- *Hardware* dedicado para funções nos casos em que a implementação por *software* pode se tornar lenta;
- *Hardware* de testes e manutenção.

Neste trabalho, o sistema embarcado tem a função de calcular, em tempo real, a duração da revolução da roda e do pedal de uma bicicleta usando sensores *reed switch* como interface. Esses dados são enviados – via comunicação bluetooth utilizando o módulo HC-05 – para um smartphone com aplicativo Android, sendo que a implementação deste também faz parte do projeto. O aplicativo utiliza esses valores recebidos para calcular, diretamente, a velocidade e a cadência da bicicleta e, indiretamente, outras informações úteis ao ciclista. Além disso, o *software* implementado para o smartphone possui outras funcionalidades envolvendo mapas, GPS e persistência de estatíticas, portanto, sendo uma ótima opção para ser usado durante vários percursos pelos ciclistas.

A figura 1.1 mostra um diagrama simplificado do projeto desenvolvido. Ímãs são fixados nos alvos do projeto: pedal e roda. O campo magnético produzido por eles é captado pelo sensor *reed switch*. Sendo assim, o microcontrolador MSP430 precisa, resumidamente[1], medir o tempo entre duas interrupções do sensor, isto é, calcular a duração da revolução. Feito isso, esses valores são enviados para o módulo HC-05 através de uma interface serial, sendo que este tem o papel de converter os dados em comunicação bluetooth. Por último, os dados calculados no MSP430 chegam no smartphone e são submetidos a outros tipos de processamento, sendo finalmente, mostrados ao usuário.

Os testes conduzidos foram divididos em etapas. Os primeiros foram feitos isolando cada unidade, com o objetivo de testar aspectos, como: precisão do cristal, algoritmo do microcontrolador e comunicação *bluetooth*. Por fim, um teste prático envolvendo todos os componentes foi feito e os resultados mostraram que o sistema proposto tem potencial para se tornar uma solução para o ciclista que deseja monitorar o seu percurso.

1.2 Estrutura do trabalho

Este trabalho tem como objetivo cobrir a teoria necessária e a implementação dos dois componentes do projeto: sistema embarcado e aplicativo Android. O conteúdo está organizado da seguinte forma:

[1] Um algoritmo mais robusto é implementado para lidar com alguns casos, isso será explicado no decorrer deste documento

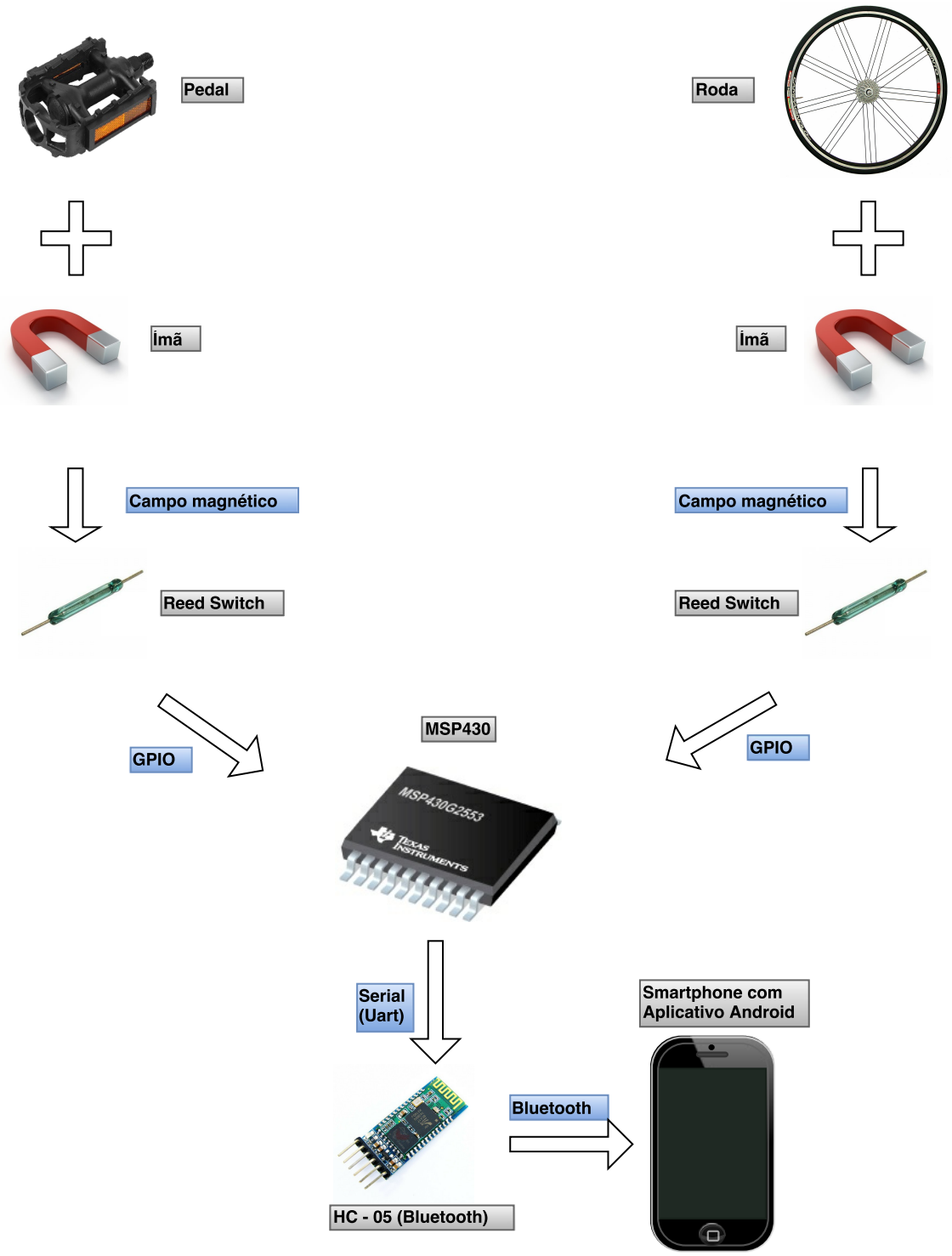

Figura 1.1: Diagrama simplificado do projeto proposto

- **Fundamentos teóricos:** este capítulo tem como objetivo mostrar a teoria necessária para o entendimento do projeto. O meio de comunicação bluetooth, o sistema operacional Android, o microcontrolador MSP430 e os sensores são alguns dos tópicos que serão abordados.
- **Sistema embarcado: implementação:** descreve os aspectos mais importantes na implementação do sistema embarcado, que tem como função medir o tempo de revolução da roda e do pedal usando sensores *reed switch* e enviar esses valores ao *smartphone*. Para facilitar a explicação do sistema descrito, o capítulo é dividido em duas seções: circuito elétrico e algoritmo utilizado no MSP430. Como sugerem os nomes, abordam questões de projeto para a implementação do *hardware* e *software* que formam o sistema embarcado.
- **Aplicativo Android: implementação:** este capítulo mostra o processo de implementação do aplicativo Android para o *smartphone*. Algumas seções abordam aspectos gerais de qualquer *software*, arquitetura, padrões de projeto e banco de dados, por exemplo. Outras são mais específicas e abordam aspectos tecnológicos do sistema Android, como por exemplo, *bluetooth*, mapas e *shared preferences*.
- **Resultados:** descreve os principais testes feitos no sistema embarcado e aplicativo Android, sendo que essa descrição mostra quais os objetivos de cada teste e os resultados obtidos deles.
- **Conclusões e trabalhos futuros:** este capítulo resume os principais pontos do trabalho e mostra que o projeto tem potencial para ser uma alternativa ao ciclista que deseja monitorar seu percurso, levando em consideração os resultados obtidos. Por fim, mostra melhorias que podem ser incorporadas para tornar o projeto em um produto real.

Capítulo 2

Fundamentos teóricos

Este capítulo tem como objetivo mostrar a teoria necessária para o entendimento do projeto. O meio de comunicação bluetooth, o sistema operacional Android, o microcontrolador MSP430 e os sensores são alguns dos tópicos que serão abordados ao decorrer do capítulo.

Primeiramente, é apresentado o meio de comunicação *bluetooth*, um breve histórico, como funciona e suas características e vantagens em relação aos outros meios de comunicação.

Nas seções seguintes, uma delas é dedicada ao sistema operacional Android, mostrando elementos de sua arquitetura e componentes principais de um aplicativo. Já na outra seção, são abordados os padrões de projeto fundamentais no desenvolvimento do aplicativo Android no projeto.

Nas últimas seções, o foco são os elementos do sistema embarcado. Uma seção abordando o microcontrolador utilizado na implementação do sistema embarcado, o MSP430: arquitetura, componentes principais e elementos de programação necessários para o desenvolvimento do software embarcado. E finalmente, a última seção fala sobre o módulo HC-05 e o sensor *reed switch*, essenciais no circuito elétrico.

2.1 Bluetooth

Bluetooth é um padrão de tecnologia que possibilita comunicação sem fio em pequenas distâncias entre dispositivos eletrônicos a partir de ondas de rádio. Criado pela empresa sueca Ericsson em 1994, foi originalmente concebido como uma alternativa sem fio a cabos de dados RS-232.[7]

Essa tecnologia tornou-se muito utilizada em vários tipos de dispositivos. Entre eles, estão: os **dispositivos inteligentes** (computadores, celulares, *tablets*), **periféricos** (teclados, *joysticks*, câmeras, fones de ouvido) e **aplicações embarcadas** (travas elétricas para automóveis, sistemas industriais).[8]

A comunicação do bluetooth é feita através de uma banda de frequência ISM (*Industrial*, *Scientific*, *Medical*), que opera em 2,4 GHz (não licenciada). Isso permite ser uma tecnologia usada mundialmente. Além disso, utiliza um esquema FHSS (Frequency hopping spread-spectrum), que permite o "salto de frequência" entre vários canais. Em muitos países há 79 canais disponíveis, enquanto alguns possibilitam apenas 23 canais. Como existem vários dispositivos utilizando a banda ISM, um dispositivo com bluetooth

não pode operar num canal por mais que 0,4 segundo dentro de qualquer período de 30 segundos, isso minimiza as interferências.[8]

Para atender a diversos tipos de aplicações, o alcance do bluetooth é dividido em três classes[1]:[9]

- **Classe 3**: alcance de 1 metro.
- **Classe 2**: alcance de 10 metros – encontrada principalmente em dispositivos móveis. A classe predominante, funciona com apenas 2,5 mW de potência.
- **Classe 1**: alcance de 100 metros – usada principalmente em aplicações de sistemas industriais.

2.1.1 Consumo

Bluetooth foi projetado para ser uma tecnologia de baixo consumo de energia. Isso é reforçado na especificação, permitindo que a transmissão de rádio seja desligada quando inativo.[9]

A tabela 2.1 mostra uma comparação entre Bluetooth e Wi-Fi (outra tecnologia de comunicação sem fio):

Tabela 2.1: Comparativo de consumo de energia entre Bluetooth e Wi-Fi

	Bluetooth	Wi-Fi
Potência de saída típica	1 – 10 mW	30 – 100 mW
Corrente absorvida típica	1 – 35 mA	100 – 350 mA

Fonte: Adaptado de [10]

Essa é uma das principais vantagens do bluetooth, por isso é altamente utilizado em aplicações portáteis em que a vida útil da bateria é limitada.

2.1.2 Rede bluetooth: a piconet

Um dispositivo bluetooth pode operar no modo **mestre** (*master*) ou **escravo** (*slave*). Um máximo de oito dispositivos – sete escravos ativos e um mestre – podem trabalhar juntos, formando uma piconet. O mestre é responsável por selecionar uma frequência, a sequência de "saltos de frequência" entre canais e o *timing* dos escravos. A figura 2.1 mostra possíveis configurações de uma piconet.

Piconets podem se conectar com outras, assim formando uma *scatternet*. Nessa configuração, um escravo pode fazer parte de mais de uma piconet, enquanto um mestre pode pertencer apenas a uma única piconet. Piconets se comunicam por meio de um nó comum entre elas. Um nó pode se comportar como mestre em uma piconet e escravo em várias outras. Na figura 2.2, por exemplo, o nó "M/S" é comum a duas piconets, fazendo papel de mestre na piconet esquerda e sendo escravo na piconet direita.

[1] Dispositivos de diferentes classes podem se comunicar, desde que se respeite o limite da classe de menor alcance.

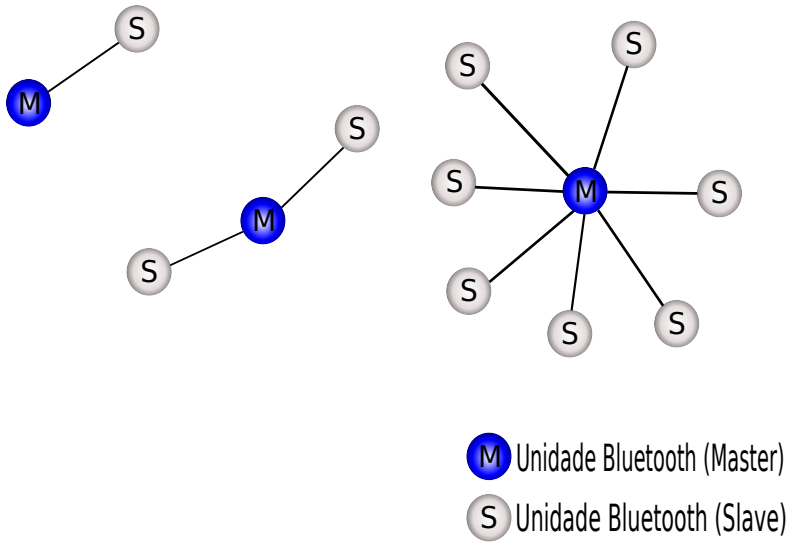

Figura 2.1: Algumas configurações de piconets. Adaptado de: [11]

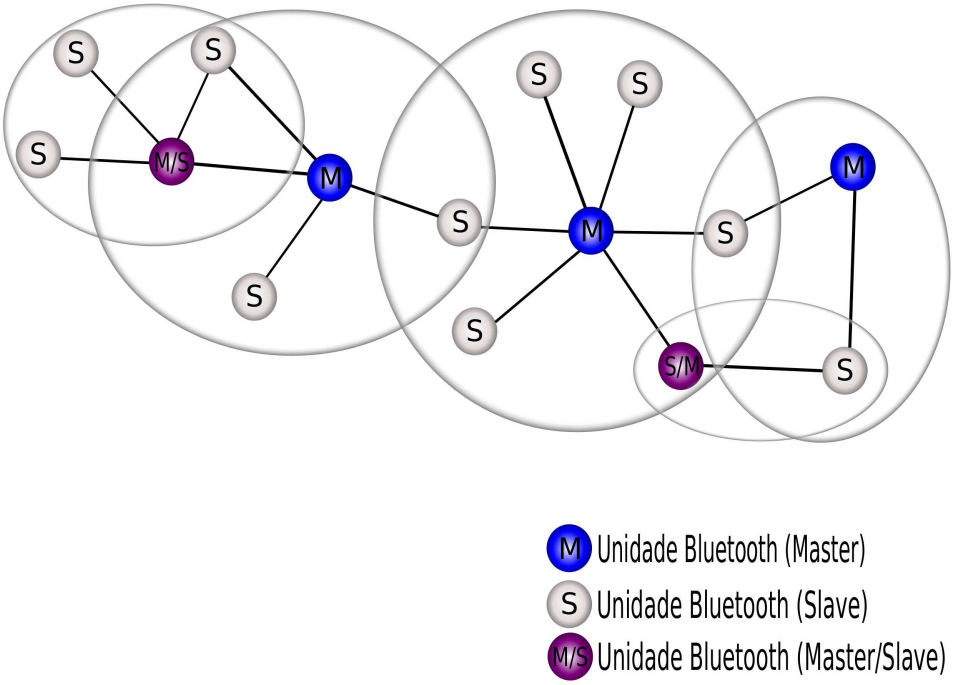

Figura 2.2: Conexão entre piconets formando uma *scatternet*. Adaptado de: [11]

2.2 Android

Android é um sistema operacional *open source* para dispositivos móveis – smartphones, tablets, relógios – baseado em linux. Teve seu desenvolvimento iniciado pela empresa Android Inc. Em 2005, a empresa foi adquirida pelo Google, responsável por liderar o desenvolvimento do sistema operacional atualmente.[12]

Figura 2.3: Logomarca do Android. Fonte: [12]

A primeira versão beta do Android Software Development Kit (SDK) – kit de desenvolvimento para o sistema operacional Android – foi lançada pelo Google em novembro de 2007. Já a primeira versão comercial, Android 1.0, foi lançada apenas em setembro de 2008.[13]

A figura 2.4 mostra as camadas da arquitetura do sistema operacional Android. O nível de abstração aumenta da base (*kernel*) para o topo da figura (aplicações). Começando pela base, tem-se:

- **Linux Kernel**: Como dito anteriormente, o Android é baseado no sistema operacional linux. Essa é a camada responsável pelos *drivers* do sistema e não interage com o usuário.
- **Libraries e Android Runtime**: Contém as bibliotecas nativas do sistema. SQLite – usado como repositório para armazenamento de dados –, SSL (*Security Socket Layer*) – segurança em aplicações que utilizam internet – são exemplos nessa camada. No mesmo nível está o Android Runtime que possui a **Máquina Virtual Dalvik** que possibilita a cada aplicação Android o seu próprio processo.
- **Application Framework**: É a camada responsável por gerenciar recursos utilizados pelas aplicações de usuário.
- **Applications**: E por último a camada de aplicação. Navegadores, agenda de contatos, câmera, calculadora e vários outros estarão presentes aqui.

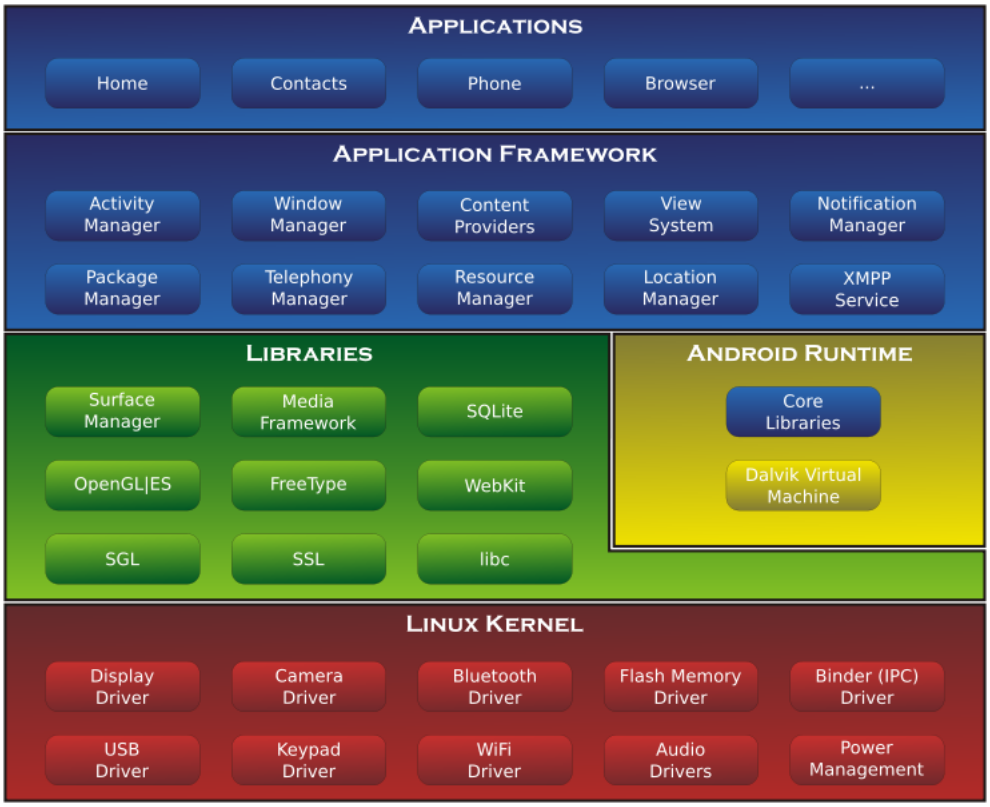

Figura 2.4: Arquitetura do sistema operacional Android. Fonte: [14]

2.2.1 Aplicativos: fundamentos

A grande maioria dos aplicativos (apps) Android são escritos usando a linguagem de programação Java. O código em Java e outros recursos – imagens, ícones, etc. – são compilados usando a ferramenta Android SDK. O produto disso é um arquivo APK (Android Package) que poderá ser utilizado para instalar a aplicação no dispositivo com o sistema Android.[15]

Um app pode possuir quatro tipos de componentes essenciais (não é necessário que um app tenha todos os quatros componentes), cada componente tem diferentes propósitos e seu próprio ciclo de vida, isto é, fases que ele passa da criação até ser destruído durante a execução do aplicativo. Esses componentes são:[15]

- **Activities**: Uma activity representa uma única tela com uma interface de usuário. Por exemplo, um aplicativo de e-mail pode ter uma activity para ler e-mails, outra para mostrar a lista de e-mails e outra para mandar novos e-mails.
- **Services**: Um service é um componente que executa em *background* para realizar operações de longa duração ou executar trabalhos para processos remotos. Um service não possui interface com usuário. Como exemplo, um service pode ser uma música sendo reproduzida enquanto o usuário vê sua letra na tela do dispositivo. O importante de um service é não bloquear a interação do usuário com a activity.
- **Content Providers**: Um content provider gerencia um conjunto de dados do aplicativo. O aplicativo pode armazenar dados em arquivos, no banco de dados SQLite, na Web ou outro meio de persistência. Através do content provider, outros aplicativos podem consultar ou até mesmo modificar esses dados (se isso for permitido pelo content provider). Ele também será útil mesmo que seu aplicativo não deseje compartilhar dados, isto é, apenas queira salvar dados privados no dispositivo. Por exemplo, o próprio sistema Android fornece um content provider que gerencia as informações de contato do usuário. Logo, qualquer aplicativo com as devidas permissões pode consultar parte do content provider para ler ou escrever informações sobre um contato particular.
- **Broadcast Receivers**: Um broadcast receiver é um componente que responde a eventos propagados pelo sistema operacional. Podem ser gerados pelo próprio sistema – por exemplo, um broadcast que anuncia que a bateria está fraca. Os aplicativos também podem iniciar um broadcast – por exemplo, para comunicar a outros dispositivos que alguns dados já foram baixados e estão prontos para uso.

Um aspecto positivo do Android é o fato de aplicativos poderem utilizar componentes de outros. Por exemplo, se um aplicativo deseja utilizar a câmera do dispositivo para tirar uma fotografia, muito provavelmente haverá outro aplicativo que já implementa essa funcionalidade. Portanto, pode-se utilizar desse componente em vez de criar uma nova activity para isso.[15]

2.3 Padrões de projeto

No desenvolvimento de um *software* – um aplicativo Android, por exemplo – pode surgir problemas que se repetem em vários outros programas. Nesse contexto, uma boa

alternativa é recorrer a soluções padronizadas para problemas já conhecidos e recorrentes. Essas soluções são conhecidas como padrões de projeto, e por serem tão difundidas, essas soluções recebem nomes específicos.

2.3.1 Model-View-Presenter

O padrão MVP (Model-View-Presenter) é um padrão de projeto que tem como objetivo separar as regras de negócio dos elementos de interface do usuário. Além disso, ainda permite desacoplar a própria lógica da interface do usuário, que se torna independente da tecnologia utilizada para apresentação.[16] Em resumo, tem -se:

- **View:** Responsável por renderizar o *model* para o usuário.
- **Presenter:** Lida com as interações do usuário e a partir disso manipula o *model* e atualiza a *view*. Logo, é o intermediário entre *view* e *model*.
- **Model:** Mantém os dados da aplicação e possui métodos para acessá-los.

A *view* conhece e mantém a instância do *presenter*, porém não tem nenhum conhecimento do *model*. O *presenter* se comunica com a *view* através de uma interface, garantindo assim que a lógica fique independente da implementação da apresentação.[16]

2.3.2 Injeção de dependência

No paradigma de programação orientada a objetos, tem-se o padrão de projeto conhecido como: **injeção de dependência**. Trata-se de um padrão que tira a responsabilidade de uma classe de criar suas próprias dependências, isto é, inversão de controle. Logo, as dependências de uma classe serão criadas e injetadas por outra parte do software.

Há dois métodos mais comuns para injetar dependências, são eles:

- **Construtores:** Na instância de uma classe, as dependências são passadas via construtor.
- **Setters:** Em qualquer momento, um método *setter* pode ser chamado para atribuir a dependência.

Sua principal vantagem está no desacoplamento dos módulos do software. Se uma classe não tem conhecimento em como criar sua dependência, alterações feitas nesta, não afetará diretamenta aquela classe. Outro benefício é a possibilidade de fazer testes unitários, em que *mocks*[2] poderão ser injetados, garantindo o isolamento do teste.

2.4 MSP430

MSP430 é uma família de microcontroladores produzida pela Texas Instruments, projetados especificamente para serem baratos e possuírem baixo consumo de energia – requisito fundamental em sistemas embarcados. Possui compilador para linguagem C, portanto

[2]Mocks são objetos "falsos" que imitam os verdadeiros e possuem um comportamento controlado, são utilizados para isolar uma unidade

pode ser programado em uma linguagem de alto nível, o que garante maior produtividade. O foco deste trabalho é o MSP430G2553 (pertencente a família MSP430G2) e ele será usado como exemplo em situações que se deseja apresentar definições mais específicas.

O MSP430 incorpora uma CPU RISC (conjunto reduzido de instruções) de 16 bits, periféricos e um sistema de *clock* flexível. A memória faz parte da arquitetura von-Neumann, isto é, dados e instruções compartilham uma mesma região de memória.[17]

A figura 2.5 mostra o diagrama de blocos do MSP430G2x53. Destacando alguns componentes e tendo como base os valores específicos do MSP430G2553, tem-se:[17]

- **Clock system**: O sistema de *clock* foi projetado principalmente para aplicações de baixo consumo de energia. Em que o ACLK (auxiliary clock) é conectado, geralmente, a um cristal de 32768 Hz – XIN e XOUT no diagrama. O SMCLK (sub-main clock) pode ser selecionado via software e pode auxiliar tarefas secundárias, como transmissão serial. E por último, o MCLK (Master clock) que possui alta velocidade (até 16 MHz) e é utilizado pela CPU.
- **CPU**: Onde as instruções são executadas. É alimentado pelo *Master clock*.
- **JTAG** e **Spy-Bi-Wire**: Ambos são interfaces para programar a memória *flash* do microcontrolador.
- **Memória flash** e **RAM**: O bloco de memória flash é utilizado para persistência (software, por exemplo) e a memória RAM para informações voláteis, possuindo 16 KB e 512 B, respectivamente.
- **ADC**: Um conversor analógico para digital com até 10 bits de precisão.
- **GPIO's**: Três portas com propósitos gerais.
- **Brownout protection**: É um módulo que reinicia o dispositivo quando a tensão de entrada está em níveis perigosos. Por exemplo, quando a alimentação é aplicada ou removida do circuito.
- **Watchdog**: Um timer de 16 bits que tem como função primária reiniciar o sistema se algum problema ocorrer no software.
- **Timer's**: Dois timers que devem ser alimentados com um clock (ACLK ou SMCLK, por exemplo) e possuem uma ampla variedade de configurações.
- **Módulo serial**: Implementa três tipos de comunicação serial, que são: UART, I2C e SPI.
- **MAB** e **MDB**: Refere-se aos barramentos de instruções (Memory Address Bus) e dados (Memory Data Bus), respectivamente.

2.4.1 Launchpad

O *launchpad* é um *kit* de desenvolvimento que permite a programação e o *debug* do MSP430 de forma fácil, através da interface USB. A figura 2.6 mostra um *launchpad* da família MSP430G2, entre suas características estão: *push button*, led's, pinos extras de alimentação, soquete para encaixe do microcontrolador.

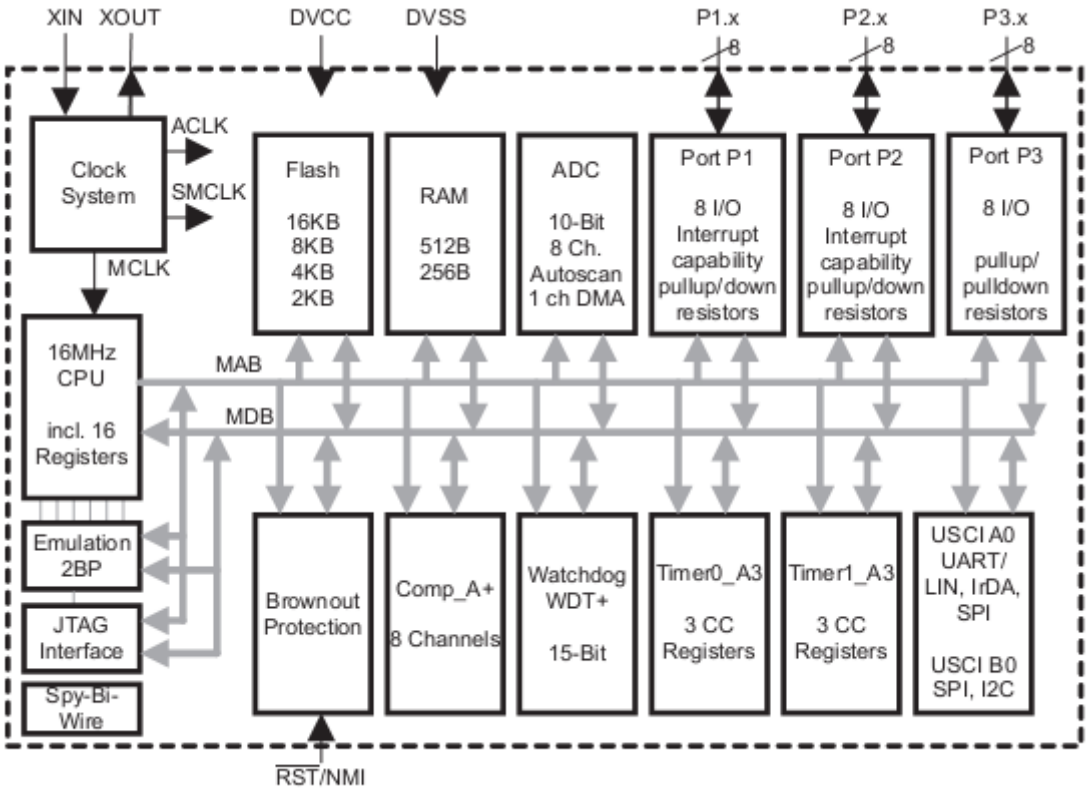

Figura 2.5: Diagrama de Bloco do MSP430G2x53. Fonte: [18]

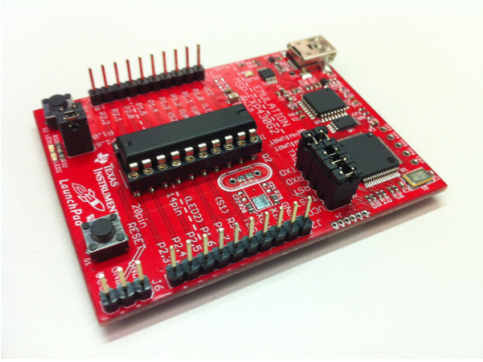

Figura 2.6: Launchpad MSP430G2. Fonte: [19]

Portanto, quando o programa já estiver funcionando adequadamente, pode-se retirar o microcontrolador do *launchpad* e colocá-lo numa placa de circuito impresso ou *protoboard*, por exemplo.

2.4.2 Programando o MSP430

As discussões a seguir são baseadas no livro [20] e nos manuais da Texas Instruments [17] e [18].

Timer's no MSP430

Os timer's no MSP430 possuem uma grande quantidade de parâmetros de configuração. O que pode ser muito flexível em vários casos, pode também ser confuso em determinadas situações. Existem dois tipos de timer's, timer_A e timer_B, o foco desse trabalho será o timer_A (as diferenças são pequenas, logo é possível com uma pequena leitura adicional aprender o timer_B).

O timer_A possui um registrador de 16 bits responsável pela contagem do tempo, chamado TAR – tem a capacidade de contar de 0 a 0xFFFF. São possíveis quatro modos de operação, descritos na tabela:

Tabela 2.2: Modos de operação do timer_A

Modo	Descrição
Stop	O timer é interrompido.
Up	O timer repetidamente conta de zero até o valor em TACCR0.
Continuous	O timer repetidamente conta de zero até 0xFFFF.
Up/down	O timer conta repetidamente de zero até o valor em TACCR0 e de TACCR0 até zero.

Fonte: Adaptado de [17]

O **TACCR0** é conhecido como registrador de capture/compare zero. Há mais dois registradores de capture/compare, porém o TACCR0 é o que possui maior prioridade. O valor colocado nele será o máximo contado no modo **up** e **up/down**.

Como no MSP430 a maioria das configurações são feitas através de registradores, é importante conhecer os principais responsáveis por definir o comportamento do timer utilizado. Os principais registradores são mostrados a seguir:

- **TACTL, registrador de controle**: através desse registrador é possível definir o clock fonte (TACLK, ACLK, SMCLK ou INCLK), um divisor para o clock fonte (1, 2, 4 ou 8) e o modo de operação (*stop*, *up*, *continuous*, *up/down*). Além disso, pode-se zerar o conteúdo do TAR e habilitar interrupções.
- **TAR, registrador de contagem**: o contador do timer_A.
- **TACCRx, registrador x capture/compare**: no **modo captura**, o valor de TAR será copiado para esse registrador. No **modo comparação**, TACCRx armazena o dado para a comparação do valor de TAR.

- **TACCTLx, registrador de controle capture/compare**: nesse registrador é possível definir se será utilizado o modo captura e em qual borda será feita (subida, descida ou ambas), o sinal específico para fazer a captura (geralmente especificado para qual pino do microcontrolador) e se isso será sincronizado. Além disso, terá as configurações de output (OUT bit value, Set, Toggle/reset, Toggle, Reset, Toggle/set, Reset/set), caso o modo pretendido seja comparação. E por último, tem-se acesso a interrupções pendentes.
- **TAIV, registro do vetor de interrupções**: armazena as interrupções pendentes, como TACCR1, TACCR2 e TAIFG (timer overflow). A interrupção do TACCR0 não está nesse registrador, pois possui uma *flag* especial no registrador TACCTLx.

O trecho de código a seguir, mostra um exemplo de programa utilizando alguns dos registradores apresentados:

```
1  /* O exemplo a seguir, coloca um led no pino P1.0
2  para mudar de estado (ligado, desligado) a cada
3  1 segundo */
4
5  #include <msp430.h>
6
7  int main(void) {
8      WDTCTL = WDTPW | WDTHOLD;    /*  Watchdog timer */
9      P1DIR |= BIT0; /* Define P1.0 como pino de output */
10
11     BCSCTL1 |= DIVA_0; /* Clock do cristal de 32768 Hz */
12     BCSCTL3 |= XCAP_3;
13
14     TA0CCR0 = 4096 - 1;
15     TA0CTL |= TASSEL_1 | ID_3 | MC_1;
16     TA0CCTL0 |= CCIE;
17
18     while(1){
19         __low_power_mode_0();
20     }
21 }
22
23 #pragma vector = TIMER0_A0_VECTOR
24 __interrupt void interrupt_timer0(){
25     P1OUT ^= BIT0; /* Toggle led */
26     __low_power_mode_off_on_exit();
27 }
```

As instruções responsáveis por configurar o timer estão nas **linhas 14, 15 e 16**. Na **linha 15** é onde configura-se o registrador de controle. As opções podem ser vistas com detalhes no datasheet[17], nesse exemplo a configuração foi a seguinte:

- **TASSEL_1**: ACLK alimenta o timer. Nesse exemplo, ACLK é obtido de um cristal de 32768 Hz (linhas 11 e 12).
- **ID_3**: Indica que o clock será dividido por 8, portanto, tem-se uma frequência de 4096 Hz.
- **MC_1**: *Up mode*, tendo o TA0CCR0 (linha 14) como limite.

Portanto, com uma frequência de 4096 Hz e contando de 0 até 4095, tem-se um intervalo de exatamente 1 segundo entre interrupções.

Um outro método de uso dos timers nesse microcontrolador é através de captura de eventos. O **modo captura** é utilizado em situações onde se necessita medir o período de tempo entre eventos, entre a subida e descida da borda de um sinal, por exemplo. Em geral, isso deve ser feito usando um clock de frequência muito superior (deve ser uma frequência conhecida), que garante uma boa precisão da medida. A figura 2.7 mostra esse esquema:

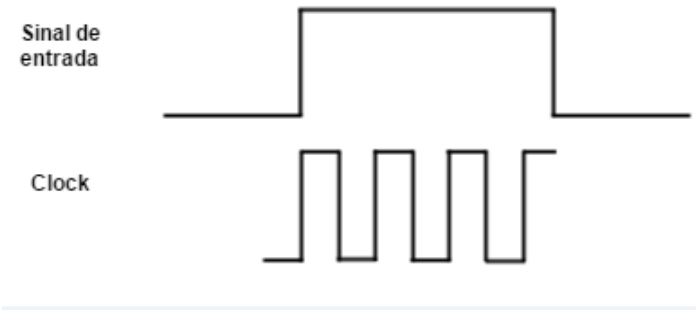

Figura 2.7: Medida de um período usando o modo captura

Como o clock possui uma frequência conhecida, o microcontrolador consegue medir o período entre a subida e descida da borda do sinal de entrada, simplesmente contando o número de períodos feito pelo clock. A precisão do período do sinal de entrada capturado será aumentada a medida que se diminui o período do clock.

O sinal de entrada deve ser selecionado no registrador TACCTLx através dos bits CCISx – essas entradas estarão conectadas a pinos externos ou sinais internos, o *datasheet* específico deve ser consultado. No registrador TACCTLx, ainda é possivel escolher qual a borda de captura (subida, descida ou ambos) e se o sinal de captura será sincronizado com o *clock* utilizado (uma prática recomendável para evitar condição de corrida).[17]

Quando uma captura ocorre, o valor do registrador TAR é copiado para o registrador TACCRx e a flag de interrupção CCIFG é ativada. Como o TAR é finito, isto é, conta até 65535 em *continuous mode* ou até TACCR0 no *up mode*, se não houver nenhuma captura nesse intervalo, ocorrerá um *overflow* que deverá ser tratado pelo programador.

Portanto, é uma questão de projeto decidir uma frequência para o *clock* que garanta uma boa precisão na medida, porém consiga fazer capturas sem precisar tratar uma quantidade exagerada de overflow's.

O fragmento de código a seguir mostra uma possível configuração do *timer 1* em modo captura:

```
1 /* Exemplo de como configurar o timer em modo captura.
2 Se nenhuma captura acontecer durante 1 segundo,
3 ocorre overflow */
4
5 TA1CCR0 = 4096 - 1;
6 TA1CTL |= TASSEL_1 | ID_3 | MC_1 | TAIE;
7 TA1CCTL1 |= CM_2 | SCS | CAP | CCIS_0 ;
8 TA1CCTL1 |= CCIE;
```

A **linha 5** define o período de tempo do *timer* – configurado para *up mode*. A **linha 6** escolhe o ACLK, divisor 8 para o clock, *up mode* e habilita a interrupção por *overflow*, respectivamente. Na **linha 7**, tem-se que a borda onde será feita a captura é a descida, habilitado o sincronismo entre *clock* e sinal, modo captura habilitado e, por fim, de onde vem o sinal ao qual será feita a captura. Finalmente, na **linha 8** habilita-se a interrupção de captura.

O próximo trecho de código mostra a estrutura para capturas as interrupções geradas por essa configuração:

```
1  /* Exemplo de estrutura para tratar as
2  interrupcoes no modo captura */
3
4  #pragma vector = TIMER1_A1_VECTOR
5  #pragma vector = TIMER1_A0_VECTOR
6  __interrupt void capture_overflow_timer1(){
7      switch(TA1IV){
8          case TA1IV_TAIFG:
9              /* Trate a captura */
10             break;
11         case TA1IV_TACCR1:
12             /* Trate o overflow */
13             break;
14     }
15 }
```

Essa função está associada a duas diretivas, isto é, deverá tratar tanto interrupção gerada por uma captura, quanto interrupção gerada por overflow. Logo, um condicional *switch* é utlizado na **linha 7** para encaminhar os diferentes eventos, sendo que a identificação do evento é obtido através do registrador TA1IV.

2.5 Módulos e sensores

Nesta seção é dedicado um espaço para falar do sensor *reed switch*, essencial para detectar revoluções tanto na roda, quanto no pedal da bicicleta, e do módulo HC-05, responsável por integrar a comunicação bluetooth ao sistema embarcado.

2.5.1 Reed Switch

O reed switch é uma chave/interruptor (abre e fecha) que pode ser normalmente aberto ou normalmente fechado. Se o dispositivo escolhido para um circuito elétrico é o normalmente aberto, significa que ele fechará o circuito apenas quando for colocado próximo a um campo magnético produzido por um ímã ou eletroímã. O normalmente fechado funciona de maneira inversa, isto é, permanece fechado até ser submetido a um campo magnético.

Figura 2.8: Reed Switch. Fonte: [19]

Pode ser utilizado em aplicações de eletrônica, como alarmes, trancas elétricas, portas, velocímetros, etc.

É um componente simples e barato, custa em média R$ 1,00. Em muitos casos, precisa-se de apenas um resistor (*pull-up* ou *pull-down*) de configuração adicional.

Ele é composto de uma cápsula e duas lâminas de um material ferromagnético (ligas de níquel e ferro). As duas lâminas são colocadas muito próximas, sem que haja contato entre elas, com uma extremidade afixada no vidro e mergulhadas num gás inerte, para não sofrerem oxidação ou deformação mecânica – aumentando a durabilidade.[21] A figura 2.9 ilustra seu funcionamento:

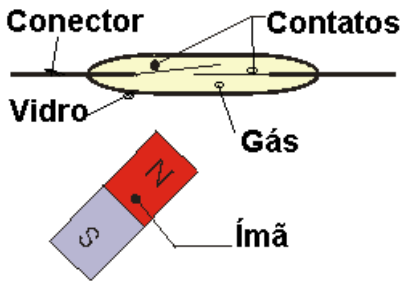

Figura 2.9: Acionamento do Reed Switch. Fonte:[21]

2.5.2 Módulo Bluetooth HC-05

O módulo HC-05 é um componente que torna transparente uma comunicação via bluetooth. Ele possui duas interfaces: serial UART e bluetooth. Logo, pode-se abstrair a complexidade do protocolo bluetooth e trabalhar apenas com a interface serial – uma tecnologia mais simples que é implementada na maioria dos microcontroladores.

Implementa o bluetooth v2.0 e pode operar tanto como escravo quanto mestre. Se enquadra na classe 2, ou seja, a comunicação possui um limite de 10 metros. Possui um regulador de tensão (na placa externa, versão que já vem soldada para ligar em uma *protoboard*, por exemplo), logo pode ser alimentado com uma fonte de 3,0 – 5,0 V.[22]

Custa em média R$ 40,00 e não necessita de configuração adicional para ser ligado a um microcontrolador.

Capítulo 3

Sistema embarcado: implementação

O objetivo do sistema embarcado é medir o tempo de revolução da roda e do pedal usando sensores *reed switch*. Além disso, enviar em tempo real, esses valores ao smartphone através da comunicação bluetooth utilizando o módulo HC-05.

Para facilitar a explicação do sistema descrito, o capítulo é dividido em duas seções: circuito elétrico e algoritmo utilizado no MSP430. Como sugerem os nomes, abordam questões de projeto para a implementação do *hardware* e *software* que formam o sistema embarcado.

A primeira seção foca em detalhes como alimentação, sensores e módulos que são ligados ao microcontrolador. O motivo de ter escolhido uma determinada bateria e como foram resolvidos problemas que surgiram com os sensores, por exemplo, são tópicos que são discutidos ao decorrer da próxima seção.

A última seção é dedicada a explicar o *software* embarcado no microcontrolador. É apresentada tanto uma discussão de alto nível, em relação a implementação do protocolo em conjunto com o smartphone, quanto uma mais baixo nível, sobre o algoritmo para calcular o tempo recebendo interrupções do sensor *reed switch*.

3.1 Circuito elétrico

3.1.1 Alimentação

Segundo o *datasheet* do MSP430G2553,[18] ele deve operar com uma tensão entre 1,8 e 3,6 V (sendo 3,3 V o valor mais comum). O módulo bluetooth HC-05 também precisa ser alimentado, e opera perfeitamente numa tensão em torno de 3,0 V.

A primeira opção foi utilizar duas pilhas tamanho AAA em série, obtendo 3,0 V de entrada. O problema nessa configuração é o decaimento da tensão nas pilhas durante o uso, isto é, a tensão de cada pilha começa em 1,5 V, entretanto, no final da sua vida útil ela apresenta apenas 0,8 V.[23] Em aproximadamente 2,6 V, o módulo HC-05 já começa a apresentar problemas em conectar-se com o smartphone, portanto, essa opção foi descartada por não ser possível utilizar o recurso total das pilhas.

A segunda alternativa foi utilizar um regulador de tensão de 3,3 V – o suficiente para manter o microcontrolador e o módulo bluetooth funcionando. O escolhido foi o LD1117#33, e segundo o *datasheet*, opera com uma tensão de entrada entre 4,75 e 10,0 V.[24] Seguindo essa especificação, optou-se pela bateria de 9,0 V, em que mesmo no fim

da sua vida útil ainda apresenta 5,0 V,[25] portanto, toda energia da bateria pode ser utilizada.

A figura 3.1 mostra o esquemático do módulo de alimentação do circuito. Os capacitores são recomendações do fabricante para garantir estabilidade. Além disso, possui uma chave para ligar e desligar o circuito e um *LED* vermelho para indicar que o sistema está ligado.

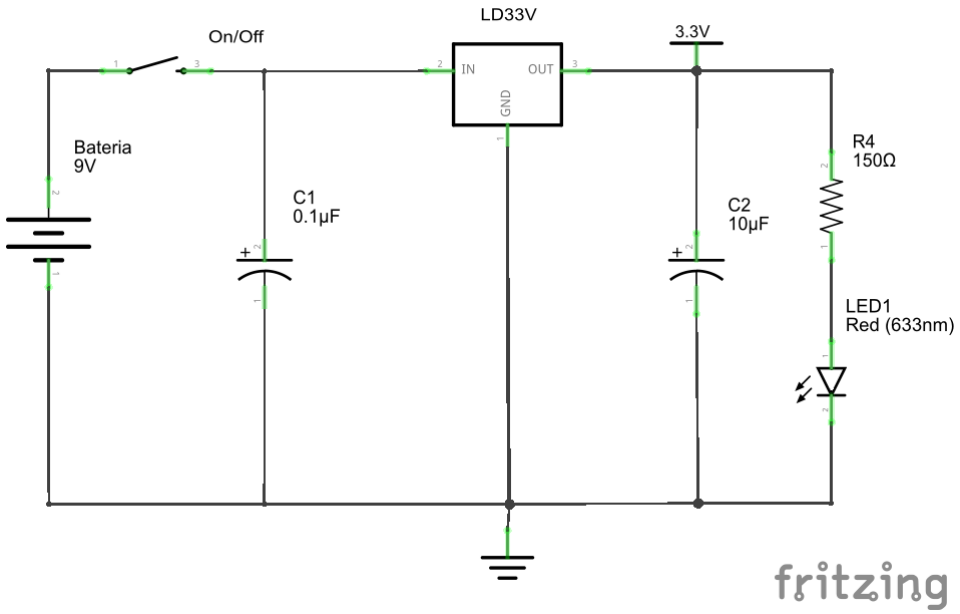

Figura 3.1: Módulo de alimentação do circuito elétrico

3.1.2 Microcontrolador

Na segunda parte do circuito encontra-se o microcontrolador, o *reed switch* e o módulo bluetooth. A figura 3.2 mostra o esquemático do circuito, onde a tensão de alimentação já está regulada para 3,3 V.

Os **pinos 1 e 20** são responsáveis pela alimentação do MSP430. Nos **pinos 18 e 19** está o cristal de 32768 Hz, oscilador que será utilizado pelo ACLK, sendo este utilizado para medida do tempo de revolução da roda e do pedal da bicicleta. O **pino 16** é de $\overline{RESET}$, portanto, um resistor de *pull-up* é utilizado para que o microcontrolador funcione adequadamente. Nos **pinos 3 e 9** estão ligados os *reed switch's* que auxiliarão no cálculo do tempo de revolução do pedal e da roda, respectivamente. E finalmente, no **pino 4** – serial TX – está ligado ao serial RX do módulo HC-05, além disso, o módulo também possui seus próprios pinos de alimentação.

Para cada *reed switch* foi utilizado um circuito de *debouncer*. Esse circuito é uma combinação de um resistor com capacitor, isto é, um filtro passivo passa-baixa. Com os valores escolhidos, a **constante de tempo** do circuito é

$$\tau = R \times C = 10k \times 0,1\mu = 1ms \tag{3.1}$$

Portanto, com a constante de tempo sendo 1 ms, o capacitor carrega em aproximadamente cinco vezes esse tempo, ou seja, 5 ms. O suficiente para eliminar o *bounce* do sensor sem afetar o cálculo do tempo de revolução do pedal e da roda da bicicleta, já que esse tempo está muito abaixo do considerado normal para essas medidas.

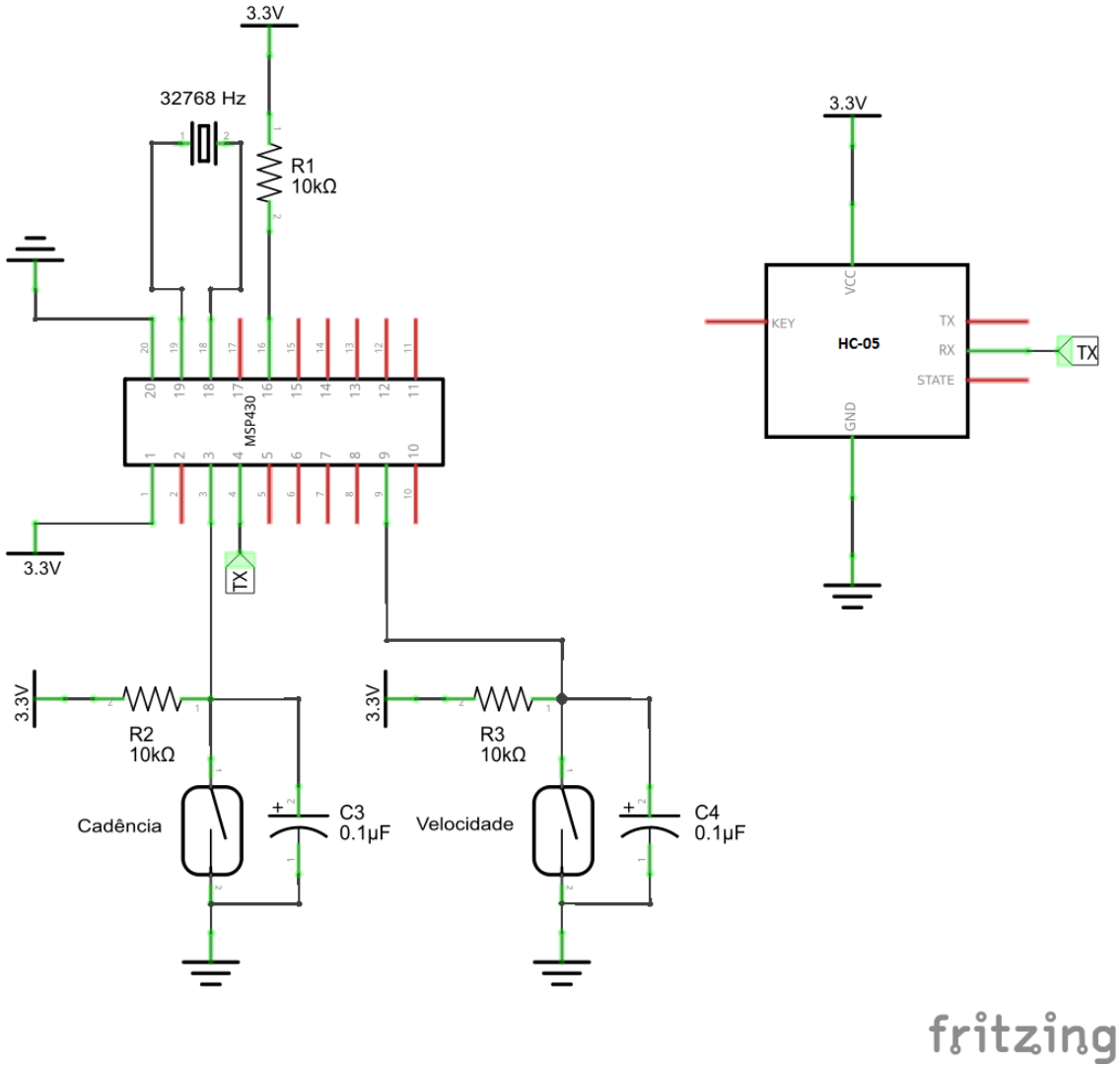

Figura 3.2: Circuito do microcontrolador com a alimentação já regulada

3.1.3 Placa de circuito impresso

Após projetado o circuito, testado em *protoboard* e placa perfurada, foi desenvolvido o *layout* para confeccionar a placa de circuito impresso. A figura 3.3 mostra o resultado final.[1]

Por se tratar de um circuito simples e de baixa frequência, houve poucos aspectos para se preocupar. Um deles foi manter as trilhas sem curvas de noventa graus, mantendo assim

[1] A trilha em vermelho na imagem representa uma camada acima, porém, pode ser substituída por um pequeno fio

um "bom *design*" – isso evita reflexões e outros problemas com sinais de alta frequência. O outro aspecto, foi manter o cristal o mais perto possível do microcontrolador, como recomendado pelo fabricante.

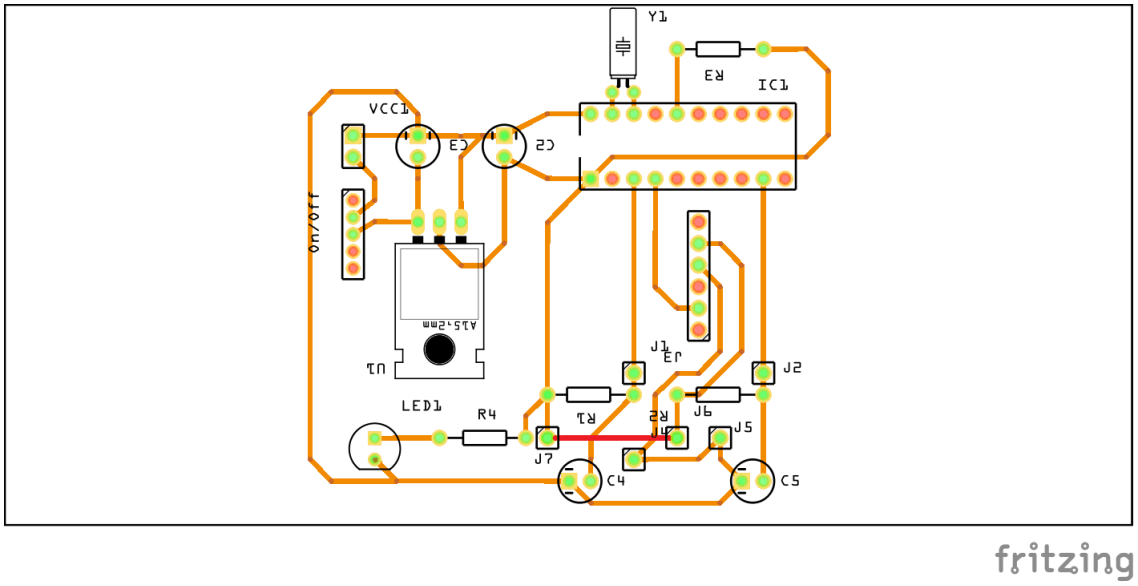

Figura 3.3: *Layout* da placa de circuito impresso (visto por cima)

A placa de circuito impresso foi produzida utilizando uma placa de fenolite cobreada de uma face, o resultado é mostrado nas figuras 3.4 e 3.5, em que a primeira mostra a vista superior da placa e a segunda mostra o fundo, onde estão as trilhas do circuito.

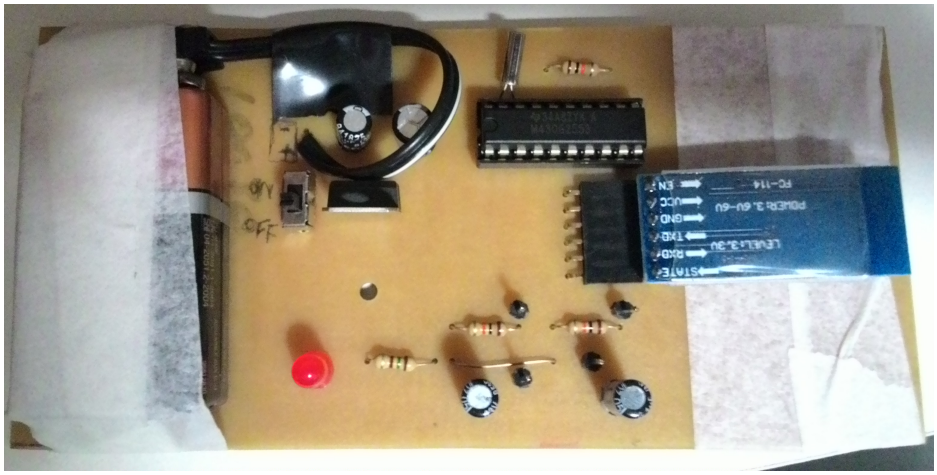

Figura 3.4: Placa de circuito impresso vista por cima

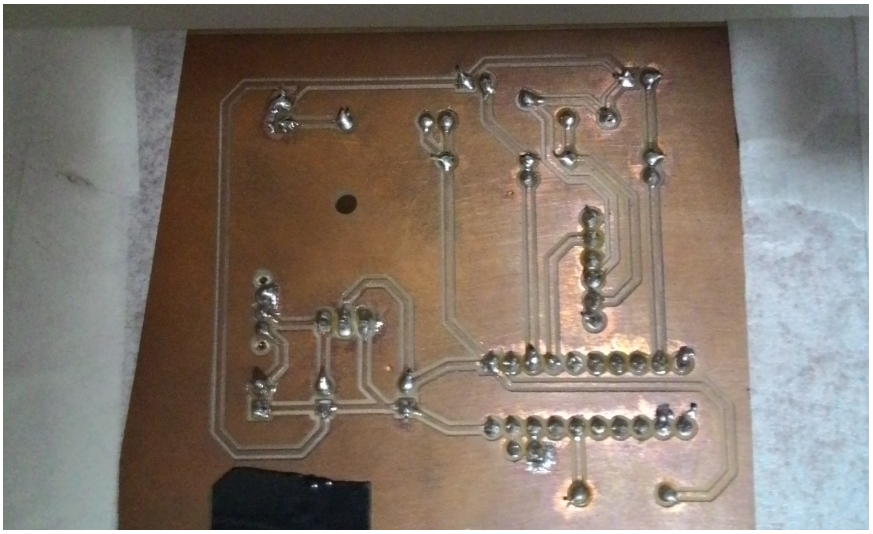

Figura 3.5: Placa de circuito impresso vista por baixo

3.2 Algoritmo utilizado no MSP430

Em resumo, o algoritmo utilizado no microcontrolador realiza os seguintes passos:

1. Configuração dos clock's, pinos de entrada e saída, módulo serial Uart e timer's;
2. Permanece em modo de baixo consumo de energia enquanto não há interrupção causada por um **reed switch**;
3. Após a interrupção, ela é tratada e o tempo obtido é enviado para o **módulo bluetooth HC-05** – responsável por enviar esse conteúdo via bluetooth ao smartphone – através do módulo serial Uart. Após o envio, volta ao estado anterior.

3.2.1 Medindo o tempo

A figura 3.6 mostra como o tempo é medido a partir dos sinais obtidos do *reed switch*.

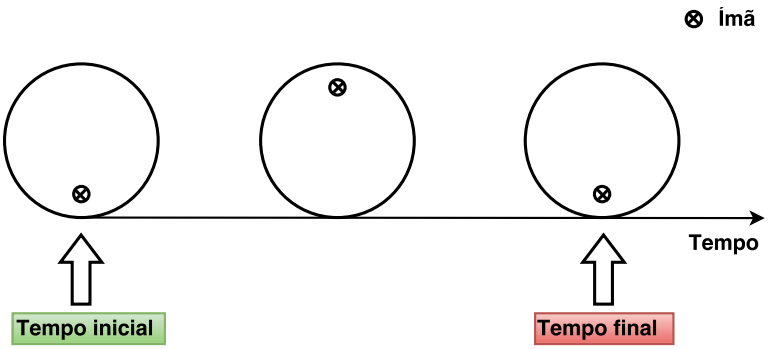

Figura 3.6: Tempo medido entre interrupções do *reed switch*

Nesse esquema, o círculo maior pode ser considerado tanto um giro de pedal, quanto um giro da roda da bicicleta. O primeiro círculo mostra quando o ímã passa pelo *reed*

switch na primeira vez, sendo considerado o **tempo inicial**. E no último círculo, o giro foi completo, isto é, outra vez o ímã passou pelo sensor magnético, o **tempo final**. Portanto, o tempo de revolução pode ser calculado por

$$\text{Tempo de revolução} = \text{Tempo final} - \text{Tempo inicial} \tag{3.2}$$

Um problema em utilizar apenas essa abordagem é identificar quando a roda ou o pedal está parado. Se for utilizado um tempo de espera pequeno, como por exemplo, 1 segundo, o sistema fica limitado e não pode medir velocidades ou cadências baixas. A equação mostra o mínimo suportado para esse tempo (considerando uma roda de 28 polegadas cujo valor da circunferência é aproximadamente 2,23 metros)

$$\text{Cadência mínima (RPM)} = \frac{60}{\text{Tempo (s)}} = \frac{60}{1} = 60,0 \tag{3.3}$$

$$\text{Velocidade mínima (km/h)} = \frac{\text{Tamanho da roda (m)}}{\text{Tempo (s)}} \times 3,6 = \frac{2,23}{1} \times 3,6 = 8,028 \tag{3.4}$$

Por outro lado, utilizando um intervalo grande de espera, 10 segundos, por exemplo, o usuário deve esperar um longo tempo para ter uma nova atualização do sistema.

Portanto, o objetivo foi unir essas duas abordagens, escolhendo um tempo de espera em que fosse possível calcular uma velocidade de aproximadamente 1 km/h e uma cadência próxima de 10 RPM, valores muito baixos, mas que foram considerados relevantes no projeto do sistema embarcado. Sendo assim, reorganizando as equações anteriores tem-se (ainda considerando uma roda de tamanho igual a 28 polegadas)

$$\text{Tempo (s)} = \frac{60}{\text{Cadência mínima (RPM)}} = \frac{60}{10} = 6,0 \tag{3.5}$$

$$\text{Tempo (s)} = \frac{\text{Tamanho da roda (m)}}{\text{Velocidade mínima (km/h)}} \times 3,6 = \frac{2,23}{1} \times 3,6 = 8,028 \tag{3.6}$$

Para efeito de simplificação, foi considerado um tempo igual a 8 segundos tanto para cadência quanto para a velocidade da bicicleta, sendo que para a cadência ainda consegue-se um ganho de capacidade, isto é, 7,5 RPM de cadência mínima e para a velocidade fica muito próximo do mínimo estipulado, não atrapalhando a performance do sistema.

Considerando o modo de medir o tempo e os problemas que precisam ser resolvidos, foi implementado o algoritmo[2] a seguir:

1. O timer deve contar em um intervalo de zero a x segundo(s) – x sendo 1, 2, 4 ou 8[3] –, numa frequência fixa e no modo captura. Sendo que contar até 8 segundos é equivalente a estar parado, tanto no cálculo da velocidade, quanto no valor da cadência.

[2] O algoritmo utilizado para cálculo do tempo de revolução da roda e do pedal são idênticos, mudando poucos parâmetros na implementação.

[3] Esses valores de intervalos (além do 8 segundos) foram utilizados, pois são uma boa divisão para o intervalo do tempo, e além disso, podem ser calculados através de deslocamentos de bits no software.

2. Se houver uma interrupção do **reed switch**, salve esse tempo e envie ao **módulo bluetooth HC-05**. Senão, envie a informação de *overflow*. Zere o contador do timer;

3. Corrija o intervalo do timer, tendo como *feedback* o tempo do item anterior. Volte ao primeiro item.

A intenção do algoritmo é se adaptar às variações na velocidade, isto é, se a velocidade é alta, o timer pode ter um intervalo pequeno antes de ocorrer *overflow*, porém, se a velocidade está muito baixa, o algoritmo precisa de um intervalo maior de contagem para capturar essa medida, no entanto, durante a adaptação do timer, o smartphone ainda possui um *feedback*. O aplicativo Android processa toda a informação de *overflow* como um valor qualquer, exceto o tempo de 8 segundos, que ele considera como velocidade zero.

Portanto, se ocorrer um *overflow* em um intervalo de 2 segundos, por exemplo, esse tempo é enviado para o smartphone e ele processa a informação como se fossse uma nova leitura feita através do sensor, em seguida, atualiza a interface do usuário com esse valor. O algoritmo não zera o timer nesse evento e como a leitura do sensor será em mais do que 2 segundos, a próxima atualização do smartphone será totalmente condizente.

A figura 3.7 mostra um fluxograma do algoritmo utilizado para realizar o procedimento 3 – corrigir o intervalo de contagem do timer. O tempo inicial escolhido é de 8 segundos, que indica que a bicicleta está parada.

Implementação no MSP430

No MSP430 foi utilizado uma frequência de 4096 Hz, isto é, a frequência do TAxCLK (utilizando um cristal externo de 32 768 Hz) combinada com um divisor igual a 8. O timer conta em *continuous mode*, modo captura na borda de descida do sinal e a fonte da captura sincronizada com o *clock*. Isso garante o cálculo do tempo de revolução entre o tempo inicial e final do *reed switch*.

Além disso, é necessário utilizar outra interrupção de comparação utilizando o registrador TAxCCRx[4] para mudar o tempo de *overflow* entre 1, 2, 4 e 8 segundos. A tabela 3.1 mostra os valores de TAxCCRx para cada tempo

Tabela 3.1: Valores do TAxCCRx para cada tempo de *overflow*

Tempo (s)	TAxCCRx
1	4095
2	8191
4	16383
8	32767

Valores do registrador TAxCCRx para cada tempo de *overflow*

Portanto, apenas mudando o valor presente no registrador TAxCCRx é possível adaptar o intervalo de contagem do timer.

[4]Timer's diferentes entre a implementação do tempo da roda e do pedal

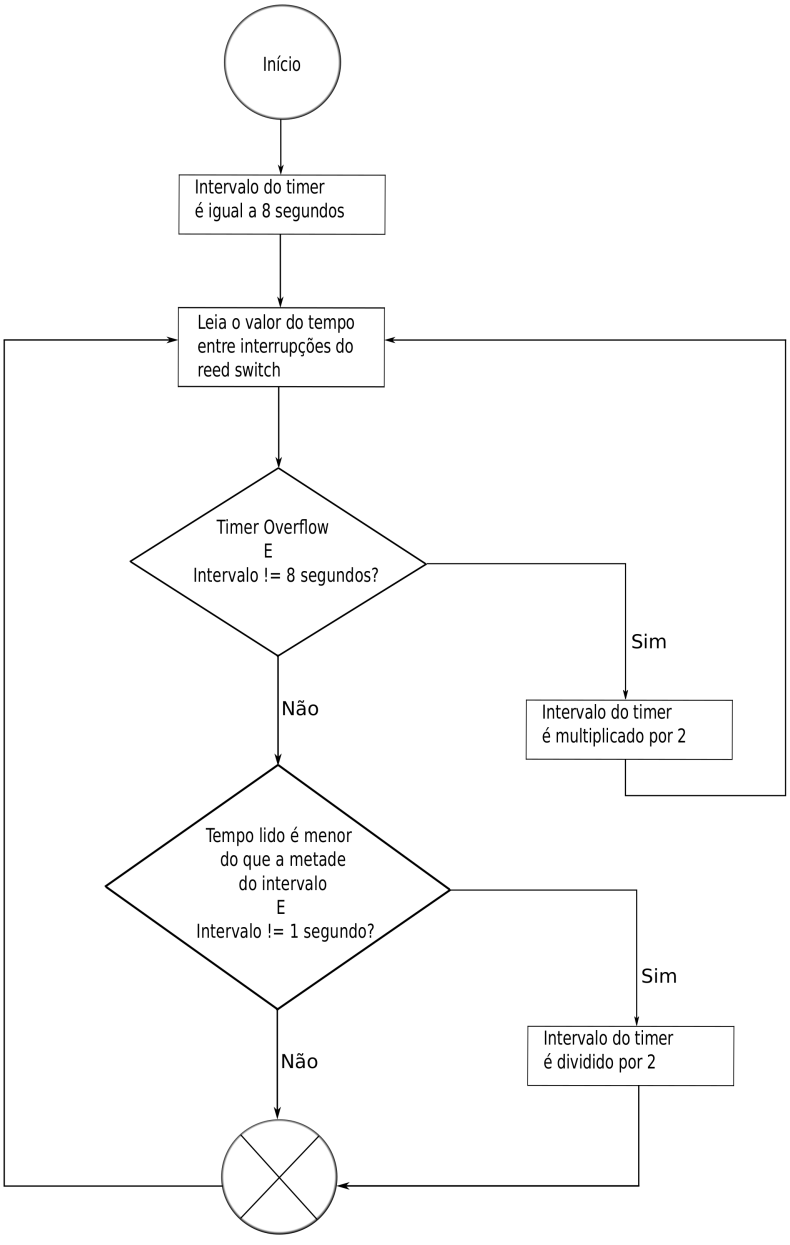

Figura 3.7: Algoritmo de adaptação do intervalo do timer.

3.2.2 Enviando dados para o smartphone

Toda transferência de dados feita entre microcontrolador e smartphone segue um protocolo simples. Primeiramente, a comunicação é unidirecional, isto é, os dados sempre serão enviados do microcontrolador para o smartphone. Além disso, ele estipula que a informação deve sempre estar encapsulada na forma de um pacote, como mostra a figura 3.8.

Figura 3.8: Pacote utilizado pelo protocolo de comunicação entre microcontrolador e smartphone

O pacote possui 3 campos, cada campo possui 1 byte – usando o módulo serial, apenas 1 byte pode ser enviado por vez. O **primeiro campo** é uma constante que identifica de qual sensor está chegando aquela informação, portanto deve estar *hardcoded* em emissor e destinatário. O **segundo e terceiro campo** trata-se do tempo obtido através do sensor, dividida entre dois bytes – mais e menos significativo. O tempo enviado ao smartphone ainda não está em segundos, logo, deverá ser processado pelo aplicativo que conhece a frequência utilizada no microcontrolador.

Para enviar dados via bluetooth ao smartphone, antes deve-se enviar os dados para o módulo HC-05 através da comunicação serial UART. A configuração utilizada foi a seguinte:

- **Dados:** Sem bit de paridade, LSB enviado primeiro, 8 bits de dados, 1 bit de parada e modo assíncrono;
- **Clock utilizado:** SMCLK;
- **Baud rate:** 9600.

Usando uma taxa de 9600 bps, sendo que cada pacote possui 27 bits (9x3, incluindo bits de parada), cada envio ao módulo bluetooth leva aproximadamente 2,81 m s.

Além de configurar o módulo UART, é necessário uma rotina para enviar os dados. Para enviar um byte, foi implementado os passos a seguir:

1. Coloque o byte no registrador TX do módulo serial UART;
2. Espere até que todo o envio tenha sido feito.

Capítulo 4

Aplicativo Android: implementação

Este capítulo mostra o processo de implementação do aplicativo Android para o *smartphone*. Algumas seções abordam aspectos gerais de qualquer *software*, arquitetura, padrões de projeto e banco de dados, por exemplo. Outras são mais específicas e abordam aspectos tecnológicos do sistema Android, como por exemplo, *bluetooth*, mapas e *shared preferences*.

Além disso, a última seção é dedicada a explicar todas as telas do aplicativo e como o usuário deve interagir com elas. Há imagens de todas as telas, para que o leitor possa acompanhar durante a descrição de cada.

4.1 Camada de apresentação

O aplicativo foi desenvolvido utilizando o padrão MVP – utilizado para desacoplar a camada de apresentação. A figura 4.1 mostra um diagrama genérico do padrão por baixo de cada tela (activity/fragment) do software.

Nesse diagrama, a interface IView deve ser implementada por uma Activity ou Fragment (que também são responsáveis por manter a instância do Presenter). A implementação será totalmente passiva, ou seja, apenas repassa as interações feitas pelo usuário e mostra resultados processados pelo Presenter.

O Presenter é a classe intermediária, em que foi necessário um esforço em mantê-lo livre da tecnologia de apresentação (Activity ou Fragment). Em teoria deveria ser apenas em "*plain java*", porém isso não é uma tarefa fácil no Android, já que quase tudo depende de classes globais como a Context.

A implementação do IModel é onde reside as regras de negócio ou apenas um *interactor* para a persistência. Geralmente, possui várias dependências, que não estão presentes no diagrama – preferências do usuário, bluetooth, etc. Em muitas ocasiões, tarefas assíncronas estarão executando nesse espaço, portanto, ele precisa de uma referência para IListener, implementada pelo Presenter, para atualizar a interface de usuário quando necessário.

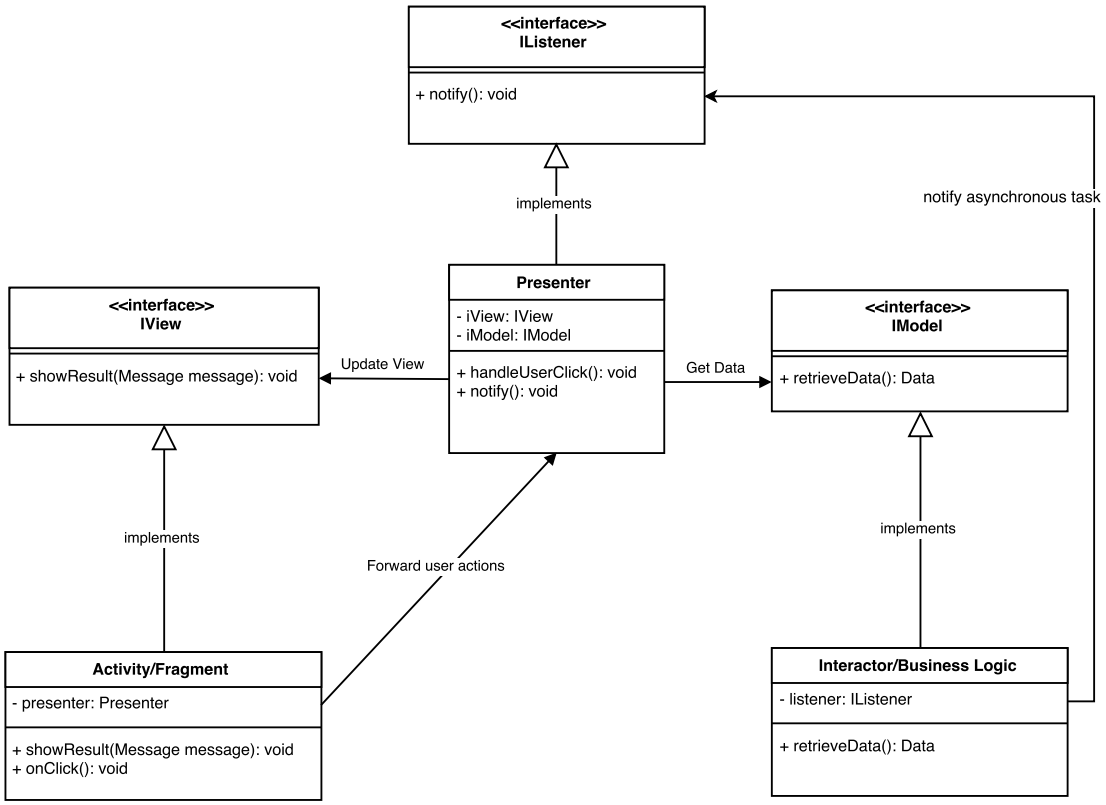

Figura 4.1: Diagrama de interação entre os componentes do Model-View-Presenter no contexto do Android

4.2 Injetando dependências com o Dagger

Com o objetivo de desenvolver um código desacoplado e testável, foi utilizado amplamente o padrão de projeto **injeção de dependência**. E para reduzir o *boilerplate* que vem com o padrão e facilitar algumas dependências, optou-se pela utilização do *framework* Dagger.

O Dagger instancia as classes da aplicação e satisfaz suas dependências, utilizando a anotação @Inject para identificação. As dependências são descritas por métodos que possuem a anotação @Provides, e esses métodos estão dentro de módulos – classes especiais que são anotadas por @Module.[1] Cada método "ensina" como o Dagger deve prover uma determinada dependência. Além disso, o framework faz a verificação em tempo de compilação, portanto, qualquer problema relacionado a criação das dependências será mostrado antes da execução.[26]

O Dagger foi muito importante na codificação do MVP, já que a implementação da view deve manter a instância do presenter, porém não pode conhecer o model. Portanto, por padrão, na activity ou fragment sempre tem uma anotação

```
1 @Inject Presenter presenter;
```

E no módulo responsável por essa dependência, tem-se o seguinte método

```
1 @Provides
2 public Presenter providePresenter(View view, Model model){
3     return new Presenter(view, model);
4 }
```

É necessário também informar ao Dagger qual a fonte das instâncias da view e do model, que pode ser feito em outros módulos. Nesse aplicativo, outro módulo tem a tarefa de prover o model, logo deve ficar

```
1 @Module(library = true)
2 public class ModelModule {
3     @Provides
4     public Model provideModel(){
5         return new Model();
6     }
7 }
```

O atributo *library* sendo verdadeiro, indica que há dependências que serão usadas por outros módulos.

4.3 Implementação do bluetooth

Para implementar a comunicação bluetooth foram criadas três interfaces e suas devidas implementações. A escolha de utilizar interfaces se baseia no fato de que se for preciso mudar de tecnologia – bluetooth low energy, por exemplo – não afetará drasticamente o restante do código, já que é possível apenas mudar a implementação das interfaces.

[1] Todas essas anotações fazem parte do padrão Java JSR-330.

A representação gráfica pode ser vista na figura 4.2, que mostra o diagrama de classes UML para esse módulo do programa[2]. A seguir são definidas as responsabilidades de cada interface:

- **IBluetoothSetup:** Interface responsável por retornar os dispositivos pareados.
- **IBluetoothConnection:** Interface que permite criar uma conexão bluetooth com outro dispositivo.
- **IBluetoothConnected:** Interface que possibilita a leitura dos dados recebidos do dispositivo ao qual se está conectado.

Com relação aos recursos utilizados da API do Android, destacando a necessidade de cada classe, tem-se:

- **BluetoothAdapter:** Classe que representa o adaptador bluetooth do dispositivo local. Essa classe é utilizada para habilitar/desabilitar o adaptador bluetooth e buscar dispositivos que estejam pareados, por exemplo.[27]
- **BluetoothDevice:** Representa um dispositivo bluetooth remoto, isto é, o dispositivo ao qual será estabelecida uma conexão. Essa classe permite criar uma conexão bluetooth, parear com o dispositivo e permite ver dados do dispositivo remoto (nome, endereço MAC), por exemplo.[28]
- **BluetoothSocket:** Muito similar aos *sockets TCP*, é a classe responsável por gerenciar uma conexão bluetooth. A partir dela, o aplicativo pode ler os dados que são enviados pelo sistema embarcado na bicicleta.[29]

Nesse esquema apresentado na figura 4.2, as interfaces utilizadas por outros módulos do aplicativo não tem conhecimento da implementação do bluetooth no Android – todas as dependências da API do android estão nas classes que mantém contrato com as interfaces –, portanto, evita-se um alto acoplamento entre as diferentes partes do software.

[2] Alguns métodos, atributos e até mesmo relações com outras classes foram omitidas para manter a simplicidade.

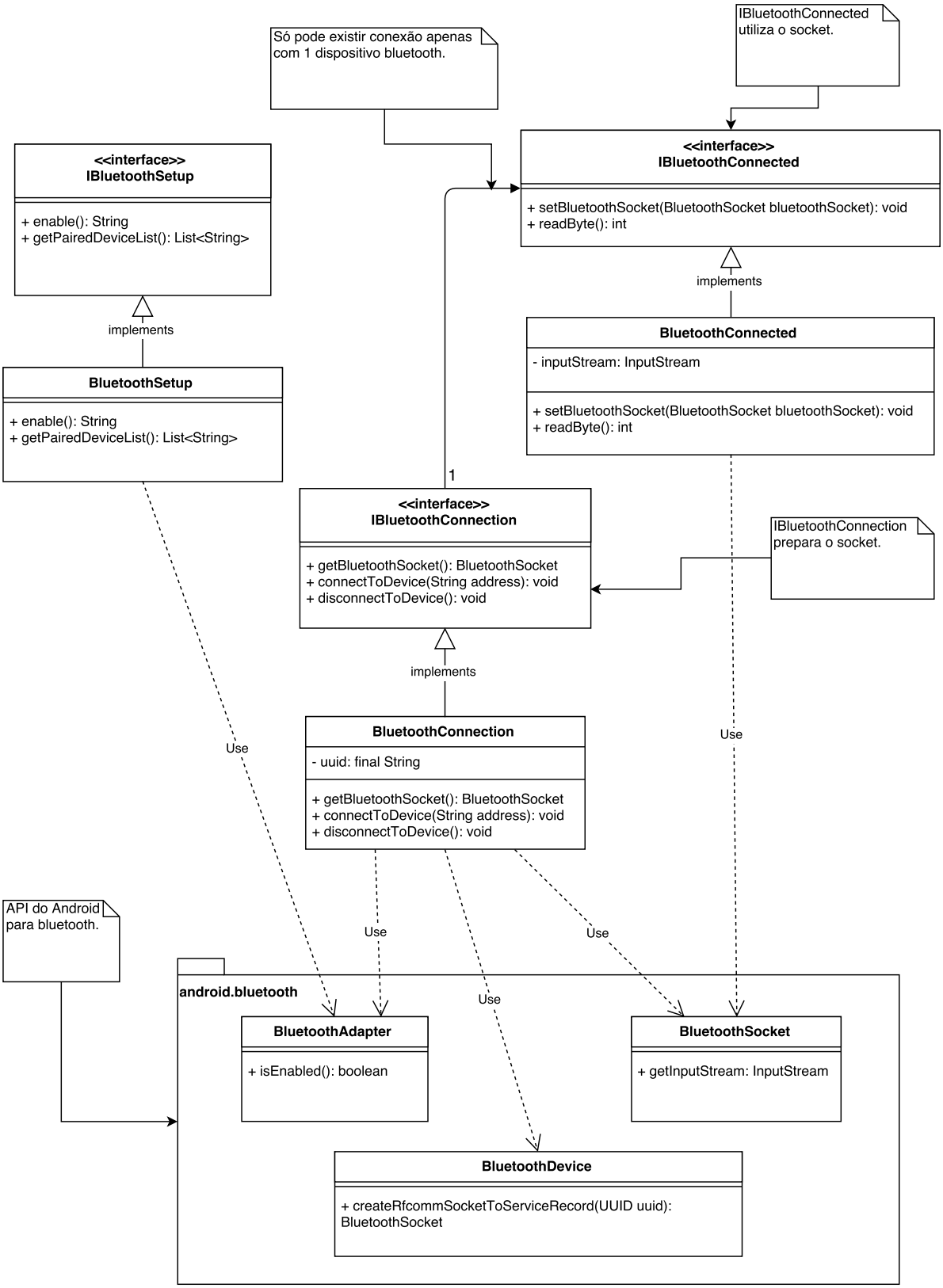

Figura 4.2: Diagrama de classes do módulo bluetooth

4.4 Preferências do usuário

O Android permite um jeito fácil de persistir dados no aparelho, usando **shared preferences**. Esse método foi utilizado para salvar as preferências do usuário – diâmetro da roda, cadência desejada e dispositivo bluetooth para conexão.

O android possui uma classe que deve ser utilizada para implementar essa técnica, SharedPreferences. Uma outra classe foi criada para usá-la, e ser a interface de todo aplicativo para utilizar preferências compartilhadas. A figura 4.3 mostra a dependência e métodos relevantes de cada classe.

A classe UserSharedPreferences mantém três atributos constantes para armazenar a chave de cada valor armazenado, inclusive a chave que identifica a preferência. Os métodos **retrieveBluetoothMacAddress()**, **retrieveWheelSize()** e **retrieveDesiredCadence()** – nomes autoexplicativos – utilizam **getString()**, **getInt()** e **getFloat()**, respectivamente, para buscar os valores salvos. Já o método **save()**, utiliza os métodos **putInt()**, **putFloat()** e **putString()** que estão em uma interface chamada Editor, que será criada na chamada do método **edit()** de SharedPreferences.

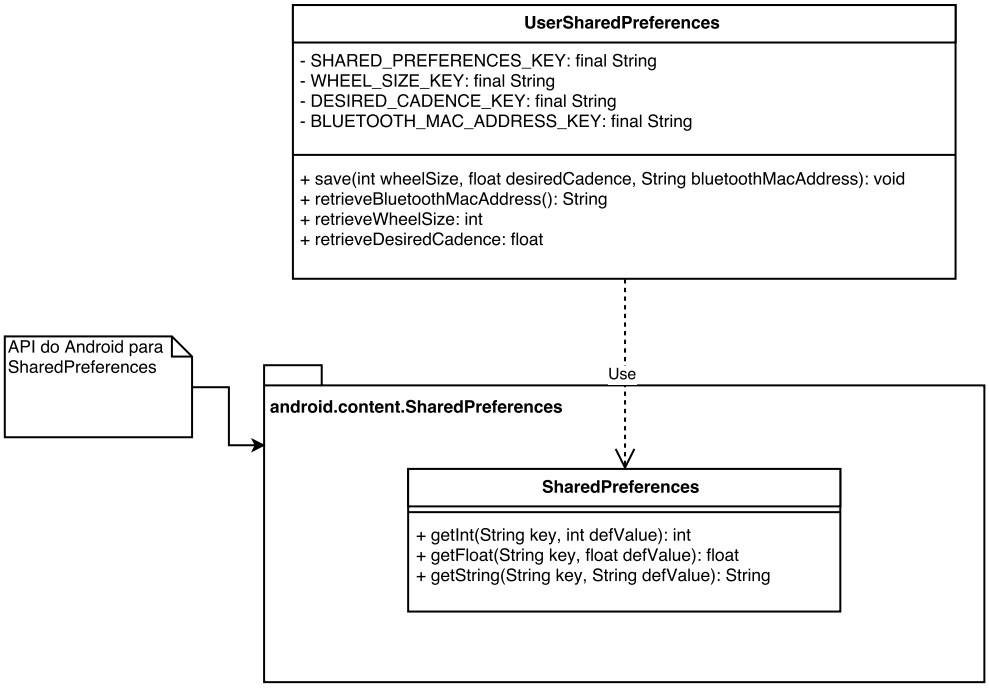

Figura 4.3: Diagrama das preferências do usuário utilizando SharedPreferences

As funcionalidades que tem dependências desses valores utilizarão os dados salvos no smartphone, porém o usuário poderá mudar essas preferências sempre que desejar. Essa estratégia se baseia no fato de que, na maioria das vezes, um usuário estará utilizando a mesma bicicleta com o mesmo sistema embarcado e já tem um histórico de cadência preferida. A tabela 4.1 mostra um exemplo de preferência aceita pelo aplicativo.

Tabela 4.1: Preferências do usuário: chave x valor

	Chave	Valor
Diâmetro da roda	wheelSize	28
Cadência desejada	desiredCadence	90.0
Endereço MAC do dispositivo remoto	bluetoothMacAddress	30:15:01:16:03:90

Exemplo de preferências salvas pelo usuário

4.5 Interação com o sistema embarcado

O aplicativo possui uma classe chamada BikeModel que, resumidamente, representa a bicicleta dentro do código do aplicativo Android. Ela implementa uma interface chamada IBikeModel, que será a referência do Presenter, além disso, possui outros métodos que são utilizados para atualizar o estado da bicicleta. Essa classe possui quatro dependências, são elas: **IBluetoothConnection**, **IBluetoothConnected**, **UserSharedPreferences** e **DatabaseHelper**.

Ela utiliza o endereço MAC lido de UserSharedPreferences para conectar ao sistema embarcado através de IBluetoothConnection e, após isso, recebe os dados a partir de IBluetoothConnected. Utiliza-se o diâmetro da roda – outro dado obtido de UserSharedPreferences – para calcular a velocidade e distância percorrida pela bicicleta, além disso, utiliza a cadência desejada, também salva pelo usuário como preferência, para indicar se o usuário deve mudar de marcha para manter aquela cadência. A classe DatabaseHelper é utilizada para persistir dados estatísticos no banco de dados SQLite.

Essa classe possui o método **readForever()**, que implementa uma nova *thread* e recebe todos os dados enviados pelo sistema embarcado na bicicleta, portanto, deve permanecer executando durante toda a comunicação entre as partes. Utilizando métodos dessa classe, ele realiza as seguintes rotinas:

1. Identifica o sensor (roda ou pedal) e decodifica o tempo (através dos dois bytes de dados);
2. De acordo com o sensor, atualiza atributos do model: cadence, speed, distance, averageSpeed e averageCadence. Utilizando-se de métodos privados, como: **calculateSpeed()** e **calculateCadence()**.
3. Por fim, atualiza a view com os valores modificados.

4.5.1 Fórmulas utilizadas

A classe BikeModel é responsável pela grande maioria dos cálculos matemáticos do aplicativo. As fórmulas mostradas a seguir são utilizadas para processar os dados lidos do microcontrolador[3]

$$\text{Tempo (s)} = \frac{\text{Valor recebido do microcontrolador}}{\text{Frequência utilizada no microcontrolador}} \tag{4.1}$$

[3]Algumas simplificações foram feitas no software para diminuir o número de instruções

onde **Frequência utilizada no microcontrolador** é igual a 4096 Hz.

$$\text{Tamanho da roda (m)} = \frac{\text{diâmetro da roda (pol)} \times 2,54 \times \pi}{100} \tag{4.2}$$

$$\text{Cadência (RPM)} = \frac{60}{\text{Tempo}} \tag{4.3}$$

$$\text{Velocidade (km/h)} = \frac{\text{Tamanho da roda (m)}}{\text{Tempo (s)}} \times 3,6 \tag{4.4}$$

Para saber se o usuário está dentro da cadência desejada, utiliza-se

$$0,9 \times \text{Cadência desejada} < \text{Cadência atual (RPM)} < 1,1 \times \text{Cadência desejada} \tag{4.5}$$

4.6 Mapas

Para melhorar a experiência do usuário com o sistema embarcado, foi incluído uma funcionalidade que permite a criação de percursos que ele gostaria de fazer durante um trajeto de bicicleta. Para implementar os mapas no aplicativo, foi utilizado a API do Google Maps v2.[30]

Assim que escolher criar um novo percurso, ele terá um mapa e poderá marcar quantos pontos desejar. Após escolher todos os pontos, apenas deve informar um nome para aquele percurso. Esses pontos serão salvos na mesma sequência em que foram escolhidos no mapa. O usuário pode manter sua localização via internet ou GPS, dependendo da configuração do seu smartphone.

Após criado um novo percurso, ele estará disponível para ser acessado pelo usuário. O objetivo será passar por todos os pontos escolhidos, na mesma ordem. Isso foi implementado utilizando um *array list*, em que cada ponto alcançado pelo usuário, este é removido da lista e consequentemente do mapa.

Cada ponto no mapa é representado por dois parâmetros: latitude e longitude. A cada mudança de posição do usuário, um método *listener* é invocado, logo, é possível fazer as modificações de acordo com o seu movimento, como mover a câmera (visualização do mapa) e verificar se ele alcançou algum ponto no mapa. Para determinar se durante o trajeto o usuário passou por um ponto específico, é considerado uma margem de erro, como mostra as equações a seguir

$$\text{Latitude do ponto} - 0,000030 \leq \text{Latitude do usuário} \leq \text{Latitude do ponto} + 0,000030 \tag{4.6}$$

$$\text{Longitude do ponto} - 0,000030 \leq \text{Longitude do usuário} \leq \text{Longitude do ponto} + 0,000030 \tag{4.7}$$

Isso garante que o usuário não precisará passar exatamente no mesmo local do ponto.

4.7 Persistência utilizando banco de dados

4.7.1 Modelo de dados

A primeira coisa a se definir em um projeto de banco de dados é o diagrama relacional entre as entidades. A figura 4.4 mostra o diagrama conceitual do modelo de dados utilizado.

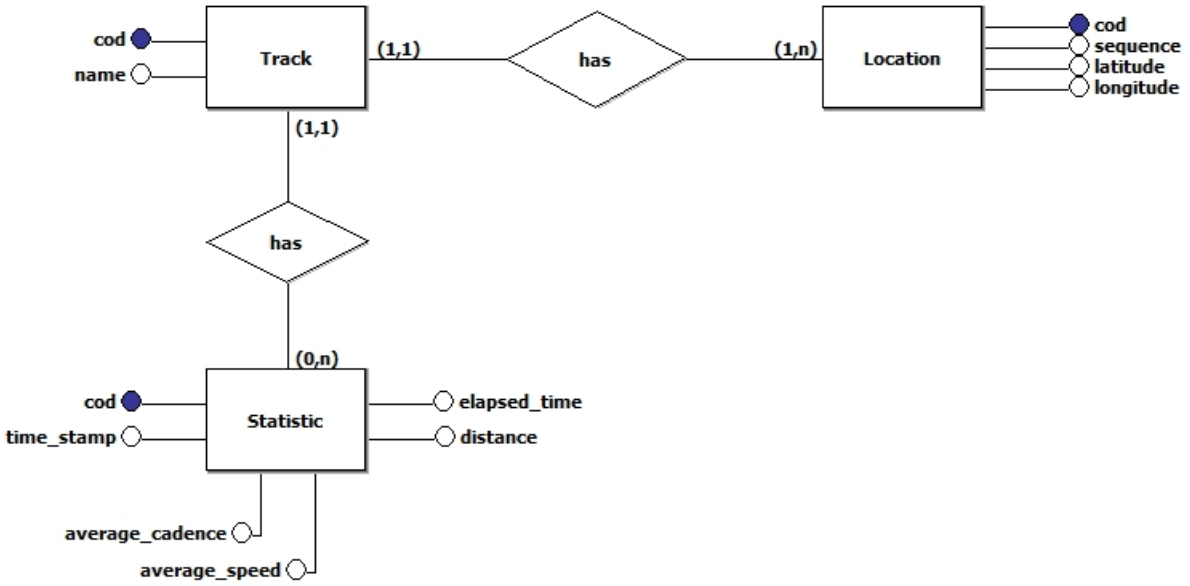

Figura 4.4: Modelo de dados conceitual implementado no software brModelo.[31]

Em seguida, o esquema transformado em tabelas (relacional), que poderá ser implementado em um banco de dados relacional utilizando a linguagem *SQL*

- Track: (**cod**, name)
- Location: (**cod**, cod_track, sequence, latitude, longitude)
- Statistic: (**cod**, cod_track, time_stamp, average_cadence, average_speed, elapsed_time, distance)

As chaves primárias estão em negrito e chaves estrangeiras sublinhadas. Nesse esquema, cod_track referencia a tabela Track.

Cada localização representa um ponto no mapa, e como é mostrado na cardinalidade, um percurso pode ter várias delas. O conjunto de localizações define um percurso. Cada localização tem sua latitude e longitude, utilizadas para localizá-la no mapa, além disso, possui uma sequência, que define a ordem em que será considerada no percurso.

Além disso, uma trajetória pode ter várias estatísticas relacionadas a ela. Serão classificadas por data e hora, e possui diversos atributos que servem de comparação para o mesmo percurso. Portanto, se em um determinado dia, o usuário fez o percurso em 30 minutos, por exemplo, ele poderá tentar reduzir esse tempo em uma outra ocasião.

4.7.2 Utilizando o SQLite

O SQLite é um banco de dados relacional nativo do sistema operacional Android. Foi utilizado para implementar e manipular o modelo de dados descrito.

Para utilizá-lo no Android, deve-se criar um classe que estende a classe **SQLiteOpenHelper**. Essa classe será responsável por criar as tabelas e atualizá-las durante a instalação do aplicativo no smartphone. Além disso, implementa as quatro operações básicas: *create*, *retrieve*, *update* e *delete*. Essa classe é a **DatabaseHelper**, o único meio que deve ser utilizado para persistir dados no SQLite. Ela é instanciada no momento em que a aplicação é criada e injetada como *singleton* pelo Dagger, ou seja, vive durante todo o ciclo de vida do aplicativo.

4.8 Interação do usuário com o aplicativo

A seguir são apresentadas explicações sobre todas as telas presentes no aplicativo e como o usuário deve interagir com elas. No final da seção estão presentes as imagens de todas elas.

A tela inicial é mostrada na figura 4.5. O botão **NEW TRACK** dá acesso a uma nova tela onde é possível marcar um percurso no mapa, cada percurso salvo através dessa segunda tela será salvo no banco de dados e mostrado na lista abaixo do botão.

O ícone  dá acesso a tela onde o usuário pode definir suas preferências.

E finalmente, como dito anteriormente, cada elemento da lista representa um percurso. Ao se dar um *click* longo em algum item, um *menu* é aberto, onde é possível deletar o item ou ir para outra tela e ver as estatísticas associadas ao percurso. Com um *click* normal, o usuário é mandado para a tela onde tem acesso ao mapa e as medidas feitas pelo sistema embarcado.

A figura 4.6 mostra o mapa onde dever ser marcado os pontos do novo percurso. O usuário pode clicar no ícone  para que a câmera se aproxime de sua localização, além de poder fazer gestos já conhecidos para dar *zoom* ou mover o mapa. Assim que estiver na localização desejada, o usuário poderá dar um *click* no mapa para adicionar um ponto ou um *click* longo para remover o último ponto adicionado. Após definir o percurso, ele deve clicar no botão **SAVE TRACK**, e finalmente, dar um nome apropriado para salvá-lo no banco de dados.

A tela mostrada na figura 4.7 é onde o usuário deverá definir suas preferências. Há dois campos para inserir o tamanho da roda da bicicleta e cadência desejada, além disso, uma lista com botão de rádio para escolher um dispositivo bluetooth. Ele sempre poderá voltar nesse lugar para editá-las, e para salvar as mudanças feitas, deverá clicar no botão **SAVE PREFERENCES**.

A tela de estatísticas é mostrada na figura 4.8. Trata-se de uma lista extensível, organizada pela data e hora de um percurso salvo. O usuário poderá ver os detalhes de um percurso feito, clicando no item desejado da lista.

E finalmente, é mostrado na figura 4.9 as telas principais do aplicativo. Essas telas são divididas em duas abas e podem ser acessadas apenas deslizando o dedo pelo smartphone. Na aba **SENSOR** tem-se todas as medidas que são calculadas através dos dados enviados pelo sistema embarcado, enquanto na aba **MAP**, é possível visualizar a localização do

usuário e os pontos aos quais ele deverá passar, sendo que o de cor verde sinaliza o próximo local a ser alcançado. O botão **START TRACK** estabelece uma conexão bluetooth com o sistema embarcado, após isso, a aba **SENSORS** já se atualiza com os dados fornecidos na comunicação. O cronômetro se inicia assim que o usuário cruzar o primeiro ponto do percurso e pausa assim que o ciclista passa pelo ponto final, sendo que isso também é um gatilho para salvar todas as estatísticas do percurso automaticamente.

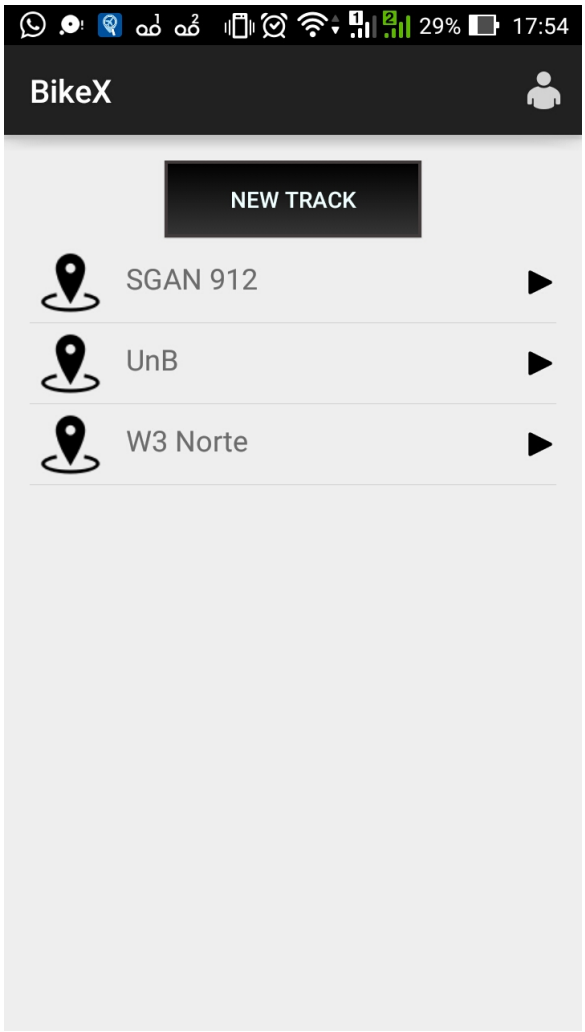

Figura 4.5: Tela inicial do aplicativo Android

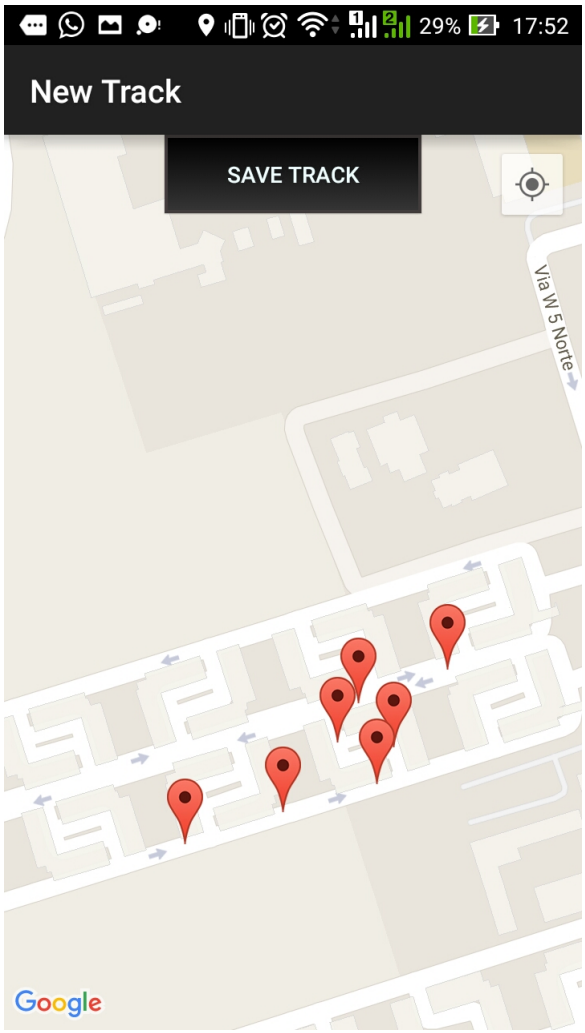

Figura 4.6: Tela de novo percurso do aplicativo Android

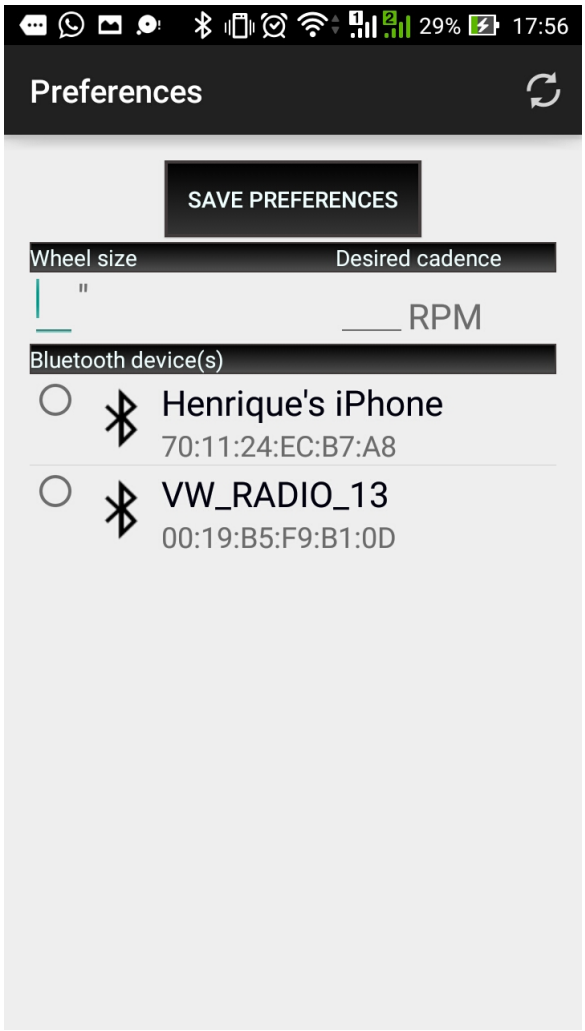

Figura 4.7: Tela de preferências do aplicativo Android

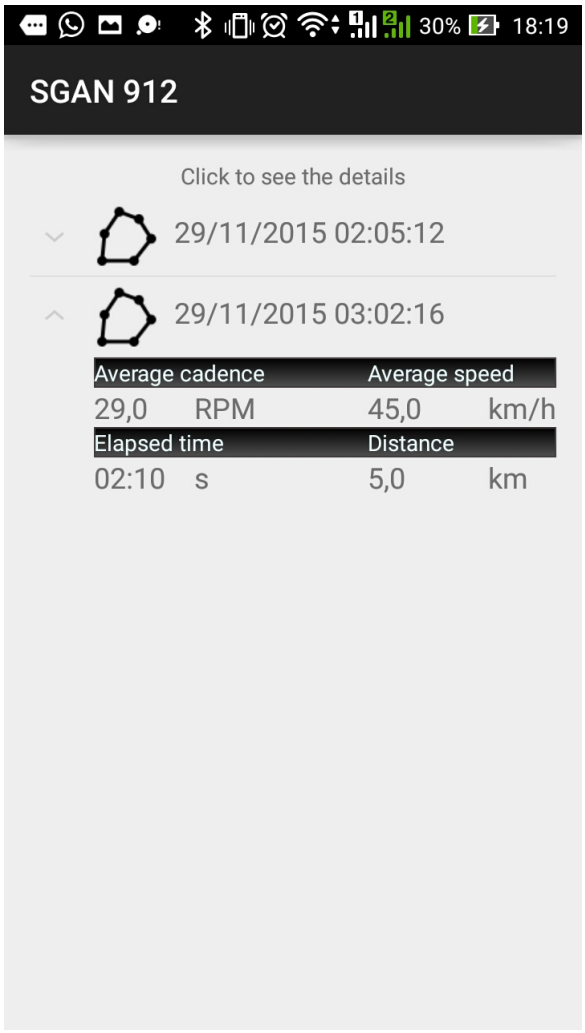

Figura 4.8: Tela de estatísticas do percurso do aplicativo Android

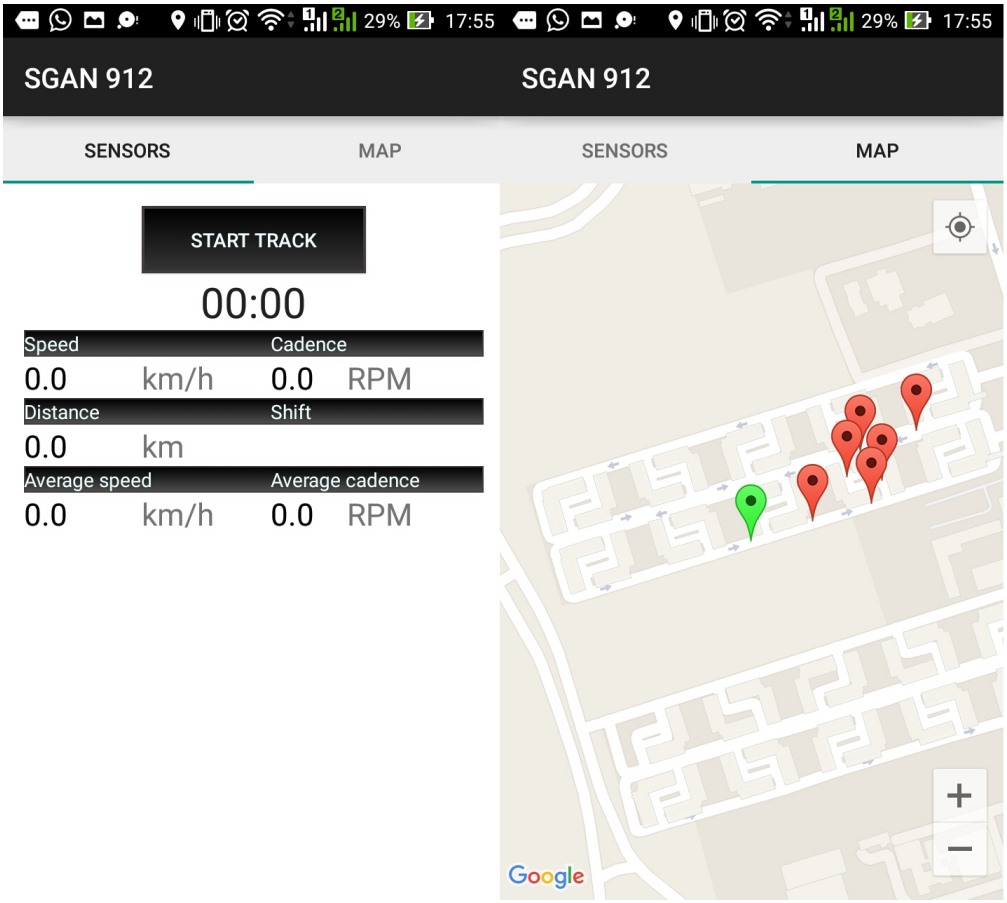

Figura 4.9: Tela principal do aplicativo Android

Capítulo 5

Resultados

Este capítulo mostra os resultados dos principais testes feitos com o sistema embarcado e aplicativo Android.

Os primeiros testes tentam isolar algum componente do sistema em geral, sendo que o primeiro tem um foco na precisão do cristal e no algoritmo utilizado no sistema embarcado, logo, o sensor magnético foi trocado pelo gerador de funções. No segundo experimento, o objetivo é observar como o algoritmo se comporta durante mudanças acentuadas na cadência, sendo que esta foi gerada por um outro microcontrolador. [1]

O terceiro teste coloca tudo na prática, isto é, o sistema embarcado é colocado na bicicleta e a cadência e velocidade são comparadas com outro produto.

5.1 Simulando a cadência e velocidade utilizando um gerador de funções

Este primeiro teste foi para determinar se a cadência e velocidade estavam sendo calculadas adequadamente. A escolha do gerador de funções baseou-se no fato de ser um equipamento mais robusto e menos sujeito a erros de precisão. Logo, além de verificar a precisão do circuito, foi possível determinar que o cristal do sistema embarcado estava bem calibrado. A figura 5.1 mostra local onde foi conduzido os testes.

Como discutido no capítulo 3, o microcontrolador está preparado para capturar eventos do sinal na borda de descida, isto é, quando o sensor magnético é acionado, o sinal cai de *Vcc* para 0 V. A figura 5.2 mostra o sinal utilizado no gerador de funções que foi colocado nos pinos do sistema embarcado em que deveria ser ligado o sensor *reed switch* no projeto completo, em seguida, a figura 5.3 mostra o resultado da cadência igual a 60,0 RPM – aplicativo ainda em fase de desenvolvimento –, em que o sinal de entrada são pulsos gerados a cada 1 segundo.

[1] Em ambos, a largura do pulso que simula o acionamento do *reed switch* foi escolhida levando em consideração apenas que deve ser maior do que a constante de tempo do circuito de *debouncer*, ou seja, maior do que 5 ms.

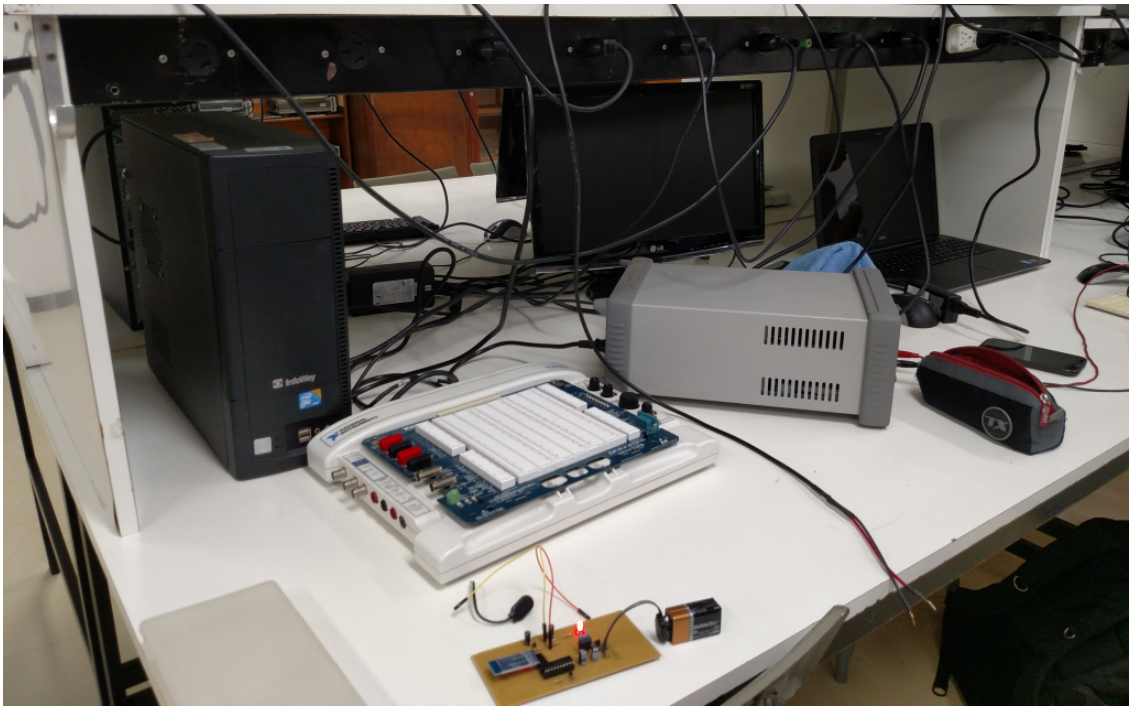

Figura 5.1: Bancada utilizada para realizar os testes

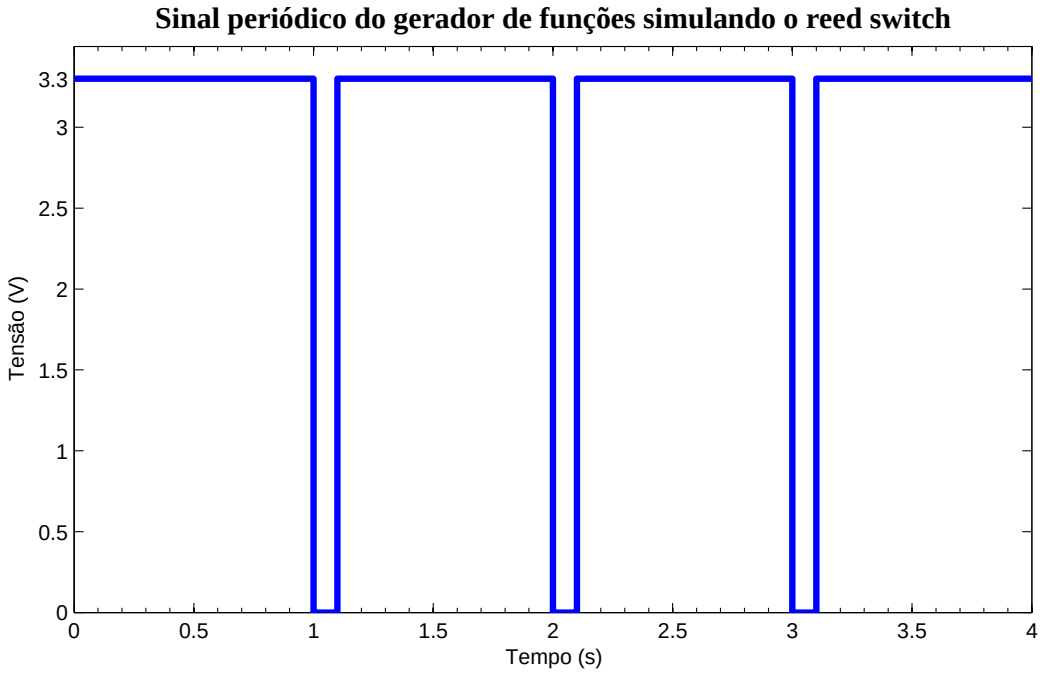

Figura 5.2: Sinal utilizado no gerador de funções para simular o sensor magnético

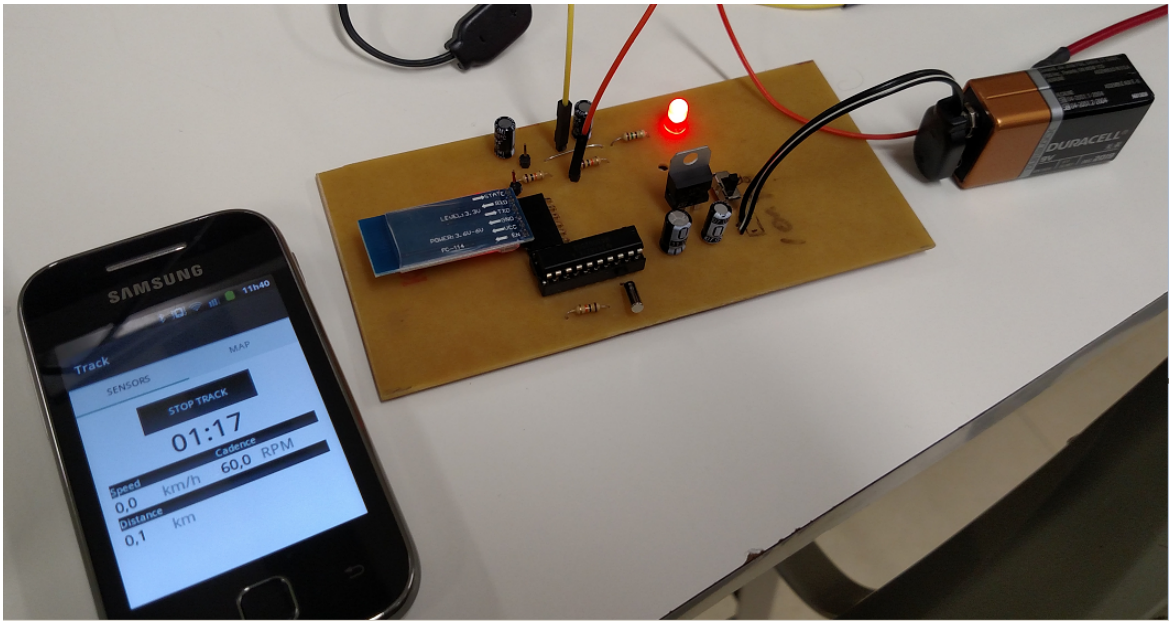

Figura 5.3: Teste de cadência utilizando o gerador de funções

5.2 Simulando a cadência por um sinal gerado por outro microcontrolador

Neste teste do software embarcado do microcontrolador, foi preciso simular dois componentes do sistema: *reed switch* e aplicativo Android. No caso do **reed switch**, pela dificuldade de gerar um sinal com período conhecido, foi utilizado outro microcontrolador para produzí-lo. Já o segundo componente, foi substituído por um programa simples utilizando a ferramenta para desenvolvimento de protótipos, Processing. A comunicação entre MSP430 e o Processing é serial UART (através da porta USB), portanto, foi possível reutilizar o código que já estava presente para comunicar-se com o módulo bluetooth HC-05.

Os testes a seguir mostram gráficos do sensor de cadência[2] – mais fácil de analisar a partir do tempo –, porém, os mesmos testes foram feitos para a velocidade.

5.2.1 Cadência múltipla de 2

A figura 5.5 mostra o gráfico de uma simulação com os valores salvos do Processing. O algoritmo utilizado no microcontrolador que substitui o *reed switch*, foi programado para gerar pulsos em intervalos que geram valores de cadência múltiplos de 2. Logo, o primeiro pulso levou 3,0 segundos, o que equivale a uma cadência de 20,0 RPM, já o segundo pulso, levou 1,5 segundo (de 3,0 a 4,5 no gráfico), ou seja, 40,0 RPM, e assim por diante. A figura 5.4 mostra o sinal descrito que substitui o sensor magnético.

Os valores referentes aos pontos vermelhos são enviados via serial pelo microcontrolador testado, logo, tudo que chegava era processado e impresso em tela pelo Processing.

[2]Para os valores de cadência enviados ao smartphone, foi utilizado um gráfico em que as ligações entre os pontos são lineares, levando em consideração o que mais se aproxima da realidade.

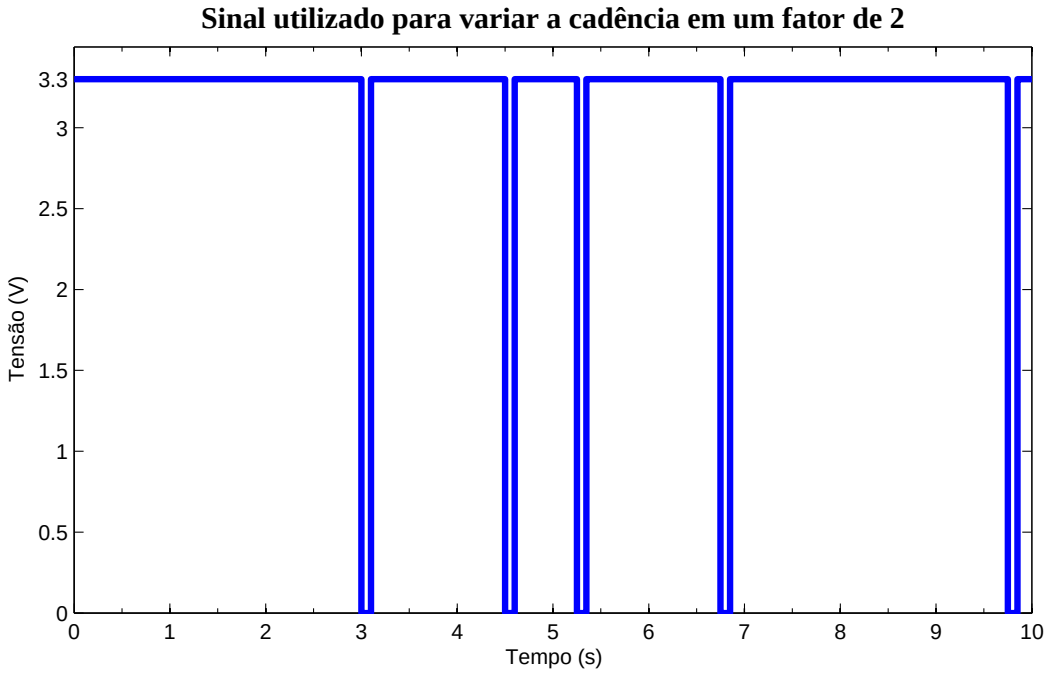

Figura 5.4: Gráfico do sinal utilizado para cadência variar em um fator de 2

Assim, foi possível observar o tempo de resposta do algoritmo ao se adaptar a diferentes velocidades.

Os pontos em que as cadências são 30 e 60 RPM, não são geradas a partir do sinal do microcontrolador substituto. Como o algoritmo do sistema embarcado se adapta, explicado no capítulo 3, contando até 1 segundo não há nenhum sinal de interrupção, portanto, ele envia um *feedback* para o smartphone, o mesmo ocorre na comparação com 2 segundos.

5.2.2 Cadência durante uma queda brusca

Como visto no capítulo 3, um problema da abordagem simplista seria o tempo de espera até o sistema detectar que o pedal da bicicleta está parado, portanto, nesse teste o objetivo é verificar como o algoritmo implementado reage a uma parada imediata. A figura 5.6 mostra o sinal simulando o *reed switch*, em que até o tempo zero havia um período de 0.375 segundo, o que equivale a uma cadência de 160,0 RPM, e após isso, nenhum outro sinal do sensor.

A figura 5.7 mostra a resposta do algoritmo numa simulação onde o usuário está a uma cadência de 160,0 RPM e para de pedalar imediatamente – isto é feito desligando o MSP430 que simula o sensor *reed switch*. Os pontos vermelhos mostram o tempo que é enviado ao smartphone e a saída que este mostra ao usuário, ou seja, mesmo sem o sinal do *reed switch*, o microcontrolador continua atualizando o smartphone.

Em um primeiro pensamento, o usuário deveria esperar um determinado intervalo de tempo para que o algoritmo determinasse que a cadência era zero, oito segundos, por exemplo. Porém, nessa implementação, pode-se observar pelo gráfico, que a atualização da cadência vista pelo usuário torna-se mais sutil e agradável.

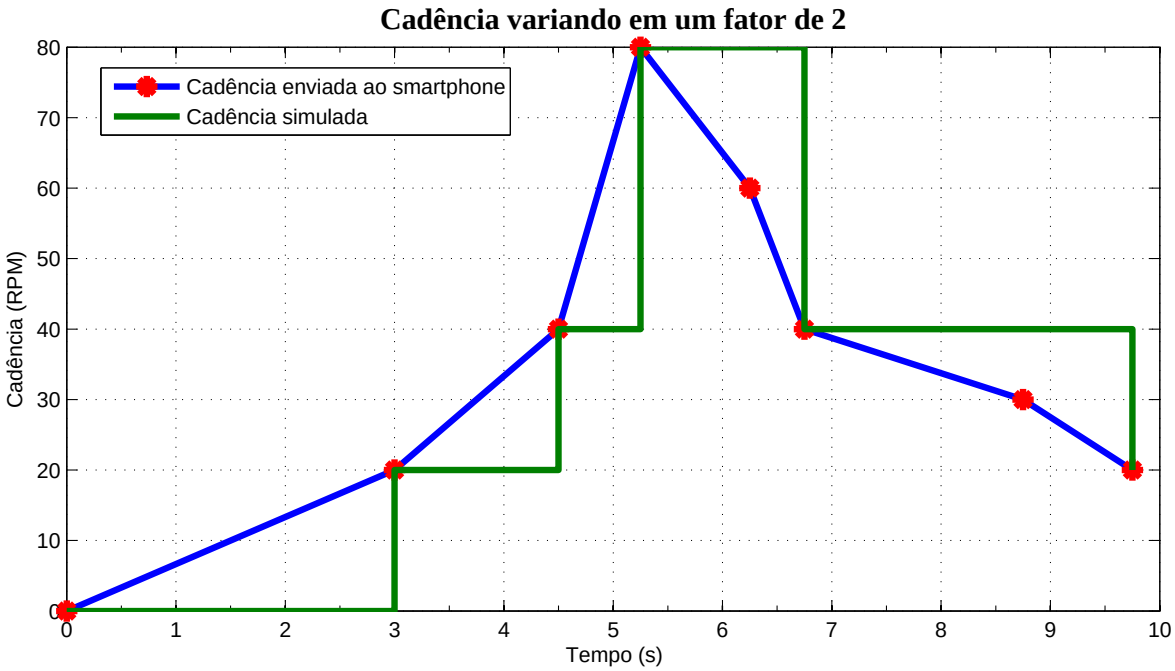

Figura 5.5: Gráfico da cadência variando em um fator de 2

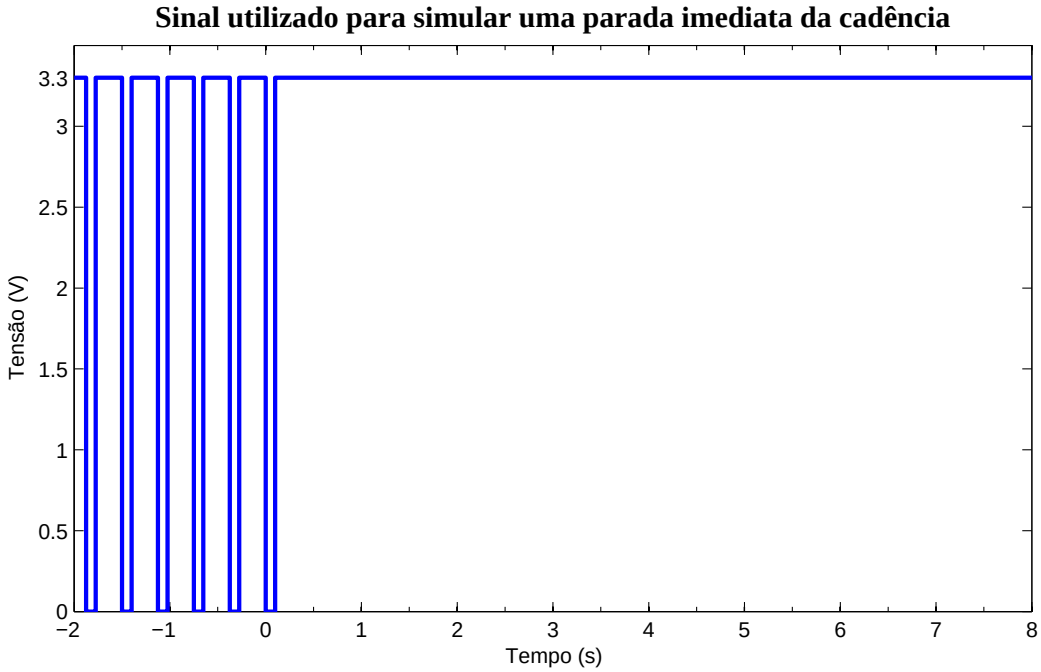

Figura 5.6: Gráfico do sinal utilizado simular a cadência em uma queda brusca

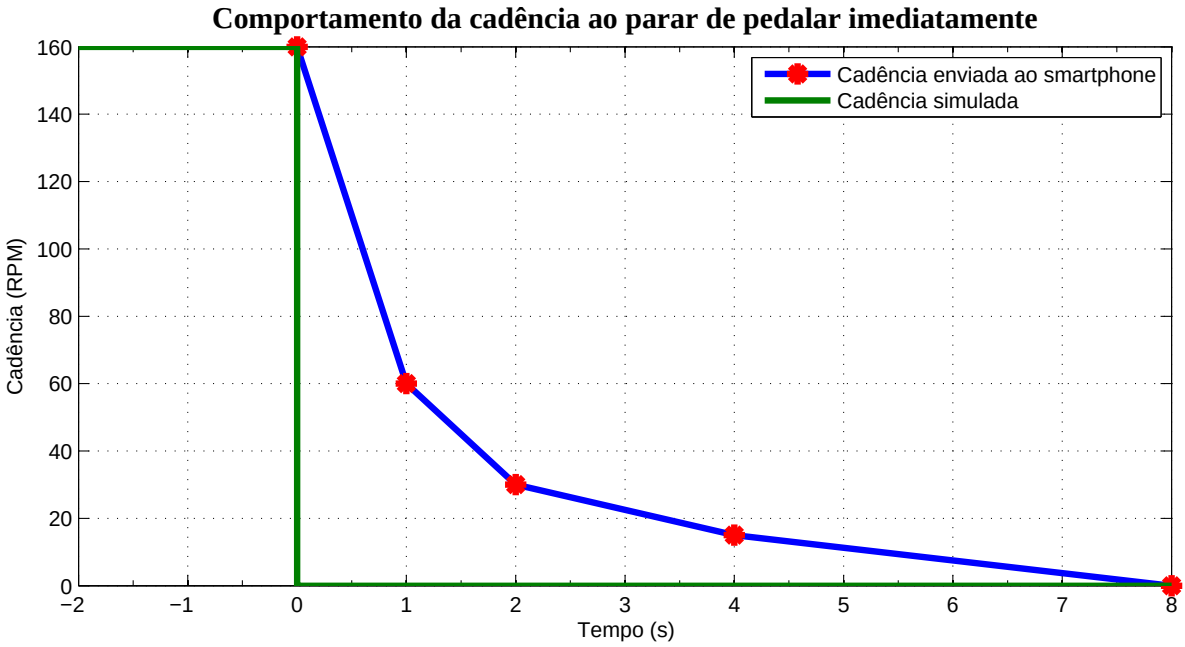

Figura 5.7: Gráfico da simulação de uma queda brusca na cadência

5.3 Testando o aplicativo Android

Os métodos principais do aplicativo Android, ou seja, os responsáveis por fazer a interação com o sistema embarcado (cálculo do tempo, velocidade e cadência, por exemplo), foram submetidos a testes unitários, utilizando o *framework* Mockito – que facilita a criação de *mocks* para os testes. O objetivo de cada teste é isolar uma unidade (utilizando *mocks* que imitam as dependências reais) e submetê-la a entradas que produzem resultados conhecidos.

As funcionalides mapas e GPS foram testadas marcando um percurso e percorrendo-o para verificar que o aplicativo estava identificando o trajeto do usuário.

5.4 Testando o sistema embarcado na prática

Este teste foi feito utilizando uma bicicleta e o sistema embarcado completo, ou seja, uma simulação de como o produto seria utilizado na prática. O objetivo foi comparar os valores de cadência e velocidade com um produto da Garmin, que já está no mercado há muito tempo e utiliza outros meios para calcular essas medidas – acelerômetro e GPS. A figura 5.8 mostra a configuração utilizada que permite observar os dois ao mesmo tempo.

Os testes foram conduzidos no Eixo Rodoviário de Brasília (DF-002), em um domingo, quando ele é fechado para os carros e se transforma em um local de lazer. A figura 5.9 mostra o cenário onde os experimentos foram feitos.

Uma dificuldade encontrada no experimento foi a fixação do sistema embarcado na bicicleta. Por tratar-se de um protótipo, foi dado prioridade ao *software* e ao *hardware* do produto, sem levar muito em consideração a parte estética. Portanto, foi utilizado uma fita crepe para manter o circuito preso a bicicleta, e assim, evitando que ele caisse durante

Figura 5.8: Os dois sistemas utilizados para comparação

Figura 5.9: Eixo Rodoviário de Brasília: local onde os testes foram conduzidos

os experimentos. O resultado disso pode ser visto na figura 5.10. Essa configuração foi utilizada para testar a velocidade, é possível ver o sensor *reed switch* – ligado ao circuito por dois *jumpers* vermelho e preto – próximo ao ímã que está parafusado no raio da bicicleta.

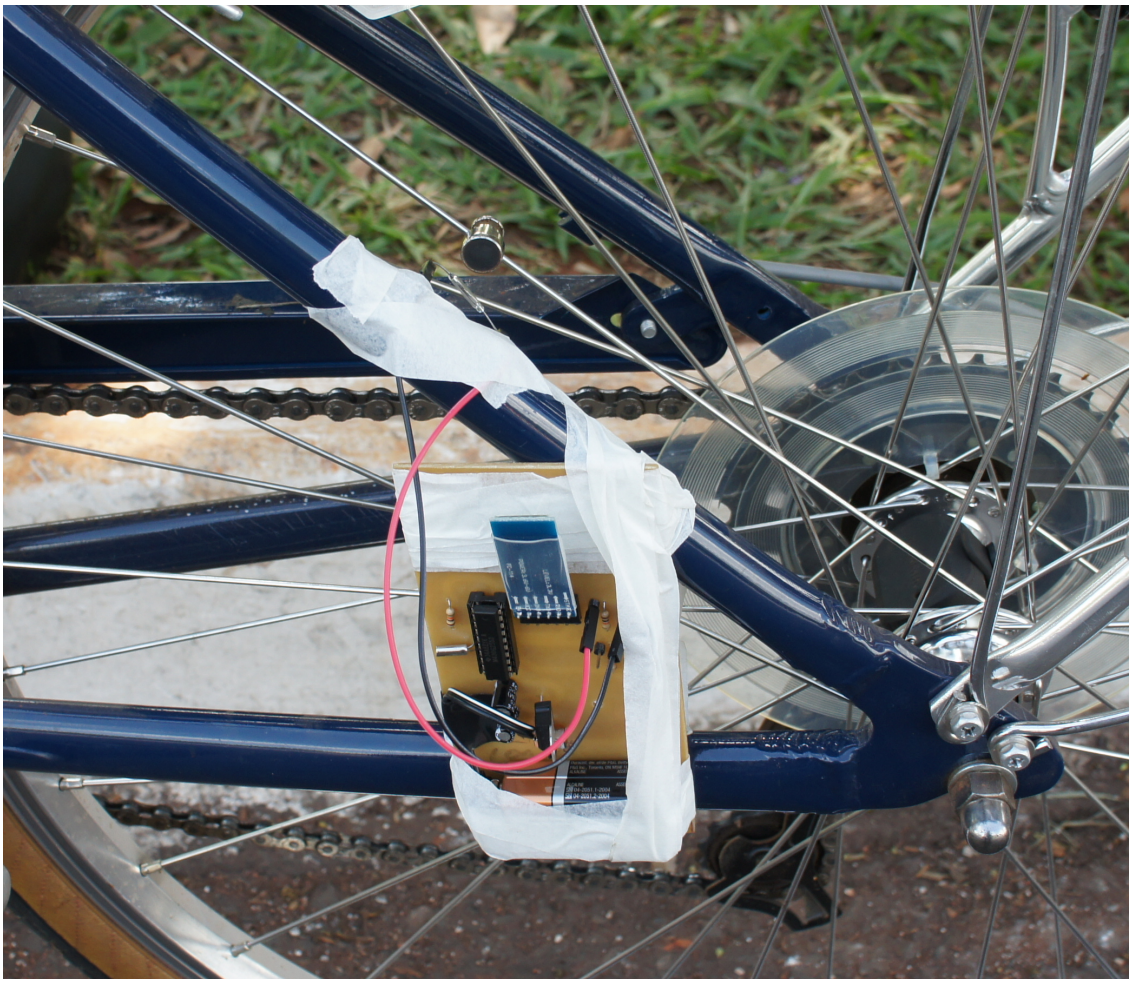

Figura 5.10: Sistema embarcado fixado na bicicleta para medir a velocidade

Para a cadência, apenas moveu-se o *reed switch* e o ímã de lugar, sendo que este último agora é preso no pedal da bicicleta. A figura 5.11 mostra como ficou na prática.

Os resultados foram satisfatórios e dentro do esperado. Os valores estavam bem próximos entre os dois produtos comparados, sendo a cadência a que mais se aproximou, já que para fazer essa medida, não tem-se nenhum erro associado ao tamanho da roda.[3] Outro aspecto notável foi a conexão bluetooth, que se mostrou estável durante todo o período de testes, sendo que em nenhum momento foi preciso restabelecer a conexão por motivo de falha.

[3] O aplicativo Android só estava preparado para receber o tamanho da roda como um inteiro em polegadas, isso contribuiu para a pequena diferença na velocidade entre os produtos, por não se considerar alguns centímetros do pneu.

Figura 5.11: Sistema embarcado fixado na bicicleta para medir a cadência

Capítulo 6

Conclusões e trabalhos futuros

Sendo o ciclismo um meio de transporte em constante crescimento e a necessidade que surge em monitorar esse tipo de atividade, este projeto tem o objetivo de ser uma solução para uma potencial demanda entre os ciclistas que necessitam monitorar algumas medidas durante os percursos. Nesse cenário, o sistema tem como finalidade medir a cadência e velocidade da bicicleta com uma boa precisão utilizando um sistema embarcado específico e usar a praticidade e baixo custo do *smartphone* com aplicativo Android para ser toda a interface do usuário.

A solução foi dividida em dois componentes: sistema embarcado e aplicativo Android. Sendo o primeiro, um circuito elétrico utilizando um microcontrolador MSP430 para calcular o tempo de revolução da roda e do pedal, sendo que sensores *reed switch* foram utilizados para captar essas revoluções. O segundo é um *software* para *smartphone* que utiliza os valores recebidos do sistema embarcado para calcular, diretamente, a velocidade e a cadência da bicicleta e, indiretamente, outras informações úteis ao ciclista. Além disso, o *software* implementado para o *smartphone* possui outras funcionalidades envolvendo mapas, GPS e persistência de estatíticas. A comunicação entre eles é via *bluetooth* e o módulo HC-05 é o responsável por transformar a comunicação serial que sai do microcontrolador em *bluetooth* que chega ao *smartphone*.

Os dois componentes do projeto, sistema embarcado e aplicativo Android, foram concluídos com sucesso. Os testes comprovaram que o sistema embarcado está com uma boa precisão e equivalente a um produto real. O aplicativo Android também se mostrou robusto, visto que, durante os testes finais não houve falhas que afetassem o uso do produto. Um fator de destaque foi a inclusão de percursos utilizando mapas e GPS no *software* do *smartphone*, aumentando a imersão do ciclista durante o uso da aplicação. Além disso, a conexão bluetooth feita com o módulo HC-05 mostrou-se estável durante todos os experimentos. Portanto, a solução final tem potencial para ser uma alternativa ao ciclista que deseja monitorar seu desempenho, trajeto e guardar estatísticas para consultas futuras.

Por tratar-se de um protótipo, o sistema embarcado ainda precisa de algumas melhorias para transformar-se em um produto real. Primeiramente, o seu tamanho pode ser reduzido utilizando algum microcontrolador com *bluetooth low energy* no mesmo circuito integrado, por exemplo, consequentemente, isso também economizaria energia, podendo trocar a bateria por uma de menor porte. Na questão mecânica, ainda necessita de um suporte para facilitar a tarefa de fixação na bicicleta.

Referências

[1] Incentivo ao uso da bicicleta: uma tendência mundial. http://sustentarqui.com.br/urbanismo-paisagismo/incentivo-ao-uso-da-bicicleta-uma-tendencia-mundial/. Acesso em: 15 de dezembro de 2015. 1

[2] Oliver O'Brien, James Cheshire, and Michael Batty. Mining bicycle sharing data for generating insights into sustainable transport systems. *Journal of Transport Geography*, 34:262 – 273, 2014. 1

[3] Minhas trilhas. https://play.google.com/store/apps/details?id=com.google.android.maps.mytracks&hl=pt_BR. Acesso em: 15 de dezembro de 2015. 1

[4] Strava. http://www.strava.com/mobile. Acesso em: 15 de dezembro de 2015. 1

[5] Garmin: ciclismo. https://buy.garmin.com/pt-BR/BR/fitness-e-outdoor/ciclismo/cIntoSports-cCycling-p1.html. Acesso em: 15 de dezembro de 2015. 1

[6] MAURIZIO DI PAOLO EMILIO. Embedded system design. *Electronics World*, 120:30–32, February 2014. 2

[7] Fast-facts. http://www.bluetooth.com/Pages/Fast-Facts.aspx. Acesso em: 20 de junho de 2015. 5

[8] P. McDermott-Wells. What is Bluetooth? *IEEE Potentials*, 23(5):33–35, December 2005. 5, 6

[9] Bluetooth basics. http://www.bluetooth.com/Pages/Basics.aspx. Acesso em: 20 de junho de 2015. 6

[10] E. Ferro and F. Potorti. Bluetooth and wi-fi wireless protocols: a survey and a comparison. *IEEE Wireless Communications*, 12(1):12–26, February 2005. 6

[11] Bluetooth. http://ows.edb.utexas.edu/site/collaborative-bluetooth-edumanet/bluetooth. Acesso em: 21 de junho de 2015. 7

[12] Introduction to android. http://www.beginandroid.com/intro.shtml. Acesso em: 23 de julho de 2015. 7

[13] The android story. http://www.xcubelabs.com/infographic-android-story. Acesso em: 23 de julho de 2015. 8

[14] The android architecture. https://commons.wikimedia.org/wiki/File:Android-System-Architecture.svg. Acesso em: 23 de julho de 2015. 8

[15] Application fundamentals. http://developer.android.com/intl/ru/guide/components/fundamentals.html. Acesso em: 24 de julho de 2015. 9

[16] Valentin Corneliu Pau, Marius Iulian Mihailescu, and Octavian Stanescu. Model View Presenter Design Pattern. *Journal of Computer Science & Control Systems*, 3(1), May 2010. 10

[17] Texas Instruments. *MSP430x2xx Family: User's Guide*, December 2004. 11, 13, 14, 15

[18] Texas Instruments. *MSP430G2x53 e MSP430G3x13: Device specific*, April 2011. 12, 13, 18

[19] Creative commons. http://creativecommons.org/licenses/by-nc-sa/3.0/. Acesso em: 06 de agosto de 2015. 12, 16

[20] John H. Davies. *MSP430 Microcontroller Basics*. Elsevier, 2008. 13

[21] Reed switch. http://www.if.ufrgs.br/mpef/mef004/20061/Cesar/SENSORES-Reed-switch.html. Acesso em: 03 de agosto de 2015. 17

[22] ITead Studio. *HC-05: Bluetooth to Serial Port Module*, July 2010. 17

[23] Duracell. *Alkaline-Manganese Dioxide Battery: Coppertop, size AAA*. 18

[24] STMicroelectronics. *LD1117 SERIES*, December 2005. 18

[25] Duracell. *Alkaline-Manganese Dioxide Battery: Coppertop, size 9V*. 19

[26] Dagger. http://square.github.io/dagger/. Acesso em: 17 de outubro de 2015. 29

[27] BluetoothAdapter | Android Developers. http://developer.android.com/intl/ru/reference/android/bluetooth/BluetoothAdapter.html. Acesso em: 08 de outubro de 2015. 30

[28] BluetoothDevice | Android Developers. http://developer.android.com/intl/ru/reference/android/bluetooth/BluetoothDevice.html. Acesso em: 08 de outubro de 2015. 30

[29] BluetoothSocket | Android Developers. http://developer.android.com/intl/ru/reference/android/bluetooth/BluetoothSocket.html. Acesso em: 08 de outubro de 2015. 30

[30] Google Maps Andoid API. https://developers.google.com/maps/documentation/android-api/. Acesso em: 14 de novembro de 2015. 34

[31] brModelo. http://www.sis4.com/brmodelo/download.aspx. Acesso em: 13 de novembro de 2015. 35

Apêndice A

Códigos fontes

O código fonte do aplicativo Android e do microcontrolador MSP430 foram colocados em um repositório online e podem ser vistos a partir dos *links* a seguir:

Aplicativo Android: https://github.com/charlesbts/BikeX-Android

Microcontrolador: https://github.com/charlesbts/BikeX-MSP430