UNIVERSIDADE ESTADUAL DO MARANHÃO – UEMA
CENTRO DE CIÊNCIAS TECNOLÓGICAS
CURSO DE ENGENHARIA DE COMPUTAÇÃO

ANTONIO IAGO LEMOS GASPAR

DESENVOLVIMENTO DE APLICATIVO PARA AUXÍLIO DE LOCOMOÇÃO DE PESSOAS COM DEFICIÊNCIA VISUAL

São Luís, MA
2016

ANTONIO IAGO LEMOS GASPAR

DESENVOLVIMENTO DE APLICATIVO PARA AUXÍLIO DE LOCOMOÇÃO DE PESSOAS COM DEFICIÊNCIA VISUAL

Trabalho de Graduação apresentado ao Curso de Engenharia de Computação da Universidade Estadual do Maranhão(UEMA), como requisito para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: MSc. REINALDO DE JESUS DA SILVA

Coorientador: Msc. MÁRIO RENATO NUNES CRISPIM DA SILVA CHAVES LIMA

São Luís, MA

2016

Gaspar, Antônio Iago Lemos.
Desenvolvimento de aplicativo para auxílio de locomoção de pessoas com deficiência visual / Antônio Iago Lemos Gaspar.–São Luís, 2016.
77 f

Monografia(Graduação) – Curso de Engenharia de Computação, Universidade Estadual do Maranhão, 2016.

Orientador: Prof. Msc. Reinaldo de Jesus da Silva.

1.Aplicativo. 2.Deficiente visual. 3.Locomoção. 4.Acessibilidade. I.Título

CDU:004.4-056.262

ANTONIO IAGO LEMOS GASPAR

DESENVOLVIMENTO DE APLICATIVO PARA AUXÍLIO DE LOCOMOÇÃO DE PESSOAS COM DEFICIÊNCIA VISUAL

Trabalho de Graduação apresentado ao Curso de Engenharia de Computação da Universidade Estadual do Maranhão(UEMA), como requisito para obtenção do grau de Bacharel em Engenharia de Computação.

Trabalho aprovado. São Luís, MA, 28 de janeiro de 2015:

MSc. REINALDO DE JESUS DA SILVA
Orientador

Msc. MÁRIO RENATO NUNES CRISPIM DA SILVA CHAVES LIMA
Coorientador

Msc. DIÓGENES CARVALHO AQUINO
Primeiro membro

Dedicatória

Este trabalho é dedicado a Deus, aos meus pais e familiares por todo o apoio que me deram nesta grande jornada.

Agradecimentos

A Deus, por ter me concedido experiências fantásticas de aprendizado, paciência e perseverança e por proporcionar esta grande conquista em minha vida.

Ao meu orientador e co-orientador, que se dispuseram a dedicar parte de seu tempo para acompanhar esta pesquisa e por agregar bastante valor a mesma.

A todos os meus familiares e amigos, que durante essa jornada me apoiaram e me deram forças para atingir meus objetivos, meu muito obrigado.

"Comece fazendo o que é necessário,
depois o que é possível e de repente
você estará fazendo o impossível."
(São Francisco de Assis)

Resumo

A seguinte pesquisa propõe o desenvolvimento de um aplicativo que possa guiar deficientes visuais de sua posição atual até o destino desejado. Abordam-se as tecnologias utilizadas para o desenvolvimento da aplicação. Destacam-se conceitos acerca da deficiência visual e dos processos que compoem a atividade de locomoção. Verifica-se na presente pesquisa o desenvolvimento da aplicação de forma a proporcionar acessibilidade.

Palavras-chaves: Aplicativo, deficiente visual, locomoção, acessibilidade.

Abstract

The following research proposes the development of an application that can guide the blind from your current location to the desired destination. It addresses the technologies used for application development. Stand out concepts about visual impairment and processes that make up the transportation activity. It is noted in this study the development of the application so as to provide accessibility.

Key-words: Application, Visually impaired, Locomotion, Accessibility.

Sumário

1 INTRODUÇÃO

A deficiência visual é um problema que atinge uma parte considerável da população brasileira. Com a deficiência visual, vem a limitação de locomoção. Os deficientes são mais dependentes de outras pessoas para realizar tarefas que exigem movimentação.

Sengundo o censo do IBGE de 2010, cerca de 45 milhões de pessoas possuem algum tipo de deficiência. Dessas 45 milhões de pessoas, 35 milhões possuíam deficiência visual. Em relação ao gênero, cerca de 16% da população masculina sofria de deficiência visual enquanto que a população feminina era de 21,4%. Essas pessoas foram cada vez mais sendo amparadas e vistas pela sociedade. Atualmente, cerca de 63,7% dos homens que possuem deficiência visual estão trabalhando. Cerca de 43,9% das mulheres que possuem deficiência visual possuem um emprego. Vemos que cada vez mais esta população participa da sociedade ativamente, contribuindo para o crescimento da mesma e, assim como ela participa, ela também está disposta a investir em si próprio, de forma que ela possa crescer não só profissionalmente como também pessoalmente, buscando cada vez mais ferramentas ou métodos que possam auxiliá-la a superar suas limitações físicas. Em se tratando de um deficiente visual, a maior limitação, podemos dizer assim, é a de locomoção. Buscar superar essas limitações é importante, pois quanto mais ele estiver preparado, ele puder realizar tarefas antes difíceis por conta da dificuldade de locomoção, mais autoestima e confiança o deficiente visual terá, contribuindo de forma indireta para o desenvolvimento do país.

Diversas são as ferramentas que foram desenvolvidas para ajudar deficientes visuais a se locomoverem. Temos por exemplo o projeto *GuideMe*, um projeto realizado pelos estudantes da USP de São Carlos: Francisco Monaco, Rene Pinto, Luiz Henrique Nunes e Heitor Freitas, que consiste em um dispositivo acoplado a uma câmera e, através do reconhecimento de padrões, reconhece obstáculos próximos e informa o usuário sobre esses obstáculos para um possível desvio (PAULO, 2014). Entretanto, até o momento do desenvolvimento da presente pesquisa, o projeto foca-se em detecção de obstáculos, não informando uma possível rota ou então a direção que o usuário deve seguir para chegar ao seu destino.

A presente pesquisa consiste em propor uma solução caracterizada como "*wearable*" - tecnologia vestível - que informará o caminho que o usuário deverá seguir através de um dispositivo até alcançar o seu destino. Esta funcionalidade será proporionada através do aplicativo FollowVibe conectado a uma luva adaptada com micromotores que indicarão a direção por meio de vibração.

As principais contribuições que pretende-se com o desenvolvimento desta pesquisa são:

- Proporcionar um sistema que auxilie deficientes visuais a locomoverem-se a lugares desejados de forma mais independente;

- Incentivar o aprimoramento do sistema de locomoção ou o desenvolvimento de projetos

que sigam a área de acessibilidade por parte de futuros desenvolvedores;

- Produzir conhecimento no desenvolvimento de aplicações móveis que trabalham no ambiente da acessibilidade.

1.1 Problema

Como garantir locomoção independente ao deficiente visual sem utilizar a audição do mesmo?

1.2 Hipótese

O desenvolvimento de um sistema móvel que determine uma rota dados dois pontos e forneça o direcionamento por interface tátil.

1.3 Objetivos

1.3.1 Objetivo Geral

Esta pesquisa tem como objetivo desenvolver um aplicativo móvel para o sistema operacional Android com o intuito de auxiliar pessoas com deficiência visual a se locomoverem até um destino desejado.

1.3.2 Objetivos Específicos

- Estudar os mecanismos de desenvolvimento de aplicativos para o sistema operacional Android;
- Desenvolver um aplicativo Android que atenda quesitos de acessibilidade ao deficiente visual;
- Realizar testes no aplicativo em um ambiente mapeado;
- Interpretar os resultados obtidos e avaliar a viabilidade do uso do aplicativo.

1.4 Justificativa

O que motivou a desenvolver esse trabalho foi a falta de opções no mercado de dispositivos para deficientes visuais que contemplassem a mobilidade com foco no georeferenciamento e a utilização de interface tátil.

A maioria das soluções oferecidas pelo mercado são focadas ou na utilização da audição como interface ou na detecção de obstáculos.

1.5 Apresentação da pesquisa

Esta pesquisa está dividida em 5 capítulos, dispostos da seguinte maneira a saber:

- O capítulo 1 aborda a introdução da pesquisa. Ela aponta os objetivos gerais e específicos da mesma, a motivação que impulsionou o desenvolvimento do projeto, as contribuições desta pesquisa e trabalhos relacionados que seguem a mesma área de auxílio aos deficientes visuais;

- O capítulo 2 trabalha com a fundamentação teórica que serve de base de conhecimento para o desenvolvimento do projeto. Apresentamos os conceitos da orientação a objetos, utilizados no desenvolvimento do aplicativo, os conceitos da linguagem de programação Java, que possui como paradigma de programação a orientação a objetos e conceitos relacionados ao desenvolvimento de aplicações Android, plataforma escolhida para o desenvolvimento do aplicativo. Veremos como funciona uma aplicação, o ciclo de vida dela no sistema e as bibliotecas necessárias utilizadas para desenvolver o aplicativo;

- O capítulo 3 aborda a pesquisa em si. Será discorrido sobre o projeto, seu modelo conceitual e a estrutura da aplicação desenvolvida;

- O capítulo 4 mostrará os resultados dos testes feitos e a avaliação dos resultados, mostrando se o sistema é viável ou se ainda é necessário mais aprimoramentos;

- O capítulo 5 traz as conclusões e considerações finais da pesquisa.

No apêndice temos o código-fonte do aplicativo desenvolvido. Todos os métodos e classes referenciados nesta pesquisa monográfica podem ser encontrados na seção apêndice para análise.

1.6 Trabalhos relacionados

Para o desenvolvimento do projeto, necessitamos buscar trabalhos que seguem o mesmo objetivo, ou seja, auxílio de pessoas com deficiências visuais na modalidade de locomoção. Observamos como eles trabalham e, a partir desta análise, verificaremos como o nossa pesquisa pode trazer um valor diferenciado para este objetivo.

O estudante Luiz Ricardo Fiera, da Universidade do extremo sul catarinense (Unesc) propos em seu trabalho de conclusão de curso um aplicativo que ajuda o deficiente visual a se locomover até um destino escolhido, utilizando para isso um guia auditivo (FAPESP, 2015). O aplicativo chama-se ViaVoz. A aplicação funciona da seguinte forma: o usuário escolherá, dentre uma série de opções, o destino desejado. Por meio do guia auditivo, o deficiente visual poderá se deslocar da sua origem até o destino escolhido. O processo de mapeamento é realizado por meio de sensores espalhados pelo ambiente. O software possui algoritmos que detectam os sinais dos sensores e constrói a melhor rota para o usuário (VIAVOZ, 2015).

Em Curitiba foi desenvolvido um trabalho de conclusão de curso pelo aluno Alessandro Cardozo Bueno, a respeito de uma bengala eletrônica que auxilia deficientes visuais a desviarem de obstáculos durante a locomoção (BUENO, 2010). A bengala emite ondas ultrassônicas que detectam um obstáculo acima ou abaixo da linha da cintura do usuário. Com dois micromotores, a vibração de um ou de outro indicará se o obstáculo está acima ou se está abaixo do usuário (BUENO, 2010). Como vimos, o trabalho de Alessandro busca resolver um problema de detecção de obstáculos, que é complementar à pesquisa proposta na presente monografia, que propõe uma solução no quesito locomoção de um ponto "A"ao ponto "B".

No estado do Pará, no Centro Universitário do Pará (CESUPA), os estudantes José de Sousa Ribeiro Filho, Rodrigo Vieira Cavalcante, Alessandra Natasha e Rejane de Barros, da Área de Ciências Exatas e Tecnologia, escreveram um artigo sobre o projeto Argos - auxílio à locomoção de deficientes visuais a partir de pulseira microcontrolada (FILHO et al., 2011). Este projeto consiste em uma pulseira que estará conectada a um microcontrolador. Este, por sua vez, conterá sensores que detectarão obstáculos acima da altura da cintura do deficiente visual. Quando detectado, o microcontrolador envia um sinal para a pulseira, que vibrará informando o usuário do obstáculo. Este projeto, assim como o projeto proposto por Alessandro utilizando uma bengala, ajuda a descobrir obstáculos abaixo e acima da altura da cintura. Entretanto, o foco destes projetos está na detecção de obstáculos. O projeto desenvolvido nesta presente pesquisa pode complementar os projetos propostos acima, possibilitando um conjunto completo e seguro de locomoção, onde o usuário poderá chegar a um destino desejado de forma mais independente, evitando possíveis obstáculos durante o caminho.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção iremos explanar os conceitos que giram em torno do projeto desta monografia. Eles proporcionam uma base sólida e um bom alicerce para sustentar a ideia que o projeto apresenta. Iremos abordar alguns conceitos de deficiência visual, conceitos sobre as tecnologias utilizadas como Java e Android, e conceitos de orientação a objetos, que proporcionam um código mais elegante, coerente e de fácil manutenção.

2.1 Principais dificuldades de um deficiente visual

"A deficiência visual, em qualquer grau, compromete a capacidade da pessoa de se orientar e de se movimentar no espaço com segurança e independência" (GIL, 2000). Sá (2005) afirma que, de acordo com suas pesquisas, um dos principais meios de auxílio à locomoção de deficientes visuais é a bengala. Os deficientes visuais dependem ainda de outras pessoas para identificar ruas, endereços, avisos e obstáculos, dentre outros. São constantes os riscos que os deficientes visuais correm ao se locomoverem.

Segundo Telford e Sawrey (1972) existem fatores que atingem indivíduos com baixa ou nenhuma visão: a compreensão social, a educação por meios não-visuais e a mobilidade independente. Destes, o foco desta pesquisa encontra-se em solucionar o problema da mobilidade independente.

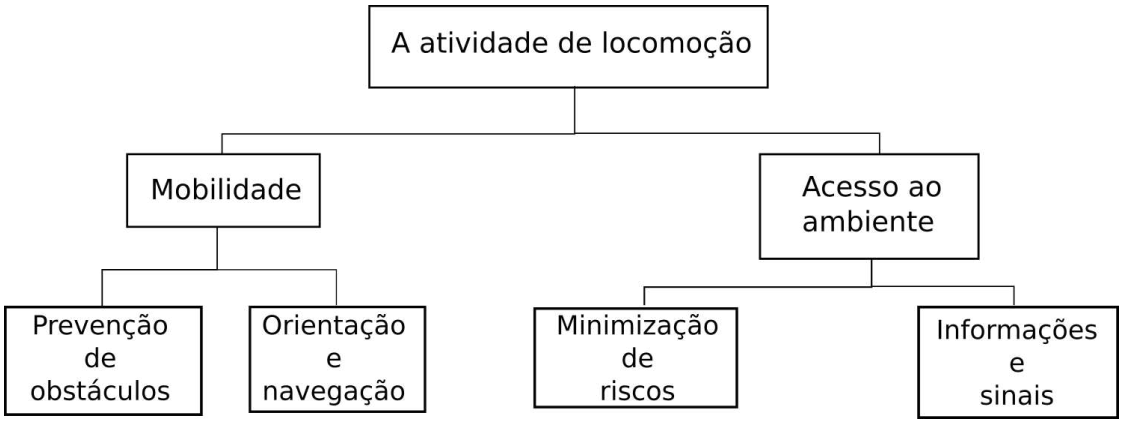
2.2 Mobilidade

O dia-a-dia da sociedade envolve, dentre outras atividades, a atividade de locomoção. A locomoção é o ato de mover-se de um local para outro (HERSH; JHONSON, 2008). O processo de locomoção envolve diversas etapas. Tomemos como exemplo uma viagem de casa até a padaria. Este processo pode ser descrito pelos seguintes passos:

- Saber o caminho a ser seguido;
- Desviar dos obstáculos;
- Modificar a direção quando necessário;
- Reconhecer o local de chegada;
- Realizar a tarefa de comprar pão;
- Realizar os passos 1 a 4 até o destino casa.

Essas tarefas são incluídas em duas categorias principais: mobilidade e acesso ao ambiente. Vejamos a seguir na figura 1 as categorias e subcategorias do processo de locomoção.

Figura 1 – Os processos inerentes na atividade de locomoção adaptada.



Fonte: (HERSH; JHONSON, 2008)

O escopo desta pesquisa envolve as tarefas de mobilidade, em especial as tarefas de orientação e navegação. Prever obstáculos é a habilidade de reconhecer possíveis objetos perigosos no ambiente em um período de tempo necessário para evitá-los. Orientação refere-se a habilidade de manter posições consistentes no espaço em intervalos curtos de comprimento, ou seja, a capacidade de uma pessoa de se locomover em uma rota realizando o menor número de desvios por causa de obstáculos. Para esta tarefa são necessários rotas ideais sem obstáculos. A navegação consiste na habilidade de determinar uma rota de locomoção. É denominado pela informação e estratégia de como sair de um ponto A e chegar ao ponto B (HERSH; JHONSON, 2008).

Para o deficiente visual, manter posições consistentes e definir rotas torna-se um trabalho muito difícil, devido a sua limitação. A presente pesquisa visa trabalhar na solução para este problema, abstraindo o processo de locomoção e o processo de formulação da rota, ou seja, o sistema irá realizar o processo de determinação da rota e o processo de direcionamento.

2.3 Programação orientada a objetos

A orientação a objetos teve início nos anos 60 na Noruega, com Kristen Nygaard e Ole-Johan Dahl. Eles desenvolveram a linguagem Simula 67, que implementava conceitos de classe e herança (WEBGOAL, 2009).

Entretanto, a orientação a objetos se popularizou através de Alan Kay, considerado como um dos criadores do termo “programação orientada a objetos” (WEBGOAL, 2009). Alan desenvolveu em Simula 67 e se interessou bastante pelos conceitos ali abordados. Juntando com seus conhecimentos de biologia e matemática, Alan formulou sua analogia “algébrico-biológica” (WEBGOAL, 2009).

De uma forma geral, Alan Kay pensou em construir um sistema que funcionasse a partir de componentes autônomos que poderiam cooperar entre si para realizar uma tarefa. A partir

disso, ele estabeleceu os seguintes princípios de orientação a objetos (BEZERRA, 2007):

- Qualquer coisa é um objeto;
- Objetos realizam tarefas através de requisições de serviços a outros objetos;
- Cada objeto pertence a uma determinada classe. Uma classe agrupa objetos similares;
- Uma classe possui comportamentos associados ao objeto;
- Classes são organizadas hierarquicamente.

Atualmente o paradigma orientado a objeto é um dos mais utilizados pela comunidade de desenvolvedores. O ponto chave é a reusabilidade de código: componentes anteriormente desenvolvidos podem ser aproveitados para futuras codificações, melhorando a clareza do código, a produtividade no desenvolvimento e a manutenibilidade do sistema.

Segundo Bezerra (2007), um sistema de software orientado a objetos consiste em objetos que colaboram entre si para realizar funcionalidades do sistema. Cada objeto é responsável por tarefas específicas, que juntas podem resolver tarefas mais complexas. Por exemplo, para construir um prédio, precisamos de arquitetos, engenheiros, mestres-de-obra, pedreiros, eletricistas, encanadores, etc. Cada indivíduo possui características, como grau de conhecimento, altura, idade, cor dos olhos, sexo, etc, e possui uma série de ações, por exemplo, realizar projeto, sentar tijolo, preparar massa. Quando estes elementos se unem e realizam suas tarefas, podemos atingir a tarefa de construir o prédio. De forma semelhante funciona na programação orientada a objetos: os objetos interagem entre si para realizar tarefas solicitadas pelo usuário do sistema.

Quando falamos de objetos, alguns conceitos são importantes para entendermos como todo esse paradigma funciona de fato na programação. Os conceitos principais são: classe, objeto, encapsulamento, herança e polimorfismo. Vejamos com mais detalhes cada um deles.

2.3.1 Classe

O mundo é constituído de diversas coisas. Como exemplo, podemos citar um homem, uma mulher, um cachorro, um carro, uma geladeira. Mas entender cada um especificamente é um trabalho dispendioso. Portanto, o ser humano desenvolveu uma capacidade incrível de agrupar coisas semelhantes. A esse agrupamento denominamos classe. A classe representa a ideia por trás de vários elementos semelhantes, descrevendo os atributos e responsabilidades em comum. Segundo Bezerra (2007), a classe representa um molde a partir do qual essas coisas são construídas. Dentro da programação orientada a objetos, a classe é justamente um molde para a construção de objetos. Ela agrupa atributos e funcionalidades em comum para um conjunto de objetos.

2.3.2 Objeto

Vimos anteriormente que a classe representa a ideia comum a todos os elementos de um grupo. Cada elemento desse grupo representa uma concretização da classe. Dizemos que cada elemento é uma instância da classe. A essa instância damos o nome de objeto. O objeto é, portanto, um exemplar originado da classe. Por exemplo, temos a classe cavalo. Nela estão descritas todas as características e ações que um cavalo pode ter ou realizar. Logo após vemos dois cavalos, um branco e alto e o outro marrom com pintas brancas. Cada cavalo descrito é uma instância da ideia cavalo.

2.3.3 Encapsulamento

O encapsulamento é o conceito em orientação a objetos que define que detalhes internos de classes devem ser ocultos para os objetos (CARVALHO, 2014a). Dessa forma, o conhecimento da implementação interna da classe é desnecessário do ponto de vista do objeto (CARVALHO, 2014a). Isto significa que um objeto, quando solicita uma determinada tarefa a outro, não precisa saber sobre como esta tarefa será realizada, mas sim o que esta tarefa irá retorna para ele. Ao encapsularmos o comportamento, fornecemos uma interface de comunicação para que objetos externos possam solicitar tarefas entre si.

Tomemos como exemplo uma caixa preta de avião. Ela nos fornece uma interface apropriada para solicitarmos tarefas como salvar dados do voo e obtê-los posteriormente. Entretanto, não sabemos como ela faz isso: fica encapsulado. Isso é muito importante, pois ao encapsularmos nossos objetos, evitamos que solicitações externas modifiquem de forma equivocada o estado do mesmo. Além disso, nossa aplicação torna-se mais flexível, uma vez que alterações serão feitas internamente ao objeto, mas sua interface de acesso será sempre a mesma.

2.3.4 Herança

A herança é a capacidade de criar uma classe absorvendo características e métodos de uma outra classe. A classe que herda as características e operações é chamada de subclasse. A classe existente que serviu de base para a subclasse é chamada de superclasse. Outras terminologias podem ser utilizadas, como classe básica para a superclasse e classe derivada para a subclasse, ou classe pai e classes filhas (DEITEL; DEITEL, 2010). Segundo Carvalho (2014b), a herança possibilita que uma classe compartilhe seus atributos, métodos e outros membros para as classes filhas.

Com a herança podemos reaproveitar o código existente. O tempo de desenvolvimento é reduzido, gerando, consequentemente, um aumento na produtividade (DEITEL; DEITEL, 2010). Vejamos um exemplo da realidade: a relação entre pessoas e funcionários. Cada pessoa possui características como idade, nome, sexo, etc. Os funcionários possuem as mesmas características, com algumas a mais próprias deles. Podemos dizer, então, que o funcionário é uma pessoa. Ao afirmar isso, dizemos que entre pessoa e funcionário existe uma hierarquia onde pessoa é a

entidade mais genérica e funcionário é uma especialização de uma pessoa, pois possui as mesmas características, com alguns atributos a mais e métodos específicos inerentes à sua entidade. Podemos dizer, então, que funcionário herda de pessoa.

2.3.5 Polimorfismo

A palavra polimorfismo é originário do grego (*poli* = muitas, *morphos* = formas) (WIKIPÉDIA, 2015c). O polimorfismo permite que classes possam se comportar como outras classes. Assim, é possível tratar várias classes diferentes de maneira homogênea. Tecnicamente falando, classes polimórficas possuem os mesmos métodos, mas implementados conforme suas necessidades. A entidade que utilizará o método não necessariamente precisa ter conhecimento da classe concreta que implementa o método, mas sim da classe abstrata que representa a interface. Vamos imaginar o seguinte exemplo: em um sistema existem as classes peixe, sapo e pássaro. Estas classes implementam a superclasse animal, que contém um método chamado mover. Todas as classes filhas implementam o método mover de acordo com suas necessidades. Ao desenvolver o programa, necessitamos que um animal se mova. Todos aqueles que são animais irão se mover. Entretanto, cada um realizará o movimento de maneira única. Mesmo o programa emitindo a mesma mensagem "mover" acaba possuindo diversas formas de resultado, daí o termo polimorfismo (DEITEL; DEITEL, 2010).

Com este conceito, podemos desenvolver sistemas bastante extensíveis. Novas classes podem ser desenvolvidas e adicionadas aos contextos gerais do sistema, com pouca ou nenhuma modificação, desde que as novas classes implementem os métodos das classes mais genéricas (DEITEL; DEITEL, 2010). Podemos imaginar como um contrato: uma nova classe assina o contrato de uma classe mais genérica para representá-la, mas é necessário que ela tenha capacidade de exercer as tarefas que a classe genérica define.

2.4 Java

A contribuição mais importante dada pelos microprocessadores foi a possibilidade de desenvolver computadores pessoais. Os computadores afetaram consideravelmente a vida das pessoas e a maneira que organizações conduzem e gerenciam seus negócios (DEITEL; DEITEL, 2010).

Deslumbrando esse cenário, não só de computadores inteligentes mas de eletrônicos inteligentes, a Sun Microsystems, em 1991, financiou um projeto de pesquisa interno que culminou em uma nova linguagem de programação, baseada em C++, que foi chamada inicialmente de Oak, devido ao fato de que seu criador e líder do projeto, James Gosling, homenagear um carvalho que ele podia admirar através da janela de seu escritório (DEITEL; DEITEL, 2010). Mais tarde, descobriu-se que já existia uma linguagem de programação com esse nome. Alguns dias mais tarde, a equipe de desenvolvimento da nova linguagem visitou uma cafeteria local. O

café era exportado das ilhas Java. Neste contexto, foi sugerido como nome para linguagem o nome das ilhas cujo o café era exportado. O nome ganhou fama com o tempo.

O projeto de desenvolvimento passou por algumas dificuldades. Inicialmente ele foi concebido com o objetivo de integrar aparelhos eletrônicos através de uma única linguagem de programação. No entanto, a indústria de eletrônicos inteligentes não estava se desenvolvendo como a empresa havia previsto. Por outro lado, a internet se desenvolveu e se popularizou rapidamente. Foi neste ponto que Gosling repensou o projeto e adaptou a linguagem Java para prover dinamismo às páginas web, que na época utilizavam apenas HTML para a estruturação do conteúdo. Após o lançamento da linguagem, em 1995, diversas empresas e programadores começaram a utilizá-la. Grandes empresas, como a IBM, anunciaram o suporte para a tecnologia Java (WIKIPÉDIA, 2015b).

2.5 Android

O Android é um sistema operacional desenvolvido pela empresa Android, Inc., que era mantida pela Google financeiramente. Primeiramente, o time de desenvolvimento da Android, Inc. começou um projeto de um sistema operacional para câmeras digitais, mas logo perceberam que este mercado não era grande o suficiente. Então, o grupo começou a desenvolver um sistema operacional para dispositivos móveis (WIKIPÉDIA, 2015a).

Em 17 de agosto de 2005, a Google adquiriu a Android, Inc. Em 5 de novembro de 2007, foi fundada a Open Handset Alliance, um consócio entre empresas que desenvolvem para o amadurecimento do sistema operacional Android. Dentre essas empresas estão a Google, a Sony, a Samsung e operadoras de telefonia como a Sprint Nextel e a T-mobile. Esta aliança foi formada com o objetivo de propor um padrão aberto do sistema operacional, para que as empresas possam customizar o Android de acordo com suas necessidades e ainda sim receber as atualizações desenvolvidas pela Google. A primeira versão do Android foi desenvolvida com base na versão do kernel linux 2.6.25, instalado no celular HTC Dream, o primeiro smartphone com o sistema operacional, lançado em 22 de outubro de 2008 (WIKIPÉDIA, 2015a).

Em 2010, a Google lançou o seu primeiro smartphone, chamado de Nexus, em parceria com a HTC, que fabricou o celular. Desde então, novas versões do dispositivo foram desenvolvidas em parceira com outras empresas. Por exemplo, o Nexus 5 foi desenvolvido pela LG e o Nexus 7 foi desenvolvido pela Asus. A cada lançamento de um novo modelo, a Google demonstra as últimas atualizações do sistema operacional. A cada grande atualização, o sistema recebe um codinome, em ordem alfabética, baseado em um doce (WIKIPÉDIA, 2015a).

2.5.1 Fundamentos do desenvolvimento de aplicativos para Android

Os aplicativos Android são escritos em linguagem Java. O Android SDK compila o código em um arquivo APK – um arquivo Android Package – que contém todo o conteúdo

da aplicação. Este arquivo contém o sufixo .apk e é utilizado para instalar a aplicação em dispositivos com o sistema operacional Android.

O sistema operacional Android baseia-se em um sistema Linux multiusuário, onde cada aplicação é um usuário do sistema. Elas possuem uma identificação única no sistema, mas esta identificação somente é conhecida pelo sistema operacional. Assim o Android pode gerenciar as permissões de acesso a arquivos para cada usuário. Cada aplicação executa em um processo, e cada processo possui uma máquina virtual, que trabalha isoladamente das outras. Por padrão, um processo é aberto se algum componente da aplicação necessita ser executado e é interrompido caso o componente não seja mais necessário ou quando o sistema operacional necessita de memória para executar um outro aplicativo (DEVELOPERS, 2015a).

Seguindo este raciocínio, o Android implementa o princípio do menor privilégio. Isto significa que cada aplicação terá permissão para utilizar apenas os recursos necessários para realizar sua tarefa, e nada mais do que isso. Isso proporciona um ambiente bastante seguro, onde aplicações não irão acessar áreas para as quais não lhes foram permitidas.

Entretanto, podemos dar permissões para que aplicativos acessem os recursos do Android e também é possível que duas aplicações compartilhem dados entre si. É possível, por exemplo, que duas aplicações compartilhem dados entre si através do compartilhamento do mesmo ID de usuário do sistema. Dessa maneira, elas poderão acessar os arquivos umas das outras. Irão compartilhar o mesmo processo e a mesma máquina virtual, mas é necessário que elas sejam assinadas com o mesmo certificado (DEVELOPERS, 2015a).

2.5.2 Componentes de uma aplicação

Os componentes são os blocos essenciais de uma aplicação Android. Cada componente é uma forma que o sistema dispõe para acessar o aplicativo. Nem todos os componentes possuem uma interface com o usuário, e alguns dependem dos outros, mas cada um possui sua própria identidade e desempenha um papel específico na aplicação. Em conjunto, colaboram para o comportamento geral da aplicação.

Existem quatro tipos de componentes. Cada um possui uma tarefa específica no sistema e um ciclo de vida diferente, que determina como são criados e destruídos. A seguir veremos cada um desses componentes (DEVELOPERS, 2015a):

1. **Activity** - Uma atividade representa uma tela com uma interface de usuário. Por exemplo, podemos ter uma atividade que lista os contatos da agenda, outra atividade que realiza uma chamada e outra que cadastra um novo contato. Embora trabalhem juntas para desempenhar o objetivo geral do sistema, cada uma é independente da outra. Assim, uma outra aplicação pode executar qualquer uma dessas atividades se ela tiver permissões para isso. Por exemplo, uma aplicação de mensagens pode solicitar a atividade de cadastro para cadastrar o número relativo à mensagem recebida.

2. **Services** - Serviços são componentes que executam em segundo plano, realizando processamentos longos ou executando tarefas que não impeçam que o usuário utilize outras aplicações. Por exemplo, um serviço que executa uma música enquanto o usuário navega pela internet. Um serviço pode ser instanciado por um outro componente, como uma atividade, ou então o componente pode associar-se ao serviço para interagir com ele.

3. **Content providers** - Gerenciam os dados de uma aplicação. Podemos armazenar dados em um banco de dados SQLite, em um arquivo de sistema, na *web* ou em algum local de armazenamento que a aplicação possua acesso. O provedor de conteúdo gerencia o acesso a esses locais de armazenamento, provendo um meio para aplicações poderem ler ou escrever dados, se o provedor permitir que isso aconteça. Por exemplo, o sistema fornece um que gerencia a lista de contatos salvos. Caso a aplicação tenha as permissões devidas, ela pode acessar esse provedor e obter dados da lista e/ou escrever um dado novo.

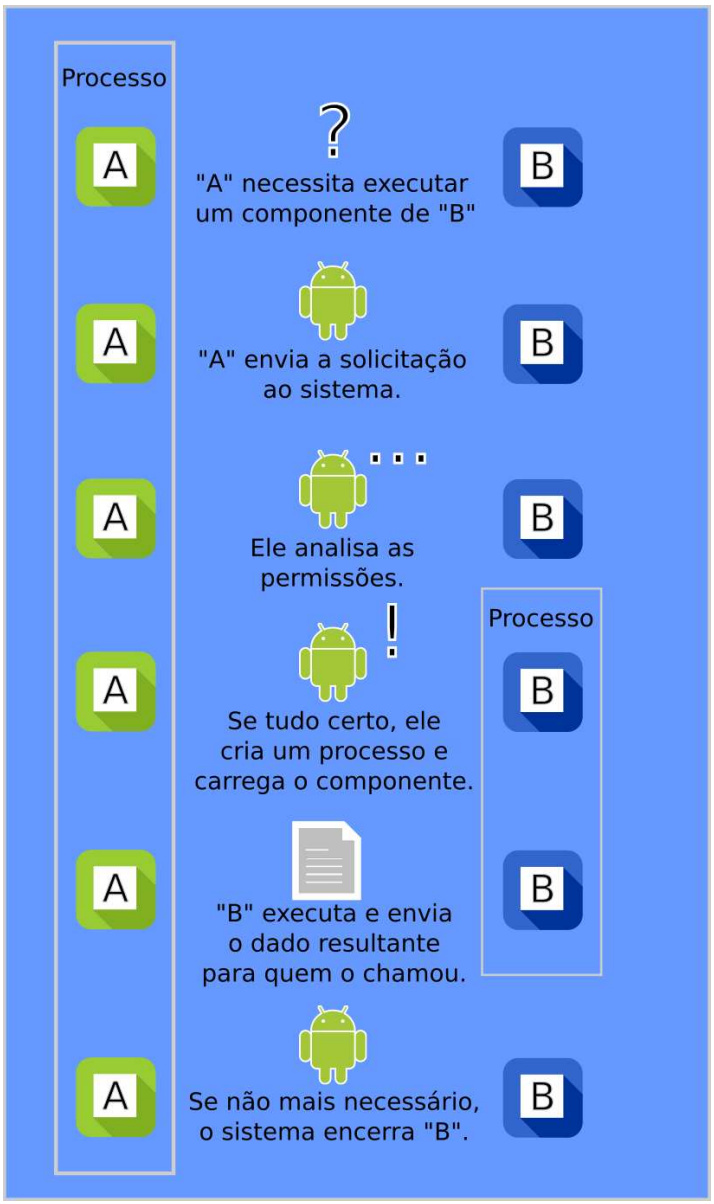
4. **Broadcast receivers** - São componentes que reagem a alguma publicação do sistema. Muitos *broadcasts* são disparados pelo sistema, como uma notificação de bateria baixa, ou uma notificação de desligamento de tela, entre outras. Uma aplicação também pode iniciar um *broadcast*. Por exemplo, ela pode emitir uma notificação para que outras aplicações saibam que um determinado recurso está pronto para ser usado. Embora *broadcasts* não possuam uma interface com o usuário, eles geram notificações para ele. Geralmente são utilizados como um caminho para que outros componentes sejam executados em relação ao evento ocorrido.

Um aspecto importante do sistema Android é que uma aplicação pode executar um componente de outra aplicação. Por exemplo, se você deseja tirar uma foto em seu aplicativo, você não necessita criar uma aplicação que faça isso. Talvez já exista um aplicativo que faça essa tarefa. Você tão pouco necessita realizar uma codificação para acoplar o código da aplicação de câmera. Simplesmente iniciamos a atividade da aplicação câmera que captura a foto. Uma vez tirada, ela retorna a foto para a sua aplicação. Para o usuário, é como se a aplicação de captura fizesse parte da aplicação.

Quando o sistema executa um componente, ele cria um processo para a aplicação (caso não exista) e carrega as classes necessárias para executar o componente. Portanto, quando um aplicativo inicia um componente de outro, este componente executará no processo cuja aplicação ele faz parte, e não no processo do aplicativo que o chamou. Percebe-se que as aplicações não possuem um ponto de entrada único, como uma função *main*.

Como cada aplicação executa em seu próprio processo, com restrições de acesso a outros arquivos, elas não iniciam diretamente um componente de outra aplicação. Quem faz isso é o sistema. Portanto, se uma aplicação necessitar executar um componente, ela envia uma mensagem ao sistema, informando a necessidade de executá-lo (DEVELOPERS, 2015a). O sistema então o inicia. A figura 2 mostra como funciona este ciclo de inicialização de componentes.

Figura 2 – Esquema de inicialização de um componente a partir de um aplicativo.



Fonte: do autor, 2015

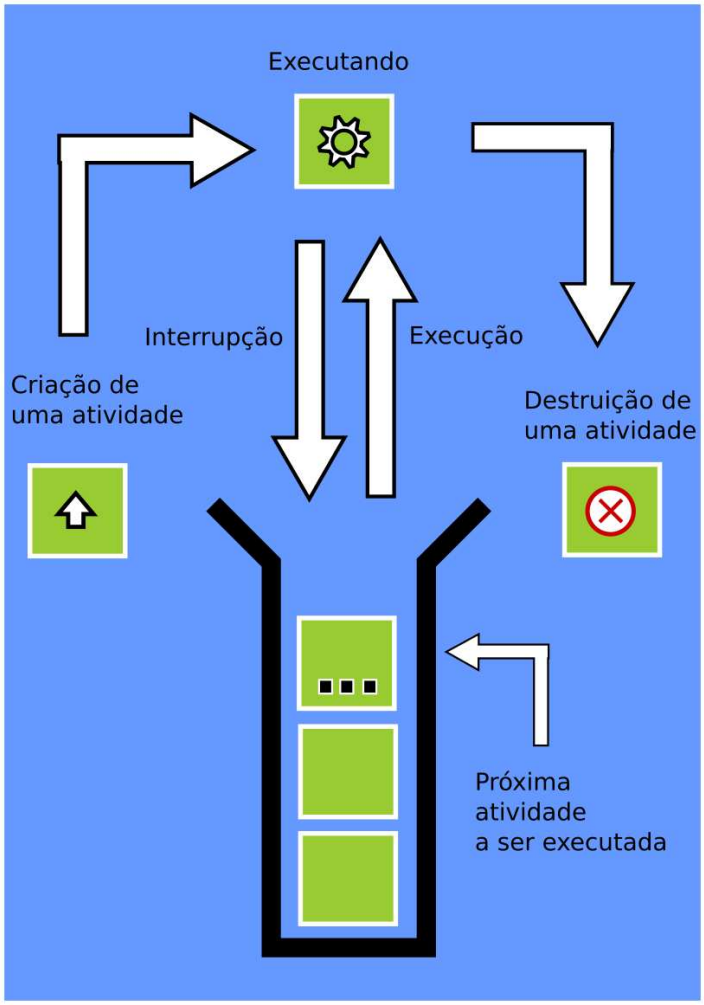
2.5.3 Ciclo de vida de uma atividade

A atividade é um componente da aplicação que permite a interação do usuário com a mesma. Discar um telefone, tirar uma foto, enviar um e-mail, cada uma dessas tarefas são realizadas através deste componente. Uma aplicação pode requisitar funcionalidades que a atividade atual não é capaz de realizar. Então, pode ser solicitada outra atividade responsável por tais tarefas. Quando uma atividade é interrompida para que outra possa executar, a última é adicionada à pilha de retorno (DEVELOPERS, 2015b).

Uma pilha é um conceito de estrutura de dados onde o primeiro elemento que entra é o último a sair. A execução de atividades funciona de forma semelhante. Cada atividade

interrompida é adicionada à pilha. Quando ela é fechada, atividade é retirada da pilha e a próxima será notificada, retomando sua execução. Vejamos a seguir como funciona o fluxo de execução de atividades na figura 3:

Figura 3 – Esquema de funcionamento da pilha de execução. Quando uma atividade é executada, a que estava anteriormente em primeiro plano é interrompida e alocada no topo da pilha de execução. Quando a atividade é destruída, a que está no topo da pilha é executada novamente.



Fonte: do autor, 2015

Há diversas formas de uma atividade ser notificada e então realizar uma ação por meio dessa notificação. Esse conjunto de formas são gerenciados pelo ciclo de vida da atividade, onde encontram-se estados que correspondem a uma chamada de retorno diferente: quando o sistema está criando uma atividade, interrompendo, retomando ou destruindo (DEVELOPERS, 2015b). Esse gerenciamento é bastante importante, pois certas tarefas só podem ser executadas mediante um estado da atividade. Por exemplo: certas aplicações, como o caso da aplicação que este trabalho apresenta, necessitam utilizar recursos do celular como o GPS. Segundo o próprio site de desenvolvedores do Android, este recurso consome bastante energia da bateria (DEVELOPERS,

2015c). Portanto é um recurso que deve ser manuseado somente quando necessário. Quando a atividade é interrompida, aproveitamos o estado de interrupção para desligar o serviço de localização.

Abaixo ilustramos como funciona o ciclo de vida de uma atividade. Podemos monitorar três *loops* aninhados de execução de uma atividade:

- **Tempo de vida de uma atividade:** Podemos monitorar todo o tempo de vida de uma atividade. Ele sempre inicia através do método *onCreate()* e é finalizada no método *onDestroy()*. Toda a inicialização de atributos deve ser executada no método *onCreate()*. Algumas atividades utilizam recursos do celular. Portanto, quando ela é destruída, devemos liberar estes recursos no método *onDestroy()*.

- **Tempo de vida visível de uma atividade:** Acontece entre a chamada do método *onStart()* e *onStop()*. Durante este tempo, a atividade estará disponível para o usuário interagir com a aplicação. Utilizamos o *onStop()*, por exemplo, para liberar recursos. A diferença para o *onDestroy()* é que as instâncias dos recursos liberados podem ser retomados através da reinicialização da atividade.

- **Tempo de vida em primeiro plano:** Acontece entre a execução do *onResume()* e *onPause()*. Durante esse tempo o usuário tem acesso a atividade que tem foco de interação (DEVELOPERS, 2015b). Uma atividade pode transitar entre o primeiro e o segundo plano de execução. Por exemplo: quando uma caixa de diálogo é exibida, a atividade chama o método *onPause()* e então vai para segundo plano. Segundo os desenvolvedores do Android, devemos atribuir execuções leves a esses métodos, evitando que o usuário tenha que esperar pela finalização dessas tarefas.

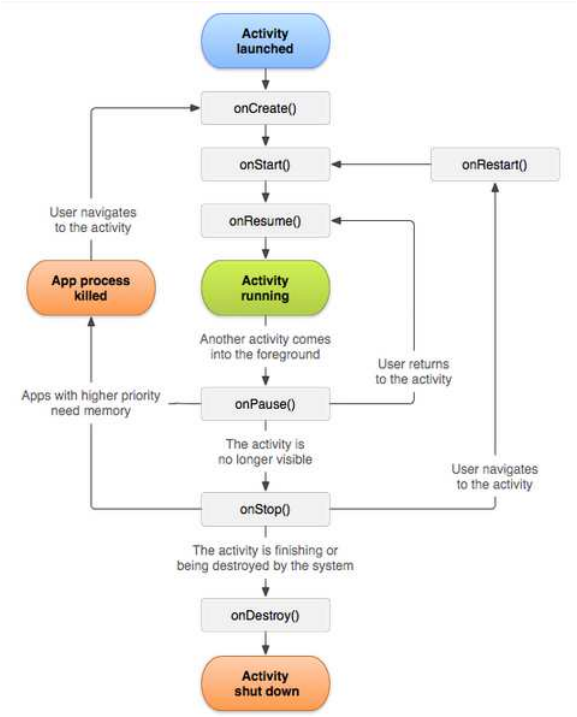
2.5.4 Google Play Services

O Google Play Services é um conjunto de ferramentas que possibilitam a integração de recursos e APIs que a Google oferece para os desenvolvedores. Com o Google Play Services, podemos ter acesso a APIs como Google Maps, Google Plus e muitas outras. Ele possui uma plataforma automática de atualizações através de APKs disponibilizados na Play Store (DEVELOPERS, 2015e).

A biblioteca contém interfaces para os serviços do Google e permite que os desenvolvedores obtenham permissões dos usuários para que eles acessem essas interfaces. A biblioteca que fica do lado do cliente pode receber atualizações e o desenvolvedor pode utilizar as novas ferramentas à medida que as funcionalidades são disponibilizadas. O desenvolvedor tem a opção também de não receber essas atualizações, caso sua aplicação não necessite (DEVELOPERS, 2015e).

A biblioteca contém as APIs e executa no aparelho como um serviço, em background. Podemos interagir com o ele por meio da biblioteca cliente. O Google Play Services é obtido

Figura 4 – Ciclo de vida de uma atividade no Android.



Fonte: (DEVELOPERS, 2015b).

através da Play Store, portanto não está acoplado a versões do sistema operacional desenvolvido por outras fabricantes. Em geral, aparelhos que rodam o Android 2.3 em diante podem instalar o aplicativo e receberem atualizações frequentemente. Versões mais antigas que o Android 2.3 não suportam o Google Play Services. A figura 5 demonstra o fluxo de atualizações (DEVELOPERS, 2015e).

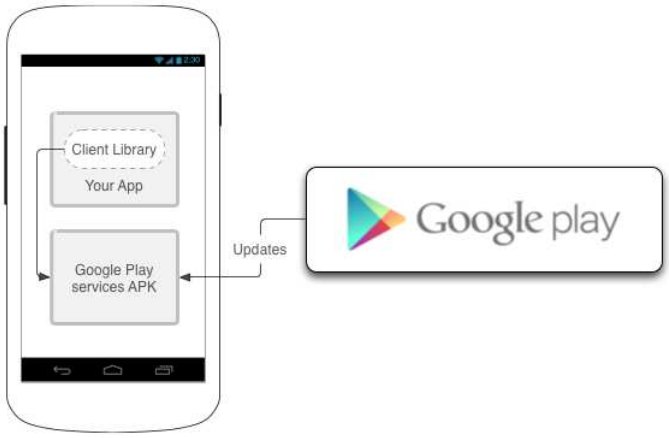
Com o Play Services, podemos utilizar diversas ferramentas e APIs da Google sem se preocupar com o suporte das versões. Assim, permite que os desenvolvedores foquem no que é mais importante: a experiência do usuário.

Para utilizar o Google Play Services, devemos baixar os pacotes necessários para o desenvolvimento. Utilizamos o SDK Manager para realizar esta tarefa. Logo após, devemos declarar em nosso arquivo de compilação que usaremos o Google Play Services em nosso projeto.

2.5.5 API de direção

Através da API Directions do Google, podemos solicitar rotas de um ponto A ao ponto B. Ela é extremamente útil para aplicações que indicam melhores caminhos ao usuário, ou caminhos que obedeçam certas restrições. Além disso, podemos solicitar rotas por meio de vários modos de locomoção: a pés, de bicicleta, de carro ou transporte público. As direções são especificadas por meio de origens, destinos e pontos entre essas coordenadas. Podemos especificar a origem e o destino por meio de texto, por exemplo, “São Luís, MA” ou “Bacabal,

Figura 5 – Fluxo de atualizações do Google Play Services.



Fonte: (DEVELOPERS, 2015e).

Figura 6 – Definindo o Google Play Services em nosso projeto.

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.0.0'
    compile 'com.google.android.gms:play-services:7.5.0'
    compile 'com.mcxiaoke.volley:library:1.0.+'
}
```

Fonte: do autor, 2015.

MA, Brazil", ou por meio de coordenadas latitude/longitude (DEVELOPERS, 2015f).

Entretanto, a API Directions foi desenvolvida para calcular rotas mediante informações fornecidas pelo usuário anteriormente. Ela não foi desenvolvida para calcular rotas em tempo real. Seu funcionamento é bem simples: construímos uma requisição via protocolo HTTP para os servidores do Google. Nesta requisição conterão a origem e o destino da rota. Enviamos a requisição e recebemos como resposta dados em formato *json* referentes à rota: as coordenadas que formam o caminho, os pontos conhecidos na rota, o tempo médio para chegar ao destino a pés, de bicicleta ou em um veículo automotor. Com esses dados, é possível desenhar a rota no mapa, por exemplo. Nosso trabalho busca mostrar que é possível também indicar a direção ao destino, utilizando o conjunto de pontos utilizados.

Para utilizar a API, é necessário possuir uma conta Google. Devemos cadastrá-la no console de desenvolvedores da Google. Em seguida, é necessário ativar a API. Então, o sistema enviará uma chave que identificará e permitirá que a aplicação acesse os servidores de direção.

2.5.6 API do Google Maps

Com a API do Google Maps, podemos adicionar mapas baseados nos dados dos servidores de mapas do Google. A API gerencia o acesso aos servidores, fornece manipuladores para o mapa da aplicação e provê a visualização do mesmo. Podemos adicionar marcadores, desenhar trajetos e mostrar diferentes formas de visualização ao usuário. Para utilizar o serviço, precisamos de uma chave de ativação, que funciona como um ID, que identifica a aplicação nos serviços da Google e permite o acesso a esses recursos (DEVELOPERS, 2015g).

3 O PROJETO DISPLAY LIGHT

O projeto Display Light foi concebido no início do ano de 2014. Procurando meios de desenvolver uma experiência de locomoção mais independente para usuários que possuem deficiência visual, idealizamos o que é hoje o aplicativo FollowVibe. Por meio deste projeto, percebemos o quão importante é o segmento de acessibilidade móvel.

O projeto Display Light propõem uma interface acessível ao portador de deficiência visual onde ele terá mais liberdade de locomoção em primeira instância. Queremos que o Display Light não seja somente isso. Desejamos que ele possa impulsionar e inspirar o desenvolvimento de outras tecnologias móveis direcionadas à acessibilidade.

O escopo desse trabalho visa demonstrar um dos ramos do projeto: o sistema FollowVibe. O objetivo é guiar deficientes visuais até um destino escolhido, utilizando um aplicativo para *smartphone* conectado a um Arduino que manipulará uma luva, fazendo-a vibrar conforme a direção que deve ser seguida. O aplicativo se conectará ao serviço de GPS para obter a localização atual do usuário, utilizará a API do Google Maps para mostrar visualmente o andamento da execução do sistema e utilizará a API Directions da Google para obter as rotas escolhidas pelo usuário.

3.1 Deficiência visual

A deficiência visual é uma situação irreversível de baixa resposta visual, que pode ser causada por fatores congênitos ou hereditários. Mesmo com tratamento clínico ou utilização de óculos para a correção, o indivíduo não retorna suas funções visuais completamente. A diminuição da resposta visual pode ser leve, moderada, severa, profunda e ausência total da visão, que constitui a cegueira (AÇÃO, 2014). Resposta visual profunda também é categorizada como visão subnormal ou baixa visão.

Segundo decreto nº 3.298/1999, artigo 4º Inciso 3, a cegueira é caracterizada por acuidade visual igual ou menor que 0,05 no melhor olho, com a melhor correção óptica; a baixa visão por acuidade visual entre 0,3 e 0,05 no melhor olho, com a melhor correção óptica. Em casos nos quais a somatória da medida do campo visual entre os dois olhos for igual ou menor que 60º e todos os outros casos anteriores dá-se o nome de deficiência visual (BRASIL, 1999).

Segundo a Organização Mundial da Saúde, existem quatro níveis de funções visuais:

1. Visão Normal;
2. Deficiência visual moderada;
3. Deficiência visual severa;
4. Cegueira.

A visão moderada junto com a visão severa constituem o grupo da baixa visão. Vejamos a seguir a tabela que contém os índices de grau de visão juntamente com a classificação de cada grau:

Tabela 1: Tabela de valores de classificação de função visual.

Categoria	**Menor que**	**Igual ou maior que**
Deficiência Visual mínima ou nenhuma deficiência visual		6/18, 3/10(0,3), 20/70
Deficiência visual moderada	6/18, 3/10(0,3), 20/70	6/60, 1/10(0,1), 20/200
Deficiência visual severa	6/60, 1/10(0,1), 20/200	3/60, 1/20(0,05), 20/400
Cegueira	3/60, 1/20(0,05), 20/400	1/60, 1/50(0,02), 5/300 (20/1200)

FONTE: (ORGANIZATION, 2014).

3.2 Modelo conceitual do aplicativo

A figura 7 demonstra o conceito de funcionamento do sistema FollowVibe. Foi definido inicialmente o desenvolvimento do aplicativo na plataforma Android. Dispositivos com este sistema operacional são mais acessíveis à população. A plataforma de desenvolvimento e programação é java, uma linguagem bastante robusta e também bastante difundida no meio acadêmico. Isto facilita que outros desenvolvedores possam dar continuidade ao projeto.

O aplicativo foi idealizado para funcionar na seguinte ordem de execução:

- O usuário inicia a aplicação em seu smartphone;
- A aplicação carrega os dados necessários para a correta execução, como os recursos de localização e os recursos do Google Maps;
- Uma vez que o sistema está pronto para uso, o aplicativo emitirá um som avisando que o usuário pode falar o destino desejado;
- O usuário informa o destino desejado ou um comando para o sistema;
- Se o usuário informa um comando, o sistema executa o comando;
- Se o usuário informa o destino, o sistema calculará a rota;
- Após a rota ser calculada, o usuário receberá atualizações de direção até alcançar o destino.

Na próxima seção demonstraremos como o aplicativo funciona, a nível de programação, para que as ações acima possam ser realizadas.

Figura 7 – Modelo conceitual do sistema.

Fonte: do autor, 2015

3.3 Diagrama de classes do aplicativo

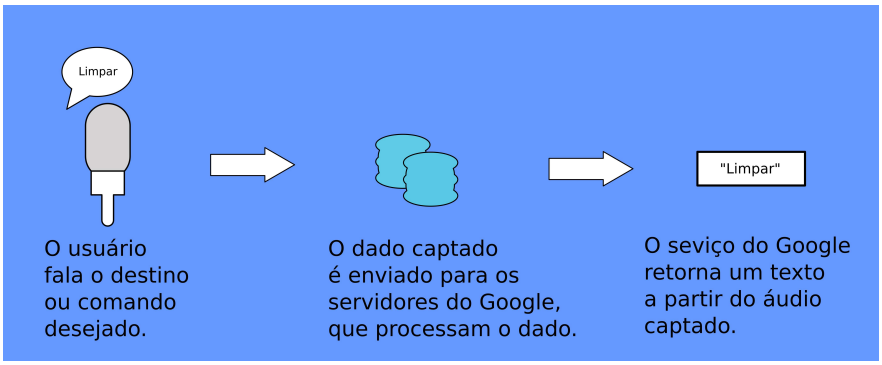
A classe principal do sistema é a MapActivity. Ela é responsável por inicializar todos os componentes do sistema, como o mapa e o serviço de GPS, e responsável por manipular os elementos gráficos apresentados para o usuário. Ao iniciar o aplicativo, o mapa é apresentado ao usuário e sua posição inicial é apresentada. O sistema apresenta duas opções de interação entre o usuário e o sistema: uma através da fala e outra através do mapa. As funcionalidades apresentadas ao usuário são: solicitar destino e limpar a rota. Ambas as funcionalidades são realizadas através de comando por voz. Vejamos cada uma das funcionalidades a seguir. Além delas, temos também a funcionalidade indicar a rota que ocorre internamente ao sistema e não necessita de uma interação direta do usuário com o mesmo. Todos os métodos e classes descritos aqui podem ser encontrados na seção apêndice desta pesquisa.

3.3.1 Solicitar o destino

A primeira ação que o usuário deve fazer ao iniciar o processo de chegar ao destino desejado é escolher o mesmo. Para isso, ao iniciar o sistema, o aplicativo fornecerá um recurso de entrada de voz para que ele fale o destino desejado. Esta ação é feita através do método da classe MapActivity getVoiceLocal. Este método basicamente inicia um objeto Intent para requisitar o reconhecedor de voz do sistema. O Intent é um objeto responsável por iniciar novas atividades. Através dele, passamos dados que queremos enviar para uma nova atividade e então iniciamos a mesma. A atividade recebe os dados e então entra em execução. Precisamos disso para solicitar o reconhecedor de voz nativo do Android. Iremos inicializá-lo, obter a entrada de voz, convertê-la para texto e enviar o resultado de volta para nossa aplicação.

O identificador de voz nativo captura a fala do usuário e envia esses dados para os servidores do Google, que processam a informação. Logo após, eles retornam um texto, resultado do processo, para o reconhecedor, que por sua vez retorna o texto para a aplicação. A seguir, na figura 8, mostraremos a sequência de passos para o reconhecimento de voz.

Figura 8 – Passos para o reconhecimento de voz no sistema Android.



Fonte: do autor, 2015

Uma vez que nossa aplicação obteve o texto, ela irá procurar em um banco de dados o destino previamente cadastrado. Uma vez que ele encontrou o destino, o sistema irá calcular a rota a partir do ponto atual do usuário. A seguir veremos em detalhes como funciona o processo de obtenção dessa rota.

3.3.2 Obter a rota e desenhar a rota

Após solicitar o local de destino, o sistema estará apto a obter a rota. Para isso, o usuário precisa estar conectado a internet. Ao obter a rota, o sistema realizará uma requisição ao serviço de Direção do Google, pedindo os dados da rota específica. Os serviço necessita de três parâmetros: o ponto de partida, que é a posição inicial do usuário, obtida internamente no sistema; o ponto de chegada, obtido através do destino cujo ponto está previamente cadastrado, e a chave de API. Com este três parâmetros, criamos uma *url* e realizamos uma requisição do tipo *get* ao serviço. O serviço retorna a requisição com um *json* contendo os dados referentes à rota. Dentre os parâmetros do *json*, temos o parâmetro *points*, composto por uma *string* codificada que representa todas as coordenadas da rota. Precisamos decodificar essa *string* para obter esses pontos. Esta tarefa está a cargo da classe ParseJson.

A classe ParseJson possui um método chamado parseGoogleMapsJson, que recebe o *json* de requisição e retorna os pontos da rota em uma lista de coordenadas. O funcionamento interno foi obtido da comunidade de desenvolvedores da plataforma *maps* (STACKOVERFLOW, 2013). Uma vez que o sistema possui esta lista de pontos, ele está apto a desenhar a rota no mapa. Essa tarefa é atribuída à classe Route. Ela é responsável por desenhar as linhas a partir de

pontos cedidos a ela. Portanto, devemos iterar sobre a lista de pontos e fornecer um a um para a classe Route. Ao final da iteração, teremos a rota desenhada no mapa.

3.3.3 Limpar a rota

Uma vez que o usuário chegou ao destino ou queira modificar a rota, ele deverá antes limpar o sistema de dados antigos para que os mesmos não atrapalhem na requisição de novas rotas. Para realizar este procedimento, basta que o usuário clique no mapa. Novamente, o reconhecedor de voz surgirá na tela. O usuário deve então falar "limpar" para o sistema. Pronto, o sistema se encarregará de deixá-lo pronto para a solicitação de novas rotas.

3.3.4 Indicar a rota

Após marcar o destino, obter a rota e desenhá-la na tela, o sistema automaticamente iniciar o processo de direcionamento. Para realizar esta tarefa, a classe MapActivity precisa implementar a interface LocationListener. Esta interface define métodos que serão utilizados pelo serviço de georreferenciamento utilizado no sistema, o GPS. O serviço precisa receber uma classe que implementa a interface, pois ele terá a certeza que os métodos que ele conhece serão implementados. Implementamos a LocationListener na classe MapActivity e passamos ela ao serviço de georreferenciamento. Ele irá chamar o método onLocationChanged, implementado na classe MapActivity, toda vez que a posição do usuário alterar, segundo algumas condições impostas ao serviço.

O GPS enviará atualizações a todo momento, se assim o programador desejar. No entanto, por recomendação da documentação para desenvolvedores Android, é interessante aplicar condições para a atualização de posicionamento. Cada atualização será processada pelo sistema e retornará uma resposta para o usuário. Este processamento, dependendo da aplicação, pode demandar um intervalo de tempo maior. Isso apresenta o risco de o usuário não estar mais no local em que a informação era necessária. Uma vez que essa informação é atrasada, poderá gerar dúvidas ou causar sérios problemas. Para contornar o problema de execuções dispendiosas, o serviço de georreferenciamento permite que o desenvolvedor coloque condições de atualização. A recomendação da documentação, para sistemas que trabalham com posicionamento de pessoas que andam a pés, é utilizar o tempo de atualização de 10 segundos, tempo de atualização rápida de 5 segundos e definir a precisão do GPS como alta. Isso permite que cada execução possa retornar o resultado para o usuário sem que uma outra atualização aconteça. Devemos colocar um tempo de atualização rápida de 5 segundos porque o tempo normal de atualização pode ser afetado por outros processos que estejam utilizando o serviço de georreferenciamento. Com esta definição teremos uma prioridade a mais para receber a atualização (DEVELOPERS, 2015d).

A partir de agora, iremos definir para o sistema o que ele deve fazer com a nova posição do usuário. Iremos decidir para onde o usuário deve seguir, a partir do ponto atual do mesmo, tendo como base o vetor de pontos que obtemos do serviço de Direção do Google. Essa responsabilidade foi atribuída à classe PositionUtil. A classe possui três métodos: getClo-

sestPointOfLatLngList, getDirectionRelationNoth, indicateDirection. Vejamos como cada um opera.

O método getClosestPointOfLatLngList recebe como parâmetro uma lista de pontos que correspondem à rota e um ponto que corresponde à posição atual do usuário. Ela irá retornar um ponto, contido na lista de pontos da rota, que corresponde ao ponto mais próximo da rota que o usuário está. Para realizar o processo de direcionamento, iremos considerar este ponto retornado.

Uma vez que o sistema possui a localização atual do usuário na rota, devemos indicar a direção do próximo ponto. Para isso, utilizaremos a função getDirectionRelationNorth. Ela recebe dois parâmetros: o ponto atual e o próximo ponto da rota. Internamente a função aplica cálculos de direção, e, baseado na direção norte, ou seja, uma vez que o usuário esteja voltado para o norte, o método indica para qual direção ele deve seguir. O valor de retorno é um inteiro, que representa a direção. Abaixo vejamos a tabela de correspondência dos valores inteiros e as respectivas direções:

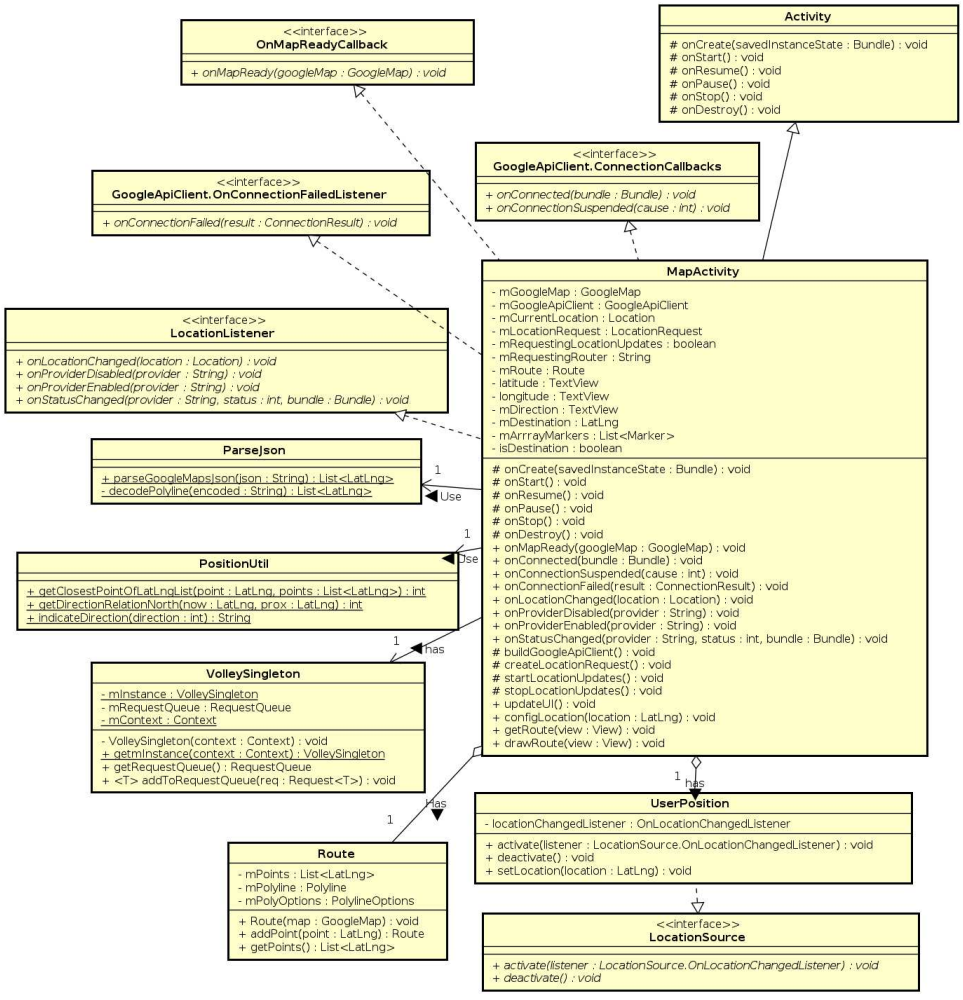
Tabela 2: Tabela de valores de possíveis direções.

Valor	**Direção**
0	Parado
1	Frente
2	Nordeste
3	Direita
4	Sudeste
5	Atrás
6	Sudoeste
7	Esquerda
8	Noroeste
9	Alerta

FONTE: do autor, 2015.

Para efeito de análise e testes do aplicativo, foi desenvolvido o método indicateDirection, que recebe como parâmetro um inteiro, preferencialmente o valor de saída do método getDirectionRelationNorth, e retorna um texto indicando a direção. O texto enviado respeita a regra de correspondência da tabela 2. O algoritmo executará até que o ponto final seja alcançado. Vejamos a seguir o diagrama de classes do sistema, contendo as classes principais e as classes facilitadoras do processo de direção.

Figura 9 – Diagrama de classes do sistema.



Fonte: do autor, 2015

4 TESTE E ANÁLISE DE RESULTADOS

A próxima seção descreverá o processo de testes do sistema FollowVibe. Serão descritos os dispositivos utilizados para realizar a execução do aplicativo e do software de direcionamento de vibrações. Serão descritos também o ambiente de testes e os resultados obtidos, bem como a análise destes resultados.

4.1 Ambiente de testes

Para realizar os testes, necessitamos operar o sistema em ambientes mapeados pelo Google Maps. As rotas são calculadas nestes ambientes mapeados. Realizamos testes na Universidade Estadual do Maranhão, campus Paulo VI. Mapeamos quatro pontos dentro da universidade para a realização dos testes, que foram o Centro de Ciências Agrárias, definido no sistema como “agronomia”, o Centro de Ciências Tecnológicas, definido no sistema como “engenharia” e a Uemanet, definido como “uemanet”.

Foi utilizado uma rede de dados móvel para realizar as requisições pela internet. O dispositivo utilizado foi um *smartphone* que possui o sistema operacional Android versão 4.4.2 KitKat, com memória RAM de 2 *gigabytes* e memória de armazenamento interno de 8 *gigabytes*. O *smartphone* possui o serviço de GPS e o serviço de reconhecimento de voz. Alguns recursos utilizados pelo sistema não funcionam muito bem em versões do Android que utilizam a API de desenvolvimento 17, ou seja, versões do android 4.2.2 para baixo não irão executar o software. As versões do Android compatíveis com o sistema são as versões KitKat(4.4) e superiores.

4.2 Avaliação e testes de resultados

Para realização dos testes, cadastramos no aplicativo três destinos diferentes localizados dentro do campus Paulo VI da Universidade Estadual do Maranhão. Os pontos cadastrados são: “agronomia”, que indica o Centro de Ciências Agrárias; “engenharia”, que indica o Centro de Ciências Tecnológicas e “uemanet”, que indica o centro de educação à distância da universidade. Isto significa que deve ser falado ao aplicativo uma das palavras previamente cadastradas.

Para realizar os testes navegamos pelas seguintes rotas: do Centro de Ciências Tecnológicas para o Centro de Ciências Agrárias; do Centro de Ciências Agrárias para o Centro de Ciências Tecnológicas e do Centro de Ciências Agrárias para o centro de educação à distância. A rota é indicada no aplicativo tendo como referência o norte geográfico. Em todos os testes um dos problemas que encontramos foi a dificuldade de conexão com a internet para obtenção da rota e processamento da fala de entrada do usuário.

Para a primeira rota obtemos os resultados da tabela 3. Observando os dados da tabela, para esta rota, obtemos 81,25% de acertos na indicação da rota.

Tabela 3: Tabela de valores de possíveis direções para a rota "engenharia - agronomia".

Valor desejado	**Valor real**
Direita	Direita
Direita	Direita
Direita	Direita
Direita	Direita
Direita	Sudoeste
Direita	Sudoeste
Sudoeste	Sudoeste
Sudoeste	Sudoeste
Sudoeste	Sudoeste
Direita	Sudoeste
Direita	Direita
Direita	Direita
Direita	Direita
Direita	Direita
Direita	Direita
Destino	Destino

FONTE: do autor, 2015.

Para a segunda rota, obtemos os resultados da tabela 4. Observando os dados da tabela, para esta rota, obtemos 84,25% de acertos na indicação da rota.

Tabela 4: Tabela de valores de possíveis direções para a rota "agronomia - engenharia".

Valor desejado	**Valor real**
Esquerda	Esquerda
Esquerda	Esquerda
Esquerda	Esquerda
Esquerda	Esquerda
Esquerda	Esquerda
Noroeste	Esquerda
Noroeste	Noroeste
Noroeste	Noroeste
Noroeste	Noroeste
Noroeste	Noroeste
Noroeste	Noroeste
Esquerda	Noroeste
Esquerda	Noroeste
Esquerda	Esquerda
Esquerda	Esquerda
Esquerda	Esquerda
Esquerda	Esquerda
Esquerda	Esquerda
Destino	Destino

FONTE: do autor, 2015

Para a terceira rota, obtemos os resultados da tabela 5. Observando os resultados, para esta rota, obtemos 78,12% de acertos de indicações.

Tabela 5: Tabela de valores de possíveis direções para a rota "agronomia - uemanet".

Valor desejado	**Valor real**
Sudeste	Sudeste
Sudeste	Sudeste
Sudeste	Sudeste
Sudeste	Sudeste
Sudeste	Sudeste
Sudeste	Sudeste
Sudeste	Sudeste
Sudeste	Sudeste
Sudeste	Sudeste
Sudeste	Sudeste
Leste	Sudeste
Leste	Sudeste
Leste	Nordeste
Leste	Nordeste
Nordeste	Nordeste
Nordeste	Nordeste
Nordeste	Nordeste
Nordeste	Nordeste
Nordeste	Nordeste
Nordeste	Nordeste
Nordeste	Nordeste
Nordeste	Nordeste
Nordeste	Nordeste
Nordeste	Nordeste
Nordeste	Nordeste
Nordeste	Nordeste
Nordeste	Nordeste
Nordeste	Nordeste
Nordeste	Nordeste
Nordeste	Destino
Nordeste	Destino
Nordeste	Destino
Destino	Destino

FONTE: do autor, 2015

Analisando os resultados, obtivemos um resultado satisfatório. A aplicação não retirou o usuário da rota principal. Apesar de eventuais contradições entre o desejado e o valor real, o usuário não sofreu nenhuma alteração brusca de direção. A interface está respondendo bem aos comandos. O usuário praticamente não precisou interagir com a mesma, bastando apenas fornecer o local desejado.

Entretanto, ocorreu algumas falhas na indicação do caminho certo. Alguns fatores contribuíram para que isso ocorresse:

- Utilização de uma abordagem tradicional. Foi desenvolvido um algoritmo que podemos categorizar como tradicional, ou seja, não utiliza nenhuma técnica amparada pelo conceito de inteligência artificial;

- Limitação do GPS. Por causa da precisão do GPS, a chegada ao destino mostrava-se precocemente determinada. Percebemos que o GPS apresenta um erro de precisão de cinco metros, o que acusava rapidamente a chegada ao destino.

5 CONCLUSÃO

Esta pesquisa apresentou um modelo conceitual de um sistema que tem como objetivo auxiliar deficientes visuais a se locomoverem de um ponto "A"para o ponto "B". Para alcançarmos este objetivo, desenvolvemos um aplicativo que define a rota e indica a direção para o destino desejado, utilizando uma luva tátil com micromotores de vibração que indica o caminho por meio da vibração desses motores.

Foram explicados neste trabalho conceitos sobre orientação a objetos, paradigma utilizado para a implementação do aplicativo. Foram abordados conceitos e fundamentos em programação para dispositivos Android. Vimos os componentes de uma aplicação e o ciclo de vida de uma atividade. Além disso, analisamos as bibliotecas necessárias para o desenvolvimento do aplicativo FollowVibe: a API de direção e a API do Google Maps. Espera-se que o produto desenvolvido seja utilizado como base para estudos na área de tecnologias para acessibilidade e também como base para estudos em tecnologias móveis.

Foi descrito nesta pesquisa o sistema FollowVibe. Ele aborda o aplicativo e a luva como interface tátil de direcionamento. O sistema indica a direção que o deficiente visual deve seguir. Através do identificador de voz, o usuário solicita o destino desejado ou um comando específico. Uma vez que a rota até o destino é calculada, o sistema inicia o processo de direcionamento.

Foram realizados testes com o sistema na Universidade Estadual do Maranhão. Foram mapeados três pontos do campus Paulo VI, a saber: o Centro de Ciências Tecnológicas, o Centro de Ciências Agrárias e a Uemanet. Elaboramos uma tabela de valores desejados e comparamos com os valores reais do aplicativo para análise. Conforme observado, obtemos resultados satisfatórios. Além disso, a partir dos testes desenvolvemos melhorias no processo de direcionamento, tornando menos confuso para o usuário.

5.1 Trabalhos Futuros

O trabalho seguirá em desenvolvimento. Existem diversas possibilidades de melhorias para a robustez do sistema. Para este primeiro momento, elencamos as seguintes propostas, a saber:

- Implementar uma lógica *fuzzy* para determinar a precisão da direção, tendo em vista que a solução atual para o cálculo do direcionamento utiliza uma abordagem tradicional;
- Analisar a estudar os sensores disponibilizados pela API do Android, como o giroscópio, o acelerômetro ou vetor de rotação, para analisar o posicionamento do usuário e a partir dela indicar a direção, sem necessariamente tomar como referencial o norte geográfico.

Referências

AÇÃO, D. em. *Deficiência visual*. 2014. Disponível em: <http://www.deficientesemacao.com/deficiencia-visual>. Acesso em: 18.11.2015.

BEZERRA, E. *Princípios de análise e projeto de sistemas com UML*. [S.l.]: Elsevier, 2007.

BRASIL. *Decreto nº 3.298 de 20 de dezembro de 1999*. 1999. Disponível em: <http://www.planalto.gov.br/ccivil_03/decreto/d3298.htm>. Acesso em: 18.11.2015.

BUENO, A. C. *Bengala Eletrônica para Deficientes Visuais*. 2010. Acesso em: 20.12.2015.

CARVALHO, W. B. de. *Conceitos - Encapsulamento: programação orientada a objetos*. 2014. Disponível em: <http://www.devmedia.com.br/conceitos-encapsulamento-programacao-orientada-a-objetos/18702>. Acesso em: 26.11.2015.

CARVALHO, W. B. de. *Conceitos e exemplos - Herança: programação orientada a objetos - Parte 1*. 2014. Disponível em: <http://www.devmedia.com.br/conceitos-e-exemplos-heranca-programacao-orientada-a-objetos-parte-1/18579>. Acesso em: 26.11.2015.

DEITEL, P.; DEITEL, H. *Java: como programar*. [S.l.]: Pearson, 2010.

DEVELOPERS, A. *Application Fundamentals*. 2015. Disponível em: <http://developer.android.com/guide/components/fundamentals.html>. Acesso em: 25.02.2015.

DEVELOPERS, A. *Atividades*. 2015. Disponível em: <http://developer.android.com/intl/pt-br/guide/components/activities.html>. Acesso em: 11.11.2015.

DEVELOPERS, A. *Location Strategies*. 2015. Disponível em: <http://developer.android.com/intl/pt-br/guide/topics/location/strategies.html>. Acesso em: 11.11.2015.

DEVELOPERS, A. *Receiving Location Updates*. 2015. Disponível em: <http://developer.android.com/intl/pt-br/training/location/receive-location-updates.html>. Acesso em: 04.12.2015.

DEVELOPERS, G. *Google APIs for Android.* 2015. Disponível em: <https://developers.google.com/android/guides/overview>. Acesso em: 14.09.2015.

DEVELOPERS, G. *The Google Maps Directions API.* 2015. Disponível em: <https://developers.google.com/maps/documentation/directions/intro?hl=pt-br>. Acesso em: 02.10.2015.

DEVELOPERS, G. *Introduction to the Google Maps Android API.* 2015. Disponível em: <https://developers.google.com/maps/documentation/android-api/intro>. Acesso em: 16.09.2015.

FAPESP. *Aplicativo que auxilia locomoção de deficientes visuais é lançado na UNESC*. 2015. Disponível em: <http://www.fapesc.sc.gov.br/aplicativo-que-auxilia-locomocao-de-deficientes-visuais-e-lancado-na-unesc/>. Acesso em: 20.12.2015.

FILHO, J. de S. R. et al. *ARGOS - Auxílio à locomoção de deficientes visuais a partir de pulseira microcontrolada*. 2011. Acesso em: 20.12.2015.

GIL, M. *Cadernos da TV Escola: Deficiência Visual*. [S.l.]: Ministério da Educação, 2000.

HERSH, M. A.; JHONSON, M. A. *Assistive Technology for Visually Impaired and Blind People*. [S.l.]: Springer, 2008. Acesso em: 22.11.2015.

ORGANIZATION, W. H. *Visual impairment and blindness.* 2014. Disponível em: <http://www.who.int/mediacentre/factsheets/fs282/en/>. Acesso em: 18.11.2015.

PAULO, U. de S. *Projeto de auxílio à pessoas com deficiência visual vence concurso INTEL de tecnologia*. 2014. Disponível em: <http://www.saocarlos.usp.br/index.php?option=com_content&task=view&id=18671&Itemid=121>. Acesso em: 06.10.2015.

Sá, E. D. de. *Acessibilidade: as pessoas cegas no itinerário da cidadania. In: Inclusão: Revista da Educação Especial.* [S.l.]: MEC, v. 1, n. 1, p. 13-18, 2005.

STACKOVERFLOW. *Decoding polyline with new Google Maps API.* 2013. Disponível em: <http://stackoverflow.com/questions/15924834/decoding-polyline-with-new-google-maps-api>. Acesso em: 04.03.2015.

TELFORD, C. W.; SAWREY, J. M. *O Indivíduo Excepcional*. [S.l.]: Zahar Editores, 1972.

VIAVOZ. *ViaVoz*. 2015. Disponível em: <http://www.guiaviavoz.com.br>. Acesso em: 20.12.2015.

WEBGOAL. *Origem da orientação a objetos*. 2009. Disponível em: <http://www.webgoal.com.br/origem-da-orientacao-a-objetos>. Acesso em: 26.10.2015.

WIKIPÉDIA. *Android*. 2015. Disponível em: <https://pt.wikipedia.org/wiki/Android>. Acesso em: 12.03.2015.

WIKIPÉDIA. *Java (linguagem de programação)*. 2015. Disponível em: <https://pt.wikipedia.org/wiki/Java_(linguagem_de_programaç~ao)>. Acesso em: 13.02.2015.

WIKIPÉDIA. *Polimorfismo*. 2015. Disponível em: <https://pt.wikipedia.org/wiki/Polimorfismo>. Acesso em: 10.10.2015.

Apêndices

APÊNDICE A – Código-fonte do aplicativo FollowVibe

```
  # -*- coding: utf-8 -*-
/**
 * Classe MapActivity
 *
 */
package com.example.antonio.darkdisplayv2;

import android.app.FragmentTransaction;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.location.Location;
import android.os.Bundle;
import android.speech.RecognizerIntent;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;
import android.widget.Toast;

import com.android.volley.Request;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.location.LocationListener;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.maps.CameraUpdate;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.GoogleMapOptions;
import com.google.android.gms.maps.MapFragment;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.model.CameraPosition;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;

import org.json.JSONException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

// Codigo de conexao e comunicacao com arduino desenvolvido por Eric do Androider.
// Site: http://android-er.blogspot.com.br/2014/
```

```
// 09/send-data-from-android-to-arduino-uno.html
//

public class MapActivity extends AppCompatActivity implements
        OnMapReadyCallback,
        LocationListener,
        GoogleApiClient.ConnectionCallbacks,
        GoogleApiClient.OnConnectionFailedListener {

    public static final int RESULT_SPEECH = 1;
    private GoogleMap       mGoogleMap;
    private GoogleApiClient mGoogleApiClient;
    private Location        mCurrentLocation;
    private LocationRequest mLocationRequest;
    private boolean         mRequestingLocationUpdates;
    private String          mRequestingRouter;
    private Route           mRoute;
    private LatLng          mDestination;
    private List<Marker>    mArrayMarkers;
    private boolean         isDestination;
    private Map<String, LatLng> pontosDestino;

    private TextView latitude;
    private TextView longitude;
    private TextView mDirection;
    private TextView voiceText;
    private int      previousDirection;

    /*Variaveis para a comunicacao com o arduino*/

    private static final int targetVendorID = 2341;
    private static final int targetProductID = 43;

    UsbDevice deviceFound          = null;
    UsbInterface usbInterfaceFound = null;
    UsbEndpoint endpointIn         = null;
    UsbEndpoint endpointOut        = null;

    private static final String
     ACTION_USB_PERMISSION = "com.android.example.USB_PERMISSION";

    PendingIntent mPermissionIntent;
    UsbInterface usbInterface;
    UsbDeviceConnection usbDeviceConnection;

    private final BroadcastReceiver mUsbReceiver =
     new BroadcastReceiver() {

        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            if (ACTION_USB_PERMISSION.equals(action)) {

                Toast.makeText(MapActivity.this, "ACTION_USB_PERMISSION",
                        Toast.LENGTH_LONG).show();

                synchronized (this) {
```

```
                UsbDevice device = (UsbDevice) intent
                        .getParcelableExtra(UsbManager.EXTRA_DEVICE);

                if (intent.getBooleanExtra(
                        UsbManager.EXTRA_PERMISSION_GRANTED, false)) {
                    if (device != null) {
                        connectUsb();
                    }
                } else {
                    Toast.makeText(MapActivity.this,
                            "permission denied for device " + device,
                            Toast.LENGTH_LONG).show();
                }
            }
        }
    }
};

private final BroadcastReceiver mUsbDeviceReceiver = new BroadcastReceiver() {

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (UsbManager.ACTION_USB_DEVICE_ATTACHED.equals(action)) {

            deviceFound = (UsbDevice) intent
                    .getParcelableExtra(UsbManager.EXTRA_DEVICE);
            Toast.makeText(
                    MapActivity.this,
                    "ACTION_USB_DEVICE_ATTACHED: \n"
                            + deviceFound.toString(), Toast.LENGTH_LONG)
                    .show();

            connectUsb();

        } else if (UsbManager.ACTION_USB_DEVICE_DETACHED.equals(action)) {

            UsbDevice device = (UsbDevice) intent
                    .getParcelableExtra(UsbManager.EXTRA_DEVICE);

            Toast.makeText(MapActivity.this,
                    "ACTION_USB_DEVICE_DETACHED: \n" + device.toString(),
                    Toast.LENGTH_LONG).show();

            if (device != null) {
                if (device == deviceFound) {
                    releaseUsb();
                }else{
                    Toast.makeText(MapActivity.this,
                            "device == deviceFound, no call releaseUsb()\n" +
                                    device.toString() + "\n" +
                                    deviceFound.toString(),
                            Toast.LENGTH_LONG).show();
                }
            }else{
                Toast.makeText(MapActivity.this,
                        "device == null, no call releaseUsb()",
```

```
                        Toast.LENGTH_LONG).show();
                }
            }
        }

};


@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_map);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    voiceText = (TextView) findViewById(R.id.textVoice);

    pontosDestino = new HashMap<String, LatLng>();
    previousDirection = 99;

    pontosDestino.put("agronomia", new LatLng(-2.583090, -44.208368));
    pontosDestino.put("engenharia", new LatLng(-2.582919, -44.209320));
    pontosDestino.put("uemanet", new LatLng(-2.580460, -44.206854));
    pontosDestino.put("letras", new LatLng(-2.575889, -44.209983));

    // Configuracoes iniciais do Google Maps
    //
    GoogleMapOptions options = new GoogleMapOptions();
    options.mapType(GoogleMap.MAP_TYPE_HYBRID);

    // Colocando o fragment que contera o mapa na atividade
    //
    MapFragment mMapFragment = MapFragment.newInstance(options);

    // Setando a interface que contera o metodo que sera
    // chamado quando o mapa estiver pronto
    //
    mMapFragment.getMapAsync(this);

    FragmentTransaction fragmentTransaction =
            getFragmentManager().beginTransaction();
    fragmentTransaction.add(R.id.mapContainer, mMapFragment);
    fragmentTransaction.commit();

    // Criando uma instancia de GoogleApiClient
    //
    this.buildGoogleApiClient();

    // Criando um objeto de configuracoes de atualizacao de rotas
    //
    this.createLocationRequest();

    // Obtendo os textViews de latitude e longitude
    //
    latitude      = (TextView) findViewById(R.id.textCurrentLatitude);
    longitude     = (TextView) findViewById(R.id.textCurrentLongitude);
    mDirection    = (TextView) findViewById(R.id.textDirection);
```

```
        mArrayMarkers = new ArrayList<Marker>();
        isDestination = false;

        // Configuracao da conexao com o arduino
        //
        // register the broadcast receiver
        mPermissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(
                ACTION_USB_PERMISSION), 0);
        IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
        registerReceiver(mUsbReceiver, filter);
        registerReceiver(mUsbDeviceReceiver, new IntentFilter(
                UsbManager.ACTION_USB_DEVICE_ATTACHED));
        registerReceiver(mUsbDeviceReceiver, new IntentFilter(
                UsbManager.ACTION_USB_DEVICE_DETACHED));

        // Conexao com o arduino;
        //
        connectUsb();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the
        // action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_map, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    @Override
    protected void onStart() {
        super.onStart();
        mGoogleApiClient.connect();
    }

    @Override
    protected void onResume() {
        super.onResume();
    }

    @Override
    protected void onPause() {
        super.onPause();
```

```
}

@Override
protected void onStop() {

    super.onStop();

    this.stopLocationUpdates();
    mGoogleApiClient.disconnect();

    mRequestingLocationUpdates = false;
}

@Override
protected void onDestroy() {
    releaseUsb();
    unregisterReceiver(mUsbReceiver);
    unregisterReceiver(mUsbDeviceReceiver);
    super.onDestroy();
}

@Override
public void onMapReady(GoogleMap googleMap) {
    mGoogleMap = googleMap;
    mRoute     = new Route(googleMap);

    this.configClickAction();
}

@Override
public void onConnected(Bundle bundle) {
    mCurrentLocation = LocationServices
     .FusedLocationApi
     .getLastLocation(mGoogleApiClient);

    latitude.setText("Lat: "+mCurrentLocation.getLatitude());
    longitude.setText("Lng: "+mCurrentLocation.getLongitude());

    this.startLocationUpdates();
    mRequestingLocationUpdates = true;

    configLocation(new LatLng(mCurrentLocation.getLatitude(),
     mCurrentLocation.getLongitude()));

    getVoiceLocal();
}

@Override
public void onConnectionSuspended(int i) {
    // Do nothing
}

@Override
public void onLocationChanged(Location location) {
    mCurrentLocation = location;
    updateUI();
```

```
    if(!isDestination) {
        if(mRoute.getPoints().size() != 0) {
            int index = PositionUtil.getClosestPointOfLatLngList(
                    new LatLng(location.getLatitude(), location.getLongitude()),
                    mRoute.getPoints());

            Log.i("DATA","Index: "+
             index+
             "- Size: "+mRoute.getPoints().size()+
             " Points:
            "+mRoute.getPoints().toString());

            if(index == mRoute.getPoints().size() - 1) {
                isDestination = true;
                mDirection.setText("Chegou ao destino!");
            } else {
                int direction = PositionUtil.getDirectionRelationNorth(
                        mRoute.getPoints().get(index),
                        mRoute.getPoints().get(index + 1));

                // Enviando os dados para a luva
                //
                direction = keepDirection(direction);
                sendDirectionToGlove(direction);

                String way = PositionUtil.indicateDirection(direction);
                mDirection.setText(way);
            }
        } else {
            mDirection.setText("Aguardando...");
        }

    } else {
        mDirection.setText("Chegou ao destino!");
    }

    configLocation(new LatLng(location.getLatitude(),location.getLongitude()));
}

@Override
public void onConnectionFailed(ConnectionResult connectionResult) {

}

protected synchronized void buildGoogleApiClient() {
    mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .addApi(LocationServices.API)
            .build();
}

protected void createLocationRequest() {
    mLocationRequest = new LocationRequest();
    mLocationRequest.setInterval(10000);
    mLocationRequest.setFastestInterval(5000);
    mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
```

```
}

protected void startLocationUpdates() {
    LocationServices.FusedLocationApi.requestLocationUpdates(
            mGoogleApiClient, mLocationRequest, this);
}

protected void stopLocationUpdates() {
    LocationServices
    .FusedLocationApi
    .removeLocationUpdates(mGoogleApiClient, this);
}

public void updateUI() {
    latitude.setText("Lat: " + mCurrentLocation.getLatitude());
    longitude.setText("Lng: " + mCurrentLocation.getLongitude());
}

public void configLocation(LatLng location) {
    CameraPosition camera = new CameraPosition.Builder()
            .target(location)
            .zoom(17)
            .build();
    CameraUpdate cameraUpdate = CameraUpdateFactory.newCameraPosition(camera);

    this.mGoogleMap.setMyLocationEnabled(true);
    this.mGoogleMap.animateCamera(cameraUpdate);

    UserPosition userPosition = new UserPosition();
    mGoogleMap.setLocationSource(userPosition);
    userPosition.setLocation(location);
}

public void getRoute() {

    if(mDestination != null) {
        String url = "https://maps.googleapis.com/maps/api/directions/json?" +
                "origin=" +mCurrentLocation.getLatitude()+","
                +mCurrentLocation.getLongitude()+
                "&destination="+mDestination.latitude+","
                +mDestination.longitude+"&sensor=false&key=" +
                "AIzaSyDeOrRD5AtzVZAtonCcC1aPFIbfZzkVEHY&mode=walking";

        // Montando a requisicao, que recebera uma resposta do tipo String
        //
        StringRequest request = new StringRequest(Request.Method.GET, url,
                new Response.Listener<String>() {
                    @Override
                    public void onResponse(String responseValue) {

                        mRequestingRouter = responseValue;
                        drawRoute();
                        Toast.makeText(MapActivity.this,
                         "Rota obtida com sucesso.", Toast.LENGTH_LONG).show();
                    }
                }, new Response.ErrorListener() {
            @Override
```

```
            public void onErrorResponse(VolleyError error) {

                // Mostrando no console de erros o que aconteceu
                //
                Log.i("ERROR_REQUEST", error.toString());

                // Mostrando ao usuario que nao
                // foi possivel realizar a requisicao
                //
                Toast.makeText(MapActivity.this,
                "Nao foi possivel realizar a requisicao da rota.",
                Toast.LENGTH_LONG).show();
            }
        });

        VolleySingleton.getmInstance(this).addToRequestQueue(request);
    } else {
        Toast.makeText(this,
        "Nao e possivel obter a rota: Indique a origem e o destino primeiro.",
        Toast.LENGTH_LONG).show();
    }
}

public void drawRoute() {
    // Verificando se a resposta foi atribuida
    //
    if (mRequestingRouter == null) {
        Toast.makeText(this,
        "Rota nao foi obtida, obtenha primeiramente a rota",
        Toast.LENGTH_LONG).show();
        return;
    }

    // Realizando a tentativa de transformar a string em um objeto json
    //
    try {
        List<LatLng> list = ParseJson.parseGoogleMapsJson(mRequestingRouter);

        for (LatLng point : list) {

            // Passando os pontos para o objeto de desenho
            //
            mRoute.addPoint(point);
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
}

public void configClickAction() {
    mGoogleMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
        @Override
        public void onMapClick(LatLng latLng) {
            getVoiceLocal();
        }
    });
}
```

```
public void resetMap() {
    mDestination      = null;
    mRequestingRouter = null;
    isDestination     = false;
    mRoute.clearRoute();
    for(Marker mMarker : mArrayMarkers) {
        mMarker.remove();
    }
}

public void getVoiceLocal() {
    Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, "pt-BR");

    try {
        startActivityForResult(intent, RESULT_SPEECH);
        voiceText.setText("");
    } catch (ActivityNotFoundException e) {
        Toast.makeText(getApplicationContext(),
                "Opps! Your device doesn't support Speech to Text",
                Toast.LENGTH_SHORT).show();
    }
}

public void getVoiceLocal(View view) {
    getVoiceLocal();
}

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    switch(requestCode) {
        case RESULT_SPEECH: {
            if(resultCode == RESULT_OK && data != null) {
                ArrayList<String> text = data.getStringArrayListExtra(
                RecognizerIntent.EXTRA_RESULTS);
                String destino = text.get(0);
                voiceText.setText(destino);

                doMappedAction(destino);
            }
        }
        break;
    }
}

public void doMappedAction(String action) {

    String lowerCase = action.toLowerCase();
    if(lowerCase.equals("limpar")) {
        resetMap();
        return;
    }

    for(String key : pontosDestino.keySet()) {
```

```
            if(key.equals(lowerCase)) {
                mDestination = pontosDestino.get(lowerCase);
                getRoute();
            }
        }
    }

    private void connectUsb() {

        Toast.makeText(MapActivity.this, "connectUsb()", Toast.LENGTH_LONG)
                .show();

        searchEndPoint();

        if (usbInterfaceFound != null) {
            setupUsbComm();
        }

    }

    private void releaseUsb() {

        Toast.makeText(MapActivity.this, "releaseUsb()", Toast.LENGTH_LONG)
                .show();

        if (usbDeviceConnection != null) {
            if (usbInterface != null) {
                usbDeviceConnection.releaseInterface(usbInterface);
                usbInterface = null;
            }
            usbDeviceConnection.close();
            usbDeviceConnection = null;
        }

        deviceFound = null;
        usbInterfaceFound = null;
        endpointIn = null;
        endpointOut = null;
    }

    private void searchEndPoint() {

        usbInterfaceFound = null;
        endpointOut = null;
        endpointIn = null;

        // Search device for targetVendorID and targetProductID
        if (deviceFound == null) {
            UsbManager manager = (UsbManager) getSystemService(Context.USB_SERVICE);
            HashMap<String, UsbDevice> deviceList = manager.getDeviceList();
            Iterator<UsbDevice> deviceIterator = deviceList.values().iterator();

            while (deviceIterator.hasNext()) {
                UsbDevice device = deviceIterator.next();

                if (device.getVendorId() == targetVendorID) {
                    if (device.getProductId() == targetProductID) {
```

```
                    deviceFound = device;
                }
            }
        }
    }

    if (deviceFound == null) {
        Toast.makeText(MapActivity.this, "device not found",
                Toast.LENGTH_LONG).show();
    } else {
        String s = deviceFound.toString() + "\n" + "DeviceID: "
                + deviceFound.getDeviceId() + "\n" + "DeviceName: "
                + deviceFound.getDeviceName() + "\n" + "DeviceClass: "
                + deviceFound.getDeviceClass() + "\n" + "DeviceSubClass: "
                + deviceFound.getDeviceSubclass() + "\n" + "VendorID: "
                + deviceFound.getVendorId() + "\n" + "ProductID: "
                + deviceFound.getProductId() + "\n" + "InterfaceCount: "
                + deviceFound.getInterfaceCount();

        // Search for UsbInterface with Endpoint of USB_ENDPOINT_XFER_BULK,
        // and direction USB_DIR_OUT and USB_DIR_IN

        for (int i = 0; i < deviceFound.getInterfaceCount(); i++) {
            UsbInterface usbif = deviceFound.getInterface(i);

            UsbEndpoint tOut = null;
            UsbEndpoint tIn = null;

            int tEndpointCnt = usbif.getEndpointCount();
            if (tEndpointCnt >= 2) {
                for (int j = 0; j < tEndpointCnt; j++) {
                    if (usbif.getEndpoint(j).getType() ==
                    UsbConstants.USB_ENDPOINT_XFER_BULK) {
                        if (usbif.getEndpoint(j).getDirection() ==
                        UsbConstants.USB_DIR_OUT) {
                            tOut = usbif.getEndpoint(j);
                        } else if (usbif.getEndpoint(j).getDirection() ==
                        UsbConstants.USB_DIR_IN) {
                            tIn = usbif.getEndpoint(j);
                        }
                    }
                }

                if (tOut != null && tIn != null) {
                    // This interface have both USB_DIR_OUT
                    // and USB_DIR_IN of USB_ENDPOINT_XFER_BULK
                    usbInterfaceFound = usbif;
                    endpointOut = tOut;
                    endpointIn = tIn;
                }
            }
        }
    }
}

private boolean setupUsbComm() {
```

```
    // for more info, search SET_LINE_CODING and
    // SET_CONTROL_LINE_STATE in the document:
    // "Universal Serial Bus Class Definitions for Communication Devices"
    // at http://adf.ly/dppFt
    final int RQSID_SET_LINE_CODING = 0x20;
    final int RQSID_SET_CONTROL_LINE_STATE = 0x22;

    boolean success = false;

    UsbManager manager = (UsbManager) getSystemService(Context.USB_SERVICE);
    Boolean permitToRead = manager.hasPermission(deviceFound);

    if (permitToRead) {
        usbDeviceConnection = manager.openDevice(deviceFound);
        if (usbDeviceConnection != null) {
            usbDeviceConnection.claimInterface(usbInterfaceFound, true);

            int usbResult;
            usbResult = usbDeviceConnection.controlTransfer(0x21,
                    RQSID_SET_CONTROL_LINE_STATE, // SET_CONTROL_LINE_STATE
                    0, // value
                    0, // index
                    null, // buffer
                    0, // length
                    0); // timeout

            Toast.makeText(
                    MapActivity.this,
                    "controlTransfer(SET_CONTROL_LINE_STATE): " + usbResult,
                    Toast.LENGTH_LONG).show();

            // baud rate = 9600
            // 8 data bit
            // 1 stop bit
            byte[] encodingSetting = new byte[] { (byte) 0x80, 0x25, 0x00,
                    0x00, 0x00, 0x00, 0x08 };
            usbResult = usbDeviceConnection.controlTransfer(0x21, // requestType
                    RQSID_SET_LINE_CODING, // SET_LINE_CODING
                    0, // value
                    0, // index
                    encodingSetting, // buffer
                    7, // length
                    0); // timeout
            Toast.makeText(MapActivity.this,
                    "controlTransfer(RQSID_SET_LINE_CODING): " + usbResult,
                    Toast.LENGTH_LONG).show();
        }

    } else {
        manager.requestPermission(deviceFound, mPermissionIntent);
        Toast.makeText(MapActivity.this, "Permission: " + permitToRead,
                Toast.LENGTH_LONG).show();
    }

    return success;
}
```

```
    private void sendDirectionToGlove(int direction) {
        if(deviceFound != null){

            int message = direction;
            byte[] bytesOut = ByteBuffer.allocate(4).putInt(message).array();
            int usbResult = usbDeviceConnection.bulkTransfer(
                    endpointOut, bytesOut, bytesOut.length, 0);

        }else{
            Toast.makeText(MapActivity.this,
                    "deviceFound == null",
                    Toast.LENGTH_LONG).show();
        }
    }

    private int keepDirection(int direction) {
     if(previousDirection == direction) {
     return 1;
     } else {
     previousDirection = direction;
     return direction;
     }
    }
}



/**
 * Classe ParseJson
 *
 */

package com.example.antonio.darkdisplayv2;

import com.google.android.gms.maps.model.LatLng;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.List;

public class ParseJson {

    // Retorna o caminho entre a origem e o destino em forma de list de LatLngs ,
    // para ser consumido
    // pelo map application, e entao desenhar o caminho no mapa
    //
    public static List<LatLng> parseGoogleMapsJson(String json)
     throws JSONException {

        JSONObject result = new JSONObject(json);
        JSONArray routes = result.getJSONArray("routes");

        long distanceForSegment = routes.getJSONObject(0).getJSONArray("legs")
                .getJSONObject(0).getJSONObject("distance").getInt("value");
```

```
        List<LatLng> lines = new ArrayList<LatLng>();

        JSONArray steps = routes.getJSONObject(0)
         .getJSONArray("legs")
         .getJSONObject(0)
         .getJSONArray("steps");
        for(int i = 0; i < steps.length(); i++) {
            String polyline = steps.getJSONObject(i)
             .getJSONObject("polyline")
             .getString("points");

            for(LatLng p : decodePolyline(polyline)) {
                lines.add(p);
            }
        }

        return lines;
    }

    // Responsavel por pegar o conjunto de pontos criptografados e decolifica-los
    // uma list de LatLng. Metodo obtido da comunidade de desenvolvedores
    //
    private static List<LatLng> decodePolyline(String encoded) {

        List<LatLng> poly = new ArrayList<LatLng>();
        int index = 0, len = encoded.length();
        int lat = 0, lng = 0;

        while (index < len) {
            int b, shift = 0, result = 0;
            do {
                b = encoded.charAt(index++) - 63;
                result |= (b & 0x1f) << shift;
                shift += 5;
            } while (b >= 0x20);
            int dlat = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
            lat += dlat;

            shift = 0;
            result = 0;
            do {
                b = encoded.charAt(index++) - 63;
                result |= (b & 0x1f) << shift;
                shift += 5;
            } while (b >= 0x20);
            int dlng = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
            lng += dlng;

            LatLng p = new LatLng((((double) lat / 1E5)),
                    (((double) lng / 1E5)));
            poly.add(p);
        }

        return poly;
    }
}
```

```
/**
 * Classe PositionUtil
 *
 */

package com.example.antonio.darkdisplayv2;

import com.google.android.gms.maps.model.LatLng;

import java.util.List;


public class PositionUtil {

    public static int getClosestPointOfLatLngList(LatLng point,
     List<LatLng> points) {

        int index = 0;

        double minDistance = Math.sqrt(
         Math.pow((point.latitude - points.get(0).latitude), 2) +
         Math.pow((point.longitude - points.get(0).longitude), 2));

        for(int i = 0; i < points.size(); i++) {
            double auxDistance = Math.sqrt(
             Math.pow((point.latitude - points.get(i).latitude), 2) +
             Math.pow((point.longitude - points.get(i).longitude), 2));

            if(auxDistance < minDistance) {
                index = i;
                minDistance = auxDistance;
            }
        }

        return index;
    }

    /**
     * Funcao que retorna a direcao que o usuario deve seguir, em relacao ao norte,
     * na rota
     *
     * 0 - Parado, aguardando informacoes
     * 1 - Para frente
     * 2 - Para o nordeste, frente e direita
     * 3 - Para a direita
     * 4 - Para o sudeste, atras e direita
     * 5 - Para tras
     * 6 - Para o sudoeste, atras e esquerda
     * 7 - Para a esquerda
     * 8 - Para o noroeste, frente e esquerda
     *
     * */
    public static int getDirectionRelationNorth(LatLng now, LatLng prox) {
```

```
        int direction = 0; // Significa parado - o usuario deve aguardar informacoes

        if (now.latitude < prox.latitude) { // Frente
            if(now.longitude < prox.longitude) { // Direita
                direction = 2;
            } else if (now.longitude > prox.longitude) { // Esquerda
                direction = 8;
            } else { // Somente para frente
                direction = 1;
            }
        } else if (now.latitude > prox.latitude) { // Atras
            if(now.longitude < prox.longitude) { // Direita
                direction = 4;
            } else if (now.longitude > prox.longitude) { // Esquerda
                direction = 6;
            } else { // Somente para tras
                direction = 5;
            }
        } else { // Ou para esquerda ou para direita
            if(now.longitude < prox.longitude) { // Direita
                direction = 3;
            } else if (now.longitude > prox.longitude) { // Esquerda

                direction = 7;
            } else { // Parado, aguardando informacao
                direction = 0;
            }
        }

        return direction;
    }

    public static String indicateDirection(int direction) {
        String retorno = "";

        switch (direction) {
            case 0: retorno = "Parado"; break;
            case 1: retorno = "Frente"; break;
            case 2: retorno = "Nordeste"; break;
            case 3: retorno = "Direita"; break;
            case 4: retorno = "Sudeste"; break;
            case 5: retorno = "Atras"; break;
            case 6: retorno = "Sudoeste"; break;
            case 7: retorno = "Esquerda"; break;
            case 8: retorno = "Noroeste"; break;
        }
        return retorno;
    }
}



/**
 * Classe Route
 *
 */
```

```
package com.example.antonio.darkdisplayv2;

import android.graphics.Color;

import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Polyline;
import com.google.android.gms.maps.model.PolylineOptions;

import java.util.ArrayList;
import java.util.List;

public class Route {

    private List<LatLng>    mPoints;
    private Polyline        mPolyline;
    private PolylineOptions mPolyOptions;

    public Route(GoogleMap map) {
        mPoints      = new ArrayList<LatLng>();
        mPolyOptions = new PolylineOptions();
        mPolyline    = map.addPolyline(mPolyOptions);
        mPolyline.setColor(Color.BLUE);
    }

    public Route addPoint( LatLng point ) {
        if( point != null ) {
            mPoints.add(point);
            mPolyline.setPoints(mPoints);
        }
        return this;
    }

    public List<LatLng> getPoints() {
        return mPoints;
    }

    public void clearRoute() {
        this.getPoints().clear();
        mPolyline.setPoints(mPoints);
    }
}


/**
 * Classe UserPosition
 *
 */

package com.example.antonio.darkdisplayv2;

import android.location.Location;
import android.location.LocationManager;

import com.google.android.gms.maps.LocationSource;
import com.google.android.gms.maps.model.LatLng;
```

```
public class UserPosition implements LocationSource {

    private OnLocationChangedListener locationChangedListener;
    @Override
    public void activate(OnLocationChangedListener onLocationChangedListener) {
        this.locationChangedListener = onLocationChangedListener;
    }

    @Override
    public void deactivate() {

    }

    public void setLocation(LatLng location) {
        Location loc = new Location(LocationManager.GPS_PROVIDER);

        loc.setLatitude(location.latitude);
        loc.setLongitude(location.longitude);

        if(this.locationChangedListener != null) {
            this.locationChangedListener.onLocationChanged(loc);
        }
    }
}
```

```
/**
 * Classe VolleySingleton
 *
 */

package com.example.antonio.darkdisplayv2;

import android.content.Context;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.toolbox.Volley;


public class VolleySingleton {
    private static VolleySingleton mInstance;
    private RequestQueue mRequestQueue;
    private static Context mContext;

    private VolleySingleton(Context context) {
        mContext = context;
        mRequestQueue = getRequestQueue();
    }

    public static synchronized VolleySingleton getmInstance(Context context) {
        if(mInstance == null) {
            mInstance = new VolleySingleton(context);
        }

        return mInstance;
```

```
    }

    public RequestQueue getRequestQueue() {
        if(mRequestQueue == null) {
            mRequestQueue =
             Volley.newRequestQueue(mContext.getApplicationContext());
        }

        return mRequestQueue;
    }

    public <T> void addToRequestQueue(Request<T> req) {
        getRequestQueue().add(req);
    }
}
```

APÊNDICE B – Código-fonte da placa Arduino

```
int incomingByte = 0;   // Armazenar valor da porta USB

void setup() {
        pinMode(2, OUTPUT); //Dedo Polegar
        pinMode(3, OUTPUT); //Dedo Indicador
        pinMode(4, OUTPUT); //Dedo Médio
        pinMode(5, OUTPUT); //Dedo Anelar
        pinMode(6, OUTPUT); //Dedo Mínimo
        Serial.begin(9600);     // Abre a porta serial e atribui valor para 9600bps
}

void loop() {

                    digitalWrite(2, LOW);   // Desliga motor (LOW é voltagem baixa)
                    digitalWrite(3, LOW);
                    digitalWrite(4, LOW);
                    digitalWrite(5, LOW);
                    digitalWrite(6, LOW);

        if (Serial.available() > 0) { // Envia dado apenas se receber dado

                incomingByte = Serial.read(); // Ler dado e atribui à variável

                if(incomingByte == 0) //Compara dado da variável com '0'
                {
                    digitalWrite(2, LOW);
                    digitalWrite(3, LOW);
                    digitalWrite(4, LOW);
                    digitalWrite(5, LOW);
                    digitalWrite(6, LOW);
                    delay(1000);     // Espera 1000 milisegundos ( 1segundo )
                }

                if(incomingByte == 1) //NORTE
                {
                    digitalWrite(2, LOW);
                    digitalWrite(3, LOW);
                    //Liga motor de vibração (HIGH é voltagem alta)
                    digitalWrite(4, HIGH);
                    digitalWrite(5, LOW);
                    digitalWrite(6, LOW);
                    delay(1000);
                }

                if(incomingByte == 2) //NORDESTE
                {
                    digitalWrite(2, LOW);
                    digitalWrite(3, HIGH);
                    digitalWrite(4, HIGH);
                    digitalWrite(5, LOW);
                    digitalWrite(6, LOW);
                    delay(1000);
```

```
}

if(incomingByte == 3) //LESTE
{
    digitalWrite(2, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    digitalWrite(5, LOW);
    digitalWrite(6, LOW);
    delay(1000);
}

if(incomingByte == 4) //SUDESTE
{
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, LOW);
    digitalWrite(5, LOW);
    digitalWrite(6, LOW);
    delay(1000);
}

if(incomingByte == 5) //SUL
{
    digitalWrite(2, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(4, HIGH);
    digitalWrite(5, LOW);
    digitalWrite(6, HIGH);
    delay(1000);
}

if(incomingByte == 6) //SUDOESTE
{
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    digitalWrite(5, HIGH);
    digitalWrite(6, HIGH);
    delay(1000);
}

if(incomingByte == 7) //OESTE
{
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    digitalWrite(5, LOW);
    digitalWrite(6, HIGH);
    delay(1000);
}

if(incomingByte == 8) //NOROESTE
{
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, HIGH);
```

```
                digitalWrite(5, HIGH);
                digitalWrite(6, LOW);
                delay(1000);
            }

            if(incomingByte == 9) //PARADA
            {
                digitalWrite(2, HIGH);
                digitalWrite(3, HIGH);
                digitalWrite(4, HIGH);
                digitalWrite(5, HIGH);
                digitalWrite(6, HIGH);
                delay(1000);
            }
        }
}
```