ÁREA DE INFORMÁTICA

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - ADS

PATRÍCIA DE AMORIM PEREIRA

THIAGO SANTOS GONÇALVES

APLICATIVO PARA SIMULAÇÃO DE DIRECIONAMENTO DE VEÍCULOS UTILIZANDO GPS

TRABALHO DE CONCLUSÃO DE CURSO - TCC

CARAGUATATUBA

2016

APLICATIVO PARA SIMULAÇÃO DE DIRECIONAMENTO DE VEÍCULOS UTILIZANDO GPS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo, da Área de Informática, do Instituto Federal de São Paulo.

Orientador:
Prof. Dr. Ederson Rafael Wagner

CARAGUATATUBA

2016

TERMO DE APROVAÇÃO

APLICATIVO PARA SIMULAÇÃO DE DIRECIONAMENTO DE VEÍCULOS UTILIZANDO GPS

por

PATRÍCIA DE AMORIM PEREIRA
THIAGO SANTOS GONÇALVES

Este Trabalho de Conclusão de Curso (TCC) foi apresentado(a) em 16 de fevereiro de 2016 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas (ADS). Os candidatos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados, a qual após deliberação, considerou o trabalho aprovado.

_____
Ederson Rafael Wagner
Prof. Orientador

_____
Mario Tadashi Shimanuki
Presidente

_____
Denny Azevedo
Membro

“Que a força esteja com você”.

JR., George Walton Lucas, 1977.

Dedicamos este trabalho à nossas famílias, amigos, a nosso orientador, que em tanto nos confiou e nos ajudou e a todos os professores que, mais que profissionais, são nossos exemplos de vida.

AGRADECIMENTOS

Agradeço primeiramente, a minha família, por sempre estar presente e incentivar minha sede de conhecimento. Ao corpo docente do IFSP que, tive a honra de conhecer, desde que fiz ensino técnico. Especialmente aos, mais que professores, Ederson Rafael Wagner (orientador e amigo), Nelson Alves e Lineu Mialaret. A nosso terceiro componente desse TCC e grande amigo, Igor Camargo, pela força, amizade, companheirismo e comprometimento durante os três anos de curso e aos anos de amizade afora. Aos amigos que fizemos ao longo de nossa jornada no curso ADS. Àqueles que tive a felicidade de conhecer neste câmpus do IFSP Caraguatatuba, no ensino técnico: Bruno Campos, Bruno Justen, Leandro Castilho, Milena Negrão e Henrique Azevedo. Aos meus incentivadores natos, que todos os dias me inspiram com suas existências e histórias de vida, Carla Nogueira, Eunice Feliciano, Beatriz Alcantara, Anna Claudia, Luis Filipe (Brtt), Bruno Tavares, João Donizetti, Ronildo Arcelino e João Cardozo. E por último e não menos importante ao meu suporte em todos esses anos, pela grande amizade, confiança, respeito e companheirismo Thiago Santos, o qual tenho a honra de dividir e concluir este trabalho.

Patrícia Amorim

Gostaria de agradecer ao nosso orientador o Prof. Dr. Ederson, um exemplo de pessoa a ser seguido, sou muito grato pela honra de ter tido alguém tão inteligente como orientador. Ao Prof. Eduardo por todo o suporte com Android, sua ajuda foi fundamental para a conclusão desse trabalho. A minha família por estar ao meu lado nos meus momentos de estresse e dificuldade. Aos meus colegas de trabalho, em especial a minha “chefe” Nélia, por todos os sábios conselhos e puxões de orelha. Ao Igor, que sempre esteve junto conosco, nas alegrias e tristezas, nas apresentações de trabalho e nos salvando nas dificuldades. Aos meus amigos que sempre estão ao meu lado Luiz, Kalil, Fabricio, Djey, Tamires, Luana I., Leticia, Myrella e Luana N. Ao melhor grupo de pessoas do mundo, Bruno Campos, Bruno Justen, Leandro, Milena e Henrique, vocês são os melhores. E finalmente Patrícia, palavras não podem expressar o quão importante é a sua amizade para mim, obrigado por toda a parceria e comprometimento ao longo de todos esses anos de amizade, sem dúvidas eu dificilmente teria chegado até aqui sem o seu apoio e ajuda no decorrer desses anos.

Thiago Santos

RESUMO

Este trabalho de conclusão de curso visa o desenvolvimento de um software que processe informações sobre posicionamento, e calcule os próximos passos, transmitindo os dados para um protótipo robótico baseado em microcontolador arduino, que possua um protocolo de entrada e tratamento de dados padrão com o projeto, para que o mesmo chegue ao destino de uma rota pré-definida, sem intervenção humana. Através desse trabalho, será possível contribuir com a comunidade pesqueira da região do litoral norte do estado de São Paulo, demonstrando a possibilidade de implantação do *software* no *smartphone* e no microcontrolador na embarcação permitindo assim, aos pescadores, elaborarem uma rota no dispositivo telefônico e sua embarcação poder realizá-la sozinha.

Tem por objetivos a produção de código fonte em linguagem Java, que pode interpretar a localização inicial, através do receptor *GPS* do *smartphone*, e assim, calcular como realizar a rota já definida pelo usuário. A demonstração se dará através do *software* indicando os próximos pontos a serem percorridos.

O processamento do *software* por meio da indicação das ações futuras do protótipo robótico são suficientes para comprovar seu pleno funcionamento e, aplicabilidade de implantação no protótipo robótico e, futuramente em embarcação.

Palavras-chave: Aplicativo. Android. Arduino. *GPS*.

ABSTRACT

This final project has the purpose to develop a software that process information about position, calculate next steps and send data to a robotic prototype. The robotic prototype is based in an Arduino microcontroller. This prototype has an input protocol and a pattern treatment with the software. As a result, the robot can arrive to the destiny without human intervention. Through this project, it will be possible to contribute with the fishing community from the northern coast of Sao Paulo. The contribution demonstrates the possibility of implementation of the software in smartphone and microcontroller on vessel. Consequently, a fisherman can draw up a route on smartphone, and his boat can navigate by itself.

The software will be developed in programming language Java. This source code can interpret the initial location through the GPS of the smartphone; and so, processing a default route from the user. The demonstration will be through the software showing the next points that will be followed.

As a final point, the processing of the software through the indication of the future action of the robot prototype are enough to prove the robot operation. Also, the applicability implementation in robotic prototype, and in the future, the implementation in vessel.

Keywords: App. Android. Arduino. *GPS*.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

LISTA DE ABREVIATURAS

A.S.	*Anti-spoofing*
I. A.	Inteligência Artificial
R. F.	Rádio Frequência
S. M. S.	*Short Message Service*
S. O.	Sistema Operacional
S.A.	*Selective Availability*
θ	Théta

LISTA DE SIGLAS

ASF	*Apache Software Foundation*
C/A	*Coarse Aquisition*
EPPROM	*Erasable Programmable Read-Only Memory*
PPS	*Precise Positioning Service*
PWM	*Pulse With Modulation*
SPS	*Standard Positioning Service*
SRAM	*Static Random Access Memory*
XML	*Extensible Markup Language*

LISTA DE ACRÔNIMOS

AAF	*American Air Force*
CPTEC	Centro de Previsão de Tempo e Estudos Climáticos
CPU	*Central Processing Unit*
DoD	*Department of Defense*
GPS	*Global Positioning System*
IBGE	Instituto Brasileiro de Geografia e Estatística
IDE	*Integrated Development Environment*
INPE	Instituto Nacional de Pesquisas Espaciais
JVM	*Java Virtual Machine*
MAC	*Media Access Control*
MASER	*Microwave Amplification by Stimulated Emission of Radition*
MCS	*Master Control Station*
NAVSTAR	*Navigation System with Time And Ranging*
NIMA	*National Imagery and Mapping Agency*
OHA	*Open Handset Alliance*
PCB	*Printed Circuit Board*
PRN	*Pseudo Random Noise*
RAM	*Random Access Memory*
RF	Radio Frequência
ROM	*Read-Only Memory*
SDK	*Software Development Kit*
SIG	Sistema de Informação Geográfica
SQLite	*Structured Query Language*
TI	Tecnologia da Informação
VANT	Veículo Aéreo Não Tripulado

SUMÁRIO

1. INTRODUÇÃO

Este trabalho de conclusão de curso visa o desenvolvimento de um *software* que simule e esteja configurado para enviar dados para um protótipo robótico desenvolvido em microcontrolador arduino, com o objetivo de guiar ao destino de uma rota pré-definida. Através desse trabalho, será possível contribuir com a comunidade pesqueira da região do litoral norte, no sentido de desenvolver um sistema que poderá ser aplicado a suas embarcações e sendo capaz, assim, de direcioná-los em alto mar.

O posicionamento foi um problema em que o homem sempre procurou solucioná-lo para de fato, saber onde está. Hoje com as tecnologias de posicionamento global, pode-se saber, em tempo real, com base em um sistema referencial, onde está determinado objeto ou pessoa. A comunidade pesqueira do litoral norte, sendo uma das maiores do estado de São Paulo, conforme levantamentos realizados pelo Instituto de Pesca do Governo do Estado de São Paulo, em termos de produção e comercialização do pescado, utiliza de métodos antigos e pouco precisos com relação a posicionamento geográfico. Através dos dados que serão explanados, se entenderá o quanto uma melhoria em seus métodos poderia atrair uma produção maior, fazendo com que a renda e a economia da região aumente.

Nos próximos capítulos serão apresentados o contexto em que esse trabalho está inserido, bem como os conceitos necessários para desenvolvê-lo. Explanando também as funções utilizadas e conceitos científicos que foram imprescindíveis para embasar a solução do problema apontado. Os processos de construção do sistema, sua documentação, metodologia e todo o contexto científico-histórico em que está inserido, bem como sua adaptação para o meio.

1.1 Objetivos do Trabalho

Esse trabalho tem por objetivo geral estudo e pesquisa científica no desenvolvimento de *software* em linguagem Java para sistema operacional Android, além da possibilidade de integração com o microcontrolador arduino, como por exemplo, o protótipo robótico construído pelo aluno Pedro Henrique Nakanishi, aluno do 5º período do curso de Análise e Desenvolvimento de Sistemas, o qual o desenvolveu como requisito para outra disciplina, fazendo com que o mesmo navegue em uma rota sem intervenção humana, diante da rota pré-estabelecida.

Os objetivos específicos são:

- Conhecimento, pesquisa e produção de *softwares* em Java para S.O. Android;
- Conhecimento, pesquisa e produção de código para controle;
- Produção de conhecimento em autonomia e robótica;
- Utilização de metodologias ágeis de produção, avaliando sua aplicabilidade e eficácia em projetos.

1.2 Justificativas

O objetivo da produção desse *software* para possível implantação no protótipo robótico desenvolvido em paralelo pelo Sr. Pedro Henrique Nakanishi, vai além das comprovações e elaborações de algoritmos. Abaixo, serão apresentados dentro dos contextos demográficos, geográficos, sociais e culturais os motivos pelos quais o *software* poderá ser implantado em embarcações pesqueiras de pequeno porte.

1.2.1 O litoral norte

Como mostra a pesquisa e levantamento do Instituto Brasileiro de Geografia e Estatística (2011), o litoral paulista faz fronteira com o estado do Rio de Janeiro, pertencendo também a mesorregião do Vale do Paraíba. Com população estimada (em 2011 pelo IBGE) de 286.163 mil habitantes, sendo divido entre as quatro

cidades: Caraguatatuba, Ilha Bela, São Sebastião e Ubatuba. Na Figura 1, observa-se a área total do litoral norte que possui cerca de 1.947,702 km².

Figura 1 - Mapa do litoral norte do estado de São Paulo.
Fonte: CPTEC INPE. Disponível em: <http://nridaln.cptec.inpe.br/> Acesso em nov. 2015.

A Serra do Mar domina a paisagem da microrregião e seu clima é oceânico deixando as temperaturas entre 20 e 30ºC, tendo um índice pluviométrico moderado, considerado chuvoso.

1.2.2 A comunidade pesqueira

Conforme estudo realizado por Diegues (2000), o meio em que a população se instala e sua natureza que a cerca definirá o tipo de relacionamento que haverá entre os dois. A natureza fornece ao homem, meios de subsistência, de trabalho e produção e meios de produzir os aspectos materiais das relações sociais.

O caiçara, definido como o habitante da zona litorânea, possui um território descontínuo, marcado por vazios aparentes que são usadas para a pesca. E na maioria das vezes, declaradas como unidades de conservação. Os pescadores estão espalhados por todo o litoral, rios e lagos. Ainda que exerçam outras atividades econômicas, seu modo de vida é baseado na pesca. Os pescadores, sobretudo os artesanais, praticam a pequena pesca, cuja produção em parte é consumida pela família e em parte é comercializada. A unidade de produção é, em

geral, a familiar, incluindo na tripulação conhecidos e parentes mais longínquos. Apesar de grande parte deles viverem em comunidades litorâneas, não urbanas, alguns moram em bairros urbanos ou periurbanos, construindo aí uma solidariedade baseada na atividade pesqueira.

1.2.3 A pesca na região

O Instituto de Pesca do Governo do Estado de São Paulo (2015) apresenta os dados de pesquisas realizadas com a comunidade pesqueira do litoral de São Paulo, compreendendo os municípios: Bertioga, Cananéia, Caraguatatuba, Cubatão, Iguape, Ilha Comprida, Ilhabela, Itanhaém, Monguagá, Peruíbe, Praia Grande, Santos/Guarujá, São Sebastião, São Vicente e Ubatuba.

A Tabela 1, fornecida pelo Instituto de Pesca sobre a produção de pescado na comunidade pesqueira do litoral norte:

Tabela 1- Quantidade de pescado nos anos de 2012, 2013 e 2014 no litoral norte do estado de São Paulo, dividido por municípios.

Município	**Quantidade de pescado por ano**		
	2012	**2013**	**2014**
Caraguatatuba	156.900,75 kg	106.868,94 kg	106.292,99 kg
Ilhabela	738.552,31 kg	760.503,79 kg	602.586,32 kg
São Sebastião	717.286,24 kg	461.707,55 kg	403.099,36 kg
Ubatuba	2.459.739,25 kg	2.341.485,95 kg	3.728.363,09 kg
Totais	4.072.478,55 kg	3.670.566,23 kg	4.840.311,76 kg

Fonte: Instituto de Pesca. Disponível em: <http://www.propesq.pesca.sp.gov.br/>. Acesso em nov. 2015.

Compreende-se pela análise que a produção só aumenta com o passar dos anos. Esses dados referem-se a pesca artesanal, que é realizada por pequenos pescadores e suas embarcações de pequeno porte. A produção de pescado no litoral norte representou 15% da produção total do estado de São Paulo no ano de 2012, e hoje, chega a representar cerca de 21%.

A Tabela 2 apresenta os valores financeiros relacionados a pesca na região:

Tabela 2 - Valor estimado da arrecadação com a venda do pescado no litoral norte do estado de São Paulo, dividido por município.

Município	Valor da produção por ano		
	2012	2013	2014
Caraguatatuba	R$ 1.484.809,51	R$ 1.252.933,62	R$ 1.600.018,85
Ilhabela	R$ 4.094.560,24	R$ 5.174.827,43	R$ 4.625.422,88
São Sebastião	R$ 4.332.013,76	R$ 3.970.961,42	R$ 4.259.997,38
Ubatuba	R$ 10.714.225,50	R$ 8.856.097,72	R$ 11.411.857,48
Totais	R$ 20.625.609,01	R$ 19.254.820,19	R$ 21.897.296,59

Fonte: Instituto de Pesca. Disponível em: <http://www.propesq.pesca.sp.gov.br/>. Acesso em nov. 2015.

Percebe-se que a estimativa de venda do pescado representa 20% do valor de todo o estado de São Paulo. A Figura 2 mostra um gráfico em que entendem-se os valores movimentados no Litoral Norte em relação a todo o estado de São Paulo:

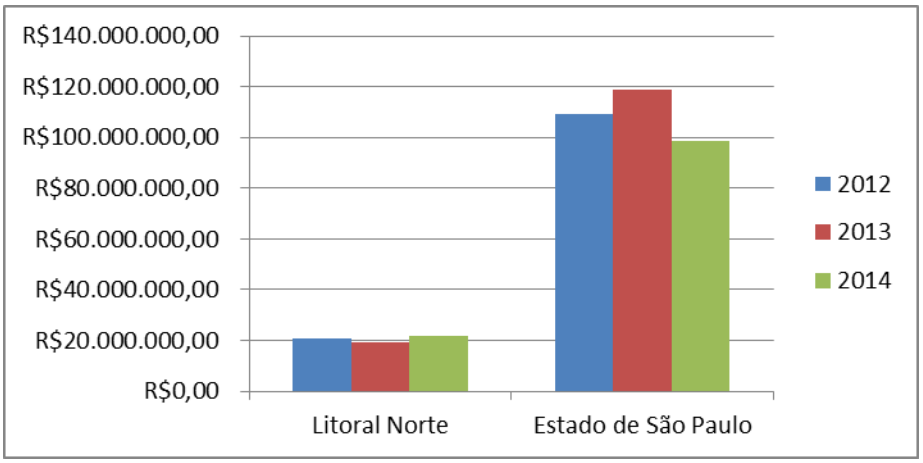

Figura 2 - Gráfico que representa a estimativa de valores arrecadados com a venda do pescado no litoral norte em relação a arrecadação de todo o estado de São Paulo.
Fonte: Elaborado pelos autores.

Analisando os dados fornecidos pelo Instituto de Pesca (2015), compreende-se a grande movimentação financeira que a pesca gera em todo o litoral paulista, sendo portanto, fonte de sustento de inúmeras famílias.

2. EMBASAMENTO TEÓRICO

Nesse capítulo, serão apresentados os conceitos específicos e científicos que foram utilizados para a elaboração do *software* que irá simular a transmissão de dados e direcionamento de protótipo robótico construído em microcontrolador arduino.

2.1 Arduino

Arduino é um microcontrolador em placa única que consiste em uma prototipagem simples e livre para o controlador (MCROBERTS, 2011). Em seu *site* Arduino (2015), diz que sua prototipagem é rápida, baseada em *hardware* de fácil utilização. Desenvolvido pelo *Interation Design Institute Ivrea*, sua origem se deu da idéia de ferramenta fácil para estudantes sem conhecimentos em eletrônica e programação. Sua plataforma de código aberto recebe entradas desde sensores, botões e até mesmo mensagens S.M.S. e mensagens de *Twitter*, processar as informações recebidas e transformá-las em saídas que podem enviar sinais para motores executarem tarefas. Seu *hardware* é barato, que têm versões, inclusive que podem ser montadas à mão, sem necessidade de ferramentas, custando menos de US$ 50,00 (cinquenta dólares).

Conforme Santos (2009), o Arduino é um sistema que interage com o ambiente através de seu *hardware* e *software*, em suma, uma plataforma de computação física ou embarcada.

Devido a grande quantidade de material acessível e pelo baixo custo dessa plataforma é que existem muitos projeto de robótica que são desenvolvidos com Arduino, com a intenção de provar hipóteses de estudos e pesquisas científicas. A utilização do microcontrolador Arduino como plataforma, adicionando o *software* desenvolvido em Java, por esse projeto e esses alunos, irão demonstrar a possibilidade de realização de um trajeto sem intervenção humana, utilizando somente códigos de localização e comparação de posicionamento.

Segundo Arduino (2015) demonstra em seu *site*, para receber e interpretar essas entradas é que existe uma ferramenta que vai proporcionar um ambiente de desenvolvimento para que seja elaborada uma aplicação com as instruções para que o *hardware* as faça. A ferramenta disponível para programação na plataforma Arduino é *open-source*, baseada na linguagem de programação C, também chamada pelo mesmo nome, e está em grande crescimento devido às contribuições dos usuários de todo o mundo. O diferencial do *software* de outros microcontroladores, é que ele apresenta facilidade de entendimento para novatos e contempla as complexidades da programação para usuários avançados que querem produzir resultados mais robustos. É uma ferramenta multiplataforma, que pode ser executada nos sistemas operacionais iOS, Windows e Linux.

Segundo Santos (2009), o Arduino conecta-se a leds, *displays*, botões, interruptores, motores, sensores de temperatura, sensores de pressão, sensores de distância, receptores *GPS*, módulos ethernet ou qualquer outro dispositivo que emita dados ou possa ser controlado, estendendo as suas funcionalidades. Os *shields*, placas que podem ser conectadas em cima do Arduino estendendo suas capacidades, também estendem os pinos até o topo de suas próprias placas de circuito, para que se possa continuar tendo acesso a todos eles. Não é estritamente necessário o uso de *shields*, também é possível fazer o circuito utilizando uma *protoboard*, *stripboard*, *veroboard* ou criando uma PCB (“*Printed Circuit Board*”: placa de circuito impresso), conforme a Figura 3, que representa o microcontrolador conectado a *shields*.

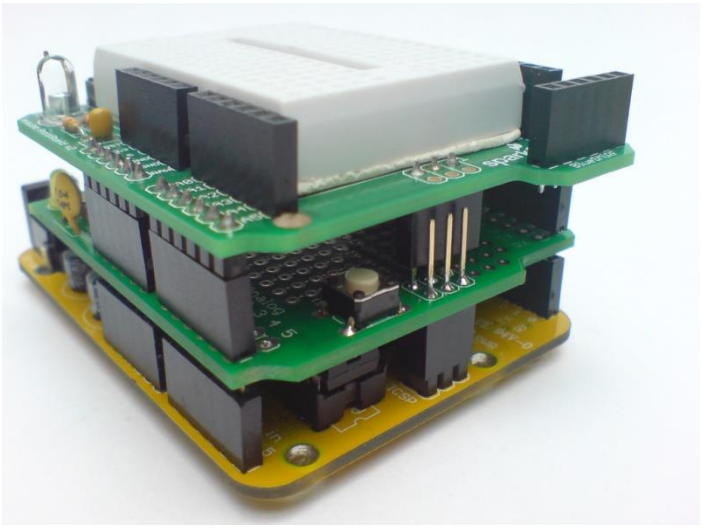

Figura 3 - Arduino conectado a *shields* e *protoboard*.
Fonte: Wikipedia. Disponível em: <https://upload.wikimedia.org/wikipedia/commons/d/df/Arduino_Protoboard_Shields.jpg>. Acesso em nov. 2015.

A Figura 4 ilustra o *hardware* de um arduino uno que é um kit de desenvolvimento baseado no controlador Atmega328P.

Figura 4 - Arduino UNO.
Fonte: Site do Arduino. Disponível em: <https://www.arduino.cc/en/Guide/Windows>. Acesso em nov. 2015.

2.1.1 Atmega 328P

Conforme Santos (2009), um microcontrolador é desenhado e construído de forma a integrar diversos componentes em um único circuito integrado. Não necessita que adicione componentes externos que permitam suas funcionalidades.

Por análise entende-se a grande quantidade de funcionalidades que podem ser reunidas em um único circuito, por exemplo: memória RAM, memória ROM, CPU,

portas e etc. Na Figura 5, segue diagrama de blocos da arquitetura dos microcontroladores Atmega tecnologia AVR.

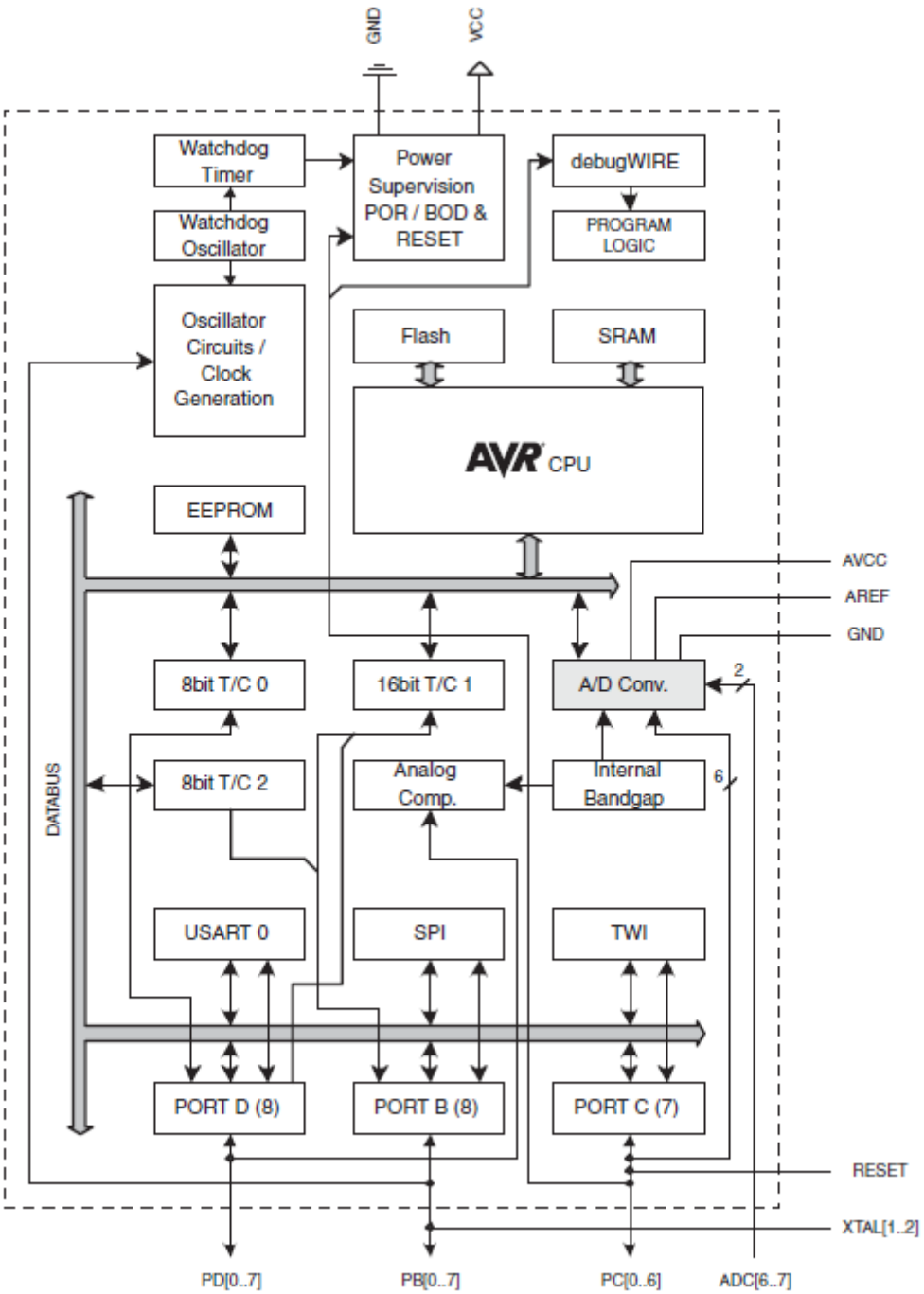

Figura 5 - Arquitetura do microcontrolador Atmega.
Fonte: Blog DQSoft. Disponível em: <http://dqsoft.blogspot.com.br/2011/07/microcontrolador-atmel-atmega328-parte.html>. Acesso em nov. 2015.

Segundo Quadros (2011), a separação de dados e programas é uma característica da arquitetura Harvard. Somente as instruções armazenadas na memória *flash* são

executadas, não sendo possível executar código na SRAM. Outros modelos Atmega possuem memórias tipo EEPROM que estão diretamente ligadas a via de conexão aos periféricos, não sendo acessada pelas instruções normais de acesso a memória.

Como Santos (2009) explica, as diferentes memórias que podem compor um microcontrolador são a *flash*, SRAM e EEPROM, que estão descritas no Quadro 1:

Quadro 1 - Tipos de memórias utilizadas em Arduino.

Flash	É aqui que o sketch e o bootloader são armazenados
SRAM	Onde são criadas e modificadas as variáveis ao longo da execução do *sketch*
EEPROM	Pode ser usada para guardar informação (não volátil)

Fonte: Site Arduino. Disponível em: <http://www.arduino.cc>. Acesso em nov. 2015.

2.1.2 Pinagem/Conectores

Como diz Santos (2009), um arduino possui pinos (conectores) que fazem as ligações com *shields* ou *protoboards,* ou mesmo alimentação, que recebem dados exteriores ou enviam resultados de processamento.

Observa-se na Figura 6, os 29 (vinte e nove) conectores/pinos encontrados em um Arduino UNO:

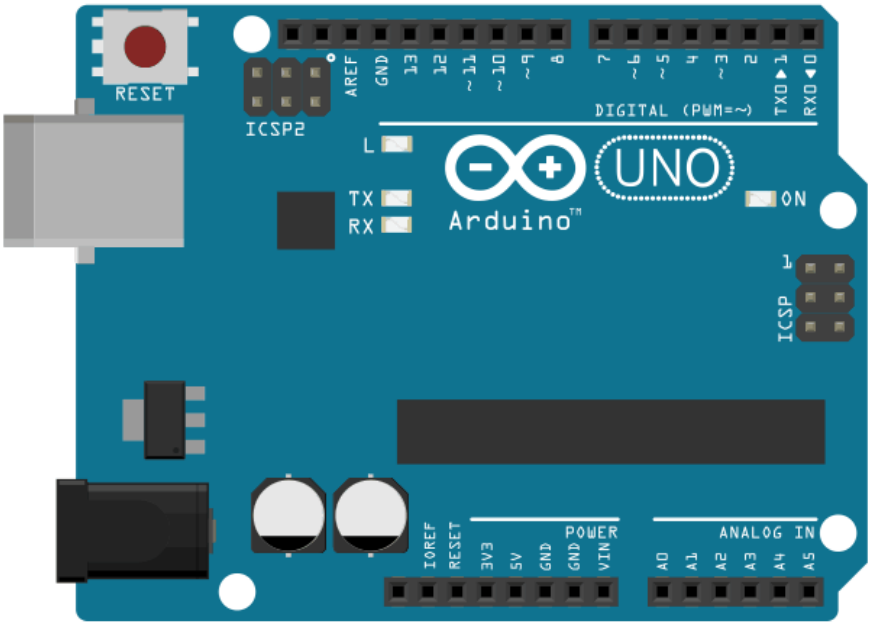

Figura 6 - Conectores/Pinos de um microcontrolador Arduino UNO.
Fonte: *Site* Embarcados. Disponível em: <http://www.embarcados.com.br/arduino-entradasaidas-digitais/>. Acesso em fev. 2016.

Conforme Souza (2013), em um Arduino UNO, existem 14 (catorze) entradas ou saídas digitais, numeradas de 0 a 13. Nelas podem ser conectadas teclas de entrada, *LED's* para saída de informações, desde que devidamente configuradas. Desses pinos, cerca de 6 (seis) são PWM (*Pulse Width Modulation*), os quais são explanados mais adiante nesse trabalho.

Também possui 6 (seis) pinos analógicos para conectar com interfaces analógicas com resolução de 10 bits, numeradas de A0 a A5. Para mudar a referência desses pinos pode-se utilizar o conector AREF (*Analog Reference*).

Para alimentação possui 3 (três) pinos GND (*Ground*) que têm função de referência terra, evitando possíveis curtos circuitos; 1 (um) pino de alimentação de 5v e 1 (um) de alimentação de 3.3v; 1 (um) pino de *RESET*, com função de resetar externamente o arduino; 1 (um) pino IOREF para utilização pelos *shields*, selecionando a interface apropriada adequando a tensão da placa para utilização do *shield*; 1 (um) pino VIN para alimentação externa através de um *shield* ou bateria.

2.1.3 *PWM*

A Modulação por Largura de Pulso (MLP), também conhecida como *Pulse Width Modulation* (PWM), é uma técnica para obter resultados em forma de onda periódica. Esse resultado nada mais é do que um padrão ligado/desligado para simular voltagens entre totalmente ligado e desligado, essa variação ocorre de acordo com o tempo que existe a emissão do sinal (ligado) e o tempo que não a emissão de qualquer sinal (desligado), como na Figura 7, representada.

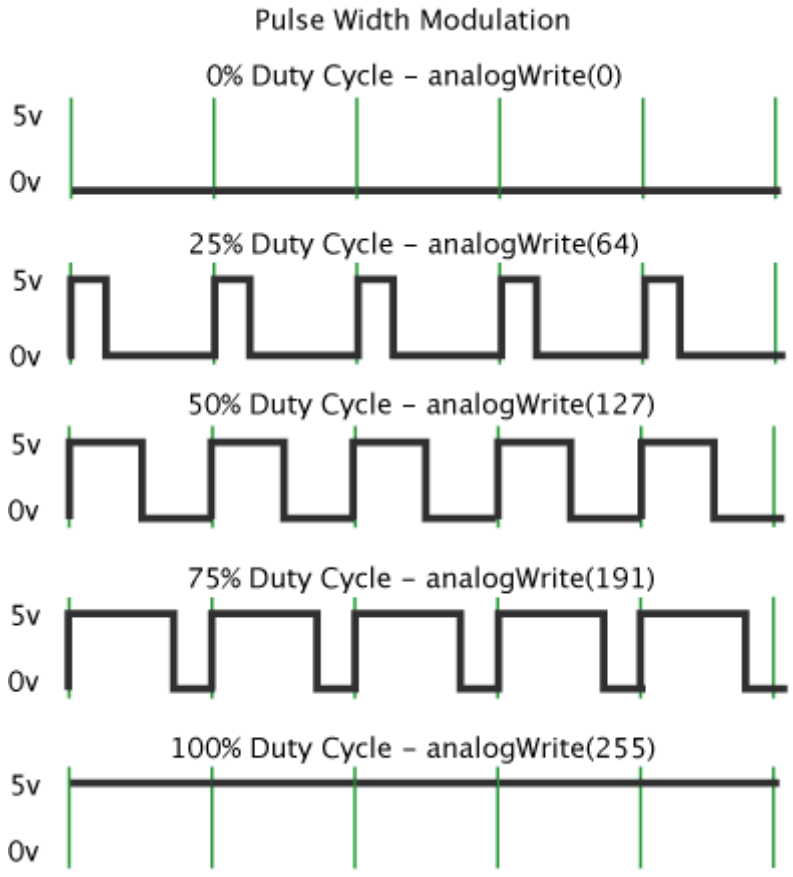

Figura 7 - Representação de diferentes valores de *PWM*.
Fonte: Site Arduino. Disponível em: <https://www.arduino.cc/en/Tutorial/PWM>. Acesso em nov. 2015.

2.2 Android

Android é um sistema operacional baseado em Linux e foi lançado em 2007. É executado no *kernel* 2.6 do sistema operacional Linux. Conforme afirma Lecheta (2013), o sistema operacional Android gerencia a memória, processos, *threads* e segurança de arquivos, pastas, redes e *drivers*. Sua segurança é baseada no Linux,

ou seja, cada processo é resultado da execução de uma aplicação, que possui uma *thread* dedicada a ele. Suas aplicações possuem um usuário criado no sistema operacional para cada uma delas, sendo assim, cada usuário possui apenas acesso a estrutura de diretórios de uma aplicação, sendo que nenhum outro usuário pode ter acesso a essa mesma aplicação. Além disso, o *kernel* controla todo o uso da memória no dispositivo. A Figura 8, ilustra a aparência do sistema operacional Android em um smartphone.

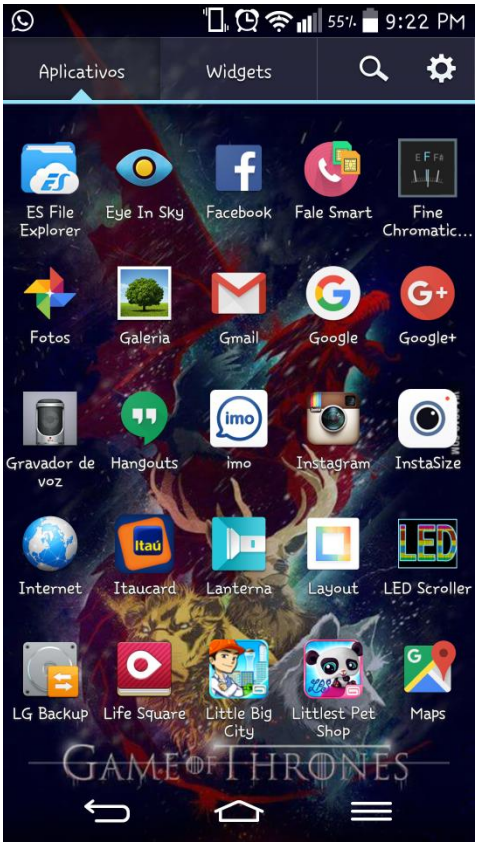

Figura 8 - Sistema operacional Android em *smartphone* e seus aplicativos.
Fonte: Elaborado pelos autores.

O sistema operacional Android surgiu por um grupo de empresas conhecido como OHA – *Open Handset Alliance* que, hoje, tem 87 (oitenta e sete) empresas dos ramos de operadoras móveis, fabricantes de celulares, empresas de semicondutores, empresas de softwares e comercialização. O grupo líder em tecnologia móvel, conta com empresas como LG, Nextel, Telefonica, Acer, Intel, Google, Sony Ericsson, Samsung dentre outras, disponível em seu site OHA (2015).

Sendo um sistema operacional de código aberto (*open-source*) – Licença ASF (*Apache Software Foundation*), diferente de *software* livre, Google (2015), em seu

site, afirma que o objetivo principal do Android é ter uma plataforma aberta para as operadoras, empresas fabricantes e desenvolvedores que querem tornar suas ideias realidade, tendo um produto que corresponde às necessidades reais dos usuários. Contudo, essa capacidade de personalização pode descaracterizar o sistema operacional Android, bem como conduzir a implementações incompatíveis. Para isto, a empresa Google mantém um programa de compatibilidade Android, onde é solicitado às empresas, uma exigência mínima para tornar seu produto desenvolvido a partir do código fonte do Android, um produto compatível.

Um ponto interessante no S.O. Android, talvez o mais importante, como afirma DiMarzio (2008), é que sua arquitetura permite que as aplicações e os pacotes com núcleo do sistema sejam executados com mesma prioridade. Além disso, cada aplicativo é executado usando muito pouco da máquina virtual do Android. Na Figura 9, representada, é demonstrada a arquitetura do sistema operacional Android.

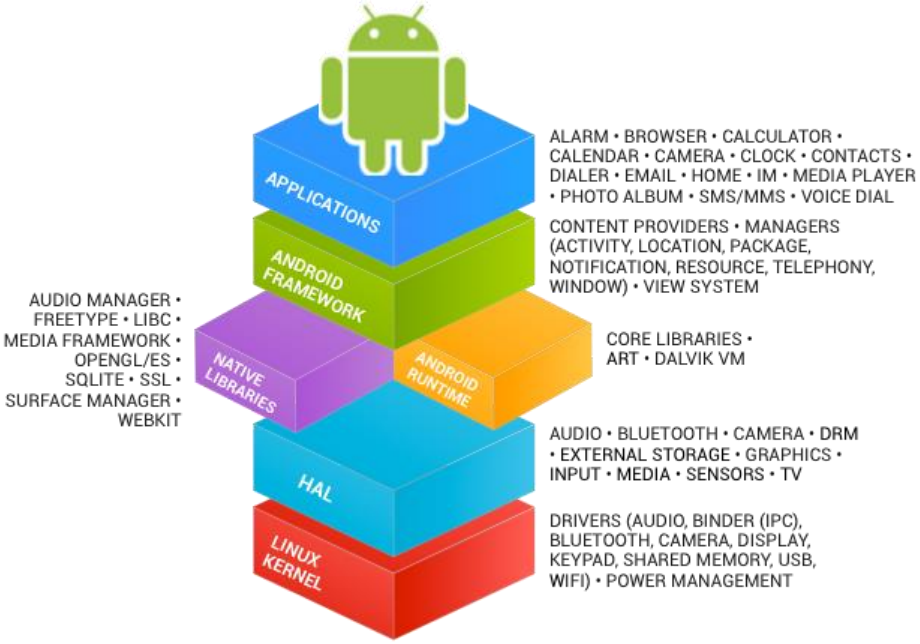

Figura 9 - Arquitetura do sistema operacional Android.
Fonte: Google. Disponível em: <http://developer.android.com/index.html>. Acesso em nov. 2015.

O SDK (*Software Development Kit*) e as bibliotecas fazem com que os programadores utilizem e tenham acesso a todos os recursos que o S.O. também tem acesso, ou seja, *GPS*, discador do telefone dentre outros. Através disso, os aplicativos acabam por não ter restrições, podendo ser desenvolvidos utilizando

dessas ferramentas. Na Figura 10, encontra-se a descrição detalhada do S.O. Android.

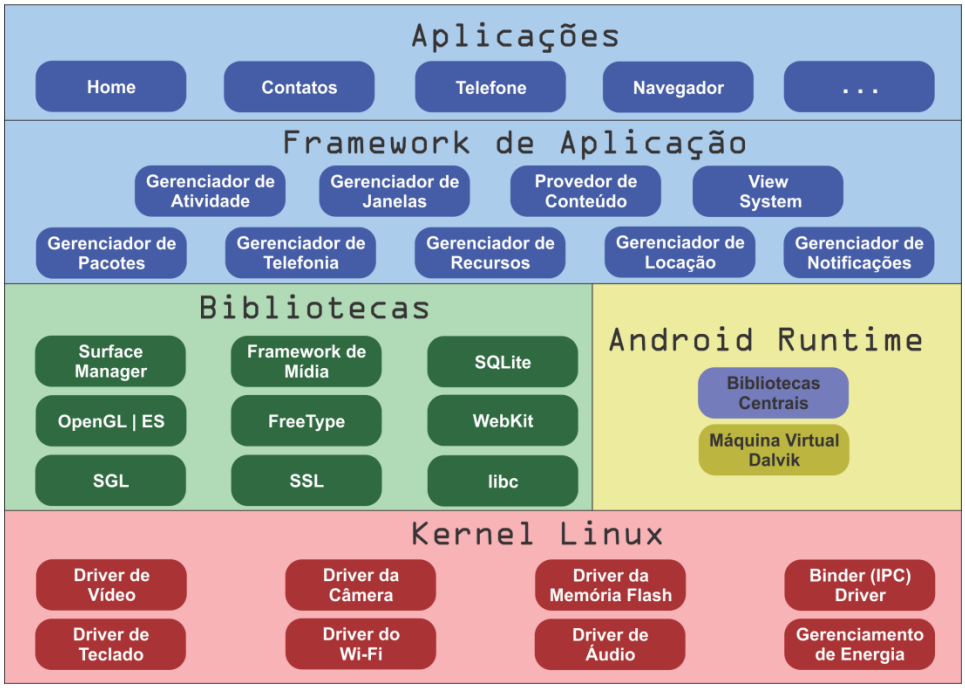

Figura 10 - Descrição do sistema operacional Android detalhada.
Fonte: Elaborada pelos autores.

Para desenvolver aplicações para Android é necessário conhecimento em programação orientada a objetos, pelo fato da utilização da linguagem Java e conhecimento em XML, que é o conceito do *layout* da interface do usuário.

Para executar as aplicações, Lecheta (2013) diz que o S.O. utiliza uma máquina virtual para compilar, sendo essa a máquina virtual Dalvik. Ela já vem instalada junto ao sistema operacional e utiliza uma arquitetura baseada em registros. É otimizada para executar os aplicativos nos dispositivos móveis, uma vez que a construção dessas aplicações é em linguagem Java.

A aplicação desenvolvida em Java tem seu *bytecode* (.class) compilado e convertido para a extensão ".dex" (Dalvik *executable*). Após, toda a aplicação e seus recursos, são compactados em um arquivo no formato ".apk" (*Android Package File*), o qual está pronto para distribuição e instalação nos dispositivos com S.O. Android, como ilustra a Figura 11.

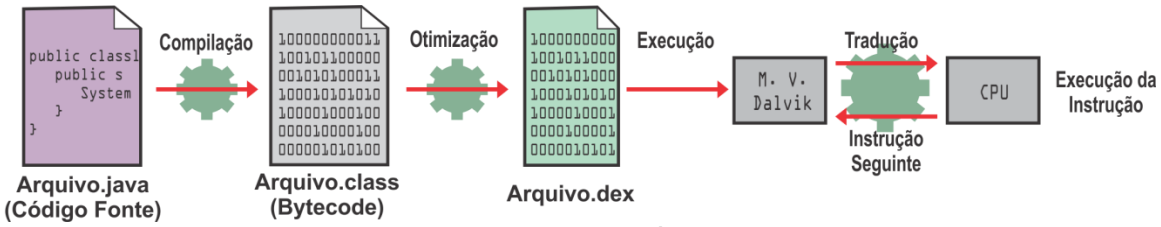

Figura 11 - Como funciona a máquina virtual Dalvik.
Fonte: Elaborada pelos autores.

O SDK do Android é bem completo e possui emulador para simular a aplicação desenvolvida no ambiente do dispositivo do celular. Em linguagem Java e utiliza o banco de dados *SQLite* para as aplicações que necessitam.

Conforme as estatísticas da Google (2015), a cada dia, mais de um milhão de usuários ligam seus *smartphones* com S.O. Android pela primeira vez em mais de 190 (cento e noventa) países ao redor do mundo. Android possui a maior comunidade de desenvolvedores do mundo, com muitas empresas envolvidas, incluindo também, a comunidade *open-source* do S.O. Linux.

Com um mercado aberto, o Android comercializa as aplicações de forma rápida e fácil para os desenvolvedores, os quais podem controlar toda a venda dos produtos, direcionando ou não a um público específico.

Sendo assim, um S.O. para dispositivos móveis com uma variedade extensa de material disponível para desenvolvimento de aplicações.

2.3 Localização Global

A localização foi um dos primeiros problemas científicos levantados pelo ser humano, movido pelo interesse de saber onde estava e pelo conhecimento de novas terras. Com o desenvolvimento da navegação marítima surgiram os instrumentos de orientação e por fim o *GPS – Global Positioning System* – Sistema de Posicionamento Global (ALBUQUERQUE; SANTOS, 2003).

Albuquerque e Santos (2003) relatam que, a corrida espacial deu início em 4 de outubro de 1957, com o lançamento do primeiro satélite o Sputnik I, que permitiu o rastreamento da órbita do satélite. Posteriormente outro pesquisador fez o inverso, através da órbita, determinou as coordenadas terrestres, surgindo assim o conceito

GPS. Os setores da microeletrônica e comunicação via satélites contribuíram para a ideia e assim foram lançados outros satélites artificiais.

2.3.1 Geoprocessamento e SIG's

Segundo Albuquerque e Santos (2003), geoprocessamento é a informática especializada para tratar de dados espaciais (mapas) e dados alfanuméricos, ou seja, um conjunto de *softwares* e equipamentos computacionais que são utilizados para implantação de Sistemas de Informações Geográficas (SIGs). Ele caracteriza-se por duas funções topológicas, que realizam operações de pertinência, proximidade e interseção entre elementos de mapas distintos.

O sistema de informação geográfica é o produto gerado da entrada de dados nos sistemas de geoprocessamento. É ilustrado na Figura 12, a função de cada conceito no sistema de posicionamento global:

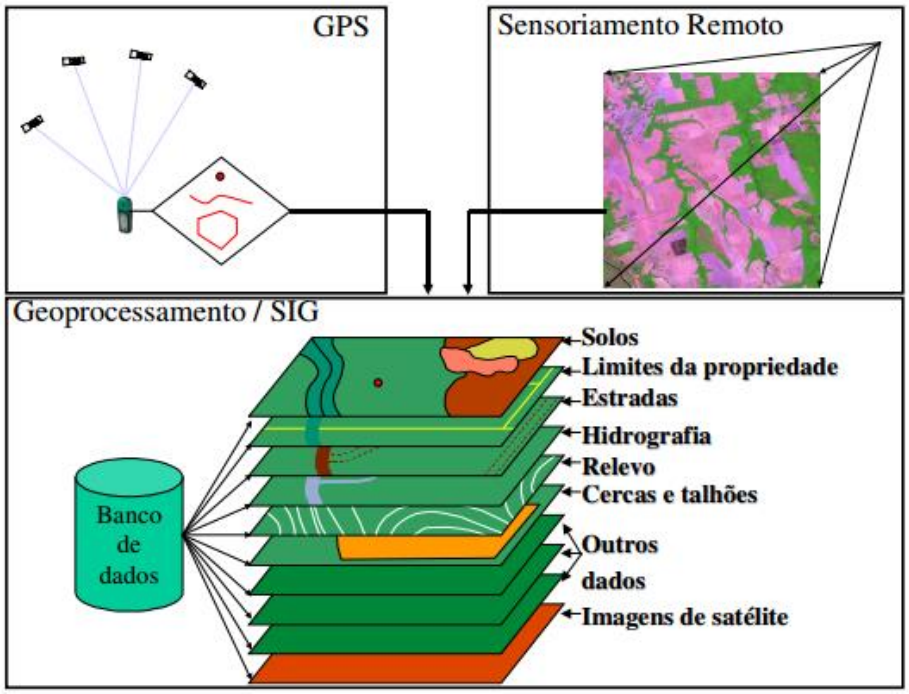

Figura 12 - *GPS*, sensoriamento remoto, geoprocessamento e SIG.
Fonte: Albuquerque e Santos (2003).

No ambiente SIG, o ambiente pode ser representado por meio de:

- Pontos: elementos pontuais como: uma árvore, uma ponte, um silo, um poste, um foco de incêndio etc;
- Linhas: elementos lineares como: um rio, uma estrada, uma linha de transmissão de energia, rede etc;
- Polígonos: áreas fechadas (linhas fechadas): propriedades, município, lago, uma área chuvosa e etc;
- Imagem: superfícies contínuas e variáveis no espaço: cobertura vegetal, solos, relevo contínuo etc;

Esses elementos são armazenados em arquivos vetoriais digitais.

O *GPS* é o instrumento que irá colher as informações de modo preciso, e, até o momento o mais eficiente para juntar dados especializados pontuais, lineares e poligonais. Por exemplo, para georreferenciar um curso de água, é necessário coletar a rota da mesma e, para tal, é necessária a utilização de *GPS*. Os dados colhidos através do *GPS* são transformados em vetoriais que compõem as diretrizes dos SIGs. Contudo, o *GPS* é uma ferramenta dos SIGs e não somente deles, também utilizado como meio de orientação de deslocamento de navios, aviões, automóveis e mesmo por pedestres (ALBUQUERQUE; SANTOS, 2003).

2.3.2 *GPS*

O *NAVSTAR GPS* (*NAVigation System with Time And Ranging* – Global Positioning System) ou seja, Sistema de navegação com tempo e variação – Sistema de Posicionamento Global, foi desenvolvido pelo Departamento de Defesa dos Estados unidos da América – DoD (*Department of Defense*) para atender as forças armadas no sistema de navegação. Em razão do alto nível de informações, e da precisão das mesmas, muitos outros países e comunidades aderiram utilizando receptores *GPS* (ALBUQUERQUE; SANTOS, 2003).

Com o sistema *GPS* implantado, qualquer usuário que possua um receptor, seja no celular, *tablet* ou qualquer dispositivo a seu dispor, tem no mínimo, 4 satélites para rastreamento, sob qualquer condição meteorológica.

Para que o sistema *GPS* exiba as coordenadas de um ponto na superfície terrestre é

relativamente simples: 3 satélites mais próximos ao ponto que tenha receptor calculam suas distâncias entre eles e entre a antena receptora; na geometria, três pontos que não pertencem ao mesmo plano são suficientes para determinar o posicionamento do ponto receptor, sendo necessário inserir um satélite, tornando assim 4 satélites, sendo que esse quarto faz a sincronia dos relógios, assim, tornando essa coordenada precisa com relação ao tempo, pois a terra está em rotação (ALBUQUERQUE; SANTOS, 2003). Assim, ilustra a Figura 13, retratando a constelação de satélites ao redor do planeta Terra.

Figura 13 - Constelação de satélites do sistema *GPS*.
Fonte: Site Garmin. Disponível em: <http://www.garmin.com/aboutGPS>. Acesso em nov. 2015.

2.3.3 Serviços SPS e PPS

O Sistema *GPS*, originalmente projetado para uso militar, foi liberado para utilização geral no ano de 1980, pelo presidente Ronald Reagan. Porém na época, por questões de segurança do país, sua acurácia era limitada através da adoção de recursos de *AS* (*Anti-spoofing*) e *SA* (*Selective Availability* – Disponibilidade Seletiva), sendo o primeiro um processo de criptografia do código de medir distância e o outro, manipulação de mensagens de navegação e frequência de relógios, resultando em erros de aproximadamente de 100 a 140 metros. Às 0 hora do dia 02

de maio de 2000 foi abolida a limitação da acurácia, melhorando-a em 10 vezes (ALBUQUERQUE; SANTOS, 2003).

Os serviços do *GPS* eram divididos em: *SPS* e *PPS*. O *SPS – Standard Positioning Service* – Serviço de Posicionamento Padrão, é um serviço disponível a todos usuários comuns e gratuito. O *PPS – Precise Positioning Service* – Serviço de Posicionamento Preciso, é um serviço de uso militar e usuários autorizados, pago.

2.3.4 Segmentos dos *GPS*

Albuquerque e Santos (2003), afirmam que o sistema *GPS* é dividido em 3 segmentos principais: o Segmento Espacial, o Segmento de Controle e o Segmento de Usuários, os quais são representados pela Figura 14 e serão explanados posteriormente.

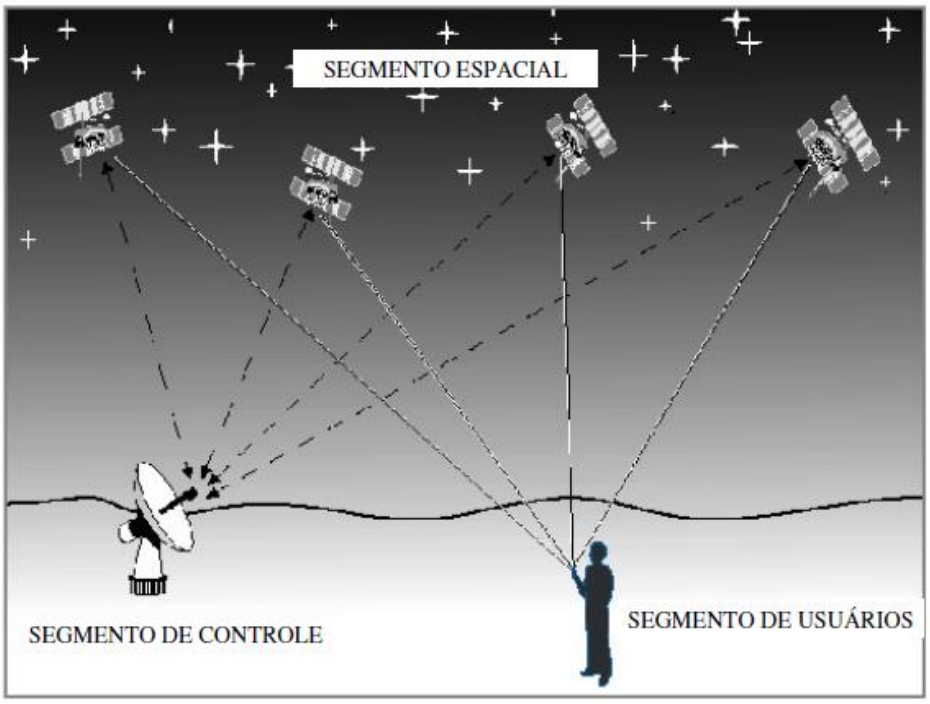

Figura 14 - Os segmentos do sistema *GPS*.
Fonte: Albuquerque e Santos (2003).

2.3.5 Segmento Espacial

São 24 (vinte e quatro) satélites ativos, em 6 (seis) órbitas elípticas, com planos

orbitais inclinados em 55º (cinquenta e cinco graus) em relação ao equador, altitude média de 20.200km, período orbital de 12 (doze) horas siderais (11:58 minutos do tempo solar), velocidade tangencial de 14.000k/h. Isto garante que 24 (vinte e quatro) horas por dia em qualquer lugar do planeta terra tenha-se a disposição 4 satélites, como representado pelas Figuras 15 e 16, a distribuição dos satélites e a distribuição orbital dos mesmos, respectivamente (ALBUQUERQUE; SANTOS, 2003).

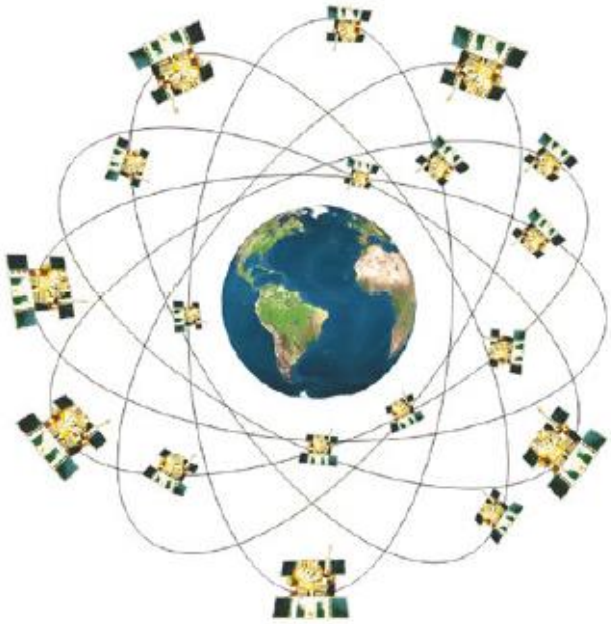

Figura 15 - Distribuição dos satélites.
Fonte: Albuquerque e Santos (2003).

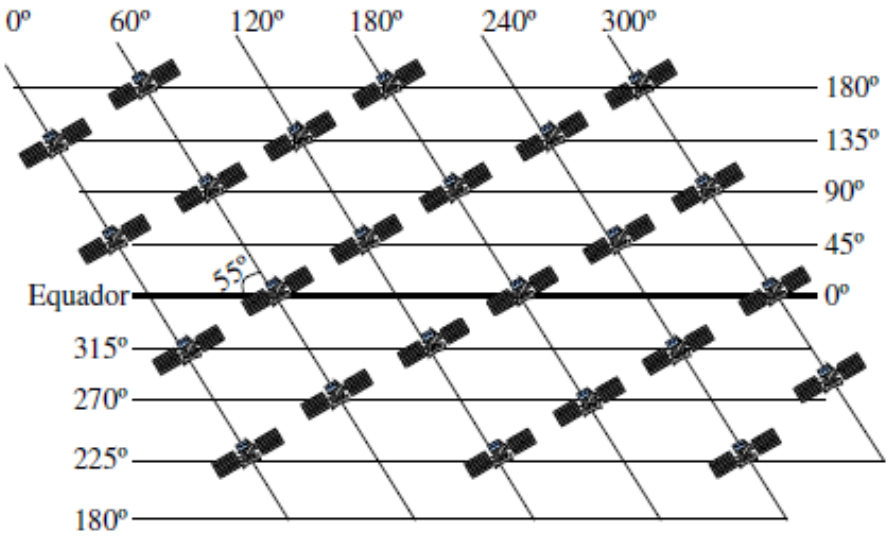

Figura 16 - Distribuição orbital dos satélites.
Fonte: Albuquerque e Santos (2003).

2.3.6 Códigos e mensagens transmitidas

O PRN (*Pseudo Random-Noise* – ruído falsamente aleatório) é uma sequência binária, gerado por um algoritmo, univocamente identificado. Junção dos códigos C/A (*Coarse Aquisition* – fácil aquisição) e P (*Precise* ou *Protected* – preciso ou protegido). C/A é parte de um grupo de códigos em que cada satélite possui o seu. De certa forma faz com que os receptores identifiquem cada satélite. A partir do C/A é possível ter as distâncias identificadas em dispositivos comuns, como celulares, sendo que seu sinal não é criptografado.

O código P é transmitido de forma criptografada sendo utilizado somente por usuários autorizados, sendo eles militares, pois dão o posicionamento preciso.

Além dos códigos, existem as mensagens de navegação, chamadas efemérides que, informam a posição do satélite a cada instante (ALBUQUERQUE; SANTOS, 2003).

2.3.7 Segmento de Controle

Esse segmento tem por objetivo monitorar e controlar o sistema de satélites. Segundo Albuquerque e Santos (2003) “determinar o tempo *GPS*; corrigir os relógios, predizer as efemérides e, atualizar as mensagens de navegação”.

Como observa-se na Figura 17, compreendendo-se que o sistema de controle é formado por: uma estação de controle central (*MCS* – *Master Control Station*), localizada em Colorado Springs; cinco estações monitoras (Hawaii, Colorado Springs, Ascension Island, Diego Garcia e Kwajalein) que pertencem a AAF (*American Air Force*) e sete estações do NIMA (*National Imagery and Mapping Agency*).

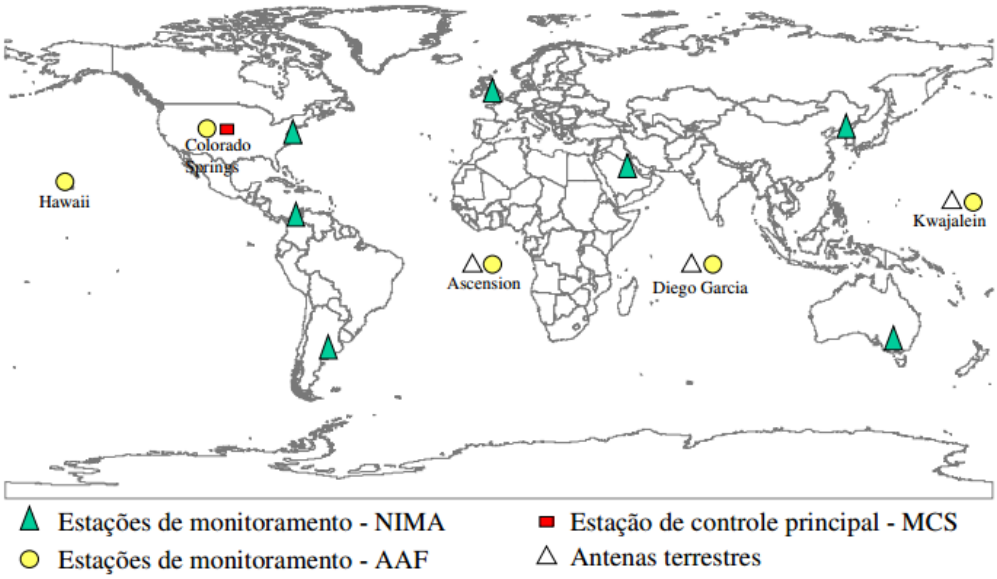

Figura 17 - Segmento de controle do *GPS*.
Fonte: Albuquerque e Santos (2003).

Segundo Albuquerque e Santos (2003), cada estação monitora rastreia todos os satélites visíveis e transmite os dados para a estação de controle principal via sistema de comunicação. Esses dados de controle determinam as órbitas, corrigem os relógios, e atualizam as mensagens de navegação. Essas efemérides de determinação de órbita são precisas em ordem de centímetros.

A cada 12 horas um satélite dá uma volta no planeta, sendo possível passar em algumas estações de monitoramento, através disso é possível controlar sua altitude, posição e velocidade. São encontradas alterações causadas por atração gravitacional, da Lua e do Sol e pressão da radiação solar.

2.3.8 Segmento de usuários

O segmento de usuários é constituído pelos receptores *GPS*, podendo ser dividido em usuários civis e militares (ALBUQUERQUE; SANTOS, 2003). Ele tem a utilidade mais variada do *GPS*, desde deslocamentos automobilísticos quanto a aplicações para descobertas e etc. Quanto aos usos militares são para manobras de combate e treinamento (mísseis, deslocamento de tropas e etc.).

2.3.9 Posicionamento

Albuquerque e Santos (2003) afirmam que posicionamento é o processo de posicionar-se em relação a um referencial específico, determinando a posição de um objeto/pessoa.

Para calcular o posicionamento de um receptor *GPS* na superfície da terra é necessário que se saiba as distâncias entre os satélites, o receptor e a posição de cada satélite no espaço cartesiano. Os satélites são como pontos referenciais para realização desse cálculo. Esse processo chama-se trilateração eletrônica: tendo um ponto de distância desconhecida e outros 3 pontos com posições conhecidas, é possível saber a posição do desconhecido.

2.3.10 Trilateração para satélites

Albuquerque e Santos (2003) utilizaram de um exemplo para ilustrar o processo de trilateração: sabe-se que um ponto desconhecido em uma superfície esférica tem a distância de 20.000km do satélite 1, como ilustra a Figura 18:

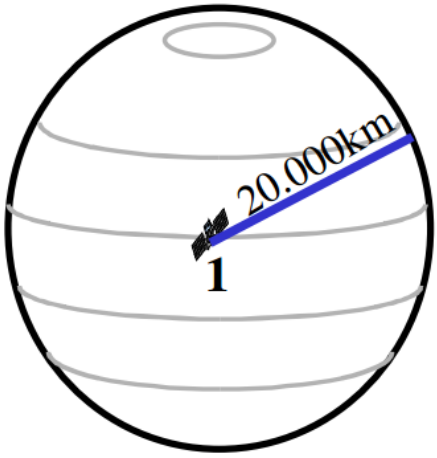

Figura 18 - Distância do ponto desconhecido do Satélite 1 na esfera.
Fonte: Albuquerque e Santos (2003).

Sabendo ainda que esse mesmo ponto desconhecido está a 21.000km do satélite 2. Cada satélite tem mais pontos dos quais possuem essas distâncias, ou seja, pontos que possuem 20.000km de distância do satélite 1 e 21.000km de distância do satélite 2. Com esses pontos possíveis forma-se um círculo possível do ponto

desconhecido na esfera, conforme a Figura 19:

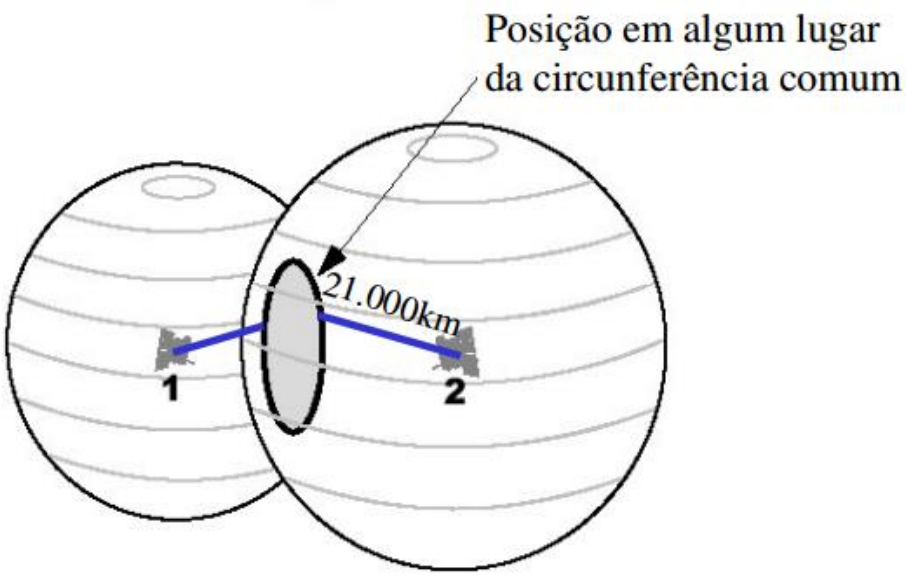

Figura 19 - Ponto desconhecido em uma área identificada pelos dois satélites e suas distâncias.
Fonte: Albuquerque e Santos (2003).

Sabendo ainda que a distância desse ponto desconhecido é de 22.000km de um terceiro satélite, reduz-se esse círculo a somente dois pontos possíveis que possuem as 3 distâncias iguais com relação aos satélites 1, 2 e 3. Porém, contudo, ainda não se sabe a posição do ponto desconhecido. A diferença que faz o reconhecimento do ponto correto é que um dos dois pontos está em um espaço impossível de se aceitar, ou seja, está a uma grande distância da superfície da esfera, fazendo com que o outro ponto seja aceito como o ponto desconhecido (ALBUQUERQUE; SANTOS, 2003), como ilustra a Figura 20.

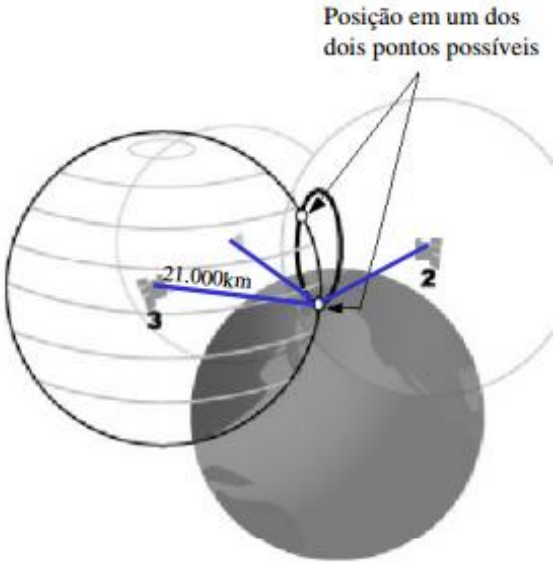

Figura 20 - Ponto desconhecido reduzido a dois pontos identificado pelo terceiro satélite.
Fonte: Albuquerque e Santos (2003).

Esses são os princípios utilizados pelos receptores *GPS*, de cálculo de suas posições.

2.3.11 Tempo na função *GPS*

Albuquerque e Santos (2003) afirmam que a sincronia dos relógios dos satélites é de extrema importância para cálculo de coordenadas precisas. Para tal os mesmos utilizam de padrões de frequência estáveis, baseados em pulsos atômicos do Césio e Rubídio. Os mais atuais poderão utilizar o MASER de hidrogênio (*Microwave Amplification by Stimulated Emission of Radition*) que é alto padrão em questão de frequência.

Os relógios utilizados para cálculos nos satélites possuem acuracidade em nanosegundos (10^{-9}), pois se o receptor e o satélite estiverem fora de sincronia por apenas 1/100 do segundo, representando uma diferença de 3.000km errados no cálculo da distância.

Para que não tenha essa diferença de cálculo, os satélites possuem relógios atômicos que são sintonizados de acordo com o relógio da Estação de Controle Central. Essa medida de tempo é uma precisão de ordem de 10^{-12} segundos, que são pulsos atômicos dos cristais radioativos Césio ou Rubídio. Os relógios custam em torno de 100mil dólares cada, sendo que cada satélite tem 4 (quatro) relógios,

assegurando o trabalho preciso.

2.4 Navegação Autônoma

Robô é uma construção eletrônica e mecânica que através de programação é capaz de cumprir e realizar tarefas. Dessa forma entende-se que eles são construídos para auxiliar o ser humano em suas tarefas diárias ou mesmo em tarefas com um certo nível de periculosidade. Daí surgem os tipos de dispositivos robóticos e uma de suas características tais como controle e autonomia (AZEVEDO *et al.*, 2015).

Com toda tecnologia que se tem e todas as descobertas realizadas pela ciência, hoje ainda há áreas que continuam intrigantes, sendo objeto de pesquisas constantes.

A propriedade “autonomia” é objeto de pesquisa e tema desse trabalho. Cazangi (2004), afirma que o significado de autonomia é a capacidade de governar-se pelos próprios meios. Sistemas biológicos são em geral autônomos, isto é, se mantem pelos seus próprios mecanismos: auto-organização.

Um objeto ser autônomo quer dizer que ele é automático, que é a capacidade de perceber um ambiente e impactá-lo visando o cumprimento de tarefas. Ele se autodirige com base na sua capacidade de cumprir uma tarefa.

Devido o fato de que qualquer ser autônomo depende do ambiente que o cerca que são dados níveis de autonomia, pois nem todo ser ou objeto é totalmente autônomo.

2.4.1 Histórico

Crestani Jr. (2001), explica que na década de 60 é que começou a era dos robôs móveis. Um dos membros dessa geração foi o robô Shakey, desenvolvido por em Stanford Research Institute. Shakey navegava em um ambiente com objetos demarcados, tomava decisões de navegações baseados em inferência simbólica e fazia leituras sensoriais para construir e manter o modelo interno do ambiente de navegação. Os primeiros robôs tinham métodos de raciocínio deliberativo, ou seja, construía uma representação do mundo em seu interior para assim, poder planejar como atingiria o objetivo proposto. Porém toda a base de código era cheia de regras,

contemplando todas as situações possíveis, isso gerava um sistema amplamente complexo e exigia muita capacidade de armazenamento de informações, sem garantia de sucesso na implementação.

A partir da década de 80, foi elaborada uma nova arquitetura em que há um estímulo, um comportamento, como consequência e logo, uma resposta à esse estímulo. Porém, as tarefas mais complexas podiam, no entanto, terem estímulos suprimidos, bem como as respostas também, pelo fato de ter vários estímulos e comportamentos a serem tomados.

As literaturas ainda identificam como sistemas híbridos aqueles que possuem características deliberativas e reativas desconsiderando inteligência computacional.

Em casos de sistemas de navegação autônomos, Crestani Jr. (2001, p.5) afirma “simulações servem como base para conclusões que levem a boas previsões do comportamento real”.

Essas referências são apenas para o embasamento teórico no contexto histórico da robótica, umas vez que inteligência artificial não faz parte desse trabalho.

2.4.2 Navegação autônoma de robôs

Cazangi (2004) explica que a navegação autônoma em robôs consiste em elaboração de sistemas de tomadas de decisão para execução de uma determinada tarefa, que envolvem a configuração do ambiente, como o robô foi construído, programação e critérios de desempenho.

O robô tem que conhecer a topologia do ambiente em que atuará. A navegação autônoma, bem como algumas fórmulas matemáticas de tomadas de decisões, prevê o uso de sensores, para conhecimento do território. Teoricamente o robô tem de navegar pelo ambiente sem auxílio externo, estruturando seus próximos passos baseados nos algoritmos impostos nele. A utilização de sensores ainda não é um fato no desenvolvimento desse trabalho, trata-se de implantação futura.

2.5 Linguagem Java

Na década de 90, a linguagem Java foi desenvolvida pela empresa *Sun Microsystems*. Descendente da linguagem de programação C, possui estrutura parecida. Linguagem de programação orientada a objetos que resolve muitos problemas, de forma estruturada, em menos tempo e com códigos de qualidade (CAELUM, 2015).

O nome Java veio de uma ilha do pacífico onde se produz um café especial que tornou-se popular entre os profissionais de T.I. (tecnologia da informação), sendo então, homenageado no nome da linguagem.

A linguagem Java possui uma máquina virtual (JVM) – *Java Virtual Machine*. Essa JVM permite que qualquer computador possa executar o programa criado em linguagem Java, ou seja, seu computador pode ter qualquer sistema operacional que executará da mesma forma, ilustrada na Figura 21.

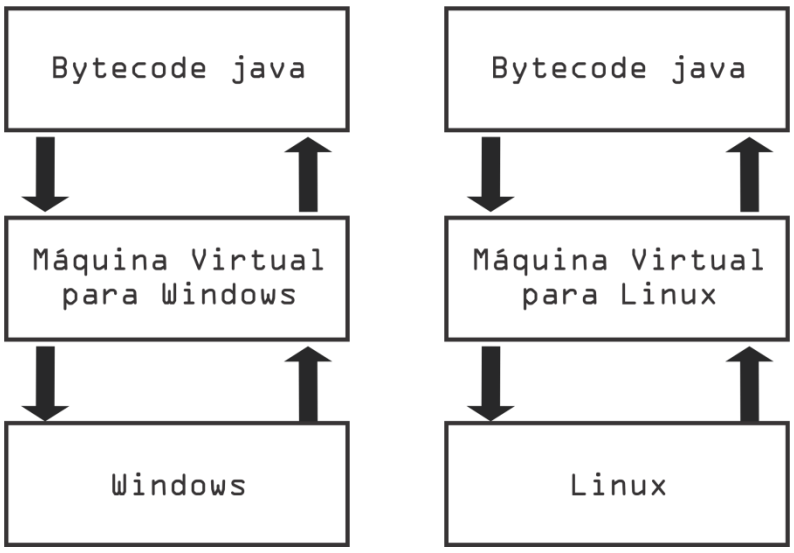

Figura 21 - Funcionamento de *software* em linguagem Java.
Fonte: Elaborado pelos autores.

A JVM não é só um interpretador, e sim responsável também por gerenciar memória, pilhas de execução, *threads* e etc. Ela faz com que sua aplicação seja executada sem que se envolva com o sistema operacional, explicando assim, sua portabilidade para qualquer computador ou dispositivo (CAELUM, 2015).

Originalmente foi criada para utilização em *browsers* móveis. Foi popularizando-se para dispositivos, tornando hoje a linguagem mais utilizada de desenvolvimento de

aplicativos para dispositivos móveis (MASSAGO; SCHÜTZER, 2015).

No Brasil a linguagem foi bem adotada tendo fóruns totalmente em português com mais de 100 mil usuários e mais de 1 milhão de mensagens nos fóruns.

Em 1994 a empresa tinha ideia de aplicativos para dispositivos móveis, televisões e etc., não sendo bem aceito na época, porém aos poucos foi introduzindo no mercado que pedia mais mobilidade e flexibilidade. Hoje o Java domina aplicações para celulares com mais de 2.5 bilhões de dispositivos compatíveis.

Por ser a linguagem mais flexível do mercado é que o S.O. Android utiliza dessa linguagem para desenvolvimento de suas aplicações, uma vez que é o objetivo principal da linguagem, o trabalho com dispositivos.

3. DESENVOLVIMENTO

Para realizar a simulação da comunicação entre *smartphone* e um protótipo robótico, foi desenvolvido um aplicativo baseado na linguagem de programação Java, na IDE *Android Studio*. O aplicativo utiliza os dados recebidos pelo *GPS* do *smartphone*, mostra seu ponto atual, e através disto, determina e aponta qual o próximo ponto no qual o protótipo robótico desenvolvido em arduino deverá perseguir, e mostra ao usuário a direção na tela do *smartphone* para chegar ao destino previamente determinado.

Na Figura 22, ocorre a explanação simbólica do esquema de comunicação e simulação da transmissão dos dados.

Figura 22 - Exemplo da comunicação dos satélites ao protótipo robótico.
Fonte: Elaborada pelos autores.

3.1 Ferramentas

Neste ítem serão descritas as ferramentas utilizadas para a construção do *software*, bem como suas aplicações e recursos. O código fonte será citado assimilando as funções em que foram utilizadas.

3.1.1 Android Studio

O Android Studio é a IDE oficial do S.O. Android para desenvolvimento de aplicações voltadas para este sistema operacional de dispositivos (ANDROID, 2015). Sendo baseado na tecnologia Intellij IDEA, esta IDE possui um suporte muito vasto para o usuário, desde assistente de código inteligente a debugger para emuladores e dispositivos reais (INTELLIJ, 2015). O Android Studio possui um suporte avançado para lidar com a comunicação da aplicação com o *GPS* e com *bluetooth* do dispositivo, tal suporte facilita o desenvolvimento de aplicações ao ter funções que já informam a distância entre dois pontos e as funções de socket e *bluetooth* para conexão entre dispositivos.

3.1.2 *Bluetooh*

Bluetooh é um recurso utilizado para transmissão de dados entre dispositivos. É necessário parear os dispositivos antes de iniciar a troca de dados. Um escaneameanto dos dispostivos disponíveis é realizado pelo *host* (aquele que irá ser o responsável por iniciar a conexão) e após encontrar o dispositivo com o qual deseja parear, é necessário requisitar a conexão, que normalmente deve ser aceita por ambos dispositivos. O *bluetooth* foi o meio de transmissão de dados escolhido para realizar a transmissão de dados entre o *smartphone* e microcontrolador. Esses dados enviados contêm informações sobre potência e inclinação em que os servo motores irão trabalhar para fazer com que o protótipo robótico ande.

Para realizar a conexão *bluetooth* entre *smartphone* e arduino, foi necessário criar uma variável responsável por receber os dados do adaptador bluetooth presente no smartphone, através da linha de código: BluetoothAdapter.getDefaultAdapter(). Dessa forma, o aplicativo pode listar os dispositivos já pareados pelo *smartphone*

anteriormente, com o código: myBluetooth.getBondedDevices(). Essa lista é apresentada numa tabela, como mostrado na Figura 23, onde o nome e endereço MAC dos dispositivos são exibidos para o usuário selecionar. Após escolhido o dispositivo correto, com o comando: list.add(bt.getName() + ”\n” + bt.getAddress()), é possível inserir o dispositivo na lista. A conexão é realizada através de um socket, sob o código: BluetoothSocket btSocket. Este comando cria o atributo socket btSocket, que após realizar a conexão, permite que posteriormente, sejam enviadas as informações para o dispositivo através da linha de código: btSocket.getOutputStream().write(“#T127T255”.toString().getBytes()).

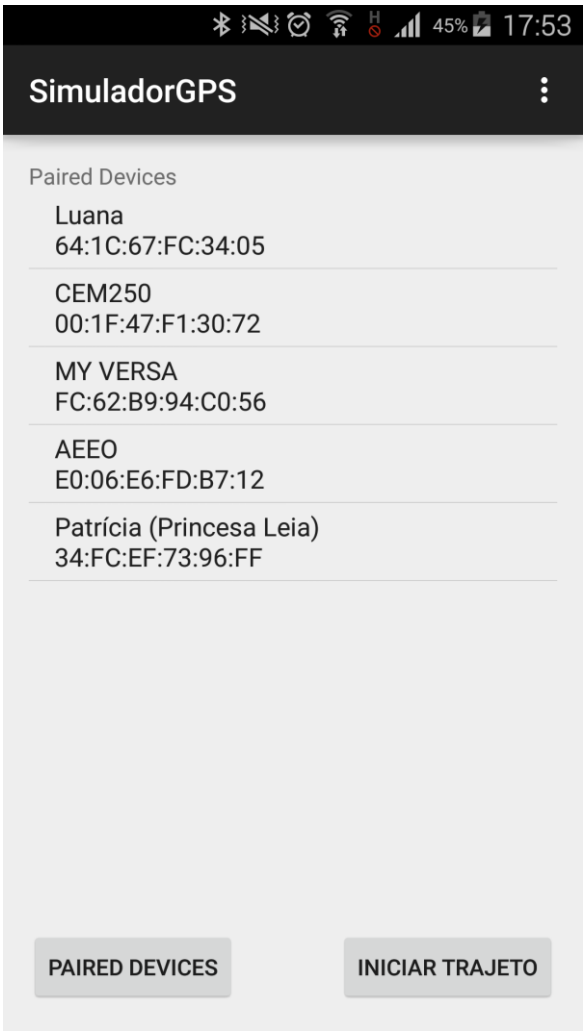

Figura 23 - Lista de dispositivos pareados.
Fonte: Elaborado pelos autores.

3.1.3 *GPS*

O *GPS* implantado no *smartphone* recebe os dados da sua posição atual e o aplicativo faz a comparação entre essa posição atual e a posição de destino. O valor retornado é um ângulo com valor entre 180º e -180º. Com esse grau de inclinação é possível indicar qual direção deverá tomar para se chegar ao seu destino. O aplicativo desenvolvido para simular a navegação informa qual a direção ele deve seguir, através de uma imagem na tela. No caso do protótipo robótico, o aplicativo enviaria via *bluetooth* qual a velocidade e rotação para cada um dos servos responsáveis pela direção do mesmo.

No aplicativo, a comparação entre a posição atual e o destino é obtida através de: resultBearingAtual = locat.bearingTo(destinoFinal), onde a variável resultBeatingAtual armazena o valor. A variável locat do tipo Location é recebida pelo metódo e armazena as informações do *GPS*. A função .bearingTo(destinoFinal) retorna o ângulo entre a posição atual do *GPS* e a variável destinoFinal, que também é do tipo Location e possui a latitude e longitude do destino. Este ângulo retornado possui um valor entre 180º e -180º, e assim chega-se ao final do cálculo mostrando em tela a direção a ser seguida.

3.1.4 Protótipo Arduino

O prótotipo arduino que está sendo desenvolvido pelo aluno Pedro Henrique Nakanishi, possui um módulo *bluetooth* na placa do microcontrolador, tornando possível a troca de informações entre um *smartphone* e o protótipo, que ao receber esses comandos os repassa para um motor *shield,* por exemplo. É necessario que eles possuam protocolos para recebimento e/ou envio de dados, esses protocolos irão enviar ou receber as informações.

O microcontrolador arduino, implantado no protótipo robótico, é o responsável por receber as informações enviadas e processadas pelo *smartphone* e transmiti-las para os servos que irão direcioná-lo. Esses dados recebidos tratam-se das velocidades de rotação para cada um dos dois servos localizados no eixo dianteiro do protótipo robótico. Essas informações são passadas através do seu *shield bluetooth*. Os valores passados podem variar de 0 a 255, tratando-se de sua

potência, parametrizados por protocolo de trabalho.

Para informar a velocidade de rotação dos servo motores, é necessário utilizar o seguinte protocolo, que é iniciado pelo simbolo cardial (#), seguido de um caractere para determinar a rotação do primeiro motor que é simbolizado com “F” para frente e “T” para trás, seguido de três números inteiros que são referentes ao valor de PWM, o mesmo deverá ser feito para o segundo motor e para finalizar o protocolo, é necessário inserir um asterisco (*). Exemplo: #F127F255*. Na Figura 24, é demonstrado como as informações de velocidade são processadas pelo microcontrolador.

```
//Atribuindo os valores de PWM
void atribuiE(String str){
 //Função separaEsq separa o valor de PWM para motor esquerdo
if(separaEsq(str) > 0){
 //caso direcao seja F = Ir para frente
  if (separaDirecaoE(str) == "F") {
    digitalWrite(mA1, LOW);
    analogWrite(mB1, separaEsq(str));
  }
  else if (separaDirecaoE(str) == "T") {
    analogWrite(mA1, separaEsq(str));
    digitalWrite(mB1, LOW);
  } else {
    Serial.println("Erro ao tentar identificar o sentido");
  }
//caso esteja recebendo número abaixo ou igual a 0
 }else{
  digitalWrite(mA1, LOW);
  analogWrite(mB1, 1);
 }
}
```

Figura 24 - Código para recebimento dos dados do aplicativo.
Fonte: Nakanishi (2015).

3.1.5 Servo Motor

Os servo motores ilustrados na Figura 25, possuem eixos integrados e um eixo que permite um controle preciso de seus movimentos. Os eixos dos servos motores mais comuns podem ser posicionados em diversos ângulos, normalmente variando entre 0 a 180 graus. A velocidade do seu eixo é determinada pela rotação dos servos (ARDUINO, 2015).

Os servos integrados às esteiras são os responsáveis pela direção e velocidade do mesmo. As informações de velocidade para cada um dos servos irá determinar para que lado o protótipo robótico irá virar. Essa informação é enviada pelo aplicativo no smartphone que foi previamente processada pela aplicação.

Figura 25 - Exemplo de servo motor integrado a uma roda.
Fonte: Site Arduinolandia. Disponível em: <http://www.arduinolandia.com.br/servo-motor-mg90s>. Acesso em nov. 2015.

3.2 O aplicativo, seu funcionamento e seu código fonte

Nesse item é explanado o aplicativo como um todo, sua aplicação, execução, funcionalidades e seu código fonte sendo comentado.

3.2.1 O aplicativo

Ao iniciar o aplicativo, será realizada uma verificação se o recurso de *bluetooth* do dispositivo está ligado, caso não esteja, será solicitado que o usuário o ative, conforme na Figura 26.

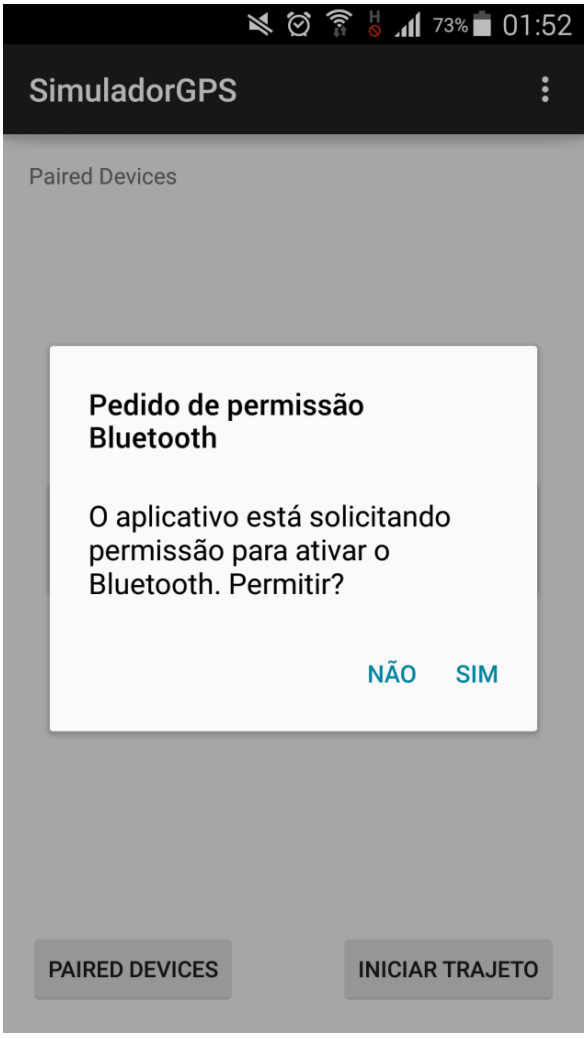

Figura 26 - Solicitação para ativar o *bluetooh*.
Fonte: Elaborado pelos autores.

A lista para seleção de dispositivos pareados pelo *bluetooth*, irá aparecer quando o usuário clicar no botão “*Paired Devices*”, como consta na Figura 23. Após selecionar o dispositivo, o usuário deve iniciar a navegação, clicando no botão “Obter localização”, na nova tela exibida, exemplificado na Figura 27.

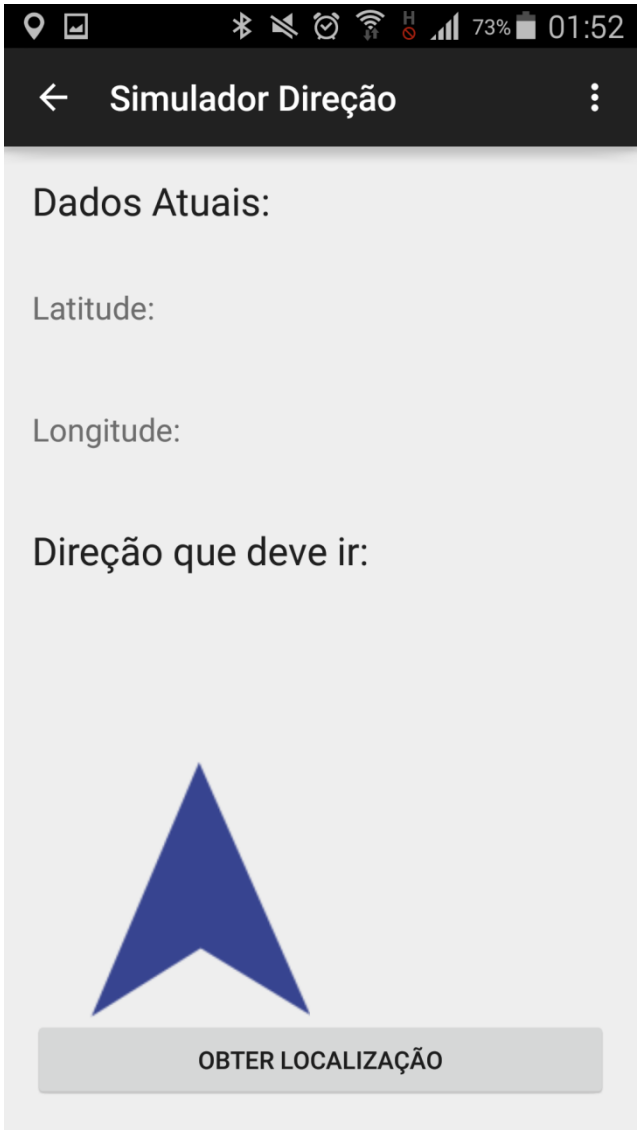

Figura 27 - Iniciar *GPS*.
Fonte: Elaborado pelos autores.

A captura dos dados do *GPS* será iniciada após a execução do método "*startGPS*", que começará a informar ao usuário, por meio de uma seta, qual será a direção que ele deverá seguir para chegar ao seu destino.

Os dados do *GPS* são tratados com o auxílio de uma variável do tipo *Location*, que é iniciada por uma *LocationListener*, responsável por verificar se houve alguma alteração nos dados do mesmo. O método *onLocationChanged* é executado a cada mudança de posição do *GPS*.

Para conseguir informar qual direção o usuário deve seguir, o aplicativo leva em

conta o passo anterior dado pelo usuário, verificando se a distância aumentou ou não e se houve algum desvio na direção que estava sendo seguida. Após essa verificação, é realizada uma rotação na seta se necessário corrigir o caminho do usuário. Para comparar tanto o ângulo de inclinação quanto a distância entre o ponto atual e o destino, foi necessário usar variáveis para armazenar os dados atuais e após utilizá-los, atribuir para variáveis responsáveis por armazenar o valor anterior.

Além disso, é necessário converter o valor recebido pelo comando *bearingTo(destinoFinal)* para um valor entre 0 e 360, como mostrado na Figura 28.

```
resultBearingAtual = locat.bearingTo(destinoFinal);
if (resultBearingAtual < 0) {//Caso o bearing seja um valor negativo, atualiza ele
    directionAtual = resultBearingAtual + 360; // conversao de norte verdadeiro (-180 --- 180)
}
```

Figura 28 - Conversão do ângulo retornado pelo *GPS*.
Fonte: Elaborado pelos autores.

Tal conversão é feita utilizando a seguinte condição: se o valor retornado pelo *bearingTo* for menor que 0, será adicionado a uma nova variável, o valor do ângulo retornado pelo *bearingTo* adicionado o valor 360. Essa conversão transforma, por exemplo, um ângulo -135 para o valor 225.

O protocolo para envio desses dados via *bluetooth* entre qualquer outro dispositivo devidamente preparado para recebe-los, está exemplificado no código, mas não está ativo pois o protótipo robótico não estava em estágio de finalização para os testes de conexão.

3.2.2 Funcionalidades

As funcionalidades de um aplicativo são tarefas que ele pode cumprir facilmente, facilitando a vida do usuário. No caso desse aplicativo, suas funcionalidades consistem em:

- Aplicativo de fácil utilização pelo usuário, calculando a rota em segundo plano, fazendo com que o usuário apenas ajuste em um único botão;

- Possibilidade de conexão a outro dispositivo via *bluetooth*, devidamente parametrizado conforme ilustra esse projeto, como por exemplo o protótipo robótico construído em microcontrolador arduino pelo Sr. Pedro Henrique Nakanishi;
- Processamento e envio de dados para movimentação de motores ligados a placa microcontroladora arduino;
- Cálculo de rota através de ângulos polares, independente da posição do smartphone;
- Mostra em tela a direção em que o dispositivo conectado via *bluetooh* vai seguir.

3.2.3 Requisitos do aplicativo

Requisitos são condições em que o sistema estabelece para que determinada funcionalidade seja executada, ou determinado objetivo seja atingido.

Os requisitos funcionais deste aplicativo são:

a) O aplicativo deve cadastrar uma rota definida pelo usuário (entrada);
b) O aplicativo deve processar a rota, definindo o próximo ponto em que o usuário deverá ir para atingir o destino final, ou mesmo um protótipo robótico no caso de utilização do microcontrolador arduino;
c) O aplicativo deve possuir um protocolo para envio de dados de velocidade, potência e direção para um protótipo robótico previamente configurado para interpretar esses dados;
d) O aplicativo deve demonstrar em tela a direção que deverá ser percorrida (saída);
e) O aplicativo deve proporcionar comunicação via *bluetooh*;
f) O aplicativo deve proporcionar opção de alteração de rota pelo usuário;

No entanto, no desenvolvimento, não foi possível atingir todos os requisitos pela complexidade do aplicativo. Os requisitos não atingidos foram “**a**” e “**f**” pelo fato da não implantação da API do Google Maps, que permite utilização dos produtos da Google para desenvolvimento de aplicações, no caso utilização dos serviços de

mapeamento. Para contemplar a funcionalidade do projeto, optou-se por ter uma rota definida em código fonte, para teste do mesmo.

No entanto ainda há os requisitos não funcionais que são propriedades temporais e espaciais, sendo elas:

a) O aplicativo deverá ser implementado em linguagem Java;
b) O aplicativo deverá processar a rota em até 60 (sessenta) segundos;
c) O aplicativo deverá apontar para o destino em tela para que o usuário acompanhe e demonstre que seus cálculos de rota estão corretos ou não.

Sendo que todos os requisitos não funcionais foram contemplados nesse projeto.

3.2.4 Métodos testados

Diversos meios para calcular qual seria o próximo passo do usuário foram testados, dentre eles, um conjunto de condições lógicas "*if* e *else*", para verificar se houve um aumento na latitude e longitude atual utilizando a latitude e longitude anterior. Após a verificação desses dados, era necessário informar ao usuário que ele deveria virar a esquerda ou a direita, para corrigir a sua rota, conforme o destino inserido. Este método apesar de ser simples, é eficaz, mas era necessário utilizar muitas linhas de código para ter uma precisão maior. Outra alternativa testada, foi usar uma variável do tipo "*GeomagneticField*", que é responsável por pegar a posição do usuário com relação ao norte verdadeiro. Através do comando "*getDeclination*" é possível armazenar o ângulo de inclinação entre o usuário e o norte verdadeiro. Apesar de sua precisão, infelizmente para uma utilização móvel, em que o usuário precisa se deslocar até seu destino, a posição do usuário em relação ao norte verdadeiro não auxilia tanto para conseguir dizer se o usuário está indo no caminho certo ou não.

3.2.5 Alguns pontos importantes sobre o código e suas funções

A determinação de qual será o caminho que o usuário deverá seguir para chegar ao seu destino previamente informado, se dá pelo aplicativo. Ele calcula o ângulo de

inclinação entre o ponto atual do *smartphone* e o ponto de destino, essa informação é retornada pelo *GPS* do mesmo. Esse ângulo não leva em consideração a orientação do aparelho, apenas qual a inclinação que deverá ser tomada pelo usuário para chegar ao seu destino pelo caminho mais curto possível. A cada segundo é informado um novo valor pelo *GPS*, com isso, sendo possível determinar se o usuario está se aproximando ou não do seu destino.

O cálculo para estimar o ângulo é realizado através da conversão das coordenadas geográficas informadas pelo *GPS* para coordenadas polares. Este cálculo pode ser realizado convertendo as coordenadas geográficas para distância e posteriormente descobrindo o valor do ângulo (θ) entre os dois pontos, em que “0” representa a posição atual do usuario e P o destino. O cálculo é realizado com as seguintes funções representadas pela Figura 29:

$$
\begin{aligned}
x &= r \cdot \cos\theta \\
y &= r \cdot \sin\theta \\
r &= \sqrt{x^2 + y^2} \\
\theta &= \arctan\left(\frac{y}{x}\right)
\end{aligned}
$$

Figura 29 - Funções para descobrir o valor de θ.
Fonte: Site Wikipédia. Disponível em: <https://pt.wikipedia.org/wiki/Coordenadas_polares>. Acesso em nov. 2015.

Com o valor de θ, o aplicativo informa para o usuário, no caso da simulação, qual direção ele deve seguir para chegar ao seu destino, tal informação é passada através setas que irão direcioná-lo. No caso do protótipo robótico em si, deverão ser passadas informações sobre potência e direção em que os servo-motores deverão girar.

A Figura 30 que demonstra o funcionamento do aplicativo, simulando as informações que o protótipo robótico receberá:

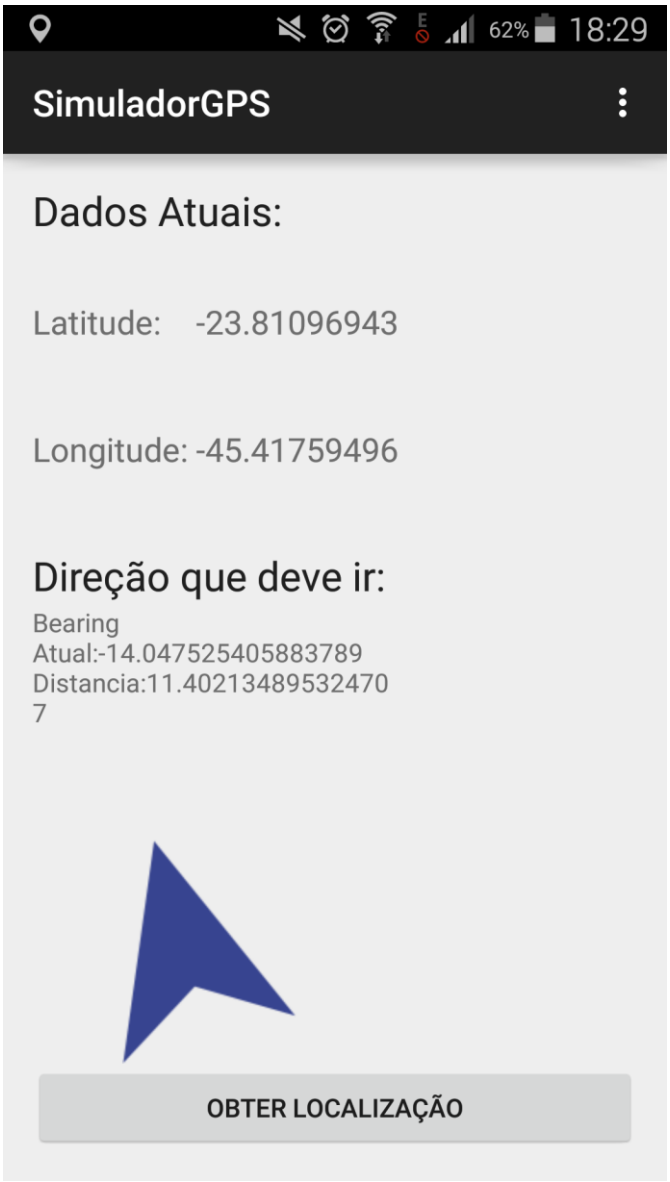

Figura 30 - *Software* detectando a posição atual e indicando a próxima direção em que o protótipo robótico deverá ir.
Fonte: Elaborada pelos autores.

3.2.6 A utilização para o usuário

Após iniciar o aplicativo, o usuário irá receber deverá dar um passo para atualizar as informações com esse seu primeiro passo. O aplicativo irá avaliar se houve um acréscimo na distancia e a partir disso, juntamente com o ângulo retornado pelo GPS, irá informar a direção que o usuário deverá seguir, conforme demonstrado na Figura 31.

Figura 31 - Direção informada pelo aplicativo.
Fonte: Elaborado pelos autores.

Após seguir a direção e se distanciar um pouco uma nova direção foi informada, dessa vez, solicitando que o usuário seguisse o caminho inverso, conforme a Figura 32.

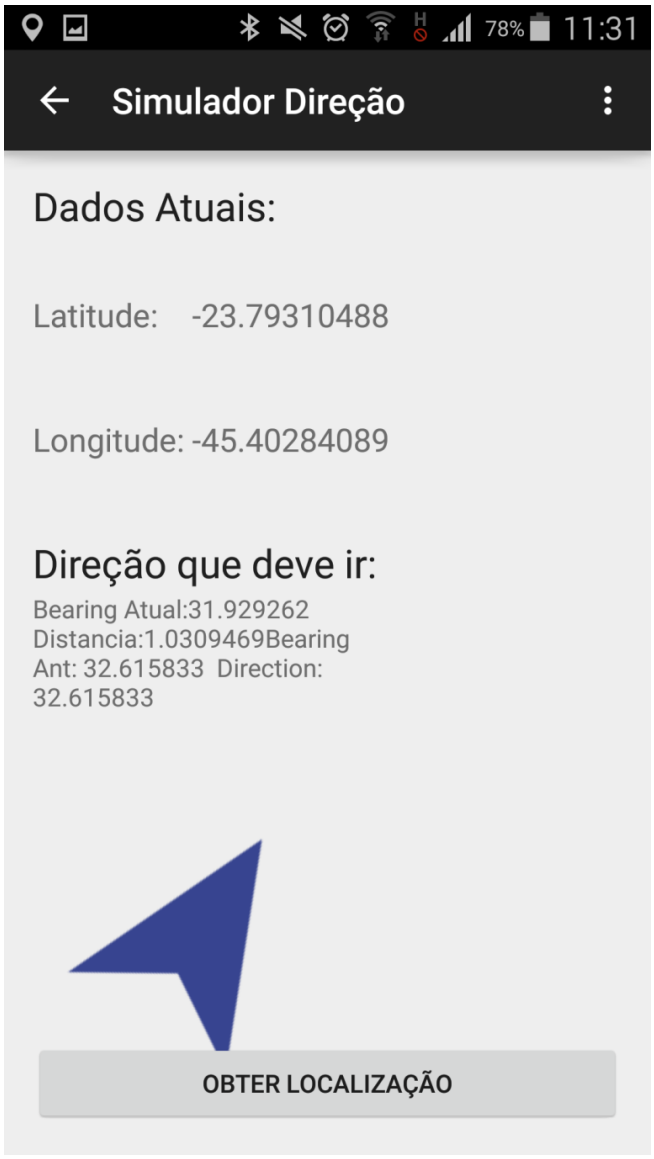

Figura 32 - Correção da rota após desvio do usuário.
Fonte: Elaborado pelos autores.

3.2.7 Metodologia de Desenvolvimento: Scrum

Esse trabalho foi desenvolvido através da metodologia ágil *scrum*. Nessa metodologia de gestão e planejamento de projetos de software a interação entre os indíviduos é maior, software entregue em módulos em funcionamento, a colaboração com o cliente é maior, e a resposta a mudanças é rápida. Essa metodologia foi criada em 2001 (TELES, 2008). No *scrum* os projetos são divididos em ciclos, mensais ou semanais. Esses ciclos ou iterações são chamados *sprint*. Dentro dos *sprints* contém um conjunto de atividades que serão executadas. Mais alguns

conceitos serão abaixo descritos:

- *Product Backlog*: lista de funcionalidades a serem implementadas em um projeto;
- *Sprint Planning Meeting*: reunião de planejamento para levantamento do *product backlog* e as prioridades de desenvolvimento;
- *Product Owner*: pessoa que prioriza e define os itens do *product backlog*;
- *Sprint Backlog*: compromisso do *scrum team* em forma de lista de tarefas extraídas do *product backlog*;
- *Daily Scrum*: reunião breve, geralmente de manhã entre a equipe, geralmente de pé, durando cerca de 15 minutos;
- *Sprint Review Meeting*: reunião de apresentação das funcionalidades implementadas;
- *Sprint Retrospective*: reunião após o final de um *sprint* revisando boas implementações e as que podem ser melhoradas;

Para colocar as tarefas, prazos e quem as executaria utiliza-se a ferramenta online Trello, disponível em http://www.trello.com, através do cadastro no site, tem-se acesso às facilidades da ferramenta. Consiste em criar um quadro e colocar os “cartões”. Cada cartão tem a tarefa a ser executada, a pessoa ou pessoas relacionadas e a data limite conforme descrito na Figura 33 relacionada e detalhada na Figura 34, respectivamente.

Figura 33 - Imagem geral do Trello.
Fonte: Elaborado pelos autores.

Figura 34 - Detalhamento da imagem geral com os cartões das atividades, prazos e integrantes que irão realiza-las.
Fonte: Elaborado pelos autores.

As reuniões *sprint planning meeting* aconteciam via “Skype” (aplicativo para comunicação via vídeo e voz) quinzenalmente para planejamento das ações, as funcionalidades *product backlog* foram colocadas no Trello para cumprimento da equipe .

As reuniões do tipo *daily scrum*, aconteciam via “Whatsapp” (aplicativo de mensagens instantâneas para *smartphone*), na qual a equipe determinava rapidamente as tarefas diárias.

Semanalmente aconteciam as reuniões *sprint review meeting*, sendo essas presenciais ou mesmo via Skype, com o orientador desse trabalho para apresentação das funcionalidades implementadas no projeto e implementação de novas metas ou funcionalidades.

4. CONCLUSÕES E TRABALHOS FUTUROS

Através dos capítulos desse trabalho, é perceptível a complexidade em que envolve este aplicativo. Esse aplicativo é capaz de realizar a comunicação entre um *smartphone* e um microcontrolador arduino através de um módulo *bluetooth*. Não foram realizados testes com um microcontrolador, mas o protocolo de comunicação utilizado como mostrado anteriormente, foi fornecido pelo aluno Pedro Henrique, que está desenvolvendo o protótipo de robótico utilizando arduino. Tal protocolo permite que qualquer dispositivo arduino que possua um módulo *bluetooth* e esteja seguindo o mesmo, possa receber os dados do aplicativo e utiliza-los para enviar a servo motores ou para qualquer outro sensor que possa se beneficiar desses dados. Um bom exemplo de implementação, seria integrar o aplicativo a um microcontrolador arduino implantado em uma embarcação pesqueira de pequeno porte, sendo responsável por controlar a direção do leme através dos dados enviados pelo aplicativo. O arduino seria o responsável por modificar a direção em que o leme está apontando, com isso mudando a direção da embarcação para a solicitada pelo aplicativo. Tal implementação auxiliaria os barcos pesqueiros, em momentos de dificuldades do usuário guiar a embarcação manualmente, seja por problemas climáticos ou qualquer outro problema que venha impossibilitá-lo de guiar a embarcação.

O desafio de desenvolver um aplicativo para uma plataforma que não se está habituado foi simplificado ao utilizar uma IDE com uma comunidade tão grande quanto a do Android Studio, bem como a do Arduino.

O aplicativo funciona corretamente podendo indicar o próximo passo em que o eventual protótipo robótico seguiria, em simulações realizadas andando pelo quarteirão através da tela do software e setas indicativas. A comprovação é suficiente para concluir que o protótipo robótico chegaria ao destino final sem necessidade de intervenção humana, uma vez que será capaz de movimentar-se utilizando funções que captam a posição atual e a comparam com o destino final

inserido pelo usuário do *smartphone*. A amostragem em tela comprova seu pleno funcionamento pelo fato de que o protótipo robótico somente recebe as informações e as traduz em forma de velocidade e rotação para os servos motores, sendo que o processamento total é realizado no *smartphone* e não no microcontrolador.

Com isto é possível comprovar que com os devidos ajustes em um microcontrolador arduino, é possível realizar a comunicação entre ele e o aplicativo desenvolvido, cabendo ao desenvolvedor do microcontrolador configurar como serão utilizados os dados enviados pelo aplicativo. Na Figura 35 tem-se a explicação esquemática:

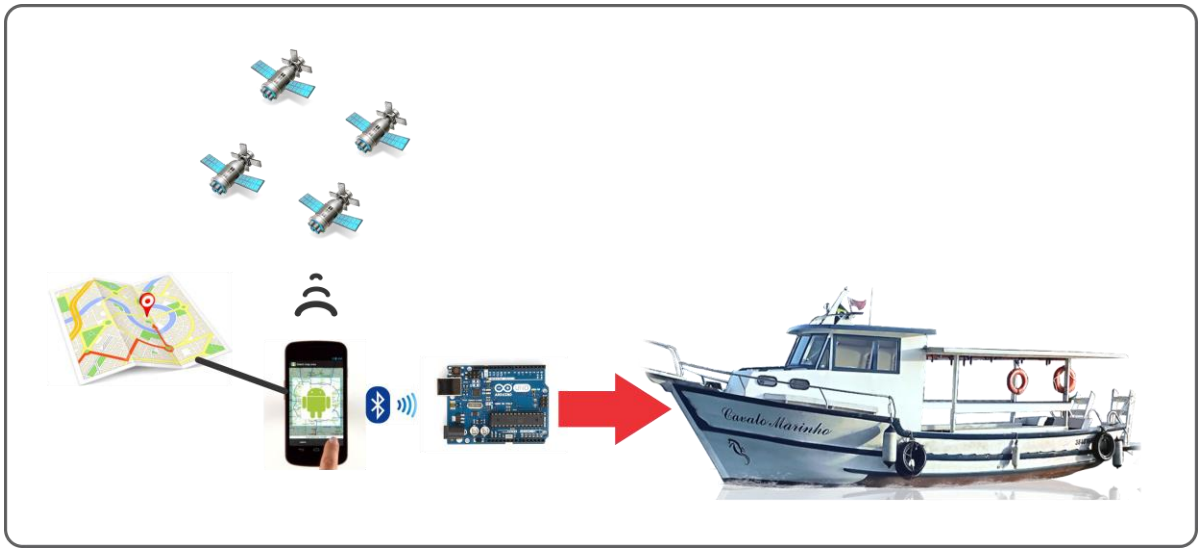

Figura 35 - Funcionamento do sistema aplicado em embarcações pesqueiras.
Fonte: Elaborado pelos autores.

Contudo ainda há muito que se fazer para refinar o projeto como um todo, adicionar sensores que evitem colisões em terrenos com obstáculos, refinamento de código para correções de erros de rotas e adicionar ainda funcionalidades de reconhecimento territorial tornando suas aplicações inúmeras e não somente nos navios pesqueiros.

Com isto coloca-se como trabalhos futuros seguindo as pesquisas com VANTs – Veículos Aéreos Não Tripulados bem como, robôs de mapeamento, e de cunho de descobertas em locais de alta periculosidade, locais inabitados e marítimos.

5. REFERÊNCIAS

ALBUQUERQUE, Paulo C. G.; SANTOS, Cláudia C. dos. M**ini Curso *GPS* para iniciantes**. In: SIMPÓSIO BRASILEIRO DE SENSORIAMENTO, 11. 2003, Belo Horizonte. Disponível em: <http://mtc-m12.sid.inpe.br/col/sid.inpe.br/jeferson/2003/06.02.09.16/doc/publicacao.pdf>. Acesso em: 10 nov. 2015.

ARDUINO. **Arduino Uno**, 2015. Foto, color. Disponível em: <https://www.arduino.cc/en/Guide/Windows>. Acesso em: 05 nov. 2015.

ARDUINO. **Site da Empresa**. 2015. Disponível em: <https://www.arduino.cc/>. Acesso em: 01 nov. 2015.

ARDUINO. **Site da Empresa**. 2015. Disponível em: <https://www.arduino.cc/en/Tutorial/PWM>. Acesso em: 01 nov. 2015.

Arduinolandia. **Site da Empresa**, 2015. Foto, color. Disponível em: <http://www.arduinolandia.com.br/servo-motor-mg90>. Acesso em: 22 nov. 2015.

AZEVEDO, S.; AKYNARA, A.; PITTA, R. **Minicurso: Introdução a robótica educacional**. *Sociedade Brasileira para o Progresso da Ciência*, São Paulo, 2015. Disponível em: <http://www.sbpcnet.org.br/livro/62ra/minicursos/MC%20Samuel%20Azevedo.pdf>. Acesso em 20 nov. 2015.

Bluetooh. **Site da Empresa**. 2015. Disponível em: <http://www.bluetooth.com/what-is-bluetooth-technology/bluetooth>. Acesso em: 22 nov. 2015.

Caelum. **Site da Empresa**, 2015. Disponível em: <https://www.caelum.com.br/apostila-java-orientacao-objetos/o-que-e-java/#2-1-java>. Acesso em: 22 nov. 2015.

CAZANGI, Renato R. **Uma proposta evolutiva para controle inteligente em navegação autônoma de robôs**. Campinas, UNICAMP, 2004.

CPTEC/INPE. **Litoral Norte do Estado de São Paulo**, 2015. Mapa, color. Disponível em: <http://nridaln.cptec.inpe.br/>. Acesso em: 01 nov. 2015.

DATASHEET. **Arquitetura do Microcontrolador Atmel**, 2011. Imagem, color. Disponível em: <http://dqsoft.blogspot.com.br/2011/07/microcontrolador-atmel-atmega328-parte.html>. Acesso em: 05 nov. 2015.

DIEGUES, Antonio Carlos (Org.). **Os saberes tradicionais e a biodiversidade no Brasil**. São Paulo: MMA/COBIO/NUPAUB/USP, 2000. 211 p.

DIMARZIO, Jerome J. F. **Android:** a programmer's guide. United States of Americ, McGraw-Hill Companies, 2008.

Garmin. **Site da Empresa**, 2015. Imagem, color. Disponível em: <http://www.garmin.com/aboutGPS/>. Acesso em: 10 nov. 2015.

Google. **Site da Empresa**. 2015. Disponível em: <http://developer.android.com/>. Acesso em: 05 nov. 2015.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATISTICA. **Base de Dados**. 2011. Disponível em: <http://www.ibge.gov.br/home/estatistica/populacao/estimativa2011/estimativa.shtm> . Acesso em: 01 nov. 2015.

Instituto de Pesca do Estado de São Paulo. PROGRAMA DE MONITORAMENTO DA ATIVIDADE PESQUEIRA MARINHA E ESTUARINA. **Base de Dados**. 2015. Disponível em: <http://www.propesq.pesca.sp.gov.br/>. Acesso em: 01 nov. 2015.

Intellij. **Site da Empresa**. 2015. Disponível em: <https://www.jetbrains.com/idea/features/android.html>. Acesso em: 22 nov. 2015.

JR. CRESTANI, Paulo R. **Sistemas inteligentes de navegação autônoma:** uma abordagem modular e hierárquica com novos mecanismos de memória e aprendizagem. Campinas, UNICAMP, 2001.

LECHETA, Ricardo R. **Android:** aprenda a criar aplicações para dispositivos móveis com o android SDK. 3ª edição, São Paulo, Editora Novatec, 2013.

MASSAGO, Sadao; SCHÜTZER, Waldeck. **UFSCar**, 2015. Disponível em: <http://www.dm.ufscar.br/profs/waldeck/curso/java/>. Acesso em: 22 nov. 2015.

MCROBERTS, Michael. **Arduino básico**. 1ª edição, São Paulo, Editora Novatec, 2011.

NAKANISHI, Pedro H. **Protótipo robótico para projetos de navegação autônoma**. Caraguatatuba, IFSP, 2015. 15 p.

Open Handset Alliance. **Site da Empresa**. 2015. Disponível em: <http://www.openhandsetalliance.com/>. Acesso em: 05 nov. 2015.

QUADROS, Daniel. **Microcontrolador Atmel ATmega328:** parte 1, 2011. Disponível em: <http://dqsoft.blogspot.com.br/2011/07/microcontrolador-atmel-atmega328-parte.html>. Acesso em: 05 nov. 2015.

SANTOS, Nuno Pessanha. **Arduino:** introdução e recursos avançados. Escola Naval, Departamento de Engenheiros Navais, 2009.

SOUZA, Fabio. Arduino – entradas e saídas, 2013. Imagem, color. Disponível em: <http://www.embarcados.com.br/arduino-entradasaidas-digitais/>. Acesso em 21 fev. 2015.

SOUZA, Fabio. **Arduino** – entradas e saídas, 2013. Embarcados. Disponível em: <http://www.embarcados.com.br/arduino-entradasaidas-digitais/>. Acesso em 21 fev. 2015.

TELES, Vinícius M. **Manifesto Ágil**, 2008. Disponível em: < http://www.desenvolvimentoagil.com.br/xp/manifesto_agil>. Acesso em: 22 nov. 2015.

WIKIPEDIA. **Arduino com protoboard e shields**, 2015. Foto, color. Disponível em: <https://upload.wikimedia.org/wikipedia/commons/d/df/Arduino_Protoboard_Shields.jpg>. Acesso em: 01 nov. 2015.

WIKIPEDIA. **Funções das relações das coordenadas**, 2015. Imagem, preto e branco. Disponível em: < https://pt.wikipedia.org/wiki/Coordenadas_polares>. Acesso em: 10 nov. 2015.

6. APÊNDICE A – CÓDIGO FONTE DO APLICATIVO

Classe: “AndroidManifest.xml"

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.thiago.simuladorgps" >

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
    <uses-permission android:name="android.permission.BLUETOOTH"/>


    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".deviceList"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
        <activity
            android:name=".gpsControl"
            android:label="Simulador Direção"
            android:parentActivityName=".deviceList" >
        </activity>
    </application>

</manifest>
```

Classe: “deviceList.java”

```
package com.example.thiago.simuladorgps;

import android.content.Intent;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.Set;


public class deviceList extends ActionBarActivity {


    Button btPaired;
    ListView devicelist;

    private BluetoothAdapter myBluetooth = null;
    private Set<BluetoothDevice> pairedDevices;
    public static String EXTRA_ADDRESS = "device_address";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.device_list);

        //puxando dados da classe R
        btPaired = (Button) findViewById(R.id.button);
        devicelist = (ListView) findViewById(R.id.listView);

        //Verifica se o dispositivo possui bluetooth e pega seus valores
        myBluetooth = BluetoothAdapter.getDefaultAdapter();

        if (myBluetooth == null) {

            Toast.makeText(getApplicationContext(), "Bluetooth Device Not
Available", Toast.LENGTH_LONG).show();


            finish();
        } else if (!myBluetooth.isEnabled()) {
            //Solicita ao usuário que ligue seu bluetooth, caso não esteja
ligado.
            Intent turnBTon = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(turnBTon, 1);
        }
```

```
        btPaired.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                pairedDevicesList();
            }
        });



    }

    private void pairedDevicesList() {
        pairedDevices = myBluetooth.getBondedDevices();
        ArrayList list = new ArrayList();

        if (pairedDevices.size() > 0) {
            for (BluetoothDevice bt : pairedDevices) {
                list.add(bt.getName() + "\n" + bt.getAddress()); //Nome e
Endereço do dispositivo
            }
        } else {
            Toast.makeText(getApplicationContext(), "No Paired Bluetooth
Devices Found.", Toast.LENGTH_LONG).show();
        }

        final ArrayAdapter adapter = new ArrayAdapter(this,
android.R.layout.simple_list_item_1, list);
        devicelist.setAdapter(adapter);
        devicelist.setOnItemClickListener(myListClickListener); //Metodo
acionado quando o usuário clica em algum dispositivo pareado da lista.

    }

    private AdapterView.OnItemClickListener myListClickListener = new
AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView<?> av, View v, int arg2, long
arg3) {
            // Pega o endereço MAC do bluetooth
            String info = ((TextView) v).getText().toString();
            String address = info.substring(info.length() - 17);

            // Intent criada para iniciar a próxima activity.
            Intent i = new Intent(deviceList.this, gpsControl.class);

            //Troca a activity.
            i.putExtra(EXTRA_ADDRESS, address);

            startActivity(i);
        }
    };


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        getMenuInflater().inflate(R.menu.menu_activity_list, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
```

```
        int id = item.getItemId();

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

        return super.onOptionsItemSelected(item);
    }
}
```

Classe: “gpsControl.java”

```
package com.example.thiago.simuladorgps;

import android.content.Context;
import android.content.Intent;
import android.hardware.GeomagneticField;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;

public class gpsControl extends AppCompatActivity {

    private TextView tvLatitude;
    private TextView tvLongitude;
    private TextView tvLatitudeAtual;
    private TextView tvLongitudeAtual;
    private TextView tvDifLat;
    private TextView tvDifLong;
    private TextView tvDirecao;
    private Button btLocalizar;
    private ImageView imgView;

    GeomagneticField geoField;
    private Double latInicial;
    private Double longInicial;
    private Double difLat;
    private Double difLong;
    private Double difLatAtual;
    private Double difLongAtual;
    private Double latFinal = -23.81081938;
    private Double longFinal = -45.41759083;
    private Double latAtual;
    private Double longAtual;
    private Double resultLat;
    private Double resultLong;
    private float heading;

    float resultBearingAnt;
    float resultBearingAtual;
    float resultDistance;
    float resultDistanceAnt;
    float direction;
    float directionAnt;
    float directionAtual;
    float difDirection;

    public LocationListener lListener;
    Location destinoFinal = new Location("");//provider name is unecessary
```

```
    public boolean comecouAMover = false;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_gps_control);
        setupElements();
        startGPS();
    }

    @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_main, 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);
    }


    // Importando os elementos da classe R
    public void setupElements(){
        tvLatitude = (TextView) findViewById(R.id.tvLatitude);
        tvLongitude = (TextView) findViewById(R.id.tvLongitude);
        tvLatitudeAtual = (TextView) findViewById(R.id.tvLatitudeAtual);
        tvLongitudeAtual = (TextView) findViewById(R.id.tvLongitudeAtual);
        tvDirecao = (TextView) findViewById(R.id.tvDirecao);


        imgView = (ImageView)findViewById(R.id.imgViewDir);
        imgView.setImageResource(R.drawable.s); //starta a seta

        btLocalizar = (Button) findViewById(R.id.btLocalizar);
        btLocalizar.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                startGPS();
            }
        });



        //passa os valores para a variavel
        destinoFinal.setLatitude(latFinal);
        destinoFinal.setLongitude(longFinal);
    }
```

```
    //Método que faz a leitura de fato dos valores recebidos do GPS
    public void startGPS(){

        LocationManager lManager =
(LocationManager)getSystemService(Context.LOCATION_SERVICE);
        lListener = new LocationListener() {
            public void onLocationChanged(Location locat) {
                updateView(locat);
                if (comecouAMover) {
                    mover(locat);
                } else {
                    comecouAMover = true;
                    primeiroPasso(locat);
                }

                /* Tentativa de usar o norte verdadeiro como base inicial,
não deu certo pois para funcionar o smartphone deveria ao menos começar
virado para o norte.
                geoField = new GeomagneticField(
                        Double.valueOf(locat.getLatitude()).floatValue(),
                        Double.valueOf(locat.getLongitude()).floatValue(),
                        Double.valueOf(locat.getAltitude()).floatValue(),
                        System.currentTimeMillis()
                );
                heading = geoField.getDeclination();*/
            }
            public void onStatusChanged(String provider, int status, Bundle
extras) {}
            public void onProviderEnabled(String provider) {}
            public void onProviderDisabled(String provider) {}
        };
        lManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000,
5, lListener);
    }

    //  Método que faz a atualização da tela para o usuário.
    public void updateView(Location locat){
        latAtual = locat.getLatitude();
        longAtual = locat.getLongitude();

        tvLatitudeAtual.setText(latAtual.toString());
        tvLongitudeAtual.setText(longAtual.toString());

        //Atualiza os valores para comparação
        difLatAtual = latAtual - latFinal;
        difLongAtual = longAtual - longFinal;
    }



    public void primeiroPasso(Location locat){
    /*    latInicial = latAtual;
        longInicial = longAtual;
        difLat = latInicial - latFinal;
        difLong = longInicial - longFinal;
        */
        resultBearingAnt = locat.bearingTo(destinoFinal);
        resultDistanceAnt = locat.distanceTo(destinoFinal);
        mover(locat);
        if (resultBearingAnt < 0) {
            directionAtual = resultBearingAnt + 360; // conversao de norte
```

```
verdadeiro (-180 --- 180)
        }else{
            directionAtual = resultBearingAnt;
        }
        directionAnt = directionAtual;
    }

    public void mover(Location locat){
        //resultLat = difLat - difLatAtual;
        //resultLong = difLong - difLongAtual;
        //difLatAtual = latAtual - latFinal;
        //difLongAtual = longAtual - longFinal;

        resultDistance = locat.distanceTo(destinoFinal);

        resultBearingAtual = locat.bearingTo(destinoFinal);
        if (resultBearingAtual < 0) {//Caso o bearing seja um valor
negativo, atualiza ele
            directionAtual = resultBearingAtual + 360; // conversao de
norte verdadeiro (-180 --- 180)
        }else{
            directionAtual = resultBearingAtual;
        }
        if(resultDistance > resultDistanceAnt){
            difDirection = directionAtual - directionAnt;
                if((difDirection > 3) || (difDirection < 3)){//Colocando
uma margem de 3° para mais ou para menos, para assumir que o usuario está
na direção certa, mas no sentido oposto
                    if(directionAtual >= 180){
                        direction = directionAtual - 180;
                    }
                    direction = directionAtual + 180;
                }else if((difDirection > 4) && (difDirection <= 15)){
                    direction = directionAtual + 5 ;
                }else if((difDirection > 15) && (difDirection <= 30)) {
                    direction = directionAtual + 10;
                }else if((difDirection > 30) && (difDirection <= 45)) {
                    direction = directionAtual + 15;
                }else if((difDirection > 45) && (difDirection <= 60)) {
                    direction = directionAtual + 25;
                }else if((difDirection > 60) && (difDirection <= 75)) {
                    direction = directionAtual + 30;
                }else if((difDirection > 75) && (difDirection <= 90)) {
                    direction = directionAtual + 45;
                }else if((difDirection <= -4) && (difDirection > -15)){
                    direction = directionAtual - 5 ;
                }else if((difDirection <= -15) && (difDirection > -30)) {
                    direction = directionAtual - 10;
                }else if((difDirection <= -30) && (difDirection > -45)) {
                    direction = directionAtual - 15;
                }else if((difDirection <= -45) && (difDirection > -60)) {
                    direction = directionAtual - 25;
                }else if((difDirection <= -60) && (difDirection > -75)) {
                    direction = directionAtual - 35;
                }else if((difDirection <= -75) && (difDirection > -90)) {
                    direction = directionAtual - 45;
                }
        }else{
            direction = directionAtual;
        }
        /*if(resultDistance > resultDistanceAnt){
```

```
            if(directionAnt - directionAtual < 1){//para verificar se não
houve uma diferença significativa na direção que o usuario esta indo
                if(direction > 180){
                    direction = direction - 180;
                }else if(direction  <= 180){
                    direction = direction + 180;
                }
            }
          }else{*/
             // direction = directionAnt - directionAtual;
        //}
        //  direction = directionAnt - directionAtual;


       //imgView.setImageResource(R.drawable.s); //seta a img smp apontando
para a frente, para poder reiniciar o posicionamento
       //direction = resultBearingAtual - heading;

          //if (direction < 0) {
          //    direction = direction + 360;
          //}

          imgView.setRotation(direction);

          tvDirecao.setText("Bearing Atual:" + resultBearingAtual + "
Distancia:" + resultDistance + "Bearing Ant: " + resultBearingAnt +"
Direction: "+direction);

          resultBearingAnt = resultBearingAtual;
          resultDistanceAnt = resultDistance;
          directionAnt = directionAtual;


     //   if((resultLat > 0.001)&&(resultLat <= 0.003)&&(resultLong >
0.001)&&(resultLong <= 0.003)){
      /*
         if(resultBearingAtual < 11.25 && resultBearingAtual > -11.25) {
              imgView.setImageResource(R.drawable.s);
           //   imgView.setRotation(60);
          }else if(resultBearingAtual < 37.5 && resultBearingAtual > 11.25) {
              imgView.setImageResource(R.drawable.ssw);
          //    imgView.setRotation(60);
          }else if(resultBearingAtual < 56.75 && resultBearingAtual > 37.5) {
              imgView.setImageResource(R.drawable.sw);
          //    imgView.setRotation(60);
          }else if(resultBearingAtual < 78.75 && resultBearingAtual > 56.75)
{
              imgView.setImageResource(R.drawable.wsw);
          //    imgView.setRotation(60);
          }else if(resultBearingAtual < 101.25 && resultBearingAtual > 78.75)
{
              imgView.setImageResource(R.drawable.w);
          //    imgView.setRotation(60);
          }else if(resultBearingAtual < 123.75 && resultBearingAtual >
101.25) {
              imgView.setImageResource(R.drawable.wnw);
          //    imgView.setRotation(60);
          }else if(resultBearingAtual <146.25 && resultBearingAtual > 123.75)
{
              imgView.setImageResource(R.drawable.nw);
          //    imgView.setRotation(60);
```

```
        }else if(resultBearingAtual <168.75 && resultBearingAtual > 146.25)
{
            imgView.setImageResource(R.drawable.nnw);
        //    imgView.setRotation(60);
        }else if(resultBearingAtual <-168.75 && resultBearingAtual >
168.75) {
            imgView.setImageResource(R.drawable.n);
        //    imgView.setRotation(60);
        }else if(resultBearingAtual <-146.25 && resultBearingAtual > -
168.75) {
            imgView.setImageResource(R.drawable.s);
        //    imgView.setRotation(180);
        }else if(resultBearingAtual <-123.75 && resultBearingAtual > -
146.25) {
            imgView.setImageResource(R.drawable.ne);
        //    imgView.setRotation(60);
        }else if(resultBearingAtual <-101.25 && resultBearingAtual > -
123.75) {
            imgView.setImageResource(R.drawable.ene);
        //    imgView.setRotation(60);
        }else if(resultBearingAtual <-78.75 && resultBearingAtual > -
101.25) {
            imgView.setImageResource(R.drawable.e);
        //    imgView.setRotation(60);
        }else if(resultBearingAtual <-56.25 && resultBearingAtual > -
78.75){
            imgView.setImageResource(R.drawable.ese);
        //    imgView.setRotation(60);
        }else if(resultBearingAtual <-31.75 && resultBearingAtual > -
56.25){
            imgView.setImageResource(R.drawable.se);
        //    imgView.setRotation(60);
        }else if(resultBearingAtual <-11.25 && resultBearingAtual > -
31.75){
            imgView.setImageResource(R.drawable.sse);
        //    imgView.setRotation(60);
        }*/
    }
}
```

Classe: “Layout device_list.xml”

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
    android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".DeviceList">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Paired Devices"
        android:id="@+id/textView"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Paired Devices"
        android:id="@+id/button"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <ListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/listView"
        android:layout_centerHorizontal="true"
        android:layout_above="@+id/button"
        android:layout_below="@+id/textView" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Iniciar Trajeto"
        android:id="@+id/btTrajeto"
        android:layout_below="@+id/listView"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />
</RelativeLayout>
```

Classe: “Layout activity_gps_control.xml”

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
    android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity"
    >


    <Button
        android:id="@+id/btLocalizar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Obter localização"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Latitude: "
        android:id="@+id/tvLatitude"
        android:layout_alignTop="@+id/tvLatitudeAtual"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Longitude: "
        android:id="@+id/tvLongitude"
        android:layout_marginTop="45dp"
        android:layout_below="@+id/tvLatitude"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:id="@+id/tvLatitudeAtual"
        android:text="Adasdadsa"
        android:layout_toRightOf="@+id/tvLongitude"
        android:layout_toEndOf="@+id/tvLongitude"
        android:layout_marginTop="63dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:id="@+id/tvLongitudeAtual"
        android:text="dawdawd"
```

```
            android:layout_alignTop="@+id/tvLongitude"
            android:layout_alignRight="@+id/tvLatitudeAtual"
            android:layout_alignEnd="@+id/tvLatitudeAtual" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceLarge"
            android:text="Direção que deve ir:"
            android:id="@+id/textView2"
            android:layout_below="@+id/tvLongitude"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:layout_marginTop="42dp" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceLarge"
            android:text="Dados Atuais:"
            android:id="@+id/textView"
            android:layout_alignParentTop="true"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceSmall"
            android:text="em branco"
            android:id="@+id/tvDirecao"
            android:layout_centerVertical="true"
            android:layout_alignRight="@+id/tvLongitudeAtual"
            android:layout_alignEnd="@+id/tvLongitudeAtual" />

        <ImageView
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/imgViewDir"
            android:layout_alignParentLeft="true"
            android:layout_alignWithParentIfMissing="false"
            android:layout_above="@+id/btLocalizar"
            android:layout_below="@+id/tvDirecao"
            android:layout_alignRight="@+id/textView2"
            android:layout_alignEnd="@+id/textView2"
            android:layout_marginTop="60dp"
            android:layout_alignParentStart="false" />

    </RelativeLayout>
```