**UNIVERSIDADE FEDERAL DOS VALES DO JEQUITINHONHA E MUCURI**
**SISTEMAS DE INFORMAÇÃO**
**ÁLVARO LUIZ MARINHO PEREIRA**

# DESENVOLVIMENTO DE UMA PLATAFORMA DE SENSORIAMENTO MÓVEL DE BAIXO CUSTO COM O ARDUÍNO

**DIAMANTINA**
**2016**

**ÁLVARO LUIZ MARINHO PEREIRA**

# DESENVOLVIMENTO DE UMA PLATAFORMA DE SENSORIAMENTO MÓVEL DE BAIXO CUSTO COM O ARDUÍNO

Monografia apresentada ao curso de Sistemas de Informação da Universidade Federal dos Vales do Jequitinhonha e Mucuri – UFVJM, como pré-requisito para obtenção do grau de bacharel, sob orientação do Professor Mestre Rafael Santin

**Diamantina**
**2016**

**ÁLVARO LUIZ MARINHO PEREIRA**

# DESENVOLVIMENTO DE UMA PLATAFORMA DE SENSORIAMENTO MÓVEL DE BAIXO CUSTO COM O ARDUÍNO

Monografia apresentada ao curso de Sistemas de Informação da Universidade Federal dos Vales do Jequitinhonha e Mucuri – UFVJM, como pré-requisito para obtenção do grau de bacharel.

**COMISSÃO EXAMINADORA**

_____

Prof. MSc. Rafael Santin

_____

Prof. Dr. Alessandro Vivas Andrade

_____

Profª. Dr. Luciana Pereira de Assis

Aprovado em: ____/____/_____

Ao meu avô, Geraldo (in memorian).

## AGRADECIMENTO

Agradeço a Deus por ter me propiciado tudo nessa vida, aos meus pais, Auresto e Sônia, pelo apoio e suporte ao longo dessa caminhada, à minha irmã Anna Clara por todos os conselhos e conversas e à minha família que sempre se mostrou presente.

Agradeço aos meus amigos de Jequitinhonha que me acompanharam durante o meu crescimento. Às novas amizades feitas durante a graduação que se mostraram verdadeiros e fieis, em especial à Ana Lisa, Barbara e Marcia. Aos integrantes das repúblicas Amoriçoca e Farrapos (e agregados) que acrescentaram significativamente na minha experiência de vida. Ao grupo Xotear pelos momentos de descontração, em especial à Geniny e Fabrício, pelos ensinamentos.

Agradeço também aos professores que nos guiaram ao longo dessa jornada, aos amigos do curso de Sistemas de Informação que compartilharam momentos difíceis, em especial Diego Guilherme, Renato e Ricardo. Ao meu orientador Rafael Santin que aceitou esse desafio e se mostrou sempre paciente e compromissado na elaboração desse trabalho. Aos amigos do SIGA Ensino/ADM que ajudaram na construção do meu aprendizado em tão pouco tempo.

*“Nunca deixe ninguém te dizer que você não pode fazer algo. Se você tem um sonho, precisa protege-lo. As pessoas não conseguem vencer e vão dizer que você também não vai vencer. Se quiser alguma coisa, corre atrás.”*

À Procura da Felicidade

## RESUMO

Este trabalho envolve o desenvolvimento de um aplicativo para Sistema Operacional Android atrelado com a criação de uma plataforma robótica baseada em Arduíno. Tem como principal objetivo desenvolver um sistema capaz de manipular o robô através de sinais digitais emitidos pelas interfaces de comunicação Bluetooth de um smartphone com sistema operacional Android.

O projeto tem como motivação o crescimento do uso do sistema operacional Android nos dispositivos móveis em todo o mundo, juntamente com o grande crescimento do uso da plataforma Arduíno na área da robótica para a prototipação de novos produtos.

O sistema foi desenvolvido para a manipulação de uma plataforma robótica, com os movimentos controlados por comandos direcionais ou pelo acelerômetro do celular. Além disso, foi implementado um modo autônomo que permite que o robô se locomova no ambiente, sem a interação do usuário.



Palavras-chave: Android, Arduíno, Robótica, Reciclagem.

## ABSTRACT

This work involves the development of an application for the Android operating system combined with the creation of a Arduino based robotic platform. It has as its main goal develop a system capable of manipulating a robot through digital signals emitted by the bluetooth communication interfaces of a Android based smartphone.

The project has as motivation the global use growth of the Android OS on mobile devices and Arduino based platforms in robotics for prototyping new products.

The system was developed for manipulating a robotic platform, with movements controlled directional commands or by the mobile accelerometer. In addition, a autonomous mode that allows the robot to move in the environment, without user interaction.



Key Words: Android, Arduíno, Robotic, Recycling.

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ARM | Advanced RISC Machine |
| bps | Bytes Por Segundo |
| cm | Centímetros |
| dBm | Decibel Milliwatt |
| EEPROM | Electrically-Erasable Programmable Read-Only Memory |
| Ghz | Giga-hertz |
| GND | Ground |
| GPS | Global Positioning System |
| kHz | Quilohertz |
| KΩ | Quilo-ohm |
| LED | Light Emitting Diode |
| m/s | Metros Por Segundo |
| mA | Miliampere |
| Mhz | Mega-hertz |
| mm | Milímetro |
| NASA | National Aeronautics and Space Administration |
| NFC | Near Field Communication |
| pF | Picofarads |
| RISC | Reduced Instruction Set Computer |
| SD | Secure Digital |
| SDK | Software Development Kit |
| SMS | Short Message Service |
| SRAM | Static Random Access Memory |
| USB | Universal Serial Bus |
| v | Volt |
| VCC | Voltagem Corrente Contínua |
| W | Watts |

## LISTA DE ILUSTRAÇÕES

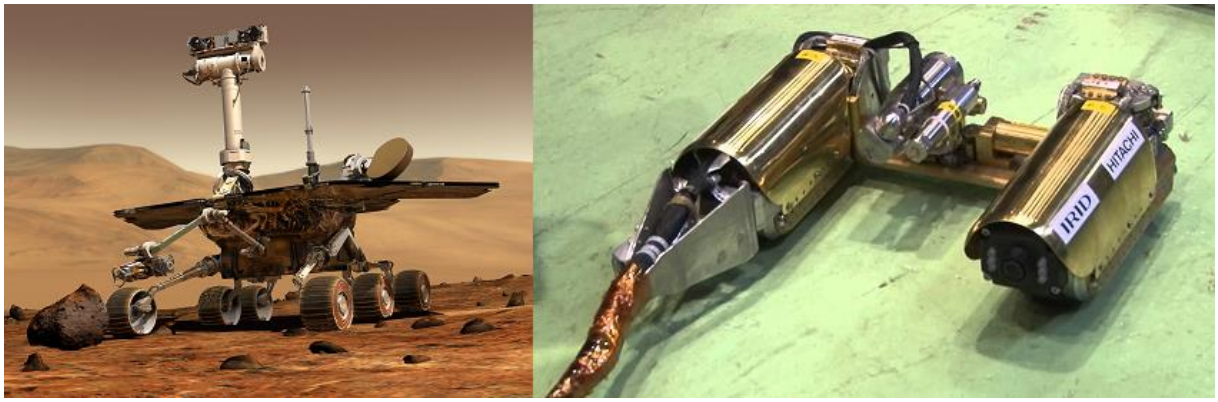
## LISTA DE TABELAS

**SUMÁRIO**

# 1. INTRODUÇÃO

Os avanços das tecnologias móveis permitem que cada vez mais robôs sejam utilizados para auxiliarem e facilitarem as atividades humanas. Além disso, os avanços na área de robótica permitem que robôs possam realizar tarefas perigosas, como o acesso a lugares inóspitos ou que oferecem alto risco a vida humana. Como os veículos exploradores enviados à Marte pela NASA que são equipados com diversos equipamentos para exploração de ambiente extraterrestre (NASA, 2015). Outro exemplo é o robô Tepco da Tokyo Eletric Power que foi usado para retirar as barras de combustível do reator da usina nuclear de Fukushima, onde os níveis de radiação atuais impedem que uma pessoa realize diretamente essa tarefa (COMPANY, 2015). Na Figura 1 vemos o Mars Rover e o Tepco.

*Figura 1 - a) Mars Rover b) Tepco*




*Fonte:* (NASA, 2015) *e* (COMPANY, 2015)

Porém o custo de produção e/ou compra desses equipamentos é altíssimo. Alguns drones e jipes não tripuláveis utilizados em monitoramento de plantações podem variar de US$100 mil a US$ 200 mil dependendo da configuração escolhida. Atualmente o mercado desses robôs é praticamente dominado por empresas estrangeiras como a americana Agribotix, a suíça senseFly, a francesa Airinov. Vale destacar a brasileira XMobots que nos últimos anos vem ganhando espaço na venda de três modelos de drones que custam de R$170 mil a R$450 mil (Xmobots, 2015).

Entretanto, é perceptível o surgimento de elementos eletrônicos configuráveis com Arduíno que facilitaram o desenvolvimento de diversas aplicações, inclusive para robótica. Além disso, o custo para desenvolvimento com a plataforma Arduíno é baixo e o uso de componentes em módulos permite o seu reaproveitamento. Os trabalhos de Moreira et. al e Gomes e Tavares (2009) mostram a facilidade e rapidez no desenvolvimento de produtos eletrônicos.

A integração da plataforma Arduíno com dispositivos móveis, como os celulares, tem avançado muito nos últimos anos, permitindo a execução de aplicações cada vez mais complexas em dispositivos de baixo custo. Visto que os celulares oferecem alto poder de processamento, e uma vasta gama de sensores e interfaces de conexão, favorecendo a criação e integração entre diferentes aplicações. Em (BEGHINI, 2013) podemos ver um exemplo de automação residencial de baixo custo utilizando o Arduíno e um celular com Android.

Assim, o presente trabalho apresenta o desenvolvimento de uma plataforma de sensoriamento móvel de baixo custo para facilita o estudo de robótica. Os principais materiais utilizados no desenvolvimento deste trabalho consistem nos componentes do Arduíno e peças recicladas de equipamentos de informática.

Embora existam na literatura alguns exemplos de plataforma de sensoriamento, deparamos com o problema da falta de padronização no processo de desenvolvimento, como as questões da utilização dos controladores e os smartphones. Além disso, este trabalho apresenta as técnicas utilizadas para o controle da plataforma, que é realizado sem fio, por meio de uma aplicação de celular.

## 1.1. OBJETIVOS

### 1.1.1. GERAL

O objetivo geral deste trabalho é o desenvolvimento de uma plataforma robótica móvel de baixo custo, que seja adaptável para o uso de diferentes sensores e motores, sendo controlados remotamente por uma aplicação de celular.

### 1.1.2. ESPECÍFICOS

Os objetivos específicos deste trabalho são:

- Estudo dos conceitos e teorias envolvidos neste projeto, como os elementos de circuitos integrados, capacitores, cristais, fontes de tensão e etc;
- Aprender a utilizar a plataforma Arduíno, como o microcontrolador Atmega e a sua linguagem de programação;
- Desenvolvimento do código para Arduíno que irá receber os eventos provenientes dos diversos sensores existentes no celular;
- Desenvolvimento de uma aplicação para dispositivos móveis contendo as funcionalidades que são executadas na plataforma;
- Montagem da plataforma robótica com componentes provenientes da reciclagem de outros equipamentos.

## 1.2. JUSTIFICATIVA

O notável avanço da robótica nos últimos anos, juntamente com a popularização de dispositivos móveis está proporcionando o aparecimento de técnicas de interação cada vez mais intuitivas. Esses mecanismos estão empregados em nosso cotidiano nas mais variadas situações como o Roomba da iRobot, que consiste num um aspirador de pó robótico que limpa a casa sozinho (IROBOT, 2002) ou o carro F015 Luxury in Motion da Mercedes-Benz que não apenas consegue se deslocar sem interferência humana, como permite um alto grau de interação com os passageiros (MERCEDES-BENZ, 2015).

Assim, percebe-se a importância de um estudo sobre o desenvolvimento de plataformas robóticas integradas a dispositivos móveis que auxiliem em tarefas de monitoramento e/ou reconhecimento de ambientes. Na atualidade existem muitas formas de integração com a plataforma Android como a utilização da plataforma Arduíno, que oferecem diversas tecnologias como a comunicação sem fio por meio de Bluetooth, Wi-Fi e NFC, além de diversos sensores, como infravermelho, sonar, entre outros.

Esta grande variedade de formas de comunicação e sensores possibilitam a criação de diversos novos produtos e serviços, utilizando estas tecnologias. Por isso, o uso do Arduíno é cada vez mais visto em projetos de robótica e automação, sendo um dos principais fatores que incentivaram o desenvolvimento deste projeto.

## 1.3. ORGANIZAÇÃO DO TRABALHO

O Capítulo 2 apresenta a estrutura do sistema operacional Android, da plataforma Arduíno e um breve histórico acerca da robótica, para um melhor entendimento e contextualização, expondo um referencial teórico do trabalho.

No Capítulo 3 são apresentas as etapas de desenvolvimento, os materiais, as linguagens e ferramentas usadas na criação da plataforma robótica e do aplicativo de controle e suas características singulares.

O Capítulo 4 são apresentados as dificuldades enfrentadas, os testes realizados e os resultados do projeto.

Por último no Capítulo 5 apresenta as conclusões e sugestões para trabalhos futuros.

# 2. REFERENCIAL TEÓRICO

Todos os conceitos necessários para o entendimento do presente trabalho são apresentados neste capítulo.

A sessão 2.1 trata sobre o surgimento do Sistema Operacional Android, sua arquitetura e os componentes de sua aplicação. A sessão 2.2 descreve a plataforma Arduíno, suas variações e o projeto standalone.

Já sessão 2.3 trata assuntos acerca da robótica, com uma introdução, seu histórico, robótica móvel e os componentes de um robô dando ênfase nos circuitos integrados, microcontrolador, ponte H e motores DC.

## 2.1. SISTEMA OPERACIONAL ANDROID

O Google, observando o crescimento no uso de dispositivos móveis, uniu-se a grandes fabricantes de aparelhos celulares como a LG, Samsung e Sony formando um grupo chamado de Open Handset Alliance, com intenção de padronizar uma plataforma de código aberto e livre para celulares, justamente para atender a todas as expectativas e tendências do mercado atual (Android Open Source Project, 2015).

Inicialmente, o sistema operacional foi desenvolvido pela empresa Android Inc., que foi comprada pela Google em 2005. Em 2007, essa plataforma tornou-se a primeira plataforma de código aberto de desenvolvimento para dispositivos móveis. Essa plataforma possui diversos componentes com uma variada disponibilidade de bibliotecas e interface gráfica, além de um kit de desenvolvimento chamado Android SDK com ferramentas e APIs (Interface de Programação de Aplicativos) necessárias para criar aplicações (LECHETA, 2010)

### 2.1.1. ARQUITETURA ANDROID

O Google refere-se ao Android como um conjunto de software, pois sua arquitetura é constituída por camadas e componentes no qual suportam funções e serviços específicos do sistema operacional.

Inicialmente temos o núcleo (kernel), onde encontramos os programas de gerenciamento de memória, o software de gerenciamento de energia, configurações de segurança, vários drivers de hardware, entre outros. Seguindo pela camada das bibliotecas nativas (coleção de subprogramas utilizados no desenvolvimento) que contém as instruções que dizem ao dispositivo como lidar com diferentes tipos de dados.

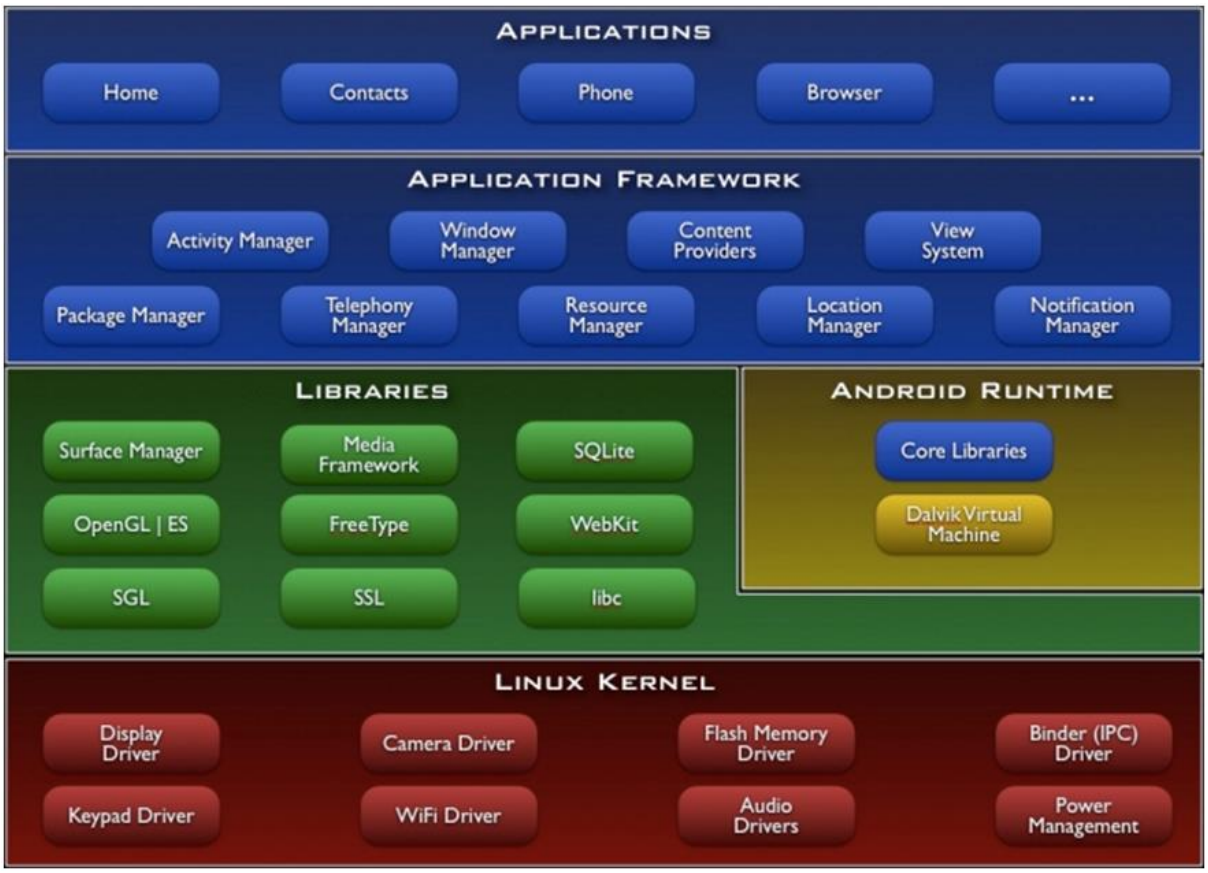
A camada de execução permite que as aplicações baseadas na plataforma sejam executadas. Um dos componentes desta camada são as *core libraries*, que disponibilizam uma interface Java utilizada para programação. Já o outro componente é a Dalvik Virtual Machine, que é uma máquina virtual para suporte à execução de aplicações.

Posteriormente a camada dos frameworks de aplicação que provê um conjunto de bibliotecas para acessar os diversos recursos do dispositivo como interface gráfica, serviços de telefonia, localizador (GPS), armazenamento no cartão SD, entre outros.

Por último a camada de aplicação que é a camada de interação entre o usuário e o dispositivo móvel, nela encontramos as funções básicas do dispositivo, aplicativos cliente de e-mail, programa de SMS, calendário, mapas, navegador, contatos, entre outros (LECHETA, 2010).

A Figura 2 esquematiza essa arquitetura e seus componentes.

*Figura 2 - Arquitetura Android*




*Fonte: http://www.mobiltec.com.br/*

### 2.1.2. APLICAÇÃO ANDROID

Uma aplicação Android é construída a partir de diversos componentes. Cada componente é uma ferramenta pelo qual o sistema pode interagir com sua aplicação, tendo identidade própria, relacionamento com outros componentes e suas próprias regras de funcionamento que ajudam a definir o comportamento geral da sua aplicação.

Há quatro tipos diferentes de componentes de aplicação. Cada tipo tem a sua finalidade e um ciclo de vida distinto que define como o componente é criado e destruído.

Segundo (Android Developers, 2015), os quatro componentes de uma aplicação são:

A. Activity (atividade): representa uma tela com uma interface de usuário. Embora as atividades trabalhem juntas para formar uma experiência de usuário coesa na aplicação, cada uma é independente das outras. Para criar uma atividade é necessário criar um arquivo de *layout*, que contém um ViewGroup ou Views internos, todos posicionados de acordo com a usabilidade da sua aplicação. Após a criação do *layout*, é necessário criar a subclasse de Activity, que será invocada quando o usuário solicitar. Uma atividade pode iniciar outros componentes de sua aplicação, como: outras Atividades, Serviços, entre outros;
B. Service (serviço): é um componente que executa operações de longo prazo ou trabalho para processos remotos em segundo plano. Um serviço não fornece uma interface de usuário. Por exemplo, um serviço deve tocar música em plano de fundo enquanto o usuário está em uma aplicação diferente. Outro componente, como uma atividade, pode iniciar o serviço e deixá-lo rodando ou anexá-lo para poder interagir com ele. Um serviço é implementado como uma subclasse do componente Service.
C. Content providers (provedores de conteúdo): gerencia um conjunto de dados da aplicação, disponibilizando-os para outras aplicações. Este componente permite que outras aplicações consultem ou até mesmo modifiquem os dados. Por exemplo, o sistema Android fornece um provedor de conteúdo que gerencia as informações de contato do usuário. Não há obrigatoriedade em criar um provedor de conteúdo para lidar com os dados de sua aplicação, apenas se quiser compartilhá-los. Um provedor de conteúdo é implementado como uma subclasse do Content providers de conteúdo e deve implementar um conjunto padrão de APIs que habilitam outras aplicações a realizarem as transações.

D. Broadcast receivers: é um componente que responde a anúncios disparados a todo o sistema. Um *broadcast receiver* ao ser iniciado fica observando se determinado evento geral acontece. É diferente do componente Atividade, que somente observa eventos gerados pelas suas Views. Este tipo de componente pode receber eventos gerados pelo código da aplicação, de outras aplicações ou mesmo do sistema operacional Android. Um exemplo é o “SCREEN_OFF”, gerado quando a tela é bloqueada. Embora *broadcast receivers* não exibam uma interface de usuário, elas podem criar uma notificação na barra de estado, para o alertando o usuário sobre a ocorrência de algum evento broadcast.

## 2.2. ARDUÍNO

Arduíno é uma plataforma de prototipagem eletrônica de código livre composta por um hardware (placa controladora) e software (ambiente de desenvolvimento) de fácil utilização. Destinado a todo tipo de pessoa interessada em desenvolver projetos ou ambientes interativos (Arduíno, 2015).

Possui uma grande quantidade de portas de entrada/saída e terminais que permitem a conexão com vários dispositivos externos como sensores, luzes, motores, dentre outros componentes para a criação de inúmeros projetos. O Arduíno pode ser alimentado via um cabo USB conectado no computador, uma bateria externa ou uma fonte de alimentação de até 12v (MONK, 2013).

### 2.2.1. VARIAÇÕES DO ARDUÍNO

O Arduíno é fabricado pela companhia italiana Smart Projects que oferecem 15 modelos comercialmente do dispositivo. A seguir serão apresentados os diversos modelos e suas peculiaridades:

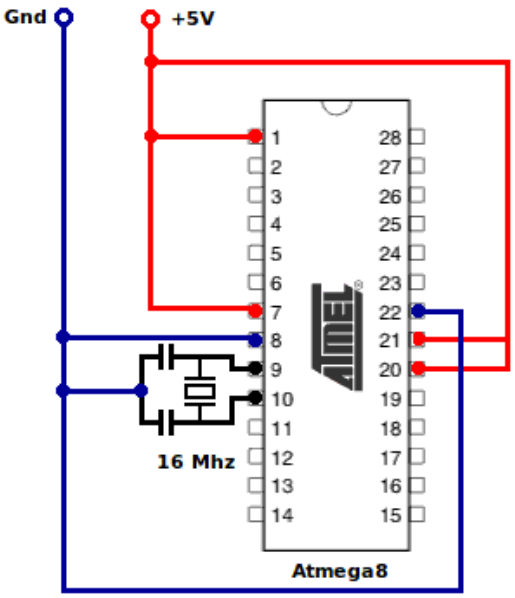
- Serial Arduíno - A versão original da placa serial Arduíno;
- Arduíno Extreme: Esse modelo permite a conexão com um número maior de componentes em relação aos modelos anteriores;
- Arduíno NG - Oferece o conversor de USB para Serial exigindo menos componentes externos;
- Arduíno BT - Oferece a interface bluetooth para comunicação;
- Arduíno Mini/Micro/Nano - versão em miniatura;
- LilyPad Arduíno - projeto minimalista para aplicações em produtos têxteis;
- Arduíno Diecimila - Permite um menor consumo de energia da placa quando alimentado por uma fonte externa;
- Arduíno Duemilanove - Seleciona automaticamente a fonte de alimentação apropriada (USB ou fonte de alimentação externa);
- Arduino PRO - A placa vem sem conectores pré-montados permitindo ao usuário o uso de diversos tipos de conectores ou cabos soldados diretamente;
- Arduino Uno - Permite a troca do chip microcontrolador, facilitando a sua substituição em em caso de dano, ou a sua utilização em projetos dedicados.
- Arduíno Leonardo - Possui 12 portas analógicas e 20 digitais, além do conector micro USB, que emular dispositivos USB;
- Arduino Yun - perfeita para usar ao projetar os dispositivos conectados;
- Arduino Mega - Possui uma conexão USB dedicada à ligação com dispositivos baseados em Android;
- Arduíno Due - Basedo no microcontrolador ARM, esse modelo possui maior capacidade de processamento.
- Arduíno 101 (ou Genuíno) - Possui bluetooth, acelerômetro e giroscópio integrados.

Além desses modelos, é possível construir manualmente as placas, visto que os projetos de hardware estão disponíveis sob uma licença de código aberto, permitindo a personalização de acordo com as necessidades do usuário (Arduíno, 2015).

### 2.2.2. ARDUÍNO STADALONE

Como visto em (Placa Standalone, 2015) a palavra Standalone ou stand alone significa autônomo, ou seja, completamente autossuficiente e no contexto do Arduíno refere-se à independência do seu projeto da placa de programação, ou seja, é a montagem do Atmega328 juntamente com poucos componentes. Devido ao fato do standalone ser formado por poucos componentes, as possibilidades de formas e tamanhos são inúmeras. Na Figura 3 tem-se o esquema de um standalone:

*Figura 3 - Esquema Standalone*




*Fonte: http://blogdonatanael.blogspot.com.br/*

Os seguintes componentes são utilizados para a montagem standalone:

- 1 resistor de 10KΩ;
- 2 capacitores cerâmicos de 22pF;
- 1 cristal de 16Mhz;
- 1 microcontrolador Atmega328 com bootloader;

## 2.3. ROBÓTICA

### 2.3.1. INTRODUÇÃO

A definição de robô é complexa. É bem provável que uma pessoa saiba reconhecer um dispositivo robótico, mas tenha dificuldade em defini-lo. Como Joseph F. Engelberger, considerado por muitos como um dos pais da robótica por ter construído e vendido o primeiro robô industrial, o Unimate em 1950 (AZEVEDO, 2015), disse:

> *"Eu não posso definir um robô, mas eu reconheço um quando o vejo"*.

O termo robô tem origem na palavra checa "*robota*", que significa "trabalho forçado" e foi pela primeira vez usado pelo Checo Karel Capek (1890-1938) numa Peça de Teatro - R.U.R. (Rossum's Universal Robots) - estreada em janeiro de 1921 (Praga). Segundo o dicionário da Universidade de Oxford, robô é uma máquina, programável por um computador, capaz de resolver uma série complexa de ações automaticamente (Oxford Dictionaries, 2015). Outra definição bastante interessante é a da Sociedade Brasileira para o Progresso da Ciência que afirma que um robô é um dispositivo autônomo ou semiautônomo, capaz de realizar trabalhos de acordo com um controle humano, controle parcial com supervisão, ou de forma autônoma (ALFARO, 2006).

Assim, percebemos que um robô é um conjunto de dispositivos, sejam eles mecânicos ou elétricos, que realizam trabalhos de uma forma automática ou pré-programada a fim de auxiliar o ser humano em suas diversas tarefas, das mais simples às mais perigosas e complexas.

A robótica é uma área de pesquisa que lida com os problemas relacionados a concepção, construção, operação, e aplicações de robôs. A robótica contribui para o desenvolvimento de diversos sistemas como aparelhos eletrodomésticos, eletrônicos, automóveis, entre outros.

### 2.3.2. HISTÓRIA DA ROBÓTICA

Os primeiros relatos de modelos robóticos que se tem notícia foram na civilização grega. Tinham aparência humana e/ou animal que usavam sistemas de pesos e bombas pneumáticas, apesar de não ter nenhuma necessidade prática ou econômica para época. Vale ressaltar o matemático e engenheiro grego Ctesibius, que viveu cerca de 285-222 a.C. em Alexandria e criou diversos esquemas robóticos, sendo o mais famoso destes a clepsidra ou relógio de água, um dos primeiros sistemas criados pelo homem para medir o tempo.

Posteriormente, cientistas árabes buscaram atribuir funções aos robôs que fossem ao encontro das necessidades humanas.

Em 1495, Leonardo DaVinci desenvolveu uma grande pesquisa da anatomia humana que permitiu aplicar esse conhecimento para a criação de articulações mecânicas. Como resultado deste estudo, surgiram diversos exemplares de bonecos que moviam as mãos, os olhos e as pernas, e que conseguiam realizar ações simples como escrever ou tocar alguns instrumentos.

Segundo as definições modernas, o barco teleoperado criado por Nikola Tesla e demonstrado em uma exibição no ano de 1898 no Madison Square Garden é considerado o primeiro robô (História da Robótica, 2015).

Em 1942, foi usado pela primeira vez o termo “robótica” pelo cientista e escritor Isaac Asimov, numa pequena história intitulada "Runaround". Em 1950 Asimov publicou uma história intitulada "I Robot". Nela, ele propôs a existência de três leis como condição de coexistência dos robôs com os seres humanos, como prevenção de qualquer perigo que a inteligência artificial pudesse representar à humanidade, às quais acrescentou, mais tarde, a lei zero (NATALIE, 2005).

As leis propostas são:

- 1ª Lei: Um robô não pode ferir um ser humano ou, por omissão, permitir que um ser humano sofra algum mal.
- 2ª Lei: Um robô deve obedecer as ordens que lhe sejam dadas por seres humanos, excepto nos casos em que tais ordens contrariem a Primeira Lei.
- 3ª Lei: Um robô deve proteger sua própria existência desde que tal proteção não entre em conflito com a Primeira e Segunda Leis.

- Lei Zero: Um robô não pode fazer mal à humanidade e nem, por inação, permitir que ela sofra algum mal.

Com o grande avanço da revolução industrial, no século XVIII, as fábricas buscavam equipar-se com máquinas capazes de realizar e reproduzir automaticamente, determinadas tarefas, como na indústria têxtil com o aparecimento dos primeiros teares mecânicos.

### 2.3.3. ROBÓTICA MÓVEL

Nas últimas duas décadas, após o avanço de robôs industriais, a robótica móvel vem chamando bastante atenção dos estudiosos, que tem concentrado esforços no seu desenvolvimento.

Um fator importante é a introdução de capacidades de mobilidade e autonomia para se adaptar ao ambiente, o que possibilita uma gama de opções de novas aplicações na robótica móvel.

Por exemplo, seu uso em aplicações industriais (transporte automatizado e veículos de carga autônomos), domésticas (aspiradores de pó e cortadores de grama automáticos), urbanas (transporte público, cadeiras de rodas robotizadas), militares (sistemas de monitoramento remoto, transporte de suprimentos e de armamento em zonas de guerra) e de segurança (controle e patrulhamento de ambientes, resgate e exploração em ambientes hostis), entre outros (WOLF, SIMÕES, *et al.*, 2009).

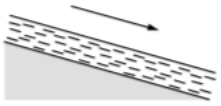
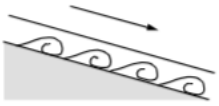
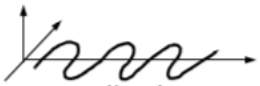
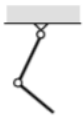
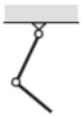
Robôs móveis podem ser classificado de acordo com o ambiente em que eles movem:

- Robôs de terra: possuem rodas ou pernas, se assemelhando como os humanos, animais ou insetos.
- Robôs aéreos: referenciados como veículos aéreos não-tripulados (VANTS ou UAVs).
- Robôs subaquático: veículos subaquáticos autônomos (AUVs).

Uma fonte de inspiração no desenvolvimento de mecanismos de locomoção são os sistemas biológicos. No entanto, replicando a natureza a este respeito é extremamente difícil.

Na Figura 4 mostra-se a classificação de alguns mecanismos de locomoção quanto ao movimento de suas contrapartes biológicas (SIEGWART, NOURBAKHSH e SCARAMUZZA, 2004):

*Figura 4 - Base Natural para os mecanismos de locomoção*

| Type of motion | Resistance to motion | Basic kinematics of motion |
|---|---|---|
| Flow in a Channel  | Hydrodynamic forces | Eddies  |
| Crawl  | Friction forces | Longitudinal vibration  |
| Sliding  | Friction forces | Transverse vibration  |
| Running  | Loss of kinetic energy | Oscillatory movement of a multi-link pendulum  |
| Jumping  | Loss of kinetic energy | Oscillatory movement of a multi-link pendulum  |
| Walking  | Gravitational forces | Rolling of a polygon (see figure 2.2)  |

*Fonte:* (SIEGWART, NOURBAKHSH e SCARAMUZZA, 2004)

### 2.3.4. COMPONENTES DE UM ROBÔ

Para a confecção de um sistema robótico móvel, é necessário ter em mente os principais componentes que ele pode possuir (ABREU, 2002):

- Controlador – É um dispositivo que faz de interface entre o exterior do robô e o seu funcionamento interno. Possui um microprocessador e memória para execução de seu(s) programa(s).
- Sensores – Um dispositivo que permite coletar informações correspondentes a um estímulo físico/químico de maneira específica e mensurável analogicamente; os sensores mais comuns são os de toque, rotação, som (microfone), ultrassom, luz, entre outros.
- Atuadores – É um dispositivo que produz movimento, atendendo a comandos que podem ser manuais, elétricos ou mecânicos. Podem ser motores de diversos tipos, como mecânicos, elétricos, hidráulicos ou pneumáticos.
- Manipuladores – conjunto de corpos ligados por juntas, formando cadeias cinemáticas que definem uma estrutura mecânica. Normalmente possuem um ou mais atuadores em sua estrutura.
- Engrenagens – elementos mecânicos compostos de rodas dentadas.
- Eixo ou Polia – peça que une um motor a engrenagens ou rodas.
- Fonte de energia.
- Fiação – Fios com a finalidade de transmitir sinais entre o controlador, os sensores e os atuadores, e também para a alimentação desses componentes.
- Estrutura – Conjunto de peças de tamanho, formato e cor diversas que serve como base para sustentar o controlador, sensores, atuadores, manipuladores, baterias, geradores, fiação, eixos e engrenagens.

#### 2.3.4.1. CIRCUITO INTEGRADO

Circuito integrado (ou simplesmente C.I.) é um circuito eletrônico composto principalmente por dispositivos semicondutores como transistores, diodos, resistores e capacitores, anexados em uma pequena placa de silício e selado em um bloco de plástico ou cerâmica com terminais que são conectados aos seus componentes por pequenos fios condutores. Com as mais diversas funções e aplicações na indústria, os circuitos integrados estão disponíveis em diversos formatos e tamanhos (encapsulamentos).

O circuito integrado foi inventado por Jack Kilby, da Texas Instruments que decidiu agrupar todos os componentes do circuito eletrônico em um único bloco monolítico feito do mesmo material reduzindo os custos, a energia, melhorando o desempenho e a fabricação poderia ser padronizada. Robert Noyce, da Fairchild Semiconductor, também apresentou sua idéia de circuito integrado, cerca de meio ano depois de Kilby. O projeto de Noyce resolvia alguns problemas do design usado por Kilby, fazendo uso do silício, que se tornaria o padrão da indústria (INSTRUMENTS, 2015)

##### 2.3.4.1.1. MICROCONTROLADOR

Um microcontrolador pode ser descrito como um pequeno microcomputador integrado em um único chip, capaz de realizar controle de máquinas e equipamentos eletro-eletrônicos através de programas, podendo ser empregado em aplicações das mais diversas.

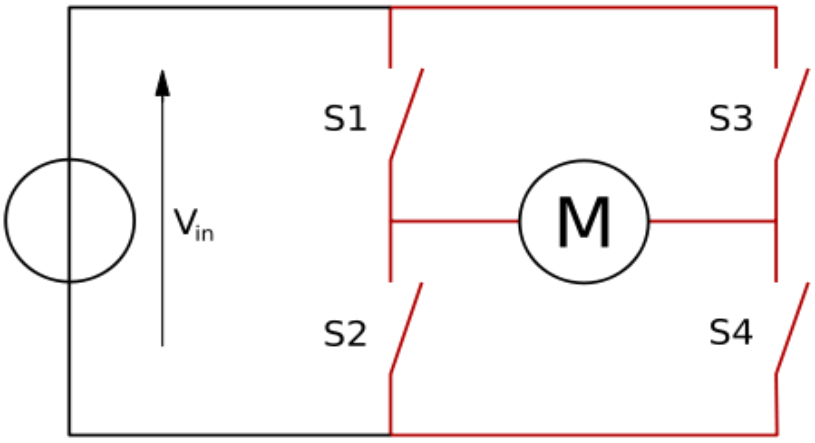
É composto por uma Unidade Central de Processamento (CPU), um sistema de clock para dar sequência às atividades da CPU, uma memória para armazenamento de instruções e para manipulação de dados, entradas e saídas para comunicar a CPU com informações do mundo externo, e o programa (firmware) para definir um objetivo ao sistema (DENARDIN, 2015)

#### 2.3.4.1.2. PONTE H

Ao ligarmos um motor de corrente continua (DC) em uma bateria, percebemos que ele gira numa única direção e numa velocidade constante. Se desejamos alterar o sentido da rotação do motor, basta ligar os terminais do motor de forma invertida. Uma forma automatizada de fazermos isso é utilizando um circuito conhecido como ponte H.

Este circuito tem esse nome pois é composta por 4 chaves mecânicas ou eletrônicas posicionadas formando a letra “H” sendo que cada chave se localiza num extremo e o motor é posicionado no meio como mostra a Figura 5:

*Figura 5 - Esquemático de uma Ponte H*

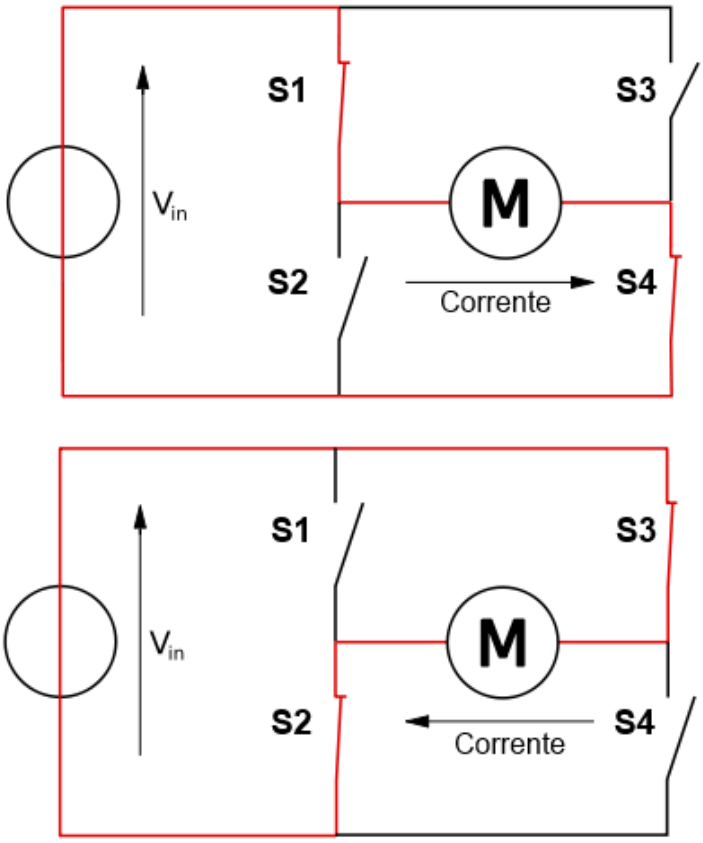



*Fonte: http://blog.filipeflop.com/*

Além de controlar o sentido da rotação de um motor (sentido da corrente) pode-se determinar a polaridade da tensão e a tensão em um dado sistema ou componente. Esse circuito pode ser construído utilizando qualquer tipo de componente que simule uma chave liga-desliga como relés, mosfets ou transistores.

O seu funcionamento se dá quando as 4 chaves são acionadas de forma alternada (S1 e S4 ou S2 e S3). Para cada acionamento de pares de chave o motor gira em um sentido, as chaves S1 e S2 assim como as chaves S3 e S4. Na Figura 6 mostra esse funcionamento.

*Figura 6 - Funcionamento de uma Ponte H*




*Fonte: http://blog.filipeflop.com/*

Deve-se ficar atento para não acionar as chaves de um mesmo lado do “H” simultaneamente (S1 e S2 ou S3 e S4). Isso faz com que o fluxo da corrente vá direto do polo positivo para o negativo, causando um curto-circuito danificando o motor, a fonte de alimentação e até os componentes eletrônicos envolvidos no circuito (BRAGA, 2005).

#### 2.3.4.2. MOTORES DC (CORRENTE CONTÍNUA)

Motor DC é qualquer mecanismo de uma classe de máquinas elétricas que converte energia elétrica de corrente contínua em energia mecânica. Uma característica marcante dessa classe é utilizar as forças produzidas por campos magnéticos. Quase todos os tipos de motores de corrente contínua têm de algum mecanismo interno, eletromecânico ou eletrônico, para alterar periodicamente a direção do fluxo de corrente na parte do motor.

Motores de corrente contínua foram os primeiros tipos amplamente utilizado, uma vez que poderiam ser alimentados a partir de sistemas de distribuição de energia de corrente contínua de iluminação existentes. A velocidade de um motor de corrente contínua pode ser controlada de várias maneiras, utilizando uma tensão de alimentação variável ou alterando a força de corrente nos seus enrolamentos de campo.

Pequenos motores DC são usados em ferramentas, brinquedos e eletrodomésticos. Os motores maiores DC são utilizados na propulsão de veículos eléctricos, elevadores e monta-cargas, ou em unidades de laminadores de aço. O advento da tecnologia tem feito a substituição de motores de corrente contínua com motores de corrente alternada (AC) devido o menor custo e ampla aplicações (ELETRONICS, 2015).

# 3. METODOLOGIA

A seguir é apresentado um resumo das etapas realizadas para o desenvolvimento do trabalho

1. Estudo sobre os fundamentos da Robótica, Eletrônica e tecnologias da arquitetura do Android e Arduíno;
2. Leitura de artigos e trabalhos relacionados com o projeto;
3. Elaboração do pré-projeto;
4. Realização de testes de comunicação do Modulo bluetooth, usando a arquitetura do Arduíno e um aplicativo de envio de dados;
5. Implementação do aplicativo de controle do robô e sua comunicação com o Arduíno;
6. Testes no demais componentes eletrônicos, como o sensor ultrassônico e motores;
7. Montagem em protoboard dos componentes no robô;
8. Confecção da placa de circuito impresso e a finalização da fabricação do robô;
9. Testes finais das operações executadas pelo robô.

## 3.1. PLATAFORMA ROBÓTICA

Visando a reutilização de elementos computacionais descartados, a plataforma controlável desenvolvida para este projeto é formada por uma carcaça de uma gravadora de CD/DVD de computador que além de leve e rígida, possui encaixes de parafusos que permitiram fixar as rodas.

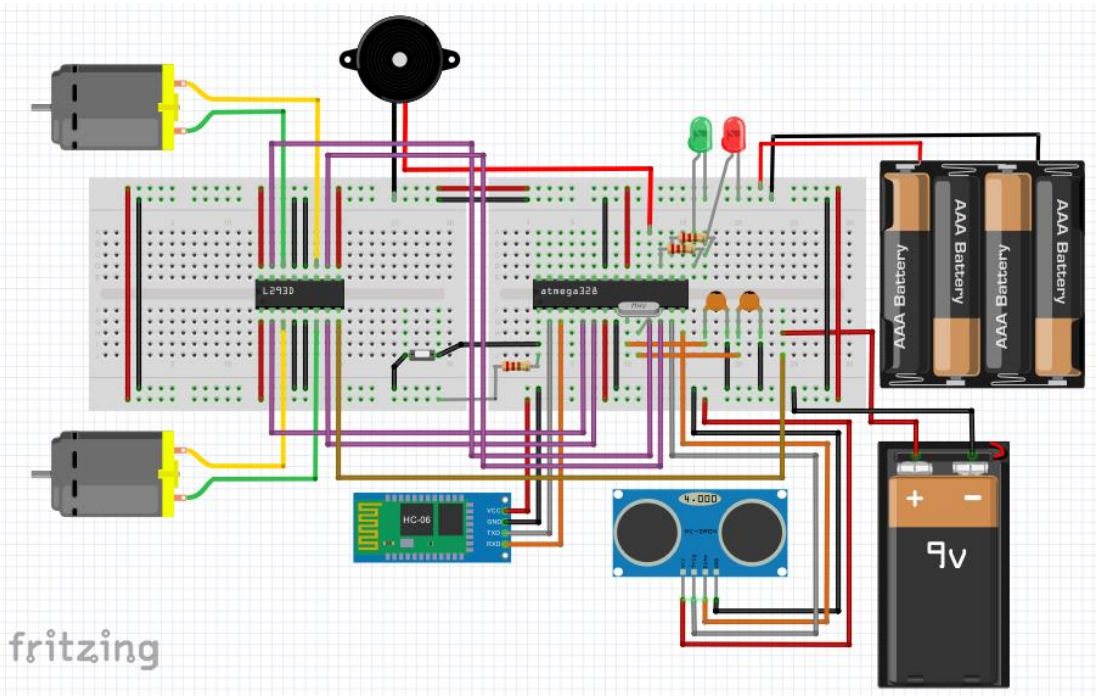
Para fixar os motores na carcaça foram utilizadas braçadeiras, uma roda castor foi acoplada como apoio na parte traseira e fixada com estruturas de madeira como é mostrado na Figura 7:

*Figura 7 - a) Roda Casto b) Braçadeiras*

As rodas dos robôs foram elaboradas a partir de dois discos de HD reciclados. Outros materiais reaproveitados foram utilizados, como elásticos que foram adaptadas como polias para girar as rodas.

Além disso, foi utilizado um esquema standalone com ATMega328, uma ponte H que utiliza o circuito integrado L293D, um módulo ultrassônico HC-SR04, um módulo de comunicação Bluetooth HC-06, uma buzina, vários LEDs e as baterias necessárias, como visto na Figura 8:

*Figura 8 - Esquema na protoboard*

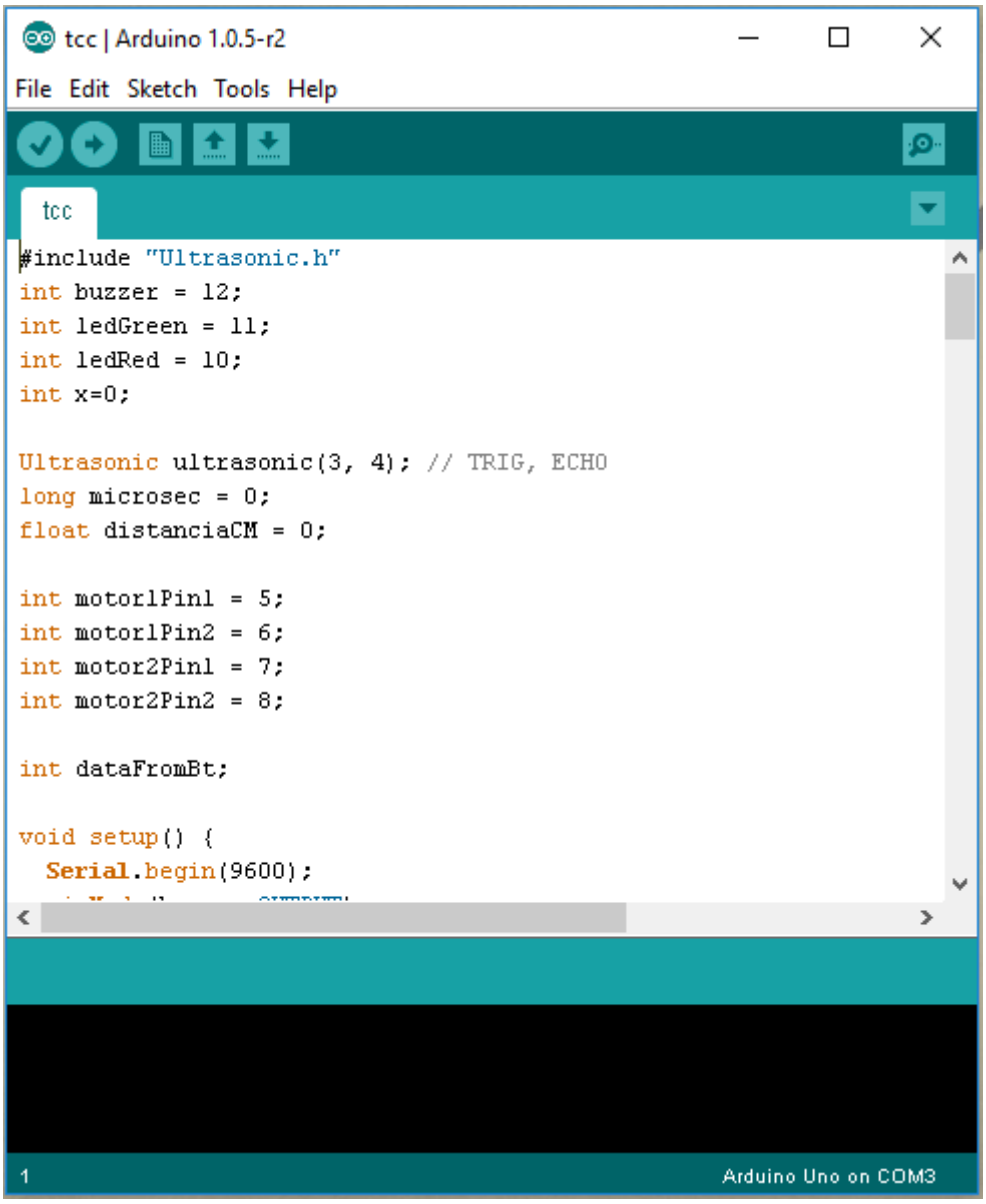



### 3.1.1. AMBIENTE DE DESENVOLVIMENTO ARDUÍNO

Segundo Schmidt (2011) Arduíno IDE é um software multiplataforma feito em Java e baseado no Processing, uma linguagem de programação com o objetivo de ensinar noções básicas em um contexto visual, e Wiring, uma plataforma de prototipagem eletrônica de hardware livre sendo de fácil manuseio. Ele é formado por um editor de texto com recursos de realce de sintaxe, parênteses/chaves correspondentes e identação automática, um compilador, um monitor de porta serial e um carregador.

A Figura 9 apresenta o layout do Arduíno IDE:

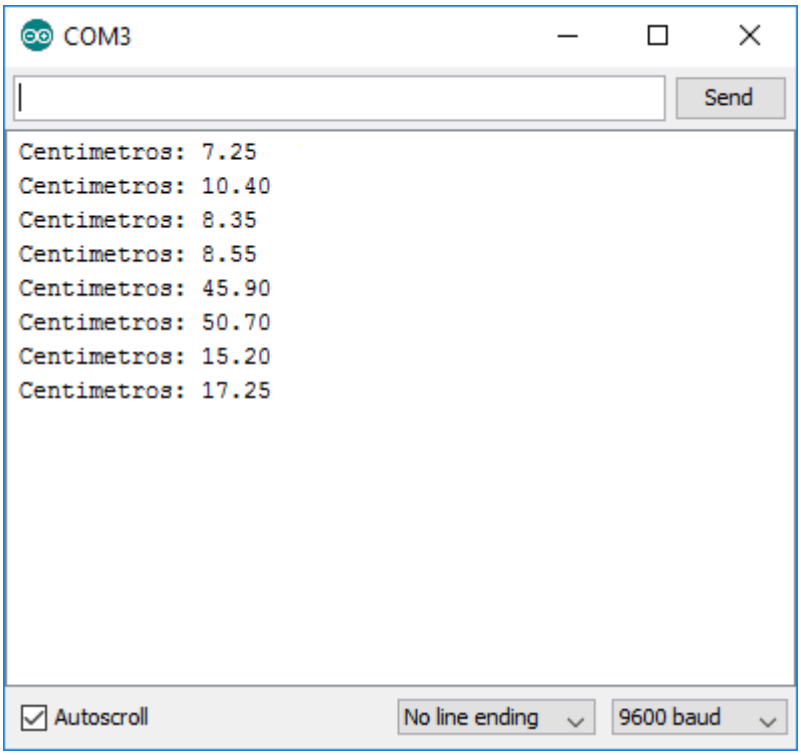
*Figura 9 - Layout do Arduino IDE*




Uma funcionalidade bastante útil desta IDE é o Monitor de Porta Serial. Essa funcionalidade permite ao Arduíno se comunicar com um computador através de uma conexão serial no qual será aberto uma janela com os valores de retorno dessa comunicação.

O monitor é mostrado na Figura 10:

*Figura 10 - Tela do Serial Monitor*




O código que controla as ações do Arduíno é feito em uma linguagem de programação chamada Wiring (similar ao C/C++) e pode ser desenvolvida no software Arduíno IDE, que controla uma interface gráfica simples e intuitiva.

O arquivo gerado deve estar no formato “.ino” e contem a declaração de duas funções básicas:

- **setup()**: Método que será executada uma única vez, assim que a placa é iniciada ou resetada. Tem como finalidade inicializar variáveis, definir os modos de entrada ou saída dos pinos, indicar bibliotecas, entre outros;
- **loop()**: Um laço infinito que trata as ações e eventos implementados dinamicamente. Utilizada para controlar de forma ativa a placa Arduíno.

### 3.1.2. COMANDOS E DECISÕES

Como mencionado, um código de Arduíno é basicamente composto de duas funções principais. No código do robô, além dos métodos básicos *setup()*, com as inicializações das variáveis e *loop()* recebendo os dados pelo módulo bluetooth e tratando no IF apropriado, temos:

- forward: Aciona os motores para girar para frente;
- backward: Aciona os motores para girar para trás e liga os LED's traseiros;
- left: Aciona o motor direito para girar para frente e aciona o motor esquerdo para trás;
- right: Aciona o motor esquerdo para girar para frente e aciona o motor direito para trás;
- stopAll(): Para todos os motores e apaga todos os LED's;
- ledOn(): Liga os LED's frontais;
- ledOff(): Desliga os LED's frontais;
- backLedOn(): Liga os LED's trazeiros;
- backLedOff(): Desiga os LED's trazeiros;
- buzz(): Aciona a buzina por 0,2 segundos;
- navegar(): cria um loop e fica realizando a leitura do sensor ultrassônico (convertidos em centímetro). Se a leitura for maior que 10, continua andando pra frente, caso contrário interrompe todos os motores, pisca os LEDs frontais, seleciona aleatoriamente o número 1 ou 2 e para chamar os métodos *left()* ou *right()* por 1,5 segundos. Após a escolha da função, interrompe tudo novamente e aciona os motores para andar para frente.

O código completo utilizado na plataforma robótica encontra-se no Anexo B.

## 3.2. PLATAFORMA DE CONTROLE

Para manipular o robô, é necessário que os comandos sejam enviados de algum controlador. Neste trabalho, foi proposto utilizar um smartphone com Android para enviar os comandos de controle através de sinais Bluetooth. O seu desenvolvimento foi utilizando as linguagens Java e XML na IDE Android Studio.

### 3.2.1. LINGUAGENS UTILIZADAS

#### 3.2.1.1. JAVA

A linguagem de programação escolhida para o desenvolvimento da aplicação de controle do robô foi o Java. O Java é uma linguagem de programação desenvolvida por James Gosling da Sun Microsystems em 1995. Um dos principais objetivos dessa linguagem é ser de fácil entendimento possibilitando que novos programadores criassem soluções desde o início da codificação. Java é orientada a objetos e interpretada, ou seja, não é compilada para código nativo, ela é compilada para um bytecode que é executado por uma máquina virtual (DEITEL e DEITEL, 2010).

Algumas vantagens chamaram atenção na escolha dessa linguagem, como a portabilidade que permite rodar o mesmo código em qualquer plataforma, sintaxe similar a C/C++, simplicidade na especificação, é distribuída com um vasto conjunto de bibliotecas (ou APIs), tem um grande suporte da comunidade e um grande acervo de materiais de estudo tanto mídias físicas, quanto digitais, gratuitos e pagos.

#### 3.2.1.2. XML

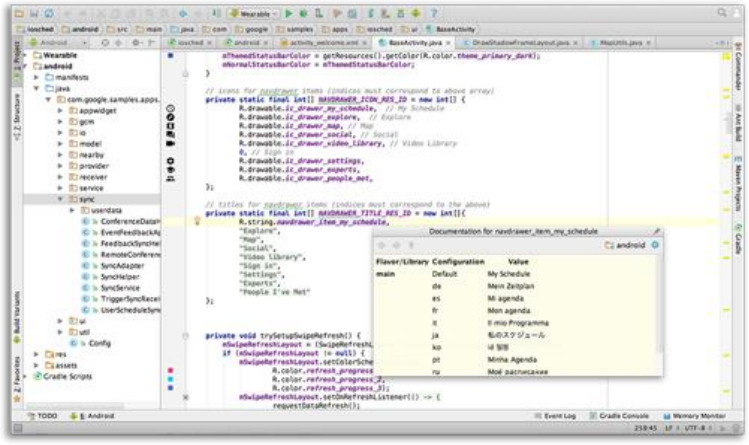
Outra linguagem utilizada nesse projeto foi o XML. Segundo (CONSORTIUM, 2015) o XML (Extensible Markup Language) é uma linguagem de marcação utilizada principalmente em documentos hierarquicamente organizados, como textos e banco de dados. O XML trabalha com o conceito de tags (rótulos) que servem como marcação de um determinado comando ou estrutura. O Android faz uso do XML para a criação da interface gráfica (tela) da aplicação.

### 3.2.2. AMBIENTE DE DESENVOLVIMENTO ANDROID

Anunciado durante a Google I/O de 2013 o Android Studio é o ambiente de desenvolvimento integrado (IDE) oferecido pela própria empresa para criação na plataforma Android. É disponibilizado gratuitamente sob a Licença Apache 2.0 que possibilita o uso, redistribuição e modificação no código, sem que haja retrospectividade (SABINO e KON, 2009).

As funcionalidades do software incluem a edição inteligente de códigos, que é capaz de realizar, refatorar e analisar códigos avançados, recursos para design de interface de usuário e análise de performance, integração com o serviço de controle de versões GitHub, e a criação de aplicações para diferentes plataformas como smartphones, tablets, TVs, relógios e carros (Android Developers, 2015). A Figura 11 apresenta o *layout* do Android Studio:

*Figura 11 - Layout do Android Studio*



*Fonte:* (Android Developers, 2015)

Uma outra opção para o desenvolvimento de aplicações Android bastante conhecida é o Eclipse IDE. Porém, como visto em (BLOG, 2015), a Google descontinuou o suporte ao Eclipse, decidindo focar na própria plataforma de desenvolvimento de aplicações. Outra vantagem do Android Studio é quanto ao recurso conhecido como drag and drop de criação de *layout*, podendo arrastar e soltar uma view na tela e acompanhado essa modificação no “preview”, tornando praticamente desnecessária a execução do emulador ou do dispositivo para visualizar o resultado.

### 3.2.3. LEVANTAMENTO DE REQUISITOS

Na realização do levantamento dos requisitos que o aplicativo deveria atender olhou-se primeiramente os materiais e equipamentos (sensores, atuadores, CI's) que já se tinha em mãos e os de fácil obtenção.

Durante o processo de desenvolvimento, diversas versões do aplicativo foram geradas para verificar a integração com o robô.

Os seguintes requisitos foram levantados:

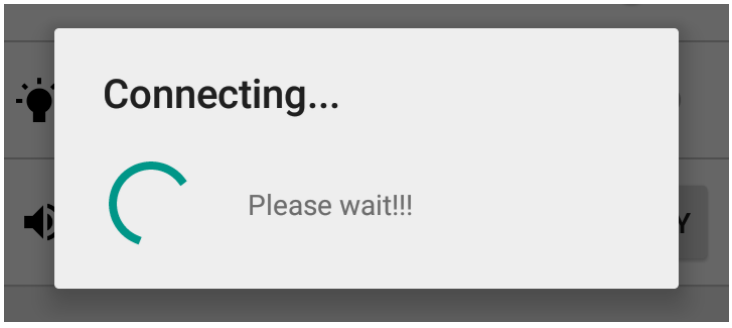
- Conexão direta com o robô;
- Permitir o controle do robô com botões direcionais ou por uso do acelerômetro;
- Acionar um modo autônomo do robô;
- Acionar LED's e buzina;

### 3.2.4. COMANDOS DE CONTROLE

O software *ROGControl* utilizou as seguintes classes para representação das suas funcionalidades: MainActivity e ConnectBT;

A classe ConnectBT tem a finalidade de realizar a conexão entre a aplicação e o modulo bluetooth do robô. Essa classe herda e reimplementa as funções da classe *AsyncTask* que auxilia a comunicação de uma thread qualquer com a UI thread (thread de modificação da interface gráfica). Gerenciando a thread de conexão com a de interface gráfica, podemos fazer o uso de um *ProgressDialog* (caixa de progresso) para indicar que a conexão está sendo realizada. A Figura 12 demonstra o uso do *ProgressDialog:*

*Figura 12 - ProgressDialog*



A classe MainActivity é a classe principal do programa e representa a única tela do aplicativo. Nela, são criados e inicializados as Views (Switchs, Buttons, TextView, ImageButtons e ImageViews), que serão acessados manipulados pela classe. Após isso, a aplicação interage com o usuário quando o mesmo clica em um botão ou switch.

Para controlar os eventos/ações de um switch é utilizado o método setOnCheckedChangeListener que pertence a classe View. O aplicativo possui 4 switchs, sendo eles e suas respectivas funções:

- Connect
    - Ativado: executando a thread de conexão da Classe ConnectBT e chama a função *setAllOn()* para habilitar todos os outros componentes.
    - Desativado: chama a função *Disconnect()* para encerrar a conexão e a função *setAllOff()* para desabilitar todos os outros componentes;

- Navigate
    - Ativado: envia para o robô o caractere “N” através da função *writeData()*, chama a função *setDirectionalOff()* para desabilitar os botões direcionais, desabilita o switch Accelerometer e emite um pop-up informativo.
    - Desativado: envia para o robô o caractere “S” através da função *writeData(),* chama a função *setDirectionalOn()* para habilitar os botões direcionais, habilita o switch Accelerometer e emite um pop-up informativo.

- Accelerometer:
    - Ativado: habilita a função *registerListener* da classe *SensorManager* que começa a registar os dados do sensor, chama a função *setDirectionalOff()* para desabilitar os botões direcionais, desabilita o switch Navigate e emite um pop-up informativo.
    - Desativado: habilita a função *unregisterListener* da classe *SensorManager* que encerra o registro dos dados do sensor, chama a função *setDirectionalOn()* para habilitar os botões direcionais, habilita o switch Navigate e emite um pop-up informativo.

- Led
    - Ativado: envia para o robô o caractere “1” através da função *writeData().*
    - Desativado: envia para o robô o caractere “0” através da função *writeData().*

Quando se instancia um botão, o método setOnClickListener que pertence a classe View é utilizado para atribuir ação. O aplicativo possui apenas o botão *Buzzer* que faz o uso deste. Caso pressionado ele envia para o robô o caractere “Z” através da função *writeData().*

Outro método utilizado nesse projeto para atribuir uma ação aos botões foi o *setOnTouchListener* que também pertence a classe View e quando utilizado, permite atribuir uma ação quando pressionar o botão e outra quando soltar. Os botões do direcional *Forward*, *Backward*, *Left* e *Right* fazem o uso desse método. Quando pressionados, envia para o robô os caracteres “F”, “B”, “L” e “R” respectivamente através da função *writeData()*.

Outra função presente na classe é a *CheckBt()* que tem a finalidade de verificar se o dispositivo tem bluetooth, caso tenha, verifica se está ativado. Ela é instanciada sempre que o aplicativo inicia ou caso o bluetooth tenha sido desativado.

Mais detalhes estão no Anexo C.

## 3.3. COMPONENTES ELETRÔNICOS

### 3.3.1. SENSOR ULTRASSÔNICO

Esse dispositivo é capaz de medir distâncias de 2cm a 4m com ótima precisão. Este módulo possui um circuito pronto com emissor e receptor acoplados e 4 pinos (VCC, Trigger, ECHO, GND). Na Figura 13 temos um módulo HC-SR04

*Figura 13 - Módulo Ultrassônico HC-SR04*


*Fonte: http://www.filipeflop.com/*

Para realizar a medição, basta manter o pino Trigger em nível logico alto por mais de 10µs. Assim o sensor emitirá uma frequência de 40 kHz e detecta se há um pulso de retorno. Neste intervalo de emissão e recebimento do sinal o pino ECHO ficará em nivel logico alto. Logo o cálculo da distância pode ser feito de acordo com o tempo em que o pino ECHO permaneceu em nível logico alto após o pino Trigger ter sido colocado em nível logico alto novamente.

$$\text{Distância} = \frac{\text{Tempo ECHO em HIGT} + \text{Velocidade do Som } (340\ \text{m/s})}{2}$$

No cálculo, a divisão por 2 deve-se ao fato que a onda é enviada e recebida, logo ela percorre 2 vezes a distância desejada.

Segundo (ELECFREAKS, 2015) as especificações estão na Tabela 1:

*Tabela 1 - Especificações do Módulo Ultrassônico HC-SR04*

| **Especificações** | |
|---|---|
| Tensão de alimentação | 5v DC |
| Corrente de Operação | 2mA |
| Frequência de Operação | 40kHz |
| Ângulo de efeito | 15º |
| Alcance | 2cm ~ 4m |
| Precisão | 3mm |

### 3.3.2. MICROCONTROLADOR ATMEGA328

Para o desenvolvimento desse trabalho utilizou-se o ATmega328 que é o microcontrolador padrão no Arduíno UNO e satisfaz as necessidades do projeto. Segundo (ATMEL, 2015) as principais características desse microcontrolador são:

- É baseado em uma arquitetura RISC avançada;
- Executa instruções em um ciclo de clock;
- Velocidade de clock de 0 a 20 MHz entre 4,5 e 5,5V;
- 32 KBytes de memória flash;
- 2 KBytes de SRAM;
- 1 KByte de EEPROM;
- 32 registradores;

### 3.3.3. PONTE H L293D

No projeto, foi utilizado o CI L293D que possui internamente 2 Ponte H e suporta uma corrente de saída de 600mA por canal, ou seja, será possível controlar até 2 motores com 600mA cada, com até 35V de alimentação. A Na Tabela 2 temos as especificações do CI (INSTRUMENTS, 2015):

*Tabela 2 - Especificações do CI L293D*

| **Especificações** | |
|---|---|
| Tensão de Operação | 4 ~ 35v |
| Controle | 2 Motores DC<br>1 Motor de Passo |
| Tensão Lógica | 5v |
| Limites de Temperatura | -20º ~135º |
| Potência Máxima | 25W |

### 3.3.4. MODULO BLUETOOTH HC-06

Este modulo permite uma comunicação wireless entre o Arduíno e outro dispositivo com interface bluetooth, como smarthphone, computador ou tablet, por exemplo. As informações recebidas pelo módulo são repassadas ao Arduíno (ou ao microcontrolador) via serial. O alcance do módulo segue o padrão da comunicação bluetooth, que é de aproximadamente 10 metros.

O HC - 06 já vem montado sobre uma placa, para facilitar o uso, que possui antena embutida e 4 pinos: VCC, GND, RX e TX, os dois últimos utilizados para comunicação serial e com tensão convertida para 5v para ser usado com o Arduíno e microcontroladores semelhantes.

Esse módulo só opera em modo slave (escravo) diferentemente do HC-05 que pode ser configurado nos modos master (mestre), slave (escravo) e loopback:

- Modo Master (Mestre): O módulo pode se conectar a outros dispositivos bluetooth.
- Modo Slave (Escravo): O módulo apenas recebe conexões de outros dispositivos bluetooth.
- Modo Loopback: O módulo recebe os dados do módulo Master e envia de volta esses mesmos dados. É um modo utilizado geralmente para testes.

Na Figura 14 temos um módulo HC-06 e na Tabela 3 as suas especificações, segundo (ELECTRONICS, 2015):

*Figura 14 - Módulo Bluetooth HC-06*

*Fonte: www.baudaeletronica.com.br*

*Tabela 3 - Especificações do Módulo HC-06*

| **Especificações** | |
|---|---|
| Tensão de Alimentação | 3,6 ~ 6v |
| Cobertura de sinal | 10m |
| Frequência | Banda de 2.4Hz |
| Potência de Transmissão | 4dBm, Classe 2 |
| Taxa máxima de transmissão serial | 1382400bps |
| Configuração padrão | 9600bps<br>Senha 1234 |

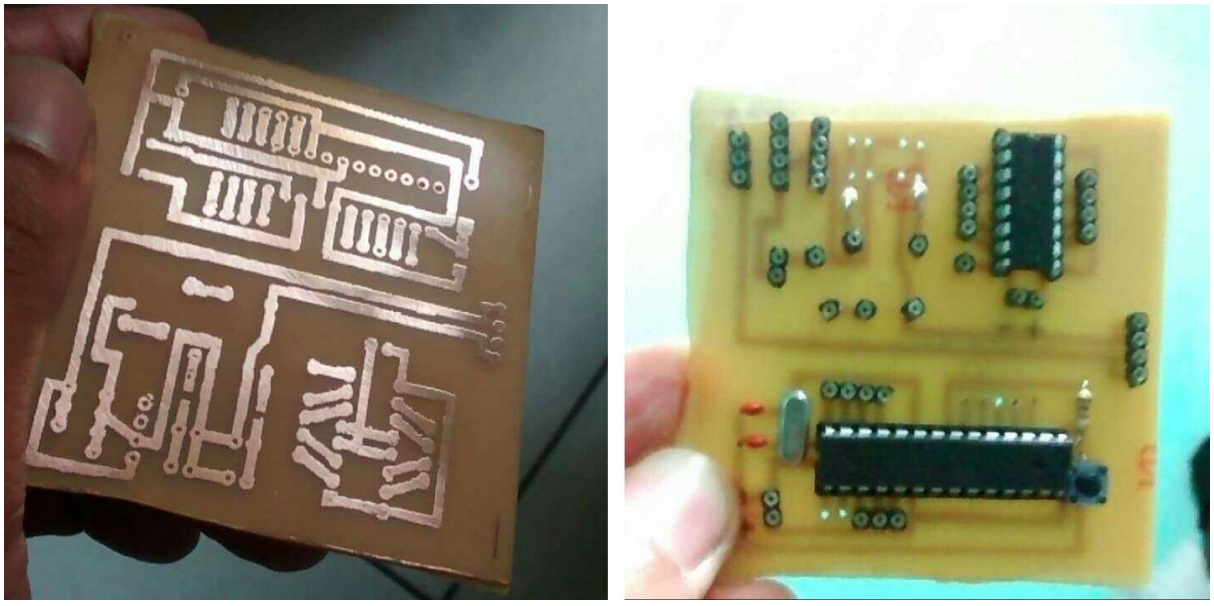
## 3.4. DESENVOLVIMENTO DA PLACA DE CIRCUITO IMPRESSO

Computadores, smartphones, brinquedos, carros, sistemas de segurança, são apenas alguns exemplos de dispositivos que são indispensáveis atualmente. E uma coisa que todos eles possuem em comum é serem formados por placas de circuito impresso.

Também conhecidas como PCB's, essa placa é formada com material isolante (fenolite ou fibra de vidro) com uma camada de material condutor (cobre, por exemplo), que visa substituir os chassis de metal que eram utilizados em aparelhos antigos para as interligações do circuito eletrônico.

O presente projeto desenvolveu uma placa de circuito para as funcionalidades da plataforma robótica seguindo e adaptando os seguintes procedimentos descritos em (COSTA, 2015). Estes procedimentos utilizam um método artesanal, pois é o mais simples e econômico de se confeccionar uma PCB, já que utiliza ferramentas de fácil obtenção. A placa desenvolvida pode ser vista nas Figuras 15 e no Anexo A o esquema utilizado.

*Figura 15 - a) Placa de Circuito Impressa b) Placa com os Componentes*



# 4. RESULTADOS E DISCUSSÃO

## 4.1. PLATAFORMA ROBÓTICA

Como resultado desse projeto, obtivemos uma plataforma robótica comandada por um aplicativo Android. Os movimentos da plataforma podem ser executados pelo usuário por comandos direcionais ou por acelerômetros do smartphone. Além disso, foi desenvolvido um modo autônomo, que movimenta a plataforma pelo ambiente, evitando colisão. Além disso foram adicionados alguns dispositivos de alerta, como um emissor sonoro e luzes para sinalizar o estado do robô. A Figura 16 mostra a plataforma desenvolvida no trabalho.

*Figura 16 - Robô R.O.G.*

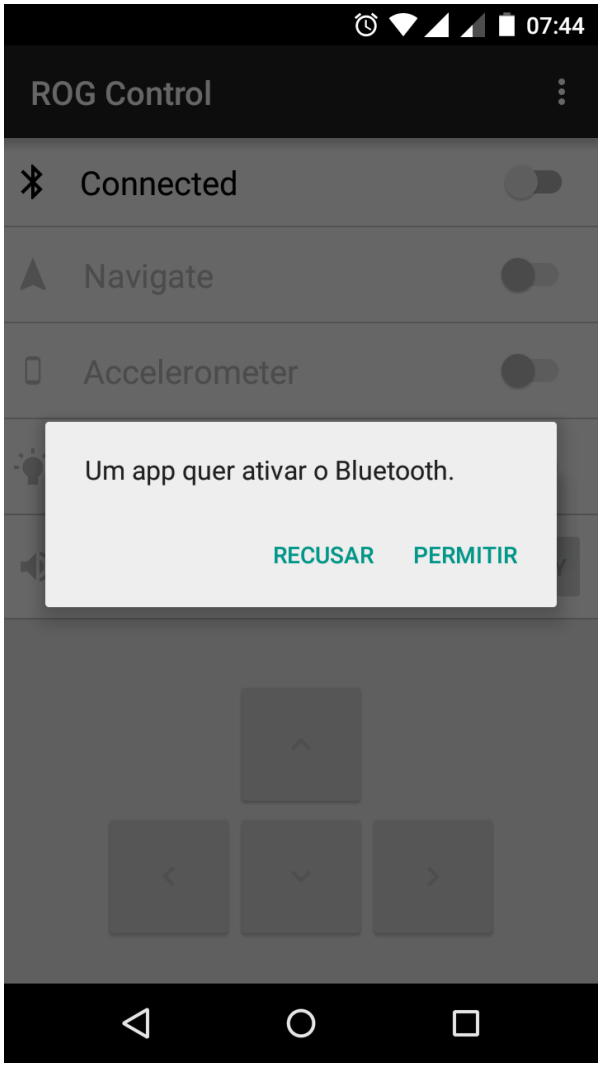
## 4.2. APLICATIVO DE CONTROLE

A plataforma é controlada pelo aplicativo desenvolvido denominado de ROGControl. Os detalhes desse aplicativo são apresentados nessa seção.

### 4.2.1. INTERFACE DE USUÁRIO

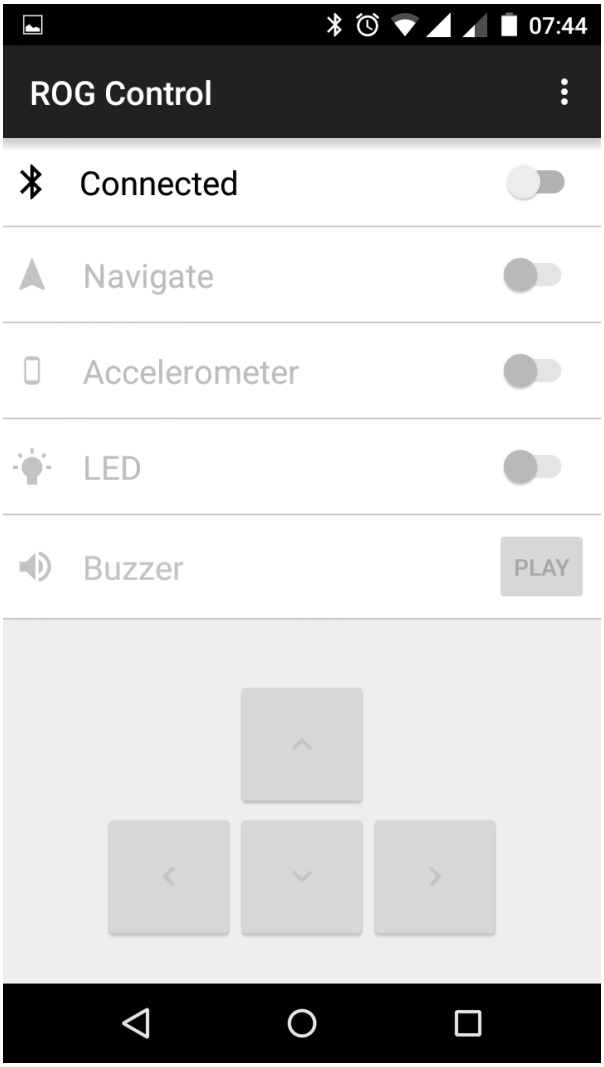
O aplicativo possui uma única tela que exerce a iteração com o usuário. Nessa tela, inicialmente, o aplicativo solicita ao usuário para permitir a ativação do bluetooth. Caso aceite, o sinal é acionado e o pop-up desaparece. A Figura 17 mostra este pop-up em funcionamento:

*Figura 17 - Pop-up de ativação Bluetooth*



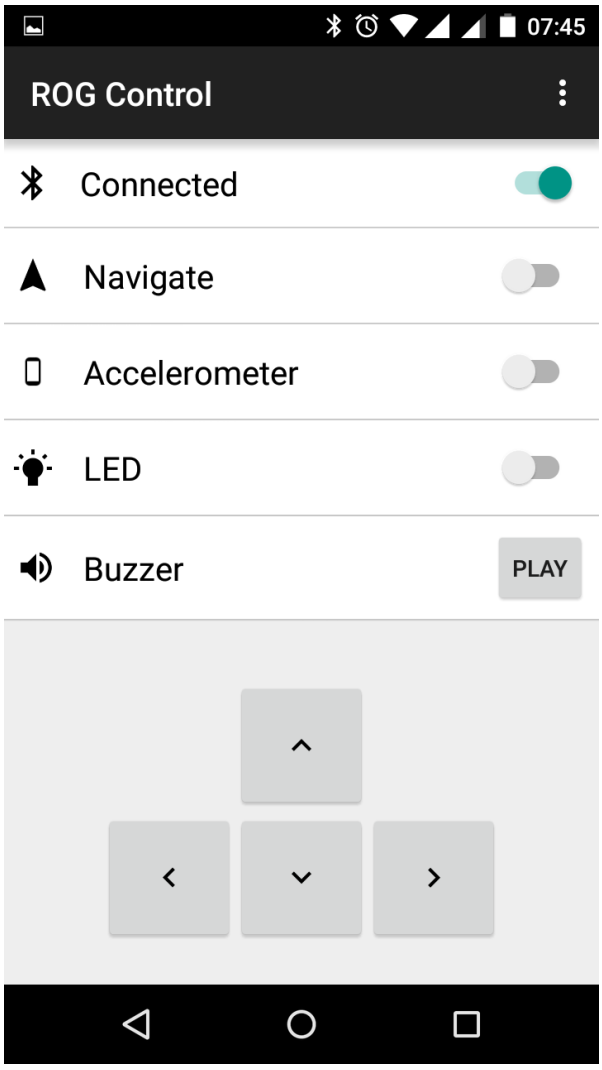
Após permitir a ativação do bluetooth, a tela apresenta as funcionalidades de controle bloqueadas, disponibilizando apenas o switch de conexão como visto na Figura 18:

*Figura 18 - Tela após a ativação do Bluetooth*



Quanto a conexão com o robô é estabelecida, como mostra a Figura 19, as interações de controle são disponibilizadas para o usuário. Dentre elas está o modo autônomo, o controle pelo acelerômetro e um direcional para o controle manual. O aplicativo também permite tocar uma buzina e acender os LED's.

*Figura 19 - Tela após a conexão com a plataforma robótica*



## 4.3. COMPARAÇÃO DE CUSTOS

Como um dos objetivos deste trabalho é a diminuição de custos, a seguir é mostrado na Tabela 4 um comparativo entre os custos do robô desenvolvido com peças recicladas e um robô montados com peças prontas. Os preços foram obtidos em lojas que fornecem no Brasil, como Solda Fria, Instituto Digital, Filipe Flop e Mercado Livre.

*Tabela 4 - Comparativo entre os custos*

| **Robô Reciclado** | | | **Robô sem reciclagem** | | |
|---|---|---|---|---|---|
| **Qnt.** | **Descrição** | **Custo** | **Qnt** | **Descrição** | **Custo** |
| **01** | Chassi + Rodas* | R$ 0,00 | 01 | Kit Chassi 2WD Robô | R$ 109,90 |
| **01** | Placa Standalone<br>• Placa de cobre<br>• Percloreto de ferro<br>• Kit Atmega<br>• Barra de pino<br><br>TOTAL | R$ 1,87<br>R$ 4,49<br>R$ 14,70<br>R$ 0,69<br>R$ 21,75 | 01 | Arduíno UNO | R$ 119,75 |
| **01** | Sensor Ultrasonico | R$ 10,80 | 01 | Sensor Ultrasonico | R$ 10,80 |
| **01** | Módulo Bluetooth | R$ 26,80 | 01 | Modulo Blueetooth | R$ 26,80 |
| **01** | Led* | R$ 0,00 | 01 | Kit Led | R$ 3,04 |
| **01** | Jumpers* | R$ 0,00 | 01 | Kit Jumpers | R$ 12,73 |
| **01** | Buzzer | R$ 1,09 | 01 | Buzzer | R$ 1,09 |
| **01** | Ponte H - L293D | R$ 6,90 | 01 | Ponte H - L293D | R$ 6,90 |
| **01** | Suporte para 4 pilhas | R$ 5,00 | 01 | Suporte para 6 pilhas | R$ 8,00 |
| **01** | Bateria | R$ 11,49 | 01 | Protoboard | R$ 14,18 |
| **01** | Elásticos* | R$ 0,00 | - | - | - |
| **01** | Braçadeiras + Parafusos | R$ 4,25 | - | - | - |
| **Total** | | **R$ 88,08** | **Total** | | **R$ 313,19** |
| ***Componentes reciclados total ou parcialmente** | | | | | |

## 4.4. DIFICULDADES E PROBLEMAS

### 4.4.1. DESVANTAGENS NO USO DO ARDUÍNO

Inicialmente foi proposto a utilização direta do Arduíno UNO na plataforma robótica. No decorrer do projeto, algumas discussões começaram a surgir, como a subutilização de uma placa com tamanho potencial. Outro fator foi o preço da mesma (cerca de R$100) que consequentemente impactaria no preço final do produto distanciando assim de um dos objetivos principais do projeto, o baixo custo na confecção do robô.

Então houve-se a necessidade de buscar alternativas de fácil manuseio, como o Arduíno, porém com um custo muito menor. Além disso o peso e o tamanho são questões importantes a serem observadas, levando ao conhecimento do Arduíno StandAlone.

### 4.4.2. CAIXA DE REDUÇÃO

Um problema que causou bastante transtorno foi na execução do movimento do robô. Procuramos, inicialmente, algo prático e de fácil construção. O mais trivial foi conectar o motor diretamente nas rodas. Contudo, não obtivemos sucesso, pois o motor não tinha o torque necessário para movimentar a roda nem o peso do próprio robô. Com isso, buscamos uma maneira de aumentar o torque do motor com algum mecanismo. A utilização de uma caixa de redução sanaria nossos problemas.

Segundo (GOMES, 2012) a caixa de redução ou caixa de velocidades é um equipamento mecânico que serve para multiplicar ou desmultiplicar a velocidade de rotação do motor permitindo transformar a potência em força ou velocidade, dependendo da necessidade.

As caixas de redução só são vendidas em conjunto com os motores o que aumentaria muito o custo do robô.. A ideia então seria de confeccionar nossa própria caixa de redução.

Ao analisarmos o funcionamento da caixa de redução, constatamos que a complexidade de manusear e fabricar engrenagens foge do escopo do trabalho, pois

necessitaria de experiência e ferramentas apropriadas, como uma furadeira de bancada, fresadora, um torno, entre outros.

Assim, a solução encontrada para o aumento do torque do motor foi utilizar um esquema de polia. De acordo com Sampaio e Calçada (2005) polia são máquinas simples utilizadas para facilitar a execução de um trabalho transferindo força e energia cinética. É constituída por um disco giratório de material rígido, metal, plástico ou madeira, lisa ou sulcada em sua periferia. Acionada por um fio, correia ou corrente metálica a polia gira em um eixo, transferindo movimento e energia a outro objeto. Se conectarmos a outra polia de diâmetro igual ou não, o trabalho pode ser equivalente ao de uma engrenagem.

### 4.4.3. ALIMENTAÇÃO DOS MOTORES

Boa parte do processo de construção do robô foi utilizado uma fonte cabeada de 12v ligada diretamente na tomada. No momento de embarcar os dispositivos surgiu a questão sobre qual seria a melhor fonte para ser acoplada na plataforma.

Em primeira instância, decidiu-se utilizar pilhas, pelo baixo preço. Para alimentar os dois motores, são necessárias 8 pilhas AA’s que aumentam excessivamente o peso do robô, reduzindo a autonomia da plataforma. A opção viável foi a utilização de uma bateria alcalina de 9V. Por ser pequena e leve, adequou-se perfeitamente ao projeto.

# 5. CONCLUSÃO

Neste trabalho foram abordados temas relevantes acerca da tecnologia utilizada para o desenvolvimento de uma plataforma móvel de sensoriamento utilizando o Arduíno. A construção do robô envolveu diferentes áreas de conhecimento, o que torna o Arduíno uma boa plataforma de aprendizado à robótica de uma maneira prática e funcional. Durante a elaboração do projeto foi necessário solucionar diversos problemas de software, elementos eletrônicos e mecânicos para atingir os objetivos propostos, dentro das restrições impostas no objetivo do trabalho.

Além do desafio de construir o robô, tem-se a dificuldade de desenvolver um sistema de controle coerente, que permita diversas formas de manipulação através de vários sensores.

O crescente uso de smartphones e o posicionamento atual do sistema operacional Android no mercado, adicionados ao grande acervo de documentação das bibliotecas e exemplos que se encontram na Internet, foi uma das motivações para sua escolha no desenvolvimento do sistema de controle.

A integração dessas plataformas e a construção do projeto foi feito de maneira gradativa e modular, realizando teste a cada adição de um novo componente, certificando de seu funcionamento correto.

Como trabalho futuro serão adicionados novos sensores na plataforma para melhorar sua percepção e movimentação, como sensores de luminosidade e câmera para registrar a trajetória do robô. Acrescentar novas maneiras de interação e comunicação, como módulos wifi permitindo o controle pela internet.

# REFERÊNCIAS


ABREU, P. Robótica Industrial: Aplicações Industriais e Robôs. **Dissertação de Mestrado em Instrumentação, Automação e Controle**, Cidade do Porto, 2002.

ALFARO, S. C. A. Robôs em Projetos Tecnológicos. **Sociedade Brasileira para o Progressoda Ciência**, Florianópolis, n. 58, Junho 2006.

ANDROID. Android Developers, 2015. Disponivel em: <http://developer.android.com/>. Acesso em: 10 Dezembro 2015.

ANDROID. Android Open Source Project, 2015. Disponivel em: <http://source.android.com/>. Acesso em: 28 Novembro 2015.

ARDUÍNO , 2015. Disponivel em: <https://www.arduino.cc/>. Acesso em: 25 Novembro 2015.

ATMEL. **ATmega328**, 2015. Disponivel em: <http://www.atmel.com/devices/atmega328.aspx>. Acesso em: 6 Dezembro 2015.

AZEVEDO, S. A. A. P. R. Minicurso: Introdução a Robótica Educacional, 2015. Disponivel em: <http://www.sbpcnet.org.br/livro/62ra/minicursos/MC%20Samuel%20Azevedo.pdf>. Acesso em: 5 Dezembro 2015.

BEGHINI, L. B. Automação Residencial de baixo custo por meio de dispositivas móveis com sistema operacional Android. **Monografia**, Universidade de São Paulo, n. 1, 2013.

BLOG, A. D. An update on Eclipse Android Developer Tools, 2015. Disponivel em: <http://android-developers.blogspot.com.br/2015/06/an-update-on-eclipse-android-developer.html>. Acesso em: 18 Dezembro 2015.

BRAGA, N. C. **Eletrônica Básica para Mecatrônica**. São Paulo: Saber, 2005.

COMPANY, T. E. P. **Development of a technology to investigate inside the Reactor Primary Containment Vessel (PCV)**. [S.l.]. 2015.

CONSORTIUM, W. W. W. Extensible Markup Language, 2015. Disponivel em: <https://www.w3.org/XML/>. Acesso em: 20 Dezembro 2015.

COSTA, A. Fazendo Placas de Circuito Impresso (PCB) através de processos caseiros, mas com aparência profissional, 2015. Acesso em: 15 Dezembro 2015.

DEITEL, P. J.; DEITEL, H. M. **Java como programar**. 8ª. ed. [S.l.]: Prentice Hall, 2010.

DENARDIN, G. W. Microcontroladores, 2015. Disponivel em: <http://www.joinville.udesc.br/portal/professores/eduardo_henrique/materiais/apostila_micro_do_Gustavo_Weber.pdf>. Acesso em: 9 Dezembro 2015.

ELECFREAKS. Ultrasonic Ranging Module HC-SR04, 2015. Disponivel em: <http://users.ece.utexas.edu/~valvano/Datasheets/HCSR04b.pdf>. Acesso em: 6 Dezembro 2015.

ELECTRONICS, C. JY-MCU Bluetooth to UART Wireless Serial Port Module, 2015. Disponivel em: <http://upcommons.upc.edu/bitstream/handle/2099.1/23666/Annex3Datasheet%20m%20dul%20Bluetooth%20JY-MCU.pdf?sequence=7>. Acesso em: 10 Dezembro 2015.

ELETRONICS, J. Mabuchi Motor RF-370CA, 2015. Acesso em: 10 Dezembro 2015.

GOMES, E. L. B.; TAVARES, L. A. Uma solução com Arduino para controlar e monitorar processos industriais. **Controle e Instrumentação**, n. 185, p. 77-79, 2013.

GOMES, L. A. Projecto de uma caixa de velocidades close ratio para um automóvel de competição. **Dissertação de Mestrado em Integrado em Engenharia Mecânica**, Porto, 2012.

HISTÓRIA da Robótica, 2015. Disponivel em: <http://www.citi.pt/educacao_final/trab_final_inteligencia_artificial/historia_da_robotica.html>. Acesso em: 6 Dezembro 2015.

INSTRUMENTS, T. L293D Quadruple Half-H Drivers, 2015. Disponivel em: <http://users.ece.utexas.edu/~valvano/Datasheets/L293d.pdf>. Acesso em: 9 Dexembro 2015.

INSTRUMENTS, T. The Chip that Jack Built, 2015. Disponivel em: <http://www.ti.com/corp/docs/kilbyctr/jackbuilt.shtml>. Acesso em: 9 Dezembro 2015.

IROBOT. Roomba Vacuum Cleaning Robot, 2002. Disponivel em: <http://www.irobot.com/For-the-Home/Vacuum-Cleaning/Roomba.aspx>. Acesso em: 10 Dezembro 2015.

LECHETA, R. R. **Google Android:** aprenda a criar aplicações para dispositivos móveis com o Android SDK. 1ª. ed. [S.l.]: Nvatec, 2010.

MERCEDES-BENZ, T. The Mercedes-Benz F 015 Luxury in Motion, 2015. Disponivel em: <https://www.mercedes-benz.com/en/mercedes-benz/innovation/research-vehicle-f-015-luxury-in-motion/>. Acesso em: 10 Dezembro 2015.

MONK, S. **Programação com Arduino:** Começando com Sketches. [S.l.]: Bookman, 2013.

MOREIRA, A. S.; M.PORTELA, A.; SILVA, R. Uso da plataforma Arduíno no desenvolvimento de soluções tecnológicas para pesquisas de dados atmosféricos na Amazônia. **Perspectiva Amazônica**, v. 3, n. 5, p. 119-126, 2013.

NASA. Mars Exploration Rovers Mission, 2015. Disponivel em: <http://mars.nasa.gov/mer/home/>. Acesso em: 04 Dezembro 2015.

NATALIE, K. **Scientific American Brasil:** Exploradores do Futuro - Isaac Asimov. [S.l.]: Duetto, 2005.

OXFORD Dictionaries, 2015. Disponivel em: <http://oxforddictionaries.com/>. Acesso em: 5 Dezembro 2015.

PLACA Standalone, 2015. Disponivel em: <http://www.placastandalone.com.br/>. Acesso em: 9 Dezembro 2015.

SABINO, V.; KON, F. **Licenças de Software Livre, História e Características.** Instituto de Matemática e Estatística da USP. São Paulo. 2009.

SAMPAIO, J. L.; CALÇADA, C. S. **Física**. 2ª. ed. São Paulo: Atual, 2005.

SCHMIDT, M. **Arduino:** a quick-start guide. [S.l.]: Pragmatic Bookshelf, 2011.

SIEGWART, R.; NOURBAKHSH, I. R.; SCARAMUZZA, D. **Introduction to autonomous mobile robots**. 2ª. ed. Cambridge: MIT Press, 2004.

SILVA, J. R. N. D. Lixo eletrônico: um estudo de responsabilidade ambiental no contexto do instituto de educação ciência e tecnologia do Amazonas. **I Congresso Brasileiro de Gestão Ambiental**, novembro 2010.

WOLF, D. F. et al. Robótica Móvel Inteligente: Da Simulação às Aplicações no Mundo Real. **Jornada de Atualização em Informática**, Bento Gonçalves, junho 2009.

XMOBOTS , 2015. Disponivel em: <http://www.xmobots.com/>. Acesso em: 05 Dezembro 2015.

# ANEXO A – IMAGENS PARA CONFECÇÃO DA PCB

*Figura 20 - Esquema gerado pelo software Fritzing*

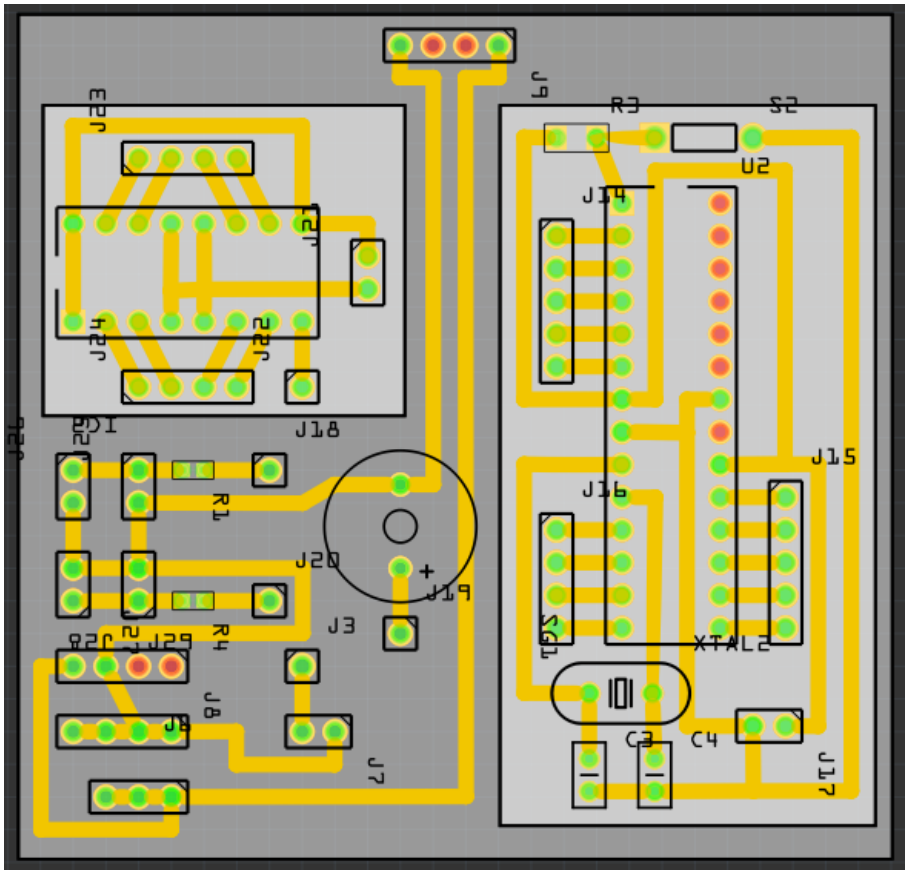
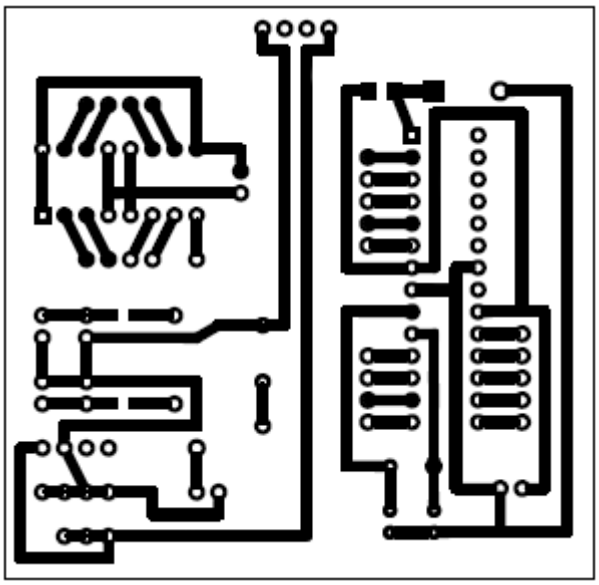



*Figura 21 - Figura utilizada na impressão da placa*



# ANEXO B – CÓDIGO FONTE DO ROBÔ

```
#include "Ultrasonic.h"
int buzzer = 12;
int ledGreen = 11;
int ledRed = 10;
int randNumber;
int x=0;

Ultrasonic ultrasonic(3, 4); // TRIG, ECHO
long microsec = 0;
float distanciaCM = 0;

int motor1Pin1 = 5;
int motor1Pin2 = 6;
int motor2Pin1 = 7;
int motor2Pin2 = 8;

int dataFromBt;

void setup() {
   Serial.begin(9600);
   randomSeed(analogRead(0));
   pinMode(buzzer,OUTPUT);
   pinMode(ledGreen, OUTPUT);
   pinMode(ledRed, OUTPUT);
   pinMode(motor1Pin1, OUTPUT);
   pinMode(motor1Pin2, OUTPUT);
   pinMode(motor2Pin1, OUTPUT);
   pinMode(motor2Pin2, OUTPUT);
}

void foward(){
   Serial.println("forward");
   digitalWrite(motor1Pin1, LOW);
   digitalWrite(motor1Pin2, HIGH);
   digitalWrite(motor2Pin1, LOW);
   digitalWrite(motor2Pin2, HIGH);
}
```

```
void backward(){
    Serial.println("backward");
    digitalWrite(motor1Pin1, HIGH);
    digitalWrite(motor1Pin2, LOW);
    digitalWrite(motor2Pin1, HIGH);
    digitalWrite(motor2Pin2, LOW);
    backLedOn();
}

void left(){
    Serial.println("left");
    digitalWrite(motor1Pin1, LOW);
    digitalWrite(motor1Pin2, HIGH);
    digitalWrite(motor2Pin1, LOW);
    digitalWrite(motor2Pin2, LOW);
}

void right(){
    Serial.println("right");
    digitalWrite(motor1Pin1, LOW);
    digitalWrite(motor1Pin2, LOW);
    digitalWrite(motor2Pin1, LOW);
    digitalWrite(motor2Pin2, HIGH);
}

void stopAll(){
    Serial.println("stop");
    digitalWrite(motor1Pin1, LOW);
    digitalWrite(motor1Pin2, LOW);
    digitalWrite(motor2Pin1, LOW);
    digitalWrite(motor2Pin2, LOW);

    backLedOff();
    ledOff();
}

void ledOn(){
    Serial.println("led on");
    digitalWrite(ledGreen, HIGH);
}

void ledOff(){
    Serial.println("led off");
    digitalWrite(ledGreen, LOW);
}
```

```
void backLedOn(){
   Serial.println("led on");
   digitalWrite(ledRed, HIGH);
}

void backLedOff(){
   Serial.println("led off");
   digitalWrite(ledRed, LOW);
}

void buzz(){
   Serial.println("buzzer");
   digitalWrite(buzzer, HIGH);
   delay(200);
   digitalWrite(buzzer, LOW);
}

void navegar(){
   do{
      dataFromBt = Serial.read();
      randNumber = random(1, 4);
      if (dataFromBt == 'S'){
         x=0;
         stopAll();
         ledOff();
      }
      else{
         microsec = ultrasonic.timing();  //Lendo o sensor
         distanciaCM = ultrasonic.convert(microsec, Ultrasonic::CM);  //Convertendo a distância em CM
         if (distanciaCM <= 10) {
            ledOn();
            stopAll();
            if (randNumber <= 2)  left();
            if (randNumber >= 4)  right();
            delay(2000);
            stopAll();
            ledOff();
            foward();
         }
         if (distanciaCM > 10)  foward();
      }
   }while( x==1 );
}
```

```
void loop() {
    dataFromBt = Serial.read();
    if(dataFromBt == 'N'){
        x=1;
        navegar();
    }

    if(dataFromBt == '1') ledOn();
    if(dataFromBt == '0') ledOff();
    if(dataFromBt == 'F') foward();
    if(dataFromBt == 'B') backward();
    if(dataFromBt == 'L') left();
    if(dataFromBt == 'R') right();
    if(dataFromBt == 'S') stopAll();
    if(dataFromBt == 'Z') buzz();
}
```

# ANEXO C – CÓDIGO FONTE DO APLICATIVO

```
import android.app.AlertDialog;
import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.Color;
import android.graphics.PorterDuff;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.view.MotionEvent;
import android.view.View.OnClickListener;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.Switch;
import android.widget.TextView;
import android.widget.Toast;

import java.io.IOException;
import java.io.OutputStream;
import java.util.UUID;

public class MainActivity extends AppCompatActivity implements SensorEventListener {

    private AlertDialog alerta;
    ProgressDialog progress;
    ImageButton Forward, Backward, Left, Right;
    Button Buzzer;
    TextView BuzzerText;
    Switch Connect, Navigate, Led, Acc;
    ImageView Bt_icon, Nav_icon, Led_icon, Buz_icon, Acc_icon;
```

```
private SensorManager mSensorManager;
private Sensor mAccelerometer;
boolean statusConection = false;

private String dataToSend;
private BluetoothAdapter mBluetoothAdapter = null;
private BluetoothSocket btSocket = null;
private OutputStream outStream = null;
private static String address = "98:D3:31:B4:35:0E";
private static final UUID MY_UUID = UUID
        .fromString("00001101-0000-1000-8000-00805F9B34FB");

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

    Connect = (Switch) findViewById(R.id.connect);
    Navigate = (Switch) findViewById(R.id.navigate);
    Led = (Switch) findViewById(R.id.led);
    Acc = (Switch) findViewById(R.id.acc);
    Buzzer = (Button) findViewById(R.id.buzzer);
    BuzzerText = (TextView) findViewById(R.id.textView);

    Forward = (ImageButton) findViewById(R.id.buttonForward);
    Backward = (ImageButton) findViewById(R.id.buttonBackward);
    Left = (ImageButton) findViewById(R.id.buttonLeft);
    Right = (ImageButton) findViewById(R.id.buttonRight);

    Bt_icon = (ImageView) findViewById(R.id.bt_icon);
    Nav_icon = (ImageView) findViewById(R.id.nav_icon);
    Led_icon = (ImageView) findViewById(R.id.led_icon);
    Acc_icon = (ImageView) findViewById(R.id.acc_icon);
    Buz_icon = (ImageView) findViewById(R.id.buz_icon);

    mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

    setAllOff();
```

```
Connect.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            new ConnectBT().execute();
            statusConection = true;
            setAllOn();
        } else {
            Disconnect();
            setAllOff();
        }
    }
});

Navigate.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            dataToSend = "N";
            writeData(dataToSend);
            Toast.makeText(getApplicationContext(),
                    "Auto mode activated!", Toast.LENGTH_SHORT).show();
            Acc.setEnabled(false);
            setDirectionalOff();
        }
        else {
            dataToSend = "S";
            writeData(dataToSend);
            Toast.makeText(getApplicationContext(),
                    "Auto mode disabled", Toast.LENGTH_SHORT).show();
            Acc.setEnabled(true);
            setDirectionalOn();
        }
    }
});

Led.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            dataToSend = "1";
            writeData(dataToSend);
        } else {
            dataToSend = "0";
            writeData(dataToSend);
        }
    }
});
```

```
Buzzer.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        dataToSend = "Z";
        writeData(dataToSend);
    }
});

Forward.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                dataToSend = "F";
                writeData(dataToSend);
                break;
            case MotionEvent.ACTION_UP:
                dataToSend = "S";
                writeData(dataToSend);
                break;
        }
        return false;
    }
});

Backward.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                dataToSend = "B";
                writeData(dataToSend);
                break;
            case MotionEvent.ACTION_UP:
                dataToSend = "S";
                writeData(dataToSend);
                break;
        }
        return false;
    }
});
```

```
    Left.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN:
                    dataToSend = "L";
                    writeData(dataToSend);
                    break;
                case MotionEvent.ACTION_UP:
                    dataToSend = "S";
                    writeData(dataToSend);
                    break;
            }
            return false;
        }
    });
    Right.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN:
                    dataToSend = "R";
                    writeData(dataToSend);
                    break;
                case MotionEvent.ACTION_UP:
                    dataToSend = "S";
                    writeData(dataToSend);
                    break;
            }
            return false;
        }
    });
}

public void Accelerometer(View view) {
    boolean on = ((Switch) view).isChecked();
    if (on) {
        mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_UI);
        setDirectionalOff();
        Navigate.setEnabled(false);
        Toast.makeText(getApplicationContext(),
                "Accelerometer activated!", Toast.LENGTH_SHORT).show();
    }
```

```
    else {
        mSensorManager.unregisterListener(this);
        Navigate.setEnabled(true);
        setDirectionalOn();
        Toast.makeText(getApplicationContext(),
                "Accelerometer disabled!", Toast.LENGTH_SHORT).show();
    }
}

public void setAllOff(){
    Connect.setChecked(false);

    Navigate.setEnabled(false);
    Led.setEnabled(false);
    Acc.setEnabled(false);
    Buzzer.setEnabled(false);
    BuzzerText.setTextColor(Color.parseColor("#C3C3C3"));
    Nav_icon.setColorFilter(0xffC3C3C3, PorterDuff.Mode.SRC_IN);
    Led_icon.setColorFilter(0xffC3C3C3, PorterDuff.Mode.SRC_IN);
    Acc_icon.setColorFilter(0xffC3C3C3, PorterDuff.Mode.SRC_IN);
    Buz_icon.setColorFilter(0xffC3C3C3, PorterDuff.Mode.SRC_IN);

    setDirectionalOff();
}

public void setAllOn() {
    Connect.setChecked(true);

    Navigate.setEnabled(true);
    Led.setEnabled(true);
    Acc.setEnabled(true);
    Buzzer.setEnabled(true);
    BuzzerText.setTextColor(Color.parseColor("#000000"));
    Nav_icon.setColorFilter(0xff000000, PorterDuff.Mode.SRC_IN);
    Led_icon.setColorFilter(0xff000000, PorterDuff.Mode.SRC_IN);
    Acc_icon.setColorFilter(0xff000000, PorterDuff.Mode.SRC_IN);
    Buz_icon.setColorFilter(0xff000000, PorterDuff.Mode.SRC_IN);

    setDirectionalOn();
}
```

```
public void setDirectionalOff(){
    Forward.setEnabled(false);
    Backward.setEnabled(false);
    Left.setEnabled(false);
    Right.setEnabled(false);

    Forward.setColorFilter(0xffC3C3C3, PorterDuff.Mode.SRC_IN);
    Backward.setColorFilter(0xffC3C3C3, PorterDuff.Mode.SRC_IN);
    Left.setColorFilter(0xffC3C3C3, PorterDuff.Mode.SRC_IN);
    Right.setColorFilter(0xffC3C3C3, PorterDuff.Mode.SRC_IN);
}

public void setDirectionalOn(){
    Forward.setEnabled(true);
    Backward.setEnabled(true);
    Left.setEnabled(true);
    Right.setEnabled(true);

    Forward.setColorFilter(0xff000000, PorterDuff.Mode.SRC_IN);
    Backward.setColorFilter(0xff000000, PorterDuff.Mode.SRC_IN);
    Left.setColorFilter(0xff000000, PorterDuff.Mode.SRC_IN);
    Right.setColorFilter(0xff000000, PorterDuff.Mode.SRC_IN);
}

protected void CheckBt() {
    mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

    if (mBluetoothAdapter == null) {
        Toast.makeText(getApplicationContext(),
                "Bluetooth null!", Toast.LENGTH_SHORT).show();
    }

    if (!mBluetoothAdapter.isEnabled()) {
        Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableBtIntent, 1);
    }
}

protected void Disconnect(){
    if (outStream != null) {
        try {
            outStream.close();
        }
        catch (Exception e) {}
        outStream = null;
    }
```

```
    if (btSocket != null) {
        try {
            btSocket.close();
        }
        catch (Exception e) {}
        btSocket = null;
    }

    statusConection = false;
}

protected void writeData(String data) {
    try {
        outStream = btSocket.getOutputStream();
    } catch (IOException e) {
        Toast.makeText(getApplicationContext(), "Bug before sending message",
                Toast.LENGTH_SHORT).show();
    }

    String message = data;
    byte[] msgBuffer = message.getBytes();

    try {
        outStream.write(msgBuffer);
    } catch (IOException e) {
        Toast.makeText(getApplicationContext(), "Bug while sending message",
                Toast.LENGTH_SHORT).show();
    }
}

@Override
protected void onDestroy() {
    super.onDestroy();
    Disconnect();
}

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

```
@Override
public void onStop() {
    super.onStop();

    if (Acc.isChecked()) {
        Acc.setChecked(false);
        Navigate.setEnabled(true);
        setDirectionalOn();
    }

    if (Navigate.isChecked()) {
        Navigate.setChecked(false);
        Acc.setEnabled(true);
        setDirectionalOn();
    }

    mSensorManager.unregisterListener(this);

    if (statusConection) {
        dataToSend = "S";
        writeData(dataToSend);
    }
}

@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) {
    int id = item.getItemId();

    if (id == R.id.action_about) {

        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle("About");
        builder.setMessage("App by Alvaro Marinho - alvaro.marinho@live.com");
        builder.setPositiveButton("Ok", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                alerta.dismiss();
            }
        });
        alerta = builder.create();
        alerta.show();
    }
    return super.onOptionsItemSelected(item);
}

@Override
public void onSensorChanged(SensorEvent event) {

    Float x = event.values[0];
    Float y = event.values[1];
    Float z = event.values[2];

    if( x > -3 && x < 3 && z > -4 && z < 4) {        // STOP
        dataToSend = "S";
        writeData(dataToSend);
    }

    if (z > 5 && z <= 9) { // FORWARD
        dataToSend = "F";
        writeData(dataToSend);
    }

    if (z < 0 && z >= -9) { // BACKWARD
        dataToSend = "B";
        writeData(dataToSend);
    }

    if (x > 3 && x <= 9) { // LEFT
        dataToSend = "L";
        writeData(dataToSend);
    }
```

```
        if (x < -2 && x >= -9) { // RIGHT
            dataToSend = "R";
            writeData(dataToSend);
        }
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }

    private class ConnectBT extends AsyncTask<Void, Void, Void> {

        private boolean ConnectSuccess = true; //if it's here, it's almost connected

        @Override
        protected void onPreExecute() {
            progress = ProgressDialog.show(MainActivity.this, "Connecting...", "Please wait!!!");  //show a
progress dialog
        }

        @Override
        protected Void doInBackground(Void... params) {
            BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
            mBluetoothAdapter.cancelDiscovery();
            try {
                btSocket = device.createRfcommSocketToServiceRecord(MY_UUID);
                btSocket.connect();
            }
            catch (IOException e) {
                ConnectSuccess = false;
            }
            return null;
        }
```

```
        @Override
        protected void onPostExecute(Void result) {
            super.onPostExecute(result);

            if (!ConnectSuccess) {
                Toast.makeText(getApplicationContext(),"Connection Failed! Try again.",
Toast.LENGTH_SHORT).show();
                setAllOff();
            }
            else {
                Toast.makeText(getApplicationContext(), "Connected.", Toast.LENGTH_SHORT).show();
            }
            progress.dismiss();
        }
    }
}
```