Universidade de São Paulo

Escola de Engenharia de São Carlos

Departamento de Engenharia Elétrica

Desenvolvimento de um sistema de comando wireless baseado em Android

Aluno:

Ricardo Marques Ferretti da Costa (nºUSP:6911341)

Orientador:

Evandro L. L. Rodrigues

São Carlos

Novembro de 2014

Ricardo Marques Ferretti da Costa

Desenvolvimento de um sistema de comando wireless baseado em Android

Trabalho de Conclusão de Curso
Apresentado à Escola de Engenharia de São Carlos
Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em Sistemas de Energia e Automação
ORIENTADOR: Evandro Luis Linhari Rodrigues

São Carlos
2014

C837d

Costa, Ricardo Marques Ferretti da
Desenvolvimento de um sistema de comando wireless baseado em Android / Ricardo Marques Ferretti da Costa; orientador Evandro Luis Linhari Rodrigues. São Carlos, 2014.

Monografia (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2014.

1. ARM. 2. Android. 3. Bluetooth. 4. Comando de Robô. 5. Comando Wireless. I. Título.

FOLHA DE APROVAÇÃO

Nome: Ricardo Marques Ferretti da Costa

Título: "Controle de Wireless de robô via dispositivo móvel com sistema operacional Android"

Trabalho de Conclusão de Curso defendido e aprovado em 27/11/2014,

com NOTA 9,0 (NOVE, ZERO), pela Comissão Julgadora:

Prof. Associado Evandro Luís Linhari Rodrigues - (Orientador - SEL/EESC/USP)

Prof. Dr. Marcelo Andrade da Costa Vieira - (SEL/EESC/USP)

Prof. Dr. Valdir Grassi Júnior - (SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

Dedicatória

Dedico este trabalho a meus pais e minha irmã, que sempre me apoiaram durante os momentos de dúvida, e que sempre estiverem comigo durante esta jornada que foi a graduação.

Agradecimentos

Gostaria de agradecer primeiramente à minha família e amigos por sempre se mostrarem presentes e me ajudando sempre que possível.

Gostaria também de agradecer a todos os professores que estiverem presentes durante minha graduação e contribuíram para minha formação. Em especial a meu orientador Evandro, que me apoiou a seguir um ano de intercâmbio, que semeou a idéia deste trabalho.

Resumo

Este trabalho teve como objetivo desenvolver uma forma de se obter controle wireless de pequenos motores de baixa tensão utilizando-se um *smartphone* com sistema operacional Android. Para que este objetivo fosse alcançado, foi necessário desenvolvimento em conjunto tanto de software como de hardware. Em questão de hardware, utilizou-se um microcontrolador ARM LPC1114 da família Cortex-M0, uma PCB como driver para os motores, um módulo Bluetooth conectado ao microcontrolador responsável pela comunicação wireless, e um *smartphone* com sensor acelerômetro e comunicação Bluetooth. O celular torna-se cada vez mais presente na vida pessoal, e a utilização do sensor acelerômetro se justifica pelo intuito de colocar o robô em movimento através do posicionamento do celular como um volante. Em relação ao software, foram utilizadas as linguagens de programação C para o microcontrolador, Java para o dispositivo móvel Android, além do software Eagle CadSoft para projeto do driver dos motores. Os resultados alcançados demonstram que a combinação *smartphone* e microcontrolador não apenas funciona, como também abre caminho para outras soluções que envolvam esses hardwares.

Palavras-chave: ARM, Android, Bluetooth, Comando de Robô, Comando Wireless

Abstract

This final work had as its main goal to develop a way to obtain wireless control of small low voltage motors using an Android smartphone. In order to achieve this goal, it was developed both software (microcontroller and Android) and hardware (PCB for handling motors) together. In relation to the hardware, it was used an ARM LPC1114 microcontroller of the Cortex-M0 family, a PCB as a driver to the motors, a Bluetooth module used for wireless communication, and an Android smartphone with accelerometer and Bluetooth communication capabilities. On the software side, it was used C language for the ARM microcontroller, Java language for the Android device, and the Eagle CadSoft for the development of the PCB. The results obtained show that this microcontroller and smartphone combination can not only be achieved, but also opens new ways for solutions that incorporate these hardwares.

Keywords: ARM, Android, Bluetooth, Wireless Command, Robot Command

Lista de Figuras

Lista de Abreviaturas e Siglas

API	*Application Programming Interface*
ARM	*Advanced RISC Machine*
CI	*Circuito Integrado*
DC	*Direct Current*
FIFO	*First In First Out*
GPIO	*General Purpose Input Output*
GPS	*Global Positioning System*
GSM	*Global System for Mobile Communications*
IDE	*Integrated Development Environment*
OS	*Operating System*
PCB	*Printed Circuit Board*
PWM	*Pulse Width Modulation*
RAM	*Random Access Memory*
RISC	*Reduced Instruction Set Computer*
SDK	*Software Development Kit*
UART	*Universal Asynchronous Receiver Transmitter*
UHF	*Ultra High Frequency*

Sumário

1. Introdução

Nas últimas décadas, presenciou-se um forte avanço tecnológico nas áreas de sistemas eletrônicos e embarcados, que se tornam cada vez mais úteis e presentes na vida moderna. O intuito deste trabalho é de investigar o funcionamento de sistemas microprocessados combinados a um dispositivo cada vez mais presente nestes últimos anos, o celular, hoje denominado *smartphone.*

Desde a invenção do primeiro microprocessador, o Intel 4004 em 1971, nota-se o quão presentes estes se tornaram atualmente. Seja em eletrodomésticos, automóveis, sistemas de segurança ou telecomunicações, os microcontroladores estão sendo largamente utilizados para tornar bens de consumo, serviços, ou processos industriais mais inteligentes, mais robustos, e mais confiáveis. Além do mais, de acordo com a Grand View Research (GRANDVIEWRESEARCH, 2014), uma empresa de pesquisa na área de mercados, “o mercado global de microcontroladores foi estimado em 17.393,6 milhões de unidades em 2013, e é esperado alcançar 39 milhões em 2020”, o que mostra que esse segmento é promissor para o futuro. Com isso, percebe-se a importância que deve ser atribuída a este tipo de hardware em específico, e o quão valiosa é a pesquisa acerca de soluções que o envolvam.

Outro segmento de novas tecnologias que vem crescendo consideravelmente é o de *smartphones*. O *smartphone* trata-se de um telefone celular móvel com maior poder computacional e conectividade do que um telefone celular básico. Entre os recursos mais utilizados estão acesso à internet, câmera digital, GPS, Wi-Fi etc.

Para gerenciar tantas funcionalidades, esses *smartphone*s possuem um sistema operacional. Dentre os mais presentes, estão Android, iOS, Windows Phone e BlackBerry. Uma pesquisa realizada pela *International Data Corportation,* líder global em pesquisa de inteligência de mercado na área de TI, revelou dados importantes sobre o mercado de sistemas operacionais para *smartphones*. A Tabela 1 mostra as respectivas fatias de mercado de sistemas operacionais para

smartphone, e seus crescimentos, onde nota-se que o Android é atualmente o mais utilizado e com o maior crescimento de 2013 para 2014.

Sistemas operacionais mais utilizados no mundo, Unidades Vendidas, e Fatias de Mercado (em milhões)

	Segundo Trimestre de 2014		**Segundo Trimestre de 2013**		
Sistema Operacional	Volume de Vendas	Fatia de Mercado	Volume de Vendas	Fatia de Mercado	Crescimento 2013/2014
Android	255,3	84,7%	191,5	79,6%	33,3%
iOS	35,2	11,7%	31,2	13,0%	12,7%
Windows Phone	7,4	2,5%	8,2	3,4%	-9,4%
BalckBerry	1,5	0,5%	6,7	2,8%	-78,0%
Others	1,9	0,6%	2,9	1,2%	-32,2%
Total	301,3	100,0%	240,5	100,0%	25,3%

Tabela 1: Fatias de Mercado dos principais sistemas operacionais para *smartphone*. Baseado na notícia retirado de (IDC, 2014)

Já a Figura 1 contém um gráfico que mostra a quantidade de dispositivos eletrônicos fabricados globalmente por sistema operacional, onde nota-se um domínio do sistema Android.

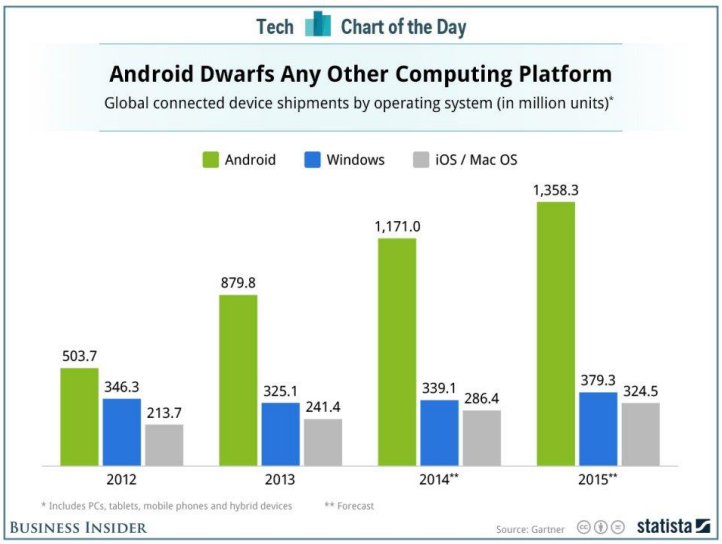


Figura 1: Dispositivos Eletrônicos fabricados globalmente por sistema operacional

Fonte: (http://www.businessinsider.com/androids-share-of-the-computing-market-2014-3#ixzz3HO0X0Tlt)

O Android foi desenvolvido pela Google, uma das maiores empresas de software do mundo. Este OS (*Operating System*, do inglês, sistema operacional) também não está atrelado a um único fabricante de *smartphone*s, o que permite que o consumidor final possa escolher entre as várias opções de celular em que este sistema está inserido. Outra característica é que o Android é um sistema de código aberto (também mencionado como *opensource*), o que acarreta em programadores e desenvolvedores do mundo inteiro contribuindo para o desenvolvimento deste sistema, além de gerar grande troca de informações e soluções, possibilitando um sistema cada vez mais robusto. Os fabricantes de celular também usufruem do fato deste sistema ser *opensource*, pois assim podem customizá-lo a fim de encaixá-lo melhor no hardware que desenvolveram, o que é algo muito atraente nesta indústria.

1.1. Objetivos

O objetivo final deste trabalho é fazer o controle de um robô por meio do celular Android. A fim de se desenvolver o trabalho proposto, alguns objetivos foram colocados:

1. Obtenção de leituras do Acelerômetro do celular.
2. Estabelecer conexão Bluetooth entre celular e microcontrolador.
3. Montagem de um protótipo de ponte-H responsável pelo controle do motor DC.
4. Fazer controle liga/desliga de 1 motor DC via acelerômetro do celular.
5. Desenvolver esquemático e layout de uma placa de circuito impresso (do ingl”es, *printed circuit* board ou PCB) englobando o módulo de controle dos motores.
6. Manufaturar a PCB e soldar componentes.
7. Integrar todos subsistemas e realizar testes finais.

1.2. Motivação

Baseando-se nos dados apresentados na introdução, este trabalho mostra-se como uma forma de aprofundar conhecimentos em soluções que envolvam microcontroladores e programação para Android.

O uso de microcontroladores neste trabalho é de uso abrangente, pois trata-se de controle de 2 motores DC. Esta abordagem é utilizada em sistemas das mais diversas áreas como robótica, automação industrial, brinquedos etc. A escolha de um microcontrolador ARM foi feita com o intuito de se aprofundar numa arquitetura mais complexa, maior poder de processamento, e ao mesmo tempo fornecer um hardware capaz de aplicações mais sofisticadas para trabalhos futuros, o que mostra vantagens com relação a hardwares mais simples porém populares, como o Arduino. Por isso, o ARM mostra-se como uma opção mais robusta em se tratando de plataformas expansíveis.

No caso da programação em Java para Android, o trabalho mostra como estabelecer uma conexão *Bluetooth* entre um *smartphone* e um microcontrolador,

artifício que pode ser utilizado em diversas aplicações wireless. A decisão de se escolher Android foi com base no crescimento da popularidade deste software, o que torna parte das aplicações deste trabalho mais abrangentes.

2. Conceitos Técnicos

Apresentada a importância que deve ser dada a essas tecnologias, são propostos três subsistemas para uma investigação mais detalhada acerca do tema proposto: um módulo de controle de motor DC de baixa tensão, um microprocessador ARM Cortex-M0, e um *smartphone* Android Samsung Galaxy S4 mini.

Nesta seção, alguns assuntos e termos técnicos que foram de alta importância para o desenvolvimento deste trabalho são abordados. O diagrama de blocos mostrado na Figura 2 explicita quais foram os subsistemas utilizados, sendo que as áreas circuladas foram onde houve desenvolvimento, e a seguir cada bloco será detalhado mais à fundo.

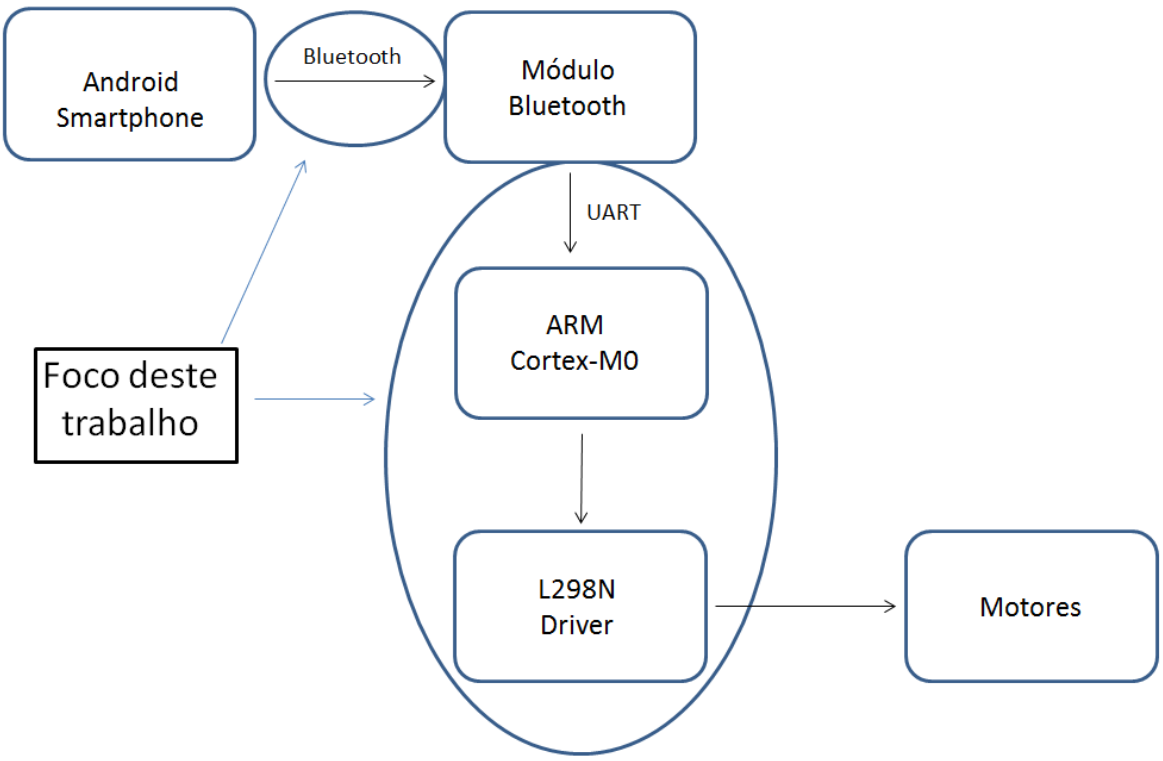


Figura 2: Diagrama de Blocos

2.1. Android

O Android é uma plataforma voltada para dispositivos móveis, totalmente aberta e livre, que foi divulgada em 5 de Novembro de 2007. O sistema Android foi, inicialmente, desenvolvido pela Google, mas atualmente essa plataforma é mantida pela OHA (Open Handset Alliance), um grupo constituído por cerca de 80 empresas, as quais se uniram para inovar e acelerar o desenvolvimento de aplicações e serviços, com o objetivo de trazer aos consumidores uma experiência mais rica em termos de recursos, e menos dispendiosa em termos financeiros para o mercado móvel (SILVA, L. A., 2012).

O Android SDK é uma ferramenta de desenvolvimento que disponibiliza um conjunto de APIs necessárias para desenvolver aplicações para a plataforma Android, utilizando linguagem Java. O Eclipse, IDE de desenvolvimento em Java, pode incorporar essa SDK, e assim, pode ser usado para fazer desenvolvimento de aplicativos para esse dispositivo móvel. Seguindo esse caminho, neste trabalho foi desenvolvido um aplicativo que atendesse ao projeto. Alguns recursos encontrados nessa plataforma são:

- Application Framework: Permite a reutilização e substituição de componentes.
- Dalvik Virtual Machine: É uma máquina virtual Java voltada para dispositivos móveis.
- Suporte multimídia: A plataforma já oferece para áudio, vídeo e formatos de imagens.
- Telefonia GSM.
- Bluetooth, EDGE, 3G, 4G e Wi-Fi (dependente de hardware).
- Câmera, GPS, compasso e acelerômetro (dependente de hardware).

Uma curiosidade é que o Android foi projetado em cima da versão 2.6 do kernel do Linux para os serviços centrais do sistema, tais como segurança, gestão de memória e gestão de processos. O kernel também atua como uma camada de abstração entre o hardware e o restante do software (ANDROID,2014).

2.2. Bluetooth

O Bluetooth é uma tecnologia de comunicação wireless de baixo custo e baixo consumo para troca de dados de curta distância, originalmente desenvolvida para substituir dispositivos conectados à cabo. Essa comunicação acontece através de ondas de rádio UHF, com frequência de banda em torno de 2.4GHz.

A versão 1.0 da especificação Bluetooth saiu em 1999, mas foi inventado pela Ericsson em 1994. O estudo da Ericsson naquela época procurou usar conexões à rádio. A comunicação à rádio não é direcional, e não precisa de uma linha de visão, portanto possuía vantagens significativas em relação ao infravermelho que era usado em alguns outros dispositivos.

Fruto deste estudo foi a especificação para tecnologia wireless Bluetooth. O nome da especificação foi dada em homenagem a Harald Blatand (Blatand significa Bluetooth em Dinamarquês). Harald foi um rei viking do século X que uniu Dinamarca e Noruega. O nome foi adotado pois acreditava-se que a tecnologia Bluetooth iria unificar as indústrias de telecomunicações e computação. (BRAY, J. STURMAN, C. F., 2002)

Uma característica importante do Bluetooth é a possibilidade de se conectar dispositivos de diferentes fabricantes uns aos outros. Essa tecnologia tomou conta de soluções que envolvem comunicação sem fio para aplicações como fones de ouvido, caixas de som, mouses, impressoras, teclados, entre outros.

2.3. Arquitetura ARM

ARM é uma arquitetura de microprocessadores baseada no sistema RISC (Reduced Instruction Set Computing), o que favorece o seu uso em sistemas embarcados. Esses fatores oferecem ampla vantagem do ARM em relação a outras arquiteturas, o que se traduziu na utilização em larga escala desse processador em *smartphones*, laptops, tablets e outros sistemas embarcados que necessitam de alta eficiência da bateria (YIU, J., 2011).

A história dessa arquitetura começa nos anos 80, quando uma empresa britânica de computadores, a Acorn Computers, desenvolveu o primeiro ARM, baseado no trabalho de pesquisa feito pela Berkley RISC, um projeto de pesquisa sobre microprocessadores RISC na University of Berkeley. A partir daí, o desenvolvimento continuou, fusões com outras empresas foram feitas, e em 1991 foi lançada a primeira versão comercial, o ARM6. O interessante é que a empresa atual, ARM Limited, não fabrica os chips físicos, mas desenvolvem o projeto das arquiteturas e os licenciam para outras empresas, vendendo apenas sua propriedade intelectual.

Além disso, dentro da arquitetura ARM, existe a família Cortex-M. Estes microprocessadores RISC 32 bits são voltados para atender demandas em relação a consumo, custo, desempenho e facilidade de uso. Os processadores ARM Cortex-M0 foram desenhados para terem capacidade de oferecer soluções “ultra-low-power”, alta conectividade (Ethernet, USB, *low-power wireless*) e uso de sensores analógicos (sensores de toque e acelerômetros). Essas aplicações requerem estreita integração de funcionalidades analógicas e digitais para processar e comunicar dados. Microprocessadores existentes de 8 bits ou 16 bits geralmente não suportam tais aplicações sem que haja código muito extenso e alta frequência de clock, o que causa aumento de consumo e de calor dissipado. O Cortex-M0 entra como uma solução para isso, além de oferecer baixo custo e estender tempo de bateria. Portanto não é surpreendente que o Cortex-M0 está disponível numa ampla e crescente gama de produtos (ARM, 2014).

A eficiência energética do Cortex-M0 é decorrente do fato de que ele consegue terminar uma tarefa em menos ciclos por segundo. Isso significa que dispositivos que utilizam o Cortex-M0 podem permanecer mais tempo em “sleep-mode”, resultando em menor consumo.

Além da eficiência, a fácil usabilidade do Cortex-M0 é característica marcante deste processador. Desenhado para ser programado em C, o código pode ser compilado por diversas IDEs, e suas rotinas de interrupção e instruções são completamente determinísticas, permitindo que aplicações que precisam de

“timing” muito preciso possam ser desenvolvidas. Essas informações podem ser encontradas no site do fabricante.

2.4. L298N – Driver

O L298, como mostrado na Figura 3, é um circuito integrado usado para aplicações de até 50V e 2A, para ativar cargas de natureza indutiva como relés, solenoides, motores DC e motores de passo (L298 DATASHEET, 2000).

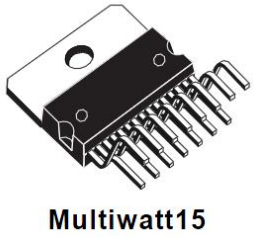


Figura 3: Circuito Integrado – L298N

Apesar do microprocessador Cortex-M0 tomar as decisões de ligar/desligar os motores, ele não é capaz de fornecer tensão e correntes suficientes para acionar os motores. Por isso, é necessário o projeto de uma placa de circuito impresso que inclua um chip como o L298 para que este sim acione os motores, sem danificar os circuitos eletrônicos do microcontrolador.

Com base no *datasheet* do L298, e utilizando-se ferramentas de CAD, projetou-se o seguinte circuito simplificado mostrado na Figura 4. Esta imagem, retirada do *datasheet*, utiliza apenas 1 canal do CI L298N, mas no projeto foram utilizados os 2 canais: 1 canal por motor.

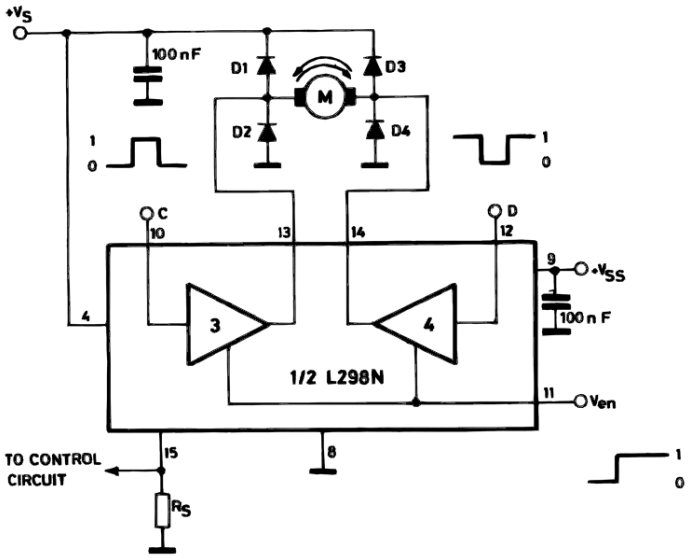


Figura 4: Circuito Simplificado

O CI ainda oferece uma saída, chama de “Current Sense” que, colocando-se um resistor, pode-se medir a corrente que passa pelo motor acionado, e a partir disso aplicar um controle sobre a velocidade do motor.

3. Descrição do projeto e dos materiais utilizados

Este projeto conta com os seguintes materiais principais: o kit de desenvolvimento NXP LPCXpresso 1114, o módulo Bluetooth BlueSMiRF da SparkFun, um celular Samsung Galaxy S4 mini com Android OS, um conjunto de motores 5V com rodas e um módulo de controle de motores DC.

3.1. LPCXpresso 1114

O kit LPCXpresso 1114 é um kit de desenvolvimento de aplicações para o Microprocessador ARM Cortex-M0, como mostrado na Figura 5. A família ARM Cortex-M oferece opções de processadores eficientes, de fácil usabilidade e alta compatibilidade, desenhado para atender as demandas de aplicações do futuro. O Cortex-M0 é o menor da família. Este é um microprocessador de 32 bits, que trabalha em até 50 MHz. Possui 56kb de memória RAM, 42 portas do tipo GPIO, interface serial UART e uma unidade de gerenciamento de potência que permite trabalhar a potências muito baixas (NXP,2014).

Este kit é uma excelente opção para quem deseja fazer uma prototipação rápida para algum projeto. Ele possui uma IDE baseada em Eclipse muito fácil de utilizar, além de oferecer opções de debug, step-by-step, entre outras funcionalidades que permitem que o desenvolvedor tenha maior fluidez no desenvolvimento.

O código trabalhado neste Kit foi em linguagem C, aprendido em algumas disciplinas da Universidade, e aperfeiçoado em outras disciplinas como *Aplicações de Microprocessadores II.* Uma vez consolidado o conhecimento para programação de microcontroladores, mesmo que haja mudança de hardware, torna-se intuitivo o desenvolvimento de códigos para projetos desse tipo. Além disso, o kit possui diversas documentações que clarificaram como alcançar os resultados neste trabalho.

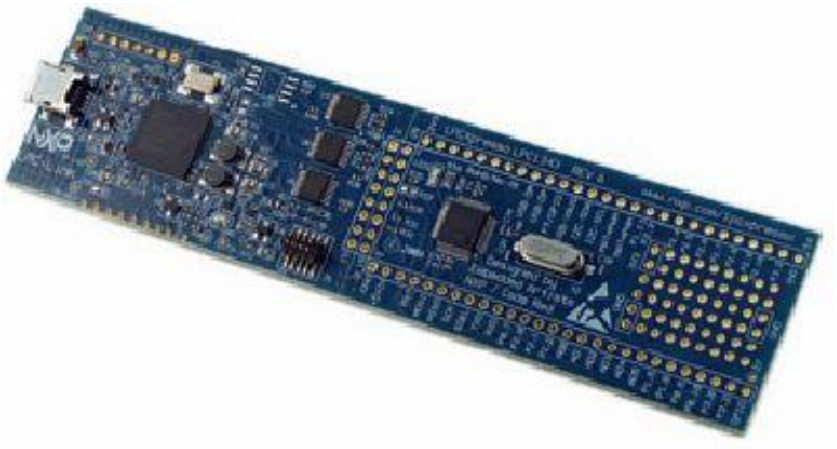


Figura 5: Kit ARM LPC1114

3.2. BlueSmiRF – módulo Bluetooth

Conectado diretamente via UART ao kit LPC1114, encontra-se um módulo Bluetooth chamado BlueSMiRF, igual ao mostrado na Figura 6. Este é um módulo Bluetooth wireless que utiliza o chip RN-42, com capacidade de até 115200bps, trabalhando com frequências em torno de 2.4GHz (SPARKFUN,2014), possibilitando à comunicação entre o kit LPC1114 e o celular que será descrito a seguir. Este módulo também é de baixa potência, podendo trabalhar de 3.3V a 6V.

Este módulo mostrou-se muito conveniente ao projeto, porque simplificou ao máximo a comunicação Bluetooth, bastando ler a porta UART do LPC1114 para receber os dados provenientes do celular.

Figura 6: Módulo BlueSmiRF

3.3. *Smartphone* Android

Outro equipamento utilizado foi um celular Samsung Galaxy S4 mini operando com sistema operacional Android. Este dispositivo, mostrado na Figura 7, é equipado com sistemas de *Bluetooth*, além de possuir hardware que trabalha com um acelerômetro. O celular é a forma de controle do usuário por meio de um aplicativo que tome conta disso.

Figura 7: *Smartphone* Android Samsung Galaxy S4 mini

O controle é realizado via leitura do sensor acelerômetro do celular. O vetor de gravidade é projetado nos eixos X, Y e Z mostrados na Figura 8. Com isso, é possível obter a posição espacial do celular, que é usada para comandar dois

motores de forma intuitiva, onde o usuário segura o celular como um volante, e pode inclinar o celular para esquerda, direita, para frente ou para trás.

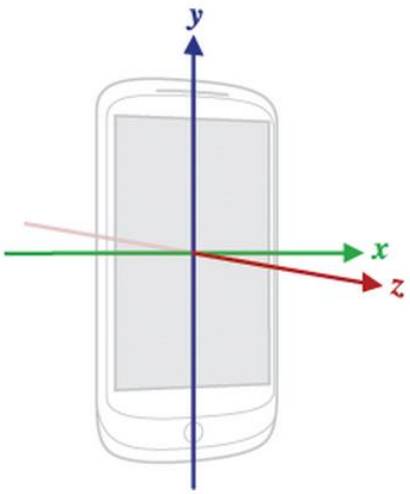


Figura 8: Eixos do sensor acelerômetro.

Finalmente, foi projetada uma placa de circuito impresso responsável por fazer a interface entre o microcontrolador e os motores, cujo esquemático está mostrado na Figura 9. Esta placa contém um circuito integrado composto por uma ponte H, e foi conectada a motores DC de baixa tensão, com o intuito de mostrar a solução proposta em funcionamento.

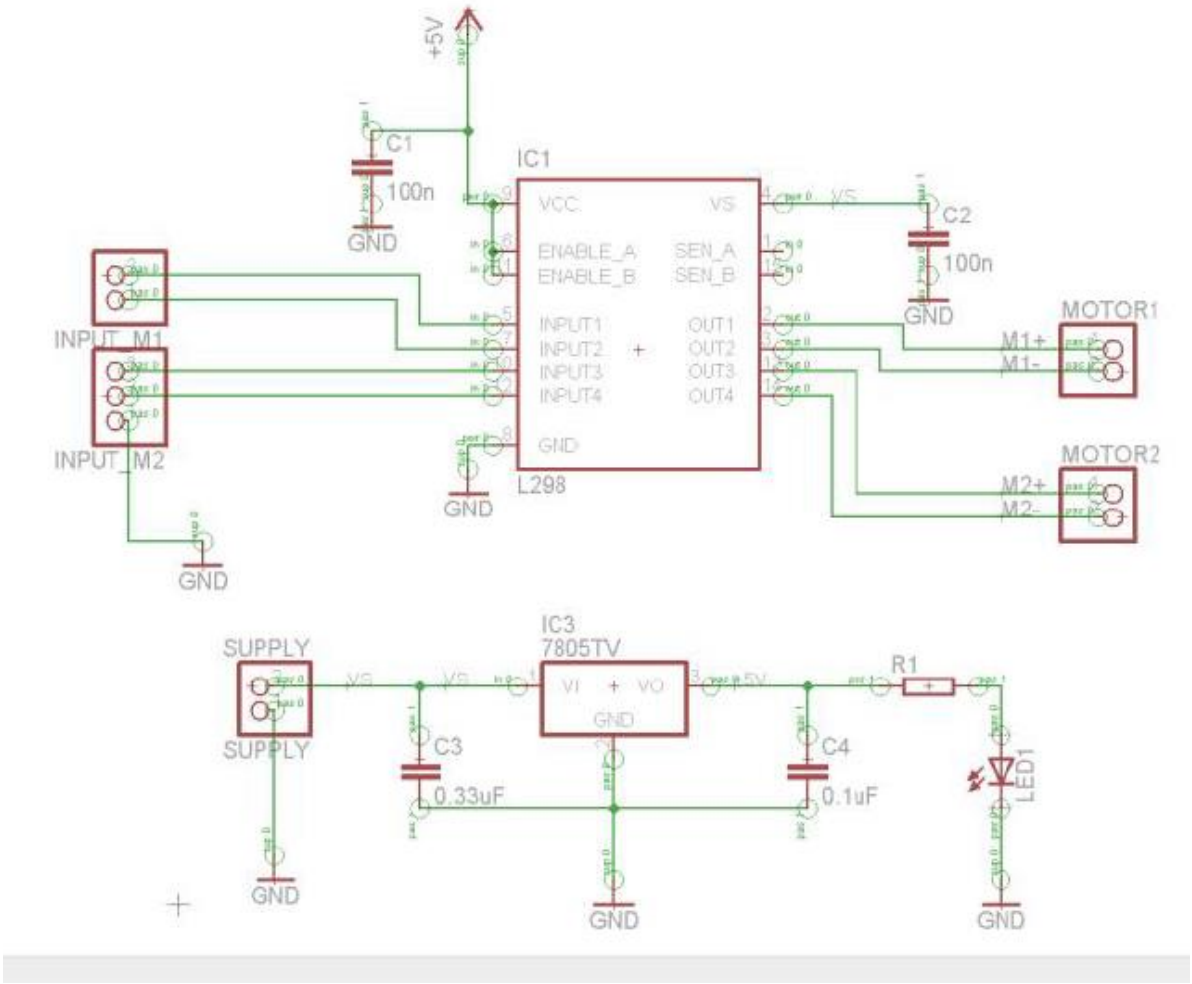


Figura 9: Esquemático da Placa de Circuito Impresso projetada.

4. Métodos

Neste capítulo, serão abordados os métodos utilizados para o desenvolvimento do trabalho proposto. Como forma de simplificação, os métodos estarão separados em cada subsistema desenvolvido. Como ilustrado anteriormente, os subsistemas são: ARM LPC1114, *Smartphone* Android, e o Driver.

4.1. ARM LPC1114

Primeiramente, com o objetivo de se familiarizar com o microcontrolador em questão e desenvolver o software final a ser utilizado nele, utilizou-se sua IDE chamada LPCXpresso, uma IDE baseada em Eclipse fornecida pela NXP, fabricante do microprocessador LPC1114 utilizado.

Este ambiente de desenvolvimento é bastante intuitivo, com funções para *debug,* e com exemplos de código que foram utilizados com o intuito de se aprofundar na arquitetura do hardware.

Alguns exemplos de código que foram utilizados foram programas que trabalham com interrupções, comunicação serial, timers, GPIO etc. Estes programas foram fornecidos pelo próprio fabricante, e uma vez compreendidos, foram customizados de diferentes maneiras para agregar ao projeto.

A seguir, a Figura 10 apresenta a tela principal da IDE. Nesta IDE foi possível desenvolver código em C utilizado como versão inicial para testes.

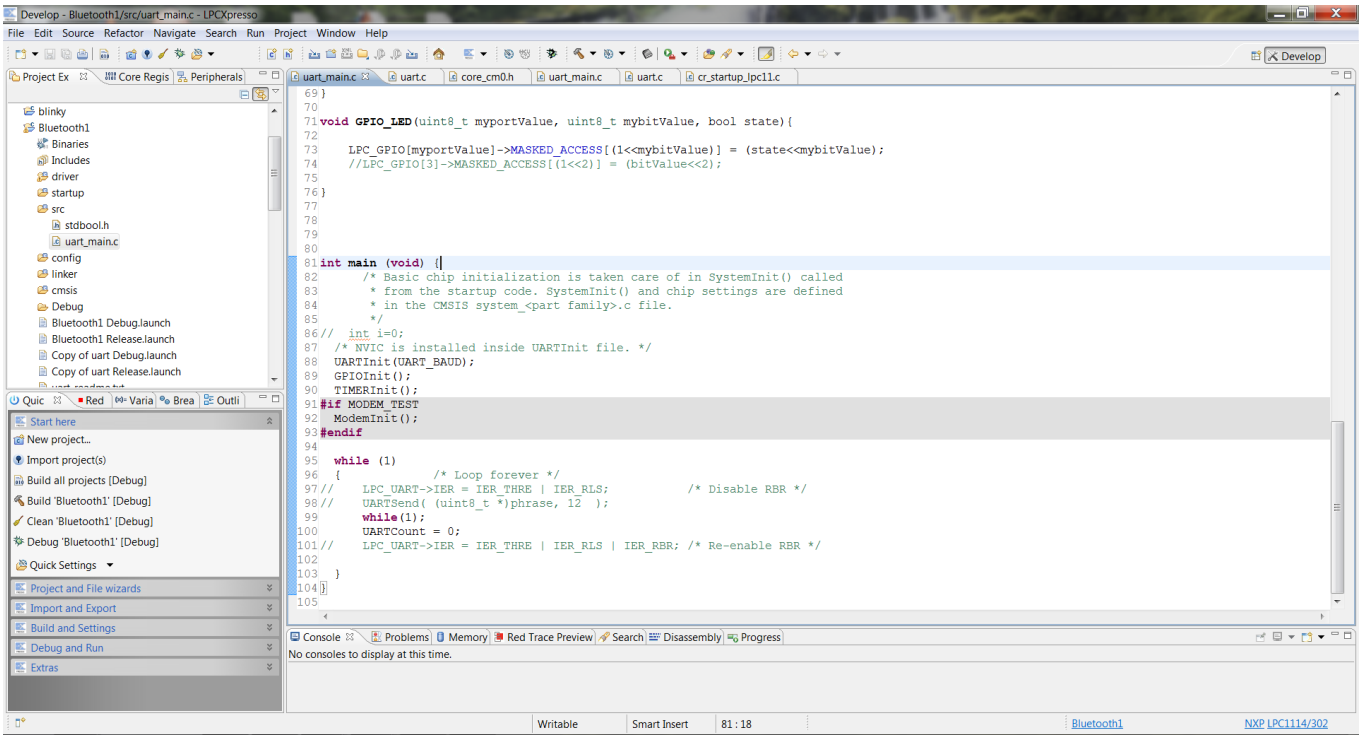


Figura 10: Tela principal da IDE LPCXpresso

O principal código a ser customizado foi o programa chamado uart.c, encontrado no apêndice A deste trabalho. Esse código é uma *Application Programming Interface* (API) responsável por tratar de toda comunicação UART realizada pelo microprocessador. Este código implementa as seguintes funções:

- UART_IRQHandler : Trata das interrupções geradas pelo canal de comunicação UART. Verifica se há erros, se houve overflow do buffer de recebimento, e disponibiliza o dado recebido pelo canal. É ativada sempre que há uma flag de interrupção pelo registrador NVIC (*Nested Vectored Interrupt Controller*)
- ModemInit : Inicialização do hardware da porta UART0 como modem.
- UARTInit : Inicializa comunicação UART, ativa interrupções e define todos os parâmetros como clock, paridade, baud rate, stop bits, FIFO etc.

Dado o código disponibilizado, a customização partiu da função UART_IRQHandler. Dentro desta função, o programa poderia ser customizado a tomar qualquer tipo de ação uma vez que um dado fosse recebido via UART. O objetivo então é que o processador tomasse decisões baseado exatamente no dado que fosse recebido. Diferentes dados resultariam em diferentes ações.

Então, para fins de testes, o seguinte procedimento foi feito:

1. Conexão do módulo SparkFun Bluetooth ao canal de comunicação UART0
2. Usando um aplicativo chamado Bluetooth S2 (terminal Bluetooth), foi possível conectar-se ao microcontrolador com o *smartphone*, e enviar e receber dados. Esse aplicativo foi encontrado na “Play Store”, loja de aplicativos do Google. É um aplicativo gratuito e de fácil usabilidade.
3. A partir do recebimento de dados, foi implementado um código fonte para controle de um LED pelo celular. O microcontrolador recebe dados via Bluetooth do celular, e dependendo dos dados, toma decisões pré-determinadas. Controlou-se a frequência e ciclo de trabalho de um LED.

A partir deste método, constatou-se que era de fato possível fazer essa conexão celular/microcontrolador, e dessa forma abriu-se um caminho para fazer uso de funcionalidades mais complexas do celular para tomada de decisões do microcontrolador.

Um detalhe importante é que, para se fazer o controle do LED, foi necessário o uso das portas GPIO (*General Purpose Input Output*). Isto é de grande importância pois esse mesmo tipo de porta seria utilizado posteriormente para fazer o acionamento dos motores de baixa tensão.

Na arquitetura ARM Cortex-M0, para se fazer uso das portas GPIO, são necessários dois passos: 1 – Habilitar clock para as portas GPIO e 2 – Definir direção da porta, *input* ou *output.* Com isso, pode-se fazer uso dos pinos de GPIO. Tais configurações podem ser encontradas no anexo, no programa uart_main.c na função GPIOInit().

A partir daqui, os seguintes objetivos foram alcançados: conexão Bluetooth entre celular e microcontrolador, tomada de decisões do microcontrolador a partir da interrupção gerada pela porta UART, e controle dos pinos de GPIO. Daqui em diante, em relação ao ARM LPC1114, o código foi trabalhado para atingir o resultado final.

Portanto, implementou-se uma função com o comando de “switch”, para tomada de decisões a partir do dado recebido via UART, como descrito no pseudo-código a seguir:

```
switch(comandoCelular){   //Comando enviado pelo Celular,
                          //recebido via Bluetooth

     case 'Parado':       //Celular manda motor parar
          Desliga   porta 1
          Desliga   porta 2
          Desliga   porta 3
          Desliga   porta 4

break;

     case 'Frente':       //Celular manda motores ligarem para
frente
          Liga           porta 1
          Desliga        porta 2
          Liga           porta 3
          Desliga        porta 4

break;

     case 'Tras':         //Celular manda motores ligarem para
trás
          Desliga        porta 1
          Liga           porta 2
          Desliga        porta 3
          Liga           porta 4

break;

     …

     …

     …
```

```
}
```

O celular envia comandos via UART e a função switch decide o que deveria ser feito. A partir do comando recebido, o microcontrolador liga ou desliga determinadas portas de GPIO. Essas portas foram conectadas ao Driver projetado, dessa forma fazendo os motores girarem para a direção desejada: frente, trás, esquerda ou direita. O fluxograma da Figura 11 exemplifica o funcionamento do algoritmo utilizado no software do microcontrolador.

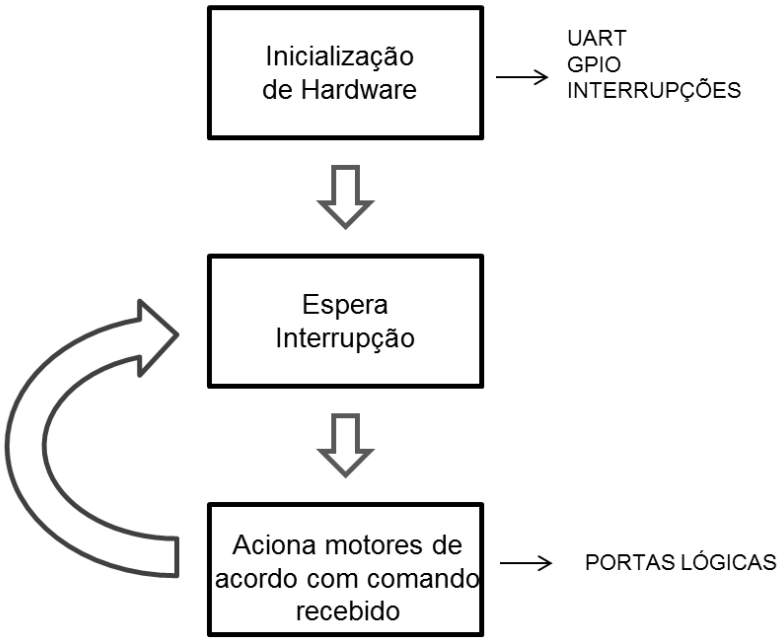


Figura 11: Fluxograma do microcontrolador

Através deste método discutido nesta seção, conseguiu-se a conexão Bluetooth entre celular e microcontrolador, e o controle dos pinos de GPIO para acionamento dos motores.

4.2. *Smartphone* Android

Para criação do aplicativo a ser utilizado no celular, foi utilizada a IDE Eclipse, um ambiente desenhado para programação em diversas linguagens de programação, incluindo Java, e faz parte do kit de desenvolvimento de software recomendado para desenvolvedores Android. Através da própria ferramenta Eclipse, que é gratuita, é possível fazer o download do Android *Software Development Kit* (SDK).

O Android SDK possui todas as bibliotecas necessárias para se fazer o desenvolvimento de um aplicativo para Android. A linguagem utilizada é Java, mas o conhecimento dessa linguagem não é suficiente para se desenvolver um aplicativo. É necessário ter noções de como é um ciclo de vida deste programa, além de entender as bibliotecas disponíveis e o hardware em si. Todavia, toda essa documentação pode ser encontrada no site developer.android.com, o site oficial para desenvolvedores em Android, e que contém a descrição de todas as bibliotecas, além de tutoriais e passo-a-passo de como utilizar cada módulo de um dispositivo.

Por meio do site oficial, e da ferramenta Eclipse, foram trabalhados alguns aplicativos muito simples com o objetivo de familiarização com a linguagem Java e com aplicações para este sistema operacional. A Figura 12 mostra como é a ferramenta Eclipse. Percebe-se que ela é muito similar à IDE LPCXpresso, devido ao fato de que esta última foi baseada no Eclipse.

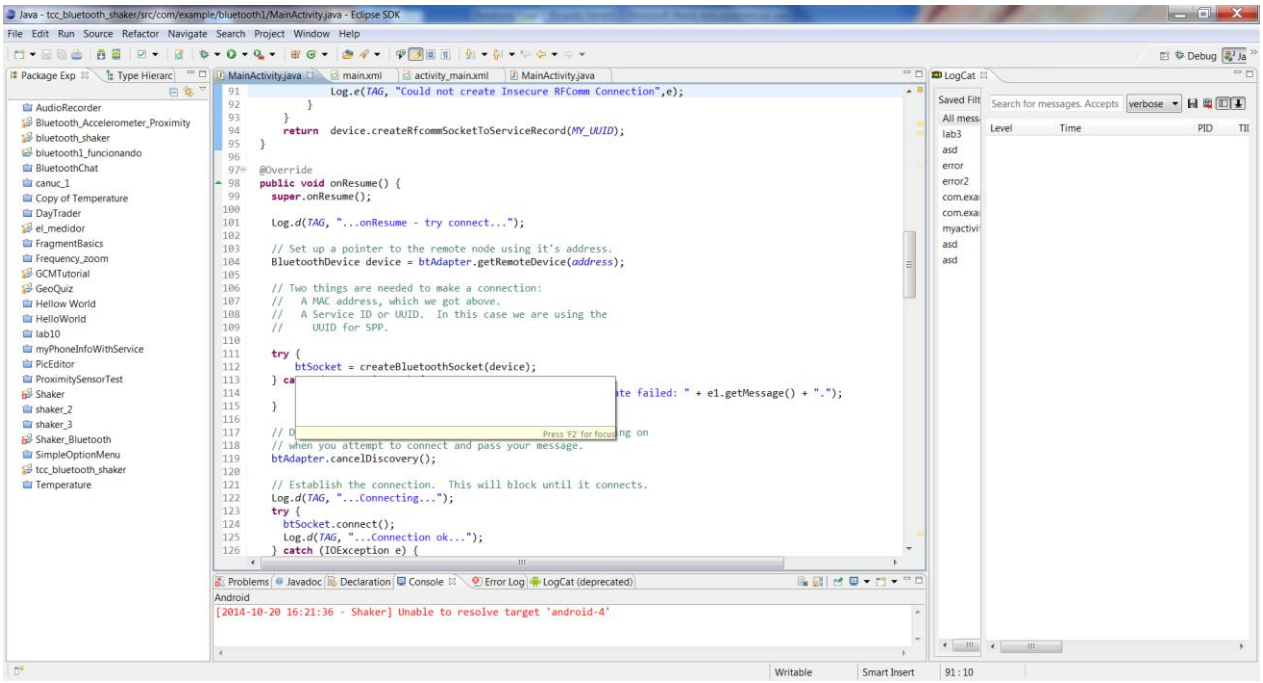


Figura 12: Tela Principal da IDE Eclipse

O programa possui três partes principais que são responsáveis pelo resultado obtido: definição de variáveis e inicialização dos hardwares (Bluetooth e Acelerômetro), conexão ao módulo Bluetooth, e por último leitura dos valores do acelerômetro e envio de dados.

Além das definições de variáveis, que são comuns em qualquer desenvolvimento de software, a inicialização dos hardwares é algo característico na programação Java para Android. É necessário criar objetos, que funcionam como “handlers” (do inglês, manipulador) e recebem como argumento o hardware em questão. No caso, foram criadas variáveis do tipo *handler* chamadas *btAdapter* e *mAccelerometer,* responsáveis por fazer a interface para o Bluetooth e o sensor Acelerômetro, respectivamente.

Uma observação a ser feita é que durante essa inicialização de hardwares, uma sequência de comandos específica se fez necessária:

```
this.mWakeLock =
pm.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK, "My Tag");
this.mWakeLock.acquire();
```

Esses comandos dizem ao aplicativo que a tela não deve girar quando o celular for posto numa posição horizontal. Isso se deve ao fato do funcionamento do ciclo de vida do aplicativo, exemplificado abaixo:

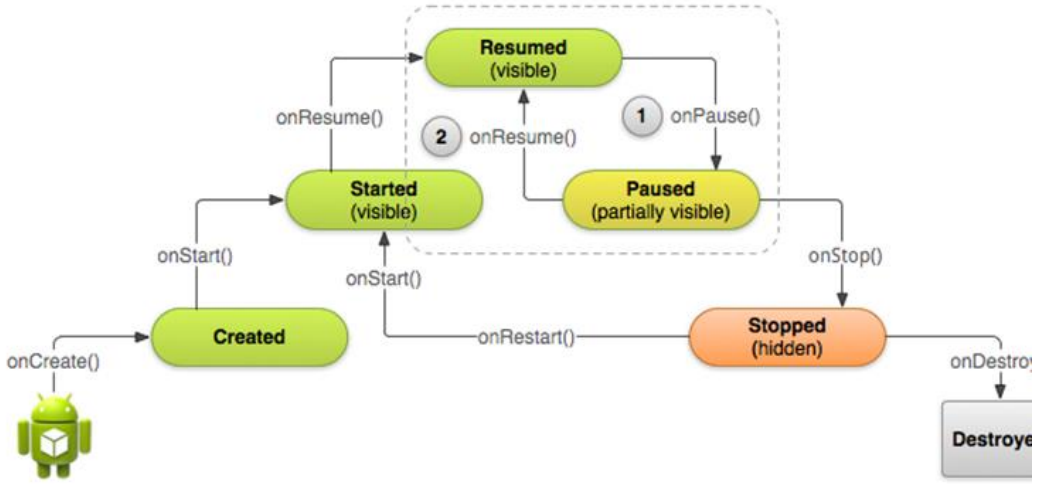


Figura 12: Ciclo de Vida do Aplicativo

Toda vez que o celular é posto para girar a tela, automaticamente as funções onPause() e onStop() são convocadas, e logo após as funções onRestart() e onResume(), voltando para o funcionamento normal do aplicativo. Isso acontece porque o aplicativo deve refazer todo o *layout* do programa, para que ele encaixe na tela horizontal do celular (PHILIPS, 2014). O problema que isso gerou para o projeto é que quando a tela era girada, e a função onStop() é convocada, a conexão Bluetooth era perdida, e poderia não ser retomada dependendo da distância do módulo, causando erros. Então a solução proposta foi manter a tela do celular numa posição fixa, como é feito em alguns aplicativos que não giram a tela. Assim o erro não foi mais encontrado.

Com relação à conexão Bluetooth, o programa teve como base um exemplo que pode ser encontrado no site http://forum.xda-developers.com/showthread.php?t=2660904. Dentro deste programa, os seguintes passos são necessários:

- Declarar a permissão Bluetooth: É necessário que essa permissão seja feita para que haja qualquer comunicação Bluetooth.

- Configuração Bluetooth: Verificar se o celular suporta Bluetooth, e se ele está ativado.
- Encontrar dispositivos: Isso pode ser feito tanto procurando dispositivos pareados como também através do “Device Discovery”.
- Inicializar e gerir uma conexão: A partir daqui, os dispositivos estão conectados e podem realizar troca de dados.

Com a ajuda da documentação encontrada no site oficial e alguns programas de exemplo, foi implementada a conexão Bluetooth entre celular e o módulo BlueSmiRF da Sparkfun.

Por último, durante o *runtime* do aplicativo, era necessário fazer leituras dos valores do acelerômetro e enviar o respectivo comando desejado para o microcontrolador, a depender desses valores lidos. Depois de inicializado o hardware do Acelerômetro, o Android possui uma função chamada onSensorChanged(). A lógica desta função é similar a de uma interrupção: toda vez em que o valor retornado pelo sensor mudar, essa função é chamada.

Assim, essa função é constantemente chamada durante o tempo em que o aplicativo roda. Além de fazer a leitura, a função também envia através do canal Bluetooth o comando para o robô. Os eixos do acelerômetro utilizados foram o Z (faz o robô ir para frente ou para trás) e o Y (esquerda ou direita). Os valores de Z e Y podem variar de -10 a +10. A figura 13 mostra o celular na posição X=-10, Y=0 e Z=0. Repare que a gravidade está no sentido negativo de X. A lógica utilizada está mostrada no fluxograma da Figura 14.

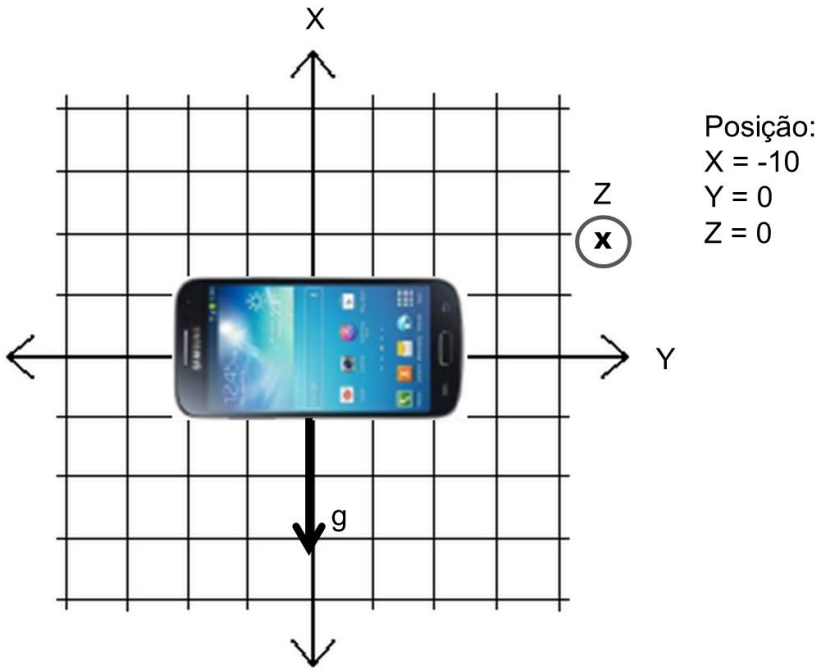


Figura 13: Fluxograma da lógica de tomada de decisões

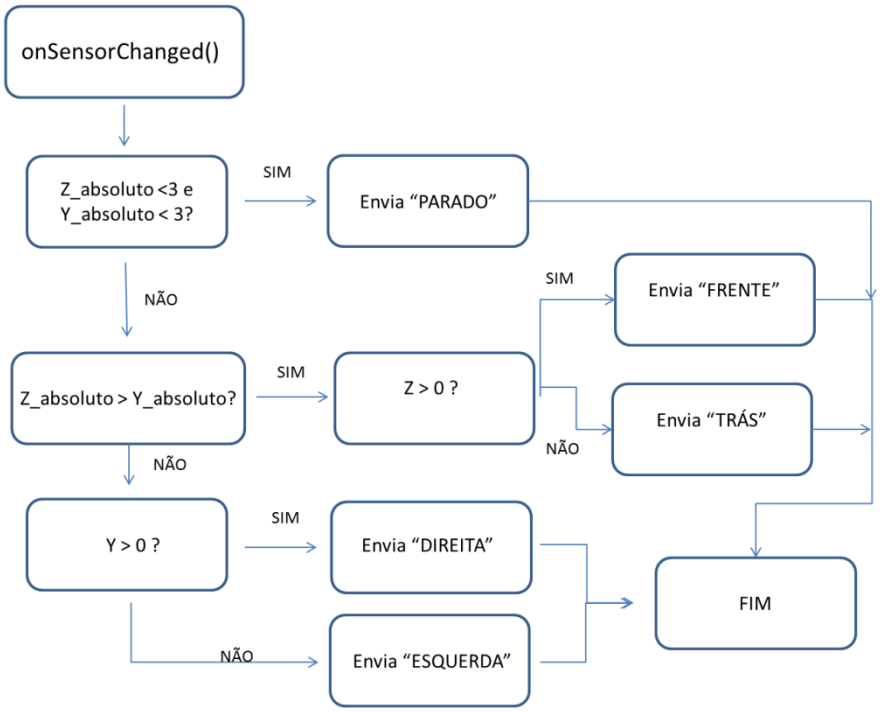


Figura 14: Fluxograma da lógica de tomada de decisões

Dessa forma, o aplicativo foi implementado com sucesso, enviando os comandos de acordo com a posição espacial do celular. Assim, o usuário pode fazer o controle do robô.

4.3. Driver L298N

O Driver L298N foi concebido a partir do fato de que o microcontrolador não pode acionar os motores através de suas portas lógicas. Pesquisando entre os CIs disponíveis, encontrou-se o L298, circuito que foi projetado justamente para esta aplicação.

Inicialmente, utilizou-se uma *Protoboard* para possibilitar os primeiros testes. A *protoboard* contou com o seguinte circuito simplificado abaixo, onde o principal CI é o L298, que trata-se de uma ponte-H usada para controlar motores DC.

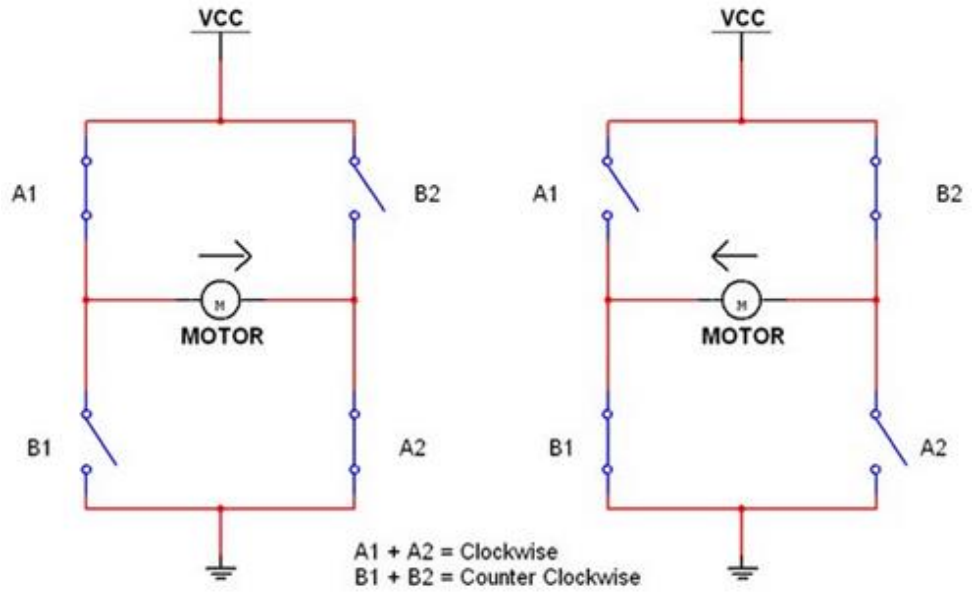


Figura 15: Circuito simplificado de uma ponte-H

Uma vez com o circuito montado na *Protoboard*, foi feito o controle de um pequeno motor de 5V. Por meio de um sinal PWM desenvolvido em software, foi controlada a velocidade do motor, utilizando-se o aplicativo para Android

anteriormente desenvolvido. Com isso, visualizou-se o conceito de acionar motores via celular, e o projeto ganhou mais consistência.

Assim que a *protoboard* foi capaz de provar o conceito, decidiu-se então projetar uma placa de circuito impresso, com o objetivo de reduzir tamanho e peso, pois este subsistema estaria alocado dentro do robô, próximo aos motores.

Para que tal circuito fosse projetado, utilizou-se o software chamado Eagle CadSoft (http://www.cadsoftusa.com/), um software de CAD para projetos de PCBs. Tal programa permite o usuário montar o esquemático desejado, utilizando especificações desejadas para cada componente, e gera um arquivo de *layout*, onde os componentes são interligados e dispostos da melhor maneira. A seguir segue duas imagens da tela de trabalho do Eagle, tanto para o layout (Figura 16), quanto para o esquemático (Figura 17).

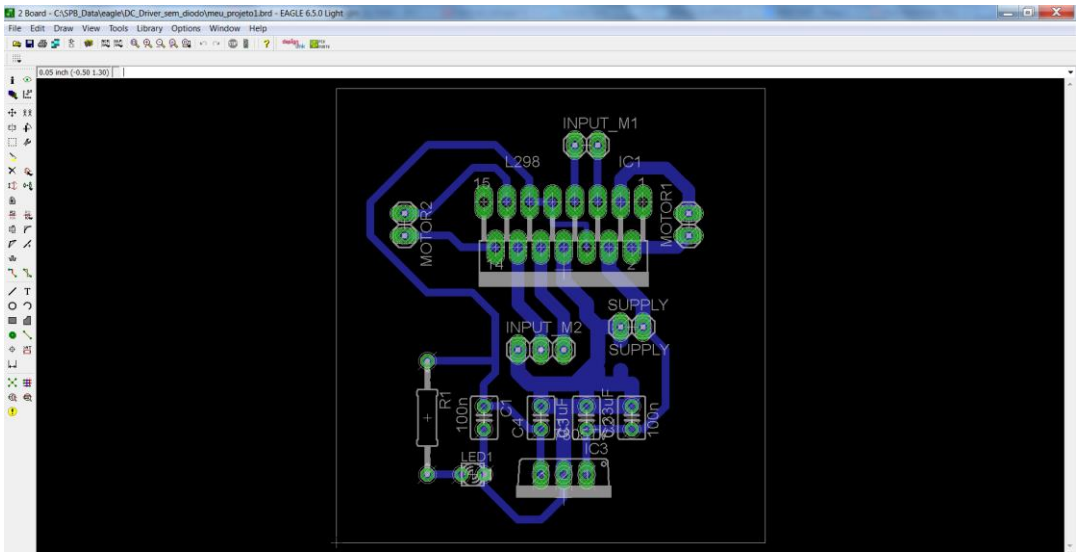


Figura 16: Tela do software Eagle para layout

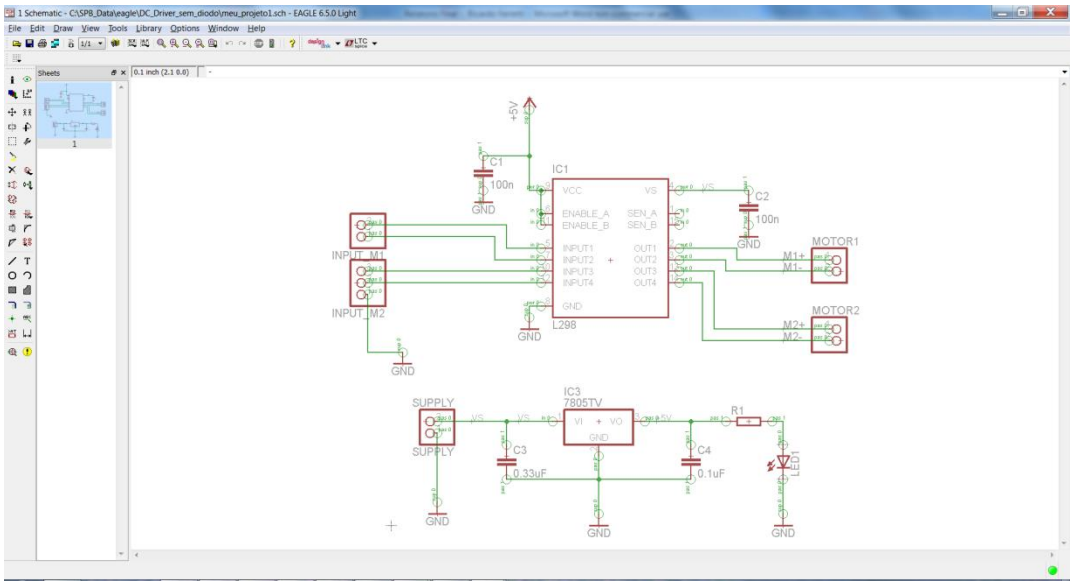


Figura 17: Tela fo software Eagle para circuito esquemático

Com relação ao projeto da PCB, alguns pontos precisam ser considerados. Primeiramente, o CI L298N oferece um pino de saída chamado “Current Sense”, que é por onde a corrente que passa pelo canal vai para o *ground*. O datasheet explica que este pino pode ser utilizado para se medir a corrente que passa pelo motor, utilizando-se um resistor, e assim podendo fazer um controle de velocidade em malha fechada. Optou-se por não fazer uso deste pino, mas é algo que poderia ser desenvolvido num trabalho futuro.

Outro ponto em destaque é um erro que este projeto sofreu, mas que pôde ser corrigido. O *datasheet* declara que diodos externos devem ser colocados em volta do motor, como na Figura 18.

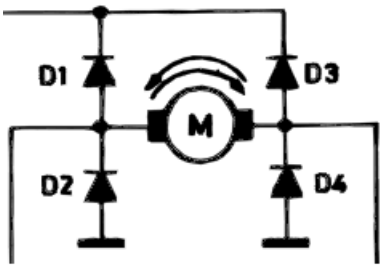


Figura 18: Diodos externos

Como se pode constar no esquemático, tais diodos não foram colocados devido a um erro no projeto, e com isso foi necessário fazer uma adaptação da PCB, para evitar futuros problemas.

4.4. Sistema completo

Para finalizar esse capítulo, segue um apanhado geral do que foi discutido nos métodos anteriores. O diagrama da Figura 19 mostra, então, como o sistema final foi projetado, mostrando como cada subsistema comunica-se com o outro.

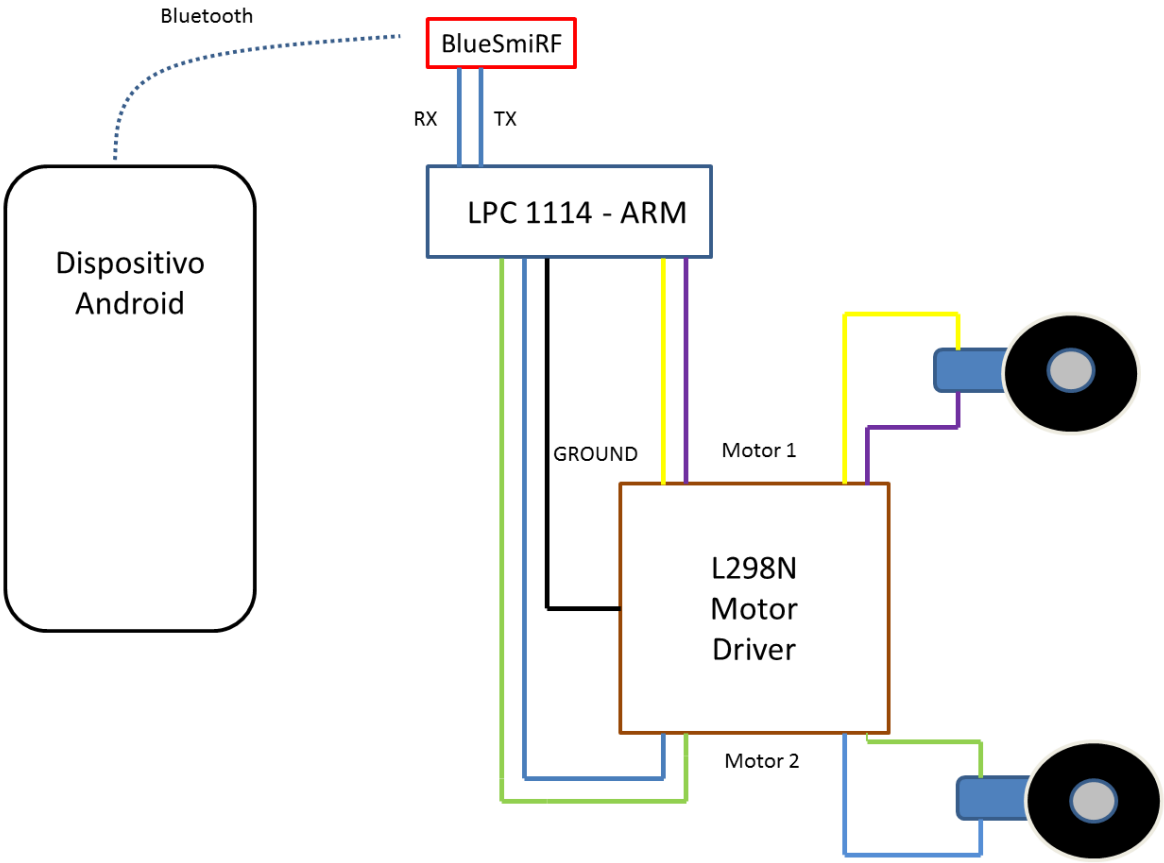
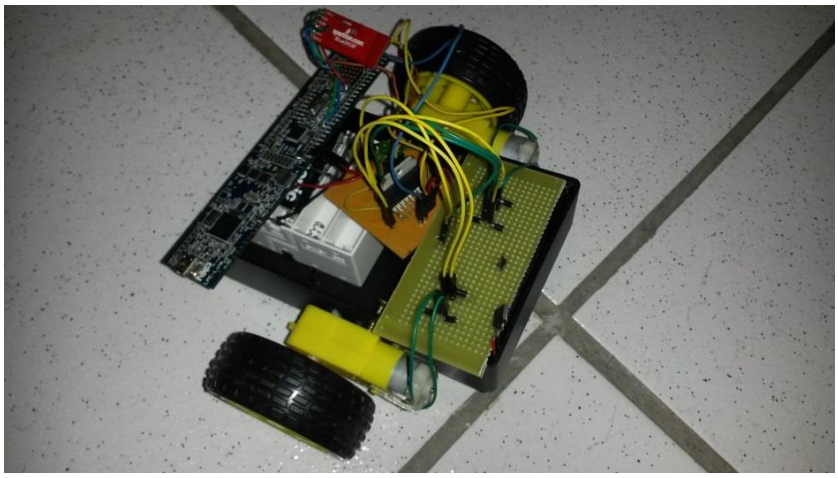


Figura 19: Diagrama do sistema completo

5. Resultados e Discussões

Tendo em vista os objetivos listados, juntamente com os materiais e métodos mencionados até aqui, foi possível atingir a meta final proposta por este trabalho. Conseguiu-se então controlar um pequeno robô através de um *smartphone* Android, fazendo-o ir para frente, trás, direita e esquerda, através do acionamento de pequenos motores de baixa tensão. A Figura 20 mostra o robô completo.



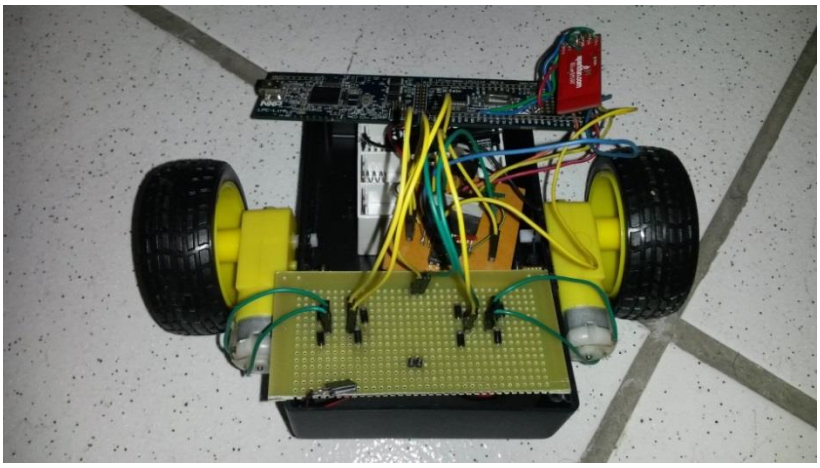
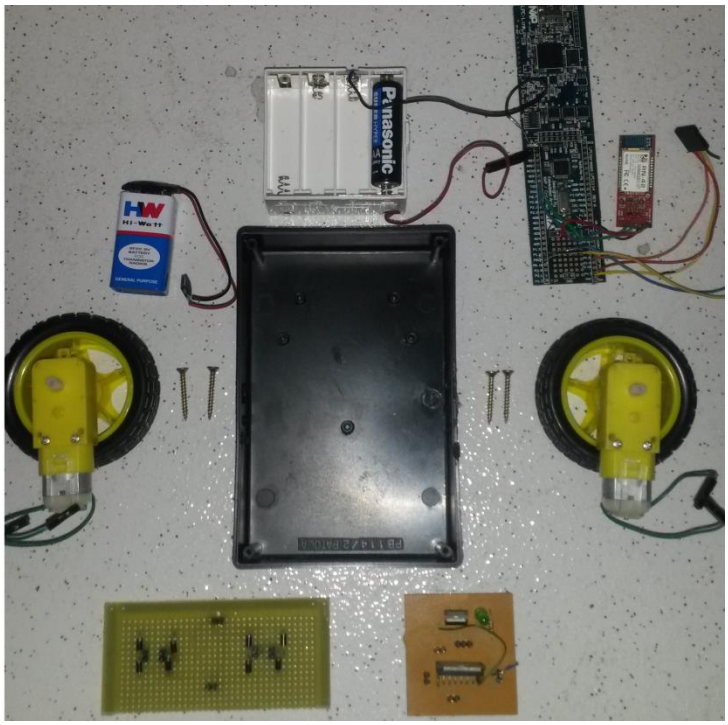


Figura 20: Robô montado



O robô é composto pelas seguintes partes:

- 2 pequenos motores.
- 1 bateria de 9V com conector
- Uma plataforma de plástico preta.
- 4 parafusos para fixação dos motores na plataforma
- Kit LPCXpresso 1114 conectado ao módulo BlueSmiRF
- Conjunto de 3 pilhas AA de 1,5V.
- PCB para acionamento de motores
- Ponte de diodos para conexão com os motores.

Como um primeiro resultado pode-se citar a placa de circuito impresso, que foi projetada utilizando-se o software Eagle PCB Design, e manufaturada utilizando-se os recursos fornecidos pelo departamento de Engenharia Elétrica da USP como a placa de cobre e a máquina de furação e desenho. Abaixo segue uma imagem da placa recém manufaturada.

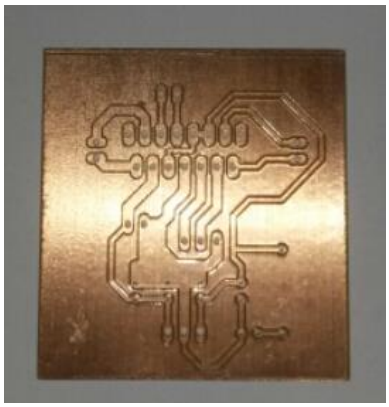


Figura 21: Placa de Circuito Impresso manufaturada

Após ter sido manufaturada, os componentes previamente listados foram soldados à placa mostrada na Figura 21. Esse hardware funcionou muito bem, apesar de alguns ajustes serem necessários. Ela fez então a interface entre microcontrolador e motores, acionando-os quando comandada.

Para realizar essa interface entre microcontrolador e *driver*, foram necessários 5 pinos do LPC1114: 2 sinais para cada motor, e 1 para o *ground.* O motivo de serem necessários 2 pinos de controle por motor está na forma como o CI L298N funciona, exemplificado na tabela abaixo.

Entradas		Função
V_enable = H	M1 = H ; M2 = L	Motor p Frente
	M1 = H ; M2 = L	Motor p Trás
	M1 = M2	Parada do Motor
V_enable = L	M1 = X ; M2 = X	Parada do Motor

L = Nível Lógico Baixo
H = Nível Lógico Alto
M1 e M2 = Pinos de entrada do motor

Tabela 1: Tabela verdade para controle de direção do motor

A tabela mostra que quando um pino está em estado lógico alto, e o outro baixo, o motor gira em uma direção, e vice-versa. No caso em que os pinos estão no mesmo estado lógico, o motor para rapidamente.

Optou-se por não utilizar modulação de largura de pulso (PWM) para os motores pela maior simplicidade de implementação e para realizar curvas. Portanto, utilizou-se apenas nível lógico alto ou baixo para ligar ou desligar os motores, não havendo controle de velocidade. Para realizar curvas dessa forma, basta ligar os motores em sentidos contrários.

Além disso, o software que foi desenvolvido para ser utilizado no microcontrolador funcionou da forma esperada. Ele encontra-se no Apêndice. Este software é responsável por comunicar-se serialmente com o módulo *Bluetooth*, e a partir desta leitura de dados, emitir um sinal via sua porta lógica de GPIO. Tal sinal foi utilizado para comunicar ao *driver* a direção em que os motores deveriam girar.

A linguagem C mostrou-se efetiva o suficiente para alcançar o resultado desejado. Em se tratando de um microcontrolador, essa linguagem é uma das mais difundidas, além de ser relativamente simples.

O kit LPC1114 também foi uma boa escolha, visto que ele possui diversos pontos favoráveis, como a simplicidade para programação, bastando uma conexão USB, e também por ser um hardware que possui várias funcionalidades.

Outro resultado que pode ser citado é o desenvolvimento de um aplicativo em Java para Android que permite a leitura de dados do sensor acelerômetro do celular, e a comunicação entre celular e microcontrolador. O código utilizado pode-se encontrar no Apêndice B deste trabalho, e uma *screenshot* da tela do aplicativo é mostrada na Figura 22, onde os três números são as leituras dos eixos X,Y e Z do sensor. O eixo X não foi utilizado, e está mostrado meramente para ilustração. A Figura 23 exemplifica como a posição do celular representa o comando enviado para o robô. A Figura 24 mostra a conexão realizada com sucesso entre celular e módulo Bluetooth.

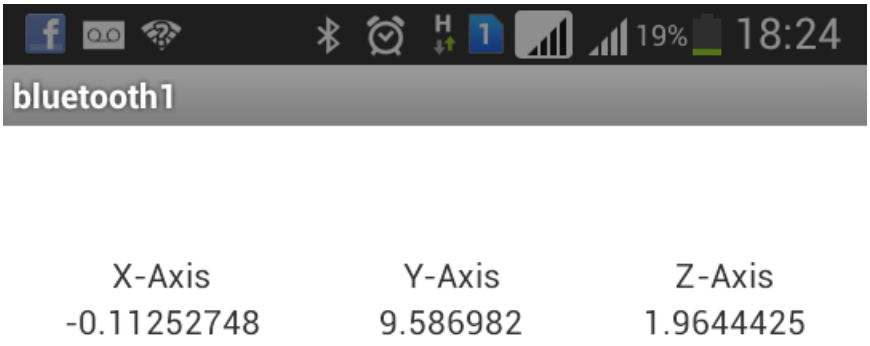


Figura 22: Screenshot da tela do aplicativo

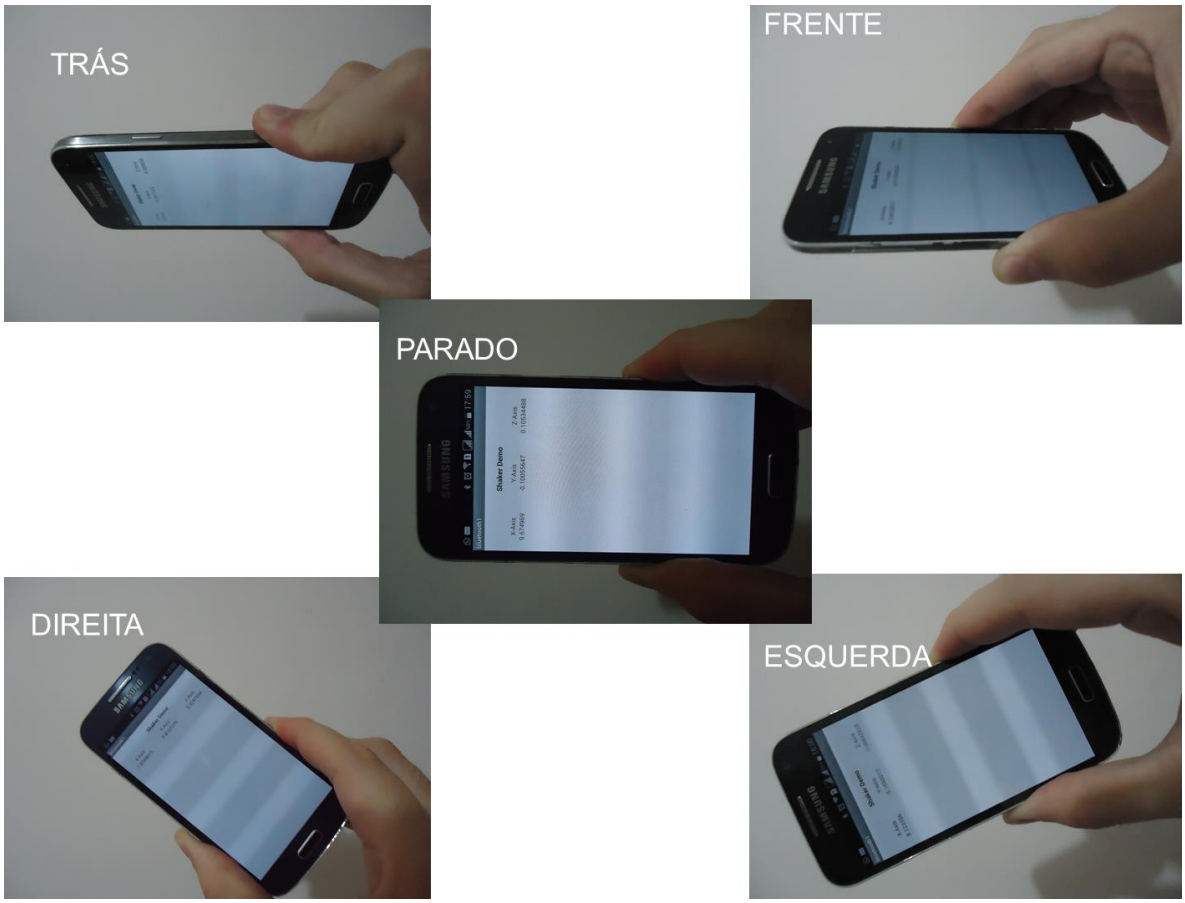


Figura 23: Posição do celular referente a cada comando.

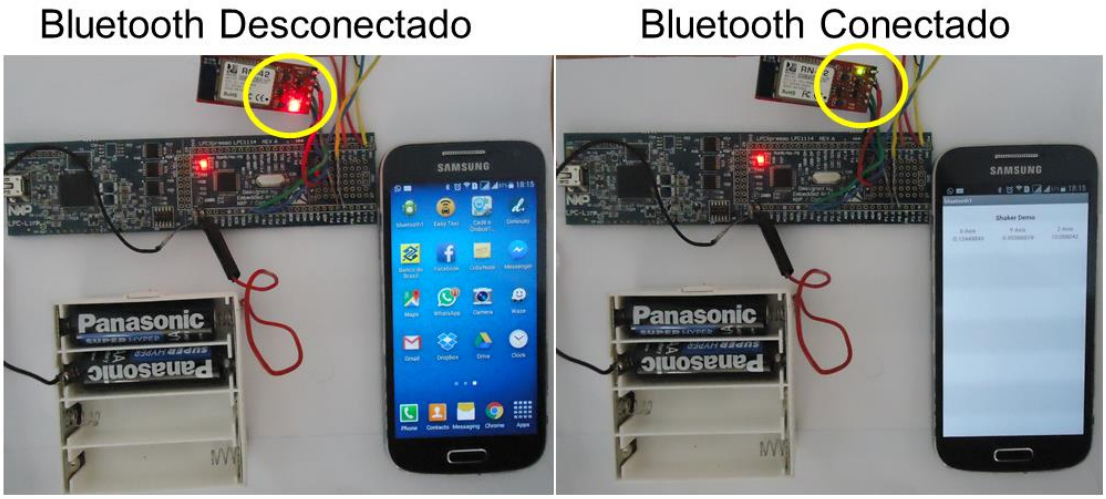


Figura 24: Conexão entre celular e módulo Bluetooth

Apesar de Java não ser uma linguagem simples, o *Android*, por ser um sistema *opensource*, possui uma quantidade extremamente grande de documentação, fórums, entre outras ferramentas que ajudaram na programação para esta plataforma. Assim, apesar de que um conhecimento avançado de Java ajudaria muito no desenvolvimento deste trabalho, ele não foi necessário, visto que a internet fornece todo o conhecimento que se deseja. O que definitivamente fez a diferença aqui foram conhecimentos em linguagem de programação orientada a objeto, o que facilitou a compreensão de códigos fonte.

Além do código responsável pelo funcionamento do programa, o Android também necessita de um código responsável pelo *layout* e interface com o usuário. Essa porção de código é programada em XML, mas o Eclipse também possui uma ferramenta que permite a construção de um *layout* sem necessidade de conhecimentos em programação nessa linguagem. O *layout* possui 1 Título principal (nome do programa), 3 subtítulos (Eixos X, Y e Z), e 3 objetos do tipo *TextView* (responsáveis por apresentar os valores provenientes do acelerômetro).

Finalmente, a conexão entre celular e microcontrolador foi realizada com sucesso, podendo ser verificada através dos comandos enviados ao robô via Bluetooth. Portanto, o robô foi o meio pelo qual a conexão se mostrou funcionando. Alguns fatores que foram decisivos para o desenvolvimento deste trabalho foram conhecimentos básicos de Java e Android, conhecimentos de microcontroladores e prática em programação em C para este hardware, e por fim noções de desenvolvimento de hardware e PCB.

6. Conclusões

Durante a realização do trabalho, ficou clara a necessidade de se ter um conhecimento um tanto generalista para lidar com plataformas diferentes. Além da programação em Java e em C, conhecimentos em microcontroladores e noções de desenvolvimento de hardware mostraram-se indispensáveis.

A partir dos resultados obtidos, o trabalho mostrou-se executável e com objetivos realistas, através dos materiais e métodos propostos. Em outras palavras, abriu-se um canal de comunicação entre celular e microcontrolador, podendo-se desenvolver soluções que não só possam necessitar de ambos os dispositivos, mas também usem mais funcionalidades que eles oferecem.

O desafio de se desenvolver um aplicativo para Android foi superado graças à quantidade de informações disponíveis para essa plataforma *opensource*. O aplicativo final forneceu uma interface muito simples, de forma que qualquer pessoa possa manuseá-lo e controlar o robô.

Além do mais, o microcontrolador mostrou-se de uso prático e intuitivo, como era de se esperar, e foi capaz de atender aos requisitos esperados. Algumas melhorias ainda podem ser feitas, como a mudança para um código-fonte mais enxuto, e a implementação de funções que envolvam “sleep-mode” para um menor consumo, e portanto um maior tempo de bateria. O programa final teve um tamanho total de 3056 bytes, o que mostra um consumo de memória de cerca de 9,5% dos 32kB totais. Isso mostra que o microcontrolador ainda oferece significativo espaço de memória para se desenvolver aplicações mais sofisticadas e complexas.

Com relação ao tempo de resposta do robô, primeiramente algumas considerações devem ser feitas. A primeira consideração é que o canal de comunicação Bluetooth não é de duas vias. Ou seja, apenas o celular manda informações para o microcontrolador, mas não o contrário. Isso dificulta muito a obtenção e medição do tempo de resposta. A segunda consideração é que por mais que fosse possível obter tal medição, ela não poderia ser exata para todos os

casos, visto que o sistema operacional Android não é de tempo preditivo (*real time operating system*), o que causaria tempos de resposta diferentes para cada caso.

Para finalizar, esse trabalho ofereceu uma oportunidade de implementação de vários conhecimentos obtidos durante o curso de Engenharia Elétrica, além de envolver assuntos fora da grade, como a programação em Java. Portanto, o projeto foi essencial não só para colocar em prática os conhecimentos adquiridos, como também abriu portas para assuntos ainda não tão difundidos no curso, além de oferecer alguma experiência em projetos de engenharia.

6.1 Trabalhos Futuros

O que este trabalho fez foi abrir um caminho de comunicação entre um hardware de baixo nível (microcontrolador e motores) e outro de alto nível e alta complexidade (*smartphone* Android). A partir daqui, com essa comunicação já estabelecida, outros trabalhos podem desenvolver soluções mais complexas, que envolvam, por exemplo, o *smartphone* embarcado ao robô. Algumas sugestões são:

- Abrir comunicação de 2 vias entre celular e microcontrolador.
- Uso do GPS do celular para navegação autônoma do robô com celular embarcado.
- Uso da internet, para se fazer controle do robô de qualquer lugar com acesso a tal com celular embarcado.

Referências Bibliográficas

ANDROID. **Site oficial para desenvolvedores Android.** Disponível em:
< http://developer.android.com/index.html > Acesso em 05.Mai.2014

ARM. **Cortex-M Series.** Disponível em
< http://www.arm.com/products/processors/cortex-m/index.php >
Acesso em 11.Out.2014

BRAY, J. STURMAN, C. F. **Bluetooth 1.1 – Connect without cables.** 2ª edição. Prentice Hall. 2002.

BUSINESSINSIDER. **This Chart Shows Google's Incredible Domination Of The World's Computing Platforms.** Disponível em:
< http://www.businessinsider.com/androids-share-of-the-computing-market-2014-3#ixzz3HO0X0Tlt > Acesso em 02.Jun.2014

GRANDVIEWRESEARCH**. Microcontroller Market Analysis By Product (4/8-bit, 16-bit, 32-bit), By Application (Automotive, Consumer Electronics, Industrial) And Segment Forecasts To 2020.** Disponível em:
<http://www.grandviewresearch.com/industry-analysis/microcontroller-market>
Acesso em 30.Out.2014

IBTIMES. **Android Vs. iOS: What's The Most Popular Mobile Operating System In Your Country?** Disponível em:
< http://www.ibtimes.com/android-vs-ios-whats-most-popular-mobile-operating-system-your-country-1464892 > Acesso em 02.Jun.2014

IDC**. Worldwide *Smartphone* Shipments Edge Past 300 Million Units in the Second Quarter; Android and iOS Devices Account for 96% of the Global Market, According to IDC.** Disponível em
< http://www.idc.com/getdoc.jsp?containerId=prUS25037214 >
Acesso em 10.Out.2014

NXP. **LPC111x/LPC11Cxx User manual.** Disponível em:
< http://www.nxp.com/documents/user_manual/UM10398.pdf >
Acesso em 05.Jun.2014

PHILIPS, B. HARDY, B. **Android Programming: The Big Nerd Ranch Guide.** 2013.

SILVA, Luciano Alves da. **Aprenda Passo a Passo a Programar em Android: Guia Essencial para Desenvolvedores**. 2ª Edição. 2012.

SPARKFUN**. Bluetooth Modem - BlueSMiRF Silver.** Disponível em
<https://www.sparkfun.com/products/12577> Acesso em 03.Mar.2014

XDA-DEVELOPERS. **Accelerometer Data.** Disponível em
<http://forum.xda-developers.com/showthread.php?t=2660904>
Acesso em 02.Mar.2013

YIU, Joseph. **The Definitive Guide to the ARM Cortex-M0**. 2nd Edtion. 2011.

Apêndice A - Software para LPC1114 – LPCXpresso (Arm Cortex-M0)

Programa 1 - Uart_main.c

```
#include "driver_config.h"
#include "target_config.h"
#include "system_LPC11xx.h"
#include "uart.h"
#include <stdbool.h>


#ifdef __USE_CMSIS
#include "LPC11xx.h"
#endif

extern volatile uint32_t UARTCount;
extern volatile uint8_t UARTBuffer[BUFSIZE];
static LPC_GPIO_TypeDef (* const LPC_GPIO[4]) = { LPC_GPIO0, LPC_GPIO1,
LPC_GPIO2, LPC_GPIO3 };


void GPIOInit(void)
      {
      LPC_SYSCON->SYSAHBCLKCTRL |=(1<<6);       //enables clock to GPIO
      LPC_GPIO[0]->DIR|=(1<<7);                       //set pin 0.7 as
output
      LPC_GPIO[3]->DIR|=(1<<2);                       //set pin 3.2 as
output
      LPC_GPIO[1]->DIR|=(1<<8);                       //set pin 1.8 as
output
      LPC_GPIO[0]->DIR|=(1<<6);                       //set pin 0.6 as
output
      LPC_GPIO[3]->DIR|=(1<<3);                       //set pin 3.3 as
output
      LPC_GPIO[2]->DIR|=(1<<8);                       //set pin 2.8 as
output
      GPIO_LED(3,3,0);
      GPIO_LED(2,8,0);
      GPIO_LED(1,8,0);
      GPIO_LED(0,6,0);
}

void TIMERInit(){
      LPC_SYSCON->SYSAHBCLKCTRL|=(1<<9);  //Enables Clock for 32-bit
counter/timer 0

      NVIC_EnableIRQ(TIMER_32_0_IRQn);
      LPC_TMR32B0->CTCR=0x00;       //Selects Timer Mode, every rising
edge of PCLK
      LPC_TMR32B0->TCR=0x02;        //Reset Timer
      LPC_TMR32B0->IR =0xff;        //Reset interrupts
      LPC_TMR32B0->MCR=0x03;        //Config Match Control Register to
interrupt on MR0 and TC will reset if match occurs
```

```
      LPC_TMR32B0->MR0= 1500 * ((SystemCoreClock/(LPC_TMR32B0->PR+1)) /
10000); //1ms time
      LPC_TMR32B0->TCR=0x01;        //START TIMER

}

void GPIO_LED(uint8_t myportValue, uint8_t mybitValue, bool state){

      LPC_GPIO[myportValue]->MASKED_ACCESS[(1<<mybitValue)] =
(state<<mybitValue);


}


int main (void) {

  UARTInit(UART_BAUD);
  GPIOInit();
  TIMERInit();
#if MODEM_TEST
  ModemInit();
#endif

  while (1)
  {                     /* Loop forever */

        while(1);
        UARTCount = 0;
//

  }
}
```

Programa 2 - uart.c

```
/****************************************************************************
*****
 *   uart.c:  UART API file for NXP LPC11xx Family Microprocessors
 *
 *   Copyright(C) 2008, NXP Semiconductor
 *   All rights reserved.
 *
 *   History
 *   2009.12.07  ver 1.00    Preliminary version, first Release
 *
****************************************************************************
*****/
#include "driver_config.h"
#if CONFIG_ENABLE_DRIVER_UART==1
#include "uart.h"
#include "stdbool.h"

volatile uint32_t UARTStatus;
volatile uint8_t  UARTTxEmpty = 1;
volatile uint8_t  UARTBuffer[BUFSIZE];
```

```
volatile uint32_t UARTCount = 0;

#if CONFIG_UART_DEFAULT_UART_IRQHANDLER==1
/*****************************************************************************
*****
** Function name:       UART_IRQHandler
**
** Descriptions:        UART interrupt handler
**
** parameters:                None
** Returned value:            None
**
*****************************************************************************
****/

void updatePWM(int AccelerometerValue){

      switch( AccelerometerValue){


      case 'S':
            GPIO_LED(3,3,0);
            GPIO_LED(2,8,0);
            GPIO_LED(1,8,0);
            GPIO_LED(0,6,0);
            break;

      case 'F':
            GPIO_LED(3,3,0);
            GPIO_LED(2,8,1);
            GPIO_LED(1,8,1);
            GPIO_LED(0,6,0);
            break;

      case 'B':
            GPIO_LED(3,3,1);
            GPIO_LED(2,8,0);
            GPIO_LED(1,8,0);
            GPIO_LED(0,6,1);
            break;

      case 'E':
            GPIO_LED(3,3,1);
            GPIO_LED(2,8,0);
            GPIO_LED(1,8,1);
            GPIO_LED(0,6,0);
            break;

      case 'D':
            GPIO_LED(3,3,0);
            GPIO_LED(2,8,1);
            GPIO_LED(1,8,0);
            GPIO_LED(0,6,1);
            break;

                  }
```

```
}


void TIMER32_0_IRQHandler(void) {
      LPC_TMR32B0->IR =0xff;  //Reset interrupts
      LPC_TMR32B0->TCR=0x00;        //STOP TIMER
      LPC_TMR32B0->TCR=0x02;        //Reset Timer
/*
      //Decide direção: Para qual direção o motor irá girar
      if(direction){
            //GPIO_LED(0,6,0); //Zera porta correspondente (assegurar que
porta estará em nível baixo!)
            //GPIO_LED(3,2,0); //Zera porta correspondente (assegurar que
portas estarão em nível baixo!)
            portValue=3;
            bitValue=2;

      }
      else{
            //GPIO_LED(3,2,0);  //Zera porta correspondente (assegurar
que porta estará em nível baixo!)
            //GPIO_LED(0,6,0); //Zera porta correspondente (assegurar que
porta estará em nível baixo!)
            portValue=0;
            bitValue=6;
      }

      //Intensidade: Sinal de PWM na porta com direção já definida
      if(on==1){
            GPIO_LED(0,7,1);
            GPIO_LED(portValue,bitValue,1);
            on=0;
            LPC_TMR32B0->MR0= (TIMER_FREQUENCY)*(LED_ON) *
((SystemCoreClock/(LPC_TMR32B0->PR+1)) / 10000); //1ms time
      }
      else if (on==0){
            GPIO_LED(0,7,0);
            GPIO_LED(portValue,bitValue,0);
            on=1;
            LPC_TMR32B0->MR0= (TIMER_FREQUENCY)*(LED_OFF) *
((SystemCoreClock/(LPC_TMR32B0->PR+1)) / 10000); //1ms time
      }*/
      LPC_TMR32B0->TCR=0x01;
}

void UART_IRQHandler(void)
{
  int Buffer=0;
  uint8_t IIRValue, LSRValue;
  uint8_t Dummy = Dummy;

  IIRValue = LPC_UART->IIR;

  IIRValue >>= 1;             /* skip pending bit in IIR */
  IIRValue &= 0x07;                 /* check bit 1~3, interrupt
identification */
  if (IIRValue == IIR_RLS)          /* Receive Line Status */
```

```
  {
    LSRValue = LPC_UART->LSR;
    /* Receive Line Status */
    if (LSRValue & (LSR_OE | LSR_PE | LSR_FE | LSR_RXFE | LSR_BI))
    {
      /* There are errors or break interrupt */
      /* Read LSR will clear the interrupt */
      UARTStatus = LSRValue;
      Dummy = LPC_UART->RBR;  /* Dummy read on RX to clear
                                                 interrupt, then bail out
*/
      return;
    }
    if (LSRValue & LSR_RDR)   /* Receive Data Ready */
    {
      /* If no error on RLS, normal ready, save into the data buffer. */
      /* Note: read RBR will clear the interrupt */
      UARTBuffer[UARTCount++] = LPC_UART->RBR;
      if (UARTCount == BUFSIZE)
      {
        UARTCount = 0;        /* buffer overflow */
      }
    }
  }
  else if (IIRValue == IIR_RDA)     /* Receive Data Available */
  {
    /* Receive Data Available */

              Buffer = LPC_UART->RBR;
              updatePWM(Buffer);

    if (UARTCount == BUFSIZE)
    {
      UARTCount = 0;          /* buffer overflow */
    }
  }
  else if (IIRValue == IIR_CTI)     /* Character timeout indicator */
  {
    /* Character Time-out indicator */
    UARTStatus |= 0x100;            /* Bit 9 as the CTI error */
  }
  else if (IIRValue == IIR_THRE)    /* THRE, transmit holding register
empty */
  {
    /* THRE interrupt */
    LSRValue = LPC_UART->LSR;       /* Check status in the LSR to see if
                                               valid data in U0THR or
not */
    if (LSRValue & LSR_THRE)
    {
      UARTTxEmpty = 1;
    }
    else
    {
      UARTTxEmpty = 0;
    }
  }
```

```
  return;
}
#endif

/*****************************************************************************
*****
** Function name:       ModemInit
**
** Descriptions:        Initialize UART0 port as modem, setup pin select.
**
** parameters:                None
** Returned value:            None
**
*****************************************************************************
****/
void ModemInit( void )
{
  LPC_IOCON->PIO2_0 &= ~0x07;    /* UART I/O config */
  LPC_IOCON->PIO2_0 |= 0x01;     /* UART DTR */
  LPC_IOCON->PIO0_7 &= ~0x07;    /* UART I/O config */
  LPC_IOCON->PIO0_7 |= 0x01;     /* UART CTS */
  LPC_IOCON->PIO1_5 &= ~0x07;    /* UART I/O config */
  LPC_IOCON->PIO1_5 |= 0x01;     /* UART RTS */
#if 1
  LPC_IOCON->DSR_LOC    = 0;
  LPC_IOCON->PIO2_1 &= ~0x07;    /* UART I/O config */
  LPC_IOCON->PIO2_1 |= 0x01;     /* UART DSR */

  LPC_IOCON->DCD_LOC    = 0;
  LPC_IOCON->PIO2_2 &= ~0x07;    /* UART I/O config */
  LPC_IOCON->PIO2_2 |= 0x01;     /* UART DCD */

  LPC_IOCON->RI_LOC     = 0;
  LPC_IOCON->PIO2_3 &= ~0x07;    /* UART I/O config */
  LPC_IOCON->PIO2_3 |= 0x01;     /* UART RI */

#else
  LPC_IOCON->DSR_LOC = 1;
  LPC_IOCON->PIO3_1 &= ~0x07;    /* UART I/O config */
  LPC_IOCON->PIO3_1 |= 0x01;     /* UART DSR */

  LPC_IOCON->DCD_LOC = 1;
  LPC_IOCON->PIO3_2 &= ~0x07;    /* UART I/O config */
  LPC_IOCON->PIO3_2 |= 0x01;     /* UART DCD */

  LPC_IOCON->RI_LOC = 1;
  LPC_IOCON->PIO3_3 &= ~0x07;    /* UART I/O config */
  LPC_IOCON->PIO3_3 |= 0x01;     /* UART RI */
#endif
  LPC_UART->MCR = 0xC0;          /* Enable Auto RTS and Auto CTS. */

  return;
}

/*****************************************************************************
*****
** Function name:       UARTInit
```

```
**
** Descriptions:        Initialize UART0 port, setup pin select,
**                      clock, parity, stop bits, FIFO, etc.
**
** parameters:                UART baudrate
** Returned value:            None
**
*****************************************************************************
****/
void UARTInit(uint32_t baudrate)
{
  uint32_t Fdiv;
  uint32_t regVal;

  UARTTxEmpty = 1;
  UARTCount = 0;

  NVIC_DisableIRQ(UART_IRQn);

  LPC_IOCON->PIO1_6 &= ~0x07;    /*  UART I/O config */
  LPC_IOCON->PIO1_6 |= 0x01;     /* UART RXD */
  LPC_IOCON->PIO1_7 &= ~0x07;
  LPC_IOCON->PIO1_7 |= 0x01;     /* UART TXD */
  /* Enable UART clock */
  LPC_SYSCON->SYSAHBCLKCTRL |= (1<<12);
  LPC_SYSCON->UARTCLKDIV = 0x1;     /* divided by 1 */

  LPC_UART->LCR = 0x83;             /* 8 bits, no Parity, 1 Stop bit */
  regVal = LPC_SYSCON->UARTCLKDIV;

  Fdiv = (((SystemCoreClock*LPC_SYSCON-
>SYSAHBCLKDIV)/regVal)/16)/baudrate ;     /*baud rate */

  LPC_UART->DLM = Fdiv / 256;
  LPC_UART->DLL = Fdiv % 256;
  LPC_UART->LCR = 0x03;       /* DLAB = 0 */
  LPC_UART->FCR = 0x07;       /* Enable and reset TX and RX FIFO. */

  /* Read to clear the line status. */
  regVal = LPC_UART->LSR;

  /* Ensure a clean start, no data in either TX or RX FIFO. */
// CodeRed - added parentheses around comparison in operand of &
  while (( LPC_UART->LSR & (LSR_THRE|LSR_TEMT)) != (LSR_THRE|LSR_TEMT) );
  while ( LPC_UART->LSR & LSR_RDR )
  {
      regVal = LPC_UART->RBR; /* Dump data from RX FIFO */
  }

  /* Enable the UART Interrupt */
  NVIC_EnableIRQ(UART_IRQn);

#if CONFIG_UART_ENABLE_INTERRUPT==1
#if CONFIG_UART_ENABLE_TX_INTERRUPT==1
  LPC_UART->IER = IER_RBR | IER_THRE | IER_RLS; /* Enable UART interrupt
*/
#else
```

```
  LPC_UART->IER = IER_RBR | IER_RLS;      /* Enable UART interrupt */
#endif
#endif
  return;
}

/*****************************************************************************
*****
** Function name:       UARTSend
**
** Descriptions:        Send a block of data to the UART 0 port based
**                      on the data length
**
** parameters:          buffer pointer, and data length
** Returned value:      None
**
*****************************************************************************
****/
void UARTSend(uint8_t *BufferPtr, uint32_t Length)
{

  while ( Length != 0 )
  {
        /* THRE status, contain valid data */
#if CONFIG_UART_ENABLE_TX_INTERRUPT==1
        /* Below flag is set inside the interrupt handler when THRE
occurs. */
      while ( !(UARTTxEmpty & 0x01) );
        LPC_UART->THR = *BufferPtr;
      UARTTxEmpty = 0;  /* not empty in the THR until it shifts out */
#else
        while ( !(LPC_UART->LSR & LSR_THRE) );
        LPC_UART->THR = *BufferPtr;
#endif
      BufferPtr++;
      Length--;
  }
  return;
}
#endif

/*****************************************************************************
******
**                            End Of File
*****************************************************************************
*****/
```

Apêndice B - Software para Samsung Galaxy S4 mini

MainActivity.java

```
package com.example.bluetooth1;

import java.io.IOException;
import java.io.OutputStream;
import java.lang.reflect.Method;
import java.util.UUID;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.content.Intent;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Build;
import android.os.Bundle;
import android.os.PowerManager;
import android.util.Log;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity implements SensorEventListener {
  private static final String TAG = "bluetooth1";

  Button btnOn, btnOff;

  private BluetoothAdapter btAdapter = null;
  private BluetoothSocket btSocket = null;
  private OutputStream outStream = null;
```

```
  private float mLastX, mLastY, mLastZ;
  private boolean mInitialized;
  private SensorManager mSensorManager;
  private Sensor mAccelerometer;
  private final float NOISE = (float) 0.5;
  protected PowerManager.WakeLock mWakeLock;

  // SPP UUID service
  private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

  // MAC-address of Bluetooth module (you must edit this line)
  private static String address = "00:06:66:46:5D:B0";

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);

    final PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
    this.mWakeLock = pm.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK, "My Tag");
    this.mWakeLock.acquire();




    btAdapter = BluetoothAdapter.getDefaultAdapter();
    checkBTState();
/*
    btnOn.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        sendData("9");
        Toast.makeText(getBaseContext(), "Turn on LED", Toast.LENGTH_SHORT).show();
      }
    });

    btnOff.setOnClickListener(new OnClickListener() {
```

```
    public void onClick(View v) {
      sendData("0");
      Toast.makeText(getBaseContext(), "Turn off LED", Toast.LENGTH_SHORT).show();
    }
  });*/

  mInitialized = false;
  mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
  mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
  mSensorManager.registerListener(this, mAccelerometer , SensorManager.SENSOR_DELAY_NORMAL);

 }

 private BluetoothSocket createBluetoothSocket(BluetoothDevice device) throws IOException {
    if(Build.VERSION.SDK_INT >= 10){
       try {
          final Method  m = device.getClass().getMethod("createInsecureRfcommSocketToServiceRecord",
new Class[] { UUID.class });
          return (BluetoothSocket) m.invoke(device, MY_UUID);
       } catch (Exception e) {
          Log.e(TAG, "Could not create Insecure RFComm Connection",e);
       }
    }
    return  device.createRfcommSocketToServiceRecord(MY_UUID);
 }

 @Override
 public void onResume() {
  super.onResume();

  Log.d(TAG, "...onResume - try connect...");

  // Set up a pointer to the remote node using it's address.
  BluetoothDevice device = btAdapter.getRemoteDevice(address);

  // Two things are needed to make a connection:
  //   A MAC address, which we got above.
  //   A Service ID or UUID.  In this case we are using the
```

```
    //     UUID for SPP.

    try {
        btSocket = createBluetoothSocket(device);
    } catch (IOException e1) {
        errorExit("Fatal Error", "In onResume() and socket create failed: " + e1.getMessage() + ".");
    }

    // Discovery is resource intensive.  Make sure it isn't going on
    // when you attempt to connect and pass your message.
    btAdapter.cancelDiscovery();

    // Establish the connection.  This will block until it connects.
    Log.d(TAG, "...Connecting...");
    try {
      btSocket.connect();
      Log.d(TAG, "...Connection ok...");
    } catch (IOException e) {
      try {
        btSocket.close();
      } catch (IOException e2) {
        errorExit("Fatal Error", "In onResume() and unable to close socket during connection failure" +
e2.getMessage() + ".");
      }
    }

    // Create a data stream so we can talk to server.
    Log.d(TAG, "...Create Socket...");

    try {
      outStream = btSocket.getOutputStream();
    } catch (IOException e) {
      errorExit("Fatal Error", "In onResume() and output stream creation failed:" + e.getMessage() + ".");
    }
    mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
  }
```

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

  Log.d(TAG, "...In onPause()...");

  if (outStream != null) {
    try {
      outStream.flush();
    } catch (IOException e) {
      errorExit("Fatal Error", "In onPause() and failed to flush output stream: " + e.getMessage() + ".");
    }
  }

  try     {
    btSocket.close();
  } catch (IOException e2) {
    errorExit("Fatal Error", "In onPause() and failed to close socket." + e2.getMessage() + ".");
  }
  mSensorManager.unregisterListener(this);
}

private void checkBTState() {
  // Check for Bluetooth support and then check to make sure it is turned on
  // Emulator doesn't support Bluetooth and will return null
  if(btAdapter==null) {
    errorExit("Fatal Error", "Bluetooth not support");
  } else {
    if (btAdapter.isEnabled()) {
      Log.d(TAG, "...Bluetooth ON...");
    } else {
      //Prompt user to turn on Bluetooth
      Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
      startActivityForResult(enableBtIntent, 1);
    }
  }
}
```

```
  private void errorExit(String title, String message){
    //Toast.makeText(getBaseContext(), title + " - " + message, Toast.LENGTH_LONG).show();
    finish();
  }

  private void sendData(String message) {
    byte[] msgBuffer = message.getBytes();

    Log.d(TAG, "...Send data: " + message + "...");

    try {
      outStream.write(msgBuffer);
    } catch (IOException e) {
      String msg = "In onResume() and an exception occurred during write: " + e.getMessage();
      if (address.equals("00:00:00:00:00:00"))
        msg = msg + ".\n\nUpdate your server address from 00:00:00:00:00:00 to the correct address on line 35
in the java code";
        msg = msg +  ".\n\nCheck that the SPP UUID: " + MY_UUID.toString() + " exists on server.\n\n";

        errorExit("Fatal Error", msg);
    }
  }
 /*SENSOR ACCELEROMETER*/
  @Override
          public void onAccuracyChanged(Sensor sensor, int accuracy) {
                  // can be safely ignored for this demo
          }

  @Override
          public void onSensorChanged(SensorEvent event) {
            TextView tvX= (TextView)findViewById(R.id.x_axis);
            TextView tvY= (TextView)findViewById(R.id.y_axis);
            TextView tvZ= (TextView)findViewById(R.id.z_axis);
                  float x = event.values[0];
                  float y = event.values[1];
                  float z = event.values[2];

                  //Decide se o movimento é "Forward" ou "Backward"
```

```
                int absZ= Math.abs((int) z);
                int absY= Math.abs((int) y);

                if(absZ < 3 && absY < 3)
                        sendData("S");  // Fica parado.

                else{
                        if(absZ > absY) {                       // Movimento é p/ Frente ou p/ Trás
                                if (z < 0)
                                        sendData("B"); // Vai para trás (Backward)
                                else
                                        sendData("F"); // Vai para frente (Forward)
                        }
                        else{                                   //Movimento é p/ Esquerda ou Direita
                                if (y < 0)
                                        sendData("E"); // Vai para Esquerda
                                else
                                        sendData("D"); // Vai para Direita
                        }
                }

                tvX.setText(Float.toString(x));
                tvY.setText(Float.toString(y));
                tvZ.setText(Float.toString(z));



}
 @Override
 public void onDestroy() {
    this.mWakeLock.release();
    super.onDestroy();
 }

}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
      <TextView
            android:paddingTop="20dip"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:textSize="16sp"
          android:textStyle="bold"
          android:gravity="center"
            android:text="Shaker Demo"/>
      <TableLayout
            android:paddingTop="10dip"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:stretchColumns="*">
            <TableRow>
                  <TextView
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:textSize="14sp"
                  android:text="X-Axis"
                  android:gravity="center"/>
            <TextView
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:textSize="14sp"
                  android:text="Y-Axis"
                  android:gravity="center"/>
            <TextView
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:textSize="14sp"
                  android:text="Z-Axis"
                  android:gravity="center"/>
            </TableRow>
            <TableRow>
                  <TextView
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:id="@+id/x_axis"
                  android:gravity="center"/>
            <TextView
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:id="@+id/y_axis"
                  android:gravity="center"/>
            <TextView
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
```

```
                android:id="@+id/z_axis"
                android:gravity="center"/>
        </TableRow>
    </TableLayout>
    <ImageView
        android:paddingTop="10dip"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/image"
        android:layout_gravity="center"
        android:visibility="invisible"/>
</LinearLayout>
```