Universidade do Estado do Rio de Janeiro

Centro de Tecnologia e Ciências

Faculdade de Engenharia

Lenielson Rodrigues de Sousa

Acionamento dos Motores CC de uma Embarcação Teleoperada

Rio de Janeiro

2016

Lenielson Rodrigues de Sousa

Acionamento dos Motores CC de uma Embarcação Teleoperada

Projeto de graduação apresentado, como requisito parcial para obtenção do grau de Engenheiro Eletricista, à Faculdade de Engenharia, da Universidade do Estado do Rio de Janeiro.

Orientador: Prof. Dr. José Paulo Vilela Soares da Cunha

Rio de Janeiro

2016

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

S725 Sousa, Lenielson Rodrigues de.
Acionamento dos motores CC de uma embarcação teleoperada / Lenielson Rodrigues de Sousa. – 2016.
125f.

Orientador: José Paulo Vilela Soares da Cunha.
Projeto Final (Graduação) - Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.
Bibliografia p. 106-108

1. Engenharia elétrica. 2. Motores - Acionamento eletrônico. 3. Padrão IEEE 802.15.4 - Protocolo ZigBee. I. Cunha, Jose Paulo Vilela da. II. Universidade do Estado do Rio de Janeiro. III. Título.

CDU 621.3

Lenielson Rodrigues de Sousa

Acionamento dos Motores CC de uma Embarcação Teleoperada

Projeto de graduação apresentado, como requisito parcial para obtenção do grau de Engenheiro Eletricista, à Faculdade de Engenharia, da Universidade do Estado do Rio de Janeiro.

Aprovado em 19 de janeiro de 2016.

Banca Examinadora:

_____

Prof.Dr. José Paulo Vilela Soares da Cunha (Orientador)

Faculdade de Engenharia - UERJ

_____

Prof.Dr. Jorge Duarte Pires Valério

Faculdade de Engenharia - UERJ

_____

Prof. Dra. Maria Dias Bellar

Faculdade de Engenharia - UERJ

Rio de Janeiro

2016

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus por ter me guiado. Por ter me dado força, coragem e perseverança para realizar meu sonho de ser engenheiro.

Agradeço aos meus pais, fruto de minha motivação, José Rodrigues e Zulene Lourenço pelo suporte em todos os momentos e aos meus irmãos (Thiago, Rodrigo, João e Marcelo).

Ao meu orientador professor Dr. José Paulo Vilela Soares da Cunha pela paciência e os valorosos conselhos.

A FAPERJ e ao CNPq pelos recursos necessários a realização deste trabalho.

A minha namorada, amiga e companheira Viviane de Brito.

A todos os professores que me incentivaram a estudar mesmo com todas adversidades, em especial aos professores Renata Borelli (Primário) e o professor Paulinni (Ensino médio).

A todos os amigos que fiz durante minha jornada na Faculdade de Engenharia da UERJ, nos estágios e trabalhos onde passei.

Aos meus velhos amigos da oitava série. Eterna T84.

A meu amigo Paulo Batista por sempre acreditar no meu potencial e pela amizade de longa data.

DEDICATÓRIA

Dedico esse trabalho à memória do meu amigo Daniel Vieira de Moura.

RESUMO

SOUSA, Lenielson Rodrigues de. *Acionamento dos Motores CC de uma Embarcação Teleoperada*. 125f. Projeto Final (Graduação em Engenharia Elétrica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2016.

Neste projeto de graduação é desenvolvido um sistema de acionamento para os motores de uma embarcação teleoperada. O sistema realiza a aquisição de dados de sensores, controle do motor e comunicação baseado em plataforma microcontrolada Arduino. O acionamento dos motores é realizado por um conversor CC-CC ponte H. Foi desenvolvido um controle remoto sem fio para envio de comandos aos motores e recebimento das leituras dos sensores. Este também usa plataforma microcontrolada Arduino para sua implementação. A comunicação do controle remoto com os motores é realizada por meio de uma rede de trabalho com protocolo de comunicação sem fio denominada ZigBee. Para a criação da rede sem fio foram utilizados módulos XBee em modo de comunicação API (application programming interface) que permite o endereçamento dos módulos na rede e a troca de dados de forma confiável.

Palavras-chave: Acionamento de motor. Propulsor. Arduino. ZigBee. XBee.

ABSTRACT

SOUSA, Lenielson Rodrigues de. *Acionamento dos Motores CC de uma Embarcação Teleoperada*. 125f. Projeto Final (Graduação em Engenharia Elétrica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2016.

In this undergraduate degree final project, a motor drive system of a remotely operated surface vessel is developed. The system performs the acquisition of sensor data, control and communication by means of Arduino microcontroller. The drive circuit of the motors is performed by a H-bridge Dc-Dc converter. A wireless remote control was developed to send commands to the motors and receive readings. This is also accomplished by an Arduino platform. The communication of the remote control with the motors is carried out by a wireless network with a communication protocol called ZigBee. The wireless network is composed of XBee modules on API (application programming interface) communication mode that allows the addressing of the modules on wireless network and reliable data exchange.

Keywords: Motor drive. Thruster. Arduino. ZigBee. XBee.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Introdução

Nosso planeta possui uma grande e rica biodiversidade que intriga o ser humano desde os seus primórdios, para o entendimento das complexas relações entre as diversas formas de vida, têm-se o desafio de estudar seus ecossistemas em especial os mares, rios e a atmosfera. Para a obtenção de dados para as pesquisas sobre fenômenos naturais, são utilizados satélites que propiciam uma observação de forma remota ou através de observações experimentais. Para obtenção de dados *in loco* que são necessários para a calibração dos instrumentos de monitoração remota (HIGINBOTHAM et al., 2008) geralmente utilizam-se bóias ou embarcações especificas para esta finalidade, sendo que o lançamento e operação destes sistemas são bastante onerosos. Com o avanço da tecnologia especialmente no campo da eletrônica embarcada, existem diversos veículos de pesquisa não tripulados que trazem uma opção para a realização de pesquisas e aquisição de dados de forma segura, precisa e menos invasiva. Destes veículos pode-se destacar: aviões, veículos terrestres, submarinos e embarcações de superfície, sendo este último o foco para este projeto de graduação. Os veículos de superfície podem ser divididos em dois tipos:

- USV (*Unmanned Surface Vehicle*);
- ASV (*Autonomous Surface Vehicle*).

Os USVs são veículos sem tripulação, mas com comandos humanos realizados a distância. Os ASVs podem realizar as tarefas que foram designadas sem intervenção direta de humanos. As vantagens no uso desses veículos não tripulados são especialmente percebidas quando são aplicados para obtenção de dados e realização de pesquisas em ambientes poluídos, ambientes em condições climáticas extremas ou quando se faz necessário um longo período de tempo para obtenção de informações. Mais alguns exemplos de aplicações dos USVs e ASVs são mostrados a seguir:

- Estudos marinhos e fluviais;
- Inspeção de cascos de navios ancorados;
- Vigilância costeira contínua e sem a presença humana;
- Caçar minas subaquáticas em regiões que apresentem muito risco para as tripulações de embarcações;

- Indústria petrolífera.

Figura 1: USV usado para fins militares. (Extraído de http://defense-update.com/products/s/stingray.htm).

Figura 2: ASV usado na aquisição de dados meteorológicos. (Extraído de http://asvglobal.com/product/c-enduro/).

A utilização de veículos de superfície não tripulados já é uma realidade e se torna uma tendência em todo o mundo para as mais diversas aplicações, neste contexto surge a realização e motivação para este projeto.

1. Objetivo

Este trabalho tem como objetivo desenvolver um sistema eletrônico para o controle de velocidade dos motores de corrente contínua a serem utilizados no Projeto de Pesquisa: Embarcações Não Tripuladas para Monitoração Ambiental e Defesa. O motor utilizado (Figura 3(a)) conta originalmente com uma haste de comando rotativa que comuta bobinas internas do motor por meio de uma chave e efetua a mudança de 5 marchas a frente e duas a ré.

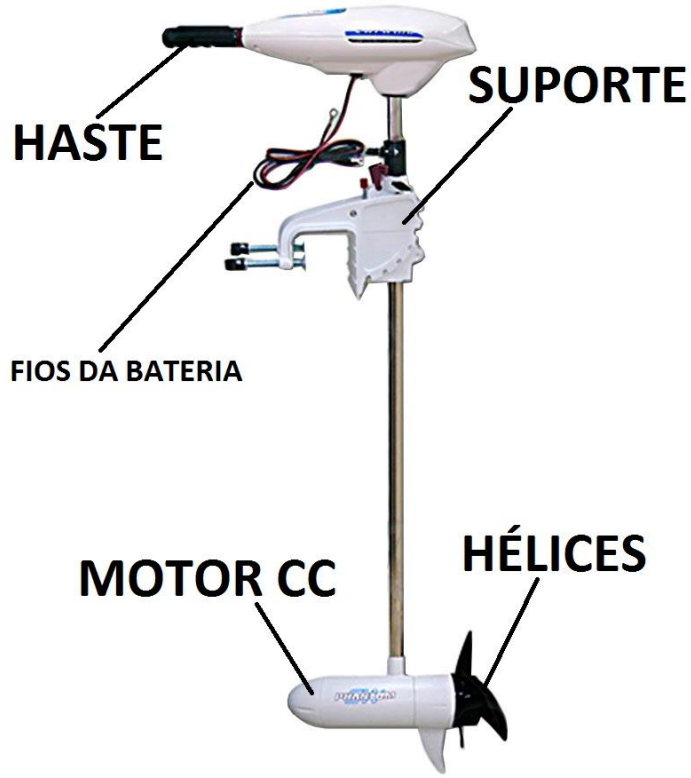


Figura 3: Motor da embarcação.

(SCHULTZE, 2012) realizou em seu projeto de graduação um estudo das características da embarcação objeto deste projeto, e implementou um sistema de comando de velocidade melhorado e de baixo custo utilizando relés, um microcontrolador e um circuito de interface.

O sistema desenvolvido neste Projeto de Graduação é capaz de realizar o controle velocidade da embarcação de forma progressiva, sem gerar solavancos, uma vez que os motores devem trabalhar em conjunto a fim de melhorar a estabilidade, precisão nos movimentos e facilitar o posicionamento da embarcação. Foi aplicada a técnica de controle conhecida como modulação

por largura de pulso (*Pulse Width Modulation* - PWM). Este controle é aplicado em um conversor do tipo ponte H completa que é capaz de controlar a magnitude e a polaridade da tensão contínua de saída (MOHAN et al., 2002, p. 188), ou seja, controla o valor médio da tensão que alimenta o motor e também é capaz de fazer a reversão do sentido de deslocamento da embarcação.

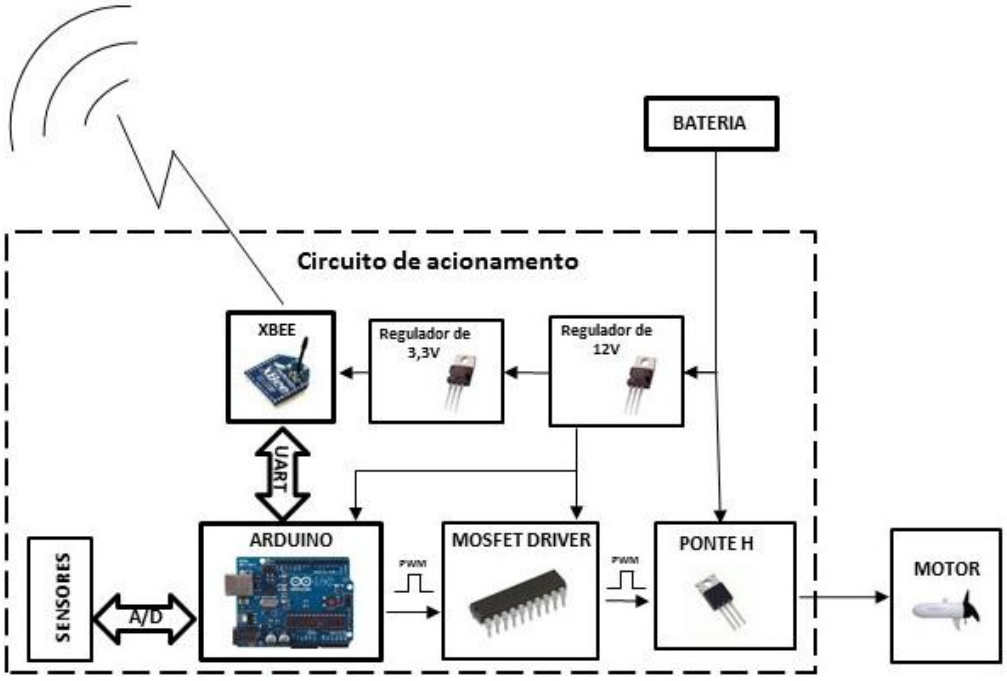


Figura 4: Diagrama em blocos do circuito de acionamento de um motor.

O sistema desenvolvido é composto pelas seguintes partes. Placa de acionamento dos motores e controle remoto sem fio.

- Placa de acionamento.

A embarcação pode contar com dois ou mais motores de corrente contínua. Para cada um desses motores foi desenvolvido um circuito eletrônico para realizar o acionamento conforme visto na Figura 4.

Uma plataforma microcontrolada Arduino gera os pulsos PWM, lê informações dos sensores e realiza a comunicação da embarcação com um controle remoto *wireless* através de um protocolo de comunicação sem fio denominado *ZigBee* que é implementado pelo módulo *XBee*.

- Controle remoto sem fio.

A Figura 5 mostra o diagrama em blocos do controle remoto sem fio desenvolvido. Suas principais funções são enviar comandos aos circuitos de acionamentos dos motores, receber os dados dos sensores e os apresentar ao operador. Com isso, é possível saber a qualquer momento as condições de funcionamento dos propulsores da embarcação.

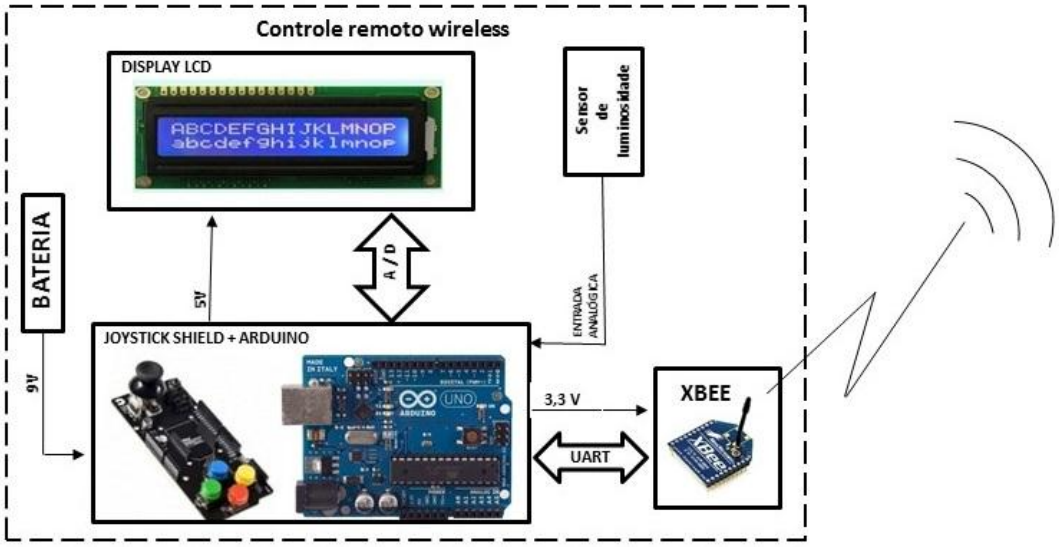


Figura 5: Diagrama em blocos do controle remoto sem fio.

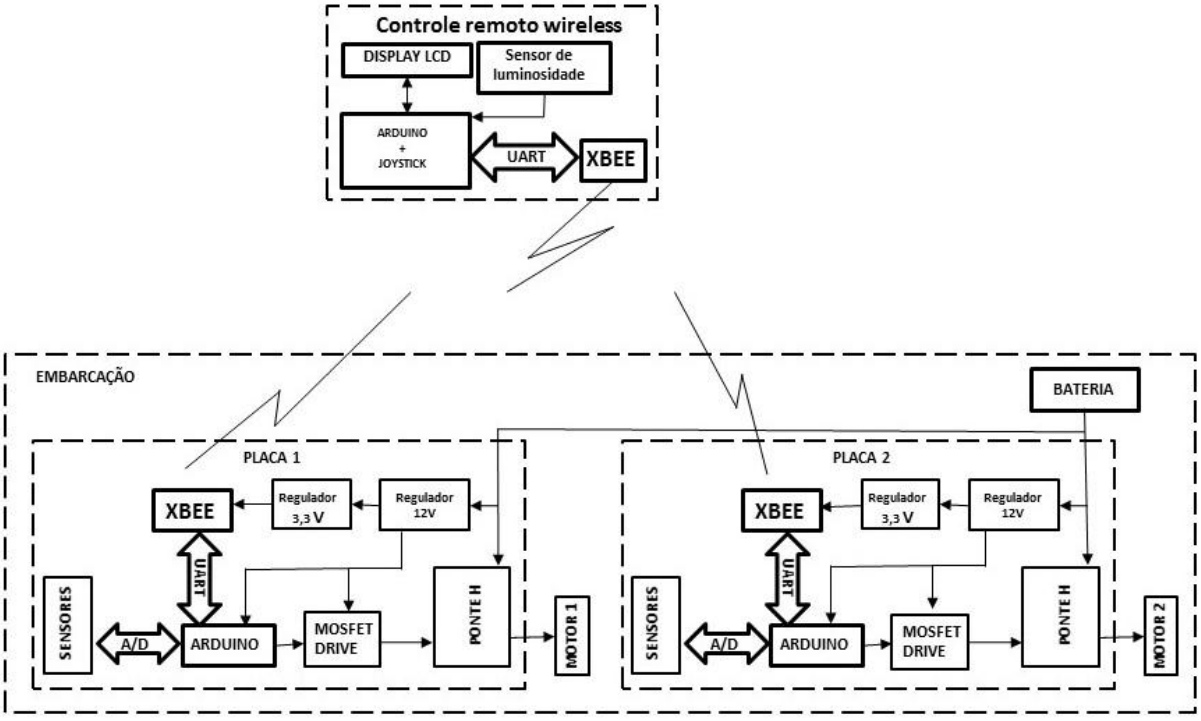


Figura 6: Diagrama em blocos do controle dos propulsores.

O diagrama em blocos do sistema completo pode ser visto na Figura 6.

1.1. Escopo do projeto

No início, esse trabalho de graduação foi dividido em dois eixos principais *hardware* e *software*. A parte de *hardware* inicialmente ficou a cargo do aluno de graduação Daniel Vieira de Moura e a parte de *software* com o autor do presente projeto. Estas duas partes iriam

compor o projeto como um todo. Infelizmente, o aluno Daniel faleceu no decorrer do projeto antes de sua conclusão. Nosso orientador sugeriu que se mantivesse o formato de divisão inicial, sendo aproveitado o valoroso esforço do aluno Daniel na montagem do protótipo do circuito de acionamento e ponte H.

O escopo desse projeto abrange os seguintes itens:

- Descrição do *hardware e software* desenvolvidos;
- Testes de funcionamento, correção e sugestões de solução de problemas encontrados na placa de circuito de acionamento e na montagem do protótipo;
- Desenvolvimento de *software* em plataforma Arduino para receber comandos de velocidade e sentido de rotação, efetuar o controle dos motores, realizar aquisição dos sinais dos sensores e sua transmissão usando protocolo sem fio *ZigBee*.
- Desenvolvimento de *software* e *hardware* com plataforma Arduino para recebimento e leitura dos sensores, envio de comandos para os motores*;*
- Configuração de todos os compontes do sistema.

1.2. Organização do texto

O texto foi organizado como segue. O Capítulo 2 apresenta os sensores utilizados e suas características. O Capítulo 3 descreve o circuito de acionamento do motor. O capítulo 4 descreve as características do microcontrolador Arduino e sua programação. O Capítulo 5 trata a comunicação eletrônica sem fio e descreve o protocolo *ZigBee*, módulos XB*ee* e sua programação. O Capítulo 6 descreve o sistema desenvolvido. O Capítulo 7 descreve os principais testes realizados. O Capítulo 8 apresenta as conclusões, algumas contribuições que o projeto pode oferecer e propostas para trabalhos futuros.

2. Sensores

Um sensor geralmente é definido como um dispositivo que recebe um estímulo e responde com um ou sinal elétrico (FRADEN, 2003, p. 2). O objetivo de um sensor é receber uma entrada (estímulo) e a partir disto gerar um sinal eletrico compatível com os circuitos no qual estão inseridos, quando se tem interesse em medir alguma grandeza física seja essa de natureza elétrica ou não utilizamos os sensores que convertem essa grandeza de interesse em sinais elétricos de saída em forma de corrente, tensão ou carga para que estes possam ser amplificados, processados, analisados ou utilizados de acordo com a necessidade específica de cada aplicação. O termo sensor deve ser distiguindo de transdutores (FRADEN, 2003, p. 3), transdutor converte um tipo de energia em outro enquanto um sensor converte qualquer forma de energia em energia elétrica, há basicamente dois tipos de sensores: sensores diretos e complexos, um sensor direto é o que emprega um determinado efeito fisico para fazer uma conversão direta de energia, já um sensor complexo emprega um ou mais transdutores de energia antes de um sensor direto ser empegado para gerar um sinal elétrico de saída em seu último estágio, podemos ver esse tipo de sensor na Figura 7.

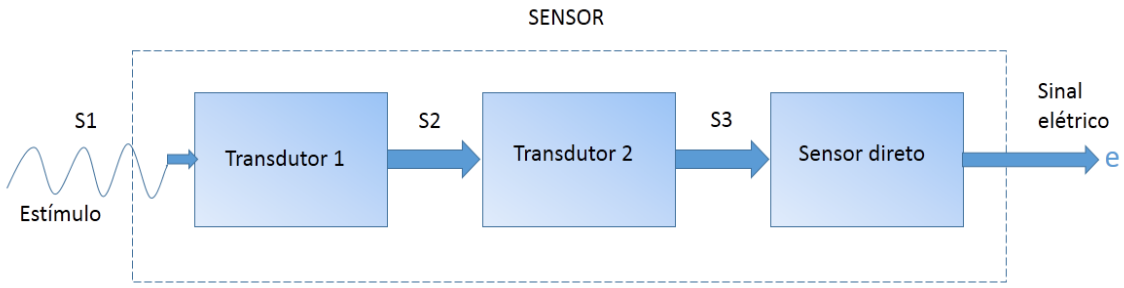


Figura 7: Exemplo de um sensor complexo onde o último estágio de saída é um sensor simples, fonte (FRADEN, 2003, p. 4).

Neste projeto de graduação serão utilizados sensores para medir três 3 grandezas físicas:

- Corrente;
- Tensão;
- Temperatura.

As leituras das grandezas físicas mensionadas são feitas nas entradas analógicas do conversor A/D (analógico / digital) do microcontrolador Arduino com a seguinte equação:

$$V = \frac{V_{REF}}{2^n} E_{Ai} \quad (1)$$

Na qual, V é o valor da grandeza que queremos medir em volts, *n* é a resolução do conversor A/D e V_{REF} é a tensão de referência. No caso do Arduino UNO *n*=10 *bits* e V_{REF} vem padronizado com o valor de 5V que é a tensão de alimentação V_{CC}, mas pode ser alterada para um outro valor de interesse na faixa de 0 a V_{CC} aplicada no pino A_{REF}.

E_{Ai} é o valor inteiro resultante da conversão A/D da entrada analógica Ai, onde i = (0, 1, 2, ..., n-1) e n é o número de entradas analógicas do Arduino empregado. A leitura é um número binário que esta na faixa de 0 a 1023. Substituindo-se os valores anteriormente citados obtem-se em *milivolts*:

$$V = \frac{V_{REF}}{2^n} E_{Ai} = \frac{5}{2^{10}} E_{Ai} = 4{,}88\, E_{Ai} \quad (2)$$

Ou seja, o passo incremental das leituras para um V_{REF} de 5V é de 4,88mV e, como será visto, é aproximadamente a metade do fator de escala do sensor de temperatura (10mV/ºc) e aproximadamente um quarto valor do fator de escala do sensor de corrente (20mV/A) e a tensão da bateria e nos *MOSFETs* é na ordem de alguns volts o que em princípio torna o valor de V_{REF} = 5V adequado ao projeto.

2.1. Sensor de corrente

Para que o sistema funcione corretamente e com segurança precisamos medir a corrente da bateria e a corrente nos motores de forma que estas apresentem valores esperados de acordo com o regime de funcionamento em que o motor se encontre. Suponhamos que a hélice do motor fique presa numa rede, por exemplo, e que o rotor trave, o aumento de corrente e consequentemente o aquecimento gerado em decorrência disto poderia danificar os circuitos e o motor. Em testes realizados no laboratório, foi observado que os motores podem atingir picos de correntes de até 80A na partida, logo o sensor de corrente para esta aplicação precisa ser bastante robusto e suportar as solicitações de correntes exigidas pelo motor. O sensor escolhido é o ACS756SCA-100B-PFF-T (ROSÁRIO, 2013, p.22) do fabricante *Allegro™ microsystens*. Este sensor funciona com o principio do efeito HALL, é um sensor bipolar capaz de medir corrente continua ou alternada na faixa de -100A a +100A. Abaixo temos um esquema das aplicações típicas desse sensor. A tensão de saída V_{out} varia linearmente de V_{CC}/ 2 a 0 para correntes negativas e de V_{CC}/ 2 até V_{CC} para correntes positivas na proporção de 20mV/A.

Os pontos A e B da Figura 9 são ligados em série com o motor, o resistor R8 e o capacitor C10 que corrrespondem ao R_F e C_F da Figura 8, respectivamente, formam um filtro passa-baixas para atenuar ruídos que poderiam prejudicar a leiturada do microprocessador e o capacitor C11 que corrresponde ao C_{BYP} é um capacitor de *bypass* para eliminar os componentes de alta frequência vindos da fonte de 5V que causariam variação na alimentação (ALLEGRO MICROSYSTEMS, 2011, p. 4) do sensor e com isso variações em suas leituras.

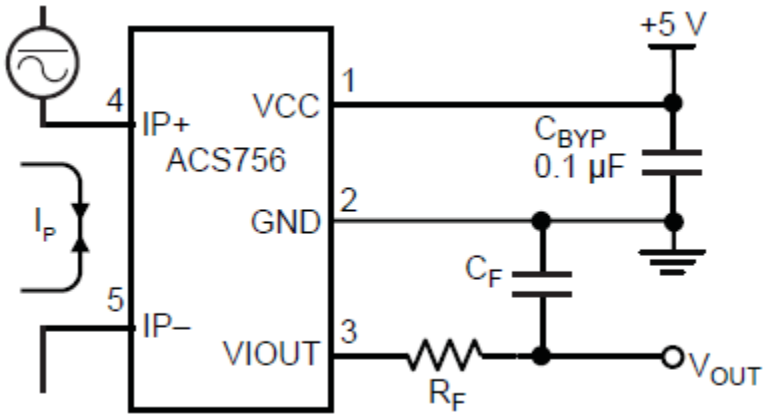


Figura 8: Esquema de ligação do sensor de corrente, extraído de ALLEGRO MICROSYSTEMS, 2011, p. 1

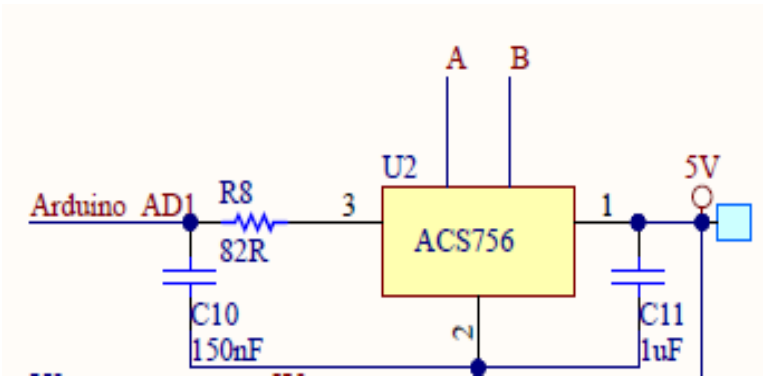


Figura 9: Sensor de corrente (Parte do diagrama do sistema vide Apêndice A).

A leitura desse sensor é realizada na entrada analógica A1 do Arduino, através da seguinte equação:

$$I = \frac{V_{REF}}{2^{10}} \frac{1}{0{,}02} (E_{A1} - 512) \tag{3}$$

Na qual E_{A1} é a leitura analógica feita pela porta A_1 do Arduino. I é a leitura em em amperes da corrente.

Repare que em (3) há essencialmete duas mudanças em relação á equação (1). A primeira é o fator de escala do sensor de 0,02 (20mV/A) que realiza a conversão da leitura de acordo com a característica de saída do sensor utilizado, sem esse ajuste leríamos a corrente como um valor de 0 a 1023 multiplicada pelo fator de 4,88mV, mas não teríamos uma noção imediata da corrente medida. A segunda mudança é a correção com o fator 512 que possibilita o ajuste de zero ampères exatamente neste valor, valores acima de 512 correspondem a correntes positivas e valores abaixo de 512 correspondem a correntes negativas. Para a medição da corrente na bateria foi utilizada a porta analógica A0 do arduino e valem as mesmas considerações que foram feitas para o sensor de corrente dos motores.

2.2. Sensor de temperatura

Optamos por um sensor de temperatura muito simples, confiável e de baixo custo o LM35 da marca TEXAS INSTRUMENTS, visto na Figura 10. É um sensor de temperatura de precisão, cuja saída é uma tensão linearmente proporcional a temperatura com um incremento de 10mV / ºC. Não requer nenhum tipo de calibração para obter temperaturas com precisão de ¼ ºC em temperatura ambiente, pode operar numa faixa (*Full range*) de temperatura que varia de -55ºC a 150ºC, ou em uma configuração básica (Figura 11) no qual trabalha na faixa de operação de 2ºC a 150ºC (com uma fonte de 4V a 30V) o que é mais que suficiente para nossa aplicação. Drena apenas 60uA o que é irrisório em termos de consumo de corrente se comparado com o circuito completo e seu auto aquecimento é de apenas 0,1ºC.

Abaixo, na Tabela 1, são apresentadas as caracteristicas extraídas do manual deste sensor e na Figura 12 vemos um trecho do circuito que corresposde ao sensor de temperatura. Nesse sensor foi adicionado um filtro RC (ROSÁRIO, 2013, p. 27) a fim de atenuar interferências eletromagnéticas.

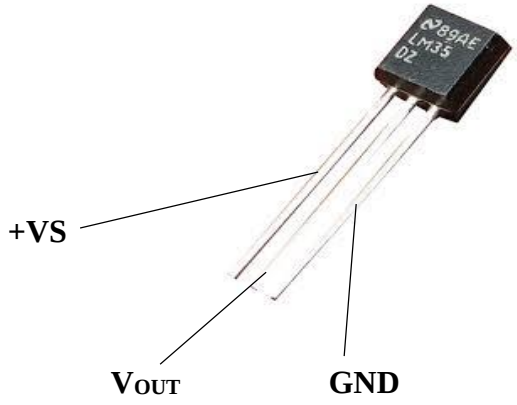


Figura 10: Sensor de temperatura LM 35.

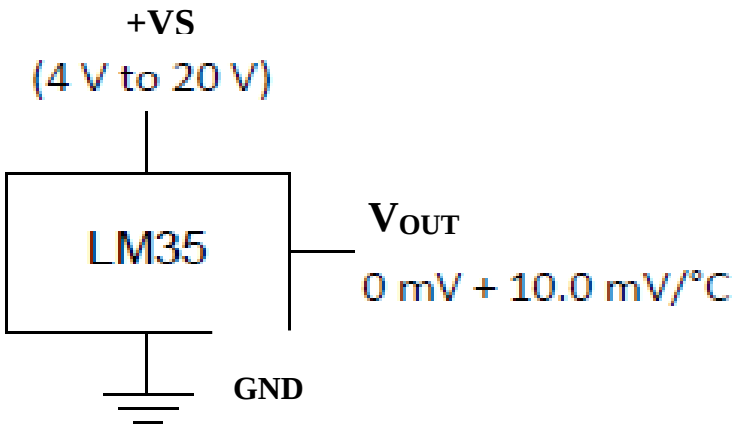


Figura 11: Configuração básica do sensor temperatura, extraído do manual (TEXAS ISTRUMENTS, 2013)

Tabela 1: Características do LM35, extraído do manual (TEXAS ISTRUMENTS, 2013)

Característica	**Valor**
Precisão	$\pm 0{,}6$ºC
Sensibilidade	$10mV/$ºC
Corrente quiescente	56µA
Coeficiente de temperatura da corrente quiescente	0,39µA/ºC
Faixa de operação	-55 ºC a 150ºC

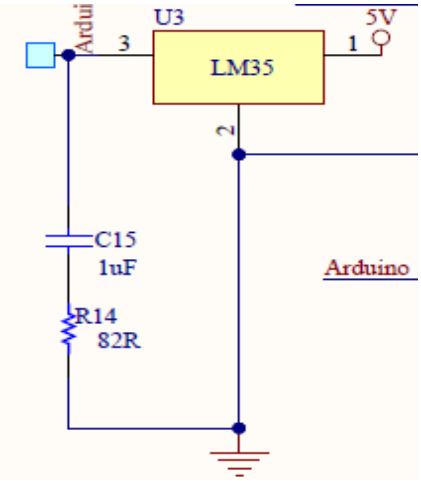


Figura 12: Parte do circuito que mostra o sensor de temperatura LM35, vide Apêndice A.

A leitura desse sensor é feita na entrada analógica A5 do Arduino. Há uma diferença em relação as outras leituras, pois não fizemos a leitura direta da entrada visto que há uma certa variação nos resultados (ROSÁRIO, 2013, p. 26). Demostrou-se ser mais precisa a leitura realizando uma repetição para acumular 100 amostras e depois extraímos a média aritmética. A equação recursiva (4) é responsável pelo acúmulo das amostras e no final desse *loop* a equação (5) faz a conversão para temperatura corrigindo com o fator de escala do sensor LM35 (0,01 V/ºC) e calcula a média.

$$T_1 = T_1 + E_{A5} \tag{4}$$

$$T_2 = \frac{V_{REF}}{2^{10}} \frac{1}{0{,}01} \frac{T_1}{N_1} \tag{5}$$

Onde;

T_1 é a temperatura medida acumulada,

T_2 é a temperatura média medida em graus Celsius,

E_{A5} é a leitura analógica feita pela porta A5 do arduino,

N_1 é a quantidade de amostras.

2.3. Sensor de tensão

O sensor de tensão utilizado no projeto é um simples divisor de tensão, onde a tensão de saída V_{OUT} é calculada com a seguinte equação:

$$V_{OUT} = \frac{R_2}{(R_1 + R_2)} V_{IN} \quad (6)$$

Onde;

V_{IN} é a tensão de entrada,

V_{OUT} é a tensão de saída,

R_1, R_2 são os resitores.

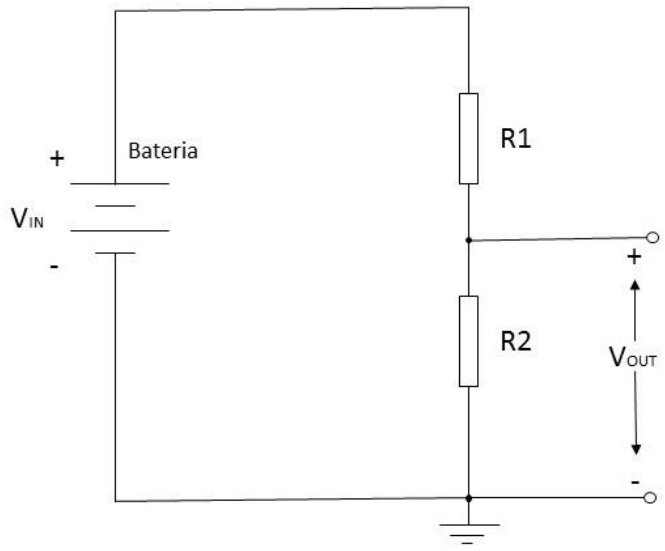


Figura 13: Divisor de tensão.

Como mostra a Figura 14 lemos a tensão em três pontos diferentes: na bateria onde com essa medida podemos fazer uma análise da potência instântanea da embarcação (pois medimos também a corrente em cada motor), essa tensão é medida através do divisor resistivo formado por R11 e R15 onde lê-se AD2, será conectado á entrada analógica 2 do microprocessador arduino onde o cálculo para efetuar as medidas será realizados via software que será explicado no Capítulo 4. As outras duas tensões medidas são nos dois "braços" da ponte H, onde os pontos de conexão são os pontos B e C da figura 14, com essas tensões temos condições de saber informações sobre o acionamento dos *MOSFETs*, qual o seu estado, o sentido de rotação dos motores, qual o valor do *duty cycle*, o valor médio da tensão e consequentemente podemos fazer o controle da velocidade da embarcação

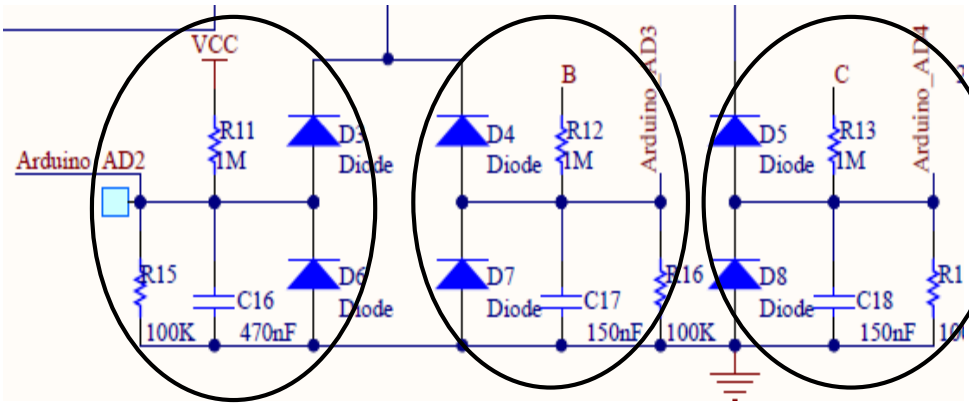


Figura 14: Destaque dos sensores de tensão (trecho extraído do circuito completo no Apêndice A).

O cálculo da leitura da tensão feita no microprocessador tem as seguintes equações:

1) Na bateria:

$$V_{OUT} = \frac{V_{REF}}{2^{10}} E_{A2} \tag{7}$$

Da equação (6) temos,

$$V_{OUT} = \frac{R_2}{(R_1 + R_2)} V_B \tag{8}$$

Onde;

$V_B = V_{IN}$ = tensão na bateria

Logo:

$$V_B = \frac{(R_1 + R_2)}{(R_2)} \frac{V_{REF}}{2^{10}} E_{A2} \tag{9}$$

R1= 1MΩ;

R2= 100kΩ.

2) Na ponte H

$$V_C = \frac{V_{REF}}{2^{10}}\ E_{A3} \tag{10}$$

$$V_A = \frac{V_{REF}}{2^{10}}\ E_{A4} \tag{11}$$

Onde;

V_A é a tensão medida no braço 1 da ponte H;

V_C é a tensão medida no braço 2 da ponte H;

E_{A3} é a leitura analógica feita pela porta A3 do arduino;

E_{A4} é a leitura analógica feita pela porta A4 do arduino.

Repare que em (6) a fração de R_2 / (R_2+R_1) gera um fator multiplicativo menor que 1. Isso é feito para que a tensão de entrada seja atenuada, o que permite leituras de tensões maiores que 5V (que é o limite do Arduino) sem danificá-lo.

As escolhas dos resitores foram feitas de modo a drenar o menor valor possível de corrente e que ofereça uma boa margem de leitura, escolhemos um valor de R1 = 1MΩ e de R2 = 100kΩ que atenua o valor a tensão de entrada em aproximadamente 11 vezes, ou seja conseguiremos medir até 55V o que dá ao *hardware* maleabilidade na leitura de tensões.

Na Tabela 2 pode ser observado o resumo dos tipos de sensores, portas do Arduino que são utilizados e em quais locais as variáveis são medidas de acordo com diagrama do circuito de acionamento apresentado no apêndice e no CD que acompanha este projeto de graduação.

Tabela 2: Resumo da aplicação dos sensores.

Tipo do sensor	**Portas do Arduino**	**LOCAL**
Corrente	A0	Motor
	A5	Bateria
Tensão	A2, A3	Ponte H (Motor)
	A1	Bateria
Temperatura	A4	*MOSFETs*

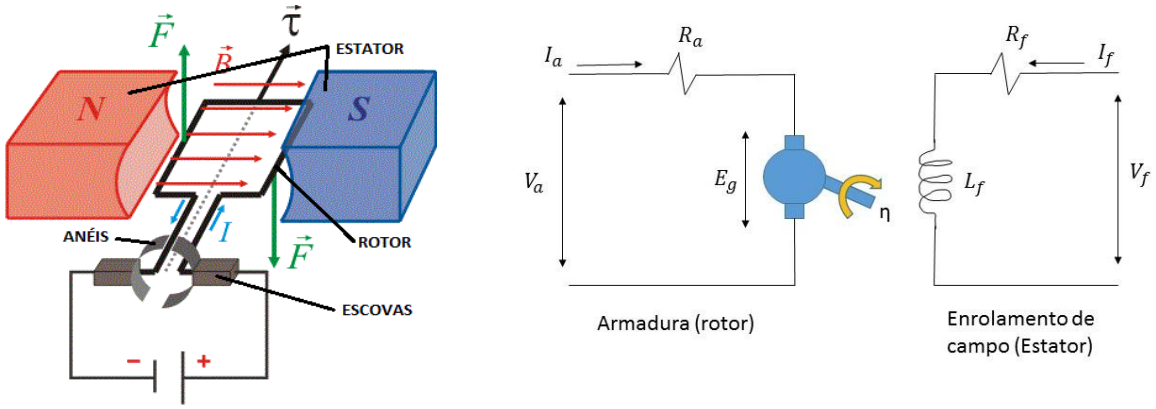
3. Acionamento do motor de corrente contínua

O motor de corrente contínua (motor CC) é uma máquina girante e sua função principal é transformar energia elétrica em energia mecânica (KOSOW, 1982, p. 38), para um entendimento inicial do funcionamento dos motores CC é preciso conhecer seus fundamentos. O funcionamento de um motor CC esta fundamentado no princípio de que quando um condutor percorrido por uma corrente de intensidade I é atravessado por um campo magnético $\vec{B}$ este condutor fica sujeito a uma força $\vec{F}$, o sentido da força é determindo pela direção da corrente ou do campo magnético e pode ser verificado de acordo com a regra da mão esquerda para motor (KOSOW, 1982, p. 28). Os motores CC são divididos basicamente nas seguintes partes:

- Armadura (rotor);
- Estator (campo)
- Comutador e escovas

Na Figura 15 (a) vemos o detalhe construtivo de um motor elementar muito simples, a armadura (rotor) é a parte girante deste motor constituída por uma única espira, repare que quando a armadura é percorrida pela corrente I atua uma força de módulo $\vec{F}$ em cada lado da espira, o sentido dessa força é determinado pela direção da corrente I conforme descrito anteriormente, as bobinas de campo (estator) foram substituídas por imã permanente que gera um campo magnético, isto é feito geralmente em pequenos motores, esse modo de criar o campo magnético tembém facilita o entendimento, porém este campo pode ser gerado por bobinas ligadas a uma fonte externa ou a mesma fonte de tensão da armadura, os anéis são segmentos condutores isolados entre si, neste caso temos dois anéis que são responsáveis por inverter o sentido da corrente em cada semiciclo, e as escovas são responsáveis por aplicar a tensão elétrica ás espiras da armadura estas .

Os motores de corrente contínua possuem diversas características de funcionamento como torque, corrente, velocidade e etc, que podem ser obtidas de formas variadas de acordo como excitamos as suas bobinas de campo, basicamente podemos fazer isto das seguintes formas:



(a) (b)

Figura 15: (a) motor elementar (extraído e adaptado de http://www.thephysicsforum.com/classical-physics/7162-faraday-law-magnetic-force.html), (b) Motor CC excitação independente

- Excitação independente

Figura 15 (b) neste arranjo são necessário duas fontes de tensão independentes V_a e V_f, onde V_a é a fonte que alimenta a armadura e V_f é a fonte que alimenta o enrolamento do campo. Os parâmetros mostrados na figura são:

I_a é a corrente de armadura;

R_a é a resistencia ôhmica de armadura;

E_g é uma tensão induzida denominada Força contra-eletromotriz, já que de acordo com a lei de F*araday* um conjunto de espiras girando num campo magnético induz uma tensão devido ao fluxo magnético.

R_f, I_f, e L_f é a resistêcia, corrente e indutância de campo respectivamente, este motor tem as seguintes equações fundamentais:

$$T = K \, Ø \, I_a \tag{12}$$

Onde;

T é o torque, K é uma constante que depende do projeto construtivo do motor, $Ø$ é o fluxo magnéticoe e I_a é a corrente de armadura.

$$E_g = k \, Ø \, n \tag{13}$$

Onde,

$$\emptyset = L_f \, I_f \tag{14}$$

E_g é a força contra-eletromotriz, *n* é a velociade do motor.

$$E_g = V_a - \, I_a \, R_a \tag{15}$$

Combinando (13), (14), (15) temos;

$$n = \frac{V_a - \, I_a \, R_a}{k \, L_f I_f} \tag{16}$$

Com a equação (16) vemos que o controle de velocidade de um motor CC pode ser feito de três formas distintas, controlando a tensão de armadura, a resistência de armadura o fluxo do campo atravès da corrente do campo, e como vemos se a corrente de campo for nula a velocidade do motor tende ao infinito, ou seja, o motor dispara e esse é um ponto importante a ser observado no controle de velocidade dos motores de corrente contínua. O torque pode controlado através da corrente de armadura equação (12).

Os outros tipos de motrores CC têm equações fundamentais similares ao de excitação independente e estes no que tange o controle de velocidade a análise é similar.

- Motor série (universal)

É um tipo de motor CC bastante utilizado onde se exige muito torque, como em tração de veículos metroviários, por exemplo, seu alto torque provém do fato da sua bobina de campo estar ligada em série com o circuito da armadura, como vemos na equação (12) o torque de um motor CC é proporcional ao produto do fluxo Ø por *k* e I_a, no entanto, nesta configuração o fluxo também é proporcional a I_a, ou seja, o torque fica como vemos na equação (17) proporcional ao quadrado da corrente de armadura.

$$T = k \, I_a^2 \tag{17}$$

Este motor tem a peculiaridade de ser utizado tanto com corrente contínua como corrente alternada o que lhe dá o título de ser universal.

- Motor excitação paralelo ou *shunt*

De acordo com (PATANE, 2010, p. 24) os motores com excitação paralela são utilizados em aplicações onde o controle de velocidade não é crítico pois este motor requer sistemas muitos simples para controlar sua velocidade ao custo de não se ter um controle extremamente preciso.

3.1. Motor escolhido para o projeto

(SHULTZE, 2012, p. 32) em seu projeto de gradução descreve com clareza o processo de escolha do tipo de motor utilizado na embarcação desse projeto, ressaltanto que a presença de escovas é um fator limitante da confiabilidade, rendimento e vida útil se comparado com motores sem escova devido aos arcos elétricos gerados no momento da comutação das bobinas da armadura. Porém, o alto preço e a indisponibilidade dos motores sem escovas no mercado nacional inviabilizariam o projeto, se esse fosse escolhido, com isso foi escolhido um motor CC com escovas tomando como prerrogativas além do baixo preço de aquisição, a disposição no mercado nacional para fututras reposições, a facilidade no controle de velocidade e a facilidade de manutenção. O motor escolhido é um *Marine Sports Phantom* 44lbs – Água Salgada (figura 16).

Figura 16: Motor Marine Sports Phantom 44lbs – Água Salgada.

As principais características desse motor são mostrada na Tabela 3.

Tabela 3: Dados básicos do motor, adaptado de (SCHULTZE, 2012, p. 46).

Marine Sport Phantom 44 lbs – água salgada		
Parâmetro	**Valor**	**Unidade**
Empuxo	20	kgf
Tensão	12	volt
Peso	15	kg
Hélice	3 pás plásticas	--

3.2. Circuito de acionamento

O circuito de acionamento é um bloco responsável por fazer a interface entre o microcontrolador e os propulsores da embarcação, três funções essênciais deste circuito devem ser destacadas:

- Sentido de deslocamento: a topologia escolhida deve ser capaz de acionar os motores tanto no sentido para frente (avante), quanto para trás (ré).
- Ele deve adequar os níveis de tensão e corrente para que seja entregue ao motor a quantidade de potência necessária para uma dada velocidade requisitada pelo operador.
- O circuito deve ser robusto e confiável e contar com proteções caso ocorra uma falha inesperada no sistema.

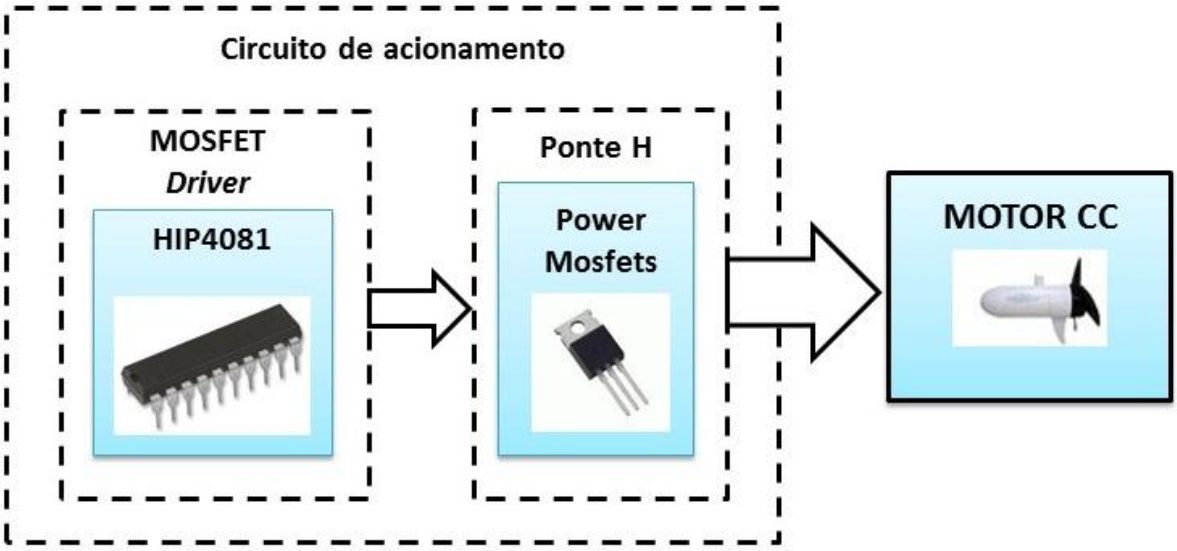


Figura 17: Diagrama em bloco do circuito de acionamento.

3.2.1. Conversores CC-CC

Os conversores CC-CC são circuitos que tranformam uma tensão CC fixa de entrada em uma tensão CC variável em sua saída (RASHID, 1999, p. 371). São amplamente utilizados no controle de tração dos mais diveros tipos de veículos elétricos, guindastes, empilhadeiras etc, por oferecerem controle de aceleração suave, alta eficiência e resposta dinâmica rápida.

Quando os conversores CC-CC são utilizados para acionar máquinas de corrente contínua, especialmente motores CC, geralmente são denominados de *choppers*. Os *choppers* podem operar com a função de abaixar ou elevar a tensão de uma fonte CC.

3.2.2. Classificação dos Conversores CC-CC

Segundo (RASHID, 1999, p. 382) os *choppers* podem ser classificados em classe A, B, C, D e E, de acordo com o seu quadrante de operação, estes quadrantes podem ser definidos dentro do plano de operação do torque em função da velocidade, no entanto, se considerarmos que a máquina de corrente contínua possui um fluxo constante no entreferro podemos considerar esse plano como sendo o valor médio da corrente de armadura I_a em função da tensão de armadura E_a (POMILIO, 2014, p. 4 - 4) como visto na Figura 18.

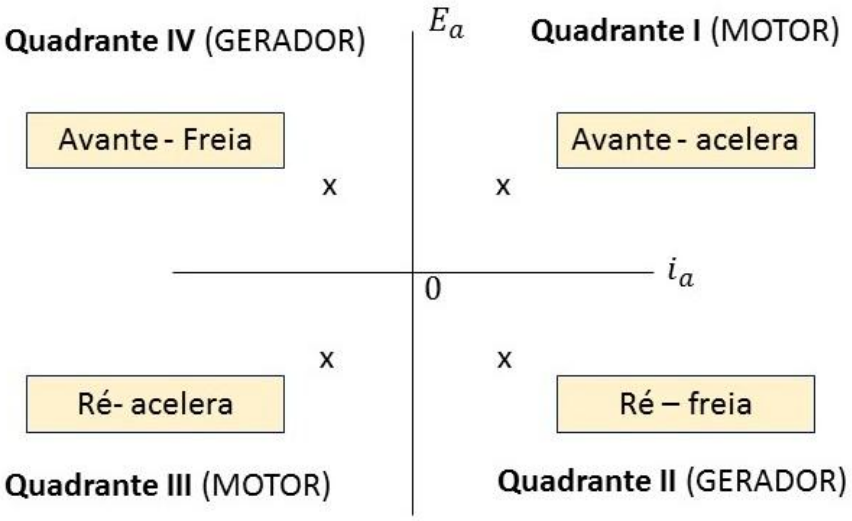


Figura 18: Quadrantes de operação de uma máquina CC.

- Conversores classe A

Operam no quadrante I, pois são capazes de acionar o motor em apenas um sentido de rotação, ou seja, o torque só pode ser desenvolvido em um sentido

- Conversores classe B

Operam no quadrante IV, assim como o classe A são capazes de acionar o motor em apenas um sentido de rotação, porém conseguem inverter o sentido do torque desenvolvido, ou seja, invertendo o sentido da corrente de armadura fazendo com que o motor seja frenado.

- Conversores classe C

Operam nos quadrante I e IV, com este tipo de conversor podemos fazer tanto frenagem quanto acelaração em apenas um sentido.

- Conversores classe D

Operam nos quadrante I e II, com este tipo de conversor não podemos fazer frenagem, não há como iverter o sentido da corrente.

- Conversores classe E

Operam nos quadrante I, II, III e IV, com este tipo de coversor pode-se acelarar e frear tanto para á frente (avante) quanto para traz (ré).

3.2.3. Escolha do conversor CC-CC

Fazendo uma análise dos tipos de conversores passíveis de escolha para o projeto, o único candidato possível é o da classe E, pois é o único que satisfaz a primeira condição para o circuito de acionamento que estipulamos no início desse Capítulo, além de que com essa topologia tem-se a possibilidade futura de implementar um sistema de frenagem, como por exemplo, frenagem regenerativa onde a energia do sistema retorna para a fonte na hora da parada ou desacelaração desde que a fonte seja receptiva, visto que, para um sistema autonômo encontrar formas aternativas de armazenar e otimizar o uso da energia disponível é crucial para o sucesso desse sistema.

3.2.4. Ponte H

O conversor classe E é tambem conhecido como ponte H, tem esse nome porque as chaves junto com a carga formam um "H" como pode ser observado na Figura 19 abaixo. O sentido de rotação do motor CC é feito através do acionamento coordenado das chaves, o controle é feito de tal forma que as chaves S1 e S4 fechem para que o motor gire em um sentido, e S3 e S2 fechem para que o motor gire no outro, a tensão média entregue nos terminas do motor e por consequência a velocidade é controlada pelo tempo em que as chaves permacem ligadas e desligadas.

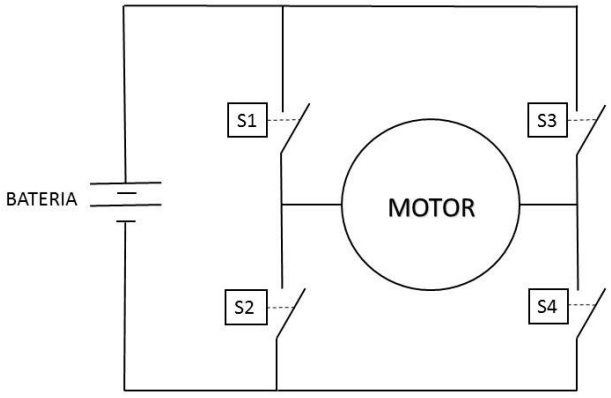


Figura 19: Ponte H.

3.2.5. Modulação por largura de pulso

A modulação por largura de pulso ou (*Pulse Width Modulation* - PWM) é um método de controle que atua em um dispositivo que funciona como chave ligando e desligando este dispositivo em uma frequência fixa. Se o tempo em que a chave permanece ligada é T_{ON} e o

período total considerado é T, a relação entre T_{ON} / T é denominada *duty cycle* ou ciclo de trabalho (D). A tensão média entregue nos terminais do motor pode ser expressa como.

$$E = V\,\frac{T_{on}}{T} \tag{18}$$

V é a tensão da fonte.

$$D = \frac{T_{on}}{T}100\% \tag{19}$$

Definindo-se *duty cycle* reescreve-se (18) como;

$$E = V\,D \tag{20}$$

Na Figura 20 vemos sinais PWM com frequência fixa e alguns valores percentuais do *duty cycle*, como podemos notar quanto maior maior o tempo no estado ligado maior a tensão entregue nos terminais do motor até atingir D = 100%, onde a tensão entregue ao motor é igual a tensão de alimentação.

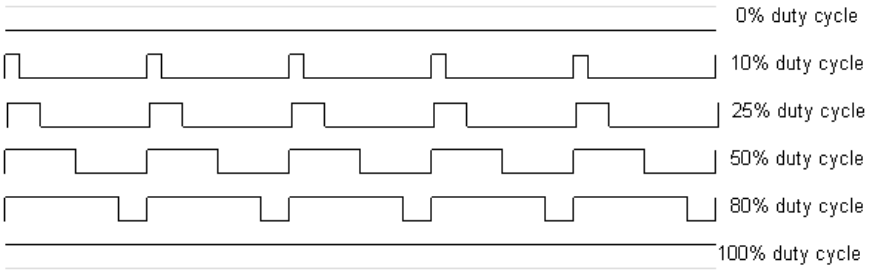


Figura 20: Pulsos PWM

3.2.6. MOSFETs de potência

Nos tópicos anteriores as chaves descritas foram consideradas ideais, pois nenhum consumo de energia para elas foi considerado, no entanto, na prática não é isto que acontece qualquer dispositivo que escolhessemos dentre as tecnologias atuais disponíveis para efetuarem o papel de chave teriam as seguintes perdas:

- Perda na condução – perda relativa ao comportamento da chave quando está ligada e algum efeito que faça surgir uma queda de tensão entres seus terminais.

- Perda na comutação – perdas relativas as etapas de ligar e desligar a chave, são intrínsecas as características da chave e da frequência de operação.
- Perda no cionamento – perda relativa aos circuitos de acionamento do dispositivo.

Com isso em mente, a escolha de um dispositivo adequado para o projeto é uma questão delicada, as perdas para acionar um dispositivo podem ser tão grandes a ponto de tornar um projeto energeticamente inviável ou pouquissímo eficiente, principalmente as perdas por concdução e comutação que tiram diretamente parte da energia disponível á carga.

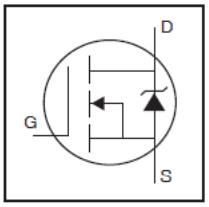


Figura 21: Símbolo gráfico para um *MOSFET*, (INTERNATIONAL RECTIFIER, 2002, p .1).

Um transístor de efeito de campo (*field effect transistor* - FET) é um disposito semicondutor de três terminais, porta (*gate* - *G*) , dreno (*drain* - *D*), e fonte (*source* - *S*) mostrado na Figura 21 ele se divide em dois grupos *JFETs* e *MOSFETs*, e os *MOSFETs* se dividem em dois subgrupos, depleção e intensificação.

(RASHID, 1999, p. 344) destaca que os *MOSFETs* de potência têm encontrado aplicações crescentes em conversores de alta frequência e baixa potência, e normalmente nesses tipos de aplicações o transístor de efeito de campo metal óxido semicondutor ou simplesmente *MOSFET* são aplicados. O *MOSFET* é um dispositivo controlado por tensão ao contrário dos tradicionais transístores de junção bipolar (*Bipolar Junction Transistor* - *BJT*) que são controlados por corrente, isso lhe confere muita facilidade em seu emprego, porém, são dispositivos mais caros, mais complexos e de manuseio delicado em consequência de descargas eletrostáticas. Para seu acionamento simplesmente é aplicado uma tensão no terminal de entrada (*gate)* maior que uma tensão de limiar V_{Th} desse modo, é permitido a passagem de corrente I_D entre seus terminais dreno e fonte. Quando um *MOSFET* esta conduzindo ele se comporta como um resistor variável de valor R_{on} e se atravessado por uma corrente de intensidade I a potência que ele terá que dissipar em sua condução será:

$$P = I^2 R_{on} \tag{21}$$

Valores de *Ron* para *MOSFETs* de ótima qualidade variam de 2 a 5 mΩ, no caso desse projeto de graduação, por exemplo, teremos uma corrente máxima em regime permanente de aproximadamente 30A e R_{on} por volta de 3,7mΩ, com esses valores têm-se uma potência dissipada em cada *MOSFET* devido a condução de P = 3,34W que é uma potência pequena em relação a potência do motor e que pode ser facilmente dissipada com trocadores de calor de pequeno porte.

Por ser um dispositivo controlado por tensão isto lhe confere uma alta impedância de entrada que reflete em uma pequena corrente de gate quando está em condução, na verdade quase zero, porém na comutação existe uma corrente entrando e saindo do gate devido ao fato de haver uma capacitância parasita entre os terminais *gate* e *source*, a energia armazenada nessa capacitância é dissipada no circuito de acionamento pois geralmente não há elementos dissipativos no inteiror dos *MOSFETs*. No entanto, a capacitância entre *gate* e *source* deve ser carregada com valor de tensão acima do limiar V_{Th} (essa tensão é a própria tensão do gate) para que haja condução, e descarregada a um valor abaixo de V_{Th} para que haja corte no, o tempo de carga T_{on} e o tempo descarga T_{off} depende diretamente da capacitância do *MOSFET* e do circuito de acionamento, geralmente esses tempos estão na ordem de dezenas de nano segundos, que confere ao *MOSFET* velocidade de chaveamento alta oferecendo menos perdas na comutação em relação aos transistores de junção bipolar.

3.2.7. Escolha do MOSFET de potência

Todas essas vantagens favorecem a escolha do uso do *MOSFET* neste projeto de graduação, mesmo sendo um transístor mais caro. O *MOSFET* escolhido para a aplicação é o IRF1404Z da *international rectifier*® (Figura 22) as características do manual desse *MOSFET* podem ser conferidas na tabela abaixo.

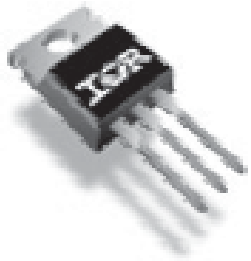


Figura 22: *MOSFET* IRF1404Z.

Tabela 4: Características do manual IRF1401Z (INTERNATIONAL RECTIFIER, 2012, p. 2).

Electrical Characteristics @ T_J = 25°C (unless otherwise specified)

	Parameter	Min.	Typ.	Max.	Units	Conditions
$V_{(BR)DSS}$	Drain-to-Source Breakdown Voltage	40	—	—	V	V_{GS} = 0V, I_D = 250µA
$\Delta V_{(BR)DSS}/\Delta T_J$	Breakdown Voltage Temp. Coefficient	—	0.033	—	V/°C	Reference to 25°C, I_D = 1mA
$R_{DS(on)}$	Static Drain-to-Source On-Resistance	—	2.7	3.7	mΩ	V_{GS} = 10V, I_D = 75A ③**
$V_{GS(th)}$	Gate Threshold Voltage	2.0	—	4.0	V	V_{DS} = V_{GS}, I_D = 150µA
gfs	Forward Transconductance	170	—	—	V	V_{DS} = 25V, I_D = 75A**
I_{DSS}	Drain-to-Source Leakage Current	—	—	20	µA	V_{DS} = 40V, V_{GS} = 0V
		—	—	250		V_{DS} = 40V, V_{GS} = 0V, T_J = 125°C
I_{GSS}	Gate-to-Source Forward Leakage	—	—	200	nA	V_{GS} = 20V
	Gate-to-Source Reverse Leakage	—	—	-200		V_{GS} = -20V
Q_g	Total Gate Charge	—	100	150		I_D = 75A**
Q_{gs}	Gate-to-Source Charge	—	31	—	nC	V_{DS} = 32V
Q_{gd}	Gate-to-Drain ("Miller") Charge	—	42	—		V_{GS} = 10V ③
$t_{d(on)}$	Turn-On Delay Time	—	18	—		V_{DD} = 20V
t_r	Rise Time	—	110	—		I_D = 75A**
$t_{d(off)}$	Turn-Off Delay Time	—	36	—	ns	R_G = 3.0 Ω
t_f	Fall Time	—	58	—		V_{GS} = 10V ③

3.2.8. Circuito de acionamento dos MOSFETs

Para o acionamento dos mosfets da ponte H foi utilizado o circuito integrado HIP4081 que é um *driver* para *MOSFETs* canal n de média voltagem e alta frequência, é capaz de operar de 8V a 80V e pode fornecer picos de corrente de gate de até 2,5A (INTERSIL, 2003, p. 1), controla independentemente cada *MOSFET* da ponte H e pode operar até a frequencia de 1MHz, tem encapsulamento *DIP* ou *SOIC* com 20 pinos, conta com várias soluçoes prontas para o acionamento de ponte H, um diagrama em blocos com apenas a metade A da ponte H é mostrado abaixo na Figura 24, a metade B é idêntica a A, e foi suprimida visando facilitar o entendimento das explicações que se seguem.

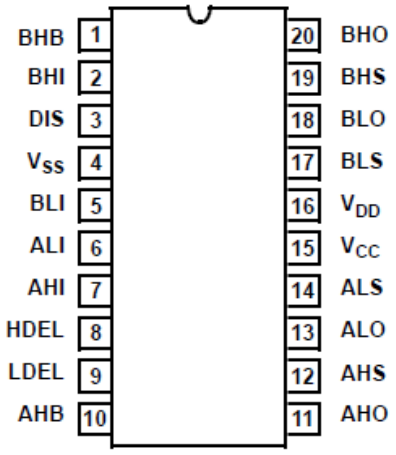


Figura 23: Pinos do HIP4081, extraído de (INTERSIL, 2002, p. 1).

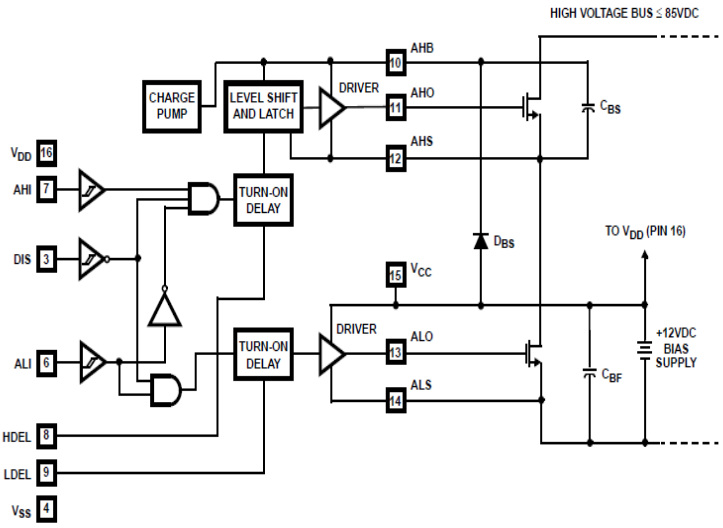


Figura 24: Diagrama em bloco simplificado do HIP 4081, extraído de (INTERSIL, 2002, p. 2).

- Alimentação e terra

O pino 15 (V_{CC}) é o positivo da fonte de alimentação do circuito integrado, em seu *datasheet* (INTERSIL, 2015, p. 6) é recomendado que o coloquemos no mesmo potêncial que o pino 16 (V_{DD}) que seria o positivo da alimentação do drive dos *mosfets* inferiores, estes pino podem ter um faixa máxima de tensões entre -0,3V a 16V e tensão de operação recomendada entre 6V e 16V em relação ao pino 4 (V_{SS}) que é a refêrencia de terra.

- Entradas e saídas Lógicas

O HIP 4081 tem 4 entradas lógicas ALI, BLI, AHI, e BHI que controlam as saídas para os Gates dos *MOSFETS*, as saídas são nos pinos ALO, BLO, AHO, BHO respectivamente, essas entradas aceitam nivel logico de 0 a 15V.

O pino *DIS* (*Disable*) é uma proteção que coloca em nível baixo todas as saídas quando estiver habilitado, em nivel lógivo “1”, independente do estado das entradas, abaixo na tabela 3.1 vemos a lógica de funcionamento do disable em relação as entradas dos sinais de controle.

Tabela 5: Entradas lógicas e tabela verdade

ALI, BLI	**AHI, BHI**	**DIS**	**ALO, BLO**	**AHO, BHO**
X	X	1	0	*0*
1	X	0	1	*0*
0	1	0	0	*1*
0	*0*	*0*	*0*	*0*

Legenda: X = não importa (*Don´t care)*, 1 = Nível alto, 0 = Nível baixo.

Observe que quando *ALI* e *BLI* estão com nível lógico "1" , ou seja, estão ligados não importa qual o estado lógico de *AHI* e *BHI* os controles de saída dos *MOSFETs* superiores são forçados a desligar, por exemplo, de acodordo com a tabela 5 vamos supor que *ALI* = 1 então, *ALO* = 1 e *AHO* = 0, isso advém de uma proteção que o circuito integrado tem contra *shoot-through* (curto circuito), isso também sugere uma simplicação no circuito de controle da ponte H, já que controlando apenas as duas entradas lógicas inferiores estamos controlando as saídas superiores e inferiores do mesmo braço da ponte H, bastanto apenas colocar um resistor de *pull up* nas entradas *AHI* e *BHI* que fixa o nível lógico em 1 dessas entradas, essa sugestão é indicada pelo fabricante em seu *aplication note* (INTERSIL, 2003, p. 2) e foi utilizada neste projeto, do contrário teríamos que controlar os 4 *MOSFETs* aumentando de forma não muito significativa a complexidade do software, porém gastando mais portas PWM do microprocesador e aumentando a probabilidade de falhas.

Dois pino muito importantes são os pinos HDEL e LDEL que são responsáveis pelo ajuste do *dead-time* ou tempo morto que é o atraso na mudança de estado dos *MOSFETs* afim de evitar curto-cicuito na bateria através dos *MOSFETs* que estão no mesmo braço da ponte H, se estes conduzirem ao mesmo tempo. Portanto, tem-se que esperar que um *MOSFET* corte a condução para que o outro do mesmo ramo comece a conduzir. Esse tempo é ajustando baseando-se no gráfico de resistência versus tempo dado na figura 25 extraído do *application notes* do fabricante do circuito integrado.

O *MOSFET* escolhido para o projeto foi visto no item anterior, este *MOSFET* tem um tempo de comutação de *on* para *off* (Td(off)- turn-off delay time) da ordem de 36ns (INTERNATIONAL, 2012, p. 2) e um tempo de comutação de *off* para *on* (Td(on) turn–on

delay time) de 18ns, “entra-se” no gráfico com o *dead-time* requerido e encontra-se o valor do resitor adequado em HDEL e LDEL para tal, observando o gráfico na Figura 25 vemos que o resistor adequado estaria um pouco a cima de 50kΩ, por não ser um valor comercial foi colocado um resistor de 56kΩ e testando o circuito com esse valor de resistência foi notado um aquecimento excessivo nos *MOSFETs*, sobretudo, os superiores. Após testados vários valores de resistores foi observado que o resistor mais adequado seria um de 270KΩ, valores abaixo destes provocam aquecimento.

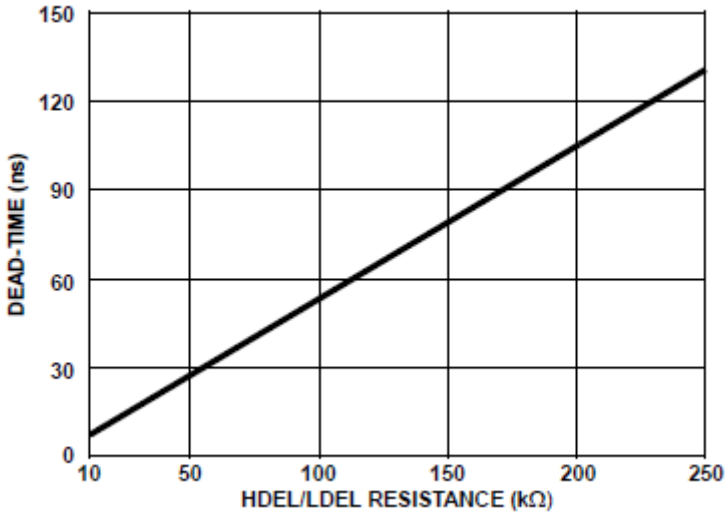


Figura 25: Mínimo Dead Time x DEL resistance, extraído de (Intersil, application note, 2003, p. 2).

- *Charge pump*

Segundo (KORMANN, 2003, p. 1) *charge pumps* são notadamente conhecidos e empregados em circuitos dobradores de tensão e inversores de baixa potência como por exemplo em circuitos com correntes de saída abaixo de 50mA, o funcionamento de um circuito *charge Pump* baseia-se no chaveamento de capacitores, que num primeiro momento carrega-se em paralelo a uma tensão de entrada e num segundo momentos descarrega em série com essa mesma tensão, na Figura 26 podemos observar uma topologia básica de um duplicador de tensão que em uma primeira fase Ø1 (carga) as chaves S2 e S3 estão fechadas, o capacitor C_F carrega-se idealmente com uma tensão de valor V_{IN} enquanto o capacitor C_{OUT} fornece a alimentação da carga e descarrega-se, em uma segunda fase Ø2 (transferência) S1 e S4 são fechadas e o capacitor C_F fica em série com V_{IN}, logo, C_{OUT} carrega-se com $2V_{IN}$ caracterizando um dobrador de tensão.

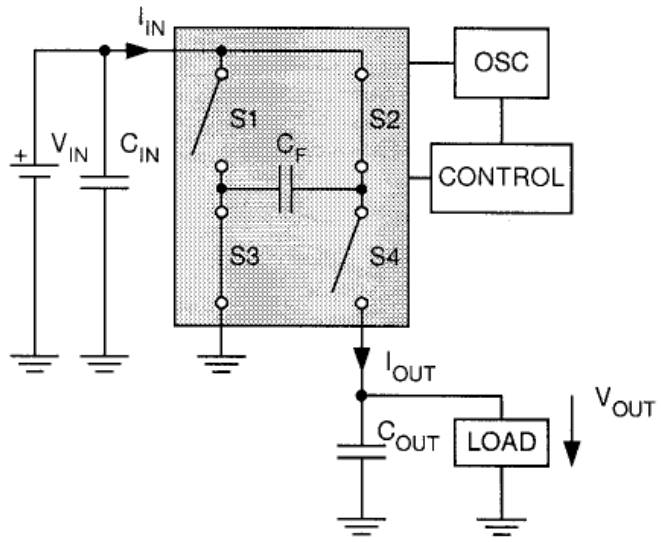


Figura 26: Topologia básica de *charge Pump* dobrador de tensão extraído de (KORMANN, 2003, p. 1).

O Hip 4081 tem dois circuitos *charge pumps* um para cada *MOSFET* superior. Cada *charge Pump* comuta o capacitor dobrador CBS ligados aos pinos AHB, AHO, BHB, BHO. Sem esse circuito quando os *MOSFETs* inferiores entrassem no corte, e os *MOSFETs* superiores fossem acionados não conseguiriam a tensão de VGS necessária para o correto acionamento, pois, perdem a referência de terra. Para contornar solucionar esse problema o capacitor CBS (*bootstrapping*) carrega-se com o dobro do valor de V_{CC} através do diodo DBS enquanto o *MOSFET* inferior estava em estado ligado, assim CBS fornece a tensão e a carga necessária para o correto acionamento dos *MOSFETs* superiores quando solicitado.

Um ponto de extremamente importante a se destacar é que o *duty cycle* do PWM que aciona os *MOSFET*s inferiores não poderá em nenhuma hipótese atingir 100%, visto que, nessa condição estes estão o tempo todo em estado ligado, não permitindo que os capacitores CBS se carreguem, tornando o acionamento dos *MOSFETs* superiores passíveis de falhas.

4. Arduino

O Arduino é uma plataforma microcontrolada baseada no *chip ATEMEGA* da empresa *ATMEL*, a parte de *hardware* da plataforma Arduino é muito mais conhecida, porém o termo Arduino refere-se à composição de *hardware* e *software*. O Arduino foi desenvolvido na Itália em 2005 inicialmente com o intuíto de facilitar o desenvolvimento de projetos eletrônicos para pessoas leigas (MARGOLIS, 2012, p. 1) ou com pouco conhecimento de *softwares* e *hardwares* eletrônicos como *designers*, músicos, decoradores e DJ's (do inglês, *Disc Jockey*).

Por fazer parte do conceito de plataforma *open-source* (Em português: fonte livre) onde todas as informações a respeito de esquemas de montagens e códigos de aplicação são compartilhadas em fóruns na *internet*, tutoriais, *cookbooks*, o Arduino esta se tornando cada vez mais popular entre os desenvolvedores não só amadores e vem ganhando espaço em aplicações ligadas á artes visuais, automação doméstica, robótica e outros, além de que, o preço a aquisição e a expansão das funcionalidades da plataforma são bastante acessíveis.

Há uma grande disponibilidade de tipos placas Arduino no mercado para as mais variadas aplicações, algumas plataformas Arduino estão apresentadas na figura 27, porém este trabalho tem como foco um tipo de Arduino, o Arduino UNO que fornece uma ótima base de entendimento de qualquer outra plataforma Arduino.

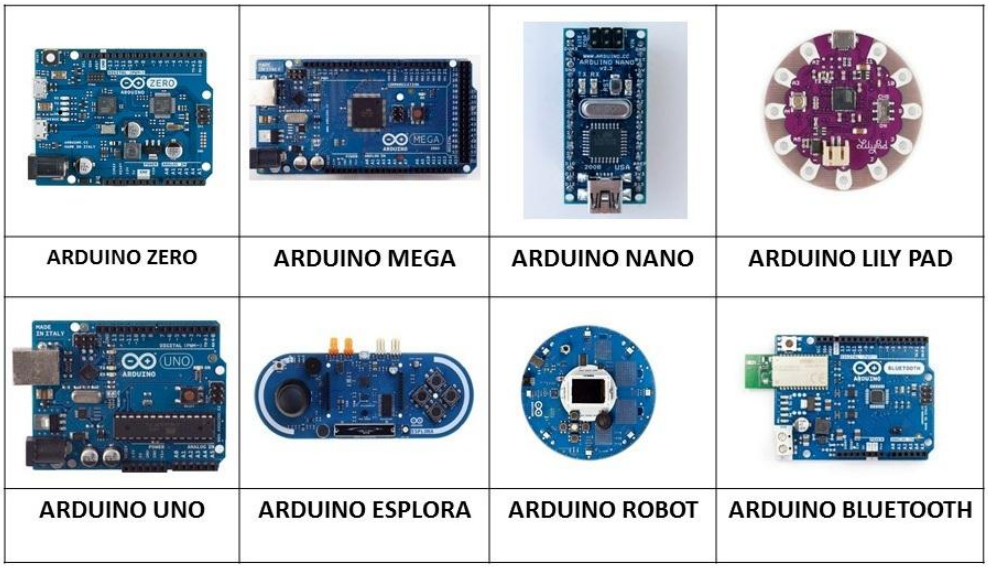


Figura 27**:** Alguns tipos de Arduinos.

4.1. Características do Arduino UNO

Dentre todos os tipos de Arduinos um dos mais utilizados é o UNO, principalmente para os iniciantes em eletrônica embarcada, essa plataforma apresenta um bom número de portas analógicas e digitais, mas um dos seus principais destaques refere-se a grande compatibilidade com os *Shields* (que será explicado a seguir) existentes no mercado para Arduinos, e o baixo preço em relação a alguns outros Arduinos para aplicações comparáveis como por exemplo o Arduino MEGA, na Tabela 6 é apresentado as principais características elétricas do Arduino UNO.

Tabela 6: Características elétricas Arduino UNO.

CARACTERÍSTICA	VALORES	UNIDADE
Tensão de operação	5	volts
Tensão de entrada (faixa recomendada)	7 a 12	volts
Tensão de entrada (limite)	6 a 20	volts
Corrente contínua por pino (entrada e saída)	40	mA
Corrente de saída no pino de 3,3V	50	mA
Frequência do *clock*	16	MHz

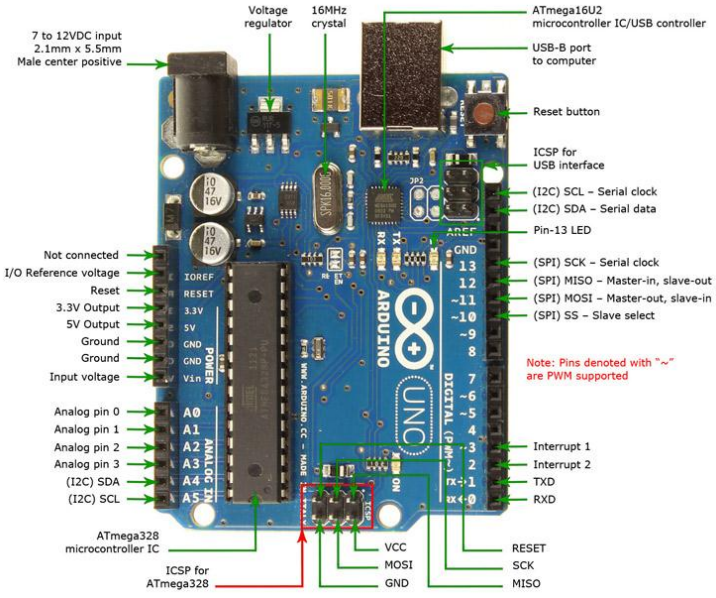


Figura 28: Arduino UNO (extraído de http://www.jameco.com/Jameco/workshop/circuitnotes/CN-arduino-uno.html).

Na Figura 28 vemos em destaque o Arduino UNO que conta com o microcontrolador ATMEGA 328 que é o principal componente da plataforma, este microcontrolador vem encaixado em um soquete e pode ser removido facilmente após algum defeito ou após ser efetuada sua gravação, podendo utilizá-lo em alguma aplicação *stand alone* (autônoma), onde com poucos componentes eletrônicos o microcontrolador funciona independente da placa Arduino. A alimentação do Arduino pode ser feita via *USB*, ou por uma fonte CC externa inserida em um conector de 2.1 mm com positivo no centro desde que respeitados os limites de voltagem em sua entrada, ainda podendo opcionalmente alimentá-lo pelo pino V_{IN}, sendo que o Arduino desliga automaticamente os pinos referentes a alimentação da interface *USB* (*universal serial bus*) de forma a proteger um computador ou outro dispositivo que por ventura esteja conectado ao Arduino. Algo muito útil é o fornecimento de uma saída de 3,3V que pode ser aproveitado para alimentação de algum sensor ou outro componente que exija esse nível de tensão de alimentação com limite máximo de 50mA.

O UNO conta com 6 entradas analógicas com um conversor A/D com resolução de 10 bits, 14 pinos digitais que podem ser configurados como entrada ou saída, 6 destes pinos podem ser usados como PWM *(pulse width modulation)*, alguns pinos podem assumir funções especializada como visto a seguir:

- Serial: pino 0 (RX) e pino 1 (TX). Usados para receber (RX) e transmitir (TX) dados seriais TTL.
- Interruptores Externos: pino 2 e pino 3. Estes pinos podem ser configurados para disparar uma interrupção de acordo com alguma variação sensível pelo circuito.
- *SPI*: pino 10 (SS), pino 11 (MOSI), pino 12 (MISO), pino 13 (SCK). Estes pinos dão suporte à comunicação SPI.
- *LED*: 13. Há um *led* integrado ao pino digital 13. Quando este pino está no valor alto este *led* está aceso, quando o pino está em valor baixo o *led* está apagado.
- *I^2C:* pino A4 (SDA) e pino A5 (SCL). Fornecem suporte a comunicação I^2C.
- A_{REF}: Voltagem de referência para as entradas analógicas.
- *Reset*: Para fazer um reset no microcontrolador um valor lógico baixo deve ser aplicado nesse pino. Tipicamente usado para adicionar um botão de reset nos *Shields* montados sobre a placa original.

4.2. Instalação do Arduino

Para fazer a instalação do Arduino no ambiente *Windows* deve-se baixar os *drivers* que vem junto com a interface de desenvolvimento do site oficial (ARDUINO, 2009) e após isso seguir os seguintes passos (BUORO, 2013):

1. Conecte o Arduino e espere o *Windows* começar o processo de instalação.
2. Após alguns momentos o processo irá falhar.
3. Clique em menu iniciar e abra o painel de controle
4. No painel de controle, acesso item Sistema e logo após clique em Gerenciador de dispositivos.
5. Procure dentre os itens Portas de Comunicação (COM & LPT) um item chamado Arduino UNO (COMxx)
6. Clique com o botão direito nesse item e escolha "Atualizar *Driver*"
7. Escolha a opção "Navegar no meu computador em busca dos *drivers*"
8. Aponte para a subpasta *Drivers* e escolha o arquivo **"ArduinoUNO.inf"**,
9. O Windows terminará a instalação dos drivers

4.3. Programação do Arduino

Para realizar a programação do Arduino é preciso baixar a *IDE* (do inglês, *Integrated Development Environment*) que é o ambiente de desenvolvimento da plataforma, o arquivo pode ser baixado do site oficial do Arduino https://www.arduino.cc/en/Main/Software não necessita fazer instalação apenas abrir o arquivo executável "arduino.exe", na figura 29 (a) vemos a tela inicial da *IDE* Arduino e seus botões, é nesta *IDE* que escrevemos os programas que são chamados de *sketches*, compilamos utilizando o compilador que já vem na própria *IDE* e se estiver tudo certo carregamos (*upload*) o programa para dentro do microcontrolador do Arduino através da porta *USB* para que se inicie a execução do programa.

Há 5 guias principais na *IDE* Arduino:

- *File* (Arquivo)

Nesta aba encontramos opções como abrir um arquivo existente, iniciar um novo projeto, abrir programas que foram abertos recentemente, salvar projeto atual, além de abrir muitos exemplos de projetos que vêm junto com a *IDE* quando fazemos seu download.

- *Edit* (Editar)

Nesta aba encontramos opções cortar, copiar, copiar para fórum, localizar, comentar que é muito útil para comentar partes do programa e assim encontrar problemas ou simplesmente comentar para compor informação ao arquivo, e outras opções.

- *Sketch*

Nesta aba encontramos as opções de verificar/compilar o *sketch* atual, carregar, entre outras. A opção em destaque dentro dessa guia é a de inserir novas bibliotecas na *IDE* arduino além das que já vem pré-instaladas.

Bibliotecas são soluções prontas desenvolvidas para facilitar certas aplicações, como exemplo para ligarmos um *display lcd*, um módulo *bluetooth*, um sensor, ou um módulo *XBee* para comunicação sem fio Z*igBee*. Para incluirmos uma biblioteca é necessário referencia-la no topo do programa antes da função *setup*, como exemplo, o trecho de programa a seguir inclui a biblioteca *LiquidCristal* e define quais pinos do Arduino serão usados para acionar o *display* e escrever textos em sua tela.

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(13, 11, 10, 4, 3, 2);
```

A *IDE* Arduino é baseada em *JAVA* que é uma linguagem de programação orientada a objetos e está por trás das facilidades que as bibliotecas proporcionam, qualquer pessoa que programe na linguagem *JAVA* pode fazer sua própria biblioteca para facilitar uma implementação específica.

- *Tools* (Ferramentas)

Nesta aba encontramos as opções como: autoformatação, arquivar sketch, corrigir codificação e recarregar, escolher o tipo de placa Arduino, escolha da porta utilizada pelo Arduino, gravar bootloader e outros. Uma ferramenta que se destaca nessa guia é o monitor serial, é nesse monitor que o Arduino pode se comunicar com o computador podendo enviar e receber dados e comandos de forma serial.

- *Help* (Ajuda)

Nesta aba encontramos as opções de ajuda, resolução de problemas, referências, entre outros.

(a) (b)

Figura 29: (a) IDE Arduino (b) Arduino conectado ao computador.

Na estrutura padrão de um programa Arduino há duas funções principais que têm que existir em qualquer programa:

- *Setup*

Nessa função fazemos as declarações de pinos, se são pinos de entradas ou pinos de saídas, ajustamos a velocidade de comunicação serial e qualquer outro ajuste que nossa implementação necessite, a função *SETUP* é executada apenas uma vez quando ligamos o Arduino ou o reiniciamos.

- *Loop*

Nessa função é onde escrevemos efetivamente nosso programa, ela é executada continuamente em um *loop* eterno.

As funções *setup* e *loop* são do tipo *void*, ou seja, não retornam nenhum tipo de valor e nem necessitam de argumentos.

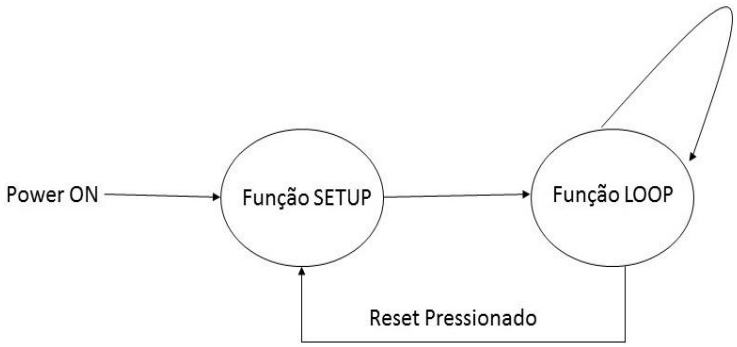


Figura 30: Principais funções Arduino.

4.4. *Shields* para Arduino

Os *Shields* são placas que agregam novas funcionalidades aos Arduinos, são geralmente encaixados em cima das placas Arduinos através de pino conectores, essas novas funcionalidades podem ser para acionamentos de pequenos motores, *displays LCD*, placa *touch screen*, módulos para conexão com a internet, módulos *GPS*, módulos *XBee* para comunicação sem fio e etc.

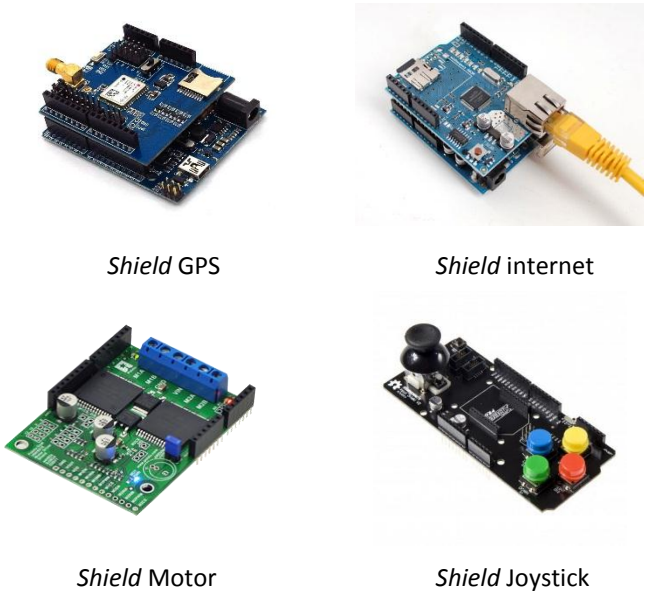


Figura 31: Exemplos de *shields* para Arduino.

O *shield* com maior relevância para este projeto de graduação é o *wireless sd card* que são utilizados para conexão dos módulos *XBee* nos Arduinos e possibilitam criação da rede de comunicação sem fio *ZigBee* que é descrito no Capítulo 5, este *shield* permite o encaixe direto do módulo *XBee* (figura 32), e é compatível com qualquer módulo *XBee* de qualquer série, conta com regulador de tensão que converte a tensão de alimentação padrão do Arduino que é de 5V para 3.3V para alimentar os módulos *XBee*, e faz a conversão desses níveis lógicos para a comunicação serial, possui um botão de *reset*, *leds* para a indicação de que esta ligado (PWR), para indicar a potência do sinal recebido RSSI (do inglês, *Received signal strength indicator*), para indicar transmissão e recepção de dados Tx e Rx, possui também uma chave de seleção de 2 posições que faz a multiplexação da UART (do inglês, *Universal Asynchronous Receiver Transmitter*), para transmissão e recepção de dados sem fio a chave deve estar na posição

MICRO ou na posição USB para comunicação com o Arduino através do cabo USB com o computador, com a chave nessa posição fazemos o *upload* dos programas escritos na IDE do Arduino (ver seção 4.3).

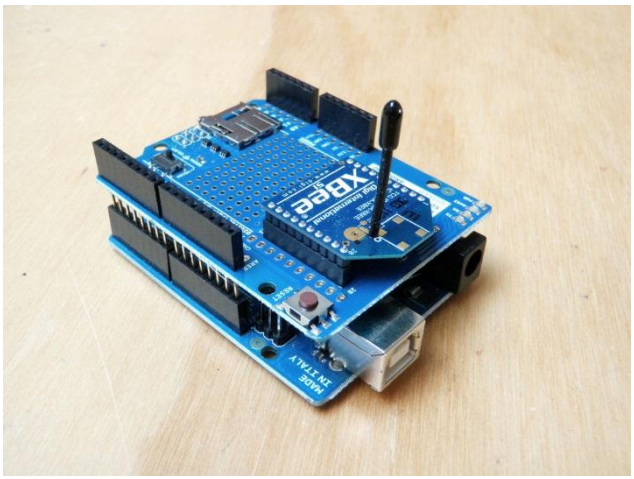


Figura 32: *Shield wireless* sd card.

5. Comunicação sem fio

Comunicação é uma palavra derivada do termo latino *communicare* que significa, partilhar algo, participar, pôr em comum, a comunicação no âmbito humano é o meio pelo qual as pessoas conseguem se entender e pode se tornar um instrumento de integração, instrução e troca mútua de informações. A busca por uma forma de comunicação eficaz capaz de difundir informação entre dois ou mais indivíduos que não ficasse restrita frente as grandes distâncias que separavam vilarejos, cidades ou até mesmo países sempre foi um desafio para as sociedades humanas ao longo da história, sendo estas barreiras rompidas a partir da segunda metade do século XIX graças as descobertas de homens como Nicola Tesla, James Clerk Maxwell, e Heinrich Hertz no campo do eletromagnetismo.

Os avanços tecnológicos no que cerne a comunição sem fio surgiu primeiramente com a motivação de desenvolver e aprimorar a comunicação a longas distâncias, o primeiro sistema de telegrafia sem fio foi desenvolvido por Guglielmo Marconi um físico e inventor italiano em 1897, no qual conseguiu manter contato contínuo com um návio no canal da mancha na inglaterra (RAPPAPORT, 2009, p. 20). Desde então a comunicação sem fio experimentou um crescimento espectacular, hoje em dia as pessoas conseguem se comunicar com dispositivos móveis em tempo real estando em praticamente qualquer lugar do planeta e o que se observa é um constante crescimento na comunicação sem fio não só entre seres humanos, mas entre dispositivos eletrônicos que interagem entre si, principalmente em redes sem fio de curto e médio alcance com as mais variadas finalidades e é neste contexto que surge a comunicação eletrônica sem fio.

A comunicação eletrônica sem fio consiste basicamente em fazer dois ou mais dispositivos eletrônicos se comunicarem usando o espaço como meio físico de transmissao de informações (RAMOS, 2010, p. 1), este processo esta fundamentado no princípio de que quando se varia o campo elétrico em um circuito emissor este propaga uma onda eletromagnética no espaço, e nesta onda pode-se "embutir" uma mensagem a ser transmitida, se um outro dispositivo consegue captar essas ondas e entender a mensagem contida nas mesmas pode-se dizer que esses dois dispositivos estabeleceram comunicação sem fio.

Segundo (RAMOS, 2012, p. 17) as técnicas de comunicação que utilizam cabeamento vem perdendo espaço em algumas aplicações em detrimento dos sistemas de comunicação sem fio, e isso esta diretamente ligado a baixa infraestrutura necessária para implementar um sistema de comunicação sem fio.

5.1. Redes WPAN

As redes WPAN (do inglês, W*ireless Personal Area Network*) são redes pessoais sem fio, estão incluídas na norma 802.15 do *IEEE* (*Institute of Electrical and Eletronics Engineers*) foram inicialmente concebidas para interconectar dispositivos como notebooks, celulares, fones de ouvios, teclado e outros, dispostos em um pequeno raio de alcance tipicamente 10 metros

no máximo (ZUCATO, 2009, p. 26). O *bluetooth* foi a primeira rede *WPAN* proposta, porém, esta limitado a uma taxa de transmissão de 1Mbps, e para aplicações onde é necessária uma maior taxa de transferência de dados foi criado a rede UWB (do inglês, *Ultra Wide Band*) podendo tranferir até 480Mbps num raio de 3m, entretanto, essas duas redes não são adequadas para dispositivos onde o baixo consumo de energia é extremante relevante e não necessite de altas taxas de transmissão de dados, como por exemplo rede de sensores industriais, e para prencher esta lacuna foi desenvolvido o padrao IEEE 802.15.4 que é uma subclasse das redes WPAN sendo denominado LR-WPAN *(Low cost wireless personal area network*) que é a base para o padrão *ZigBee* que será visto a seguir.

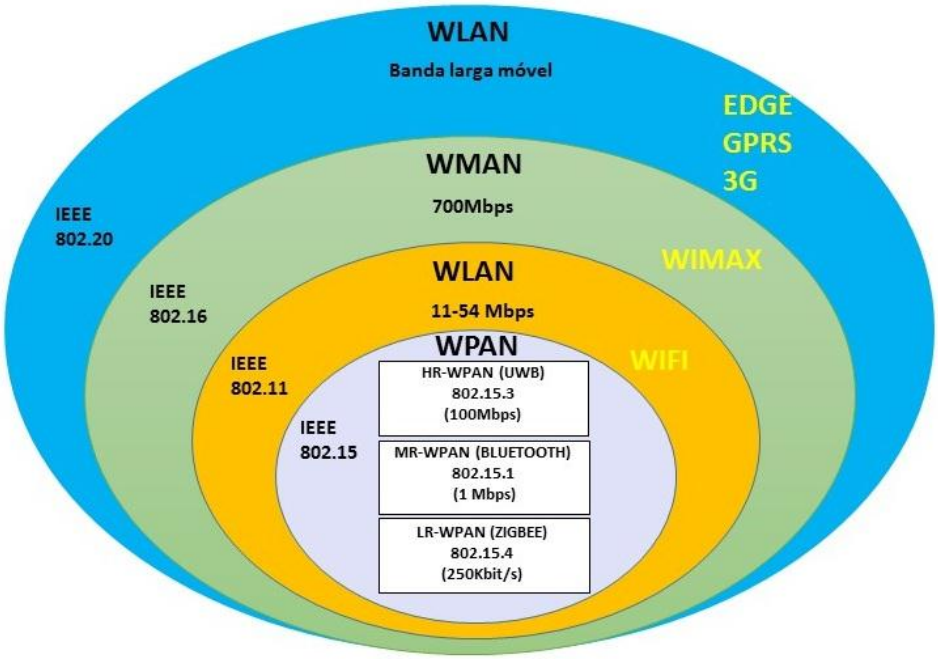


Figura 33: Redes wireless.

Tabela 7: Diferenças entre o *Bluetooth* e o padrão *ZigBee*.

	Taxa bps	Alcance (m)	Nº Nós	Duração bateria	Aplicação típca
802.15.4 (Zigbee)	250 k	10 a 100	65535	1 a 7 dias	Rede de sensores sem fio.
802.15.1 (Bluetooth)	1 M	1 a 10	7	10^2 a 10^3 dias	Fones, mouse sem fio

5.2. Padrão IEEE 802.15.4

O 802.15.4 ou *LR-WPAN* é um padrão de comunicação sem fio criado pelo *IEEE* (*Institute of Electrical and Eletronics Engineers*), é neste padrão que são definidas as especificações das camadas físicas (*PHY*) e de controle de acesso ao meio (*MAC*) das redes LR-*WPAN*.

É utilizado em aplicações que requerem baixo consumo de potência e que possuem baixa taxas de transmissão de dados, seus principais objetivos e vantagens (IEEE, 2011) são:

- Fácil instalação
- Confiabilidade na transferência de dados
- Custo extremamente baixos
- Baixo consumo de energia
- Protocolo simples e fexivel

5.2.1. Camada física

Na camada física (PHY) é são definidas as características de *hardware* e os comandos elétricos, sua principal função é adequar as informaçõs externas para a camada de controle de acesso ao meio (MAC), dentre as principais funções podem ser citadas as seguintes (RAMOS, 2012):

- Recepção e transmissão de dados digitais;
- Ativação/ desativação do transceptor de rádio;
- Detecção de energia no canal a ser usado;
- Seleção do canal a ser utilizado;
- Indicação da qualidade das conexões através dos pacotes recebidos (*link quality indication*-LQI);
- Técnicas de múltiplo acesso ao canal.

5.2.2. Camada de controle de acesso ao meio

A camada MAC (Medium Access Control) é responsavel por tudo que envolve o canal físico de comunicação podemos citar como principais funcionalidades dessa camada:

- Fornecimento de uma conexão confiável entre duas entidades MAC;
- Manipulação do mecanismo da técnica de acesso ao canal;

- Associação/ desassociação de PAN (Personal Area Network);
- Sincronismo de *beacon*;
- Geraçao de *beacon*;
- Segurança do dispositivo.

5.2.3. Propriedades do IEEE 802.15.4

O IEEE 802.15.4 opera na banda de frequência ISM (*Industrial Scientific Medical*), está lincensiado para trabalhar em três faixas, sendo 868 MHz com 1 canal de comunicação utilizado na europa, 915 MHz com 10 canais de comunicação usada na américa do norte e na américa do sul, e 2.4 GHz que é a única banda que pode operar em todo o mundo usando 16 canais e taxa de transmissão de 250 Kbps, por se tratar de uma banda utilizada na indústria, ciência e medicina os dispositivos que operam nessas faixas não precisam de licensa para funcionarem.

5.3. O padrão *ZigBee*

O padrão Z*igBee* é um padrão de comunicação sem fio que foi desenvolvido em 2002 quando grandes empresas do setor de eletrônicos se reuniram e formaram uma aliança denominada de *alliance ZigBee*, esta aliança surgiu com o objetivo de padronizar um sistema de comunicação sem fio de baixo custo e baixa potência de transmissão voltado para o setor de automação, e tornar este sistema compatível independente quem fabricasse o *hardware*. O Z*igBee* é um padrão aberto, ou seja, livre para aquisição de toda documentação e as especificações estão disponível para o público em geral, mas para uma empresa usar a marca *ZigBee* esta deve associar-se a *alliance ZigBee* e pagar uma taxa anual de acordo com um dos três tipos de membros que é disponibilizado para associação.

O *ZigBee* segue o padrão de camada *OSI-ISO* entretando, não tem as sete camadas que normalmente se tem nesse padrão, possui apenas as camadas essências visando manter o baixo custo, a baixa complexidade e o baixo consumo de energia.

As duas camadas de nível mais alto foram produzidas pelo protocolo Z*igBee* e as duas camadas inferiores pelo protocolo do *IEEE* 802.15.4 ou seja, dois tipos de protocolos integram o padrão de comunicação *ZigBee*.

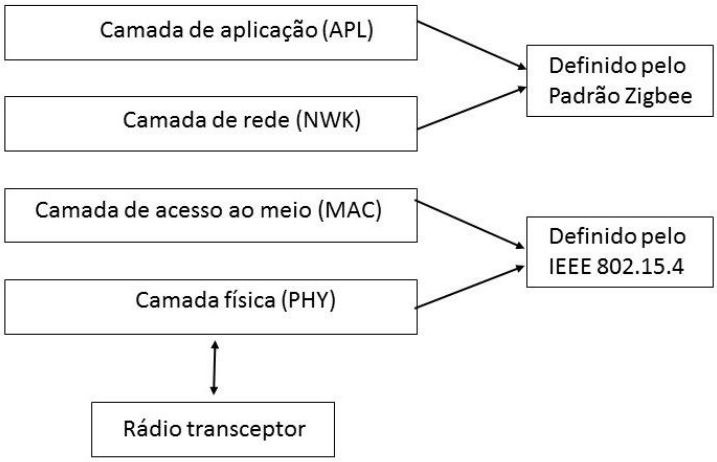


Figura 34: Camadas de protocolo da rede *ZigBee* (Ramos, 2012, p. 48).

5.3.1 Tipos de dispositivos *ZigBee*

Podemos identificar dois tipos de dispositivos físicos no padrão *ZigBee.*

- *Full Function Device* (FFD)
- *Reduced Function Device* (RFD)

O tipo de dispositivo no padrão *ZigBee* esta diretamente ligado a quantidade de atividades que este dispositivo pode realizar e quais tarefas estão armazenadas na pilha de instrução em seu microcontrolador (RAMOS, 2012, p. 45). Dispositivos FFD são capazes de formar uma rede, gerenciar e rotear dados, enquanto que os dispostivos RFD podem apenas juntar-se a rede e assumir funções simples como comutar uma chave ou acender uma lâmpada, exercendo atividades que exigem pouco poder de processamento, memória e consequentemente pouca energia.

5.3.2 Papéis dos dispositivos

Podemos ter os seguintes papéis uma rede zigbee:

✓ **Coordenador**

O coordenador é responsável por formar a rede , selecionar um acanal de comunicação e um endereço específico, permite o acesso ou não de outros dispositivos que pretendam se associar á rede por ele formada, o papel de coordenador é exercido por dispositivos do tipo FFD

como descrito anteriormente, por conseguinte possui a pilha de instrução completa em sua memória.

- ✓ **Roteador**

Os roteadores são responsavéis por ofereceram rotas alternativas aos pacotes de dados que trafegam na rede, são dispositovos do tipo *FFD*.

- ✓ **Dispositivos finas**

Exercem poucas tarefas e geralmente estão ligados a sensores, são dispositivos do tipo *RFD* cujo conjunto de instruçoes carregados em sua memória são bastante reduzidos, este tipo de dispositivo pode apenas se comunicar com dispositivos do tipo FFD necessitando destes para entrarem na rede.

5.3.3 Topologia de rede *ZigBee*

A topologia de rede foi estabalecida pela camada de rede do protocolo *IEEE* 802.15.4 (*IEEE*, 2011, p. 8), portanto uma rede *ZigBee* pode assumir duas topologias sendos estas estrela ou topologia *peer-to-peer*.

Numa topologia estrela dispositivos finais apenas se comunicam com o coordenador, onde os dados não podem ser retransmitidos pelos roteadores, na topologia peer-to-peer dispositivos *FFDs* (coordenadores e roteadores) podem se comunicar com o dispositivo *FFD* mais próximo, sendo nesta rede os dispositivos exercendo todos os papéis do padrão.

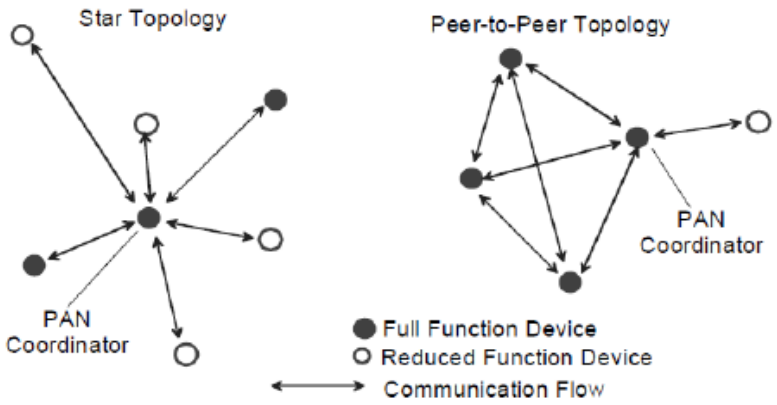


Figura 35: Topologia rede *ZigBee*, fonte (IEEE, 2011, p. 9).

Dependendo de como é organizado a inclusão de novos dispositivos na rede, a topologia peer-to-peer pode assumir duas subdivisões (RAMOS, 2012, p. 50), topologia rede *mesh* e topologia de rede em árvore.

Na topologia *mesh* a rede pode se ajustar automaticamente tanto na inicialização quanto na entrada de novos módulos na rede, módulos *FFDs* podem se comunicar com módulos *FFDs* da rede que estão ao seu alcance físico, o único impedimento para não haver comunicação é a potência de radiação, dispositivos finais podem usar roteadores para se comunicar com o coordenador Figura 36.

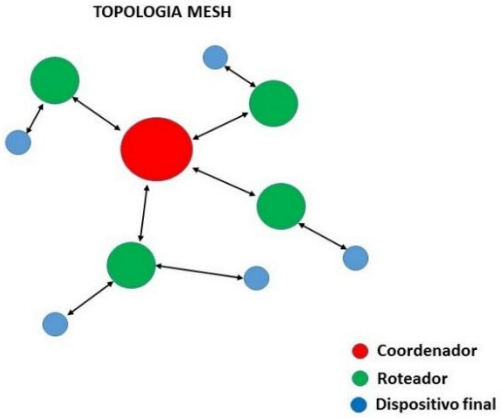


Figura 36: Topologia *mesh* (malha).

Na topologia árvore a rede cresce de forma organizada e metódica, os roteadores funcionam como galhos e dispositivos finais como folhas. O coordenador assume o papel de nó mestre da rede e se algum dispositivo final deseja se juntar a rede este deve procurar o coordenador mais próximo que atua como módulo pai para recebê-lo, nesta topologia a rede pode se expandir com inclusão de novos coordenadores.

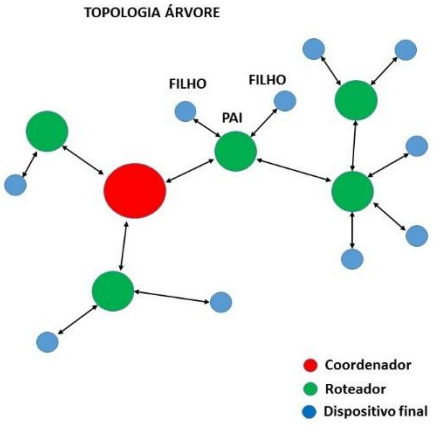


Figura 37: Topologia árvore (*cluster tree*).

5.3.4 Formação e junção a uma rede *ZigBee*

Para formação de uma rede *ZigBee* é necessário primeiro que haja um módulo *FFD* configurado como coordenador da rede, esse módulo gera uma rede de trabalho chamada *PAN*, esta *PAN* possui um endereço único e específico chamado de *PAN ID* é através da *PAN ID* que se consegue diferenciar as redes *ZigBee* formadas, esse endereço é armazenado em cada módulo ao se juntar a rede. Vale ressaltar que em cada rede formada só há um coordenador.

Ao energizar o coordenador ele inicia o processo de criação da rede de trabalho fazendo uma varredura (*energy scan*) em todos os canais de frequência para detectar os níveis de energia em cada canal (DIGI INTERNATIONAL, 2008, p. 19), os canais com níveis excessivos de energia são descartados da lista de candidatos a iniciar a rede.

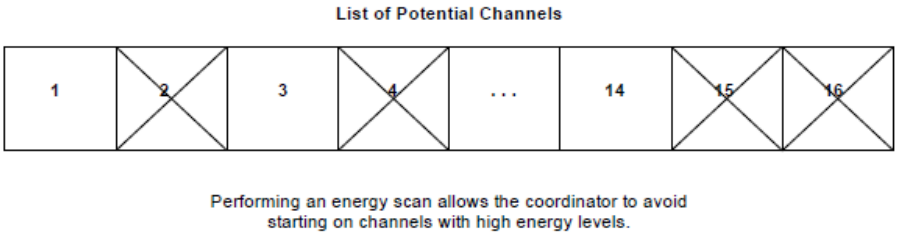


Figura 38: Lista de canais potencias para rede ZigBee, canais possiveis:1, 3, 14
fonte (DIGI INTERNATIONAL, 2008, p. 19).

Após finalizar o *energy scan* o coordenador realiza uma outra varredura nos canais restantes (*pan scan*) enviando um pacote em *broadcast* (ver item **5.3.6.1 Transmissão broadcast**) se algum coordenador ou roteador receber esse pacote enviará um pacote de resposta, neste pacote contém entre outras coisas a *PAN ID* e a informação dizendo se é ou não permitida entrada de dispositivos a rede qual pertecem. Feito isto, o coordenador analisa todas as respostas recebidas e tenta iniciar uma *PAN* e então poderá permitir que roteadores e dispositivos finais se juntem para formar a rede de trabalho.

5.3.5. Endereçamento dos dispositivos *ZigBee*

Há dois tipo de endereços de dispositivos numa rede ZigBee.

5.3.5.1. Endereço de 64 bits

É um endereço fixo e único que os dispositivos Z*igBee* recebem no momento de sua fabricação.

5.3.5.2. Endereço de 16 bits

É um endereço dinâmico que os dispositivos recebem do coordenador ou roteador no momento em que aderem a um *PAN*, sendo também denominado como endereço de rede.

5.3.6. Tipos de transmissão de dados *ZigBee*

Há duas formas de transmitir dados no padrão *ZigBee*.

5.3.6.1 Transmissão *broadcast*

Neste tipo de transmissão os pacotes de dados serão enviados a todos os dispositivos finais da rede, para garantir que todos os dispositivos finais irão receber os dados enviados, os coordenadores e roteadores quando recebem o comando para transmitir em *broadcast* retransmitem três vezes o pacote.

5.3.6.2 Transmissão *unicast*

Neste tipo de transmissão os pacotes só podem ser enviados de um módulo remetente a um módulo destinatário conhecendo o seu endereço de 16 bits, para isso, é preciso descobri-lo da seguinte forma: O módulo que deseja enviar o pacote, faz primeiramente uma busca pela rede através de uma transmissão em *broadcast* contendo o endereço de 64 bits do módulo destinatário, ao receber esse pacote os módulos da rede comparam seu endereço de 64 bits com o que foi enviado, o módulo correspondente retorna com o seu endereço de 16 bits e posteriormente o pacote lhe é enviado.

5.3.7. Modos de operação dos dispositivos *ZigBee*

Os dispositivos numa rede *ZigBee* podem operar em cinco modos de operação diferentes (DIGI INTERNATIONAL, 2013), o modo de operação reflete diretamente o consumo de energia dos módulos e por conseguinte da rede.

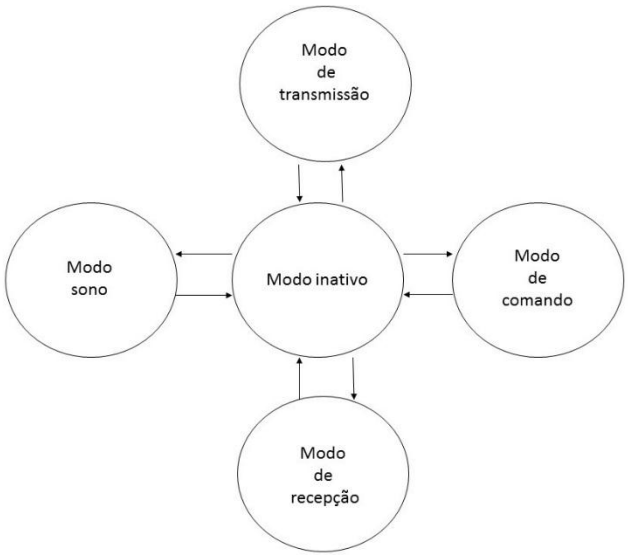


Figura 39: Modos de operação, adaptado de (DIGI INTERNATIONAL, 2013, p. 23).

5.3.7.1 Modo de repouso

É neste modo que se tem o menor consumo de energia, os módulos ficam "dormindo" até serem solicitados, observa-se que um estudo de quando dormir ou acordar deve ser realizado principalmente em redes razoavelmente grandes, visto que quando nesse modo dados não podem ser roteados ou transmitidos podendo prejudicar o desempenho da rede.

5.3.7.2. Modo inativo

Um módulo fica neste modo quando não esta transmitindo nem recebendo dados, quando o módulo sai do modo inativo ele vai para outro modo de acordo com as seguintes condições:

- ✓ Modo de transmissão (Se dados seriais estão vindo do *buffer*);
- ✓ Modo de recepção (Se recebeu dados válidos através da antena);
- ✓ Modo de repouso (Se Condições para o modo de repouso são encontradas);
- ✓ Modo de comando (Se é iniciada uma sequência de dados de comando);

5.3.7.3. Modo de recepção

O módulo entra em modo de recepção quando um dado válido é recebido, após isso o dado é transferido para o *buffer* de transmissão serial.

5.3.7.4. Modo de comando

Para modificar ou ler parâmetros dos módulos na rede, estes devem estar em modo de comando (*Digi international*, 2013, p. 27) neste modo os dados são interpretados como linhas de comando, há dois de modo de comando, *AT comand mode* e *API comand mode*, de acordo com o modo de comunicação (ver item **5.6. Modo de comunicação dos módulos XBee**).

5.3.8 Perfil de aplicação *ZigBee*

Foi visto que as camadas mais altas do padrão *IEEE 802.15.4* são implementadas pelo usuário no caso o Z*igBee alliance*, porém estas camadas são flexíveis no que diz respeito à aplicação, ou seja, pode-se desenvolver uma aplicação para uma finalidade específica, com isso o *ZigBee alliance* desenvolveu diversos perfis para o padrão *ZigBee* de acordo o tipo de trabalho que o protocolo irá realizar, segue alguns exemplos:

- *ZigBee health care;*
- *ZigBee home automation;*
- *ZigBee remote control;*
- *ZigBee smart energy;*

5.3.9 Segurança na rede *ZigBee*

Numa rede ZigBee os dados podem ser codificados com uma chave de encriptação de 128 bits, nessa rede codificada dispositivos podem receber a chave de duas formas:

- Pré-configuração;
- Recebem por *RF* no momento que aderem a *PAN*

5.4 Módulos XBee

O módulo *XBee* é uma plataforma embarcada de rádio frequência baseado nas definições do *IEEE* 802.15.4 e o *ZigBee alliance*. Existem diversas empresas que produzem módulos *XBee*, porém, a pinagem o espaçamento entre os pinos, e até mesmo a função de cada pino são iguais.

Figura 40 : Módulo *XBee*, fonte (MAXSTREAM, 2007).

A junção de *hardware*, protocolo e *firmware* define o produto *XBee* final, e a qual série ele pertence, existem diversas séries, entrentanto, vale destacar que um módulo pertecente a uma série não se comunica com outro módulo de série diferente. Existem também módulos *XBee* que contam com protocolos proprietários como o *Digimesh* e o *Multipoint* da *Digi international*.

Tabela 8: Pinos e funções dos módulos *XBee*, fonte (DIGI INTERNATIONAL, 2013, p. 7).

Pino	Nome	Direção	Descrição
1	VCC	-	Alimentação elétrica constante
2	DOUT	SAÍDA	Saída de dado UART
3	DIN / $\overline{CONFIG}$	ENTRADA	Entradad de dado UART
4	DIO 12	SAÍDA	Entrada/saída digital
5	$\overline{RESET}$	ENTRADA	Reset do módulo(pulso de pelo menos 200 ms)
6	RSSI PWM/DIO 10	SAÍDA	Potência do sina RX ou Entrada/saída digital 10.
7	DIO11	ENTRADA	Entrada/saída digital 11
8	RESERVADO	-	Desconectado
9	$\overline{DTR}$ / SLEEP_RQ / DIO8	ENTRADA	Controle de modo sono, ou entrada/saída digital 8.
10	GND	-	Ligação ao terra
11	DIO4	AMBAS	Entrada/saída digital 4
12	$\overline{CTS}$ / DIO7	AMBAS	Entrada/saída digital 7
13	ON / $\overline{SLEEP}$	SAIDA	Indicação do estado do módulo ou saída digital
14	VREF	ENTRADA	Entrada de tensão de referência
15	ASSOCIAÇÃO / DIO 5	AMBAS	Indicação de associação ou entrada/saída digital 5
16	RTS/DIO6	AMBAS	Controle de fluxo que requita dados UART de envio, caso habilitado é entrada/saída digital 6.
17	AD3/DIO3	AMBAS	Entrada analógica 3 ou Entrada/saída digital 3.
18	AD2/DIO2	AMBAS	Entrada analógica 2 ou Entrada/saída digital 2.
19	AD1/DIO1	AMBAS	Entrada analógica 1 ou Entrada/saída digital 1.
20	AD0/DIO0	AMBAS	Entrada analógica 0 ou Entrada/saída digital 0.

5.4.1. Classificação dos módulos *XBee*

Os módulos *XBee* podem ser classificados quanto a sua potência de transmissão como *XBee* ou *XBee-Pro.* Os módulos *XBee-Pro* têm uma maior potência de transmissão tanto em ambientes internos quanto em ambientes externos em relação aos módulos *XBee*, naturalmente são mais caros e são usados quando a aplicação requer distância consideráveis no tráfego das informações. Cada série pode ter os dois tipos de *hardwares*, abaixo vemos uma tabela de comparação das características entre esses dois tipos. Existem no mercado atualmente módulos *XBee* até a série 6, porém, neste projeto de graduação serão descritos apenas os hardwares das série 1 e série 2 que são notadamente os mais usuais e dão uma boa base de entendimento para módulos de qualquer outra série.

Tabela 9: Comparação entre módulos *XBee* e *XBee-Pro*, fonte (RAMOS, 2012, p. 66).

	802.15.4 (S1)		**ZigBee S2**	
	Xbee	Xbee-Pro	Xbee	Xbee-Pro
Distâncias internas	30 m	90 m	40 m	90 m
Distâncias Externas	90 m	1600 m	120 m	3200 m
Potência de transmissão	1mW(0 dBm)	63 mW	2mW	50 mW (17 dBm)
Sensibilidade de recepção	- 92dBm	-100dBm	-96 dBm	-102 dBm
Taxa de transmissão	250 Kbps	250Kpbs	250 Kpbs	250 Kpbs

5.4.1.1. Módulos Série 1 (S1)

Os módulos da série 1 suportam os protocolos *IEEE* 802.15.4 e *Digimesh* e como já mencionado anteriormente nesta série temos *XBee* e *XBee-Pro* distribuídos da seguinte maneira (RAMOS, 2012, p. 70):

- **XB24 (*XBee*)**

É formado pelo *hardware* da série 1 com o protocolo *IEEE* 802.15.4 com as principais características:

- ✓ Baixa potência de transmissão;
- ✓ Aceita topologias peer-to-peer, point-to-point, point-to-multipoint;
- ✓ Aceita os papéis de coordenador e dispositivos finais.

- **XBP24 (*XBee*-Pro)**

É formado pelo *hardware* da série 1 com o protocolo IEEE 802.15.4, diferenciando-se XB24 apenas no quesito de potência de transmissão, possuindo seguintes as principais características:

- ✓ Alta potência de transmissão;
- ✓ Aceita topologias peer-t-peer, point-to-point, point-to-multipoint;
- ✓ Aceitam os papéis de coordenador e dispositivos finais.

- **XB24 DM (*XBee*)**

É formado pelo *hardware* da série 1 com o protocolo proprietário *Digimesh*, as principais características são:

- ✓ Baixa potência de transmissão;
- ✓ Aceita topologias peer-t-peer, point-to-point, point-to-multipoint, *mesh;*
- ✓ Aceita os papéis de roteador e dispositivos finais.

- **XBP24 DM (*XBee*-Pro)**

É formado pelo *hardware* da série 1 com o protocolo proprietário *Digimesh*, as principais características são:

- ✓ Alta potência de transmissão;
- ✓ Aceita topologias peer-t-peer, point-to-point, point-to-multipoint, *mesh;*
- ✓ Aceita os papéis de roteador e dispositivos finais.

5.4.1.2. Série 2 (S2)

Os módulos da série 2 são a junção do *hardware* da série 2 com o protocolo Z*igBee* e podem identificados como:

- **XB24 ZB (*XBee2 ZigBee*)**

Características:

- ✓ Baixa potência de transmissão (até 120m);
- ✓ Pode assumir as topologias árvores ou *mesh*;
- ✓ Pode assumir os papéis de coordenador, roteador, ou dispositivo final.

- **XBP24 ZB (*XBee*-Pro2 *ZigBee*)**

Características:

- ✓ Alta potência de transmissão (até 3200m);
- ✓ Pode assumir as topologias árvores ou *mesh*;
- ✓ Pode assumir os papéis de coordenador, roteador, ou dispositivo final.

- **XB24 SE (XBee2 smart energy)**

É formado pelo hardware da série 2 com o protocolo *ZigBee* com o perfil *smart energy*.

Características:

✓ Baixa potência de transmissão (até 120m);
✓ Pode assumir as topologias árvores ou *mesh*;
✓ Pode assumir os papéis de coordenador, roteador, ou dispositivo final.

- **XBP24 SE (*XBee-Pro2 smart energy*)**

É formado pelo hardware da série 2 com o protocolo *ZigBee* com o perfil *smart energy*.

Características:

✓ Alta potência de transmissão (até 3200m);
✓ Pode assumir as topologias árvores ou mesh;
✓ Pode assumir os papéis de coordenador, roteador, ou dispositivo final.

5.4.1.3. Série 2B

São a junção do hardware da série 2B com o protocolo *ZigBee*.

- **XBP24 ZB (XBee-Pro S2B)**

Características:

✓ Alta potência de transmissão (até 3200m);
✓ Pode assumir as topologias árvores ou mesh;
✓ Pode assumir os papéis de coordenador, roteador, ou dispositivo final;
✓ Possui 32 KB de memória flash, e 2 KB de memória RAM (do inglês, random access memory).

- **XBP24 SE (*XBee*-Pro S2B *smart energy*)**

É a junção do hardware da série 2B com o protocolo *ZigBee* com o perfil *smart energy*, e características iguais ao módulo XBP24 ZB.

5.4.1.4. Série 2C

São a junção do hardware da série 2C com o protocolo *ZigBee* e está dividido da seguinte forma:

- **XB24 ZB SMT (*XBee* S2C)**

Características:

✓ Média potência de transmissão (até 1200m);
✓ Pode assumir as topologias árvores ou *mesh*;
✓ Pode assumir os papéis de coordenador, roteador, ou dispositivo final;
✓ Possui 32 KB de memória flash, e 2 KB de memória RAM (do inglês, *random access memory*);

✓ Comunicação *SPI* com taxa de 5Mbps.

- **XBP24 ZB SMT (*XBee* S2C)**

Características semelhantes ao XB24ZB SMT, com exceção da potência de transmissão que atinge até 3200m.

5.5. Comunicação serial dos módulos *XBee*

Os módulos *XBee* se comunicam com outros dispositivos através de comunicação serial assíncrona (RS 232), esses módulos podem se comunicar com qualquer de nível lógico compatível com UART (*Universal Asynchronous Receiver/Transmitter*).

Qualquer dispositivo que tenha interface UART pode enviar dados para os módulos XBee através do pino DIN (data in), os dados entram no módulo de forma assíncrona e devem estar em *idle* nível lógico alto para começarem a ser transmitidos (Figura 42).

Cada *byte* consiste de um *start byte* (nível lógico baixo), oito bits de dados com os bits menos significantes primeiro e um bit de parada em nível lógico alto. O módulo UART realiza tarefas como verificação de tempo e paridade que são necessárias para comunicação de dados. Os módulos XBee têm dois pequenos *buffers* (DIGI INTERNATIONAL, 2008, p. 11) para armazenar dados vindo da comunicação serial ou dados RF.

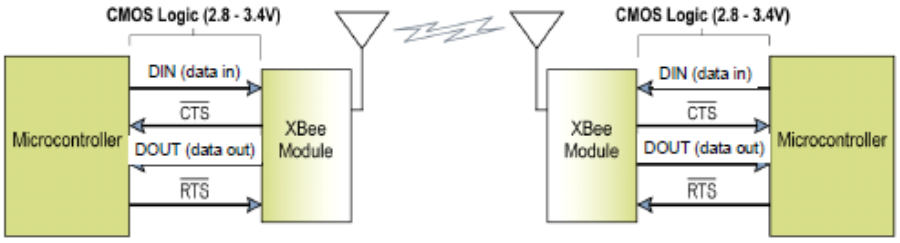


Figura 41: Diagrama de fluxo de dado UART, fonte (DIGI INTERNATIONAL, 2008, p. 11**).**

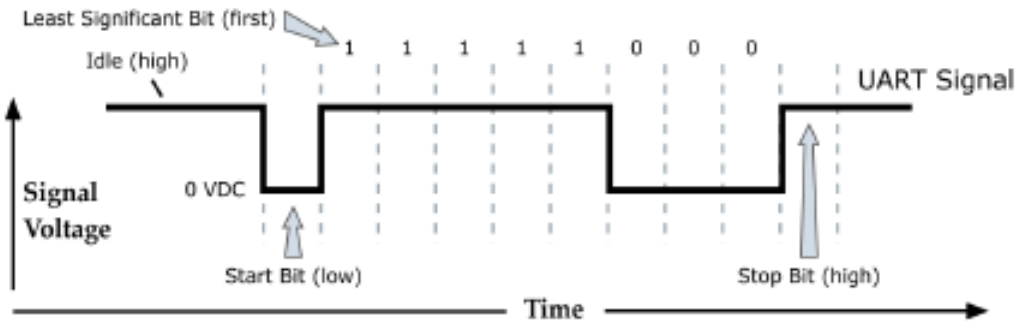


Figura 42: Estrutura de um pacote de dados UART. (DIGI INTERNATIONAL,2008, p. 11).

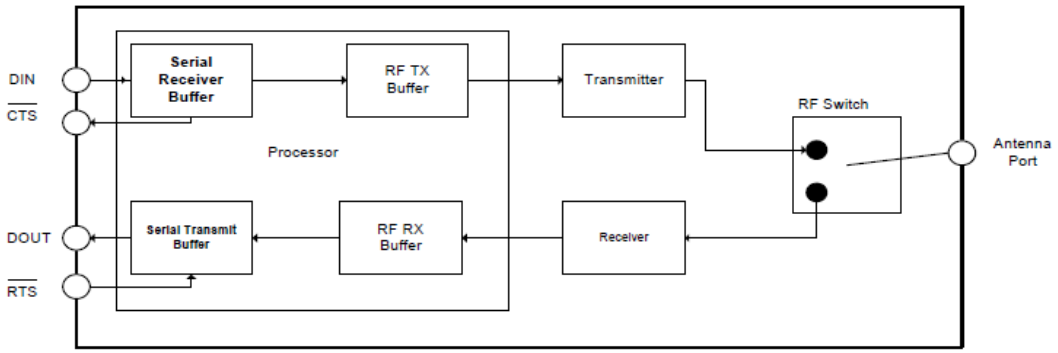


Figura 43: Fluxo interno de dados. (DIGI INTERNATIONAL,2008, p. 11).

5.6. Modo de comunicação dos módulos *XBee*

Módulos *XBee* suportam dois modos de comunicação serial, modo transparente e modo API.

5.6.1 Modo transparente

Neste modo de comunicação os bits que vem da *UART* são diretamente transmitidos para outros módulos como se houvesse um *link serial* cabeado.

Este modo de transmissão pode ser escolhido quando deseja-se criar uma rede simples para comunicação, como por exemplo com dois módulos *XBee*, em que não há preocupação com interpretação de pacotes de dados recebidos nem transmitidos, nem preocupação com endereçamento ou roteamento de dados.

5.6.1 Modo API

Neste modo de comunicação as informações enviadas e recebidas são implementadas através de pacotes de dados com uma estrutura pré-estabelecida. A estruturação e explicação específica de todos os pacotes API foge ao escopo desse trabalho, sendo apresentado com detalhes apenas os pacotes que foram utilizados para realização desse projeto.

Este modo de transmissão pode ser usado quando deseja-se criar uma rede *ZigBee* mais completa e segura em relação as redes em modo transparente, no modo API podemos endereçar cada pacote a um nó específico da rede, receber sinalizações de sucesso ou falha de pacotes enviados, configurar e ler parâmetros de módulos remotos, saber de qual módulo da rede é cada pacote recebido, entre outros. A seguir é apresentado o formado padrão de um pacote API e sua estrutura básica. No modo API há duas classes de comunicação, sem caractere sinalizado ou com caractere sinalizado, no modo com caractere sinalizado alguns caracteres precisam ser escapados (*escape characters*) quando transmitidos ou recebidos para não interferirem na estrutura de dados da comunicação UART, neste projeto usa-se o modo sem caractere sinalizado.

5.6.1.1. Estrutura dos pacotes em modo API sem caractere sinalizado

Os dados são organizados da seguinte forma:

Tabela 10: Estrutura dos pacotes API.

<table>
<tr><td>Start delimiter (byte1)</td><td colspan="2">Lenght (byte 2 até 3)</td><td colspan="2">Frame data (byte 4 até n)</td><td>Check Sum (byte n+1)</td></tr>
<tr><td>0x7E</td><td>MSB</td><td>LSB</td><td colspan="2">API–specific structure</td><td>1 byte</td></tr>
<tr><td></td><td></td><td></td><td>cmdID</td><td>cmdDATA</td><td></td></tr>
</table>

MSB= most significant byte (byte mais significante), LSB= least significant byte (byte menos significante).

- Start delimiter (Delimitador inicial)

Byte que indica ao módulo que recebe o pacote que se iniciou uma comunicação em modo API.

- Lenght (comprimento)

Bytes que indicam o comprimento do frame data, esse campo é constituído por dois *bytes*.

- Frame data (estrutura do pacote)

Este campo está dividido em duas partes o cmdID e o cmdDATA.

- ✓ CmdID – informa aos módulos *XBee* remotos qual comando API deverá ser executado, sendo estes comandos apresentados na tabela seguinte:

Tabela 11: Comandos *API*.

Nome do comando API ID	**Função**	**Valor hexadecimal**
AT command	Configura salvando ou lê parâmetro do módulo local	0x08
AT command-Queue parameter value	Configure sem salvar ou lê parâmetros de um módulo local	0x09
Zigbee transmit Request	Solicita uma transmissão a um endereço	0x10
Explicit Addressing zigbee comand frame	Permite que end point ou cluster ID sejam especificados por um pacote de transmissão	0x11
Remote AT comand Request	Solicita ou configura parâmetros em um módulo remoto com comandos AT	0x17
Create Source Route	Cria uma rota de comunicação	0x21

AT comand Response	Responde a uma solicitação de um comando AT	0x88
Modem Status	Indica o status do módulo para um determinado evento	0x8A
Zigbee transmit status	Indica status de transmissão do último pacote	0x8B
Zigbee Receive Packet (A0=0)	Recebe um pacote pela UART do módulo	0x90
Zigbee Explicit Rx (A0=1)	O módulo envia esse pacote quando recebe um pacote do tipo Zigbee Transmit Packet	0x91
Zigbee IO Data Sample Rx indicator	Módulo envia pela UART o estado lógico das portas digitais e valores da conversão analógica	0x92
Xbee Sensor Read Indicator (A0=0)	Recebe leitura de sensores	0x94
Node Identification Indicator (A0=0)	Recebe identificações dos módulos remotos	0x95
Remote Command Response	Recebe em resposta ao Remote Command Request	0x97
Over-the-Air- Firmware Update Status	Fornece a indicação do status da atualização do firmware	0xA0
Route Record Indicator	É recebido quando o módulo envia um Zigbee Route Record command	0xA1
Many-to-One Route Request Indicator	É enviado quando many-to-one Route request command é recebido	0XA3

- CmdDATA

Este campo é composto pelo endereço de 16 *bits* e o endereço de 64 *bits* (seção 5.3.5) quando efetuada comunicação com módulos remotos.

- *Check Sum*

Este *byte* tem a função de certificar se os pacotes recebidos pelos módulos estão coerentes com aquilo que foi enviado, para isso o *Checksum* faz uma verificação nos dados, os campos *Start delimiter* e *Lenght* não entram nesta verificação.

5.7. Configuração dos módulos *XBee*

Para configurar os módulos foi utilizado o *software XCTU* da empresa *Digi international* em ambiente *Windows* 7 e um adaptador *USB-SERIAL* da empresa *sparkfun* denominado *XBee explorer usb*, esse adaptador tem a função de fazer a comunicação entre o módulo *XBee* a ele conectado e o computador.

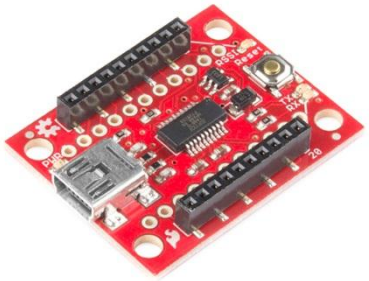


Figura 44: *XBee explorer USB*, fonte (https://www.sparkfun.com/products/11812).

O programa *XCTU* em sua versão mais recente pode ser baixado no *site* mostrado abaixo, após abrir a página e clicar no botão *DOWNLOAD XCTU*, também tem um vídeo tutorial em inglês que ensina as funcionalidades do *software*. Depois de baixar o instalador é necessário executá-lo e seguir a instalação aceitando os termos de uso do *software* e outras etapas, após a instalação ser concluída, para iniciar a programação dos módulos pode-se abrir o programa usando o atalho criado na área de trabalho aparecerá a tela mostrada na Figura 45.

Site: http://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu

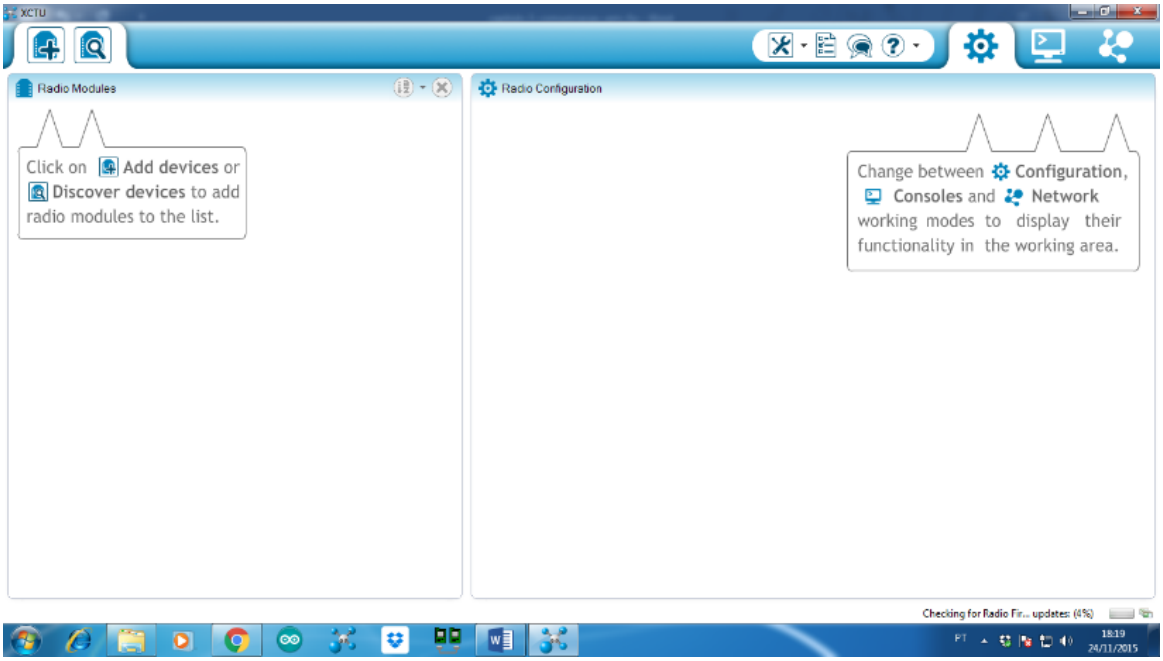


Figura 45: Tela inicial do programa XCTU.

O próximo passo é encaixar o módulo *XBee* no adaptador *explorer* e liga-lo ao computador através de um cabo *USB*.

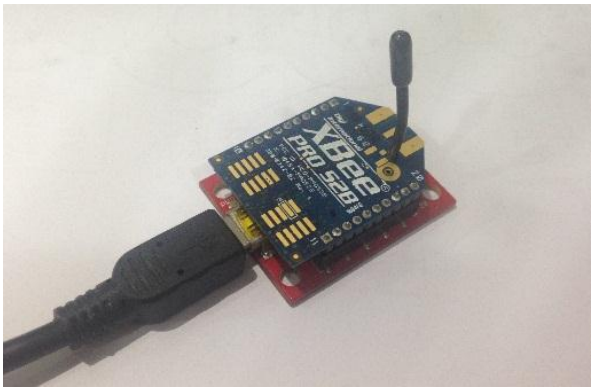


Figura 46: Adaptador *explorer* + módulo *XBee*.

No programa *XCTU* deve-se clicar no ícone do canto esquerdo superior com o símbolo de +, abrirá um a janela para escolha da porta *USB* na qual está ligado o módulo *XBee*, escolha a porta correta e aperte o botão *finish* o programa deverá reconhecer o módulo a ele ligado, ao clicar em cima da figura do módulo encontrado, o programa ler automaticamente os parâmetros nele configurados como pode ser visto abaixo.

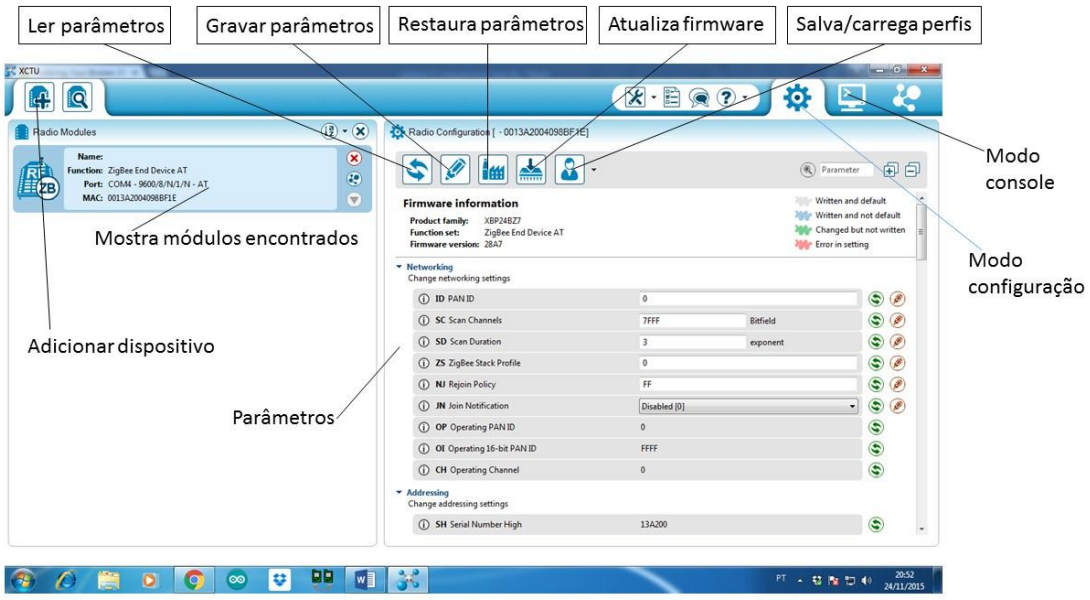


Figura 47: Aba modo configuração do XCTU.

Na etapa de configuração dos módulos foram feitos dois tipos de configurações, a primeira para o coordenador *API* da rede e a segunda para os dois módulos *XBee* que operam como dispositivo final *API* (*end device API)*. Os módulos *XBee* vem de fábrica para funcionarem como dispositivos finais em modo AT (transparente) por ter maior uso neste modo devido a facilidade na comunicação.

Neste projeto de graduação foram usados módulos *XBee* série 2 e foram configurados para operarem em modo API, justificando-se nas vantagens que este modo oferece. Estas vantagens foram apresentadas na seção anterior. Para operarem em modo API o *firmware* dos módulos precisa ser atualizado de acordo com o tipo de dispositivos que irão desempenhar e estes são:

- ✓ Coordenador

Após feitas as etapas de reconhecimento do módulo pelo software *XCTU* deve-se apertar o botão indicado como atualizar *firmware*, uma lista de opções de *firmware* será apresentada para escolha de acordo com o tipo de família do dispositivo, para o coordenador foi escolhido e instalado o *firmware* mais recente indicado com a função Z*igBee cordinator* API cuja versão é 21A7 a seguir é mostrada uma tabela com os parâmetros que foram alterados e carregados no módulo coordenador.

Tabela 12: Parâmetros do módulo coordenador.

Parâmetro	Ajuste	Ajuste padrão	Função
PAN ID	FF	0	Identificação da PAN
NI	Coordenador	‘ ’	Nome do dispositivo na rede
AP	2	1	(2) Modo API sem caractere sinalizado

- ✓ Dispositivos finais

As tabelas a seguir apresentam as configurações dos módulos remotos *XBee* que funcionam como dispositivo final com o nome de *ZigBee end device API* com a versão mais recente 29A7.

Tabela 13: Parâmetros do módulo motor_esquerdo.

Parâmetro	Ajuste	Ajuste padrão	Função
PAN ID	FF	0	Identificação da PAN
NI	Motor_esquerdo	‘ ’	Nome do dispositivo na rede
AP	2	1	(2) Modo API sem caractere sinalizado
DH	0013A200	0	Parte alta do endereço de 64 bis do módulo destino (coordenador)
DL	4098BF1E	0	Parte baixa do endereço de 64 bis do módulo destino (coordenador)

Tabela 14: Parâmetros do módulo motor_direito.

Parâmetro	Ajuste	Ajuste padrão	Função
PAN ID	FF	0	Identificação da PAN
NI	Motor_direito	‘ ’	Nome do dispositivo na rede
AP	2	1	(2) Modo API sem caractere sinalizado
DH	0013A200	0	Parte alta do endereço de 64 bis do módulo destino (coordenador)
DL	4098BF1E	0	Parte baixa do endereço de 64 bis do módulo destino (coordenador)

5.7.1 Testes com o *software XCTU*

Com este *software* além de configura parâmetros e atualizar o *firmware* dos módulos *XBee*, na aba *console mode* podemos criar uma rede *ZigBee*, adicionar dispositivos, enviar e receber pacotes, estes recursos são uma ótima forma de testar a comunicação da rede Z*igBee* a Figura 48 mostra a rede formada pelo coordenador e os dois dispositivos finais conectados a ele.

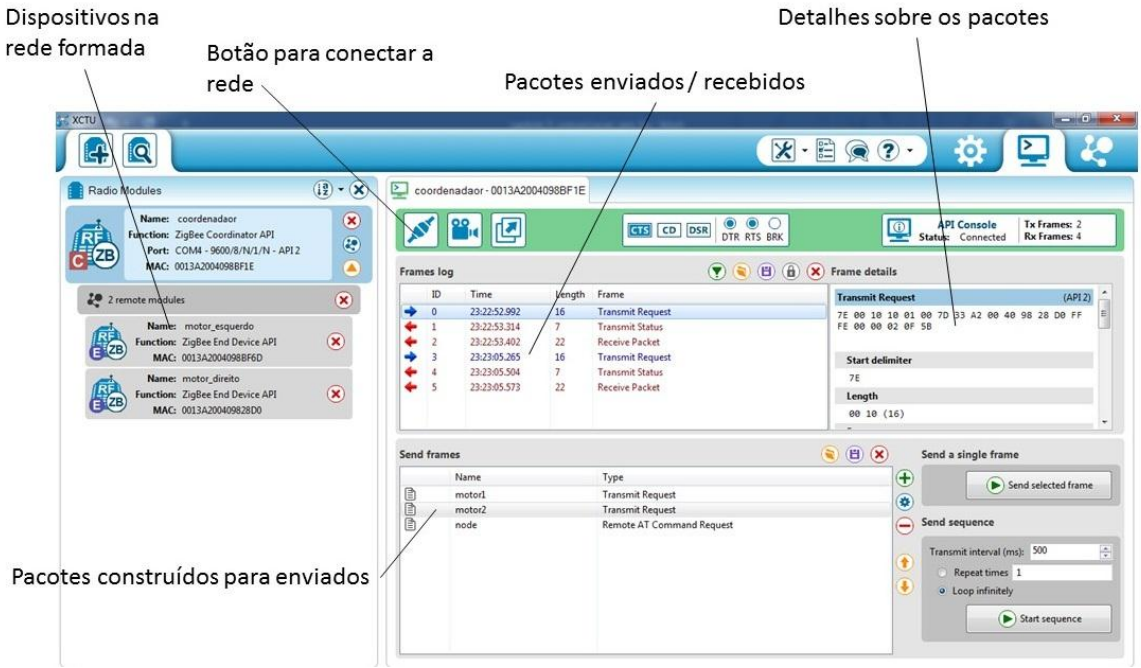


Figura 48: Rede *ZigBee* no software XCTU.

No exemplo acima no campo descrito como pacotes enviados/recebidos (*frame log*), os pacotes em azul são pacotes enviados e os pacotes em vermelho são os pacotes recebidos em resposta os pacotes ficam registrados nesse campo e têm uma identificação sequencial (*ID*), este projeto conta com três tipos de pacotes API*:*

1) *Transmit Request* (0x10);
2) *Receive Packet* (0x90);
3) *Transmit status* (0x8b);

Ainda tomando como base o exemplo acima um pacote (*ID*=0) do tipo *transmit request* é enviado ao módulo *XBee* motor_direito, um pacote de resposta (ID=1) do tipo *transmit status* é recebido automaticamente para informar se o módulo remoto recebeu com sucesso ou não o pacote enviado, logo em seguida o módulo remoto motor_direito envia outro pacote do tipo *receive Packet* (*ID=2*) contendo seu endereço e as informações sobre as leituras dos sensores. O pacote de ID=3 é enviado ao módulo motor_esquerdo e o mesmo processo acontece.

Pacote *transmit request* (ID = 0) do exemplo:

Pacote*: 7E 00 10 10 01 00 7D 33 A2 00 40 98 28 D0 FF FE 00 00 02 0F 5B*

Start delimiter: 7E

Length: 00 10 (16)

Frame type: 10 (Transmit Request)

Frame ID: 01 (1)

64-bit dest. address: 00 13 A2 00 40 98 28 D0

16-bit dest. address: FF FE

Broadcast radius: 00 (0)

Options: 00

RF data: 02 0F (dados enviados)

Checksum: 5B

Pacote *Transmit Status* (ID = 1) do exemplo:

Pacote: 7E 00 07 8B 01 EB 8B 1C 00 40 A1

Start delimiter: 7E

Length: 00 07 (7)

Frame type: 8B (Transmit Status)

Frame ID: 01 (1)

16-bit dest. address: EB 8B

Tx. retry count: 1C (28)

Delivery status: 00 (Success)

Discovery status: 40 (Extended timeout discovery)

Checksum: A1

Pacote *receive packet* (ID = 2) do exemplo:

Pacote:7E 00 16 90 00 7D 33 A2 00 40 98 28 D0 EB 8B 41 02 25 01 B0 FF E1 02 45 01 C7 6C

Start delimiter: 7E

Length: 00 16 (22)

Frame type: 90 (Receive Packet)

64-bit source address: 00 13 A2 00 40 98 28 D0

16-bit source address: EB 8B

Receive options: 41

RF data: 02 25 01 B0 FF E1 02 45 01 C7 (dados recebidos das leituras dos sensores)

Checksum: 6C

Como visto, com o *software XCTU* podemos enviar e receber pacotes e ajustar a velocidade de cada motor enviando os valores de velocidade desejado, mas fazer o controle de velocidade dos motores de forma que este atinja toda gama de velocidade se torna uma tarefa difícil e impraticável somente com este programa, é preciso desenvolver alguma forma de comandar os motores da embarcação e esta ideia será um dos objetivos do Capítulo 6.

6. Sistema desenvolvido

O sistema desenvolvido neste projeto de graduação tomou como base a divisão em dois eixos, *hardware* e *software* que serão explicados em detalhes ao longo deste Capítulo.

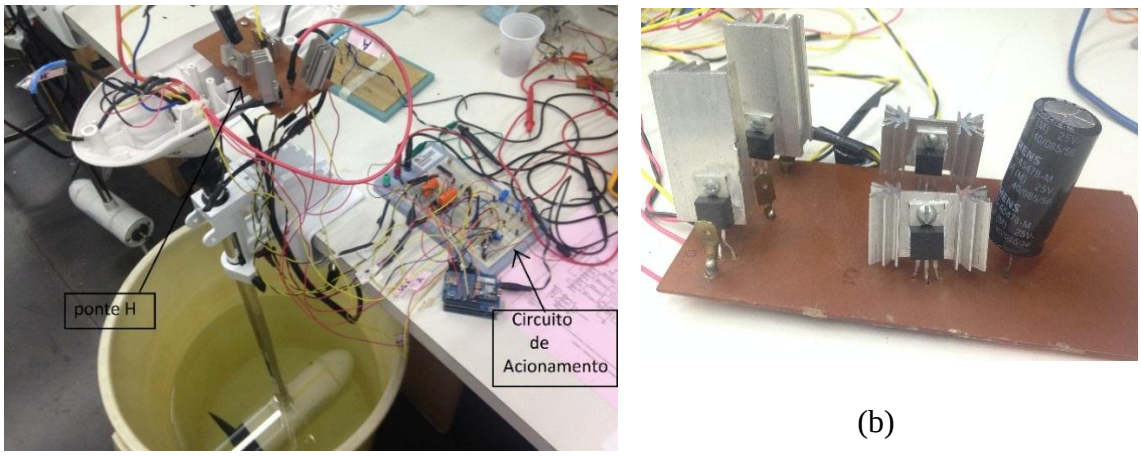
6.1. *Hardware*

6.1.1. Protótipo

A Figura 49 mostra o protótipo dos circuitos de acionamento e ponte H desenvolvidos, as descrições das partes integrantes desse circuito, além das justificativas das escolhas de cada componente foram descritas no Capítulo 3. O protótipo foi montado tomando como base o *application note* do circuito integrado HIP4081, e alguns componentes tiveram seus valores finais escolhidos com base em tentativas de acerto e erro, visto que valores calculados nem sempre refletem as condições reais de todas as variáveis do sistema e tornam-se críticos mesmo para pequenos desvios, esses componentes foram:

- ✓ Resistores de *Gate*;
- ✓ Capacitor de *Bootstraping*;
- ✓ Diodo de *Bootstraping*;
- ✓ Os resistores HDEL e LDEL (ajustam o *dead-time)*;

Seus primeiros valores testados causaram a queima de alguns *MOSFETs*, principalmente os resistores que excitam os *gates*. Seus valores finais foram escolhidos por oferecerem melhor desempenho e podem ser consultados no diagrama do projeto em anexo no final desse projeto de graduação.



(a) (b)

Figura 49: Protótipo desenvolvido: (a) circuito de acionamento e sensores; (b) ponte H.

No mesmo *protoboard* onde está o circuito de acionamento também foi montado o circuito dos sensores, com exceção dos sensores de corrente que foram montados juntos a placa da ponte H, já que estes medem as correntes que entram no motor e a corrente que sai da bateria, os detalhes sobre os sensores foram descritos no Capítulo 2, nesta etapa os dados destes sensores foram lidos através da tela do monitor serial do Arduino que apresentou leituras coerentes das variáveis medidas.

Embora tenha bastante fios (sujeitos a ruídos) e a conexões dos componentes feita por meio de um *protoboard*, este protótipo apresentou funcionamento razoável que motivou e justificou a montagem de uma placa de circuito impresso.

6.1.2. Circuito para acionamento dos motores

Com base no protótipo do circuito de acionamento e ponte H como visto anteriormente, foi desenvolvida por empresa de prototipagem eletrônica uma placa de circuito impresso profissional de dupla camada em fibra de vidro para realizar o acionamento dos motores, esta placa pode ser vista na figura 50.

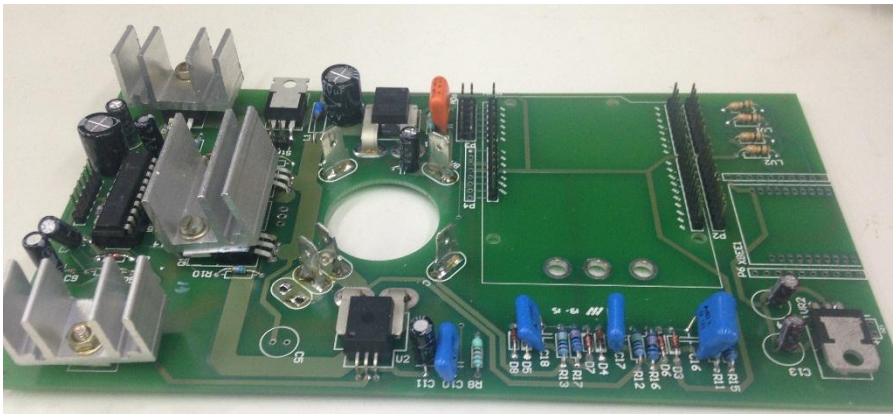


Figura 50: placa de circuito impresso.

A figura a seguir mostra as principais partes da placa e um destaque com o Arduino e o módulo *XBee* anexados a placa.

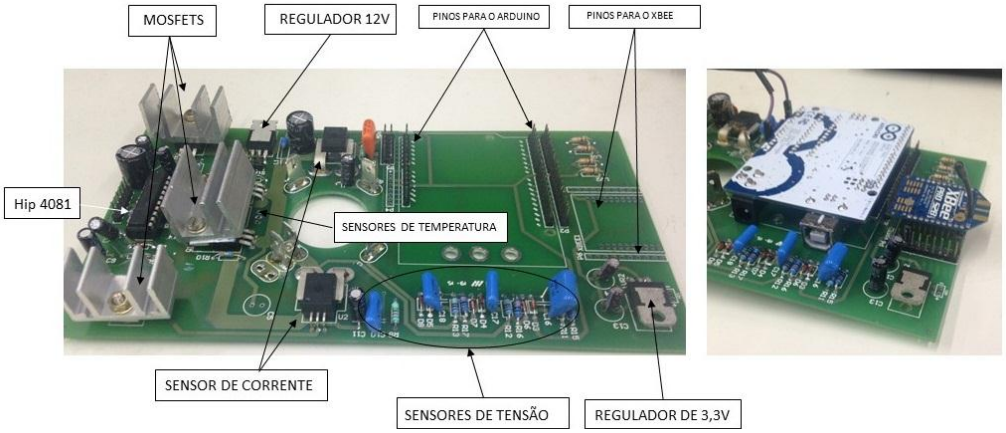


Figura 51: (a) Descrição das partes da placa; (b) Arduino e módulo *XBee* anexados.

É importante observar que para alimentação de todos os sensores foi utilizado a saída regulada de 5V que Arduino oferece em um de seus pinos, isso facilitou a montagem da placa além de torna-la mais barata, embora o Arduino também ofereça uma linha de 3,3V preferiu-se alimentar o módulo *XBee* através de um regulador de 3,3V exclusivo, para não sobrecarregar o regulador do Arduino, o regulador de 3,3V também oferece alimentação para um dos lados de um conversor de nível lógico necessário para "interfacear" o Arduino e o módulo *XBee* já que estes trabalham com nível lógico diferente.

A alimentação do *chip HIP*4081, assim como a alimentação do Arduino é fornecida pela bateria através de um regulador de 12V (figura 52), observamos um conector (*strap)* que pode colocar em curto-circuito os pinos V_{IN} e V_{OUT} do regulador de 12V, essa ferramenta torna-se útil quando o circuito está ligado com apenas uma bateria de 12V, neste cenário temos que desabilitar o regulador pois acionado ele mantém a tensão em sua saída com um valor abaixo de 12V, e isso impede o bom funcionamento do *chip HIP*4081, para o cenário em que mais de uma bateria é conectada para fornecer ao sistema tensões de 24V ou 36V, por exemplo, o *strap* é removido isto será visto com mais detalhes à frente no item 6.2.1.

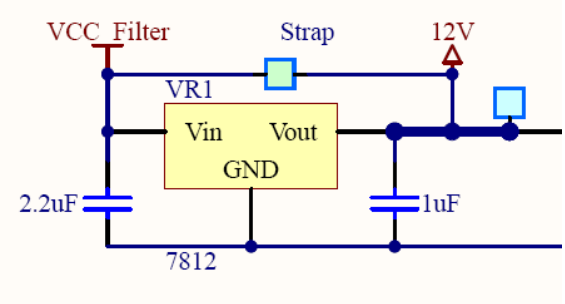


Figura 52: Detalhe do *strap* de *by-pass* do regulador de 12V.

6.1.2.1 Correções de problemas do circuito de acionamento

Ao ser colocada em funcionamento a placa do circuito de acionamento apresentou alguns problemas que serão relatados:

✓ Sensores

Com a placa ligada, mas com os *MOSFETs* chaveadores da ponte H desligados a leitura do sensor de temperatura e dos sensores de tensão apresentavam valores corretos, porém ao aplicar o menor sinal PWM nos *MOSFETs*, todas as leituras de tensão, temperatura e corrente ficavam inconsistentes deixando claro que a ponte H ao ser ligada estaria causando interferência nas leituras dos sensores. Após uma longa investigação foi detectado que o problema estava na interconexão das referências de terras, terra de instrumentação dos sensores com o terra da parte de potência da ponte H, a solução mais imediata e elegante encontrada foi desconecta-los, e isso foi feito e pode ser visto a seguir, observe que a trilha foi interrompida.

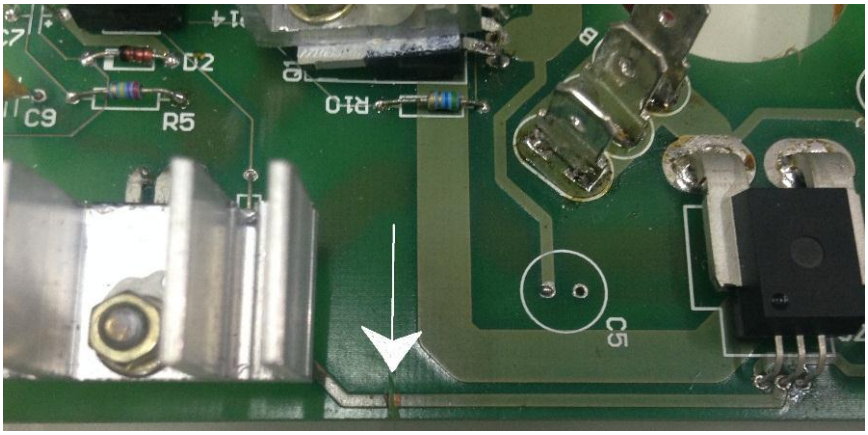


Figura 53: Separação das referências de terra.

Após isso o problema foi resolvido interligando o referêncial de terra da parte de instrumentação em apenas um ponto, vale destacar que esse problema não foi apresentado no protótipo pois a ponte H e o circuito dos sensores estão fisicamente separados figura 49 (a).

✓ Temperatura das trilhas

Outro problema encontrado foi o aquecimento demasiado de algumas trilhas que interligam os componentes da ponte H e nos terminais de conexão da bateria, A análise de temperatura das trilhas foram feitas com um monitor termográfico (FLIR TG165, 2014). Para este teste foi aplicado sinal PWM com uma taxa de trabalho de 70% (*duty cycle*) e as temperaturas em pontos específicos foram sendo analisadas constantemente num intervalo de cinco minutos para frente, e três minutos para ré, momento em que a placa começou a apresentar cheiro de queimado forçando seu desligamento. A maior temperatura registrada é mostrada abaixo e uma outra figura que serve como referencial mostrando o mesmo trecho da placa analisado.

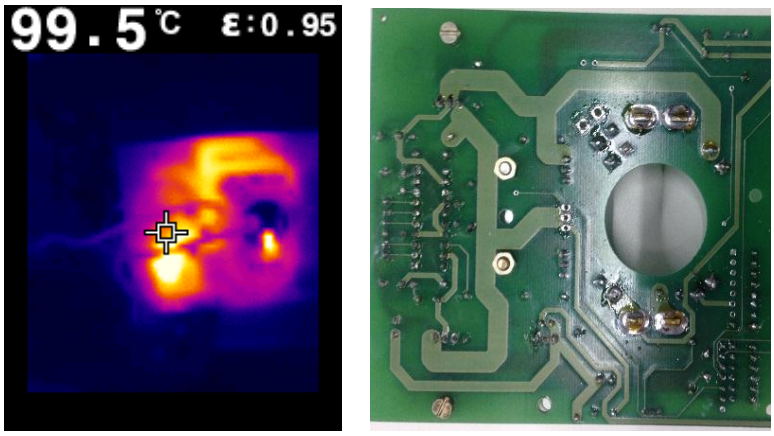


Figura 54: Maior temperatura para ré PWM de 70% em 3 minutos.

A solução encontrada foi colocar fio de cobre de 4mm² de seção reta em paralelo com alguns trechos das trilhas, como mostrado na figura abaixo.

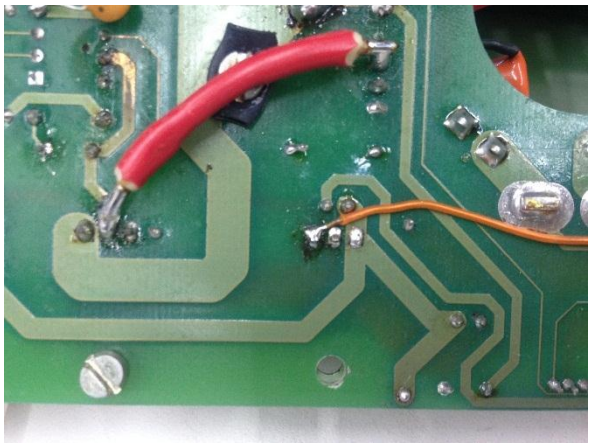


Figura 55: Fio em paralelo com a trilha.

✓ Troca dos capacitores de *Bootstraping*

Os capacitores de *bootstrapping* da placa inicialmente eram capacitores eletrolíticos, estes capacitores não são apropriados para aplicações cuja frequência esteja em algumas dezenas de kHz que é o caso já que a frequência do PWM do sistema é de 31kHz, foi observado que a capacitância diminui consideravelmente com o aumento da frequência, foi observado também que este efeito se potencializa em capacitores de qualidade ruim que estão disponíveis no comercio eletrônico atualmente. Os capacitores de *bootstrapping* são componentes críticos e variações em seus valores nominais implicam em variação no desempenho dos *MOSFETs* superiores já que esses capacitores fornecem momentaneamente tensão para a correta polarização dos mesmos, uma solução encontrada para contornar esse problema foi a substituição por capacitores de tântalo, cuja curva de resposta de frequência é plana para a frequência requerida, além de serem fisicamente muito menores para a mesma capacitância. A figura abaixo mostra a variação da capacitância em relação ao aumento da frequência para o capacitor eletrolítico em questão, para medição foi usado um *LCR meter* que o laboratório *LEPAT-UERJ* disponibiliza.

Figura 56: testes dos capacitores eletrolíticos.

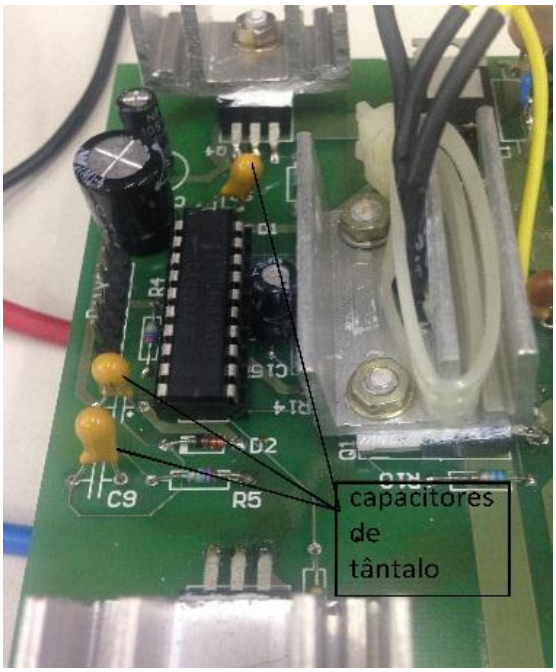


Figura 57: capacitores substituídos.

✓ Banco de capacitores

A fim de evitar que correntes alternadas fiquem transitando na ponte H dissipando energia em forma de calor, foi implementado um banco de capacitor simples para prover um filtro para estas correntes. Para implementar o banco de capacitores foi utilizado 6 capacitores eletrolíticos de 4700µf x 35V em paralelo, foi adicionado terminais de encaixe para sua fácil desconexão ao circuito.

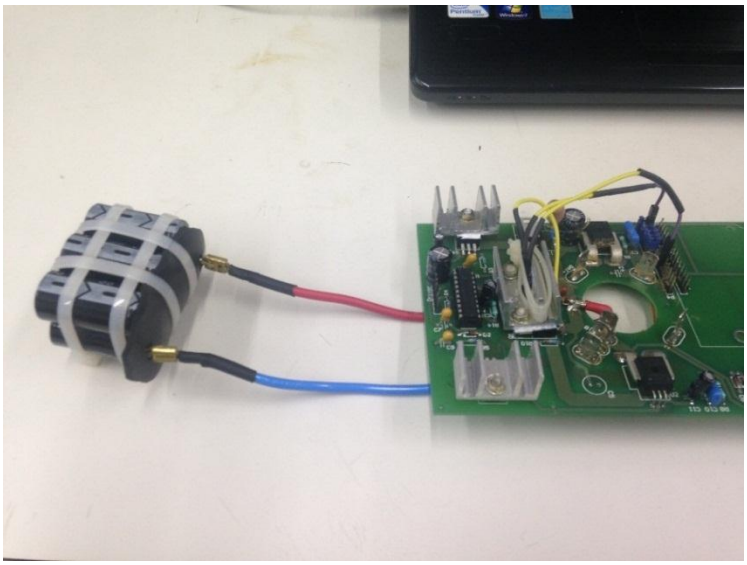


Figura 58: Banco de capacitores.

6.1.3. Controle remoto sem fio *ZigBee*

Para facilitar o controle dos propulsores da embarcação foi desenvolvido um controle remoto com as seguintes funções:

- ✓ Iniciar e coordenar a rede *ZigBee*;
- ✓ Enviar os comandos de velocidades e sentido da embarcação;
- ✓ Efetuar as leituras dos sensores;

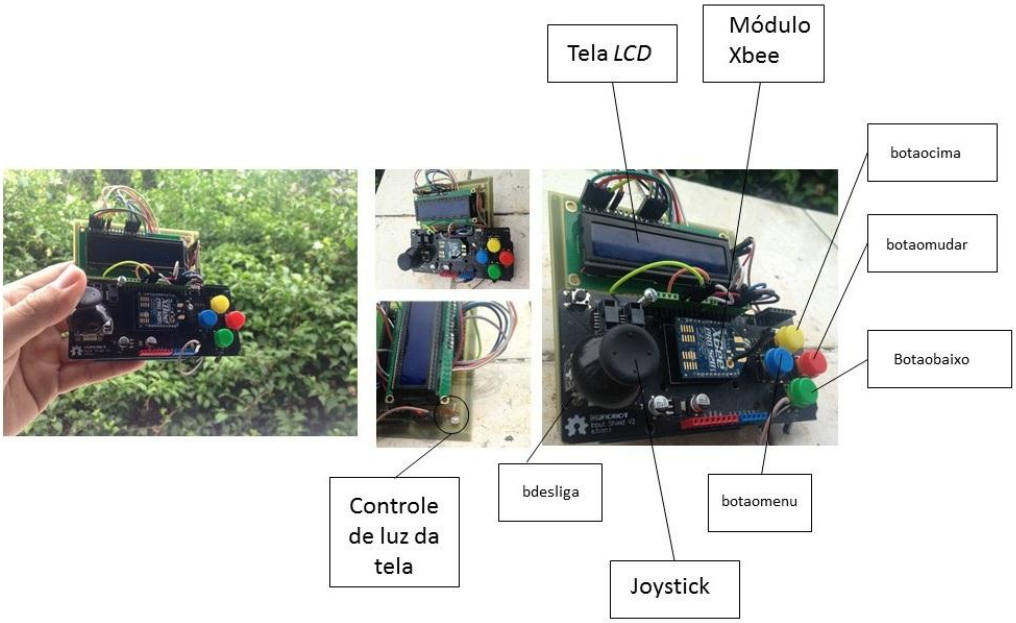


Figura 59: Controle remoto *wireless ZigBee*.

6.1.3.1. Partes integrantes do controle remoto sem fio *ZigBee*

O controle remoto é composto pelas seguintes partes, Arduino uno, *shield joystick* para Arduino, tela de cristal líquido de duas linhas por dezesseis colunas (16x2), módulo *XBee*, circuito para controle automático da luz de fundo, placa de fenolite (base).

- Arduino *joystick shield*

Esse *shield* produzido pela empresa *DFROBOT* foi escolhido para o projeto por oferecer um controle de direção intuitivo com o *joystick* de dois eixos, porém o grande diferencial desse *shield* é que ele oferece um conector apropriado para adição de um módulo *XBee* para aplicações com comunicação sem fio. Além do *joystick* de dois eixos que é implementado com potenciômetros de precisão, este *shield* conta também com mais cinco botões:

- ✓ Botão *UP* (amarelo) - ligado à porta digital 8 do arduino.
- ✓ Botão *DOWN* (verde) – ligado a porta digital/analógica A1.
- ✓ Botão *LEFT* (azul) – ligado a porta digital 9.
- ✓ Botão *RIGHT* (vermelho) – ligado a porta digital 12.

✓ Botão eixo Z (acionado pressionando o joystick) - ligado a porta A0

Neste projeto os botões serão identificados como botaocima, botaobaixo, botaomenu, botaomudar, bdesliga respectivamente. O *shield* conta também com um botão de *reset* para reiniciar o Arduino.

O eixo X do *joystick* é lido pela porta analógica A2, e o eixo Y é lido pela porta analógica A3 do Arduino, para o eixo Z tem botão que pode ser acionado pressionando o *joystick* este é lido pela porta analógica A0. Esse *shield* tem uma chave de duas posições (*PROG* ou *RUN*) para multiplexar o uso da porta serial do microcontrolador. A posição *prog* deve ser usada para carregar os *sketches* para o Arduino, por exemplo, e a posição *run* para fazer a comunicação com o *XBee*.

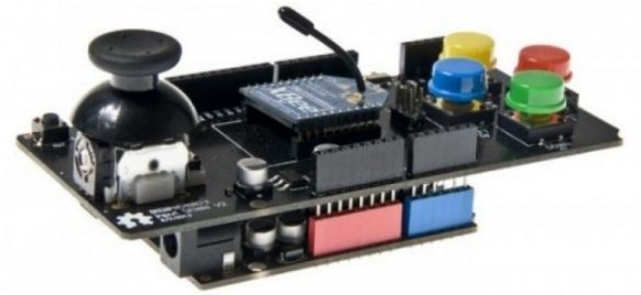


Figura 60: *Shield joystick* com Arduino.

- Display de cristal líquido (*LCD*)

É necessário que se tenha alguma forma de realizar as leituras das variáveis medidas pelos sensores, além de poder monitorar a velocidade e sentido de deslocamento da embarcação para isto foi utilizado um *display* de cristal líquido que realiza essa tarefa de forma simples.

Figura 61: display LCD, fonte (https://www.baudaeletronica.com.br/acessorios/lcds/display-lcd-16x2-azul.html).

Este display *LCD* tem um controlador integrado da *Hitachi* o HD44780 (Hitachi, 1998), é formado por 32 células onde são escritos os caracteres, cada célula é composta por uma matriz de 8x5 pixels. Existem diversos fabricantes, porém a grande maioria segue um padrão quanto à disposição dos pinos e suas funções sendo 14 pinos quando o *display* não tem a função de luz de fundo (*backlight*) e 16 pinos quando tem.

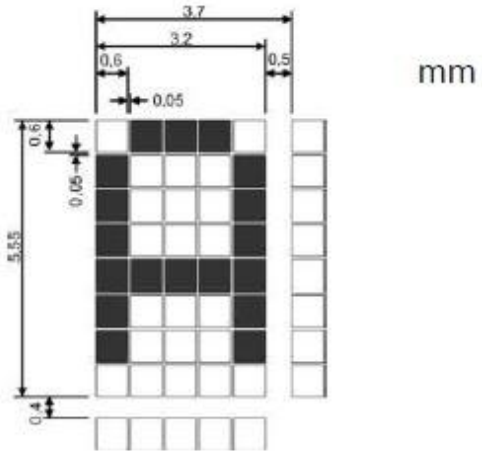


Figura 62: Matriz de pixels de *display LCD*.(BASTOS, 2012, p. 10).

Tabela 15: Pinos de um display LCD 16x2, (SPARKFUN, 2008, p. 5).

Nº pino	**Símbolo**	**Função**
1	VSS	Aterramento
2	VDD	Alimentação
3	V0	Ajuste de contraste
4	RS	Sinal de seleção do registrador
5	R/W	Sinal de seleção de escrita ou leitura
6	E	Sinal habilitador (Enable)
7~10	DB0~DB3	Dados de escrita ou leitura menos significativos
11~14	DB4~DB7	Dados de escrita ou leitura mais significativos
15	LED+ ou A (Anodo)	Alimentação para a luz de fundo (backlight)
16	LED- K (catodo)	Terra para a luz de fundo

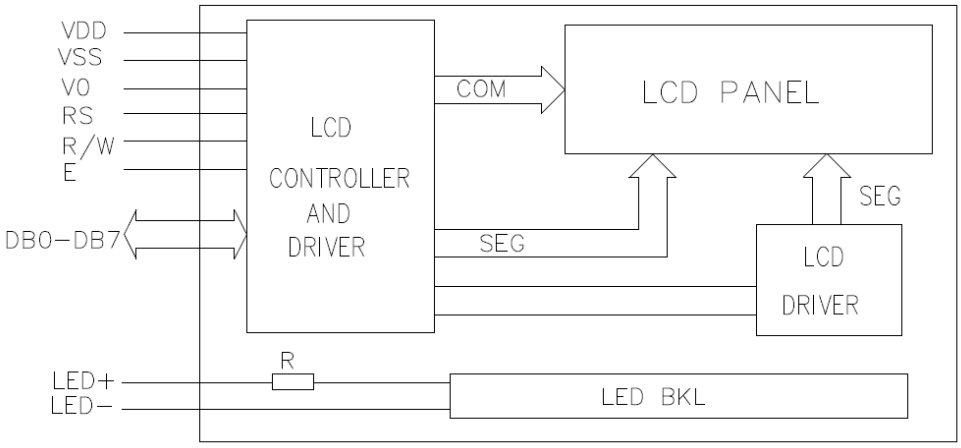


Figura 63: Diagrama em blocos do display, fonte (SPARKFUN, 2008, p. 5)

Existem duas formas possíveis de implementar comunicação usando esse tipo de *display LCD*

1) Enviar um *byte* por vez com a configuração do que deverá ser escrito;
2) Enviar um dois *nibbles* de 4 *bits* com a configuração do que será escrito.

A segunda opção foi criada para usar menos portas dos microcontroladores, nos dois casos a comunicação é realizada inserindo os dados nos pinos (DB0 a DB7 com byte e DB4 a DB7 com 2 *nibbles*), o RS e o R/W com 0 e 1 e informando ao controlador que deve ser feita uma operação de leitura (BASTOS, 2012, p. 18). Este informe é feito elevando o nível do pino de *enable* de 0 para 1 e retornando-o para 0.

Para comunicar o Arduino com o *display* foi utilizada uma biblioteca chamada *liquid Crystal* que utiliza o formato de envio de 2 *nibbles* para comunicação, esta biblioteca vem pré-instalada no *software* Arduino (*IDE* Arduino).

Normalmente no pino V_0 (pino 3) usa-se um potenciômetro para ajuste do contraste, neste projeto foi utilizado uma saída analógica do Arduino que insere um sinal PWM neste pino e faz o ajuste de contraste aumentando ou diminuindo seu valor através das teclas do controle de acordo com a necessidade do operador.

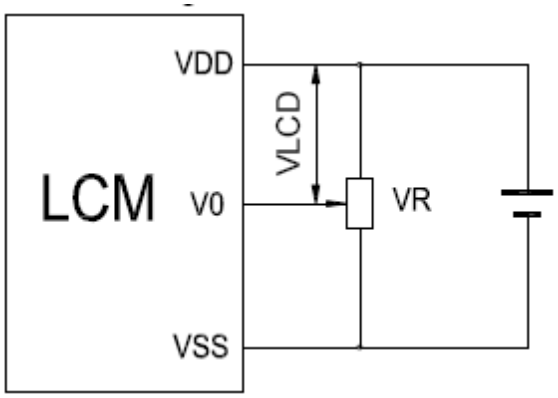


Figura 64: Controle nível de contraste com potenciômetro (SPARKFUN, 2008, p. 6)

- Circuito para controle automático da luz de fundo

A luz de fundo do *display* pode ser ajustada pelo operador de três formas diferentes:

✓ Desligada;
✓ Ligada;
✓ Automática;

No modo ligado o ajuste é feito de forma que o *display* apresente o máximo de luminosidade permanentemente ligada, e no modo desligado o mínimo de luminosidade ou seja a luz é desligada.

No modo automático é utilizado um sensor de luminosidade, para ajustar a intensidade da luz de fundo automaticamente de acordo com o nível de luminosidade ambiente (LDR - *Light Dependent Resistor*), esta funcionalidade é útil pois quando os motores são operados em ambientes com alta a ou baixa luminosidade a luz tela se adequa oferecendo melhor visualização do conteúdo apresentado, além da economia de energia pois o controle é alimentado por bateria.

Figura 65: (a) LDR (b) Função de luz *display*.

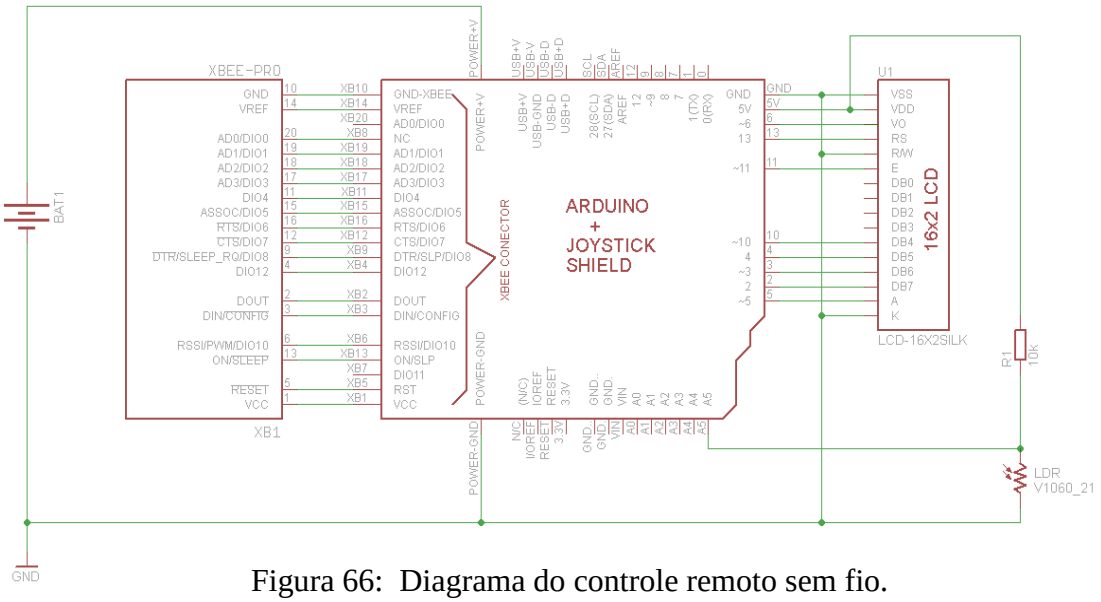


Figura 66: Diagrama do controle remoto sem fio.

6.2. *Software*

Os *softwares* foram desenvolvidos na plataforma Arduino e foram divididos em duas partes: *software* da placa de acionamento dos motores e *software* do controle remoto *sem fio ZigBee*.

6.2.1. *Software* da placa de acionamento dos motores.

Este *software* é executado pelos Arduinos que controlam os motores da embarcação, têm a função de receber e validar os pacotes *ZigBee* com informações para o acionamento dos motores, além de aquisitar e enviar as leituras dos sensores ao coordenador da rede *ZigBee*. Serão apresentados trechos do *software* e suas principais funções de acordo com o diagrama do *software* apresentado na figura 67.

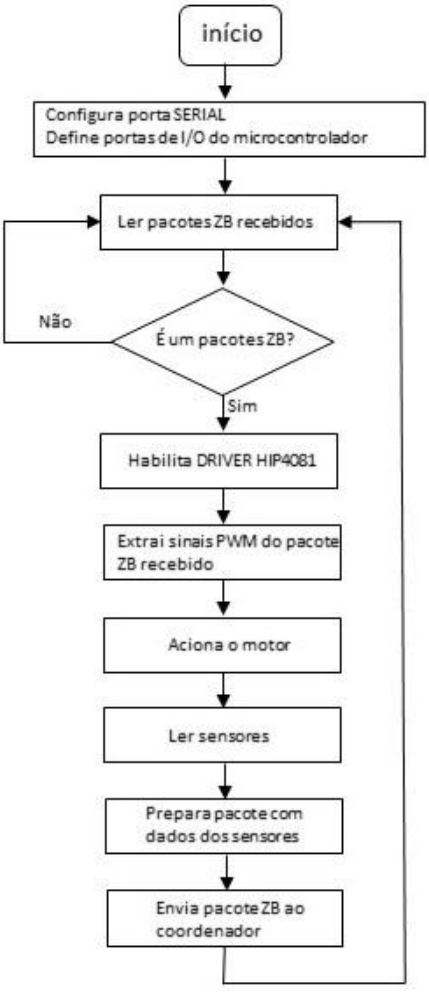


Figura 67: Arquitetura do software.

A comunicação no modo API entre módulos *XBee* não é uma tarefa muito trivial, visto que a comunicação é estruturada por meio de pacotes pré-estabelecidos com diversos campos e cálculos como foi apresentado no Capítulo 5, para facilitar o desenvolvimento deste projeto foi utilizado uma biblioteca chamada Xbee.h, o ganho de tempo com o uso desta biblioteca foi decisivo para a sua conclusão. A biblioteca pode ser encontrada no site do desenvolvedor https://github.com/andrewrapp/xbee-arduino.

- **Configurar comunicação serial**

A programação em plataforma Arduino é extremamente facilitada e a maioria das funções necessária para ajuste e funcionamento de um *software* são implementadas em apenas uma linha de comando isto é visto na função para configurar a comunicação com a porta serial, o número entre parentes representa a taxa de transmissão de dados nesse caso 9600 bps. Também é necessário iniciar a comunicação serial do módulo *XBee*, estando estes códigos dentro da função *setup* do Arduino.

```
Serial.begin(9600);
xbee.begin(Serial);
```

Nesta etapa também faz a configuração dos modos de como pinos do Arduino irão trabalhar se são entradas ou saídas.

```
pinMode(DIS, OUTPUT);
pinMode(statusLed, OUTPUT);
pinMode(errorLed, OUTPUT);
pinMode(ledcalibragem,OUTPUT);
 //declaração dos pinos digitais
pinMode(frente, OUTPUT);
pinMode(re, OUTPUT);
pinMode(BHI, OUTPUT);
pinMode(AHI, OUTPUT);
```

- **Frequência PWM**

Como visto no Capítulo 4 o Arduino UNO tem 6 pinos que podem ser usados como saídas PWM, essas saídas têm frequência pré-estabelecidas mas podem ser alteradas para valores definidos pelo fabricante do microcontrolador, a frequências das saídas são controladas por três contadores TCCR0B, TCCR1B e TCCR2B estes contadores controlam os pinos;

✓ Pinos 3 e 11 - controlados pelo contador TCCR2B tem as seguintes frequências 31KHz, 3.9KHz, 976Hz, 488Hz (padrão), 244Hz, 122Hz,30Hz.
✓ Pinos 5 e 6 - controlados pelo contador TCCR0B tem as seguintes frequências 62,5KHz, 7.8KHz, 976Hz (padrão), 488Hz, 244Hz, 61Hz.
✓ Pinos 9 e 10 - controlados pelo contador TCCR1B tem as seguintes frequências 31KHz, 3.9KHz, 976Hz, 488Hz (padrão), 30Hz.

Em testes inicias o acionamento dos motores mostrou-se extremamente ruidoso e foi percebida uma vibração além da normalidade na ponte H e isto foi atribuído a baixa frequência de operação das saídas PWM utilizadas nos pinos 5 e 6 do Arduino, a seguir foi feito um estudo comparativo se em algum valor de frequência e largura de pulso (*duty cycle*) nas saídas, causariam uma diferença acentuada na perda de energia e por consequência aumento na temperatura de operação dos *MOSFETs*, porém nada significativo foi percebido neste sentido, em relação ao ruído e vibração o circuito apresentou uma melhora significativa na frequência de 31kHz (faixa não audível), também ficou decidido mudar o par de pinos que fazem as saídas PWM inicialmente arbitradas nos pinos 5 e 6 para os pinos 9 e 10, o motivo é que o *site* oficial do Arduino recomenda que não seja alterada a frequência padrão do temporizador TCCR0B

que controla os pinos 5 e 6, pois funções do Arduino relacionados a temporização e *delay* podem apresentar falhas. O trecho de programa altera a frequência PWM dos pinos 9 e 10 para 31kHz.

```
TCCR1B = TCCR1B & B11111000 | B00000001;
```

- **Ler pacotes *ZigBee***

Esta função é executada dentro da função *loop*, ou seja, é executada continuamente e tem como finalidade ler possíveis pacotes *ZigBee* a cada *loop* do programa principal, recebendo um pacote *ZigBee* a função aguarda 500ms.

```
(xbee.readPacket(500))
```

- **Habilitar *DRIVER HIP*4081**

Como foi visto no Capítulo 3, o circuito integrado HIP4081 tem um pino chamado *DIS* (*disable*) que funciona como habilitador ou desabilitado das saídas PWM da ponte H, este pino é colocado em nível lógico alto através do pino 7 do Arduino o que lhe confere a condição de desabilitado enquanto o Arduino executa a função *setup*, entretanto após executar as rotinas de inicialização e se há um comando recebido através de um pacote de dados a ser executado, o Arduino precisa habilitar as saídas da ponte H e isto é feito colocando nível lógico baixo neste pino, além de habilitar as saídas *AHI* e *BHI* que controlam os *MOSFETs* superiores da ponte H.

```
digitalWrite(DIS,LOW);// Habilita o MOSFET driver HIP4081
digitalWrite(AHI,HIGH);// habilita AHI
digitalWrite(BHI,HIGH);// habilita BHI
```

- **Leitura dos pacotes recebido**

Após serem recebidos e validados os pacotes *ZigBee*, a informação contida nestes precisam ser extraídas isto é feito com a linha seguinte código.

```
y1 =controle_tensao( rx.getData(0));
y1=~y1;
y2=controle_tensao( rx.getData(1));
y2=~y2;
```

A função rx.getData(0) extrai o *byte* de dados menos significativo e a função rx.getData(1) o segundo byte menos significativo, estes dados são tratados pela função controle_tensão e armazenados nas variáveis y1 e y2 respectivamente.

A função *controle_tensão* foi desenvolvida para que o sistema possa operar com mais de uma bateria de 12 volts, ou com um nível de tensão mais elevado em sua alimentação, porém nesta condição o Arduino não pode produzir ciclos de trabalho (*duty cycle*) cujo valor médio da tensão de saída seja maior que 12V que é a tensão nominal do motor senão este queimaria, para isto a função pega a leitura de tensão da bateria feita pelo Arduino e a usa para ponderar a largura de pulso PWM gerada, por exemplo, se o motor estiver sendo alimentado por duas baterias de 12 volts de forma que estas forneçam juntas 24 volts, a largura do pulso PWM gerada será no máximo 50% que entrega 12 volts ao motor.

```
int controle_tensao(int b)
{
float fator
=(tensao_bat_1*constante_tensao_bateria/tensao_nominal_motor);
if (fator > 1.0)
{
b = b*(1.0/fator);
}
return b;
}
```

A função *controle_tensão* recebe um parâmetro de entrada o inteiro b (*byte* recebido), e retorna o mesmo parâmetro b depois de ser feito a ponderação mencionada acima.

Após isso a variável y1 recebe o *byte* referente ao valor do pulso PWM para a frente, a variável y2 recebe o byte referente ao valor do pulso PWM para ré, estes bytes precisam ser invertidos porque o circuito HIP 4081 tem uma porta inversora em cada uma das entradas referentes a esses pinos que recebem estes sinais (ALI, BLI), sem a inversão o acionamento ocorre, porém, uma das consequências observado é que na condição de motor parado ao invés de ter 0 volts em cada braço da ponte H (tensão aplicada no motor) têm-se 12 volts.

- **Acionamento dos motores**

O acionamento dos motores é feito simplesmente excitando os *MOSFETs* com o valor do pulso PWM que foi recebido pelo módulo *XBee* remoto. Isto é realizado pela função *analogWrite* do Arduino que tem dois parâmetros, o número da porta de saída PWM e o valor do pulso contido nos *bytes*, foi usado um pino de saída PWM do Arduino para cada sentido de deslocamento (frente = pino 9 e ré = pino 10) implementados pelos seguintes trechos do programa:

✓ Frente;

```
analogWrite(frente, y1);
```

✓ Ré;

```
analogWrite(re,y2);
```

- **Limitador de corrente no motor**

Para evitar efeitos de altas correntes por tempo prolongado, foi implementado um limitador de corrente que limita em 30A a corrente máxima do motor. A ação deste limitador esta sintetizada nas figuras abaixo para alguns valores de correntes.

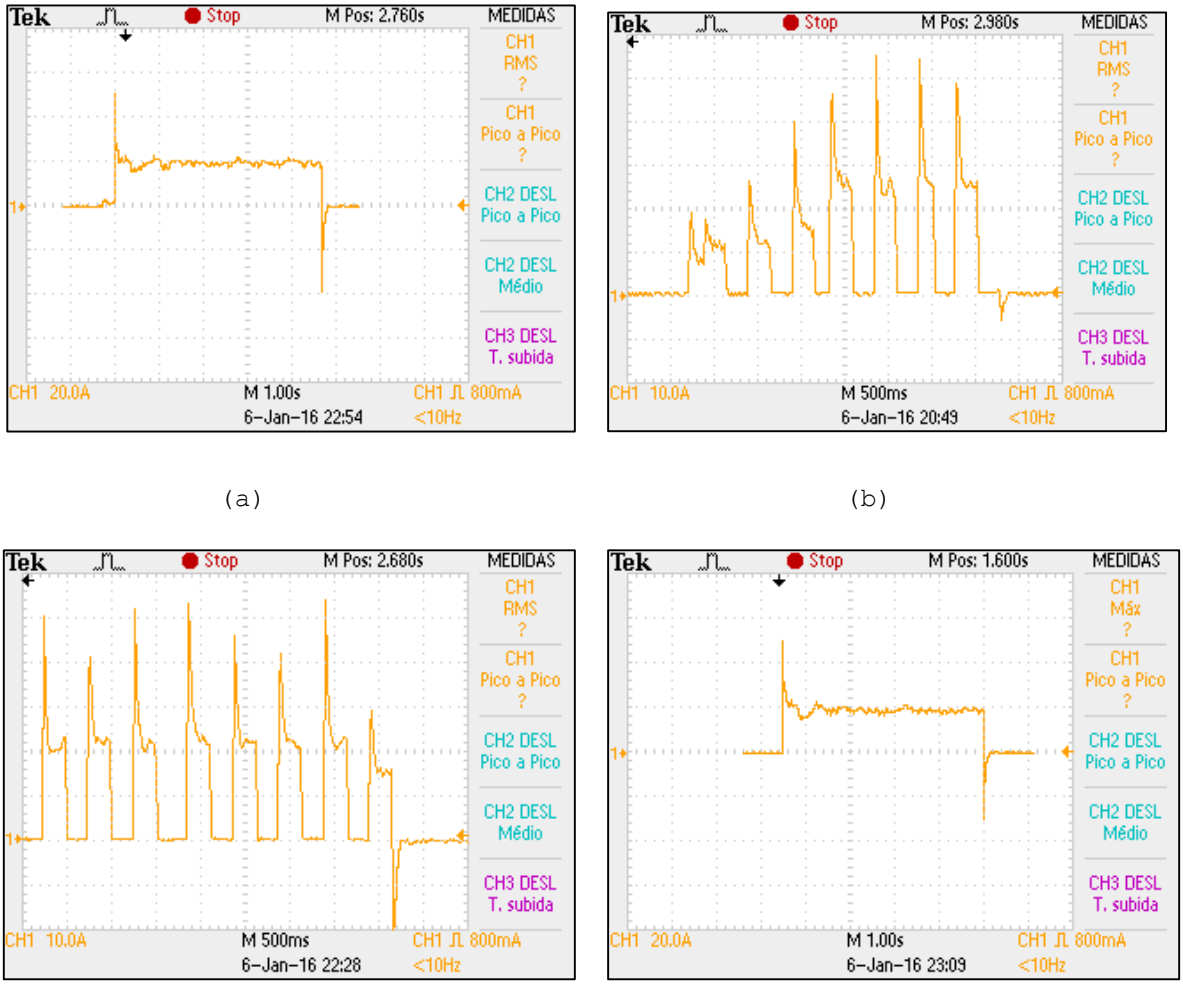


Figura 68: Limitador de corrente no motor com limites em: (a) sem limite, (b) 10A, (c) 20A, (d) 30A.

- **Aquisição dos dados dos sensores**

A aquisição dos dados lido por cada sensor é realizada com uma leitura analógica do Arduino e foi explicada no Capítulo 2.

O conversor analógico-digital do Arduino tem uma resolução de 10 bits isso quer dizer que os dados lidos pelos sensores são compostos por 10 *bits* de informação, no entanto, os pacotes *ZigBee* só podem ser compostos por *bytes* (8 bits), para resolver isso de forma que não haja perda de informação tendo que suprimir 2 bits, os 10 bits das leituras foram divididos em dois *bytes*, por exemplo digamos que seja preciso enviar o valor da leitura do sensor temperatura mostrado abaixo.

Exemplo de leitura analógica dos sensores

1	1	0	1	1	0	0	1	1	0

10 bits a serem enviados

Figura 69: Valor genérico lido pelo sensor de temperatura.

Para isso, foram criados os seguintes trechos;

```
temp = analogRead(A4);
payload[0] = temp >> 8 & 0xff;
payload[1] = temp & 0xff;
```

Esses trechos são executados pelo programa, e inicialmente gera o byte [0] (*payload* [0]) mais significativo, realiza um deslocamento de 8 *bits* a direita dos 10 *bits* iniciais e em seguida a operação lógica *booleana AND* com o valor hexadecimal 0xFF (máscara).

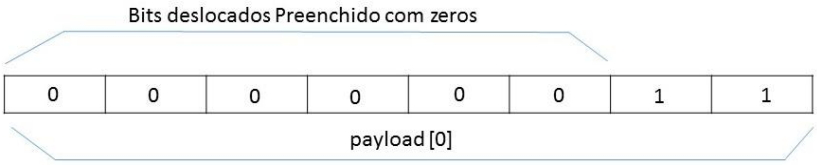


Figura 70: *Payload* [0].

Para o *byte* [1] apenas a máscara é necessária, a fim de extrair os 8 *bits* e gerar o payload [1].

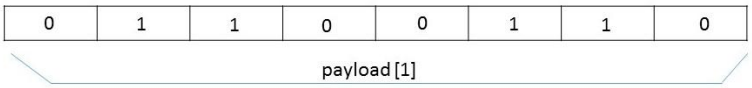


Figura 71: *Payload* [1].

- **Envio do pacote *ZigBee***

Após ser construído o pacote deve ser enviado ao coordenador da rede *ZigBee*, e isto é feito pelo seguinte trecho no programa.

```
xbee.send(zbTx);
```

6.2.2. *Software* do controle remoto sem fio *ZigBee*.

A arquitetura principal do *software* do controle remoto sem fio foi baseada em uma máquina de estados, seu funcionamento foi divido em um menu, onde cada etapa desse menu representa um estado da máquina de estados. Cada estado atual executa uma série de tarefas e apresenta informações referentes a esse estado de acordo com as entradas recebidas.

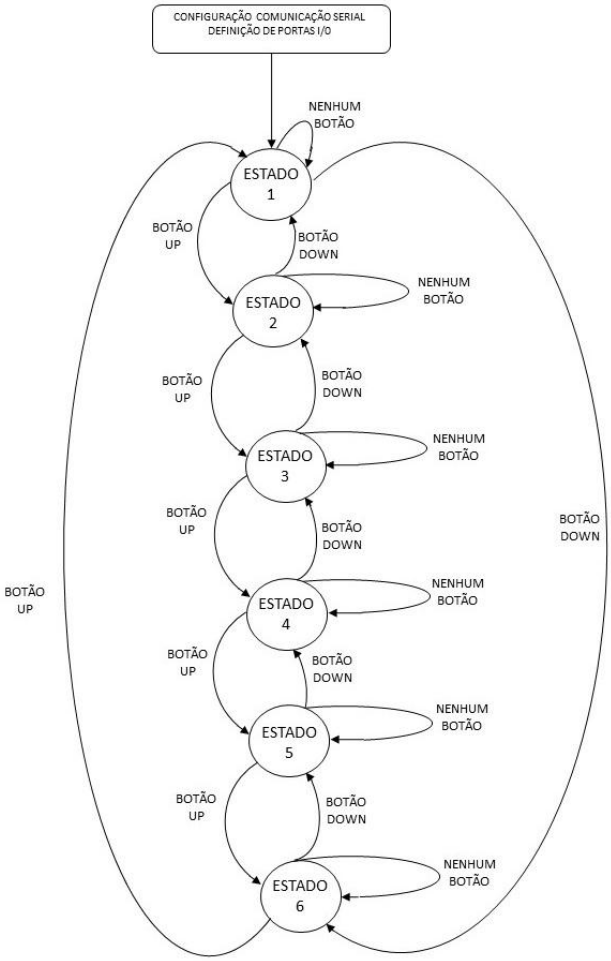


Figura 72: Software do controle remoto sem fio.

- **Inicialização do *software***

A inicialização da comunicação serial é similar a explicada no tópico anterior com a diferença que foi utilizada a biblioteca Arduino *liquid crystal*, para facilitar a implementação de escrita no *display* de cristal líquido (*LCD*), a biblioteca é incluída ao arquivo com o seguinte trecho no topo do programa.

```
#include <LiquidCrystal.h>
```

E a declaração dos pinos que o Arduino deverá ligar no *display*, declarado também no topo do programa:

```
LiquidCrystal lcd(13, 11, 10, 4, 3, 2);
```

Após isso o seguinte código é inserido para criar um objeto do tipo *LCD* de 2 linhas e 16 colunas, esse trecho é inserido dentro da função *setup*.

```
lcd.begin(16, 2);
```

Alguns outros ajustes são feitos antes da função *setup* ser executada, como definição dos pinos, definição dos endereços dos módulos remotos *XBee*, inclusão de bibliotecas e inicialização de variáveis.

Quando o programa executa a função *setup* ele cria alguns caracteres que serão utilizados nos menus, aciona os *pull up* internos dos pinos digitais do Arduino que irão ler os comandos do *display*, liga a luz de fundo, ajusta o nível de contraste e exibe uma mensagem de inicialização da rede *ZigBee*. Após isso a função *loop* é iniciada e o programa principal começa a ser executado.

- **Estados do menu do controle remoto**

O programa principal inicia fazendo uso de uma função *switch* cujo trecho do início da função é mostrado abaixo.

```
switch (state){
case1:
switch (qual_botao()) {
case bdesliga:
break;
case botaomudar:
break;
case botaocima:
lcd.clear(); proximo_estado(2); // antes de mudar de tela, é
necessário limpar o
break;                    // display com a função lcd.clear()
case botaobaixo:
lcd.clear(); proximo_estado(6);
break;
```

Como pode ser visto o parâmetro de entrada da primeira função *switch* é a variável *state*, essa variável foi inicializado com o valor 1, isso determina que a função *switch* sempre inicie o programa pelo estado 1 que é o menu inicial do programa, e após descobrir qual o estado atual do menu está sendo executado, o programa faz uso de uma nova função *switch* cujo parâmetro de entrada é uma função chamada *qual_botao ( )* que foi criada para descobrir se algum botão foi apertado, em seguida um novo *case* executa a ação determinada pelo botão que foi apertado, o exemplo do trecho acima mostra o *case* para 4 botões, bdesliga, botaomudar, botaocima e botaobaixo, por exemplo se o botão bdesliga for pressionado o programa não faz nada, se o botão botaomudar for pressionado o programa não faz nada, se o botão botaocima for pressionado o programa limpa a tela e passa para o estado 2 e se o botão botaobaixo for pressionado o programa limpa a tela e passa para o estado 6.

Se nenhum botão for pressionado o estado permanece no estado atual, uma função chamada *próximo_estado* ( ) foi criada para determinar o estado seguinte e o conteúdo que será apresentado na tela em cada estado.

✓ **Estado 1 do *software* do controle remoto (Tela 1)**

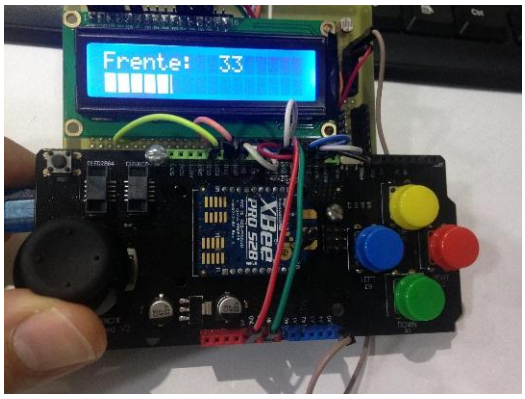


Figura 73: Tela 1 do menu do controle remoto.

Nesse estado são mostradas a velocidade e a direção da embarcação, a velocidade é mostrada com valores que variam de 0 (parado) e 100 (velocidade máxima), a velocidade também é mostra numa barra gráfica que será descrita mais à frente. Para atuar na velocidade e sentido da embarcação é preciso que nesse estado os pacotes contendo estas informações sejam continuamente enviados, para cumprir essa tarefa as funções *tx_motor_esquerdo* () e *tx_motor_direito* () foram criadas, essas funções enviam informações a um módulo com endereço específico.

A função chamada *mapeamento ()* foi idealizada para "mapear" os valores das leituras dos potenciômetros do *joystick* de forma a adequar esses valores possibilitando o cálculo da contribuição de cada motor no deslocamento total da embarcação, outra tarefa desta função é gerar uma zona morta quando o *joystick* estiver na posição central, ou seja, um valor PWM igual a zero é enviado aos motores quando *joystick* estiver em pequenas regiões centrais evitando assim latência no deslocamento da embarcação quando deveria estar parada.

Os botões do controle remoto têm as seguintes funções nesse estado:

- botaocima – vai para o estado 2;
- botaobaixo – vai para o estado 6;
- botaomudar - não faz nada;
- botaomenu – não faz nada;
- bdesliga – não faz nada;

✓ **Estados 2 e 3 do *software* do controle remoto (Tela 2 e Tela 3)**

O funcionamento desses estados é similar, sua função é mostrar na tela *LCD* valores lidos pelos sensores nos módulos *XBee* remotos, os sensores utilizados neste projeto foram descritos no Capítulo 2. No estado 2 temos as leituras dos sensores do motor esquerdo e no estado 3 temos as leituras dos sensores do motor direito.

As funções *rx_motor_esquerdo* () e *rx_motor_direito* () foram criadas para efetuarem o recebimento e a leituras dos sensores em cada um dos seus estados, elas os recebem apenas se o endereço constante for o do módulo remoto correspondente. As funções de envio de pacotes também são executadas nesses estados isso permite que o operador possa monitorar as variáveis (temperatura, corrente e tensão) de um determinado motor enquanto sua velocidade e sentido de deslocamento variam.

As informações são apresentadas na tela na seguinte sequência, temperatura nos *MOSFETs*, tensão na bateria, tensão no motor, corrente na bateria e corrente no motor.

Figura 74: Tela 2 do menu do controle remoto.

Como foi descrito na seção anterior os dados recebidos de cada sensor estão contidos em dois *bytes*, porém a informação está contida apenas em 10 *bits* desses 2 *bytes*, para facilitar a reconstrução dos 10 *bits* com o valor correto das variáveis medidas foi criada uma função chamada *combina ()*, o trabalho dessa função é usar esses dois *bytes* como parâmetros de entrada e combina-los em um inteiro de 10 *bits* corresponde a leitura.

Os botões têm as seguintes funções nesses estados;

- botaocima – vai para o estado 3 ou 4;
- botaobaixo – vai para o estado 1 ou 2;
- botaomudar - não faz nada;
- botaomenu – não faz nada;
- bdesliga – não faz nada;

✓ **Estado 4 do *software* do controle remoto (Tela 4)**

Esse estado foi criado para variar a velocidade em passos de 10 em 10 por cento da velocidade máxima dos motores tanto para a frente quanto para a ré, essa funcionalidade é útil quando se deseja manter os motores acionados em uma velocidade fixa, em testes por exemplo. Uma função chamada *incrementa_velocidade ()* foi criada para implementar essa funcionalidade, também foi criada uma função chamada *desliga ()* para possibilitar o desligamento imediato dos motores acionando o botão bdesliga a qualquer momento independentemente do valor que a função que incrementa a velocidade esteja, por exemplo imagine que se queira desligar os motores no momento em que estes estão a 40% da velocidade máxima, pois aconteceu algo inesperado, basta acionar o botão bdesliga que um pacote com as informações para desligamento são enviados aos dois motores imediatamente, a outra opção seria decrementar a velocidade até 0 o que poderia demorar a ponto de danificar o sistema.

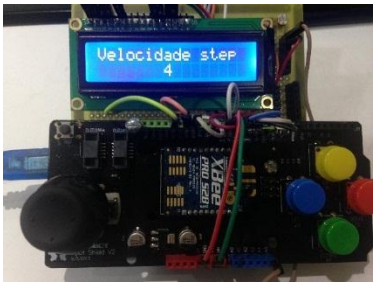


Figura 75: Tela 4 do menu do controle remoto.

Os botões têm as seguintes funções nesse estado;

- botaocima – vai para o estado 5;
- botaobaixo – vai para o estado 3;
- botaomudar - incrementa a velocidade;
- botaomenu – decrementa a velocidade;
- bdesliga – desliga os motores;

- **Estado 5 do *software* do controle remoto (Tela 5)**

O controle remoto sem fio *ZigBee* conta com um controle da iluminação do display, sendo que este controle pode ligar a luz de fundo, desligar ou operar de forma automática. Uma função chamada controle_luz () foi criada para auxiliar esse controle.

```
void controle_luz (int m)
{
if (m==2)
{
p= map(analogRead(luz),0,1023,255,0);
}
else
{
p =m*255;
}
analogWrite(Display,p);
}
```

Quando ligamos o controle remoto ou quando o estado atual é o estado 5 a variável chamada variavel1 é iniciada com o valor 1, a função *controle_luz ()* vista acima, recebe esse valor unitário como parâmetro de entrada m do tipo *int* (inteiro), isso faz com que na saída PWM do Arduino referente ao pino *Display* (pino 5) seja escrita com o valor p=255 isso acende o display com a intensidade máxima e é essa a função "Luz:ligada" que aparece escrito no *LCD*.

Quando o botão chamado botaomenu é apertado no estado 5 a variável1 é decrementada, ganhando o valor 0 que é novamente o parâmetro da função *controle_luz ()*, porém, agora apresenta o valor 0 como saída PWM e apaga o display da tela *LCD* escrevendo "Luz: desligada" na tela.

Se o botão chamado botaomudar é apertado a variável1 é incrementada, e se estava em 0 passa ao valor 1, seguindo os passos descritos anteriormente, e se estava em 1 passa ao valor 2 e novamente este é assumido pela função *controle_luz ()* e são executados os trechos dentro da função *if* que "mapeia" o valor da luminosidade lida pela porta analógica do Arduino denominada luz (porta A5) de 0 a 1023 em um valor digital de 0 a 255 que novamente é entregue ao pino de saída PWM, quando concluída esta etapa é escrito na tela "Luz: automática" e a luz de fundo da tela é proporcional ao nível de luminosidade ambiente.

A função *controle_luz ()* é executada em todos os estados para possibilitar que os ajustes funcionem em todos os menus.

```
case botaomudar:
lcd.clear();
variavel1 ++ ;
break;
```

```
case botaomenu:
lcd.clear();
variavel1 --;
break;

default:
if (variavel1>2 || variavel1<=0)
{
 variavel1=0 ;
}
controle_luz(variavel1);
proximo_estado(5);
}
break;
```

Outra funcionalidade interessante desse estado é a barra gráfica que se desloca na tela de forma crescente ou decrescente de acordo com o valor de entrada da função *bargrapgh ()*, esta função recebe um parâmetro do tipo *double* que pode variar de 0 a 100 e imprime uma barra gráfica na tela para interpretação imediata da grandeza monitorada.

Os botões do controle têm as seguintes funções nesse estado;

- botaocima – vai para o estado 6;
- botaobaixo – vai para o estado 4;
- botaomudar - ajusta controle de luz;
- botaomenu – ajusta controle de luz;
- bdesliga – desliga os motores;

✓ **Estado 6 do *software* do controle remoto (Tela 6)**

O estado 6 implementa o controle de contraste, este controle é executado pela função *controle_contraste ()* que recebe como parâmetro de entrada valores da variável chamada variavel2 que pode variar de 0 a 100, onde o botão chamado botaomenu decrementa e o botão chamado botaomudar incrementa esta variável, o nível de contraste é apresentado de forma percentual e de forma gráfica na tela. Foi observado que a porcentagem que apresenta um bom contraste para leitura dos caracteres da tela é de aproximadamente 55% sendo esse o ajuste inicial, mas isso pode variar de acordo com a percepção do observador e do ambiente de operação podendo ser facilmente ajustado para o valor mais conveniente.

```
void controle_contraste(int m )
{
int a = m*(255/100);
analogWrite(contraste,a);
}
```

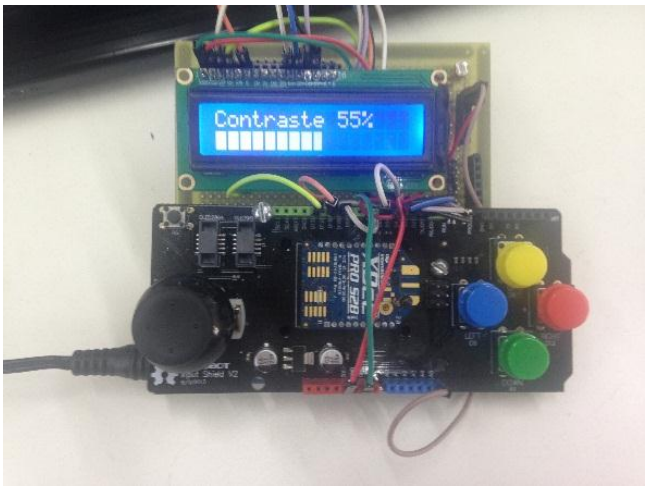


Figura 76: Tela 6 do menu do controle remoto remoto.

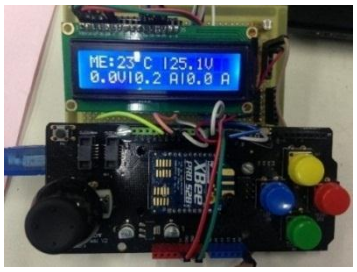
Os botões do controle têm as seguintes funções nesse estado;

- botaocima – vai para o estado 1;
- botaobaixo – vai para o estado 5;
- botaomudar - aumenta o contraste da tela;
- botaomenu – diminui o contraste da tela;
- bdesliga – desliga os motores;

7. Testes gerais

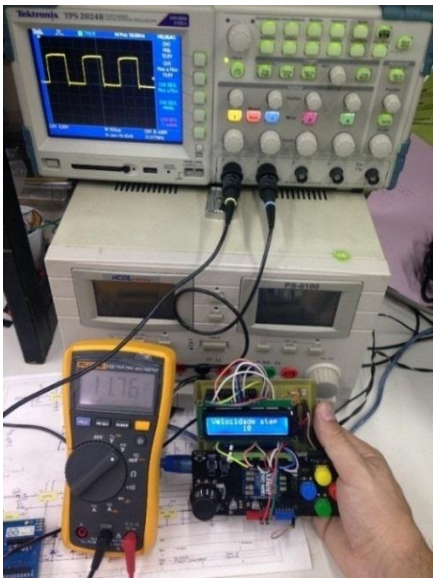
Todos os testes e experimentos deste projeto de graduação foram realizados no laboratório LEPAT – UERJ com apenas um motor submerso em um balde com água suficiente para gerar condições que simule aproximadamente a operação em um ambiente aquático real, o acionamento do segundo motor foi testado através de *leds* em protoboard, análise no osciloscópio e voltímetro.

- **Teste com duas baterias em série**



(a)

(b)

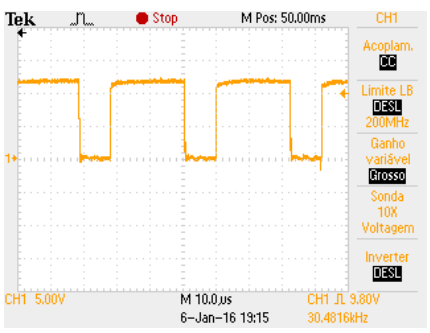


(c)

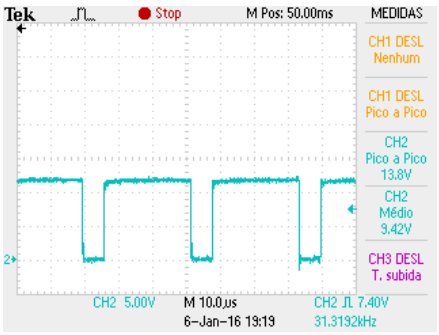
Figura 77: Testes com duas baterias: (a) Leitura da tensão de entrada (25,1V); (b) baterias em série; (c) Tensão entregue ao motor (11, 76V) e largura de pulso PWM em aproximadamente 50%.

Com duas baterias o sistema apresentou resultados bastante satisfatórios limitando a largura do pulso PWM em aproximadamente 50% isso entregou um nível de tensão de alimentação ao motor em torno de 12 volts conforme esperado, quando toda velocidade foi solicitada.

- **Pulsos PWM aplicado nos *gates* dos *MOSFETs***



(a)

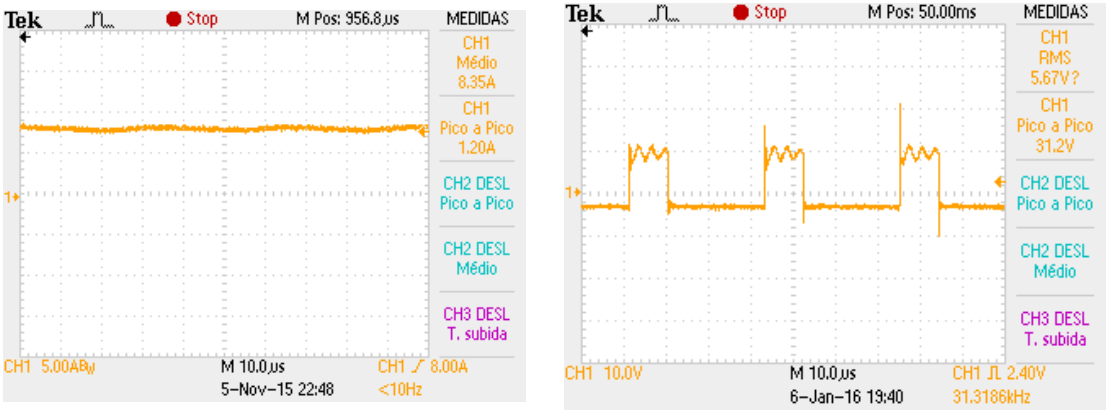


(b)

Figura 78: Pulsos PWM (a) frente com duty cycle = 30% (b) ré com duty cycle = 20%.

Os pulsos aplicados nos *gates* dos *MOSFETs* se apresentaram da forma esperada, destacando que estes pulsos estão invertidos como foi explicado no *software*.

- **Forma de onda da tensão e corrente no motor**



(a) (b)

Figura 79: Formas de ondas do motor (a) Corrente (b) tensão.

- **Consumo do circuito com motor desligado**

O consumo dos circuitos de acionamento, Arduino, circuito de aquisição dos sensores e *XBee* está em torno de 110mA com uma bateria de 12V, esse nível de consumo por um longo prazo pode ser preocupante do ponto de vista energético.

- **Alcance da comunicação *ZigBee***

A comunicação com os módulos *Xbee* se mostrou estável até aproximadamente uns 40 metros em ambientes internos com paredes (laboratório) o que está de acordo com o *data sheet*, após esta distância a comunicação apresentou falhas. Infelizmente, não foi possível realizar testes em campo aberto, mas é provável que se consiga alcançar a distância de 120 metros especificadas para os módulos *XBee* série 2.

8. Conclusão

Os *softwares* e os circuitos desenvolvidos neste projeto de graduação são de baixo custo de aquisição e apresentaram bastante robustez durante os testes, onde os *MOSFETs* têm um papel de destaque por apresentarem excelentes características como a baixíssima resistência em condução e desempenho muito satisfatório.

Durante a fase de testes com a ponte H o *driver* HIP4081 se mostrou bastante confiável reduzindo a chance de um eventual acidente por erros no acionamento dos *MOSFETs* devido as proteções que este componente oferece e isso trouxe mais liberdade e tranquilidade para os testes.

Um dos maiores desafios desse projeto era encontrar um protocolo de comunicação sem fio que fosse capaz de controlar pelo menos dois motores de forma segura, barata e estável e o protocolo *ZigBee* cumpriu essa tarefa através dos módulos *XBee*. Foi necessário bastante pesquisa para entender o modo de comunicação API dos módulos *XBee*, sendo nitidamente percebida a falta de bibliografia em língua portuguesa acerca deste assunto.

A plataforma embarcada Arduino se mostrou extremamente simples e confiável tanto no que se refere ao s*oftware* quanto ao *hardware*, com vasta bibliografia, bibliotecas e exemplos de aplicação além dos *shields* oferecerem ótimas soluções como foi o caso do controle remoto sem fio desenvolvido. O tempo de resposta dos comandos de velocidade é pequeno, tornando o controle dos motores divertido e intuitivo.

A proposta do projeto era realizar um sistema eletrônico para o controle de velocidade de uma embarcação teleoperada e este objetivo foi alcançado com sucesso.

8.1. Contribuições deste trabalho

Esse trabalho pode contribuir no que diz respeito ao acionamento da maioria dos motores de corrente contínua, observando que as especificações dos componentes podem ser alteradas e adaptadas às características de uma aplicação específica. Também pode contribuir no que diz respeito ao modo de comunicação API dos módulos *XBee* com plataforma Arduino, pois exemplifica a criação da rede *ZigBee*, a parametrização dos módulos e as configurações de envio, recebimento e tratamento dos pacotes API.

8.2. Propostas para trabalhos futuros

Como sugestão de trabalho futuro pode-se propor a criação de um *software* escrito preferencialmente em linguagem de programação orientada a objeto para o gerenciamento da rede *ZigBee*. Este *software* seria instalado no *notebook* a bordo da embarcação que estaria ligado ao módulo coordenador da rede *ZigBee* através de uma porta USB, enviaria os comandos de acionamento aos motores e receberia os dados das leituras dos sensores, por exemplo, através de internet sem fio aumentando o alcance físico da comunicação da embarcação teleoperada.

REFERÊNCIAS

ALLEGRO MICROSYSTEMS. 2011, **data sheet ACS756**, Disponível em: <http://www.allegromicro.com/~/Media/Files/Datasheets/ACS756-Datasheet.ashx>.

ARDUINO, 2009, **ArduinoBoardMega**, Disponível em: http://arduino.cc/en/Main/ArduinoBoardMega

ARDUINO, 2009, **ArduinoBoardUno**, Disponível em: http://arduino.cc/en/Main/ArduinoBoardUno

BASTOS, 2012, Alex Vidigal Bastos, **LCD (liquid cristal display)**, slides e apostila da matéria SISTEMAS EMBUTIDOS (UFOP), disponível em: http://www.decom.ufop.br/alex/arquivos/bcc425/slides/LCD.pdf

BUORO, Angelo Sabbatini; 2013, **Controle dos motores e acionamento sem fios de uma pequena embarcação**, 2013. Projeto de Graduação em Engenharia Eletrônica - UERJ.

CUNHA, J. P. V. S; 1992, **Projeto e Estudo de Simulação de um Sistema de Controle a Estrutura Variável de um Veículo Submarino de Operação Remota**. Rio de Janeiro: COPPE/UFRJ.

DIGI INTERNATIONAL, 2008, **XBee™ ZNet 2.5/XBee-PRO™ ZNet 2.5 OEM RF Modules**, Product Manual v1.x.4x - *ZigBee* Protocol, For OEM RF Module Part Numbers: XB24-BxIT-00x.

DIGI INTERNATIONAL, 2013, **XBee®/XBee-PRO® RF Modules**, **Product Manual v1.xEx - 802.15.4 Protocol,** For RF Module Part Numbers: XB24-A...-001, XBP24-A...-001

FRADEN, J, 2003, **Handbook of Modern Sensors**. 3. ed. Califórnia: Springer.

HIGINBOTHAN, J. R.; Moisan, J. R.; Schirtzinger, C.; Linkswiler, M.; Yungel, J.; Orton, P; 2008, Update **on the Development and Testing of a New Long Duration Solar Powered Autonomous Surface Vehicle**, Oceans 2008, pp. 1-10, Setembro, Quebec, Canadá.

INTERNATIONAL RECTIFIER, 2012, **data sheet Power Mosfet IRF 1104**, disponível em:

<http://www.irf.com/product-info/datasheets/data/irf1404zpbf.pdf>

INTERSIL, 2003, **HIP4081 80V high frequency H- bridge drive**r, application note. Disponível em: http://www.hvlabs.com/files/HIP4081application.pdf

INTERSIL, 2015, **80V/2.5A Peak, High Frequency Full Bridge FET Driver**, datasheet. Disponível em: https://www.intersil.com/content/dam/Intersil/documents/hip4/hip4081a.pdf

KORMANN, Brigitte, 2003, **High-Efficiency, Regulated Charge Pumps for High-Current Applications**, copyright © 2003, Texa instruments.

KOSOW, Irvin Lionel, 1982, **Máquinas elétricas e transformadores,** 4º edição**,** tradução Felipe Luiz Ribeiro Daielle, Ed Globo.

MOHAN, N; 2002. **Power Electronics: Converters, Applications, and Design.** 2. ed. Michigan: Wiley.

MARGOLIS, Michael, 2011, **Arduino Cookbook**, Second edition, December.

MAXSTREAM, 2007, **XBee™/XBee-PRO™ OEM RF Modules**, Product Manual v1.xAx - 802.15.4 Protocol, IEEE® 802.15.4 OEM RF Modules by MaxStream.

NATIONAL SEMICONDUCTOR CORPORATION. 2010, **LM35 - Precision Centigrade Temperature Sensors,** Disponível em: <http://www.ti.com/lit/ds/symlink/lm35.pdf>.

PATANÉ, Edson João, 2008, **Implementação de controle de velocidade em malha fechada para motores de corrente contínua utilizando sistemas de aquisição de dados**, Dissertação de mestrado, Escola de Engenharia Mauá.

POMILIO, 2014, José Antenor, **Eletrônica de potência**, livro disponível em: http://www.dsce.fee.unicamp.br/~antenor/elpot.html

RASHID, Muhammad H., **Eletrônica de potência: Circuitos, dispositivos e aplicações**, tradução Carlos Alberto Favato, Makron books do Brasil.

RAMOS, Jadeilson de Santana bezerra, 2012, **Instrumentação eletrônica sem fio: transmitindo dados com módulos *XBee Zigbee* e PIC16F877A**, Editora ÉRICA Ltda.

RAMOS, Jadeilson de Santana bezerra, 2010, **Sistema digital de instrumentação sem fio de uma mola helicoidal TI-NI usando tecnologia zigbee, usb e Linux**, Dissertação de mestrado, Universidade federal de Pernambuco.

RAPPAPORT, Theodore S., 2009, **Comunicação sem fio princípios e práticas**, 2ª Edição, Tradução Daniel Vieira, Pearson Education do Brasil Ltda.

ROSÁRIO, Rafael Vida de Castro; 2013, **Sistema para monitoração de uma embarcação não tripulada**, Projeto de Graduação em Engenharia Eletrônica - UERJ.

SCHULTZE, H. J. 2012, **Projeto e Construção de uma Embarcação Teleoperada,** Projeto de Graduação em Engenharia Eletrônica - UERJ.

SPARKFUN, 2008, **Specification of LCD module,** XIAMEN AMOTEC DISPLAY CO., LTD.

TEXAS INSTRUMENTS, 2013, **Data Sheet**, LM35 Precision Centigrade Temperature Sensors.

ZUCATO, Fábio Labegalinni, 2009, **Rede ZigBee gerenciada por sistema de monitoramento remoto utilizando TCP/IP e GPRS**, Dissertação de mestrado, Escola de Engenharia de São Carlos da Universidade de São Paulo.

APÊNDICES

Apêndice A – Diagramas do circuito de acionamento

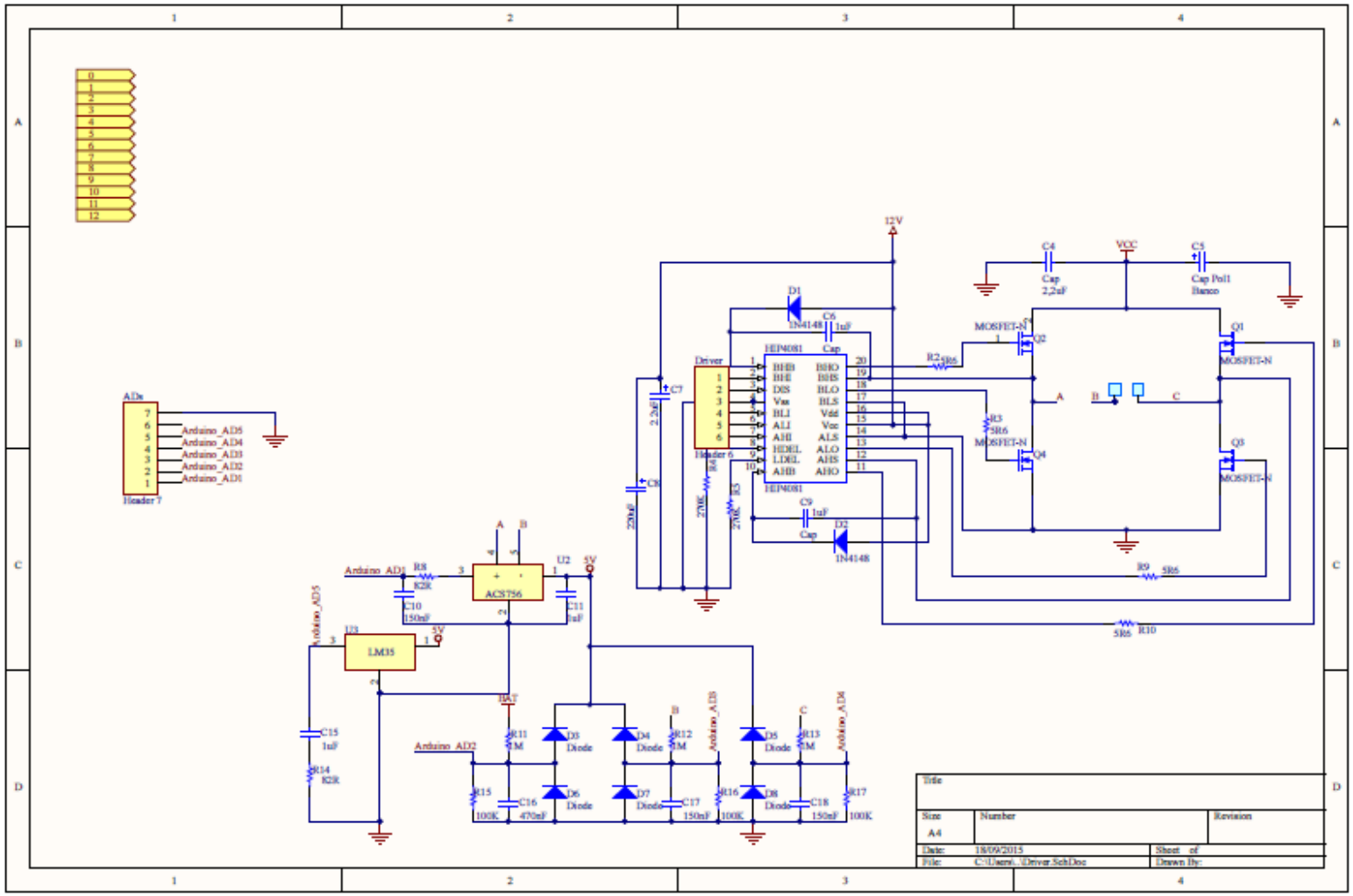


Figura 80: Ponte H e circuito dos sensores

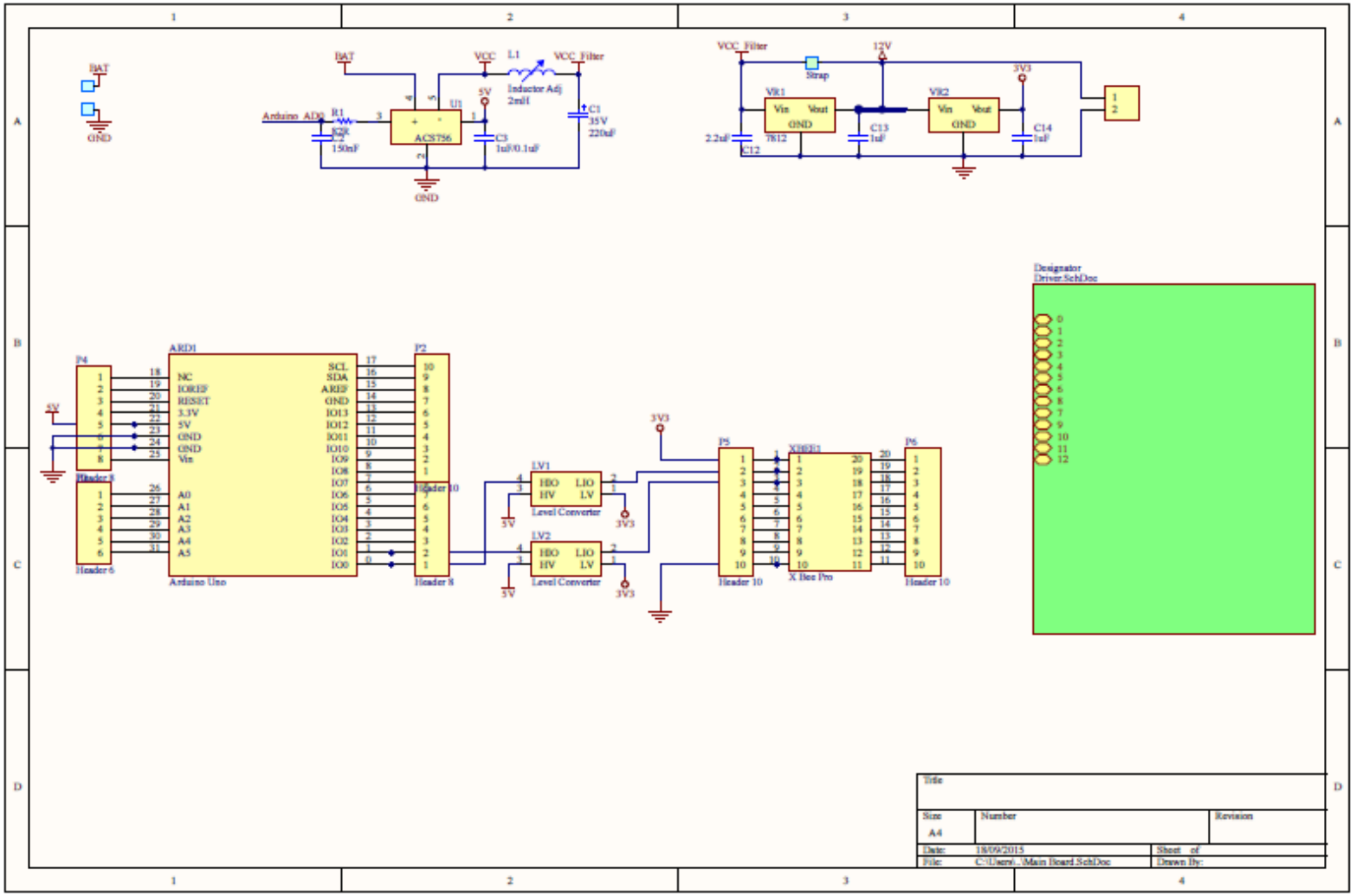


Figura 81: Arduino e XBee

Apêndice B – Softwares Arduinos - XBee

Arduino – Dispositivo final ZigBee (motor)

```
// AUTOR: Lenielson Rodrigues de Sousa
// Engenharia eletrônica - UERJ
// data: 26/10/2015
//=============================================================================================
int n=0;
float Vcc=4.994;// tensao vcc medida
float tensao_nominal_motor = 12.0;
//float tensao_entrada_medida=0.0;
int amostras = 100;
int temp = 0; //**********
int tensao_bat_1 = 0.0;
const float constante_tensao_bateria = (Vcc/1023)*11;
int tensao_Va = 0;
int tensao_Vc = 0;
int tensao_motor_1=0;
int corrente_bat_1 = 0;
int corrente_motor_1 = 0;
int y1=0;
int y2=0;
unsigned int dado1=0x00;//0x02;// 0x02 0f = e igual a 527 centro do potenciometro
unsigned int dado2=0x00;//0x0f;
unsigned int dado3=0x00;//0x01;
unsigned int dado4=0x00;//0x5f;
int combined= 0;
const unsigned int DIS = 7; // HABILITADOR DO DRIVER
const unsigned int AHI = 6; // NIVEL LOGICO "1" PARA A PORTA AHI DO HIP4081
const unsigned int BHI = 5; // NIVEL LOGICO "1" PARA A PORTA AHI DO HIP4081
const unsigned int frente = 9; //
const unsigned int re = 10;
const unsigned int led =12;
const unsigned int errorLed = 4;
const unsigned int statusLed = 8;

#include <XBee.h>
XBee xbee = XBee();
//=============================================Endereço Xbee Coordenador
uint8_t payload[] = {0,0,0,0,0,0,0,0,0,0};// Array payload dados a serem enviados
//=== SH + SL endereço do XBEE destino
XBeeAddress64 addr64 = XBeeAddress64(0x0013a200,0x4098bf1e);
ZBTxRequest zbTx = ZBTxRequest(addr64, payload, sizeof(payload));
ZBTxStatusResponse txStatus = ZBTxStatusResponse();
XBeeResponse response = XBeeResponse();//= cria objeto de respota reusavel
ZBRxResponse rx = ZBRxResponse();
ModemStatusResponse msr = ModemStatusResponse();
//=============================================================SETUP
void setup()
{
Serial.begin(9600); // Inicia porta serial à 9600 bps:
 xbee.begin(Serial);
 TCCR1B = TCCR1B & B11111000 | B00000001;//altera frequencia PWM das saidas 9 e 10 para 31Khz
 pinMode(DIS, OUTPUT);
 pinMode(statusLed, OUTPUT);
 pinMode(errorLed, OUTPUT);
  //declaração dos pinos digitais
 pinMode(frente, OUTPUT);
 pinMode(re, OUTPUT);
 pinMode(BHI, OUTPUT);
 pinMode(AHI, OUTPUT);
 flashLed(statusLed, 3, 50);//ligando
 parado();
 }
 //============================================================================== LOOP
void loop()
{

 (xbee.readPacket(500)); //=== Ler continuamente algum pacote Zigbee (ZB)

if (xbee.getResponse().isAvailable()) //==recebeu algo
{
```

```
if (xbee.getResponse().getApiId() == ZB_RX_RESPONSE)//== é um Pacote ZB?(API ID 0x90 -
received packet)
{
 xbee.getResponse().getZBRxResponse(rx); //==Aloca os dados recebidos na classe rx
if (rx.getOption() == ZB_PACKET_ACKNOWLEDGED)
{
  // pisca o led de status se o remetente teve pacote recebido
flashLed(statusLed, 1, 10);
}
else
{
parado();
}
tensao_bat_1 = analogRead(A1); // mede tensao na bateria
corrente_motor_1=(analogRead(A0)- 512);// corrente no motor
if (corrente_motor_1 <= 123)// Limitador de corrente em 30A
{
digitalWrite(DIS,LOW);// Habilita o MOSFET driver HIP4081
digitalWrite(AHI,HIGH);// habilita AHI
digitalWrite(BHI,HIGH);// habilita BHI
//========================================== Extrai dados recebidos
y1 =controle_tensao( rx.getData(0));
y1=~y1;
analogWrite(frente, y1);
y2=controle_tensao( rx.getData(1));
y2=~y2;
analogWrite(re,y2);
}
else
{
parado();
}

//=======================================inicia a aquisição dos dados dos sensores
temp = analogRead(A4); // temperatura dos mosfets
payload[0] = temp >> 8 & 0xff;
payload[1] = temp & 0xff;
//=======================Leitura e armazenamento do sensor de tensão no payload
payload[2] = tensao_bat_1 >> 8 & 0xff;
payload[3] = tensao_bat_1 & 0xff;
//================================================= tensão no motor
tensao_Va = analogRead(A2); // tensao no braço esquerdo da ponte H (Mosfets Q2 e Q4)
tensao_Vc = analogRead(A3); // tensao no direito da ponte H (Mosfets Q1 e Q3)
tensao_motor_1= (tensao_Va - tensao_Vc);
payload[4] = tensao_motor_1 >> 8 & 0xff;
payload[5] = tensao_motor_1 & 0xff;
//=========================Leitura e armazenamento do sensor de corrente no payload
corrente_bat_1=(analogRead(A5)- 512);// corrente na bateria
payload[6] = corrente_bat_1 >> 8 & 0xff;
payload[7] = corrente_bat_1 & 0xff;
//=================================================
payload[8] = corrente_motor_1 >> 8 & 0xff;
payload[9] = corrente_motor_1 & 0xff;
//=================================================
xbee.send(zbTx);//==========>>>>>>>>>>>>>>>>>>>>>>>>> Envia o pacote para ocoordenador
//=================================================
if (xbee.readPacket(500)) //<<<<
{
if (xbee.getResponse().getApiId() == ZB_TX_STATUS_RESPONSE)
{
xbee.getResponse().getZBTxStatusResponse(txStatus);
if (txStatus.getDeliveryStatus() == SUCCESS)
{
//digitalWrite(statusLed, HIGH); //====== Sucesso no envio!!!!!
} else
{
flashLed(errorLed, 2, 100);
}
}
}
else
{
 parado();
 flashLed(errorLed, 2, 100);
}
}
```

```
}
}

//============================================================ FINAL DO LOOP

//========== Função que gera um inteiro de 10 bits a partir de 2 bytes do Payload
int combina(unsigned int x_high, unsigned int x_low)
{
  int combined;
  combined = x_high;
  combined = combined*256;
  combined |= x_low;
  return combined;
}
//======================================== Função para o estado parado
void parado ()
{
digitalWrite(DIS,HIGH);
digitalWrite(AHI,LOW);// HABILITA AHI
digitalWrite(BHI,LOW);// HABILITA BHI
digitalWrite(frente,253);
digitalWrite(re,253);
}
//================================================Função para piscar LED de sinalização
void flashLed(int pin, int times, int wait) {

 for (int i = 0; i < times; i++)
 { digitalWrite(pin, HIGH);
  delay(wait);
  digitalWrite(pin, LOW);
   if (i + 1 < times)
   { delay(wait); }}}

//=====================================================================
int controle_tensao(int b)
{
float fator =(tensao_bat_1*constante_tensao_bateria/tensao_nominal_motor);
if (fator > 1.0)
{
b = b*(1.0/fator);
}
return b;
}
```

Arduino – Coordenador ZigBee (controle remoto)

```
// Autor:  Lenielson Rodrigues de Sousa
// Engenharia eletrônica UERJ
//23/10/2015
//============================== Definição dos pinos dos botões do LCD
#include <LiquidCrystal.h>
#define Display  5
#define contraste 6
#define botaomenu    9
#define botaomudar  12
#define botaocima      8
#define botaobaixo    A1
#define bdesliga A0
#define luz A5

#define bMenu0    90  // Valor de referência que a
#define bChange0  91  // função qual_botao() passa
#define bUp0      92  // indicando que um botão foi
#define bDown0    93  // solto
#define bdesliga0 94
boolean aMenu, aChange, aUp, aDown,adesliga;//=================Grava o último valor lidos nos
botões.
int variavel=0;// variável a ser alterada pelo menu
int variavel1=1;
int variavel2=55;
char state=1;// variável que guarda posição atual do menu
LiquidCrystal lcd(13, 11, 10, 4, 3, 2);  // Declaração do objeto tipo lcd
//================================== Inclui biblioteca XBEE e cria o objeto XBee
```

```
#include <XBee.h>
XBee xbee = XBee();
//=====================================Endereço XBee motor esquerdo
uint8_t payload[] = {0x02,0x0f,0x01,0x5f};//==== Array payload dados a serem enviados
// SH + SL endereço do XBEE destino
XBeeAddress64 addr64 = XBeeAddress64(0x0013a200,0x4098bf6d );
ZBTxRequest zbTx = ZBTxRequest(addr64, payload, sizeof(payload));
long end_motor_esquerdo;
//================================================ Endereço Xbee motor direito
XBeeAddress64 addr64_2 = XBeeAddress64(0x0013a200,0x409828d0 );
ZBTxRequest zbTx_2 = ZBTxRequest(addr64_2, payload, sizeof(payload));
//=================================================================
ZBTxStatusResponse txStatus = ZBTxStatusResponse();
XBeeResponse response = XBeeResponse();
ZBRxResponse rx = ZBRxResponse();
//=============================================Variaveis de controle
int a,b,c,d,p,x,y = 0;
const unsigned int eixo_y =2;
const unsigned int eixo_x =3;
float temperatura_1 = 0;
float tensao_bat_1=0;
float tensao_motor_1=0;
float corrente_bat_1=0;
float corrente_motor_1=0;
float temperatura_2 = 0;
float tensao_bat_2=0;
float tensao_motor_2=0;
float corrente_bat_2=0;
float corrente_motor_2=0;
float Vcc=4.994;
float raiz=0;
int amostras =10;
int n=0;
//============================== caracteres para gerar barra gráfica
#define lenght 16.0
double percent=100.0;
unsigned char vb;
unsigned int parte;
byte p1[8] = {
  0x10,
  0x10,
  0x10,
  0x10,
  0x10,
  0x10,
  0x10,
  0x10};

byte p2[8] = {
  0x18,
  0x18,
  0x18,
  0x18,
  0x18,
  0x18,
  0x18,
  0x18};
byte p3[8] = {

  0x1C,
  0x1C,
  0x1C,
  0x1C,
  0x1C,
  0x1C,
  0x1C,
  0x1C};
byte p4[8] = {
  0x1E,
  0x1E,
  0x1E,
  0x1E,
  0x1E,
  0x1E,
  0x1E,
  0x1E};
```

```
byte p5[8] = {
  0x1F,
  0x1F,
  0x1F,
  0x1F,
  0x1F,
  0x1F,
  0x1F,
  0x1F};
//================================================SETUP
void setup()
{
Serial.begin(9600);
xbee.setSerial(Serial);
lcd.begin(16, 2);  // Iniciando a biblioteca do LCD
controle_contraste(variavel2);
lcd.createChar(0, p1);
lcd.createChar(1, p2);
lcd.createChar(2, p3);
lcd.createChar(3, p4);
lcd.createChar(4, p5);
pinMode(Display,OUTPUT);
pinMode(contraste,OUTPUT);
pinMode(luz,  INPUT);
pinMode(botaomenu,  INPUT);
pinMode(botaomudar,INPUT);
pinMode(botaocima,    INPUT);
pinMode(botaobaixo,  INPUT);
digitalWrite(botaomenu,  HIGH);// Aciona o pull-up interno
digitalWrite(botaomudar,HIGH); // dos botões
digitalWrite(botaocima, HIGH);
digitalWrite(botaobaixo, HIGH);
digitalWrite(Display, HIGH);  // Liga a luz do display.
pinMode(eixo_y, OUTPUT);
pinMode(eixo_x, OUTPUT);
lcd.setCursor(0,0);
lcd.print("<<<LIGANDO>>>");
lcd.setCursor(0,1);
lcd.print("<<REDE ZIGBEE>>");
delay(2000);
}
//==================== LOOP PRINCIPAL <<<<<<<<<<<<<<<<<<<<<<<<<<<<
void loop()
{
switch (state) {  // checa qual tela atual
case 1:  //==================================executado quando na TELA 1
switch (qual_botao()) {
case bdesliga:
break;
case botaomudar:

break;
case botaomenu:

break;
case botaocima:
lcd.clear(); proximo_estado(2);
break;
case botaobaixo:
lcd.clear(); proximo_estado(6);
break;
default:   // Caso nenhum botão tenha sido apertado permanece no estado 1
proximo_estado(1);
tx_motor_esquerdo();
delay(100);
tx_motor_direito();
delay(100);
controle_luz(variavel1);
}
break;
case 2: //========================================== executado quando na TELA 2
switch (qual_botao()) {
case bdesliga:
lcd.clear();
desliga();
break;
```

```
case botaomudar:
break;
case botaocima:
lcd.clear(); proximo_estado(3);
break;
case botaobaixo:
lcd.clear(); proximo_estado(1);
break;
default:
proximo_estado(2);
tx_motor_esquerdo();
delay(80);
rx_motor_esquerdo();
proximo_estado(2);
tx_motor_direito();
delay(80);
controle_luz(variavel1);
}
break;
case 3:  //================================================= executado quando na TELA 3
switch (qual_botao()) {
case bdesliga:
lcd.clear();
desliga();
break;
case botaomudar:
break;
case botaocima:
lcd.clear(); proximo_estado(4);
break;
case botaobaixo:
lcd.clear(); proximo_estado(2);
break;
default:
proximo_estado(3);
tx_motor_direito();
delay(110);
rx_motor_direito();
tx_motor_esquerdo();
controle_luz(variavel1);
}
break;
case 4: //================================================ executado quando na TELA 4
switch (qual_botao()) {
case bdesliga:
lcd.clear();
desliga();
break;
case botaomenu:
lcd.clear();
variavel--;
if (variavel<-10)
{
  variavel=0;
}
incrementa_velocidade (variavel);
break;
case botaomudar:
lcd.clear();
variavel++ ;
if (variavel>10)
{
  variavel=0;
}
incrementa_velocidade(variavel);
break;
case botaocima:
lcd.clear(); proximo_estado(5);
break;
case botaobaixo:
lcd.clear(); proximo_estado(3);
break;
default:
proximo_estado(4);
controle_luz(variavel1);
}
```

```
break;

case 5: //======================================== executado quando na TELA 5
switch (qual_botao())
{
case bdesliga:
lcd.clear();
desliga();
break;
case botaocima:
lcd.clear(); proximo_estado(6);
break;

case botaobaixo:
lcd.clear(); proximo_estado(4);
break;

case botaomudar:
lcd.clear();
variavel1 ++ ;
break;

case botaomenu:
lcd.clear();
variavel1 --;
break;

default:
if (variavel1>2 || variavel1<=0)
{
 variavel1=0 ;
}
controle_luz(variavel1);
proximo_estado(5);
}
break;

case 6: //=================================== executado quando na TELA 6
switch (qual_botao())
{
case bdesliga:
lcd.clear();
desliga();
break;
case botaocima:
lcd.clear(); proximo_estado(1);
break;

case botaobaixo:
lcd.clear(); proximo_estado(5);
break;

case botaomudar:
lcd.clear();
variavel2 ++ ;
break;

case botaomenu:
lcd.clear();
variavel2 --;
break;

default:
if (variavel2>100 || variavel2<=0)
{
 variavel2=0 ;
}
controle_contraste(variavel2);
controle_luz(variavel1);
proximo_estado(6);
}
break;
}
}
//============================================ FIM DO LOOP
//========================== Função que diz qual botão foi apertado
```

```
char qual_botao() {
if (aMenu!=digitalRead(botaomenu)) {
aMenu=!aMenu;
if (aMenu) return bMenu0; else return botaomenu;
} else
if (aChange!=digitalRead(botaomudar)) {
aChange=!aChange;
if (aChange) return bChange0; else return botaomudar;
} else
if (aUp!=digitalRead(botaocima)) {
aUp=!aUp;
if (aUp) return bUp0; else return botaocima;
} else
if (aDown!=digitalRead(botaobaixo)) {
aDown=!aDown;
if (aDown) return bDown0; else return botaobaixo;
} else

if (adesliga!=digitalRead(bdesliga)) {
adesliga=!adesliga;
if (adesliga) return bdesliga0; else return bdesliga;
}
return 0;
}
//============================== Função para descobrir o estado e imprimir informações
void proximo_estado(char index) {
state = index;  // Atualiza a variável state para a nova tela
switch (state) { // verifica qual a tela atual e exibe o conteúdo correspondente
case 1: //=================================================TELA 1
lcd.clear();
lcd.setCursor(0,0);
mapeamento();

if (y==0 && x==0)
{
lcd.print("Parado:");
}
if (y>0 && x==0)
{
lcd.print("Frente:");
}
if (y<0 && x==0)
{
lcd.print("Re:");
}
if (y>0 && x>0)
{
lcd.print("Frente D:");
}
if (y>0 && x<0)
{
lcd.print("Frente E:");
}
if (y<0 && x>0)
{
lcd.print("Re D :");
}
if (y<0 && x<0)
{
lcd.print("Re E :");
}

if (y==0 && x>0)
{
lcd.print("Direita:");
}
if (y==0 && x<0)
{
lcd.print("Esquerda:");
}

lcd.setCursor(9,0);
lcd.print(raiz,0);
bargraph (raiz);

xbee.readPacket(100);
```

```
if (xbee.getResponse().getApiId() == ZB_TX_STATUS_RESPONSE || xbee.getResponse().getApiId() ==
ZB_RX_RESPONSE)
{
if ( txStatus.getDeliveryStatus() == SUCCESS)
{
lcd.setCursor(13,0);
lcd.print(">|<");
 }
}

break;
case 2: //================================================= TELA 2
lcd.setCursor(0,0);
lcd.print("ME: ");
lcd.setCursor(3,0);
lcd.print(temperatura_1,0);
lcd.write(B11011111); //Simbolo de graus celsius
lcd.print("C");
lcd.setCursor(8,0);
lcd.print("|");
lcd.setCursor(9,0);
lcd.print(tensao_bat_1+0.26,1);
lcd.print("V");
//linha 1
lcd.setCursor(0,1);
lcd.print(tensao_motor_1,1);
lcd.print("V");
lcd.setCursor(4,1);
lcd.print("|");
lcd.setCursor(5,1);
lcd.print(corrente_bat_1,1);
lcd.setCursor(9,1);
lcd.print("A");
lcd.setCursor(10,1);
lcd.print("|");
lcd.setCursor(11,1);
lcd.print(corrente_motor_1,1);
lcd.setCursor(15,1);
lcd.print("A");
break;
case 3: //================================================= TELA 3
lcd.setCursor(0,0);
lcd.print("MD: ");
lcd.setCursor(3,0);
lcd.print(temperatura_2,0);
lcd.write(B11011111); //Simbolo de graus celsius
lcd.print("C");
lcd.setCursor(8,0);
lcd.print("|");
lcd.setCursor(9,0);
lcd.print(tensao_bat_2,1);
lcd.print("V");
//linha 1
lcd.setCursor(0,1);
lcd.print(tensao_motor_2,1);
lcd.print("V");
lcd.setCursor(4,1);
lcd.print("|");
lcd.setCursor(5,1);
lcd.print(corrente_bat_2,1);
lcd.setCursor(9,1);
lcd.print("A");
lcd.setCursor(10,1);
lcd.print("|");
lcd.setCursor(11,1);
lcd.print(corrente_motor_2,1);
lcd.setCursor(15,1);
lcd.print("A");

break;
case 4: //================================================== TELA 4
lcd.setCursor(0,0);
lcd.print("Velocidade step");
lcd.setCursor(7,1);
lcd.print(variavel, DEC);  // mostra o valor de "variavel"
break;
```

```
case 5: //======================================================== TELA 5
lcd.setCursor(0,0);
lcd.print("Luz:");
if (variavel1==1)
{
lcd.setCursor(4,0);
lcd.print("Ligada");
}
else if (variavel1==0)
{
lcd.setCursor(4,0);
lcd.print("Desligada");
}
else if (variavel1==2)
{
lcd.setCursor(6,0);
lcd.print("Automatica");
p=map(p,0,255,0,100);
bargraph(p);
}
break;
case 6: //======================================================== TELA 6
lcd.setCursor(0,0);
lcd.print("Contraste");
lcd.setCursor(12,0);
lcd.print("%");
lcd.setCursor(10,0);
lcd.print(variavel2);
bargraph(variavel2);
break;
default: ;
}
}
//===== Função para mapear leituras dos potenciômetros do joystick e definir dead band
void mapeamento ()
{
a = analogRead(eixo_y);//leitura da posição Y do joystick
b = analogRead(eixo_x);//leitura da posição X do joystick
int dead_band_superior_y = 530;//== Zona morta, para evitar pequenas latências no deslocamento
int dead_band_inferior_y = 524;//==dos motores quando joystick estiver na posição
central(parado)
int dead_band_superior_x = 504;
int dead_band_inferior_x = 498;
//=======================================
if (a<dead_band_inferior_y)
{
y = map (a,523,0,0,254); // 0 a 254 porque a largura do PWM não pode ser 100%
}
else if (a> dead_band_superior_y)
{
y = map (a,531,1023,0,-254);
}
else if ( a>= dead_band_inferior_y || a <= dead_band_inferior_y)
{
y=0;
}
//=======================================
if (b < dead_band_inferior_x)
{
x = map (b,499,0,0,254);
}
else if (b > dead_band_superior_x)
{
x= map (b,505,1023,0,-254);
}
else if ( b>= dead_band_inferior_x || b <= dead_band_inferior_x)
{
x=0;
}
//=======================================
if (x!=0 && y!=0)//calcula velocidade diagonais na escala de 0 a 100
{
raiz=sqrt(sq(x*100/254)+sq(y*100/254));
raiz=map(raiz,0,141,0,100);
}
else
```

```
{
raiz=sqrt(sq(x*100/254)+sq(y*100/254));
}
}
//================================= função para enviar pacotes para o motor esquerdo
void tx_motor_esquerdo()
{
 mapeamento();
 //========================== parado
if (x==0 && y==0)
{
payload[0] = 0 ;
payload[1] = 0;
}
//========================== frente
if (x==0 && y>0)
{
payload[0] = y ;
payload[1] = 0;
}
//========================== re
else if (x==0 && y<0)
{
y=-y;
payload[0] = 0 ;
payload[1] = y;
}
//========================== frente direita
else if (x>0 && y>0)
{
payload[0] = y ;
payload[1] = 0;
}
//========================== frente esquerda
else if (x<0 && y>0)
{
x=-x;
payload[0] = x ;
payload[1] = 0;
}
//========================== re direita
else if (x>0 && y<0)
{
payload[0] = 0 ;
payload[1] = x;
}
//========================== re esquerda
else if (x<0 && y<0)
{
 y=-y;
payload[0] = 0 ;
payload[1] = y;
}
//========================== direita
else if (x>0 && y==0)
{
payload[0] = x ;
payload[1] = 0;
}
//========================== esquerda
else if (x<0 && y==0)
{
x=-x;
payload[0] = 0 ;
payload[1] = x;
}

xbee.send(zbTx);//================ Envia
pacote

}

//==== função para receber pacotes do motor esquerdo e extrair informações dos sensores
void rx_motor_esquerdo()
{
xbee.readPacket();
```

```
if(xbee.getResponse().isAvailable())
{
xbee.getResponse().getApiId() == ZB_RX_RESPONSE;
xbee.getResponse().getZBRxResponse(rx); // Extrai pacote de retorno
XBeeAddress64 end_motor_esquerdo = rx.getRemoteAddress64();
end_motor_esquerdo.getLsb();

if (end_motor_esquerdo.getLsb() == 0x4098bf6d )
 {
//================================================== Temperatura
int temperatura_1_1 = combina (rx.getData(0),rx.getData(1))*(Vcc/0.01/1023);
for (n = 0; n < amostras; n++)
{
temperatura_1 =temperatura_1+temperatura_1_1 ; // temperatura dos mosfets
}
temperatura_1=(temperatura_1/amostras);
//==================================================Tensao
float r13=995000;//MEDIDO X TEORICO(1M ohm)
 float r17=99600;//MEDIDO X TEORICO(100k ohm)
 float r12=998000;//MEDIDO X TEORICO(1M ohm)
 float r16=99900;//MEDIDO X TEORICO(100k ohm)
 float r11=999700;//MEDIDO X TEORICO(100k ohm)
 float r15=99900;//MEDIDO X TEORICO(10k ohm)
 float relacao_1=((r13+r17)/r17);
 float relacao_2=((r12+r16)/r16);
 float relacao_3=((r11+r15)/r15);
tensao_bat_1 = combina (rx.getData(2),rx.getData(3))*(11)*(Vcc/1023);
tensao_motor_1 = combina (rx.getData(4),rx.getData(5))*(11)*(Vcc/1023);
//================================================== Corrente
corrente_bat_1 = combina (rx.getData(6),rx.getData(7))*(Vcc/0.02/1023);
corrente_motor_1 = combina (rx.getData(8),rx.getData(9))*(Vcc/0.02/1023);
//==================================================
}
}
}
//==================================================
void tx_motor_direito()
{
mapeamento();
//=========================== parado
if (x==0 && y==0)
{
payload[0] = 0 ;
payload[1] = 0;
}
 //=========================== frente
if (x==0 && y>0)
{
payload[0] = y ;
payload[1] = 0;

}
//=========================== re
else if (x==0 && y<0)
{
y=-y;
payload[0] = 0 ;
payload[1] = y;

}
//=========================== frente direita
else if (x>0 && y>0)
{
payload[0] = x ;
payload[1] = 0;

}
//=========================== frente esquerda
else if (x<0 && y>0)
{
payload[0] = y ;
payload[1] = 0;

}
//=========================== re direita
else if (x>0 && y<0)
```

```
{
y=-y;
payload[0] = 0 ;
payload[1] = y;

}
//========================== re esquerda
else if (x<0 && y<0)
{
x=-x;
payload[0] = 0 ;
payload[1] = y;

}
//========================== direita
else if (x>0 && y==0)
{
 payload[0] = 0 ;
payload[1] = x;

}
//========================== esquerda
else if (x<0 && y==0)
{
x=-x;
payload[0] = x ;
payload[1] = 0;
}

xbee.send(zbTx_2);//========================== Envia pacote
}

//=============================================================================================
void rx_motor_direito()
{
  xbee.readPacket();
if  (xbee.getResponse().isAvailable())
{
xbee.getResponse().getApiId() == ZB_RX_RESPONSE;
xbee.getResponse().getZBRxResponse(rx); // Extrai pacote de retorno
XBeeAddress64 addr64_2 = rx.getRemoteAddress64();
addr64_2.getMsb();
addr64_2.getLsb();
if (addr64_2.getLsb() == 0x409828d0)
 {

//============================================================ Temperatura
int temperatura_2_2 = combina(rx.getData(0),rx.getData(1))*(Vcc/0.01/1023);
for (n = 0; n < amostras; n++)
{
temperatura_2 =temperatura_2+temperatura_2_2 ; // temperatura dos mosfets
}
temperatura_2=(temperatura_2/amostras);
//=================================================== Tensao
float r13=995000;//MEDIDO X TEORICO(1M ohm)
 float r17=99600;//MEDIDO X TEORICO(100k ohm)
 float r12=998000;//MEDIDO X TEORICO(1M ohm)
 float r16=99900;//MEDIDO X TEORICO(100k ohm)
 float r11=999700;//MEDIDO X TEORICO(100k ohm)
 float r15=99900;//MEDIDO X TEORICO(10k ohm)
 float relacao_1=((r13+r17)/r17);
 float relacao_2=((r12+r16)/r16);
 float relacao_3=((r11+r15)/r15);
tensao_bat_2 = combina (rx.getData(2),rx.getData(3))*(11)*(Vcc/1023);
tensao_motor_2 = combina (rx.getData(4),rx.getData(5))*(11)*(Vcc/1023);
//=================================================== Corrente
corrente_bat_2 = combina (rx.getData(6),rx.getData(7))*(Vcc/0.02/1023);
corrente_motor_2 = combina (rx.getData(8),rx.getData(9))*(Vcc/0.02/1023);
//===================================================
}
}
}
//======================================== função que extrai 10 bits do Payload
int combina(unsigned int x_high, unsigned int x_low)
{
  int combined;
```

```
  combined = x_high;
  combined = combined*256;
  combined |= x_low;
  return combined;
}
//========================== Função Step para incrementar a velocidade de 10 em 10%
void incrementa_velocidade (int m )
{
int var;
int n = m*(255/10);

if (n<0)
{
var = -n;
payload[0] = 0;
payload[1] = var;

xbee.send(zbTx);//Envia pacote ZB para o modulo1
xbee.send(zbTx_2);//Envia pacote ZB para o modulo1
}
else
{
var=n;
payload[0] = var;
payload[1] = 0;
xbee.send(zbTx);//Envia pacote ZB para o modulo1
xbee.send(zbTx_2);//Envia pacote ZB para o modulo1
}
}
//=============================================função para desligar os propulsores
void desliga()
{
payload[0] = 0;
payload[1] = 0;

xbee.send(zbTx);
xbee.send(zbTx_2);
variavel=0;
}
//========================================== função para o controle da luz de fundo
void controle_luz (int m )
{
if (m==2)
{
p= map(analogRead(luz),0,1023,255,0);
}
else
{
p =m*255;
}
analogWrite(Display,p);
}
//=======================================   função para o controle do contraste
void controle_contraste(int m )
{
int a = m*(255/100);
analogWrite(contraste,a);
}
//========================================= função para gerar a barra gráfica no LCD

void bargraph(double graph)
{
unsigned char vb;
lcd.setCursor(0,1);
double va=lenght/100*graph;
// Desenhando retangulos preto no LCD
if (va>=1) {
for (int i=1;i<=va;i++)
{
lcd.write(4);
  vb=i;
}
va=va-vb;
}
int parte=va*5;
// Desenhando a coluna
```

```
switch (parte) {
case 0:
break;
case 1:
lcd.print((char)0);
break;
case 2:
lcd.write(1);
break;
case 3:
lcd.write(2);
break;
case 4:
lcd.write(3);
break;
}
//clearing line
for (int i =0;i<(lenght-vb);i++) {
lcd.print(" ");
}
}
```