UNIVERSIDADE

Estácio de Sá

RODRIGO DUQUE ESTRADA DA SILVA D'ALMEIDA

CONTROLADOR DE VELOCIDADE REVERSÍVEL PARA MOTORES DC COM ESCOVA

Rio de Janeiro

2014

RODRIGO DUQUE ESTRADA DA SILVA D'ALMEIDA

CONTROLADOR DE VELOCIDADE REVERSÍVEL PARA MOTORES DC COM ESCOVA

Trabalho de conclusão de curso apresentado à Universidade Estácio de Sá como requisito parcial para a conclusão do curso de Graduação em Engenharia Elétrica Enf. em Computação.

ORIENTADOR: Prof. André Sarmento Barbosa

Rio de Janeiro

2014

RODRIGO DUQUE ESTRADA DA SILVA D'ALMEIDA

CONTROLADOR DE VELOCIDADE REVERSÍVEL PARA MOTORES DC COM ESCOVA

Trabalho de conclusão de curso apresentado à Universidade Estácio de Sá como requisito parcial para a conclusão do curso de Graduação em Engenharia Elétrica Enf. em Computação.

Aprovada em 24/05/2014

BANCA EXAMINADORA

_____

Prof. MSc. André Sarmento Barbosa
Universidade Estácio de Sá

_____

Prof. MSc. Gilberto Rufino de Santana
Universidade Estácio de Sá

_____

Prof. MSc. Guibson Maria Diniz Navas
Universidade Estácio de Sá

_____

Prof. MSc. Júlio Cezar de Oliveira Medeiros
Universidade Estácio de Sá

A todos que me deram apoio e incentivo ao longo desta longa jornada.

AGRADECIMENTOS

A todos os familiares e amigos que me apoiaram ao longo do curso, à Banca Avaliadora, ao Professor Gilberto Rufino, ao meu orientador Prof. André Sarmento pela ajuda em todo o processo, aos professores Robson Moraes, Antônio Carlos Castañon, Júlio Cesar Medeiros e William Roger pelos conhecimentos adquiridos durante o curso e usados nesse projeto. À equipe de Robótica RioBotz, e todos os seus integrantes, a qual tenho orgulho de ter feito parte e ter tido além de experiências acadêmicas, experiência de vida. Aos Professores Marco Antonio Meggiolaro, Mauro Schwanke da Silva e Mauro Esperanza, por todo conhecimento prático adquirido por todos esses anos e a todos aqueles que me ajudaram diretamente neste projeto: Eduardo von Ristow, Felipe Maimon, Matheus Marques e Michel Feinstein. Agradecimentos especiais a Alexandre Ormiga, Guilherme Rodrigues e Guilherme Porto pelos conhecimentos passados sobre rádios.

RESUMO

Este projeto consiste em um controlador de velocidade variável, reversível para motores DC de 12V até 50V, que suporte corrente contínua de até 120A, e que seja operado por rádios padrão PPM ou dispositivos que produzam tal sinal. O controlador é fisicamente dividido em dois módulos: o módulo de controle e o módulo de potência. O módulo de controle recebe o sinal PPM, converte em sinais PWM para serem enviados ao módulo de potência também faz o controle de corrente e interrompe a transmissão dos sinais PWM em caso de perda de sinal. O módulo de potência é onde se encontram os drivers de controle, opto-acopladores, os MOSFET's de potência e o regulador de tensão chaveado. Este módulo é opto-acoplado, o que significa que a parte de controle é fisicamente isolada da parte de potência, isolando assim a parte do circuito de potência, que utiliza alta tensão e corrente da parte de controle que utiliza baixa tensão e corrente. Ser modular permite utilizar os módulos em conjunto ou separado em outros projetos, dando mais opção de escolha a quem pretende usar o controlador em um projeto de engenharia.

Palavras-chave: Controlador de velocidade. Modular. MOSFET. PPM. PWM. Arduino. Ponte H.

ABSTRACT

This project consists of a variable speed controller , for 12V DC motors up to 50V, supporting currents up to 120A , and is operated by radio using PPM or using devices that produce the same signal. The controller is physically divided into two modules: the control module and the power module. The control module receives the PPM signal , converts it into PWM signals to be sent to the power module also makes the current control and stops the transmission of PWM signals in case of signal loss . The power module is where the drivers control opt couplers , the power MOSFET 's and the voltage regulator switched . This module is opt coupled, meaning that the control part is physically isolated from the power part , thus isolating the power circuit part which uses high voltage and current, of the control part that uses low voltage and current. Being modular allows using modules together or separately in other projects , giving more options of choose who you want to use the controller in an engineering project .

Keywords: Speed controller. Module. MOSFET. PPM. PWM. Arduino. H Bridge

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

DC – *Direct Current.*

AC – *Alternate Current.*

PPM – *Pulse Position Modulation.*

PWM – *Pulse With Modulation.*

FET – *Field Effect Transistor.*

SCR – *Silicon Controlled Rectifier.*

MOSFET – *Metal Oxide Semiconductor Field Effect Transistor.*

LED – *Light Emitting Diode.*

TVS – *Transient Voltage Suppression.*

ALU – *Aritimetic Logical Unit.*

IGBT – *Insulated Gate Bipolar Transistor.*

MCU – *Micro Controller Unit.*

CI – *Circuito integrado.*

I/O – *Input / Output.*

ROM – *Read Only Memory.*

RAM – *Random Access Memory*

EEPROM – *Electrically-Erasable Programmable Read-Only Memory*

SUMÁRIO

1 – INTRODUÇÃO

As aplicações tanto na indústria como na robótica necessitam de motores e atuadores para a realização de tarefas. Os motores de corrente contínua são utilizados em diversas aplicações de automação e robótica.

Na indústria os motores, tanto AC como DC, podem ser usados no controle de esteiras em linhas de produção assim como em processos automáticos de embalagem do produto, o que requer muitas vezes um controle de velocidade e sentido do motor. Motores de corrente contínua também são utilizados em robôs tele operados como robôs de exploração submarina (ROV) figura 1.1, robôs de inspeção em lugares perigosos ou de difícil acesso aos seres humanos figura 1.2, robôs de desarmamento de bombas figura 1.3, robôs de inspeção com câmeras figura 1.4 dentre outros. Todas essas aplicações descritas acima utilizam motores que necessitam de um controle de velocidade e sentido para sua completa operação.

Figura 1.1 - ROV da empresa Subsea 7.

Figura 1.2 - Robô ambiental da empresa Petrobrás.

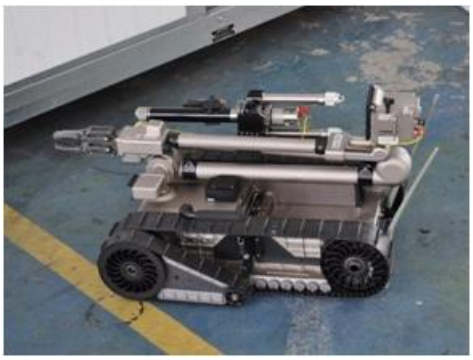


Figura 1.3 - Robô anti-bombas comprado pelo governo brasileiro durante a Copa das Confederações.

Figura 1.4 - Robô de inspeção da empresa Sondeq.

2 – FUNCIONAMENTO DE MOTORES DC COM ESCOVA

Motores DC com escova ou motores de corrente contínua (CC), são dispositivos que se utilizam das forças de atração e repulsão de ímãs para gerar movimento.

Para gerar movimento o motor precisa gerar um campo magnético através de suas bobinas. Para isso é preciso aplicar uma tensão às bobinas, o que produz uma força de atração e repulsão entre a bobina e os ímãs gerando torque, fazendo as bobinas e o eixo (rotor) girarem ao ponto em que a bobina que estava em um pólo seja repelida ao pólo oposto onde o sistema fica em equilíbrio. Para manter um movimento constante um motor precisa de um comutador, para trocar os pólos da bobina continuamente, mantendo sempre uma força de repulsão entre as bobinas e os ímãs figura 2.1.

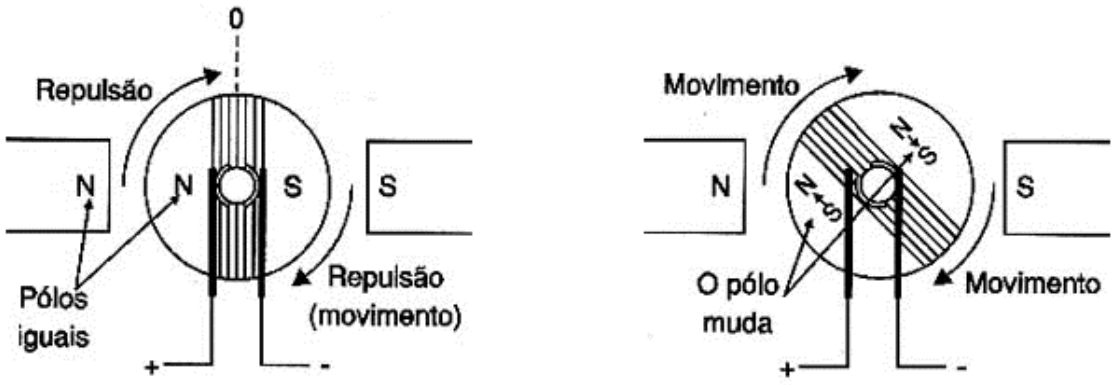


Figura 2.1 – Funcionamento de motor DC.

A velocidade do motor depende da diferença de potencial aplicada às bobinas. Quanto maior a tensão aplicada, mais intensamente as bobinas e os imãs vão se repelir, fazendo o motor girar mais rápido. O Sentido de rotação depende da polaridade da tensão aplicada às bobinas.

A corrente consumida pelo motor DC é maior quanto mais esforço ele fizer para manter ou iniciar o movimento, sendo a menor corrente possível quando o motor gira livremente (corrente sem carga), e a maior corrente possível quando o motor recebe tensão, mas não se move (corrente de stall). [1]

2.1 – FORMAS DE ACIONAMENTO

Existem várias formas de se acionar um motor DC figura 2.2. A mais simples é ligá-lo direto a uma fonte de tensão contínua. A menos que se esteja apenas testando o motor não é a melhor maneira de se controlá-lo em uma aplicação.

Figura 2.2 - Motor DC da empresa Maxon.

Para aplicações simples pode-se acionar o motor através de interruptores, relés figura 2.3, solenóides ou outros dispositivos mecânicos que fechem o circuito fazendo a corrente passar pelo motor. Porém nesse método não é possível o controle nem da velocidade nem do sentido do motor. Pode ser utilizado tanto em motores AC como DC. Como exemplo temos algumas máquinas da indústria que usam motores AC, como a serra circular, a esmerilhadeira, o esmeril, a furadeira de bancada entre muitas outras. Como exemplos de aplicação na indústria automobilística que usam motores DC têm os motores da ventoinha do radiador e da bomba que circula a água pelo radiador dos veículos.

Figura 2.3 - Relé automotivo

Quando há a necessidade de controle do sentido, pode-se utilizar os mesmos elementos descritos acima numa configuração em que se possa alterar a polaridade do motor. Normalmente é feito com interruptores figura 2.4. Utilizando um interruptor de duas posições em configuração cruzada figura 2.5 é possível reverter o sentido do motor de forma totalmente mecânica. Ainda assim a velocidade do motor não varia nessa configuração. Esse tipo de configuração pode ser usado para o acionamento de um portão de garagem onde a velocidade pode ser contínua, mas é necessário o controle do sentido.

Figura 2.4 - Interruptor deslizante duas posições.

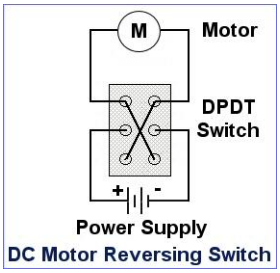


Figura 2.5 - Esquema de reversão de polaridade.

Quando se deseja o controle de velocidade de um motor, é necessário fazer a tensão variar ou limitar a corrente que circula nele. Uma maneira simples de variar a velocidade de um motor é utilizar um reostato figura 2.6, é uma resistência variável que suporta potências maiores que os potenciômetros e que limitará, dependendo do ajuste, mais ou menos a corrente que circula pelo motor variando sua velocidade. Porém, esse método não é muito prático em certas aplicações, pois o reostato tem que dissipar uma potência que, dependendo do motor, pode ser muito elevada exigindo reostatos maiores e mais pesados. Uma aplicação comum para os reostatos é controlar a velocidade de ventiladores.

Figura 2.6 - Reostato de potência

Outra possibilidade de controlar a velocidade de um motor é a utilização de um transistor em conjunto com um potenciômetro e um resistor fixo figura 2.7. A grande vantagem é que dessa forma a potência dissipada no potenciômetro é mínima, cabendo ao transistor suportar a corrente utilizada pelo motor, que torna o circuito bem menor. O resistor fixo serve para não deixar o valor mínimo de resistência ser zero e para que a corrente de base do transistor não seja alta demais. O potenciômetro serve para variar a tensão de base fazendo com que a corrente do emissor ao coletor varie. Os transistores podem ser do tipo NPN e PNP figura 2.8. [2]

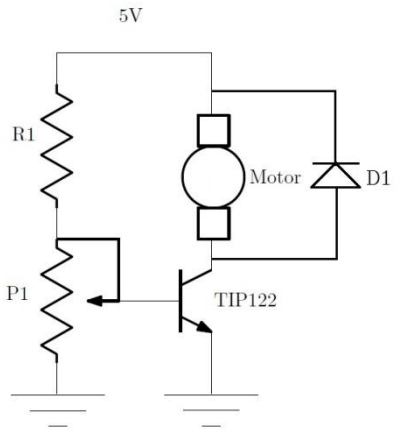


Figura 2.7 - Controlador de velocidade com transistor.

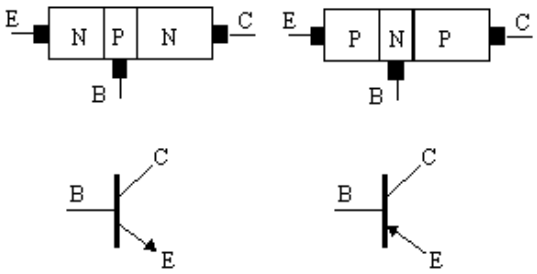


Figura 2.8 - Representação de transistores NPN e PNP.

As equações 1 e 2 dão as correntes e tensões respectivamente no terminais dos transistores:

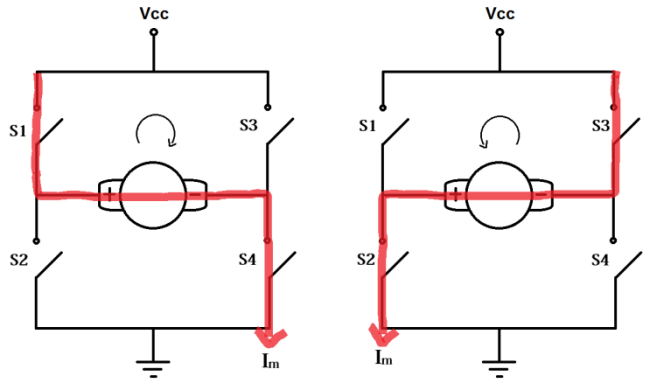
$$I_E = I_C + I_B \quad (1)$$

$$V_{CE} = V_{BE} + V_{CB} \quad (2)$$

2.2 – FUNCIONAMENTO DA PONTE H

Quando há a necessidade de um controle completo do motor, ou seja, sentido de rotação e velocidade, é utilizado a técnica da ponte H.

A ponte "H" consiste em quatro elementos em uma configuração que lembra a letra H (figura 2.9). Uma analogia de fácil entendimento é considerar esses elementos como chaves que abrem e fecham o circuito definindo o caminho que a corrente circula pelo motor. Considerando que somente uma chave de cada lado pode ser acionada por vez, para evitar curto-circuito, temos quatro estados possíveis a serem considerados.



Estados possíveis

S1	S2	S3	S4	Sentido
1	0	1	0	Parado
1	0	0	1	Horário
0	1	0	1	Parado
0	1	1	0	Anti-Horário

Figura 2.9 – Funcionamento da ponte H.

Uma vez definido o funcionamento da ponte H é preciso substituir os interruptores por um dispositivo que varie a tensão ao invés de conectar e desconectar a alimentação da rede. Para essa função se utiliza o transistor que consegue limitar a corrente ou variar a tensão ligando e desligando rapidamente, dependendo do tipo, para controlar a velocidade do motor. Os tipos de transistor e suas características serão explicados em outro capítulo, por ora o importante é o conceito por traz da ponte H e seu funcionamento.

Na figura 2.10 é possível ver uma ponte H com transistores NPN.

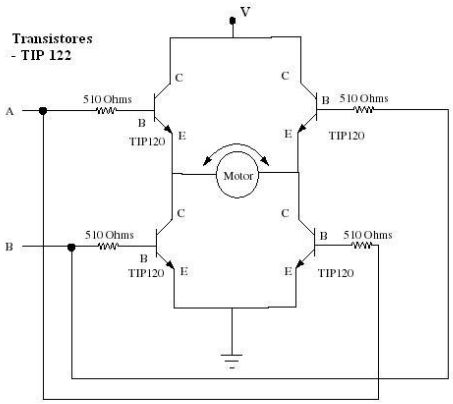


Figura 2.10 – Ponte H com transistores TIP 122 do tipo NPN.

Neste caso os transistores estão agrupados em pares opostos, como mostrado no sentido de condução da figura 2.10. Quando aplicamos uma tensão de base em A o motor gira em um sentido e com velocidade que depende da tensão de entrada, e em B para o sentido oposto e com velocidade também dependendo da tensão de entrada. Porém, se for aplicada uma tensão em A e B ao mesmo tempo, os transistores fecham o circuito entre os dois terminais da fonte de tensão provocando curto circuito. Essa é uma condição proibida de acionamento do circuito, pois provoca a destruição do mesmo.Para evitar que aconteça, deve-se se certificar que quando uma porta recebe tensão a outra deve ficar desativada.

Outra configuração possível é a meia ponte H (fig. 2.11), que corresponde ao controle da velocidade de apenas um sentido do motor e normalmente é empregada onde não há a necessidade de reverter o sentido de rotação, mas deseja-se controlar a velocidade. [3]

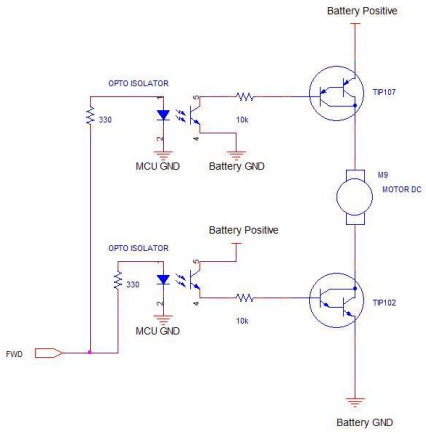


Fig.2.11 – Meia ponte H.

2.3 – PWM - MODULAÇÃO POR LARGURA DE PUSLSO

Uma das formas de se variar a tensão de acionamento do motor é a modulação por largura de pulso que vem da sigla em inglês PWM (*Pulse With Modulation*). Esse método consiste em ligar e desligar um dispositivo a uma frequência fixa entre as mudanças de estado. O *duty-cycle*, ou ciclo ativo, é o intervalo em que o dispositivo fica em estado lógico 1, ou seja ligado, e nos dá a porcentagem da tensão de saída do dispositivo como pode ser visto na figura 2.12.

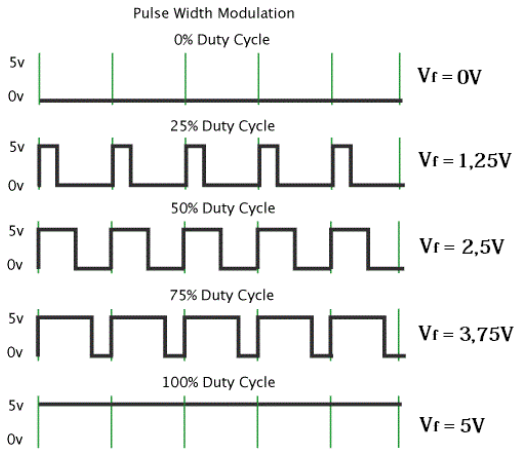


Figura 2.12 – Sinais PWM.

Para poder ligar e desligar um motor rapidamente normalmente são utilizados dispositivos de estado sólido como transistores bipolar, FET de potência, SCR, entre outros. Esses componentes semicondutores têm a capacidade de ligar e desligar muito mais rapidamente se comparados com dispositivos eletromecânicos como relés e solenóides. Quanto maior a frequência de atualização do PWM, mais rapidamente o dispositivo varia sua tensão dando uma resposta mais natural. A figura 2.13 mostra um sinal PWM que gera uma senóide. [4]

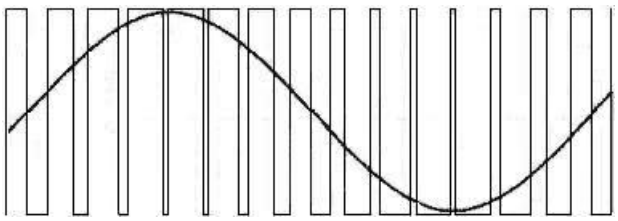


Figura 2.13– Senóide gerada por PWM.

3 – CIRCUITO DE POTÊNCIA:

A Eletrônica de potência é a área da eletrônica que trata do processamento da energia elétrica e visa obter maior eficiência e qualidade. Pode ser empregado no controles de sistemas enormes com milhares de volts até sistemas eletrônicos bem pequenos de baixa tensão. Os métodos que são empregados em eletrônica de potência baseiam-se na utilização de dispositivos semicondutores operados em regime de chaveamento, para realizar um controle no fluxo de energia e a conversão de formas de onda de tensões e correntes entre fontes e cargas. [5]

Especificamente para este projeto (figura 3.1), a finalidade do módulo de potência é o controle da tensão e corrente enviadas a um motor DC para que se possa controlar a velocidade e o sentido de rotação, dando total liberdade de movimentação ao motor. Nesse capítulo serão apresentados os componentes do circuito de potência, suas funções e o motivo da escolha de cada componente.

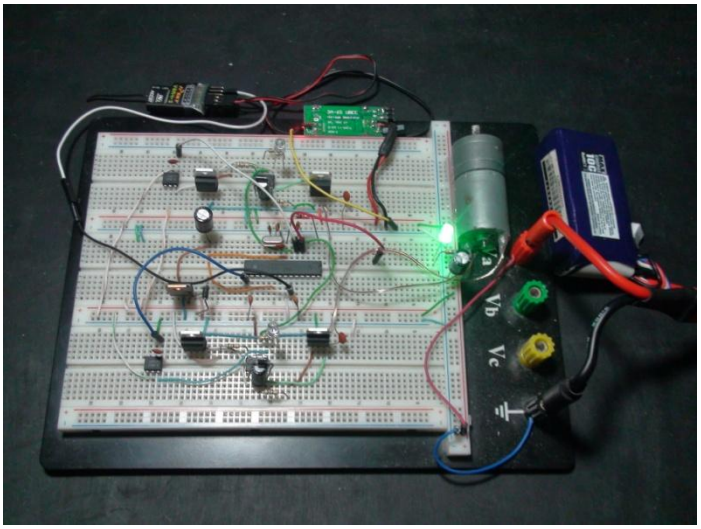


Figura 3.1 – Circuito de potência montado em *protoboard*.

3.1 – TRANSISTORES

Na atualidade de forma geral circuitos de potência para controle de velocidade de motores ou utilizam IGBT's (figura 3.3) ou transistores MOSFET (figura 3.2). Ambos controlam a velocidade por chaveamento sendo o IGBT, de uma forma geral, empregado em tensões maiores que 1000V e os MOSFET's em tensões abaixo de 250V. Mesmo sendo empregado em tensões maiores o IGBT possui em geral uma freqêuncia de chaveamento menor que os MOSFET's.

Figura 3.2 - Transistor IGBT

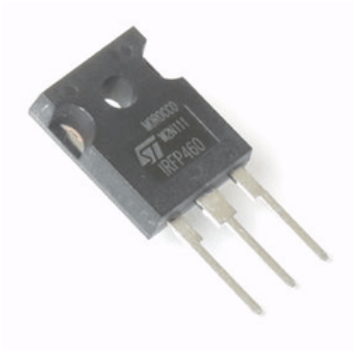


Figura 3.3- Transistor MOSFET

A taxa de chaveamento em hertz é a velocidade na qual o componente conduz e deixa de conduzir pois é o tempo que ele fica conduzindo em função do tempo que ele fica sem conduzir e nos dá uma variação de tensão PWM como visto no capítulo 2.

Como o objetivo do projeto é projetar um circuito que consiga acionar um motor de até 50,4V, com variação de velocidade e sentido que suporte correntes de até 120A, e que possua duty-cycle de frequências da ordem dos kHz a melhor opção é o transistor do tipo MOSFET, que será visto em detalhes a seguir.

Um transistor de efeito de campo metal-óxido ou MOSFT, é um amplificador de tensão e não de corrente como ocorrem nos transistores bipolares. Enquanto a corrente de coletor de um transistor comum é função da corrente de base, em um transistor de efeito de campo, a corrente de dreno é função da tensão de comporta variando a tensão de saída. Um esquema representativo pode ser visto na figura 3.4.

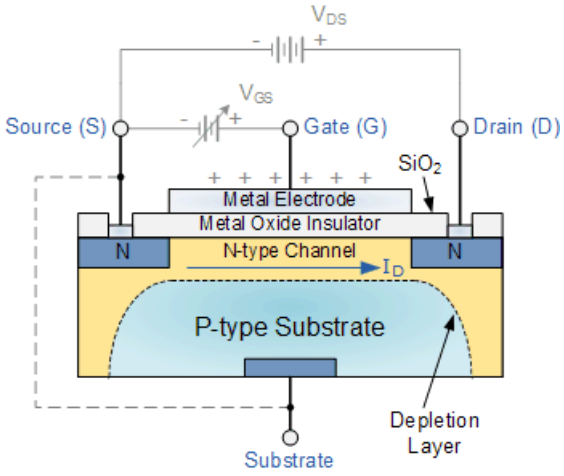


Figura 3.4 - MOSFET canal P.

Uma película fina de óxido de metal isola a região de comporta da região do canal que liga o dreno ao source. Dependendo da polaridade usada nos materiais semicondutores podemos ter MOSFET de canais N ou P como visto na figura 3.5. [6]

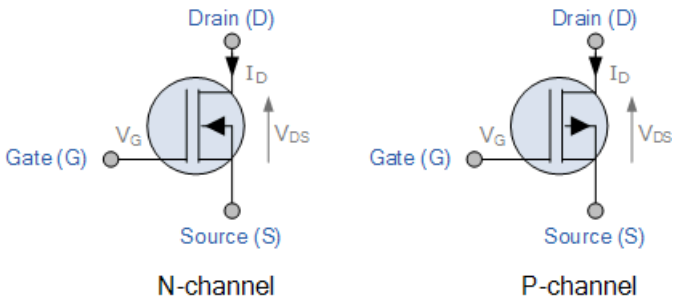


Figura 3.5 - MOSFET canal P e canal N.

Características de projeto

Para se escolher um transistor para um projeto de ponte H é preciso levar em conta diversos fatores como: Tensão máxima, corrente máxima suportada, resistência interna e potência máxima dissipada.

A tensão máxima suportada mostra qual a maior tensão que o MOSFET consegue chavear o motor. Mas é importante levar em conta que nunca devemos trabalhar na tensão máxima do MOSFET sempre sendo prudente considerar uma margem de segurança, pois os dados dos componentes são fornecidos em condições ideais que não ocorrem na prática. Sendo assim é prudente usar uma margem de 50 a 75% da tensão total estipulada pelo fabricante.

A corrente máxima suportada pelo MOSFET é a corrente máxima que seu encapsulamento suporta sem ser danificado, e assim como a tensão, é importante considerar uma margem de segurança pois o fabricante considera condições ideais, sendo que na prática existem perdas que podem comprometer o funcionamento ideal do componente, fazendo que ele não suporte a referida corrente. Portanto uma margem em torno de 50 a 75% do valor máximo deve ser considerada por segurança.

A resistência interna do MOSFET influencia no quanto ele vai dissipar de potência, quanto menor a resistência interna, menor a potência em watts a ser dissipada para uma mesma corrente. Portanto, quanto menor a resistência interna, melhor a dissipação de calor.

A potência máxima dissipada determina qual a máxima potência em watts pode ser dissipada do MOSFET sem danificar o mesmo. A potência dissipada é dada pela equação 3. É importante considerar a corrente máxima pretendida e a resistência interna para poder determinar qual a potência a ser dissipada. [7]

$$2\,(I)^2\,R \qquad (3)$$

Onde **I** é a corrente em ampares e **R** a resistência em ohms.

Dependendo da potência dissipada os MOSFETS vão necessitar de um dissipador, ou até mesmo um dissipador combinado com ventoinha para não aquecerem demais danificando eles mesmos e os demais componentes da placa. [8]

Tendo em vista as características de um MOSFET e o que se deseja para o projeto, o componente escolhido é o IRFS3107 da International Rectifier (figura 3.6). [9]

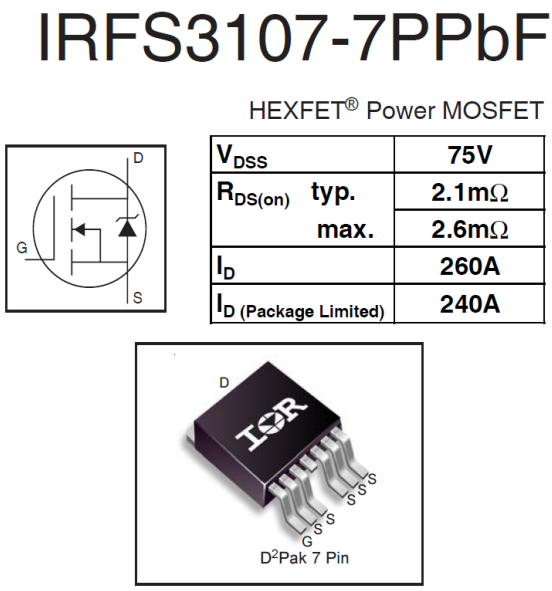


Figura 3.6 - MOSFET IRFS3107.

Esse MOSFET possui uma tensão máxima de 75V, e como se pretende uma tensão máxima de 50.4V, será utilizado aproximadamente 62,2% da tensão máxima, ainda dentro dos 75% recomendados.

A corrente máxima suportada por esse transistor é de 260A, mas vale notar que o fabricante informa que o limite suportado pelo encapsulamento de plástico é 240A. Como se pretende uma corrente contínua de 120A será utilizado apenas 50% da corrente máxima do MOSFET.

A resistência interna é de apenas 2.6mΩ, que é baixa, fazendo o componente dissipar energia de forma mais eficiente. Sendo assim, usando a fórmula 3 temos:

$$2\ (120A)^2\ 0{,}0026\Omega \qquad (3)$$

Que nos dá 74,88W de potência máxima dissipada, o que pode parecer muito, mas que pode ser dissipado com um dissipador metálico adequado e uma ventoinha, pois ainda está abaixo do valor dissipado pela maioria dos processadores atuais.

3.2 – DRIVER DE CONTROLE E OPTO-ACOPLADORES

Para que se possa usar apenas MOSFETS de um único tipo (P ou N) é preciso de um *driver* que gerencie o acionamento para um correto funcionamento da ponte H. Além de permitir a utilização de MOSFETS de mesmo canal, sendo o mais comum a utilização dos de canal N. Os *drivers* modernos possuem outra vantagem pois bloqueiam a condição proibida não deixando os transistores errados acionarem ao mesmo tempo.

Tendo em vista as característica do MOSFET escolhido o *driver* selecionado para acionar a ponte H é o IRS2184 também da International Rectifier (figura 3.7). [10]

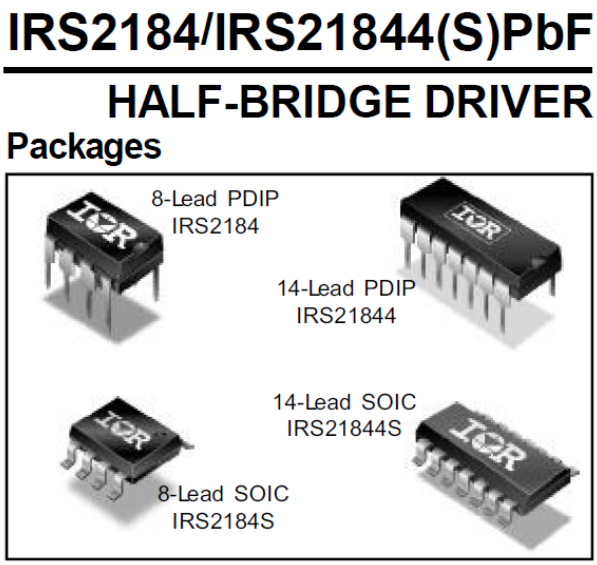


Figura 3.7 - Driver IRS2184.

Esse *driver* pode trabalhar com uma grande gama de tensões como 3.3V, utilizada nos processadores da família ARM, e 5V utilizados em diversos micro processadores, dando a ele uma grande variedade de opções.

A tensão de trabalho do *gate* é de 10 a 20V, e como se deseja uma tensão mínima de 12V não há problema desde que se use um regulador que mantenha a tensão próxima dos 12V os *drivers* ficarão devidamente alimentados. Como a tensão máxima de alimentação dos MOSFETS permitida pelo *driver* é 600V, fica dentro dos 50.4V de tensão máxima requerida.

Cada *driver* controla apenas meia ponte como visto na figura 3.8. Sendo assim, são necessários dois *drivers* para controlar uma ponte completa. Para que não haja problemas é necessário que o software de controle acione apenas os MOSFETS da parte alta de um lado e da parte baixa do outro e vice versa para não haverem curtos-circuitos , mas em caso de serem acionados os dois lados ao mesmo tempo, os *drivers* possuem uma proteção desligando o seu lado da ponte.

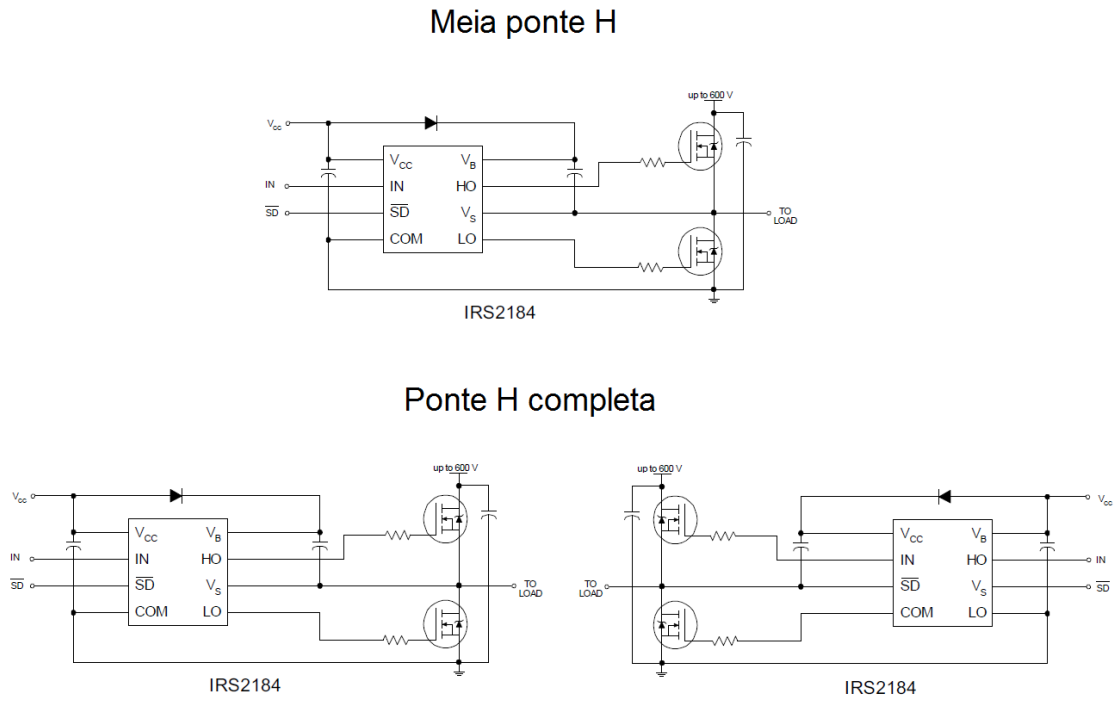


Figura 3.8- Ponte H com o IRS2184.

Os opto-acopladores possuem um papel muito importante no circuito pois eles interligam a parte de sinal à parte de potência, e são o intermédio entre o microprocessador e o *driver*. Assim é como se o microcontrolador controlasse apenas leds e o *driver* reconhecesse os comandos e fizessem os transistores controlarem a parte que demanda mais potência do circuito. Dessa forma podemos minimizar as chances de danos ao microprocessador separando as partes de baixa corrente e tensão das partes de alta tensão e corrente do circuito como visto na figura 3.9.

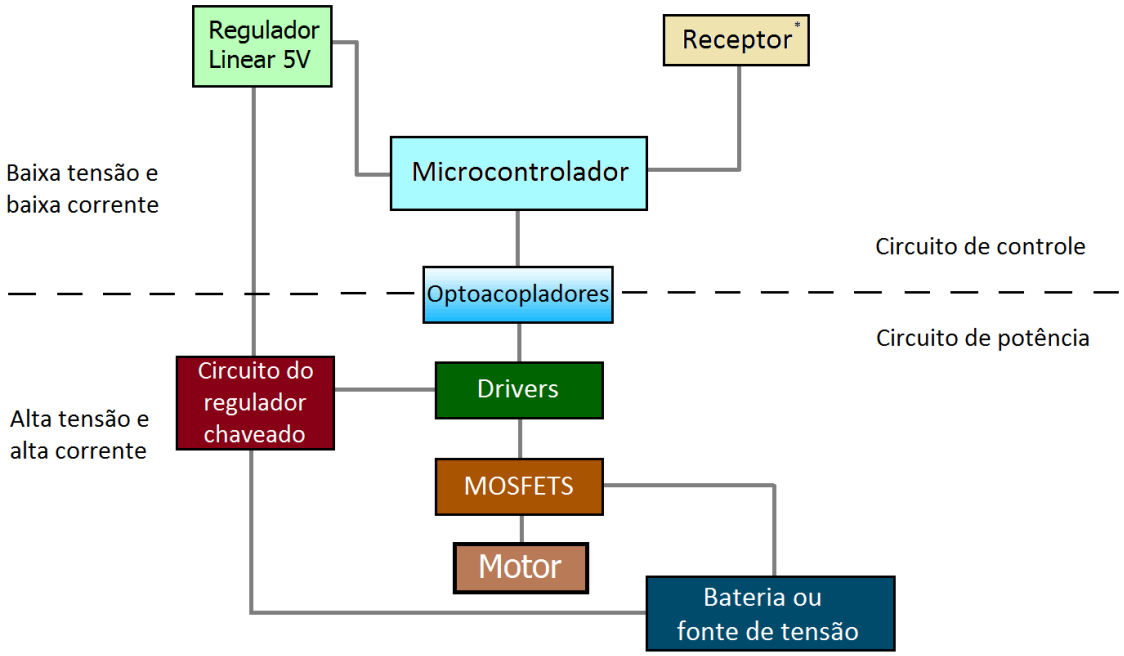


Figura 3.9 - Divisão entre circuito de controle e potência.

O opto-acoplador escolhido é o 4N25 (figura 3.10) por ser bem comum, possuir as características necessárias e funcionar com tensões bem baixas. [11]

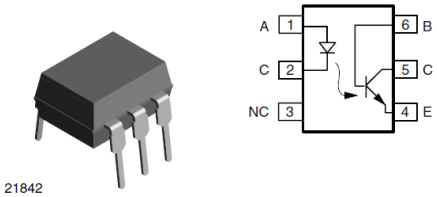


Figura 3.10 - Opto-acoplador 4N25.

3.3 – ACIONAMENTO DE PONTE H

O pino IN é a entrada de sinal para as partes alta (HO) e baixa (LO) da ponte, sendo que está em fase com a parte alta da ponte. O pino SD também é entrada de sinal, mas controla a parte baixa da meia ponte H. Para o acionamento da ponte o microcontrolador manda sinal para 4 pinos que são capazes de acionar os dois lados da ponte e interromper seu funcionamento. Um pino PWM que manda sinal pelo SD para cada lado da ponte, para que se possa controlar a velocidade no lado da ponte desejado e um pino de *enable* que, que serve para habilitar ou desabilitar o lado da meia ponte utilizado a ponte como visto na figura 3.11.

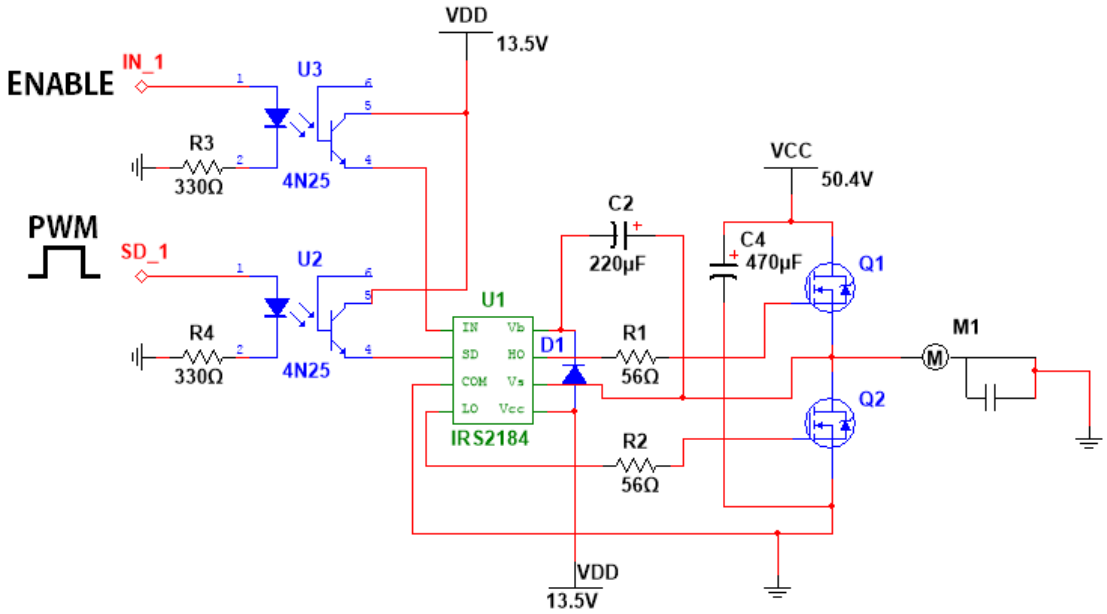


Figura 3.11 - Acionamento de meia ponte H.

O sinal PWM é enviado para o opto-acoplador, e a saída do sinal é posteriormente enviada ao pino de entrada do *driver,* que por sua vez interpreta o sinal e varia a tensão dos MOSFET's de acordo com o PWM. Para ativar um lado da ponte basta enviar um sinal lógico 1 no pino IN que neste caso é de 5V por ser a tensão de trabalho do microcontrolador e estar dentro do especificado pelo fabricante segundo o *datasheet* do componente. Para desativar a ponte basta enviar um sinal lógico 0 no pino IN que corresponde a 0V. Uma característica comum à esse *driver* e a vários outros *drivers* de MOSFET é que o sinal máximo suportado do PWM é de 99%. Portanto, se o sinal extrapolar

esse valor e for a 100% o *driver* simplesmente não manda sinal para os MOSFET's logo é preciso via programação limitar o sinal a 99% ou menos.

É importante lembrar que meia ponte H controla apenas um sentido de rotação do motor, para se ter controle de reversão de sentido e velocidade é preciso de uma ponte H completa que nada mais é do que duas meia ponte H.

Quando o lado esquerdo por exemplo recebe o sinal PWM no pino SD e um sinal lógico 1 no pino IN do *driver* e o outro lado recebe um sinal lógico 1 no pino SD e um sinal lógico 0 no pino IN do outro *driver*. A corrente I1 passa pelo transistor Q1 da parte alta (HO) e vai até o transistor da parte baixa (LO) Q4 passando pelo motor. Quando o motor para de receber sinal PWM (PWM = 0%) uma corrente I2, chamada corrente de *freewheeling* fica circulando na parte baixa da ponte H passando pelos transistores Q2 e Q4 através dos diodo de *freewheeling* internos dos MOSFETs. Essa corrente flui apenas por um instante até que o motor pare suavemente, evitando uma parada brusca que sobre aqueceria o motor e poupando a bateria de dissipar a corrente já que o motor se comporta como uma carga indutiva. O funcionamento pode ser visto na figura 3.12.

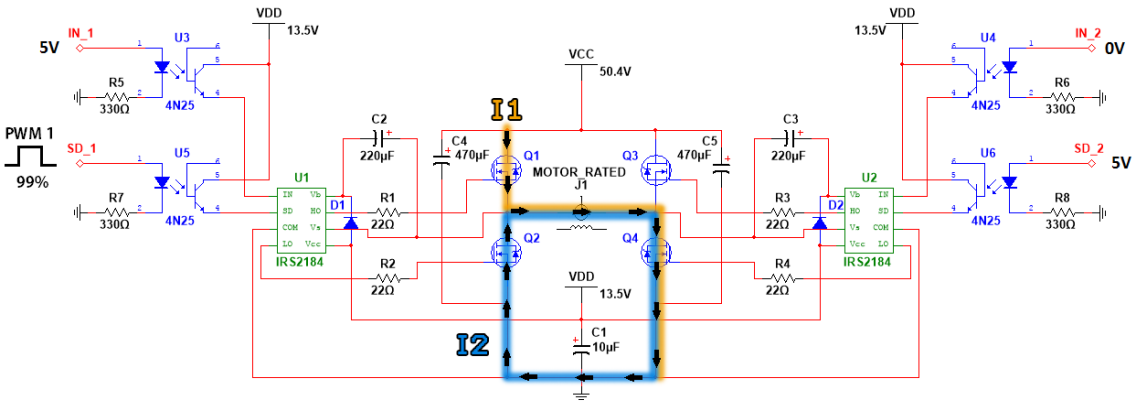


Figura 3.12 - Funcionamento da ponte H completa.

Todo o funcionamento deve ser coordenado via *software* para que quando um lado da ponte estiver mandando pulsos PWM maiores que zero o sinal IN deve ser 1 e o IN do *driver* oposto 0 para não haver curto circuito, pois caso isso ocorra os *drivers* não vão acionar os MOSFET's da maneira correta, o que pode acarretar danos ao circuito. [10]

3.4 – CIRCUITO BOOTSTRAP E TRANSIENTES

A tensão Vbs (a diferença de tensão entre os pinos VB e Vs no do driver) fornece a alimentação para o circuito na parte alta (HO) do controle do driver. O fornecimento deve estar na faixa de 10-20V para assegurar que o CI de controle funcione corretamente, fazendo com que funcionem os circuitos de detecção de subtensão para Vbs e para assegurar que o CI não deixe a tensão Vbs cair muito. Esta tensão de alimentação é uma fonte de Vbs flutuante que fica no topo da tensão Vs (que de forma geral é uma onda quadrada de alta frequência). Para garantir o fornecimento da tensão flutuante Vb pode ser usado um circuito de *bootstrap*, que consiste de um diodo um resistor e um capacitor como visto na figura 3.13. [12]

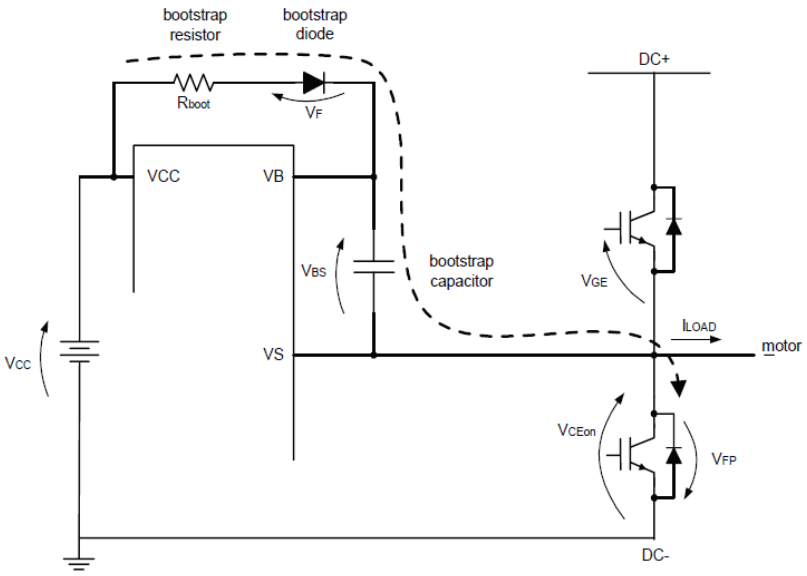


Figura 3.13 - Circuito de bootstrap

Esse método traz algumas limitações no *duty-cycle*, mas se os componentes forem devidamente dimensionados não irão ocorrer problemas no circuito.

Para calcular o capacitor de *bootstrap* precisamos primeiro resolver as equações 4 e 5 para obter o valor mínimo do capacitor de *bootstrap* na equação 6 conforme consta no *aplication note* dt04-4 da International Rectifier.

$$Q_{TOT} = Q_G + Q_{LS} + (I_{LK\text{-}GS} + I_{QBS} + I_{LK} + I_{LK\text{-}DIODE} + I_{LK\text{-}CAP} + I_{DS}) \text{ x } T_{HON} \quad (4)$$

$$\Delta V_{BS} \leq V_{cc} - V_F - V_{GSmin} - V_{DSon} \quad (5)$$

$$C_{BOOT} \geq \frac{Q_{TOT}}{\Delta V_{BS}} \quad (6)$$

Considerando os valores abaixo referentes ao MOSFET e o *driver* utilizados e que foram extraído dos seus respectivos *datasheet* temos:

I_{QBS} = 150 uA
I_{LK} = 50 uA
Q_{LS} = 38 nC
Q_g = 200 nC
$I_{LK\text{-}GS}$ = 100 nA
$I_{LK\text{-}DIODE}$ = 100 uA
$I_{LK\text{-}CAP}$ = 0 (para capacitores cerâmicos)
I_{DS} = 150 uA
T_{HON} = 100 uS

V_{CC} = 12 V
V_F = 1 V
V_{DSon} = 10 V
V_{GSmin} = 2 V

Substituindo os valores na equação 4 temos:

$$Q_{TOT} = 200 + 38 + (100 + 150000 + 50000 + 100000 + 0 + 150000) \text{ x } 10^{-4} \quad (4)$$

$$Q_{TOT} = 280{,}01 \text{ nC} \quad (4)$$

Substituindo os valores na equação 5 temos:

$$\Delta V_{BS} \leq 12 - 1 - 2 - 10 \quad (5)$$

$$\Delta V_{BS} \leq 1 \quad (5)$$

Tendo os valores de Q_{TOT} e ΔV_{BS} podemos obter o valor mínimo do capacitor de *bootstrap* através da equação 6:

$$C_{BOOT} \geq \frac{280,01}{1} \quad (6)$$

$$C_{BOOT} \geq 280,01\ nF \quad (6)$$

Tendo em vista o valor mínimo do capacitor de *bootstrap* é importante lembrar que os motores em carga consomem quantidades significativas de energia provocando quedas de tensão no circuito e podendo descarregar o capacitor de *bootstrap* rápido demais, fazendo com que o circuito não funcione de forma adequada. Para isso é preciso usar uma valor de capacitor mais elevado. Através do protótipo construído foi possível determinar uma valor ideal. Valores a cima de 22 uF se mostraram eficientes, mas colocando uma boa margem de segurança, para garantir o bom funcionamento do circuito capacitores de 220 uF se demonstraram melhores em conjunto com resistores de 22 ohm nas partes alta (HO) e baixa (LO) do *driver*, que devem ser resistores de valor baixo. [13]

Transientes

Quando uma carga repentina entra no sistema ocorre uma variação de tensão até o sistema estabilizar, esse período chamamos de transiente. Para se evitar o transiente indesejado nos componentes vitais do circuito é altamente recomendado usar capacitores de acoplamento e desacoplamento, além de ser desejável capacitores de desacoplamento de valores elevados que liguem o positivo do circuito ao terra para amenizar os transientes. Foram usados dois capacitores de 470 uF entre o positivo e negativo das duas meias ponte H para amenizar ao máximo a queda de tensão provocada, além de capacitores de desacoplamento em cada *driver* e opto-acoplador. Um diodo TVS entre os terminais do motor evita os picos de tensão transiente muito elevados protegendo o circuito de tensões acima do especificado. [14]

3.5 – REGULADOR DE TENSÃO

Apesar da fonte de tensão poder variar de 12 a 50.4V os *drivers* e opto-acopladores precisam de uma tensão que não extrapolem seus limites de funcionamento. Por esse motivo é preciso de um regulador que mantenha a tensão próxima dos 12V que é um valor adequado para os dois componentes.

Um regulador linear comum poderia no máximo regular de 30V para 12V, e ainda sim dissiparia tanto calor que precisaria de um grande dissipador. Para regular grandes tensões como 50.4V sem dissipar muito calor a melhor opção é um regulador chaveado.

Reguladores chaveados funcionam de forma mais eficiente dissipando menos calor, conseguindo regular tensões maiores e com uma eficiência em média de 85% contra no máximo 40% dos reguladores lineares.

Para regular tensões de até 50.4V para 12V foi escolhido o regulador LM2576-HV da National Semiconductor. Na figura 3.14 é demonstrado o esquema de ligação. [15]

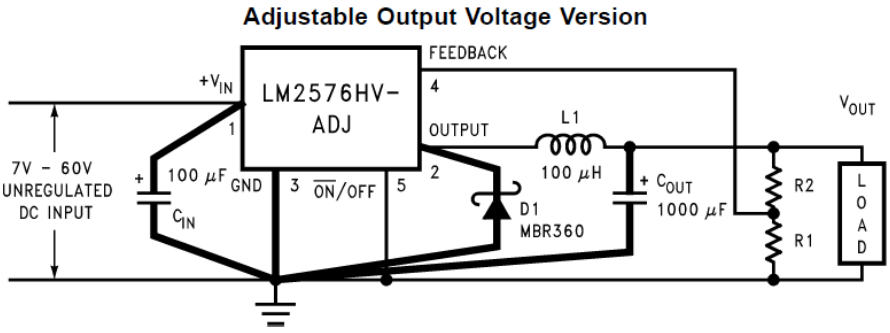


Figura 3.14 - Regulador chaveado ajustável.

Segundo o *datasheet* do componente a fórmula 7 nos dá o valor da tensão de saída em função dos resistores R1 e R2 .

$$\mathbf{R2} = \mathbf{R1}\left(\frac{\mathbf{V_{OUT}}}{\mathbf{V_{ref}}} - \mathbf{1}\right) \qquad (7)$$

Considerando V_{ref} = 1,23V, R1 = 1K e Vout = 12,5V temos o valor de R2:

$$\mathbf{R2} = \mathbf{1000}\left(\frac{\mathbf{12.5}}{\mathbf{1{,}23}} - \mathbf{1}\right) = \mathbf{9162.6\ ohms \cong 10K} \qquad (7)$$

4 – CIRCUITO DE CONTROLE

O módulo de controle é responsável por controlar a parte de potência gerenciando o funcionamento dos *drivers*, opto-acopladores e MOSFET's através de sinais PWM enviados aos opto-acopladores e o sinal de *enable* enviado aos *drivers*. O circuito de controle também é responsável por receber o sinal do rádio através da modulação PPM, interpretá-lo e convertê-lo em sinais PWM para o motor, e em caso de falha ou perda de sinal interromper seu funcionamento, como será visto mais a frente. Esse módulo é composto por um microcontrolador, um regulador de tensão 5V, capacitores de desacoplamento e um botão usado para fazer a calibragem do sinal.

A placa micro controladora escolhida para o projeto é a Arduino Nano (figura 4.1) pois é pequena, já possui interface serial FTDI e é compatível com várias bibliotecas que serão úteis ao projeto.

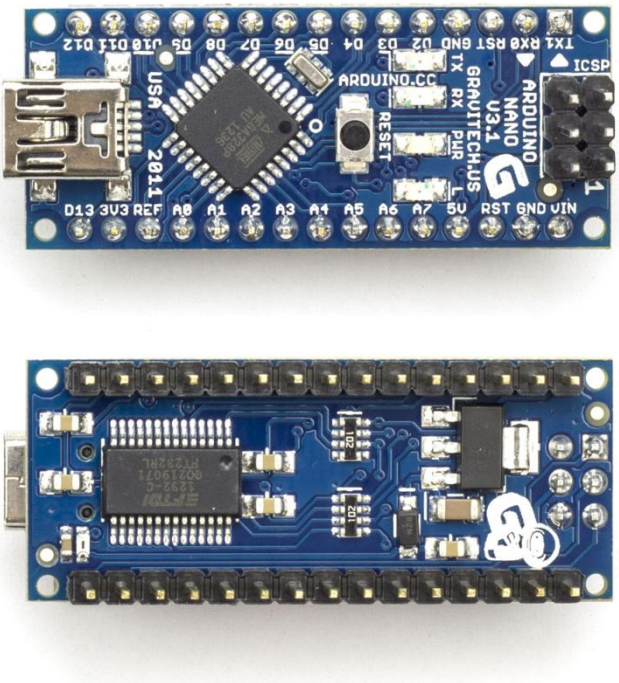


Figura 4.1 - Placa Arduino Nano.

4.1 – SOFTWARE ARDUINO

O Arduino surgiu na Itália em 2005 com o intuito de ser uma plataforma de prototipagem de fácil acesso. É composto basicamente por microcontroladores que em geral são os AVR da empresa Atmel que e em sua maioria são de 8 bits. Utiliza uma linguagem C/C++ e tem interface IDE multiplataforma escrita em Java derivada dos projetos Processing e Wiring, que contem um *debugger* e realce de sintaxe para facilitar a visualização. Dentro da IDE também é possível encontrar exemplos de códigos e de bibliotecas. A estrutura básica é; setup() - que define parâmetros de inicialização e loop() - que é a execução cíclica do programa. Abaixo na figura 4.2 pode-se visualizar a IDE do Arduino. [17]

Figura 4.2 - Interface da IDE Arduino.

O exemplo acima é o próprio programa de controle que será explicado mais adiante.

4.2 – MICROCONTROLADOR

Um microcontrolador, que também é conhecido como MCU pode ser considerado um computador em um único *chip* pois contém processador, memória e dispositivos de entrada e saída, também conhecidos como I/O. Ele pode ser programado para executar funções específicas e suas características de funcionamento dependem da arquitetura e do fabricante.

Microcontroladores tornaram possível a automatização de muitos dispositivos como máquinas de lavar, ar condicionados, geladeiras e muitos outros por serem pequenos e baratos. Uma estrutura de microcontrolador genérica pode ser vista na figura 4.3.

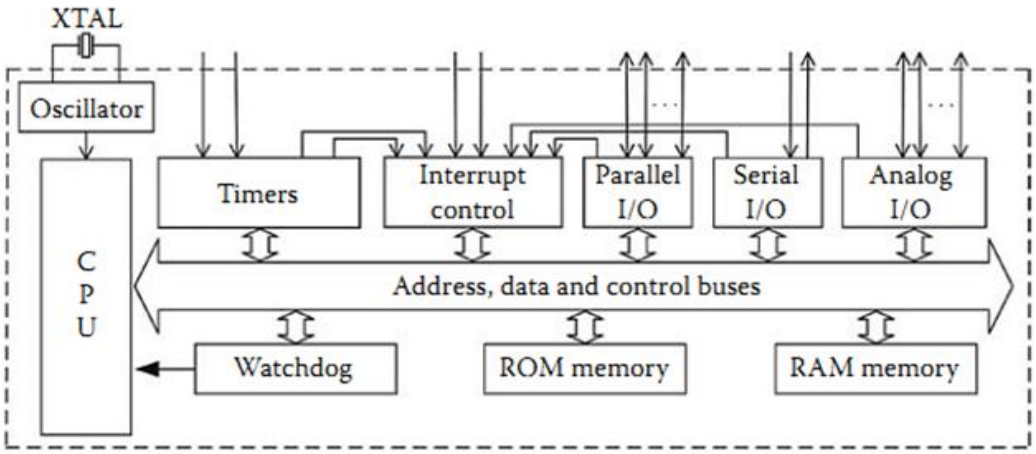


Figura 4.3 - Estrutura genérica de um microcontrolador.

Os microcontrolador em geral possui um oscilador interno, mas podem operar com um oscilador externo que garante uma maior precisão e se devidamente implementado tem uma boa imunidade a ruídos. As I/O podem ser paralelas, seriais ou analógicas dependendo do tipo de microcontrolador, podendo ter mais de um tipo de I/O. A memória ROM ou EPROM é a memória permanente do microcontrolador, onde é gravado o *bootloader*, já a memória RAM é a memória temporária que se apaga quando o microcontrolador é desligado. [18]

O microcontrolador escolhido para o projeto é o ATMEGA328P que é o mesmo utilizado na placa Arduino UNO e na placa Arduino Nano. Ele é compatível com a grande maioria de bibliotecas do Arduino e pode ser encontrado nas versões PDIP e SMD (figura 4.4)

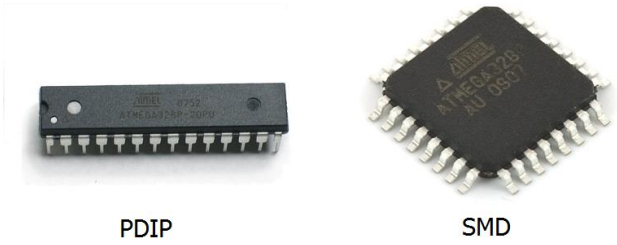


Figura 4.4 - ATMEGA328P.

Para funcionar com a interface Arduino o microcontrolador precisa ter o *bootloader* do Arduino gravado. O *bootloader* é um pequeno programa que é gravado na memória permanente do microcontrolador. Através desse programa que o microcontrolador se comunica com a interface de desenvolvimento, no caso do Arduino com a Arduino IDE. A própria IDE do Arduino possui uma opção de gravar o *bootloader* correspondente ao microcontrolador. Na figura 4.5 pode ser visto a pinagem do ATMEGA328P original e a utilizada no Arduino.

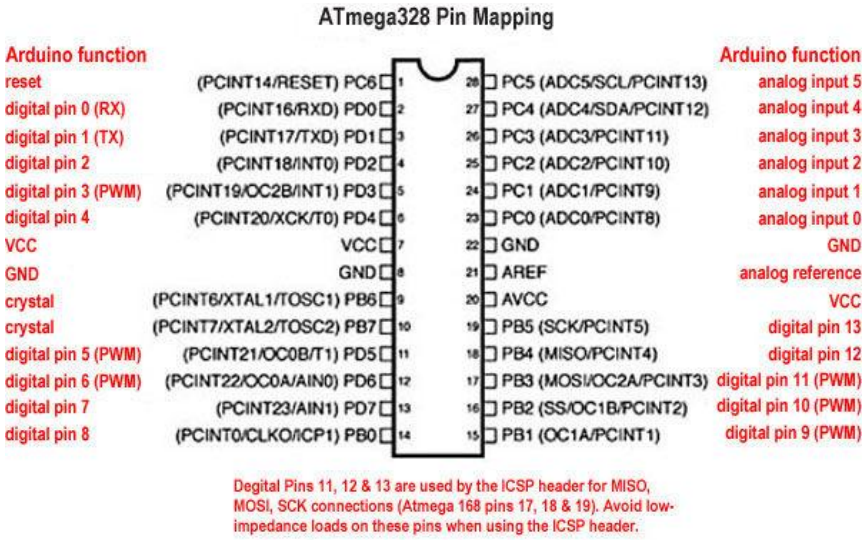


Figura 4.5 - Pinagem do ATMEGA328P.

4.3 – PROCESSAMENTO DIGITAL DE SINAIS

Para controlar o circuito o usuário precisa de um dispositivo que gere sinais PPM como rádios comuns de modelismo, mas primeiramente é preciso entender o funcionamento da modulação PPM e como os rádios com este padrão funcionam.

Modulação PPM

A modulação PPM ou *Pulse Posittion Modulation* é uma modulação de sinal que pode ser usada tanto para transmissões digitais como analógicas. Pode ser empregada em comunicações óticas, controles remoto infra-vermelhos, transmissores de rádio e outras aplicações onde se deseja o mínimo de interferência possível. [19]

Em uma transmissão PPM todos os pulsos tem que ter a mesma amplitude e o mesmo comprimento, o parâmetro que varia é o atraso entre os pulsos, o que determina o valor do sinal é a soma dos pulsos considerando seu atraso, como visto na figura 4.6.

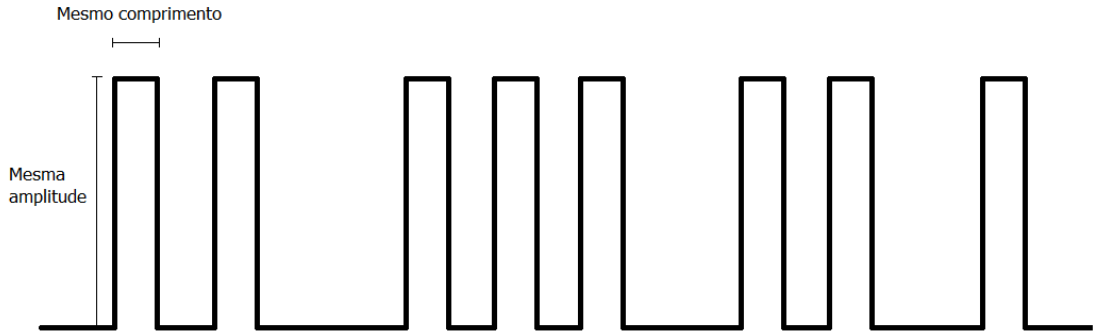


Figura 4.6 - Pulso PPM

Para sinais digitais só são necessárias duas variações no atraso dos pulsos para representar 0 e 1 lógicos. Já no sinal analógico são necessárias n variações no atraso dos pulsos onde n depende da precisão requerida. Na figura 4.7 é possível ver essa diferença. [20]

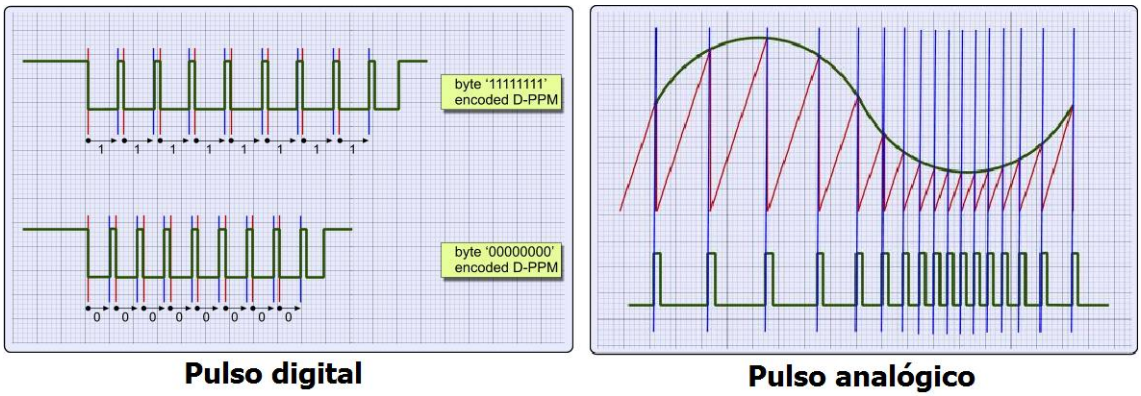


Figura 4.7 - Pulso PPM digital e analógico

Rádios PPM

Rádios que usam a modulação PPM geralmente usam um *frame* de 20ms por canal, ou seja o tempo total do *frame* por canal é de 20ms, o que significa que se o rádio possuir 6 canais o *frame* completo é de 100ms como visto na figura 4.8.

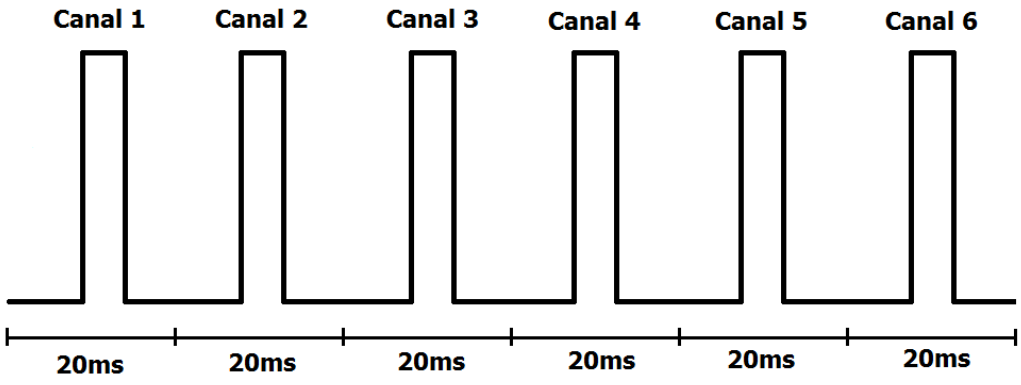


Figura 4.8 - Pulso PPM de um rádio de 6 canais.

Em cada *frame* o que de fato importa para a interpretação do sinal é o tempo que o mesmo fica em nível lógico alto, uma vez que se sabe que a cada 20ms é um *frame* que corresponde a um canal. O pulso fica em estado lógico alto de 1 a 2ms, sendo 1,5ms a posição central do *stick*, 1ms a posição mais baixa e 2ms a posição mais alta como visto na figura 4.9.

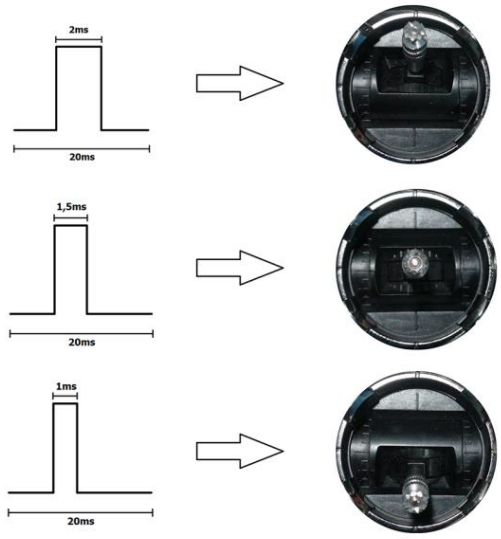


Figura 4.9- Sinais PPM em um rádio de modelismo.

Qualquer valor entre 1 e 1,5ms é referente ao movimento do para cima e qualquer valor entre 1,5 e 2ms se refere ao movimento do *stick* para baixo. Qualquer valor intermediário pode existir variando com a precisão do rádio. Na figura 4.3.5 podemos ver um exemplo de rádio de modelismo que utiliza o padrão PPM. [21]

Figura 4.10 - Rádio Turnigy 9XR.

Uma vez compreendido o funcionamento do sinal PPM e dos rádios, é preciso converter esses sinais em pulsos PWM respeitando o sentido de rotação do motor. Portanto da metade do *stick* para cima, ou seja, de 1,5 a 2ms, deseja-se que o motor gire para um lado e da metade do *stick* para baixo, ou seja de 1 a 1,5ms, se deseja que o motor gire para o outro lado. Sendo assim temos um sinal PWM para o motor girar no sentido horário e outro PWM para o motor girar no sentido anti-horário como pode ser visto na figura 4.11.

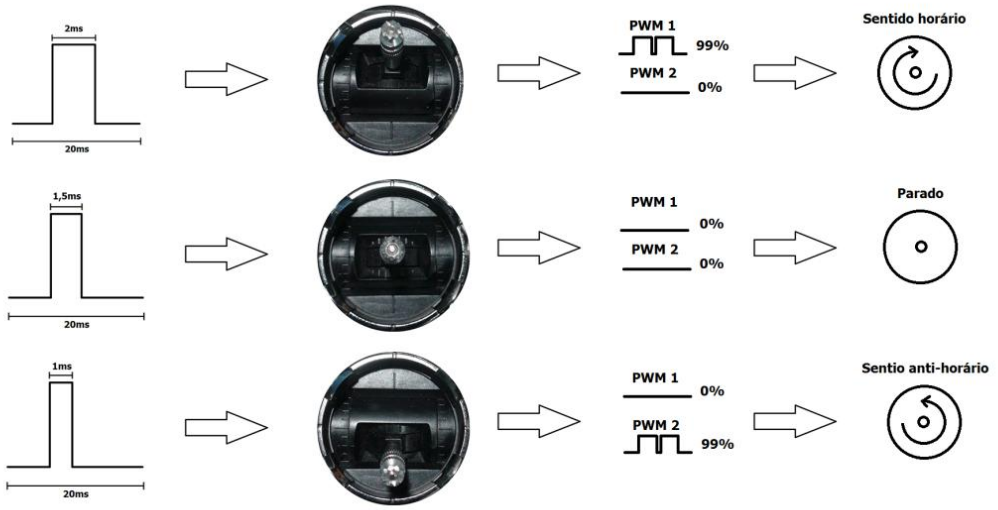


Figura 4.11 - Conversão de sinal PPM para PWM.

Essa conversão será feita pelo programa que lê o sinal PPM através do canal desejado do rádio e transformará esse sinal em dois sinais de PWM. Para sinais de 1 a 1,5ms é criado um PWM que vai de 0 a 99% em proporção a esse sinal e outro PWM, é criado para sinais de 1,5 a 2ms e o mesmo varia de 0 a 99% em proporção a esse sinal. É bom lembrar que é desejável se considerar uma zona morta próximo do meio do *stick* para que se tenha uma zona segura onde o motor está parado.

4.4 – HARDWARE

Para o correto funcionamento do módulo de controle é preciso além do microcontrolador um regulador 5V para garantir sua devida alimentação e proteger contra sobre tensão, capacitores de desacoplamento para proteger o circuito dos transientes de tensão e um diodo no positivo da alimentação do regulador para proteger o módulo de ser ligado em polaridade invertida.

É importante considerar o uso de um capacitor do pino que recebe os sinais PPM ao terra, pois sem ele caso o fio seja removido e o circuito ainda esteja ligado, o microcontrolador pode captar ruídos e mandar sinais aleatórios para o motor. Portanto, por questões de segurança esse capacitor é altamente recomendado como se pode ver na figura 4.12.

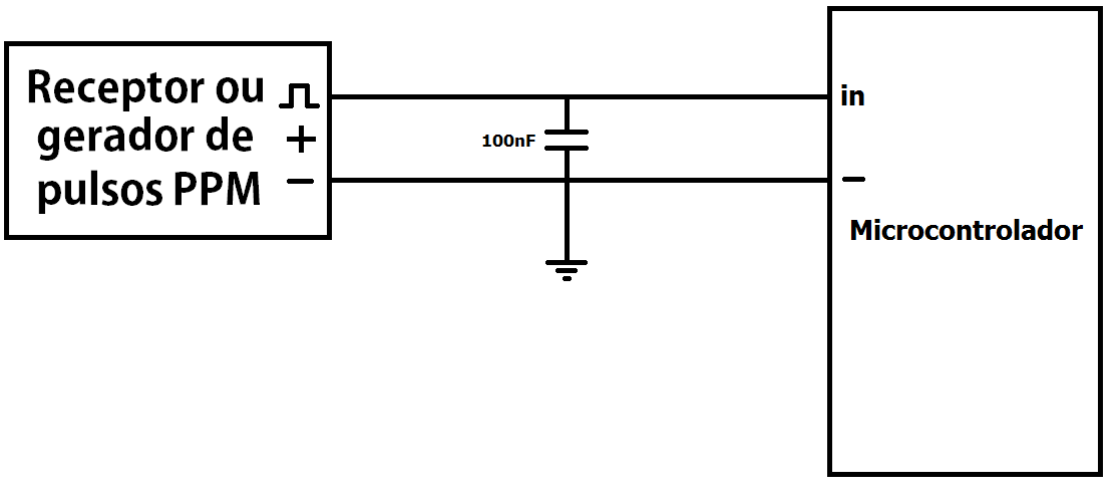


Figura 4.12 - Capacitor de desacoplamento do sinal.

Também é importante conectar o negativo do dispositivo ao negativo do microcontrolador para se ter um terra comum do circuito.

Pode ser implementado também o uso de capacitores de desacoplamento em todas as I/O utilizadas do microcontrolador com a mesma finalidade de evitar ruídos externos que comprometam o funcionamento.

Apesar de ser um padrão o sinal de 1 a 2ms nos rádios de modelismo esse valor na prática pode variar um pouco de rádio para rádio, portanto é interessante ter uma forma de calibrar esse sinal toda vez que se deseja usar um rádio diferente. Para isso foi utilizado um simples botão de toque que quando pressionado antes da inicialização entra em um modo de calibragem que tem a finalidade de definir os novos valores mínimos e máximos do sinal.

O circuito completo de controle pode ser visto na figura 4.13.

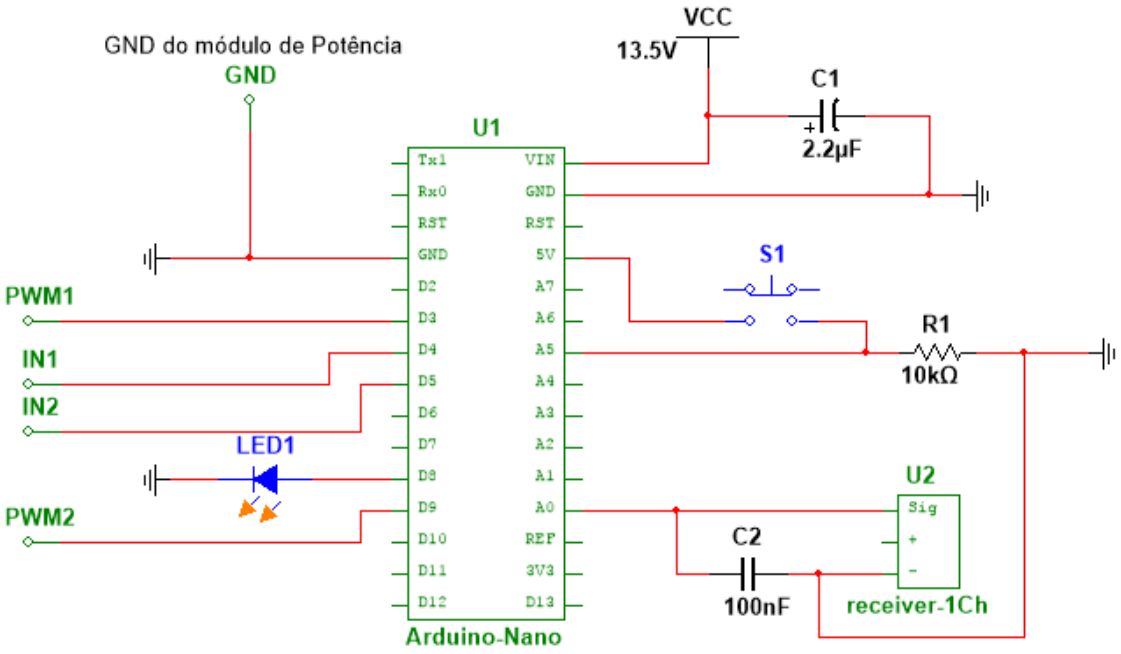


Figura 4.13 - Circuito de controle.

Neste caso como o Arduino Nano possui um regulador 5V interno não há a necessidade de se utilizar um, porém a tensão de entrada não deve passar de 18V para não super aquecer o regulador e danificar o circuito. Como o regulador chaveado foi projetado para mandar uma faixa de 12 a 13,5V não será um problema. É importante lembrar que os sinais PWM só funcionam em portas PWM do microcontrolador como as portas D3 e D9. [22]

O LED1 é o indicador de quando o modo de calibração está acionado.

4.5 – PROGRAMA

Tendo definido todo o circuito, tanto do módulo de potência quanto do módulo de controle agora só resta definir um programa que faça todas as rotinas especificadas de acionamento do motor. Para uma melhor compreensão o programa será apresentado em várias partes sendo que uma versão completa se encontra no apêndice B.

Bibliotecas utilizadas

Primeiramente é preciso entender o funcionamento das bibliotecas utilizadas e suas finalidades no programa.

Biblioteca EEPROM

Permite que sejam gravadas dados na memória EEPROM do microcontrolador. Esses dados não são apagados quando o microcontrolador é desligado, portanto será útil gravar o valor da última calibragem feita para que possa ser acessada pelo programa durante a inicialização. Os dados gravados no espaço da EEPROM são de um byte, ou seja oito bits que dão valores de *0* a *255*.

Biblioteca PWM

Essa biblioteca permite alterar a frequência padrão de PWM do Arduino que é próxima dos 500Hz. A vantagem de se fazer isso é poder chavear a frequências maiores. Através do protótipo feito foi possível perceber que frequências próximas de 3kHz deram uma resposta melhor do que frequências mais altas ou baixas, mas isso varia dependendo do MOSFET e do *driver* utilizado. A desvantagem dessa biblioteca é que ela altera os *timers* internos do microcontrolador, porém, possui uma função chamada *InitTimersSafe()* que preserva as funções de manutenção do tempo. Sendo assim das 6 portas PWM do Arduino Nano apenas duas continuam funcionando adequadamente, mas agora a frequência pode ser escolhida, e como só é preciso de duas portas PWM não há problemas em usá-la. [23]

Declaração de bibliotecas e variáveis

```
#include <EEPROM.h>
#include <PWM.h>

int32_t frequency = 3000;  // Frequência de chaveamento do PWM em Hz.
```

Essa parte do programa declara as bibliotecas EEPROM e PWM para serem utilizadas e a função *int32_t* da biblioteca PWM seta a frequência em 3KHz.

```
int PPM = A0; // Sinal de entrada PPM.
int PWM1 = 3; // Porta de saída 1 PWM.
int PWM2 = 9; // Porta de saída 2 PWM.
int IN1 = 4; // Entrada 1.
int IN2 = 5; // Entrada 2.
int pulse = 0;  // Sinal de entrada convertido em 8 bits.
int fail_safe = 0; // Função que identifica o estado de failsafe.
int PWM_1 = 0;  // Sinal PWM 8 bits de saída 1.
int PWM_2 = 0;  // Sinal PWM 8 bits de saída 2.
int led = 8;  // Led indicativo de estado.
const int  buttonPin = A5;  // Botão de calibragem.
float Min = 0;  // Posição da EEPROM contendo o valor mínimo.
float Max = 1;  // Posição da EEPROM contendo o valor máximo.
int maximo = 1997;  // Valor máximo de sinal de entrada.
int minimo = 980;  // Valor mínimo de sinal de entrada.
int buttonState = 0;  // Estado do botão.
int ponte_1 = 0; // Estado do IN1.
int ponte_2 = 0; // Estado do IN2.
int PWM_old1 = 0; // Estado anterior do PWM_D.
int PWM_old2 = 0; // Estado anterior do PWM_E.
int flag_1 = 0; // Flag do PWM_D.
int flag_2 = 0; // Flag do PWM_E.
int enable_1 = 0; // Função do debugger.
int enable_2 = 0; // Função do debugger.
```

Está parte do programa determina quais portas serão usadas para os sinais PWM (SD), para o *enable* (IN), sinal de entrada, botão de calibragem e LED indicativo. Também são definidas variáveis auxiliares que ajudam a definir os parâmetros do programa.

void setup()

Essa é a parte do programa onde será feita a rotina de inicialização do microcontrolador, ou seja, toda vez que ele for ligado ele passa primeiro por esse programa que possui parâmetros de inicialização e definição de valores.

```
void setup() {

    InitTimersSafe();  // Inacialização dos timers.
    SetPinFrequencySafe(PWM1, frequency);  // Setando frequência do PWM1.
    SetPinFrequencySafe(PWM2, frequency);  // Setando frequência do PWM2.
    pinMode(PWM1, OUTPUT);
    pinMode(PWM2, OUTPUT);
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(sd, OUTPUT);
    pinMode(PPM, INPUT);
    pinMode(led, OUTPUT);
    pinMode(buttonPin, INPUT);
   buttonState = digitalRead(buttonPin);  // Define estado do botão como leitura do pino correspondente.
```

Nesta parte do programa são inicializados os *timers* pela biblioteca PWM, e posteriormente são atribuídas as novas frequências de funcionamento para as portas PWM1 e PWM2.

A função *pinMode()* define se a porta vai funcionar como entrada ou saída. Posteriormente a variável *buttonState* é atribuída a porta do botão de calibragem.

Calibragem do sinal

```
  if (buttonState == HIGH)  // Verifica se o botão foi pressionado.
 {
    delay(100);
    digitalWrite(led, HIGH);  // Liga o led de calibragem.
    delay(2000);
    pulse = pulseIn(PPM, HIGH);  // Lê o sinal do receptor.
    EEPROM.write(Max, pulse/8);  // Grava na EEPROM o valor máximo
dividido por 8 para caber em 8 bits.


 // Pisca o led para avisar que o valor máximo foi computado.
    for (int i=0; i <= 4; i++)
    {
       delay(100);
       digitalWrite(led, LOW);
       delay(100);
       digitalWrite(led, HIGH);
    }
```

Essa rotina verifica se o usuário manteve o botão de calibragem pressionado antes de ligar o circuito, se estava pressionado então ele continua a função *if*. O led se ascende indicando que o modo de calibragem está ativo, posteriormente o usuário tem dois segundos para colocar o *stick* na posição máxima desejada. Após os dois segundos é lido o valor atual do pulso dividido por oito pois o dado precisa caber em 8 bits, como são valores teoricamente entre mil e dois mil precisam ser de alguma forma comprimidos para caber em oito bits. Esse valor é gravado na posição *1* da EEPROM, e o led pisca várias vezes para indicar que aquele dado foi gravado com sucesso. A função *pulseIn()* devolve o valo que um sinal ficou em estado alto (HIGH) ou baixo (LOW), nesse caso precisamos saber o tempo que o sinal ficou em nível alto para detectar a variação de 1 a 2ms.

Depois a mesma coisa deve ser feita, agora para gravar o sinal da posição mínima. apos a espera de dois segundos e o LED ter piscado várias vezes ele aguarda dois segundos novamente e depois faz o mesmo procedimento para armazenar o valor mínimo na posição *0* da EEPROM.

```
    delay(2000);
    pulse = pulseIn(PPM, HIGH);  // Lê o sinal do receptor.
    EEPROM.write(Min, pulse/8);  // Grava na EEPROM o valor mínimo dividido por 8 para caber em 8 bits.

  // Pisca o led para avisar que o valor mínimo foi computado.

    for (int i=0; i <= 4; i++)

    {

      delay(100);
      digitalWrite(led, LOW);
      delay(100);
      digitalWrite(led, HIGH);

    }

      delay(500);
      digitalWrite(led, LOW);
      delay(200);

  }

// Passando pela calibragem ou não o programa define os valores máximos ou mínimos.

// Leitura do valor mínimo e máximo da calibragem.

maximo = (EEPROM.read(Max) * 8) + 2;
minimo = (EEPROM.read(Min) * 8) + 2;
```

Para se voltar ao valor original do sinal é preciso multiplicá-lo por oito novamente, mas como divisões não exatas não deixam restos existem perdas, o número dois somado ao final da expressão é um fator de compensação para essas perdas e o erro obtido ao final passa a ser desprezível.

void loop ()

Essa é a parte do programa onde serão feitas as rotinas de loop, ou seja, as rotinas que funcionam enquanto o circuito estiver ligado.

```
void loop() {

// Desativa botão de calibragem durante a execução do loop.

  buttonState == LOW;

 // Lê o sinal do receptor.

  pulse = pulseIn(PPM, HIGH);
  fail_safe = pulseIn(PPM, HIGH);

// Converte em 8 bits.

  pulse = map(pulse, minimo, maximo, 0, 255);

// Define o PWM da meia ponte esquerda e direita e zona morta.

     PWM_D = pulse;
     PWM_E = pulse;

// Re mapeando PWM_D e PWM_E para 8 bits considerando uma zona morta.

     PWM_D = map(PWM_D, 0, 104, 253, 0);
     PWM_E = map(PWM_E, 150, 255, 0, 253);
```

O sinal PPM foi remapeado pela função map() para uma faixa de *0* a *255* (8 bits), e depois as variáveis PWM_D e PEM_E foram remapeadas, uma para a parte baixa do sinal e a outra para a parte alta, note que foi definida uma zona morta para dar suavidade ao controle e evitar a reversão imediata. Esse mapeamento também evita que os dois sinais PWM sejam enviados ao mesmo tempo.

```
// Condições descartadas (fora dos 8 bits).

 if(pulse > maximo)
   pulse = maximo;
 if(pulse < minimo)
   pulse = minimo;
 if(PWM_D <= 0)
   PWM_D = 0;
 if(PWM_D > 253)
   PWM_D = 253;
 if(PWM_E <= 0)
   PWM_E = 0;
 if(PWM_E > 253)
   PWM_E = 253;
```

São descartadas todas as condições fora de intervalo ou que não interessam, isso evita que o PWM seja de 100% ou que seja menor que 0%. Evita também que o pulso seja maior que o máximo e menor que o mínimo.

Fail-safe

A variável *fail_safe* nada mais é que uma cópia da variável pulse, e só foi criada para facilitar a compreensão do programa. Quando o receptor está sem sinal o valor enviado é menor que zero, portanto é fácil determinar pela função abaixo quando ocorre uma condição de *fail-safe*. O *enable* é habilitado desligando os dois lados da ponte e por segurança os dois sinais PWM são setados em 0 e o LED de calibragem fica acesso indicando perda de sinal.

Essa função evita que em caso de perda de sinal o robô ou dispositivo fique descontrolado evitando riscos a segurança do usuário ou dos demais.

```
// Fail-safe (perda de sinal).

 if(fail_safe == 0)

 {

    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
    digitalWrite(PWM1, 0);
    digitalWrite(PWM2, 0);
    digitalWrite(led, HIGH);
```

Para evitar a reversão instantânea do motor, o que pode causar um curto circuito na ponte H, foram utilizadas duas variáveis para cada lado da ponte que identificam o sentido de rotação, são elas *PWM_old* e *flag*. A função *if* testa qual lado está sendo acionado no momento e atribui a variável *PWM_old* do lado acionado como 0 e a do lado oposto como 1, identificando que antes o motor girava no lado oposto. Também atribui a variável *flag* do lado que está acionado como 1 indicando que agora está girando para esse lado. Quando o lado é revertido, ou seja, o outro lado da ponte é acionado uma outra lógica *if* checa três condições: Se a flag desse lado é 1, se a variável *PWM_old* do lado oposto é 1 e se a variável *PWM_old* desse lado é 0 (ou seja, se estava antes girando para o lado oposto).

```
// Delay na reversão.

if (PWM_1 > 0)
{
   PWM_old1 = 0;
   PWM_old2 = 1;
   flag_1 = 1;
}
if (PWM_2 > 0)
{
   PWM_old1 = 1;
   PWM_old2 = 0;
   flag_2 = 1;
}
if (flag_2 == 1 && PWM_old1 == 0 && PWM_old2 == 1)
{
   flag_2 = 0;
   ponte_1 = 255;
   ponte_2 = 0;
   delay(120);
}
if (flag_1 == 1 && PWM_old1 == 1 && PWM_old2 == 0)
{
   flag_1 = 0;
   ponte_1 = 0;
   ponte_2 = 255;
   delay(120);
}
```

Depois são enviados aos opto-acopladores os sinais PWM que determinam a velocidade e sentido do motor.

```
// Definindo sinais PWM.

    pwmWrite(PWM1, PWM_1);

    pwmWrite(PWM2, PWM_2);

    analogWrite(IN1, ponte_1);

    analogWrite(IN2, ponte_2);
```

Por último é criada uma lógica que garante que o motor não pare bruscamente deixando a corrente circular pelos diodos de *freewheeling.*

```
// Freewheelin (Coast).

if (PWM_1 > 0)

{

  ponte_1 = 255;
  ponte_2 = 0;
  digitalWrite(PWM2, HIGH);

}

else if (PWM_2 > 0)

{

  ponte_1 = 0;
  ponte_2 = 255;
  digitalWrite(PWM1, HIGH);

}

else if (PWM_1 == 0 && PWM_2 == 0)

{

  ponte_1 = 0;
  ponte_2 = 0;

}

   delay(5); // Artifical break da CPU
```

Essa parte do programa serve apenas como diagnóstico para ver se as portas estão funcionando adequadamente. As informações são printadas no serial monitor do Arduino (que pode ser visualizado no serial monitor do Windows, Linux ou MAC).

```
// Debugger.
    enable_1 = digitalRead(IN1);
    enable_2 = digitalRead(IN2);
    Serial.print("PWM1 - ");
    Serial.print(PWM_1);
    Serial.print("  PWM2 - ");
    Serial.print(PWM_2);
    Serial.print("  IN1 - ");
    Serial.print(enable_1);
    Serial.print("  IN2 - ");
    Serial.println(enable_2);

}
```

5 – CONCLUSÃO E PROPOSTAS

Tendo em vista todas as etapas do processo pode-se concluir que são necessárias várias áreas do conhecimento para o projeto de um circuito desse tipo, entre elas: elétrica, eletrônica, circuitos analógicos, circuitos digitas, microcontroladores, processamento digital de sinais, eletrônica de potência e programação.

Mesmo com a dificuldade de se encontrar componentes no Brasil existe bastante material disponível sobre controladores de motores, o que tornou possível a pesquisa e desenvolvimento desse projeto.

Foram desenvolvidos dois protótipos. O primeiro em *protoboard* e o segundo foi construído em placas de circuito impressos universais como pode ser visto na figura 5.1.

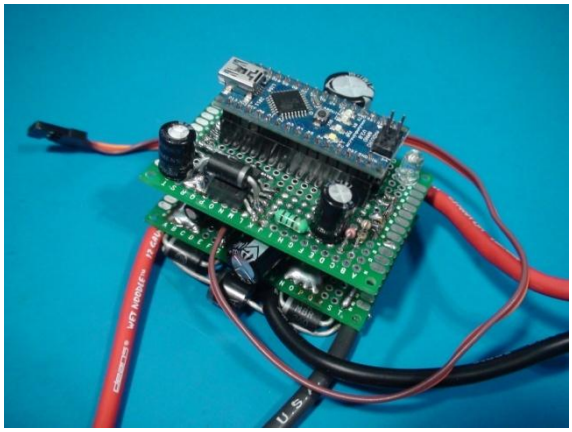


Figura 5.1 - Protótipo feito em placa de circuito impresso universal.

Como o sistema é modular as duas placas são fixadas entre si por conectores de barra de pinos que conectam as partes comuns às placas de controle e potência, como pode ser visto na figura 5.2.

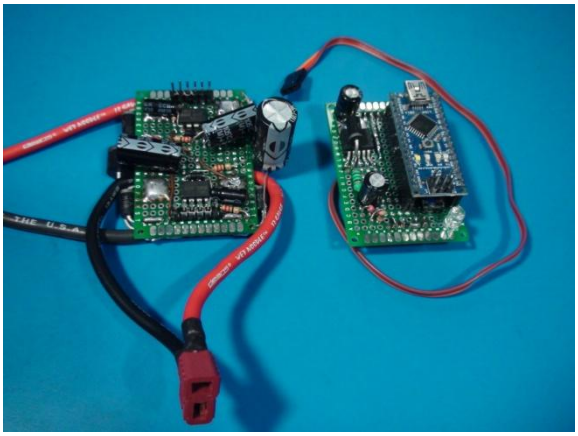


Figura 5.2 - Módulos de controle e potência.

Como sugestão para futuros trabalhos nessa área, eu recomendo a implementação de um controle de corrente com a finalidade de garantir que o limite de corrente estipulado não seja extrapolado, mesmo que seja um limite com uma boa margem de folga um controle desse tipo pode ajudar a prevenir danos por curto ou defeito do motor. Uma forma simples de fazer esse controle pode ser visto na fig. 5.3.

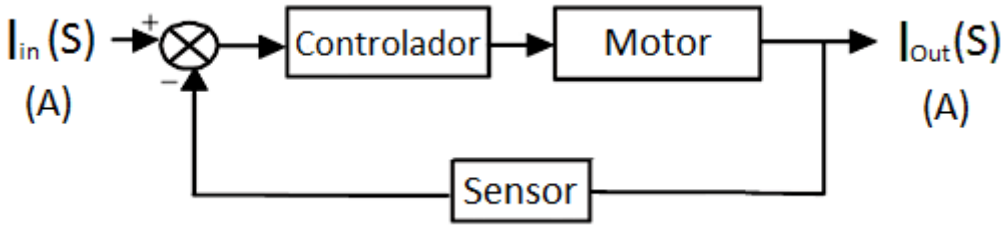


Figura 5.3 - Malha fechada do controle de corrente.

Dessa forma é possível detectar a corrente e caso ela extrapole o limite o controlador interfere no sinal PWM diminuindo seu valor e consequentemente a tensão do motor, até que a corrente fique abaixo do valor estipulado novamente fechando a malha mantendo este mesmo ciclo de varredura da corrente.

Este material teve como objetivo ser desenvolvido da forma mais clara, objetiva e ilustrativa possível para a melhor compreensão, utilizando-se de várias figuras e esquemáticos que evidenciem o conhecimento a ser transmitido.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] 1 autor:

BRAGA, Newton. http://www.newtoncbraga.com.br: Artigos: ART476. (acessado em 27/03/2014)

[2] 1 autor:

BRAGA, Newton. http://www.newtoncbraga.com.br: Artigos: MEC073A. (acessado em 27/03/2014)

[3] 1 autor:

BASCONCELLO, Daniel. http://www.robotizando.com.br/artigo_ponte_h_pg1.php (acessado em 27/03/2014)

[4] 1 autor:

HIRZEL, Timothy. http://arduino.cc/en/Tutorial/PWM (acessado em 27/03/2014)

[5] 1 autor:

MICHELS, Leandro. http://www.joinville.udesc.br/portal/professores/michels/materiais/EPO1___Aula_01_04.pdf (acessado em 13/04/2014)

[6] 1 autor:

BAUSCH, Jeffrey. http://www.electronicproducts.com/Analog_Mixed_Signal_ICs/Discrete_Power_Transistors/MOSFET_vs_IGBT.aspx

[7] 2 autores:

BLAKE, Carl; BULL, Chris. http://www.irf.com/technical-info/whitepaper/choosewisely.pdf (acessado em 27/03/2014)

[8] 1 autor:

COLTON, Shane.
http://scolton.blogspot.com.br/2011/01/victor-883107-7.html (acessado em 18/04/2014)

[9] International Rectifier IRFS3107 datasheet.

http://www.irf.com/product-info/datasheets/data/irfs3107-7ppbf.pdf (acessado em 02/03/2014)

[10] International Rectifier IRS2184 datasheet

http://www.irf.com/product-info/datasheets/data/irs2184.pdf (acessado em 02/03/2014)

[11] Vishay 4N25 datasheet

http://www.vishay.com/docs/83725/4n25.pdf (acessado em 09/03/2014)

[12] 1 autor:

ADAMS, Jonathan
http://www.irf.com/technical-info/designtp/dt98-2.pdf (acessado em 12/03/2014)

[13] 3 autores:

A. MERELLO; A. RUGGINENTI; M. GRASSO
http://www.irf.com/technical-info/designtp/dt04-4.pdf (acessado em 13/03/2014)

[14] Mecatrônica Atual

http://www.mecatronicaatual.com.br/educacao/910-transientes-de-tensao (acessado em 17/04/2014)

[15] Dimension Engineering

https://www.dimensionengineering.com/info/switching-regulators (acessado em 17/04/2014)

[16] LM2576-HVdatasheet

http://www.ti.com/lit/ds/symlink/lm2576.pdf (acessado em 17/04/2014)

[17] Arduino website

http://www.arduino.cc/ (acessado em 04/02/2014)

[18] 1 autor:

BRAIN, Marshall
http://electronics.howstuffworks.com/microcontroller.htm (acessado em 13/04/2014)

[19] 1 autor:

OLIVEIRA, Alessandro
http://paginapessoal.utfpr.edu.br/alessandro/disciplinas-do-semestre/principios-de-comunicacao/aulas/terceira-prova/modulacao%20pam-%20pwm%20e%20ppm.pdf (acessado em 24/04/2014)

[20] 1 autor:

LARARIDS, Giorgos
http://www.pcbheaven.com/wikipages/Pulse_Position_Modulation/ (acessado em 06/04/2014)

[21] endurace-rc.com

http://www.endurance-rc.com/ppmtut.php (acessado em 12/04/2014)

[22] 1 Autor:

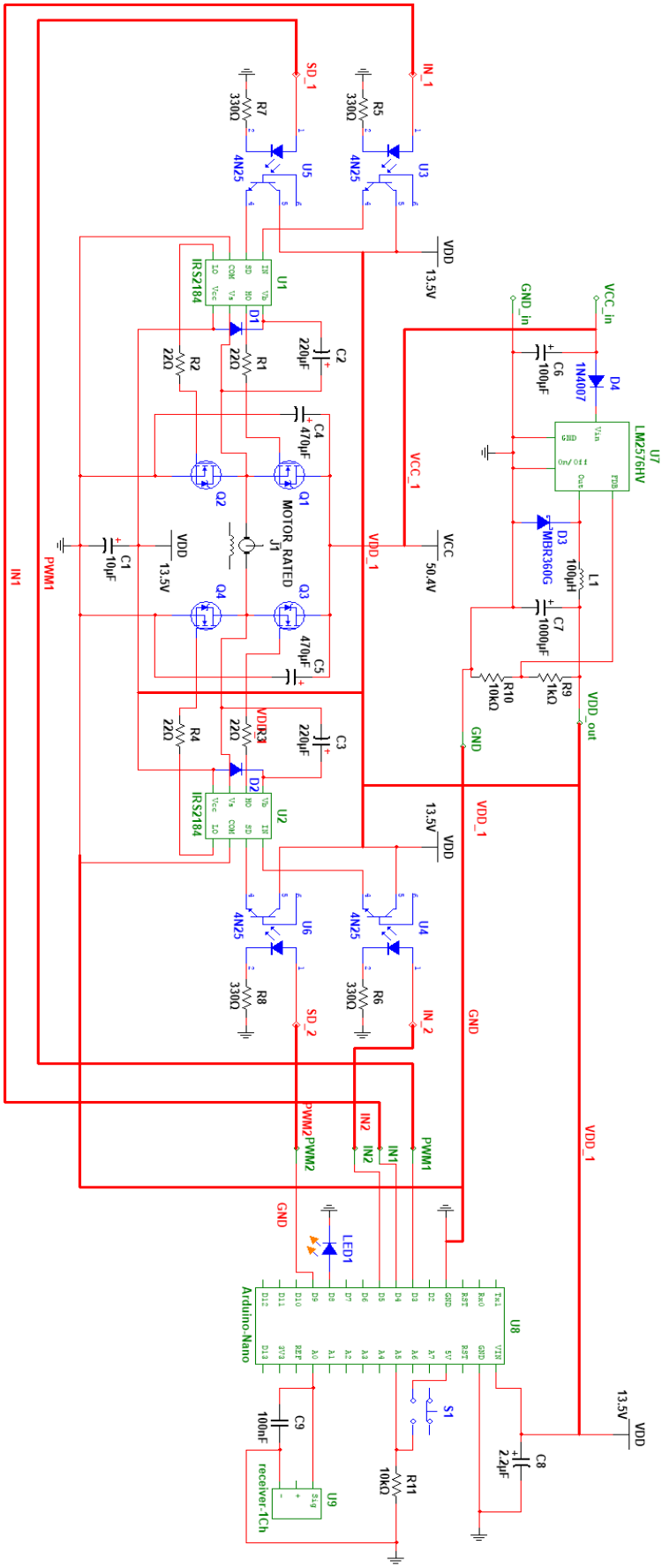
BERTINI, Luiz
http://doradioamad.dominiotemporario.com/doc/O_Capacitor.pdf (acessado em 06/01/2014)

[23] Arduino Fórum

http://forum.arduino.cc/index.php?topic=117425.0 (acessado em 06/01/2014)

Referência complementar: OSMC (Open Source Motor Control)
https://groups.yahoo.com/neo/groups/osmc/info (acessado em 15/03/2014)

APÊNDICE A – ESQUEMÁTICO COMPLETO



APÊNDICE B – LISTA DE COMPONENTES

COMPONENTE	QUANTIDADE	PREÇO ESTIMADO U$/un.
IRFS3107	4	6,5
IRS2184	2	3
4N25	4	0,4
LM2576-HV	1	5,8
Resistor 22ohm	4	0,05
Resistor 330ohm	2	0,05
Resistor 1K	2	0,05
Resistor 10K	1	0,05
Capacitor cerâmico 100nF	1	0,02
Capacitor Eletrolítico 220uF 25V	2	0,86
Capacitor Eletrolítico 470uF 63V	2	1,8
Capacito Eletrolítico 100uF 63V	1	1,2
Capacitor Eletrolítico 1000uF 25V	1	1,6
Capacitor Eletrolítico 2,2uF 25V	1	0,35
Capacitor Eletrolítico 10uF 25V	1	0,42
Indutor 100uH	1	0,87
Diodo 1N4007	3	0,15
Diodo Schottky 1N5822	1	0,47
Push-button	1	0,2
TOTAL	33	49,75

APÊNDICE C – PROGRAMA COMPLETO

```
//-_-_-_-Simparts Delaytronics SP50V120A-_-_-_-

#include <EEPROM.h>
#include <PWM.h>
int32_t frequency = 3000;   // Frequência de chaveamento do PWM em Hz.

int PPM = A0; // Sinal de entrada PPM.
int PWM1 = 3; // Porta de saída 1 PWM.
int PWM2 = 9; // Porta de saída 2 PWM.
int IN1 = 4; // Entrada 1.
int IN2 = 5; // Entrada 2.
int pulse = 0;  // Sinal de entrada convertido em 8 bits.
int fail_safe = 0; // Função que identifica o estado de failsafe.
int PWM_1 = 0;  // Sinal PWM 8 bits de saída 1.
int PWM_2 = 0;  // Sinal PWM 8 bits de saída 2.
int led = 8;  // Led indicativo de estado.
const int  buttonPin = A5;  // Botão de calibragem.
float Min = 0;  // Posição da EEPROM contendo o valor mínimo.
float Max = 1;  // Posição da EEPROM contendo o valor máximo.
int maximo = 1997;  // Valor máximo de sinal de entrada.
int minimo = 980;  // Valor mínimo de sinal de entrada.
int buttonState = 0;  // Estado do botão.
int ponte_1 = 0; // Estado do IN1.
int ponte_2 = 0; // Estado do IN2.
int PWM_old1 = 0; // Estado anterior do PWM_D.
int PWM_old2 = 0; // Estado anterior do PWM_E.
int flag_1 = 0; // Flag do PWM_D.
int flag_2 = 0; // Flag do PWM_E.
int enable_1 = 0; // Função do debugger.
int enable_2 = 0; // Função do debugger.

void setup() {

    Serial.begin(9600);
    InitTimersSafe();  // Inacialização dos timers.
    SetPinFrequencySafe(PWM1, frequency);  // Setando frequência do PWM1.
    SetPinFrequencySafe(PWM2, frequency);  // Setando frequência do PWM2.
```

```
    pinMode(PWM1, OUTPUT);
    pinMode(PWM2, OUTPUT);
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(PPM, INPUT);
    pinMode(led, OUTPUT);
    pinMode(buttonPin, INPUT);

   buttonState = digitalRead(buttonPin);  // Define estado do botão como leitura do pino correspondente.

//------------------------------- Calibragem ----------------------------------

  if (buttonState == HIGH)  // Verifica se o botão foi pressionado.
  {
    delay(100);
    digitalWrite(led, HIGH);  // Liga o led de calibragem.
    delay(2000);
    pulse = pulseIn(PPM, HIGH);  // Lê o sinal do receptor.
    EEPROM.write(Max, pulse/8); // Grava na EEPROM o valor máximo dividido por 8 para caber em 8 bits.

  // Pisca o led para avisar que o valor máximo foi computado.
    for (int i=0; i <= 4; i++)
    {
      delay(100);
      digitalWrite(led, LOW);
      delay(100);
      digitalWrite(led, HIGH);
    }
    delay(2000);
    pulse = pulseIn(PPM, HIGH);  // Lê o sinal do receptor.
    EEPROM.write(Min, pulse/8); // Grava na EEPROM o valor mínimo dividido por 8 para caber em 8 bits.

  // Pisca o led para avisar que o valor mínimo foi computado.
    for (int i=0; i <= 4; i++)
    {
      delay(100);
      digitalWrite(led, LOW);
      delay(100);
      digitalWrite(led, HIGH);
    }
```

```
        delay(500);
        digitalWrite(led, LOW);
        delay(200);
  }
//-------------------------------------------------------------------------------

// Leitura do valor mínimo e máximo da calibragem.
   maximo = (EEPROM.read(Max) * 8) + 2;
   minimo = (EEPROM.read(Min) * 8) + 2;

}

void loop() {

// Desativa botão de calibragem durante a execução do loop.
  buttonState == LOW;

// Lê o sinal do receptor.
  pulse = pulseIn(PPM, HIGH);
  fail_safe = pulseIn(PPM, HIGH);

// Converte em 8 bits.
  pulse = map(pulse, minimo, maximo, 0, 255);

// Define o PWM da meia ponte esquerda e direita e zona morta.
     PWM_1 = pulse;
     PWM_2 = pulse;

// Re mapeando PWM_D e PWM_E para 8 bits considerando uma
zona morta.
     PWM_1 = map(PWM_1, 0, 104, 253, 0);
     PWM_2 = map(PWM_2, 150, 255, 0, 253);

// Condições descartadas (fora dos 8 bits).
     if(pulse > maximo)
       pulse = maximo;
     if(pulse < minimo)
       pulse = minimo;
     if(PWM_1 <= 0)
       PWM_1 = 0;
     if(PWM_1 > 253)
       PWM_1 = 253;
     if(PWM_2 <= 0)
```

```
      PWM_2 = 0;
    if(PWM_2 > 253)
      PWM_2 = 253;

// Fail-safe (perda de sinal).
 if(fail_safe == 0)
 {
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
    digitalWrite(PWM1, 0);
    digitalWrite(PWM2, 0);
    digitalWrite(led, HIGH);
 }

// Delay na reversão.
if (PWM_1 > 0)
{
    PWM_old1 = 0;
    PWM_old2 = 1;
    flag_1 = 1;
}
if (PWM_2 > 0)
{
    PWM_old1 = 1;
    PWM_old2 = 0;
    flag_2 = 1;
}
if (flag_2 == 1 && PWM_old1 == 0 && PWM_old2 == 1)
{
    flag_2 = 0;
    ponte_1 = 255;
    ponte_2 = 0;
    delay(120);
}
if (flag_1 == 1 && PWM_old1 == 1 && PWM_old2 == 0)
{
    flag_1 = 0;
    ponte_1 = 0;
    ponte_2 = 255;
    delay(120);
}
```

```
// Definindo sinais PWM.
      pwmWrite(PWM1, PWM_1);
      pwmWrite(PWM2, PWM_2);
      analogWrite(IN1, ponte_1);
      analogWrite(IN2, ponte_2);

// Freewheelin (Coast).
 if (PWM_1 > 0)
 {
    ponte_1 = 255;
    ponte_2 = 0;
    digitalWrite(PWM2, HIGH);
 }
 else if (PWM_2 > 0)
 {
    ponte_1 = 0;
    ponte_2 = 255;
    digitalWrite(PWM1, HIGH);
 }
 else if (PWM_1 == 0 && PWM_2 == 0)
 {
    ponte_1 = 0;
    ponte_2 = 0;
 }

    delay(5); // Artificial break da CPU.

// Debugger.
    enable_1 = digitalRead(IN1);
    enable_2 = digitalRead(IN2);
    Serial.print("PWM1 - ");
    Serial.print(PWM_1);
    Serial.print("  PWM2 - ");
    Serial.print(PWM_2);
    Serial.print("  IN1 - ");
    Serial.print(enable_1);
    Serial.print("  IN2 - ");
    Serial.println(enable_2);

}
```