LUCAS CARNEIRO NOVAES

Desenvolvimento do *Software* e do *Hardware* de um Módulo de Controle para um Robô Pendular

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica com Ênfase em
Sistemas de Energia e Automação

ORIENTADOR: Prof. Dr. Eduardo do Valle Simões

São Carlos
2014

Novaes, Lucas Carneiro
N935d Desenvolvimento do software e do hardware de um módulo de controle para um robô pendular / Lucas Carneiro Novaes; orientador Eduardo do Valle Simões. São Carlos, 2014.

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

1. Robótica educacional. 2. Driver de motores. 3. Sensor de ultrassom. 4. Arduino. 5. PWM. 6. Módulo de controle. I. Título.

FOLHA DE APROVAÇÃO

Nome: Lucas Carneiro Novaes

Título: "Desenvolvimento do Software e do Hardware de um módulo de controle para um robô pendular"

Trabalho de Conclusão de Curso defendido e aprovado em 25 / 11 / 2014,

com NOTA 9,5 (nove, cinco), **pela Comissão Julgadora:**

Prof. Dr. Eduardo do Valle Simões - (Orientador - SSC/ICMC/USP)

Prof. Dr. José Carlos de Melo Vieira Júnior - (SEL/EESC/USP)

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

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

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Este trabalho trata do projeto e implementação do módulo de controle de uma plataforma robótica pendular de duas rodas projetada pelo Laboratório de Computação Reconfigurável do ICMC-USP. O módulo de controle inclui todo o conjunto de *hardware* e *software* de baixo nível que permita ao robô receber comandos de navegação de um computador de mais alto nível via conexão sem fio *Bluetooth*, interpretar esses comandos e convertê-los em seu deslocamento. A locomoção do robô é função de dois motores de corrente contínua que são acionados por um microcontrolador através do *driver* de potência, também integrante do módulo de controle. A execução dos comandos de navegação recebidos pelo Arduino, plataforma microcontrolada da qual se faz uso neste projeto, fica sempre condicionada à existência ou não de obstáculos na trajetória do robô. A detecção de obstáculos em potencial é feita por dois módulos de aferição de distância baseados em pulsos ultrassônicos. A plataforma robótica para a qual esse módulo de controle foi desenvolvido tem aplicação em projetos educacionais dentro dos contextos de tecnologia, inovação e robótica.

Palavras-chave: Robótica Educacional, Driver de Motores, Sensor de Ultrassom, Arduino, PWM, Módulo de Controle.

Abstract

This is a project on both design and implementation of a control module for a two-wheeled mobile robotic device designed by the Reconfigurable Computing Laboratory at ICMC-USP. The control module designed includes a whole set of hardware and low level software which allows the robot to receive navigation instructions sent by a high level software running on a computer through Bluetooth link, to process those instructions and to convert them into its mobility. Two DC motors, which are driven by a microcontroller through a power driver, are responsible for this robot's mobility, being that device also part of the control module. The proper execution of navigation instructions received by an Arduino, microcontroller of which it is made use in this project, is submitted to whether there is an obstacle on its way or not. The detection of potential obstacles is done by two ultrasonic range modules. The robotic device for which this control module was designed is intended to be part of educational projects within technology, innovation and robotic contexts.

Keywords: Educational Robotics, DC Motor Driver, Ultrasonic Range, Arduino, PWM, Control Module.

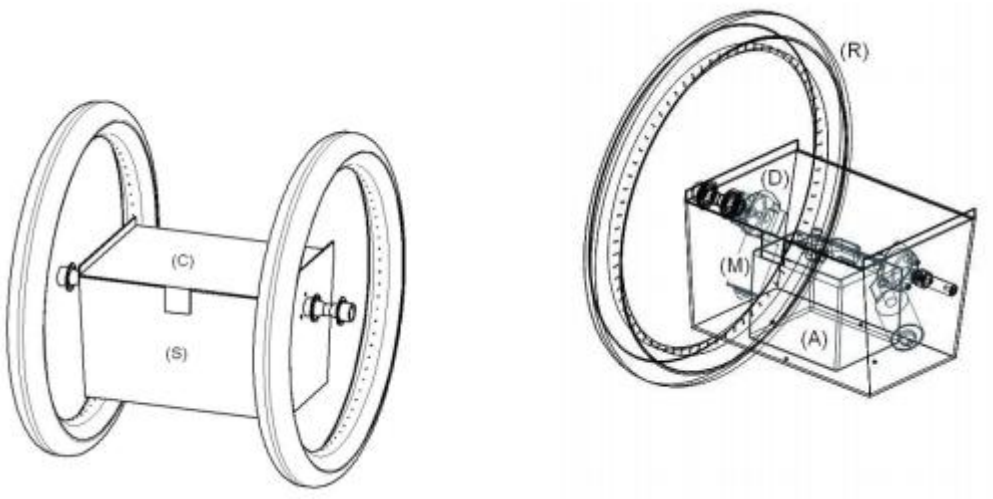
1. Introdução

A introdução da robótica em projetos educacionais ou pedagógicos nas escolas de ensino básico é um recurso relativamente novo que surge em resposta à demanda por uma nova classe de competências de que nossa sociedade necessita. Essas competências, de modo genérico, estão relacionadas com a real compreensão do conhecimento teórico, tornando-o concreto através de planejamento, projeto e criação. Através da robótica, a informação que tradicionalmente é tratada apenas em textos de livros didáticos pode ser convertida em algo concreto pelo contato direto com a tecnologia (CHELLA, 2002). Além da robótica, atualmente discute-se muito a respeito de novas metodologias e/ou metodologias complementares de ensino, visto que os modelos tradicionais de ensino estão ficando defasados com relação às novas necessidades de nossa sociedade, que está cada vez mais integrada e globalizada. Assim, um ambiente de robótica educacional surge como uma extensão do ambiente de sala de aula, onde as propostas de atividades e projetos trazem mais dinamismo e significado aos conteúdos de – principalmente, mas não limitado a, - Ciências Exatas. Os projetos e atividades educacionais envolvendo robótica não são por si só a mudança na metodologia de ensino, mas constituem um ambiente no qual o estudante pode encontrar sua identidade ou motivação que o instiga a saber mais, levando-o a questionar seus professores em sala de aula (HERNÁNDEZ, 1998) (SIMÕES, 2008).

As escolas particulares atualmente vêm adquirindo cada vez mais kits robóticos educacionais de vários fabricantes e seus alunos têm sido introduzidos a esta tecnologia desde os primeiros anos de sua educação. Já as escolas públicas têm mais dificuldade em arcar com o custo de kits robóticos e seus alunos vêm sendo deixados de lado destas novas metodologias de ensino. Este fato traz a demanda por uma tecnologia nacional de baixo custo e robustez, que possa ser operada por professores sem treinamento específico e com baixo custo de manutenção. Sendo assim, este trabalho apresenta uma das fases de um projeto da Universidade de São Paulo – USP que visa produzir a tecnologia necessária para viabilizar a utilização de plataformas robóticas como ferramenta de ensino em escolas públicas.

Este trabalho descreve o projeto e implementação de um módulo de controle para uma plataforma robótica pendular de duas rodas, desenvolvido no Laboratório de Computação Reconfigurável (LCR) do Instituto de Ciências Matemáticas e de Computação (ICMC) da USP. Esse robô pendular nasceu como parte de um projeto educacional que visa despertar a curiosidade e o interesse de estudantes de escolas públicas no Brasil em relação à robótica (SIMÕES, 2008). O projeto educacional faz uso de robôs que possam ser comandados pelos estudantes com o intuito de resolver problemas didáticos de modo geral, incluindo jogos com interação entre múltiplos robôs em que as crianças são responsáveis pela navegação dos mesmos. Dadas as características dessas atividades e a interação com os estudantes, as plataformas robóticas devem ser suficientemente robustas. Assim, o

robô pendular, desenvolvido e patenteado pelo LCR-ICMC, possui seu chassi em forma de uma caixa retangular feita em chapas de aço de espessura 4 mm, à qual dois motores, suas respectivas rodas (rodas convencionais de bicicleta) e o conjunto de baterias estão acoplados. Esse *design* de robô mostrou-se bastante eficiente e flexível para navegação, mesmo em terrenos irregulares, sendo facilmente manobrável pelas crianças. A Figura 1 mostra o projeto do chassi da plataforma robótica pendular de duas rodas utilizada, no qual: (S) indica a caixa de aço e (C) a sua tampa, (R) mostra uma das rodas acopladas, (M) indica um dos dois motores e (D) sua caixa de redução, e (A) mostra a posição do conjunto de baterias (SIMÕES, 2008).



(a)

(b)

Figura 1: (a) Esboços do projeto e (b) imagem real da plataforma robótica pendular de duas rodas

Além dos dois motores e do conjunto de baterias, vai embarcado nessa plataforma robótica o módulo de controle desenvolvido, foco deste Trabalho de Conclusão de Curso. Por módulo de controle desse robô, inclui-se todo o conjunto *hardware/software* que permita ao mesmo receber comandos de

controle de um computador de mais alto nível, interpretá-los, processá-los e convertê-los em forma de movimento do robô através do devido acionamento de seus motores. Também fazem parte desse conjunto sensores de proximidade baseados em pulsos de ultrassom. Esses sensores de proximidade são responsáveis por identificar potenciais obstáculos na trajetória do robô. Seus sinais são lidos, filtrados e pré-processados pelo *software* de controle em tempo real, rodando no microcontrolador embarcado no robô. O *software* que roda nesse mesmo microcontrolador é também o responsável por receber os comandos de navegação enviados por um computador via tecnologia de comunicação sem fio *Bluetooth* (Bluetooth SIG). Esse computador (*notebook, smartfone, tablet* ou qualquer outro equipamento capaz de executar o programa e dotado de dispositivo *Bluetooth*), no qual futuramente um *software* de mais alto nível responsável pela navegação estará em execução, poderá estar também embarcado no robô ou posicionado em local remoto dentro do raio de alcance da comunicação *Bluetooth*. O microcontrolador, por sua vez, aciona devidamente os dois motores através do *driver* de corrente projetado neste trabalho.

Microcontroladores de sistemas embarcados em geral não trabalham com potência suficiente para acionar motores e atuadores diretamente, salvo em poucas exceções em que esses dispositivos são de baixíssima potência. A locomoção do robô pendular de duas rodas deste projeto é função de dois motores independentes de 24 V_{CC}. Seu microcontrolador, por sua vez, trabalha de 0 a 5 V_{CC}. Portanto, faz-se necessário o uso de uma interface que converta os comandos do microcontrolador para níveis de tensão e corrente que sejam capazes de acionar os motores. Essa interface é feita pelo *driver* de corrente, item que constitui o módulo de controle deste trabalho. Esse dispositivo recebe do microcontrolador um sinal de direção e um sinal de velocidade para cada um dos dois motores, além da alimentação (V_{CC} e Terra) para o lado de lógica *TTL* da placa (Balch, 2003). Os sinais de velocidades seguem a abordagem da modulação por largura de pulso (Pulse-Width Modulation – *PWM*) (Ahmed, 2000).

Embora existam *drivers* de motores disponíveis comercialmente, decidiu-se que fosse projetado e construído um dispositivo próprio para atender à corrente solicitada pelos motores desse robô que fosse de baixo custo, uma vez que se pretende construir um número grande de robôs para utilização em escolas públicas. Aliado a essa necessidade, foi levado também em consideração o intuito de serem concretizados conhecimentos teóricos adquiridos durante o curso.

Neste trabalho, optou-se por um modelo de microcontrolador Arduino. Assim como o Arduino, existem diversas plataformas microcontroladas que integram diversas funcionalidades e ferramentas do processo de programação de microcontroladores, implicando em simplicidade no uso dos mesmos. Mas a escolha do Arduino em projetos de sistemas embarcados tem sido frequente tanto por amadores, quanto por profissionais e professores, devido a diversas vantagens apresentadas por ele. Uma delas é a natureza *open-source* de seu projeto, implicando em sistemas microcontrolados de

muito baixo custo que estão em constante otimização. Todas as versões de Arduino possuem seus projetos, tanto de *hardware*, quanto de *software*, disponíveis para quem deseje construir seu próprio protótipo. Placas originais do Arduino UNO R3, fabricadas na Itália pela empresa Smart Projects (Smart Projects | About us), por exemplo, são encontradas no Brasil custando em torno de R$50,00; as versões denominadas "clone" desse mesmo modelo chegam a custar em torno de R$30,00 (custos levantados em agosto de 2014). Além disso, sua *IDE* (*Integrated Development Environment*, que em português significa Ambiente de Desenvolvimento Integrado), escrita em JAVA, tem característica multiplataformas, podendo ser executada nos sistemas operacionais Windows, Macintosh OSX e Linux. Esse ambiente de desenvolvimento é de uso intuitivo, de fácil entendimento para usuários iniciantes, embora também possa ser explorado mais afundo por usuários mais experientes. Além de sua característica *open-source* ou código aberto, seu *software* pode ser expandido e/ou modificado por usuários mais experientes com inclusão de bibliotecas em C++, ou ainda avançar através da programação em AVR-C. Originalmente seus projetos são baseados em microcontroladores ATMEGA8 e ATMEGA168, embora possam também ser substituídos por projetistas que queiram desenvolver suas próprias plataformas de microcontroladores baseados nas características do projeto Arduino (Banzi, M. , Cuartielles, D. *et al*.). Neste documento, far-se-á referência à plataforma microcontrolada Arduino como microcontrolador Arduino, como é comumente feito por projetistas e demais usuários dessa tecnologia.

A fonte de alimentação do robô móvel pendular deste trabalho é constituída por duas baterias de 12 V_{CC}, de 7 Ah de capacidade cada. A associação em série dessas duas baterias provê 24 V_{CC} necessários à alimentação da parte de potência do *driver*, que aciona os motores. Derivam-se 12 V_{CC} de uma das baterias para a alimentação do microcontrolador, que através de um regulador de tensão integrado à sua placa fornece 5 V_{CC} para os componentes *TTL* de sua própria placa e dos demais dispositivos conectados a ela: sensores de ultrassom, *driver* e também do dispositivo de comunicação *Bluetooth*.

Ao final deste projeto, o módulo de controle deverá permitir que o robô pendular navegue segundo os comandos enviados por um computador com *Bluetooth*. Entretanto, o cumprimento dos comandos recebidos pelo robô ficará sempre condicionado à avaliação da existência de obstáculos em sua trajetória com o intuito de proteger a si mesmo e a pessoas transitando. A continuidade deste projeto será baseada no desenvolvimento de um software de alto nível que codifique uma inteligência artificial capaz de permitir ao robô sua navegação autônoma dentro de um ambiente de trabalho, cumprindo tarefas pré-especificadas.

1.1. Objetivos

Um dos objetivos deste trabalho é o projeto e implementação de um módulo de acionamento de motores de corrente contínua, ou *driver*, para os dois motores de 24 V responsáveis pela locomoção do robô, incluindo a manufatura da placa de circuito impresso integrando todo o circuito. Cada um dos motores é conectado a uma das duas pontes-H na placa. O *driver,* que faz a interface entre microcontrolador e motores, deve ser capaz de receber dois sinais em nível *TTL* do microcontrolador para cada um dos dois motores: um sinal de direção ou sentido de rotação e um sinal de *PWM*, que resulta na velocidade do respectivo motor. São entradas para o *driver*, portanto, quatro sinais ao todo, além da alimentação para o lado *TTL* da placa (5 V). Além das duas pontes-H e das portas de entrada do *driver*, esse dispositivo deve conter um circuito que interprete os sinais de direção e *PWM* e, então, acione devidamente a respectiva meia-ponte. Cada meia-ponte, que somadas formam a chamada ponte-H, deve ser responsável pelo acionamento de seu respectivo motor em um único e determinado sentido. Serão apresentadas duas versões implementadas de *drivers*. Entretanto, a primeira versão, nomeada projeto I, não apresentou desempenho esperado nos testes de validação e, por essa razão, deu-se início ao projeto II, foco deste trabalho e que será abordado com mais detalhes.

Outro foco deste trabalho é a avaliação da viabilidade do uso dos sensores baseados em ultrassom, ou sonares, como ferramentas na estimação da distância entre o robô e obstáculos em sua trajetória. Esses módulos de sensor ultrassônico são compatíveis com microcontroladores baseados em Arduino e têm sido amplamente utilizados por profissionais e programadores amadores em seus projetos com Arduino, pois, entre outras características, possuem um baixo consumo de potência (cerca de 70 mW). Já existem bibliotecas especiais para o trabalho com esses sensores baseados em ultrassom, que podem ser baixadas no site oficial da comunidade Arduino e inclusas nos projetos através de sua *IDE*. Entretanto, essas bibliotecas foram desenvolvidas com base na operação de módulos de ultrassom de outro fabricante, de maneira que, para que não se corresse o risco de alguma incompatibilidade no acionamento e leitura do sonar utilizado, optou-se pelo desenvolvimento das rotinas próprias para o dispositivo utilizado (Rajput, 2012).

Também faz parte deste trabalho o desenvolvimento de um *software* de baixo nível, a ser executado em tempo real no microcontrolador embarcado na plataforma robótica, que faça a leitura dos sensores de ultrassom e o processamento de seus sinais. Além disso, esse *software* deve conter uma rotina que permita ao microcontrolador comunicar-se com um computador remoto via conexão *Bluetooth*, de onde virão os comandos de navegação para o robô. Esse *software* deve também interpretar esses comandos e gerar os respectivos sinais de direção e de *PWM* que acionarão os dois motores através do *driver*.

A comunicação do computador (que deve ser hábil para comunicação *Bluetooth*) com o microcontrolador é estabelecida através do uso de um módulo de transmissão e recepção *Bluetooth*

conectado ao Arduino, que faz a conversão entre esse protocolo sem fio e o protocolo padrão de comunicação serial desse microcontrolador.

1.2. Organização do documento

O capítulo 2 deste documento: “Projeto de *hardware* e *software* do módulo de controle do robô pendular de duas rodas” trata dos requisitos de projeto, do *design* e dos conceitos teóricos por trás de cada parte que integra o módulo de controle, tanto em nível de *hardware* quanto em *software*.

A manufatura das placas de circuito impresso das duas versões de *drivers* e a implementação prática dos demais componentes do módulo de controle do robô serão abordados no capítulo 3, nomeado “Implementação de *hardware* e *software* do módulo de controle do robô pendular de duas rodas”.

O capítulo 4 - “Resultados”, trata dos desempenhos e de resultados de testes de cada um dos componentes do módulo de controle do robô, bem como do funcionamento do sistema interligado.

Por fim, as conclusões obtidas a partir dos resultados alcançados neste trabalho serão expostas no capítulo 5. A continuidade deste projeto será um assunto também abordado nesse capítulo.

2. Projeto de *hardware* e *software* do módulo de controle do robô pendular de duas rodas

Este capítulo explica o funcionamento teórico de cada componente que integra o módulo de controle do robô. São evidenciados as necessidades que justificam a existência de cada componente e os requisitos de projeto previamente estabelecidos para o devido funcionamento da plataforma robótica. A Figura 2 sumariza a integração do módulo de controle projetado e implementado, cujas partes integrantes serão detalhadas.

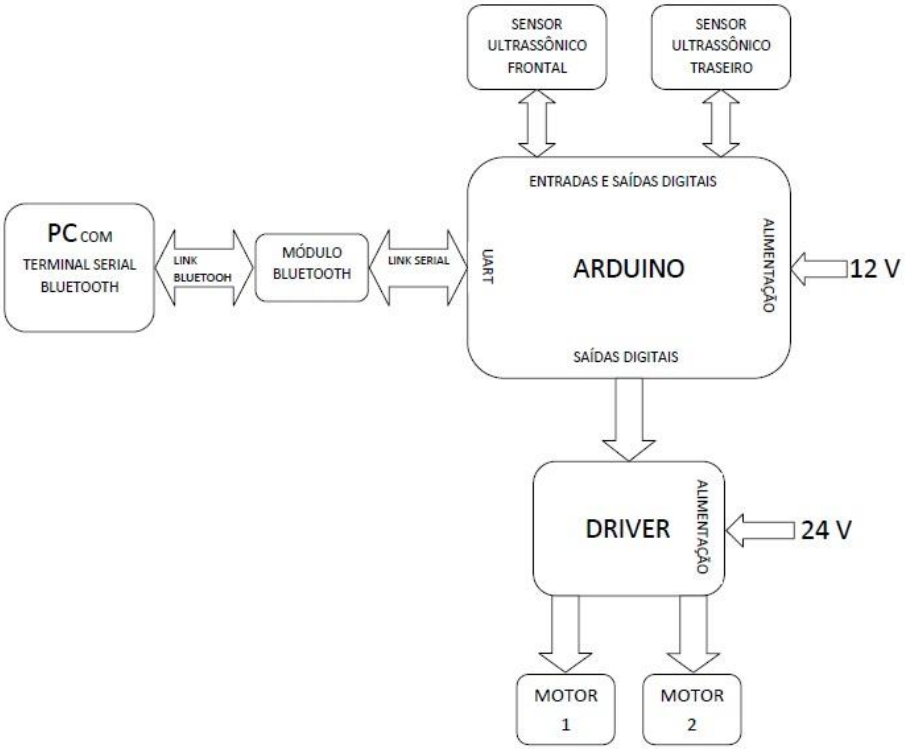


Figura 2: Diagrama esquemático do módulo de controle.

2.1. *Driver* dos motores

Os dois motores responsáveis pela locomoção do robô pendular são fabricados pela Bosh, modelo CEP F 006 WM0 310 (Datasheet F006 WM0 310). Trata-se de um motor de corrente contínua, de tecnologia *brushless*. Embora ele tenha sido desenvolvido para suprir a indústria automotiva no acionamento de limpadores de para-brisas, ele é amplamente utilizado em inúmeras aplicações envolvendo automação e robótica. Opera com tensão nominal de 24 V e corrente nominal de 5 A , embora essa dependa da carga imposta ao eixo do motor. Para dimensionamento do circuito do *driver*, fez-se necessário estimar o valor de corrente com a qual se trabalharia. Sendo esse um requisito de projeto, ensaios foram feitos para se determinar a corrente solicitada pelos dois motores

quando esses aplicavam o torque necessário para o robô locomover-se. O procedimento desse teste está descrito no capítulo 3.

Constatou-se que os motores trabalham com um valor de corrente bem abaixo da nominal, em torno de 1,1 A quando em velocidade máxima e com todo o conjunto de componentes do robô já embarcados, o que constituí uma carga para os motores. Na situação mais crítica, quando o robô sai do repouso e acelera quase que instantaneamente até sua velocidade máxima (aplicando-se diretamente 24 V nos seus terminais), o pico de corrente observado com duração de cerca de 1 segundo foi em torno de 2,4 A. Na prática, essa situação de partida direta é evitada em *software*, com implementação de aceleração em rampa. Considerando-se os dados obtidos, por segurança estabeleceu-se dimensionar as trilhas do *driver* para que suportassem a corrente nominal do motor, de 5 A, visto ser possível que eventualmente mais carga seja adicionada ao chassi do robô. O dimensionamento das trilhas foi feito com uso de um *toolbox* de desenvolvimento de placas de circuito impresso disponível na *webpage* da *Electrical Engineering Comunity* (External PCB Trace Width).

Os motores CC empregados, assim como a maioria dos motores CC, podem girar nos dois sentidos, dependendo apenas da polaridade aplicada nos seus terminais. Invertendo-se a polaridade, inverte-se o sentido. A topologia eletrônica denominada ponte-H, ou ponte completa, é uma estratégia largamente empregada para se fazer o acionamento de motores CC de modo que a inversão da polaridade da tensão aplicada aos terminais dessas cargas seja facilmente realizada (Inoue & Takao, 2004). Conceitualmente, a configuração de uma ponte-H está apresentada na Figura 3, em que a conexão dos terminais positivo e negativo são feitas com chaves genéricas. A semelhança dessa topologia com a letra H é a razão de seu nome.

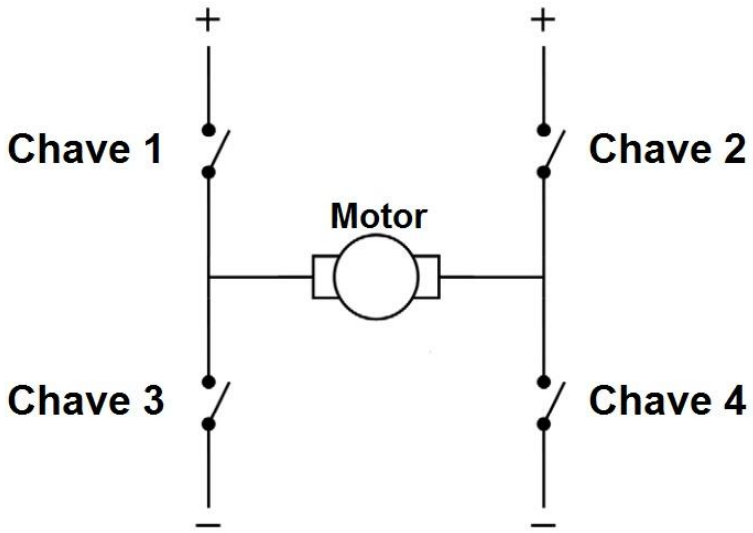


Figura 3: Conceito de ponte-H

Suponha que o motor gire no sentido horário quando passa por ele uma corrente que, de acordo com a Figura 4, flui da esquerda para a direita. Nesse caso, as chaves 1 e 4 devem estar fechadas, enquanto as chaves 2 e 3 devem permanecer abertas. Nessa configuração, ter-se-ia o polo positivo aplicado ao terminal esquerdo do motor e o negativo ao direito. De modo análogo, o motor gira em sentido anti-horário quando as chaves 2 e 3 estão fechadas e as chaves 1 e 4 abertas. Essa situação é ilustrada na Figura 5.

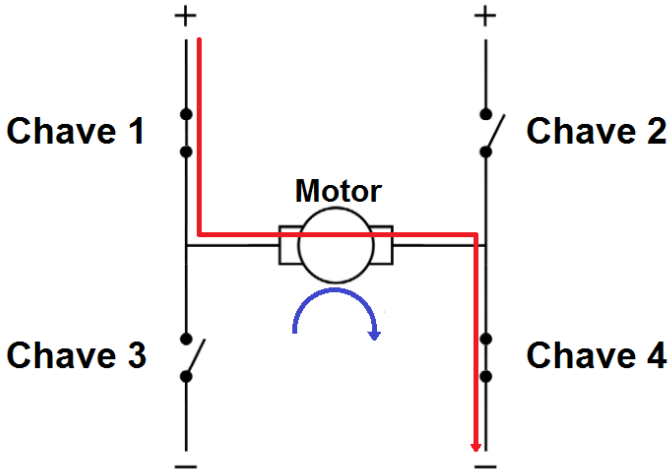


Figura 4: Meia-ponte fechada pelas chaves 1 e 4

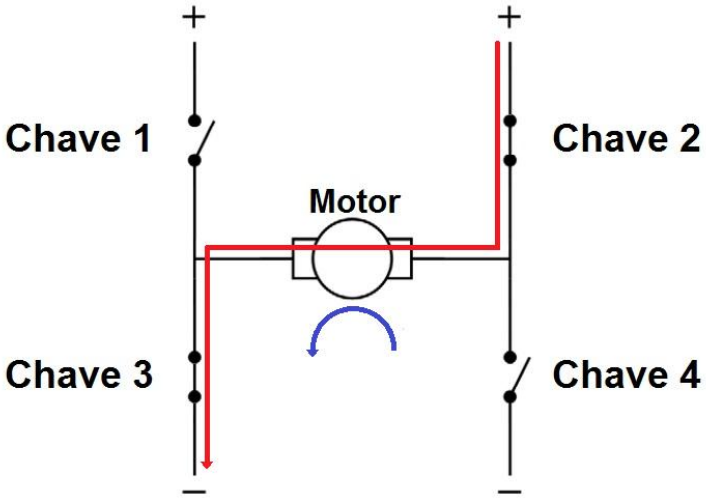


Figura 5: Meia ponte fechada pelas chaves 2 e 3

Existem diversos modos de operação da ponte-H. Contudo, independentemente do modo de operação, o que deve ser garantido é que as duas chaves do mesmo lado da ponte não sejam fechadas simultaneamente, pois essa situação caracterizaria um curto-circuito na fonte. Dependendo da aplicação, o componente que faz o papel de chave deve ser escolhido apropriadamente. Podem ser

chaves mecânicas, como relés, ou componentes semicondutores, como transistores de junção bipolar (*BJT*), transistores de efeito de campo (*MOSFET*), entre outros. O modo de acionamento dessas chaves, a velocidade de operação delas, bem como limitações de corrente e dissipação de potência são fatores considerados na escolha da chave adequada.

Por tratar-se de um *driver* de potência que trabalharia numa frequência relativamente alta, chaves mecânicas não se aplicariam neste projeto. Um componente adequado a esta aplicação, embora não único, é o *MOSFET* de potência (Sedra & Smith, 2000). A Figura 6 apresenta a topologia de ponte-H utilizada com seus componentes especificados, de modo que o motor é conectado aos terminais marcados com um "X". As funções das chaves 1, 2, 3 e 4 são desempenhadas pelos *MOSFETs* Q_1, Q_2, Q_3 e Q_4, respectivamente. As funções dos diodos D_1, D_2, D_3 e D_4 serão descritas posteriormente.

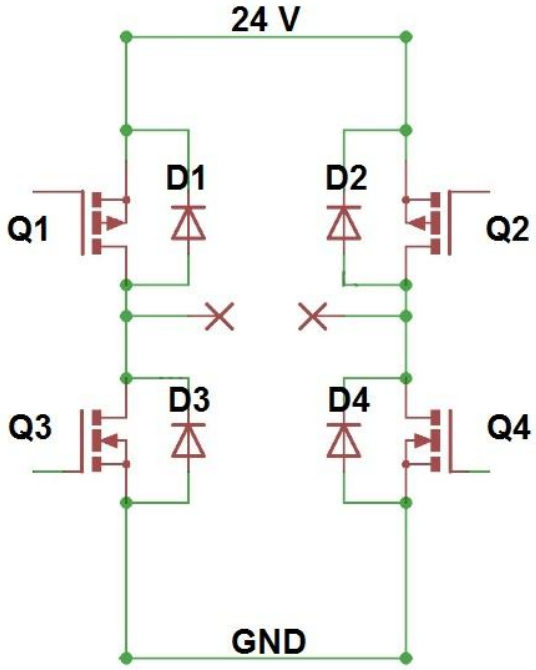


Figura 6: Topologia de ponte-H do projeto

Os *MOSFETs* de potência da parte superior da ponte-H diferem daqueles na parte inferior. Q_1 e Q_2 são denominados *MOSFETs* do tipo P, ou *PMOS*, enquanto Q_3 e Q_4 são denominados *MOSFETs* tipo N, ou *NMOS* (Barbi, 2006) (Sedra & Smith, 2000). As denotações P e N referem-se ao tipo de dopagem do material semicondutor do corpo desses transistores. As regiões de operação dos *MOSFETs* que interessam a esta aplicação são as regiões de corte, - análoga à chave aberta, e saturação, - análoga à chave fechada. Para esclarecer como colocá-los nesses modos de operação, observe a Figura 7, que indica as três principais partes dos *MOSFETs*, a saber: *drain*, *source* e *gate*,

que chamadas aqui de dreno, fonte e comporta, respectivamente. À esquerda tem-se um PMOS e à direta um NMOS.

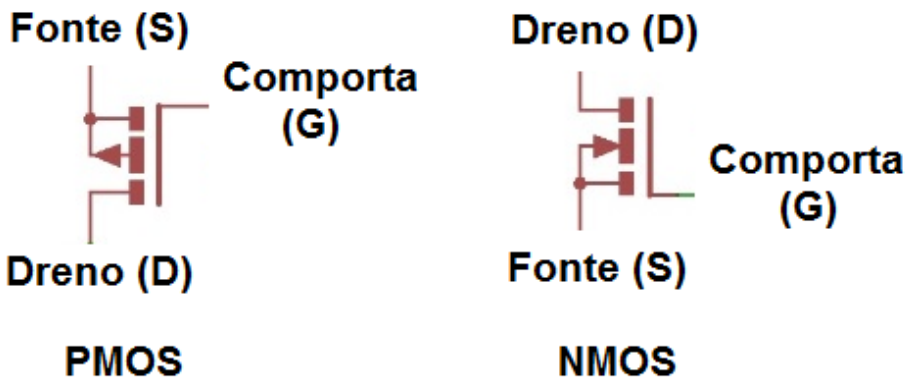


Figura 7: Representação gráfica dos MOSFETs tipo-P (à esquerda) e tipo-N (à direita)

De modo geral, o modo de acionamento de um MOSFET se dá pela diferença de tensão entre os terminais fonte e comporta, ou (S) e (G). Genericamente, para operar no modo de corte (chave aberta), essa diferença de tensão deve ser a menor (em módulo) possível, que geralmente varia entre 0 e 4 V. Para operação no modo de saturação (chave fechada), essa diferença deve ser grande o suficiente, que para vários modelos é de no mínimo 12 V.

Referindo-se novamente à Figura 6, tem-se que o terminal fonte ou (S) de um MOSFET tipo P está conectado ao polo positivo da fonte, de 24 V. Portanto, para operação no modo de corte, deve-se garantir no terminal comporta (G) no mínimo 20 V. Na prática, por segurança, projeta-se para que essa tensão de acionamento para modo de corte seja muito próxima de 24 V. Em contrapartida, para operação em modo de saturação, deve-se garantir no terminal comporta (G) no máximo 12 V. Na prática, para garantir que o transistor esteja na região de saturação, projeta-se para que essa tensão chegue próxima a 0 V.

Mais uma vez, observando-se a ponte-H na Figura 6, observa-se que o terminal (S) de um transistor NMOS está conectado ao polo negativo da bateria, ou GND. Assim, para que opere no modo de corte, o terminal comporta (G) deve estar em um potencial de no máximo 4 V, preferindo-se projetar para que fique o mais próximo de 0 V. De outro modo, para operação no modo de saturação, leva-se o terminal comporta (G) para um potencial de, no mínimo, 12 V. Na prática, opta-se por chegar o mais próximo de 24 V, ampliando-se bastante a diferença de potencial entre os terminais G e S.

A natureza indutora de um motor de corrente contínua, que é a carga a ser acionada pela ponte-H, não permite que uma corrente fluindo por suas bobinas seja interrompida instantaneamente, o que aconteceria se o circuito da Figura 5 abrisse as chaves 3 e 2, por exemplo. Se isso acontecesse, a tensão nos terminais do motor, diretamente proporcional à taxa de variação da corrente, tenderia a infinito, podendo queimar os transistores. A fim de corrigir esse problema, os diodos D_1, D_2, D_3 e D_4 são adicionados em paralelo com aos *MOSFETs*, com polarização reversa. Desse modo, quando houver o desligamento de um dos lados da ponte, seja para inversão do sentido ou simplesmente para parar o motor, os diodos fornecem um caminho alternativo para que a corrente remanescente no motor se dissipe no circuito (Barbi, 2006) (HV Floating MOS-Gate Driver ICs).

Suponha que o circuito real da ponte-H (Figura 6) esteja na configuração da Figura 5: Meia ponte fechada pelas chaves 2 e 3. No instante em que os transistores 2 e 3 (Q_2 e Q_3) cortarem, o circuito indutivo do motor forçará que a corrente continue fluindo da direita para a esquerda. Mas como todas as chaves estão abertas, o caminho tomado pela corrente é partindo do GND, passando por D_4 e D_1 até o polo positivo, devolvendo parte de energia à fonte. O intervalo de tempo durante o qual isso ocorre é muito curto, praticamente igual ao tempo levado para que o motor pare de girar (Barbi, 2006) (HV Floating MOS-Gate Driver ICs).

Uma vez que as devidas considerações sobre o funcionamento e operação da ponte-H foram abordadas, as seções seguintes exibem os fundamentos dos dois projetos de acionamento das pontes-H do *driver*.

2.1.1. Circuito de controle das pontes-H: projeto I

O primeiro projeto feito para o devido acionamento das pontes-H fazia uso de um circuito lógico sequencial com portas lógicas e *flip-flops* tipo-D (*FF's* tipo-D ou simplesmente *FF's-D*) (Tocci, 2011). Esse circuito lógico foi projetado de modo a ser sincronizado por um sinal de *clock*, que nesse projeto também era uma entrada do *driver*. Portanto, além da alimentação e das quatro entradas mencionadas anteriormente (dois sinais de direção e dois sinais de *PWM*), essa versão de *driver* possuía uma entrada adicional, o *clock*, fornecida pelo microcontrolador. A parte lógica que controla uma das pontes-H está simplificada na Figura 8 (o esquemático completo está em anexo: apêndice C).

Conceitualmente, o circuito foi projetado de modo que a realimentação de *flip-flops* garantisse que, antes que uma inversão de sentido solicitada pelo microcontrolador fosse transmitida à ponte, ocorresse um estado de abertura de todas as chaves da ponte-H durante um intervalo de tempo de um *clock*. Esse *design* serviu, entretanto, principalmente para melhor entendimento do funcionamento da ponte-H. Apesar de operar dentro do esperado em regime de trabalho normal, este circuito apresentou algumas falhas de funcionamento para transitórios como quedas na tensão de alimentação devido a travamento de eixo no motor, que podiam fazer com que os dois flip-flops acionassem os transistores

do mesmo lado da ponte H simultaneamente. Isso ocasionava a queima dos mesmos devido à grande corrente que circulava entre Vcc (24V) e o terra. Por essa razão, este projeto foi abandonado em função de uma solução mais robusta, apresentada a seguir.

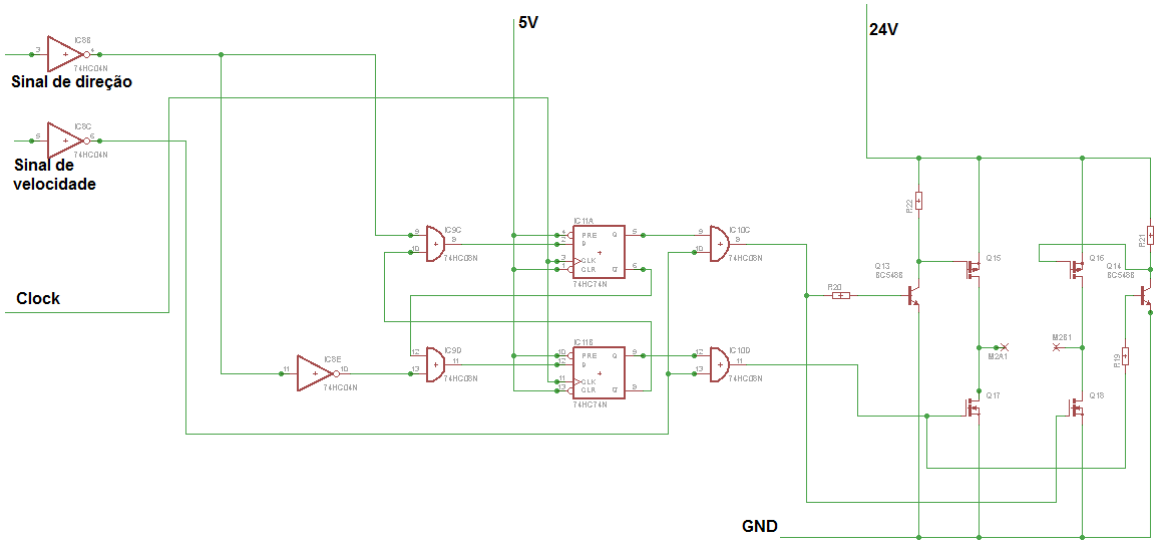


Figura 8: Circuito lógico sequencial no acionamento da ponte-H

2.1.2. Circuito de controle das pontes-H: projeto II

O segundo projeto do circuito de acionamento das pontes-H deixou de lado a ideia de trabalhar-se com um circuito lógico sequencial. Como já mencionado, embora se possa operar uma ponte completa de diferentes métodos, neste projeto de *driver*, os pares de chaves 1 - 4 ou 2 – 3 serão acionados sempre ao mesmo tempo. Portanto basta que o sinal de velocidade, que fará o chaveamento dos transistores segundo a modulação por largura de pulso, seja acoplado ao par 1 – 4 ou ao par 2 – 3. O par de chaves sem sinal de *PWM*, ou de modo análogo, com sinal de velocidade em zero, deverá estar em estado de corte. O modo adotado para se fazer a seleção de qual par de chaves deve ser acionado em função do sinal de direção foi o de fazer uso de um relé. Essa estratégia é ilustrada no esquemático da Figura 9.

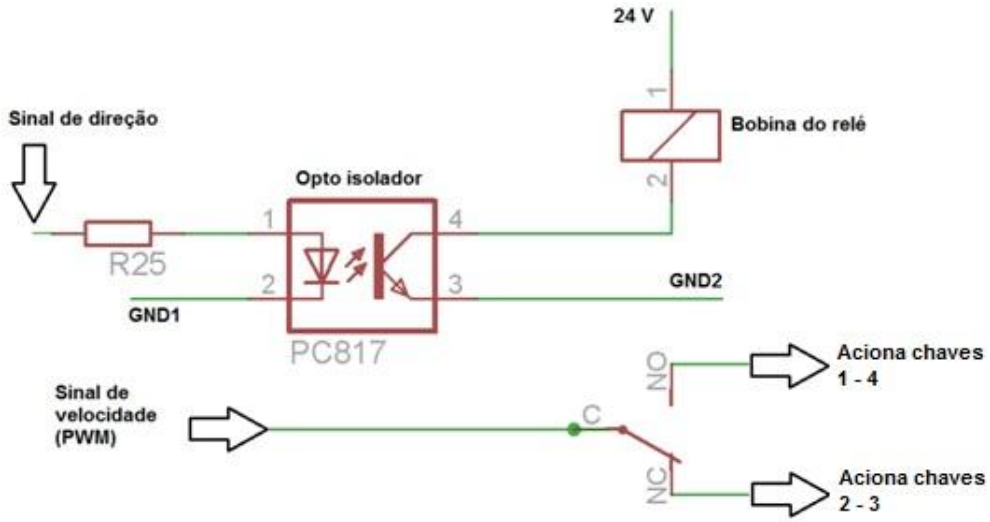


Figura 9: Sentido de rotação e chaveamento do relé

O sinal de direção aciona a bobina do relé através de um opto acoplador ou opto isolador, que tem a função de isolar eletricamente o lado do *driver* que recebe os sinais do microcontrolador do lado de potência da placa. Quando o sinal de direção está em tensão 0 V (nível lógico zero), o *LED* do opto isolador não emite luz e o foto-transistor está em corte (Optocoupler Datasheet_LTV8x6). Nessa situação, não flui corrente pela bobina do relé e, portanto, sua chave conecta o terminal C (comum) ao terminal NC. O sinal de *PWM*, conectado ao terminal C, é enviado a um dos pares de chaves da ponte-H. Em contrapartida, quando o sinal de direção está em 5 V (nível lógico 1), o *LED* do opto isolador emite luz e satura o foto-transistor, fazendo com que circule corrente pela bobina do relé. Nessa situação, a chave do relé muda de posição e conecta o sinal de *PWM* ao terminal NO, que por sua vez está ligado ao outro par de chaves da ponte-H.

A ponte-H propriamente dita também é acionada pelo sinal de *PWM* através de opto-acopladores, que além da finalidade de proteção do lado da placa conectado ao microcontrolador, também permite que os *MOSFETs* da ponte sejam acionados por sinais de 0 a 24 V, o que garante sua devida operação nos modos de corte ou saturação. A Figura 10 mostra o esquemático quase completo (haverá mais uma modificação decorrente dos testes de validação do dispositivo) desse segundo projeto de *driver* (que pode ser visto com mais clareza na folha anexa: apêndice A), e a Figura 11 evidencia os optos isoladores conectados à ponte.

Além dos componentes já descritos, para melhor condicionamento dos sinais recebidos do microcontrolador, o *driver* conta com um *buffer* de entrada 74LS244 (Datasheet 74LS244).

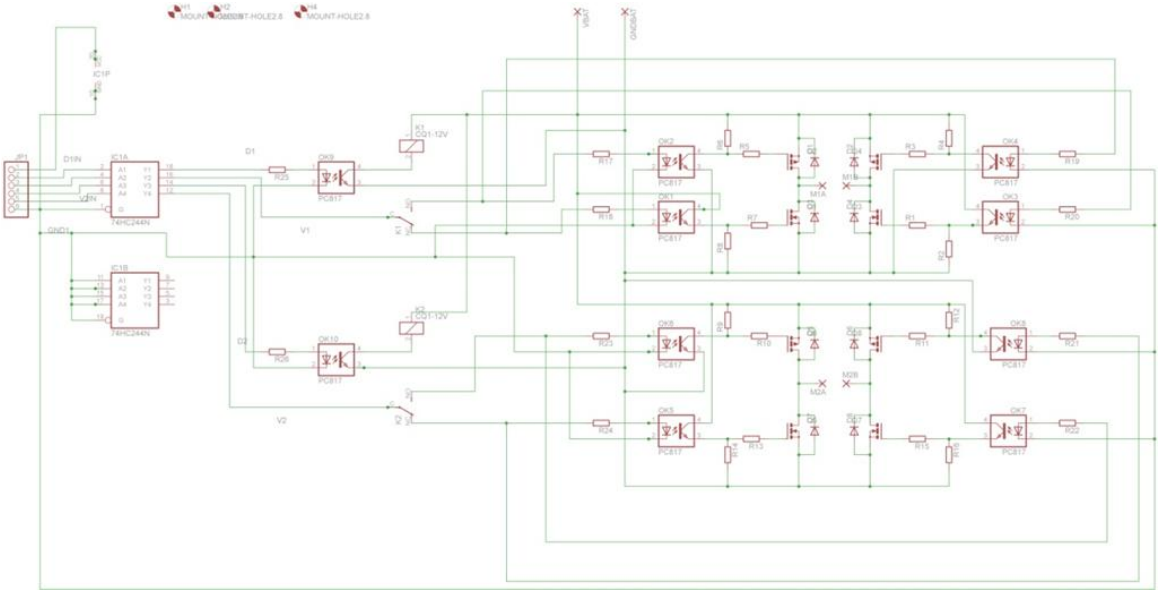


Figura 10: Esquemático do segundo driver (ver Apêndice A)

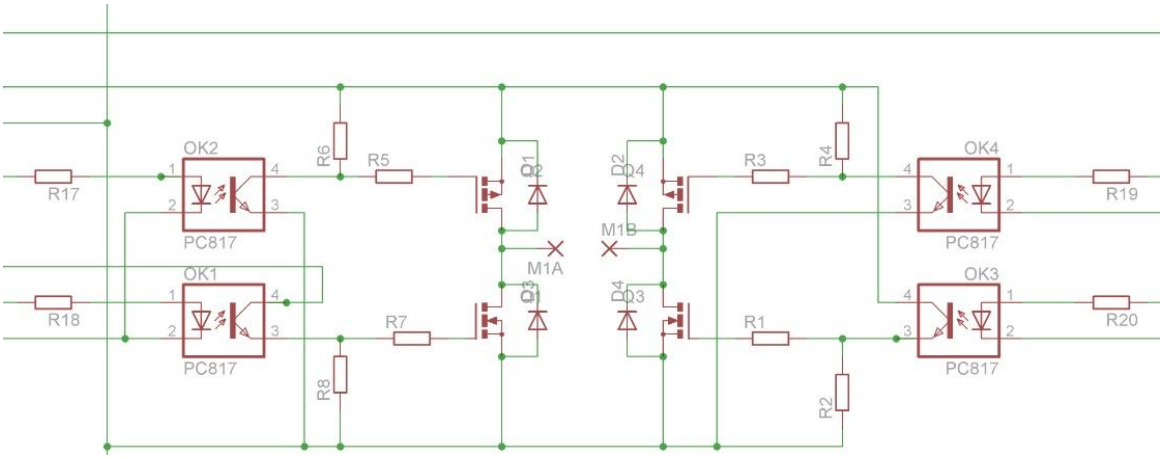


Figura 11: Detalhe da ponte-H do segundo projeto de driver

2.2. Sensor ultrassônico

Com o intuito de proteger o robô contra colisões, o sensor escolhido para aferição de distâncias trabalha com tecnologia de emissão de pulsos ultrassônicos e recepção dos pulsos refletidos, como é feito por alguns animais na natureza. Conhecendo-se a velocidade de propagação do som no ambiente de trabalho do sensor (em torno de 340 m/s), através do intervalo de tempo entre a emissão e a recepção de um dado pulso é possível calcular-se a distância entre a fonte e o obstáculo (Rajput, 2012).

Quando uma leitura é solicitada ao sensor, esse emite um trem de pulsos numa determinada frequência e aguarda pela recepção dessa mesma sequência de pulsos. Para essa identificação, o

circuito de recepção faz uso de um sensor pizoelétrico que compara os sinais recebidos com os emitidos (HC-SR04 Datasheet, 2012).

Uma das vantagens do uso desse módulo de sonar é o fato de seu desempenho não ser afetado pela luz solar, ou pela existência de corpos negros que absorvem o espectro de luz usado em sensores baseados em reflexão de ondas eletromagnéticas, como sensores de infravermelho. Outra vantagem é conveniência de ter-se integrados numa única placa os circuitos de emissor e receptor, bem como de um circuito de controle. Apenas basta que o microcontrolador, que faz a leitura do sensor, dê o devido comando de acionamento do sensor e leia sua saída, que é um pulso único de largura proporcional à distância do obstáculo (HC-SR04 Datasheet, 2012).

Figura 12: Módulo de sensoriamento por ultrassom

2.3. Modulação por largura de pulso - *PWM*

A modulação por largura de pulsos, cujo acrônimo *PWM* usualmente difundido significa *Pulse-Width Modulation*, é uma técnica aplicada em geral no controle do valor médio de uma forma de onda genérica. Será abordado aqui, entretanto, o uso dessa técnica no controle de velocidade de motores CC de pequeno porte, categoria em que se encaixam os motores responsáveis pela locomoção da plataforma robótica deste trabalho.

A velocidade desempenhada por um motor CC depende do valor médio da tensão aplicada em seus terminais. A modulação por largura de pulso permite que se faça o controle desse valor médio de tensão sem, contudo, que se modifique o valor da tensão nominal aplicada aos terminais do motor. Ao invés de se aplicar um valor de tensão CC constante ao motor, essa técnica entrega a ele pulsos

retangulares com tensão igual à sua tensão nominal, como se estivéssemos ligando-o e desligando-o repetidamente. Entretanto, a velocidade desse ligar e desligar deve ser suficientemente alta para não permitir que o torque do motor chegue a zero, isto é, a frequência da onda retangular deve ser suficiente para que o motor não pare de girar durante o intervalo de tempo em que a onda retangular está em 0 V. (Ahmed, 2000).

A largura dessa onda retangular, ou de outro modo, a largura desse pulso em nível alto é diretamente proporcional ao nível médio de tensão enxergada pelo motor (Ahmed, 2000). À razão entre a largura de pulso e o período de repetição dessa onda deu-se o nome de razão cíclica, ou *duty-cycle*. A Figura 13 exemplifica esses conceitos.

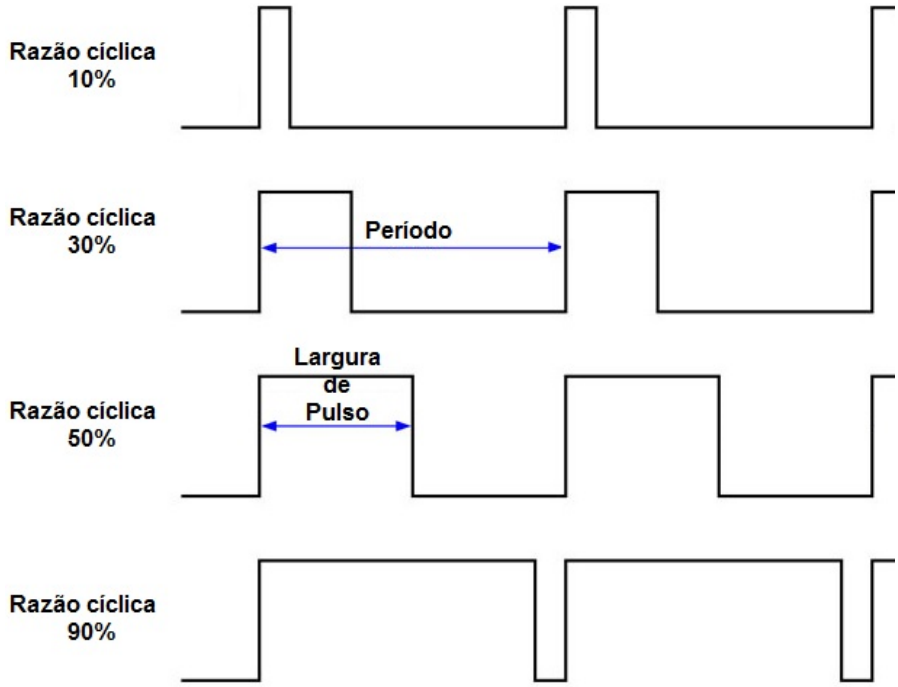


Figura 13: Definição da razão cíclica de um sinal PWM

A grandeza razão cíclica (DC) é calculada como:

$$d = {}^{T_{ON}}/_{T} \qquad \textbf{Equação 1}$$

Tal que T_{ON} é largura do pulso e T é o período da onda retangular, ambos em unidades de tempo. Assim, a relação entre a tensão média observada pelo motor (V_O) e a sua tensão nominal (V_I) é dada por:

$$V_O = d.V_I \qquad \textbf{Equação 2}$$

A frequência da onda modulada por largura de pulso deve estar de acordo com a carga a ser acionada, neste caso, um motor CC. A frequência de operação deve ser tal que permita que a corrente

no motor não seja composta por harmônicos que impliquem em seu mau funcionamento, o que implica em garantir que a largura de pulso da corrente seja a mesma. Na prática, deve-se observar que o motor gire sem torque pulsante, suavemente.

2.4. Módulo de comunicação *Bluetooth*

O dispositivo usado para permitir a comunicação do Arduino com o computador é um módulo de conversão entre os protocolos *UART* 9600 bps, com um *start bit*, oito *bits* de dado, um *stop bit* e sem paridade, e o protocolo *UART Bluetooth* (Bluetooth UART Interface Datasheet).

Devido à sua característica de dispositivo escravo, necessita que a comunicação seja iniciada pelo dispositivo mestre, que no caso deste projeto será um computador hábil a enviar caracteres ASCII (Tocci, 2011) via porta serial com conversor *Bluetooth*. Não seria possível, por exemplo, que dois microcontroladores se comunicassem via link Bluetooth com o uso de dois desses módulos, por exemplo.

Esse módulo de conversão pode comunicar-se com qualquer dispositivo mestre que possua um conversor *Bluetooth* embarcado ou *built-in*, como *laptops*, *tablets* e *smartfones*. No caso de computadores mais antigos, existem adaptadores externos, chamados de *Bluetooth dongles*, que transformam qualquer porta *USB* em um emissor/receptor *Bluetooth*.

2.5. *Software* do microcontrolador embarcado

O *software* a ser executado em tempo real no microcontrolador da plataforma robótica deve constantemente ler sua porta serial com o intuito de receber os comandos de navegação do robô, transmitidos por um computador. Deve ser capaz de interpretar os comandos recebidos e, então, gerar sinais de *PWM* e de direção que serão responsáveis por acionar cada um dos dois motores de locomoção do robô pendular através do *driver* de corrente.

A interpretação e execução dos comandos de navegação, entretanto, devem ficar condicionadas à existência ou não de obstáculos na trajetória estimada do robô, de modo que sua integridade e a segurança de pessoas em seu ambiente de trabalho sejam garantidas. Portanto, este *software* deve, periodicamente, acionar e fazer a leitura dos sensores ultrassônicos, responsáveis pela aferição da distância entre o robô e possíveis obstáculos em seu trajeto. Havendo um obstáculo suficientemente próximo do robô, o microcontrolador deve pará-lo, configurando um reflexo condicionado de prioridade máxima (Simões, 2000). O desenvolvimento mais detalhado do *software* será descrito no capítulo 3, no qual suas necessidades e funções são apresentadas durante a fase de implementação.

3. Implementação de *hardware* e *software* do módulo de controle do robô pendular de duas rodas

Este capítulo aborda o desenvolvimento prático dos componentes do módulo de controle da plataforma robótica, incluindo *hardware* e *software*. O correto funcionamento individual de cada um desses componentes do módulo, cujos fundamentos teóricos foram discutidos no capítulo anterior, é essencial para o devido funcionamento e bom desempenho do robô pendular como um todo. Os testes de validação desses componentes serão também descritos.

3.1. Prototipagem do *driver*: projeto I

O esquemático do circuito eletrônico da primeira versão do *driver*, cujos fundamentos do projeto já foram apresentados, foi implementado no *software* de projeto e edição de *layouts* de placas de circuito impresso chamado *Eagle*, desenvolvido pela CadSoft (About Us | CadSoft).

Uma vez que o circuito de interesse é desenvolvido no ambiente de esquemático, esse programa cria um projeto de *layout* de placa a partir dele, com todas as devidas conexões elétricas e dimensões reais dos componentes eletrônicos e seus respectivos *footprints*, nome dado aos pontos de soldagem. Há uma grande gama de informações de componentes eletrônicos necessárias a um projeto de *layout* em bibliotecas carregadas por esse *software*. Entretanto, caso o usuário esteja trabalhando com algum componente não existente nessas bibliotecas, o mesmo pode criar o *footprint* referente a esse componente. A disposição ou posicionamento de cada componente na placa fica por conta do usuário, mas o roteamento das trilhas que fazem as conexões elétricas pode ser feita automaticamente pelo programa, de acordo com predefinições estabelecidas pelo projetista, como largura de e distância mínima entre trilhas. Mais informações sobre esse programa podem ser encontradas na *webpage* do fabricante (About Us | CadSoft).

Antes que se iniciasse o desenvolvimento do *layout*, entretanto, fez-se a prototipagem do circuito numa placa de prototipagem (geralmente conhecida como *protoboard*), de modo que o esquemático do circuito desenhado previamente no *Eagle* foi de grande ajuda como referência para as devidas conexões. A placa de prototipagem utilizada, que pode ser vista na Figura 14, tem ainda um conjunto de chaves para geração de sinais binários *TTL*, o que facilitou o estudo do circuito lógico sequencial.

Esse circuito lógico aciona um par de chaves (par de chaves que constitui uma meia-ponte, e.g. chave 1 e chave 4, Figura 3) de cada vez, conceito este que também foi aplicado no segundo projeto do *driver*. Além disso, a realimentação de *flip-flops* tipo-D, sincronizados por um sinal de *clock*, faz com que ocorra um estado de desligamento dos dois lados da ponte sempre que o microcontrolador solicite uma inversão de sentido de rotação dos motores. Desse modo, ainda que em

software essa solicitação de inversão seja instantânea, o *hardware* impõe um atraso na execução dessa inversão, cujo intervalo de tempo é igual a um ciclo de *clock*.

Durante a prototipagem do circuito lógico sequencial em *protoboard*, observou-se o devido funcionamento do mesmo. Para averiguação adequada dos resultados, todos os sinais foram gerados pelas chaves da placa de prototipagem, de modo que se pudesse conferir as saídas das portas lógicas a cada chaveamento. Contudo, a desvantagem desse método foi a impossibilidade de se avaliar o funcionamento do circuito com sinal de velocidade modulado por largura de pulso. Assim, quando se fez a conexão do circuito lógico à ponte-H, observou-se o devido funcionamento da mesma, porém, em modo *ON-OFF*, isto é, com duty-cycle 100% ou 0%. Os erros ocorridos em decorrência dessa abordagem serão descritos no capítulo de resultados.

As fases seguintes deste primeiro projeto de *driver*, que são basicamente a geração do *layout*, a manufatura da placa de circuito impresso a partir dele e a soldagem dos componentes, são idênticas às respectivas fases do segundo projeto de *driver*, que será abordado em sequência.

Figura 14: Protoboard com gerador de sinais

3.2. Prototipagem do *driver*: projeto II

Como previamente abordado, este segundo projeto de *driver* descartou o uso de circuitos lógicos sequenciais no controle da ponte-H. Entretanto, manteve-se a ideia preliminar de se acionar as chaves da ponte aos pares (chaves 1 e 4 ou chaves 2 e 3), tal que não seja possível acionar os dois pares ao mesmo tempo. O componente que faz a seleção de para qual par de chaves o sinal de *PWM* é enviado é um relé de 24 V (HJR-3FF-S-H Datasheet).

Depois de esboçado o circuito em papel, fez-se o esquemático detalhado do mesmo no *software Eagle*. Usando-se esse esquemático como referência (que pode ser visto na Figura 10 ou nas

folhas de projeto no Apêndice A), implementou-se o circuito na placa de prototipagem da Figura 14, que possui geradores de sinais *TTL*. É importante salientar que, antes da conexão de cada componente do circuito, fez-se o devido dimensionamento dos resistores. Esse dimensionamento, além de baseado nas especificações de corrente obtidas nas folhas de dados de cada componente, foi fundamentado em testes empíricos. Por exemplo, os resistores conectados aos optos acopladores foram dimensionados com base em testes de operação em corte e saturação dos foto-transistores de saída (Optocoupler Datasheet_LTV8x6).

Outros exemplos de testes foram a verificação do acionamento da bobina do relé conectada ao coletor do foto-transistor do opto acoplador, bem como do acionamento dos *MOSFETs* da ponte-H através dos optos acopladores. Neste *design* de *driver* observou-se o devido controle de operação dos *MOSFETs* de potência com as faixas de tensões adequadas para colocá-los em corte ou saturação, segundo especificações das folhas de dados dos componentes escolhidos (IRF640 Datasheet) (IRF9640 Datasheet).

Observando-se a Figura 11, nota-se que lógica de acionamento dos *MOSFETs* de potência tipo-P (parte superior da ponte) é diferente do modo como se aciona os *NMOS* (parte inferior da ponte). Para que sejam chaves normalmente abertas, as comportas (G) dos *PMOS* devem estar em 24 V, mesmo potencial de suas fontes (S). Em contrapartida, para funcionarem como chaves normalmente abertas, os transistores *NMOS* devem ter suas comportas (G) no potencial 0 V. Do modo como se vê as conexões dos optos acopladores aos respectivos *MOSFETs* tem-se essa lógica assegurada, de modo que seu desempenho foi comprovado com a prototipagem em placa provisória.

Uma vez finalizada a fase de testes em placa de prototipagem, veio a fase de desenvolvimento do *layout*, cujo ambiente de projeto, como já brevemente descrito, é gerado a partir do esquemático. O impasse dessa fase foi a limitação na disponibilidade de alguns recursos do *Eagle* devido ao fato de que se fez uso da versão livre e gratuita deste *software*. Essa versão, por exemplo, tem um limite dimensional dentro do qual se pode dispor os componentes e fazer-se o roteamento das trilhas. Devido à existência de muitos componentes (ver projeto em anexo: apêndices A e B), foi necessário que a placa de circuito impresso fosse projetada em duas camadas. Essa estratégia tornou possível, mas ainda não simples, a geração de um *layout* funcional. O resultado pode ser observado no projeto em anexo (Apêndice B).

A partir do *design* pronto, manufaturou-se a placa de circuito impresso manualmente pelo método de transferência térmica do *layout* para a placa de cobre e corrosão com ácido perclórico. A Figura 15 resume os resultados de etapas desse processo. Detalhes sobre esse processo de manufatura de placas de circuito impresso podem ser encontrados em (Tutorial PCI).

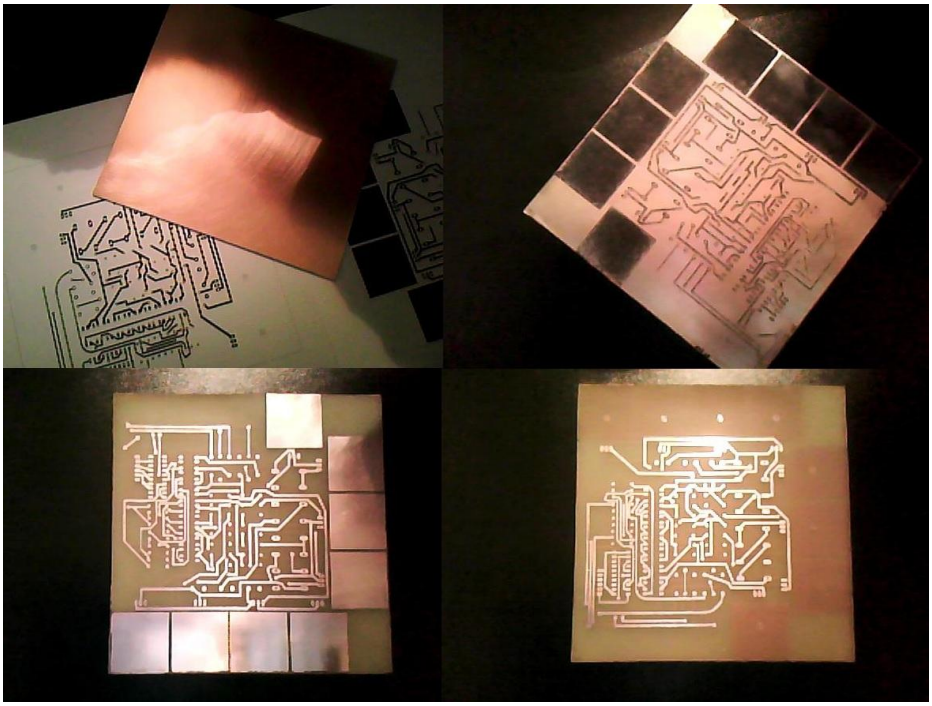


Figura 15: Resultados resumidos das etapas de manufatura da PCI

As etapas seguintes foram a furação da placa e soldagem dos componentes e vias, como são chamadas as conexões elétricas que ligam trilhas da face superior a trilhas da face inferior da placa. Uma vez que esse processo foi concluído, inspecionou-se a placa em procura de possíveis curtos-circuitos ou outras falhas decorrentes da construção manual da placa, como excesso de corrosão de algumas trilhas e pontos de soldagem de pinos de componentes, corrigindo-se essas falhas quando encontradas.

Quando finalizadas a inspeção e a correção, iniciou-se a fase de testes do *driver*. Inicialmente, fez-se o acionamento do dispositivo sem a conexão de cargas à ponte-H de modo a observar a devida propagação dos sinais de comando até o devido chaveamento da ponte. A geração desses sinais ocorreu a partir das chaves da placa de prototipagem e a alimentação do lado de potência do *driver* foi fornecida por uma fonte com limitação de corrente e proteção contra curtos-circuitos. Os resultados destes testes serão discutidos no capítulo 4.

O próximo passo foi a conexão dos motores às pontes. Ainda com a alimentação de 24 V fornecida pela fonte controlada com corrente limitada em 2A e com os sinais de comando sendo gerados por chaves, fez-se o acionamento dos motores através do *driver*. Durante a inversão de sentido de rotação dos motores, feita pela comutação dos sinais de direção dos motores que chaveiam os contatos dos relés, ouviu-se um ruído de “repique” dos mesmos (esse “repique” ocorre devido a múltiplos chaveamentos dos contatos do relé). Porém, quando a alimentação do circuito de potência da

placa passou a ser fornecida diretamente pelo conjunto de baterias, esse repique no chaveamento não foi observado.

Dessa observação concluiu-se que a limitação de corrente da fonte é problemática durante a inversão de rotação dos motores, processo em que os motores solicitam correntes mais elevadas durante um curto intervalo de tempo, reduzindo a tensão da fonte. Embora esse efeito não tenha sido observado com a alimentação provida pelas baterias (que não limitam a corrente), especulou-se sobre a possibilidade desse fenômeno ser observado quando a carga da bateria fosse reduzida devido a um longo período de funcionamento do robô. Um quadro agravante desse cenário seria, quando em carga de bateria reduzida, os motores ficarem travados por algum obstáculo, por exemplo, o que solicitaria altas correntes da bateria, levando a uma queda significativa de tensão da mesma e que, por fim, poderia implicar em mau chaveamento dos relés. Isso faria com que o robô revertesse o sentido de movimento, liberando os motores que voltam a consumir corrente baixa. Mas se isso acontecer, o relé volta a ser acionado, fazendo com que o robô colida novamente com o obstáculo, repetindo todo o processo.

O chaveamento adequado do relé utilizado é função da corrente que flui pela sua bobina. Observando a Figura 9, nota-se que essa corrente é controlada pelo ganho de amplificação de corrente propiciado pelo foto-transistor do opto acoplador. Esse componente oferece ganho de, no máximo, seis vezes. Assim, com o objetivo de se aumentar esse ganho e, portanto, assegurar-se o devido acionamento da bobina do relé, foi realizada a seguinte mudança no circuito (Figura 16):

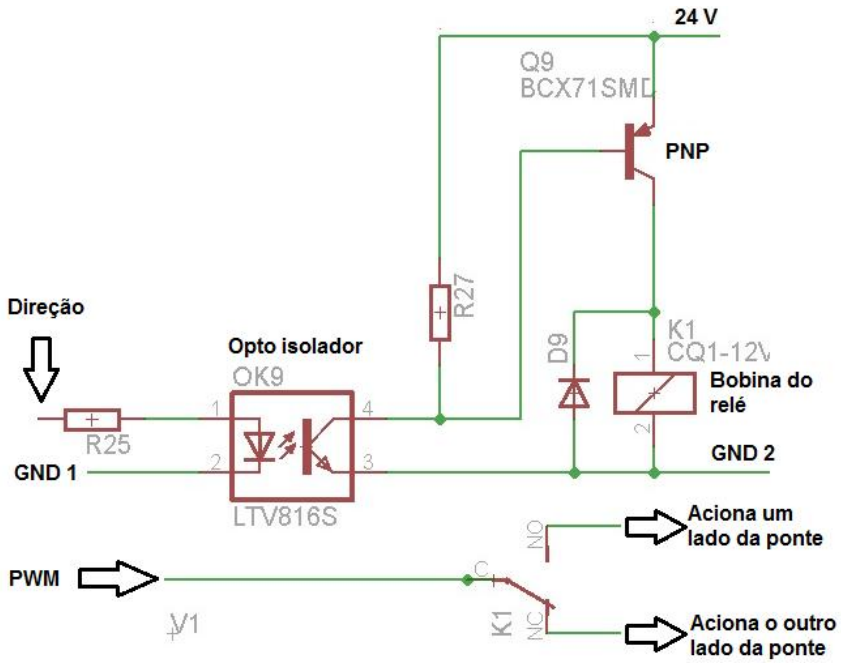


Figura 16: Bobina do relé

Observa-se que agora a bobina do relé é acionada por um transistor de junção bipolar tipo-P (Datasheet BC558). Esse componente apresenta uma razão de transferência de corrente de, no mínimo, 110 vezes, assegurando melhor excitação para a bobina do relé. A função do diodo em paralelo com essa bobina é a de proteger o circuito de tensões induzidas provocadas pelo chaveamento do relé, operando da mesma maneira que descrita para os diodos da ponte-H. Embora ele não apareça no circuito da Figura 9, ele foi soldado à placa do *driver*. De modo análogo, a modificação proposta na excitação da bobina do relé foi sobreposta à placa de circuito impresso do *driver*, interrompendo-se devidamente as conexões elétricas necessárias e soldando-se o *BJT* e o resistor adicionais, um conjunto para cada relé.

Feita essa modificação à placa, submeteu-se a mesma a novos testes de desempenho. Dessa vez, contudo, o acionamento do *driver* foi feito diretamente a partir de um Arduino, no qual foi executada uma rotina simples de diferentes combinações de rotações dos motores, variando-se ambos os sinais de direção e velocidade modulada por largura de pulso. A Figura 17 ilustra o esquema elétrico de conexão dos motores e do microcontrolador ao *driver* durante esse teste. Os resultados serão discutidos no capítulo seguinte.

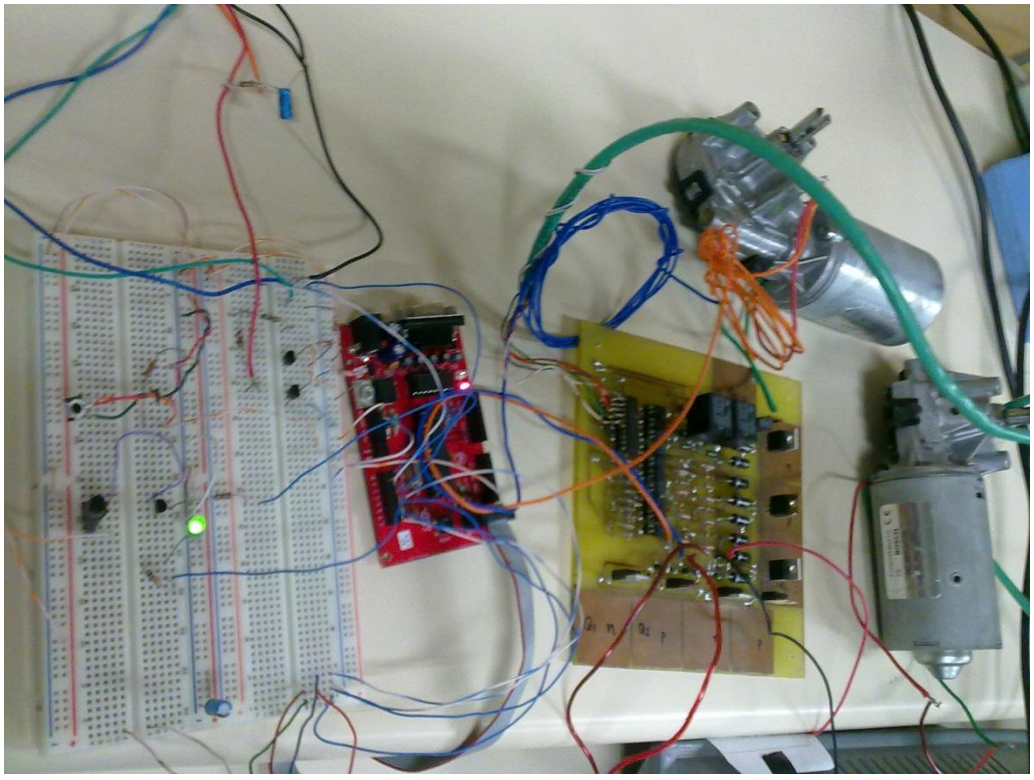


Figura 17: Acionamento dos motores pelo microcontrolador através do *driver*

3.3. Sensor ultrassônico: acionamento e leitura

O módulo de aferição de distância por ultrassom de que se fez uso neste projeto possui apenas quatro pinos: dois deles para a alimentação, que deve ser de 5 V, um pino de entrada e um pino de saída. É entrada para esse módulo um sinal denominado *trigger*, ou pulso de disparo. Como o nome sugere, esse sinal indica para o sensor que ele deve iniciar o processo de aferição. A saída do módulo é um sinal denominado *echo*, cuja tradução em português é "eco". A leitura desse sinal pelo microcontrolador permite calcular a distância do objeto ao sensor.

O sinal de *trigger* enviado ao sensor deve ser um pulso em nível lógico alto (5 V) cuja duração mínima deve ser de 10 us. Uma vez detectado o disparo, o módulo emite um sinal ultrassônico na frequência 40 kHz contendo 8 pulsos e imediatamente leva sua saída, *echo*, para nível lógico alto (HC-SR04 Datasheet, 2012). Assim que seu circuito de detecção recebe a reflexão desse trem de pulsos emitido, leva sua saída para nível zero. Desse modo, tem-se o tempo decorrido para que o trem de pulsos percorra duas vezes a distância entre o emissor e o obstáculo, onda a reflexão do sinal emitido ocorreu. A Figura 18 apresenta o diagrama de tempo desse dispositivo, disponibilizado na folha de dados do mesmo.

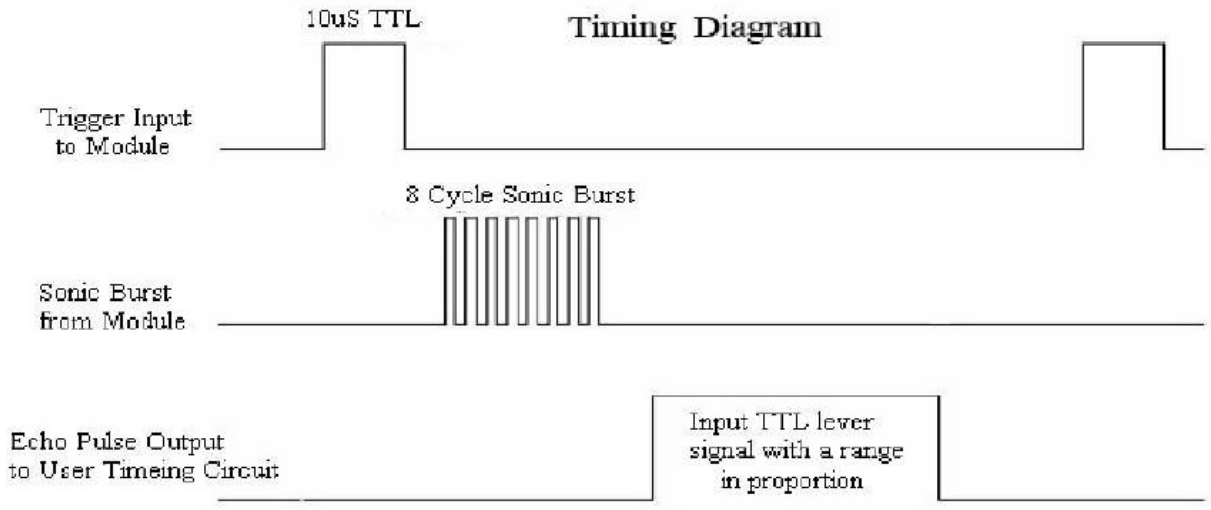


Figura 18: Diagrama de tempo na operação do sensor de ultrassom

Portanto, assumindo que o som se propague no ar com velocidade de 340 m/s, pode-se calcular a distância como:

$$\boldsymbol{Distância\ (m) = 340\left(\frac{m}{s}\right).\frac{\Delta t}{2}(s)} \quad \textbf{Equação 3}$$

Onde Δt é o tempo de duração do sinal *echo* em nível alto. Como se trabalha com intervalos de tempo da ordem de microssegundos, que representam distâncias da ordem de centímetros, a conta prática que se fez no *software* foi:

$$\boldsymbol{Distância\ (cm) = \Delta t(us)/58,2} \quad \textbf{Equação 4}$$

Assim, implementou-se uma rotina simples de acionamento e leitura dos dois sensores que seriam usados no robô, um para detecção de obstáculos frontais quando o robô se movimenta para frente e outro para detecção de obstáculos atrás do robô quando esse se locomove em sentido reverso.

O teste, cujos resultados serão tratados no capítulo seguinte, foi inicialmente realizado individualmente para cada sensor, com o intuito de se determinar o *range* ou faixa dentro da qual o módulo tem desempenho favorável. Esse teste foi realizado em sala com temperatura ambiente, ventilada e iluminada pela luz do sol, sendo as distâncias aferidas pelo sensor comparadas com a medida de uma fita métrica. A segunda etapa do teste foi a aferição da leitura de ambos os sensores ao mesmo tempo. Nessa situação, segundo descrição da folha de dados do módulo, deve-se observar o ângulo de abertura do sinal dentro do qual se faz a emissão e recepção do mesmo. Esse ângulo de abertura é de 15 º, de modo que o sensor tem melhor desempenho na detecção de objetos que estejam dentro de um cone imaginário de abertura total igual a 30 º. Para que não haja interferência no desempenho de detecção dos sensores, quando acionados ao mesmo tempo, deve-se garantir o devido posicionamento deles de modo que se respeite essa abertura. De fato, durante o teste com ambos os sensores, observou-se que quando posicionados lado a lado, os valores de distâncias lidos eram totalmente discrepantes com os valores reais. Entretanto, como para essa aplicação na plataforma robótica os dois sensores estarão dispostos a 180 º um do outro, não se observou esse problema.

Baseado nas informações da folha de dados desse módulo de ultrassom, o tempo mínimo recomendado entre duas leituras consecutivas do sensor é de 60 milissegundos. Esse atraso foi implementado com o uso de funções de *delay* do Arduino.

3.4. Implementação do *software*

Este subcapítulo aborda a integração do dispositivo de comunicação *Bluetooth* ao módulo de controle da plataforma robótica, a geração do *PWM* e sinais de direção para o *driver* dos motores e o sensoriamento da trajetória do robô. Ou seja, esse subcapítulo trata do desenvolvimento do *software* de controle da plataforma robótica como um todo.

Para o Arduino, o protocolo do *Bluetooth* é invisível, pois o dispositivo que transmite e recebe dados via *link Bluetooth* se comunica com o microcontrolador via protocolo serial de 9600 bps, oito *bits* de dados, um *start bit*, um *stop bit* e sem paridade, que é o protocolo padrão de comunicação serial do Arduino. Assim, o programa executado nesse microcontrolador não necessita tratar do protocolo *Bluetooth*. Desse modo, as bibliotecas padrões de comunicação serial desse microcontrolador já permitem a ele a comunicação com um dispositivo mestre (no caso deste projeto, um *laptop* tipo IBM-PC) sem que tenhamos que nos preocupar com o protocolo *Bluetooth*. A única informação de configuração da comunicação serial que o código fonte deve incluir é o *baud rate* ou taxa de

transferência de dados, de 9600 bps. A conexão física entre o módulo de conversão *Bluetooth* e o Arduino deve ser feita de modo que os pinos TX e RX do Arduino sejam respectivamente conectados aos pinos RX e TX do módulo de conversão, o que também é estabelecido pelo padrão *UART* (Tocci, Sistemas Digitais - Princípios e Aplicações.). A alimentação do conversor deve ser de 3,3 V, fornecida também pelo Arduino.

A biblioteca de comunicação serial do Arduino disponibiliza diversas funções de leitura e escrita na porta serial, de modo que não foi preciso o desenvolvimento de nenhuma função especial para estabelecer-se a comunicação entre o microcontrolador e o computador.

Para trabalhar-se com a geração de sinais modulados por largura de pulso, as bibliotecas padrões do Arduino disponibilizam a função *analogWrite()*, cuja sintaxe é: (Arduino analogWrite Reference)

analogWrite(pino, valor)

O parâmetro "pino" denota o pino onde a função irá gerar o sinal de *PWM* de frequência fixa cujo valor de razão cíclica é dado por:

$$\textbf{Razão cíclica} = \frac{valor}{255}.100\ (\%) \qquad \textbf{Equação 5}$$

O parâmetro "valor", portanto, deve ser um inteiro variando de 0 a 255. Essa função, quando chamada, faz com que o sinal de *PWM* seja gerado no pino solicitado até que seja chamada novamente.

O Arduino UNO, microcontrolador do qual se faz uso neste projeto, disponibiliza 14 pinos de saídas ou entradas digitais, dos quais seis deles podem ser usados com a função *analogWrite()* na geração de *PWM*. São eles os pinos 3, 5, 6, 9, 10 e 11. A frequência do sinal de *PWM* nos pinos 5 e 6 é de aproximadamente 980 Hz. Para todos os outros, essa frequência é de aproximadamente 490 Hz. Neste projeto, optou-se pelo uso dos pinos 11 e 10 (490 Hz) devido ao fato de que a geração de *PWM* nos pinos 5 e 6 (980 Hz) afete o uso de funções de atraso, das quais se faz uso no programa (Arduino UNO Overview).

Figura 19: Arduino UNO

A geração dos sinais de *trigger* para os sensores pôde ser implementada com o uso de duas das saídas digitais e a função *delayMicroseconds()*, cujo parâmetro é um número inteiro de microssegundos que se devem ser gastos nela. Configurado o disparado, a leitura de cada sensor foi feita com auxílio da função *pulseIn()*, cuja sintaxe é: (Arduino pulseIn Reference)

pulseIn(pino, nível, timeout)

O parâmetro "pino" diz qual das entradas digitais deve ser lida; "nível" informa à função por qual valor lógico ela deve esperar: alto ou baixo; o parâmetro "*timeout*" é opcional e indica por, no máximo, quanto tempo a função deve esperar pelo sinal. Se algum pulso tiver início dentro do intervalo de tempo *timeout*, ela retorna a duração desse pulso lido em microssegundos. Com essa leitura e o devido processamento, pode-se aferir a distância entre um obstáculo e o sensor de ultrassom. Se não houver algum pulso dentro da janela de tempo igual a *timeout*, a função retorna o valor zero.

O fluxograma apresentado na Figura 20 descreve o algoritmo do *software* desenvolvido para controle de todos os periféricos do módulo de controle do robô pendular e acionamento de seus atuadores através do *driver* de potência. Os blocos "Avaliar Comando e Executar" e "Reflexo Condicionado" serão detalhados a seguir.

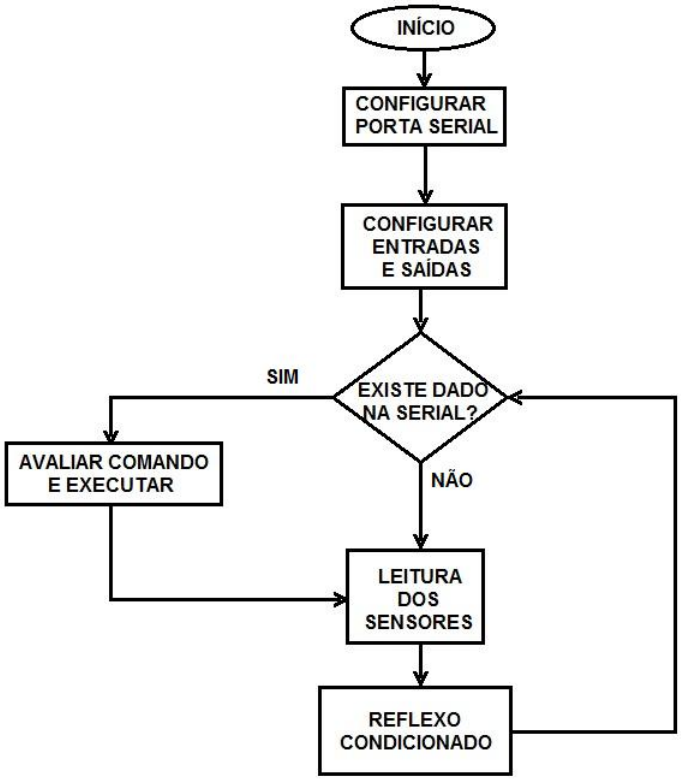


Figura 20: Fluxograma do *software* do módulo de controle

O bloco "Avaliar Comando e Executar" é chamado cada vez que se verifica a existência de algum dado na serial do microcontrolador. Essa sub-rotina identifica e executa imediatamente um comando recebido dentre uma possibilidade de sete comandos. Esses comandos são identificados pelo código ASCII de caracteres transmitidos um a um do computador para o microcontrolador via *link Bluetooth*. A Tabela 1 associa os caracteres predefinidos pelo projetista aos respectivos comandos:

Tabela 1: Resumo dos comandos de navegação

Caractere	Comando
w	Configura sentido de rotação dos motores para o robô locomover-se para frente. Não altera a velocidade.
s	Configura sentido de rotação dos motores para o robô locomover-se para trás. Não altera a velocidade.
e	Incrementa a velocidade.
q	Decrementa a velocidade.
d	Vira o robô para a direita se velocidade diferente de zero.
a	Vira o robô para a esquerda se velocidade diferente de zero.
x	Para o robô.

Os comandos “w” e “s” são executados de modo que, caso impliquem em reversão do sentido de rotação de um ou ambos os motores, a velocidade de rotação dele(s) seja alcançada por meio de rampa de aceleração. Como exemplo, suponhamos que os motores estejam sendo acionados para frente por sinais de *PWM's* com razões cíclicas de 50%. Caso o comando “s” seja recebido, o microcontrolador irá, nesta ordem, zerar a velocidade de ambos os motores (i.e. *duty cycle* 0%), inverter o sinal de rotação e, então, acelerar os motores gradualmente até que as razões cíclicas dos sinais de *PWM's* cheguem a 50% novamente. Essa estratégia em *software* evita picos de corrente no *driver* e a ocorrência de solavancos no robô. A ação de virar para a direita, por exemplo, é desempenhada por desligamento do próprio motor direito. Quando, depois de solicitado o comando de virar para a direita, por exemplo, vier a solicitação para locomoção para frente, ambos os motores serão novamente acelerados até a velocidade anterior.

O bloco “Reflexo Condicionado” representa o trecho de programa responsável por analisar a distância aferida pelos módulos de ultrassom e decidir se deve ou não atuar sobre os motores. Este módulo implementa um sistema reativo que para o robô se o operador lhe conduzir a colidir com um obstáculo. Isso é muito útil se for considerado que este robô será operado por crianças nas escolas. Se o robô estiver se locomovendo para frente e a distância aferida pelo sensor ultrassônico frontal for maior do que zero e menor do que 40 cm, os sinais de *PWM* dos dois motores são zerados. De modo análogo, se o robô estiver se deslocando para trás e algum obstáculo for detectado pelo sensor traseiro a uma distância menor do que 40 cm e maior do que zero, ambos os motores são cortados e o robô para. Nestas condições, o robô somente aceita comandos que o conduzam em direção contrária ao obstáculo. O sensor frontal não implica em nenhuma ação quando o robô desloca-se para trás nem o traseiro quando se locomove para frente.

Uma vez que o *software* (ver apêndice D) apresentou desempenho funcional na execução do algoritmo que controla todas as partes que integram o módulo de controle do robô pendular, iniciaram-se os testes com o robô no ambiente.

4. Resultados

Dentro do que foi definido na fase de *design* abordada no capítulo 2 e do que foi implementado na fase prática explicitada no capítulo 3, este capítulo apresenta os resultados obtidos no desenvolvimento do módulo de controle do robô pendular de duas rodas, bem como uma análise dos mesmos. Embora o projeto de *driver* nomeado neste documento como projeto I não tenha sido utilizado, seus resultados serão brevemente apresentados e analisados, pois serviram de base de entendimento do devido funcionamento da ponte-H.

4.1. Protótipo de *driver*: projeto I

Como discutido na seção 3.1, durante a fase de validação da placa já manufaturada, observou-se a devida operação desse *driver* no modo *ON-OFF*, sem sinal de *PWM*. Entretanto, quando se fez uso de sinal de velocidade modulado por largura de pulso, não se obteve a devida variação de velocidade de rotação dos motores. Com o uso de osciloscópio, observou-se que os sinais de *PWM* que entravam no *buffer* do *driver* chegavam aos terminais de conexão de carga na ponte-H bastante distorcidos.

Um dos erros mais graves desse *design* foi o acionamento dos *MOSFETs* tipo N, na parte inferior da ponte, com sinais de nível *TTL* (de 0 a 5 V). Como já mencionado anteriormente, a devida tensão de acionamento de um NMOS no modo de saturação requer, em geral, uma diferença de tensão entre fonte e comporta de pelo menos 12 V. Com um sinal de 5 V, o transistor operava na sua região linear de amplificação. Outro problema observado algumas vezes foi a queima dos *MOSFETs* devido ao acionamento simultâneo dos dois *MOSFETs* do mesmo lado da ponte H. Embora esse *design*, cujo projeto foi levado até a conclusão da manufatura da placa, tenha permitido o acionamento dos motores em modo *ON-OFF*, isto é, em velocidade plena ou parado, o fato de não ser possível fazer-se uso da modulação por largura de pulso implicou no início de um segundo projeto de *driver*. Contudo, esta primeira versão permitiu, além de um melhor entendimento da ponte-H, que se desenvolvesse uma bagagem de conhecimento no projeto de placas de circuito impresso, desde a geração de *layouts* até a soldagem dos componentes.

4.2. Protótipo de *driver*: projeto II

Durante os testes de validação realizados com essa segunda placa de driver observou-se o repique de chaveamento do relé quando o lado de potência da placa era alimentado pela fonte protegida de tensão, que possui como uma de suas características a limitação de corrente em 2 A. Como já previamente abordado, embora esse repique não tenha sido observado quando o circuito era alimentado pela associação de baterias, especulou-se a possibilidade de ocorrência desse fenômeno quando a carga da bateria caísse e uma alta corrente fosse solicitada pelos motores, como num caso de travamento das rodas, por exemplo. Nesse cenário, a tensão da bateria cairia mais ainda,

comprometendo a devida excitação para o chaveamento das bobinas do relé, pois as mesmas eram acionadas pelo foto-transistor do opto acoplador, elemento esse que possui um baixo ganho de corrente.

Dessa observação, fez-se uma modificação no circuito, já descrita na seção 3.2 e ilustrada na Figura 16. As devidas alterações foram realizadas diretamente na placa manufaturada. O teste de validação prosseguiu, mas com o acionamento do driver diretamente pelo Arduino. Fez-se uso de uma rotina simples e repetitiva, que acionava os motores com diferentes combinações de sentido de rotação e com diferentes razões cíclicas. Os resultados foram satisfatórios mesmo em chaveamentos mais críticos, quando se comuta os sentidos de rotação de ambos os motores ao mesmo tempo, em velocidade máxima e sem atraso em *software*. Nessa situação, o único intervalo de tempo entre a inversão de sentidos é aquele durante o deslocamento da chave do relé de uma posição para a outra, que é de no máximo 10 milissegundos (HJR-3FF-S-H Datasheet). Esse cenário, contudo, deve ser evitado sempre que possível, pois impõe uma circulação de altas correntes pelas trilhas, soldas e demais contatos da placa de circuito impresso que podem diminuir sua vida útil ou mesmo romper uma isolação e configurar um curto-circuito. As partidas suaves, como a aceleração em rampa, são estratégias recomendadas das quais se fez uso no *software*.

4.3. Sensores de ultrassom: desempenho

O teste realizado individualmente para cada um dos dois sensores foi baseado em comparar-se 10 leituras do sensor em teste para uma dada distância fixa. Dado que o robô é relativamente grande, a resolução usada foi de um centímetro. A Tabela 2: Aferição de distância pelos sensores ultrassônicos mostra os resultados das leituras de um dos sensores para distâncias fixas, de modo que resultados acima da faixa considerada foram descartados. O resultado para o segundo sensor foi tecnicamente similar ao do primeiro.

Tabela 2: Aferição de distância pelos sensores ultrassônicos

Leitura	Distâncias Reais (cm)								
	2	5	10	50	100	150	200	210	220
1	2	5	10	50	100	150	202	214	229
2	2	5	10	50	99	150	202	213	XXX
3	2	5	10	50	98	151	201	216	230
4	2	5	10	51	100	150	203	217	232
5	2	5	10	50	100	150	201	212	235
6	2	5	11	50	101	150	200	210	230
7	3	5	10	50	100	150	200	213	228
8	3	5	10	49	100	152	201	XXX	XXX
9	2	5	10	50	100	151	200	217	230
10	2	5	10	50	99	150	201	215	XXX
Média	2	5	10	50	100	150	201	214	231

O resultado médio foi calculado apenas com os valores válidos de leitura, de modo que "XXX" denota valores muito discrepantes de leitura e, para que não afetassem a média, foram descartados. Observa-se que a aferição de distâncias através do sensor de ultrassom é viável de 2 até 200 cm, limite esse que permite de maneira eficaz que se possa manobrar o robô de maneira a evitar uma colisão.

A partir desse resultado, configurou-se que o tempo máximo de espera da função que lê o sinal de *echo* é o tempo para que se detecte um obstáculo a aproximadamente 200 cm.

4.4. Desempenho do *software* do módulo de controle

Antes que o *software* fosse, de fato, implementado, muitos fluxogramas e esquemas de algoritmos foram esboçados, de modo que um funcionamento satisfatório do programa integrando todos os periféricos foi observado já na primeira versão prototipada. Entretanto, ajustes foram sendo aplicados à medida que resultados não previstos apareciam.

Por exemplo, observou-se que, embora as leituras dos sensores fossem feitas de 10 a 20 vezes por segundo, o envio de seus valores pela porta serial para o monitor do computador que envia os comandos de navegação do robô devia ser feito a cerca de uma vez por segundo, já propiciando uma boa averiguação das medidas.

O modo de comando da navegação do robô também foi alterado à medida que testes foram sendo executados. Por exemplo, a ideia inicial proposta para realização do comando de virar para

direita, por exemplo, era decrementar gradualmente a velocidade do motor direito, conforme o comando “d” fosse sendo reenviado. Para que o robô parasse de virar para a direita e retornasse a locomover-se em linha reta, o usuário deveria enviar gradualmente o comando de virar à esquerda (“a”) até que se constatasse que o robô estivesse em linha reta. Mas ao final desse processo, devido à redução de velocidade de ambos os motores durante o processo de curva, o robô deveria ser acelerado novamente. O algoritmo de curva foi, então, redefinido. Quando o comando “d” é reconhecido na porta serial, o programa salva a razão cíclica atual do motor direito e o desliga. Assim, para que o robô pendular volte a se deslocar para frente, basta que o comando “w” seja enviado a ele, de modo que o programa retornará o valor salvo de *duty cycle* ao *PWM* do motor direito.

Embora durante a fase de projeto do *software* de acionamento do *driver* já se tivesse estabelecida a ideia de partida suave, a necessidade da implementação da rampa de aceleração a cada comutação de sentido foi realizada de fato ao se notar os solavancos que o robô desenvolvia quando saía do processo de curva e quando se invertiam os sentidos de rotação de ambos os motores ao mesmo tempo. A calibração da taxa de aceleração, realizada na prática pela alteração do atraso entre os incrementos no valor de *duty cycle* até o valor final, foi feita gradualmente com o robô já montado, isto é, com todos os componentes do seu módulo de controle já embarcados em seu chassi. A Figura 21 mostra, em detalhe, o *driver* e um dos sensores no chassi do robô.

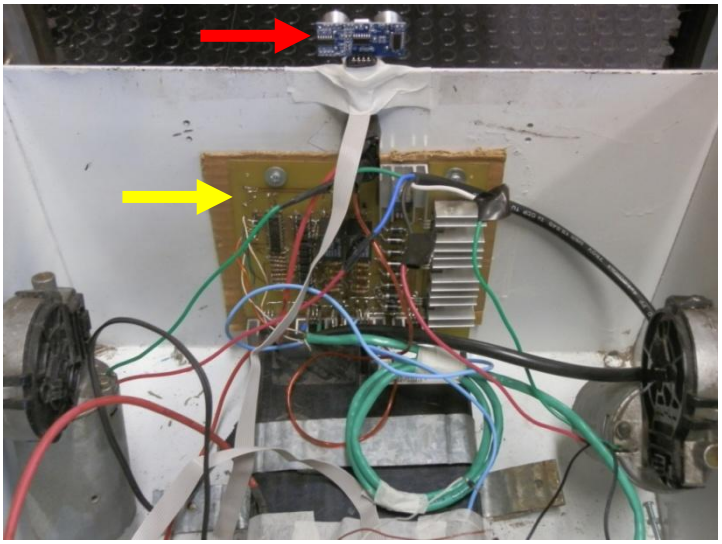


Figura 21: Chassi do robô com detalhe para o driver e um sensor de ultrassom

Os testes de navegação realizados com o robô no chão permitiram também observar que, quando em velocidade máxima, o robô consegue evitar a colisão com um obstáculo detectado a cerca de 40 cm se seu chassi. O levantamento desse parâmetro permitiu a devida calibração do controle reativo do robô.

No que diz respeito ao *link Bluetooth* estabelecido entre a plataforma robótica e o computador, não se observou perda de conexão num raio de 15 m.

5. Conclusões

Este trabalho apresentou o projeto e implementação de um módulo de controle para um robô móvel autônomo. Examinando-se o resultado final alcançado neste trabalho como um todo, observa-se que a proposta de desenvolvimento de um conjunto *hardware/software* que permitisse à plataforma robótica pendular de duas rodas apresentada receber comandos de navegação do terminal *Bluetooth* de um computador e convertê-los em deslocamento do robô condicionado pelo sensoriamento de obstáculos foi atendida. O conjunto de comandos de navegação, formado por sete caracteres próximos em um teclado padrão de computador, torna simples o controle do robô por, por exemplo, uma criança já familiarizada com esse periférico de seu computador pessoal ou de sua escola. Para permitir que a navegação seja futuramente estendida para a responsabilidade de um *software* de inteligência artificial de mais alto nível, a transmissão dos códigos ASCII referentes aos respectivos caracteres da Tabela 1 deve ser utilizada.

A função de atuar como uma interface entre o microcontrolador e os dois motores responsáveis pela locomoção do robô pendular é bem desempenhada pelo *driver* projetado e manufaturado neste trabalho. Como discutido no capitulo dos resultados, a placa desenvolvida não apresentou falha, dano ou mau desempenho nem mesmo nas situações mais críticas de chaveamento. Além de possibilitar o acionamento de dois motores ao mesmo tempo, o *driver* desenvolvido possui baixo custo comparado com os preços de dispositivos comerciais que atendam aos requisitos deste projeto, cujos valores ultrapassam R$100,00. O custo total dos componentes da placa manufaturada, levantado em agosto de 2014, foi abaixo de R$50,00, de modo que esse valor pode ainda ser reduzido caso o *driver* seja implementado com componentes *SMD (Surface Mount Device)* (Sedra & Smith, 2000), que reduziriam a área da placa de cobre necessária.

Embora o Arduino seja parte de uma categoria de microcontroladores que apresentam, entre outras características, ferramentas e funcionalidades desenvolvidas para tornar o processo de programação de microcontroladores acessível para amadores e demais usuários sem algum tipo de formação acadêmica na área, o mesmo não apresentou nenhuma limitação ao desenvolvimento do módulo de controle do robô pendular proposto. Não foi necessário nem mesmo que se incluísse alguma biblioteca especial além daquelas que já são padrões de sua *IDE*, ou que se usasse trechos de códigos mais avançados em *AVR-C*. A única limitação apresentada no uso do Arduino foi a disponibilidade de apenas duas frequências na geração de *PWM* (ver seção 3.4), das quais uma delas não afeta o uso de funções baseadas em atrasos (*delays*). É importante notar que, devido ao fato de que essas duas frequências sejam audíveis por seres humanos, há ruído sonoro durante o acionamento dos motores.

Os módulos de detecção de obstáculos baseados em pulsos ultrassônicos mostraram-se viáveis na detecção de obstáculos necessária à implementação do reflexo condicionado na navegação do robô. Com apenas dois sensores disponíveis para execução deste projeto, foi possível proteger o robô, objetos e pessoas presentes em seu ambiente de trabalho contra colisões. Entretanto, principalmente devido ao relativo grande porte dessa plataforma robótica, quando este projeto for continuado futuramente com a implementação de navegação autônoma, será necessário que mais módulos sonares sejam devidamente dispostos no chassi do robô, de modo que, além do desenvolvimento do *software* de alto nível para navegação, modificações no programa em baixo nível executado no microcontrolador embarcado também deverão ser implementadas com o intuito de se fazer a leitura e processamento dos mesmos.

De modo geral, o desempenho alcançado no comando da navegação da plataforma robótica denota que o módulo de controle desenvolvido neste trabalho para o robô pendular de duas rodas torna possível e viável o uso dessa plataforma na interação com crianças e adultos em projetos educacionais contextualizados pela importância da inserção da tecnologia no ambiente de ensino. Portando, conclui-se que os objetivos específicos deste trabalho, que são descritos pelo desenvolvimento de todos os componentes que constituem o módulo de controle da plataforma robótica pendular de duas rodas, contribuem para o objetivo mais amplo, que é o uso desse robô nos projetos educacionais mencionados.

6. Referências bibliográficas

About Us | CadSoft. (s.d.). Acesso em 25 de Jullho de 2014, disponível em EAGLE PCB Design Software: http://www.cadsoftusa.com/about-us/

Ahmed, A. (2000). Modulação por largura de pulso (PWM). In: A. Ahmed, *Eletrônica de Potência* (pp. 365-368). São Paulo: Pearson Prentice Hall.

Arduino analogWrite Reference. (s.d.). Acesso em 04 de Agosto de 2014, disponível em Arduino Reference: http://arduino.cc/en/Reference/analogWrite

Arduino pulseIn Reference. (s.d.). Acesso em 04 de Agosto de 2014, disponível em Arduino Reference: http://arduino.cc/en/Reference/pulseIn

Arduino UNO Overview. (s.d.). Acesso em 07 de Janeiro de 2014, disponível em Arduino UNO Overview: http://arduino.cc/en/Main/ArduinoBoardUno

Balch, M. (2003). *Complete Digital Design : A Comprehensive Guide to Digital Electronics and Computer System Architecture*. McGraw Hill Professional.

Banzi, M. , Cuartielles, D. et al. (s.d.). *Arduino*. Acesso em August de 2014, disponível em Arduino: http://www.arduino.cc/en/Guide/Introduction

Barbi, I. (2006). *Eletrônica de potência* (6ª ed.). Florianópolis: Edição do Autor.

Bluetooth SIG, S. |. (s.d.). *Bluetooth Special Interest Group (SIG)*. Acesso em July de 2014, disponível em https://www.bluetooth.org/en-us/members/about-sig

Bluetooth UART Interface Datasheet. (s.d.). Acesso em 21 de Junho de 2014, disponível em Bluetooth UART Interface Datasheet: http://www.tato.ind.br/files/GP-GC020.pdf

CHELLA, M. (2002). *Ambiente de robotica para aplicações educacionais com SuperLogo* . Campinas, SP: UNICAMP.

Datasheet 74LS244. (s.d.). Acesso em 10 de Dezembro de 2013, disponível em Datasheet 74LS244: http://pdf1.alldatasheet.com/datasheet-pdf/view/1617/MITSUBISHI/74LS244.html

Datasheet BC558. (s.d.). Acesso em 15 de Dezembro de 2013, disponível em Datasheet BC558: https://www.fairchildsemi.com/datasheets/BC/BC556.pdf

Datasheet F006 WM0 310. (s.d.). Acesso em Julho de 2014, disponível em Kalatec Automação: http://www.kalatec.com.br/Motor_Bosch/F006_WM0_310.pdf

External PCB Trace Width. (s.d.). Acesso em 15 de Dezembro de 2013, disponível em Electrical Engineering Comunity: http://www.eeweb.com/toolbox/external-pcb-trace-width

HC-SR04 Datasheet. (2012). Acesso em 20 de Setembro de 2013, disponível em Ultrasonic Ranging Module: http://elecfreaks.com/store/download/HC-SR04.pdf

HERNÁNDEZ, F. (1998). *Transgressão e mudança na educação: os projetos de trabalho.* Porto Alegre: Artmed.

HJR-3FF-S-H Datasheet. (s.d.). Acesso em 13 de Julho de 2014, disponível em HJR-3FF-S-H Datasheet: http://www.datasheet4u.com/download.php?id=704181

HV Floating MOS-Gate Driver ICs. (s.d.). Acesso em 05 de Agosto de 2014, disponível em International Rectifier App Notes: http://www.irf.com/technical-info/appnotes/an-978.pdf

Inoue, K., & Takao, O. (2004). *Patente Nº 6753717.*

IRF640 Datasheet. (s.d.). Acesso em 20 de Janeiro de 2014, disponível em IRF640 Datasheet: http://pdf.datasheetcatalog.com/datasheet/fairchild/IRF640.pdf

IRF9640 Datasheet. (s.d.). Acesso em 20 de Janeiro de 2014, disponível em IRF9640 Datasheet: http://pdf.datasheetcatalog.com/datasheet/fairchild/RF1S9640SM.pdf

Optocoupler Datasheet_LTV8x6. (s.d.). Acesso em Dezembro de 2013, disponível em Optocoupler Datasheet_LTV8x6: https://www.sparkfun.com/datasheets/Components/LTV-8x6.pdf

Rajput, A. (2012). *Ultrasonic SR04 library for Arudino*. Acesso em August de 2014, disponível em Ultrasonic SR04 library for Arudino: http://playground.arduino.cc/Code/SR04

Sedra, A., & Smith, K. (2000). *Microeletrônica.* São Paulo: Makron.

SIMÕES, E. d. (2008). Dispositivo Robótico Móvel Pendular de Duas Rodas. *Revista da Propriedade Industrial* .

Simões, E. V. (2000). *Development of an Embedded Evolutionary Controller to Enable Collision-Free Navigation of a Population of Autonomous Mobile Robots.* The University of Kent at Canterbury, (Tese de Doutorado).

Smart Projects | About us. (s.d.). Acesso em July de 2014, disponível em Smart Projects | About us: http://www.smartprojects.it/about_en.html

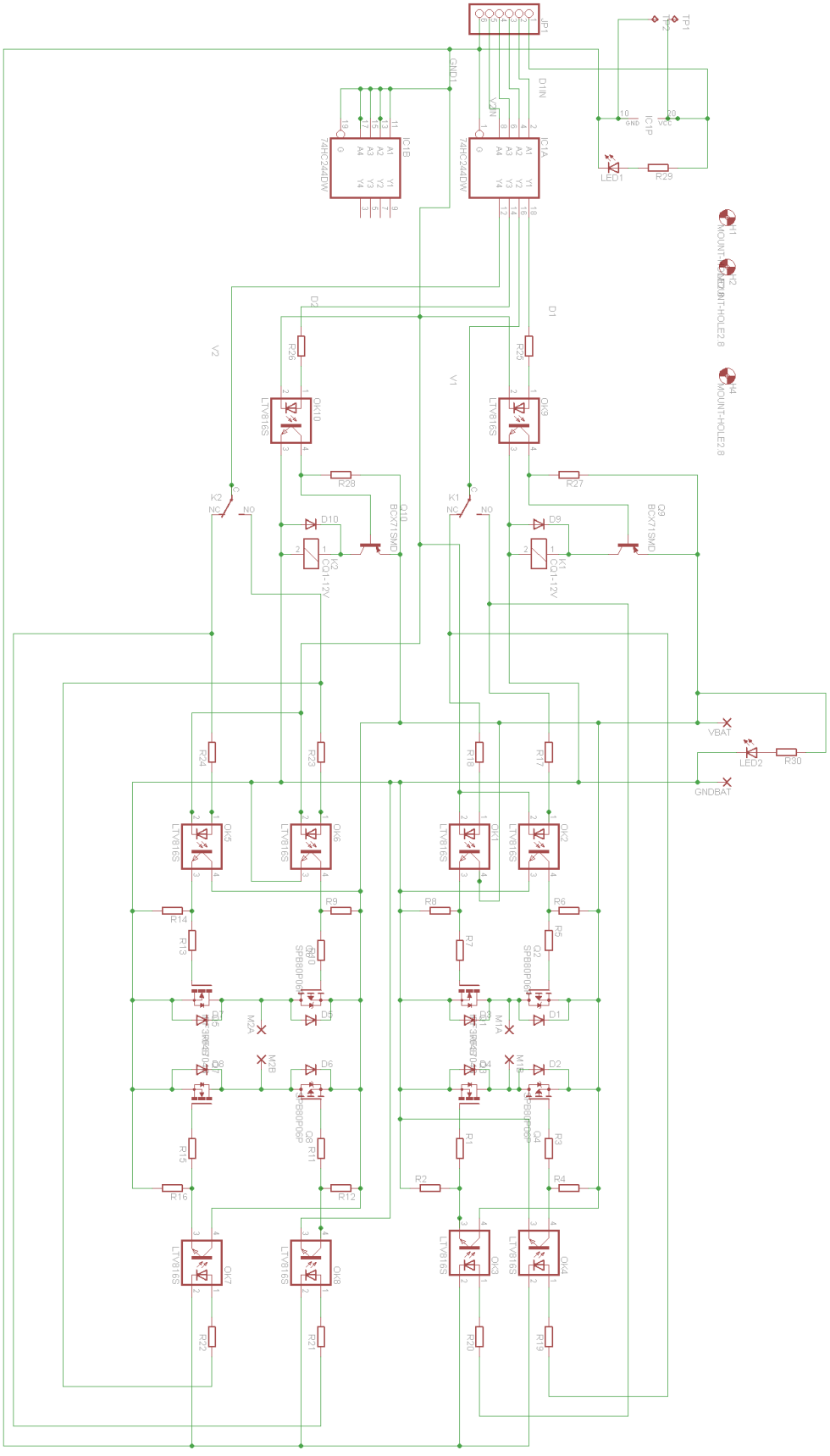
Tocci, J. R. *Sistemas Digitais - Princípios e Aplicações.* Prentice Hall do Brasil.

Tocci, J. R. (2011). *Sistemas Digitais - Princípios e Aplicações.* São Paulo: Prentice Hall do Brasil.

Tutorial PCI. (s.d.). Acesso em 15 de Dezembro de 2013, disponível em Tutorial PCI: https://www.dropbox.com/s/qiz6vjrfe6oo8s3/tutorialdeplacadecircuitoimpressopci1-111205173435-phpapp02.pdf

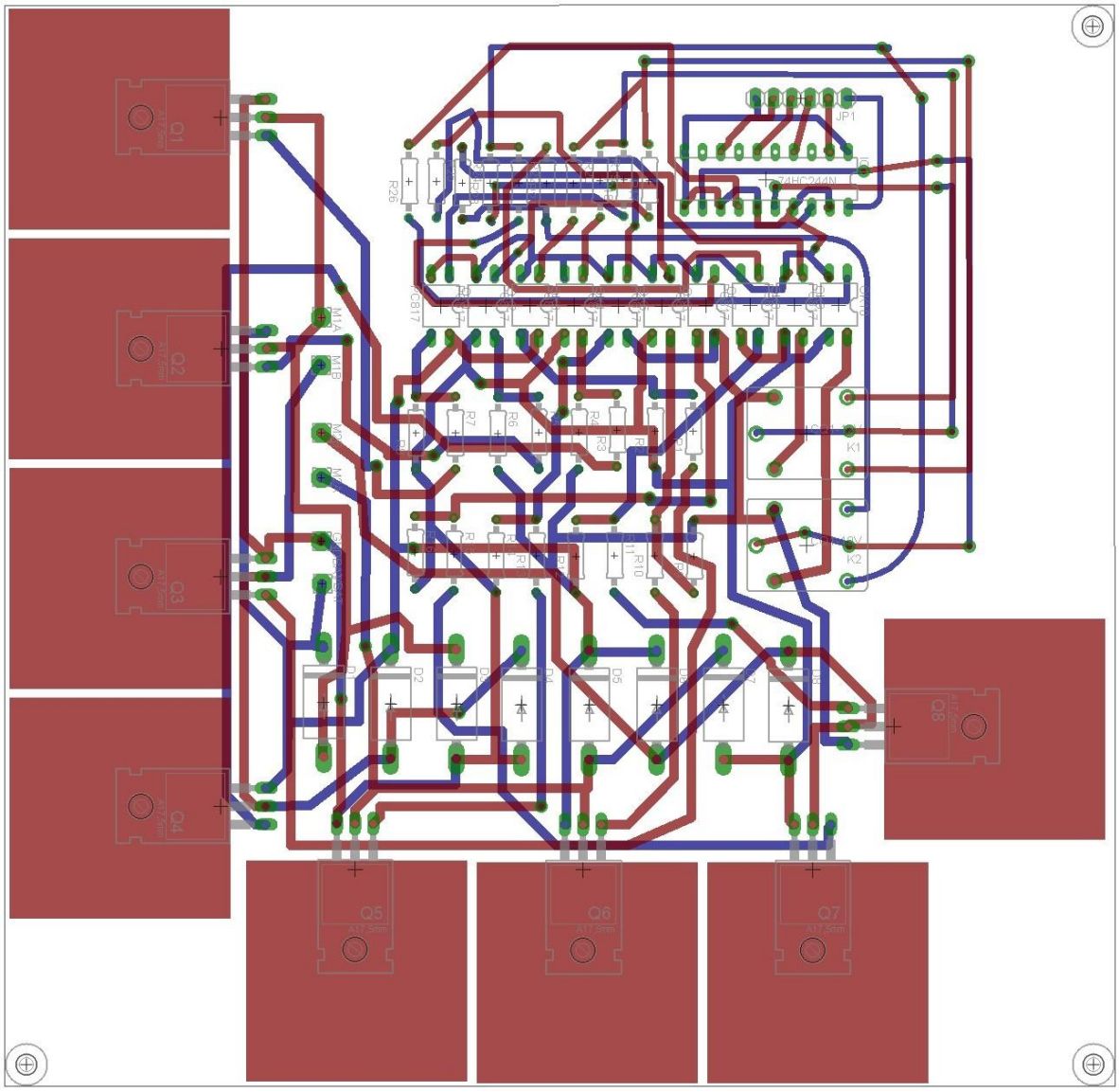
I. Apêndice A

Esquemático final do circuito do driver.



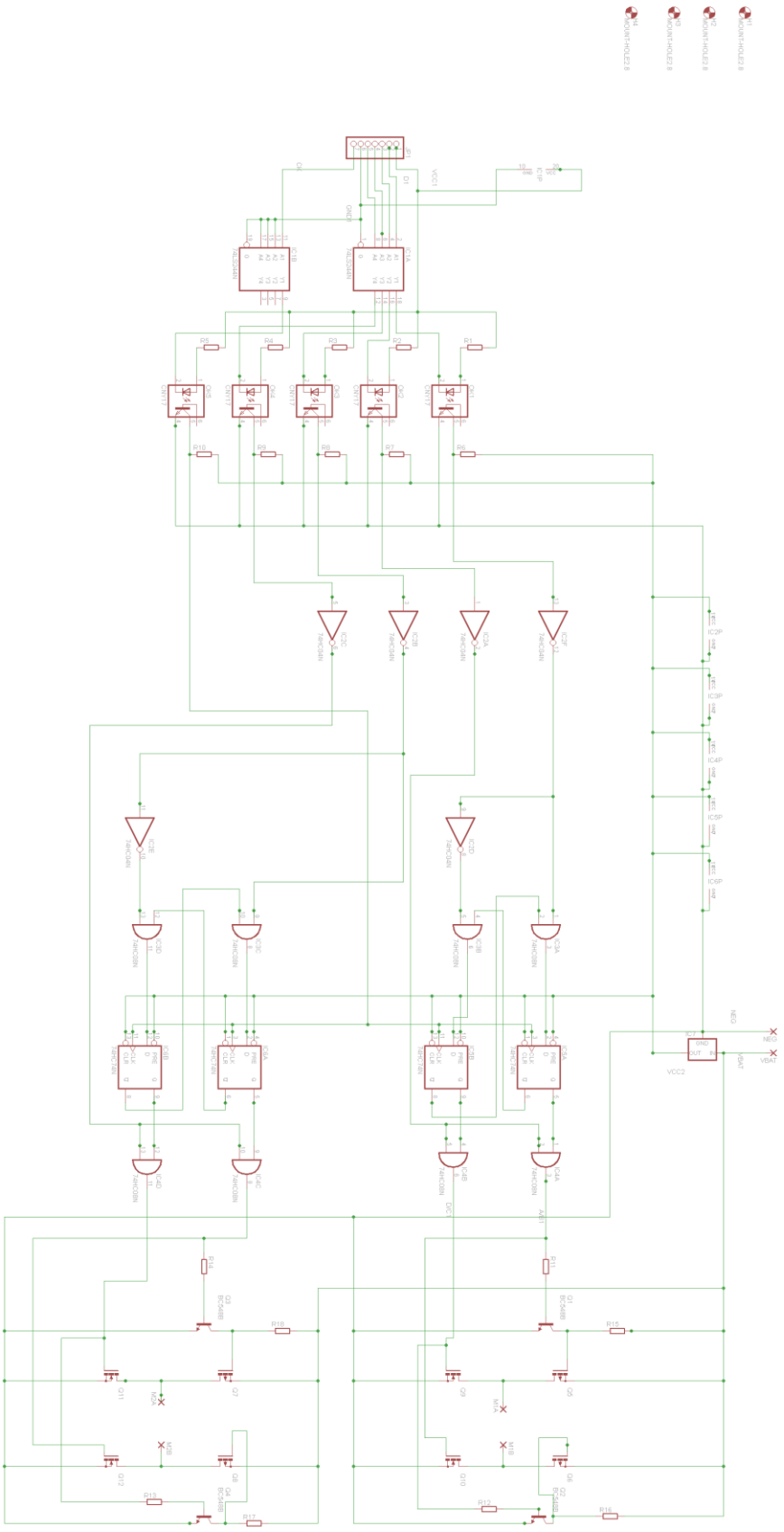
II. Apêndice B

Layout final da placa de circuito impresso.



III. Apêndice C

Esquemático da primeira versão de *driver*.



IV. Apêndice D

Software final embarcado no microcontrolador do módulo de controle.

```
int RPWM = 10;   //saída de PWM para motor direito é o pino 10

int LPWM = 11;

int Rdirout = 8; //saída de direção para motor direito é pino 8

int Ldirout = 9;

int Trigger =  7;//saída que dispara o trem de pulsos ultra-som

int Trigger2 = 6;

int Echo = 5;

int Echo2 = 4;

unsigned long microsduration;  //mede largura do pulso Echo em us

unsigned long distcm;                  //distância do obst. em cm

unsigned long microsduration2;  //mede largura do pulso Echo em us

unsigned long distcm2;                 //distância do obst. em cm

//Variáveis que dão o sentido de direção para as variáveis de saída de direção

        int Rdir = LOW; //0-> reverso ou 1-> frente

        int Ldir = LOW;



        //Duty Cycle variables (DC Left PWM e DC Right PWM): de 0 a 255 -> 0 a 100%

        int DCRPWM = 0;

        int DCLPWM = 0;

        //Variável de controle de exibição

int countdisplay = 0;

//Flag que indica modo de curva

        int flag1 = 0;


void setup()

{

        Serial.begin(9600);

        //iniciar pinos de saída para RightPWM, LeftPWM, RightDir, LeftDir, Trigger:

        //11, 10, 9, 8, 7 respectivamente
```

```
        for(int setouts = 6; setouts <= 11; setouts++)
        {
                pinMode(setouts, OUTPUT);
        }
    //iniciar pino Echo as input
    pinMode(Echo, INPUT);
    pinMode(Echo2, INPUT);
}

void loop()
{
        if (Serial.available() > 0)    //check se existe dado na serial
        {
                char bluein = Serial.read();
                 Serial.println(bluein);        //ecoa o character recebido
                switch(bluein)
                {
                        case 'w':       //set ambos para frente
                         Serial.println("frente");

                        if((Rdir == LOW)||(Ldir == LOW)||(flag1 == 1)) //se algum dos motores está em reverso ou desligado...
                                                                        //... zerar PWM antes de inverter
                                {
                                        analogWrite(RPWM, 0);
                                        analogWrite(LPWM, 0);
                                         delay(10);
                                        Rdir = HIGH;
                                        Ldir = HIGH;

                                        digitalWrite(Rdirout, Rdir);
                                        digitalWrite(Ldirout, Ldir);
                                        //rampa de aceleracao
```

```
                for(int acelera = 0; acelera <= DCRPWM; acelera = acelera + 15)
                {
                        analogWrite(RPWM, acelera);
                        analogWrite(LPWM, acelera);
                        delay(100);
                }
        }

        flag1 = 0;
        break;

case 's':   //set ambos reverso
          Serial.println("reverso");
        if((Rdir == HIGH)||(Ldir == HIGH)||(flag1 == 1)) //se algum dos motores está para frente ou desligado ...
                                                          //... zerar PWM antes de inverter
        {
                analogWrite(RPWM, 0);
                analogWrite(LPWM, 0);
                 delay(10);
                Rdir = LOW;
                Ldir = LOW;
                digitalWrite(Rdirout, Rdir);
                digitalWrite(Ldirout, Ldir);
                //rampa de aceleracao
                for(int acelera = 0; acelera <= DCRPWM; acelera = acelera + 15)
                {
                        analogWrite(RPWM, acelera);
                        analogWrite(LPWM, acelera);
                        delay(100);
                }

        }
```

```
                flag1 = 0;
                break;

case 'e':       //acelera ambos
                if(DCRPWM < 255)
                {
                        DCRPWM = DCRPWM + 15;
                 }
                if(DCLPWM < 255)
                 {
                        DCLPWM = DCLPWM + 15;
                 }
                analogWrite(RPWM, DCRPWM);
                analogWrite(LPWM, DCLPWM);
                break;

case 'q':       //desacelera ambos
                if(DCRPWM > 0)
                {
                        DCRPWM = DCRPWM - 15;
                 }
                if(DCLPWM > 0)
                 {
                        DCLPWM = DCLPWM - 15;
                }
                analogWrite(RPWM, DCRPWM);
                analogWrite(LPWM, DCLPWM);
                break;

case 'd':       //vira para direita...
                        //... desligando motor direito
```

```
                                        analogWrite(RPWM, 0);
                                        flag1 = 1;

                                        break;

                                case 'a':       //vira para esquerda...
                                                //... desligando motor esquerdo

                                        analogWrite(LPWM, 0);
                                        flag1 = 1;

                                        break;

                                case 'x':       //desliga ambos
                                        DCRPWM = 0;
                                        DCLPWM = 0;
                                        analogWrite(RPWM, DCRPWM);
                                        analogWrite(LPWM, DCLPWM);
                                        break;
                        }

                }


//Leitura do Ultra-Som e processamento
        distcm = distcm2 = 0;
          sensor1();
         sensor2();

        //Exibir leitura e verificar se dentro do range
        if(countdisplay == 10)
        {
```

```
  Serial.println("Obstaculo a frente:");
  Serial.println(distcm);
  Serial.println("Obstaculo atras:");
  Serial.println(distcm2);
  if(distcm2 > 200)
        Serial.println("Obstaculo atras out of range");
  if(distcm > 200)
    Serial.println("Obstaculo frontal out of range");
  countdisplay = 0;
 }
 //Reflexo caso andando para frente e obstáculo na trajetória a menos de 40 cm
 //Corta os motores
 if((distcm >0)&& (distcm < 40)&&(Rdir == HIGH)&&(Ldir == HIGH))
 {
        DCRPWM = 0;
        DCLPWM = 0;
       analogWrite(RPWM, DCRPWM);
       analogWrite(LPWM, DCLPWM);
 }
  //Reflexo caso andando para trás e obstáculo na trajetória a menos de 40 cm
 //Corta os motores
 if((distcm >0)&&(distcm2 < 40)&&(Rdir == LOW)&&(Ldir == LOW))
 {
        DCRPWM = 0;
        DCLPWM = 0;
        analogWrite(RPWM, DCRPWM);
        analogWrite(LPWM, DCLPWM);
 }
countdisplay++;
delay(50);

}
```

```
void sensor1(void)
{
      digitalWrite(Trigger, LOW); //garante nível low antes de disparar o trigger
      delayMicroseconds(4);
      digitalWrite(Trigger, HIGH); //início do trigger com duração de 10 us
      delayMicroseconds(15);
      digitalWrite(Trigger, LOW);  //final do trigger

      microsduration = pulseIn(Echo, HIGH, 11764); //mede largura do pulso Echo
      distcm = microsduration/58.1;        //11764 é o tempo p objeto a ~2m
}

void sensor2(void)
{
      digitalWrite(Trigger2, LOW); //garante nível low antes de disparar o trigger
      delayMicroseconds(4);
      digitalWrite(Trigger2, HIGH); //início do trigger com duração de 10 us
      delayMicroseconds(15);
      digitalWrite(Trigger2, LOW);  //final do trigger

      microsduration2 = pulseIn(Echo2, HIGH, 11764);
      distcm2 = microsduration2/58.1;
}
```