ESCOLA POLITÉCNICA
DE PERNAMBUCO

UPE - POLI
eCOMP
Engenharia de Computação
GRADUAÇÃO

MELHORIAS NO KIT EDUCACIONAL PÊNDULO INVERTIDO MONTADO COM REEE

Trabalho de Conclusão de Curso

Engenharia de Computação

Autor: Ricardo Teixeira da Silva

Orientador: Profº. Sergio Campello Oliveira

UNIVERSIDADE
DE PERNAMBUCO

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

RICARDO TEIXEIRA DA SILVA

MELHORIAS NO KIT EDUCACIONAL PÊNDULO INVERTIDO MONTADO COM REEE

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, novembro de 2012.

Agradecimentos

Ao longo da minha vida muitas pessoas contribuíram para minha formação pessoal e acadêmica. Algumas representaram forte influência e estavam sempre por perto me levantando a cada queda ou vibrando a cada conquista. Outros, talvez, não notaram a grande importância do suporte que me deram. Não conseguiria citar o nome de todos eles, todos sabem quão complexa é minha memória, ou a falta dela.

Primeiramente a família. O alicerce da minha vida. As pessoas nas quais sempre me inspirei. A minha mãe, Leninha, como gosta de ser carinhosamente chamada pelos amigos e família, agradeço a cada momento de incentivo, cada esforço para que eu pudesse ter o melhor na minha educação. Juntamente com ela meu pai, Davi Teixeira, que também trabalhou duro pra me educar e que representa um grande ícone de inspiração e formação da pessoa que me tornei e de quem trago os valores para a vida. Meu irmão Rodrigo, sempre presente mesmo estando distante, com sua mão amiga e palavras de incentivo. Obrigado, vocês são a minha fonte de inspiração sempre.

Agradeço a minha noiva, Michele de Vasconcelos, por ser esta pessoa impressionante que está comigo me incentivando e apoiando em todos os momentos. Agradeço pela força, não só para a conclusão deste trabalho, mas pela paciência em tantos outros que precisei estar ausente. Por sempre me dar aquele empurrão quando estava quase parando. Por ler e reler meus textos e dar suas opiniões sempre construtivas.

Aos amigos, e sócios, Laís Xavier, Leonardo Menezes e Teoria (Rodrigo Carneiro) que sempre me deram grande incentivo para a conclusão da minha graduação e entendendo os momentos de ausência necessários no trabalho.

A Mércio Andrade cujo Mestrado proporcionou o desenvolvimento deste trabalho e pela ajuda na compreensão do todo, para que eu pudesse ter um ponto de partida.

Não posso esquecer o meu professor e orientador Sérgio Campello, pela sugestão do tema deste trabalho me apresentado possibilidades com REEE, uma área que me identifiquei por utilizar tais recursos desde meus primeiros passos em

eletrônica na adolescência. Agradeço também por ter me guiado para que eu pudesse concluir este trabalho pontuando meus erros, para que eu pudesse corrigi-los, e afirmado meus acertos, para me incentivar ainda mais.

Também não posso deixar de mencionar os amigos que encontrei no decorrer desta jornada que é um curso de engenharia. Agradeço muito àqueles que me ajudaram a passar por tantas dificuldades, pelas viradas de noite estudando juntos, desenvolvendo os projetos, incentivando. Obrigado a todos.

Resumo

A utilização de artefatos didáticos criados a partir de Resíduos de Equipamentos Elétricos e Eletrônicos (REEE) foi considerada viável e eficaz pelo trabalho de Andrade (2012), no qual ele criou e utilizou *kits* de treinamento a partir sucatas de equipamentos eletrônicos, entre eles o *Kit* Pêndulo Invertido. Por meio de uma análise deste *kit*, este trabalho propõe a implementação de melhorias a fim de prover um artefato didático para treinamento de estudantes de Engenharia de Computação. Neste trabalho foram implementadas melhorias de *hardware* e *software*. No *hardware* foram adicionados sensores de fim de curso para que fosse possível detectar as laterais e ainda uma interface de comunicação serial RS-232. O *software* de controle embarcado no Arduino foi aprimorado. Uma rotina de inicialização foi criada para determinar o ponto de partida do sistema, foi criada uma série de códigos de comando a fim de ampliar a interação do controlador cliente e ainda a uma nova forma de envio dos dados, sendo agora por requisição, foi definida. Foram apresentados testes realizados com o *kit* original e então apresentados os impactos que as melhorias propostas representaram para a utilização do *kit* definitivo.

Abstract

The use of educational artifacts created from waste electrical and electronic equipment (WEEE) was considered practicable and effective by research conducted by Andrade (2012), in which he created and used training kits using scrap electronic equipment, including Kit Inverted pendulum. Through an analysis of this artifact, this work proposes the implementation of improvements in Inverted Pendulum Kit to provide a didactic device for the training of students in Computer Engineering. In this work were implemented hardware and software improvements. Were added to the hardware limit-switch sensors for detecting the possible lateral and also a communication interface RS-232. The embedded control software in the Arduino has been enhanced. A initialization routine was created to determine the starting point of the system, we created a series of command codes to extend client interaction controller and also a new way of sending data and is now by request, was defined. Tests with the original kit were presented, and then the effects that the proposed improvements accounted for using the final kit were shown.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Siglas

ASCII – *American Standard Code for Information Interchange*

DC – *Direct Current*

FPGA – *Field-programmable Gate Array*

GND – *Ground*

HP – *Hewlett-Packard*

IDE – *Integrated Development Environment*

LAN – *Local Area Network*

PC – *Personal Computer*

PWM – *Pulse Width Modulation;*

REEE – Resíduos de Equipamentos Elétricos e Eletrônicos

RX – *Receive X*

TIC – Tecnologia da Informação e Comunicação

TTL – *Transistor-Transistor Logic*

TX – *Transmit X*

USB – *Universal Serial Bus*

Capítulo 1

Introdução

Há um consenso entre os docentes de disciplinas de informática de que são necessários trabalhos práticos para que os alunos consigam assimilar o conhecimento transferido (SILVA *et al*., 2004). Nos cursos de Engenharia Elétrica, Mecatrônica e de Computação, que têm como disciplinas cálculo diferencial, eletrônica, controle de processos, entre outras, existe a necessidade de recursos didáticos para uso em laboratório e aulas práticas. Esses recursos, muitas vezes, têm um custo relativamente alto e, por isso, nem sempre as universidades podem adquiri-los.

Como alternativa para obtenção desses recursos, Andrade (2012) propôs a montagem de *kits* de treinamento com sucatas de dispositivos de Tecnologia da Informação e Comunicação (TIC). Essa proposta teve como objetivo a produção de artefatos para treinamento de estudantes de Engenharia de Computação e Mestrado em Sistemas, visando aproveitar a maior quantidade de componentes. No trabalho, o autor produziu três *kits* de treinamento para aplicações distintas, sendo um deles o Pêndulo Invertido – de maior complexidade quando comparado aos demais –, que pode ser aplicado em disciplinas relacionadas às áreas da Eletrônica, Controle de Processos e Inteligência Computacional. O Pêndulo Invertido tem seu princípio de funcionamento semelhante ao problema de se tentar equilibrar um bastão na ponta dos dedos ou na palma da mão. Quando o bastão tende a cair em uma determinada direção, este movimento é compensado movendo-se a mão para o sentido da inclinação, fazendo com que o bastão volte a ficar ereto (ANDRADE, 2012).

O princípio do Pêndulo Invertido pode ser encontrado em diversos exemplos reais. Alguns exemplos desta aplicação que podem ser citados são o equilíbrio de um foguete durante o lançamento e o controle de estabilização do veículo de transporte humano Segway (LIMA, 2006) ou do Double (robô da Double Robotics® que usa o *tablet* iPad para, por meio de teleconferência, levar o usuário para qualquer lugar).

1.1 Motivação

Este trabalho buscou uma motivação na utilização do Pêndulo Invertido construído por Andrade (2012), que será denominado no decorrer do documento "*Kit* Pêndulo Invertido", para o uso em aulas práticas ou projetos em diversas disciplinas do curso de Engenharia de Computação. Durante a análise das características do *kit* foram observados alguns indícios de melhorias, propostas para minimizar problemas de controle e para ampliar a compatibilidade com alguns dispositivos embarcados, possibilitando o seu uso em disciplinas com foco em microcontroladores ou FPGAs.

A estrutura mecânica básica do *kit* foi montada a partir de peças de uma impressora jato de tinta HP® *deskjet* 3420. Também foram utilizadas peças de um disco rígido, gabinetes, entre outros (ANDRADE, 2012). Da impressora foram utilizados basicamente a estrutura de movimentação do carro de impressão, o carro de impressão, sensores, fita e disco codificados, apresentados por Andrade (2012).

Primeiramente, foi observado que o sistema não apresenta um ponto de partida para a devida inicialização, sendo necessário posicionar manualmente o carro em uma posição próximo da ótima para o controle. O procedimento de posicionamento manual não garante que a posição escolhida esteja em um ponto ideal para o controle do sistema.

A forma de acoplamento de computadores pessoais (PC) para controle do pêndulo atualmente é possível apenas pela conexão USB da placa Arduino que controla o sistema, não sendo possível utilizar em dispositivos com o padrão RS-232, padrão este bastante comum em aplicações embarcadas.

Além disso, as informações são enviadas via protocolo serial de forma contínua durante a execução do dispositivo que controla o sistema, não permitindo que um controlador externo, como um PC, determine a velocidade de recebimento desses dados, minimizando assim o processamento do lado do controlador cliente.

Ainda há a necessidade do tratamento dos dados provenientes dos sensores, para que estes sejam enviados da forma adequada para a implementação do controlador de equilíbrio do pêndulo, sem que haja necessidade do conhecimento do código embarcado no Arduino.

Por fim, devido aos limites do trilho, podem ocorrer grandes colisões com as laterais, que podem ocasionar danos na estrutura física do pêndulo. Esse levantamento de características foi feito baseado em observações e testes empíricos com o *hardware* supracitado, analisando seu funcionamento e o código fonte do *firmware*, que serão descritos no Capítulo 3.

1.2 Objetivos

O objetivo deste trabalho é a implementação de modificações no *software* embarcado, visando melhorar a estrutura do código fonte e do protocolo de comunicação do sistema com o controlador externo, e no *hardware* do *Kit* Pêndulo Invertido desenvolvido por Andrade (2012), a fim de adicionar outro meio físico de comunicação.

Entre as metas específicas para este trabalho, estão:

(i) implementar uma etapa de calibração para que o carro possa inicializar automaticamente no ponto central do sistema;

(ii) adicionar sensores de fim de curso para detecção do ponto de partida e desligar o motor corrigindo comandos indevidos;

(iii) ampliar compatibilidade com maior variedade de dispositivos de controle através da criação de uma interface serial compatível com RS-232;

(iv) adaptar o código fonte proposto por Andrade (2012) para que o estado dos sensores seja enviado por requisição e não continuamente;

(v) fazer a modelagem de um diagrama de estados do funcionamento do sistema para que sirva como documentação do projeto;

(vi) criação de códigos de comando a fim de ampliar a interação do controlador cliente com o *Kit* Pêndulo Invertido.

1.3 Metodologia

A pesquisa realizada no projeto é de Natureza Aplicada, objetivando gerar conhecimentos para aplicação prática dirigida a alunos de cursos de Engenharia Elétrica, Engenharia de Computação, entre outras. Quanto ao seu objetivo, a

pesquisa é caracterizada como exploratória, fazendo uso de procedimento técnico bibliográfico e de ação.

Desta forma, contempla as seguintes etapas:

1- **Estudo aprofundado sobre funcionamento do Pêndulo Invertido em seu estado original.**

Esse estudo tem enfoque nas características presentes no dispositivo que requerem alguma melhoria, como: (i) sensores utilizados; (ii) a forma como o código embarcado trata os dados dos sensores; e (iii) o tipo de informações que são enviadas via comunicação serial.

2- **Criação de diagrama de estados do sistema.**

A criação do diagrama de estados permite visualizar de forma rápida o funcionamento do *software* embarcado possibilitando uma análise rápida dos estados possíveis e suas transições.

3- **Estudo e inclusão dos sensores.**

Na definição do estado de calibração do sistema será necessário adicionar um sensor de fim de curso, responsável por detectar o ponto de partida do carro, ou seja, a posição inicial no eixo horizontal do sistema. Esta etapa inclui uma pesquisa acerca dos possíveis sensores a serem utilizados no *kit*: Fatores como facilidade de acoplamento do sensor ao sistema, necessidade de modificações no circuito original para gerar a interrupção e facilidade na obtenção do sensor, foram levados em consideração para a escolha. Após a escolha se deu a incorporação do sensor ao sistema.

4- **Implementação da etapa de calibração do sistema.**

No estado de calibração o sistema desloca o carro para a esquerda a fim de encontrar o fim de curso do sistema, ponto em que o motor é desligado e o sistema passa para o estado de centralização. Neste estado o carro é deslocado para o centro do sistema, determinado pelo posicionamento linear do carro. O valor da posição central será obtido através das interrupções geradas pelo sensor de quadratura (Andrade, 2012).

No processo de implantação (i) foi determinado o tratamento da interrupção do sensor de fim de curso; (ii) foi feito o envio do carro para a posição inicial e, em seguida, (iii) a centralização do carro. Mais detalhes podem ser vistos na subseção 4.4, Inicialização do Sistema.

5- Adaptação do conversor RS-232->TTL.

Esta etapa tem a finalidade de fazer a implementação de uma placa conversora serial e buscar por placas disponíveis no mercado compatíveis com Arduino que sejam conversores RS-232->TTL.

6- Estudo e implementação do protocolo de requisição dos dados.

Para a reestruturação do protocolo de comunicação, o *software* embarcado no Arduino responderá às requisições feitas pelo sistema cliente, com os estados atuais dos sensores ou informações já processadas pelo sistema embarcado, tal como o ângulo do pêndulo e deslocamento horizontal. Essa abordagem por requisição será implementada objetivando dar mais flexibilidade ao sistema cliente conectado ao *Kit* Educacional Pêndulo Invertido.

7- Definição da lista de códigos de comando

Os códigos de comando serão utilizados para o controle do *Kit* Pêndulo Invertido. A partir dos códigos ações como movimentar o carro, reiniciar o controle e requisitar os dados serão possíveis, além de poder informar ao controlador cliente acerca de eventos, como a entrada em um determinado estado, e erros, como o envio de um comando inválido.

1.4 Estrutura do Documento

Este trabalho está dividido em 6 capítulos, incluindo este introdutório, que apresenta uma breve introdução ao tema, os objetivos e a metodologia aplicada em seguida o Capítulo 2 traz informações acerca do Pêndulo Invertido. Foi apresentado um contexto histórico, aplicações no mercado e academia e ainda contextualizar o seu uso em projetos educativos.

A análise preliminar do *Kit* Pêndulo Invertido proposto por Andrade (2012) será mostrada no Capítulo 3. Detalhes de seu funcionamento, forma de comunicação e quais recursos foram utilizados para a montagem foram apresentados. Ainda neste capítulo foram propostas as adaptações e melhorias que serão realizadas ao longo deste trabalho.

No Capítulo 4 será apresentado como as devidas adaptações e melhorias propostas no capítulo 3 serão implementadas. Será mostrado um diagrama de estados para o sistema que guiará as modificações no *firmware* e além do detalhamento dos códigos de comando definidos. Acerca do *hardware* do sistema foi mostrado como os sensores e a nova interface RS-232 foi adicionada.

Após a implementação das melhorias propostas o Capítulo 5 apresentará os resultados alcançados neste trabalho e as dificuldades encontradas em torno do desenvolvimento dessas melhorias e, sem seguida, a conclusão e trabalhos futuros serão comentados no Capítulo 6, mostrando as dificuldades encontradas e as possibilidades de outras modificações necessárias para melhorar o projeto.

Neste documento foi utilizado o termo “controlador cliente” para se referir a qualquer tipo de *hardware* externo que poderá ser desenvolvido ou utilizado e que se comunique com o sistema embarcado do *Kit* Pêndulo Invertido.

Capítulo 2

O Pêndulo Invertido

O objetivo deste capítulo é apresentar brevemente o contexto histórico do pêndulo invertido, bem como sua definição e características, o modo como é geralmente implementado e os tipos encontrados no mercado e no meio acadêmico. Ainda neste capítulo apresentar-se-á o *Kit* Pêndulo Invertido desenvolvido por Andrade (2012), sua estrutura e principais características funcionais.

2.1 Contexto Histórico

A estabilização do Pêndulo Invertido é uma curiosidade bem conhecida na mecânica clássica. Stephenson (1908) apontou um tipo de estabilidade dinâmica que define que um pêndulo comum, plano e rígido, cujo eixo é forçado a oscilar ao longo a linha vertical, torna-se estável na posição invertida se a amplitude e a frequência estiverem em certos intervalos. Nas pequenas e moderadas variações, a partir da posição invertida, o pêndulo não mostra tendência a virar para baixo. Sendo desviado, o pêndulo executa oscilações relativamente lentas sobre a linha vertical, além de rápidas oscilações no ponto de suspensão. Devido ao atrito estas oscilações lentas são amortecidas gradualmente e o pêndulo, eventualmente, vem para a posição vertical invertida (BUTIKOV, 2011).

A partir dessa abordagem, provavelmente primeiramente proposta por Stephenson, o autor provou que sobre uma vibração deste eixo o pêndulo pode-se preservar estável na posição vertical. Mais tarde, Kapitza (1951) forneceu quase independentemente a solução para o mesmo problema. Ele usou um simples método heurístico que alavancou o aparecimento de uma nova seção da Teoria das Oscilações Não-lineares para a mecânica vibratória (VASILKOV, 2007). Dessa forma, o comportamento extraordinário do pêndulo estava fisicamente explicado e investigado experimentalmente em detalhes (BUTIKOV, 2011). O dispositivo físico correspondente é mencionado na literatura por Vasilkov (2007) como “Pêndulo de

Stephenson-Kapitza". Na Figura 2.1 pode ser visto o pêndulo proposto por Kaptiza (1951).

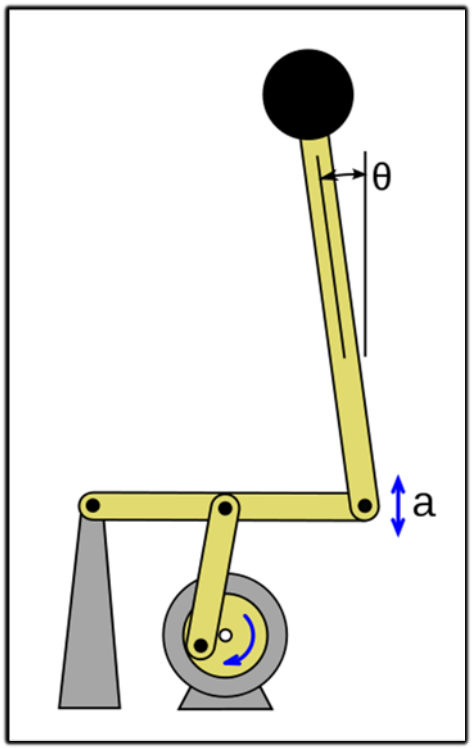


Figura 2.1 Pêndulo de Kapitza. Fonte: WIKIPEDIA, 2012.

Além do pêndulo de Stephenson-Kapitza, o qual tem o pivô sendo movimentado verticalmente, existem outras implementações do pêndulo invertido onde o controle deve ser efetuado horizontalmente, como é o caso do exemplo proposto por Poorhossein (2010), cujo objetivo é equilibrar um pêndulo, aplicando-se uma força $\vec{F}$ e ajustando o ângulo $\boldsymbol{\theta}$, montado em um carrinho móvel, mostrado na Figura 2.2. Brunauer (2007) propôs o mesmo arranjo e afirmou que ajustando a velocidade e a direção do carrinho, um controlador consegue manter o pêndulo na posição vertical.

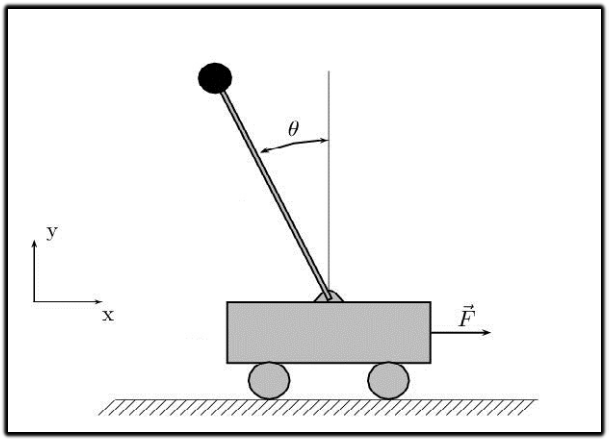


Figura 2.2 Pêndulo montado sobre um carrinho. Fonte: POORHOSSEIN, 2010.

2.2 Aplicações

Sistemas com pêndulos invertidos têm características muito interessantes, por isso muitos pesquisadores têm interesse em expandir suas pesquisas desenvolvendo sistemas mais avançados. Os sistemas com pêndulo invertido têm sido ampliados para diversas aplicações como controle de caminhada humana, sistema de pêndulo móvel, sistemas sobre atuação, entre outros (AHN, 2008).

Por isso, o ensino de controle de sistemas tornou-se mais importante na área de Engenharia de Controle, devido a tecnologias “estado-da-arte” demandarem algoritmos de controle mais sofisticados (JUNG, 2011), reforçando a necessidade de recursos educacionais para lecionar tal conteúdo.

2.2.1 Contexto Educacional

Sistemas inteligentes têm despertado bastante atenção pela comunidade de controle de sistemas. Inteligência tem sido uma palavra chave importante para a próxima fronteira de projetos de sistemas robotizados e sistemas de comunicação. Especialmente nas comunidades de controle, controles inteligentes têm crescido como uma das mais efetivas áreas. Assim, há a necessidade de educar estudantes em sistemas de controle inteligentes (LEE, 2008).

Para uma educação efetiva de métodos de controle avançados, teorias de controle devem ser explicadas por meio de demonstrações experimentais. Estudos experimentais ajudarão estudantes a entender as teorias de controle com facilidade e podem compensar a falta de simulações para aplicações de controle avançadas, uma vez que o mundo real tem que lidar com muitos problemas incertos (JUNG, 2011).

Além disso, segundo Felder (1988), há indicações de que os estudantes de engenharia são mais propensos a serem ativos do que os reflexivos. Alunos ativos são aqueles que assimilam mais o conhecimento que lhes é apresentado fazendo alguma coisa externa com a informação - discutindo, explicando ou testando algo de alguma forma. Um aluno ativo é alguém que se sente mais confortável com, ou é melhor em experimentações ativas do que em observações reflexivas (FELDER, 1988).

Sistemas de pêndulo invertido têm sido dominantemente utilizados como o sistema de base experimental, pois este tipo de sistema tem algumas características particulares. Primeiramente, uma única entrada controla duas saídas, um ângulo e uma posição, o qual é considerado como um sistema de uma única entrada e múltiplas saídas. Uma única entrada $\boldsymbol{u}$ tem que controlar duas variáveis, ângulo $\boldsymbol{\theta}$ e posição $\boldsymbol{x}$ para satisfazer o rastreamento da posição desejada enquanto balança. Assim, combinações adequadas entre o controle do ângulo e controle da posição proporcionam performances bem-sucedidas. Segundo, o sistema de pêndulo invertido é também um sistema não-linear, mas pode ser linearizado. Por fim, o sistema é de baixo custo possibilitando facilmente sua implementação (JUNG, 2011).

No mercado há diversos modelos comerciais de *kits* destinados à educação. Podem ser destacados dois que produzem o experimento do pêndulo invertido: o modelo GLIP2001, que pode ser visto na Figura 2.1, da empresa chinesa Googol Technology® e o modelo IP0, fabricado pela empresa canadense Quanser®, mostrado na Figura 2.2 (ANDRADE, 2012).

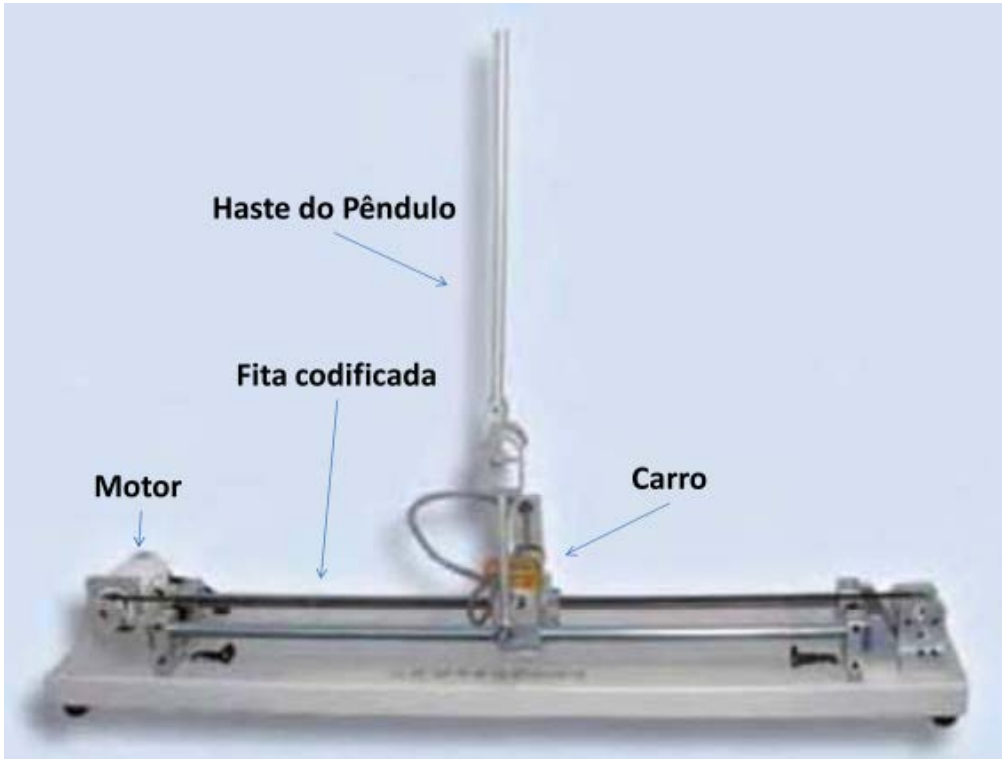
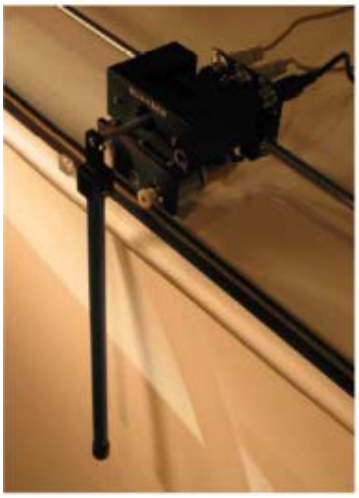
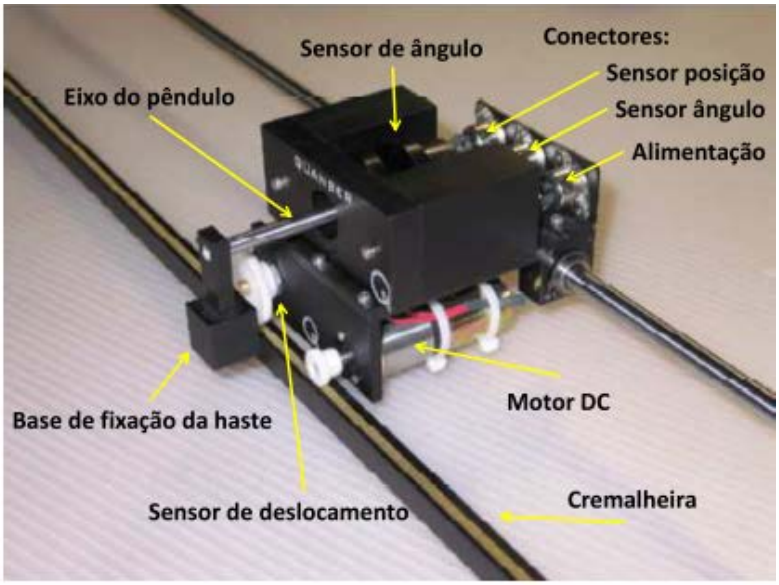


Figura 2.3 Pêndulo modelo GLIP2001 da *Googol Technology*®*.* Fonte*:* ANDRADE, 2012.



(a)



(b)

Figura 2.4 (a) Pêndulo modelo IP02 da *Quanser*® e (b) detalhe do carro de suporte. Fonte: ANDRADE, 2012.

2.2.2 Aplicações Comerciais

Existem diversas aplicações comerciais que utilizam os princípios do pêndulo invertido além das já citadas no contexto educacional. Algumas dessas aplicações são apresentadas a seguir.

Uma aplicação bastante conhecida é o veículo elétrico de transporte pessoal da empresa norte-americana *Segway*®, o *Segway* PT (*Personal Transportation*), mostrado na Figura 2.3. Neste dispositivo, o pivô do pêndulo é o eixo de uma roda ou par de rodas. Em um *Segway Human Transporter* (Segway de Transporte Humano), a roda é impulsionada por um motor elétrico. O movimento da roda, ou par de rodas é controlado de modo que o pêndulo é dinamicamente equilibrado (LIMA, 2006).

Figura 2.5 Veículo de transporte *Segway*. Fonte: SEGWAY, 2012.

Outra aplicação comercial que se pode citar é o Double, da *Double Robotics*® (www.doublerobotics.com). O Double é um dispositivo utilizado para se comunicar remotamente com pessoas ou grupos de pessoas. É utilizado em conjunto com o

iPad, *tablet* da *Apple*®, para permitir a visualização e a comunicação com o ambiente onde se encontra. O usuário deve encaixar seu iPad ao Double e o conjunto será equilibrado e controlado remotamente, podendo se locomover e caminhar virtualmente pelo ambiente, seja um escritório, uma loja ou o campus de uma universidade. A Figura 2.4 mostra o dispositivo e algumas especificações técnicas.

Figura 2.6 Double, da *Double Robotics*®. Fonte: DOUBLE ROBOTICS®, *2012*.

Além dessas, uma abordagem *open-source* de um veículo semelhante ao *Segway* citado anteriormente é o *Elektor OSPV1* (*Open Source Personal Vehicle V1)* desenvolvido pela equipe da revista *Elektor,* mostrado na Figura 2.5. O *kit* está disponível para compra e seu código fonte pode ser baixado no próprio *site da Elektor* (ELEKTOR, 2012).

Figura 2.7 *Elektor OSPV1.* Fonte: ELEKTOR, 2012.

2.3 Apresentação do *Kit* Pêndulo Invertido

O *Kit* Pêndulo Invertido foi concebido para reproduzir o experimento do pêndulo invertido como forma de reaproveitar as peças comumente presentes nas sucatas das impressoras jato de tinta. Este *kit* utiliza as peças como disco codificado, sensor de quadratura, motor e outras, comumente usadas como base por diversos modelos de impressoras a jato de tinta fabricados pela HP®, além de peças de discos rígidos, gabinete de computadores pessoais, entre outras. O *kit* foi construído baseado nos princípios de funcionamento dos modelos apresentados na subseção 2.2.1 (ANDRADE, 2012). O diagrama em blocos do *Kit* Pêndulo Invertido pode ser visto na Figura 2.8.

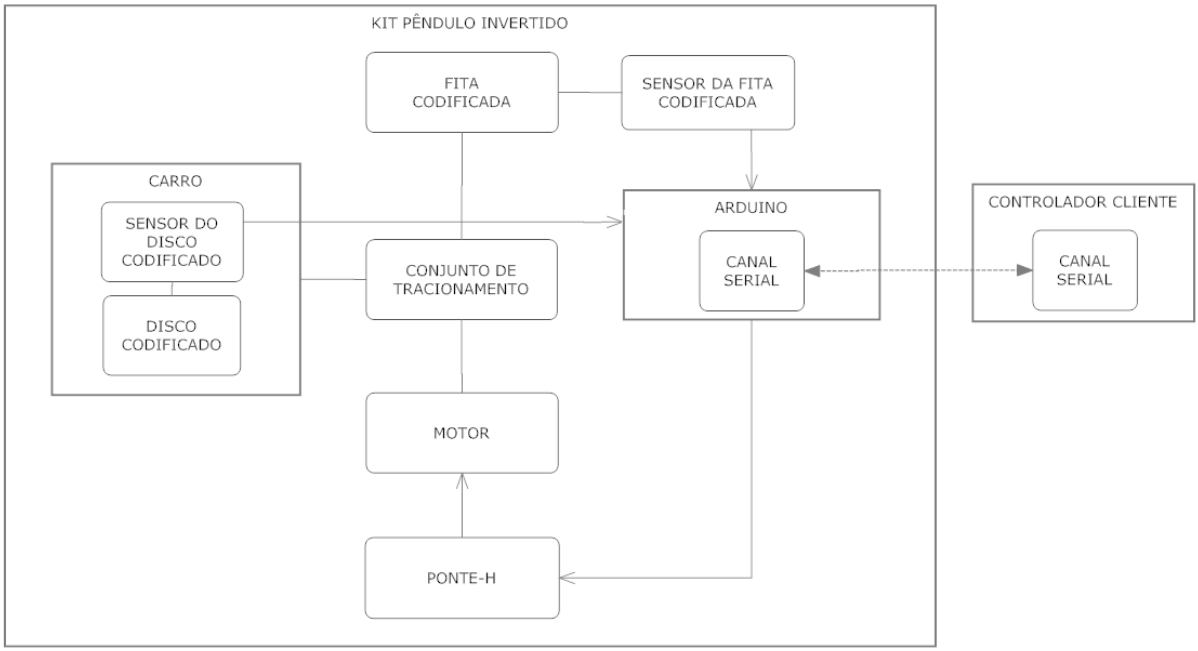


Figura 2.8 Diagrama em Blocos do *Kit* Pêndulo Invertido.

No diagrama de blocos podemos observar que a existem dois sensores no sistema: o sensor do disco codificado, que está acoplado ao carro, e o sensor da fita codificada, acoplado ao conjunto de tracionamento do carro. Ambos os sensores são ligados aos pinos de interrupção do Arduino. Podemos ver também que o motor é controlado pelo Arduino através de um circuito de ponte-h, necessário para controlar o motor nos dois sentidos. O sistema é controlado por um controlador cliente que envia comandos através do interface serial. O projeto do *Kit* Pêndulo Invertido pode ser visto na Figura 2.9.

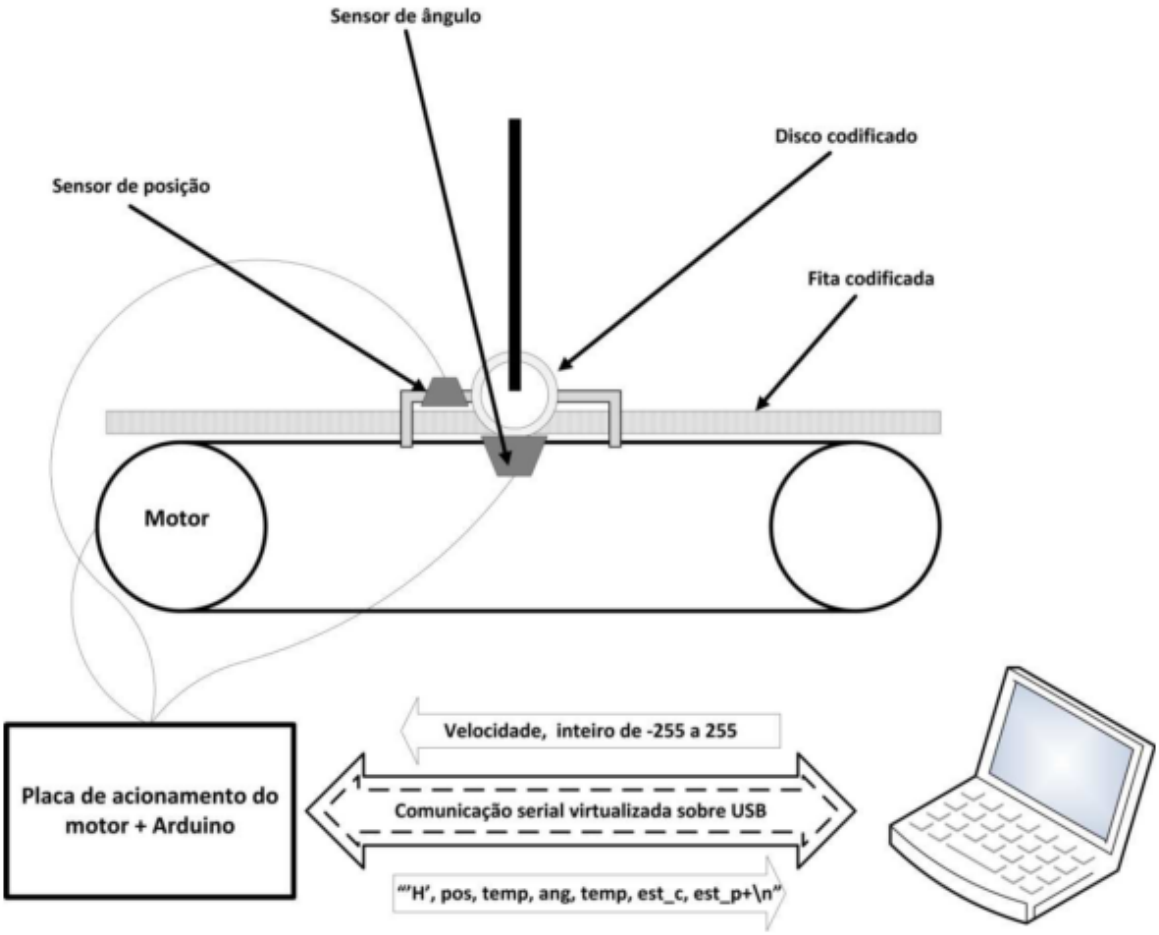


Figura 2.9 Projeto do *Kit* Pêndulo Invertido.

O ângulo de inclinação do pêndulo é obtido por meio do sensor construído a partir do sensor de tracionamento do papel da impressora. Este sensor foi fixado ao carro de impressão juntamente com o motor de um disco rígido, utilizado como pivô, que tem o papel unicamente de fixação do disco e da haste do pêndulo. Vale salientar que o motor do disco rígido não foi alimentado, estando livre para a haste balançar sem que haja qualquer força atuante por parte deste motor, como mostra a Figura 2.10. Vale salientar que o motor do disco rígido não foi alimentado, estando livre para a haste balançar sem que haja qualquer força atuante por parte deste motor (ANDRADE, 2012).

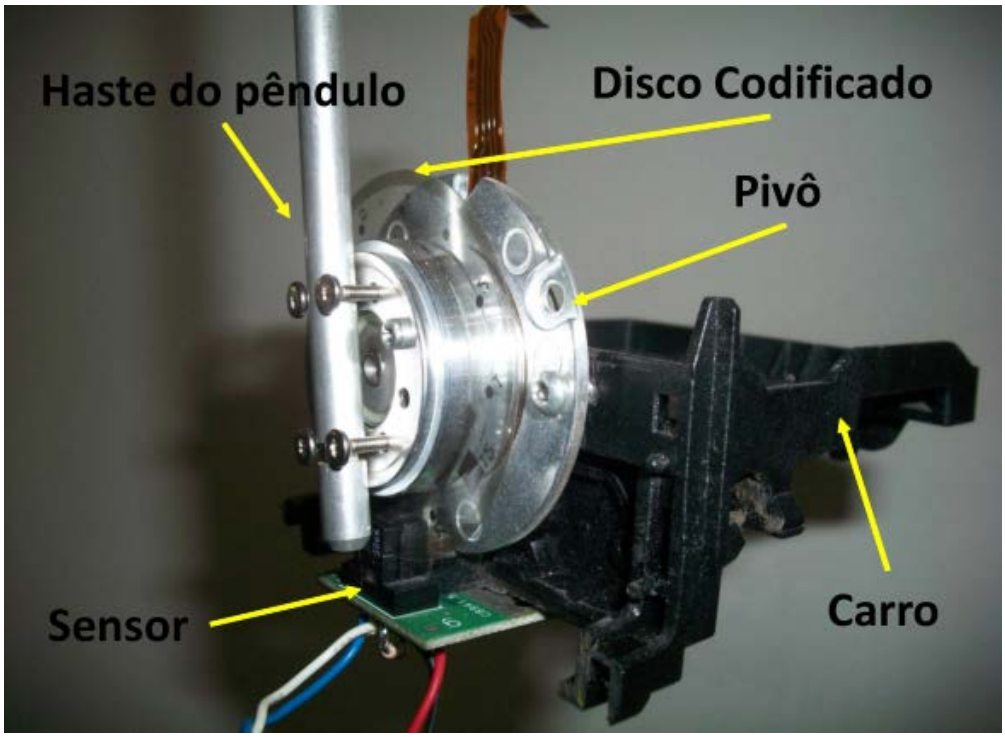


Figura 2.10 Carro de suporte do *Kit* Pêndulo Invertido (ANDRADE, 2012).

Além de determinar a posição angular, foi preciso também determinar a posição horizontal do carro em relação ao ponto inicial do sistema. Para isso, foi utilizado o próprio mecanismo da impressora composto de sensor e fita codificada. Apesar de continuar na placa do circuito original, foi preciso cortar as trilhas que fazia sua conexão ao resto do circuito. A partir disto foi feita a alteração necessária para operação do sensor (ANDRADE, 2012). O detalhe do conjunto – carro e fita codificada – pode ser visto na Figura 2.11. O disco e a fita codificada apresentam o mesmo princípio de funcionamento descrito por Andrade (2012).

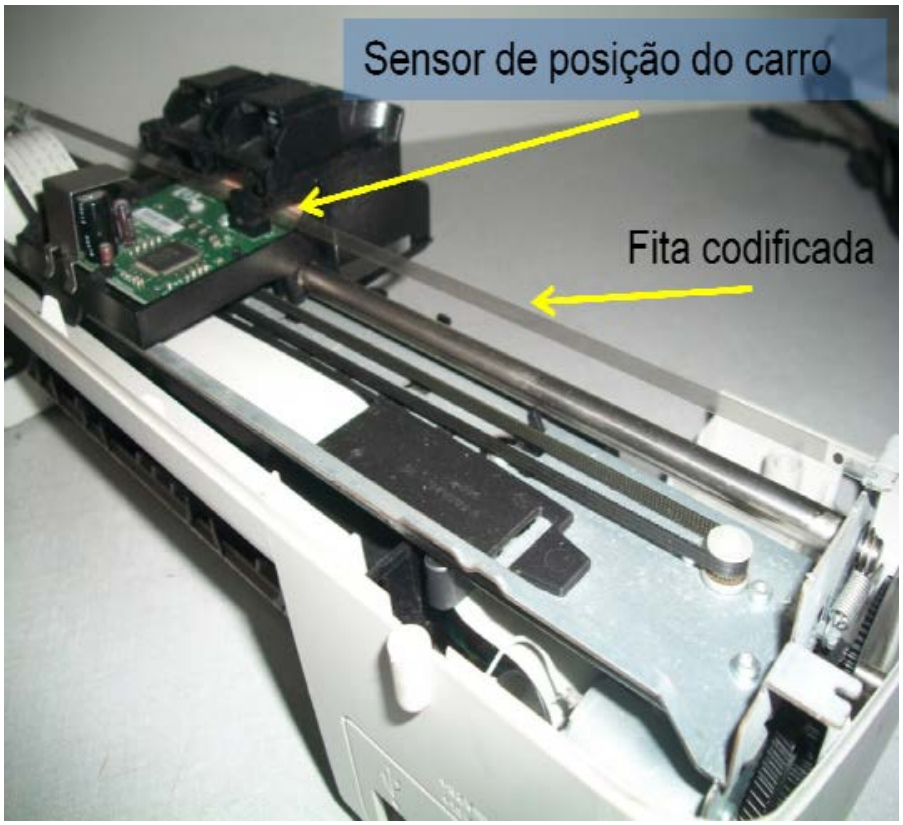


Figura 2.11 Sensor de deslocamento horizontal do *kit* (ANDRADE, 2012).

Para fazer o controle e receber os dados dos sensores foi utilizada a plataforma aberta de *hardware* e *software* Arduino, bastante utilizada em escolas e universidades. A versão utilizada foi a *Duemilanove,* baseada no microcontrolador AVR Atmega 328. As características do Arduino *Duemilanove* são apresentadas na Tabela 2.1 e o *kit* completo montado em base de madeira pode ser observado na Figura 2.12. A plataforma é compatível com os sistemas *Windows*®, *Linux* ou *OS X* (ANDRADE, 2012, *apud* SARIK, 2010).

Além da plataforma Arduino original utilizada no projeto foi preciso montar um circuito para controlar o motor DC que movimenta o carro. Este circuito, composto basicamente por uma ponte-h e quatro diodos, pode ser visto na Figura 2.13 juntamente com o microcontrolador do Arduino e os sensores de quadratura descritos por Andrade (2012).

Tabela 2.1 Características do Arduino Duemilanove (www.arduino.cc)

Microcontrolador	ATmega168/328
Tensão de operação	5V
Tensão de entrada (recomendado)	7-12V
Tensão de entrada (limites)	6-20V
Pinos digitais de E/S	14 (dos quais 6 provêm saída PWM)
Pinos analógicos de entrada	6
Corrente DC por pino de E/S	40 mA
Corrente DC por pino 3.3V	50 mA
Memória *Flash*	16 KB (ATmega168) ou 32 KB (ATmega328) dos quais 2 KB são utilizados pelo *bootloader*
SRAM	1 KB (ATmega168) ou 2 KB (ATmega328)
EEPROM	512 bytes (ATmega168) ou 1 KB (ATmega328)
Clock	16 MHz

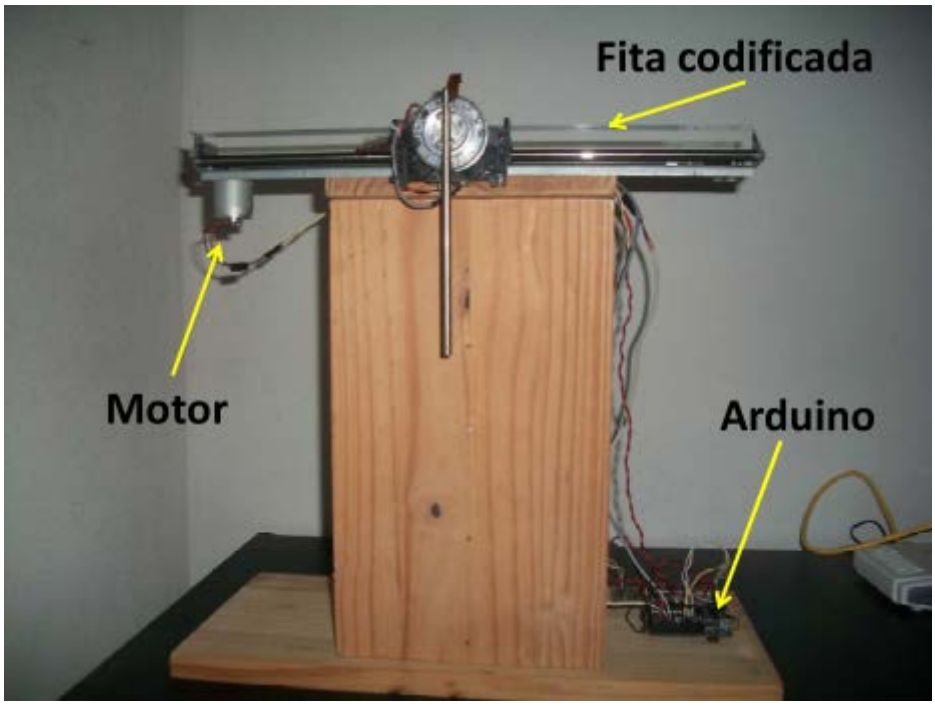


Figura 2.12 *Kit* Pêndulo Invertido montado (ANDRADE, 2012).

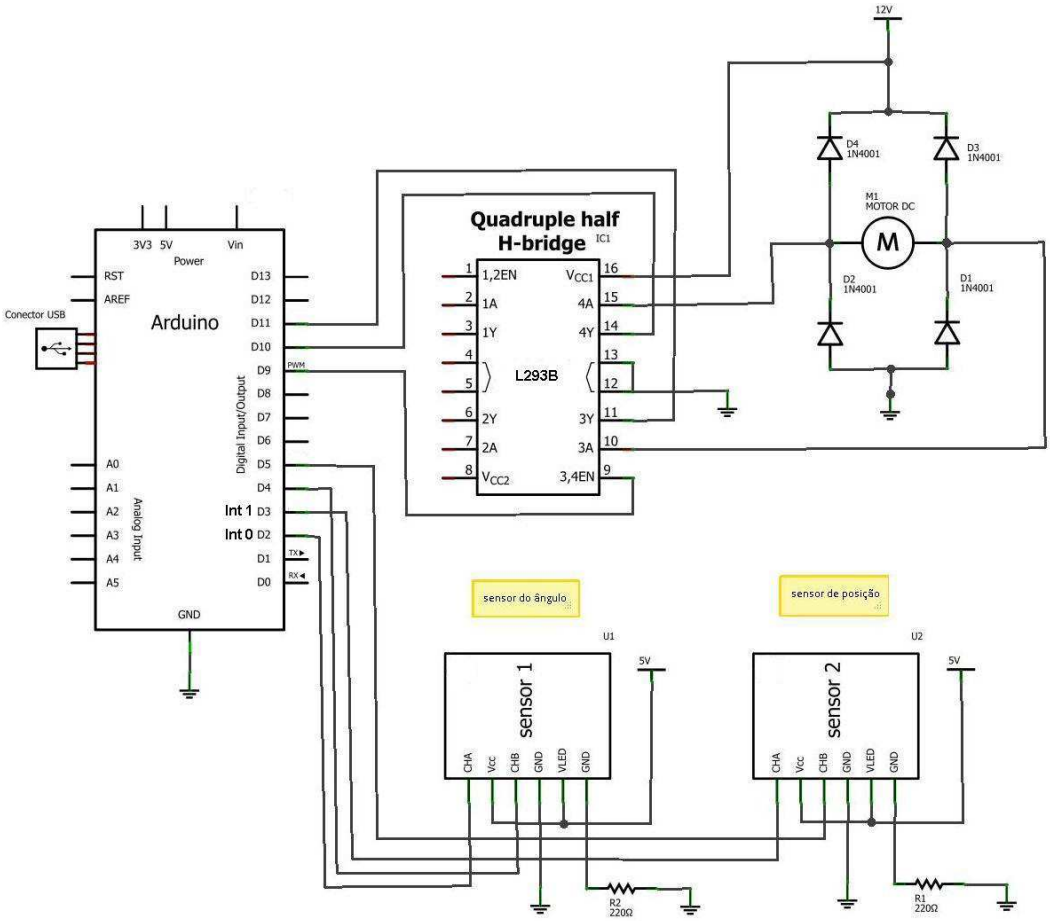


Figura 2.13 Circuito Elétrico do *Kit* Pêndulo Invertido (ANDRADE, 2012).

2.4 Resumo do Capítulo

Neste capítulo está descrita uma breve introdução acerca do que é e como funciona o pêndulo invertido. Foi apresentado o contexto histórico citando os precursores da descrição e resolução do problema. Discutiu-se sobre suas aplicações comerciais apresentando modelos de produtos já presentes no mercado e, sobretudo, sua inserção no contexto educacional no âmbito do controle de sistemas.

Capítulo 3

Análise do *Kit* Pêndulo Invertido

O *Kit* do Pêndulo Invertido proposto por Andrade (2012) teve como objetivo servir como recurso educacional para os cursos de Engenharia de Computação e Mestrado em Sistemas, reforçando assim o aprendizado ativo dos alunos dos cursos citados. Como alternativa na obtenção de recursos educacionais, e para proporcionar cada vez mais o aprendizado ativo, Andrade (2012) propôs a montagem do *Kit* do Pêndulo Invertido a partir de REEE (Resíduos de Equipamentos Elétricos e Eletrônicos).

Neste capítulo será apresentada uma análise feita do *Kit* do Pêndulo Invertido visando verificar as melhorias necessárias a fim de tornar o *kit* próprio para aplicação como recurso didático em disciplinas de Engenharia de Computação.

3.1 Análise do *Hardware*

3.1.1 Colisões

Através da análise do *Kit* Pêndulo Invertido por meio de testes práticos controlados a partir de *softwares* criados para este propósito, foi possível observar a existência de um grande problema de colisões com as laterais da estrutura metálica que sustenta todo o sistema. As colisões ocorrem devido a outro problema encontrado durante os testes que é a persistência do motor, que traciona a correia para mover o carro, em se manter ligado indefinidamente até que haja um comando de um controlador cliente para desligamento do mesmo. A persistência do motor ligado sem que haja um controle de desligamento automático pode causar diversos danos ao sistema, principalmente ao próprio motor, a correia e ao circuito *driver* do motor composto pela ponte-h.

Esse problema pode ser observado no vídeo disponível em (SILVA, 2012), no qual o *Kit* Pêndulo Invertido está conectado ao computador por meio da porta USB. O sistema recebe um comando de deslocamento através do monitor serial da própria

IDE do Arduino. Os valores para o comando de deslocamento do carro podem assumir inteiros entre -255 e 255. Os valores positivos deslocam o carro para a direita enquanto que os valores negativos deslocam o carro para a esquerda. O motor é desligado para o valor zero. A velocidade do deslocamento do carro é proporcional ao valor absoluto escolhido, sendo o valor 1 representante da menor velocidade e 255 da velocidade máxima.

Os testes foram efetuados com valores múltiplos de 51, sendo feitos então cinco testes, com os valores 51, 102, 153, 204 e 255. Esse problema pode ser observado no vídeo “Análise do *Kit* Pêndulo Invertido - Colisões” disponibilizado por Silva (2012). Observou-se que para os valores 51 e 102 não houve grande intensidade de força aplicada ao carro, fazendo com que este se desloque com uma velocidade muito pequena. A partir do valor 153 o carro passa a se deslocar com uma velocidade bem maior e que a colisão passa a ter um impacto significativo. No valor máximo o carro fica preso na lateral direita e a correia passa a sofrer uma tração muito forte, podendo ser observado o deslize desta nas engrenagens que formam o conjunto de tração do carro.

3.1.2 Interface de Comunicação

Como já foi apresentado na Seção 2.3, o *Kit* Pêndulo Invertido foi montado a partir de componentes de REEE e utiliza a plataforma Arduino para fazer o controle embarcado ao sistema. A comunicação serial do Arduino é feita através dos pinos digitais 0 e 1, que são o RX e TX respectivamente. Para fazer a interface do *kit* com um PC, a placa Arduino utilizada dispõe de uma porta USB. No entanto, não é possível efetuar o controle, com o PC ou com outro dispositivo, a partir de uma porta serial RS-232, pois não há o conector DB9, comumente utilizado para comunicação serial, no Arduino e este também não reconhece o padrão RS-232.

Através da porta RS-232 é possível efetuar a comunicação serial por diversos meios. No mercado há, por exemplo, conversores de LAN para RS-232, como é o caso dos adaptadores NETRS2321E da StarTech e do NA-4021 da MOXA©, permitindo a comunicação com o dispositivo dotado de RS-232 por uma rede local. Esses adaptadores podem ser vistos na Figura 3.1.

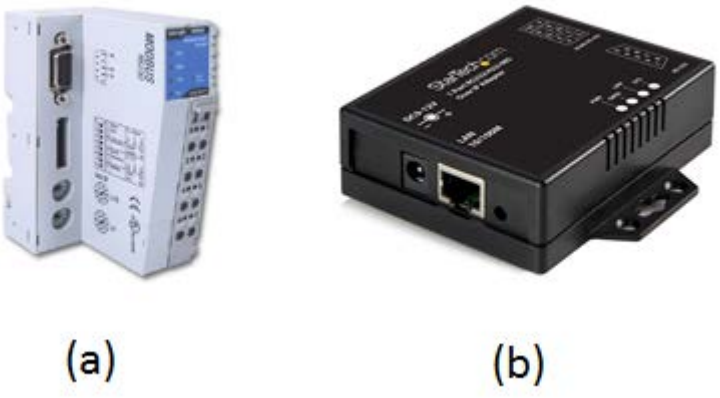


Figura 3.1 Conversores de LAN para RS-232. (a) NA-4021 (MOXA) e (b) NETRS2321E (STARTECH, 2012).

3.2 Análise do *Software* Embarcado

3.2.1 Inicialização

Durante a análise inicialmente apresentada na Seção 3.1 foi possível verificar que o sistema não é dotado de uma rotina de inicialização. A posição de referência, ou seja, a origem do sistema na horizontal é aquela onde se encontra o carro. Como esta posição é arbitrária, o controlador cliente não tem conhecimento do espaço possível de se percorrer com o carro até que haja uma colisão com algum dos lados, mesmo que este receba o valor da posição atual do carro. Apesar de ser possível determinar uma posição inicial, movendo-se o carro manualmente, esta posição seria difícil de determinar.

3.2.2 Transmissão dos Dados

A placa Arduino utilizada contém um chip FT232RL, que é responsável por fazer a interface USB para serial na placa (FTDI™ Chip). Sendo assim, o controle do *Kit* Pêndulo Invertido é controlado por uma porta COM*X*, onde *X* é o número da porta instalada no PC pelo reconhecimento do *chip* mencionado. Então, a rotina de configuração é a mesma quando o controle ocorre pela USB ou pelos pinos de comunicação serial, sendo necessário apenas executar a função `Serial.begin(speed)`, onde "*speed"* é a taxa de transmissão serial.

Assim que o sistema é ligado, os dados começam a ser transmitidos pelo sistema embarcado em um laço infinito na velocidade da taxa de comunicação serial. Este envio de informação indefinidamente, sem um controle específico, não deixa o sistema flexível para que um controlador cliente possa obter essas informações à medida que necessite, deixando o canal de comunicação sem congestionamento.

3.2.3 Comandos

Não existe qualquer comando, além do valor de controle da direção e intensidade do movimento do carro, que o cliente possa enviar para o sistema a fim gerenciar melhor seu funcionamento geral. Comandos como reinicialização do sistema, requisição dos dados e reposicionamento do carro no estado inicial não foram implementados devidos às observações já citadas nas subseções anteriores.

3.2.4 Eventos e Exceções

Por fim, não existe envio de informações acerca de eventos e exceções, para que o controlador cliente saiba de possíveis problemas ocorridos, como colisões, o não recebimento de informações e o recebimento de dados inválidos.

3.3 Resumo do Capítulo

Neste capítulo foi apresentada uma análise geral do *Kit* Pêndulo invertido. Por meio de testes práticos executados no *kit* pôde-se observar alguns pontos que necessitam de melhorias para que o sistema funcione de forma mais estável e controlada. Em relação ao *hardware* foi apresentado o problema com colisões do carro com as laterais e a interface de comunicação apenas pela USB. Na parte de *software* a falta de uma inicialização controlada, determinando o ponto de partida do carro, por exemplo, a transmissão contínua e sem controle por parte do cliente e a inexistência de um código de exceções e comandos como inicialização foram as questões abordadas. Pretende-se neste trabalho propor melhorias para os itens analisados neste capítulo. Observou-se também que, além das melhorias no *hardware* para prover o melhor sensoriamento e interface com outros dispositivos, é preciso uma revisão no *software* embarcado como um todo.

Capítulo 4

Adaptação e Implementação das Melhorias

O objetivo deste capítulo é descrever como foram feitas as adaptações e implementações necessárias no *hardware* e no *software* para as propostas de melhorias já citadas na seção Objetivos do capítulo introdutório.

4.1 Modelagem de Estados do Sistema

Para que houvesse maior confiabilidade nas mudanças propostas, sobretudo no *software*, foi desenvolvido um diagrama de estados simplificado do funcionamento do sistema. Este diagrama foi utiliza no decorrer do processo de implementação como guia.

Sete estados foram propostos para a criação do diagrama. Dos sete, três compõem uma rotinha macro de inicialização, sendo eles INICIALIZAR, CALIBRAR e CENTRALIZAR. Em outros três estados temos os estados PROCESSANDO COMANDO, ESPERA e EXCEÇÃO. O diagrama de estados completo pode ser visto na Figura 4.1.

Quando o sistema é alimentado e o Arduino inicia o processamento o estado do sistema passa a ser INICIALIZAR, no qual é verificado se o carro já se encontra na posição inicial do eixo horizontal, que no neste caso seria a extrema esquerda. Caso não esteja o sistema passa então para o estado CALIBRAR, que é responsável por comandar o carro para a posição inicial e detectar quando esta é alcançada. Quando a posição inicial é encontrada o estado do sistema é alterado para CENTRALIZAR. Neste estado o carro é comandado para se mover para a direita a fim de encontrar a posição central do sistema físico. Quando esta posição é encontrada o carro para, e o estado é alterado para ESPERA.

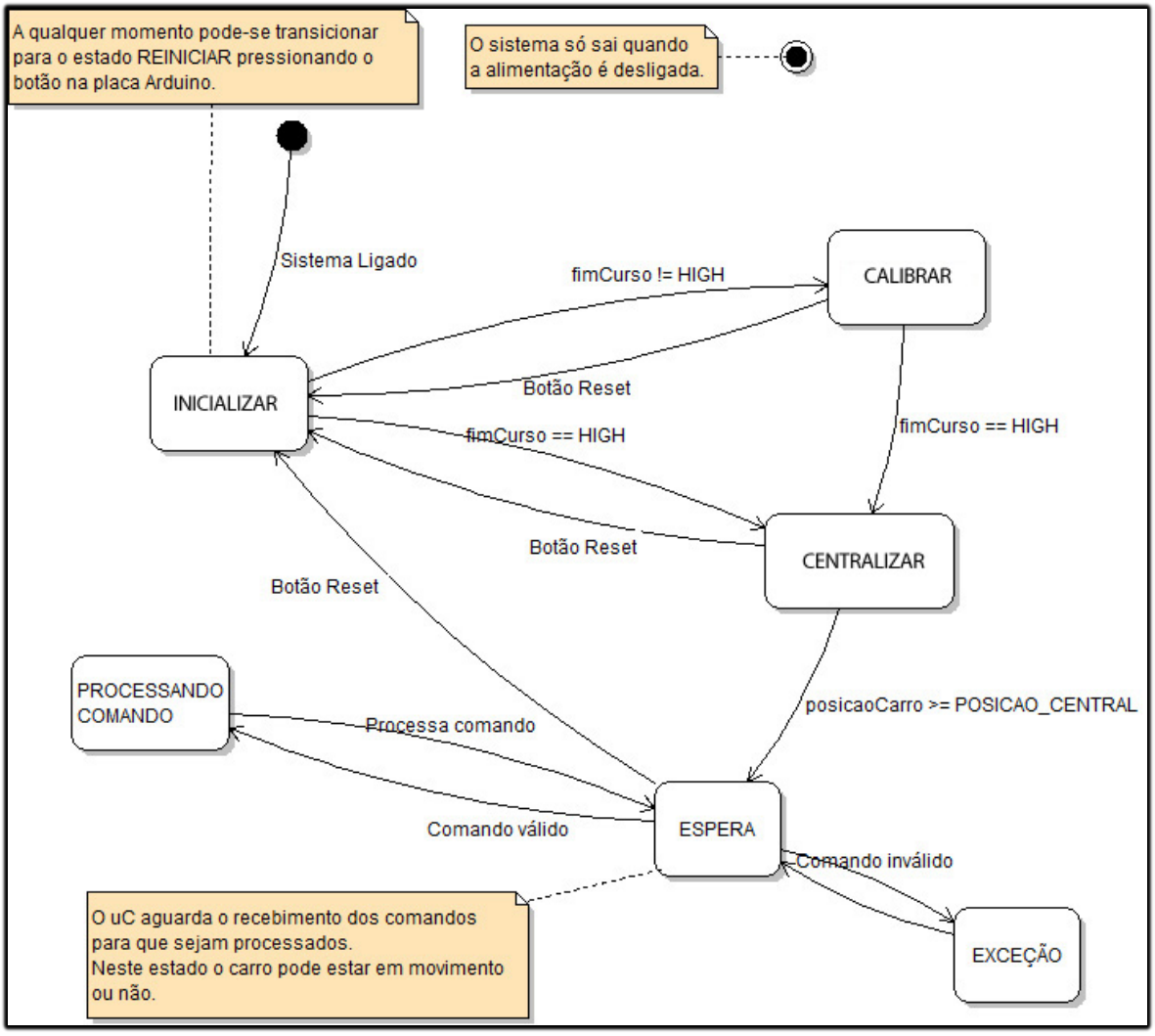


Figura 4.1 Diagrama de estados proposto para o do *Kit* Pêndulo Invertido.

O estado ESPERA é aquele que aguarda os comandos possíveis para o controle do sistema, incluindo a movimentação do carro. Se um comando inválido for enviado, o sistema passa para o estado de EXCEÇÃO e envia um comando para o controlador cliente. O estado PROCESSANDO COMANDO é alcançado quando um comando válido é enviado para o sistema embarcado, este então é processado e o estado corrente volta a ser o ESPERA.

Por fim, caso o botão de *reset* da placa Arduino seja pressionado o sistema volta para o estado INICIALIZAR após a conclusão da rotina de inicialização do Arduino.

4.2 Sensores de Fim de Curso

Para que seja possível realizar um tratamento de exceções e proteger o *hardware* de colisões foram adicionados dois sensores de fim de curso. Os sensores de fim de curso são utilizados para saber se o carro encontrou algum dos limites laterais do trilho. Caso estes limites sejam encontrados um mudança de estado nos pinos do Arduino permitirá a tomada de decisões para tratar o problema.

Para determinar qual sensor utilizar foram analisados três tipos: um sensor mecânico, um sensor óptico e um sensor magnético, conhecido como *reed switch*. O sensor mecânico utilizado foi uma chave de pressão normalmente aberta, conhecida como *micro switch,* que pode ser vista na Figura 4.2. O sensor óptico, que pode ser observado na Figura 4.3, foi retirado de uma impressora HP©. Este mesmo tipo de sensor é utilizado também nas impressoras com o propósito de identificar se a tampa estaria levantada ou não. O sensor magnético utilizado foi o *reed switch*, adquirido no mercado local de componentes eletrônicos e é mostrado na Figura 4.4.

Os critérios para a determinação de qual tipo de sensor seria utilizado foram o esforço necessário nas modificações no circuito original para gerar a interrupção, ou mudança de estado em determinado pino, a facilidade na obtenção do sensor em REEE ou no mercado e por fim a complexidade de acoplamento do sensor ao sistema físico.

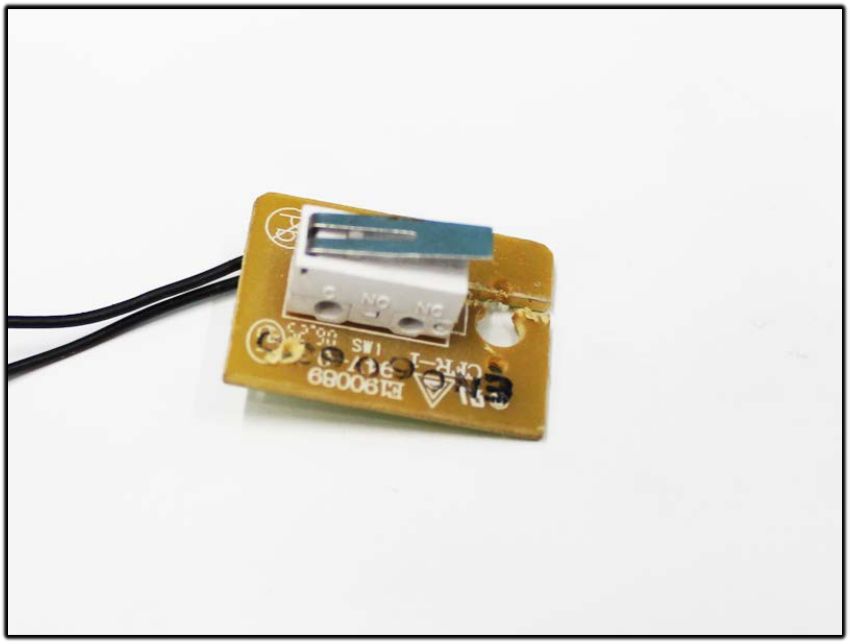


Figura 4.2 Sensor mecânico - chave normalmente aberta.

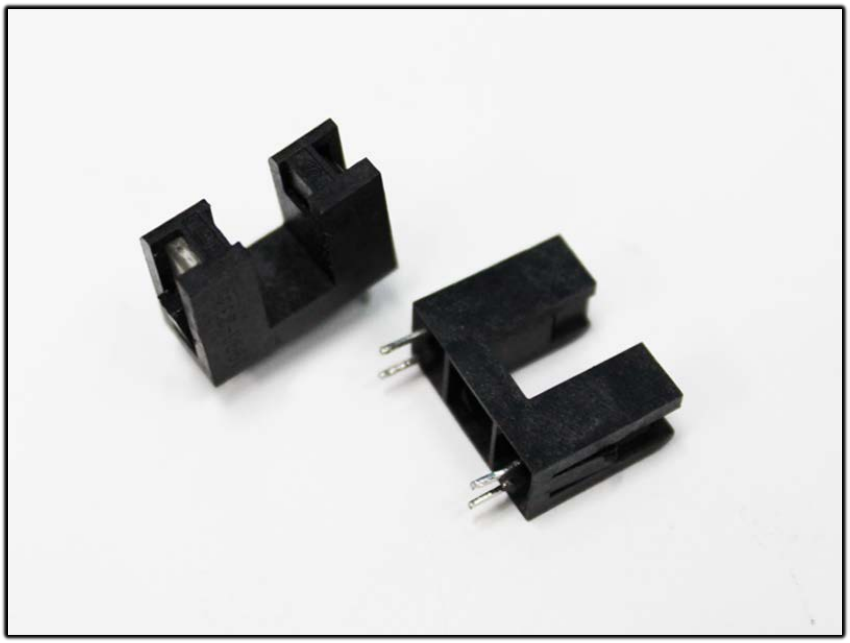


Figura 4.3 Sensor óptico de impressora HP©.

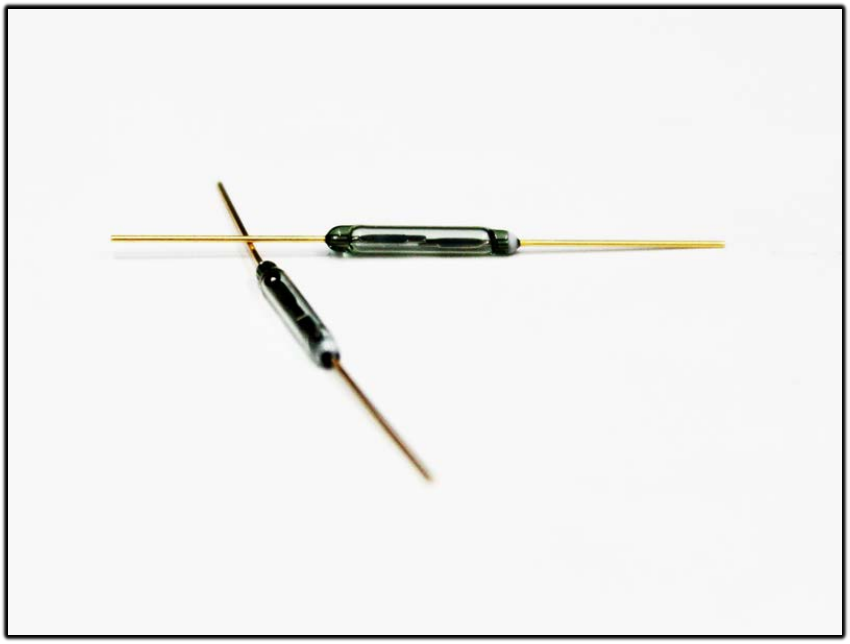


Figura 4.4 Sensor magnético *reed switch*.

4.2.1 Modificações no Circuito Original

Dos sensores analisados, a chave mecânica e o *reed switch* funcionam com o fechamento dos seus contatos metálicos. No primeiro isto ocorre por meio de um contato físico, do carro com a chave, por exemplo, e no segundo por meio de um campo magnético a certa distância do sensor. Essa característica torna o acoplamento destes ao *kit* mais fácil, pois para gerar uma interrupção nos pinos do Arduino não há necessidade de um circuito além dos fios e um resistor. Em relação ao sensor óptico, seu funcionamento foi analisado e para utilização deste, um circuito precisaria ser montado com o sensor. Além da montagem do circuito seria necessário definir como a interrupção da luz seria feita para gerar o sinal de fim de curso.

4.2.2 Aquisição dos Sensores

Não houve dificuldade em encontrar a chave ou o *reed switch* no mercado, sendo facilmente obtido em lojas de componentes eletrônicos. O sensor óptico, por ser reutilizado, foi obtido da desmontagem de uma impressora HP©, não sendo tão fácil de encontrar no mercado se comparado com os outros dois.

4.2.3 Acoplamento dos Sensores

Por fim a fixação do sensor ao sistema físico foi um ponto determinante para a definição do sensor. Percebeu-se que utilizando a chave haveria a necessidade de interação física entre o carro e a chave, o que dificultaria a estabilidade da fixação deste sensor. O acoplamento da chave e do sensor óptico à base metálica que suporta o carro também foi um ponto sem sucesso devido à falta de flexibilidade da estrutura que é basicamente de metal e não contém furos para possíveis parafusos e porcas que ajudariam na fixação do conjunto.

Após as análises foi percebido que o *reed switch* seria ideal para a utilização no projeto visto que não seria necessário montá-lo em nenhum tipo de base para depois acoplar na estrutura. Dois sensores do tipo *reed switch* foram fixados no carro utilizando apenas cola quente, um de cada lado do carro. Para ativar os sensores dois ímãs retirados de um disco rígido foram utilizados. Os ímãs foram facilmente posicionados nas laterais da estrutura metálica do trilho fazendo os sensores fecharem seus contatos quando o carro se aproxima das laterais.

Para que os sensores fossem controlados pela placa Arduino foram necessários apenas dois resistores e fios para fazer a ligação do sensor que se encontram no carro até a placa Arduino. Foram utilizados dois resistores de 1k Ohms conectados entre o *GND* e os pinos 6 e 7 do Arduino. Então, conectou-se cada *reed switch* ao resistor de seu respectivo pino.

O esquema elétrico da Figura 4.5 mostra as conexões do *reed switch* a placa Arduino. Os pinos de conexão dos sensores são o 6 e o 7. Ambos são pinos digitais configurados como entrada. Observe que foram necessários resistores de *pull-down* ligando cada pino ao *GND*. A outra extremidade dos sensores é ligada diretamente ao VCC.

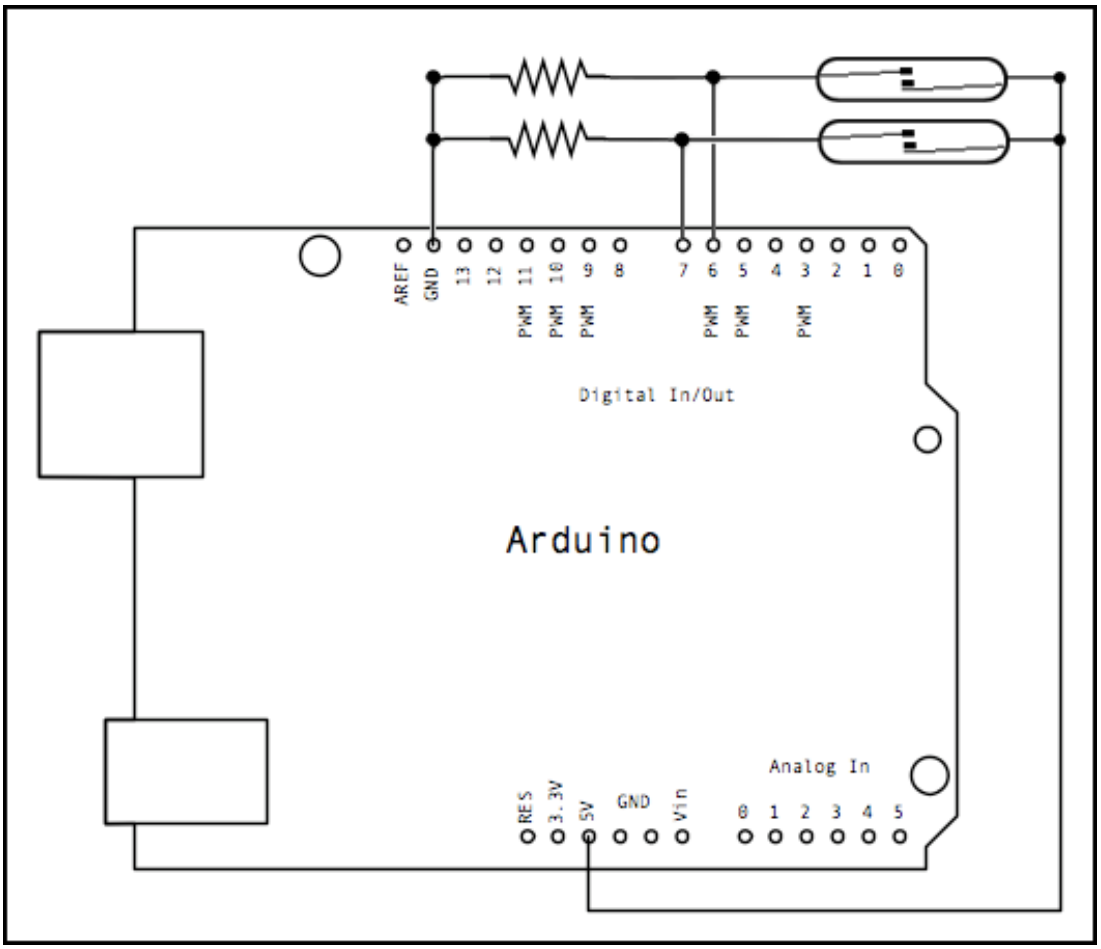


Figura 4.5 Esquema elétrico da conexão dos sensores *reed switch*. Fonte adaptada de http://arduino.cc/en/uploads/Tutorial/button_schem.png.

4.3 Interface de Comunicação RS-232

Para adição de uma comunicação serial, que possa ser utilizada em dispositivos RS-232, foi adicionada no *hardware* do sistema uma interface através de uma placa conversora para o padrão RS-232->TTL. Esta placa permite que o Arduino se comunique com PCs ou outros dispositivos, além de microcontroladores.

A conexão entre os dispositivos é estabelecida através do conector DB9, sendo necessários apenas três pinos de conexão: 2, 3 e 5. O pino 2 representa o RX, o pino 3 representa o TX e o pino 5 o GND. A placa de interface serial utilizada pode ser vista na Figura 4.6. Um cabo serial macho-fêmea foi confeccionado para fazer a conexão com PC. A Figura 4.7 mostra o diagrama do cabo.

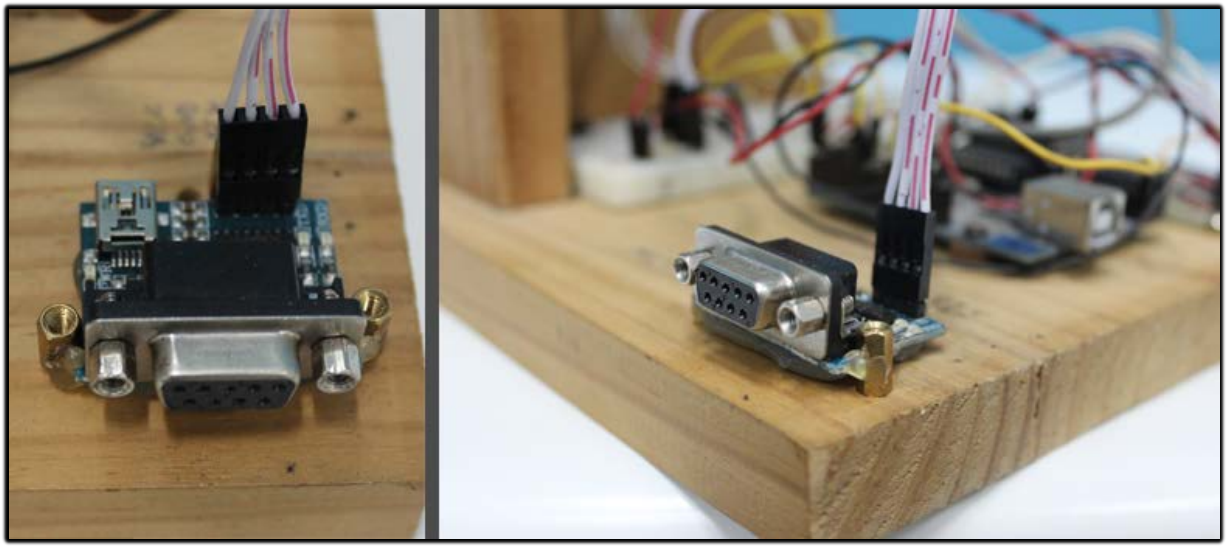


Figura 4.6 Detalhe da placa de interface RS-232.

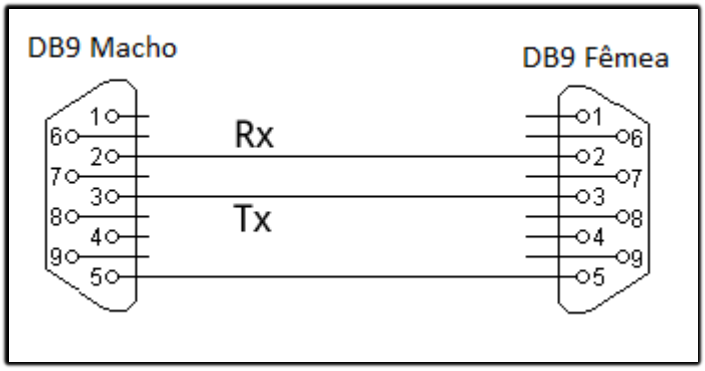


Figura 4.7 Esquema do cabo DB9 macho-fêmea.

4.4 Inicialização do Sistema

Para determinar automaticamente uma posição ótima para o carro foram criadas rotinas de inicialização do *software* embarcado. Estas rotinas estão contempladas nos três estados iniciais: INICIALIZAR, CALIBRAR e CENTRALIZAR.

Quando o sistema é alimentado e o Arduino inicia o processamento o estado passa a ser INICIALIZAR. Neste ponto o *loop* do *software* embarcado verifica se o carro já se encontra na extrema esquerda do sistema físico. Esta verificação é feita através do nível, alto ou baixo, no pino 7 no qual está conectado o sensor esquerdo. Se o carro estiver na extrema esquerda então este já se encontra na posição inicial ideal e passa ao estado CENTRALIZAR, caso contrário o estado passa a ser

CALIBRAR. No estado CALIBRAR o carro é comandado a se deslocar para a esquerda, ainda na mesma rotina, é verificado quando este chega ao fim de curso esquerdo. Quando isto ocorre o estado é alterado novamente, desta vez para o CENTRALIZACAO, que comanda o carro para a direita. O carro para quando o centro do sistema é alcançado, concluindo a inicialização do sistema e então passando para o estado ESPERA.

4.5 Requisição de Dados por Demanda

Para que seja possível controlar o carro de forma flexível ao sistema cliente foi implementado um protocolo de requisição de dados por demanda. Diferente da implementação atual, que transmitia os dados continuamente, esta implementação só envia as informações necessárias para o controle do *Kit* Pêndulo Invertido quando o controlador cliente requisita. Esta requisição é feita através de um comando enviado ao sistema embarcado do *kit.*

O envio de dados sob demanda permite ao sistema cliente definir o intervalo de obtenção das informações uma vez que implementações diferentes para o controle podem necessitar de dados em diferentes intervalos de tempo. Utilizando esta abordagem é possível deixar o canal de comunicação livre para que novos comandos possam ser enviados sem deixar o canal de comunicação congestionado com possíveis redundâncias.

Quando o sistema se encontra no estado ESPERA é possível enviar uma série de códigos de comando, sendo um deles a requisição dos dados. Estes comandos serão abordados da seção seguinte.

4.6 Definição de Códigos de Comandos e Eventos

O controle do *Kit* Pêndulo Invertido é feito através de dados enviados pelo controlador cliente através do canal de comunicação serial. Para flexibilizar este controle foram criados uma série de códigos que possibilitam comandar o carro, alternar entre as formas de recebimento dos dados, reiniciar o sistema, requisitar os dados, obter dados de configuração e ainda redefinir valores de variáveis utilizadas

para o controle. Estes códigos devem ser enviados para o sistema quando este estiver no estado ESPERA. Um código é enviado ao controlador cliente para que este saiba quando o sistema se encontra neste estado.

Os códigos foram criados seguindo o formato proposto inicialmente por Andrade (2012), os quais já utilizavam a tabela ASCII como base para definição dos comandos, assim como os códigos propostos por este trabalho. Dessa forma os caracteres enviados e recebidos pelo controlador são caracteres definidos pela tabela ASCII. O código embarcado é escrito em uma linguagem baseada em C e C++, assim a leitura dos dados, feita pelo método do `Serial.read()` do Arduino, pelo canal serial já é traduzida para caracteres ASCII. A rotina de tratamento dos dados, `void tratarEntrada()`, pode ser vista no Apêndice A.

Os comandos de controle do carro originais propostos por Andrade (2012) foram mantidos, visto que é uma forma simplifica para movimentar o carro em determinada velocidade. Essa lista de comandos para movimentar o carro para esquerda, direita e também parar é apresentada na Tabela 4.1.

Tabela 4.1 Lista de comandos de movimentação do carro.

Descrição	**Código**
Parar	0
Mover para Direita	1 à 255
Mover para Esquerda	-1 à -255

Com a implementação da requisição de dados sob demanda foi necessário criar um comandos de controle. Para facilitar a distinção dos comandos de movimentação do carro, foram utilizados valores acima de 255, iniciando de 300. O controlador cliente pode solicitar os dados ao Arduino através do envio do código de comando 300. É possível também reiniciar o controle, alterando o estado do sistema

para CALIBRAR, através do código de comando 301. A Tabela 4.2 mostra os códigos de comandos de controle.

Tabela 4.2 Comandos de controle

Descrição	**Código**
Obter Dados	300
Reiniciar	301

Uma lista de códigos de configuração foi criada para que o controlador cliente possa definir quais informações deseja receber, obter informações acerca dos valores configurados e ainda redefinir os valores iniciais dos sensores de posição linear e angular. Na Tabela 4.3 pode-se observar estes códigos de configuração. Por padrão o controlador cliente recebe os dados no novo formato, porém se precisar redefinir esta configuração deve ser enviado o código 302 para o sistema. O formato para o recebimento destes dados segue o padrão “H,PL,VL,PA,VA”, onde cada parte separada por vírgula representa uma informação e está descrita na Tabela 4.5. Para que o sistema passe a enviar os dados no formado definido por Andrade (2012) deve ser enviado o código de configuração 303. O padrão para estes dados segue a regra “H,SL,TSL,SA,TSA,EC,EP”, descrita na Tabela 4.6.

Um código de configuração para redefinição dos valores dos sensores linear e angular. A necessidade da redefinição destes dados foi percebida, pois ao entrar no estado de ESPERA, o sistema mesmo estacionário, passava a enviar o valor da posição angular diferente de zero, o que não poderia ocorrer por estar na posição inicial linear e angular. Assim, o comando 304 redefine os valores das variáveis para os valores iniciais.

Para que o controlador cliente possa obter informações sobre os valores dos limites esquerdos e direito utilizados controlar as colisões laterais, partes do disco codificado e partes da fita codificada foi criado o código 305. O padrão dos dados enviados por este comando segue a regra “C,PF,PD,LE,LD”, descrita na Tabela 4.4. É importante observar que, caso a posição do carro esteja entre os valores de LD e

LE a *flag* de fim de curso é desabilitada. A lista completa dos códigos de configuração pode ser vista na Tabela 4.3.

Tabela 4.3 Códigos de Configuração

Descrição	**Código**
Receber Novos Dados	302
Receber Dados Originais	303
Redefinir Valores	304
Obter os Valores Configurados	305

Tabela 4.4 Detalhe dos dados das variáveis de configuração

Parte	**Descrição**
C	Indica o início do conjunto de dados de configuração
PF	Quantidade de partes da fita codificada
PD	Quantidade de partes do disco codificado
LE	Limite esquerdo
LD	Limite direito

Tabela 4.5 Descrição dos Novos Dados

Parte	**Descrição**
H	Indica o início do conjunto de dados
PL	Posição linear do carro
VL	Velocidade linear do carro
PA	Posição angular (-180 a 180)
VA	Velocidade angular da haste

Tabela 4.6 Descrição dos Dados Originais

Parte	Descrição
H	Indica o início do conjunto de dados
SL	Sensor da posição linear do carro
TSL	Instante de tempo da posição linear do carro (ms)
SA	Valor do sensor angular
TSA	Instante de tempo da posição angular do carro (ms)
EC	Estado do carro. Indica de está se movendo ou não (valores 0 ou 1)
EP	Estado do pêndulo. Indica de está se movendo ou não (valores 0 ou 1)

Além dos códigos enviados para o sistema foi necessário criar códigos para eventos e exceções. Quando o sistema finaliza a inicialização e fica no estado de espera, é enviado para o controlador cliente o comando 400, informando que o evento da chegada ao estado de espera ocorreu e que o agora pode receber os códigos de comandos citados anteriormente. Além deste evento também é

informado ao controlador cliente, através do código 401, da parada do carro por fim de curso, que ocorre quando o carro chega a uma das laterais. Por fim, quando um código inválido é enviado o controle embarcado responde esta exceção com o código 402. A Tabela 4.7 apresenta os códigos para eventos e exceções.

Tabela 4.7 Eventos e Exceções do Sistema

Evento / Exceção	**Código**
Estado de Espera	400
Parada por fim de curso	401
Comando Inválido	402

A última lista de códigos definida trata da redefinição dos valores das variáveis que armazenam a quantidade de partes da fita codificada, do disco codificado e os valores dos limites esquerdo e direito. Estes comandos podem ser utilizados quando é preciso mudar estes valores devido à troca das fitas ou se as fitas sofrerem algum desgaste. Para definir cada uma destas variáveis o código é composto com duas partes sendo o valor desejado seguido de um sufixo representado por uma letra que identifica cada variável. Na Tabela 4.8 pode ser visto a lista destes códigos.

Tabela 4.8 Comandos para definição de variáveis.

Descrição da variável	**Código (**<xx> é a *string* de caracteres números ASCII do novo valor da variável)
Partes Disco Codificado	<xx>d
Partes Fita Codificada	<xx>s
Limite Esquerdo	<xx>l
Limite Direito	<xx>r

4.7 Resumo do Capítulo

Neste capítulo foram apresentadas como foram feitas as implementações e adaptações feitas no *Kit* Pêndulo invertido proposto por Andrade (2012). Foi mostrado que um diagrama de estados foi criado para auxiliar nas adaptações e modificações no código do sistema embarcado. Também se mostrou que, dos propostos, o sensor magnético *reed switch* foi a melhor opção para adaptação ao sistema e que uma interface serial unificada foi definida adicionando uma placa de interface serial RS-232. Foi feita uma descrição de como a rotina de inicialização funciona e ainda descreveu-se a requisição de dados por demanda. Por fim, a descrição dos códigos de comando necessários para melhorar a interação do sistema embarcado com o controlador cliente foi apresentada.

Capítulo 5

Resultados

Este capítulo apresenta os resultados alcançados a partir das modificações e implementações feitas no *Kit* Pêndulo Invertido proposto por Andrade (2012). As melhorias foram feitas tanto no *hardware*, adicionando recursos que possibilitassem melhorar o sensoriamento e comunicação, quanto no *software*, criando rotinas para manipular os dados dos novos sensores e ainda uma lista de códigos de comando para ampliar a comunicação entre o sistema embarcado e o controlador cliente. O código do sistema embarcado no Arduino pode ser visto no Apêndice A.

5.1 Modelagem de Estados do Sistema

A modelagem dos estados do sistema teve um papel de grande importância no decorrer deste trabalho. As modificações na estrutura funcional do *software* embarcado foram guiadas através do diagrama desenvolvido. O resultado final servirá como documentação para guiar trabalhos futuros. O diagrama final foi apresentado na Figura 4.1.

5.2 Sensores de Fim de Curso

Os sensores de fim de curso proporcionaram a implementação da rotina de inicialização modelada pelos estados de inicialização do diagrama de estados. Os sensores foram adicionados com sucesso à estrutura do carro sendo necessários apenas três fios para fazer a ligação do conjunto de sensores ao Arduino. A disposição final dos sensores no carro pode ser observada na Figura 5.1. Pode-se observar que os sensores foram facilmente adicionados ao carro e que estes ficam protegidos pela estrutura do carro evitando que sofram danos.

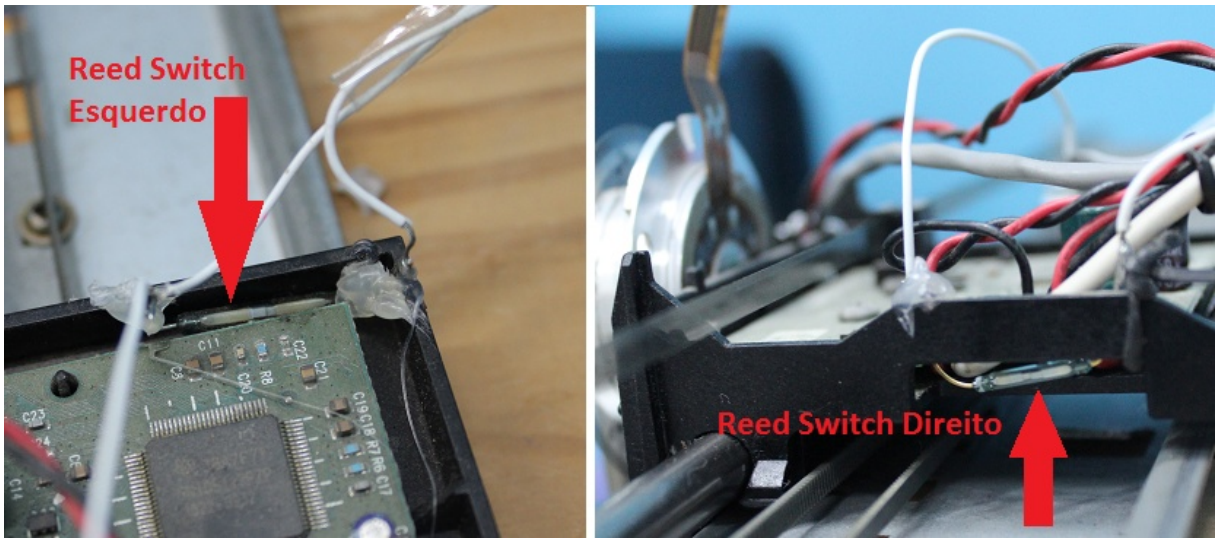


Figura 5.1 Posicionamento dos sensores de fim de curso.

Na Figura 5.2 podem ser vistos os ímãs que são responsáveis por disparar os sensores de fim de curso. Os ímãs foram facilmente acoplados e posicionados à estrutura metálica não havendo necessidade de parafusos ou quaisquer outros artifícios para fixação.

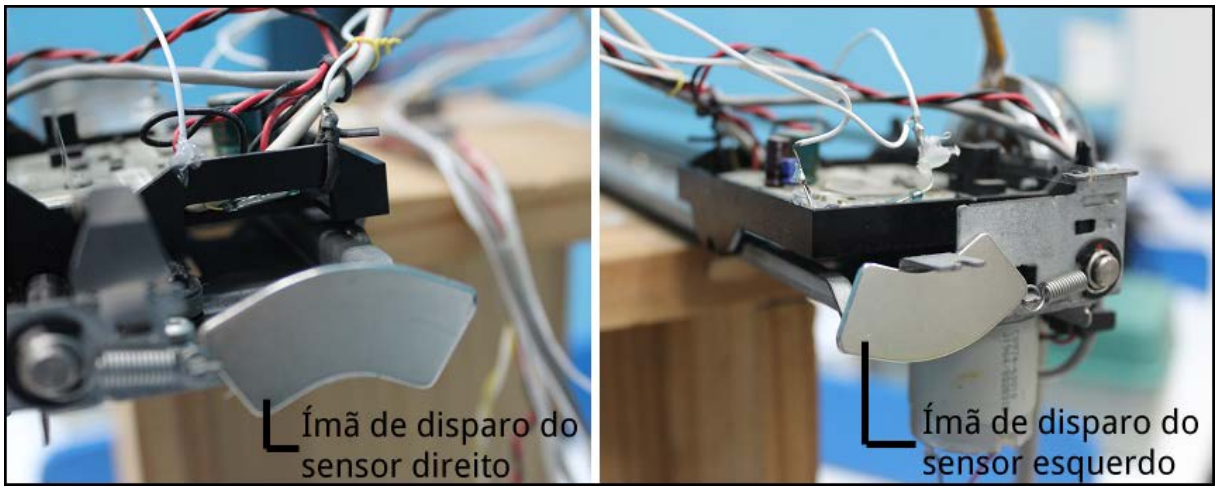


Figura 5.2 Posicionamento dos ímãs para disparo dos sensores de fim de curso.

5.3 Interface de Comunicação RS-232

Os resultados da inclusão da interface RS-232 foram obtidos por meio de testes efetuados a partir de um *software* desenvolvido para este propósito, denominado

neste trabalho de 'Controle Pendular', que tem seu código fonte apresentado no Apêndice B. Com esta ferramenta foi possível validar a comunicação do *Kit* Pêndulo Invertido com um PC dotado de porta serial RS-232 e executar os principais comandos definidos no Capítulo 4. A Figura 5.3 mostra os códigos enviados para o *kit* a partir do Controle Pendular em um PC.

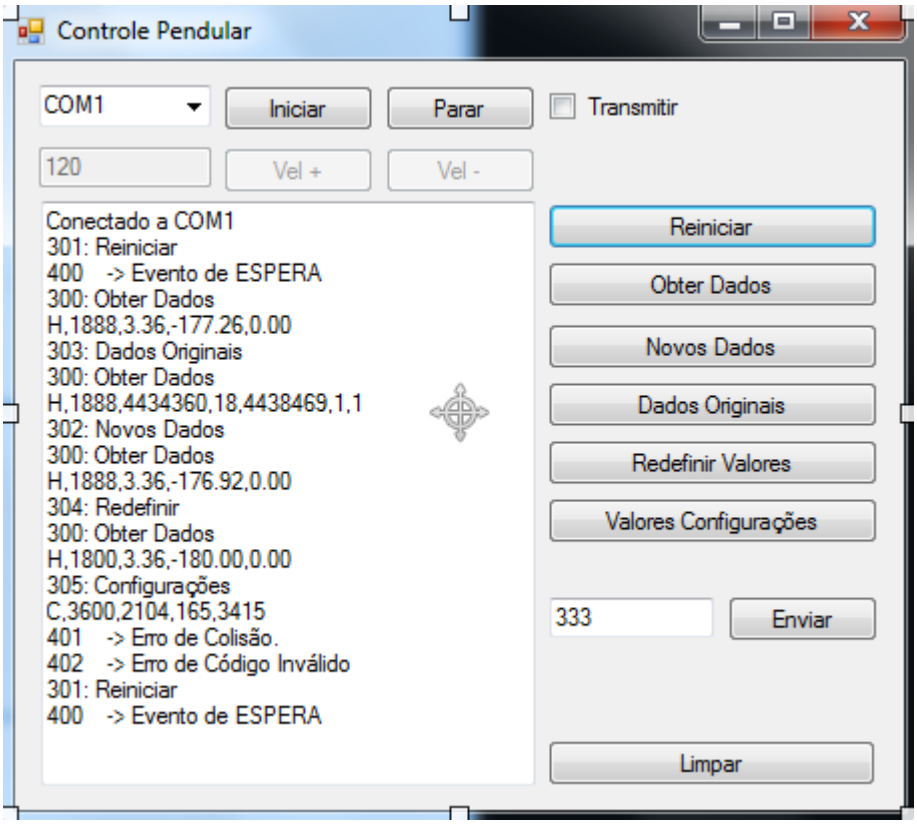


Figura 5.3 Testes da comunicação serial utilizando o *software* Controle Pendular.

No teste executado o sistema respondeu a todas as entradas dos comandos efetuados as devidas ações e enviado os dados requisitados. Pôde-se observar obtenção dos dados a partir de uma requisição, a resposta a colisões com o envio do código de exceção e a mudança para o estado ESPERA.

5.4 Inicialização do Sistema

Como proposto no capítulo anterior a implementação da etapa de inicialização do sistema proporcionou a determinação do ponto de partida do carro na extremidade

esquerda do sistema físico e em seguida sua centralização. O resultado esperado foi alcançado e pode ser percebido através da Figura 5.4, que apresenta o monitor serial da IDE do Arduino. No teste, o sistema foi inicializado chegando à posição central, o que pode ser observado pelo recebimento do código 400. Em seguida o carro foi posicionado manualmente até o limite esquerdo, disparando um evento de fim de curso, o qual é notificado pelo código 401. Em seguida o código de obtenção de dados foi enviado através do monitor serial seguidas vezes enquanto se variava manualmente a posição do carro. Cada instante é mostrado na Figura 5.5. Perceba que a legenda (a) da Figura 5.4 está relacionada com o instante (a) da Figura 5.5, a legenda (b) está relacionada com o instante (b) e assim por diante.

Assim é possível perceber que após a inicialização do sistema os valores da posição do carro são conhecidos não sendo preciso o posicionamento manual a fim de encontrar a posição ótima.

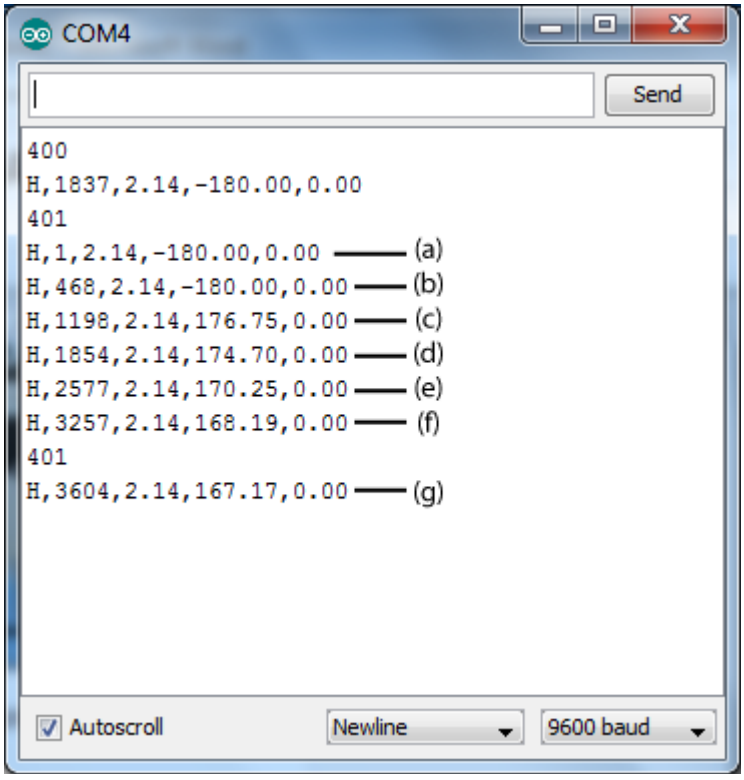


Figura 5.4 Valores obtidos a cada instante da Figura 5.5.

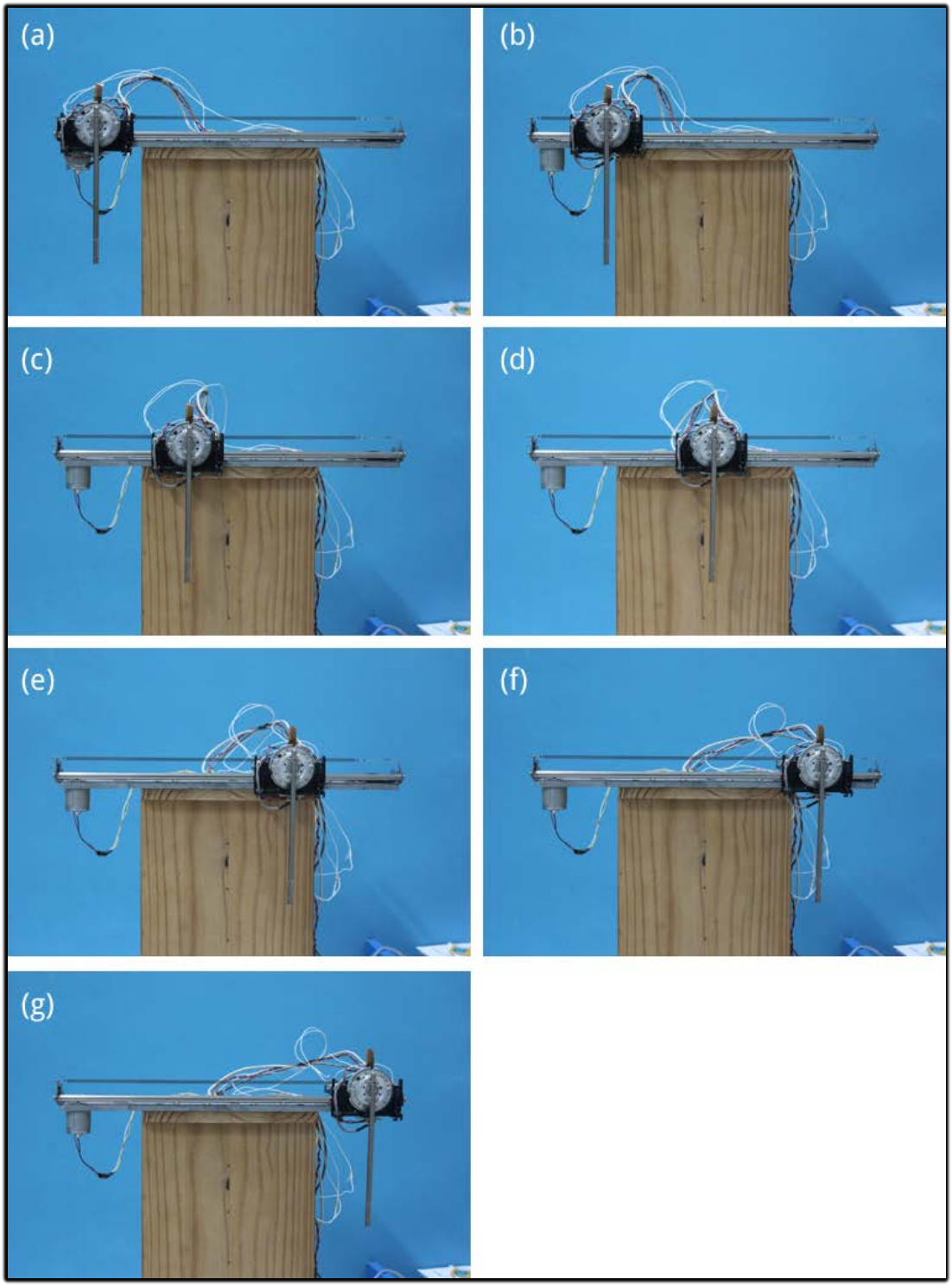


Figura 5.5 Instantes do posicionamento do carro. (a) carro na posição 1; (b) carro na posição 468; (c) carro na posição 1198; (d) carro na posição 1854; (e) carro na posição 2577; (f) carro na posição 3257; (g) carro na posição 3604.

5.5 Requisição de Dados por Demanda

A partir da implementação da requisição de dados por demanda é possível observar que o canal serial não fica transmitindo informações. A Figura 5.6 mostra os passos para efetuar a requisição por demanda através do monitor serial da IDE do Arduino. Na Figura 5.6 (a) o sistema encontra-se o estado de espera, sem que haja transmissão através do canal serial. Em (b) o código de requisição é escrito no monitor serial e em seguida enviado. Por fim, em (c), os dados são transmitidos para o requerente e então escritos no monitor serial.

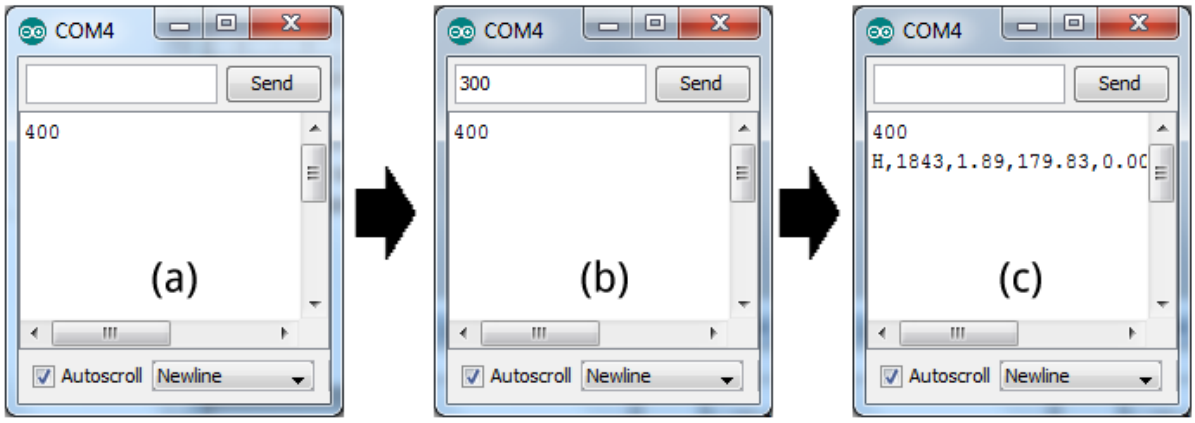


Figura 5.6 Passos da requisição por demanda. (a) sistema em estado de espera; (b) entrada do código de requisição; (c) os dados são enviados pela serial.

5.6 Códigos de Comando e Eventos

Da mesma forma que na Seção 5.3, os códigos de comando foram testados utilizando o *software* Controle Pendular. Foi observado que todos os comandos foram tratados da forma correta e suas respectivas respostas contemplaram o padrão definido na Seção 4.6. Os testes dos códigos de comando podem ser vistos no vídeo “Análise do *Kit* Pêndulo Invertido - Códigos de Comando via RS-232”, disponibilizado por Silva (2012).

5.7 Resumo do Capítulo

Neste capítulo foram mostrados os resultados das implementações propostas. Detalhes da implantação dos sensores foram mostrados através das Figuras 5.1 e 5.2. Testes da interface de comunicação RS-232 foram apresentados e o detalhamento da inicialização do sistema foi apresentado utilizando dados reais do posicionamento do carro. A requisição de dados por demanda e os códigos de comando foram discutidos e exemplificados.

Capítulo 6

Conclusão

Com a implementação das melhorias propostas neste trabalho, pôde-se observar que o *Kit* Pêndulo Invertido tornou-se um recurso didático mais robusto. Pode-se enumerar as seguintes melhorias a partir da proposta e das implementações feitas neste trabalho:

(i) O sistema se tornou resistente, pois como é possível saber localmente se o carro atingiu o limite de deslocamento o motor pode ser desligado evitando sua sobrecarga ou o rompimento da correia;

(ii) Os dados são enviados pela requisição, evitando que o canal serial fique congestionado por dados desnecessários e possibilitando que o controlador cliente controle o tempo de obtenção destes;

(iii) Códigos de comando foram definidos proporcionando ao cliente, por exemplo, saber o momento de uma colisão para que a devida providência possa ser tomada;

(iv) O tratamento de códigos inválidos foi criado para que o controlador cliente seja informado desta exceção;

(v) É possível reiniciar o sistema através de um código de comando específico, o que permite que o cliente possa interferir apenas via *software* para tratar uma exceção e recomeçar o controle;

(vi) Através da interface de comunicação RS-232 implantada é possível fazer o controle através da porta serial de um PC ou de outro dispositivo compatível com este padrão dando maior flexibilidade ao *kit*;

(vii) A criação de um diagrama de estados pode guiar a implementação do controle ou ainda outras melhorias que sejam necessárias.

A partir das melhorias efetuadas neste trabalho é possível utilizar o *Kit* Pêndulo invertido como recurso didático em disciplinas do curso de Engenharia de

Computação, no entanto é preciso observar os problemas enumerados na Seção 6.1 e analisar se algum dos itens pode comprometer o controle.

Por fim, é observado com este trabalho que é possível viabilizar o uso de recursos de REEE, nas universidades de engenharia, para equipar laboratórios através da construção de *kits*, ampliando, desta forma, o contato dos alunos com ferramentas didáticas mais próximas do mercado, visto que o *kit* foi montado com recursos que já estão sendo utilizados por produtos comercializados.

6.1 Trabalhos Futuros

No decorrer das análises e implementações feitas neste trabalho, foram observados os seguintes pontos a serem aprimorados ou corrigidos em trabalhos futuros:

(i) Um forte atrito do carro com o eixo pelo qual este desliza, necessitando de uma análise neste ponto a fim de tornar o controle possível com valores de atuação menores no carro. Produtos para lubrificação de impressoras podem ajudar a solucionar o problema. A verificação do eixo do suporta o carro também deve ser feita a fim de averiguar se o mesmo está empenado, visto que o problema é mais perceptível na metade esquerda;

(ii) Há a necessidade da fixação da estrutura de madeira do *Kit* Pêndulo Invertido em uma base a fim de manter o conjunto imóvel, pois devido à inércia o conjunto tende a se deslocar verticalmente no momento em que o carro inverte o sentido do movimento, ou para, quando está se movendo com valores muito altos;

(iii) Foi observado também que há uma inconsistência do valor da variável que representa o disco codificado. Seus valores variam de forma que, a partir do estado estacionário, ao se executar um giro completo e voltar ao mesmo ponto, o valor desta variável mantém um acúmulo, o que pode dificultar o controle de forma ideal. A causa provável deste problema é uma leve inclinação que existe do disco codificado em relação ao seu sensor. O disco codificado passa pelo sensor inclinado e gera interrupções através do sensor nos dois sentidos do giro. A correção da inclinação do disco codificado em relação ao sensor deve resolver o problema.

Bibliografia

AHN, J.; JUNG, S. **Swing-up Fuzzy Control of an Inverted Pendulum System for Control Education with an Experimental Kit**, International Conference on Control, Automation and Systems, COEX, Seoul, Korea, 2008.

ANDRADE, M. A. O. **Mitigação Do Impacto Ambiental Causado Pelo REEE : Confecção De Kits De Treinamento a Partir De Equipamentos Descartados Como Estratégia Para Tratamento Do Lixo Digital,** 2012.

BRUNAUER, L.; KREIL, W. **Inverted Pendulum**, Scientific Literature Digital Library and Search, 2007.

BUTIKOV, E. I. **An Improved Criterion for Kapitza's Pendulum Stability**, Journal of Physics A: Mathematical and Theoretical, 44, 2011.

FELDER, R. M., SILVERMAN, L. K. **Learning and Teaching Styles**, Journal of Engineering Education, 1988. v. 78, pp. 674–681.

FTDI™ Chip, **FT232R USB UART I.C.** Disponível em: < http://www.ftdichip.com >. Acesso em: 15 nov. 2012.

JUNG, S.,; AHN, J. **Remote Control of an Inverted Pendulum System for Intelligent Control Education**, International Institute of Informatics and Cybernetics, 2011, 9, pp. 49–54.

LEE, G. H.; JUNG, S. **Control of Inverted Pendulum System Using a Neuro-fuzzy Controller for Intelligent Control Education**', 2008 IEEE International Conference on Mechatronics and Automation, 2008.

LIMA, J. L., COSTA, P. G., MOREIRA, A. P. **Inverted Pendulum Virtual Control Laboratory,** 7th Portuguese Conference on Automatic Control, Instituto Superior Técnico, Lisboa, Portugal, 2006.

MOXA , **NA-4020/4021, NA-4010 and NA-4020/4021 Series**. Disponível em: <http://www.moxa.com>. Acesso em: 14 nov. 2012.

POORHOSSEIN, A., VAHIDIAN-KAMYAD, A., **Design and Implementation of Sugeno Controller for Inverted Pendulum on a Cart System**, IEEE 8th International Symposium on Intelligent Systems and Informatics, 2010, 641–646.

SARIK, J. e KYMISSIS, I. **Lab Kits Using the Arduino Prototyping Platform**, 40th SEE/IEEE Frontiers in Education Conference, 978-1-4244-6262-9/10/2010 IEEE.

SILVA, L.F.; LEITE , J.C.S.P.; BREITMAN, K.K.. **Ensino de Engenharia de Software: Relato de Experiências,** XII WEI - Workshop de Educação em Informática (XXIV SBC), Salvador-BA, 2004.

SILVA, R. T., **Análise do Kit Pêndulo Invertido - Colisões**. Disponível em: <http://www.youtube.com/watch?v=OW7SZzdnhdg>. Acesso em: 14 nov. 2012.

SILVA, R. T., **Análise do Kit Pêndulo Invertido - Códigos de Comando via RS-232**. Disponível em: <http://www.youtube.com/watch?v=qHdgr_f1AEM>. Acesso em: 19 nov. 2012.

STARTECH, **NETRS2321E 1 Port RS-232/422/485 Serial over IP Ethernet Device Server**. Disponível em: < http://intrl.startech.com/ >. Acesso em: 14 nov. 2012.

VASILKOV, V. ; CHUBINSKY, A.; YAKIMOVA, K., **The Stephenson - Kapitsa Pendulum: Area of the Attraction of the Upper Positions of the Balance**, 2007, 61–66.

WIKIPEDIA, **KAPITZA'S PENDULUM**. Disponível em: <http://en.wikipedia.org/wiki/Kapitza's_pendulum>. Acesso em: 20 set. 2012.

Apêndice A

```
/*
#############################################################################
#########################
19/11/2012 - UPE - POLI
Codigo de controle do Kit Pendulo Invertido
Aluno: Ricardo Teixeira da Silva

O dispositovo recebe valores descritos abaixo nas Tabelas de Entrada e
Saída de Dados,
e retorna dados nos padroes:
  Dados Originais: posicao, tempo ; angulo, tempo;  estado carro, estado
pendulo, line feed.
  Novos Dados: posicao,velocidade posicao, angulo (em graus), velocidade
angular

versão original por Mércio Andrade (2012)

#############################################################################
#########################

## Tabelas de Entrada e Saída de Dados ##

Se estiver no estado de espera o sistema
pode receber os seguintes comando.

-----------------------------------------
| Comandos de Movimento |    Recebe    |
-------------------------------------
| Parar                   |   0         |
| Mover para Direita    |    1 à 255   |
| Mover para Esquerda   |    -1 à -255 |
-----------------------------------------

-----------------------------------------
| Comandos de Controle  |    Recebe    |
-----------------------------------------
| Obter Dados             |   300       |
| Reiniciar             |    301       |
-----------------------------------------

-----------------------------------------
| Comandos de Configurações | Recebe    |
-----------------------------------------
| Receber Novos Dados       | 302       |
| Receber Dados Originais   | 303       |
| Redefine Valores          | 304       |
| Valores Config            | 305       |
-----------------------------------------

-----------------------------------------
| Definição das Variáveis   | Recebe    |
-----------------------------------------
| Pates Disco Codificado    | xxxxd     | disc
| Pates Fita Codificada     | xxxxs     | strip
| Limite Esquerdo           | xxxxl     | left
| Limite Direito            | xxxxr     | right
```

```
-----------------------------------------

* Dados enviados para o controle
-----------------------------------------
| Comandos de Controle  |    Envia     |
-----------------------------------------
| Estado de Espera      |    400       |
| Parada: fim de curso  |    401       |
| Comando Inválido      |    402       |
-----------------------------------------

*/

#define enPin 9 // pino enable CI L293
#define drive_1 10 // pino conectado ao IN1 do CI L293
#define drive_2 11  // pino conectado ao IN2 de CI L293

#define encoder0PinoA 2
#define encoder0PinoB 4
#define encoder1PinoA 3
#define encoder1PinoB 5

volatile int estado_carro = 0; // (1) descolando; (0) parado
volatile int estado_pendulo = 0; // (1) oscilado; (0) parado

char entrada; // armazena o valor recebido pela serial
int val = 0;           // auxiliar entrada
int direcao_carro = -1; // (0) esquerda (1) direita
long velocidade = 0;

// variaveis do encoder int 0 (deslocamento)

volatile long encoder_0_posicao = 0;
volatile long tempo_posicao;
long tempo_inicio_pos=0;
long tempo_posicao_capturado;

// variaveis do encoder int 1 (angulo)

volatile long encoder_1_angulo = 0;
volatile long tempo_angulo = millis ();
long tempo_inicio_ang=0;

volatile long delta_tempo_angulo = tempo_angulo;

// sensor de fim de curso esquerdo
const int sensorFimCursoEsqPin = 7;
// sensor de fim de curso direito
const int sensorFimCursoDirPin = 6;

int inicializado = 0;

int fimCursoEsq = 0;
int fimCursoDir = 0;
int fimCursoErro = 0;
long timeEnviaErroFimCurso;

const int ledFimcurso = 13;

////////////////////////////
```

```
// Constantes de inicialização

// posicao central
int POSICAO_FINAL = 3600;
// posicao central
int POSICAO_CENTRAL = POSICAO_FINAL / 2;
// posicao central
const int VELOCIDADE_CALIBRACAO = 125;

// Limites para o tratamento das colisões
int LIMITE_ESQ = 165;
int LIMITE_DIR = 3415;

// estados do sistema
const int INICIALIZANDO = 0;
const int CALIBRANDO = 1;
const int CENTRALIZANDO = 2;
const int ESPERA = 3;

////////////////////////////

int estadoAtual = INICIALIZANDO;
int fimCurso;

const float pi = 3.14159265358979323846264338327 95;

/// Disco codificado
int partesDisco = 2104;
float parteGrau = 360 / partesDisco;

int tiCentralizar;
int tfCentralizar;
float velocidadeCentralizar;
float velocidadeDeslocamento;
int tiDesloca;
int posicaoAnterior = 0;

// config do padrão de dados a receber (0 = novo, 1  = antigo);
int padraoInfo = 0;

void setup() {
  // Configuracao da velocidade de comunicacao serial
  Serial.begin(9600);

  // Pinos dos sensores de fim de curso
  pinMode(sensorFimCursoEsqPin, INPUT);
  pinMode(sensorFimCursoDirPin, INPUT);

  // Pinos acionamento motor DC
  pinMode(drive_1, OUTPUT); // in1
  pinMode(drive_2, OUTPUT); // in2

  // Pinos Interrupcao 0
  pinMode(encoder0PinoA, INPUT);
  pinMode(encoder0PinoB, INPUT);
  attachInterrupt(0, doEncoder0, CHANGE ); // sensor conectado a
interrupcao 0

  // Pinos iterrupcao 1
  pinMode(encoder1PinoA, INPUT);
```

```
  pinMode(encoder1PinoB, INPUT);
  attachInterrupt(1, doEncoder1, CHANGE );  // sensor conestado a
interrupcao 1

}

void loop() {

  switch (estadoAtual) {

    case INICIALIZANDO:
      fimCurso = digitalRead(sensorFimCursoEsqPin);
       if (fimCurso == HIGH) {
         estadoAtual = CENTRALIZANDO;
       } else {
         estadoAtual = CALIBRANDO;
       }
      break;

    case CALIBRANDO:
      estadoCalibracao();
      break;

    case CENTRALIZANDO:
      estadoCentralizacao();
      break;

    case ESPERA:
       estadoEspera ();
      break;
  }
}

void redefineValoresCentral () {
    encoder_1_angulo = 0;
    encoder_0_posicao = POSICAO_FINAL / 2;
}

// testa o sensor de fim de curso. quando a pino sensorFimCursoEsqPin
// for para nível alto (fimCurso == HIGH).
// quando em nível alto o carro é parado, o encoder_0_posicao vai para zero
// e o estadoAtual vai para CENTRALIZANDO
void estadoCalibracao () {
  fimCurso = digitalRead(sensorFimCursoEsqPin);
  if (fimCurso == HIGH) {
    digitalWrite(ledFimcurso, HIGH);
    desligarCarro ();
    encoder_0_posicao = 0;

    tiCentralizar = millis();

    estadoAtual = CENTRALIZANDO;
  } else {
    digitalWrite(ledFimcurso, LOW);
    moverCarro (0, VELOCIDADE_CALIBRACAO);
  }
}

// verifica a posição atual e para quando chegar na POSICAO_CENTRAL
// alterando o estadoAtual para ESPERA. O teste é com >= pois
```

```
// devido ao atrito o carro pode passar um pouco fincando aproximado.
void estadoCentralizacao () {

  if (encoder_0_posicao >= POSICAO_CENTRAL) {
    desligarCarro ();
    estadoAtual = ESPERA;

    tfCentralizar = millis();

    velocidadeCentralizar = (float)POSICAO_CENTRAL / (float)(tfCentralizar
- tiCentralizar);

    // envia o código que representa o estado de espera
    Serial.print (400);
    Serial.write ('\n');

    redefineValoresCentral ();

  } else {
    moverCarro (1, VELOCIDADE_CALIBRACAO);
  }
}

// retorna true se qualquer dos sensores de fim de curso
// for acionado caso contrario retorna false
boolean tratarFimCurso () {
  fimCursoEsq = digitalRead(sensorFimCursoEsqPin);
  fimCursoDir = digitalRead(sensorFimCursoDirPin);

  return fimCursoEsq || fimCursoDir;
}

// Recebendo dados da serial
void estadoEspera () {

  if (encoder_0_posicao > LIMITE_ESQ && encoder_0_posicao < LIMITE_DIR) {
    fimCursoErro = 0;
  }

  if (tratarFimCurso () && fimCursoErro == 0) {

    desligarCarro ();

    // Exceção Parada por fim de curso
    Serial.print (401);
    Serial.write ('\n');

    fimCursoErro = 1;

  }

    // checa se existem dados a serem recebidos
    if (Serial.available() > 0) {
      // Lendo o dado recente
      entrada = Serial.read();
      tratarEntrada ();
    }
}

void desligarCarro () {
```

```
  analogWrite( enPin, 0 );
}

void moverCarro ( int dir, long vel ) {
      // Determinanndo o sentido de giro do motor DC

      if ( velocidade > 255 || velocidade < 0) {
            velocidade = 0;
      }

      direcao_carro = dir;
      if(direcao_carro == 0) {
            digitalWrite(drive_1, HIGH);
            digitalWrite(drive_2, LOW);
            //direcao_carro = 1;
      } else {
            digitalWrite(drive_1, LOW);
            digitalWrite(drive_2, HIGH);
      }

      analogWrite( enPin, vel );
}

// Valores brutos dos sensores e timestamp
void enviarStatusSensores () {


//H,encoder_0_posicao,tempo_posicao,encoder_1_angulo,tempo_angulo,estado_ca
rro,estado_pendulo

    Serial.print ("H"); // 'H' indica inicio da mensagem
    Serial.print (","); // Separadador
    Serial.print(encoder_0_posicao); // Deslocamento horizontal
    Serial.print (",");
    Serial.print(tempo_posicao); // Tempo associada à posição
    Serial.print(",");
    Serial.print(encoder_1_angulo); //Deslocamento angular
    Serial.print (",");
    Serial.print(tempo_angulo); // Tempo associado ao angulo deslocado
    Serial.print (",");
    Serial.print (estado_carro); // (0) carro parado, (1) carro em
movimento
    Serial.print (",");
    Serial.print (estado_pendulo); // (0) pêndulo Parado, (1) Pêndulo em
movimento
    Serial.write("\n"); //insere \n no final da string enviada

    // armazenando estaddos anteriores
    //temp1 = velocidade; // armazena entrada anterior do carro
    estado_pendulo = 0; // seta o estado do pendulo para parado
    estado_carro = 0;  // estado do carro setado para parado
}

void enviarStatusValores () {

  int sinal;
  if (encoder_1_angulo < 0) {
    sinal = -1;
  } else {
    sinal = 1;
```

```
  }

  // DONE deixar o angulo entre -180 e 180
  float r = ((float)encoder_1_angulo / (float)partesDisco);
  float valorAng = (r * 360) - 180; // (r * (180 - (-180))) + (-180)

  float parteGrauRad = pi / 180 * parteGrau; // grau pra rad
  float velAngular = parteGrau / delta_tempo_angulo; // grau / (millis /
1000 = seg)

  // H,encoder_0_posicao,velocidadeCentralizar,valorAng,velAngular

  Serial.print ("H"); // 'H' indica inicio da mensagem
  Serial.print (","); // Separador
  Serial.print(encoder_0_posicao); // Deslocamento horizontal
H,3548,26569,0,27387,1,1
  Serial.print (",");
  Serial.print(velocidadeDeslocamento); // Tempo associada a  posição
  Serial.print (",");
  Serial.print(valorAng); // Deslocamento angular
  Serial.print (",");
  Serial.print(velAngular); // Tempo associado ao angulo deslocado

  Serial.write("\n"); //insere "\n" no final da string enviada

}

void enviarStatus () {

      if (padraoInfo == 1) {
            enviarStatusSensores ();
      } else {
            enviarStatusValores ();
      }

}

void enviarConfig () {
        Serial.print ("C");
        Serial.print (",");
        Serial.print(POSICAO_FINAL);
        Serial.print (",");
        Serial.print(partesDisco);
        Serial.print (",");
        Serial.print(LIMITE_ESQ);
        Serial.print (",");
        Serial.print(LIMITE_DIR);
        Serial.write("\n");
}

void  tratarEntrada () {


      // checa se é ASCII entre 0 e 9
      if(entrada >= '0' && entrada <= '9') {

            converterParaASCII (entrada);

      } else if(entrada =='d') {
```

```
            partesDisco = val;
                val = 0;

      } else if(entrada =='s') {

            POSICAO_FINAL = val;
                POSICAO_CENTRAL = POSICAO_FINAL / 2;
                val = 0;

      } else if(entrada =='l') {

            LIMITE_ESQ = val;
                val = 0;

      } else if(entrada =='r') {

            LIMITE_DIR = val;
                val = 0;

      } else if(entrada =='+') {

            dir = 1; //direcao_carro = 1;

      } else if(entrada =='-') {

            dir = 0; //direcao_carro = 0;

      } else if (entrada == 10) { // checa se recebeu o caracter newline
(\n) (ASCII = 10)

                velocidade = 0;

                if (val != 0) {

                      if (dir == 0) {
                            direcao_carro = 0;
                      } else {
                            direcao_carro = 1;
                      }
                }

                dir = -1;

            // se entrada = \n então novo comando foi lido
            if (val >= 0 && val <= 255) {

                  velocidade = val;

                   if (fimCursoEsq == 1 && direcao_carro == 1) {
                            moverCarro(direcao_carro, velocidade);
                        } else if (fimCursoDir == 1 && direcao_carro == 0) {
                            moverCarro(direcao_carro, velocidade);
                        } else if (fimCursoEsq == 0 && fimCursoDir == 0) {
                            moverCarro(direcao_carro, velocidade);
                        }

                        velocidadeDeslocamento = (float)(encoder_0_posicao -
posicaoAnterior) / (float)(millis() - tiDesloca);

                        posicaoAnterior = encoder_0_posicao;
```

```
                        tiDesloca = millis ();

            } else {

                  tratarComando (val);

            }

            val = 0; // zera val, para ser aplicada a uma nova sequência de
entrada

    }

    analogWrite(enPin, velocidade );
}

void tratarComando (int comando) {

      if (comando == 300) {
            enviarStatus (); // Obter Dados
      } else if (comando == 301) {
                inicializado = 0;
            estadoAtual = CALIBRANDO; // Reiniciar
      } else if (comando == 302) {
            padraoInfo = 0;
      } else if (comando == 303) {
            padraoInfo = 1;
      } else if (comando == 304) {
            redefineValoresCentral ();
      } else if (comando == 305) {
            enviarConfig ();
      } else {
            enviarErro ();
      }
}

void enviarErro () {

    // Exceção Comando Inválido
    Serial.print (402);
    Serial.write ('\n');
}

void converterParaASCII (char valorLido) {
      val = (val * 10) + (valorLido - '0'); // converte ASCII para valor
numérico
}

// Rotina ativada pela interrupcao 0 (deslocamento do carro)
void doEncoder0() {

    /* se A==B, entao  deslocamneto eh  para a direita
     * se A!=B, entao o deslocamento e para a esquerda.
     */
    if (digitalRead(encoder0PinoA) == digitalRead(encoder0PinoB)) {
      tempo_posicao = millis();
      encoder_0_posicao++;
    } else {
      tempo_posicao = millis();
      encoder_0_posicao--;
```

```
    }
    estado_carro=1;

}

//Rotina ativada pela  interrupcao 1 (variacao do angulo do pendulo)
void doEncoder1() {

    long agora = millis();
    delta_tempo_angulo = agora - tempo_angulo;
    tempo_angulo = agora;
    estado_pendulo = 1;

    if (digitalRead(encoder1PinoA) == digitalRead(encoder1PinoB)) {
      encoder_1_angulo++;
      if (encoder_1_angulo >= partesDisco) {
         encoder_1_angulo = 0;
      }
     } else {
      encoder_1_angulo--;
      if (encoder_1_angulo < 0) {
         encoder_1_angulo = partesDisco;
      }
    }

}
```

Apêndice B

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace Pendular
{
    public partial class Form1 : Form
    {
        private SerialPort serialPort1;
        private String RxDataString;

        private string[] parts;
        private decimal angularAnterior;

        public Form1()
        {
            InitializeComponent();

            this.angularAnterior = 0;

            this.serialPort1 = new SerialPort();
            this.serialPort1.DataReceived += this.serialPort1_DataReceived;

            this.log.Multiline = true;
            this.log.WordWrap = true;

            this.portas.DataSource = SerialPort.GetPortNames();

            this.velMenos.Enabled = false;
            this.velMais.Enabled = false;
            this.textVel.Enabled = false;
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void initUpdate()
        {
            RxDataString = "Conectado a " +
this.portas.SelectedValue.ToString();
            this.Invoke(new EventHandler(DisplayText));
        }

        private void tratarDados(string dados)
        {
            this.parts = dados.Split(',');
        }
```

```
        private void enviarDados(string value)
        {
            if (!this.serialPort1.IsOpen) return;
            this.serialPort1.WriteLine(value);
        }

        private void DisplayText(Object myObject, EventArgs myEventArgs)
        {
            this.log.AppendText(RxDataString + "\n");
        }

        private void serialPort1_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            RxDataString = serialPort1.ReadLine();

            tratarDados(RxDataString);

            if (this.parts.Length > 1)
            {
                this.Invoke(new EventHandler(DisplayText));
            }
            else if (RxDataString == "400")
            {
                RxDataString = "400    -> Evento de ESPERA";
                this.Invoke(new EventHandler(DisplayText));
            }
            else if (RxDataString == "401")
            {
                RxDataString = "401    -> Erro de Colisão.";
                this.Invoke(new EventHandler(DisplayText));
            }
            else if (RxDataString == "402")
            {
                RxDataString = "402    -> Erro de Código Inválido";
                this.Invoke(new EventHandler(DisplayText));
            }
        }

        private void iniciar_Click(object sender, EventArgs e)
        {
            if (!this.serialPort1.IsOpen)
            {
                this.serialPort1.PortName =
this.portas.SelectedValue.ToString(); // "COM4";
                this.serialPort1.BaudRate = 9600;
                this.serialPort1.Open();
            }

            if (this.serialPort1.IsOpen)
            {
                this.initUpdate();
            }

        }

        private void parar_Click(object sender, EventArgs e)
        {
            if (this.serialPort1.IsOpen)
            {
```

```
                this.serialPort1.Close();
            }
        }

        // Envia valor
        private void button3_Click(object sender, EventArgs e)
        {
            this.enviarDados(this.textValor.Text);
        }

        // Códigos de Comando

        private void button2_Click(object sender, EventArgs e)
        {
            RxDataString = "300: Obter Dados";
            this.Invoke(new EventHandler(DisplayText));
            this.enviarDados("300");
        }

        private void button1_Click(object sender, EventArgs e)
        {
            RxDataString = "301: Reiniciar";
            this.Invoke(new EventHandler(DisplayText));
            this.enviarDados("301");
        }

        private void buttonNovos_Click(object sender, EventArgs e)
        {
            RxDataString = "302: Novos Dados";
            this.Invoke(new EventHandler(DisplayText));
            this.enviarDados("302");
        }

        private void buttonOrig_Click(object sender, EventArgs e)
        {
            RxDataString = "303: Dados Originais";
            this.Invoke(new EventHandler(DisplayText));
            this.enviarDados("303");
        }
        private void button5_Click(object sender, EventArgs e)
        {
            RxDataString = "304: Redefinir";
            this.Invoke(new EventHandler(DisplayText));
            this.enviarDados("304");
        }
        private void button6_Click(object sender, EventArgs e)
        {
            RxDataString = "305: Configurações";
            this.Invoke(new EventHandler(DisplayText));
            this.enviarDados("305");
        }
        private void btLimpar_Click_1(object sender, EventArgs e)
        {
            this.log.Text = "";
        }
        private void checkBox1_CheckedChanged(object sender, EventArgs e)
        {
        }
    }
}
```