FACULDADE SATC

LUIZ HENRIQUE CASSETTARI

PROPOSTA DE EXPERIMENTO REMOTO DIDÁTICO APLICADO AO CONTROLE DE ROTAÇÃO DE MOTOR DE CORRENTE CONTÍNUA

Criciúma

Novembro – 2011

LUIZ HENRIQUE CASSETTARI

PROPOSTA DE EXPERIMENTO REMOTO DIDÁTICO APLICADO AO CONTROLE DE ROTAÇÃO DE MOTOR DE CORRENTE CONTÍNUA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica da Faculdade SATC, como parte dos requisitos à obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Me. João Mota Neto.

Coordenador do Curso: Prof. Me. André Abelardo Tavares.

Criciúma

Novembro – 2011

LUIZ HENRIQUE CASSETTARI

PROPOSTA DE EXPERIMENTO REMOTO DIDÁTICO APLICADO AO CONTROLE DE ROTAÇÃO DE MOTOR DE CORRENTE CONTÍNUA

Este Trabalho de Conclusão de Curso foi julgado adequado à obtenção do título de bacharel em Engenharia Elétrica e aprovado em sua forma final pelo Curso de Graduação em Engenharia Elétrica da Faculdade SATC.

Criciúma, 25 de Novembro de 2011.

_____

Professor e orientador João Mota Neto, Mestre.
Faculdade SATC

_____

Prof. Giovani Martins Cascaes, Mestre.
Faculdade SATC

_____

Prof. Marcel Campos Inocencio, Especialista.
Faculdade SATC

Dedico a minha família, meus amigos, colegas de classe, e professores, em especial ao meu orientador, por todo apoio e incentivo.

AGRADECIMENTOS

Gostaria de agradecer ao meu pai Márcio e a minha mãe Genésia por estarem ao meu lado sempre que precisei, por todo amor e respeito. Agradeço também ao meu irmão José e a todos familiares pelo apoio e ajuda nas horas de necessidade.

Aos meus amigos, colegas de classe, que estiveram presentes nos bons e maus momentos, ao professor Me. João Mota Neto e a todos os colaboradores da Faculdade SATC.

“Se fiz descobertas valiosas, foi mais por ter paciência do que qualquer outro talento.” (Isaac Newton).

RESUMO

Desenvolver um experimento remoto para o controle em malha fechada da rotação de um motor de corrente contínua. É apresentada a fundamentação teórica sobre experimento remoto, visando controlar o motor remotamente usando a internet como meio de comunicação. Apresentam-se também os recursos tecnológicos para desenvolver uma ferramenta de auxílio ao estudo, que permitirá o controle da rotação do motor pela internet. Desenvolveu-se um aplicativo na linguagem de programação Java para estabelecer a comunicação entre o usuário e o experimento, permitindo a ação de controle e sua visualização por meio do gráfico. O motor será controlado pela plataforma Arduino, a placa Arduino Uno possui um microcontrolador programável responsável pelo algoritmo de controle, e quando unida ao *Ethernet Shield* permite comunicar-se com a rede internet. A partir desse, desenvolveu-se o projeto e a estratégia de controle, bem como, o algoritmo de controle. O modelo de um controlador PID digital foi implementado à plataforma Arduino, assim como a lógica de controle pela internet. Validou-se o algoritmo PID implementado no Arduino Uno por meio da ferramenta computacional MATLAB, comparando os gráficos gerados pelo aplicativo e simulados, referente ao experimento na sua totalidade após interações via internet.

Palavras-chave: Experimento remoto, Java, Arduino, Controlador PID.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

SIGLAS

API	___	*Application Programming Interface*
CA	___	Corrente Alternada
CC	___	Corrente Contínua
E/S	___	Entrada/Saída
ICSP	___	*In-Circuit Serial Programming*
IP	___	*Internet Protocol*
MAC	___	*Media Access Control*
PID	___	Proporcional-Integral-Derivativo
PWM	___	*Pulse-Width Modulation*
SATC	___	Associação Beneficente da Indústria Carbonífera de Santa Catarina
SPI	___	*Serial Peripheral Interface*
TCP	___	*Transmission Control Protocol*
UDP	___	*User Datagram Protocol*
USB	___	*Universal Serial Bus*

SÍMBOLOS

A	[A]	Corrente em Amperes
V	[V]	Tensão em Volts

SUMÁRIO

1 INTRODUÇÃO

A internet proporciona uma ferramenta ampla de comunicação, podendo ser utilizada por diversas áreas e finalidades. No âmbito da educação aplica-se este recurso para desenvolvimento de aplicações que proporcionem maior interação interdisciplinar e na implementação de cursos à distância.

Contudo faz-se necessário o uso de dispositivos eletrônicos físicos com a capacidade de serem acessados e controlados pela internet. Sendo estes nomeados de experimento remoto, os alunos e professores podem utilizar esse recurso para interagir e receber informações reais de experimentos, validando teorias discutidas em sala de aula.

A proposta de experimentos remotos foca-se no auxílio de atividades práticas a serem realizadas pelos estudantes na compreensão de conteúdos curriculares ministrados nas aulas. Podendo desfrutar desse recurso em qualquer horário, pois se encontra ligado a rede mundial de computadores disponível 24 horas por dia e 7 dias por semana. Devido à disponibilidade de acesso, flexibiliza o horário do usuário ao interagir com a atividade prática virtual.

Neste contexto, o presente trabalho propõe um experimento remoto didático prático. Esse proporcionará a interação do acadêmico ao controle da rotação de um motor de corrente contínua. Desse modo, os alunos e os professores poderão utilizar essa ferramenta, disponível na internet, para estudos relacionados a disciplinas de sistema de controle.

O controle empregado ao motor, especificamente, será um controlador PID. Nele o usuário terá acesso aos parâmetros desse controlador, a fim de modificá-los e poder avaliar o desempenho do mesmo, utilizando a interface específica do experimento que permite o controle do motor e visualização da rotação por meio de gráficos.

1.1 JUSTIFICATIVA E CONTRIBUIÇÕES

O crescimento da Internet e a acessibilidade desta, abrangendo um número maior de estudantes, favorece o surgimento de novas tecnologias aplicáveis à educação. Dentre as tecnologias podemos destacar o uso de experimentos remotos, como uma alternativa para atender o aumento da demanda por experimentos laboratoriais (COOPER, 2000). Quando comparado o ambiente virtual ao laboratório real, este possui capacidade de gerar dados experimentais reais por meio de controle remoto do equipamento real automatizado por um maior número de alunos. Assim, a atividade prática ajuda tanto no aprofundamento conceitual dos acadêmicos como no desenvolvimento da sua capacidade de elaborar e desenvolver projetos. [2]

A principal contribuição deste projeto é a disponibilização de uma ferramenta de auxílio no ensino de sistemas de controle. No espaço educacional do ambiente remoto os usuários poderão usufruir de recursos visuais e de interatividade que propiciará o aprofundamento dos conceitos relacionados ao controlador PID estudados no curso de engenharia.

1.2 ORGANIZAÇÃO

- Capítulo 1: apresenta-se uma introdução sobre a proposta do experimento remoto didático, bem como os objetivos do presente trabalho;
- Capítulo 2: aborda-se a fundamentação teórica do experimento remoto, relacionada aos itens e ferramentas utilizadas para desenvolvimento do projeto;
- Capítulo 3: são realizadas a modelagem, a identificação e o desenvolvimento do experimento remoto proposto;

- Capítulo 4: são apresentados os resultados experimentais obtidos nos testes realizados para verificar o desempenho do experimento remoto, e validar a resposta do sistema;
- Capítulo 5: são expostas as conclusões do presente trabalho e sugestões para trabalhos futuros.

1.3 OBJETIVO GERAL

Implementar um experimento remoto didático que auxilie os acadêmicos na compreensão dos parâmetros do controlador PID aplicados a um motor de corrente contínua.

1.4 OBJETIVOS ESPECÍFICOS

- Estudar as técnicas aplicadas no desenvolvimento de experimentos remotos;
- Definir a arquitetura de software e hardware referente ao experimento;
- Implementar o hardware;
- Desenvolver o software;
- Implementar e validar o controlador digital PID.

2 FUNDAMENTAÇÃO TEÓRICA

O presente capítulo apresenta os conceitos necessários para compreender os recursos utilizados no desenvolvimento do experimento, sendo apresentados os fundamentos referentes ao experimento remoto como: Java, Arduino e estrutura de controle.

2.1 EXPERIMENTO REMOTO

Nas últimas décadas os métodos de ensino sofreram diversas mudanças, e houve aumento da procura por formação acadêmica, fazendo com que as instituições de ensino superior criem métodos viáveis para integrar essa população de estudantes. Assim as instituições de ensino gradativamente estão modificando os métodos de ensino/aprendizagem, visa a modificar o sistema tradicional de ensino, tornando-o flexível.

Uma maior flexibilidade nos cursos das instituições de ensino superior permite que estudantes organizem seu tempo para poderem estudar. Nesse contexto o surgimento de cursos de período noturno permite que diversos alunos trabalhem e estudem. A tecnologia favoreceu a flexibilização dos cursos, pois um vasto número de estudantes possuem acesso a rede mundial de computadores, o que permite o surgimento de novas tecnologias aplicáveis à educação. [3]

A utilização da internet para fins educativos junto com as tecnologias de comunicação colaboram para o desenvolvimento dos métodos de ensino/aprendizagem das instituições de ensino. Para Silva (2007) as principais vantagens que a tecnologia proporcionou para a educação são descritas a seguir:

- Criação de ambientes de ensino/aprendizagem mais flexíveis.
- Quebra das barreiras espaço-tempo entre o professor e os alunos.
- Rompimento dos métodos clássicos do ensino tradicional.

Atualmente algumas instituições possuem cursos de ensino a distância, que muitas vezes só há a necessidade do aluno ir à instituição para realizar

avaliações. Embora nesses cursos que possuem disciplinas relacionadas às áreas da ciência e tecnologia, há a necessidade de diversas aulas práticas laboratoriais, que demandam um horário para reunir os estudantes e o professor para a utilização do laboratório da instituição.

Em geral laboratórios possibilitam que o aluno utilize os conhecimentos teóricos em situações práticas e por meio desses valide a teoria e aprofunde o conhecimento. O aluno tendo contato com situações reais permite que tenha o confronto de idéias entre a teoria e a prática, portanto, o aluno aprende fazendo. [4]

Equipamentos utilizados em experiências práticas presenciais nas instituições de ensino possuem um custo elevado, sua aquisição, manutenção e reposição de materiais, acarretam altos custos para a instituição. Além da necessidade de um número adequado de laboratórios para atender toda a demanda de alunos. Neste contexto podemos destacar o uso de laboratórios remotos, como uma alternativa para atender o aumento da demanda por laboratórios (COOPER, 2000).

Laboratório remoto é um conjunto de equipamentos que proporciona a utilização de um experimento por pessoas fora do laboratório, ou seja, possibilita que experimentos sejam controlados e monitorados por alunos de sua própria casa, utilizando a internet como meio de comunicar-se com o experimento. Segundo Silva (2007), os laboratórios de acesso remoto buscam resolver de forma eficiente o problema na demanda por laboratórios clássicos, com os objetivos de:

> - Incrementar as atividades práticas em um curso (de forma que os alunos possam acessar a eles em qualquer horário, não somente quando esteja aberto o centro para temas docentes),
> - Reduzir os custos de gestão e manutenção dos laboratórios (ao aumentar o uso em qualquer horário aos mesmos com um pessoal menor),
> - Permitir o uso dos mesmos desde qualquer ponto geográfico de forma que se reduzam ou minimizem os custos de deslocamento, assim como a qualquer hora, permitindo desta forma resolver o problema dos fusos horários com outras zonas geográficas, e,
> - Integrar em um mesmo ambiente as aplicações docentes das práticas, experimentação e trabalho no laboratório, com as atividades propriamente docentes mediante a integração de materiais, simulações e acesso a equipamentos e dispositivos. [3]

Os laboratórios remotos permitem maior acesso ao estudante quando comparado aos laboratórios convencionais, porém o professor não acompanhará a experiência remota, havendo a necessidade de um material auxiliar que explique o

funcionamento do experimento, auxiliando na compreensão da experiência a ser desenvolvida, favorecendo a aprendizagem.

Esses laboratórios remotos com base na *web* permitem estabelecer novos padrões de ensino-aprendizagem. A educação à distância pretende disponibilizar um sistema de ensino-aprendizagem flexível, acessível e adaptável sem limitações espaciais nem temporais. [3]

2.2 JAVA

Para que qualquer linguagem de programação possa funcionar no computador, é necessário que a máquina reconheça os comandos do programa, para poder interpretar cada linha de código existente no algoritmo, permitindo que o programa possa ser executado.

Segundo Rodrigues (2008) para que o computador possa rodar um programa escrito em Java é preciso ter instalado um programa chamado Java *Runtime Enviroment* (JRE), conhecido também como Java *Virtual Machine* (JVM), que é basicamente um emulador que transforma a sequência de bytes escrito em Java para uma linguagem denominada *bytecodes*. [5]

O computador compreende e executa o *bytecode* como se fosse uma linguagem padrão da máquina, portanto qualquer computador que possua o JRE instalado consegue executar um programa escrito em Java, assim os acadêmicos deverão ter instalado o JRE para terem acesso ao experimento.

A linguagem de programação Java foi desenvolvida na década de 90, os principais objetivos são descritos a seguir: [5]

- Sintaxe similar a C/C++;
- Dinâmica;
- Orientação a objetos;
- Independência de plataforma;
- Recursos de rede internet;
- Vasto conjunto de bibliotecas.

A linguagem de programação Java não necessita de um ambiente de programação, porém será utilizado um *software* para auxiliar o desenvolvimento do aplicativo do experimento remoto proposto, esse deverá ser o *software* NetBeans, por ser de domínio público, e possuir diversos recursos que facilitam a programação como: *help*, auto-completar códigos e a interface visual de programação.

O NetBeans é um ambiente de desenvolvimento de código fonte aberto para desenvolvedores de aplicativos em diversas linguagens, como: Java, C, C++, PHP, HTML, Ruby, entre outras. O próprio software NetBeans foi criado na linguagem Java, o tornando independente da plataforma, ou seja, opera em qualquer sistema operacional que possua o JVM instalado. [6]

Segundo MARSHALL (2008), o NetBeans possui uma base para criar projetos e módulos, por possuir diversos conjuntos de bibliotecas e interfaces de programação, conhecidas como *Application Programming Interface* (API). Uma API utilizada no desenvolvimento do aplicativo Java do experimento remoto proposto é o *Swing,* esse facilita a criação de botões, janelas, blocos de texto, barras de rolagem, entre outros. Essa API permite, por exemplo, modificar visualmente utilizando o mouse o tamanho de botões, sendo desnecessário criar o código desse botão, pois a API automaticamente implementa o algoritmo do botão, facilitando o desenvolvimento do aplicativo. [6] A Figura 1 demonstra a tela principal do software NetBeans.

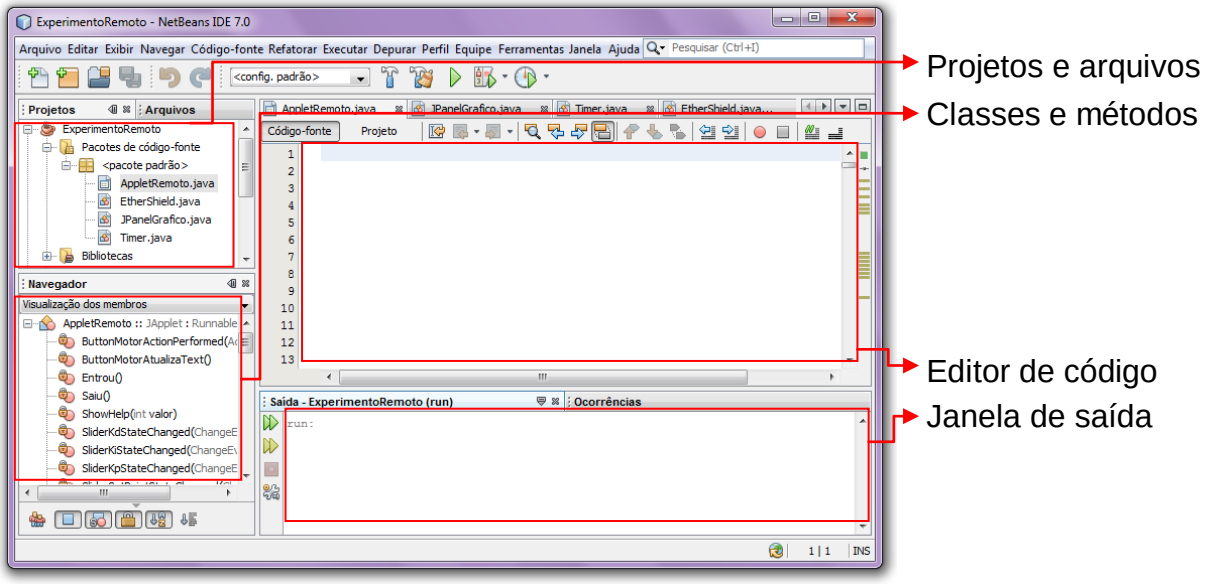


Figura 1 – Janela principal do software NetBeans.

Fonte: Do autor, 2011.

A figura anterior demonstra o *software* NetBeans e nela são destacados as janelas principais: projetos e arquivos, classes e métodos, editor de código e janela de saída. A funcionalidade de cada janela do NetBeans da Figura 1 é explicada a seguir:

- Projetos e arquivos: Permite visualizar, selecionar e modificar os arquivos existentes nos projetos, podendo copiar arquivos de um projeto ao outro.
- Classes e métodos: Possibilitam verificar os métodos existentes nos arquivos do projeto, como os parâmetros envolvidos nos métodos.
- Editor de código: Permite modificar o código de um determinado arquivo, ou seja, é o local onde é escrito o programa.
- Janela de saída: Janela padrão de saída do *software*, onde são visualizados os possíveis erros de programação de um determinado arquivo.

Esse ambiente de desenvolvimento de programação será utilizado para desenvolver o ambiente de interação entre o usuário e o experimento remoto. A linguagem utilizada para desenvolver o aplicativo será o Java, mais especificamente será um aplicativo *applet.* [6]

Applets são aplicativos que fornecem recursos para aplicações em páginas *web*, ou seja, são inseridos em uma página a fim de fornecer todos os recursos do aplicativo à página. Esse tipo de aplicativo permitirá que o usuário controle o experimento remoto ao acessar uma página *web,* que possuirá esse aplicativo *applet*.

2.3 SOCKET EM JAVA

Em Java os recursos fundamentais para comunicação com a rede são declarados pelas classes e interfaces do pacote *java.net*, por meio desse o Java oferece comunicações baseadas em fluxo que permitem aos aplicativos visualizar a rede como fluxos de dados. [6]

As comunicações baseadas em *sockets* de fluxo são visualizadas pelo aplicativo Java como uma E/S de arquivos de dados, ou seja, um programa em Java

pode enviar e ler dados da rede utilizando uma classe denominada *Socket*. Um aplicativo ao utilizar o fluxo de *socket* pode estabelece uma conexão com outro processo, enquanto esse estiver conectado os dados fluem entre os processos em fluxos contínuos. O protocolo utilizado para realizar esse fluxo é o *Transmission Control Protocol* (TCP). [7]

O TCP é um protocolo de transporte de rede que fornece a programas um serviço de comunicação de dados confiáveis com controle de fluxo. Depois de solicitar ao *socket* que estabeleça uma conexão TCP, um aplicativo pode usar a conexão para enviar e receber dados, esse protocolo garante a entrega de dados em ordem e sem duplicação. Quando os dois aplicativos terminam de usar a conexão, eles solicitam que a mesma seja terminada. [7]

Um computador que se comunica com outro utilizando TCP trocam diversas mensagens. Todas as mensagens de um TCP a outro usam o formato de segmento do TCP, incluindo mensagens que transportam dados, acknowledgments e anúncios de janela, ou mensagens usadas para estabelecer e terminar uma conexão. Cada segmento de TCP viaja em um datagrama IP. [7]

O TCP utiliza uma variedade de mecanismos para assegurar um serviço confiável. Além de um checksum em cada segmento, o TCP retransmite qualquer mensagem que seja perdida.

Esse protocolo de transporte será utilizado na comunicação do experimento remoto, pois esse garante o envio e recebimento dos pacotes. O aplicativo Java também utilizará o protocolo TCP, utilizando a classe *Socket* para realizar a conexão com o experimento remoto.

2.4 ARDUINO

Este dispositivo integra o conceito de hardware e software livre e está aberto para uso e contribuição de toda sociedade. O conceito Arduino surgiu na Itália, em 2005, com o objetivo de criar um equipamento para desenvolver projetos/protótipos construídos de uma forma menos dispendiosa do que outros sistemas disponíveis no mercado. [8]

Essa plataforma *open-source* foi projetada com a finalidade de facilitar a compreensão, programação, aplicação e multiplataforma, ou seja, podemos programá-lo em ambientes Windows, GNU/Linux e Mac OS. Destinada a qualquer pessoa interessada em criar objetos ou ambientes interativos. [9]

2.4.1 Arduino Uno

Esta placa eletrônica é baseada no chip ATmega328 fabricado pela Atmel, possui entrada/saída (E/S) digital e analógica, além de interface serial através da porta USB, permite ao usuário programar e interagir em tempo real. Os conectores de entrada e saída são expostos de maneira que possam ser interligados a outros módulos expansivos, conhecidos como *Shields*. A Figura 2 demonstra a placa Arduino Uno com destaque nos conectores de: E/S digital, referência analógica, alimentação e entrada analógica. [10]

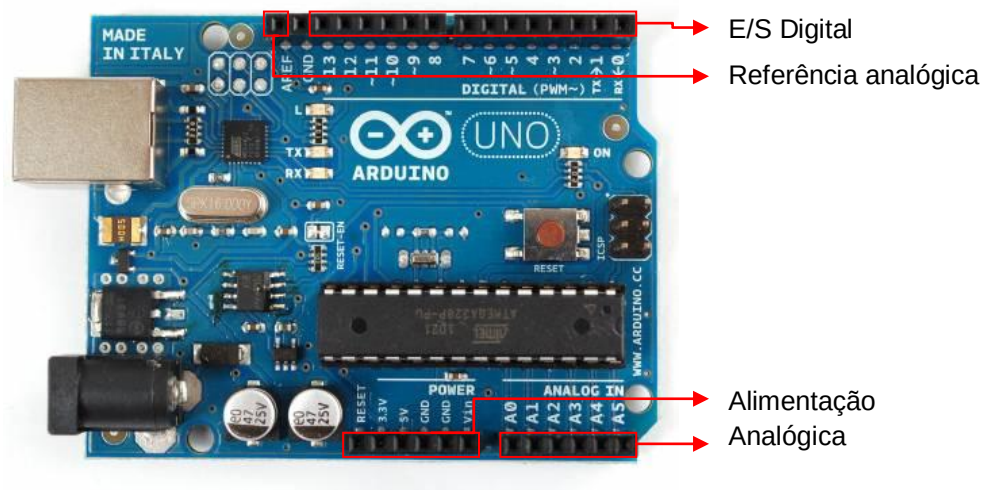


Figura 2 – Placa Arduino Uno.

Fonte: Arduino. Disponível em: <http://arduino.cc/en/Main/ArduinoBoardUno>. Acesso em: 30 ago. 2011.

O Arduino Uno que será utilizado no experimento é composto por quatorze pinos de E/S digital, seis entradas analógicas, oscilador de cristal 16 MHz,

controlador USB, uma entrada de alimentação, conector ICSP e um botão de *reset*. Para sua utilização, basta conectá-lo a um computador com um cabo USB ou ligá-lo com um adaptador AC para CC ou bateria. Na Tabela 1, encontram-se maiores informações técnicas da placa eletrônica em estudo. [8]

Tabela 1 – Características Técnicas do Arduino Uno.

Características Técnicas	
Microcontrolador	Atmega328
Tensão Operacional (recomendada)	5V
Tensão de Alimentação (recomendada)	7-12V
Tensão de Alimentação (limites)	6-20V
Pinos E/S Digitais	14 (6 podem ser saídas PWM)
Pinos de Entrada Analógica	6
Corrente Contínua por Pino E/S	40mA
Corrente Contínua para o Pino 3.3V	50mA
Memória Flash	32KB
SRAM	2KB
EEPROM	1KB
Frequência de *Clock*	16MHz

Fonte: Arduino. Disponível em: <http://arduino.cc/en/Main/ArduinoBoardUno>. Acesso em: 30 ago. 2011.

O chip Atmega328 possui conversor analógico-digital de 10 bits, nomeados como entrada analógica no Arduino Uno. O conversor suporta mapear sinais de tensões entre 0 e 5 Volts em valores inteiros entre 0 e 1023, resulta na resolução nas leituras de: 5 Volts / 1024 unidades ou, 0,0049 Volts por unidade. Esse intervalo de entrada e resolução pode ser alterado utilizando o conector de referência analógica. Esse possibilita diminuir a máxima tensão da entrada analógica de 5 Volts para uma tensão menor. Para ativar a referência analógica é preciso conectar uma tensão entre 0 e 5 Volts a entrada de referência e utilizar o código para ativar a referência externa. [10]

Outra característica importante do Arduino Uno é a facilidade de gravar novos programas no Atmega328, pois essa operação é executada na placa do Arduino Uno, sendo utilizado o conversor USB para serial e o *firmware* gravado no

chip Atmega328. Desse modo o usuário não necessita retirá-lo do *socket* ou utilizar uma placa eletrônica específica para realizar a operação.

2.4.2 Ethernet Shield

Para realizar a conexão do experimento proposto neste trabalho utilizando a rede internet como comunicação entre o usuário e o experimento, faz-se necessário adicionar a placa do Arduino Uno responsável por controlar o motor elétrico e por meio da leitura do taco gerador à placa eletrônica *Ethernet Shield.*

O principal componente empregado na placa *Ethernet* é o chip ENC28J60 da Microchip, que suporta diversos protocolos de comunicação como: TCP, UDP, entre outros. Esse chip foi projetado para permitir que qualquer dispositivo que possua comunicação *Serial Peripheral Interface* (SPI) possa estabelecer uma conexão *Ethernet*. As características técnicas do chip ENC28J60 encontram-se na Tabela 2. [11]

Tabela 2 – Características Técnicas do ENC28J60.

MAC	Sim
PHY	Sim
TX/RX RAM Buffer (Bytes)	8192
Pino de interrupção	1
LED	2
Tensão de operação (V)	3.3
Temp. Mínima (°C)	-40
Temp. Máxima (°C)	85
Comunicação	SPI
Endereço MAC Pré-Programado	Não
Mecanismos de segurança	Não
Controlador *Ethernet*	10Base-T

Fonte: Microchip ENC28J60. Disponível em: <http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en022889>. Acesso em: 30 set. 2011.

O Arduino Uno possui a comunicação SPI, que é localizada através das E/S digitais 10, 11, 12 e 13. Assim a placa *Ethernet Shield* possui o *layout* que permite o encaixe rápido, conectando todos os pinos do Arduino Uno aos respectivos pinos do *Shield*, desse modo não é necessário a utilização de cabos externos para interligar as placas. A Figura 3 apresenta duas placas sobrepostas e unidas com intuito de demonstrar o encaixe rápido. [11]

Figura 3 – Arduino Uno ligado ao Ethernet Shield.
Fonte: Do autor, 2011.

Para comunicar-se com a internet é preciso que diversos pacotes sejam trocados de forma organizada para estabelecer a conexão. Para que o *Ethernet Shield* possa trocar pacotes e se conectar com a rede é preciso incluir uma biblioteca específica da placa, que inclui funções e comandos de configurações que permitem transferência dos pacotes e conexão do *Ethernet*.

Os pacotes da rede recebidos pelo *Shield* são enviados ao Arduino Uno por SPI, esses pacotes são armazenados em um buffer interno do chip Atmega328, são analisados pela biblioteca do *Ethernet*. Esse buffer é criado ao incluir essa biblioteca, porém o tamanho do buffer é limitado pelo tamanho da SRAM sendo de 2000 Bytes. [12]

Na biblioteca do *Ethernet* possui alguns exemplos, dentre esses exemplos é utilizado um tamanho de buffer de 500 Bytes, que é suficiente para estabelecer a conexão. Nos exemplos é possível modificar o valor do endereço IP e porta de acesso do Arduino, e também o endereço MAC.

2.4.3 Ambiente de Programação do Arduino

O ambiente de programação é um *software* de desenvolvimento de algoritmos para o Arduino, assim esse programa permite criar códigos que podem ser gravados a placa. A linguagem empregada nesse ambiente de desenvolvimento de aplicações é uma versão simplificada da linguagem de programação C, possuindo o mesmo tipo de regras e funções básicas. Na Figura 4 é possível visualizar o ambiente de desenvolvimento da programação do Arduino. [8]

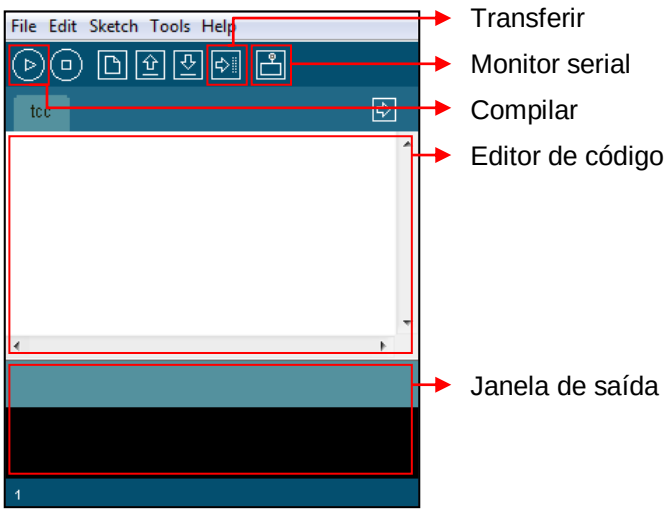


Figura 4 – Ambiente de Programação do Arduino.
Fonte: Do autor, 2011.

A figura acima demonstra o ambiente de desenvolvimento do Arduino onde são destacados os itens: transferir, monitor serial, compilar, editor de código e janela de saída. A funcionalidade de cada item da Figura 4 é explicada a seguir: [13]

- Transferir: Botão utilizado para transferir o código ao Arduino.
- Monitor serial: Botão que permite o acesso a janela de monitoramento de variáveis do programa.
- Compilar: Botão responsável por compilar o código e validar o algoritmo a fim de apontar possíveis erros de programação.
- Editor de código: Permite modificar e desenvolver o algoritmo que poderá ser transferido para o Arduino.

– Janela de saída: Local onde são apontados os erros do programa, como erro de compilação.

O algoritmo criado no programa passará por uma validação, quando pressionado o botão de compilar, o código passará por um compilador que poderá apontar erros de sintaxe e de programação. A Figura 5 apresenta um diagrama de blocos a fim de demonstrar a funcionalidade do *software*.

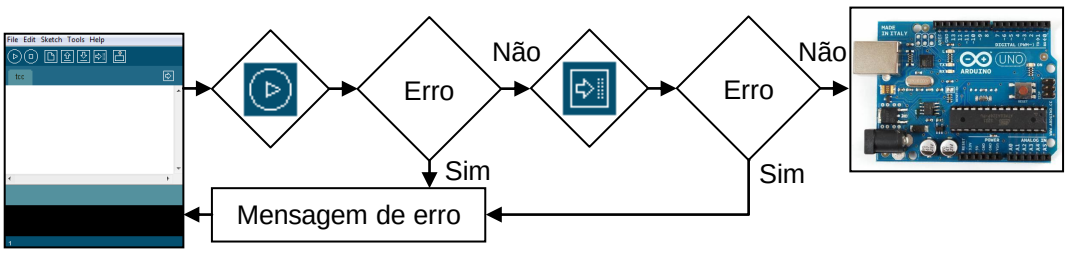


Figura 5 – Transferência e monitoramento do programa.
Fonte: Do autor, 2011.

Conforme apresentado acima ao pressionar o botão de compilar do *software* verifica o algoritmo a procura de erros, se algum erro for encontrado é mostrado uma mensagem de erro na janela de saída. Caso nenhum erro seja encontrado é possível transferir o algoritmo para o Arduino apertando o botão de transferir, assim o programa verifica a comunicação com o Arduino para enviar o código ao seu chip, caso algum erro de transmissão ocorra, o envio do algoritmo é interrompido e uma mensagem de erro aparecerá na janela de saída. [13]

Outro recurso disponível no *software* é o monitor serial, que é utilizado para estabelecer uma comunicação serial com a placa do Arduino, permitindo que mensagens sejam enviadas entre a placa e o *software.* Dessa forma é possível utilizar o monitor serial como um recurso de *debug*, podendo ser usado para receber valores de uma entrada analógica da placa ou enviar uma determinada mensagem para ligar uma saída digital. O botão para abrir a comunicação entre a placa e o *software* pode ser visualizado pela Figura 4, e a janela do monitor serial pela Figura 6. [13]

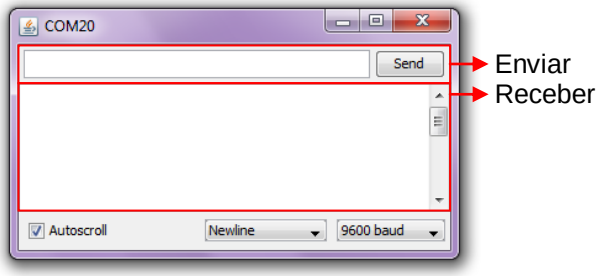


Figura 6 – Janela do monitor serial.

Fonte: Do autor, 2011.

Conforme a figura acima que demonstra a janela do monitor serial onde são destacados os itens: enviar e receber. Esses itens são descritos a seguir:

- Enviar: Bloco utilizado para escrever mensagens de texto para serem enviadas ao Arduino conectado ao computador.
- Receber: Janela de texto onde são mostradas as mensagens enviadas pela placa do Arduino.

O monitor serial será utilizado para auxiliar o andamento do experimento, pois toda a programação e funcionalidades previstas no projeto serão desenvolvidas no ambiente de programação do Arduino, como o controlador PID e a lógica de controle.

2.5 SISTEMA DE CONTROLE

Trata-se de mecanismos que interagem e monitoram o processo, com propósito de manter o sistema dentro de certos patamares aceitáveis. Segundo Nise (2009) um sistema de controle consiste em subsistemas e processos (ou plantas) construídos com o objetivo de se obter uma saída desejada com desempenho desejado, para uma entrada específica fornecida. [15] A Figura 7 mostra a representação de um sistema de controle por um bloco com entrada e saída indicadas por setas.

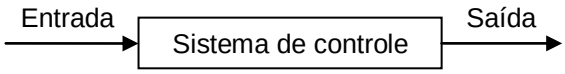


Figura 7 – Sistema de controle simplificado.
Fonte: Do autor, 2011.

Contudo a engenharia de controle baseia-se no princípio da realimentação e objetiva-se no controle de determinadas variáveis de um sistema. Embora esteja tradicionalmente ligada à engenharia elétrica, a engenharia de controle é interdisciplinar e encontra-se vinculada nos cursos de engenharias química, mecânica, aeronáutica, biomédica, educação, etc. [16]

Dentre os diversos tipos de controladores existentes pode-se destacar o controlador Proporcional-Integral-Derivativo (PID), que é o tipo de controlador de estrutura fixa amplamente utilizado nas aplicações industriais, tanto no Brasil como no mundo. Esses controladores tornaram-se disponíveis comercialmente na década de 30, e nos anos 40 estavam sendo utilizados na indústria com enorme aceitação. [17]

Apesar da importância prática do PID e da vasta quantidade de pesquisas sobre este controlador no ambiente acadêmico (BRUCIAPAGLIA, 1992, POSSER, 1998), é normal encontrar no meio industrial controladores PID mal ajustados e/ou utilizados (NORMEY-RICO, 2001). Nesse contexto apresenta-se a proposta do experimento remoto como uma ferramenta para desenvolver a habilidade perante ao controlador.

2.5.1 Estrutura de controle

Existem duas estruturas básicas de sistemas de controle: sistemas em malha aberta e fechada. Um sistema de controle de malha aberta utiliza um controlador em série com o processo que se deseja controlar, de modo que a entrada do processo deve ser um valor para que sua saída se comporte como desejado. A principal característica é que a ação de controle é independe da saída.

Observa-se que um sistema de controle deste tipo fornecera a saída desejada se não ocorrerem perturbações externas que alterem o valor da saída ou alterações paramétricas internas do sistema. Se alguma destas ocorrer, a saída sofrerá alteração, mas a atuação de controle continua exatamente a mesma. [16] A Figura 8 apresenta a estrutura básica do controlar mencionado acima:

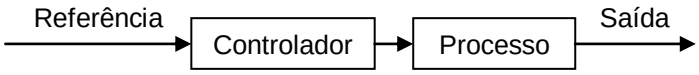


Figura 8 – Diagrama de blocos de um sistema de controle em malha aberta.
Fonte: Do autor, 2011.

Os problemas relacionados acima podem ser solucionados utilizando um sistema de controle de malha fechada, conforme Nise (2009, p.8):

> O sistema de malha fechada compensa as perturbações pela medição da resposta na saída, alimentando aquela medida no caminho de realimentação e comparando aquela resposta com a entrada na junção de adição. Se existir qualquer diferença entre as duas respostas, o sistema aciona a planta, via sinal de atuação, para realizar a correção. Caso não ocorra qualquer diferença, o sistema não aciona a planta, uma vez que a resposta da planta já é a resposta desejada. [14]

Segundo Carvalho (2000), um sistema de controle em malha fechada utiliza a realimentação da informação, uma medida adicional da saída real, a fim de compará-la com a resposta desejada do sistema. Esta estrutura é observada na Figura 9.

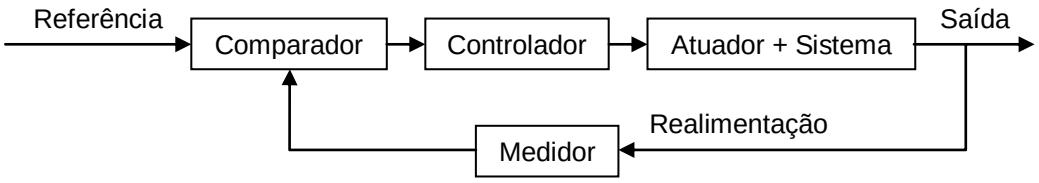


Figura 9 – Diagrama de blocos de um sistema de controle em malha fechada.
Fonte: Do autor, 2011.

A estrutura de controle citada acima possui diversos componentes, geralmente empregados em diagramas de blocos, para facilitar a visualização e

entendimento do conceito, a Figura 9 demonstra o diagrama de blocos de um sistema básico de malha fechada. Abaixo os principais itens deste sistema são descritos:

- Referência: Valor desejado inicial da variável a qual objetiva-se controlar, conhecida também como *setpoint*;
- Comparador: Componente responsável pela produção do sinal de erro, resultado entre o valor desejado e o obtido;
- Controlador: De acordo com o sinal de erro gerado pelo comparador, este dispositivo o manipula, e deste modo origina-se um sinal de controle que será aplicado no sistema, a fim de corrigir a variável a ser controlada.
- Atuador: Dispositivo que recebe o sinal de controle e converte em um sinal de potência suficiente para atuar sobre o sistema.
- Sistema: Dispositivo ou fenômeno que se deseja operar com alguma finalidade, objetivo de controle, pode ser denominado como planta ou processo.
- Medidor: Responsável pela medição e conversão da variável a ser controlada para fins de comparação e obtenção do erro de saída.

2.5.2 Estabilidade de sistemas de controle

A estabilidade em um sistema de controle é essencial para que o processo possua um desempenho adequado. Segundo Nise (2009) os sistemas de controle devem ser projetados para atender os critérios do processo, nestes destaca-se a estabilidade. Assim, um sistema é dito estável, quando submetido a uma entrada limitada ou distúrbio, geram uma saída limitada e esteja dentro dos parâmetros aceitáveis. Permitindo que a resposta ao longo do tempo tenda ao regime normal de operação da planta.

A Figura 10 mostra a resposta de um sistema estável quando submetido a um degrau. Na mesma observa-se o comportamento do regime transitório e permanente.

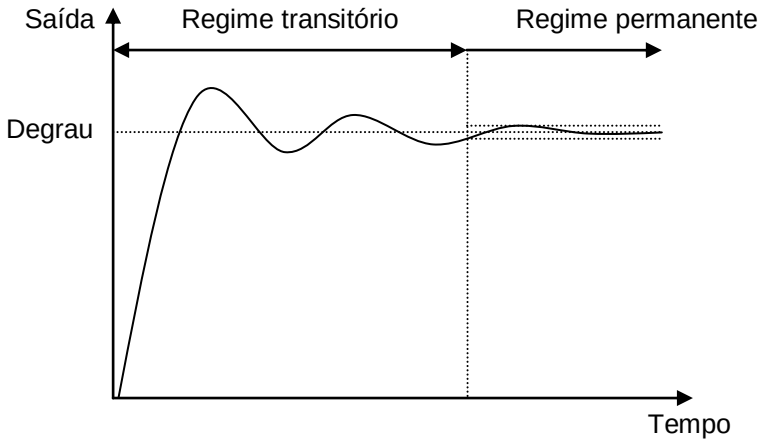


Figura 10 – Resposta de um sistema estável em regime transitório e permanente.
Fonte: Do autor, 2011.

2.5.3 Tipos de controladores

Nos processos industriais existem diversos tipos de controladores usados para atender uma infinidade de problemas industriais. Os controladores amplamente utilizados nas indústrias normalmente são sistemas de controle de malha fechada, pois possuem a realimentação do sinal e permitem resolver diversos problemas encontrados na indústria.

Dentre os controladores de malha fechada podemos destacar o controlador PID. Conforme Campos e Teixeira (2006, p.7):

> O controlador do tipo Proporcional-Integral-Derivativo (PID) é, sem dúvida, o mais usado em sistemas de malha fechada na área da indústria. As vantagens deste controlador são:
> - Bom desempenho em muitos processos;
> - Estrutura versátil;
> - Poucos parâmetros a serem sintonizados ou ajustados;
> - Fácil associação entre parâmetros de sintonia e o desempenho.

O controlador PID utiliza o erro[1] de três métodos distintos para controlar o sinal de saída: o termo proporcional (P), o integral (I) e o derivativo (D). As

[1] Sendo este o resultado da diferença entre a referência e o sinal de realimentação.

características e os efeitos dos parâmetros desses controladores são distintas e são descritas a seguir: [22]

– **Proporcional:** A saída do controlador é diretamente proporcional ao sinal do erro. O controlador proporcional pode ser comparado com um amplificador, com ganho Kp, conforme a equação 1.

$$P = K_p\, e(t) \tag{1}$$

Onde:
K_p = ganho proporcional;
e(t) = Erro;

A ação de controle proporcional não possui a habilidade de eliminar o erro em regime permanente, porém o controlador proporcional pode ser usado com bons resultados em processos de baixa ordem, utilizando um valor de Kp adequado permite acelerar a reposta do sistema. No entanto, em sistemas de ordem superior a utilização de ganhos elevados pode levar o sistema a se tornar instável. [15]

– **Integral:** A ação de controle integral faz com que a saída do controlador seja proporcional a integral do sinal do erro. Desse modo o controlador considera todos os erros anteriores dentro de um espaço de tempo definido no processo. A equação (2) demonstra a fórmula do controle integral. [15]

$$I = K_i \int_0^t e(t)\, dt \tag{2}$$

Onde:
Ki = ganho Integral;

A inclusão da ação integral a um controle de malha fechada tem por objetivo eliminar o erro em regime permanente, porém a utilização de um ganho integral elevado pode criar instabilidade no processo. A ação integral não é utilizada separada da ação proporcional. [15]

– **Derivativo:** O controle derivativo resulta em uma saída proporcional a taxa de variação do erro. Como mostra a equação (3) a taxa de variação é multiplicada pelo Kd.

$$D = K_d \frac{d\,e(t)}{dt} \tag{3}$$

Onde:

Kd = ganho derivativo;

A utilização desse controlador permite corrigir a resposta transitória do sistema. Quando usada em sistemas acionados por referências constantes, a ação derivativa não terá efeito sobre o regime permanente. Por outro lado, essa ação será predominante nos instantes em que a variação do erro é rápida. Isto acontece geralmente nas mudanças de referência ou nos momentos em que o processo é afetado por perturbações de carga. [15] A ação derivativa pode ser interpretada como uma maneira de gerar uma atuação que possa prever um determinado efeito na resposta do sistema e evitá-lo ou, ao menos, diminuí-lo.

Segundo Campos e Teixeira (2006) a união entre a ação proporcional, integral e derivativa resulta no controlador PID, que podem ser encontrados em diversos formatos. Dentre os formatos existentes o escolhido para ser implementado ao experimento foi o PID em paralelo, por ser o amplamente encontrado dentre os controladores industriais. A Figura 11 demonstra o diagrama de blocos de um sistema de controle de malha fechada utilizando um controlador PID.

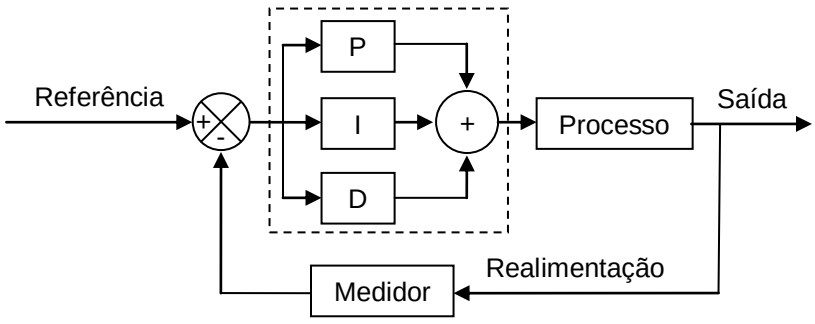


Figura 11 – Diagrama de blocos de um controlador PID.

Fonte: Do autor, 2011.

A figura anterior mostra o digrama de blocos do controlador PID, esse possui um somador que integra as ações dos controladores: proporcional, integral e derivativo. Resultando no controlador PID em paralelo, sendo a equação desse controlador é demonstrada logo abaixo pela equação 4.

$$PID = K_p\, e(t) + K_i \int_0^t e(t)\, dt + K_d\, \frac{d\, e(t)}{dt} \tag{4}$$

Utilizando a equação acima do controlador PID no domínio do tempo pode-se desenvolver uma equação de um controlador PID digital, para poder utilizá-la no Arduino. Para aproximar a equação 4 é utilizado uma taxa de amostragem com período T, assim a integral se aproxima ao somatório de todos os valores amostrados do erro multiplicado pelo período e a derivada pela diferença do erro atual e anterior dividido pelo período. A equação do PID digital é mostrada pela equação 5 [15].

$$PID = K_p\, e(t) + K_i\, T \sum_{i=0}^{t} e(i) + K_d\, \frac{e(t) - e(t-1)}{T} \tag{5}$$

Essa equação será utilizada para desenvolver um algoritmo do controlador PID digital para a plataforma do *software* do Arduino. Será utilizado para controlar a rotação do motor, permitido modificar os parâmetros do controlador PID, o ganho proporcional, integral e derivativo.

3 PROCEDIMENTOS METODOLÓGICOS

A montagem do experimento remoto foi analisada levando em consideração os fatores necessários para a realização do experimento remoto, que são basicamente o controle do motor pelo Arduino Uno e permitir ao usuário interação a mesma via internet.

Foram utilizados os conhecimentos adquiridos na literatura para obter o melhor desempenho do experimento, como informações de *hardware* e *software* do Arduino para assim implementar um algoritmo adequado de comunicação pela rede. O desenvolvimento do projeto foi dividido em etapas para facilitar a conclusão do experimento, sendo essas divididas em:

- Levantamento de parâmetros para controlar o motor;
- Controle do motor através do Arduino;
- Comunicação via internet;
- Desenvolvimento do ambiente virtual de interação;

3.1 LEVANTAMENTO DOS PARÂMETROS ELÉTRICOS PARA CONTROLAR O MOTOR

No primeiro momento o foco do projeto foi voltado para identificação das características elétricas básicas do motor e do taco gerador, a fim de avaliar e dimensionar os circuitos eletrônicos auxiliares necessários para a interação do Arduino Uno aos periféricos. Desse modo, levantou-se as características do motor CC, que possui tensão nominal de 24 V e corrente nominal de 1 A. Também foi visualizado o comportamento do taco gerador em função da tensão aplicada no motor, mostrado pela Figura 12 e o motor pela Figura 13.

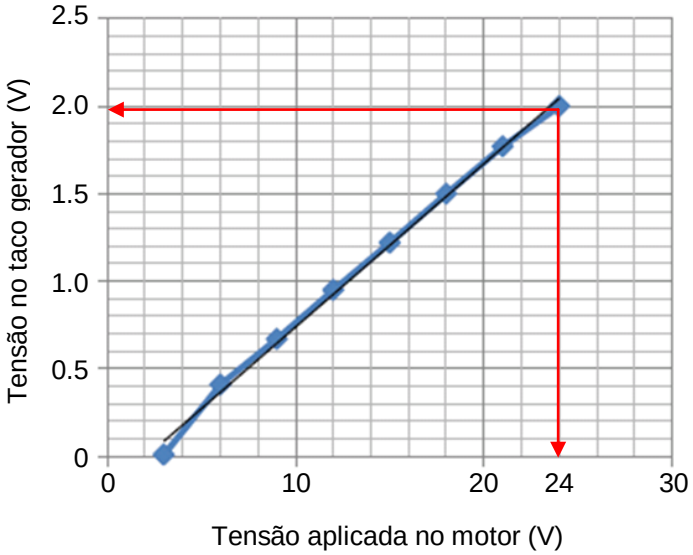


Figura 12 – Comportamento do taco gerador.

Fonte: Do autor, 2011.

Observa-se pela figura acima que a relação entre a tensão aplicada no motor e a gerada pelo taco gerador é linear, atingindo sua amplitude máxima gerada no taco gerador de 2 V com uma tensão nominal no motor de 24 V. A tensão fornecida pelo taco gerador será posteriormente conectada a uma entrada analógica do Arduino Uno, a fim de criar uma realimentação. Dessa forma fez-se necessário identificar as características da entrada analógica do Arduino para verificar o limite de tensão.

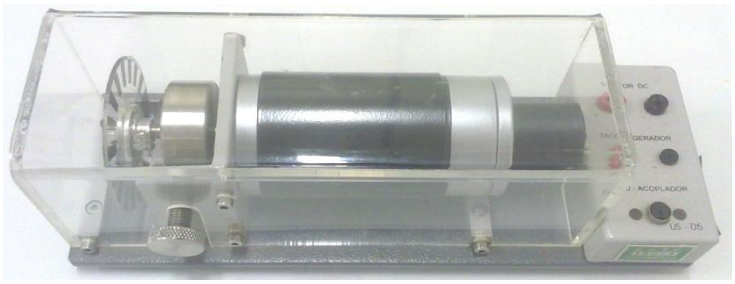


Figura 13 – Motor CC.

Fonte: Do autor, 2011.

Conforme a fundamentação, a entrada analógica possui 10 bits de resolução e tem a capacidade de ler sinais elétricos de 0 a 5 Volts. Sabendo que a tensão máxima fornecida pelo taco gerador é de 2 V, possibilitou-se utilizar o recurso

do hardware do Arduino, a entrada de referência analógica que permite diminuir o range de tensão analógico. Nesse caso conecta-se uma tensão de 2 Volts a entrada de referência analógica e a entrada analógica passou a ter uma escala de 1.95 mV, pois pode ler sinais elétricos de 0 a 2 Volts. Assim não há necessidade de utilizar circuitos de amplificação para a tensão do taco gerador.

Definida a etapa de condicionamento do sinal do taco gerador observou-se a corrente elétrica consumida pelo motor e os níveis de tensão necessários para alimentar a placa eletrônica do Arduino Uno. Abaixo é listada a descrição dos pré-requisitos necessários para posterior construção de uma placa eletrônica auxiliar.

- Tensão de alimentação do Arduino, sendo 5 Volts;
- Tensão de 2 Volts para referência analógica;
- Possuir um circuito para ligar e desligar o motor;
- Necessidade de utilizar um transistor para acionar o motor, devido a limitação de corrente elétrica na saída digital do Arduino, sendo no máximo 40 mA.

Após definido os níveis de tensões necessárias para suprir a demanda do experimento, elaborou-se o circuito com o auxilio do *software* Proteus. A Figura 14 a seguir mostra o circuito elaborado, este deve ser alimentado por uma tensão de 24 Volts alternado, possuir uma ponte retificadora de onda completa, um regulador de tensão de 24 Volts e de 5 Volts, um divisor de tensão de 2 Volts e um transistor de potência para ligar o motor.

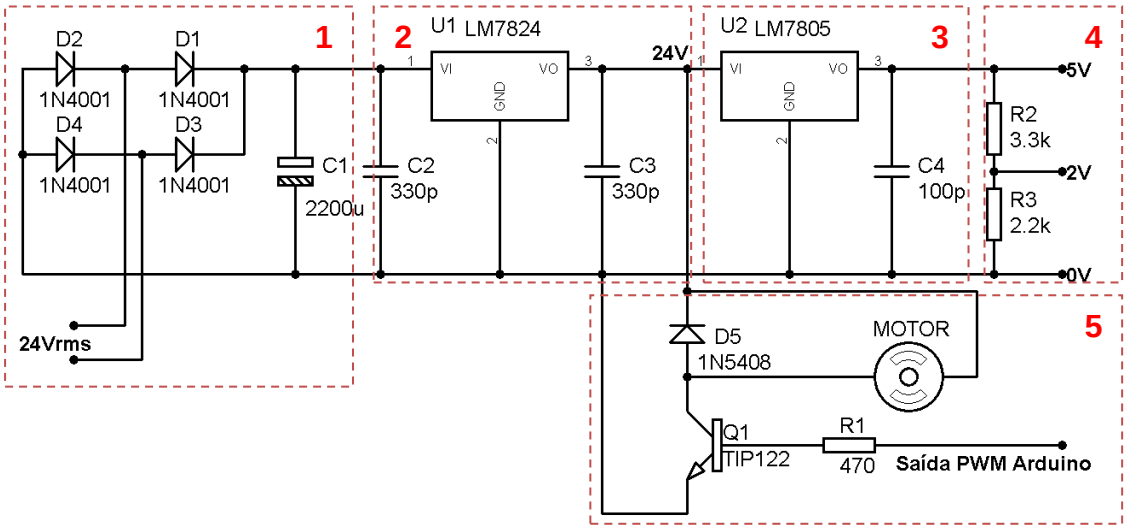


Figura 14 – Circuito da placa de alimentação.

Fonte: Do autor, 2011.

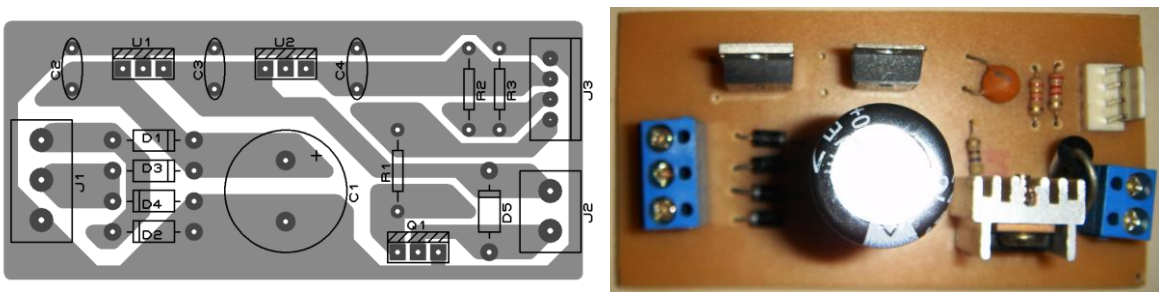
A Figura 14 mostra o circuito da placa auxiliar que é dividida em cinco circuitos, nomeados de: ponte retificadora de onda completa, regulador LM7824, regulador LM7805, divisor de tensão e controle do motor. A Tabela 3 a seguir apresenta a finalidade de cada circuito.

Tabela 3 – Descrição dos circuitos da placa de alimentação.

Circuito	Nome	Finalidade
1	Ponte retificadora de onda completa.	Converter o sinal de tensão alternado para em um sinal próximo ao contínuo.
2	Regulador LM7824.	Regular a tensão fornecida pela ponte retificadora a uma tensão de 24 Volts contínua.
3	Regulador LM7805.	Regular para 5 Volts a tensão fornecida pelo regulador LM7824.
4	Divisor de tensão.	Divide a tensão de 5 Volts fornecida pelo regulador LM7805 a uma tensão de 2 Volts.
5	Controle do motor.	Alimentar o motor e permitir ligá-lo e desligá-lo utilizando uma saída digital do Arduino.

Fonte: Do autor, 2011.

Com a conclusão da simulação da placa de alimentação auxiliar gerou-se o circuito impresso com o auxílio do *software* Proteus de acordo com a Figura 15 (a) e posteriormente a montagem da mesma. O resultado final da placa desenvolvida é mostrado pela Figura 15 (b).



(a) (b)

Figura 15 – Circuito impresso e placa de alimentação.

Fonte: Do autor, 2011.

Com a placa finalizada foi preciso validar a sua funcionalidade, utilizando um osciloscópio para mensurar os níveis de tensão da placa. Para poder alimentar a

placa, que necessita de uma tensão alternada de 24 Volts, usou-se um transformador de 220 Volts para 24 Volts, podendo assim verificar a tensão fornecida pelos reguladores de tensão contidos na placa auxiliar desenvolvida. A Figura 16 demonstra o gráfico da tensão fornecida pelo regulador LM7824.

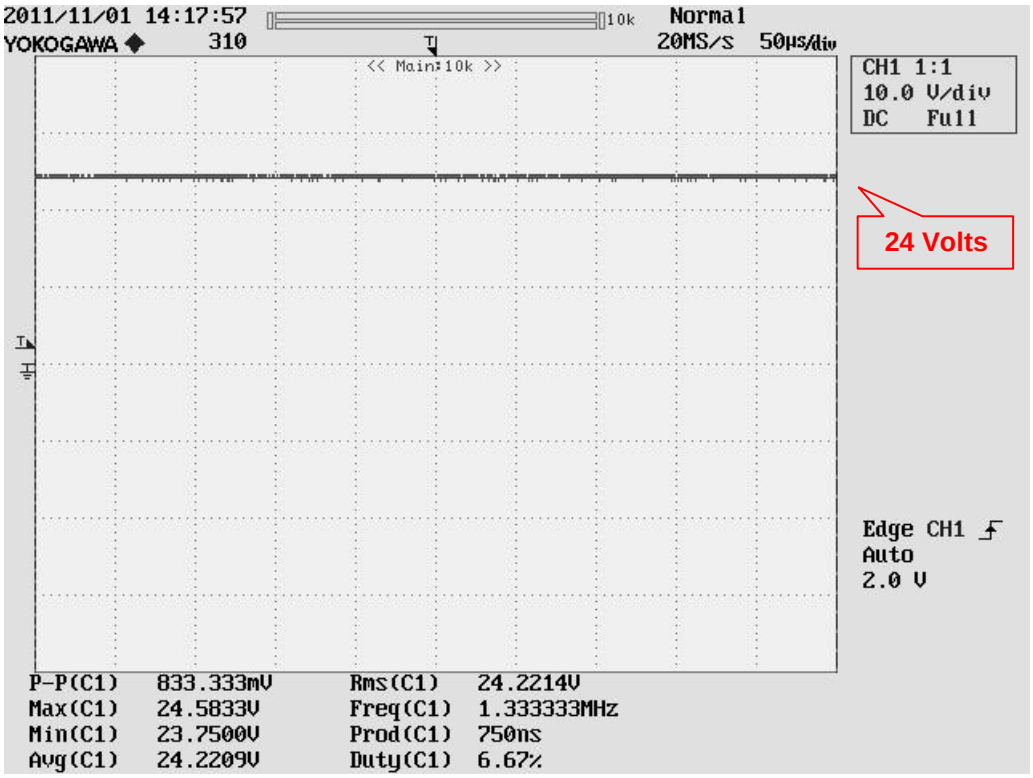


Figura 16 – Gráfico da tensão fornecida pelo regulador LM7824.

Fonte: Do autor, 2011.

O gráfico da figura acima foi retirado do osciloscópio sendo possível identificar a tensão fornecida pelo regulador de tensão LM7824. Este possui uma tensão média de 24.22 Volts CC. Essa tensão regulada será utilizada para alimentar o motor, que possui uma tensão nominal de 24 Volts, sendo assim viável usá-la. Após verificado o valor da tensão do regulador de tensão LM7824, foi mensurado a tensão fornecida pelo regulador de 5 Volts, a Figura 17 demonstra o gráfico da tensão fornecida pelo regulador de tensão LM7805.

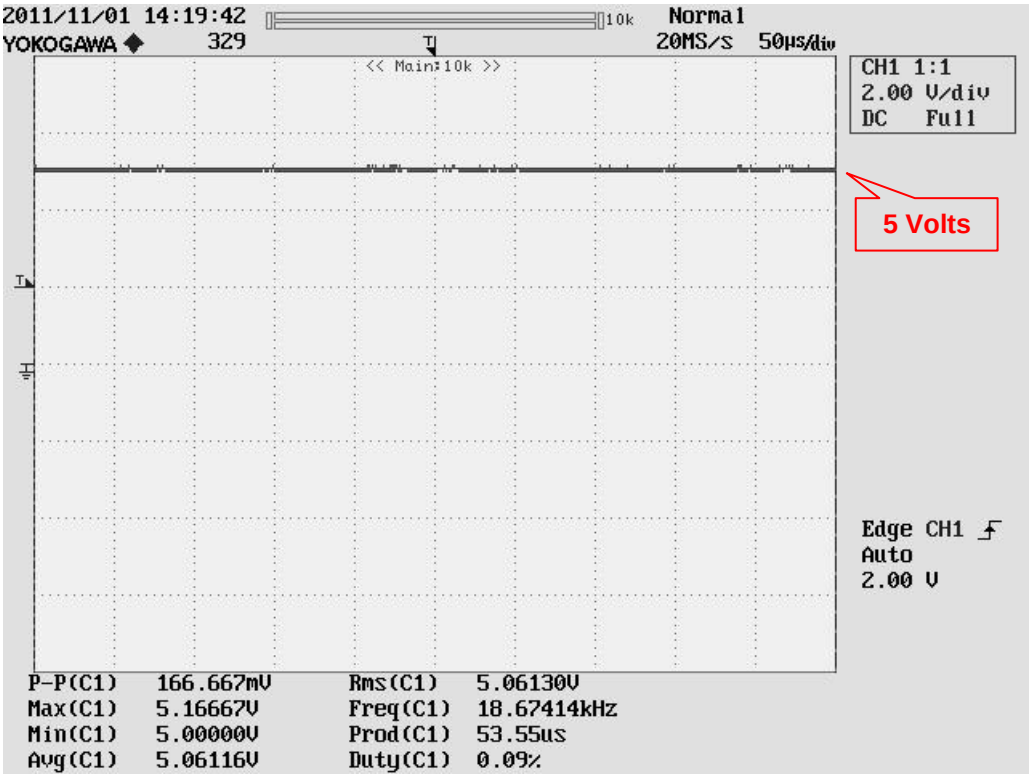


Figura 17 – Gráfico da tensão fornecida pelo regulador LM7805.
Fonte: Do autor, 2011.

O regulador de tensão LM7805 fornece uma tensão média de 5.06 Volts CC conforme o gráfico acima, visto que é um regulador de 5 Volts a sua tensão atende a demanda do processo. A medição e aprovação dos valores de tensão fornecidos pela placa eletrônica auxiliar permitem que o motor seja alimentado e controlado pelo acionamento do transistor, dessa forma o Arduino Uno poderá ligar o motor com o intuito de controlá-lo.

3.2 CONTROLE DO MOTOR UTILIZANDO O ARDUINO

A validação da placa eletrônica auxiliar permitiu a alimentação do motor. Assim foi possível utilizar o Arduino para controlar o motor. Para poder controlar o motor foram realizados diversos testes a fim de aprovar a funcionalidade da placa criada e apontar possíveis erros não previstos no projeto. Cada teste possui um

objetivo específico para controlar o motor e um resultado esperado. Esses testes utilizam o Arduino Uno para controlar o motor consistem em:

- Ligar e desligar o motor;
- Controlar a rotação do motor em malha aberta;
- Controle com realimentação utilizando a leitura da tensão gerada pelo taco gerador;
- Implementação do algoritmo de controle PID;

Primeiramente o foco foi controlar o motor utilizando uma saída digital do Arduino para ligar e desligar o motor, sem utilizar qualquer sinal gerado pelo taco gerador. Assim uma saída digital foi conectada a entrada de controle do motor, que permite ligar o motor com o acionamento do transistor. Para realizar tal função foi desenvolvido um algoritmo de controle na qual uma saída digital era acionada conforme a solicitação, permitindo controlar o motor utilizando o monitor serial que é um recurso do software do Arduino. Uma palavra específica enviada ao Arduino pelo computador permitia ligar ou desligar o motor, assim foi desenvolvido um controle manual, porém digital em malha aberta.

No segundo momento o objetivo foi controlar a rotação do motor utilizando uma saída PWM do Arduino, mas para realizar a escolha da rotação do motor foi utilizado o monitor serial da mesma forma que no primeiro teste. Um novo algoritmo foi desenvolvido e do computador foi possível enviar o valor desejado da velocidade para o Arduino, valores entre 0 e 255, ou seja, valores de 0 a 24 Volts. Assim foi possível desligar o motor com o valor 0 e girar com a velocidade nominal utilizando o valor 255. Conforme solicitado o motor visualmente girava rápido com valores maiores e lento com valores pequenos, parando em 0. Dessa forma foi possível controlar a rotação do motor manualmente. O Arduino controlava o motor, porém ainda permanecia um controle de malha aberta.

Após finalizado o controle em malha aberta o foco foi voltado a desenvolver um controle de malha fechada utilizando o sinal de tensão gerado pelo taco gerador, dessa forma foi utilizado o controle de rotação para verificar a mudança de tensão gerada pelo taco gerador conforme a velocidade do motor.

Assim a saída do taco gerador foi ligada a uma entrada analógica do Arduino. A entrada analógica 0 foi escolhida, mas poderia ser qualquer outra, isso permitiu visualizar a tensão captada pela entrada analógica no monitor serial. A medida que se modificava a rotação do motor notou-se um resultado inesperado,

pois a tensão gerada pelo taco gerador possuía uma enorme interferência, não foi possível visualizar um valor de tensão constante quando o motor girava em uma velocidade constante, mesmo em rotação nominal o valor variava aleatoriamente. Assim uma maneira para minimizar essa influência indesejada foi inserir um filtro passa baixa, permitindo diminuir o ruído captado pela entrada analógica, minimizando a influência das tensões de alta freqüência, resultando em um sinal de tensão mais constante. Optou-se por criar um filtro digital. Esse algoritmo foi desenvolvido no *software* do Arduino e pode ser visualizado nas linhas de código da Figura 18.

```
byte filtro_digital(){
  float n = (float) 0.1; // potencia do filtro passa baixa
  static float filtro; // declara variavel filtro como float estatico
  filtro = n*(analogRead(0)/4) + filtro*(1-n); // equação do filtro passa baixa
  return (byte) filtro;  // retorna valor filtrado
}
```

Figura 18 – Linhas de código do filtro digital.

Fonte: Do autor, 2011.

A implementação do filtro digital permitiu-se uma melhora na resposta do sinal, isso possibilitou o avanço para o desenvolvimento de um controle automático, ou seja, utilizar a realimentação do sinal do taco gerador e criar um controle de malha fechada. Dessa forma o sinal captado pela entrada analógica pode ser analisado a fim de controlar a rotação do motor.

O controle de malha fechada adotado para ser utilizado no experimento foi o controle Proporcional-Integral-Derivativo (PID), para isso houve a necessidade de desenvolver um algoritmo do PID para o Arduino. Utilizando a equação 5 da fundamentação teórica foi possível implementar um algoritmo de controle PID digital. O linhas do código implementado no *software* do Arduino podem ser visualizadas pela Figura 19.

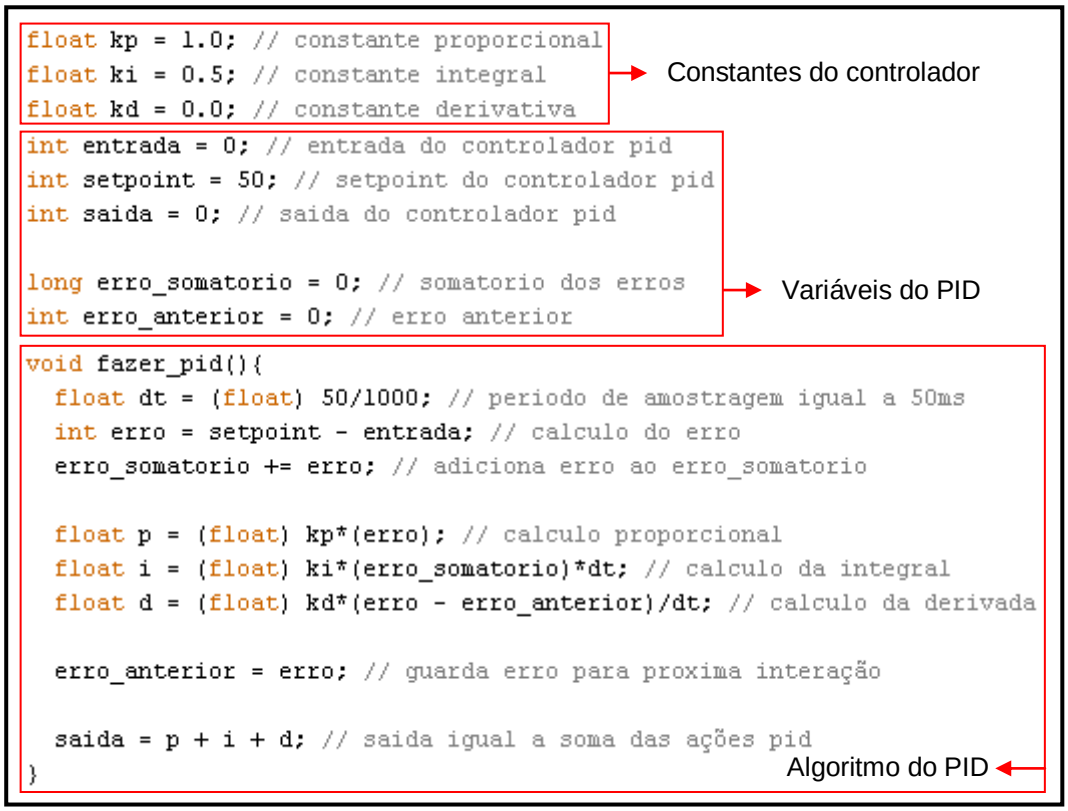


Figura 19 – Linhas de código do algoritmo PID.

Fonte: Do autor, 2011.

Com o algoritmo do controle PID digital pronto, foi possível controlar o motor utilizando o controlador, nele os parâmetros do PID e o *setpoint* eram alterados utilizando o monitor serial. Mesmo com o controlador implementado não foi possível notar que o algoritmo realmente estava funcionando adequadamente. Assim foi desenvolvido um aplicativo em Java para poder visualizar a rotação do motor, ou seja, mostrar em um gráfico o sinal analógico captado pelo Arduino e o *setpoint* escolhido.

A Figura 20 demonstra o aplicativo Java desenvolvido para visualizar a rotação do motor. Esse aplicativo utilizou a porta serial do Arduino para poder enviar valores referentes ao controle PID do motor e também para receber valores do *setpoint* e da entrada analógica do Arduino, mostrando-os na forma de gráfico.

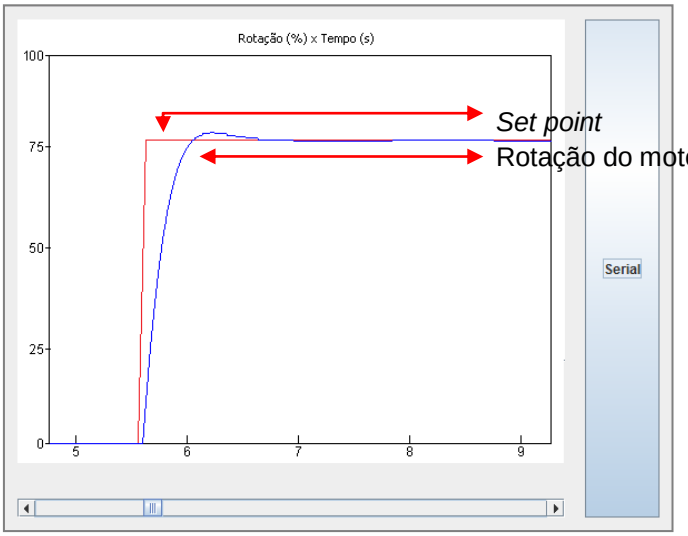


Figura 20 – Aplicativo Java para visualizar a rotação do motor.
Fonte: Do autor, 2011.

Com o auxílio desse aplicativo, desenvolvido em Java, foi possível visualizar no gráfico a resposta da rotação do motor perante o controle PID implementado. Este permiti verificar no gráfico a resposta do sistema, conforme a escolha dos parâmetros do controlar digital. A finalização do controle utilizando o Arduino permitiu o avanço do projeto para a comunicação via rede.

3.3 COMUNICAÇÃO VIA INTERNET

A etapa de comunicação via internet necessitou um vasto tempo de pesquisa relacionada a biblioteca do *Ethernet Shield*, utilizada para compreender o processo de comunicação entre o Arduino e o *Shield*, pois essa possui exemplos e comentários que ajudaram a entender a o funcionamento da comunicação entre as placas.

A comunicação entre as placas Arduino Uno e *Ethernet Shield* é feita por SPI e possuem respectivamente os chips Atmega328 e ENC28J60. O chip ENC28J60 pode ser comparado com um grande buffer, ou seja, guarda pacotes recebidos da rede. Também guarda pacotes para serem enviados, assim o buffer é

dividido em duas partes, uma para enviar e outra para receber dados da rede. Somente o chip *Ethernet* não faz a comunicação com a rede. É necessário um processador para analisar os pacotes e saber o que fazer com eles. O chip Atmega328 faz o processamento dos pacotes se utilizar a biblioteca do *Ethernet* a qual possui funções específicas para diversos tipos de comunicação como TCP e UDP, permitindo identificar o protocolo e responder o pacote recebido.

Essa biblioteca possui alguns exemplos, um deles permite criar um *web site* simples. Este foi estudado e utilizado para criar uma página *web* para controlar o motor. A página permite ligar e desligar o motor conforme mostra a Figura 21. O botão permite ligar ou desligar o motor. Um fator limitante é que o código dessa página é gravado internamente no chip Atmega328, assim como a biblioteca do *Ethernet.* Ela envia um único pacote de dados, contendo todo código da página.

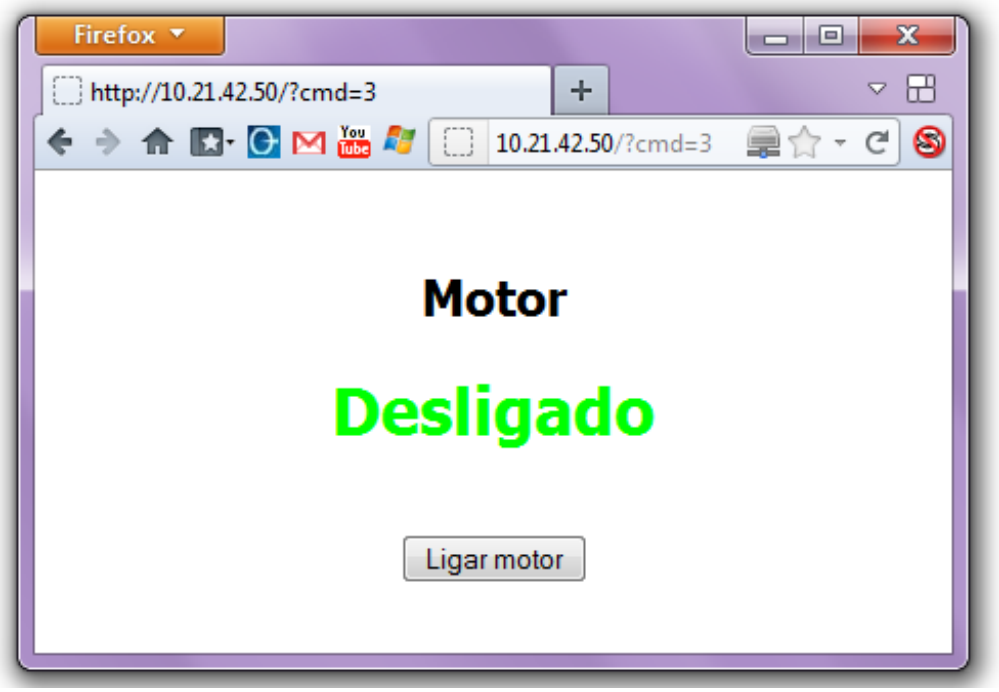


Figura 21 – Página *Web* para ligar e desligar o motor.
Fonte: Do autor, 2011.

Outro fator importante que limita o tamanho da página é que a biblioteca cria um buffer dentro do Arduino para guardar o pacote enviado e recebido. Nos exemplos da biblioteca é utilizado um buffer de 500 Bytes. O tamanho desse buffer é

limitado pela SRAM do Atmega328 que é de 2000 Bytes, dessa forma limita o tamanho do buffer e da página *web*.

Considerando que o tamanho máximo do pacote é limitado pelo buffer, optou-se por utilizar uma estratégia diferente, ao invés de desenvolver uma página *web* para controlar o motor, foi criado um aplicativo na linguagem Java para estabelecer uma conexão com o Arduino. O aplicativo desenvolvido estabelece uma conexão com o Arduino utilizando o protocolo de comunicação TCP, para isso foi utilizado um recurso da linguagem Java chamado *socket.* A biblioteca *socket* permite abrir uma conexão com um determinado IP e porta de acesso, assim foi possível enviar e receber dados do Arduino pela internet. A Figura 22 demonstra o aplicativo criado para se comunicar com o Arduino, que é similar ao monitor serial, porém se comunica com o Arduino via Ethernet.

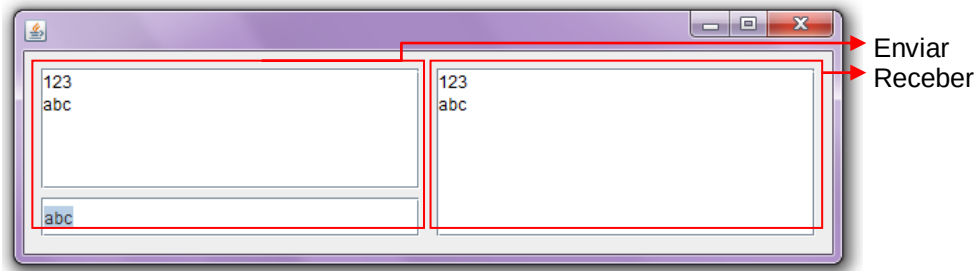


Figura 22 – Aplicativo para comunicação TCP.

Fonte: Do autor, 2011.

Semelhante ao monitor serial do *software* do Arduino, esse aplicativo possui uma janela para enviar dados e outra para receber. Conforme a figura acima é destacado nas janelas: um bloco demonstra os dados enviados e o outro os dados recebidos do Arduino. A Figura 22 mostra as mensagens enviadas e recebidas que são exatamente iguais, pois o algoritmo do Arduino foi programado para realizar um eco, ou seja, todo pacote de dados enviados ao Arduino serão retornados de volta para quem o enviou. As linhas do algoritmo para realizar essa função podem ser visualizada pela Figura 23.

```
void setup(){
  server_init(); // inicia o chip enc28j60
}

void loop(){
  if (client_receive()){ // caso recebeu algum pacote de dados
    while(client_available()){ // enquanto tiver dados para ler
      client_write(client_read()); // le o dado e escreve no buffer
    }
    client_send(); // envia o pacote
  }
}
```

Figura 23 – Linhas de código da comunicação pela rede.

Fonte: Do autor, 2011.

O algoritmo desenvolvido tornou-se simples, pois foi criado um arquivo para simplificar a biblioteca. Esse arquivo é adicionado junto ao código principal do Arduino, que inclui a biblioteca do *Ethernet Shield* e outras funções que simplificam a mesma. Todas as configurações estão presentes nesse arquivo, como configuração do MAC, IP e porta de acesso do *Shield.* Nele também pode ser modificado o tamanho do buffer do Arduino, que está configurado para 500 Bytes. O arquivo pode ser visualizado no Anexo A.

A implementação da comunicação entre o Arduino e o aplicativo Java permitiu o envio de mensagens específicas ao Arduino via internet. A Figura 24 demonstra a comunicação pela internet entre união das placas do Arduino e o aplicativo de comunicação TCP, que permite o envio de mensagens pelo usuário ao Arduino via internet. Possibilitando que o usuário troque informações com o Arduino a fim de controlar os dispositivos ligados ao Arduino.

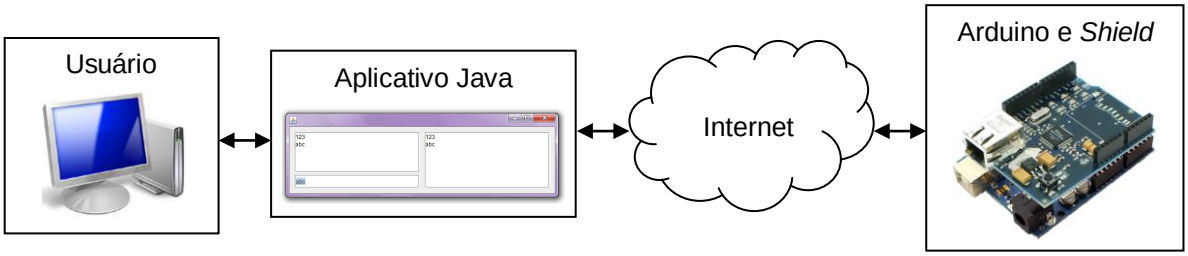


Figura 24 – Diagrama de blocos da comunicação pela internet.

Fonte: Do autor, 2011.

Conforme a figura acima o aplicativo faz a ligação entre o usuário e o experimento utilizando a internet para ligá-los. Dessa forma é possível criar um aplicativo específico, sendo esse utilizado para controlar o experimento remoto. Assim qualquer interface do aplicativo Java poderia enviar uma mensagem específica para o Arduino. Um botão, por exemplo, permite enviar essa mensagem, que para o Arduino significaria ligar o motor. Portanto, o usuário desconhece os comandos que são válidos no código do Arduino, tornando o sistema seguro.

3.4 DESENVOLVIMENTO DO AMBIENTE VIRTUAL

Finalizada a comunicação do Arduino via internet os esforços foram voltados à criação de um ambiente de interação entre o usuário e o experimento remoto. O ambiente virtual foi desenvolvido na linguagem Java, sendo esse o local utilizado para criar uma conexão entre o usuário e o experimento remoto. O usuário poderá interagir com o experimento utilizando comandos disponíveis no aplicativo, permitindo realizar a experiência no ambiente virtual utilizando a internet como meio de comunicação para receber informações do experimento e controlá-lo.

O ambiente desenvolvido em Java será utilizado especificamente para comunicação com o experimento remoto via internet ou rede local. Esse aplicativo Java é um *applet*, e permite ser hospedado em uma página *web* similar a uma figura. Assim esse aplicativo se auto-instala no computador do usuário, permitindo utilizar todos os recursos do aplicativo dentro do próprio navegador.

O aplicativo *applet* desenvolvido foi inserido em uma página conforme a Figura 25. Nessa página será possível visualizar o motor, pois será utilizado uma *webcam* que permite o usuário visualizar a rotação do motor conforme é solicitada ao aplicativo. Esse permite ligar e desligar o motor, bem como modificar os parâmetros do controlador PID e visualizar no gráfico a rotação do motor.

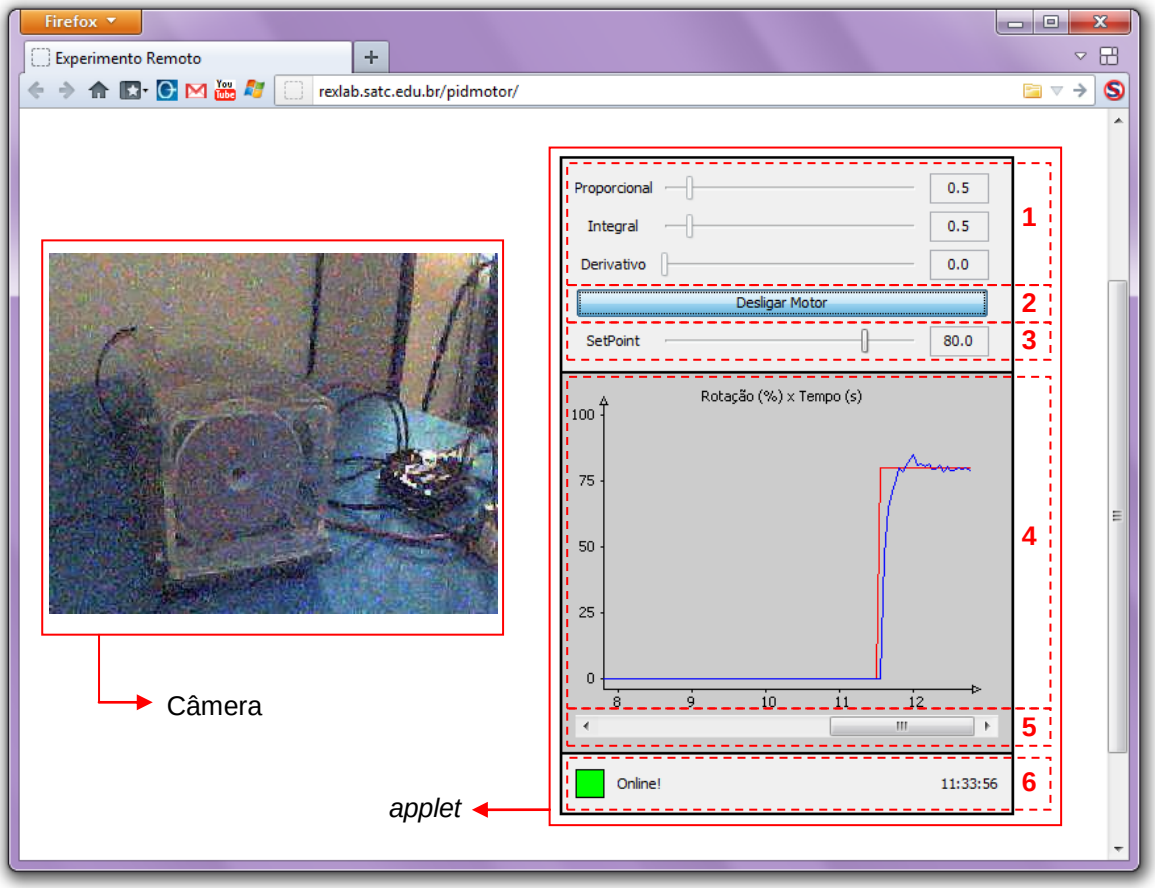


Figura 25 – Página com o aplicativo *applet*.

Fonte: Do autor, 2011.

A página *web* acima demonstra o local onde será possível visualizar o motor, a fim de verificar se realmente o motor gira. Ao lado o aplicativo *applet* que proporciona ao usuário visualizar no gráfico a rotação do motor e controlar os parâmetros do controlador PID. Esse aplicativo possui diversos mecanismos para interagir com o usuário, as funcionalidades de cada item apontado na Figura 25 são descritas na Tabela 4.

O *applet* desenvolvido possui a habilidade de se comunicar com o experimento pela rede, a interação entre o algoritmo do Arduino e o aplicativo permite controlar o motor e receber informações do mesmo. O aplicativo desenvolvido possui uma lógica de controle que é demonstrada pelo fluxograma da Figura 26.

Tabela 4 – Descrição dos blocos do ambiente desenvolvido.

Bloco	Nome	Finalidade
1	Parâmetros do controlador PID	Selecionar os valores dos parâmetros do controlador PID, ou seja, os valores das constantes, proporcional, integral, e derivativa.
2	Botão liga desliga	Ligar ou desligar o motor.
3	Seleção do *setpoint*	Escolher o valor desejado da rotação do motor onde o controlador deve atuar.
4	Gráfico da rotação	Visualizar os valores de rotação pelo tempo do motor e do *setpoint*, identificados respectivamente pela cor azul e vermelha.
5	Barra do gráfico	Paralisar o gráfico para visualizar valores anteriores de rotação.
6	*Status* do aplicativo	Mostrar o horário e mensagens do funcionamento do aplicativo.

Fonte: Do autor, 2011

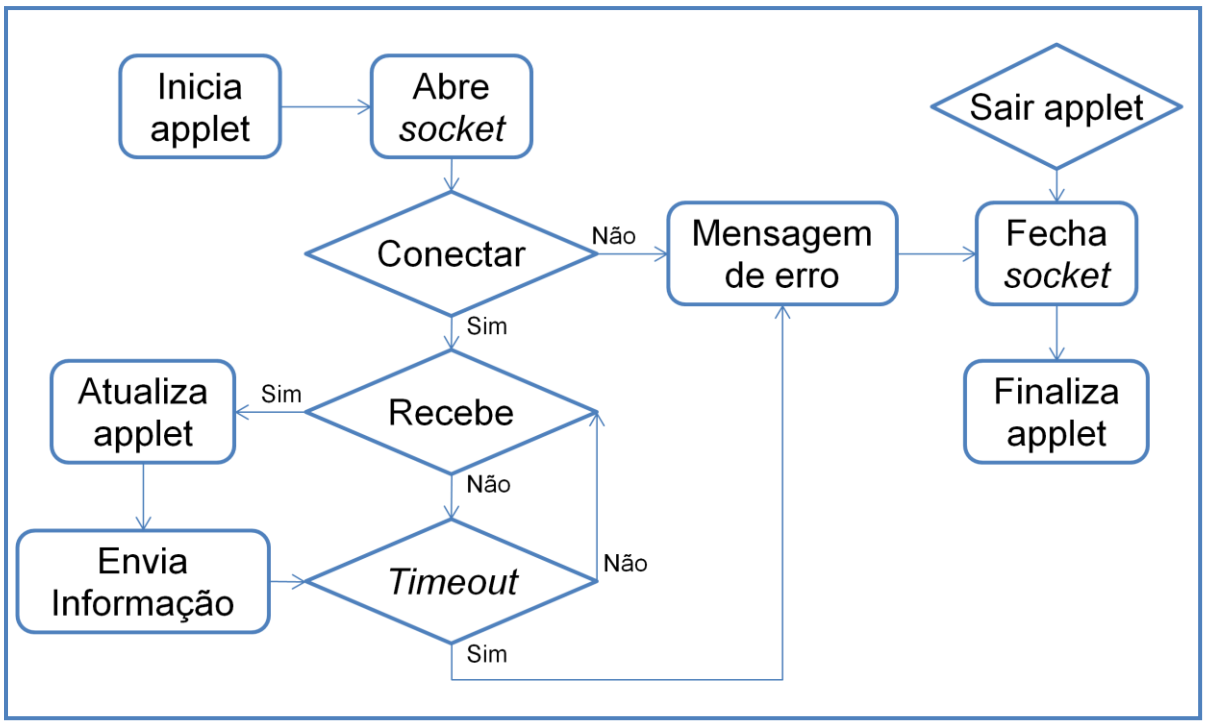


Figura 26 – Fluxograma do aplicativo *applet*.

Fonte: Do autor, 2011

O fluxograma da Figura 26 demonstra o funcionamento do *applet.* Ao entrar na página *web* o aplicativo é executado no computador do usuário e inicia os parâmetros e variáveis do mesmo, abre uma conexão TCP utilizando o método *Socket* da linguagem Java, conforme o Anexo B. Caso a conexão não seja estabelecida uma mensagem de erro é demonstrada no bloco de *Status* do aplicativo e o mesmo é finalizado.

Com a conexão estabelecida com o Arduino o *applet* entra em um *loop,* se receber alguma informação do Arduino o aplicativo é atualizado, principalmente o gráfico e são enviadas informações a placa. O *timeout* é um tempo definido utilizado para desconectar e finalizar o aplicativo quando este ficar um tempo sem receber mensagem da rede, caso ocorra, outra mensagem de erro é exibida no bloco de *Status*. Caso o usuário saia da pagina que contem o aplicativo, esse é finalizado permitindo fechar a conexão TCP.

Essa lógica empregada no aplicativo é funcional, pois o Arduino possui uma lógica que responde adequadamente ao fluxograma do aplicativo. A seguir a Figura 27 demonstra o fluxograma do funcionamento do Arduino. O algoritmo principal desenvolvido nessa placa pode ser visualizado no Anexo C.

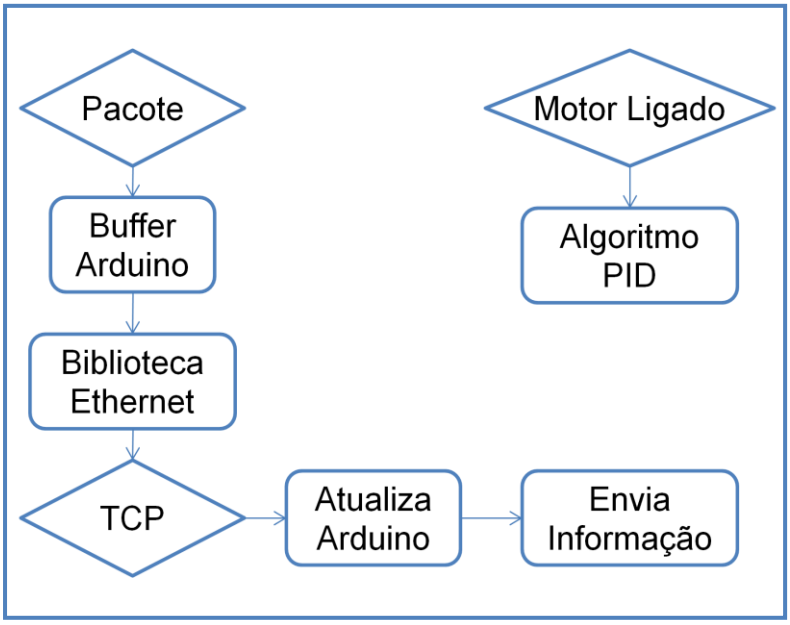


Figura 27 – Fluxograma do Arduino.

Fonte: Do autor, 2011

Conforme fluxograma do Arduino (Figura 27), o mesmo fica em um *loop* verificando se o motor está ligado e se possui alguma informação da rede para ser analisada. Se possuir alguma pacote de rede no *Ethernet Shield*, o pacote é copiado para um buffer interno no Arduino. Esse pacote é analisado pela biblioteca do *Ethernet* que permite identificar e responde os dados da rede. Caso o protocolo envolvido for o TCP, o mesmo é analisado para atualizar o Arduino, que modifica os parâmetros do controlador e *status* do motor. Depois de atualizar os parâmetros, são enviadas informações do motor para o aplicativo pela rede.

Com a comunicação estabelecida entre o Arduino e o aplicativo permitiu a instalação do experimento no laboratório remoto da SATC, assim uma página *web* foi criada, conforme a Figura 25. Dessa forma permitiu que o experimento remoto fosse acessado de locais fora da instituição, utilizando a rede internet para acessar a página e poder utilizar o aplicativo *applet* nela. Permitindo o controle da rotação do motor remotamente, sendo a *webcam* inserida permite que o usuário visualize o motor girando.

4 ANÁLISE DOS RESULTADOS

Com o experimento remoto finalizado foi possível visualizar no gráfico os valores da rotação do motor a partir da mudança dos valores dos ganhos do controlador PID e *setpoint*. A Tabela 5 demonstra os valores dos parâmetros utilizados no aplicativo Java para realizar testes no experimento remoto.

Tabela 5 – Testes realizados no experimento.

Setpoint	60%			Figura
Controlador	Kp	Ki	Kd	
P	1.0			28a
	5.0			28b
PI	5.0	0.5		29a
	5.0	2.0		29b
PID	5.0	0.5	0.5	30a
	5.0	2.0	0.5	30b
	5.0	0.5	2.0	30c
	5.0	2.0	2.0	30d

Setpoint	80%			Figura
Controlador	Kp	Ki	Kd	
P	1.0			28c
	5.0			28d
PI	5.0	0.5		29c
	5.0	2.0		29d
PID	5.0	0.5	0.5	31a
	5.0	2.0	0.5	31b
	5.0	0.5	2.0	31c
	5.0	2.0	2.0	31d

Fonte: Do autor, 2011.

Os diversos testes realizados conforme a tabela acima permitiu verificar a resposta da rotação do motor. A seguir são mostradas as figuras de cada teste realizado relacionando os valores dos ganhos do controlador PID com as figuras na referida tabela.

Com os devidos testes realizados foi possível verificar o funcionamento do controlador, permitindo assim o funcionamento adequado da ação de controle proporcional, integral e derivativo. Posteriormente será utilizado o *software* MATLAB para realizar simulações, comparando os gráficos gerados pelo aplicativo e os gráficos simulados.

A seguir a Figura 28 demonstra a resposta do motor quando utilizado um controlador proporcional puro, com valores diferentes de ganho proporcional e de *setpoint*.

A Figura 28 abaixo apresenta um controlador proporcional, conforme a Tabela 5 com alterações no valor do ganho proporcional e do *setpoint.* Por ser um controlador somente proporcional, não anula o erro em regime permanente, mantendo a dinâmica descrita na teoria de controle. Ou seja, possui uma diferença visível entre o valor da rotação do motor e do *setpoint*.

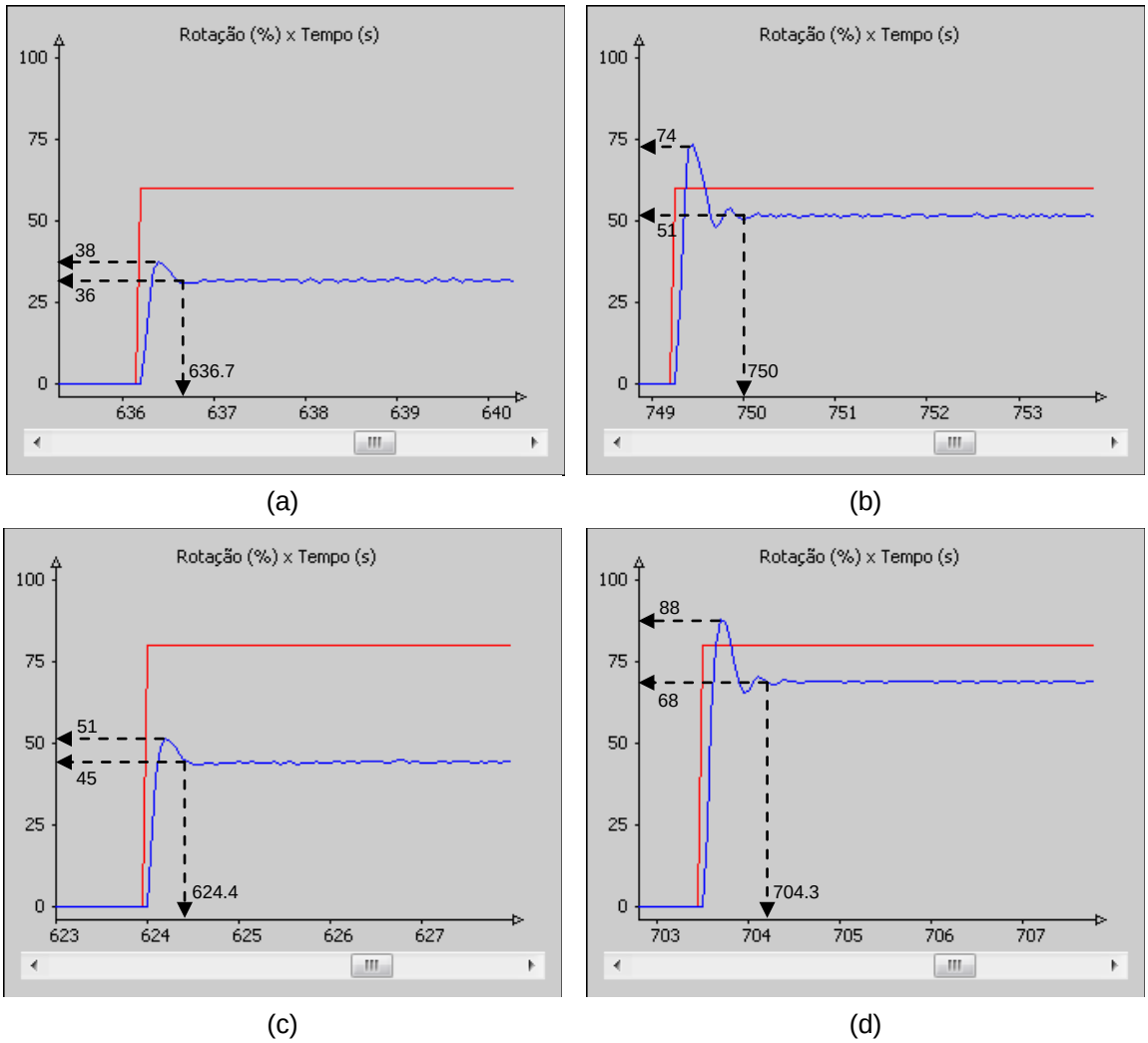


Figura 28 – Aplicativo Java com controlador P.

Fonte: Do autor, 2011.

A seguir são mostrados gráficos do aplicativo utilizando um controlador proporcional e integral, podendo demonstrar o desempenho desse controlador na Figura 29.

Com a inclusão do controlador integral, visualizado pela Figura 29, é possível verificar a sua atuação no controle da rotação do motor. O ganho

proporcional permanece constante, somente os valores do *setpoint* e do ganho integral são modificados. É visível que a adição do controlador integral ao proporcional permite levar o erro em regime permanente ao nulo, porém um ganho elevado da constante integral aumenta o valor de sobre sinal e a oscilação do sistema. O aumento do *setpoint* faz com que o sistema fique menos instável, diminuindo a oscilação no regime transitório.

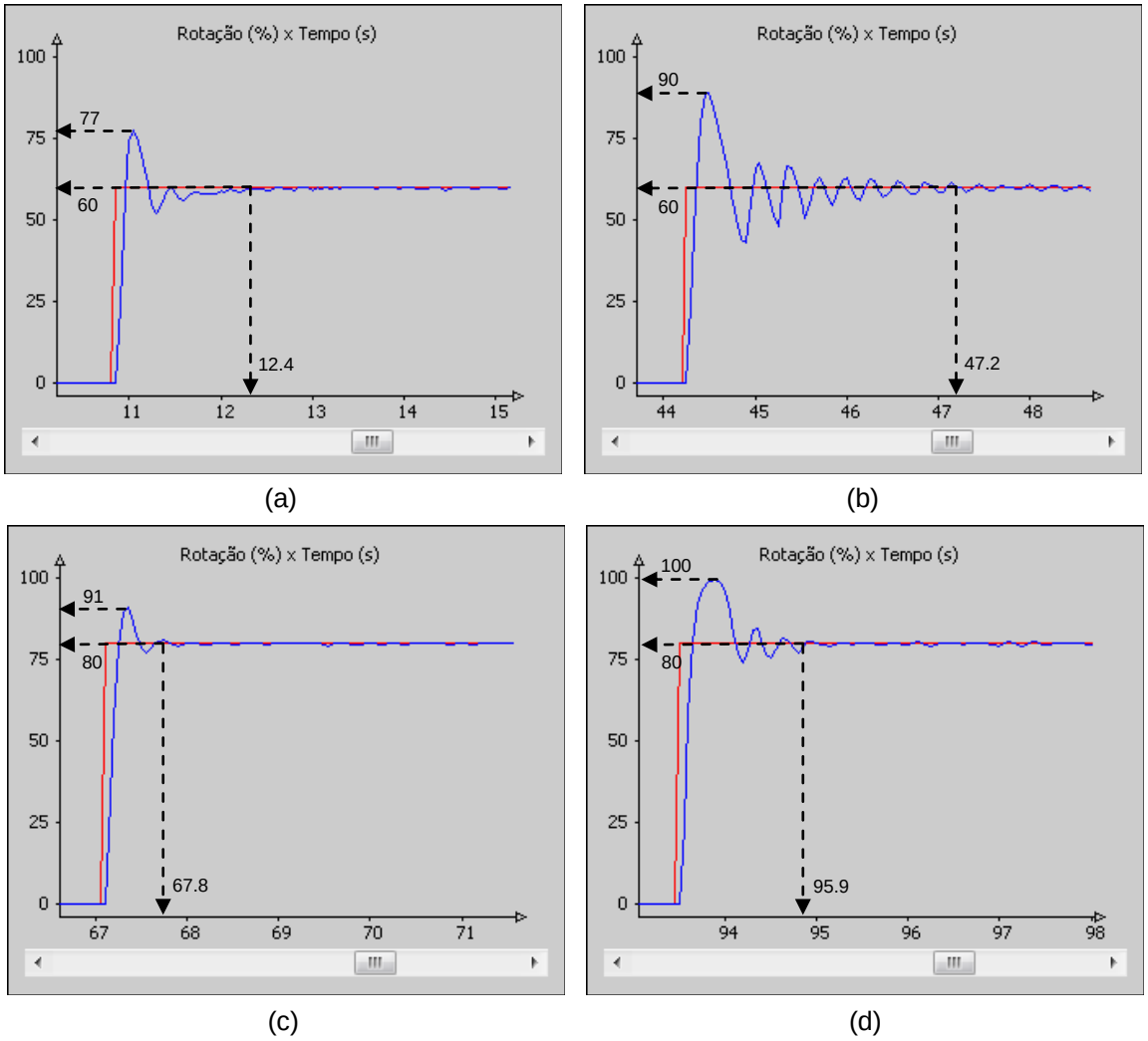


Figura 29 – Aplicativo Java com controlador PI.

Fonte: Do autor, 2011.

A seguir os gráficos do controlador PID são demonstrados pela Figura 30 e Figura 31, respectivamente utilizando um valor de *setpoint* de 60% e 80%. A Figura 30 demonstra o controlador PID quando utilizado um valor de *setpoint* de 60%, nesses gráficos são alterados os valores do ganho integral e derivativo,

enquanto o valor do ganho proporcional permanece constante. A atuação do controlador derivativo permite diminuir a oscilação do sistema e com o ganho integral elevado o sistema tende a se tornar instável.

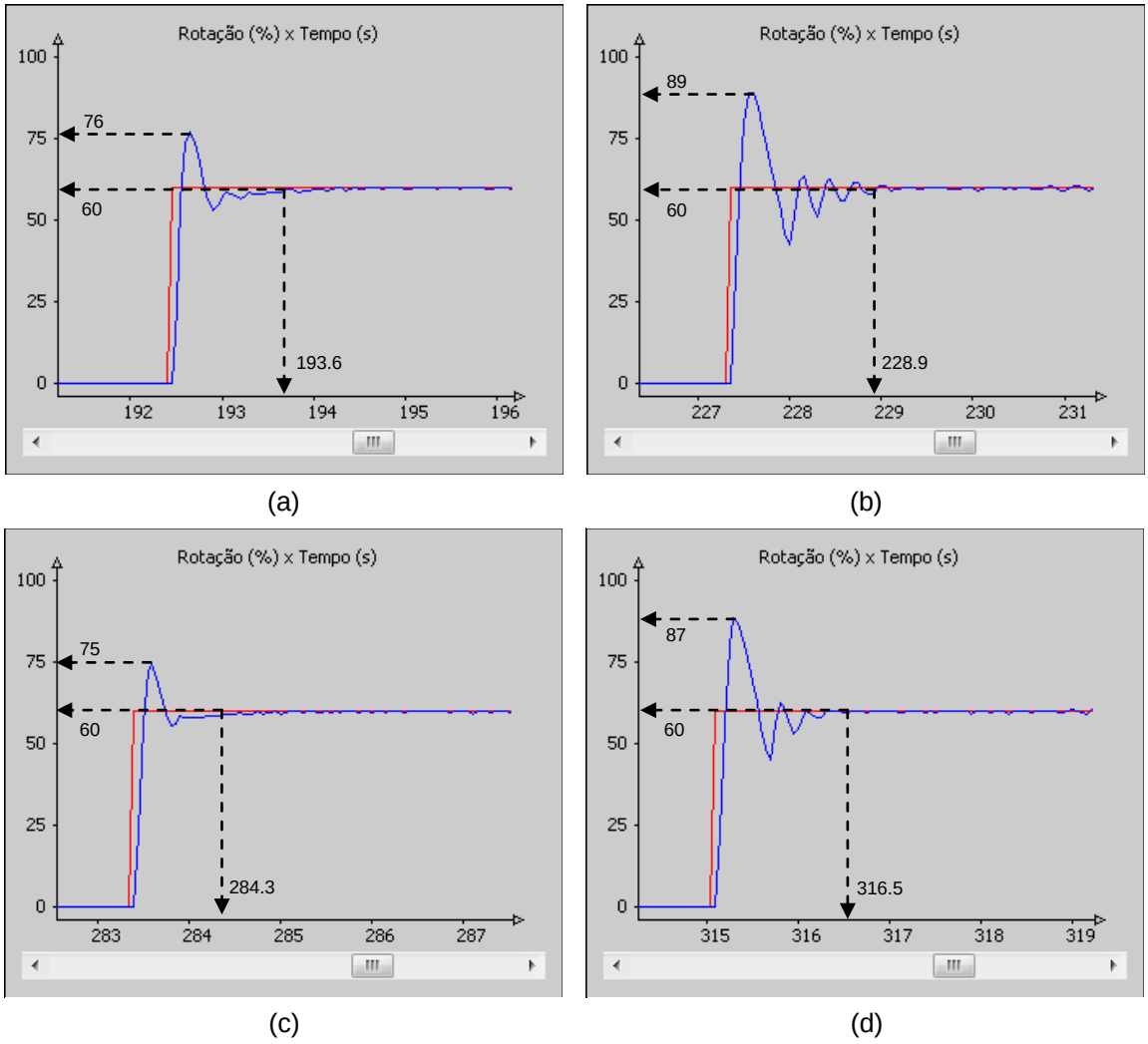


Figura 30 – Aplicativo Java com controlador PID – 60%.

Fonte: Do autor, 2011.

A Figura 31 é semelhante à Figura 30, porém possui um valor de *setpoint* superior com valor de 80%. A seguir a Figura 31 demonstra os gráficos do controlador PID em um *setpoint* de 80%. Pode-se comparar essas figuras, pois as mesmas possuem os valores de ganhos proporcional, integral e derivativo iguais. O aumento do valor de referência permite diminuir a oscilação do sistema tornando-o mais estável com valores maiores de *setpoint.*

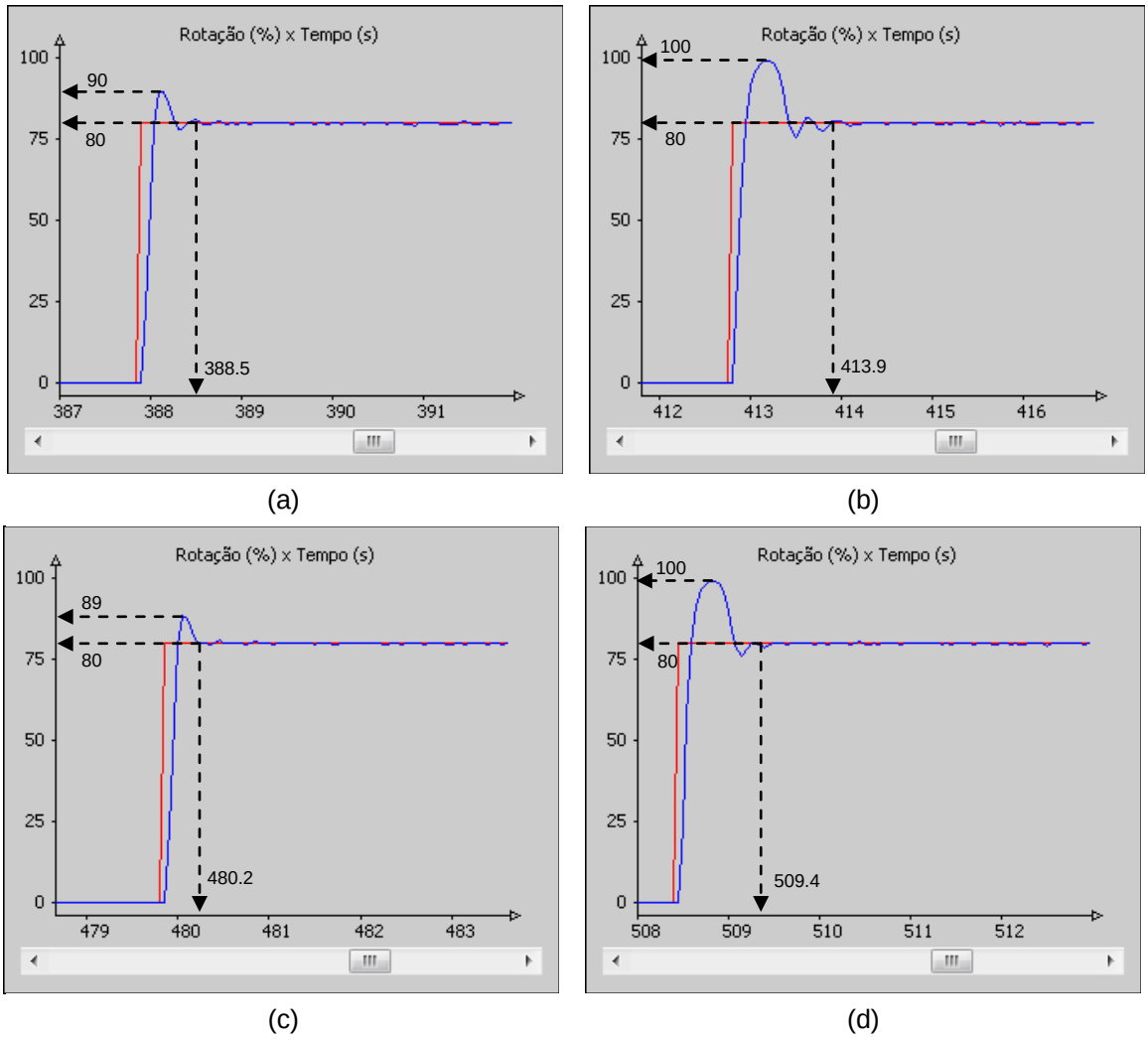


Figura 31 – Aplicativo Java com controlador PID – 80%.

Fonte: Do autor, 2011.

Visualizando os resultados gerados pelo gráfico do aplicativo desenvolvido na linguagem de programação Java, pode-se compará-lo a uma resposta de um sistema simulado. As simulações foram realizadas para avaliar o desempenho do algoritmo PID implementado no Arduino e comparar a resposta dos sistemas, real e simulado.

4.1 SIMULAÇÕES NO MATLAB

Para poder realizar simulações foi utilizado o *software* MATLAB. Para isso foi preciso identificar a função de transferência do experimento, utilizando assim o aplicativo Java, que possibilitou monitorar a resposta da rotação do motor em malha aberta, permitindo a utilização de métodos para modelar a função de transferência do sistema a partir do gráfico. A Figura 32 a seguir demonstra a resposta do sistema em malha aberta.

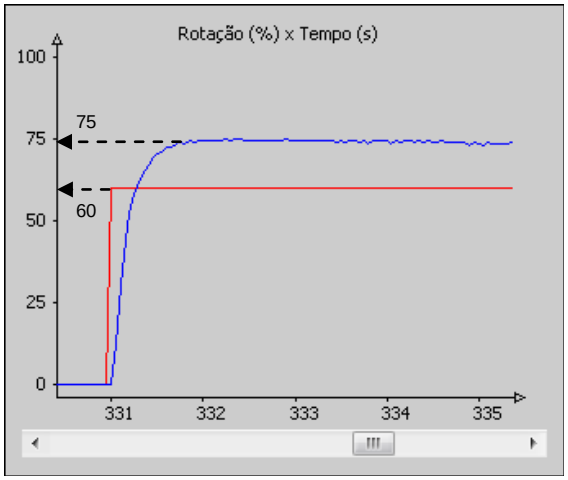


Figura 32 – Aplicativo Java com controlador em malha aberta.
Fonte: Do autor, 2011.

Com o gráfico obtido da rotação do motor referente a uma entrada de 60% da rotação máxima do motor, utilizou-se análise do gráfico para encontrar os modelos matemáticos que representem a resposta do sistema, permitindo identificar as respectivas funções de transferência. O método de Hägglund encontrado no Anexo D, modela uma função de transferência de primeira ordem e o método de Mollenkamp encontrado no Anexo E, uma função de segunda ordem.

As funções de transferência modeladas a partir do gráfico da Figura 32 são mostradas a seguir, representadas pela função de primeira ordem (H1) e de segunda ordem (H2).

$$H1(s) = \frac{1.25}{0.2 * s + 1} \qquad H2(s) = \frac{1.25}{0.00216 * s^2 + 0.1753 * s + 1}$$

Com o auxílio do *software* MATLAB foi simulado as duas funções de transferência simultaneamente, assim foi possível simular diversos tipos de controladores. A seguir a Figura 33 demonstra a simulação das funções de transferência, nela é possível visualizar o *setpoint,* H1 e H2, respectivamente pelas cores azul, vermelho e verde.

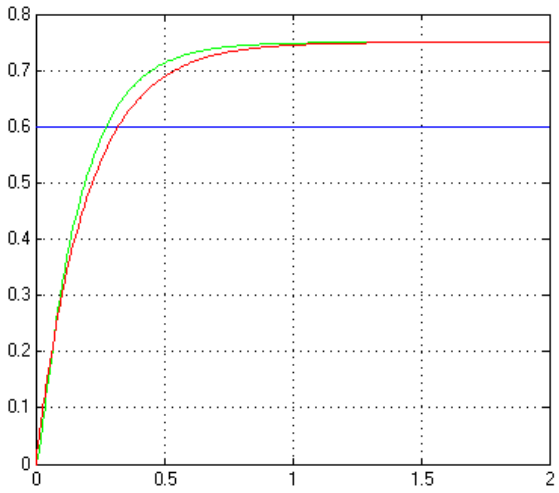


Figura 33 – Simulação das funções de transferência em malha aberta.
Fonte: Do autor, 2011.

Para comparar o sistema real com o modelado foi simulado diversos controladores utilizando as funções de transferência obtidas a partir do gráfico em malha aberta. Os testes simulados podem ser visualizados pela Tabela 6 que é similar a Tabela 5, assim pode-se comparar os gráficos da resposta do sistema real com a simulação.

A Figura 34, Figura 35, Figura 36 e Figura 37 são gráficos gerados pela simulação dos modelos matemáticos e representam sistemas de malha fechada com um controlador PID. Os parâmetros do controlador e o *setpoint* podem ser visualizados pela Tabela 6 a seguir.

Tabela 6 – Testes simulados.

Setpoint	60%			Figura
Controlador	Kp	Ki	Kd	
P	1.0			34a
	5.0			34b
PI	5.0	0.5		35a
	5.0	2.0		35b
PID	5.0	0.5	0.5	36a
	5.0	2.0	0.5	36b
	5.0	0.5	2.0	36c
	5.0	2.0	2.0	36d

Setpoint	80%			Figura
Controlador	Kp	Ki	Kd	
P	1.0			34c
	5.0			34d
PI	5.0	0.5		35c
	5.0	2.0		35d
PID	5.0	0.5	0.5	37a
	5.0	2.0	0.5	37b
	5.0	0.5	2.0	37c
	5.0	2.0	2.0	37d

Fonte: Do autor, 2011.

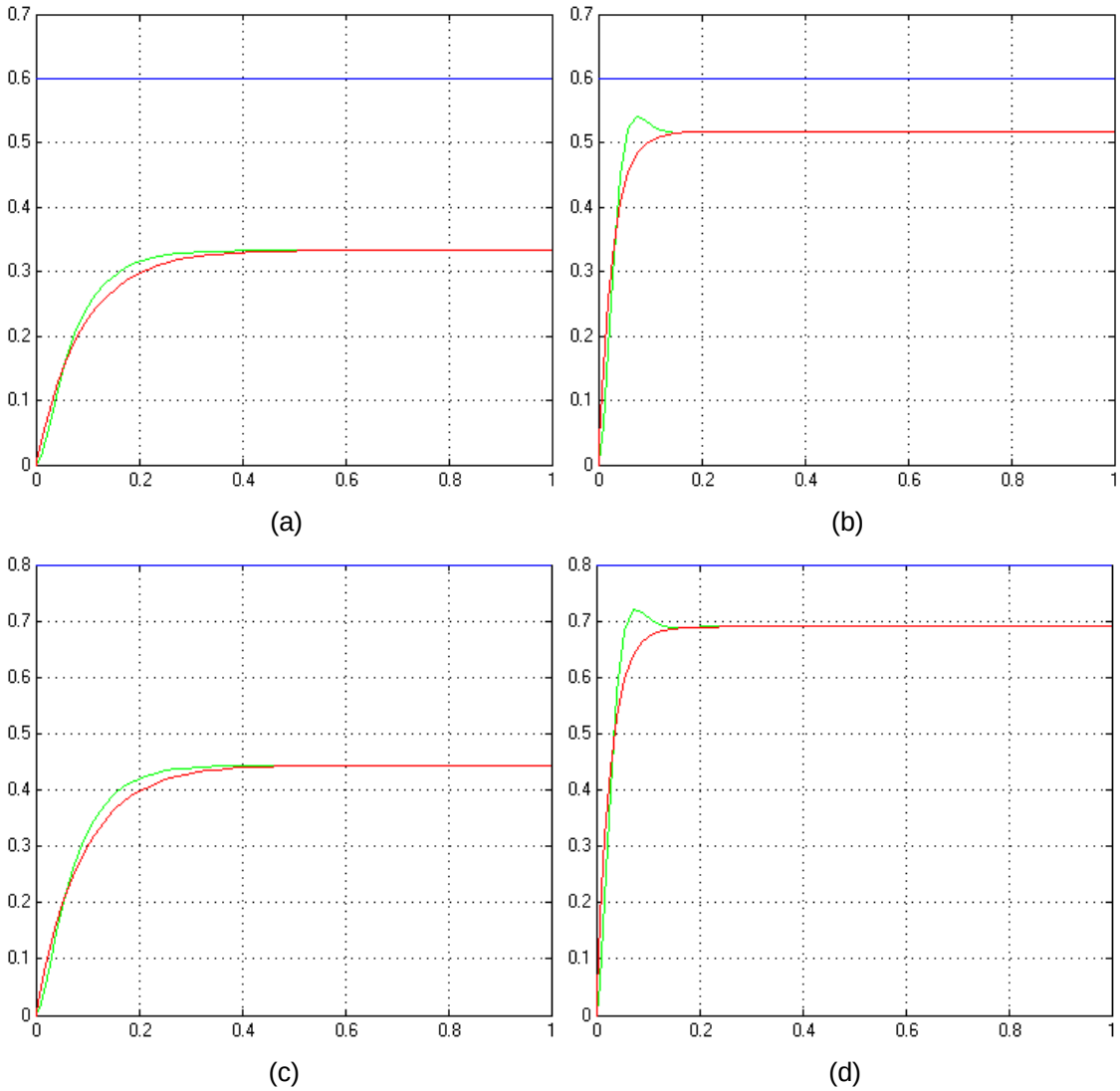


Figura 34 – Simulação do controlador P.

Fonte: Do autor, 2011.

A Figura 34 demonstra um controlador proporcional, nele é possível visualizar a diferença da função de transferência de primeira ordem e segunda ordem. Quando a constante proporcional é elevada, verifica-se o surgimento de um sobre sinal.

Comparando o grupo da Figura 28 e Figura 34 é possível visualizar que a resposta real possui um sobre sinal semelhante a função de segunda ordem, assim essa função se assemelha mais com a resposta real da rotação do motor, mesmo que as duas funções de transferência em regime permanente resultem no mesmo valor de rotação. A seguir o grupo da Figura 35 representa a simulação de um controlador proporcional e integral.

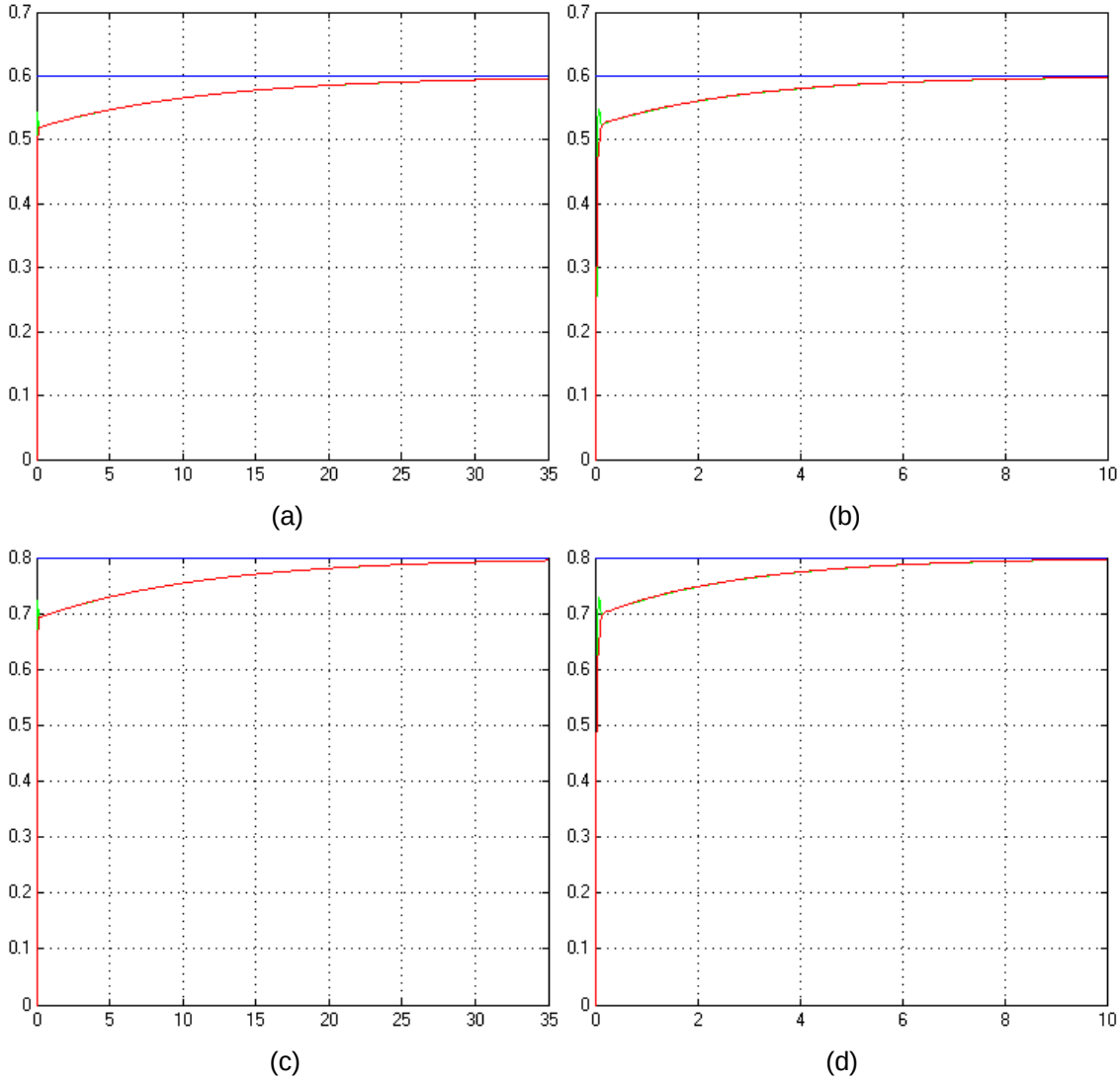


Figura 35 – Simulação do controlador PI.

Fonte: Do autor, 2011.

A adição do controlador integral pode ser visualizada pela Figura 35. A inclusão do controlador integral fez com que o erro em regime permanente se anule. Essa figura mostra o controlador PI simulado modificando o valor do ganho integral e permitindo que o sistema se estabilize mais rapidamente com o aumento desse ganho. Conforme visto na Figura 29 o sistema real também elimina o erro em regime permanente com a inclusão do controlador integral. O controlador PID simulado é demonstrado pela Figura 36 e Figura 37, respectivamente 60% e 80% da rotação.

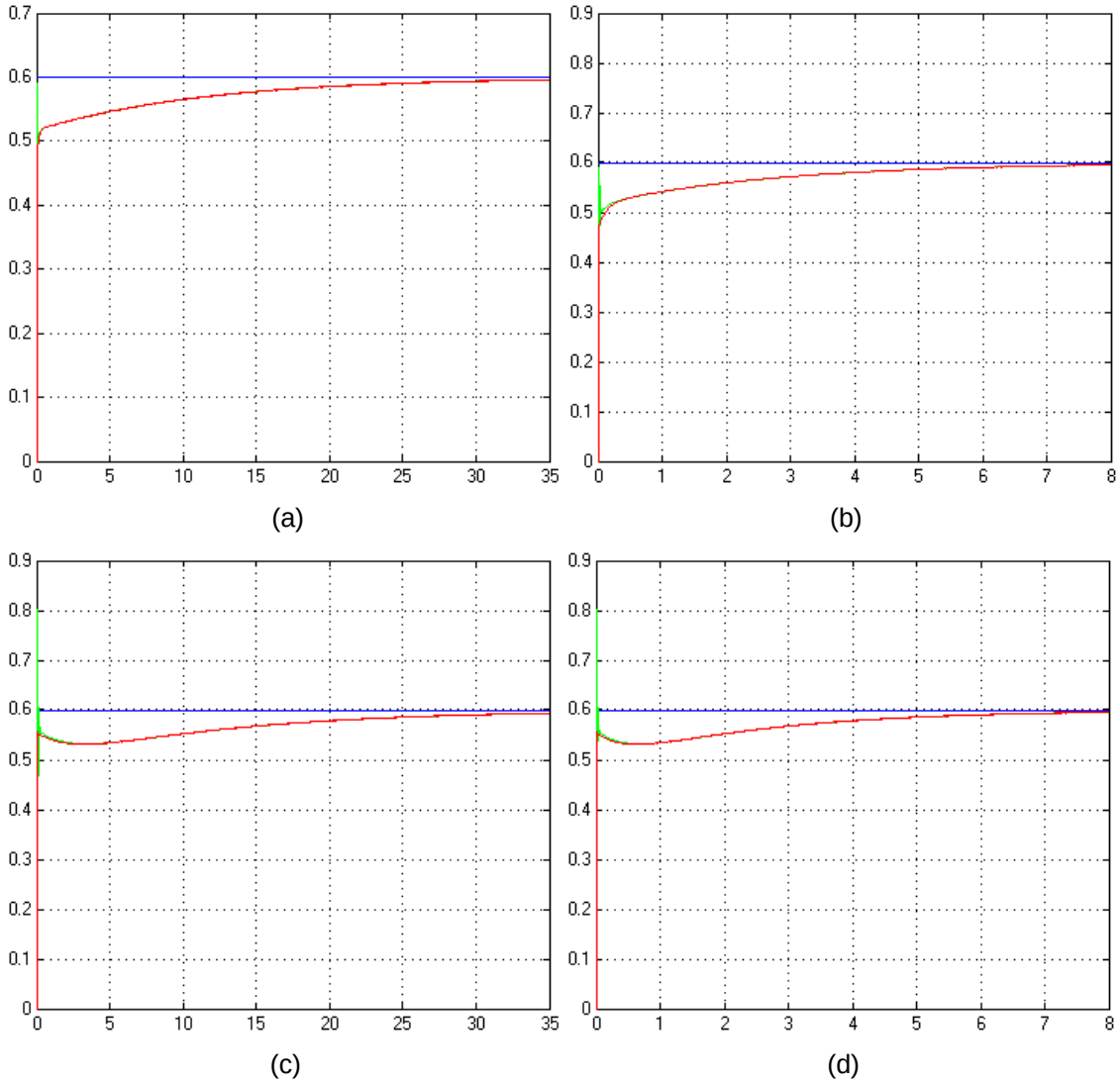


Figura 36 – Simulação do controlador PID – 60%.

Fonte: Do autor, 2011.

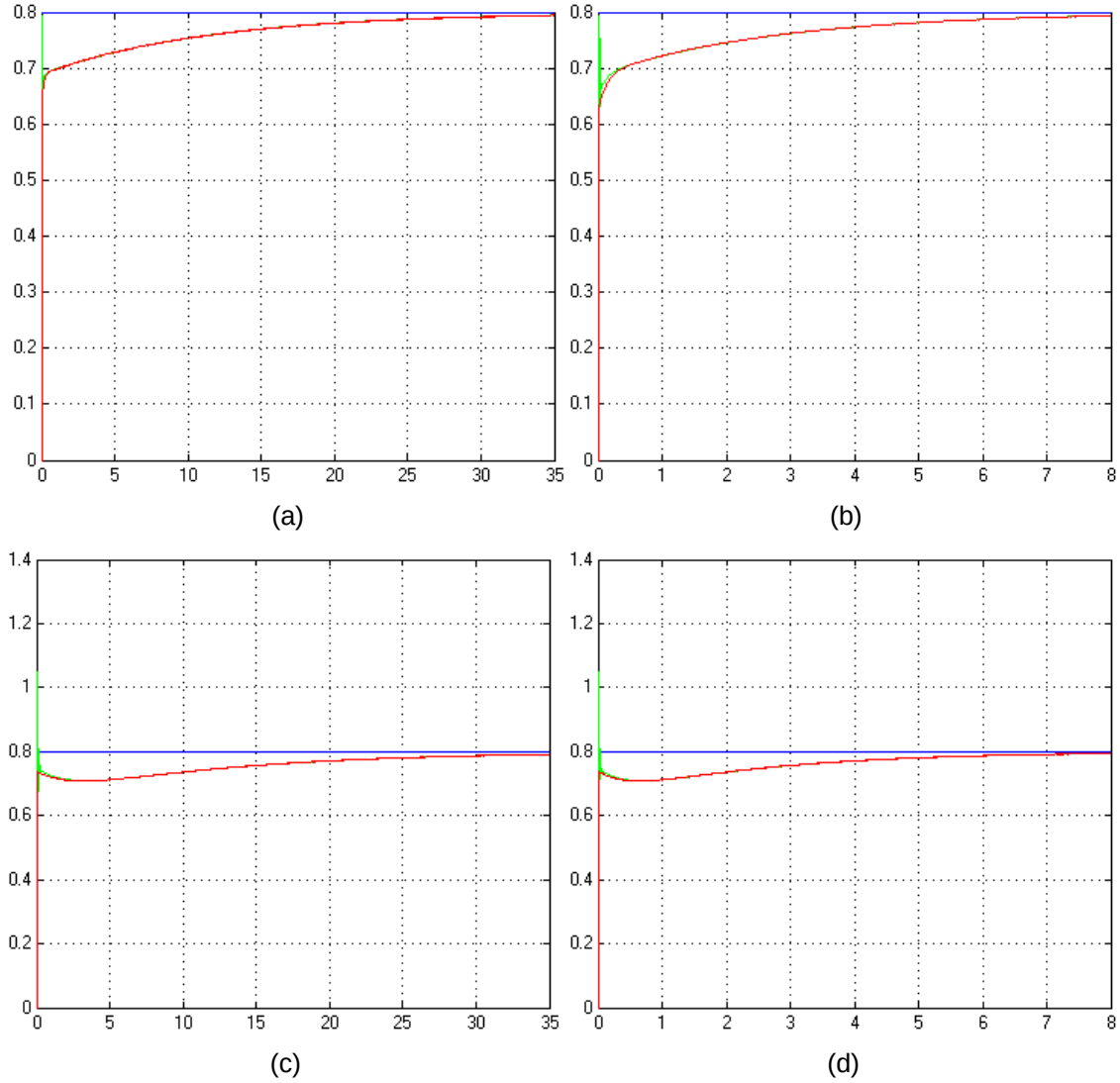


Figura 37 – Simulação do controlador PID – 80%.

Fonte: Do autor, 2011.

Quando inserido o controlador derivativo no simulador surge o sobre sinal, mais visível quando o valor ganho derivativo é maior. O controlador PID do sistema real (Figura 30 e Figura 31) e o simulado (Figura 36 e Figura 37) são similares, pois anulam o erro em regime permanente, porém o sistema real possui uma resposta rápida comparado com o sistema simulado.

4.2 CUSTOS DO PROJETO

O projeto do experimento remoto acarretou custos, pois houve a necessidade da obtenção de alguns itens para desenvolver o experimento. Dentre os itens que foram comprados para o projeto foram o Arduino Uno e o *Ethernet Shield*. O motor utilizado no experimento é de propriedade da instituição SATC. Bem como os componentes utilizados para fabricação da placa auxiliar e a fonte para alimentá-la. A seguir na Tabela 7 é mostrado o custo total, supondo que a instituição não possui-se os itens apontados anteriormente.

Tabela 7 – Custos do projeto.

Arduino Uno	R$ 88,00
Ethernet Shield	R$ 70,00
Motor	R$ 100,00
Placa auxiliar e fonte	R$ 20,00
Câmera	R$ 422,00
Total	R$ **700,00**

Fonte: Do autor, 2011.

O custo apontando acima considera a inclusão de uma câmera ao experimento para que o usuário possa visualizar o funcionamento do motor. No primeiro instante é notável que se trata de um custo elevado levando em conta que é somente um experimento, porém se trata um experimento remoto, que pode ser acessado por diversas pessoas em horários distintos e de qualquer local com acesso a rede internet.

5 CONCLUSÃO

O controlador PID empregado no experimento resultou no controle adequado da rotação do motor, conforme os ganhos utilizados e a resposta do sistema atendeu o esperado. Com as simulações feitas a partir das funções de transferência obtidas por análise gráfica foi observado a convergência das respostas do sistema físico implementado com as teóricas de sistemas de controle.

Na implementação do protótipo do sistema de controle foram aproveitados componentes já existentes na instituição. A integração da instrumentação com os elementos de potência foi feita utilizando-se um microcontrolador, que por meio de um *software* possibilita ao usuário observar o desempenho e interagir com o sistema, assim atuando sobre o controle da rotação do motor.

A implementação do software de interação e do hardware realizados permitiu que o controle fosse acessado de locais externos a rede da instituição de ensino, em diferentes horários, demonstrando o funcionamento remoto adequado.

No desenvolvimento de experimentos remotos aplicados na área da educação, em específico a sistemas de controle, observou-se algumas dificuldades relacionadas ao material bibliográfico para subsidiar os conceitos necessários ao desenvolvimento de experimentos referentes a sistemas de controle. Dentro deste contexto podemos citar os poucos livros para auxiliar na compreensão dos *softwares* utilizados para implementar a proposta.

Este trabalho pode ser aplicado em estudos futuros como um incentivo a desenvolver novos experimentos remotos, utilizando-se das ferramentas de *hardware* e *software* contidas neste. Os arquivos desenvolvidos no ambiente Arduino serão de grande ajuda para futuros desenvolvedores de experimentos remotos utilizando essa plataforma.

REFERÊNCIAS

[1] COOPER, M. The challenge of practical work in a eUniversity - real, virtual and remote experiments. In: INFORMATION SOCIETY TECHNOLOGIES (IST), 2000, Nice, France. Proceedings... Nice, 2000.

[2] SCHAF, Frederico Menine. Arquitetura para Ambiente de Ensino de Controle e Automação Utilizando Experimentos Remotos de Realidade Mista. 2006**.** 207 f. Dissertação - Departamento de Engenharia Elétrica, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2006.

[3] SILVA, Juarez Bento da. A UTILIZAÇÃO DA EXPERIMENTAÇÃO REMOTA COMO SUPORTE À AMBIENTES COLABORATIVOS DE APRENDIZAGEM.. Florianópolis 2007.

[4] AUER, M.; PESTER, A.; URSUTIU, D.; SAMOILA, C. Distributed Virtual and Remote Labs in Engineering. In: IEEE INTERNATIONAL CONFERENCE ON INDUSTRIAL TECHNOLOGY, 2003, Maribor, Slovenia. Proceedings... IEEE, 2003. p.1208-1213.

[5] RODRIGUES FILHO, Renato. Desenvolva Aplicativos com Java 6. São Paulo: Érica, 2008.

[6] MARSHALL, Brain. "Desenvolvendo Aplicativos com NETBEANS IDE 6". São Paulo: Editora Moderna, 2008.

[7] COMER, Douglas E.. Redes de Computador e Internet. Porto Alegre: Bookman, 2007.

[8] Arduino. Disponível em: <http://arduino.cc/>. Acesso em: 30 ago. 2011.

[9] FONSECA, Erika Guimarães Pereira da; BEPPU, Mathyan Motta. Arduino**.** Niterói: Universidade Federal Fluminense, 2010. 23 p.

[10] Arduino Uno. Disponível em: <http://arduino.cc/en/Main/ArduinoBoardUno>. Acesso em: 30 ago. 2011.

[11] Arduino Ethernet Shield. Disponível em: <http://arduino.cc/en/Main/ ArduinoEthernetShield>. Acesso em: 28 jul. 2011.

[12] BRUGNARI, Arthur; MAESTRELLI, Luiz Henrique Mussi. AUTOMAÇÃO RESIDENCIAL via WEB. Curitiba: Pontifícia Universidade Católica Do Paraná, 2010. 36 p.

[13] Arduino Development Environment. Disponível em: <http://arduino.cc/en/Guide/ Environment> Acesso em: 30 ago. 2011.

[14] NISE, Norman S. Engenharia de Sistemas de Controle. 5. ed. Rio de Janeiro: Livros Técnicos e Científicos Editora S.A., 2009.

[15] BOLTON, W. Engenharia de Controle. São Paulo: Makron Books do Brasil Editora Ltda, 1995.

[16] Universidade Federal de Santa Catarina (UFSC). Fundamentos de Controle Clássico. Florianópolis, 2005.

[17] BRISTOT, Vilmar Menegon. CONTROLE DE TEMPERATURA DE SECADORES DE REVESTIMENTOS CERÂMICOS ALIMENTADOS COM GÁS NATURAL. Florianópolis: Universidade Federal De Santa Catarina, 2002. 109 p.

[18] BRUCIAPAGLIA, A. H.; 1992. Desenvolvimento do Controlador PID – AA /UFSC: Processos com Atraso de Transporte Dominante. Universidade Federal de Santa Catarina, Florianópolis - Brasil.

[19] POSSER, M.S.; PID – Toolbox : Uma Ferramenta para o Ensino e Ajuste de Controladores PID's. Departamento de Engenharia Química, Universidade Federal do Rio Grande do Sul.

[20] NORMEY-RICO, J. E.; CAMACHO, E. F.; GOMEZ-ORTEGA, Juan; 1998. Robustez em Controladores Preditivos Generalizados. In: XII CONGRESSO BRASILEIRO DE AUTOMÁTICA (Set. 1998 : Uberlândia, Minas Gerais). Anais. Minas Gerais, 1998. p.157-162.

[21] DE CARVALHO, J.L. Martins. Sistemas de Controle Automático. Livros Técnicos e Científicos Editora, Rio de Janeiro, 2000.

[22] CAMPOS, Mário César M. Massa de; TEIXEIRA, Herbert C. G. Controles Típicos de Equipamentos e Processos Industriais. São Paulo: Editora Edgard Blucher Ltda., 2006.

[23] COELHO, Antônio Augusto Rodrigues; COELHO, Leandro Dos Santos, 2004. "Identificação de Sistemas Dinâmicos Lineares", Editora UFSC, Santa Catarina.

ANEXOS

ANEXO A – Arquivo “tcp_ethershield.pde”, simplifica a biblioteca do *Ethernet Shield*

```
/*
*  Arquivo: tcp_ethershield.pde
*  Versão: 2.0
*  Autor: Luiz Henrique Cassettari
*  Descrição: Arquivo utilizado para simplificar a biblioteca do ethernet shield da nuelectronics.com.
*  A biblioteca não foi modificada, somente adicionada funções para simplificar o programa principal.
*  Data: 10/11/2011
*/

// ------------------------------------------------
// Configurações do arquivo
// ------------------------------------------------

// Configuração do ethernet
byte mac[6] = {0x54,0x55,0x58,0x10,0x00,0x24};
byte ip[4] = {10,21,42,50};
int port = 80;

// Configuração do buffer
#define BUFFER_SIZE 500

// ------------------------------------------------
// Incluindo bibliotecas e variáveis
// ------------------------------------------------

#include <etherShield.h>

EtherShield es = EtherShield();
```

```
static byte buf[BUFFER_SIZE+1];

static struct {
  boolean close;
  int n_rec;
  int i_rec;
  int i_env;
} client;
// ------------------------------------------------

// ------------------------------------------------
// server_init
// ------------------------------------------------
void server_init(){
  /*inicia o chip enc28j60*/
  es.ES_enc28j60Init(mac);
  es.ES_enc28j60clkout(2); // muda o clock de saída
  es.ES_enc28j60PhyWrite(PHLCON,0x476);
  delay(100);

  es.ES_init_ip_arp_udp_tcp(mac,ip,port); // adiciona mac, ip e porta
}

// ------------------------------------------------
// client_close
// ------------------------------------------------
void client_close(){
  client.close = true;
}

// ------------------------------------------------
// client_clear
// ------------------------------------------------
void client_clear(){
```

```
  client.close = false;

  client.i_rec = 0;
  client.n_rec = 0;

  client.i_env = 0;
}

// ------------------------------------------------
// client_available
// ------------------------------------------------
int client_available(){
  return client.n_rec;
}

// ------------------------------------------------
// client_read
// ------------------------------------------------
byte client_read(){
  byte b = buf[TCP_DATA_P + client.i_rec++];
  client.n_rec--;
  return b;
}

// ------------------------------------------------
// client_write
// ------------------------------------------------
void client_write(byte b){
  buf[TCP_DATA_P + client.i_env++] = b;
}

// ------------------------------------------------
// client_read_line (string)
// ------------------------------------------------
```

```
void client_read_line(String &s){
  s = "";
  char c;
  while(client.n_rec){
      c = (char) client_read();
      if (c == '\n') break;
      s += c;
  }
}

// ------------------------------------------------
// client_print (string)
// ------------------------------------------------
void client_print_string(String &s){
  int len = s.length();
  for(int i = 0; i < len; i++)
    client_write(s[i]);
  s = "";
}

void client_println_string(String &s){
  client_print_string(s);
  client_write('\n');
}

// ------------------------------------------------
// client_print (const char)
// ------------------------------------------------
void client_print(const char *s){
  for(int i = 0; s[i] != '\0'; i++)
    client_write(s[i]);
}
void client_println(const char *s){
  client_print(s);
```

```
  client_write('\n');
}

// ---------------------------------------------
// client_print (byte)
// ---------------------------------------------
void client_print(byte b){
  String s = (int) b;;
  client_print_string(s);
}
void client_println(byte b){
  client_print(b);
  client_write('\n');
}

// ---------------------------------------------
// client_print (int)
// ---------------------------------------------
void client_print(int i){
  String s = i;
  client_print_string(s);
}
void client_println(int i){
  client_print(i);
  client_write('\n');
}

// ---------------------------------------------
// client_print (long)
// ---------------------------------------------
void client_print(long l){
  String s = l;
  client_print_string(s);
}
```

```
void client_println(long l){
  client_print(l);
  client_write('\n');
}

// ------------------------------------------------
// client_send
// ------------------------------------------------
void client_send() {

    es.ES_make_tcp_ack_from_any(buf); // envia pacote ack

    if (client.close)
      es.ES_make_tcp_ack_with_data(buf,client.i_env); // envia pacote de dados com
fim
    else
      es.ES_tcp_client_send_packet (  // envia pacote de dados sem fim
                                buf, // buffer
                                (buf[TCP_DST_PORT_H_P]<<8)                                +
buf[TCP_DST_PORT_L_P],  // porta de destino
                                port,  // porta local
                                TCP_FLAG_ACK_V | TCP_FLAG_PUSH_V, // flag
                                0,0,0,client.i_env, // tcp tamanho do dado
                                &buf[ETH_DST_MAC], // mac de destino
                                        &buf[IP_DST_P] // ip de destino
                                        );
}

// ------------------------------------------------
// client_receive
// ------------------------------------------------
int client_receive(){

  int plen;
```

```
  plen = es.ES_enc28j60PacketReceive(BUFFER_SIZE, buf);

  /*plen will ne unequal to zero if there is a valid packet (without crc error) */
  if(plen!=0){

    client_clear();

    // arp is broadcast if unknown but a host may also verify the mac address by sending it to a unicast address.
    if(es.ES_eth_type_is_arp_and_my_ip(buf,plen)){
      es.ES_make_arp_answer_from_request(buf);
      return 0;
    }

    // check if ip packets are for us:
    if(es.ES_eth_type_is_ip_and_my_ip(buf,plen)==0){
      return 0;
    }

    if(buf[IP_PROTO_P]==IP_PROTO_ICMP_V                                    && buf[ICMP_TYPE_P]==ICMP_TYPE_ECHOREQUEST_V){
      es.ES_make_echo_reply_from_request(buf,plen);
      return 0;
    }

    if (((buf[TCP_DST_PORT_H_P]<<8) + buf[TCP_DST_PORT_L_P]) != port) return 0;

    // tcp protocolo
    if ((buf[IP_PROTO_P]==IP_PROTO_TCP_V)  ){

      if (buf[TCP_FLAGS_P] & TCP_FLAGS_SYN_V){
```

```
        es.ES_make_tcp_synack_from_syn(buf);  //  make_tcp_synack_from_syn  does
already send the syn,ack
        return 0;
      }

      if (buf[TCP_FLAGS_P] & TCP_FLAGS_ACK_V){
        es.ES_init_len_info(buf); // init some data structures
        int dat_p=es.ES_get_tcp_data_pointer();
        if (dat_p==0){ // we can possibly have no data, just ack:
          if (buf[TCP_FLAGS_P] & TCP_FLAGS_FIN_V){
            es.ES_make_tcp_ack_from_any(buf);
          }
          return 0;

        }

        plen          =          ((buf[IP_TOTLEN_H_P]<<8)|(buf[IP_TOTLEN_L_P]&0xff)-
IP_HEADER_LEN-(buf[TCP_HEADER_LEN_P]>>4)*4);

        if (plen > BUFFER_SIZE-TCP_DATA_P) plen = BUFFER_SIZE-TCP_DATA_P;

        client.n_rec = plen;

        return (plen > 0);
      }
    }
  }
  return 0;
}
// ------------------------------------------------
```

ANEXO B – MÉTODO *SOCKET* NO ALGORITMO DO APLICATIVO JAVA.

O aplicativo desenvolvido se comporta como um cliente. Esse se conecta ao experimento remoto utilizando o metodo *Socket* do Java, permitindo a comunicação TCP entre os mesmos. A Figura 38 demonstra uma parte das linhas de código do aplicativo utilizada para declarar as variáveis utilizadas na comunicação com o experimento.

```
// ---------------------- Declaração de variáveis ----------------------

private static Socket client = null; // Define client como Socket

private static String ip = "10.21.42.50"; // Ip do experimento
private static int port = 80; // Porta do experimento

private static PrintWriter out = null; // Canal de saida de dados
private static BufferedReader in = null; // Canal de entrada de dados
private static boolean connected = false; // client conectado

// Buffer de dados para ler
private static StringBuffer ReadBuffer = new StringBuffer("");
private static String ReadString = "";

// Buffer de dados para enviar
private static StringBuffer SendBuffer = new StringBuffer("");
private static String SendString[] = new String[SENDSTRINGMAX];

// ---------------------------------------------------------------------
```

Figura 38 – Variáveis declaradas do aplicativo Java.

Fonte: Do autor, 2011.

Essas variáveis são utilizadas nos métodos da Figura 39, Figura 40 e Figura 41. A Figura 39 demonstra o método para se conectar ao experimento, a Figura 40 demonstra o método para enviar um dado ao experimento e a Figura 41 um método para receber dados do experimento.

```
/*
 *  Connect() : Abre conexão utilizando socket
 */

public void Connect() {
    try {
        System.out.println("Conectando..."); // Escreve no canal de saída
        client = new Socket(ip,port); // Abre conexão TCP
        // Define out e in como canal de entrada e saída de dados do client
        out = new PrintWriter(client.getOutputStream(),true);
        in = new BufferedReader(new InputStreamReader(client.getInputStream()));
        connected = true; // Conectado
        System.out.println("Conectado!"); // Escreve no canal de saída
    }
    catch (Exception exception) {
        connected = false; // Desconecatado
        System.out.println("Falha na conexão!"); // Escreve no canal de saída
    }
}
```

Figura 39 – Método *Connect* do aplicativo Java.

Fonte: Do autor, 2011.

```
/*
 *  Send() : Envia os dados do buffer
 */

public void Send(){
    if (connected){ // Se conectado
        // Envia dados do buffer para o canal de saída out
        for(int i = 0;i < SENDSTRINGMAX;i++){
            if (SendString[i] != null)
            out.print(SendString[i]); // Envia a string
            SendString[i] = ""; // Limpa a string
        }
        out.print(SendBuffer); // Envia o buffer
        SendBuffer.setLength(0); // Limpa o buffer
        System.out.println(SendBuffer); // Escreve no canal de saída
        out.flush(); // Libera dados do canal de saída
    }
}
```

Figura 40 – Método *Send* do aplicativo Java.

Fonte: Do autor, 2011.

```
/*
 *  Read() : Guarda no buffer os dados recebidos
 */

public boolean Read(){
    ReadString = ""; // Limpa string
    ReadBuffer.setLength(0); // Limpa buffer
    // Retorna false se estiver desconectado e canal de entrada for nulo
    if (!connected) return false;
    if (in == null) return false;
    try {
        if(in.ready()){ // Se canal in estiver pronto
            // Adiciona ao buffer a linha do canal de entrada
            ReadBuffer.append(in.readLine());
            System.out.println(ReadBuffer); // Escreve no canal de saída
            return true; // Retorna true
        }
    } catch (IOException ex) {}
    return false; // Retorna false
}
```

Figura 41 – Método R*ead* do aplicativo Java.

Fonte: Do autor, 2011.

ANEXO C – Arquivo “pidmotor_tcc.pde”, algoritmo principal do Arduino.

```
/*
*  Arquivo: pidmotor_tcc.pde
*  Versão: 1.0
*  Autor: Luiz Henrique Cassettari
*  Descrisão: Arquivo principal do Arduino.
*             Utilizado para controlar a rotação do motor com uma saída PWM, possui um filtro digital, algoritmo do controlador PID, e lógica de controle pela rede internet.
*  Data: 15/11/2011
*/

// ---------- defines ---------- //

#define PID_PIN 9
#define ENTRAR "on_entrar"
#define SAIR "off_sair"
#define DATA "dado"

// ---------- PID ---------- //

#include <Timer.h>

static Timer pid_timer(50);

boolean motor = false; // status do motor = ligado ou desligado

int kp = 1000; // constante proporcional
int ki = 500; // constante integral
int kd = 0; // constante derivativa

int setpoint = 204; // setpoint do controlador pid
int saida = 0; // saida do controlador pid
```

```
long erro_somatorio = 0; // somatorio dos erros
int erro_anterior = 0; // erro anterior

void fazer_pid(int entrada){

  if ((kp==0)&(ki==0)&(kd==0)){
     saida = setpoint;
     return;
  }

  int erro; // declaração do erro
  float dt = (float) 50/1000; // periodo de amostragem igual a 50ms
  float kps = (float) kp/1000;
  float kis = (float) ki/1000;
  float kds = (float) kd/1000;

  kis *= dt;
  kds /= dt;

  erro = setpoint - entrada; // calculo do erro
  erro_somatorio += (long) erro; // adiciona erro ao erro_somatorio

  float p = (float) (erro);
  float i = (float) (erro_somatorio);
  float d = (float) (erro - erro_anterior);

  saida = p*kps+i*kis+d*kds;

  erro_anterior = erro; // guarda erro para proxima interação
}

// ---------- Filtro ---------- //

static int nfiltro = 50; // valor da potencia do filtro
```

```
byte filtro_digital(){
  float n = (float) nfiltro/10000; // potencia do filtro passa baixa
  static float filtro; // declara variavel filtro como float estatico
  filtro = n*(analogRead(0)/4) + filtro*(1-n); // equação do filtro passa baixa
  return (byte) filtro;  // retorna valor filtrado
}

// ---------- Comandos net ---------- //

String string_taco = "";
static boolean ocupado = false;
static byte contador = 0;

enum data {PID_STATUS, PID_LIGA, PID_SET, PID_KP, PID_KI, PID_KD, TACO};

void envia_d(int i){
  client_print(DATA);
  client_write(' ');
  client_print(i);
  client_write(' ');
}

void ler_data(int i, long l){
  switch(i){
    case PID_LIGA:
      motor = l;
      if (motor) string_taco = "";
      break;
    case PID_SET:
      setpoint = l;
      break;
    case PID_KP:
      kp = l;
```

```
      break;
    case PID_KI:
      ki = l;
      break;
    case PID_KD:
      kd = l;
      break;
  }
}

void envia_data(int i){
  switch(i){
    case PID_STATUS:
      envia_data(PID_LIGA);
      envia_data(PID_SET);
      envia_data(PID_KP);
      envia_data(PID_KI);
      envia_data(PID_KD);
      break;
    case PID_LIGA:
      envia_d(i);
      client_println(motor);
      break;
    case PID_SET:
      envia_d(i);
      client_println(setpoint);
      break;
    case PID_KP:
      envia_d(i);
      client_println(kp);
      break;
    case PID_KI:
      envia_d(i);
      client_println(ki);
```

```
      break;
    case PID_KD:
      envia_d(i);
      client_println(kd);
      break;
    case TACO:
      client_print(DATA);
      client_write(' ');
      client_print(i);
      client_println_string(string_taco);
      break;
  }
}

void limpa_tudo(){
  erro_anterior = 0;
  erro_somatorio = 0;
  analogWrite(PID_PIN,0);
}

// ---------- setup ---------- //

void setup(){
  server_init(); // inicia o chip enc28j60
  pid_timer.Start();
  pinMode(PID_PIN, OUTPUT);
  analogWrite(PID_PIN,0);
  analogReference(EXTERNAL); // Ativa referência analogica
}

// ---------- loop ---------- //

void loop(){
  byte tacos = filtro_digital();
```

```
if (pid_timer.Get()){
  if (ocupado){
    if (100 <=contador++){
      ocupado = false;
      contador = 0;
      motor = false;
    }
  }
  if (motor){
    fazer_pid(tacos);
    if (saida < 0) saida = 0;
    else if (saida > 255) saida = 255;
    analogWrite(PID_PIN,saida);
  }
  else {
    limpa_tudo();
  }
  if (string_taco.length()<200){
    string_taco += " ";
    string_taco += (int) motor*setpoint;
    string_taco += " ";
    string_taco += (int) tacos;
  }
}

if (client_receive()){ // caso recebeu algum pacote de dados
  String string = "";

  while(client_available()){ // enquanto tiver dados para ler
    String str = "";
    client_read_line(str);
    string += str;
    string += "\n";
  }
```

```
while(string.length()){
  String str = "";
  string_line(string,str);
  if (string_test(str,"GET")){
    client_println(contador*50);
    client_println(ocupado);
    client_println(tacos);
    client_close();
    break;
  }
  if (string_test(str,ENTRAR)){
    if (ocupado){
      client_println(SAIR);
      break;
    }
    ocupado = true;
    client_println(ENTRAR);
    string_taco = "";
  }
  if (string_test(str,SAIR)){
    client_println(SAIR);
    ocupado = false;
    contador = 0;
    motor = false;
  }
  if (string_test(str,DATA)){
    contador = 0;
    int i = string_valor(str);
    if (str.length())
      ler_data(i,string_valor(str));
    envia_data(i);
  }
}
```

```
    client_send(); // envia o pacote
  }
}

// ---------- Comandos string ---------- //

boolean string_test(String &str,const char* s){
  boolean ret = true;
  int i;
  for(i = 0; s[i]!='\0'; i++){
    if (s[i] != str[i]) ret = false;
  }
  if (ret){
    if ((str[i] == ' ')|(str[i] == '\r')){
      str = str.substring(i+1);
      return true;
    }
  }
  return false;
}
void string_line(String &str, String &s){
  int n = str.indexOf('\n');
  s = str.substring(0,n);
  str = str.substring(n+1);
}
long string_valor(String &str){
  int n = str.indexOf(' ');
  if (n == -1) n = str.length();
  String s = str.substring(0,n);
  str = str.substring(n+1);
  return s.toInt();
}
// ----------  ---------- //
```

ANEXO D – MÉTODO DE HÄGGLUND.

Segundo Coelho e Coelho, (2004) esse método utiliza analise gráfica de um sistema de malha aberta para determinar um modelo matemático, a fim de encontrar uma função de transferência de primeira ordem. Resultando em uma função conforme a equação 6. [23]

$$H1(s) = \frac{K}{\tau * s + 1} \tag{6}$$

Onde:

K = ganho do sistema;

τ = constante de tempo;

O ganho proporcional é encontrado conforme a equação 7, que representa o valor da entrada (y) pela saída (u) do sistema quando em regime permanente.

$$K = \frac{y}{u} \tag{7}$$

O valor da constante de tempo o tempo de 63.2%, ou seja, tempo para a saída alcançar 63.2% da mudança total final. Abaixo Figura 42 demonstra o gráfico com as variáveis para encontrar a função de transferência de primeira ordem.

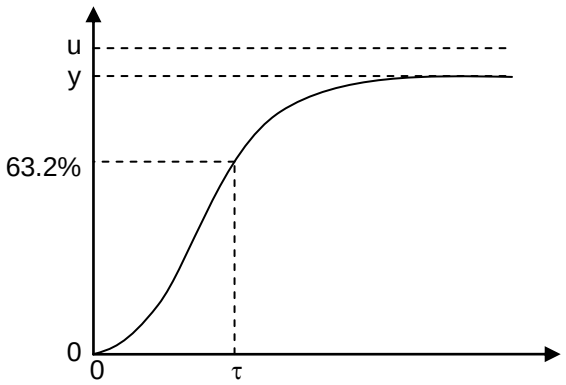


Figura 42 – Gráfico do método de Hägglund.

Fonte: Do autor, 2011.

ANEXO E – MÉTODO DE MOLLENKAMP.

O método Mollenkamp, conforme Coelho e Coelho (2004), utiliza análise gráfica similar ao método de Hägglund, porém resulta em uma função de transferência de segunda ordem. Assim é preciso identificar três pontos intermediários no gráfico, isto é, t_1 - tempo para a saída alcançar 15% da mudança total final; t_2 - tempo para a saída alcançar 45% da mudança total final; t_3 - tempo para a saída alcançar 75% da mudança total final. Com base nestes instantes de tempo, os parâmetros do modelo matemático das variáveis são calculados pelo seguinte algoritmo, utilizando a equação 8. [23]

$$x = \frac{t_2 - t_1}{t_3 - t_1} \tag{8}$$

A obtenção do coeficiente de amortecimento (ξ) do sistema em estudo é expresso pela equação 9, por meio da variável auxiliar (x).

$$\xi = \frac{0.0805 - 5.547 * (0.475 - x)^2}{x - 0.356} \tag{9}$$

Conhecendo o valor do coeficiente de amortecimento (ξ), por meio do algoritmo, determina-se os valores de f_2 (ξ) para $\xi < 1$ a equação 10, e para valores de f_2 (ξ) para $\xi \geq 1$ a equação 11.

$$f_2(\xi) = (0.708) * (2.811)^{\xi} \tag{10}$$

$$f_2(\xi) = 2.6 * \xi - 0.6 \tag{11}$$

Utilizando a equação 12 determina-se a freqüência natural do sistema.

$$w_n = \frac{f_2(\xi)}{t_3 - t_1} \tag{12}$$

$$f_3(\xi) = (0.922) * (1.66)^{\xi} \tag{13}$$

Com a obtenção do valor da variável $f_3(\xi)$ pela equação 13, determina-se a constante de tempo (θ) através da equação 14.

$$\theta = t_2 - \frac{f_3(\xi)}{w_n} \tag{14}$$

Através da equação 15, determinam-se os valores das constantes de atraso de transporte ($\tau_{1,2}$) do sistema.

$$\tau_{1,2} = \frac{\xi \pm \sqrt{\xi^2 - 1}}{w_n} \tag{15}$$

Utilizando esse conjunto de equações é possível encontrar uma função de segunda ordem.