UNIVERSIDADE LUTERANA DO BRASIL

PRÓ-REITORIA DE GRADUAÇÃO

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

João Daniel de Oliveira Klein

BANCADA DE TESTES E LEVANTAMENTO DE PARÂMETROS DE MOTORES DE CORRENTE CONTÍNUA

Canoas, Julho de 2010

JOÃO DANIEL DE OLIVEIRA KLEIN

BANCADA DE TESTES E LEVANTAMENTO DE PARÂMETROS DE MOTORES DE CORRENTE CONTÍNUA

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da ULBRA como um dos requisitos obrigatórios para a obtenção do grau de Engenheiro Eletricista

Departamento:

Engenharia Elétrica

Área de Concentração

Instrumentação eletroeletrônica

Professor Orientador:

MSc. Eng. Eletr. Paulo César Cardoso Godoy – CREA-RS: 0116822-D

Professor Co-Orientador:

Dr. Eng. Eletr. Valner João Brusamarello – CREA-RS: 078158-D

Canoas

2010

FOLHA DE APROVAÇÃO

Nome do Autor: João Daniel de Oliveira Klein

Matrícula: 051008762-0

Título: Bancada de Testes e Levantamento de Parâmetros de Motores de Corrente Contínua

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da ULBRA como um dos requisitos obrigatórios para a obtenção do grau de Engenheiro Eletricista

Professor Orientador:

MSc. Eng. Eletr. Paulo César Cardoso Godoy

CREA-RS: 0116822-D

Professor Co-Orientador:

Dr. Eng. Eletr. Valner João Brusamarello

CREA-RS: 078158-D

Banca Avaliadora:

Dr. Eng. Eletr. Marília Amaral da Silveira

CREA-RS: 050909-D

Conceito Atribuído (A-B-C-D):

MSc. Eng. Eletr. Augusto Alexandre Durgante de Mattos

CREA-RS: 088003-D

Conceito Atribuído (A-B-C-D):

Assinaturas:

Autor João Daniel de Oliveira Klein	Orientador Paulo César Cardoso Godoy
Avaliador Marília Amaral da Silveira	Avaliador Augusto Alexandre Durgante de Mattos

Relatório Aprovado em:

DEDICATÓRIA

Dedico aos meus pais por tudo o que
eles são e fizeram por mim. Sem eles,
com certeza, não teria chegado até aqui.

AGRADECIMENTOS

Este trabalho é o fruto da contribuição de muitas pessoas: pais, amigos, professores e funcionários. A todos estes, o meu reconhecimento.

Ao professor Valner, por ter motivado e idealizado este projeto, pela dedicação, paciência e colaboração no seu desenvolvimento.

Ao professor Godoy, pelas sugestões e orientações referentes, principalmente, à etapa final deste trabalho.

À professora Marília e aos professores Augusto e Gertz, pelas sugestões que proporcionaram melhorias e aperfeiçoamento dos resultados.

À Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS) que apoiou o desenvolvimento deste projeto.

EPÍGRAFE

"Se A é o sucesso, então A é igual a X mais Y mais Z. O trabalho é X; Y é o lazer; e Z é manter a boca fechada."

Albert Einstein

RESUMO

KLEIN, João Daniel de Oliveira. **Bancada de Testes e Levantamento de Parâmetros de Motores de Corrente Contínua**. Trabalho de Conclusão de Curso em Engenharia Elétrica - Departamento de Engenharia Elétrica. Universidade Luterana do Brasil. Canoas, RS. 2010.

O conhecimento dos parâmetros de uma máquina CC é de grande importância para diversos profissionais, principalmente aqueles que trabalham com o controle destes dispositivos. Pensando nisso, foi desenvolvida uma bancada para levantamento dos parâmetros elétricos e mecânicos, diferentemente do método convencional, buscando a otimização dos ensaios. O circuito aplica uma excitação em tensão do tipo degrau na armadura do motor e monitora a velocidade angular desenvolvida e a corrente elétrica até que as mesmas estabilizem. Utilizando-se da equação diferencial que descreve o funcionamento dinâmico de uma máquina CC com excitação independente, do método recursivo dos mínimos quadrados e dos dados de velocidade e corrente, é possível estimar tais parâmetros. Toda a bancada é comandada através de um computador, com software especificamente desenvolvido para este fim em linguagem C++. O algoritmo empregado no cálculo dos parâmetros apresentou grande confiabilidade na obtenção da constante de força contra-eletromotriz e resistência elétrica, com desvios-padrão relativos da ordem de 6% e erro em relação aos ensaios convencionais da ordem de 4%. No entanto, apresentou certa fragilidade no que diz respeito à obtenção da indutância e do coeficiente de atrito, com desvios-padrão relativos e erros em relação aos ensaios convencionais da ordem de 60% e 40%, respectivamente. Apesar disso, o conjunto de valores estimados gerou resultados simulados que reproduzem com boa fidelidade as curvas obtidas experimentalmente.

Palavras chave: Máquinas CC. Mínimos quadrados. Instrumentação eletrônica.

ABSTRACT

KLEIN, João Daniel de Oliveira. **Bench Test and Survey Parameters of DC Motors**. Work of Conclusion of Course in Electrical Engineering - Electrical Engineering Department. Lutheran University of Brazil. Canoas, RS. 2010.

Knowledge of the parameters of a DC machine is of great importance to many professionals, especially those working on the control of these devices. Thinking about it, a bench was developed to estimate the electrical and mechanical parameters, unlike the conventional method, seeking to optimize the tests. The circuit applies a step voltage in the armature of the motor and monitors the angular velocity developed and electrical current until they stabilize. Using the differential equation that describes the dynamic operation of a separated excitation DC motor, the method of recursive least squares and data speed and current, it is possible to estimate such parameters. The entire bench is controlled by a computer, with software specifically developed for this purpose in language C++. The algorithm used to calculate the parameters showed great reliability in obtaining the back-emf constant and electrical resistance and fragility with regard to obtaining the inductance and coefficient of friction. However, the set of estimated values generated in simulation results that reproduce with good accuracy the experimental curves.

Keywords: DC Machine. Least squares. Electronic instrumentation.

LISTA DE FIGURAS

Figura 2.1: Esboço de uma máquina CC. (Fonte: Nasar, 1984 – pag. 75) 6
Figura 2.2: Estrutura básica de uma máquina CC. (Fonte: Nasar, 1984 – pag. 76) 6
Figura 2.3: Estrutura básica de uma máquina CC de ímãs permanentes. (Fonte: Fitgerald et al., 2006 - pag. 370) 7
Figura 2.4: Modelo de uma máquina de corrente contínua. (Fonte: Basilio e Moreira, 2001 – pag. 299) 7
Figura 2.5: Estrutura fundamental da Rede Neural de Elman. (Fonte: Al-Quassar e Othman, 2008 – pag. 192) . 22
Figura 3.1: Diagrama simplificado hardware. 26
Figura 3.2: Rebobinagem do transformador 29
Figura 3.3: Fonte de alimentação 29
Figura 3.4: Circuito principal - aquisição e comunicação 30
Figura 3.5: Identificação dos pinos do MSP430F2274. (Fonte: Datasheet Texas Instruments, 2007 – pag. 3) 31
Figura 3.6: Identificação dos pinos do conversor. 32
Figura 3.7: Topologia do amplificador de potência 33
Figura 3.8: Terminais do LM 675 (Fonte: Datasheet National 33
Figura 3.9: Localização dos extensômetros em uma célula de carga e o seu comportamento quando é aplicada uma força. (Fonte: Balbinot e Brusamarello, 2007 – pag. 189) 35
Figura 3.10: Ponte de Wheatstone para uma célula de carga com quatro extensômetros. (Fonte: Balbinot e Brusamarello, 2007 – pag. 208) 35
Figura 3.11: Célula de carga tipo *"single point"*. 36
Figura 3.12: Representação do circuito interno do INA 126. (Fonte: Datasheet Burr-Brown, 1996 – pag. 1) 37
Figura 3.13: Amplificador da célula de carga. 37
Figura 3.14: Topologia básica do circuito para medição de corrente 39
Figura 3.15: Circuito para medição de corrente com divisor de tensão 39
Figura 3.16: Circuito simplificado do INA 200. (Fonte: Datasheet Burr-Brown, 2007 – pag. 1) 40
Figura 3.17: Exemplo de ligação do amplificador para monitorar corrente através de um resistor shunt. (Fonte: Datasheet Burr-Brown, 2007 – pag. 12) 40
Figura 3.18: Circuito de acionamento de uma carga resistiva. 41
Figura 3.19: Estrutura da bancada de ensaios 42
Figura 3.20: Diagrama em blocos do software 43
Figura 3.21: Fluxograma do algoritmo utilizado para filtragem 44
Figura 3.22: Sequência de cálculos para determinação dos parâmetros. 45
Figura 3.23: Sequência de comando do conversor digital-analógico. (Fonte: Datasheet Burr-Brown, 1998 – pag. 10) 50
Figura 3.24: Programador do MSP430 51
Figura 3.25: Modelo motor CC no Simulink. 51
Figura 3.26: Gráfico gerado pelo modelo da máquina CC no Simulink. 52
Figura 3.27: Diagrama em blocos que descrevem o funcionamento de um motor CC com excitação independente. 52
Figura 4.1: Gráficos simulados com parâmetros da tabela 4.2 - SERVO DC 58
Figura 4.2: Gráficos simulados com parâmetros da tabela 4.4 - SERVO DC 58
Figura 4.3: Gráficos do ensaio prático 58
Figura 4.4: Gráficos simulados com parâmetros da tabela 4.2 - SERVO DC (ampliado) 59
Figura 4.5: Gráficos simulados com parâmetros da tabela 4.4 - SERVO DC (ampliado) 59
Figura 4.6: Gráficos do ensaio prático (ampliado) 59
Figura 4.7: Gráficos simulados com parâmetros da tabela 4.5 - PACIFIC CIENTIFIC 60
Figura 4.8: Gráficos simulados com parâmetros da tabela 4.7 - PACIFIC CIENTIFIC 60
Figura 4.9: Gráficos do ensaio prático - PACIFIC CIENTIFIC. 60
Figura 4.10: Gráficos simulados com parâmetros da tabela 4.5 - PACIFIC CIENTIFIC (ampliado) 61
Figura 4.11: Gráficos simulados com parâmetros da tabela 4.7 - PACIFIC CIENTIFIC (ampliado) 61
Figura 4.12: Gráficos do ensaio prático - PACIFIC CIENTIFIC (ampliado) 61
Figura 4.13: Gráficos simulados com parâmetros da tabela 4.8 – ENGEL 62

Figura 4.14: Gráficos simulados com parâmetros da tabela 4.10 – ENGEL 62
Figura 4.15: Gráficos do ensaio prático - ENGEL 62
Figura 4.16: Gráficos simulados com parâmetros da tabela 4.8 – ENGEL (ampliado) 63
Figura 4.17: Gráficos simulados com parâmetros da tabela 4.10 – ENGEL (ampliado) 63
Figura 4.18: Gráficos do ensaio prático - ENGEL (ampliado) 63
Figura 4.19: Gráficos simulados - Ensaio 1 - SERVO DC 65
Figura 4.20: Gráficos simulados - Ensaio 2 - SERVO DC 65
Figura 4.21: Curva de rendimento do motor. 66

LISTA DE TABELAS

Tabela 4.1: Valores médios obtidos para o gerador 53
Tabela 4.2: Média e desvio-padrão - SERVO DC - 23,5V 54
Tabela 4.3: Média e desvio-padrão - SERVO DC - 21V 54
Tabela 4.4: Média e desvio-padrão - SERVO DC - 18V 55
Tabela 4.5: Média e desvio-padrão - PACIFIC CIENTIFIC - 23,5V 55
Tabela 4.6: Média e desvio-padrão - PACIFIC CIENTIFIC - 21V 55
Tabela 4.7: Média e desvio-padrão - PACIFIC CIENTIFIC - 18V 56
Tabela 4.8: Média e desvio-padrão - ENGEL - 23,5V 56
Tabela 4.9: Média e desvio-padrão - ENGEL - 21V 56
Tabela 4.10: Média e desvio-padrão - ENGEL - 18V 57
Tabela 4.11: Comparativo dos resultados obtidos com a bancada e através de ensaio convencional - SERVO DC. 64
Tabela 4.12: Comparativo dos resultados obtidos com a bancada e através de ensaio convencional - PACIFIC CIENTIFIC 64
Tabela 4.13: Comparativo dos resultados obtidos com a bancada e através de ensaio convencional - ENGEL. .. 64
Tabela 4.14: Dois ensaios distintos - motor SERVO DC. 65
Tabela 4.15: Tabela de velocidade, potência mecânica e rendimento do motor SERVO DC. 66

LISTA DE ABREVIATURAS E SIGLAS

PC: *Personal Computer* – Microcomputador

CC: Corrente Contínua

CA: Corrente Alternada

SI: Sistema Internacional de Medidas

LISTA DE SÍMBOLOS

b – Coeficiente de atrito viscoso

I – Corrente elétrica

V – Tensão elétrica

ω – Velocidade angular

R_a - Resistência de armadura

I_a - Corrente de armadura

V_a - Tensão de armadura

e_a - Força eletromotriz

τ_a - Constante de tempo elétrico da armadura

τ_{em} - Constante de tempo eletromecânico do rotor

J - Momento de inércia

K_e - Constante de força contra-eletromotriz

K_m - Constante de torque

B - Densidade de fluxo eletromagnético

l - Comprimento

F - Força

T_m - Torque mecânico

ε - Erro

SUMÁRIO

1. INTRODUÇÃO 2

2. REFERENCIAL TEÓRICO 5

2.1. A Máquina CC 5
2.1.1. Princípio de Funcionamento 5
2.1.2. Máquina CC de ímã permanente 7
2.1.3. Modelo Dinâmico de Uma Máquina de Corrente Contínua 7
2.2. Identificação de Sistemas 10
2.2.1. Mínimos Quadrados 10
2.2.2. Estimador Recursivo de Mínimos Quadrados (RMQ) 12
2.2.3. Outros Estimadores Recursivos 13
2.3. Obtenção de Parâmetros da Máquina CC 14
2.3.1. Modelo Dinâmico e Mínimos Quadrados 14
2.3.2. Modelo Dinâmico Visto Como Dois Sistemas de Primeira Ordem Desacoplados [1] 14
2.3.3. Método de Momentos [7] 18
2.3.4. Redes Neurais 21
2.3.5. Ensaio Convencional 24

3. MATERIAIS E MÉTODOS 26

3.1. Hardware 27
3.1.1. Fonte de Alimentação 27
3.1.2. Placa de aquisição e comunicação 29
- Microcontrolador MSP430F2274 30
- Conversor DAC 7614 32
- Amplificador Operacional LM 675 33
- Darlington MJ11016 33

3.1.3. Medição de Torque 34
3.1.4. Medição de Velocidade Angular 37
3.1.5. Condicionador de sinais para medição de corrente 38
3.1.6. Conversor UART – USB 41
3.1.7. Conjunto de cargas resistivas 41
3.1.8. Parte mecânica/eletromecânica 42
3.2. O Software 43
3.3. O Firmware 48
3.3.1. Programação do Microcontrolador 48
3.3.2. Software de Programação e Compilação 48
3.4. Validação do modelo 51

4. APRESENTAÇÃO E DISCUSSÃO DOS RESULTADOS 53

4.1. Procedimento 53
4.2. Resultados obtidos 54
4.2.1. Motor SERVO DC – Tensão nominal: 24V 54
4.2.2. Motor MICRO SWTCH/PACIFIC CIENTIFIC – Tensão nominal: 24V 55
4.2.3. Motor ENGEL – Tensão nominal: 52V 56
4.3. Simulações baseadas nos resultados 57
4.3.1. Motor SERVO DC 58
4.3.2. Motor MICRO SWTCH/PACIFIC CIENTIFIC 60
4.3.3. 60

4.3.4. Motor ENGEL 62
4.3.5. Considerações da análise através de simulação 64
4.4. Ensaio convencional dos motores 64
4.5. Sensibilidade da resposta com a variação de resultados 65
4.6. Avaliação de rendimento 66
5. CONSIDERAÇÕES FINAIS 67
5.1. Avaliação dos Objetivos Propostos 67
5.2. Limitações do Sistema 67
5.3. Conclusões 68
5.4. Sugestões para Trabalhos Futuros 68
REFERÊNCIAS 70
OBRAS CONSULTADAS 71
APÊNDICE A – TABELAS DE ENSAIOS 72
APÊNDICE B – DIAGRAMAS ELÉTRICOS 74
APÊNDICE C – CÓDIGO FONTE DO MICROCONTROLADOR 79
APÊNDICE D – CÓDIGO FONTE DO SOFTWARE DE ESTIMATIVA DOS PARÂMETROS 86
APÊNDICE E – CÓDIGO FONTE DO SOFTWARE DE CÁLCULO DE RENDIMENTO 97
ANEXO A – BIBLIOTECA "MATRIX.H" 105

1. INTRODUÇÃO

Esse trabalho apresenta o desenvolvimento de uma bancada para ensaios de uma máquina de corrente contínua, que seja capaz de estimar parâmetros elétricos e mecânicos de forma fácil e rápida. Em geral, a determinação desses parâmetros é feita através de ensaios que exigem conceitos específicos de mecânica e de máquinas elétricas, o que dificulta essa tarefa [1], podendo até se tornar inviável em laboratórios de controle.

Os parâmetros elétricos incluem a constante de força contra-eletromotriz, resistência elétrica e a indutância da armadura, enquanto os mecânicos são a constante de torque, o momento de inércia e o coeficiente de atrito viscoso. Outra característica bastante útil e que pode ser obtida é o rendimento de uma máquina CC. Todos esses dados são de grande utilidade para empresas que trabalham com o desenvolvimento, manutenção e aplicação de motores e também laboratórios de pesquisa diversos.

A bancada de testes desenvolvida permite o ensaio de motores de até 24V com uma potência máxima de 150W. Esses valores foram escolhidos porque muitos dos motores de pequeno e médio porte encontram-se nessa faixa de tensões e potências. Caso seja necessário o ensaio de motores maiores, basta substituir alguns dispositivos e recalibrar o sistema, pois o algoritmo para a determinação dos parâmetros de interesse continua o mesmo.

Ensaios de máquinas elétricas geralmente são realizados através de medições de tensão e corrente aplicadas à armadura com o rotor livre e travado, torque desenvolvido com o rotor travado e velocidade angular com o rotor livre. Esse tipo de ensaio necessita a ligação de vários instrumentos de medição, tais como amperímetro, voltímetro, torquímetro e tacômetro. Esse procedimento, além de ser trabalhoso, não permite que sejam levantados todos os parâmetros de forma dinâmica, nem a avaliação do seu rendimento em diferentes condições de carga, velocidade, torque, entre outros. A bancada proposta faz uma avaliação dinâmica do motor, permitindo variar a tensão aplicada à sua armadura (e a velocidade angular) e a carga mecânica aplicada no eixo do motor. Em qualquer das condições, pode ser verificado o seu rendimento.

Para se variar a carga mecânica, tem-se algumas opções, como por exemplo, acoplar um freio mecânico ao eixo do motor, acoplar um gerador e variar

uma carga resistiva aplicada a ele ou desenvolver um sistema de freio eletromagnético. Apesar de o freio mecânico ser de fácil implementação, com o uso de um disco e pastilhas, esta opção dificulta um controle automático através de software, necessitando de um atuador eletromecânico. Além disso, um sistema mecânico como o disco pode conter imperfeições ao longo de suas faces, fazendo com que a velocidade angular não seja constante ao longo de uma rotação. O uso do freio eletromagnético exige um aprofundamento do seu conhecimento e vários testes adicionais, o que demanda muito tempo além de não haver uma certeza quanto ao seu desempenho nessa aplicação. Dessa forma, optou-se pela utilização do gerador, que permite o chaveamento de cargas resistivas na sua saída, variando com isso a força contra-eletromotriz desenvolvida. Essa alternativa não apresenta as desvantagens das outras opções, sendo o seu funcionamento confiável e de fácil controle através de software.

A determinação dos parâmetros elétricos e mecânicos será feita através de equações que descrevem o modelo dinâmico de uma máquina CC, que prevê uma variação da corrente em relação à velocidade angular. Isso pode ser obtido aplicando-se uma excitação em tensão na armadura do tipo degrau, que resulta em um pico de corrente enquanto houver aceleração angular do rotor. Dessa forma, a taxa de aquisição da corrente e da velocidade angular deve ser elevada o suficiente para que se possa encontrar a solução das equações com um erro mínimo. Isso inviabiliza o uso de instrumentos de medição convencionais como multímetros e tacômetros, sendo necessário um sistema de aquisição de alta velocidade.

Tendo em vista os itens descritos anteriormente, chegou-se ao modelo da bancada que foi construída. O motor a ser ensaiado está acoplado mecanicamente a um gerador elétrico, que tem a função de inserir uma carga mecânica variável. Tanto o acionamento do motor como o controle da carga mecânica é definido pelo usuário através de um micro computador (PC), onde também se encontra o algoritmo para estimação dos parâmetros descritos anteriormente.

A máquina em teste é acionada através de um conversor digital/analógico e um amplificador de potência, que suporta picos de corrente de até dez vezes a corrente nominal do motor, para suportar a corrente de partida. A tensão e corrente aplicadas ao motor são monitoradas e enviadas a um micro computador. O mesmo ocorre com a velocidade angular, que pode ser determinada através de um taco-gerador.

Tendo posse de uma quantia razoável de pontos do ensaio, já é possível estimar os parâmetros desejados. Isso é feito através do método recursivo dos mínimos quadrados que visa obter a melhor estimativa e com um erro quadrático mínimo.

2. REFERENCIAL TEÓRICO

O princípio de funcionamento das máquinas de corrente contínua já é bem conhecido, assim como alguns métodos empregados para a determinação dos seus parâmetros elétricos e mecânicos. Muitos desses métodos são recentes e encontram-se publicados em artigos técnicos.

Ao longo desta unidade será feira uma breve revisão da máquina CC e de procedimentos empregados na estimativa das suas características.

2.1. A Máquina CC

Apesar do advento da eletrônica de potência no que diz respeito ao acionamento de máquinas CA, a máquina CC ainda está presente em várias aplicações específicas. Isso se deve às diversas características de desempenho que são oferecidas pelos diferentes métodos de excitação e grande adaptabilidade ao controle, seja ele manual ou automático.

As variadas opções de excitação resultam em diferentes características de tensão em relação à corrente ou velocidade em relação ao conjugado. A grande facilidade de controle em sistemas que exigem uma ampla faixa de velocidades e precisão de controle também é um diferencial dessa máquina. [2]

Além das características já citadas, motores CC também são usados em larga escala em dispositivos e equipamentos portáteis, alimentados por fontes CC, tais como acumuladores, onde simplificam ou até dispensam a necessidade de circuitos eletrônicos adicionais.

2.1.1. Princípio de Funcionamento

A operação da grande maioria das máquinas elétricas é baseada na interação entre condutores percorridos por correntes elétricas e campos eletromagnéticos. A ação motora, especificamente, é baseada na lei de Lorenz (2.1):

$$\vec{F} = I\vec{\ell} \times \vec{B} \tag{2.1}$$

Onde $\vec{F}$ é o vetor força que atua sobre um condutor de comprimento ativo ℓ quando percorrido por uma corrente I na presença de um campo com densidade de fluxo magnético B. Dessa forma pode-se perceber que uma espira, quando

percorrida por corrente elétrica e sujeita a um campo magnético, sofre a ação de uma força proporcional ao seu comprimento ativo.

A máquina CC é dotada de um comutador localizado, geralmente, na extremidade da sua armadura (rotor). Esse comutador, além de fornecer corrente à bobina de armadura, também tem a finalidade de manter uma polaridade unidirecional para uma escova, ou obter um conjugado unidirecional em uma bobina num campo magnético. A Figura 2.1 mostra um esboço simplificado de uma máquina CC.

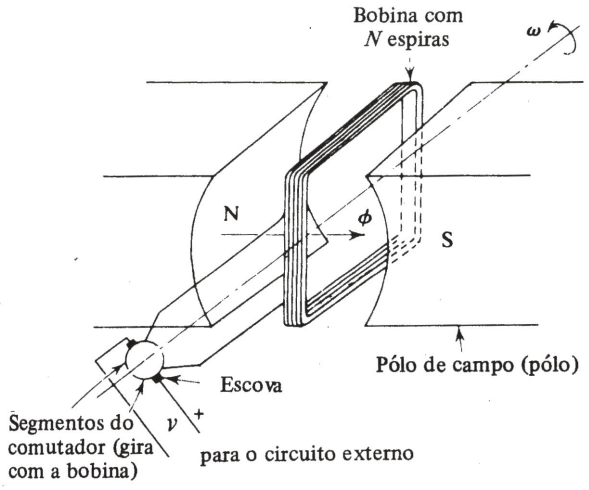


Figura 2.1: Esboço de uma máquina CC. (Fonte: Nasar, 1984 – pag. 75)

Os pólos de campo, que produzem o fluxo necessário, geralmente são montados no estator e possuem enrolamentos de campo (ou bobinas de excitação). Já o núcleo da armadura comporta o enrolamento da armadura e geralmente está no rotor. Este enrolamento é o enrolamento de carga. A Figura 2.2 mostra a estrutura típica da máquina CC. [3]

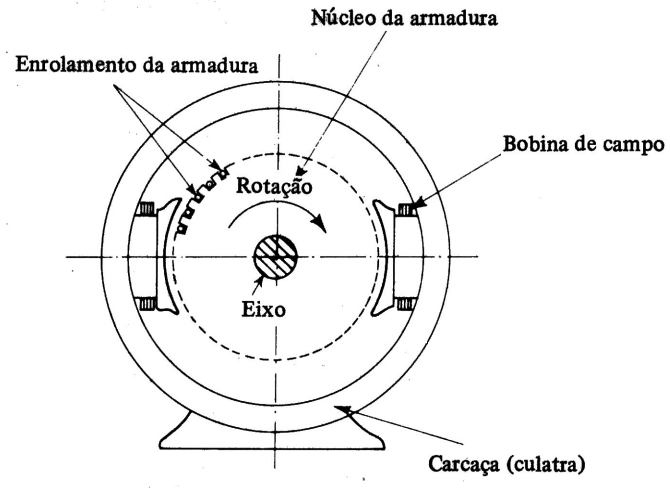


Figura 2.2: Estrutura básica de uma máquina CC. (Fonte: Nasar, 1984 – pag. 76)

2.1.2. Máquina CC de ímã permanente

As máquinas CC de ímã permanente são bastante encontradas em motores de potência reduzida. Nestes, o enrolamento de campo é substituído por ímãs permanentes. Geralmente o ímã tem formato cilíndrico, espessura uniforme e é magnetizado radialmente.

As vantagens desse tipo de máquina estão relacionadas a não necessidade de excitação externa (o que dissipa potência) e ao espaço necessário para acomodação dos ímãs, que pode ser menor do que o exigido para um enrolamento de campo.

A grande limitação das máquinas CC com ímãs permanentes está relacionada a máquinas de grande potência, onde pode haver a desmagnetização devido às correntes elevadas e ao sobreaquecimento. A Figura 2.3 ilustra a estrutura de uma máquina CC de ímã permanente. [2]

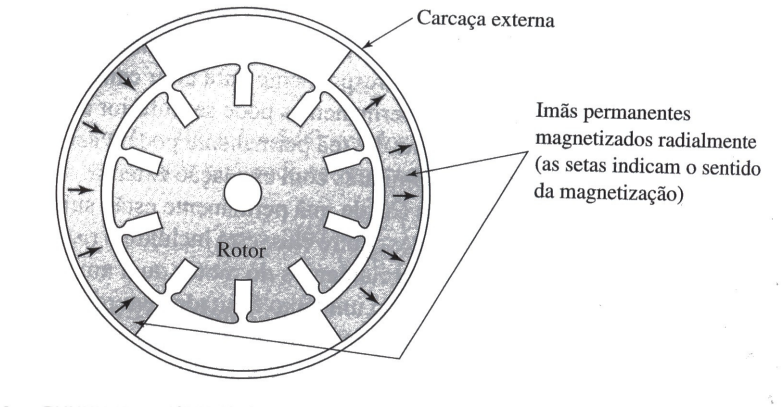


Figura 2.3: Estrutura básica de uma máquina CC de ímãs permanentes. (Fonte: Fitgerald et al., 2006 - pag. 370)

2.1.3. Modelo Dinâmico de Uma Máquina de Corrente Contínua

Um motor de corrente contínua controlado pela armadura pode ser representado pelo circuito equivalente da Figura 2.4.

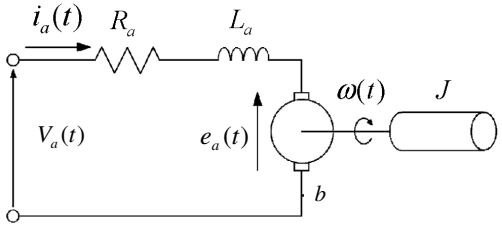


Figura 2.4: Modelo de uma máquina de corrente contínua. (Fonte: Basilio e Moreira, 2001 – pag. 299)

Onde:

R_a : resistência da armadura;

L_a : indutância da armadura;

J : momento de inércia;

b : coeficiente de atrito viscoso;

$V_a(t)$: tensão da armadura;

$i_a(t)$: corrente da armadura;

$e_a(t)$: força contra-eletromotriz;

$\omega(t)$: velocidade angular do rotor.

Aplicando-se a lei das tensões de Kirchhoff ao circuito apresentado, obtém-se a equação (2.2):

$$V_a(t) = R_a i_a(t) + L_a \frac{d}{dt} i_a(t) + e_a(t) \tag{2.2}$$

O equacionamento da parte mecânica pode ser obtido através da lei de Newton para o movimento rotacional, chegando-se à equação (2.3)

$$T_m(t) - b\omega(t) = J \frac{d}{dt} \omega(t) \tag{2.3}$$

A relação entre os parâmetros elétricos e mecânicos pode ser obtida através das equações (2.4) e (2.5), onde K_e é a constante de força contra-eletromotriz e K_m é a constante de torque.

$$e_a(t) = K_e \omega(t) \tag{2.4}$$

$$T_m(t) = K_m i_a(t) \tag{2.5}$$

Substituindo-se a equação (2.4) na equação (2.2), a equação (2.5) em (2.3) e rearranjando os termos, chega-se às equações (2.6) e (2.7) [1]:

$$L_a \frac{d}{dt} i_a = -R_a i_a(t) - K_e \omega(t) + V_a(t) \tag{2.6}$$

$$J \frac{d}{dt} \omega = K_m i_a(t) - b\omega(t) \tag{2.7}$$

Que é a mesma equação descrita por Saab [4].

Isolando as variáveis de velocidade angular e de corrente, e reescrevendo as derivadas das equações (2.6) e (2.7) na forma discreta, obtêm-se as equações (2.8) e (2.9):

$$\omega_k = -\frac{L_a}{\Delta T K_e}(i_{a_k} - i_{a_{k-1}}) - \frac{R_a}{K_e} i_{a_k} + \frac{V_{a_k}}{K_e} \tag{2.8}$$

$$i_{a_k} = \frac{b}{K_m}\omega_k + \frac{J}{\Delta T K_m}(\omega_k - \omega_{k-1}) \tag{2.9}$$

Por fim, as equações (2.8) e (2.9) podem ser escritas na forma de equações de estado, como é apresentado em (2.10). Essa forma facilita as operações matemáticas através de matrizes. Nota-se que K_m foi substituído por K_e, isso vem do fato de ambos os termos serem numericamente iguais no Sistema Internacional de Medidas (SI).

$$\begin{bmatrix} \omega_k \\ i_{a_k} \end{bmatrix} = \begin{bmatrix} -i_{a_k} & i_{a_{k-1}} & 0 & 0 & v_k \\ 0 & 0 & \omega_k & -\omega_{k-1} & 0 \end{bmatrix} \cdot \begin{bmatrix} \left(R_a + \frac{L_a}{\Delta T}\right)\cdot\frac{1}{K_e} \\ \left(\frac{L_a}{\Delta T K_e}\right) \\ \left(\frac{J}{\Delta T K_e} + \frac{b}{K_e}\right) \\ \left(\frac{J}{\Delta T K_e}\right) \\ \frac{1}{K_e} \end{bmatrix} \tag{2.10}$$

Onde ΔT é o intervalo de tempo entre o instante k e $k-1$.

Já as funções de transferência que relacionam a corrente com a tensão de entrada e a velocidade com os conjugados pode ser obtida através da Transformada de Laplace das equações (2.2) e (2.3):

$$V_a(s) = R_a I_a(s) + sL_a I_a(s) + K_e \omega(s) \tag{2.11}$$

$$K_m I_a(s) = T_m + sJ\omega(s) + b\omega(s) \tag{2.12}$$

A corrente de armadura $I_a(s)$ e a velocidade angular $\omega(s)$ podem ser obtidas através das equações (2.11) e (2.12):

$$I_a(s) = \frac{V_a(s) - K_e\omega(s)}{sL_a + R_a} \tag{2.13}$$

$$\omega(s) = \frac{K_m I_a(s) - T_m}{sJ + b} \tag{2.14}$$

2.2. Identificação de Sistemas

2.2.1. Mínimos Quadrados

Existem dois métodos principais para se determinar os parâmetros de um modelo: em batelada ou estimação recursiva. No primeiro, monta-se a matriz dos regressores (variáveis independentes) e resolve-se o problema de uma só vez, tendo posse de todos os dados de entrada e saída de um determinado sistema. O segundo caso ocorre quando os dados são medidos e disponibilizados sequencialmente.

O método dos Mínimos Quadrados é um recurso matemático que tem como objetivo encontrar uma função $g(x)$ que mais se aproxime de outra função $f(x)$. Um caso específico dessa necessidade ocorre quando a função $f(x)$ descreve um fenômeno real e deseja-se encontrar outra função $g(x)$ que melhore a aproximação, mas ainda represente o comportamento do fenômeno.

Uma aproximação pode ser feita através da equação (2.15), que exprime a função erro:

$$r(x) = f(x) - g(x) \tag{2.15}$$

Sendo $f(x)$ a função original e $g(x)$ a função que irá aproximar $f(x)$. A melhor aproximação da função $g(x)$ será aquela que minimizar o módulo da função erro. Dessa forma, busca-se minimizar o quadrado da função erro, conforme expresso na equação (2.16) [5]:

$$\min\left[\sum_x r^2(x)\right] \tag{2.16}$$

Um conjunto de funções do tipo $y = f(x)$ pode ser escrito como (2.17), onde N representa o número de restrições:

$$\begin{aligned} y_1 &= f(\chi_1) \\ y_2 &= f(\chi_2) \\ \vdots &= \quad \vdots \\ y_N &= f(\chi_N) \end{aligned} \tag{2.17}$$

No caso vetorial, $f(x):\Re^n \to \Re$ depende de uma matriz de n parâmetros, θ. Dessa forma, o conjunto de funções pode ser escrito como em (2.18):

$$\begin{aligned} y_1 &= f(x_1,\theta) \\ y_2 &= f(x_2,\theta) \\ \vdots &= \quad \vdots \\ y_N &= f(x_N,\theta) \end{aligned} \tag{2.18}$$

O conjunto de equações (2.18) ainda pode ser escrito da forma matricial (2.19):

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & x_n \end{bmatrix} \cdot \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \tag{2.19}$$

Ou, de forma simplificada (2.20):

$$Y = \varphi \cdot \theta \tag{2.20}$$

A matriz θ, que contém os parâmetros desejados, poderá ser determinada a partir da equação (2.21), desde que φ seja não singular:

$$\theta = \varphi^{-1} Y \tag{2.21}$$

No entanto, se N>n, ou seja, se houverem mais restrições N do que elementos n de θ, a matriz φ não será quadrada e consequentemente não poderá ser invertida. Entretanto, multiplicando-se ambos os lados da equação (2.21) por φ^T chega-se à equação (2.22), que é chamada de equação normal.

$$\varphi^T Y = \varphi^T \varphi \theta \tag{2.22}$$

Dessa forma, a matriz θ poderá ser determinada, para o caso sobredeterminado, a partir da equação (2.23):

$$\theta = \left[\varphi^T \varphi\right]^{-1} \varphi^T Y \tag{2.23}$$

2.2.2. Estimador Recursivo de Mínimos Quadrados (RMQ)

Esse método apresenta as vantagens de possibilitar resultados em tempo real, quando as variações do sistema são lentas, e facilita a resolução do problema numérico em relação ao método de batelada por necessitar menos recursos computacionais. Devido a estas características, juntamente com as vantagens oferecidas pelos mínimos quadrados, este foi o método escolhido para ser implementado no software de estimação.

Através do método recursivo, o valor da matriz θ_k pode ser determinado pela equação (2.24):

$$\theta_k = \theta_{k-1} + K_k \eta(k) \tag{2.24}$$

Onde K_k é uma matriz de ganho, definida por (2.25):

$$K_k = P_k \varphi(k-1) \tag{2.25}$$

E $\eta(k)$ é a inovação no instante k, e é expressa pela equação (2.26):

$$\eta(k) = y(k) - \varphi^T(k-1)\hat{\theta}_{k-1} \tag{2.26}$$

A partir da equação (2.24) percebe-se que o valor mais atualizado da matriz θ_k é simplesmente o seu valor anterior acrescido de um valor de ajuste, que é função dos novos dados e de uma ponderação dada a esse ajuste (inovação).

No entanto, para se utilizar o algoritmo recursivo ainda é necessário calcular P_k, que pode ser feito através da equação (2.27):

$$P_k = P_{k-1}\varphi(k-1)(\varphi^T(k-1)P_{k-1}\varphi(k-1)+1)^{-1}\varphi^T(k-1)P_{k-1} \tag{2.27}$$

Combinando as equações (2.24), (2.25), (2.26) e (2.27) de forma a isolar os termos K_k, $\hat{\theta}_k$ e P_k, e considerando-se que $\varphi_k = \varphi(k-1)$, obtém-se o estimador recursivo de mínimos quadrados, expresso em (2.28):

$$\left.\begin{aligned} K_k &= \frac{P_{k-1}\varphi_k}{\varphi_k^T P_{k-1}\varphi_k + 1} \\ \hat{\theta}_k &= \hat{\theta}_{k-1} + K_k\left[y(k) - \varphi_k^T\theta_{k-1}\right] \\ P_k &= P_{k-1} - K_k\varphi_k^T P_{k-1} \end{aligned}\right\} \tag{2.28}$$

Deve-se observar que, por se tratar de um método iterativo, é necessário inicializar o vetor θ_k, por exemplo, com um valor nulo ou próximo ao esperado. [6]

2.2.3. Outros Estimadores Recursivos

Existem vários algoritmos recursivos que podem ser empregados na estimação de parâmetros, entre eles pode-se citar o estimador recursivo estendido de mínimos quadrados, de variáveis instrumentais e de aproximação estocástica.

- Estimador recursivo estendido de mínimos quadrados: pode ser implementado através do conjunto de equações (2.29):

$$\begin{cases} K_k = P_{k-1}\varphi_k\left[\varphi_k^T P_{k-1}\varphi_k + 1\right]^{-1}; \\ \hat{\theta}_k = \hat{\theta}_{k-1} + K_k\left[y(k) - \varphi_k^T\hat{\theta}_{k-1}\right]; \\ P_k = P_{k-1} - K_k\varphi_k^T P_{k-1}; \\ \xi(k) = y(k) - \varphi_k^T\hat{\theta}_k \end{cases} \tag{2.29}$$

Onde o termo ξ representa o resíduo na iteração *k*.

- Estimador recursivo de variáveis instrumentais: pode ser implementado através do conjunto de equações (2.30):

$$\begin{cases} M_k = M_{k-1} - M_{k-1}z_k\left[1 + \varphi_k^T M_{k-1}z_k\right]^{-1}\varphi_k^T M_{k-1} \\ K_k = M_k z_k; \\ \hat{\theta}_k = \hat{\theta}_{k-1} + K_k\left[y(k) - \varphi_k^T\hat{\theta}_{k-1}\right]; \\ \xi(k) = y(k) - \varphi_k^T\hat{\theta}_k \end{cases} \tag{2.30}$$

Onde z_k é o vetor de variáveis instrumentais na *k*-ésima iteração.

- Estimador recursivo de aproximação estocástica: pode ser implementado através do conjunto de equações (2.31):

$$\begin{cases} K_k = \gamma_k - \varphi_k; \\ \hat{\theta}_k = \hat{\theta}_{k-1} + K_k\left[y(k) - \varphi_k^T\hat{\theta}_{k-1}\right]; \\ \xi(k) = y(k) - \varphi_k^T\hat{\theta}_k \end{cases} \tag{2.31}$$

Onde γ_k pode ser definido por $\gamma_k = Ck - \alpha$, sendo $0{,}5 \le \alpha \le 1$ e C uma constante positiva. γ_k deve ser escolhido de forma que $\lim_{k\to\infty}\gamma_k = 0$, porém para

valores de γ_k muito pequenos, a taxa de convergência do estimador também será baixa. [6]

2.3. Obtenção de Parâmetros da Máquina CC

2.3.1. Modelo Dinâmico e Mínimos Quadrados

Conhecendo-se as equações de estado que descrevem o funcionamento dinâmico de uma máquina CC, conforme mostra a equação (2.10), uma forma de se obter alguns parâmetros elétricos e mecânicos é utilizando dados de um ensaio dinâmico como entrada do sistema de equações. Assim, empregando métodos de regressão é possível estimar o valor da matriz θ.

Um método de regressão bastante útil para este fim é o dos mínimos quadrados, que tem por objetivo minimizar o módulo do erro. Isso é interessante para o caso onde há presença de ruído branco (interferências de alta frequência, chaveamentos, vibrações).

2.3.2. Modelo Dinâmico Visto Como Dois Sistemas de Primeira Ordem Desacoplados [1]

Parte das equações (2.6) e (2.7) e utiliza recursos matemáticos um pouco diferentes. Definindo a corrente de armadura $i_a(t)$ e a velocidade angular $\omega(t)$ como estados, as equações de estado baseadas em (2.6) e (2.7) ficam como expresso na equação (2.32):

$$\begin{bmatrix} \frac{d}{dt} i_a(t) \\ \frac{d}{dt} \omega(t) \end{bmatrix} = \begin{bmatrix} -\frac{R_a}{L_a} & -\frac{K_e}{L_a} \\ \frac{K_m}{J} & -\frac{b}{J} \end{bmatrix} \begin{bmatrix} i_a(t) \\ \omega(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{L_a} \\ 0 \end{bmatrix} V_a(t) \qquad (2.32)$$

Mas, como será necessário o uso de um tacômetro, cuja tensão nos seus terminais é proporcional a $\omega(t)$, como expressa a equação (2.33)

$$V_t(t) = K_t \omega(t) \qquad (2.33)$$

A equação (2.32) fica:

$$L_a \frac{d}{dt} i_a(t) + R_a i_a(t) = V_a(t) - \frac{K_e}{K_t} V_t(t) \tag{2.34}$$

$$J \frac{d}{dt} V_t(t) + b V_t(t) = K_m K_t i_a(t) \tag{2.35}$$

E definindo

$$\begin{cases} u_e(t) = V_a(t) - (K_e / K_t) V_t(t) \\ u_m(t) = K_m K_t i_a(t) \end{cases} \tag{2.36}$$

Obtém-se:

$$\begin{cases} L_a \frac{d}{dt} i_a(t) + R_a i_a(t) = u_e(t) \\ J \frac{d}{dt} V_t(t) + b V_t(t) = u_m(t) \end{cases} \tag{2.37}$$

O conjunto de equações (2.37) representa dois sistemas de primeira ordem desacoplados, o que permite definir um sistema elétrico (E) e outro mecânico (M):

$$(E)\begin{cases} \dot{x}_e(t) = A_e x_e(t) + B_e u_e(t) \\ y_e(t) = C_e x_e(t) \end{cases} \tag{2.38}$$

Onde: $x_e(t) = i_a(t)$, $A_e = -R_a / L_a$, $B_e = 1/L_a$ e $C_e = 1$

$$(M)\begin{cases} \dot{x}_m(t) = A_m x_m(t) + B_m u_m(t) \\ y_m(t) = C_m x_m(t) \end{cases} \tag{2.39}$$

Onde: $x_m(t) = V_t\ (t)$, $A_m = -b/J$, $B_m = 1$ e $C_m = 1$.

Sabendo-se que as equações de estado para um sistema discreto equivalente a um sistema de primeira ordem contínuo são dadas por (2.40):

$$\begin{cases} x(t_{k+1}) = \Phi x(t_k) + \Gamma u(t_k) \\ y(t_k) = C x(t_k) \end{cases} \tag{2.40}$$

Onde t_k e t_{k+1} denotam os instantes de amostragem e sendo $h = t_{k+1} - t_k$, tem-se que:

$$\Phi = e^{Ah} = e^{-h/\tau} \tag{2.41}$$

$$\Gamma = \int_0^h e^{Ax} B dx = \frac{K}{\tau} \int_0^h e^{-x/\tau} dx = K(1-\Phi) \tag{2.42}$$

Onde $A = -1/\tau$, $B = K/\tau$ e $C = 1$.

Por fim, as equações acima podem ser convertidas na equação a diferenças finitas:

$$y(t_k) = \Phi y(t_{k-1}) + \Gamma u(t_{k-1}) \tag{2.43}$$

Calculando-se Φ e Γ, o ganho K e a constante de tempo τ do modelo do sistema podem ser calculados utilizando-se as relações dadas em (2.41) e (2.42).

Fazendo-se a aquisição dos sinais de saída $y(t_k)$ e de entrada $u(t_k)$, para k=0,1,...,q, pode-se escrever:

$$\begin{cases} y(t_1) = \Phi y(t_0) + \Gamma u(t_0) \\ y(t_2) = \Phi y(t_1) + \Gamma u(t_1) \\ \vdots \\ y(t_q) = \Phi y(t_{q-1}) + \Gamma u(t_{q-1}) \end{cases} \tag{2.44}$$

Que podem ser agrupadas no formato matricial, expresso em (2.45):

$$\begin{bmatrix} y(t_1) \\ y(t_2) \\ \vdots \\ y(t_q) \end{bmatrix} = \begin{bmatrix} y(t_0) & u(t_0) \\ y(t_1) & u(t_1) \\ \vdots & \vdots \\ y(t_{q-1}) & u(t_{q-1}) \end{bmatrix} \begin{bmatrix} \Phi \\ \Gamma \end{bmatrix} \tag{2.45}$$

Ou, de forma simplificada:

$$b = Ax \tag{2.46}$$

O valor da matriz x, que contém os valores de Φ e Γ, pode então ser determinada a partir do método dos mínimos quadrados, fazendo-se:

$$x = \left(A^T A\right)^{-1} A^T b \tag{2.48}$$

Para a determinação da região linear do motor, aplica-se a ele degraus de tensão de amplitude V_{a_i}, $i = 0,1,2,...,n$ e mede-se os respectivos valores de tensão de saída no tacômetro V_{t_i}, $i = 0,1,2,...,n$. Em seguida, Formam-se os pares cartesianos $\left(V_{a_i}, V_{t_i}\right)$ e ajustam-se os pontos obtidos por um polinômio de grau q (arbitrário):

$$V_t = A_0 V_a^q + A_1 V_a^{q-1} + ... + A_{q-1} V_a + a_q \tag{2.49}$$

Com os dados obtidos, monta-se o sistema de equações (2.50):

$$\begin{bmatrix} V_{t_1} \\ V_{t_{21}} \\ \vdots \\ V_{t_{n1}} \end{bmatrix} = \begin{bmatrix} V_{a_1}^{q} & V_{a_1}^{q-1} & \cdots & V_{a_1} \\ V_{a_2}^{q} & V_{a_2}^{q-1} & \cdots & V_{a_{21}} \\ \vdots & \vdots & & \vdots \\ V_{a_n}^{q} & V_{an}^{q-1} & \cdots & V_{a_{n1}} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{q-1} \end{bmatrix} \tag{2.50}$$

Que pode ser facilmente solucionado pelo método dos mínimos quadrados. A região linear corresponderá ao intervalo no qual o gráfico da derivada de V_t em relação a V_a é aproximadamente paralela ao eixo das abscissas.

As constantes do motor K_m e de força contra-eletromotriz K_e possuem o mesmo valor e são obtidas fazendo o motor funcionar como gerador. A tensão medida nos seus terminais, nessas condições e sem carga, é praticamente igual à força contra-eletromotriz E_a. Assim, a determinação de K_e é feita ajustando-se os pontos (V_{t_i}, E_a) por uma reta que passa pela origem, onde K_e / K_t corresponde ao coeficiente de inclinação da reta.

Aplicando-se como sinal de entrada $V_a(t)$ pulsos de largura aleatória e fazendo a aquisição do sinal $V_t(t)$, para o sistema (E):

$$y_e(t_k) = \Phi_e y_e(t_{k-1}) + \Gamma_e u_e(t_{k-1}) \tag{2.51}$$

Sendo $\Phi_e = e^{-\frac{R_a}{L_a}h}$, $\Gamma_e = 1/R_a(1-\Phi_e)$ e $y_e(t_k) = i_a(t_k)$.

Formando-se um conjunto de equações a partir de ensaios e baseando-se na equação acima, Φ_e e Γ_e podem ser determinados empregando o método dos mínimos quadrados, conforme já exposto. Assim, R_a e L_a podem ser obtidos através das equações (2.52) e (2.53):

$$R_a = \frac{1-\Phi_e}{\Gamma_e} \tag{2.52}$$

$$L_a = -\frac{R_a h}{\ln(\Phi e)} \tag{2.53}$$

De forma análoga, para o sistema (M) tem-se:

$$y_m(t_k) = \Phi_m y_m(t_{k-1}) + \Gamma_m u_m(t_{k-1}) \tag{2.54}$$

De onde se obtém Φ_m e Γ_m. Assim, b e J podem ser encontrados a partir das equações:

$$b = \frac{K_t K_m (1 - \Phi_m)}{\Gamma_m} \tag{2.55}$$

$$J = -\frac{bh}{\ln(1 - \Phi_m)} \tag{2.56}$$

2.3.3. Método de Momentos [7]

Esse método parte do princípio que a resposta a um impulso é caracterizada por infinitos momentos, onde apenas os primeiros são necessários para a função densidade de probabilidade.

Considerando um sistema linear estável, caracterizado pela resposta ao impulso $h(t)$, tem-se:

$$H(s) = \frac{B(s)}{A(s)} \tag{2.57}$$

A equação (2.57) pode ser expandida em séries de Taylor na vizinhança de $s = j\omega_0$ e, para $\omega = 0$, resulta em:

$$H(s) = \sum_{n=0}^{\infty} (-1)^n s^n \overline{A}_n(h) \tag{2.58}$$

Onde,

$$\overline{A}_n(h) = \int_0^{+\infty} \frac{t^n}{n!} h(t) dt \tag{2.59}$$

Tomando $y(t)$ como a resposta ao degrau do sistema em estudo, propõe-se a sua identificação através da equação (2.60):

$$H(s) = \frac{Y(s)}{E(s)} = K_1 \cdot \frac{1 + b_1 s + b_2 s^2 + \ldots + b_m s^m}{1 + a_1 s + a_2 s^2 + \ldots + a_n s^n} \tag{2.60}$$

K_1 pode ser determinado a partir do teorema do valor final, fazendo-se:

$$K_1 = \lim_{t \to \infty} y(t) = y(\infty) \tag{2.61}$$

Considerando a função erro como

$$\varepsilon(t) = K_1 - y(t) \tag{2.62}$$

E introduzindo a transformada de Laplace na equação de $H(s)$ e fazendo as simplificações necessárias já, resulta na equação (2.63).

$$\varepsilon(s) = \sum_{n=0}^{\infty} (-1)^n s^n A_n(\varepsilon) \tag{2.63}$$

De acordo com as equações (2.58) e (2.63), pode-se deduzir os coeficientes da função de transferência H(s) resolvendo o sistema matricial (2.64):

$$\begin{bmatrix} K_1(a_1 - b_1) \\ K_1(a_2 - b_2) \\ \vdots \\ K_1(a_{n+1} - b_{n+1}) \end{bmatrix} = \begin{bmatrix} A_0(\varepsilon) & 0 & \cdots & 0 \\ -A_1(\varepsilon) & A_0(\varepsilon) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_n(\varepsilon) & \cdots & \cdots & \ddots \end{bmatrix} \cdot \begin{bmatrix} 1 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \tag{2.64}$$

A função de transferência, para o caso específico do motor, é expressa por (2.65).

$$H(s) = K_1 \cdot \frac{1 + b_1 s}{1 + a_1 s + a_2 s^2} \tag{2.65}$$

E então o sistema matricial pode ser reescrito como em (2.66):

$$\begin{bmatrix} K_1(a_1 - b_1) \\ K_1 a_2 \\ 0 \end{bmatrix} = \begin{bmatrix} A_0 & 0 & 0 \\ -A_1 & A_0 & 0 \\ A_2 & -A_1 & A_0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ a_1 \\ a_2 \end{bmatrix} \tag{2.66}$$

Resolvendo-se o sistema acima, obtêm-se os coeficientes a_1, a_2 e b_1, como expresso nas equações (2.67), (2.68) e (2.69):

$$a_1 = \frac{A_1 A_0 - K_1 A_2}{A_0^2 - K_1 A_1} \tag{2.67}$$

$$a_2 = \frac{-A_1 + a_1 A_0}{K_1} \tag{2.68}$$

$$b_1 = a_1 - \frac{A_0}{K_1} \tag{2.69}$$

Para determinar os parâmetros do motor, aplica-se um sinal de tensão em degrau ΔU_a e medem-se os valores iniciais e finais de corrente e velocidade. Assim, pode-se escrever:

$$U_{a_0} = R_a i_{a_0} + K\omega_0 \tag{2.70}$$

$$U_{a_1} = R_a i_{a_1} + K\omega_1 \tag{2.71}$$

Das equações (2.70) e (2.71), isolando *K*:

$$K = \frac{U_{a1} - \frac{i_{a1}}{i_{a0}} U_{a0}}{\omega_1 - \frac{i_{a1}}{i_{a1}} \omega_0} \tag{2.72}$$

E da variação entre os valores inicial e final, conclui-se que:

$$\Delta U_a = R_a \Delta i_a + K\Delta\omega \tag{2.73}$$

Com isso, pode-se reescrever as funções de transferência do motor como:

$$\Delta U_a = R_a \Delta i_a + L_a \frac{di_a}{dt} + K\Delta\omega \tag{2.74}$$

$$K\Delta i_a = J \frac{d\Delta\omega}{dt} + b\Delta\omega \tag{2.75}$$

A função de transferência da corrente de armadura é dada por:

$$H_1(s) = \frac{\Delta i_a(s)}{\Delta U_a(s)} = \frac{\frac{b}{K^2 + R_a b}\left(1 + \frac{Js}{b}\right)}{1 + \tau_m \tau_e s^2 + (\tau_m + \mu\tau_e)s} \tag{2.76}$$

E a função de transferência da velocidade angular é dada por:

$$H_2(s) = \frac{\Delta\omega(s)}{\Delta U_a(s)} = \frac{\frac{K}{K^2 + R_a b}}{1 + \tau_m \tau_e s^2 + (\tau_m + \mu\tau_e)s} \tag{2.77}$$

Onde τ_e é a constante elétrica de tempo expressa pela equação (2.78):

$$\tau_e = \frac{L_a}{R_a} \tag{2.78}$$

τ_m é a constante mecânica de tempo expressa pela equação (2.79):

$$\tau_m = \frac{R_a J}{K^2 + R_a b} \tag{2.79}$$

E μ é um coeficiente geralmente desprezível, dado pela equação (2.80):

$$\mu = \frac{R_a f}{K^2 + R_a b} \tag{2.80}$$

A partir das equações de transferência (2.76) e (2.77), pode-se deduzir b:

$$b = \frac{K\Delta i_a(\infty)}{\Delta\omega(\infty)} \tag{2.81}$$

Pela comparação dos denominadores de $H_1(s)$ e $H_2(s)$ com o denominador de $H(s)$, chega-se aos valores de a_1 e a_2:

$$a_1 = \tau_m + \mu\tau_e \tag{2.82}$$

$$a_2 = \tau_m\tau_e \tag{2.83}$$

Das equações acima se chega à equação de segunda ordem:

$$\mu\tau_e^2 - a_1\tau_e + a_2 = 0 \tag{2.84}$$

Esta equação resulta em duas raízes, sendo uma positiva e outra negativa (descartada).

Assim, é possível calcular τ_m, L_a e J.

2.3.4. Redes Neurais

Alguns pesquisadores têm adotado o método de identificação através de redes neurais. Este método permite que sejam memorizadas e emuladas as características do sistema. Isso permite que a ferramenta seja "treinada".

O procedimento descrito a seguir é baseado no trabalho de Arif e Mazin [8].

Embora existam várias estruturas possíveis para mapear redes neurais, concluiu-se que Algoritmos Genéticos (GAs) são os mais eficientes para encontrar a estrutura mínima capaz de emular corretamente o sistema dinâmico desconhecido.

A estrutura fundamental de uma Rede Neural de Elman é mostrada na Figura 2.5.

Além das entradas e saídas, a rede Elman conta com uma unidade "caixa preta" X_k e uma unidade de composição X^C.

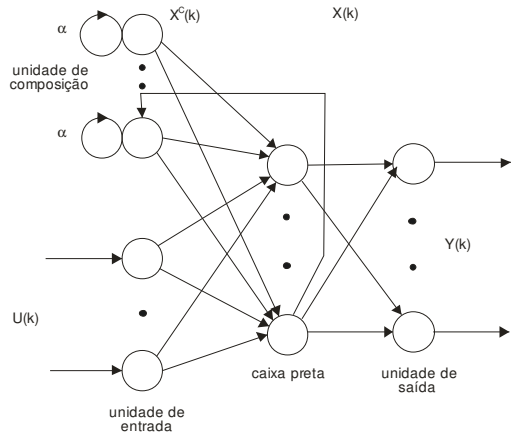


Figura 2.5: Estrutura fundamental da Rede Neural de Elman. (Fonte: Al-Quassar e Othman, 2008 – pag. 192)

As equações dinâmicas da rede podem ser escritas como:

$$X(k) = W^{XC} X^{C}(k) + W^{XU} U(k-1) \tag{2.85}$$

$$X^{C}(k) = X(k-1) \tag{2.86}$$

$$Y(k) = W^{YX} X(k) \tag{2.87}$$

Onde as matrizes W^{XC}, W^{XU} e W^{YX} definem a interconexão entre a unidade de composição e a "caixa preta", a entrada e a "caixa preta", e entre a "caixa preta" e a unidade de saída, respectivamente. As equações (2.85), (2.86) e (2.87) representam as equações de estado que descrevem o sistema dinâmico.

A fim de encontrar os valores das matrizes que definem a interconexão, a ferramenta GAs é utilizada para minimizar o erro quadrático entre a saída da rede neural e a saída atual do sistema.

Um motor CC controlado pela armadura com excitação fixada, pode ser descrito pelas equações de estado:

$$\dot{X} = FX + GU \tag{2.88}$$

$$Y = CX \tag{2.89}$$

Onde:

$$F = \begin{bmatrix} \frac{-b}{J} & \frac{K_m}{J} \\ \frac{-K_e}{L_a} & \frac{-R_a}{L_a} \end{bmatrix} \tag{2.90}$$

$$G = \begin{bmatrix} 0 \\ \dfrac{1}{L_a} \end{bmatrix} \tag{2.91}$$

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{2.92}$$

$$X = \begin{bmatrix} \omega \\ i_a \end{bmatrix} \tag{2.93}$$

Sendo b a constante de atrito viscoso, J o momento de inércia, K_m a constante de torque, K_e a constante de força contra-eletromotriz, R_a a resistência da armadura, L_a a indutância da armadura, i_a a corrente de armadura, ω a velocidade angular do motor e U a tensão de armadura.

As equações de estado equivalentes para tempo discreto são dadas pelas equações (2.94) e (2.95):

$$X_{k+1} = AX_k + bU_k \tag{2.94}$$

$$Y_k = CX_k \tag{2.95}$$

No qual as matrizes A e B podem ser expandidas:

$$A = 1 + FT + F^2 \frac{T^2}{2!} + ... + \frac{T^i}{i!} F^i \tag{2.96}$$

$$B = TG + \frac{T^2}{2!} FG + ... + \frac{T^i}{i!} F^{i-1} \tag{2.97}$$

Usualmente o tempo de amostragem ΔT é escolhido de forma a ser menor que 1. Dessa forma, as equações acima podem ser truncadas até o termo T:

$$A = \begin{bmatrix} 1 - \dfrac{b\Delta T}{J} & \dfrac{K_m \Delta T}{J} \\ \dfrac{-K_e \Delta T}{L_a} & 1 - \dfrac{R_a \Delta T}{L_a} \end{bmatrix} \tag{2.98}$$

$$B = \begin{bmatrix} 0 \\ \dfrac{\Delta T}{L_a} \end{bmatrix} \tag{2.99}$$

Ainda comparando-se as equações (2.85) e (2.87) com as equações (2.94) e (2.95), tem-se que:

$$W^{XC} = \begin{bmatrix} W_{11}^{XC} & W_{12}^{XC} \\ W_{21}^{XC} & W_{22}^{XC} \end{bmatrix} = A = \begin{bmatrix} 1 - \dfrac{B\Delta T}{J} & \dfrac{K_m \Delta T}{J} \\ \dfrac{-K_e \Delta T}{L_a} & 1 - \dfrac{R_a \Delta T}{L_a} \end{bmatrix} \tag{2.100}$$

$$W^{XU} = \begin{bmatrix} W_{11}^{XU} \\ W_{21}^{XU} \end{bmatrix} = B = \begin{bmatrix} 0 \\ \dfrac{\Delta T}{L_a} \end{bmatrix} \tag{2.101}$$

$$W^{YX} = \begin{bmatrix} W_{11}^{YX} & W_{12}^{YX} \end{bmatrix} = C = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{2.102}$$

Este método prevê que a resistência da armadura R_a seja conhecida, uma vez que pode ser obtida facilmente através de um ohmímetro. Os demais parâmetros (J, K_m, K_e, b e L_a) podem então ser estimados.

2.3.5. Ensaio Convencional

O procedimento descrito a seguir é baseado em ensaios convencionais e equações clássicas que regem o funcionamento de uma máquina CC.

A resistência de armadura, para um motor CC com excitação independente, pode ser obtida através do ensaio de rotor bloqueado. Fazendo com que a velocidade angular ω seja nula, a partir da equação (2.2) que descreve o circuito elétrico equivalente da armadura, obtém-se:

$$V_a(t) = R_a i_a(t) + L_a \frac{di_a(t)}{dt} \tag{2.103}$$

Sabendo-se que em regime permanente ($t = \infty$) a derivada da corrente é nula, pode-se isolar o termo R_a:

$$R_a = \frac{V_a}{i_a} \tag{2.104}$$

Para este ensaio, aplica-se uma tensão de armadura baixa e mede-se a corrente resultante. É necessário ter o cuidado para que a corrente não exceda a corrente nominal do motor.

Através deste mesmo ensaio, é possível obter a indutância da armadura L_a, medindo-se a constante de tempo do circuito. A constante de tempo é definida como, ao aplicar-se uma excitação em tensão do tipo degrau na armadura, o tempo necessário para a corrente atingir 63,2% do seu valor em regime.

Uma vez que a constante de tempo elétrico da armadura é dada pela equação (2.105):

$$\tau_a = \frac{L_a}{R_a} \tag{2.105}$$

Conhecendo-se a resistência de armadura, obtém-se L_a a partir da equação (2.106):

$$L_a = \tau_a R_a \tag{2.106}$$

A constante k_e é obtida com a máquina CC operando como um gerador, acionado com velocidade constante e sem carga. Como a força eletromotriz é dada pela equação (2.107):

$$e_a = k_e \omega \tag{2.107}$$

E como, nessas condições, $e_a = V_a$, pode-se isolar a constante k_e, chegando à equação (2.108):

$$k_e = \frac{V_a}{\omega} \tag{2.108}$$

Novamente excitando-se a armadura com uma tensão em degrau, porém com o rotor livre, define-se a constante de tempo eletromecânica τ_{em} como o tempo necessário para o motor atingir 63,2% da rotação em regime. Conhecendo-se τ_{em}, R_a e k_e, pode-se definir o momento de inércia como:

$$J = \frac{\tau_{em} k_e^{\,2}}{R_a} \tag{2.109}$$

Ainda a partir da constante de tempo eletromecânica é possível encontrar o coeficiente de atrito viscoso b:

$$b = \frac{J}{\tau_{em}} \tag{2.110}$$

3. MATERIAIS E MÉTODOS

Para alcançar os objetivos deste trabalho, é necessário fazer o acionamento do motor e controle da carga mecânica aplicada ao seu eixo, assim como monitorar o seu funcionamento.

Todo o sistema deverá ser operado apenas através de um micro-computador PC (do inglês *personal computer*). Dessa forma, o PC irá se comunicar com um microcontrolador através da comunicação USB e este, fará a interface com o motor, com as cargas e com os demais sensores.

As grandezas que serão monitoradas para obtenção dos parâmetros do motor são a velocidade angular, a corrente e a tensão aplicadas ao motor e o torque desenvolvido. A tensão gerada, embora não seja diretamente necessária para alcançar o objetivo final, é útil para evitar possíveis danos ao sistema que irá gerar a carga mecânica.

A Figura 3.1 mostra a forma como cada parte do hardware se relaciona entre si.

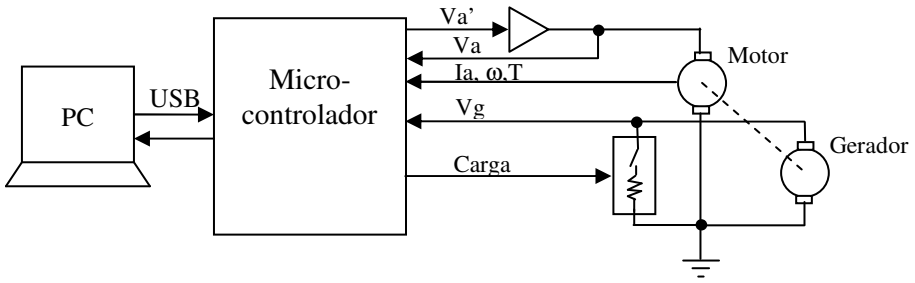


Figura 3.1: Diagrama simplificado hardware.

No computador, um software especialmente desenvolvido para este fim, faz o acionamento do motor e das cargas mediante comandos do usuário. Logo em seguida, faz a leitura dos dados enviados pelo microcontrolador e calcula os parâmetros desejados.

A metodologia empregada aqui para a obtenção dos parâmetros do motor de corrente contínua é baseada nas equações que descrevem o seu modelo dinâmico. Essas equações levam em consideração as derivadas da velocidade angular e da corrente de armadura, de forma que estas não podem ser estáveis (regime permanente). Uma forma de se obter tais derivadas não nulas é variando-se a excitação do motor. Neste trabalho será aplicada uma excitação em tensão do tipo degrau e o transitório, desde o repouso até o motor entrar em regime permanente, será monitorado.

Um dos objetivos do sistema de aquisição é fazer a amostragem dos sinais, principalmente, de tensão, corrente e velocidade angular, adquirindo o maior número de pontos possível durante o transitório.

As seções a seguir detalham cada parte do hardware e do software desenvolvidos.

3.1. Hardware

O hardware é composto pelas seguintes partes:

- Fonte de alimentação;
- Circuito principal microcontrolado;
- Condicionador de sinais para medição do torque;
- Condicionador de sinais para medição de corrente;
- Conversor UART – USB;
- Conjunto de cargas resistivas;
- Base da parte mecânica/eletromecânica;
- Microcomputador (PC).

3.1.1. Fonte de Alimentação

A fonte de alimentação é a parte responsável por fornecer energia elétrica a todo o conjunto (exceto o microcomputador). Ela tem a função de alimentar o motor e os demais circuitos eletrônicos.

Como foi definido que o motor a ser ensaiado poderá ser de até 24V e 150W, a fonte deverá ter uma saída CC com uma tensão ligeiramente acima de 24V, para compensar quedas de tensão no circuito de acionamento. Definiu-se para isto uma tensão de 30V.

A capacidade de corrente em regime permanente pode ser definida através da lei da potência (3.1):

$$P = V \cdot I \tag{3.1}$$

Onde P representa a potência em Watt, V é a tensão aplicada em Volt e I é a corrente que circula pelo circuito em Ampère.

Isolando-se I e aplicando os termos conhecidos:

$$I = \frac{150}{24} = 6{,}25A \tag{3.2}$$

Em função de custos, utilizou-se um transformador reaproveitado de um equipamento, que possui capacidade de corrente no secundário de até 13,5A, o que é satisfatório, pois já admite uma corrente até 115% acima do valor nominal previsto.

Para os circuitos eletrônicos, a fonte fornece uma tensão simétrica de ±12V. Alguns dos circuitos ainda rebaixam essa tensão para alimentação adequada de certos componentes, tais como o microcontrolador que opera com tensões na ordem de 3,3V. Tensões negativas são necessárias para evitar regiões não lineares próximas a 0V em alguns amplificadores. Nesta etapa não foi feito um levantamento de cargas por se tratar na grande maioria de componentes de baixo consumo, julgou-se 1A suficiente para alimentá-los.

Como já foi dito, o transformador foi reaproveitado de um equipamento eletrônico em desuso e precisou de alguns ajustes para adequação aos níveis de tensão desejados. Após realizar uma rebobinagem parcial, obteve-se uma tensão alternada de $30V_{RMS}$ em um circuito do secundário e $\pm 15V_{RMS}$ no outro. A Figura 3.2 mostra o transformador durante a sua rebobinagem.

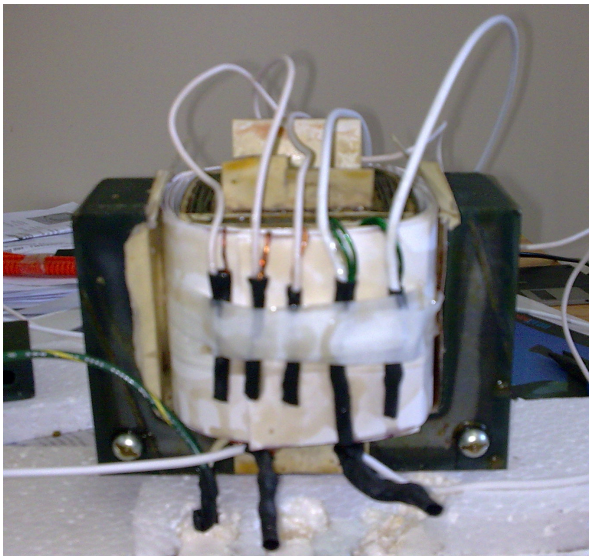


Figura 3.2: Rebobinagem do transformador

A tensão alternada dos secundários foi retificada em circuito tipo ponte de diodos e filtrada através de capacitores. Para as tensões de ±12V ainda foram utilizados os reguladores de tensão eletrônicos LM7812 e LM7912.

A Figura 3.3 mostra a fonte de alimentação montada.

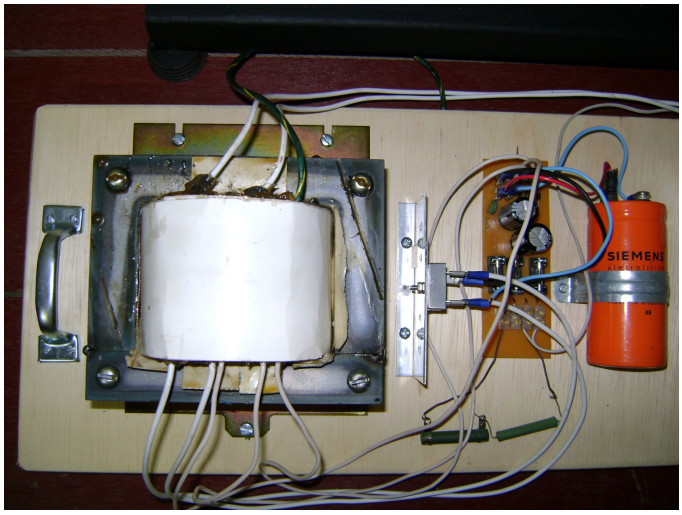


Figura 3.3: Fonte de alimentação

3.1.2. Placa de aquisição e comunicação

Essa placa é a responsável por adquirir os vários sinais desejados e enviá-los ao microcomputador assim como receber ordens do mesmo para efetuar o acionamento do motor e das cargas resistivas. Ela pode ser vista na Figura 3.4, onde aparece também o conversor USB - UART.

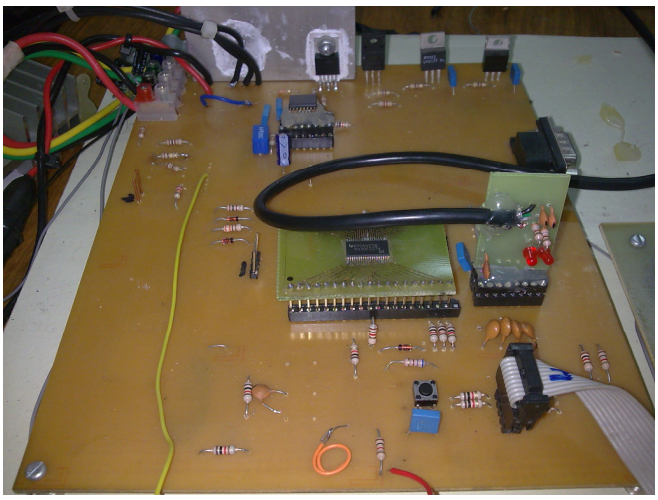


Figura 3.4: Circuito principal - aquisição e comunicação

O principal componente dessa placa é o microcontrolador. Ele recebe de forma serial os dados enviados pelo PC e interpreta qual é o comando solicitado, podendo acionar o motor, comunicar-se com o conversor digital-analógico, acionar as cargas resistivas ou realizar a conversão analógico-digital.

O microcontrolador escolhido foi o MSP430F2274 e a seguir são apresentadas algumas informações a respeito desse componente e porque utilizá-lo.

- **Microcontrolador MSP430F2274**

É um microcontrolador produzido pela Texas Instruments e cujas principais características estão listadas abaixo:

- Tensão de alimentação: de 1,8 V a 3,6 V;
- Frequência de operação de até 16 MHz;
- Interfaces de comunicação serial: UART, IRDA, SPI Síncrono e I^2C;
- Conversor analógico-digital (ADC) de 10 bits, até 200 ksps, com possibilidade de referência interna;
- 32 KB + 256 B de memória flash e 1 KB de memória RAM;
- Oscilador interno RC programável.

Além disso, pode ser programado através do protocolo *Spy-Bi-Wire*, permitindo o uso, por exemplo, dos módulos eZ430-F2013 ou eZ430-RF2500.

A identificação dos pinos do microcontrolador é indicada na figura 3.5. [Datasheet]

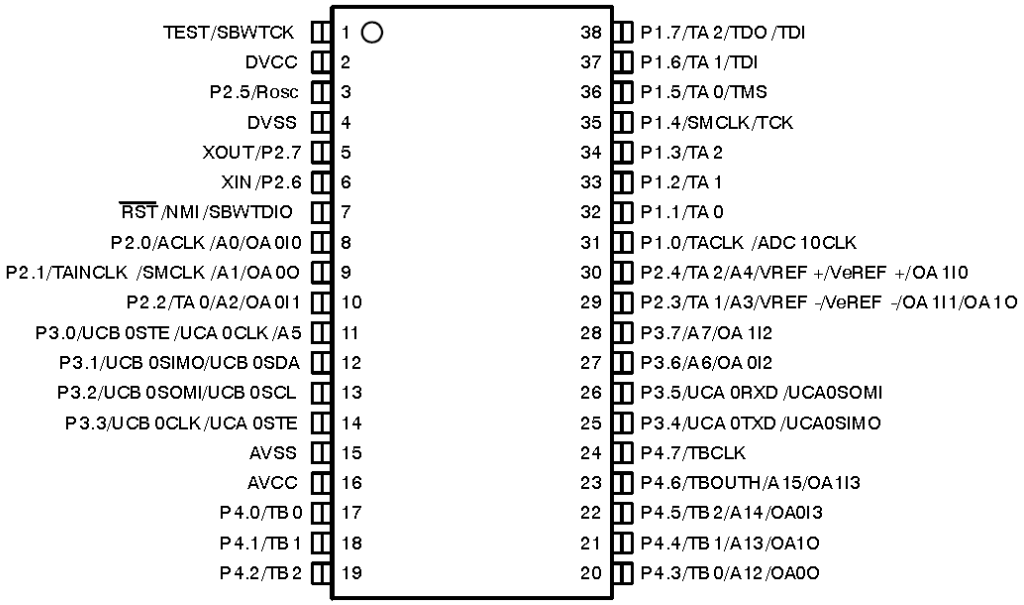


Figura 3.5: Identificação dos pinos do MSP430F2274. (Fonte: Datasheet Texas Instruments, 2007 – pag. 3)

Para que fosse possível atingir elevadas taxas de aquisição, foi necessário utilizar uma taxa de transferência de dados elevada. O microcontrolador se comunica através de comunicação UART a uma taxa de 921600 bps. Os terminais utilizados são o P3.4 (UCA 0TXD) e P3.5 (UCA 0RXD). Posteriormente, em outro circuito, o protocolo UART será convertido em USB, conforme explicado em 3.1.6.

As cargas resistivas são comandadas através da porta P4 do microcontrolador. Todos os seus pinos irão acionar os relés que farão o acionamento ou desligamento dos resistores.

A leitura dos sinais de interesse, como a tensão aplicada de fato ao motor, corrente de armadura, tensão gerada pelo gerador e pelo taco-gerador e o torque desenvolvido, é feita pelo conversor analógico-digital do próprio microcontrolador.

Os sinais de tensão aplicada ao motor, de tensão gerada pelo gerador e pelo taco-gerador, apesar de não necessitarem de circuitos condicionadores especiais, são superiores ao limite máximo das entradas analógicas do microcontrolador. Dessa forma, foram implementados divisores resistivos para estes sinais, junto com diodos Zener, para garantir que a tensão não ultrapasse o limite de 3,3V. Fato semelhante ocorre com o sinal proveniente do circuito condicionador da célula de carga, que também passa por uma redução da sua amplitude.

Os sinais referentes à corrente de entrada e tensão aplicada ao motor, força aplicada à célula de carga, rotação e tensão gerada, já condicionados, são aplicados

às entradas de conversão analógico-digital do microcontrolador A0, A1, A2, A3 e A4 respectivamente.

No microcontrolador, os pinos P1.0, P1.1, P1.2 e P1.3 são responsáveis pela comunicação com o conversor digital-analógico (DAC), que irá acionar o motor. A saída analógica do conversor é então amplificada em tensão e corrente, para atender às especificações do projeto.

O conversor digital-analógico empregado foi o DAC7614, produzido pela Burr-Brown e a seguir são apresentadas algumas de suas características e motivos que justificam a utilização.

- **Conversor DAC 7614**

É um conversor digital-analógico de quatro canais com resolução de 12 bits, comunicação serial e além disso, permite uma alimentação tanto unipolar quanto bipolar. A figura 3.6 mostra a identificação dos pinos do componente, em um invólucro tipo SOIC.

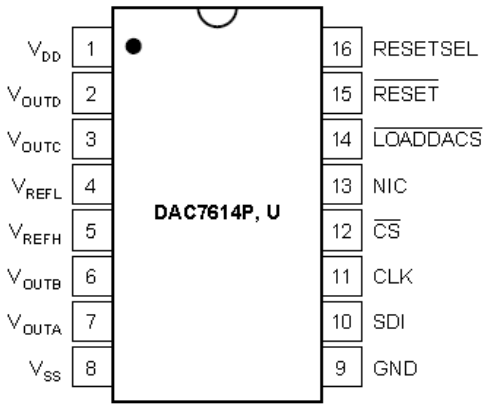


Figura 3.6: Identificação dos pinos do conversor.
(Fonte: Datasheet Burr-Brown, 1998 – pag. 4)

A topologia do amplificador utilizado é mostrada na figura 3.7.

O circuito utiliza essencialmente um amplificador operacional de potência, LM675 e um transistor bipolar darlington MJ11016, ambos dotados de dissipadores de calor. O dissipador do transistor ainda conta com o auxílio de ventilação forçada, através de um mini-ventilador (*cooler*).

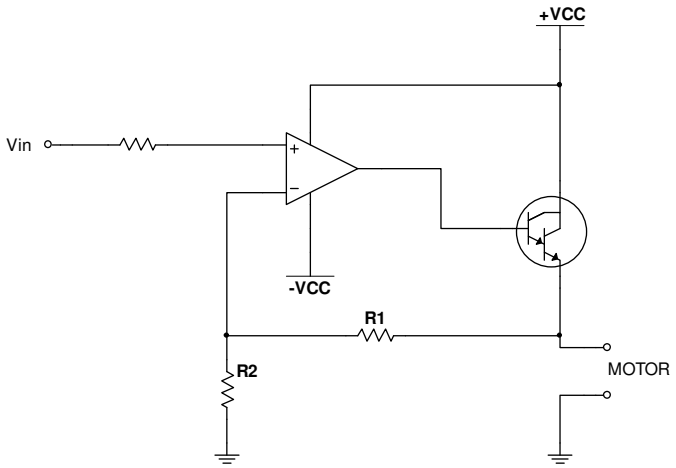


Figura 3.7: Topologia do amplificador de potência

- **Amplificador Operacional LM 675**

É um amplificador operacional de potência, capaz de operar com tensões de até 60 V e uma corrente de saída de até 3 A. Ainda possui uma capacidade de dissipação máxima de 20W e um tempo de subida (*slew rate*) de 8 V/µs. Esse componente é produzido com encapsulamento TO-220, específico para a utilização de dissipadores de calor.

A figura 3.8 apresenta a posição e função dos seus terminais.

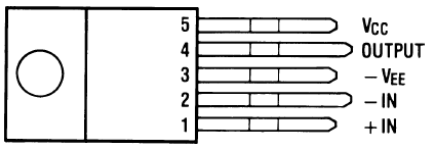


Figura 3.8: Terminais do LM 675 (Fonte: Datasheet National Semiconductor, 2004 – pag. 1)

- **Darlington MJ11016**

É um transistor bipolar de potência em configuração Darlington, que resulta em um elevado ganho de corrente. Entre as suas principais características estão:

- Máxima tensão coletor-emissor: 120V;
- Máxima corrente de coletor: 30A;
- Máxima dissipação de potência: 200W;
- Ganho CC: 200~1000;

3.1.3. Medição de Torque

Sabe-se dos princípios da física que o torque, na sua forma mais genérica, é dado pela seguinte equação:

$$T = F \cdot d \tag{3.3}$$

Isso significa que, medindo a força desenvolvida por um motor a uma distância conhecida do ponto central do seu eixo, é possível obter o torque. Dessa forma, o maior problema encontrado consiste em medir essa força.

Existem vários transdutores que podem ser empregados na medição dessa grandeza. Alguns dos principais:

- Transdutor piezelétrico: Apresentam a propriedade de gerar tensão elétrica ao serem submetidos a uma carga mecânica;
- Transdutor capacitivo: Considerando, por exemplo, duas placas metálicas e paralelas entre si, com um dielétrico entre elas susceptível de deformação quando submetido a uma carga mecânica, ocorrerá variação na capacitância do dispositivo;
- Extensômetro de resistência elétrica: Também chamado de *strain gage*, aplica o princípio da relação que existe na variação de resistência elétrica de um condutor quando submetido a uma deformação na sua região elástica.

Neste trabalho optou-se pelo emprego de uma célula de carga, que tem como princípio uma ponte de Wheatstone composta por extensômetros. Esse transdutor, assim como a própria célula de carga, será mais explorado a seguir.

A relação entre a variação relativa de resistência e a variação relativa da deformação é uma constante, sendo definida por Kelvin como:

$$\frac{\Delta R / R_0}{\Delta l / l_0} = \frac{\Delta R / R_0}{\varepsilon} = K \tag{3.4}$$

Na célula de carga do tipo viga biengastada, geralmente uma extremidade é fixada enquanto a outra fica sujeita à aplicação de forças. A localização dos extensômetros é representada por R1, R2, R3 e R4 e, quando sujeitos a uma força como indicada na Figura 3.9, R1 e R4 são submetidos a um esforço de tração enquanto R2 e R3 a um esforço de compressão.

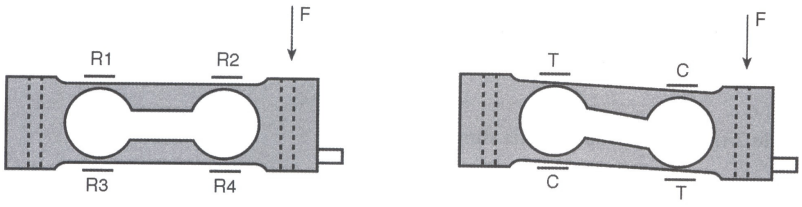


Figura 3.9: Localização dos extensômetros em uma célula de carga e o seu comportamento quando é aplicada uma força. (Fonte: Balbinot e Brusamarello, 2007 – pag. 189)

Geralmente utilizam-se quatro extensômetros iguais e conectados na forma de uma ponte de Wheatstone, como mostra a figura 3.10. Aqui fica clara a variação da resistência (ΔR) positiva ou negativa, em função da tração ou compressão a qual é submetida. A tensão de saída V_0 é dada pela equação (3.5). [9]

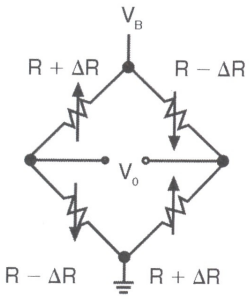


Figura 3.10: Ponte de Wheatstone para uma célula de carga com quatro extensômetros. (Fonte: Balbinot e Brusamarello, 2007 – pag. 208)

$$V_0 = V_B \frac{\Delta R}{R} \tag{3.5}$$

Usualmente, células de carga comerciais vêm com a sua sensibilidade expressa em mV/V, para a carga nominal, ou seja, uma célula que apresente sensibilidade de 2 mV/V e seja para uma carga nominal de até 5 kg, terá uma tensão de saída de 20 mV quando aplicado 10V na sua entrada e 5 kg de carga.

A célula de carga utilizada é do tipo GL-5, produzida pela Alfa Instrumentos, e possui capacidade nominal de 5 kg e sensibilidade de 2 mV/V. A capacidade mínima da célula foi obtida experimentalmente, onde constatou-se a necessidade de uma célula que suportasse aproximadamente 2 kg. Em função de custos e da maior robustez, importante inclusive para o transporte da bancada, optou-se pela célula de 5 kg.

A ponte de Wheatstone foi alimentada com 12V e a sua saída amplificada por um amplificador de instrumentação, INA126. Este sinal é posteriormente filtrado e passa por mais um estágio amplificador, onde é possível regular o ganho e o nível CC (*off-set*).

A Figura 3.11 mostra a célula de carga utilizada.

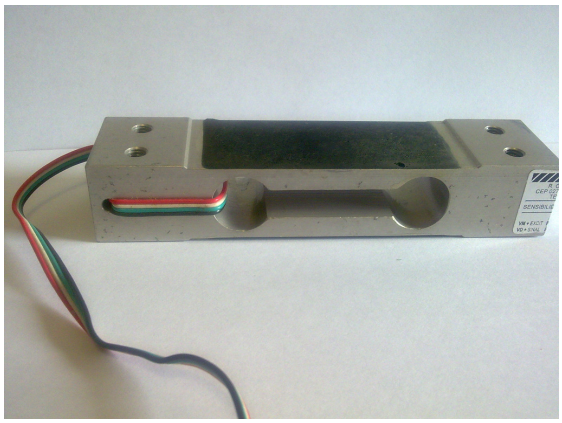


Figura 3.11: Célula de carga tipo *"single point"*.

É fácil perceber que a tensão de saída da ponte Wheatstone é da ordem de alguns mV, mais precisamente 4,8 mV/kg, o que pode ser calculado a partir da sensibilidade da célula. Isso deixa claro que o circuito condicionador de sinais também é um ponto crítico dessa etapa, pois é necessário um ganho diferencial elevado e grande rejeição a ruídos. Um elemento chave para tal é o amplificador de instrumentação.

O amplificador de instrumentação é a denominação para um circuito eletrônico, integrado ou não, que apresenta como características um elevado ganho, elevada taxa de rejeição em modo comum (imunidade a ruídos) e elevada impedância de entrada. Geralmente o ganho desse circuito é definido através de um único resistor.

Para o condicionamento do sinal de tensão da célula de carga foi empregado um amplificador de instrumentação integrado, o INA126, produzido pela Burr-Brown.

- **INA 126**

É um amplificador de instrumentação de precisão, com ganho variável entre 5 V/V e 10.000 V/V através de um resistor e apresenta pequena variação em função da temperatura. A Figura 3.12 exibe a representação simplificada do seu circuito interno e a respectiva função dos terminais.

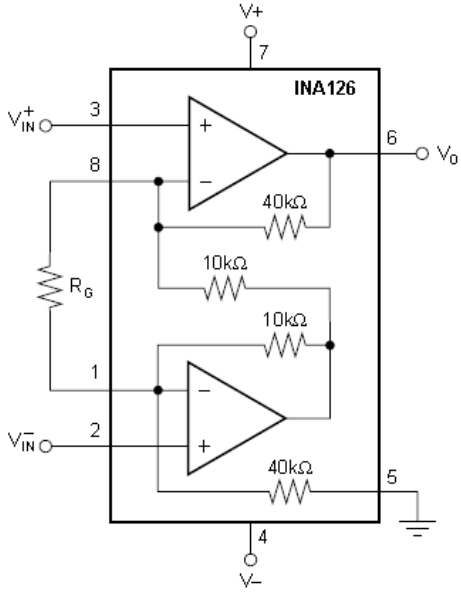


Figura 3.12: Representação do circuito interno do INA 126. (Fonte: Datasheet Burr-Brown, 1996 – pag. 1)

Onde o ganho é definido através de R_G, pela equação (3.6):

$$G = \frac{V_0}{V_{in}^{+} - V_{in}^{-}} = 5 + \frac{80k}{R_G} \tag{3.6}$$

A Figura 3.13 apresenta o diagrama simplificado do circuito condicionador da célula de carga.

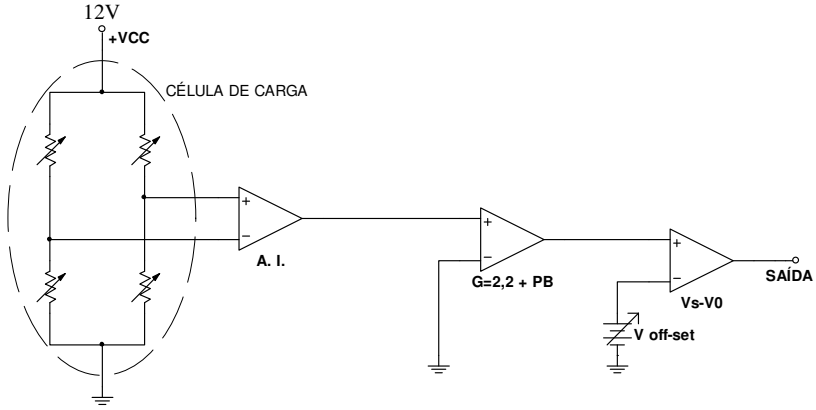


Figura 3.13: Amplificador da célula de carga.

3.1.4. Medição de Velocidade Angular

Um dos dados mais importantes neste trabalho para a obtenção dos parâmetros elétricos e mecânicos de uma máquina CC é a sua velocidade angular.

O recurso mais utilizado em sistemas de medição de velocidade angular é o tacômetro, que pode ser analógico, digital, por efeito Hall, magnetorresistivo, entre outros.

Tacômetros digitais geralmente são dispositivos sem contato e que podem utilizar diversos tipos de sensores, como sensores ópticos, indutivos, magnéticos ou *encoders*. Já os tacômetros analógicos são menos precisos que os digitais e podem ser CC ou AC. No tacômetro CC, a saída é aproximadamente proporcional à sua velocidade de rotação e a sensibilidade é expressa em V/rpm. A polaridade da tensão de saída reflete o sentido de rotação. [9]

A opção adotada foi o uso de um taco-gerador CC, que apresenta uma saída linear proporcional à velocidade angular. Como o transdutor utilizado, produzido pela Pacific Cientific, gera 2,5V a cada 1.000 rpm e motores CC de pequeno porte passam facilmente dos 4.000 rpm, isso resulta em uma tensão incompatível com as entradas de conversão analógico-digital do microcontrolador. Para solucionar isso, foi inserido um divisor resistivo, prevendo velocidades de até 10.000 rpm.

3.1.5. Condicionador de sinais para medição de corrente

Embora a idéia inicial fosse a de usar um sensor de efeito Hall para a medição de corrente, optou-se por utilizar um resistor Shunt em série com o motor. O primeiro sensor apresentou alguns problemas de instabilidade e dificuldade de uma leitura precisa, devido principalmente às características de histerese do material ferromagnético do núcleo. Já a segunda alternativa, além de atender perfeitamente às necessidades, é mais robusto, estável e o seu circuito de condicionamento é de fácil construção.

A fim de se obter uma resistência suficientemente baixa e que suportasse potências um pouco mais elevadas, foram utilizados quatro resistores de 0,1 Ω – 10W – em paralelo, obtendo assim uma resistência de 0,025 Ω. A queda de tensão sobre esses resistores, que fica na ordem de mV, é então amplificada por um INA200, que é um amplificador diferencial com ganho de 20 V/V. A saída deste vai a uma entrada de conversão analógico-digital do microcontrolador.

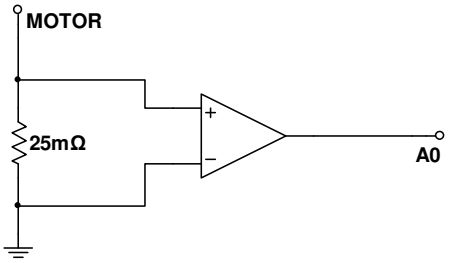


Figura 3.14: Topologia básica do circuito para medição de corrente

Considerando-se uma tensão de saída máxima igual à tensão de alimentação, 5 V, a máxima tensão diferencial na entrada do amplificador corresponde a 5/20 = 0,25V. Uma vez que, com um dos motores utilizados para os ensaios durante o desenvolvimento do protejo, já foram constatadas correntes de partida próximas de 25 A, arbitrou-se o valor de aproximadamente 40 A como sendo o máximo. Desta forma, existe uma certa flexibilidade também para outros motores, evitando o risco de saturação.

Pela Lei de Ohm:

$$V = R \times I = 0{,}025 \times 40 = 1V \tag{3.7}$$

Como esta tensão ficou acima do valor máximo para entrada do amplificador, foi inserido um divisor de tensão em paralelo com o shunt, conforme ilustra a Figura 3.15.

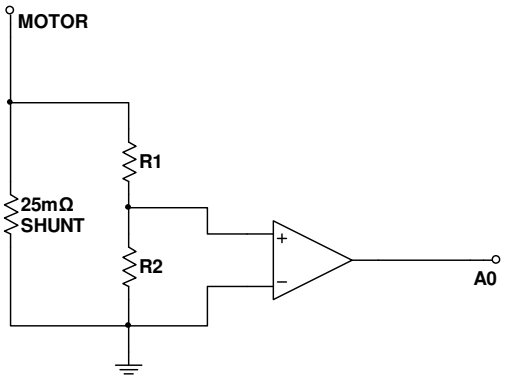


Figura 3.15: Circuito para medição de corrente com divisor de tensão

Considerando que R1+R2 seja suficientemente maior que R_{Shunt}, de forma que a diferença inserida pelo paralelo entre eles possa ser desprezada, $V_{Shunt_{máx}} = 1V$ e $V_{R2_{máx}} = 0{,}25V$. Aplicando-se as leis de Kirrchoff no circuito da figura acima, chegou-se a conclusão que R1 = 3*R2. Os resistores escolhidos foram então: R1 = 4,7 Ω e R2 = 1,5 Ω.

- **INA 200**

É um amplificador destinado à medição de corrente através de resistor *shunt*, tendo a sua saída em tensão proporcional à queda de tensão do *shunt*. Este amplificador apresenta um ganho fixo igual a 20 V/V e frequência de operação de até 500 kHz. Além disso, também possui um circuito comparador internamente. A Figura 3.16 mostra seu circuito interno simplificado e um exemplo típico de ligação para monitoração de corrente pode ser visto na figura 3.17. Observa-se por esse diagrama que quanto maior o valor de R_{Filter} em relação a R_{Shunt}, mais confiável será a resposta.

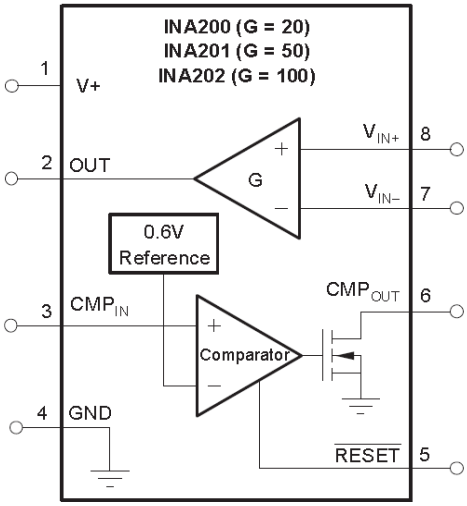


Figura 3.16: Circuito simplificado do INA 200. (Fonte: Datasheet Burr-Brown, 2007 – pag. 1)

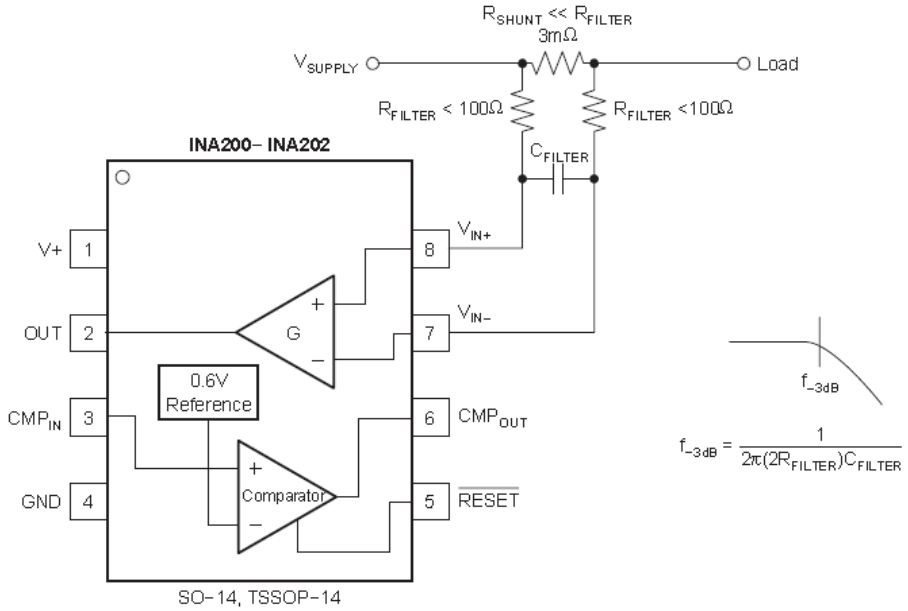


Figura 3.17: Exemplo de ligação do amplificador para monitorar corrente através de um resistor shunt. (Fonte: Datasheet Burr-Brown, 2007 – pag. 12)

3.1.6. Conversor UART – USB

Com o advento da informática, a comunicação USB se popularizou e outras alternativas, como a serial (RS-232) e a paralela (IEE 1284) foram caindo em desuso, a ponto de alguns microcomputadores possuírem apenas a USB. Pensando nisso, desenvolveu-se um hardware tal que a sua comunicação fosse compatível com esse protocolo. Para isso, foi utilizado um conversor UART – USB (*Universal Asynchronous Receiver/Transmitter – Universal Serial Bus*), que permite a comunicação com o microcontrolador através da serial UART e, com o uso de driver específico, emula uma porta serial no PC.

O componente escolhido foi o FT232R, produzido pela FTDI.

O driver que possibilita o reconhecimento desse componente como uma porta serial no PC, do ponto de vista do usuário, pode ser obtido no site do fabricante: http://www.ftdichip.com/FTDrivers.htm.

3.1.7. Conjunto de cargas resistivas

Como já foi dito, a carga mecânica a ser aplicada no eixo do motor será proveniente da força contraeletromotriz desenvolvida pelo gerador, quando submetido a uma carga elétrica. Essa carga elétrica é composta por resistores de potência que são ligados em paralelo. O comando de ligação/desligamento deles é fornecido pelo microcontrolador e o acionamento é feito através de relés. Por questão de segurança, uma vez que esse conjunto é submetido a elevadas potências e tensões (bem acima dos 3,3 V do conversor A/D), foram utilizados acopladores ópticos 4N25 para isolar as partes de comando e de potência. A Figura 3.18 apresenta o circuito de acionamento para um elemento resistivo.

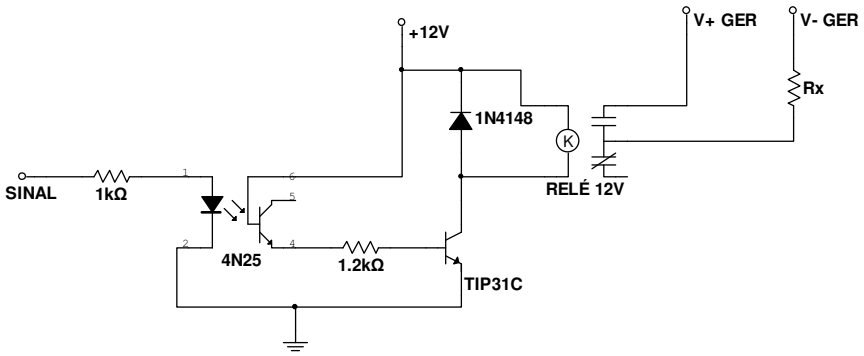


Figura 3.18: Circuito de acionamento de uma carga resistiva.

3.1.8. Parte mecânica/eletromecânica

Essa parte compreende o motor objeto dos testes, o gerador, o tacogerador, a célula de carga e a estrutura mecânica que dá sustentação a tudo isso. A figura a seguir mostra a estrutura utilizada.

O motor fica fixo na base e a sua potência mecânica é transmitida ao gerador através de um acoplamento mecânico, fixado na extremidade dos eixos do motor e do gerador. O gerador, por sua vez, é sustentado por dois rolamentos, ambos localizados na linha central do eixo, de forma a facilitar a tendência de giro do mesmo. Na periferia do gerador, a célula de carga mede a força exercida por essa tendência de giro, ao mesmo tempo em que evita que o gerador se desloque.

Conhecendo-se a força desenvolvida a uma distância conhecida dos eixos, é possível obter o torque no gerador, que é o mesmo desenvolvido pelo motor (desconsiderando-se perdas por atrito dos rolamentos).

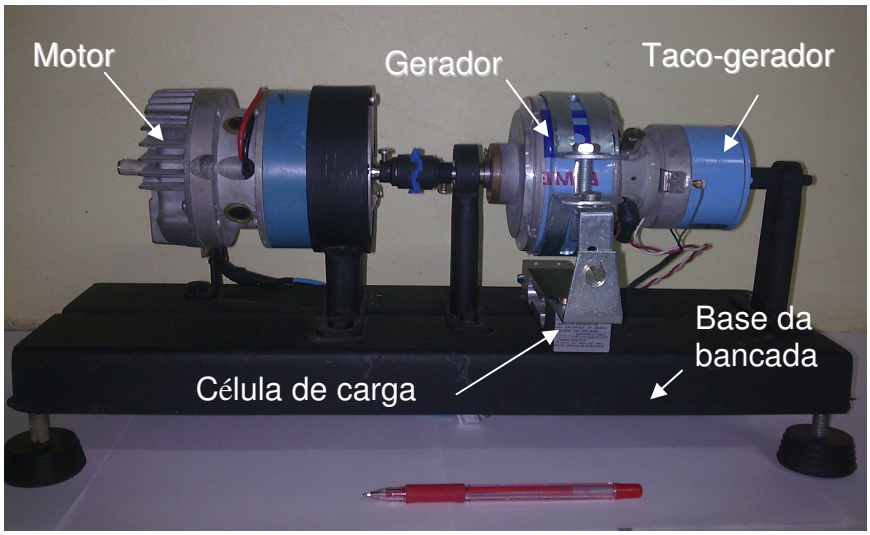


Figura 3.19: Estrutura da bancada de ensaios

Já o gerador, é na verdade um motor de corrente contínua que está sendo utilizado para a geração de corrente elétrica. O modelo deste motor é 33VM62-020-15 – Pacific Cientific.

O motor utilizado durante a construção do projeto foi um similar ao utilizado como gerador, um 33VM82-000-5 – Pacific Cientific. Este motor opera com uma tensão de até 24 V e corrente de 6,6 A.

3.2. O Software

O software implementado no microcomputador tem duas funções principais: ser a interface entre o usuário e a bancada propriamente dita e efetuar os cálculos necessários para a determinação dos parâmetros desejados. Ele foi desenvolvido com o método de programação orientado a objetos com a linguagem C++, que permite facilmente a construção de interfaces gráficas, entre outros recursos.

O compilador utilizado foi o C++ Builder, componente do pacote Borland Developer Studio 2006. Esta linguagem e compilador foram escolhidos por serem de fácil utilização, já que durante o curso tem-se algumas disciplinas onde esse conteúdo é apresentado, possui uma interface amigável, facilidade de acesso às portas de comunicação do PC e de construção de gráficos.

Na tela inicial do software desenvolvido é possível alterar as configurações de comunicação, como velocidade e bits de parada, selecionar a porta a ser utilizada (até COM9), determinar a tensão nominal do motor, iniciar e salvar o ensaio. A Figura 3.20 mostra o diagrama em blocos simplificado do software.

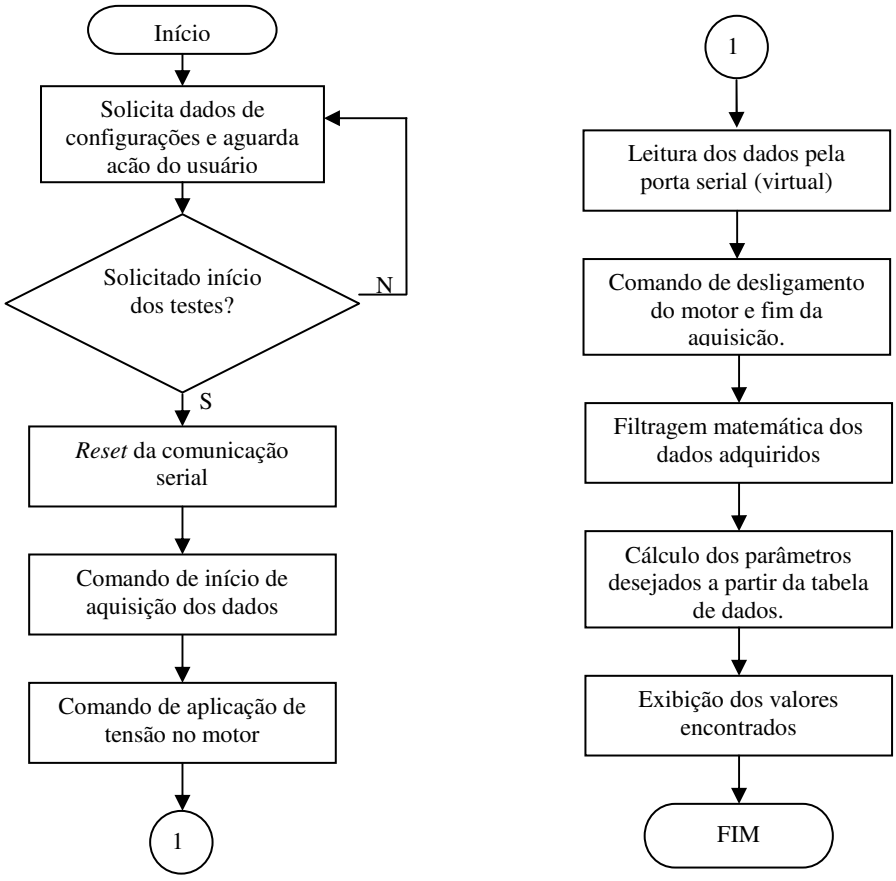


Figura 3.20: Diagrama em blocos do software

Foi elaborado um filtro passa-baixa a fim de se eliminar ruídos de alta frequência. Ele consiste na determinação de uma equação do segundo grau através de mínimos quadrados para um pequeno conjunto de vinte pontos, onde, a partir da equação obtida, calcula-se o valor da curva em um único instante de tempo. Esse procedimento se repete desde o início do conjunto de dados coletados até o seu término, avançando um ponto a cada nova iteração. A Figura 3.21 mostra a sequência utilizada para filtragem dos dados.

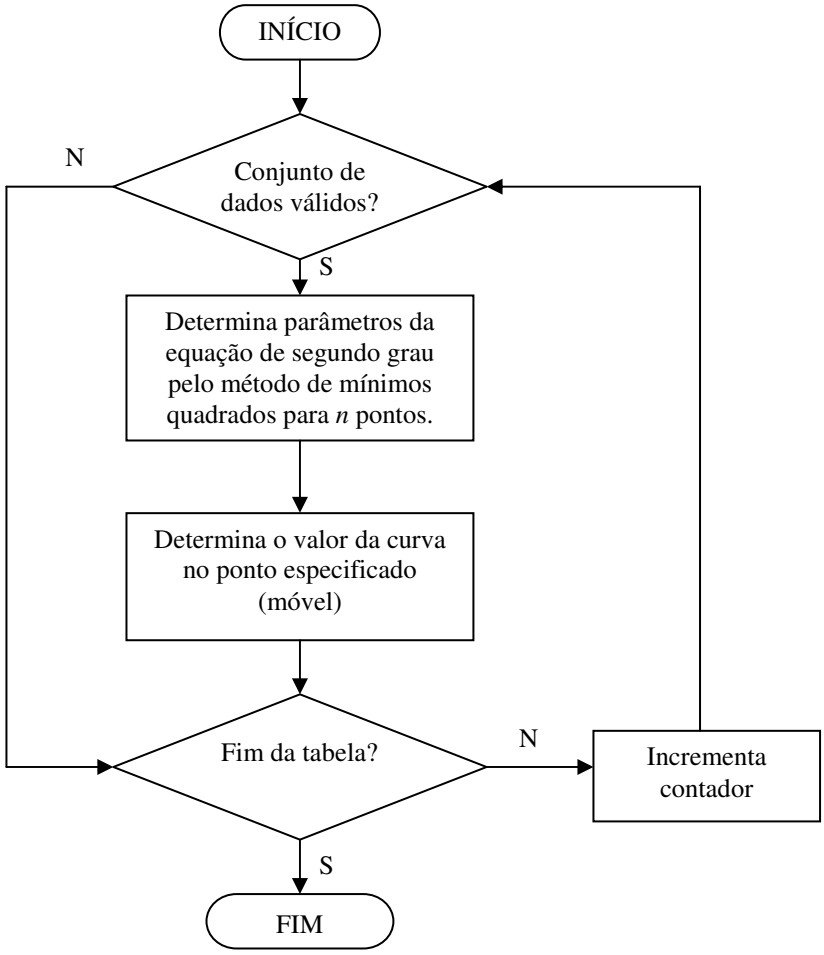


Figura 3.21: Fluxograma do algoritmo utilizado para filtragem

Tendo-se posse dos dados já filtrados, é possível utilizá-los no conjunto de equações (2.28) para determinar os parâmetros do motor. Por se tratar de um método iterativo na qual alguns valores são atualizados constantemente, algumas matrizes também necessitam ser inicializadas. A sequência de cálculos empregada é apresentada na Figura 3.22.

Esse método, como já pôde ser visto, consiste praticamente de repetidas sequências de operações matemáticas com matrizes. A fim de facilitar a

programação desses cálculos, foi obtido pela internet uma biblioteca de funções específicas para este fim, Matrix TCL Lite 1.13, desenvolvido pela Techsoft Pvt. Esta biblioteca é disponibilizada gratuitamente em www.techsoftpl.com, exclusivamente pra fins didáticos e não comerciais. As operações básicas são efetuas da mesma forma que números escalares, outras operações específicas recebem caracteres especiais.

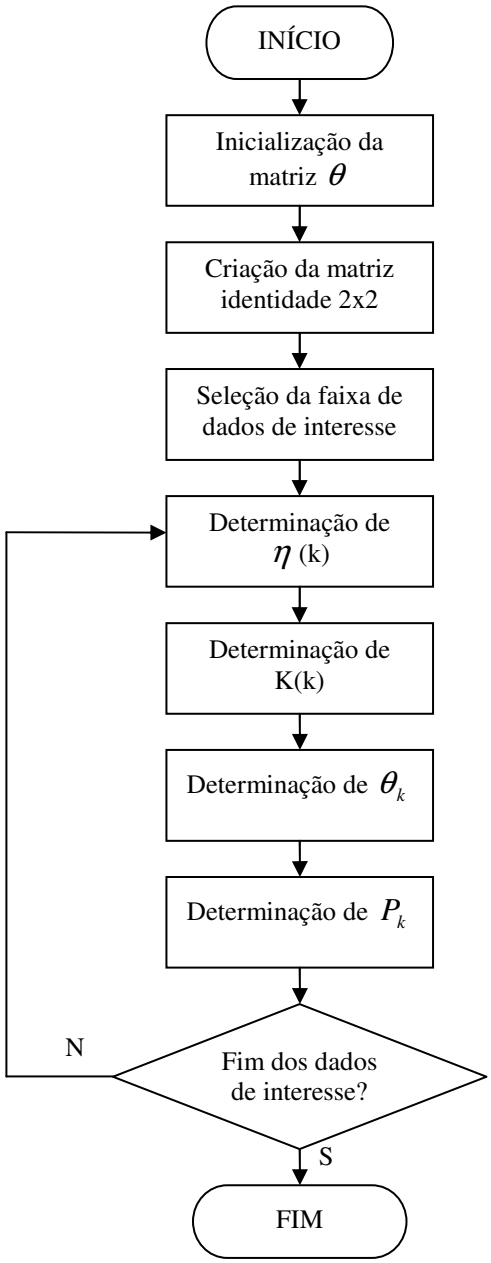


Figura 3.22: Sequência de cálculos para determinação dos parâmetros.

Abaixo são relacionadas as principais operações matriciais utilizadas:

- + →Soma;
- - →Subtração;

- * →Multiplicação;
- / →Divisão (A/B = matriz A multiplicada pela matriz inversa de B);
- ~ →Transposta.

A fim de verificar a funcionalidade de tais operações, foram realizados alguns ensaios com valores conhecidos utilizando a biblioteca obtida e os mesmos foram realizados também no MatLab. Os resultados obtidos foram idênticos.

A parte do software que executa os cálculos descritos na figura 3.22 é transcrita a seguir:

```
K_k0 = (P_k1 * phi_k0)/((mq_1 + ~phi_k0 * P_k1 * phi_k0));
er = Y_k0 - (~phi_k0 * teta_k0);
teta_k0 = teta_k0 + (K_k0 * er);
P_k1 = P_k1 - (K_k0 * ~phi_k0 * P_k1);
```

Com este procedimento se obtém, após diversas iterações, os valores que compõem a matriz θ das equações de estado que descrevem o comportamento dinâmico do motor (equação 3.8).

$$\begin{bmatrix} \omega_k \\ i_k \end{bmatrix} = \begin{bmatrix} -i_k & i_{k-1} & 0 & 0 & v_k \\ 0 & 0 & \omega_k & -\omega_{k-1} & 0 \end{bmatrix} \cdot \begin{bmatrix} \left(R + \frac{L}{T}\right) \cdot \frac{1}{K_b} \\ \left(\frac{L}{TK_b}\right) \\ \left(\frac{J}{TK_b} + \frac{b}{K_b}\right) \\ \left(\frac{J}{TK_b}\right) \\ \frac{1}{K_b} \end{bmatrix} \tag{3.8}$$

Ou, de forma simplificada, $Y = \varphi \cdot \theta$.

O procedimento adotado nessa sequência de cálculos é o chamado método recursivo dos mínimos quadrados, onde, a cada nova iteração, os valores que compõem a matriz θ são atualizados, reduzindo o erro.

Ao final das iterações, podem-se isolar os termos R, L, K_b, b e J:

$$R = \frac{\theta_{1,1} - \theta_{2,1}}{\theta_{5,1}} \tag{3.9}$$

$$L = \frac{\theta_{2,1} \times T}{\theta_{5,1}} \tag{3.10}$$

$$K_b = \frac{1}{\theta_{5,1}} \tag{3.11}$$

$$b = \frac{\theta_{3,1} - \theta_{4,1}}{\theta_{5,1}} - b_{gerador} \tag{3.12}$$

$$J = \frac{\theta_{4,1} \times T}{\theta_{5,1}} - J_{gerador} \tag{3.13}$$

Onde:

R → Resistência elétrica do motor;

L → Indutância do enrolamento;

K_b = Constante de força contra-eletromotriz;

b = Coeficiente de atrito viscoso;

J = Momento de inércia do rotor;

T = Tempo diferencial (intervalo entre duas amostras).

Nas duas últimas equações já foi extraído o erro inserido pelo rotor do gerador, que contribui em massa (inércia) e perdas por atrito, tanto do gerador em si quanto dos rolamentos.

Durante os ensaios iniciais foram verificadas algumas falhas no processo de comunicação entre o microcontrolador e o microcomputador, como perda de sincronismo após a falha no recebimento de um byte ou valores errôneos. A fim de minimizar esses efeitos, estabeleceu-se como padrão o uso dos caracteres “AA” no início de cada pacote de dados para evitar a perda de sincronismo. Durante a leitura da porta serial, lêem-se os dez bytes seguintes após a confirmação de recebimento dos caracteres “AA”. Já os valores errôneos recebidos geralmente eram 255 no byte mais significativo, onde o valor máximo, proveniente da conversão analógico-digital, seria 3. Uma forma simples de resolver isto foi descartando valores recebidos acima de 3 para o byte mais significativo.

Após o uso desses recursos de validação, não foram mais verificados problemas que pudessem prejudicar os procedimentos de análise. Uma vez que as falhas ocorriam com pequena frequência (inferior a 0,5% do total de dados

recebidos), o descarte de alguns pacotes de dados não gera consequências significativas.

3.3. O Firmware

O firmware desenvolvido para o microcontrolador foi escrito e compilado através do software IAR (IAR C/C++ Compiler for MSP430 - V3.41A/W32 [Kickstart]). Esse software, além de ser um compilador, também permite que sejam feitas simulações e a própria gravação do firmware no microcontrolador. Outro fator que contribuiu para a sua utilização foi o fato de ele ser fornecido junto com o kit de programação EZ430-F2013, cujo gravador foi utilizado.

3.3.1. Programação do Microcontrolador

Embora a ferramenta de desenvolvimento eZ430 – F2013 seja produzida para programação do MSP430F2013, ela utiliza o mesmo protocolo de programação que o MSP430F2274. Essa ferramenta dispõe de terminais de fácil acesso permitindo a sua ligação com outros microcontroladores diferentes do original e a um custo mais acessível do que as outras ferramentas semelhantes compatíveis.

Esse módulo também apresenta a grande vantagem de possuir comunicação USB.

3.3.2. Software de Programação e Compilação

Os dois principais compiladores para este fim são o *IAR Embedded Workbench* e o *Code Composer Essentials* (CCE). Optou-se por utilizar o IAR que já acompanha o kit. Este software permite criar ambientes de trabalho onde são inseridos o código principal, bibliotecas e outros componentes. Também oferece como recurso a possibilidade de emulação e de gravação direta no microcontrolador, sem a necessidade de outro software para isso.

No microcontrolador é gerada uma interrupção ao receber um caractere pelo canal de comunicação UART e, conforme o caractere poderá:

- Acionar o motor, com níveis de tensão que variam de 1,5 V até 23,5V, variando com um passo de 1,5V (até 21V) ou realizar um incremento ou decremento nessa mesma tensão, com um passo menor;

- Realizar um incremento ou decremento da resistência de carga;
- Efetuar uma única conversão analógico-digital e enviar através do canal de comunicação serial UART;
- Iniciar uma conversão analógico-digital contínua e enviar simultaneamente os dados pelo canal de comunicação serial UART.

Para que fosse possível atingir elevadas taxas de comunicação, foi utilizada como velocidade da UART a máxima taxa possível pelo Hyper Terminal (software padrão de comunicação do Windows), 921600 bits por segundo. No firmware é necessário configurar o fator de divisão N, que é definido por:

$$N = \frac{BRCLK}{baudrate} \tag{3.14}$$

Onde BRCLK corresponde à frequência do clock utilizado pela interface UART, e foi definido como sendo igual à frequência DCO, 16 MHz. Dessa forma:

$$N = \frac{16000000}{921600} = 17{,}36 \Rightarrow 17 \tag{3.15}$$

O microcontrolador possui oscilador interno programável, com frequência máxima de 16 MHz. Para operar nessa condição, os registradores "*Basic Clock System Control 1*" e "*Digitally Controlled Oscilator*" foram devidamente configurados.

Além disso, foram criadas algumas funções específicas, principalmente para facilitar a conversão analógico-digital, digital-analógica e a comunicação. Entre elas:

- void ADC_ini (void): Configura parâmetros básicos para operação do conversor analógico-digital, como o *clock* (16 MHz), modo de aquisição (sequencial), modo de transferência dos dados (contínua) e canais a serem habilitados para conversão (A0 à A4).
- void Get_ADC (unsigned int a): informa o endereço inicial onde serão armazenados os dados resultantes da conversão analógico-digital e habilita o início da conversão.
- void Delay (unsigned int a): rotina de perda de tempo, utilizada para temporização. O tempo corresponde a 1,25 + a * 0,75 µs.
- void DAC_send_bit (int val_bit): função responsável por enviar um único bit para o conversor digital-analógico de forma serial. Também gera o pulso de *clock* necessário para tal.

- void DAC_send_data (long int val_dac): esta função recebe um valor inteiro e converte ele em binários. Posteriormente, cada bit é enviado de forma sequencial, utilizando a função DAC_send_bit.

O gráfico da figura 3.23 mostra a sequência de comandos necessária para o correto funcionamento do conversor digital-analógico. As funções DAC_send_bit e DAC_send_data baseiam-se nessa sequência e temporização.

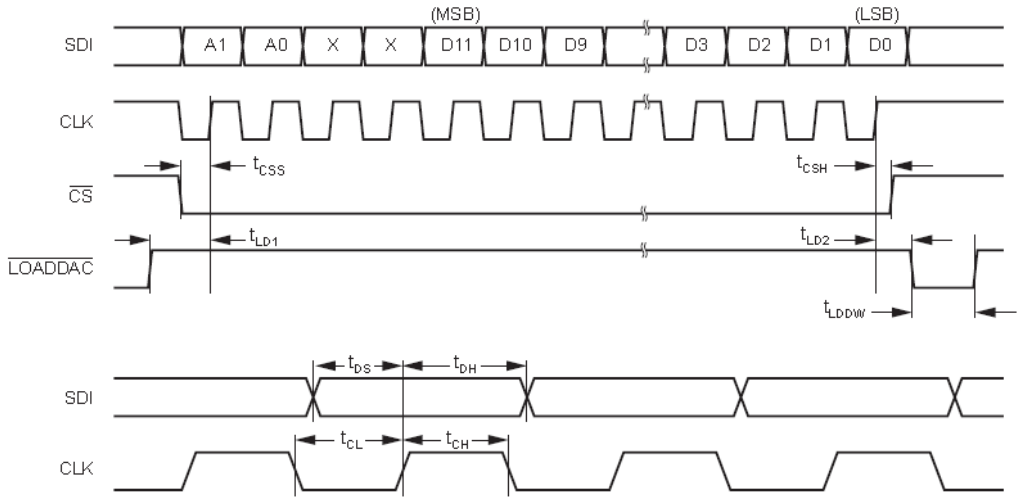


Figura 3.23: Sequência de comando do conversor digital-analógico. (Fonte: Datasheet Burr-Brown, 1998 – pag. 10)

O MSP430-F2274 possibilita a sua programação através da interface spy-by wire. Uma ferramenta comercializada pela Texas Instruments para este fim é o kit eZ430-F2013. Este kit inclui o programador em si (semelhante a um pen-drive e com conexão USB), um microcontrolador MSP430-F2013 soldado em soquete específico para o kit e um CD com o compilador IAR Embedded Workbench Kickstart.

O programador, embora seja compatível com outros microcontroladores com interface spy-by wire, a sua conexão é preparada apenas para o MSP430-F2013. Dessa forma foi necessário fazer a inserção de um novo conector, adaptado através de cabos soldados diretamente no circuito do programador, como mostra a Figura 3.24.

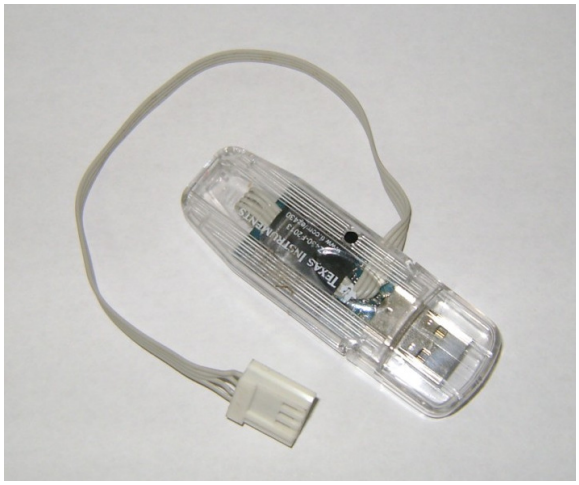


Figura 3.24: Programador do MSP430

3.4. *Validação do modelo*

A fim de validar o modelo matemático e o método de ensaio adotados, foi utilizado o Simulink, uma ferramentas do software MatLab 7.5. Essa ferramenta permite a construção de modelos de sistemas de controle utilizando diagramas em blocos e a utilização de modelos prontos de sistemas reais. Alguns deles podem ser obtidos gratuitamente no site do desenvolvedor do software: www.mathworks.com.

Aqui, especificamente, foi utilizado o *DC Motor Model*, que permite que sejam definidas a tensão de alimentação, resistência, indutância, constante de força contra-eletromotriz, momento de inércia e coeficiente de atrito viscoso. Esse modelo gera o gráfico com as curvas de corrente, velocidade angular e tensão aplicada. As Figuras 3.25 e 3.26 ilustram o seu funcionamento:

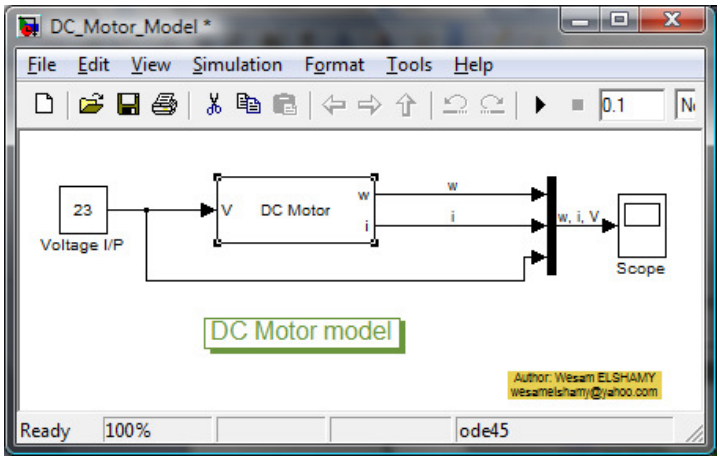


Figura 3.25: Modelo motor CC no Simulink.

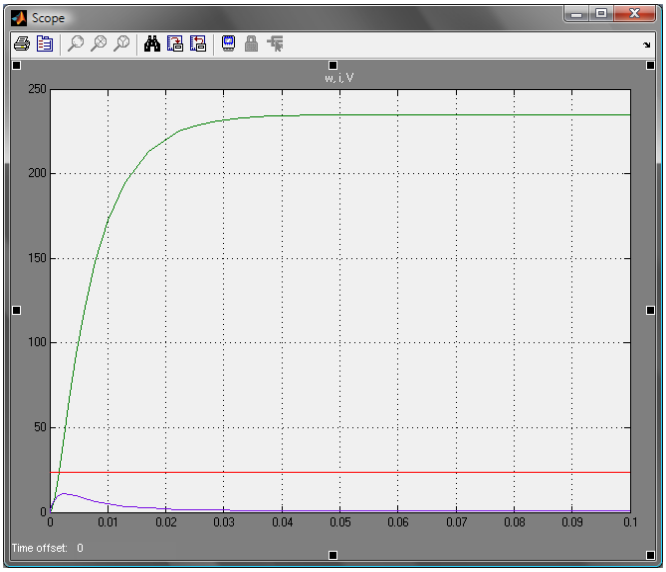


Figura 3.26: Gráfico gerado pelo modelo da máquina CC no Simulink.

Darizon [10] e Márcio [11] descrevem o motor de corrente contínua a partir do diagrama em blocos. A Figura 3.27 mostra o sistema implementado a partir destes diagramas. Uma vez que, quando testados os dois métodos, o modelo anterior e o diagrama em blocos, ambos apresentaram resultados iguais para as mesmas condições, confirmou-se a validade do modelo "*DC Motor Model*". A partir de então apenas este último foi utilizado por questões de facilidade.

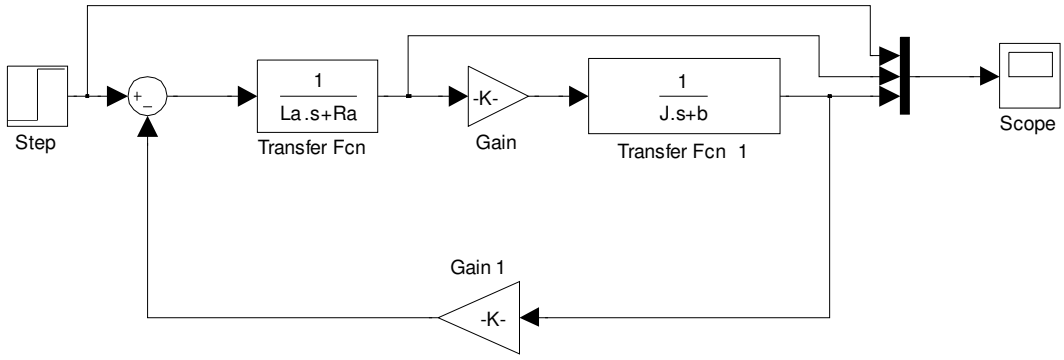


Figura 3.27: Diagrama em blocos que descrevem o funcionamento de um motor CC com excitação independente.

Com o uso de simulação através de modelos matemáticos é possível aplicar os parâmetros encontrados (calculados) nestes modelos e comparar o gráfico gerado pela simulação e o gráfico gerado pelo ensaio real do motor.

4. APRESENTAÇÃO E DISCUSSÃO DOS RESULTADOS

Nos tópicos a seguir serão apresentados os resultados obtidos durante os ensaios, tanto para os parâmetros da máquina CC quanto para o rendimento, avaliando-os do ponto de vista estatístico e fazendo comparações gráficas.

4.1. Procedimento

O motor, ao ser ensaiado, encontra-se acoplado mecanicamente ao gerador, de forma que este influencia significativamente nos resultados obtidos para os parâmetros mecânicos, não podendo ser desprezado. Assim, o primeiro motor a ser ensaiado é o próprio gerador a fim de se obter os valores de coeficiente de atrito viscoso b e de momento de inércia J.

A Tabela 4.1 apresenta as médias resultantes de alguns ensaios realizados na bancada, quando aplicado um sinal de excitação em tensão de 24V.

Tabela 4.1: Valores médios obtidos para o gerador.

	$\overline{X}$
K	$3{,}98.10^{-2}$
b	$1{,}22.10^{-4}$
R	$7{,}33.10^{-1}$
L	$4{,}74.10^{-4}$
J	$1{,}16.10^{-5}$

O software inicialmente estima os parâmetros do conjunto motor + gerador e posteriormente subtrai os valores de b e de J, obtidos neste ensaio.

O motor, ao ser fixado na bancada para testes, precisa ser alinhado a fim de permitir uma boa transferência de torque, minimizar as vibrações e efeitos de atrito. Por questões de dificuldade técnica, a estrutura da bancada possui um sistema de alinhamento bem singelo e um pouco trabalhoso. Após alinhamento visual do motor com o gerador, o mesmo foi acionado em rotações diferentes e buscou-se, através do ajuste das porcas borboleta, minimizar a vibração e ruído.

4.2. Resultados obtidos

Foram ensaiados três motores distintos, todos com campo de ímã permanente, sendo dois deles com tensão nominal de 24V e um de 52V. Todos eles foram fixados uma única vez e realizados trinta ensaios consecutivos em 23,5V, 21V e 18V. As tabelas a seguir apresentam as médias e desvio-padrão dos parâmetros estimados, considerando-se os dez primeiros ensaios válidos, ou seja, com valores positivos. Parâmetros obtidos com valores negativos foram considerados inválidos, pois não são possíveis em um sistema físico real como esse.

4.2.1. Motor SERVO DC – Tensão nominal: 24V

A Tabela 4.2 apresenta os valores médios e o desvio-padrão obtidos para cada parâmetro estimado com tensão de ensaio de 23,5 V. 13,3% dos ensaios realizados resultaram em pelo menos um parâmetro negativo, sendo desconsiderados nos cálculos.

Tabela 4.2: Média e desvio-padrão - SERVO DC - 23,5V

	$\overline{X}$	σ
K [V/rad]	$9{,}22.10^{-2}$	$1{,}12.10^{-3}$
b [N.m.s/rad]	$2{,}95.10^{-4}$	$1{,}14.10^{-4}$
R [Ω]	1,74	$7{,}09.10^{-2}$
L [H]	$1{,}77.10^{-3}$	$4{,}61.10^{-4}$
J [kg.m²]	$2{,}92.10^{-5}$	$2{,}78.10^{-6}$

A Tabela 4.3 refere-se aos ensaios com tensão de 21 V. Nestes, 3,3% dos ensaios realizados resultaram em pelo menos um parâmetro negativo, e foram desconsiderados nos cálculos.

Tabela 4.3: Média e desvio-padrão - SERVO DC - 21V

	$\overline{X}$	σ
K [V/rad]	$9{,}29.10^{-2}$	$1{,}66.10^{-3}$
b [N.m.s/rad]	$3{,}48.10^{-4}$	$1{,}74.10^{-4}$
R [Ω]	1,74	$9{,}27.10^{-2}$
L [H]	$2{,}04.10^{-3}$	$9{,}24.10^{-4}$
J [kg.m²]	$3{,}03.10^{-5}$	$5{,}90.10^{-6}$

Os valores médios e desvio-padrão obtido com tensão de ensaio de 18 V são apresentados na Tabela 4.4. Todos os dez ensaios realizados resultaram em valores de parâmetros positivos.

Tabela 4.4: Média e desvio-padrão - SERVO DC - 18V

	$\overline{X}$	σ
K [V/rad]	$9{,}41.10^{-2}$	$7{,}45.10^{-4}$
b [N.m.s/rad]	$4{,}38.10^{-4}$	$2{,}28.10^{-4}$
R [Ω]	1,76	$1{,}09.10^{-1}$
L [H]	$1{,}62.10^{-3}$	$1{,}08.10^{-3}$
J [kg.m²]	$2{,}92.10^{-5}$	$7{,}16.10^{-6}$

4.2.2. Motor MICRO SWTCH/PACIFIC CIENTIFIC – Tensão nominal: 24V

A Tabela 4.5 apresenta os valores médios e o desvio-padrão obtidos para cada parâmetro estimado com tensão de ensaio de 23,5 V. Todos os dez ensaios realizados resultaram em valores de parâmetros positivos, sendo utilizados nos cálculos.

Tabela 4.5: Média e desvio-padrão - PACIFIC CIENTIFIC - 23,5V

	$\overline{X}$	σ
K [V/rad]	$4{,}25.10^{-2}$	$6{,}90.10^{-4}$
b [N.m.s/rad]	$1{,}98.10^{-4}$	$2{,}60.10^{-5}$
R [Ω]	$8{,}43.10^{-1}$	$9{,}24.10^{-3}$
L [H]	$4{,}30.10^{-4}$	$1{,}66.10^{-4}$
J [kg.m²]	$3{,}90.10^{-6}$	$4{,}75.10^{-7}$

Já a Tabela 4.6 apresenta os resultados obtidos através dos ensaios com tensão de excitação de 21V. Nessas condições, 6,7% dos resultados obtidos apresentaram algum parâmetro negativo e não foram considerados.

Tabela 4.6: Média e desvio-padrão - PACIFIC CIENTIFIC - 21V

	$\overline{X}$	σ
K [V/rad]	$4{,}11.10^{-2}$	$7{,}18.10^{-4}$
b [N.m.s/rad]	$1{,}39.10^{-4}$	$3{,}12.10^{-5}$
R [Ω]	$8{,}33.10^{-1}$	$1{,}88.10^{-2}$
L [H]	$3{,}26.10^{-4}$	$2{,}08.10^{-4}$
J [kg.m²]	$4{,}32.10^{-6}$	$3{,}99.10^{-7}$

Os resultados obtidos através dos ensaios com tensão de 18 V são apresentados na Tabela 4.7. 30% dos ensaios resultaram em algum valor negativo e foram desconsiderados nos cálculos.

Tabela 4.7: Média e desvio-padrão - PACIFIC CIENTIFIC - 18V

	$\overline{X}$	σ
K [V/rad]	$4{,}20.10^{-2}$	$6{,}78.10^{-4}$
b [N.m.s/rad]	$1{,}12.10^{-4}$	$4{,}44.10^{-5}$
R [Ω]	$7{,}64.10^{-1}$	$3{,}73.10^{-2}$
L [H]	$2{,}27.10^{-4}$	$1{,}48.10^{-4}$
J [kg.m²]	$6{,}03.10^{-6}$	$5{,}77.10^{-7}$

4.2.3. Motor ENGEL – Tensão nominal: 52V

O terceiro motor a ser ensaiado na bancada foi o Engel, cuja tensão nominal é de 52V. Devido a limitações da bancada, ele foi ensaiado com tensões igual ou inferiores a 23,5V.

As médias e desvio-padrão obtidos para os ensaios com excitação de 23,5V são apresentados na Tabela 4.8. Nestas condições, 30% dos ensaios resultaram em valores negativos para os parâmetros e dessa forma não foram considerados nos cálculos.

Tabela 4.8: Média e desvio-padrão - ENGEL - 23,5V

	$\overline{X}$	σ
K [V/rad]	$1{,}44.10^{-1}$	$4{,}31.10^{-3}$
b [N.m.s/rad]	$4{,}81.10^{-4}$	$2{,}38.10^{-4}$
R [Ω]	3,44	$0{,}30.10^{-1}$
L [H]	$3{,}02.10^{-3}$	$1{,}78.10^{-3}$
J [kg.m²]	$3{,}35.10^{-5}$	$1{,}22.10^{-5}$

Quando submetido à tensão de excitação de 21V, a estimativa dos parâmetros não resultou em nenhum valor negativo. As médias e desvio padrão são apresentados na Tabela 4.9.

Tabela 4.9: Média e desvio-padrão - ENGEL - 21V

	$\overline{X}$	σ
K [V/rad]	$1{,}41.10^{-1}$	$2{,}50.10^{-3}$
b [N.m.s/rad]	$6{,}54.10^{-4}$	$4{,}18.10^{-4}$
R [Ω]	3,84	$3{,}57.10^{-1}$
L [H]	$8{,}24.10^{-3}$	$2{,}48.10^{-3}$
J [kg.m²]	$2{,}24.10^{-5}$	$1{,}10.10^{-5}$

Os resultados dos ensaios com tensão de 18V são apresentados na Tabela 4.10. Nestas condições, 10% dos ensaios resultaram em parâmetros negativos e foram desconsiderados nos cálculos de média e desvio-padrão.

Tabela 4.10: Média e desvio-padrão - ENGEL - 18V

	$\overline{X}$	σ
K [V/rad]	$1{,}43.10^{-1}$	$2{,}27.10^{-3}$
b [N.m.s/rad]	$8{,}83.10^{-4}$	$4{,}75.10^{-5}$
R [Ω]	3,80	$4{,}28.10^{-1}$
L [H]	$8{,}02.10^{-3}$	$2{,}78.10^{-3}$
J [kg.m²]	$2{,}41.10^{-5}$	$1{,}26.10^{-5}$

4.3. Simulações baseadas nos resultados

A fim de se validar o modelo matemático empregado na determinação dos parâmetros, os resultados obtidos nos ensaios foram utilizados como parâmetros do modelo de motor DC no MatLab. Deve-se observar que, para que seja possível comparar os gráficos gerados pela simulação com os gráficos obtidos no ensaio, é necessário considerar também o coeficiente de atrito e momento de inércia do gerador e taco-gerador.

As figuras a seguir mostram o comparativo dos gráficos obtidos através da simulação com os parâmetros estimados e do ensaio prático, para os três motores, em tensão de 23,5 V. Para cada motor foi realizada a simulação com os parâmetros obtidos a partir do ensaio com 23,5 V e com 18 V. Além disso, também são expostos os gráficos ampliados a fim de se observar melhor o comportamento da corrente de armadura.

É interessante observar os gráficos das curvas de velocidade angular (verde) e da corrente de armadura (lilás). As principais características a serem obtidas graficamente são o tempo de estabilização e os valores máximos e finais.

A fim de facilitar a visualização, todos os gráficos em uma mesma página estão na mesma escala.

4.3.1. Motor SERVO DC

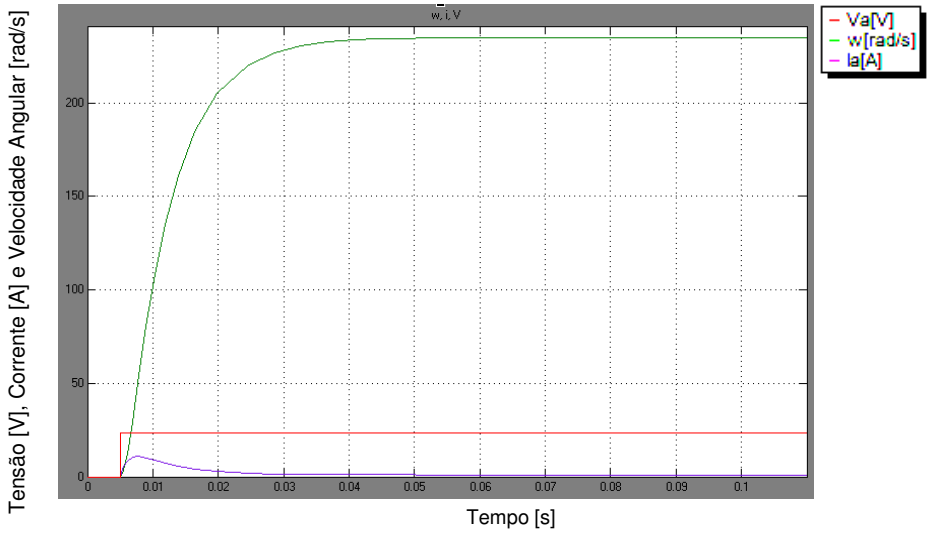


Figura 4.1: Gráficos simulados com parâmetros da tabela 4.2 - SERVO DC

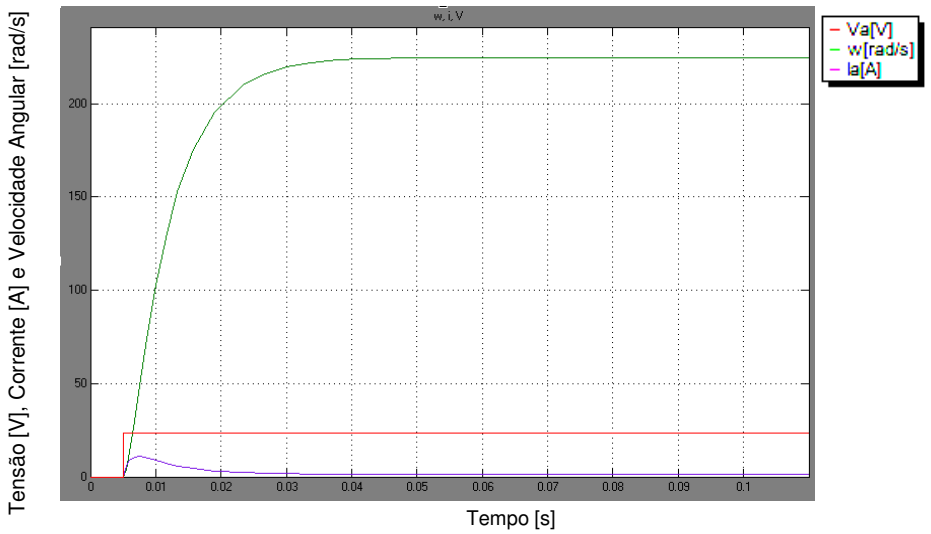


Figura 4.2: Gráficos simulados com parâmetros da tabela 4.4 - SERVO DC

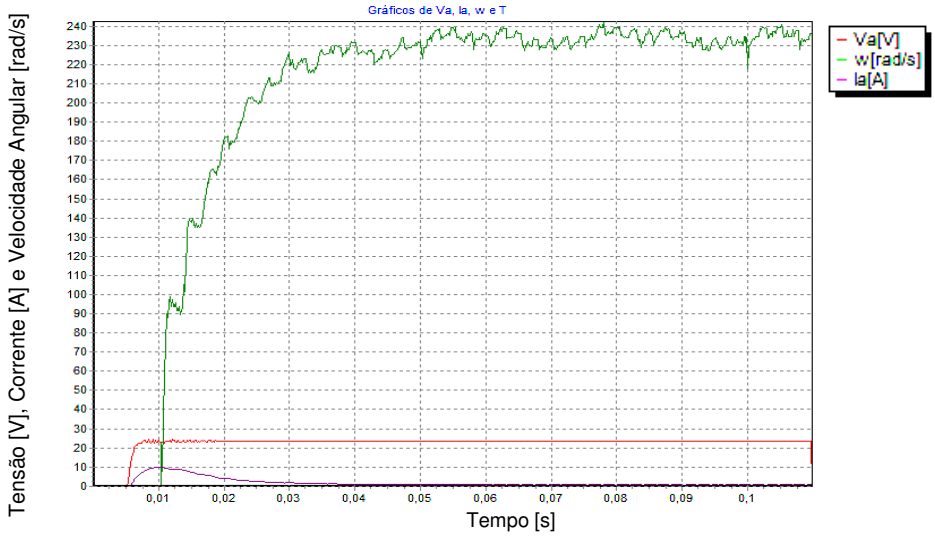


Figura 4.3: Gráficos do ensaio prático

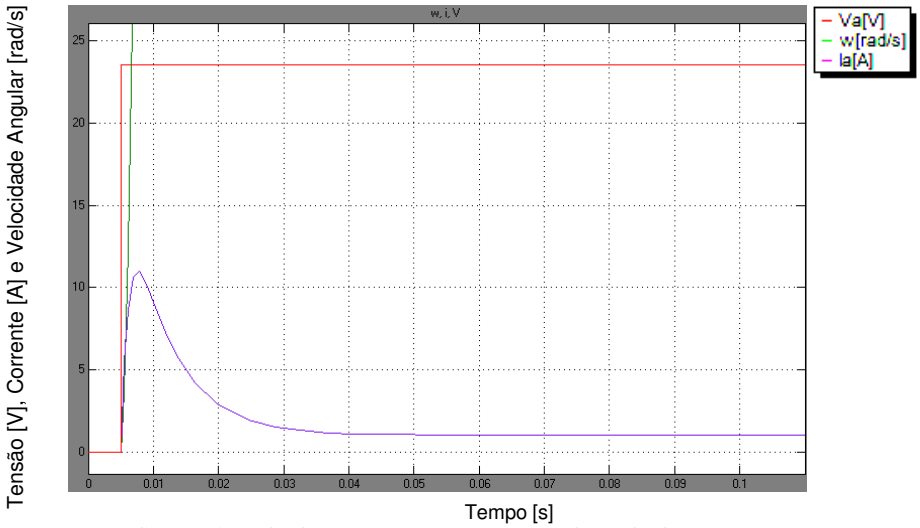


Figura 4.4: Gráficos simulados com parâmetros da tabela 4.2 - SERVO DC (ampliado)

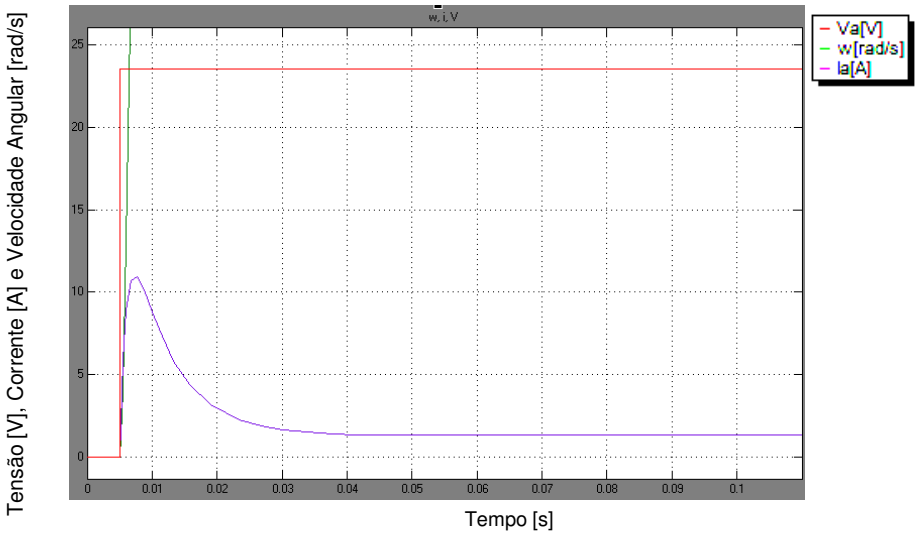


Figura 4.5: Gráficos simulados com parâmetros da tabela 4.4 - SERVO DC (ampliado)

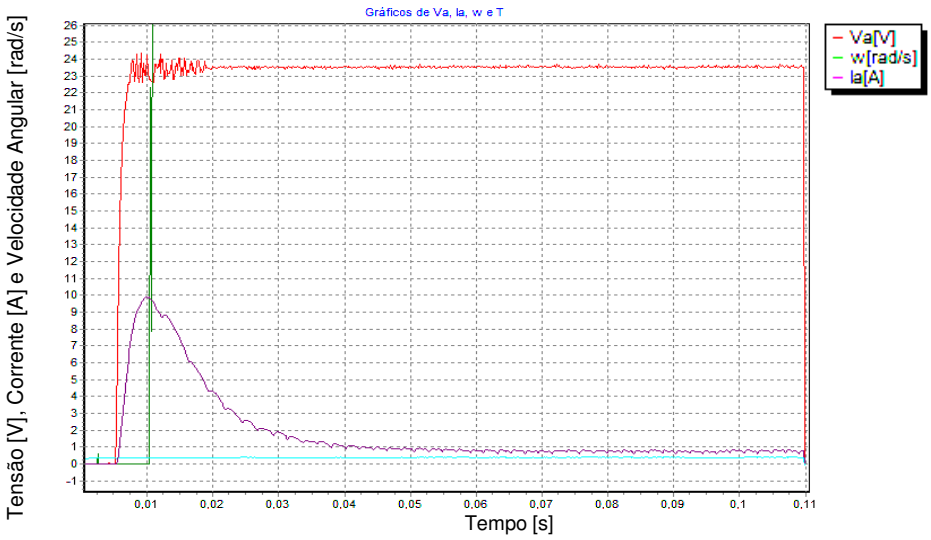


Figura 4.6: Gráficos do ensaio prático (ampliado)

4.3.2. Motor MICRO SWTCH/PACIFIC CIENTIFIC

4.3.3

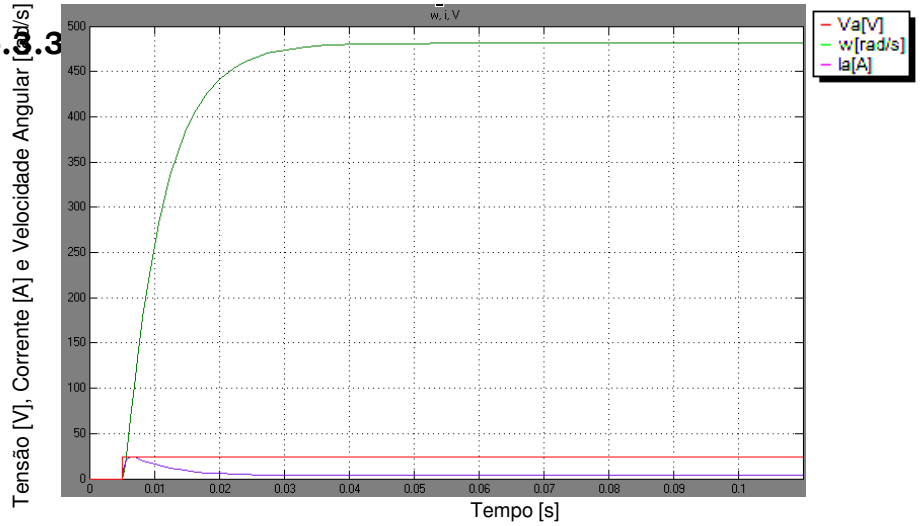


Figura 4.7: Gráficos simulados com parâmetros da tabela 4.5 - PACIFIC CIENTIFIC

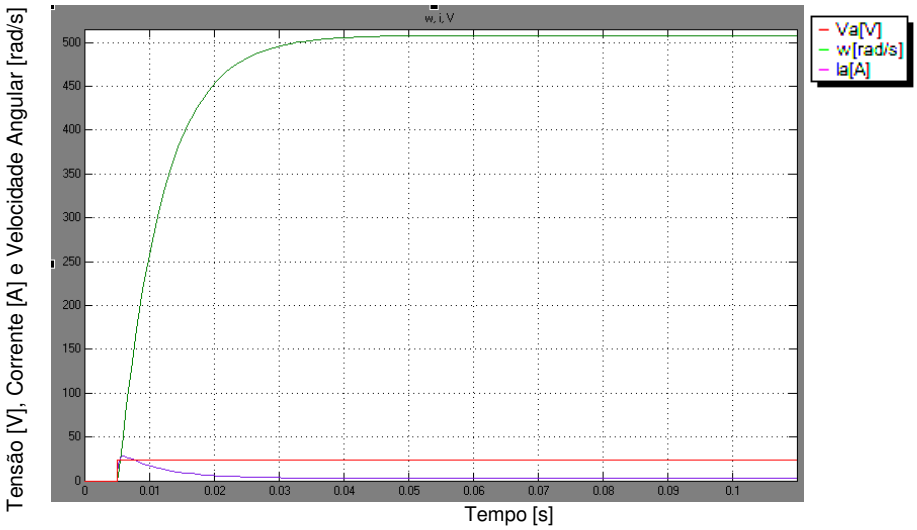


Figura 4.8: Gráficos simulados com parâmetros da tabela 4.7 - PACIFIC CIENTIFIC

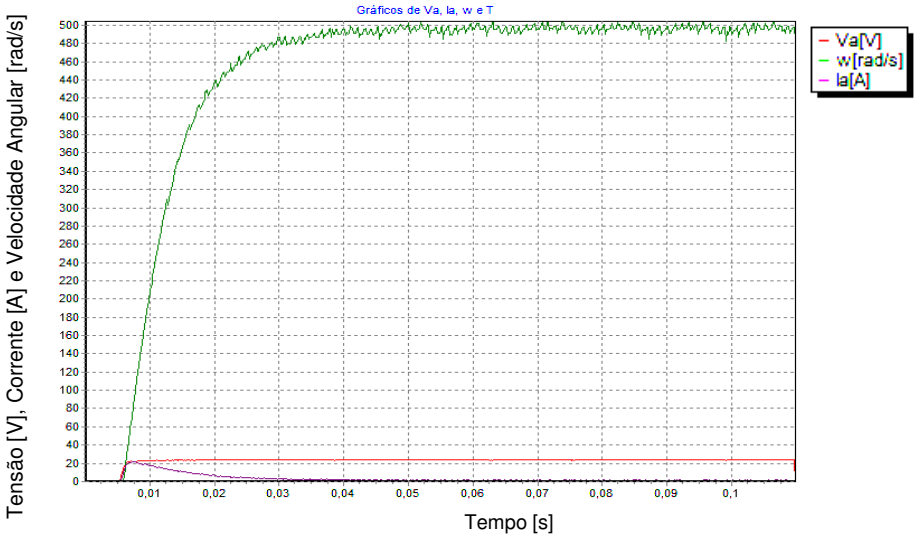


Figura 4.9: Gráficos do ensaio prático - PACIFIC CIENTIFIC.

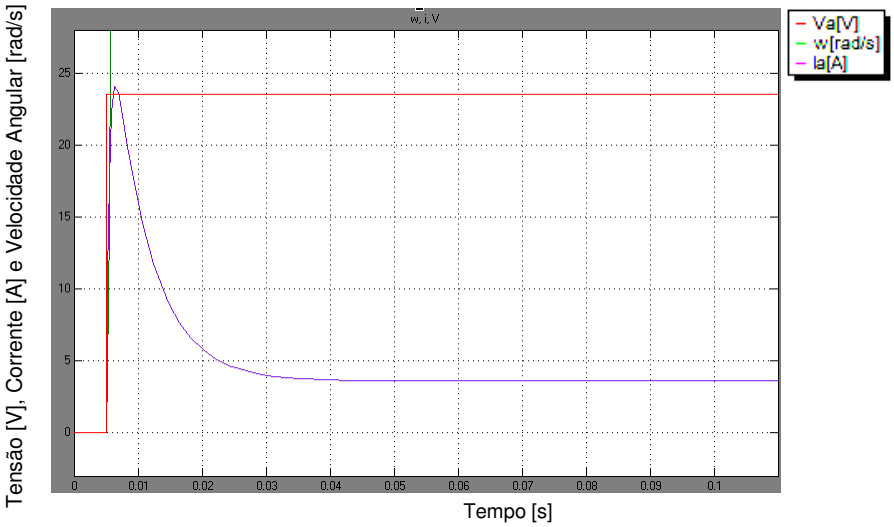


Figura 4.10: Gráficos simulados com parâmetros da tabela 4.5 - PACIFIC CIENTIFIC (ampliado)

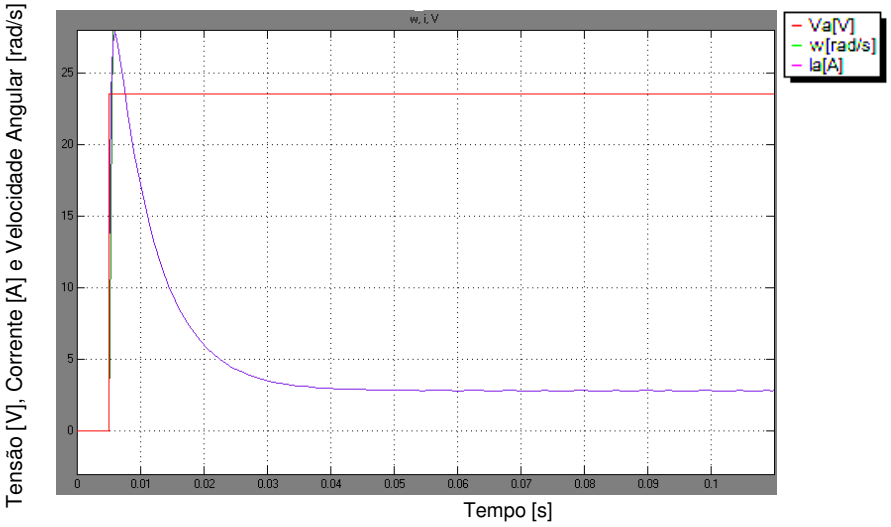


Figura 4.11: Gráficos simulados com parâmetros da tabela 4.7 - PACIFIC CIENTIFIC (ampliado)

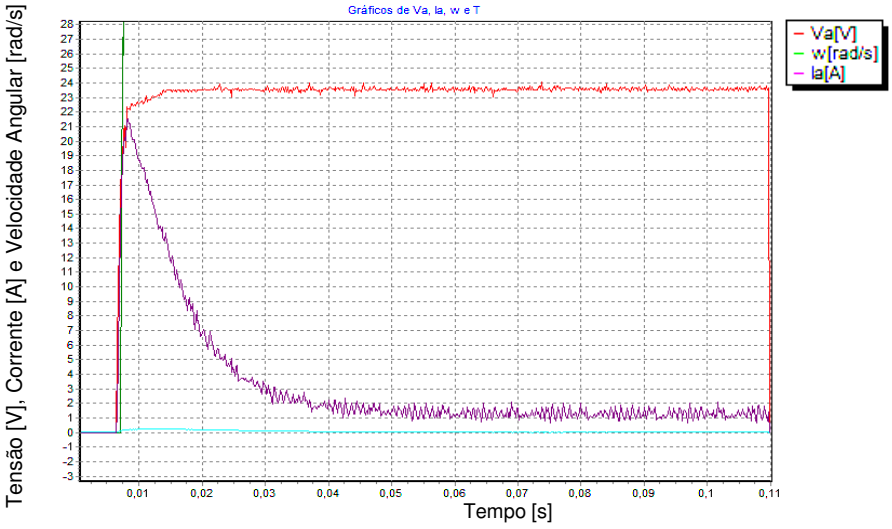


Figura 4.12: Gráficos do ensaio prático - PACIFIC CIENTIFIC (ampliado)

4.3.4. Motor ENGEL

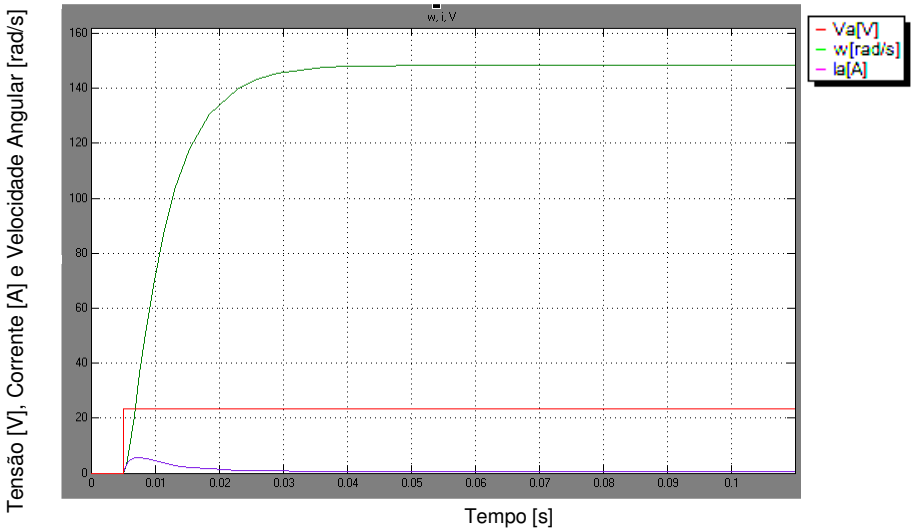


Figura 4.13: Gráficos simulados com parâmetros da tabela 4.8 – ENGEL

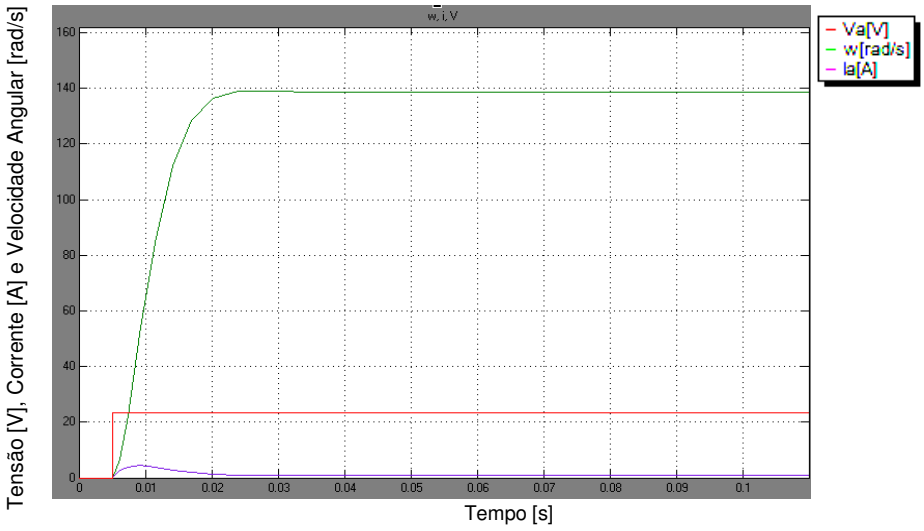


Figura 4.14: Gráficos simulados com parâmetros da tabela 4.10 – ENGEL

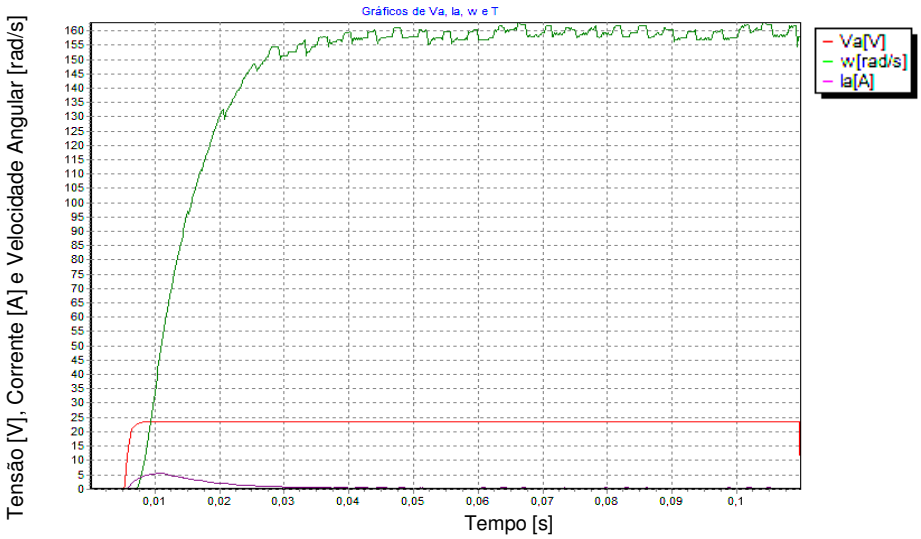


Figura 4.15: Gráficos do ensaio prático - ENGEL

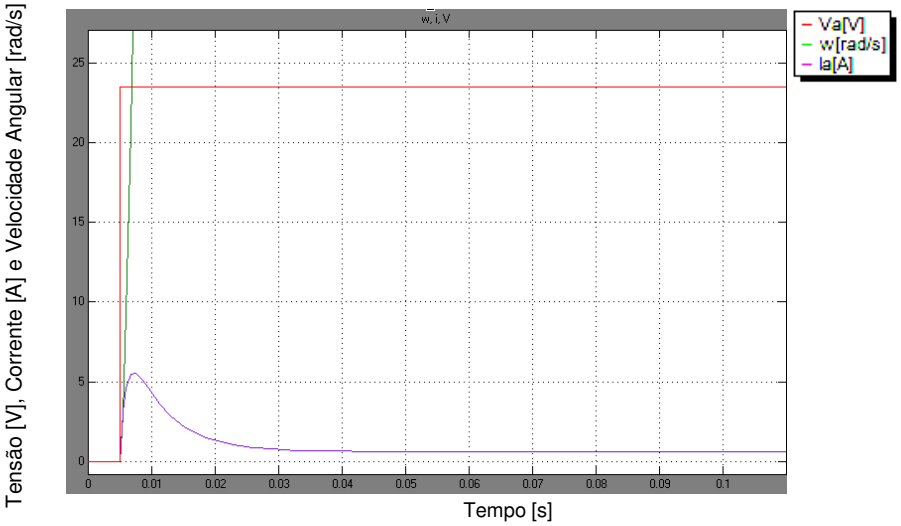


Figura 4.16: Gráficos simulados com parâmetros da tabela 4.8 – ENGEL (ampliado)

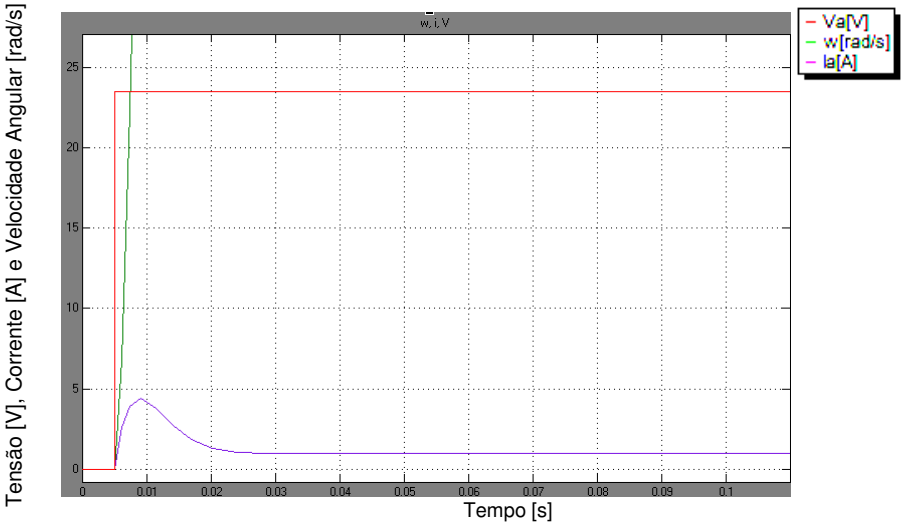


Figura 4.17: Gráficos simulados com parâmetros da tabela 4.10 – ENGEL (ampliado)

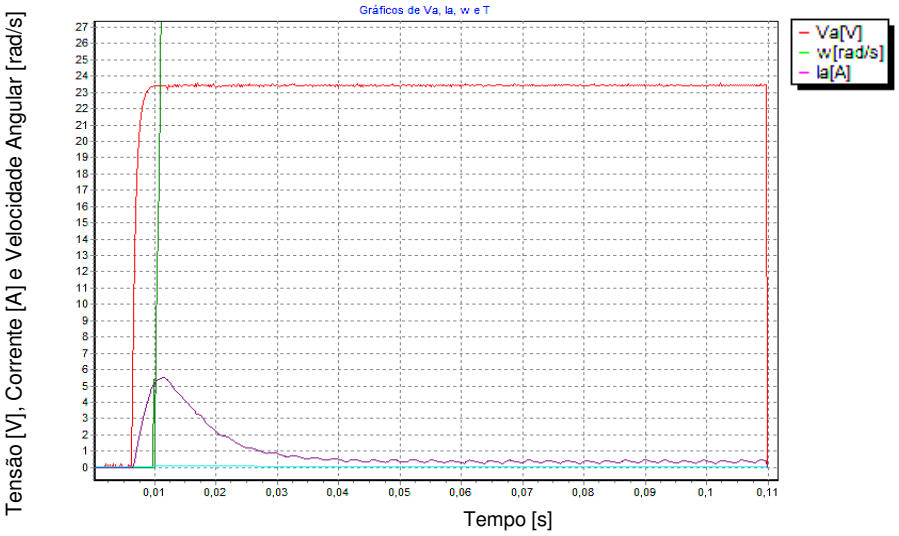


Figura 4.18: Gráficos do ensaio prático - ENGEL (ampliado)

4.3.5. Considerações da análise através de simulação

Comparando-se graficamente os resultados práticos e os simulados através dos parâmetros obtidos, verifica-se que o método matemático empregado é capaz de reproduzir com boa fidelidade os efeitos de velocidade angular e de corrente quando aplicado um sinal de excitação em degrau.

4.4. Ensaio convencional dos motores

Outra forma de se avaliar os resultados obtidos nos ensaios é comparando-os com os resultados obtidos através de ensaios convencionais dos motores. As Tabelas 4.11 a 4.13 mostram esse comparativo.

Para obtenção dos parâmetros através do ensaio convencional, foi utilizado o procedimento descrito no item 2.3.5.

→Motor SERVO DC:

Tabela 4.11: Comparativo dos resultados obtidos com a bancada e através de ensaio convencional - SERVO DC.

	Bancada	Convencional	ε_r (%)
K [V/rad]	$9{,}22.10^{-2}$	$9{,}27.10^{-2}$	0,54
b [N.m.s/rad]	$2{,}95.10^{-4}$	$3{,}48.10^{-4}$	15,23
R [Ω]	1,74	1,81	3,87
L [H]	$1{,}77.10^{-3}$	$1{,}78.10^{-3}$	0,56
J [kg.m²]	$2{,}92.10^{-5}$	$3{,}18.10^{-5}$	8,18

→Motor MICRO SWTCH/PACIFIC CIENTIFIC:

Tabela 4.12: Comparativo dos resultados obtidos com a bancada e através de ensaio convencional - PACIFIC CIENTIFIC.

	Bancada	Convencional	ε_r (%)
K [V/rad]	$4{,}25.10^{-2}$	$4{,}13.10^{-2}$	2,91
b [N.m.s/rad]	$1{,}98.10^{-4}$	$1{,}96.10^{-3}$	89,79
R [Ω]	$8{,}43.10^{-1}$	$8{,}20.10^{-1}$	2,80
L [H]	$4{,}30.10^{-4}$	-	-
J [kg.m²]	$3{,}90.10^{-6}$	$7{,}23.10^{-6}$	46,06

→Motor ENGEL:

Tabela 4.13: Comparativo dos resultados obtidos com a bancada e através de ensaio convencional - ENGEL.

	Bancada	Convencional	ε_r (%)
K [V/rad]	$1{,}44.10^{-1}$	$1{,}39.10^{-1}$	0,72
b [N.m.s/rad]	$4{,}81.10^{-4}$	$5{,}40.10^{-3}$	91,11
R [Ω]	3,44	3,50	1,71
L [H]	$3{,}02.10^{-3}$	$4{,}91.10^{-3}$	38,49
J [kg.m²]	$3{,}35.10^{-5}$	$4{,}50.10^{-5}$	25,56

4.5. *Sensibilidade da resposta com a variação de resultados*

A fim de se avaliar os efeitos da variabilidade dos parâmetros entre alguns ensaios, foi realizada a simulação com os resultados obtidos em dois ensaios do motor "Servo DC" e que apresentam grande diferença.

Tabela 4.14: Dois ensaios distintos - motor SERVO DC.

	Ensaio 1	Ensaio 2
K [V/rad]	$9{,}28.10^{-2}$	$9{,}16.10^{-2}$
b [N.m.s/rad]	$1{,}41.10^{-4}$	$2{,}30.10^{-4}$
R [Ω]	1,72	1,89
L [H]	$1{,}60.10^{-3}$	$2{,}56.10^{-3}$
J [kg.m²]	$3{,}37.10^{-5}$	$2{,}92.10^{-5}$

Os gráficos resultantes das simulações são apresentados a seguir:

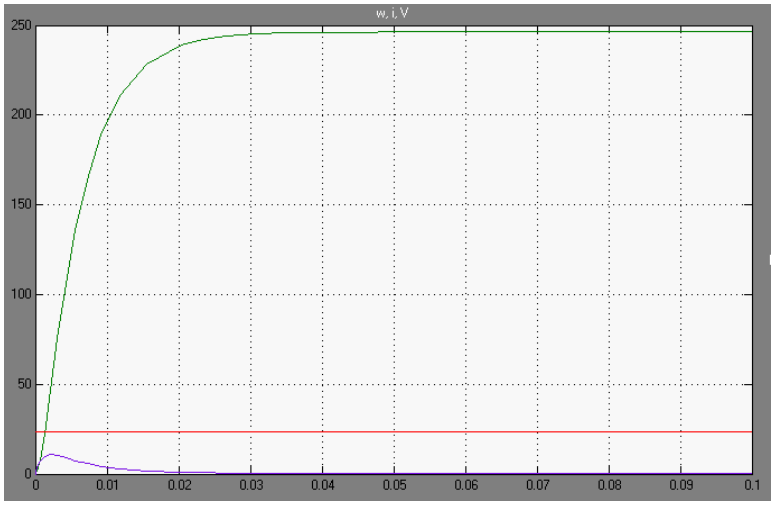


Figura 4.19: Gráficos simulados - Ensaio 1 - SERVO DC

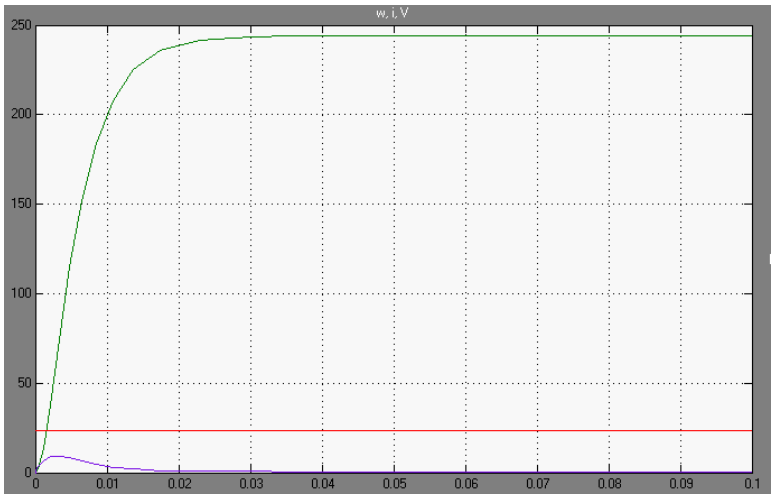


Figura 4.20: Gráficos simulados - Ensaio 2 - SERVO DC

Os gráficos das figuras 4.19 e 4.20 demonstram que, apesar da grande variação principalmente nos valores estimados para o coeficiente de atrito e para a indutância, as respostas da velocidade angular e da corrente de armadura sofreram pequena alteração.

4.6. *Avaliação de rendimento*

Além da estimativa dos parâmetros elétricos e mecânicos da máquina CC, também foram coletados os dados de rendimento para diferentes condições de rotação e carga mecânica.

Os resultados expostos aqui são baseados nos ensaios realizados com o motor "Servo DC" operando em tensão constante de 23,5V. A tabela 4.15 apresenta os dados de velocidade angular, potência mecânica e rendimento. O rendimento e a potência mecânica são definidos pelas equações (4.1) e (4.2) respectivamente.

$$\eta = \frac{P_{mec}}{P_{elétrica}} \tag{4.1}$$

$$P_{mec} = \omega \cdot T \tag{4.2}$$

Tabela 4.15: Tabela de velocidade, potência mecânica e rendimento do motor SERVO DC.

ω [rad/s]	243,6	239,1	237,5	234,2	231,9	229,0	226,7	222,6	225,3
P_{mec} [W]	2,29	6,65	10,93	13,71	20,16	25,71	28,16	31,82	38,93
η [%]	11,11	27,05	40,23	46,81	62,2	73,59	76,64	78,76	77,57

Logo a seguir, é exibida a curva de rendimento traçada a partir desses dados, no software OriginPro (considerando que a velocidade angular fosse constante).

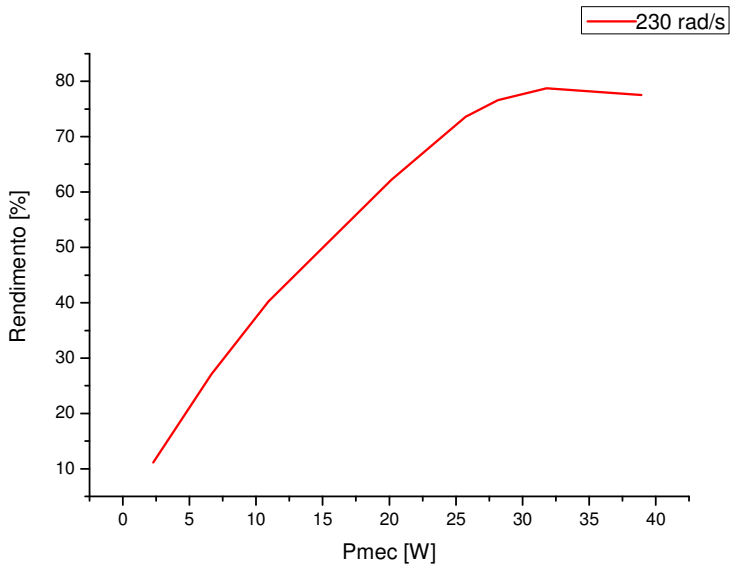


Figura 4.21: Curva de rendimento do motor.

5. CONSIDERAÇÕES FINAIS

5.1. Avaliação dos Objetivos Propostos

Ao término deste trabalho, todos os objetivos propostos inicialmente puderam ser alcançados, alguns com maior ou menor confiabilidade.

Apesar da grande variabilidade de alguns parâmetros estimados para determinados motores, como foi apresentado, isto não representou efeitos muito significativos nas curvas de corrente da armadura e velocidade angular. Dessa forma, é possível obter a equação que descreve o funcionamento dinâmico da máquina CC com razoável confiabilidade e em poucos minutos.

5.2. Limitações do Sistema

Embora não tenha sido testado, motores de potência bastante reduzida (menores que 10W), dificilmente apresentarão bons resultados tendo em vista a inércia e atrito do gerador empregado. Além disso, motores de pequeno porte também não permitem que seja feita a sua fixação junto ao suporte na bancada e nem acoplamento ao gerador.

Outras limitações técnicas:

- Velocidade angular máxima: 4500 rpm;
- Corrente de armadura máxima em regime: 7A;
- Tensão de armadura máxima: 24V;
- Tensão de armadura mínima: 3V.

Para a determinação do rendimento da máquina CC, motores de baixa rotação, abaixo de 2000 rpm, também não são recomendados, uma vez que a carga mecânica é proporcional à carga elétrica, que depende da velocidade angular do gerador.

5.3. Conclusões

Ao longo deste trabalho foi desenvolvida uma bancada para ensaios de máquinas de corrente contínua com excitação independente, capaz de estimar os seus parâmetros elétricos e mecânicos. A metodologia empregada consiste, basicamente, em aplicar uma excitação em tensão do tipo degrau na armadura de um motor CC e avaliar a sua resposta, durante o transiente, ou seja, até entrar em regime permanente. A estimativa dos parâmetros é possível utilizando-se a equação diferencial que descreve o seu funcionamento dinâmico juntamente com o método dos mínimos quadrados.

Foram ensaiados três motores, alguns deles cujas informações eram desconhecidas. Em todos os ensaios foi possível estimar, principalmente, a constante de força contra-eletromotriz e a resistência da armadura com boa confiabilidade. Já o coeficiente de atrito e a indutância da armadura, em alguns testes, apresentaram um elevado desvio-padrão que, no entanto, não tiveram influência muito significativa nas curvas de corrente e velocidade angular simuladas.

Quando comparados os parâmetros obtidos a partir de ensaios com diferentes tensões de excitação, percebeu-se que quanto maior for essa excitação, mais confiáveis serão os resultados. Dessa forma, é preferível que a realização dos testes seja feita com a tensão nominal do motor.

Em relação ao tempo necessário para o ensaio na bancada, perdeu-se aproximadamente quinze minutos para fazer o alinhamento da máquina, e um minuto para obtenção dos seus parâmetros. Já para a realização do ensaio convencional, além do tempo necessário para alinhamento, levaram-se aproximadamente quarenta minutos para a obtenção dos parâmetros.

5.4. Sugestões para Trabalhos Futuros

Sugere-se que sejam utilizadas estruturas diferentes para estimativa dos parâmetros e para levantamento da curva de rendimento.

Para estimativa dos parâmetros, é preferível que não se use um gerador, apenas um taco-gerador ou *encoder* com atrito e inércia bastante reduzidos ou, se possível, até desprezíveis quando comparados aos do motor objeto dos testes. Nesse caso, não é necessária a utilização da célula de carga.

Já para determinação da curva de rendimento, é interessante que se utilize um gerador com maior capacidade de corrente e resistências menores também, de

forma que seja possível aplicar uma carga mecânica elevada mesmo em baixas rotações. Neste caso, podem-se utilizar amortecedores a fim de minimizar a vibração.

REFERÊNCIAS

[1] Basilio, J. C.; Moreira, M. V. – Experimentos para Estimação dos Parâmetros de Motores de Corrente Contínua – Porto Alegre – Cobenge, 2001

[2] Fitzgerald, A. E.; Kingsley Jr., C.; Umans, S. D. – Máquinas Elétricas – 6ª Ed. – Porto Alegre: Bookman, 2006

[3] Nasar, Syed. A. – Máquinas Elétricas – São Paulo: McGraw-Hill do Brasil, 1984

[4] Saab, S. S.; Kaed-Bey, R. A. – Parameter Identification of a DC Motor: Na Experimental Approach - IEEE

[5] Burian, R.; Lima, A. C. – Cálculo Numérico – 1ª Ed. – Rio de Janeiro: LTC, 2007

[6] Aguirre, L. A. – Introdução à Identificação de Sistemas – 3ª Ed. – Belo Horizonte: UFMG, 2007

[7] Hadef, M.; Bourouina, A.; Mekideche, M. R. – Parameter Identification of a DC Motor Using Moments Method – International Journal of Electrical and Power Engineering – Medwell Journals, 2007

[8] Al-Qassar, A. A.; Othman, M. Z. – Experimental Determination of Electrical and Mechanical Parameters of DC Motor Using Genetic Neural Network – Journal of Engineering Science and Technology – Taylos's University College, 2008

[9] Balbinot, A.; Brusamarello, V. J. – Instrumentação e Fundamentos de Medidas – Volume 2 – 1ª Ed. – Rio de Janeiro: LTC, 2007

[10] Andrade, Darizon A. – Notas de Aula: Acionamento de Máquinas Elétricas - UFU

[11] Souza, Márcio R. de – Monografia: Estimação Utilizando Mínimos Quadrados Recursivo (MQR) e Controle de Velocidade do Motor DC Servo Trainer – UFC – Fortaleza, 2009

[12] Texas Instruments – Datasheet MSP430F2274 – Dallas, 2007

[13] Burr-Brown – Datasheet DAC7614 – Estados Unidos, 1998

[14] National Semiconductor – Datasheet LM675 – 1999

[15] Burr-Brown – Datasheet INA126 – Estados Unidos, 1996

[16] Burr-Brown – Datasheet INA200 - 2007

OBRAS CONSULTADAS

Falcone, Aurio G. – Eletromecânica – Volume 2 – São Paulo: Edgard Blücher, 1979

Toro, Vincent Del – Fundamentos de Máquinas Elétricas – Rio de Janeiro: Prentice-Hall do Brasil, 1994

APÊNDICE A – TABELAS DE ENSAIOS

As médias e desvios-padrão calculados no item 4.2. são baseados nos dados expostos nas tabelas abaixo:

→Motor SERVO DC – Tensão de excitação: 23,5V

Ensaio	1	2	3	4	5	6	7	8	9	10
K	$9{,}05.10^{-2}$	$9{,}15.10^{-2}$	$9{,}26.10^{-2}$	$9{,}19.10^{-2}$	$9{,}42.10^{-2}$	$9{,}28.10^{-2}$	$9{,}20.10^{-2}$	$9{,}36.10^{-2}$	$9{,}16.10^{-2}$	$9{,}12.10^{-2}$
b	$2{,}54.10^{-4}$	$3{,}06.10^{-4}$	$1{,}40.10^{-4}$	$3{,}57.10^{-4}$	$5{,}24.10^{-4}$	$1{,}41.10^{-4}$	$3{,}52.10^{-4}$	$3{,}6.10^{-4}$	$2{,}3.10^{-4}$	$2{,}88.10^{-4}$
R	1,83	1,76	1,70	1,66	1,71	1,72	1,70	1,69	1,89	1,77
L	$1{,}65.10^{-3}$	$1{,}54.10^{-3}$	$1{,}22.10^{-3}$	$1{,}23.10^{-3}$	$2{,}46.10^{-3}$	$1{,}6.10^{-3}$	$1{,}66.10^{-3}$	$2{,}09.10^{-3}$	$2{,}56.10^{-3}$	$1{,}70.10^{-3}$
J	$2{,}64.10^{-5}$	$2{,}87.10^{-5}$	$3{,}34.10^{-5}$	$2{,}88.10^{-5}$	$2{,}45.10^{-5}$	$3{,}37.10^{-5}$	$2{,}81.10^{-5}$	$2{,}92.10^{-5}$	$2{,}92.10^{-5}$	$2{,}96.10^{-5}$

→Motor SERVO DC – Tensão de excitação: 21V

Ensaio	1	2	3	4	5	6	7	8	9	10
K	$9{,}34.10^{-2}$	$9{,}09.10^{-2}$	$9{,}24.10^{-2}$	$8{,}98.10^{-2}$	$9{,}39.10^{-2}$	$9{,}48.10^{-2}$	$9{,}38.10^{-2}$	$9{,}41.10^{-2}$	$9{,}14.10^{-2}$	$9{,}41.10^{-2}$
b	$3{,}57.10^{-4}$	$3{,}08.10^{-4}$	$2{,}71.10^{-4}$	$2{,}21.10^{-4}$	$3{,}08.10^{-4}$	$2{,}70.10^{-4}$	$2{,}28.10^{-4}$	$8{,}25.10^{-4}$	$3{,}47.10^{-4}$	$3{,}46.10^{-4}$
R	1,84	1,84	1,81	1,71	1,70	1,55	1,78	1,81	1,69	1,67
L	$3{,}26.10^{-3}$	$2{,}59.10^{-3}$	$2{,}10.10^{-3}$	$1{,}04.10^{-3}$	$2{,}08.10^{-3}$	$6{,}71.10^{-4}$	$2{,}52.10^{-3}$	$3{,}29.10^{-3}$	$9{,}51.10^{-4}$	$1{,}86.10^{-3}$
J	$3{,}03.10^{-5}$	$3{,}11.10^{-5}$	$3{,}27.10^{-5}$	$3{,}43.10^{-5}$	$3{,}11.10^{-5}$	$3{,}40.10^{-5}$	$3{,}51.10^{-5}$	$1{,}44.10^{-5}$	$2{,}98.10^{-5}$	$3{,}00.10^{-5}$

→Motor SERVO DC – Tensão de excitação: 18V

Ensaio	1	2	3	4	5	6	7	8	9	10
K	$9{,}39.10^{-2}$	$9{,}39.10^{-2}$	$9{,}36.10^{-2}$	$9{,}36.10^{-2}$	$9{,}48.10^{-2}$	$9{,}31.10^{-2}$	$9{,}39.10^{-2}$	$9{,}56.10^{-2}$	$9{,}40.10^{-2}$	$9{,}49.10^{-2}$
b	$5{,}11.10^{-4}$	$2{,}90.10^{-4}$	$3{,}47.10^{-4}$	$4{,}08.10^{-4}$	$3{,}86.10^{-4}$	$7{,}53.10^{-4}$	$9{,}14.10^{-4}$	$2{,}40.10^{-4}$	$2{,}42.10^{-4}$	$2{,}85.10^{-4}$
R	1,76	1,65	1,78	1,69	1,83	1,97	1,85	1,59	1,77	1,70
L	$2{,}20.10^{-3}$	$7{,}81.10^{-4}$	$1{,}78.10^{-3}$	$1{,}42.10^{-3}$	$3{,}13.10^{-3}$	$3{,}83.10^{-4}$	$3{,}43.10^{-3}$	$1{,}08.10^{-3}$	$2{,}15.10^{-4}$	$1{,}82.10^{-3}$
J	$2{,}54.10^{-5}$	$3{,}29.10^{-5}$	$3{,}24.10^{-5}$	$3{,}17.10^{-5}$	$3{,}32.10^{-5}$	$1{,}64.10^{-5}$	$1{,}67.10^{-5}$	$3{,}50.10^{-5}$	$3{,}42.10^{-5}$	$3{,}39.10^{-5}$

→ Motor PACIFIC CIENTIFIC – Tensão de excitação: 23,5V

Ensaio	1	2	3	4	5	6	7	8	9	10
K	$4{,}24.10^{-2}$	$4{,}22.10^{-2}$	$4{,}33.10^{-2}$	$4{,}22.10^{-2}$	$4{,}31.10^{-2}$	$4{,}32.10^{-2}$	$4{,}21.10^{-2}$	$4{,}10.10^{-2}$	$4{,}23.10^{-2}$	$4{,}29.10^{-2}$
b	$2{,}06.10^{-4}$	$1{,}90.10^{-4}$	$1{,}72.10^{-4}$	$2{,}07.10^{-4}$	$1{,}84.10^{-4}$	$1{,}87.10^{-4}$	$1{,}80.10^{-4}$	$2{,}62.10^{-4}$	$2{,}12.10^{-4}$	$1{,}81.10^{-4}$
R	$8{,}53.10^{-1}$	$8{,}53.10^{-1}$	$8{,}29.10^{-1}$	$8{,}49.10^{-1}$	$8{,}47.10^{-1}$	$8{,}46.10^{-1}$	$8{,}39.10^{-1}$	$8{,}41.10^{-1}$	$8{,}27.10^{-1}$	$8{,}49.10^{-1}$
L	$4{,}98.10^{-4}$	$3{,}51.10^{-4}$	$3{,}74.10^{-4}$	$4{,}39.10^{-4}$	$7{,}21.10^{-4}$	$6{,}18.10^{-4}$	$3{,}79.10^{-4}$	$1{,}68.10^{-4}$	$2{,}38.10^{-4}$	$5{,}15.10^{-4}$
J	$3{,}92.10^{-6}$	$3{,}96.10^{-6}$	$4{,}12.10^{-6}$	$3{,}75.10^{-6}$	$4{,}24.10^{-6}$	$4{,}07.10^{-6}$	$4{,}21.10^{-6}$	$2{,}65.10^{-6}$	$3{,}79.10^{-6}$	$4{,}27.10^{-6}$

→ Motor PACIFIC CIENTIFIC – Tensão de excitação: 21V

Ensaio	1	2	3	4	5	6	7	8	9	10
K	$4{,}06.10^{-2}$	$4{,}17.10^{-2}$	$4{,}13.10^{-2}$	$4{,}14.10^{-2}$	$4{,}12.10^{-2}$	$4{,}03.10^{-2}$	$4{,}19.10^{-2}$	$4{,}17.10^{-2}$	$3{,}96.10^{-2}$	$4{,}10.10^{-2}$
b	$1{,}61.10^{-4}$	$1{,}70.10^{-4}$	$1{,}51.10^{-4}$	$1{,}39.10^{-4}$	$1{,}90.10^{-4}$	$8{,}68.10^{-5}$	$1{,}02.10^{-4}$	$1{,}25.10^{-4}$	$1{,}23.10^{-4}$	$1{,}41.10^{-4}$
R	$8{,}21.10^{-1}$	$8{,}55.10^{-1}$	$8{,}36.10^{-1}$	$8{,}23.10^{-1}$	$8{,}50.10^{-1}$	$8{,}10.10^{-1}$	$8{,}49.10^{-1}$	$8{,}20.10^{-1}$	$8{,}59.10^{-1}$	$8{,}10.10^{-1}$
L	$2{,}62.10^{-4}$	$6{,}76.10^{-4}$	$3{,}78.10^{-4}$	$3{,}36.10^{-4}$	$4{,}43.10^{-4}$	$2{,}73.10^{-6}$	$4{,}29.10^{-4}$	$2{,}41.10^{-4}$	$4{,}82.10^{-4}$	$1{,}08.10^{-5}$
J	$3{,}73.10^{-6}$	$4{,}01.10^{-6}$	$4{,}14.10^{-6}$	$4{,}70.10^{-6}$	$3{,}81.10^{-6}$	$4{,}50.10^{-6}$	$4{,}93.10^{-6}$	$4{,}57.10^{-6}$	$4{,}23.10^{-6}$	$4{,}60.10^{-6}$

→ Motor PACIFIC CIENTIFIC – Tensão de excitação: 18V

Ensaio	1	2	3	4	5	6	7	8	9	10
K	$4{,}12.10^{-2}$	$4{,}16.10^{-2}$	$4{,}12.10^{-2}$	$4{,}30.10^{-2}$	$4{,}30.10^{-2}$	$4{,}24.10^{-2}$	$4{,}20.10^{-2}$	$4{,}14.10^{-2}$	$4{,}16.10^{-2}$	$4{,}22.10^{-2}$
b	$6{,}68.10^{-5}$	$1{,}29.10^{-4}$	$1{,}44.10^{-4}$	$1{,}23.10^{-4}$	$5{,}78.10^{-5}$	$9{,}85.10^{-5}$	$1{,}02.10^{-4}$	$1{,}37.10^{-4}$	$1{,}99.10^{-4}$	$5{,}89.10^{-5}$
R	$7{,}96.10^{-1}$	$7{,}80.10^{-1}$	$7{,}99.10^{-1}$	$6{,}99.10^{-1}$	$7{,}03.10^{-1}$	$7{,}52.10^{-1}$	$8{,}00.10^{-1}$	$7{,}59.10^{-1}$	$7{,}89.10^{-1}$	$7{,}61.10^{-1}$
L	$1{,}87.10^{-4}$	$8{,}48.10^{-5}$	$3{,}78.10^{-4}$	$3{,}08.10^{-5}$	$1{,}54.10^{-4}$	$2{,}28.10^{-4}$	$4{,}9.10^{-4}$	$8{,}23.10^{-5}$	$3{,}53.10^{-4}$	$2{,}85.10^{-4}$
J	$6{,}02.10^{-6}$	$5{,}79.10^{-6}$	$5{,}65.10^{-6}$	$6{,}59.10^{-6}$	$6{,}74.10^{-6}$	$6{,}31.10^{-6}$	$5{,}91.10^{-6}$	$5{,}40.10^{-6}$	$5{,}07.10^{-6}$	$6{,}78.10^{-6}$

→ Motor ENGEL – Tensão de excitação: 23,5V

Ensaio	1	2	3	4	5	6	7	8	9	10
K	$1{,}46.10^{-1}$	$1{,}44.10^{-1}$	$1{,}45.10^{-1}$	$1{,}47.10^{-1}$	$1{,}39.10^{-1}$	$1{,}48.10^{-1}$	$1{,}52.10^{-1}$	$1{,}42.10^{-1}$	$1{,}41.10^{-1}$	$1{,}38.10^{-1}$
b	$7{,}94.10^{-4}$	$3{,}70.10^{-4}$	$2{,}06.10^{-4}$	$4{,}36.10^{-4}$	$1{,}17.10^{-4}$	$6{,}25.10^{-4}$	$3{,}89.10^{-4}$	$3{,}63.10^{-4}$	$7{,}66.10^{-4}$	$7{,}39.10^{-4}$
R	3,52	3,54	3,04	3,41	3,86	3,22	3,02	3,30	3,81	3,69
L	$4{,}36.10^{-3}$	$3{,}97.10^{-3}$	$3{,}69.10^{-5}$	$2{,}96.10^{-3}$	$3{,}32.10^{-3}$	$3{,}18.10^{-3}$	$1{,}59.10^{-3}$	$1{,}15.10^{-3}$	$6{,}36.10^{-3}$	$3{,}30.10^{-3}$
J	$3{,}03.10^{-5}$	$4{,}12.10^{-5}$	$4{,}53.10^{-5}$	$4{,}05.10^{-5}$	$1{,}20.10^{-5}$	$4{,}03.10^{-5}$	$4{,}59.10^{-5}$	$3{,}94.10^{-5}$	$2{,}25.10^{-5}$	$1{,}71.10^{-5}$

→ Motor ENGEL – Tensão de excitação: 21V

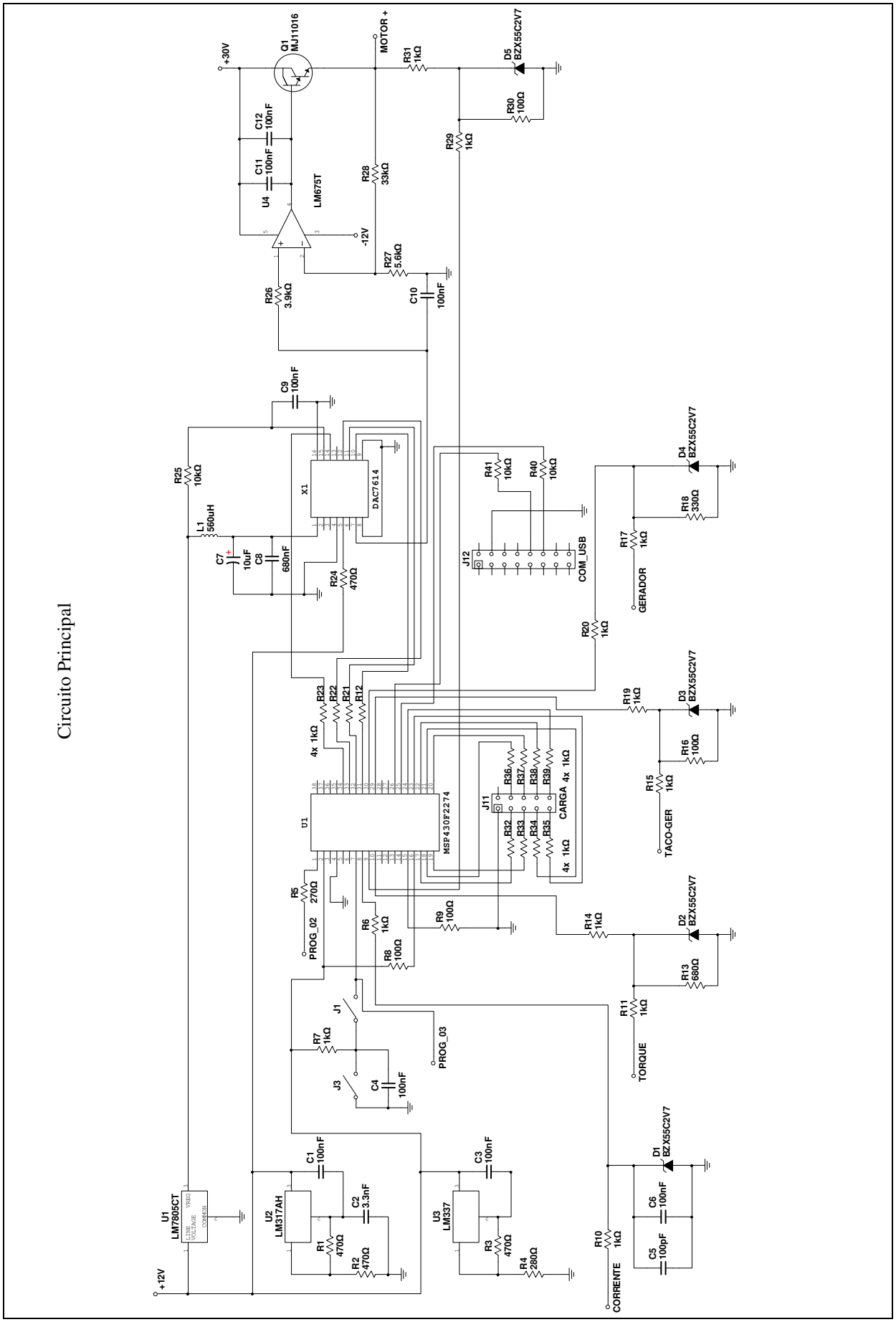
Ensaio	1	2	3	4	5	6	7	8	9	10
K	$1{,}43.10^{-1}$	$1{,}40.10^{-1}$	$1{,}42.10^{-1}$	$1{,}42.10^{-1}$	$1{,}40.10^{-1}$	$1{,}37.10^{-1}$	$1{,}43.10^{-1}$	$1{,}41.10^{-1}$	$1{,}43.10^{-1}$	$1{,}36.10^{-1}$
b	$4{,}89.10^{-4}$	$1{,}18.10^{-3}$	$1{,}01.10^{-4}$	$4{,}19.10^{-4}$	$1{,}23.10^{-4}$	$1{,}12.10^{-3}$	$5{,}74.10^{-4}$	$8{,}52.10^{-4}$	$4{,}57.10^{-4}$	$1{,}22.10^{-3}$
R	3,59	3,98	3,55	3,61	4,47	4,04	3,60	3,70	3,51	4,39
L	$6{,}53.10^{-3}$	$9{,}20.10^{-3}$	$6{,}99.10^{-3}$	$6{,}33.10^{-3}$	$1{,}43.10^{-2}$	$9{,}05.10^{-3}$	$6{,}94.10^{-3}$	$8{,}75.10^{-3}$	$5{,}65.10^{-3}$	$8{,}63.10^{-3}$
J	$3{,}39.10^{-5}$	$1{,}21.10^{-5}$	$2{,}24.10^{-5}$	$3{,}36.10^{-5}$	$1{,}28.10^{-5}$	$1{,}23.10^{-5}$	$3{,}42.10^{-5}$	$2{,}21.10^{-5}$	$3{,}38.10^{-5}$	$6{,}37.10^{-6}$

→ Motor ENGEL – Tensão de excitação: 18V

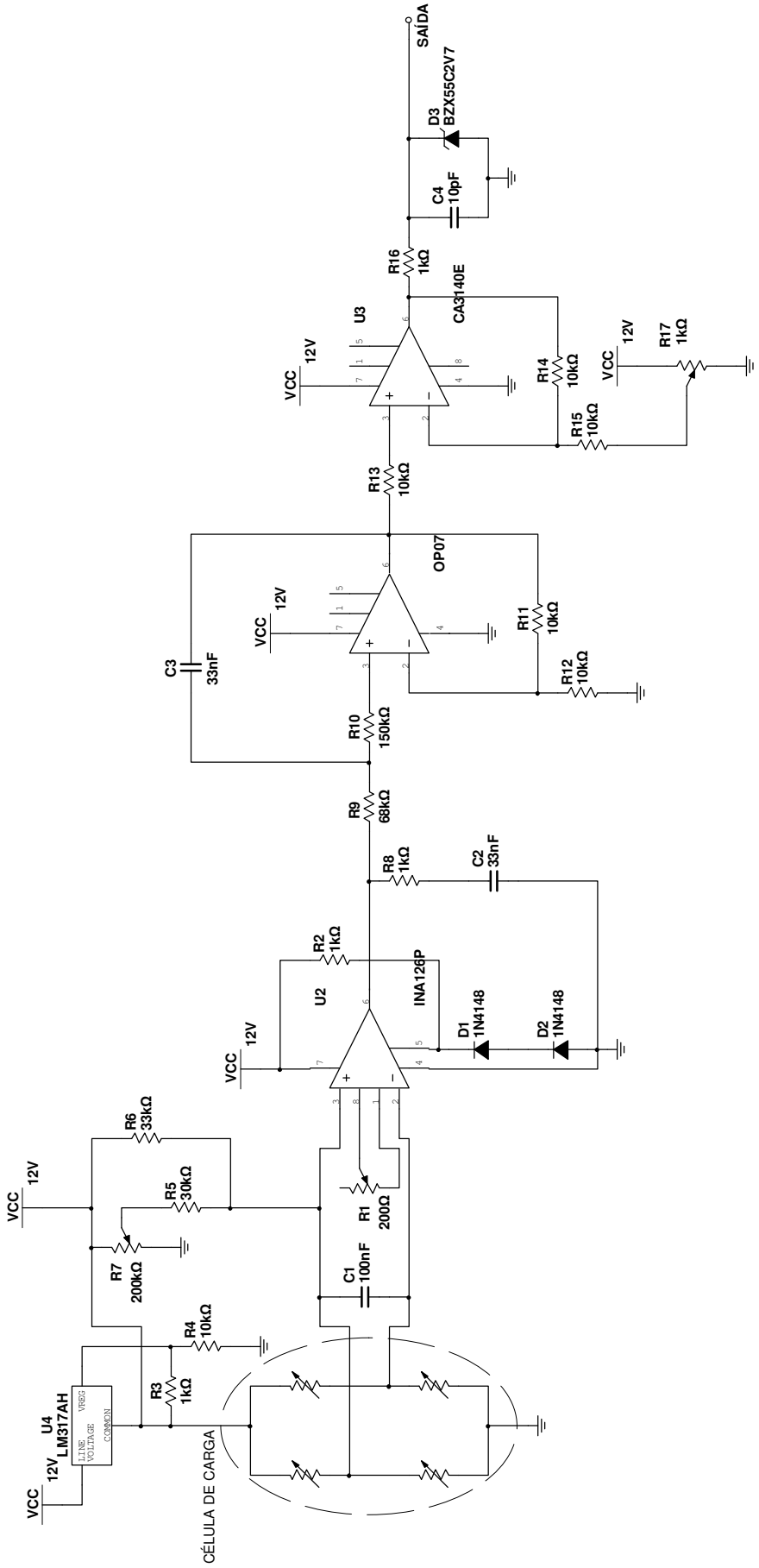
Ensaio	1	2	3	4	5	6	7	8	9	10
K	$1{,}40.10^{-1}$	$1{,}44.10^{-1}$	$1{,}44.10^{-1}$	$1{,}47.10^{-1}$	$1{,}41.10^{-1}$	$1{,}40.10^{-1}$	$1{,}43.10^{-1}$	$1{,}41.10^{-1}$	$1{,}44.10^{-1}$	$1{,}41.10^{-1}$
b	$1{,}31.10^{-3}$	$4{,}19.10^{-4}$	$1{,}15.10^{-3}$	$6{,}26.10^{-4}$	$1{,}54.10^{-3}$	$1{,}08.10^{-4}$	$1{,}35.10^{-3}$	$6{,}02.10^{-4}$	$5{,}74.10^{-4}$	$1{,}15.10^{-3}$
R	4,25	3,18	3,66	3,17	4,37	3,91	3,99	3,51	3,75	4,23
L	$1{,}10.10^{-2}$	$9{,}26.10^{-3}$	$5{,}73.10^{-3}$	$3{,}03.10^{-3}$	$1{,}13.10^{-2}$	$8{,}01.10^{-3}$	$9{,}20.10^{-3}$	$4{,}93.10^{-3}$	$7{,}16.10^{-3}$	$1{,}06.10^{-2}$
J	$1{,}25.10^{-5}$	$4{,}18.10^{-5}$	$2{,}02.10^{-5}$	$3{,}91.10^{-5}$	$9{,}38.10^{-6}$	$1{,}73.10^{-5}$	$1{,}63.10^{-5}$	$3{,}73.10^{-5}$	$3{,}46.10^{-5}$	$1{,}26.10^{-5}$

APÊNDICE B – DIAGRAMAS ELÉTRICOS

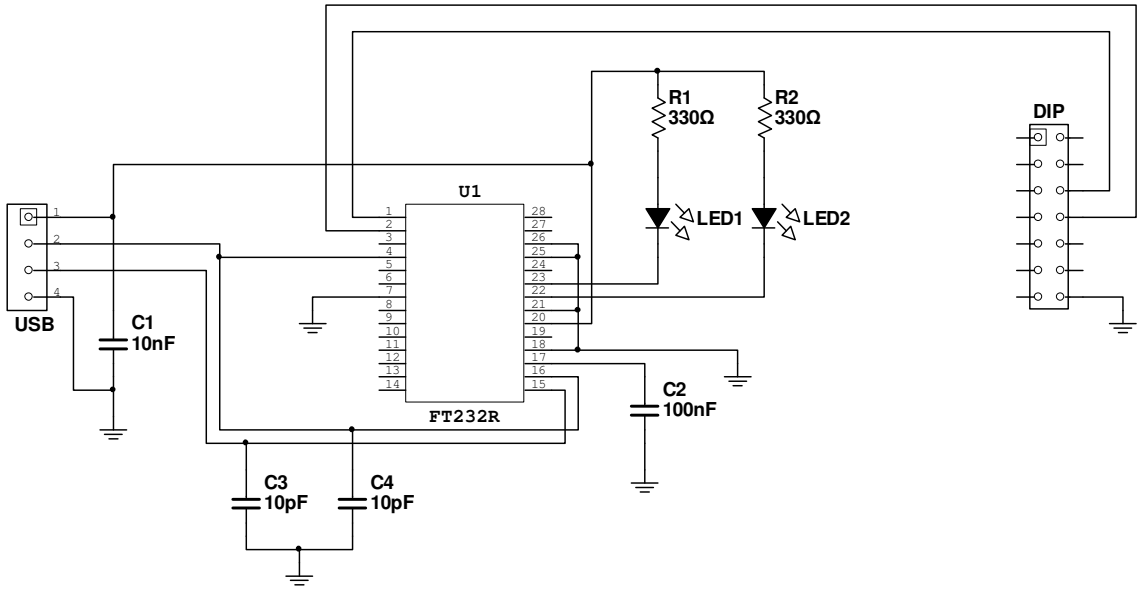
Circuito Principal



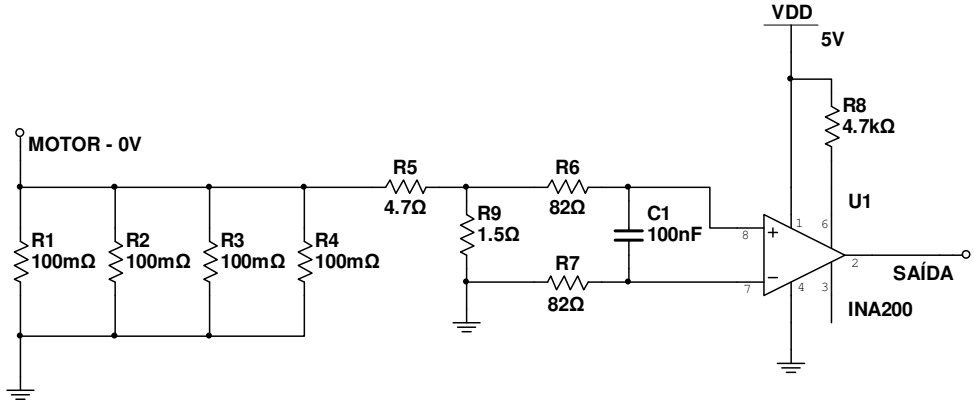
Condicionador da Célula de Carga



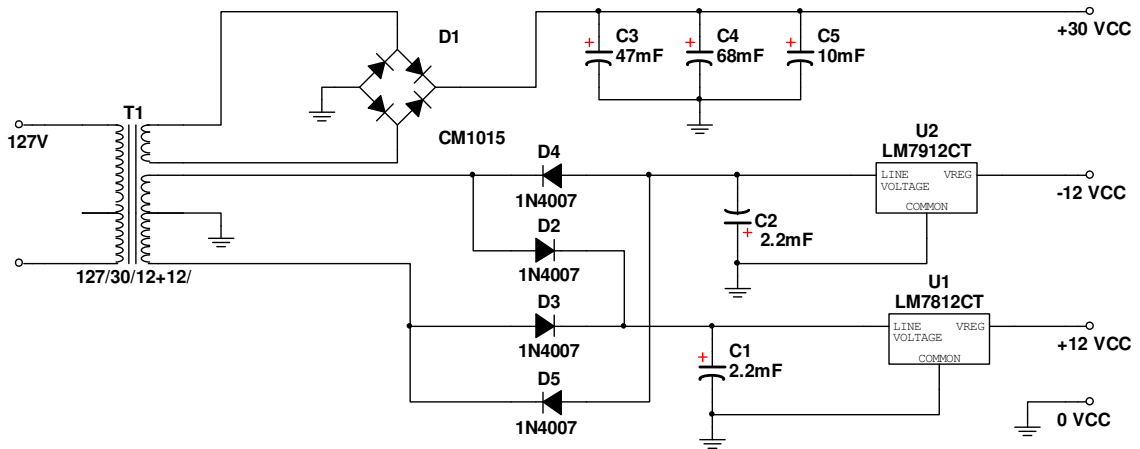
Conversor USB ←→ UART



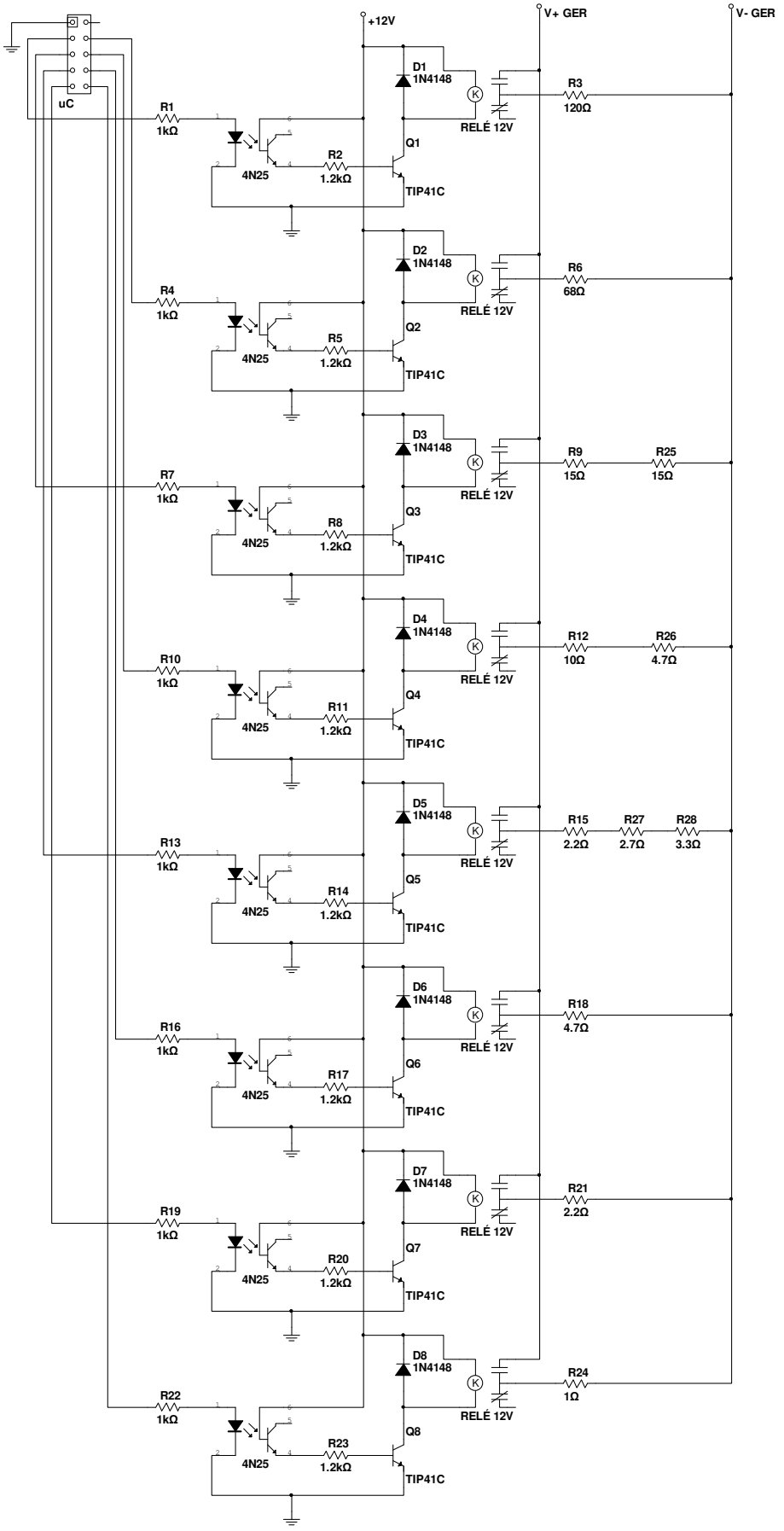
Condicionador do Sinal de Corrente



Fonte de Alimentação



Cargas Resistivas



APÊNDICE C – CÓDIGO FONTE DO MICROCONTROLADOR

```
#include "msp430x22x4.h"
char string1[13] = {0};
int i, j, k;
char *adcdado;

char adc_dado;
int cmil;
int dmil;
int mil;
int cem;
int dez;
int um;

long int Vout=0;
long int Rload=0;

void ADC_ini (void);
void Get_ADC(unsigned int a);
void Delay (unsigned int a);
void DAC_send_data (long int val_dac);
void DAC_send_bit (int val_bit);

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;             // Stop watchdog timer
  P1DIR |= 0x01;                        // Set P1.0 to output direction
  P1DIR |= 0x0f;
  P4DIR |= 0xff;
  P4OUT = 0x00;


  //Comunicaçao serial

  BCSCTL1 = CALBC1_16MHZ;
  DCOCTL = CALDCO_16MHZ;
  P3SEL = 0X30;
  //UCA0CTL1 |= UCSSEL_2;
  UCA0CTL1 |= UCSSEL_3;
  //UCA0BR0 = 104;
  //UCA0BR0 = 70;
  UCA0BR0 = 17;
  UCA0BR1 = 0;
  UCA0MCTL = UCBRS0 + UCBRS2;
  UCA0CTL1 &= ~UCSWRST;
  IE2 |= UCA0RXIE;
  P4OUT = 0x01;
  ADC_ini();
  __bis_SR_register(LPM0_bits + GIE); //nao finaliza o programa
  }
```

```
#pragma vector=USCIAB0TX_VECTOR
  __interrupt void USCI0TX_ISR(void)
  {
    UCA0TXBUF = string1[i++];

    if (i == sizeof string1 - 1)
      IE2 &= ~UCA0TXIE;
  }


#pragma vector=USCIAB0RX_VECTOR
  __interrupt void USCI0RX_ISR(void)
  {

    if (UCA0RXBUF == 'U')
    {
      while(1){

      i = 0;
      j = 0;

    Get_ADC(INCH_4);
        string1[i]='A';
        i++;
        string1[i]='A';
        i++;

        adcdado=(char *)0x401;
        adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;
        adcdado=(char *)0x400;
          adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;

        adcdado=(char *)0x403;
        adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;
        adcdado=(char *)0x402;
          adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;

        adcdado=(char *)0x405;
        adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;
        adcdado=(char *)0x404;
          adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;

        adcdado=(char *)0x407;
        adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;
        adcdado=(char *)0x406;
          adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;

        adcdado=(char *)0x409;
        adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;
        adcdado=(char *)0x408;
```

```
          adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;

          string1[i] = '\n';

      i=0;

      IE2 |= UCA0TXIE;

      Delay(250);
  }
  }

    if (UCA0RXBUF == 'u')
    {
      i = 0;
      j = 0;

      Get_ADC(INCH_4);

        string1[i]='A';
        i++;
        string1[i]='A';
        i++;

        adcdado=(char *)0x401;
        adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;
        adcdado=(char *)0x400;
          adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;

        adcdado=(char *)0x403;
        adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;
        adcdado=(char *)0x402;
          adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;

        adcdado=(char *)0x405;
        adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;
        adcdado=(char *)0x404;
          adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;

        adcdado=(char *)0x407;
        adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;
        adcdado=(char *)0x406;
          adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;

        adcdado=(char *)0x409;
        adc_dado=*adcdado;
          string1[i]=adc_dado;
          i++;
        adcdado=(char *)0x408;
          adc_dado=*adcdado;
          string1[i]=adc_dado;
```

```
        i++;

        string1[i] = '\n';

    i=0;

    IE2 |= UCA0TXIE;
  }



  //-------------------------------------------------------------
  //Controle do DA - Tensao aplicada no motor: R = incremento
  //-------------------------------------------------------------

  if (UCA0RXBUF == 'R')
  {
   if (Vout<4086)
   {
     Vout=Vout+10;
   }
   DAC_send_data(Vout);
  }

  if (UCA0RXBUF == 'S')
  {
   if (Vout<4086)
   {
     Vout=Vout+50;
   }
   DAC_send_data(Vout);
  }

  if (UCA0RXBUF == 'T')
  {
   if (Vout<4086)
   {
     Vout=Vout+100;
   }
   DAC_send_data(Vout);
  }

  if (UCA0RXBUF == '1')
  {
     Vout=510;
   DAC_send_data(Vout);
  }

  if (UCA0RXBUF == '2')
  {
     Vout=1020;
   DAC_send_data(Vout);
  }

  if (UCA0RXBUF == '3')
  {
     Vout=1530;
   DAC_send_data(Vout);
  }

  if (UCA0RXBUF == '4')
  {
     Vout=2040;
   DAC_send_data(Vout);
  }

  if (UCA0RXBUF == '5')
  {
     Vout=2550;
```

```
     DAC_send_data(Vout);
    }

    if (UCA0RXBUF == '6')
    {
       Vout=3060;
     DAC_send_data(Vout);
    }

    if (UCA0RXBUF == '7')
    {
       Vout=3570;
     DAC_send_data(Vout);
    }

    if (UCA0RXBUF == '8')
    {
       Vout=4080;
     DAC_send_data(Vout);
    }



    //-------------------------------------------------------
    //Controle do DA - Tensao aplicada no motor: r = decremento
    //-------------------------------------------------------

    if (UCA0RXBUF == 'r')
    {
     if (Vout>=10)
     {
       Vout=Vout-10;
     }
     DAC_send_data(Vout);
    }

    //-------------------------------------------------------
    //Controle da carga: L = incremento
    //-------------------------------------------------------

    if (UCA0RXBUF == 'L')
    {
     if (Rload<245)
     {
       Rload = Rload+10;
       P4OUT = Rload;
     }
    }

    //-------------------------------------------------------
    //Controle da carga: l = decremento
    //-------------------------------------------------------

    if (UCA0RXBUF == 'l')
    {
     if (Rload>=10)
     {
       Rload = Rload-10;
       P4OUT = Rload;
     }
    }

  }

    //-------------------------------------------------
    //Declaracao das funçoes ----------------------------
    //-------------------------------------------------

void ADC_ini (void)
```

```
{
  ADC10CTL1 |= INCH_4 + CONSEQ_1 + ADC10SSEL_3;
  ADC10CTL0 = 0;
  ADC10CTL0 = ADC10SHT_3 + MSC + ADC10ON + ADC10IE;
  ADC10DTC1 = 0X05;
  ADC10CTL0 |= ENC;  // Conversion enabled
  while (ADC10CTL1 & BUSY);
}

void Get_ADC(unsigned int a)
{
  ADC10CTL0 &= ~ENC;                                  //disable convertion

  ADC10SA = 0x400;

  ADC10CTL0 |= ENC + ADC10SC;                         //enable convertion
  __bis_SR_register(CPUOFF + GIE);
}

void Delay (unsigned int a)
{
  unsigned char k;
  for (k=0 ; k != a; ++k);                        //20+a*12 cycles (for
1MHz)(1.25+0.75 us)
}

//-------------------------------------------------
//Envio de um unico bit para o DAC ----------------
//-------------------------------------------------
void DAC_send_bit (int val_bit)
{
  P1OUT &= ~0x02;                               // dac_clock=0
  if(val_bit==0)                                //dac_dado
  {
    P1OUT &= ~0x01;
  }
  else
  {
    P1OUT |= 0x01;
  }
  P1OUT |= 0x02;
}


//-------------------------------------------------
//Envio do dado para o DAC ------------------------
//-------------------------------------------------

void DAC_send_data (long int val_dac)
{
  int DAC_0;
  int DAC_1;
  int DAC_2;
  int DAC_3;
  int DAC_4;
  int DAC_5;
  int DAC_6;
  int DAC_7;
  int DAC_8;
  int DAC_9;
  int DAC_10;
  int DAC_11;

  P1OUT |= 0x08;                               // dac_load=1
  P1OUT &= ~0x04;

     DAC_0=val_dac%2;
     val_dac=val_dac/2;
     DAC_1=val_dac%2;
```

```
      val_dac=val_dac/2;
      DAC_2=val_dac%2;
      val_dac=val_dac/2;
      DAC_3=val_dac%2;
      val_dac=val_dac/2;
      DAC_4=val_dac%2;
      val_dac=val_dac/2;
      DAC_5=val_dac%2;
      val_dac=val_dac/2;
      DAC_6=val_dac%2;
      val_dac=val_dac/2;
      DAC_7=val_dac%2;
      val_dac=val_dac/2;
      DAC_8=val_dac%2;
      val_dac=val_dac/2;
      DAC_9=val_dac%2;
      val_dac=val_dac/2;
      DAC_10=val_dac%2;
      val_dac=val_dac/2;
      DAC_11=val_dac;

  DAC_send_bit (0);                    //A1
  DAC_send_bit (0);                    //A2
  DAC_send_bit (0);                    //X
  DAC_send_bit (0);                    //X
  DAC_send_bit (DAC_11);                    //DAC_11
  DAC_send_bit (DAC_10);                    //DAC_10
  DAC_send_bit (DAC_9);                    //DAC_9
  DAC_send_bit (DAC_8);                    //DAC_8
  DAC_send_bit (DAC_7);                    //DAC_7
  DAC_send_bit (DAC_6);                    //DAC_6
  DAC_send_bit (DAC_5);                    //DAC_5
  DAC_send_bit (DAC_4);                    //DAC_4
  DAC_send_bit (DAC_3);                    //DAC_3
  DAC_send_bit (DAC_2);                    //DAC_2
  DAC_send_bit (DAC_1);                    //DAC_1
  DAC_send_bit (DAC_0);                    //DAC_0

  P1OUT |= 0x04;                           // dac_cs=1
  P1OUT &= ~0x08;                          // dac_load=0
}

#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);
}
```

APÊNDICE D – CÓDIGO FONTE DO SOFTWARE DE ESTIMATIVA DOS PARÂMETROS

Unit1.cpp

```
#include <vcl.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <dos.h>
#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
#include <Excel_XP.h>
// GLOBAL VARIABLES

HANDLE hComm = NULL;
TRead *ReadThread;
COMMTIMEOUTS ctmoNew = {0}, ctmoOld;

AnsiString Cont_cel;
AnsiString CellRef;

char Abre_col = 'A';
char V_out;

int Abre_linha = 1;

int con=0;
int coluna_1, coluna_2;
int linha_1;
int k;
int kk;
int jj;
int n_pulsos;
AnsiString ASData;
AnsiString ASvelocidade;
AnsiString ASbits_parada;
AnsiString salvar;

int velocidade;
int bits_parada;

float Vin_m;
float Vg_m;
float Iin_m;
float Rot_m;
float T_m;
```

```
unsigned long int BytesWritten;
unsigned char *cdata_H;
unsigned char *cdata_L;
unsigned int Length;

FILE *ptabela;
FILE *pconst;
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "Excel_XP_srvr"
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
       : TForm(Owner)
{
hComm = NULL;

          int ii=1;
          AnsiString Porta;
do
{
               Porta=AnsiString("COM")+AnsiString(ii);
               hComm = CreateFile(Porta.c_str(),
                                  GENERIC_READ | GENERIC_WRITE,
                                  0,
                                  0,
                                  OPEN_EXISTING,
                                  FILE_ATTRIBUTE_NORMAL,
                                  0);

                if(hComm != INVALID_HANDLE_VALUE)
                {
              ComboBox1->Items->Add(Porta);
              CloseHandle(hComm);
                     }

              ii++;
              } while(ii<=9);

hComm = NULL;

StringGrid2->Cells[0][0] = AnsiString("V in");
StringGrid2->Cells[1][0] = AnsiString("I in");
StringGrid2->Cells[2][0] = AnsiString("rad/s");
StringGrid2->Cells[3][0] = AnsiString("V ger");
StringGrid2->Cells[4][0] = AnsiString("T");

StringGrid1->Cells[0][0] = AnsiString("V in");
StringGrid1->Cells[1][0] = AnsiString("I in");
StringGrid1->Cells[2][0] = AnsiString("rad/s");
StringGrid1->Cells[3][0] = AnsiString("V ger");
StringGrid1->Cells[4][0] = AnsiString("T");

Image1->Picture->LoadFromFile("Diagrama2.bmp");
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1KeyPress(TObject *Sender, char &Key)
{
  TransmitCommChar(hComm, Key);
  if(Key != 13 && (Key < ' ' || Key > 'z')) Key = 0;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
 DCB dcbCommPort;
  hComm = CreateFile(ComboBox1->Text.c_str(),
```

```
                                    GENERIC_READ | GENERIC_WRITE,
                                    0,
                                    0,
                                    OPEN_EXISTING,
                                    0,
                                    0);

  if(hComm == INVALID_HANDLE_VALUE) Application->Terminate();

  for(int m=0;m<100;m++)
       {
          cdata_H = AnsiString("u").c_str();
              WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
       }

       Sleep(200);

  for(int m=0;m<300;m++)
       {
          cdata_H = AnsiString("L").c_str();
              WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
       }

  CloseHandle(hComm);

  unsigned int RxBufferSize = 300000;
  unsigned int TxBufferSize = 300000;

  SetupComm(hComm, RxBufferSize, TxBufferSize);

  GetCommTimeouts(hComm,&ctmoOld);
  ctmoNew.ReadTotalTimeoutConstant = 0;
  ctmoNew.ReadTotalTimeoutMultiplier = 1;
  ctmoNew.WriteTotalTimeoutMultiplier = 100;
  ctmoNew.WriteTotalTimeoutConstant = 10;
  SetCommTimeouts(hComm, &ctmoNew);

pconst=fopen("config_ser.dat","r");      //Abre o arquivo para escrita
fscanf(pconst,"%d\n%d",&velocidade,&bits_parada);
fclose(pconst);

  dcbCommPort.DCBlength = sizeof(DCB);
  GetCommState(hComm, &dcbCommPort);
  ASvelocidade = AnsiString(velocidade)+",N,8,"+AnsiString(bits_parada);

  BuildCommDCB(ASvelocidade.c_str(), &dcbCommPort);
  SetCommState(hComm, &dcbCommPort);

  Sleep(200);
  hComm = CreateFile(ComboBox1->Text.c_str(),
                                    GENERIC_READ | GENERIC_WRITE,
                                    0,
                                    0,
                                    OPEN_EXISTING,
                                    0,
                                    0);
  if(hComm == INVALID_HANDLE_VALUE) Application->Terminate();

  Sleep(100);

              cdata_H = AnsiString('U').c_str();
              WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
              Sleep(5);

              if(ComboBox2->Text == "3")
              {
                     V_out = '1';
              }
```

```
              if(ComboBox2->Text == "6")
              {
                     V_out = '2';
              }

              if(ComboBox2->Text == "9")
              {
                     V_out = '3';
              }

              if(ComboBox2->Text == "12")
              {
                     V_out = '4';
              }

              if(ComboBox2->Text == "15")
              {
                     V_out = '5';
              }

              if(ComboBox2->Text == "18")
              {
                     V_out = '6';
              }

              if(ComboBox2->Text == "21")
              {
                     V_out = '7';
              }

              if(ComboBox2->Text == "23.5")
              {
                     V_out = '8';
              }

              cdata_H = AnsiString(V_out).c_str();
              WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
              Sleep (100);
              ReadThread = new TRead(false);

              Sleep(2100);

              for(int m1=0;m1<500;m1++)
{
          cdata_H = AnsiString("uuuuuuuuuuuuuuuu").c_str();
              WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
              Sleep(1);
}

  Sleep(100);

              for(int m2=0;m2<500;m2++)
{
          cdata_H = AnsiString("r").c_str();
              WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
              Sleep(1);
}
              CloseHandle(hComm);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
SaveDialog1->Execute();
salvar = SaveDialog1->FileName+".xls";

ptabela = fopen(salvar.c_str(),"w");

for(long int i=1;i<StringGrid2->RowCount;i++)
{
```

```
for(long int ii=0;ii<StringGrid2->ColCount;ii++)
{
fprintf(ptabela,"%f\t",StringGrid2->Cells[ii][i].ToDouble());
}
fprintf(ptabela,"\n");
}
fclose(ptabela);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Comunicao1Click(TObject *Sender)
{

Form4->Visible = true;
pconst=fopen("config_ser.dat","r");      //Abre o arquivo para escrita
fscanf(pconst,"%d\n%d",&velocidade,&bits_parada);
Form4->ComboBox1->Text = AnsiString(velocidade);
Form4->ComboBox3->Text = AnsiString(bits_parada);
fclose(pconst);

}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button4Click(TObject *Sender)
{
for(int m=0;m<100;m++)
{
          cdata_H = AnsiString("u").c_str();
              WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
}
}
//---------------------------------------------------------------------------
```

Unit2.cpp

```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit2.h"
#include <math.h>
#include "matrix.h"
#pragma package(smart_init)

#ifndef _NO_NAMESPACE
using namespace std;
using namespace math;
#define STD std
#else
#define STD
#endif

#ifndef _NO_TEMPLATE
typedef matrix<double> Matrix;
#else
typedef matrix Matrix;
#endif

extern HANDLE hComm;
unsigned char InBuff[200000];
unsigned int tab_dados[10];
unsigned int tab_dados2[10];
int k=1;
```

```
int nk=0;
float num_1;
int num_2;
int L=1;
int gL = 0;
int status_tempo = 0;
int tempo_1;
int tempo_2;
int valor;
char aa;
char bb;
int status=0;

float Iin_m;
float Vin_ma;
float Vg_ma;
float Iin_ma;
float Rot_ma;
float T_ma;

float I_max;

float teta_0;
float teta_1;
float teta_2;
float teta_3;
float teta_4;

Matrix Y_k0(2,1);
Matrix er(2,1);
float Y_k1[2][1];

Matrix phi_k0(5,2);
float phi_k1[2][5];

Matrix teta_k0(5,1);
float teta_k1[5][1] = {0, 0, 0, 0, 0};

Matrix K_k0(5,2);
float K_k1[5][5];

Matrix P_k1(5,5);

Matrix mq_1(2,2);
Matrix mq_13(3,3);
Matrix den_K(1,1);

Matrix A_sm(21,3);                      // b = A.X
Matrix X_sm(3,1);                      //X = b/A --> X = [X2 X1 X0]
Matrix B_sm(21,1);

DWORD dwBytesRead;
//---------------------------------------------------------------------------
__fastcall TRead::TRead(bool CreateSuspended)
       : TThread(CreateSuspended)
{
}

//---------------------------------------------------------------------------
void __fastcall TRead::DisplayIt()
{
  nk = 1;

  for(long int h=12;h<120000;h++)
  {

  if((InBuff[h-12] == 'A') && (InBuff[h-11] == 'A'))
  {
  Form1->StringGrid2->RowCount = nk+1;
```

```
  if((int(InBuff[h-10])<=4)&&(int(InBuff[h-8])<=4)&&(int(InBuff[h-
6])<=4)&&(int(InBuff[h-4])<=4)&&(int(InBuff[h-2])<=4))
  {
  Form1->StringGrid2->Cells[2][nk] = (int(InBuff[h-10])*256+int(InBuff[h-
9]))*5.7609*0.10472;
  Form1->Series2->AddXY(nk*0.000166,Form1->StringGrid2->Cells[2][nk].ToDouble());
  Form1->StringGrid2->Cells[3][nk] = (int(InBuff[h-8])*256+int(InBuff[h-
7]))*0.03572;
  Form1->Series4->AddXY(nk*0.000166,Form1->StringGrid2->Cells[3][nk].ToDouble());
  Form1->StringGrid2->Cells[4][nk] = (int(InBuff[h-6])*256+int(InBuff[h-
5]))*0.00174-0.1;;         //0.001509-0.01;
  Form1->Series5->AddXY(nk*0.000166,Form1->StringGrid2->Cells[4][nk].ToDouble());
  Form1->StringGrid2->Cells[0][nk] = (int(InBuff[h-4])*256+int(InBuff[h-
3]))*0.03598;
  Form1->Series1->AddXY(nk*0.000166,Form1->StringGrid2->Cells[0][nk].ToDouble());


  Iin_m = int(InBuff[h-2])*256+int(InBuff[h-1]);

  if(Iin_m>=64)
  {
        Iin_m = Iin_m*0.02827+0.19073;
  }
  else
  {
        Iin_m = 0.00429 + 0.05537*Iin_m - 0.00127*pow(Iin_m,2) +
0.000024026*pow(Iin_m,3) - 0.00000015536*pow(Iin_m,4);
  }

  Form1->StringGrid2->Cells[1][nk] = Iin_m;

  Form1->Series6->AddXY(nk*0.000166,Form1->StringGrid2->Cells[1][nk].ToDouble());

  nk++;
  }
  }
  }

  Form1->StringGrid1->RowCount = Form1->StringGrid2->RowCount;

                           mq_13(0,0) = 1.0;
                           mq_13(0,1) = 0.0;
                           mq_13(0,2) = 0.0;
                           mq_13(1,0) = 0.0;
                           mq_13(1,1) = 1.0;
                           mq_13(1,2) = 0.0;
                           mq_13(2,0) = 0.0;
                           mq_13(2,1) = 0.0;
                           mq_13(2,2) = 1.0;

  for(int m3a=1;m3a<Form1->StringGrid1->RowCount-2;m3a++)
             {
                    if(m3a<=21)
                    {
                           Vin_ma = 0;
                           for(int m4a=0;m4a<m3a;m4a++)
                           {
                                  Vin_ma = Vin_ma + Form1->StringGrid2-
>Cells[0][m3a-m4a].ToDouble();
                           }
                           Form1->StringGrid1->Cells[0][m3a] =
AnsiString(Vin_ma/m3a);

                           Iin_ma = 0;
                           for(int m4a=0;m4a<m3a;m4a++)
                           {
                                  Iin_ma = Iin_ma + Form1->StringGrid2-
>Cells[1][m3a-m4a].ToDouble();
```

```
                        }
                        Form1->StringGrid1->Cells[1][m3a] =
AnsiString(Iin_ma/m3a);

                        Rot_ma = 0;
                        for(int m4a=0;m4a<m3a;m4a++)
                        {
                               Rot_ma = Rot_ma + Form1->StringGrid2-
>Cells[2][m3a-m4a].ToDouble();
                        }
                        Form1->StringGrid1->Cells[2][m3a] =
AnsiString(Rot_ma/m3a);

                        Vg_ma = 0;
                        for(int m4a=0;m4a<m3a;m4a++)
                        {
                               Vg_ma = Vg_ma + Form1->StringGrid2->Cells[3][m3a-
m4a].ToDouble();
                        }
                        Form1->StringGrid1->Cells[3][m3a] =
AnsiString(Vg_ma/m3a);

                        T_ma = 0;
                        for(int m4a=0;m4a<m3a;m4a++)
                        {
                               T_ma = T_ma + Form1->StringGrid2->Cells[4][m3a-
m4a].ToDouble();
                        }
                        Form1->StringGrid1->Cells[4][m3a] = AnsiString(T_ma/m3a);
                }
                else
                {
                        for(int m7a=0;m7a<=4;m7a++)
                        {
                        for(int m4a=0;m4a<=20;m4a++)
                        {
                               A_sm(m4a,0) = pow((m3a-m4a),2);
                               A_sm(m4a,1) = m3a-m4a;
                               A_sm(m4a,2) = 1;

                               B_sm(m4a,0) = Form1->StringGrid2->Cells[m7a][m3a-
m4a].ToDouble();
                        }

                        X_sm = (mq_13/(~A_sm * A_sm))* ~A_sm * B_sm;

                        Vin_ma = pow((m3a-10),2)*X_sm(0,0) + (m3a-10)*X_sm(1,0) +
X_sm(2,0);

                        Form1->StringGrid1->Cells[m7a][m3a-10] =
AnsiString(Vin_ma);
                        }



                }
                if(m3a>=22)
                {
                Form1->Series3->Add(Form1->StringGrid1->Cells[0][m3a-
13].ToDouble());
                Form1->Series7->Add(Form1->StringGrid1->Cells[1][m3a-
13].ToDouble());
                Form1->Series8->Add(Form1->StringGrid1->Cells[2][m3a-
13].ToDouble());
                Form1->Series9->Add(Form1->StringGrid1->Cells[3][m3a-
13].ToDouble());
                Form1->Series10->Add(Form1->StringGrid1->Cells[4][m3a-
13].ToDouble());
```

```
                    if((Form1->StringGrid1->Cells[1][m3a-13].ToDouble()>=I_max))
                    {
                           I_max = Form1->StringGrid1->Cells[1][m3a-13].ToDouble();
                           tempo_1 = m3a-13;
                           status_tempo = 1;
                    }

                    if((Form1->StringGrid1->Cells[1][m3a-13].ToDouble()<Form1-
>StringGrid1->Cells[1][m3a-12].ToDouble())&&(status_tempo==1))
                    {
                           status_tempo = 2;
                    }

                    if((Form1->StringGrid1->Cells[1][m3a-13].ToDouble()>Form1-
>StringGrid1->Cells[1][m3a-12].ToDouble())&&(status_tempo==2))
                    {
                           tempo_2 = m3a-13;
                           tempo_2 = tempo_2 + (tempo_2 - tempo_1)*2;
                           status_tempo = 3;
                    }

                    Form1->Label2->Caption = AnsiString(tempo_1);
                    Form1->Label3->Caption = AnsiString(tempo_2);

                    }
                    }

                    for (int t1=0; t1 <= 4; t1++)
                    {
                           for(int t2=0; t2 <=4; t2++)
                           {
                                  if(t1==t2)
                                  {
                                         P_k1(t1,t2)=10000000.0;
                                  }
                                  else
                                  {
                                         P_k1(t1,t2)=0.0;
                                  }
                                  //mq_1(t1,t2)=1;
                           }
                    }

                    for (int t3=0; t3 <= 1; t3++)
                    {
                           for(int t4=0; t4 <=1; t4++)
                           {
                                  if(t3==t4)
                                  {
                                         mq_1(t3,t4)=1.0;
                                  }
                                  else
                                  {
                                         mq_1(t3,t4)=0.0;
                                  }
                           }
                    }

                    for(int m6a=0;m6a<=4;m6a++)
                    {
                           teta_k0(m6a,0) = 0;
                    }

                    for(int m5a=tempo_1;m5a<tempo_2;m5a++)
                    {
                    phi_k0(0,0) = - Form1->StringGrid1->Cells[1][m5a].ToDouble();
                    phi_k0(1,0) = Form1->StringGrid1->Cells[1][m5a-1].ToDouble();
                    phi_k0(2,0) = 0;
                    phi_k0(3,0) = 0;
```

```
                    phi_k0(4,0) = Form1->StringGrid1->Cells[0][m5a].ToDouble() -
0.025 * Form1->StringGrid1->Cells[1][m5a].ToDouble();
                    phi_k0(0,1) = 0;
                    phi_k0(1,1) = 0;
                    phi_k0(2,1) = Form1->StringGrid1->Cells[2][m5a].ToDouble();
                    phi_k0(3,1) = -Form1->StringGrid1->Cells[2][m5a-1].ToDouble();
                    phi_k0(4,1) = 0;

                    Y_k0(0,0) = Form1->StringGrid1->Cells[2][m5a].ToDouble();
                    Y_k0(1,0) = Form1->StringGrid1->Cells[1][m5a].ToDouble();

                    K_k0 = (P_k1 * phi_k0)/((mq_1 + ~phi_k0 * P_k1 * phi_k0));
                    er = Y_k0 - (~phi_k0 * teta_k0);
                    teta_k0 = teta_k0 + (K_k0 * er);
                    P_k1 = P_k1 - (K_k0 * ~phi_k0 * P_k1);

                    teta_0 = teta_k0(0,0);
                    teta_1 = teta_k0(1,0);
                    teta_2 = teta_k0(2,0);
                    teta_3 = teta_k0(3,0);
                    teta_4 = teta_k0(4,0);
                    }

                    Form1->Label4->Caption = "K = " + AnsiString(1/teta_4);
                    Form1->Label2->Caption = "b = " + AnsiString((((teta_2-
teta_3)/teta_4) - 0.0001224));
                    Form1->Label3->Caption = "R = " + AnsiString((teta_0-
teta_1)/teta_4);
                    Form1->Label5->Caption = "L = " +
AnsiString((teta_1*0.000166)/teta_4);
                    Form1->Label7->Caption = "J = " +
AnsiString((((teta_3*0.000166)/teta_4) - 0.00001158));
             }

//---------------------------------------------------------------------------
void __fastcall TRead::Execute()
{
  FreeOnTerminate = true;

  while(1)
  {
        ReadFile(hComm, InBuff, 120000, &dwBytesRead, NULL);

        if(dwBytesRead)
        {
          InBuff[dwBytesRead] = 0; // NULL TERMINATE THE STRING
          Synchronize(&DisplayIt);
        }
   }
}
```

Unit4.cpp

```
#include <vcl.h>
#include <stdio.h>
#pragma hdrstop

#include "Unit4.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm4 *Form4;
FILE *pconst2;
//---------------------------------------------------------------------------
__fastcall TForm4::TForm4(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm4::Button1Click(TObject *Sender)
{
pconst2=fopen("config_ser.dat","w");      //Abre o arquivo para escrita
fprintf(pconst2,"%d\n%d",ComboBox1->Text.ToInt(),ComboBox3->Text.ToInt());
fclose(pconst2);
Visible = false;
}
//---------------------------------------------------------------------------

void __fastcall TForm4::Button2Click(TObject *Sender)
{
Visible = false;
}
```

APÊNDICE E – CÓDIGO FONTE DO SOFTWARE DE CÁLCULO DE RENDIMENTO

Unit1.cpp

```
#include <vcl.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <dos.h>
#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
#include <Excel_XP.h>
// GLOBAL VARIABLES

HANDLE hComm = NULL;
TRead *ReadThread;
COMMTIMEOUTS ctmoNew = {0}, ctmoOld;

AnsiString Cont_cel;
AnsiString CellRef;

char Abre_col = 'A';
char V_out;

int Abre_linha = 1;

int con=0;
int coluna_1, coluna_2;
int linha_1;
int k;
int kk;
int jj;
int n_pulsos;
AnsiString ASData;
AnsiString ASvelocidade;
AnsiString ASbits_parada;
AnsiString salvar;

int velocidade;
int bits_parada;

float Vin_m;
float Vg_m;
float Iin_m;
float Rot_m;
float T_m;

float n;
float T;
```

```
float Va;
float Ia;
float rend;
float torque_zero;

unsigned long int BytesWritten;
unsigned char *cdata_H;
unsigned char *cdata_L;
unsigned int Length;

FILE *ptabela;
FILE *pconst;

unsigned char InBuff_2[200000];
int nk_2=0;
DWORD dwBytesRead_2;
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "Excel_XP_srvr"
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
       : TForm(Owner)
{
hComm = NULL;

          int ii=1;
          AnsiString Porta;
do
{
               Porta=AnsiString("COM")+AnsiString(ii);
               hComm = CreateFile(Porta.c_str(),
                                  GENERIC_READ | GENERIC_WRITE,
                                  0,
                                  0,
                                  OPEN_EXISTING,
                                  FILE_ATTRIBUTE_NORMAL,
                                  0);



                if(hComm != INVALID_HANDLE_VALUE)
                {
              ComboBox1->Items->Add(Porta);
              CloseHandle(hComm);
                     }

              ii++;
              } while(ii<=9);

hComm = NULL;

StringGrid2->Cells[0][0] = AnsiString("V in");
StringGrid2->Cells[1][0] = AnsiString("I in");
StringGrid2->Cells[2][0] = AnsiString("RPM");
StringGrid2->Cells[3][0] = AnsiString("V ger");
StringGrid2->Cells[4][0] = AnsiString("T");
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{

  CloseHandle(hComm);

}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{

              if(ComboBox2->Text == "0")
              {
                     V_out = '0';
              }

              if(ComboBox2->Text == "3")
              {
                     V_out = '1';
              }

              if(ComboBox2->Text == "6")
              {
                     V_out = '2';
              }

              if(ComboBox2->Text == "9")
              {
                     V_out = '3';
              }

              if(ComboBox2->Text == "12")
              {
                     V_out = '4';
              }

              if(ComboBox2->Text == "15")
              {
                     V_out = '5';
              }

              if(ComboBox2->Text == "18")
              {
                     V_out = '6';
              }

              if(ComboBox2->Text == "21")
              {
                     V_out = '7';
              }

              if(ComboBox2->Text == "23.5")
              {
                     V_out = '8';
              }

              cdata_H = AnsiString(V_out).c_str();
              WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
              Sleep (100);

              if(ComboBox2->Text == "0")
              {
                     for(int m1=0;m1<500;m1++)
                     {
                        cdata_H = AnsiString("r").c_str();
                        WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
                     }
              }


  Sleep(100);

              for(int m1=0;m1<100;m1++)
{
          cdata_H = AnsiString("u").c_str();
              WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
}
```

```
 Sleep(300);

//---LEITURA DA SERIAL-------------------------------------------------------

        ReadFile(hComm, InBuff_2, 1200, &dwBytesRead_2, NULL);

  nk_2 = 1;

  for(long int h=12;h<=1200;h++)
  {
  if((InBuff_2[h-12] == 'A') && (InBuff_2[h-11] == 'A'))
  {
  StringGrid2->RowCount = nk_2+1;

  if((int(InBuff_2[h-10])<=4)&&(int(InBuff_2[h-8])<=4)&&(int(InBuff_2[h-
6])<=4)&&(int(InBuff_2[h-4])<=4)&&(int(InBuff_2[h-2])<=4))
  {
  StringGrid2->Cells[2][nk_2] = (int(InBuff_2[h-10])*256+int(InBuff_2[h-
9]))*6.0489*0.10472;
  StringGrid2->Cells[3][nk_2] = (int(InBuff_2[h-8])*256+int(InBuff_2[h-
7]))*0.03572;
  StringGrid2->Cells[4][nk_2] = (int(InBuff_2[h-6])*256+int(InBuff_2[h-
5]))*0.00174-0.1;//-torque_zero; //0.00174-0.005;              //0.001509-0.01;
  StringGrid2->Cells[0][nk_2] = (int(InBuff_2[h-4])*256+int(InBuff_2[h-
3]))*0.03598;
  StringGrid2->Cells[1][nk_2] = (int(InBuff_2[h-2])*256+int(InBuff_2[h-
1]))*0.02811;    //0.02811

  if(((int(InBuff_2[h-6])*256+int(InBuff_2[h-5]))*0.00174-0.1)<0.0)
  {
        StringGrid2->Cells[4][nk_2] = 0;
  }
  nk_2++;
  }
  }
  }
                          n = 0;
                          for(int m4a=0;m4a<100;m4a++)
                          {
                                n = n + StringGrid2->Cells[0][m4a+1].ToDouble();
                          }
                          StringGrid1->Cells[0][1] = AnsiString(n/100);
                          Label12->Caption = "Va[V] = " + AnsiString(n/100);

                          n = 0;
                          for(int m4a=0;m4a<100;m4a++)
                          {
                                n = n + StringGrid2->Cells[1][m4a+1].ToDouble();
                          }
                          StringGrid1->Cells[1][1] = AnsiString(n/100);
                          Label13->Caption = "Ia[A] = " + AnsiString(n/100);

                          n = 0;
                          for(int m4a=0;m4a<100;m4a++)
                          {
                                n = n + StringGrid2->Cells[2][m4a+1].ToDouble();
                          }
                          StringGrid1->Cells[2][1] = AnsiString(n/100);
                          Label10->Caption = "n[rad/s] = " + AnsiString(n/100);

                          n = 0;
                          for(int m4a=0;m4a<100;m4a++)
                          {
                                n = n + StringGrid2->Cells[3][m4a+1].ToDouble();
                          }
                          StringGrid1->Cells[3][1] = AnsiString(n/100);

                          n = 0;
```

```
                            for(int m4a=0;m4a<100;m4a++)
                            {
                                  n = n + StringGrid2->Cells[4][m4a+1].ToDouble();
                            }
                            StringGrid1->Cells[4][1] = AnsiString(n/100);
                            Label11->Caption = "T[N.m] = " + AnsiString(n/100);


                            Label2->Caption = "Pmec = " + AnsiString(StringGrid1-
>Cells[2][1].ToDouble()*StringGrid1->Cells[4][1].ToDouble());
                            Label3->Caption = "Pe = " + AnsiString(StringGrid1-
>Cells[0][1].ToDouble()*StringGrid1->Cells[1][1].ToDouble());

                            rend = ((StringGrid1->Cells[2][1].ToDouble()*StringGrid1-
>Cells[4][1].ToDouble())+0.0000000001)/((StringGrid1-
>Cells[0][1].ToDouble()*StringGrid1->Cells[1][1].ToDouble())+0.000001);
                            Label16->Caption = AnsiString(rend*100) + " %";

}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
SaveDialog1->Execute();
salvar = SaveDialog1->FileName+".xls";

ptabela = fopen(salvar.c_str(),"w");

for(long int i=1;i<StringGrid2->RowCount;i++)
{
for(long int ii=0;ii<StringGrid2->ColCount;ii++)
{
fprintf(ptabela,"%f\t",StringGrid2->Cells[ii][i].ToDouble());
}
fprintf(ptabela,"\n");
}
fclose(ptabela);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Comunicao1Click(TObject *Sender)
{

Form4->Visible = true;
pconst=fopen("config_ser.dat","r");      //Abre o arquivo para escrita
fscanf(pconst,"%d\n%d",&velocidade,&bits_parada);
Form4->ComboBox1->Text = AnsiString(velocidade);
Form4->ComboBox3->Text = AnsiString(bits_parada);
fclose(pconst);

}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button4Click(TObject *Sender)
{
for(int m=0;m<100;m++)
{
          cdata_H = AnsiString("u").c_str();
              WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
}
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ConeeeeeeClick(TObject *Sender)
{

 DCB dcbCommPort;
```

```
  unsigned int RxBufferSize = 300000;
  unsigned int TxBufferSize = 300000;

  SetupComm(hComm, RxBufferSize, TxBufferSize);

  GetCommTimeouts(hComm,&ctmoOld);
  ctmoNew.ReadTotalTimeoutConstant = 0;
  ctmoNew.ReadTotalTimeoutMultiplier = 1;
  ctmoNew.WriteTotalTimeoutMultiplier = 100;
  ctmoNew.WriteTotalTimeoutConstant = 10;
  SetCommTimeouts(hComm, &ctmoNew);

pconst=fopen("config_ser.dat","r");      //Abre o arquivo para escrita
fscanf(pconst,"%d\n%d",&velocidade,&bits_parada);
fclose(pconst);

  dcbCommPort.DCBlength = sizeof(DCB);
  GetCommState(hComm, &dcbCommPort);
  ASvelocidade = AnsiString(velocidade)+",N,8,"+AnsiString(bits_parada);

  BuildCommDCB(ASvelocidade.c_str(), &dcbCommPort);
  SetCommState(hComm, &dcbCommPort);

  //ReadThread = new TRead(false);

  Sleep(200);
  hComm = CreateFile(ComboBox1->Text.c_str(),
                                    GENERIC_READ | GENERIC_WRITE,
                                    0,
                                    0,
                                    OPEN_EXISTING,
                                    0,
                                    0);
  if(hComm == INVALID_HANDLE_VALUE) Application->Terminate();

  Sleep(300);

              for(int m1=0;m1<200;m1++)
{
          cdata_H = AnsiString("L").c_str();
              WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
}

//-----------------Calibração da célula de carga-------------------------------
Sleep(100);

              for(int m1=0;m1<100;m1++)
{
          cdata_H = AnsiString("u").c_str();
              WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
}

 Sleep(300);

//---LEITURA DA SERIAL-----------------------------------------------------------

        ReadFile(hComm, InBuff_2, 1200, &dwBytesRead_2, NULL);

  nk_2 = 1;

  for(long int h=12;h<=1200;h++)
  {
  if((InBuff_2[h-12] == 'A') && (InBuff_2[h-11] == 'A'))
  {
  StringGrid2->RowCount = nk_2+1;

  if((int(InBuff_2[h-10])<=4)&&(int(InBuff_2[h-8])<=4)&&(int(InBuff_2[h-
6])<=4)&&(int(InBuff_2[h-4])<=4)&&(int(InBuff_2[h-2])<=4))
  {
```

```
  StringGrid2->Cells[2][nk_2] = (int(InBuff_2[h-10])*256+int(InBuff_2[h-
9]))*6.0489*0.10472;
  StringGrid2->Cells[3][nk_2] = (int(InBuff_2[h-8])*256+int(InBuff_2[h-
7]))*0.03572;
  StringGrid2->Cells[4][nk_2] = (int(InBuff_2[h-6])*256+int(InBuff_2[h-
5]))*0.00174;              //0.001509-0.01;
  StringGrid2->Cells[0][nk_2] = (int(InBuff_2[h-4])*256+int(InBuff_2[h-
3]))*0.03598;
  StringGrid2->Cells[1][nk_2] = (int(InBuff_2[h-2])*256+int(InBuff_2[h-
1]))*0.02811;    //0.02811

  nk_2++;
  }
  }
  }

                           n = 0;
                           for(int m4a=0;m4a<100;m4a++)
                           {
                                  n = n + StringGrid2->Cells[4][m4a+1].ToDouble();
                           }
                           torque_zero = n/100;



}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject *Sender)
{

Sleep(100);

              for(int m1=0;m1<500;m1++)
{
          cdata_H = AnsiString("r").c_str();
              WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
}

Sleep(300);

              for(int m1=0;m1<100;m1++)
{
          cdata_H = AnsiString("u").c_str();
              WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
}

 Sleep(300);

//---LEITURA DA SERIAL-------------------------------------------------------

        ReadFile(hComm, InBuff_2, 1200, &dwBytesRead_2, NULL);

  nk_2 = 1;

  for(long int h=12;h<=1200;h++)
  {
  if((InBuff_2[h-12] == 'A') && (InBuff_2[h-11] == 'A'))
  {
  StringGrid2->RowCount = nk_2+1;

  if((int(InBuff_2[h-10])<=4)&&(int(InBuff_2[h-8])<=4)&&(int(InBuff_2[h-
6])<=4)&&(int(InBuff_2[h-4])<=4)&&(int(InBuff_2[h-2])<=4))
  {
  StringGrid2->Cells[2][nk_2] = (int(InBuff_2[h-10])*256+int(InBuff_2[h-
9]))*6.0489*0.10472;
  StringGrid2->Cells[3][nk_2] = (int(InBuff_2[h-8])*256+int(InBuff_2[h-
7]))*0.03572;
```

```
  StringGrid2->Cells[4][nk_2] = (int(InBuff_2[h-6])*256+int(InBuff_2[h-
5]))*0.00174;              //0.001509-0.01;
  StringGrid2->Cells[0][nk_2] = (int(InBuff_2[h-4])*256+int(InBuff_2[h-
3]))*0.03598;
  StringGrid2->Cells[1][nk_2] = (int(InBuff_2[h-2])*256+int(InBuff_2[h-
1]))*0.02811;    //0.02811

  nk_2++;
  }
  }
  }
                          n = 0;
                          for(int m4a=0;m4a<100;m4a++)
                          {
                                 n = n + StringGrid2->Cells[4][m4a+1].ToDouble();
                          }
                          torque_zero = n/100;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button6Click(TObject *Sender)
{
         cdata_H = AnsiString("L").c_str();
         WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button7Click(TObject *Sender)
{
         cdata_H = AnsiString("l").c_str();
         WriteFile(hComm, cdata_H, 1,&BytesWritten, NULL);
}
//---------------------------------------------------------------------------
```

Nota: O arquivo “Unit4.cpp” é idêntico para os dois softwares.

ANEXO A – BIBLIOTECA "MATRIX.H"

O código transcrito a seguir foi produzido pela empresa Techsoft e está disponível em www.techsoftpl.com.

```
////////////////////////////////
// Matrix TCL Lite v1.13
// Copyright (c) 1997-2002 Techsoft Pvt. Ltd. (See License.Txt file.)
//
// Matrix.h: Matrix C++ template class include file
// Web: http://www.techsoftpl.com/matrix/
// Email: matrix@techsoftpl.com
//

////////////////////////////
// Installation:
//
// Copy this "matrix.h" file into include directory of your compiler.
//

////////////////////////////
// Note: This matrix template class defines majority of the matrix
// operations as overloaded operators or methods. It is assumed that
// users of this class is familiar with matrix algebra. We have not
// defined any specialization of this template here, so all the instances
// of matrix will be created implicitly by the compiler. The data types
// tested with this class are float, double, long double, complex<float>,
// complex<double> and complex<long double>. Note that this class is not
// optimized for performance.
//
// Since implementation of exception, namespace and template are still
// not standardized among the various (mainly old) compilers, you may
// encounter compilation error with some compilers. In that case remove
// any of the above three features by defining the following macros:
//
//  _NO_NAMESPACE:  Define this macro to remove namespace support.
//
//  _NO_EXCEPTION:  Define this macro to remove exception handling
//                  and use old style of error handling using function.
//
//  _NO_TEMPLATE:   If this macro is defined matrix class of double
//                  type will be generated by default. You can also
//                  generate a different type of matrix like float.
//
//  _SGI_BROKEN_STL: For SGI C++ v.7.2.1 compiler.
//
//  Since all the definitions are also included in this header file as
//  inline function, some compiler may give warning "inline function
//  can't be expanded". You may ignore/disable this warning using compiler
//  switches. All the operators/methods defined in this class have their
//  natural meaning except the followings:
//
//  Operator/Method                         Description
//  ---------------                         -----------
//   operator ()    :   This function operator can be used as a
```

```
//                      two-dimensional subscript operator to get/set
//                      individual matrix elements.
//
//   operator !    :    This operator has been used to calculate inversion
//                      of matrix.
//
//   operator ~    :    This operator has been used to return transpose of
//                      a matrix.
//
//   operator ^    :    It is used calculate power (by a scalar) of a matrix.
//                      When using this operator in a matrix equation, care
//                      must be taken by parenthesizing it because it has
//                      lower precedence than addition, subtraction,
//                      multiplication and division operators.
//
//   operator >>   :    It is used to read matrix from input stream as per
//                      standard C++ stream operators.
//
//   operator <<   :    It is used to write matrix to output stream as per
//                      standard C++ stream operators.
//
// Note that professional version of this package, Matrix TCL Pro 2.11
// is optimized for performance and supports many more matrix operations.
// It is available from our web site at <http://www.techsoftpl.com/matrix/>.
//

#ifndef __cplusplus
#error Must use C++ for the type matrix.
#endif

#if !defined(__STD_MATRIX_H)
#define __STD_MATRIX_H

//////////////////////////////
// First deal with various shortcomings and incompatibilities of
// various (mainly old) versions of popular compilers available.
//

#if defined(__BORLANDC__)
#pragma option -w-inl -w-pch
#endif

#if ( defined(__BORLANDC__) || _MSC_VER <= 1000 ) && !defined( __GNUG__ )
#  include <stdio.h>
#  include <stdlib.h>
#  include <math.h>
#  include <iostream.h>
#  include <string.h>
#else
#  include <cmath>
#  include <cstdio>
#  include <cstdlib>
#  include <string>
#  include <iostream>
#endif

#if defined(_MSC_VER) && _MSC_VER <= 1000
#  define _NO_EXCEPTION        // stdexception is not fully supported in MSVC++ 4.0
typedef int bool;
#  if !defined(false)
#    define false  0
#  endif
#  if !defined(true)
#    define true   1
#  endif
#endif

#if defined(__BORLANDC__) && !defined(__WIN32__)
#  define _NO_EXCEPTION        // std exception and namespace are not fully
```

```
#  define _NO_NAMESPACE        // supported in 16-bit compiler
#endif

#if defined(_MSC_VER) && !defined(_WIN32)
#  define _NO_EXCEPTION
#endif

#if defined(_NO_EXCEPTION)
#  define _NO_THROW
#  define _THROW_MATRIX_ERROR
#else
#  if defined(_MSC_VER)
#    if _MSC_VER >= 1020
#      include <stdexcept>
#    else
#      include <stdexcpt.h>
#    endif
#  elif defined(__MWERKS__)
#      include <stdexcept>
#  elif (__GNUC__ >= 2 || (__GNUC__ == 2 && __GNUC_MINOR__ >= 8))
#     include <stdexcept>
#  else
#     include <stdexcept>
#  endif
#  define _NO_THROW               throw ()
#  define _THROW_MATRIX_ERROR     throw (matrix_error)
#endif

#ifndef __MINMAX_DEFINED
#  define max(a,b)    (((a) > (b)) ? (a) : (b))
#  define min(a,b)    (((a) < (b)) ? (a) : (b))
#endif

#if defined(_MSC_VER)
#undef _MSC_EXTENSIONS      // To include overloaded abs function definitions!
#endif

#if ( defined(__BORLANDC__) || _MSC_VER ) && !defined( __GNUG__ )
inline float abs (float v) { return (float)fabs( v); }
inline double abs (double v) { return fabs( v); }
inline long double abs (long double v) { return fabsl( v); }
#endif

#if defined(__GNUG__) || defined(__MWERKS__) || (defined(__BORLANDC__) &&
(__BORLANDC__ >= 0x540))
#define FRIEND_FUN_TEMPLATE <>
#else
#define FRIEND_FUN_TEMPLATE
#endif

#if defined(_MSC_VER) && _MSC_VER <= 1020   // MSVC++ 4.0/4.2 does not
#  define _NO_NAMESPACE                     // support "std" namespace
#endif

#if !defined(_NO_NAMESPACE)
#if defined( _SGI_BROKEN_STL )              // For SGI C++ v.7.2.1 compiler
namespace std { }
#endif
using namespace std;
#endif

#ifndef _NO_NAMESPACE
namespace math {
#endif

#if !defined(_NO_EXCEPTION)
class matrix_error : public logic_error
{
    public:
```

```
      matrix_error (const string& what_arg) : logic_error( what_arg) {}
};
#define REPORT_ERROR(ErrormMsg)  throw matrix_error( ErrormMsg);
#else
inline void _matrix_error (const char* pErrMsg)
{
    cout << pErrMsg << endl;
    exit(1);
}
#define REPORT_ERROR(ErrormMsg)  _matrix_error( ErrormMsg);
#endif

#if !defined(_NO_TEMPLATE)
#  define MAT_TEMPLATE  template <class T>
#  define matrixT  matrix<T>
#else
#  define MAT_TEMPLATE
#  define matrixT  matrix
#  ifdef MATRIX_TYPE
     typedef MATRIX_TYPE T;
#  else
     typedef double T;
#  endif
#endif


MAT_TEMPLATE
class matrix
{
public:
   // Constructors
   matrix (const matrixT& m);
   matrix (size_t row = 6, size_t col = 6);

   // Destructor
   ~matrix ();

   // Assignment operators
   matrixT& operator = (const matrixT& m) _NO_THROW;

   // Value extraction method
   size_t RowNo () const { return _m->Row; }
   size_t ColNo () const { return _m->Col; }

   // Subscript operator
   T& operator () (size_t row, size_t col) _THROW_MATRIX_ERROR;
   T  operator () (size_t row, size_t col) const _THROW_MATRIX_ERROR;

   // Unary operators
   matrixT operator + () _NO_THROW { return *this; }
   matrixT operator - () _NO_THROW;

   // Combined assignment - calculation operators
   matrixT& operator += (const matrixT& m) _THROW_MATRIX_ERROR;
   matrixT& operator -= (const matrixT& m) _THROW_MATRIX_ERROR;
   matrixT& operator *= (const matrixT& m) _THROW_MATRIX_ERROR;
   matrixT& operator *= (const T& c) _NO_THROW;
   matrixT& operator /= (const T& c) _NO_THROW;
   matrixT& operator ^= (const size_t& pow) _THROW_MATRIX_ERROR;

   // Miscellaneous -methods
   void Null (const size_t& row, const size_t& col) _NO_THROW;
   void Null () _NO_THROW;
   void Unit (const size_t& row) _NO_THROW;
   void Unit () _NO_THROW;
   void SetSize (size_t row, size_t col) _NO_THROW;

   // Utility methods
   matrixT Solve (const matrixT& v) const _THROW_MATRIX_ERROR;
```

```
   matrixT Adj () _THROW_MATRIX_ERROR;
   matrixT Inv () _THROW_MATRIX_ERROR;
   T Det () const _THROW_MATRIX_ERROR;
   T Norm () _NO_THROW;
   T Cofact (size_t row, size_t col) _THROW_MATRIX_ERROR;
   T Cond () _NO_THROW;

   // Type of matrices
   bool IsSquare () _NO_THROW { return (_m->Row == _m->Col); }
   bool IsSingular () _NO_THROW;
   bool IsDiagonal () _NO_THROW;
   bool IsScalar () _NO_THROW;
   bool IsUnit () _NO_THROW;
   bool IsNull () _NO_THROW;
   bool IsSymmetric () _NO_THROW;
   bool IsSkewSymmetric () _NO_THROW;
   bool IsUpperTriangular () _NO_THROW;
   bool IsLowerTriangular () _NO_THROW;

private:
    struct base_mat
    {
       T **Val;
       size_t Row, Col, RowSiz, ColSiz;
       int Refcnt;

       base_mat (size_t row, size_t col, T** v)
       {
           Row = row; RowSiz = row;
           Col = col; ColSiz = col;
           Refcnt = 1;

           Val = new T* [row];
           size_t rowlen = col * sizeof(T);

           for (size_t i=0; i < row; i++)
           {
              Val[i] = new T [col];
              if (v) memcpy( Val[i], v[i], rowlen);
           }
       }
       ~base_mat ()
       {
           for (size_t i=0; i < RowSiz; i++)
              delete [] Val[i];
           delete [] Val;
       }
    };
    base_mat *_m;

    void clone ();
    void realloc (size_t row, size_t col);
    int pivot (size_t row);
};

#if defined(_MSC_VER) && _MSC_VER <= 1020
#  undef  _NO_THROW               // MSVC++ 4.0/4.2 does not support
#  undef  _THROW_MATRIX_ERROR     // exception specification in definition
#  define _NO_THROW
#  define _THROW_MATRIX_ERROR
#endif

// constructor
MAT_TEMPLATE inline
matrixT::matrix (size_t row, size_t col)
{
  _m = new base_mat( row, col, 0);
}
```

```
// copy constructor
MAT_TEMPLATE inline
matrixT::matrix (const matrixT& m)
{
    _m = m._m;
    _m->Refcnt++;
}

// Internal copy constructor
MAT_TEMPLATE inline void
matrixT::clone ()
{
    _m->Refcnt--;
    _m = new base_mat( _m->Row, _m->Col, _m->Val);
}

// destructor
MAT_TEMPLATE inline
matrixT::~matrix ()
{
   if (--_m->Refcnt == 0) delete _m;
}

// assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator = (const matrixT& m) _NO_THROW
{
    m._m->Refcnt++;
    if (--_m->Refcnt == 0) delete _m;
    _m = m._m;
    return *this;
}

//  reallocation method
MAT_TEMPLATE inline void
matrixT::realloc (size_t row, size_t col)
{
   if (row == _m->RowSiz && col == _m->ColSiz)
   {
      _m->Row = _m->RowSiz;
      _m->Col = _m->ColSiz;
      return;
   }

   base_mat *m1 = new base_mat( row, col, NULL);
   size_t colSize = min(_m->Col,col) * sizeof(T);
   size_t minRow = min(_m->Row,row);

   for (size_t i=0; i < minRow; i++)
      memcpy( m1->Val[i], _m->Val[i], colSize);

   if (--_m->Refcnt == 0)
       delete _m;
   _m = m1;

   return;
}

// public method for resizing matrix
MAT_TEMPLATE inline void
matrixT::SetSize (size_t row, size_t col) _NO_THROW
{
   size_t i,j;
   size_t oldRow = _m->Row;
   size_t oldCol = _m->Col;

   if (row != _m->RowSiz || col != _m->ColSiz)
      realloc( row, col);
```

```
   for (i=oldRow; i < row; i++)
      for (j=0; j < col; j++)
        _m->Val[i][j] = T(0);

   for (i=0; i < row; i++)
      for (j=oldCol; j < col; j++)
        _m->Val[i][j] = T(0);

   return;
}

// subscript operator to get/set individual elements
MAT_TEMPLATE inline T&
matrixT::operator () (size_t row, size_t col) _THROW_MATRIX_ERROR
{
   if (row >= _m->Row || col >= _m->Col)
      REPORT_ERROR( "matrixT::operator(): Index out of range!");
   if (_m->Refcnt > 1) clone();
   return _m->Val[row][col];
}

// subscript operator to get/set individual elements
MAT_TEMPLATE inline T
matrixT::operator () (size_t row, size_t col) const _THROW_MATRIX_ERROR
{
   if (row >= _m->Row || col >= _m->Col)
      REPORT_ERROR( "matrixT::operator(): Index out of range!");
   return _m->Val[row][col];
}

// input stream function
MAT_TEMPLATE inline istream&
operator >> (istream& istrm, matrixT& m)
{
   for (size_t i=0; i < m.RowNo(); i++)
      for (size_t j=0; j < m.ColNo(); j++)
      {
         T x;
         istrm >> x;
         m(i,j) = x;
      }
   return istrm;
}

// output stream function
MAT_TEMPLATE inline ostream&
operator << (ostream& ostrm, const matrixT& m)
{
   for (size_t i=0; i < m.RowNo(); i++)
   {
      for (size_t j=0; j < m.ColNo(); j++)
      {
         T x = m(i,j);
         ostrm << x << '\t';
      }
      ostrm << endl;
   }
   return ostrm;
}


// logical equal-to operator
MAT_TEMPLATE inline bool
operator == (const matrixT& m1, const matrixT& m2) _NO_THROW
{
   if (m1.RowNo() != m2.RowNo() || m1.ColNo() != m2.ColNo())
      return false;

   for (size_t i=0; i < m1.RowNo(); i++)
```

```
      for (size_t j=0; j < m1.ColNo(); j++)
             if (m1(i,j) != m2(i,j))
                return false;

   return true;
}

// logical no-equal-to operator
MAT_TEMPLATE inline bool
operator != (const matrixT& m1, const matrixT& m2) _NO_THROW
{
    return (m1 == m2) ? false : true;
}

// combined addition and assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator += (const matrixT& m) _THROW_MATRIX_ERROR
{
   if (_m->Row != m._m->Row || _m->Col != m._m->Col)
      REPORT_ERROR( "matrixT::operator+= : Inconsistent matrix sizes in
addition!");
   if (_m->Refcnt > 1) clone();
   for (size_t i=0; i < m._m->Row; i++)
      for (size_t j=0; j < m._m->Col; j++)
        _m->Val[i][j] += m._m->Val[i][j];
   return *this;
}

// combined subtraction and assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator -= (const matrixT& m) _THROW_MATRIX_ERROR
{
   if (_m->Row != m._m->Row || _m->Col != m._m->Col)
      REPORT_ERROR( "matrixT::operator-= : Inconsistent matrix sizes in
subtraction!");
   if (_m->Refcnt > 1) clone();
   for (size_t i=0; i < m._m->Row; i++)
      for (size_t j=0; j < m._m->Col; j++)
        _m->Val[i][j] -= m._m->Val[i][j];
   return *this;
}

// combined scalar multiplication and assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator *= (const T& c) _NO_THROW
{
    if (_m->Refcnt > 1) clone();
    for (size_t i=0; i < _m->Row; i++)
       for (size_t j=0; j < _m->Col; j++)
           _m->Val[i][j] *= c;
    return *this;
}

// combined matrix multiplication and assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator *= (const matrixT& m) _THROW_MATRIX_ERROR
{
   if (_m->Col != m._m->Row)
      REPORT_ERROR( "matrixT::operator*= : Inconsistent matrix sizes in
multiplication!");

   matrixT temp(_m->Row,m._m->Col);

   for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < m._m->Col; j++)
      {
         temp._m->Val[i][j] = T(0);
         for (size_t k=0; k < _m->Col; k++)
            temp._m->Val[i][j] += _m->Val[i][k] * m._m->Val[k][j];
```

```
      }
   *this = temp;

   return *this;
}

// combined scalar division and assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator /= (const T& c) _NO_THROW
{
    if (_m->Refcnt > 1) clone();
    for (size_t i=0; i < _m->Row; i++)
       for (size_t j=0; j < _m->Col; j++)
           _m->Val[i][j] /= c;

    return *this;
}

// combined power and assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator ^= (const size_t& pow) _THROW_MATRIX_ERROR
{
       matrixT temp(*this);

       for (size_t i=2; i <= pow; i++)
      *this = *this * temp;

       return *this;
}

// unary negation operator
MAT_TEMPLATE inline matrixT
matrixT::operator - () _NO_THROW
{
   matrixT temp(_m->Row,_m->Col);

   for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
        temp._m->Val[i][j] = - _m->Val[i][j];

   return temp;
}

// binary addition operator
MAT_TEMPLATE inline matrixT
operator + (const matrixT& m1, const matrixT& m2) _THROW_MATRIX_ERROR
{
   matrixT temp = m1;
   temp += m2;
   return temp;
}

// binary subtraction operator
MAT_TEMPLATE inline matrixT
operator - (const matrixT& m1, const matrixT& m2) _THROW_MATRIX_ERROR
{
   matrixT temp = m1;
   temp -= m2;
   return temp;
}

// binary scalar multiplication operator
MAT_TEMPLATE inline matrixT
operator * (const matrixT& m, const T& no) _NO_THROW
{
   matrixT temp = m;
   temp *= no;
   return temp;
}
```

```
// binary scalar multiplication operator
MAT_TEMPLATE inline matrixT
operator * (const T& no, const matrixT& m) _NO_THROW
{
   return (m * no);
}

// binary matrix multiplication operator
MAT_TEMPLATE inline matrixT
operator * (const matrixT& m1, const matrixT& m2) _THROW_MATRIX_ERROR
{
   matrixT temp = m1;
   temp *= m2;
   return temp;
}

// binary scalar division operator
MAT_TEMPLATE inline matrixT
operator / (const matrixT& m, const T& no) _NO_THROW
{
    return (m * (T(1) / no));
}


// binary scalar division operator
MAT_TEMPLATE inline matrixT
operator / (const T& no, const matrixT& m) _THROW_MATRIX_ERROR
{
    return (!m * no);
}

// binary matrix division operator
MAT_TEMPLATE inline matrixT
operator / (const matrixT& m1, const matrixT& m2) _THROW_MATRIX_ERROR
{
    return (m1 * !m2);
}

// binary power operator
MAT_TEMPLATE inline matrixT
operator ^ (const matrixT& m, const size_t& pow) _THROW_MATRIX_ERROR
{
   matrixT temp = m;
   temp ^= pow;
   return temp;
}

// unary transpose operator
MAT_TEMPLATE inline matrixT
operator ~ (const matrixT& m) _NO_THROW
{
   matrixT temp(m.ColNo(),m.RowNo());

   for (size_t i=0; i < m.RowNo(); i++)
      for (size_t j=0; j < m.ColNo(); j++)
      {
         T x = m(i,j);
             temp(j,i) = x;
      }
   return temp;
}

// unary inversion operator
MAT_TEMPLATE inline matrixT
operator ! (const matrixT m) _THROW_MATRIX_ERROR
{
   matrixT temp = m;
```

```
   return temp.Inv();
}

// inversion function
MAT_TEMPLATE inline matrixT
matrixT::Inv () _THROW_MATRIX_ERROR
{
   size_t i,j,k;
   T a1,a2,*rowptr;

   if (_m->Row != _m->Col)
      REPORT_ERROR( "matrixT::operator!: Inversion of a non-square matrix");

   matrixT temp(_m->Row,_m->Col);
   if (_m->Refcnt > 1) clone();


   temp.Unit();
   for (k=0; k < _m->Row; k++)
   {
      int indx = pivot(k);
      if (indx == -1)
             REPORT_ERROR( "matrixT::operator!: Inversion of a singular matrix");

      if (indx != 0)
      {
             rowptr = temp._m->Val[k];
             temp._m->Val[k] = temp._m->Val[indx];
             temp._m->Val[indx] = rowptr;
      }
      a1 = _m->Val[k][k];
      for (j=0; j < _m->Row; j++)
      {
             _m->Val[k][j] /= a1;
             temp._m->Val[k][j] /= a1;
      }
      for (i=0; i < _m->Row; i++)
          if (i != k)
          {
             a2 = _m->Val[i][k];
             for (j=0; j < _m->Row; j++)
             {
                _m->Val[i][j] -= a2 * _m->Val[k][j];
                temp._m->Val[i][j] -= a2 * temp._m->Val[k][j];
             }
          }
   }
   return temp;
}

// solve simultaneous equation
MAT_TEMPLATE inline matrixT
matrixT::Solve (const matrixT& v) const _THROW_MATRIX_ERROR
{
   size_t i,j,k;
   T a1;

   if (!(_m->Row == _m->Col && _m->Col == v._m->Row))
      REPORT_ERROR( "matrixT::Solve():Inconsistent matrices!");

   matrixT temp(_m->Row,_m->Col+v._m->Col);
   for (i=0; i < _m->Row; i++)
   {
      for (j=0; j < _m->Col; j++)
        temp._m->Val[i][j] = _m->Val[i][j];
      for (k=0; k < v._m->Col; k++)
        temp._m->Val[i][_m->Col+k] = v._m->Val[i][k];
   }
   for (k=0; k < _m->Row; k++)
```

```
   {
      int indx = temp.pivot(k);
      if (indx == -1)
        REPORT_ERROR( "matrixT::Solve(): Singular matrix!");

      a1 = temp._m->Val[k][k];
      for (j=k; j < temp._m->Col; j++)
        temp._m->Val[k][j] /= a1;

      for (i=k+1; i < _m->Row; i++)
      {
        a1 = temp._m->Val[i][k];
        for (j=k; j < temp._m->Col; j++)
          temp._m->Val[i][j] -= a1 * temp._m->Val[k][j];
      }
   }
   matrixT s(v._m->Row,v._m->Col);
   for (k=0; k < v._m->Col; k++)
      for (int m=int(_m->Row)-1; m >= 0; m--)
      {
        s._m->Val[m][k] = temp._m->Val[m][_m->Col+k];
        for (j=m+1; j < _m->Col; j++)
           s._m->Val[m][k] -= temp._m->Val[m][j] * s._m->Val[j][k];
      }
   return s;
}

// set zero to all elements of this matrix
MAT_TEMPLATE inline void
matrixT::Null (const size_t& row, const size_t& col) _NO_THROW
{
    if (row != _m->Row || col != _m->Col)
      realloc( row,col);

    if (_m->Refcnt > 1)
      clone();

    for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
          _m->Val[i][j] = T(0);
    return;
}

// set zero to all elements of this matrix
MAT_TEMPLATE inline void
matrixT::Null() _NO_THROW
{
    if (_m->Refcnt > 1) clone();
    for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
             _m->Val[i][j] = T(0);
    return;
}

// set this matrix to unity
MAT_TEMPLATE inline void
matrixT::Unit (const size_t& row) _NO_THROW
{
    if (row != _m->Row || row != _m->Col)
      realloc( row, row);

    if (_m->Refcnt > 1)
      clone();

    for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
          _m->Val[i][j] = i == j ? T(1) : T(0);
    return;
}
```

```
// set this matrix to unity
MAT_TEMPLATE inline void
matrixT::Unit () _NO_THROW
{
    if (_m->Refcnt > 1) clone();
    size_t row = min(_m->Row,_m->Col);
    _m->Row = _m->Col = row;

    for (size_t i=0; i < _m->Row; i++)
       for (size_t j=0; j < _m->Col; j++)
           _m->Val[i][j] = i == j ? T(1) : T(0);
    return;
}

// private partial pivoting method
MAT_TEMPLATE inline int
matrixT::pivot (size_t row)
{
  int k = int(row);
  double amax,temp;

  amax = -1;
  for (size_t i=row; i < _m->Row; i++)
    if ( (temp = abs( _m->Val[i][row])) > amax && temp != 0.0)
     {
       amax = temp;
       k = i;
     }
  if (_m->Val[k][row] == T(0))
     return -1;
  if (k != int(row))
  {
     T* rowptr = _m->Val[k];
     _m->Val[k] = _m->Val[row];
     _m->Val[row] = rowptr;
     return k;
  }
  return 0;
}

// calculate the determinant of a matrix
MAT_TEMPLATE inline T
matrixT::Det () const _THROW_MATRIX_ERROR
{
   size_t i,j,k;
   T piv,detVal = T(1);

   if (_m->Row != _m->Col)
      REPORT_ERROR( "matrixT::Det(): Determinant a non-square matrix!");

   matrixT temp(*this);
   if (temp._m->Refcnt > 1) temp.clone();

   for (k=0; k < _m->Row; k++)
   {
      int indx = temp.pivot(k);
      if (indx == -1)
        return 0;
      if (indx != 0)
        detVal = - detVal;
      detVal = detVal * temp._m->Val[k][k];
      for (i=k+1; i < _m->Row; i++)
      {
        piv = temp._m->Val[i][k] / temp._m->Val[k][k];
        for (j=k+1; j < _m->Row; j++)
           temp._m->Val[i][j] -= piv * temp._m->Val[k][j];
      }
   }
```

```
   return detVal;
}

// calculate the norm of a matrix
MAT_TEMPLATE inline T
matrixT::Norm () _NO_THROW
{
   T retVal = T(0);

   for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
        retVal += _m->Val[i][j] * _m->Val[i][j];
   retVal = sqrt( retVal);

   return retVal;
}

// calculate the condition number of a matrix
MAT_TEMPLATE inline T
matrixT::Cond () _NO_THROW
{
   matrixT inv = ! (*this);
   return (Norm() * inv.Norm());
}

// calculate the cofactor of a matrix for a given element
MAT_TEMPLATE inline T
matrixT::Cofact (size_t row, size_t col) _THROW_MATRIX_ERROR
{
   size_t i,i1,j,j1;

   if (_m->Row != _m->Col)
      REPORT_ERROR( "matrixT::Cofact(): Cofactor of a non-square matrix!");

   if (row > _m->Row || col > _m->Col)
      REPORT_ERROR( "matrixT::Cofact(): Index out of range!");

   matrixT temp (_m->Row-1,_m->Col-1);

   for (i=i1=0; i < _m->Row; i++)
   {
      if (i == row)
       continue;
      for (j=j1=0; j < _m->Col; j++)
      {
        if (j == col)
           continue;
        temp._m->Val[i1][j1] = _m->Val[i][j];
        j1++;
      }
      i1++;
   }
   T  cof = temp.Det();
   if ((row+col)%2 == 1)
      cof = -cof;

   return cof;
}


// calculate adjoin of a matrix
MAT_TEMPLATE inline matrixT
matrixT::Adj () _THROW_MATRIX_ERROR
{
   if (_m->Row != _m->Col)
      REPORT_ERROR( "matrixT::Adj(): Adjoin of a non-square matrix.");

   matrixT temp(_m->Row,_m->Col);
```

```
   for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
         temp._m->Val[j][i] = Cofact(i,j);
   return temp;
}

// Determine if the matrix is singular
MAT_TEMPLATE inline bool
matrixT::IsSingular () _NO_THROW
{
   if (_m->Row != _m->Col)
      return false;
   return (Det() == T(0));
}

// Determine if the matrix is diagonal
MAT_TEMPLATE inline bool
matrixT::IsDiagonal () _NO_THROW
{
   if (_m->Row != _m->Col)
      return false;
   for (size_t i=0; i < _m->Row; i++)
     for (size_t j=0; j < _m->Col; j++)
       if (i != j && _m->Val[i][j] != T(0))
         return false;
   return true;
}

// Determine if the matrix is scalar
MAT_TEMPLATE inline bool
matrixT::IsScalar () _NO_THROW
{
   if (!IsDiagonal())
     return false;
   T v = _m->Val[0][0];
   for (size_t i=1; i < _m->Row; i++)
     if (_m->Val[i][i] != v)
       return false;
   return true;
}

// Determine if the matrix is a unit matrix
MAT_TEMPLATE inline bool
matrixT::IsUnit () _NO_THROW
{
   if (IsScalar() && _m->Val[0][0] == T(1))
     return true;
   return false;
}

// Determine if this is a null matrix
MAT_TEMPLATE inline bool
matrixT::IsNull () _NO_THROW
{
   for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
        if (_m->Val[i][j] != T(0))
           return false;
   return true;
}

// Determine if the matrix is symmetric
MAT_TEMPLATE inline bool
matrixT::IsSymmetric () _NO_THROW
{
   if (_m->Row != _m->Col)
      return false;
   for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
```

```
        if (_m->Val[i][j] != _m->Val[j][i])
           return false;
   return true;
}

// Determine if the matrix is skew-symmetric
MAT_TEMPLATE inline bool
matrixT::IsSkewSymmetric () _NO_THROW
{
   if (_m->Row != _m->Col)
      return false;
   for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
        if (_m->Val[i][j] != -_m->Val[j][i])
           return false;
   return true;
}

// Determine if the matrix is upper triangular
MAT_TEMPLATE inline bool
matrixT::IsUpperTriangular () _NO_THROW
{
   if (_m->Row != _m->Col)
      return false;
   for (size_t i=1; i < _m->Row; i++)
      for (size_t j=0; j < i-1; j++)
        if (_m->Val[i][j] != T(0))
           return false;
   return true;
}

// Determine if the matrix is lower triangular
MAT_TEMPLATE inline bool
matrixT::IsLowerTriangular () _NO_THROW
{
   if (_m->Row != _m->Col)
      return false;

   for (size_t j=1; j < _m->Col; j++)
      for (size_t i=0; i < j-1; i++)
        if (_m->Val[i][j] != T(0))
           return false;

   return true;
}

#ifndef _NO_NAMESPACE
}
#endif

#endif //__STD_MATRIX_H
```