**UNIVERSIDADE FEDERAL DO CEARÁ**

**CENTRO DE TECNOLOGIA**

**DEPARTAMENTO DE ENGENHARIA ELÉTRICA**

**CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA**

**JOSÉ RAIMUNDO DE OLIVEIRA JÚNIOR**

# SISTEMA DIDÁTICO DE SINTONIA AUTOMÁTICA E AUTO-AJUSTE DE CONTROLADOR PID EM CONTROLE DE VELOCIDADE E DE POSIÇÃO DE MOTOR CC

**FORTALEZA**

**2020**

JOSÉ RAIMUNDO DE OLIVEIRA JÚNIOR

# SISTEMA DIDÁTICO DE SINTONIA AUTOMÁTICA E AUTO-AJUSTE DE CONTROLADOR PID EM CONTROLE DE VELOCIDADE E DE POSIÇÃO DE MOTOR CC

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Orientador: Prof. Dr.-Ing. Sergio Daher

Coorientador: Me. René D. O. Pereira

FORTALEZA

2020

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

---

O47s Oliveira Júnior, José Raimundo.
Sistema didático de sintonia automática e auto-ajuste de controlador PID em controle de velocidade e de posição de motor CC / José Raimundo Oliveira Júnior. – 2020.
99 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2020.
Orientação: Prof. Dr. Sergio Daher.
Coorientação: Prof. Me. René Descartes Olimpio Pereira.

1. Sintonia automática. 2. Auto-ajuste. 3. Controle PID. 4. Motor CC. 5. Controle de motor CC. I. Título.

CDD 621.3

---

JOSÉ RAIMUNDO DE OLIVEIRA JÚNIOR

# SISTEMA DIDÁTICO DE SINTONIA AUTOMÁTICA E AUTO-AJUSTE DE CONTROLADOR PID EM CONTROLE DE VELOCIDADE E DE POSIÇÃO DE MOTOR CC

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Aprovada em: xx de xx de 20xx

BANCA EXAMINADORA

---

Prof. Dr.-Ing. Sergio Daher (Orientador)
Universidade Federal do Ceará (UFC)

---

Me. René D. O. Pereira (Coorientador)
Universidade Federal do Ceará (UFC)

---

Prof. Dr. Bismark C. Torrico
Universidade Federal do Ceará (UFC)

## AGRADECIMENTOS

Por sua importância, direciono meu primeiro agradecimento aos meus pais, José Raimundo e Odete. Agradeço por todo o esforço em relação à minha educação formal, assim como para os ensinamentos em relação ao mundo e a realidade das coisas. Em especial à minha mãe pelo exemplo de resiliência.

À Mariana pela paciência e apoio, enquanto eu desenvolvia esse trabalho e em todos os momentos compartilhados. Pela compreensão em relação a mim, e novamente pela paciência em meus debates e discussões infinitas.

À minha família e amigos, pelos momentos agradáveis e pela compreensão em minha ausência.

Aos amigos do GPAR pelo apoio, conversas e momentos. Em especial ao meu amigo René, pela ajuda nesse trabalho e aos incentivo sempre que precisei.

Agradeço a banca examinadora pela disponibilidade e, com isso, permitir que eu prossiga tranquilamente com meu trabalho.

## RESUMO

Este trabalho apresenta um sistema didático de sintonia automática e de auto-ajuste de controladores Proporcional-Integral-Derivativo (PID) no controle de velocidade e de posição de motor de Corrente Contínua (CC). O motor CC é modelado através de princípios físicos e são obtidos modelos simplificados relacionando a tensão de entrada com a sua velocidade ou com a sua posição do eixo. O sistema didático é baseado em uma placa de circuito que inclui um dsPIC30F4011 e periféricos, e conta com um programa de interface gráfica para sistema operacional Microsoft Windows 10. Técnicas de controle adaptativo de sintonia automática e de auto-ajuste são definidas e aplicadas ao sistema didático. O resultado é uma ferramenta eficaz que realiza a identificação dos modelos e sintonia dos controladores a partir de respostas em malha aberta e em malha fechada. Todo o processo é concentrado na interface gráfica, unificando as informações, o comando do sistema e a visualização dos resultados. A simplicidade da interface gráfica permite aos discentes conferir e validar os resultados obtidos. Isto torna o sistema didático uma ferramenta eficaz no ensino e na consolidação de conceitos básicos da teoria de controle, como identificação de sistemas e sintonia de controladores.



**Palavras-chave:** Sintonia automática. Auto-ajuste. Controle PID. Motor CC. Controle de motor CC

## ABSTRACT

This work presents a didactic system of auto-tuning and self-tuning of PID controllers in the control of speed and position of direct current (DC) motor. The DC motor is modeled through physical principles and simplified models are obtained by relating the input voltage to its speed or to its axis position. The didactic system is based on a circuit board that includes a dsPIC30F4011 and peripherals, and has a graphic interface program for Microsoft Windows 10 operating system. Adaptive control techniques of automatic tuning and self-adjusting are defined and applied to the didactic system. The result is an effective tool that performs model identification and tuning of controllers from open loop and closed loop responses. The whole process is concentrated in the graphic interface, unifying the information, the system command and the visualization of the results. The simplicity of the graphic interface allows the students to check and validate the results obtained. This makes the didactic system an effective tool in teaching and consolidating basic concepts of control theory, such as system identification and controller tuning.



**Keywords:** Auto-tuning. Self-tuning. PID control. DC motor. DC motor control

## LISTA DE FIGURAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| A/D | Analógico/Digital |
| CC | Corrente Contínua |
| CMOS | *Complementary Metal Oxide Semiconductor* |
| CPU | *Central Processing Unit* |
| D/A | Digital/Analógico |
| DSP | *Digital Signal Processing* |
| EEPROM | *Programmable Read-Only Memory* |
| FOPDT | *First-Order Plus Dead Time* |
| GUIDE | *Graphical User Interface Development Environment* |
| IAE | *Integrated Absolute Error* |
| IFOPDT | *Integrating First-Order Plus Dead Time* |
| LCD | *Liquid Crystal Display* |
| MAB | *Memory Adress Bus* |
| MDB | *Memory Data Bus* |
| PI | Proporcional-Integral |
| PID | Proporcional-Integral-Derivativo |
| PWM | *Pulse Width Modulation* |
| QEI | *Quadrature Encoder Interface* |
| RAM | *Random Access Memory* |
| RISC | *Reduced Instruction Set Computer* |
| SBPA | Sequencia Binária Pseudo-Aleatória |
| SISO | *Single Input Single Output* |
| TITO | *Two Input Two Output* |
| UART | *Universal Asynchronous Receiver Transmitter* |
| USB | *Universal Serial Bus* |
| USCI | *Universal Serial Communication Interface* |

## LISTA DE SÍMBOLOS

| | |
|---|---|
| $Ae$ | Área efetiva da antena |
| $\overline{oo'}$ | Eixo de giro do rotor |
| $L$ | Centroide do rotor |
| $N$ | Polo norte magnético do estator |
| $S$ | Polo Sul magnético do estator |
| $e_{xx}$ | Tensão induzida no respectivo trecho $_{xx}$ |
| $e_{tot}$ | Tensão Induzida total |
| $\omega_m$ | Velocidade angular mecânica do rotor |
| $r$ | Raio das faces cilíndricas do rotor |
| $\overline{abcd}$ | Perímetro completo da espira |
| $\overline{ab}$ | Fio da espira que liga os pontos $a$ e $b$ |
| $\overline{cd}$ | Fio da espira que liga os pontos $c$ e $d$ |
| $\overline{da}$ | Fio da espira que liga os pontos $d$ e $a$ |
| $\overline{bc}$ | Fio da espira que liga os pontos $b$ e $c$ |
| $i$ | Corrente no circuito de armadura |
| $S$ | Área superficial delimitada pelo fio da bobina $\overline{abcd}$ |
| $\mathbf{S}$ | Vetor de área de $S$ |
| $v$ | Velocidade linear do fio |
| $I$ | Comprimento do fio imerso no campo magnético |
| $\mathbf{I}$ | Vetor do comprimento do fio imerso no campo magnético |
| $l$ | Comprimento de um dos lado da bobina. |
| $\mathbf{L}_x$ | Vetor do trecho do comprimento de fio do lado “x” do vetor $\mathbf{I}$. |
| $\mathbf{B}$ | Vetor de densidade de fluxo magnético |
| $\phi$ | Fluxo magnético |
| $A_P$ | Areá superficial de um dos polos |
| $K_M$ | Constante característica da maquina CC |

| | |
|---|---|
| $K_e$ | Produto entre a constante característica da maquina CC e o fluxo $\phi$ |
| $n$ | Número de espiras da máquina |
| $E$ | Força cotra-eletromotriz |
| $V_b$ | Tensão da fonte de alimentação externa |
| $F_m$ | Força devido ao campo magnético |
| $Q$ | Carga elétrica |
| $\mathbf{J}$ | Vetor de densidade de corrente de convecção |
| $\Delta I$ | Variação de corrente |
| $\Delta S$ | Variação na seção de área superficial |
| $\rho$ | Densidade do fluxo de cargas |
| $d\upsilon$ | Derivada volumétrica |
| $d\upsilon$ | Volume infinitesimal |
| $dQ$ | Carga infinitesimal |
| $dl$ | Elemento condutor de comprimento infinitesimal |
| $F_{xx}$ | Força magnética no respectivo trecho $_{xx}$ |
| $e_{xx}$ | Tensão induzida no respectivo trecho $_{xx}$ |
| $R_a$ | Resistência de armadura |
| $L_a$ | Indutância de armadura |
| $E_{Mec}$ | Energia mecânica |
| $t$ | Tempo |
| $T$ | Torque |
| $\omega$ | Velocidade angular |
| $N_x$ | Número de dentes de engrenagem “x” |
| $V_{Lx}$ | Velocidade linear na extremidade da engrenagem “x” |
| $\pi$ | Constante matemática que representa a relação do perímetro com o diâmetro de um circulo |
| $\eta$ | Eficiência de transmissão de energia mecânica |
| $J_m$ | Momento de inércia do motor |

| | |
|---|---|
| $J_x$ | Momento de inercia do trecho “x” |
| $T_m$ | Conjugado mecânico ou torque gerado pelo motor |
| $T_x$ | Conjugado mecânico ou torque aplicado trecho “x” |
| $\omega_x$ | Velocidade angular no trecho “x” |
| $J_{eqx}$ | Momento de inércia total equivalente refletido ao trecho “x” |
| $D_{eq}$ | Constante de amortecimento total equivalente |
| $D_{eqx}$ | Constante de amortecimento total equivalente refletido ao trecho “x” |
| $FCY_{micro}$ | Frequência interna do relógio do microcontrolador |
| $PLL_{TMRx}$ | *Loop* de bloqueio de fase |
| $F_{inter}$ | Frequência de ocorrência de interrupção |
| $TX$ | Transmissor |
| $RX$ | Receptor |
| $BR$ | Taxa de transmissão |
| $K$ | Ganho estático |
| $T$ | Constante de tempo |
| $L$ | Atraso de transporte |
| $K_v$ | Ganho estático do modelo FOPDT |
| $T_v$ | Constante de tempo do modelo FOPDT |
| $L_v$ | Atraso de transporte do modelo FOPDT |
| $K_p$ | Ganho estático do modelo IFOPDT |
| $T_p$ | Constante de tempo do modelo IFOPDT |
| $L_p$ | Atraso de transporte do modelo IFOPDT |
| $u_v(t)$ | Degrau na entrada no ensaio de velocidade no tempo |
| $y_v(t)$ | Saída ou resposta no ensaio de velocidade |
| $y_v(\infty)$ | Valor final da saída no ensaio de velocidade |
| $A_{uv}$ | Amplitude do degrau aplicado no ensaio de velocidade |
| $T_{0v}$ | Soma da constante de tempo e do atraso de transporte |
| $u_p(t)$ | Pulso de entrada no ensaio de posição |

| | |
|---|---|
| $y_p(t)$ | Saída ou resposta no ensaio de posição |
| $y_p(\infty)$ | Valor final da saída no ensaio de posição |
| $t_p$ | Largura do pulso no tempo no ensaio de posição |
| $A_u$ | Amplitude do pulso no ensaio de posição |
| $T_{cv}$ | Constante de tempo de malha fechada no modelo FOPDT |
| $T_{cp}$ | Constante de tempo de malha fechada no modelo IFOPDT |
| $T_c$ | Constante de tempo de malha fechada |
| $E(s)$ | Função de transferência do sinal de erro |
| $Y(s)$ | Função de transferência da saída |
| $T_f$ | Constante de tempo do filtro derivativo |
| $r(t)$ | Referência para o controlador |
| $e(t)$ | Erro do sistema em malha fechada definido por $r(t)$ - $y(t)$ |
| $A_r$ | Amplitude do degrau aplicado na referência |
| $e_u(t)$ | Sinal característico |
| $NB$ | Banda de ruído |
| $u(t)$ | Sinal de controle |
| $e_u(t)$ | Integral do sinal de controle |
| $T_s$ | Tempo de amostragem |
| $\Delta t_u$ | Duração do pulso de entrada |
| $p$ | Quantidade de pulsos do *encoder* contados |
| $r_{tr}$ | Relação de transmissão do conjunto de engrenagens do motor |
| $r_{enc}$ | Resolução do *encoder* |
| $v_{rpm}$ | Velocidade do eixo em rpm. |

**SUMÁRIO**

# 1 INTRODUÇÃO

Os projetos modernos de engenharia possuem natureza multidisciplinar e, nesse contexto, o controle automático desempenha um papel fundamental (GUZMAN *et al.*, 2016). As aplicações em controle automático exigem um conhecimento matemático abrangente, incluindo equações diferenciais, álgebra linear, geometria diferencial, e/ou variável complexa, entre outros (BRAATZ, 2013). Consequentemente, em muitos casos, o controle automático é difícil de ser compreendido pela maioria dos alunos, especialmente nos casos em que eles têm apenas um curso de controle introdutório em seu currículo de engenharia (BISSELL, 1999). Assim, se faz necessário o desenvolvimento de alternativas para introduzir e tornar o controle automático atraente para os alunos, tal como o uso de esquemas, figuras e, principalmente, a implementação de projetos práticos com interfaces didáticas (BENCOMO, 2002).

Em várias aplicações industriais o processo controlado pode mudar no decorrer do tempo. Para sistemas com controladores de parâmetros fixos, isso pode levar à deterioração de desempenho ou até mesmo à instabilidade do sistema. Para lidar com este tipo de problema, técnicas de controle adaptativo têm sido estudadas desde os anos 1950. Nesse contexto, há uma crescente demanda para que os estudantes tenham contato com essas técnicas e práticas ainda em sua formação.

Dois tipos de áreas de controle adaptativo se destacam na indústria: o auto-tuning (sintonia automática, em português) e o self-tuning (auto-ajuste, em português) de controladores.

Técnicas de sintonia automática de controladores são importantes no comissionamento de novas plantas, onde o operador apenas efetua um comando para realizar a sintonia inicial do controlador. Elas ficaram populares nos anos 1970 com o uso de computadores digitais na indústria, quando, por exemplo, o método de sintonia de Ziegler-Nichols passou a ser realizado de forma automática.

O primeiro controlador auto-ajustável foi proposto em Kalman (1958). Mas somente nos anos 1970 esta técnica começaria a ter grandes avanços, com o trabalho de Åström e Wittenmark (1973).

O uso de técnicas de controle de velocidade e de posição de motores CC abrangem diversas aplicações. De acordo com Gieras e Wing (2002) destacam-se na robótica, automobilística, indústria de borracha, guindastes, máquinas ferramenta e entre outras.

A modelagem de motores CC é amplamente abordada na literatura, como descrito em UMANS (2014), Chapman (2012). O desenvolvimento desta modelagem é baseado nos

princípios físicos e obtém as equações diferenciais que regem o comportamento desses motores. Com a metodologia apresentada em Nise (2015), é possível usar estas equações diferenciais para obter modelos em funções de transferência. Além disso, é possível simplificar estes modelos para modelos de menor ordem. Isto resulta em modelos estáveis de primeira ordem com atraso de transporte (do inglês, *First-Order Plus Dead Time* (FOPDT)) e integrador de primeira ordem com atraso de transporte (do inglês, *Integrating First-Order Plus Dead Time* (IFOPDT)) para representar os processos de velocidade e de posição, respectivamente.

## 1.1 Objetivo Geral

Este trabalho tem como objetivo apresentar um sistema didático de sintonia automática e auto-ajuste de controladores PID no controle de velocidade e de posição de um motor CC. Esse desenvolvimento é realizado utilizando apenas instrumentos, equipamentos e conhecimentos comuns aos alunos de graduação, para assim permitir uma maior integração desse conhecimento e a familiaridade com a ferramenta.

## 1.2 Objetivos Específicos

Os objetivos específicos mostram os propósitos do trabalho, eles estão listados abaixo:

- Estudar o funcionamento e a modelagem dos motores de corrente contínua;
- Estudar a configuração e programação dos microcontroladores;
- Estudar a teoria de sinais, sistemas e controle, assim como, adquirir familiaridade com suas aplicações;
- Desenvolver um protótipo funcional para sintonia automática e auto-ajuste para motores de corrente contínua;
- Desenvolver uma interface gráfica para facilitar a operação do protótipo e melhorar a didática no ensino de teoria de controle.

## 1.3 Organização do trabalho

O trabalho está organizado da seguinte forma: O Capítulo 2 apresenta uma revisão do funcionamento e da modelagem dos motores de corrente contínua. O Capítulo 3 apresenta o microcontrolador utilizado e suas particularidades. O Capítulo 4 apresenta o método de sintonia

automática. O Capítulo 5 apresenta o método de auto-ajuste. O Capítulo 6 são apresentados o sistema didático e os resultados experimentais. Finalmente, no Capítulo 7 são apresentadas as conclusões.

# 2 MOTORES DE CORRENTE CONTÍNUA

Devido à natureza da sua construção e princípio de funcionamento, as máquinas de Corrente Contínua (CC) se destacam em sua versatilidade, permitindo em seu projeto uma grande liberdade para escolha dos parâmetros da máquina com relação a operações dinâmicas ou em regime permanente. Essa característica tornou, no passado, essa máquina a mais popular para aplicações que exigem operações em uma grande faixa de velocidade ou de controle preciso da saída do motor (UMANS, 2014). Porém, desde o final do século dezenove, os sistemas elétricos de potência em corrente alternada predominaram sobre os sistemas de potência em corrente contínua e, aliado a isso, a tecnologia baseada em eletrônica de potência avançou de forma a permitir que outras máquinas consigam atuar em aplicações que eram essencialmente exclusivas de máquinas CC (CHAPMAN, 2012). Devido aos custos elevados de manutenção e projeto das máquinas CC e ao fortalecimento da concorrência de outras máquinas a partir do uso das tecnologia de estado sólido, as máquinas CC atualmente são usadas em raras aplicações (HUGHES, 2006).

Em aplicações que exigem alto torque de partida, grande flexibilidade da saída de operação da máquina, em baixa potência e de uso intermitente as máquinas CC ainda são uma opção viável técnica e economicamente. Ao considerar importância histórica dessas máquinas, as aplicações atuais e a simplicidade de projeto e estudo destas, justifica-se a sua escolha para pequenos projetos e aplicações didáticas e, por consequência, para o presente trabalho.

As máquinas CC normalmente apresentam dois circuitos separados, a depender do tipo. O primeiro circuito está associado aos enrolamento de campo, os quais envolvem cada polo, gerando a excitação da máquina e, desse modo, um campo magnético que normalmente se mantém constante e em seu valor máximo. O outro circuito está relacionado ao rotor, e transporta a maior quantidade de corrente por estar relacionado com o trabalho ativo da máquina. Os principais tipos de máquina CC diferem entre si em apenas pequenas particularidades, principalmente com relação ao projeto do circuito de campo, uma classificação para as máquinas CC pode ser: excitação independente, excitação em paralelo, ímã permanente, excitação série e excitação composta. No primeiro caso, fontes de tensão independentes alimentam cada circuito. Para o segundo caso a mesma fonte alimenta os dois em configuração paralela. No terceiro caso a excitação é obtida a partir dos campos naturais de ímãs. No quarto caso os dois circuitos estão em série e alimentados por uma mesma fonte. Por fim, a ultima configuração apresenta dois enrolamentos de campo, com um alimentado em série com o enrolamento de armadura e outro

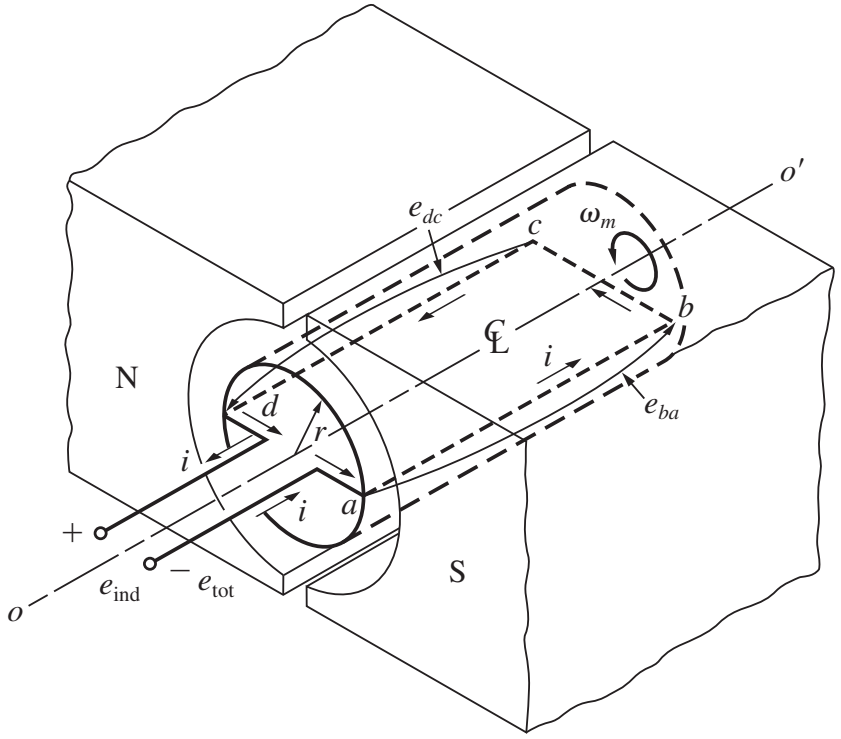
alimentado em paralelo ao circuito equivalente anterior, utilizando apenas uma fonte de tensão.

## 2.1 Força contra-eletromotriz

Esse trabalho utiliza um motor CC com excitação por ímã permanente, por se tratar de uma aplicação de baixa potência e devido a sua disponibilidade e praticidade. O desenvolvimento teórico será baseado nesse tipo de motor, porém existe um conjunto de elementos e princípios básicos compartilhados por todos eles, e assim, essa análise ainda generaliza grande parte dos princípios envolvidos nas máquinas CC.

De acordo com Chapman (2012), um dos exemplos simples de máquina rotativa CC consiste em uma única espira de fio girando em torno de um eixo fixo imerso em um campo magnético gerado por dois polos, que permite teorização suficiente para obter as equações de operação para o modelo clássico da máquina, que podem ser utilizadas para o controle, sem envolver as complexidades das máquinas reais. Na Figura 1 é possível classificar duas regiões com relação ao movimento. A primeira região permanece estática e é denominada como "estator" e a segunda região apresenta uma dinâmica rotativa e é denominada como "rotor".

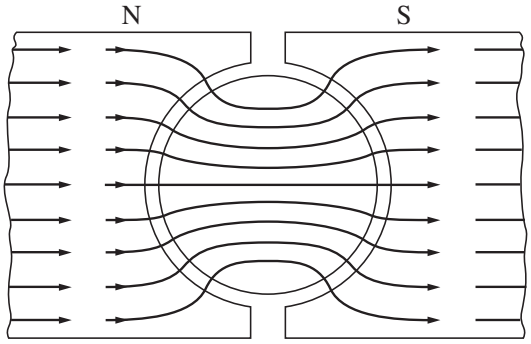
Figura 1 – Vista em perspectiva da máquina rotativa CC de espira única




Fonte: Adaptada de Chapman (2012).

Onde a linha $\overline{oo'}$ indica o eixo de simetria e o eixo de giro do rotor, o ponto *L*

Figura 2 – Vista da distribuição das linhas de campo em um corte reto transversal na máquina




Fonte: Adaptada de Chapman (2012).

indica o centroide do rotor. As indicações $N$ e $S$ são os polos magnéticos do estator norte e sul, respectivamente. Os $e_{xx}$ indicam a tensão induzida no respectivo trecho e o $e_{tot}$ é a soma dessas contribuições das tensões induzidas. Por fim, $\omega_m$ é a velocidade mecânica do rotor, $l$ é o comprimento dos trechos de fio paralelos ao eixo de rotação e $r$ é o raio das faces cilíndricas do rotor.

A espira de fio $\overline{abcd}$ exposta na Figura 1 está embutida em uma ranhura pertencente ao núcleo ferromagnético do rotor. A geometria deste núcleo é projetada de tal forma que busca manter um espaçamento uniforme entre a superfície dos polos do estator e a superfície do rotor, esse espaçamento é denominado entreferro. Como o entreferro é preenchido por ar e a relutância deste é dezenas de milhares de vezes superior ao dos materiais ferromagnéticos, e como o fluxo magnético atua de modo a minimizar a relutância do caminho percorrido, será prioritário realizar o menor caminho possível na zona do entreferro. E para esse caso, o menor caminho é sempre perpendicular à superfície do rotor e a superfície dos polos, como pode ser verificado na Figura 2. Essa característica impõe um fluxo magnético com direção essencialmente constante na região do entreferro descrita.

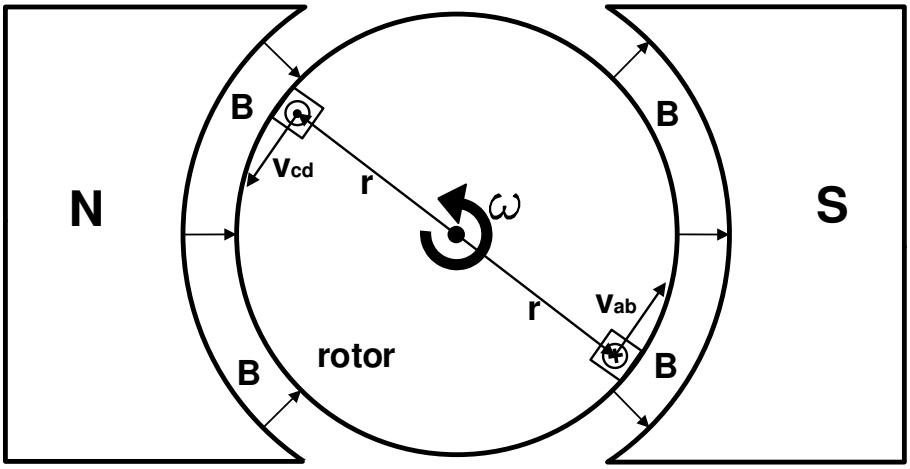
Inicialmente será analisada a tensão total induzida $e_{tot}$ nos terminais da bobina. Na Figura 1 é perceptível que a bobina é quadrada, composta pelos trechos $\overline{ab}$, $\overline{cd}$, $\overline{da}$ e $\overline{bc}$, respeitando a convenção do sentido da corrente $i$. Tem-se que a tensão induzida pode ser calculada a partir de (2.2), tratando diretamente dos princípios físicos (SADIKU, 2013). No caso particular onde tem-se uma espira em movimento (ranhuras do rotor) imersa em um campo magnético estático (entreferro) pode-se a partir da definição

$$e_{ind} = \frac{-d}{dt} \int_S \mathbf{B} \bullet d\mathbf{S}, \tag{2.1}$$

obter um caso particular simplificado

$$e_{ind} = \oint_L (\mathbf{v} \times \mathbf{B}) \bullet d\mathbf{L} = ((\mathbf{v} \times \mathbf{B}) \bullet \mathbf{I})_{\sum \mathbf{L}_x}, \tag{2.2}$$

Figura 3 – Diagrama da máquina com enfase nos vetores para o cálculo da tensão induzida



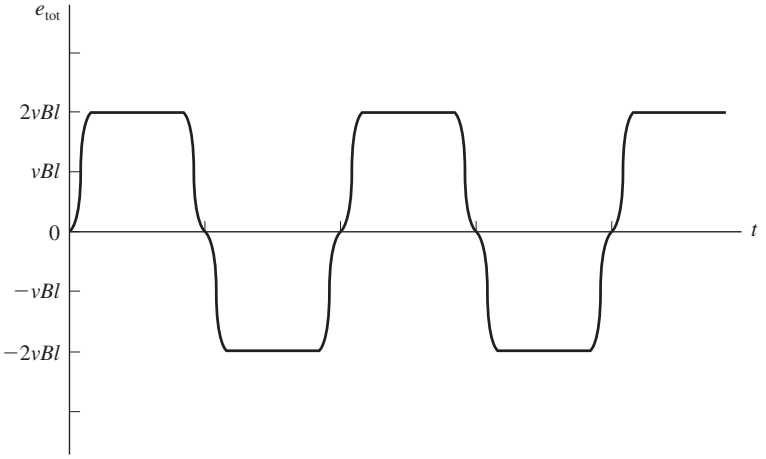

Fonte: Autoria própria.

onde, $\frac{-d}{dt}$ é a derivada temporal do resultado da integral, e o valor negativo é resultado da lei de Lenz. $e_{ind}$ é a tensão induzida, $\int_S \mathbf{B} \bullet d\mathbf{S}$ é uma integral de superfície, $S$ é a área superficial delimitada pelo fio da bobina e $\mathbf{S}$ é o vetor respectivo a essa área, que por definição, apresenta direção perpendicular ao plano da bobina. Tem-se que $\mathbf{B}$ é o vetor de densidade do fluxo magnético. O produto escalar $\mathbf{B} \bullet d\mathbf{S}$ quantifica o fluxo magnético efetivo que envolve cada superfície infinitesimal, $v$ é a velocidade linear do fio, $I$ é o vetor de tamanho do fio imerso no campo magnético e o $\mathbf{L}_x$ representa o trecho do lado "x" do vetor $\mathbf{I}$.

Aplicando (2.2) torna-se mais simples a obtenção do valor de tensão induzida, e como trata-se de uma integral de linha, pode-se dividir o cálculo por trechos obtendo as contribuições de tensão induzida. Por ser um quadrado, pode-se dividir entre os quatro lados com mesma direção. Assim o cálculo é simplificado para apenas o produto escalar entre o vetor de comprimento do respectivo lado e o resultado do produto vetorial entre a velocidade linear e o campo magnético, para cada trecho, os respectivos vetores estão expostos na Figura 3. Por fim, basta apenas somar cada contribuição para obter o total.

Para o trecho $\overline{ab}$, o produto vetorial tem como resultado um vetor na mesma direção que o vetor de comprimento $\mathbf{I}$, em concordância com a convenção corrente $i$ positivo, obtendo tensão positiva de módulo $e_{ab} = vBl$. No caso do trecho $\overline{cd}$, tem-se os mesmos módulos, mas o vetor de velocidade inverte o sentido assim como o de comprimento, obtendo o mesmo resultado de tensão que o caso anterior $e_{cd} = vBl$, esse comportamento pode ser verificado na Figura 3. Para os trechos $\overline{bc}$ e $\overline{da}$, os vetores resultantes do produto vetorial estão perpendiculares ao vetor de comprimento $\mathbf{I}$, tornando o produto escalar entre esse vetor resultante e o vetor $\mathbf{I}$ igual a zero, assim, $e_{bc} = e_{da} = 0$. A soma das contribuições apresenta o valor de $e_{ind} = vBl + 0 + vBl + 0 =$

Figura 4 – Tensão de saída nos terminar da espira



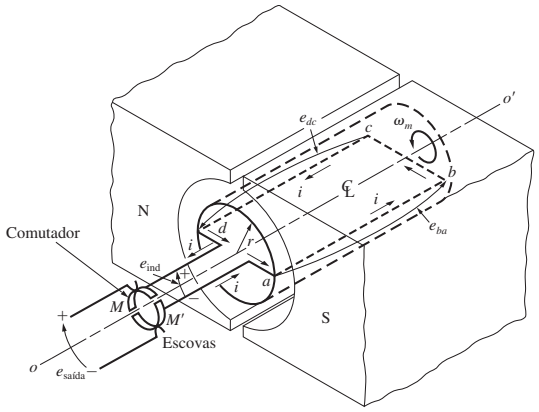

Fonte: (CHAPMAN, 2012).

$2vBl$. A convenção de corrente foi escolhida de forma que a tensão resultante fosse positiva e, desse modo, o fluxo magnético criado pela convenção de corrente é oposto à variação fluxo magnético gerado pelo movimento da bobina.

A análise vetorial realizada no paragrafo anterior pode ser repetida para diferentes posições da bobina, obtendo o perfil de tensão para toda uma revolução. No caso especifico das máquinas de corrente contínua, a direção do vetor resultante do produto vetorial $(\mathbf{v} \times \mathbf{B})$ tem sempre a mesma direção que a convenção de corrente **I** nos trechos paralelos ao eixo de rotação e valor zero nos demais trechos. Dessa forma, as mudanças possíveis no valor da tensão induzida ocorrem devido a variação do módulo dos vetores ou mudanças no sentido, ocasionadas mudança de posição da espira. Apenas o vetor de densidade de fluxo magnético **B** que não se mantém constante, seu módulo oscila no espaço de acordo com a distribuição das linhas de campo apresentadas na Figura 2 e apresenta mudança na direção do campo magnético ao trocar de polo. Por fim, o valor da tensão induzida de acordo com a posição da espira está exposta na Figura 4, nela é perceptível que a frequência elétrica é igual à frequência mecânica nesse caso de apenas dois polos magnéticos.

O módulo da tensão obtida é praticamente constante, porém apresenta valor alternado, e para esse tipo de máquina é interessante um valor continuo de tensão de saída. A tensão contínua é obtida nessas máquinas a partir de um comutador e escovas, que são capazes de formar um retificador mecânico, como na Figura 5. Assim, pode-se considerar, quando retificado, a tensão nos terminais como aproximadamente constante $e_{ind} = 2vBl$. Esse resultado pode ser reformulado em termos da velocidade angular $\omega_m$ e do fluxo magnético $\phi$ que cruza a superfície inferior dos polos $A_P$, pois para esse caso particular: $\omega_m = \frac{v}{r}$ e $\phi = \mathbf{A}_P \bullet \mathbf{B}$, onde $\mathbf{A}_P$ é a área de

Figura 5 – Vista em perspectiva da máquina CC com comutador e escovas




Fonte: (CHAPMAN, 2012).

superfície do polo (SADIKU, 2013).

Condensando os resultados anteriores obtém-se a tensão induzida para uma máquina CC de apenas uma espira como $e_{ind} = \frac{2lr}{A_P}\omega_m\phi$, onde a parcela $\frac{2lr}{A_P}$ é uma constante característica $K_M$ dependente apenas do projeto de cada máquina. A generalização da tensão induzida para uma máquina de "n" espiras pode ser obtida por superposição, pois toda a análise e as operações realizadas foram lineares, assim a tensão induzida para uma máquina real é: $e_{ind} = K_M\omega_m\phi$ e para esse caso $K_M = n\frac{2lr}{A_P}$. Devido ao princípio de origem da tensão induzida $e_{ind}$ ela também é denominada como força cotra-eletromotriz $E$, será utilizada essa nomenclatura nos tópicos posteriores.

## 2.2 Torque

No tópico anterior, foi demonstrada a existência de uma relação direta entre a força contra-eletromotriz $E$ e a velocidade angular $\omega_m$ nas condições de operação definidas para a máquina. Se o circuito de armadura da máquina for fechado, irá surgir uma corrente, seja pela diferença de potencial devido a força contra-eletromotriz $E$ ou por uma contribuição de alimentação de tensão externa $V_b$. A interação entre a corrente na bobina $i$ e a densidade de fluxo magnético gerado pelos polos $\mathbf{B}$ resulta em uma força $F_m$. Partículas carregadas eletricamente movendo-se na presença de um campo magnético são expostas a

$$F_m = Q\mathbf{v} \times \mathbf{B}, \tag{2.3}$$

onde, $F_m$ é a força devido ao campo magnético e $Q$ é a carga elétrica.

Porém, para essa aplicação tem-se uma corrente $i$, ao invés de uma carga $Q$. Assim,

utilizando o fato de que para corrente de convecção **J** tem-se a relação

$$\mathbf{J} = \frac{\Delta I}{\Delta S} = \rho \mathbf{v}, \tag{2.4}$$

onde, **J** é a densidade de corrente de convecção. $\Delta I$ é a variação de corrente e o $\Delta S$ é a variação na seção de área de superfície que a corrente percorreu, e o $\rho$ é a densidade do fluxo de cargas. Para um elemento de corrente de condução $id\mathbf{I}$ tem-se

$$id\mathbf{I} = \mathbf{J}d\upsilon, \tag{2.5}$$

onde, $i$ é a corrente de condução no fio da bobina e $d\upsilon$ é a derivada em relação á um volume infinitesimal, combinando ambos obtém-se

$$id\mathbf{I} = \mathbf{v}\rho d\upsilon = dQ\mathbf{v}, \tag{2.6}$$

onde, por definição, a densidade do fluxo de carga $\rho$ em um volume infinitesimal $d\upsilon$ é a própria carga infinitesimal $dQ$.

Por fim, em concordância com Sadiku (2013), a força sobre um elemento condutor $dl$ com corrente de condução $i$ pode ser obtida ao modificar (2.3), utilizando (2.6), obtendo a relação a seguir

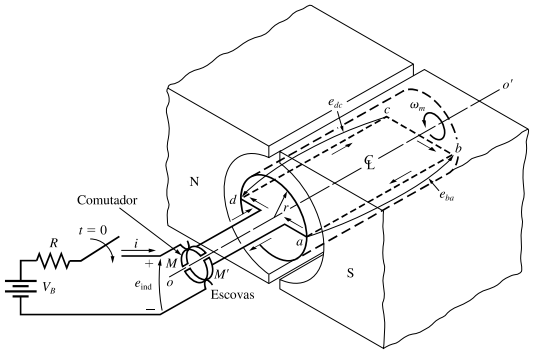
$$dF_m = dQ\mathbf{v} \times \mathbf{B} = id\mathbf{I} \times \mathbf{B}, F_m = \oint_L id\mathbf{I} \times \mathbf{B} = (i(\mathbf{I}_x \times \mathbf{B}))_{\sum \mathbf{L}_x}. \tag{2.7}$$

A partir de (2.7) torna-se mais simples a obtenção do valor da força magnética na bobina. Como trata-se de uma integral de linha, pode-se dividir o cálculo por trechos obtendo as contribuições de força separadamente. Por ser um quadrado, pode-se dividir a análise em quatro, assim o cálculo é simplificado para apenas o produto vetorial entre o vetor de comprimento do respectivo lado e o vetor de campo magnético multiplicado pelo valor de $i$, para cada trecho. Para os objetivo desse trabalho, tem-se que a máquina irá operar como motor, e assim, a corrente de armadura (corrente da bobina $i$) terá o sentido imposto pela fonte de tensão externa $V_b$, de acordo com a Figura 6.

Para simplificar as análises vetoriais, a Figura 7 exibe um diagrama baseado no motor da figura anterior. Onde as forças indicadas são respectivas a um lado e calculadas de acordo com $F_m$. A direção da corrente $i$ está indicada através de uma cruz para o sentido "entrando no plano da pagina"e um ponto para o sentido "saindo do plano da página".

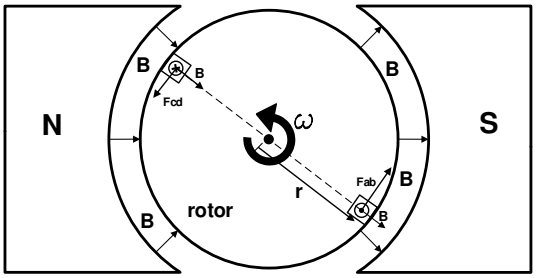
Para o trecho $\overline{ab}$, tem-se que o produto vetorial $\mathbf{I}_x \times \mathbf{B}$ tem como resultado um vetor de força com direção tangencial ao vetor de velocidade da espira, no sentido anti-horário, e de

Figura 6 – Vista em perspectiva da máquina CC alimentada por fonte externa




Fonte: (CHAPMAN, 2012).

Figura 7 – Diagrama da máquina com enfase nos vetores para o cálculo da força magnética

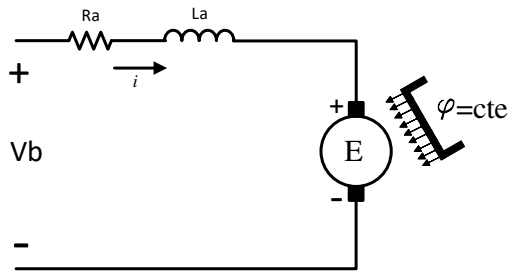



Fonte: Autoria própria.

módulo $F_{ab} = ilB$. Para o trecho $\overline{cd}$, tem-se os mesmos módulos, mas o vetor de comprimento do trecho $\mathbf{I}_{ab}$ inverte o sentido e obtendo o mesmo resultado a mesma direção tangencial a velocidade da espira e sentido de giro anti-horário. Esse comportamento pode ser verificado na Figura 7, onde os vetores estão em concordância com as análises apresentadas. Para os trechos $\overline{bc}$ e $\overline{da}$, obtém-se resultado nulo, pois no produto vetorial os vetores estão paralelo em ambos os casos. A soma das contribuições de força magnética apresenta o valor de $F_m = ilB + 0 + ilB + 0 = 2ilB$. A convenção de corrente está de acordo com a polaridade da fonte externa, pois para um motor $E < V_b$.

De forma análoga ao procedimento para o cálculo da força contra-eletromotriz $E$, pode-se generalizar o resultado da força magnética $F_m$ obtida por uma espira utilizando a superposição para o obter o valor para "n" espiras. Assim, o valor da força magnética nesse caso é de: $F_m = 2nilB$. Para motores, o valor de torque $T_m$ é, normalmente, mais utilizado que a força tangencial, assim será conveniente usar $T_m = F_m r = 2rnilB$ e substituindo $B = \frac{\phi}{A_P}$, tem-se: $T_m = K_M i\phi$ e $K_M = \frac{2rnl}{A_P}$.

Figura 8 – Circuito Equivalente de um motor CC de imã permanente




Fonte: Autoria própria.

## 2.3 Comportamento dinâmico e modelagem

A partir das equações anteriores e da definição dos circuitos da máquina foi possível montar o esquema de circuito da Figura 8, permitindo avaliar as grandezas elétricas envolvidas. A tensão nos terminais é $V_b$, a resistência de armadura $R_a$, a indutância de armadura $L_a$ e a força contra eletromotriz $E$ estão representadas na Figura 8. Onde $E$ está na ilustração do rotor, e um magneto indica o fluxo magnético do estator que complementa a natureza magnética desse funcionamento.

Utilizando as leis de Leis de Kirchhoff pode-se obter as equações diferenciais que regem o funcionamento desse circuito. a partir da analise de David Halliday, Robert Resnick (2016), tem-se que a soma de todas as variações de tensão em uma malha de circuito é nula, obtendo:

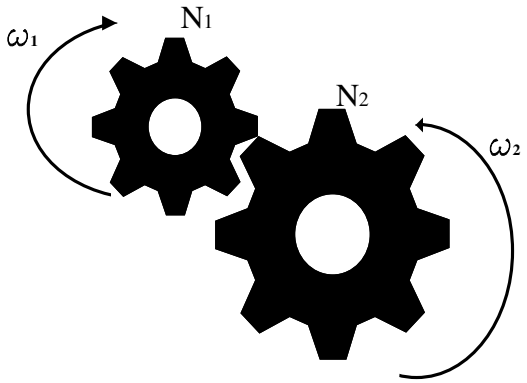
$$V_b = E + R_a i + L\frac{di}{dt}. \tag{2.8}$$

Raramente o projeto de um motor cumpre nominalmente as exigências mecânicas de uma aplicação. assim, utiliza-se alguns mecanismos para atender estas especificações, a solução comum e de baixo custo é adoção de uma caixa de redução. A energia mecânica se apresenta na forma $E_{Mec} = T\omega$t, onde $t$ é o $t$, $T$ e $\omega$ são o torque e a velocidade angular, respectivamente. De forma análoga aos transformadores elétricos, uma caixa de redução transforma os valores de torque ou de velocidade angular mantendo aproximadamente a mesma energia total. Nos transformadores elétricos a relação de transformação é através da relação entre espiras das bobinas. Para caixa de redução a relação de transformação pode ser através da relação entre os raios das polias ou os dentes das engrenagens, a depender do tipo de sistema.

Na Figura 9, tem-se o acoplamento de duas engrenagens com quantidade de dentes $N_1$ e $N_2$, respectivamente. Esse tipo de acoplamento garante a mesma velocidade linear para a superfície de ambas as engrenagens, caso não ocorra o fenômeno de patinação, que sempre deve

ser evitado. Cada engrenagem está acoplada á um eixo diferente, e como cada uma apresenta um número de dentes diferentes e a mesma velocidade linear, cada um dos eixos terá um velocidade angular diferente.

Figura 9 – Acoplamento mecânico por engrenagens com N1 e N2 dentes




Fonte: Autoria própria.

Os dentes das engrenagens estão distribuídos em sua superfície; a primeira engrenagem apresenta $N_1$ dentes e a segunda engrenagem apresenta $N_2$ dentes. A velocidade linear $V_{L1}$ em dentes/segundo demora $\frac{V_{L1}}{N_1}$ segundos para percorrer a superfície da primeira engrenagem, completando uma volta de $N_1$ dentes, analogamente a velocidade linear $V_{L2}$ demora $\frac{V_{L2}}{N_2}$ segundos para completar uma volta na segunda engrenagem. Como $V_{L1} = V_{L2}$ e a velocidade angular pode ser convertida em volta/segundo para rad/s de acordo com: $1\frac{volta}{segundo} = 2\pi\frac{radiano}{segundo}$. A partir dos resultados anteriores a relação entre as velocidades pode ser obtida por

$$\omega_1 = 2\pi\frac{V_{L1}}{N_1}, \omega_2 = 2\pi\frac{V_{L2}}{N_2} \tag{2.9}$$

utilizando a igualdade das velocidades lineares pode-se simplificar para

$$\omega_1 2\pi N_1 = V_{L1} = V_{L2} = \omega_2 2\pi, N_2, \tag{2.10}$$

e, por fim, tem-se a relação para a velocidade

$$\frac{\omega_1}{\omega_2} = \frac{N_2}{N_1}. \tag{2.11}$$

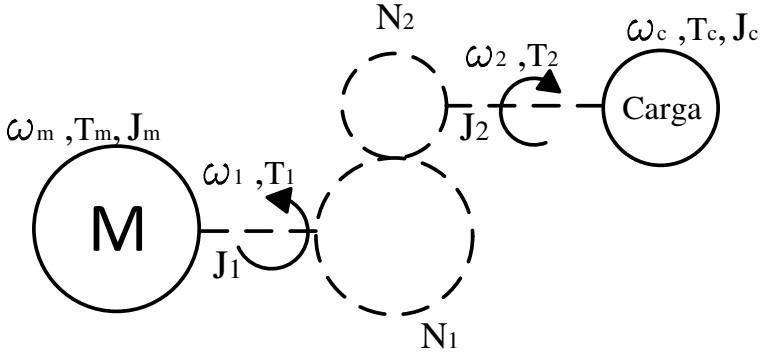
O acoplamento garante a mesma velocidade linear, então se a eficiência de transmissão for $\eta$, a redução de energia se dá pela análise do torque de acordo com

$$P_{Mec1} = T_1\omega_1, P_{Mec2} = T_2\omega_2, P_{Mec1} = \eta P_{Mec2}, \tag{2.12}$$

utilizando o resultado das velocidades, a relação de torque encontrada é

$$T_1\omega_1 = \eta T_2\omega_2, \frac{\omega_1}{\omega_2} = \eta\frac{T_2}{T_1}, \tag{2.13}$$

Figura 10 – Diagrama do conjunto mecânico




Fonte: Autoria própria.

obtendo

$$\eta \frac{T_2}{T_1} = \frac{N_2}{N_1}, \tag{2.14}$$

para o caso particular de eficiência máxima $\eta = 1$ a relação (2.14) se torna

$$\frac{T_2}{T_1} = \frac{N_2}{N_1} = \frac{\omega_1}{\omega_2}. \tag{2.15}$$

Utilizando (2.15) pode-se obter as grandezas mecânicas em uma região do sistema a partir do conhecimento dessas grandezas na região anterior. Assim, utilizando os resultados obtidos para o motor, obtém-se sequencialmente todos os valores intermediários até a carga na ponta do eixo. Um conjunto mecânico tipico, que foi utilizado no experimental deste trabalho, está representado na Figura 10. Nela são destacadas quatro zonas distintas, correspondentes ao motor, ao eixo acoplado diretamente, ao eixo após o conjunto de engrenagens e, por fim, à carga mecânica.

Utilizando a segunda lei de Newton para corpos girantes $T = J(\frac{d\omega}{dt})$ associada a (2.15) obtém-se as equações para o trecho acoplado diretamente ao eixo 1 resultando em (2.16). Como o eixo referente ao momento de inércia do motor $J_m$ e ao momento de inércia do eixo 1 $J_1$ estão acoplados e giram na mesma velocidade, eles podem ser somados e unificados em apenas uma analise. O somatório dos momentos externos sobre $J_m + J_1$ resulta em

$$Tm - T_2 \frac{N_1}{N_2} = \frac{d\omega_1}{dt}(J_m + J_1), \tag{2.16}$$

onde $T_m$ é o torque gerado pelo motor, $T_2$ é o torque de carga acrescentado do torque de inércia do resto do sistema convertida para o respectivo eixo e o $\frac{d\omega_1}{dt}$ é a aceleração angular do eixo 1.

Para o trecho após a caixa de engrenagens o eixo referente ao momento de inércia do eixo 2 $J_2$ e ao momento de inércia da carga $J_c$ estão acoplados e giram na mesma velocidade.

De forma análoga ao caso anterior tem-se

$$T_1 \frac{N_2}{N_1} = \frac{d\omega_2}{dt}(J_2 + J_c). \tag{2.17}$$

$T_1$ é o único torque externo e ele foi convertido para o respectivo eixo, $\frac{d\omega_2}{dt}$ é a aceleração angular do eixo 2.

As grandezas mecânicas buscadas para esse sistemas são: momento de inércia visto pelo motor $J_{eq2}$, o torque efetivo na ponta do eixo 2 e a velocidade angular $\omega_2$. Utilizando (2.15) e convertendo os resultados da análise realizada no eixo 2 para a referência do eixo 1 o (2.17) torna-se

$$T_2 \frac{N_2}{N_1}\frac{N_1}{N_2} = \frac{N_1}{N_2}\frac{d\omega_1}{dt}(J_2 + J_c). \tag{2.18}$$

substituindo em (2.16)

$$Tm = \frac{d\omega_1}{dt}\left(\frac{N_1}{N_2}\right)^2 (J_2 + J_c) + \frac{d\omega_1}{dt}(J_m + J_1) = \frac{d\omega_1}{dt}\left(\left(\frac{N_1}{N_2}\right)^2 (J_2 + J_c) + (J_m + J_1)\right), \tag{2.19}$$

convertendo (2.19) para a referência do eixo 2 obtém-se

$$Tm\frac{N_1}{N_2} = \frac{N_2}{N_1}\frac{d\omega_2}{dt}\left(\left(\frac{N_1}{N_2}\right)^2 (J_2 + J_c) + (J_m + J_1)\right), \tag{2.20}$$

e simplificando

$$Tm = \frac{d\omega_2}{dt}\left((J_2 + J_c) + (J_m + J_1)\left(\frac{N_2}{N_1}\right)^2\right). \tag{2.21}$$

O torque efetivo na carga $T_m$ está descrito em (2.19) convertido ao eixo 1 e em (2.21) para a referência do eixo 2. Esse ultimo resultado é o mais importante, pois os ensaios são feitos sobre a carga e ela está nesse eixo. A velocidade angular em cada eixo foi descrito por (2.15) e são proporcionais à relação dos dentes. Por fim, o momento de inércia visto pelo motor para o eixo 1 pode ser obtido por (2.22), pois por definição o termo que multiplica $\frac{d\omega_1}{dt}$ é o momento de inércia equivalente, obtendo

$$J_{eq1} = \left(\left(\frac{N_1}{N_2}\right)^2 (J_2 + J_c) + (J_m + J_1)\right), \tag{2.22}$$

e para o eixo 2, basta isolar o termo que multiplica $\frac{d\omega_2}{dt}$

$$J_{eq2} = \left((J_2 + J_c) + (J_m + J_1)\left(\frac{N_2}{N_1}\right)^2\right). \tag{2.23}$$

De acordo com Nise (2015), utilizando o procedimento análogo ao usado para obter a inércia equivalente do sistema e projetar seu valor em cada eixo, é possível realizar o procedimento para a constante de amortecimento equivalente $D_{eq}$ e a constante elástica equivalente do sistema. Se algum dos trechos não apresenta uma dessas grandezas baste zerar sua respectiva contribuição. Para o motor CC a constante elástica é desprezível, porém apresenta constante de amortecimento relevante e pode ser calculada por

$$D_{eq2} = \left( (D_2 + D_c) + (D_m + D_1) \left( \frac{N_2}{N_1} \right)^2 \right), \tag{2.24}$$

e o resultado em (2.21) com o acréscimo de $D_{eq2}$ torna-se

$$T_m = \frac{d\omega_2}{dt} J_{eq2} + \omega_2 D_{eq2}. \tag{2.25}$$

Utilizando o desenvolvimento realizado nesse capítulo, é possível obter as equações diferenciais que regem o movimento desse sistema apresentado. Uma delas é a equação que relaciona a tensão externa aplicada no circuito de armadura e a velocidade angular na carga. Nos experimentos realizados nesse trabalho, foram desenvolvidos justamente esses resultados, portanto o desenvolvimento teórico desse fenômeno é de fundamental importância para o embasamento dos resultados. Como o desenvolvimento de outras relações são demoradas e fogem do escopo desse trabalho, será desenvolvida apenas essa análise.

Usando (2.8),(2.2), (2.7), (2.21),(2.22),(2.23),(2.24),(2.25) e (2.26), pode-se relacionar, no domínio do tempo $t$, diretamente a tensão da fonte externa $V_b$ com a velocidade no eixo 2 $\omega_2$. Para simplificar as expressões será usado diretamente o momento de inércia equivalente $J_{eq2}$, a constante de amortecimento equivalente $D_{eq2}$ e o produto entre o fluxo $\phi$ e a constante da máquina $K_m$ serão unificados na constante $K_e$. Como resultado tem-se

$$V_b(t) = K_e \omega_2(t) + \left( \frac{R_a}{K_e} \right) \left( J_{eq2} \frac{d\omega_2(t)}{dt} + D_{eq2} \omega_2(t) \right) + \left( \frac{L_a}{K_e} \right) \left( J_{eq2} \frac{d^2\omega_2(t)}{dt^2} + D_{eq2} \frac{\omega_2(t)}{dt} \right), \tag{2.26}$$

convertendo o resultado anterior para o domínio da frequência, a partir da Transformada de Laplace, e organizando os termos obtém-se:

$$\frac{\omega_2(s)}{V_b(s)} = \frac{K_e}{R_a D_{eq2} \left( 1 + \frac{L_a}{R_a} s \right) \left( 1 + \frac{J_{eq2}}{D_{eq2}} s \right) + K_e^2}, \tag{2.27}$$

onde $\left( \frac{L_a}{R_a} \right)$ é a constante de tempo elétrica do sistema e $\left( \frac{J_{eq2}}{D_{eq2}} \right)$ é a constante de tempo mecânica do sistema.

Além do amortecimento $D_{eq}$ causado pelo atrito viscoso, o motor também sofre a influência do atrito de Coulomb e sua modelagem pode ser realizada através da inclusão de uma não linearidade do tipo zona-morta. Na análise que teve como resultado (2.27) tem-se um desenvolvimento e uma resposta totalmente linear, assim, qualquer entrada na tensão $V_b$ diferente de zero causará uma resposta na saída $\omega_2$ diferente de zero, reação que não ocorre nos motores reais. Neles a saída permanece zerada até que a magnitude do valor de entrada ultrapasse um certo limite, esse comportamento é denominado zona-morta (SALCEDO, 2010).

As características e a modelagem da zona-morta apresentam pouca exatidão, dependem de fatores ambientais e podem variar com o tempo. Na maioria das práticas os parâmetros da zona morta são difíceis de obter, assim uma modelagem sobre a natureza do fenômeno é pouco útil e normalmente são realizados procedimentos práticos de correção direta da zona-morta sem entrar em maiores detalhes. Os efeitos da zona-morta serão atenuados a partir da implementação de uma zona morta inversa adaptativa no *firmware* a partir de dados experimentais coletados, seu comportamento e correção serão no desenvolvimento experimental do trabalho (TAO; KOKOTOVIC, 1995).

# 3 MICROCONTROLADORES DSPIC30F

O microcontrolador dsPIC30F4011 foi escolhido a partir da disponibilidade e dos conhecimentos prévios adquiridos em disciplinas curriculares. Este microcontrolador é especializado em controle de motores, apresentando funções nativas que auxiliam diretamente nessas aplicações, justificando a importância e a escolha desse produto.

## 3.1 Introdução

O protótipo experimental desenvolvido é baseado em um sistema de hardware embarcado microcontrolado. A família de microcontroladores dsPIC30F é um produto da Microchip®. Eles são microcontroladores de 16 bits, de arquitetura Computador com um Conjunto Reduzido de Instruções (do inglês *Reduced Instruction Set Computer - Reduced Instruction Set Computer* (RISC)), que, dependendo do modelo, podem trabalhar com frequências de até 25 MHz e endereçar até 2048 KB de memória. Alguns módulos estão presentes em todos os modelos da família, que são básicos, mudando apenas a quantidade deles; porém existem alguns módulos de uso específico projetados para uma aplicação exata. De acordo com o modelo, pode-se encontrar unidades de memórias flash, temporizadores, Interface de Comunicação Serial Universal (do inglês, *Universal Serial Communication Interface* (USCI)), interface de Barramento Serial Universal (do inglês, *Universal Serial Bus* (USB)), controladores de Tela de Cristal Líquido (do inglês, *Liquid Crystal Display* (LCD)), conversores Analógico/Digital (A/D) e conversores Digital/Analógico (D/A), amplificadores operacionais, hardware multiplicador, relógio de tempo-real, controladores de supervisão de tensão de alimentação, sensor de temperatura e entre outros.

A principal vantagem dos microcontroladores da família dsPIC30F com relação a outros microcontroladores com características semelhantes é o alto processamento. Essa característica associada a um designer robusto, tamanho compacto e de baixo custo, faz dele uma boa escolha para aplicações de sistemas embarcados que tenha previsão de escalonamento, opere em ambientes limitados ou com considerável restrição de orçamento.

A Microchip® disponibiliza várias ferramentas de desenvolvimento para o dsPIC30F4011, desde manuais para toda família e códigos com exemplos de aplicações comuns abrangendo até compiladores sequenciais e simuladores. Para a implementação do *firmware* do protótipo desenvolvido, utilizou-se a ferramenta MPLAB X IDE® através de um computador com sistema operacional Microsoft Windows 10 e um gravador PICkit 3. Nessa seção serão

apresentados apenas os módulos configurados e utilizados diretamente pelo *firmware*, os quais serão descritos de acordo com o seu funcionamento e seu uso dentro da lógica implementada para o protótipo.

## 3.2 Microcontrolador dsPIC30F4011

O dsPIC30F4011 incorpora uma Unidade Central de Processamento (do inglês *Central Processing Unit - Central Processing Unit* (CPU)) tipo RISC, módulos e um flexível sistema de *clock* que se interconectam usando um Barramento de Endereços de Memória (do inglês *Memory Adress Bus - Memory Adress Bus* (MAB)) comum com arquitetura Harvard e um Barramento de Dados de Memória (do inglês *Memory Data Bus - Memory Data Bus* (MDB)), que estão identificados através de um diagrama de blocos na Figura 11.

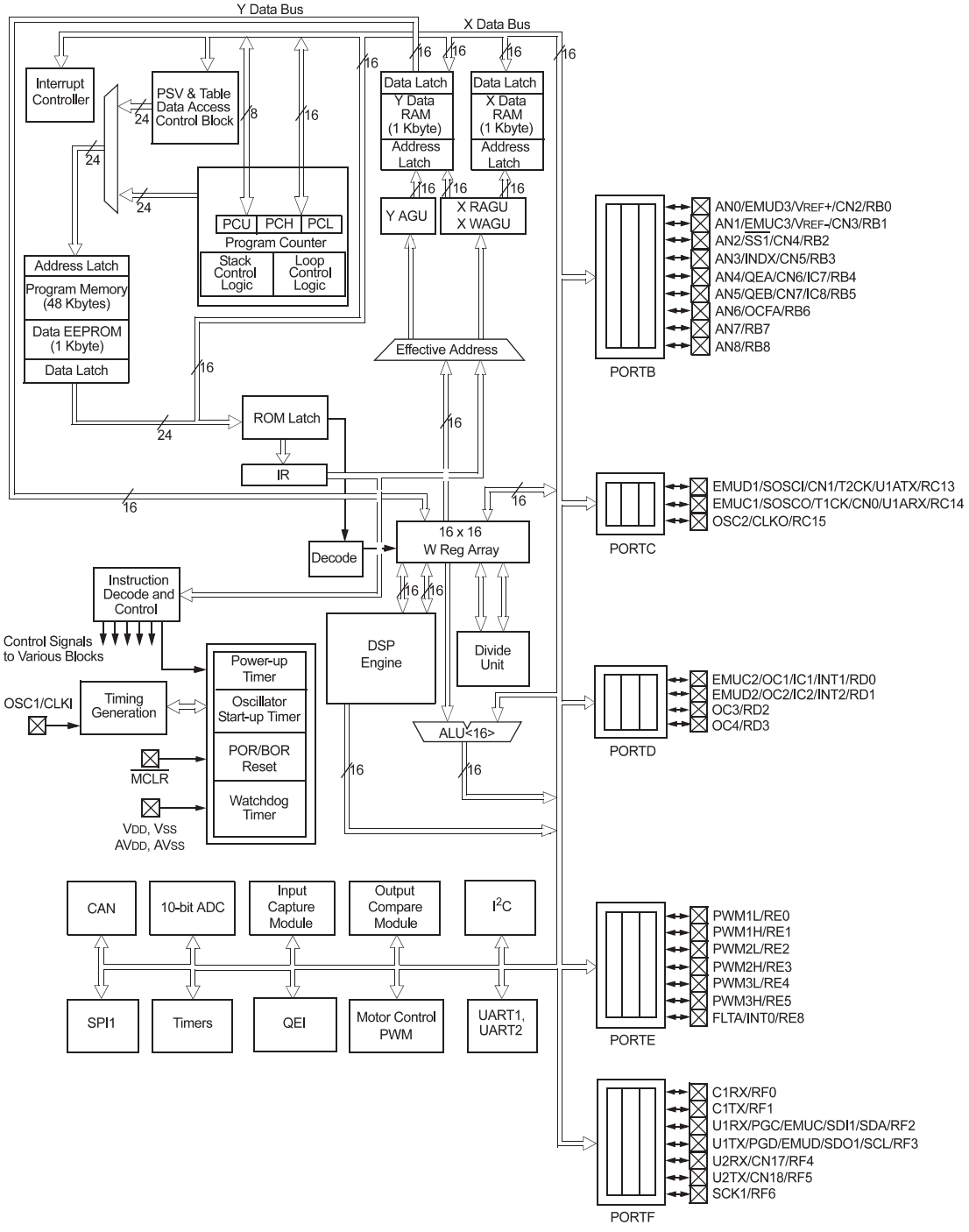
As principais características do dsPIC30F4011, podem ser identificadas em (MICROCHIP TECHNOLOGY, 2005) pois trata do *datasheet* do produto Microchip®, e estão listadas a seguir:

- arquitetura do conjunto de instruções otimizada para o compilador em linguagem de programação C com medidores de endereçamento flexíveis e com 84 instruções básicas. Com tamanho de palavra das instruções é de 24 bits e de dados são de 16 bits de largura;
- 2 Kbytes de Memória de Acesso Aleatório (do inglês *Random Access Memory - Random Access Memory* (RAM)) nativos no chip e 1 Kbite de dados não voláteis tipo Memória Somente de Leitura Apagável Eletricamente (do inglês *Electrically-Erasable Programmable Read-Only Memory - Programmable Read-Only Memory* (EEPROM));
- 30 fontes distintas de interrupção de módulos e 3 fontes de interrupção externa, apresentando hierarquia de 8 níveis de prioridade das interrupções;
- hardware de 17 bits x 17 bits de ciclo único multiplicador fracionário/inteiro, com todas as instruções do processador digital de sinais (do inglês *Digital Signal Processing - Digital Signal Processing* (DSP)) executadas em um único ciclo de máquina;
- são 20 pinos configuráveis para entrada/saída e suportam alta corrente, no máximo 25 mA;
- cinco temporizadores/contadores de 16 bits; opcionalmente, permite o emparelhamento dos temporizadores de 16 bits em módulos temporizadores de 32 bits;
- 2 módulos *Universal Asynchronous Receiver Transmitter* (UART) onde o primeiro *byte* a entrar é o primeiro a sair.
- 6 canais de saída *Pulse Width Modulation* (PWM) com 3 geradores de ciclo de trabalho,

com base de tempo dedicada;

- módulo dedicado para Codificador de Quadratura de *encoder*;
- conversor analógico-digital de 10 bits A/D com 4 entradas com *Sample and hold*;
- tecnologia *Complementary Metal Oxide Semiconductor* (CMOS), apresentando um baixo consumo de energia e com ampla faixa de tensão operacional (2,5V a 5,5V).

Figura 11 – Diagrama de blocos da Arquitetura do dsPIC30F4011




Fonte: (MICROCHIP TECHNOLOGY, 2005)

## 3.3 Sistema de *clock*

Para esse dsPIC existem três formas principais de proporcionar o *clock* necessário para o funcionamento do dispositivo. Elas são escolhidas fundamentalmente a partir do compromisso entre desempenho e consumo de energia, podendo ser influenciado também por características especificas do projeto. Em concordância com Microchip Technology (2005), as opções são descritas a seguir:

- XT: oscilador de baixa frequência/alta frequência que pode ser usado com cristais de relógio de baixa frequência ou fontes de *clock* externas de 32768 Hz ou com cristais padrão, ressonadores, ou fontes de *clock* externas na faixa de 400 kHz a 16 MHz. Maior desempenho e consumo;
- ERC: Oscilador Digitalmente Controlado interno. Disponível em todos os dispositivos. Basicamente, é um oscilador RC altamente controlável que inicia em menos de 1 $\mu s$, solução equilibrada;
- FRC: oscilador de baixa frequência, baixa potência, interno, de 32 kHz. Pode ser usado como alternativa a ERC quando a precisão de um cristal não é necessária, essa é a solução mais econômica.

A partir do *clock* principal fornecido ao dsPIC, obtém-se três saídas que serão osciladores para diferentes módulos internos. Uma breve descrição de seu ciclo de trabalho e uso estão expostas a seguir:

- Master clock, MCLK: sua fonte é selecionável por software. Pode ser dividido por 2, 4 ou 8. O MCLK é usado pela CPU e alguns módulos;
- Sub-main clock, SMCLK: assim como o MCLK, sua fonte é selecionável por software e pode ser dividido por 2, 4 ou 8. É distribuído diretamente aos módulos;
- Auxiliary clock, ACLK: sua fonte é selecionável por software como ERC ou FRC. Pode ser dividido por 2, 4 ou 8. Também é distribuído aos módulos, mas principalmente para *timers*.

## 3.4 Módulo controlador de portas

Esse microcontrolador permite utilizar até 8 pinos de entrada/saída analógicos e até 30 pinos como entrada/saída digitais. Cada pino pode ser configurado individualmente por meio de software, onde os registradores necessários para definir a configuração dependem do

respectivo pino, pois todos eles compartilham outras funções, e assim, deve-se definir diretamente sua operação.

De forma complementar, tem-se as interrupções externas. No total são três possíveis interrupções em pinos distintos, as interrupções podem ser ativadas tanto por uma mudança de bit ascendente como descendente. É possível configurar a prioridade das interrupções e a bandeira de aviso de cada uma via software.

No dsPIC30F4011, a prioridade do uso do pino é dos módulos; assim, ao habilitar o módulo a respectiva função de porta é desativada. Todos os pinos digitais que apresentam função de porta também apresentam três registradores diretamente associados: o registrador de direção TRISx, o registrador de escrita LATx e o registrador de leitura PORTx. Para os pinos analógicos os registradores associados diretamente são ADPCFG e o TRIS, onde o primeiro define se o pino de porta será analógica ou digital e o segundo define a direção do pino como entrada o saída. Todos esses registradores tem tamanho de 16 bits, porém parte desses bits estão desabilitados. Existem ainda outros registradores que interagem com os pinos de porta, nesse tópico será descrito apenas os que foram configurados.

### *3.4.1 Registrador de direção TRISx*

Cada bit do registrador representa o nível lógico do respectivo pino quando este está configurado como entrada/saída digital. Ao resetar o microcontrolador esse registrador coloca os respectivos bits em nível lógicos alto, que reapresenta saída, como medida de segurança. Tanto os pinos com função de porta digital quanto os pinos de porta analógica são configurados por esse registrador, o índice “x” distingue os diferentes registradores TRIS, que nesse microcontrolador são B,C,D,E e F.

### *3.4.2 Registrador de escrita LATx*

O nível lógico de cada pino de porta assume o valor armazenado no respectivo bit do registrador quando o pino está configurado como saída e *pullup*/*pulldown* está desabilitado. Quando o pino está configurado como entrada, o valor do respectivo bit é armazenado em um *buffer* e, quando posteriormente o pino é configurado como saída, o pino de porta assume este valor.

### *3.4.3 Registrador de leitura PORTx*

Quando a respectivo pino de porta está configurado como entrada esse registrador apresenta o nível lógico de cada um dos pinos em seus bits. Assim, o valor do bit dos PORTx é a leitura do nível logico atual do pino. Na outra situação, quando o pino de porta está configurado como saída e é escrito algo no registrador PORTx, o efeito é semelhando ao escrever diretamente no respectivo registrador LATx. Assim, ao tentar "escrever"via PORTx apenas o valor do LATx é modificado e o PORTx permanece com o nível logico real e atual do pino.

### *3.4.4 Registradores de seleção de função*

Em concordância com o que foi apresentado anteriormente, sabe-se que os pinos compartilham de conexões com funções de outros módulos e mesmo quando o pino está configurado para uma outra função, ainda é necessário selecionar a direção do pino por meio do TRISx. Os registradores que controlam a função do pino depende diretamente de quais módulos estão compartilhando este pino, Para o módulo de Modulação de Largura de Pulso (PWM, do inglês *Pulse Width Modulation*) o registrador é o PTCON. Para o módulo da interface de quadratura de *encoder* o registrador é o DFLTCON. Para o o módulo de comunicação Receptor/Transmissor assíncrono universal ( UART, do inglês *Universal Asynchronous Receiver Transmitter*) o registrador é o UxMODE e para o módulo de módulo de comparação de saída o registrador é o OCxCON.

### *3.4.5 Registradores de habilitação de interrupção IEC0<15:0>, IEC1<15:0> e IEC2<15:0>*

Através desse registradores pode-se liberar a capacidade de gerar interrupções quando ocorre uma transição de bit em um pino de entrada específico. Nesse dsPIC são 10 pinos, os CN0 até CN7, CN17 e CN18 para interrupção externas. Por padrão interrupções ficam desabilitadas na inicialização, todos as possíveis origens de interrupção são ativadas por esses registradores.

### *3.4.6 Registradores de seleção de transição da interrupção CNPUx*

Cada bit desse registrador seleciona a transição que ativa a interrupção. Quando o bit é igual a zero, a interrupção é acionada por uma transição de nível lógico baixo para alto. O inverso ocorre para o bit igual a um. Esse registrador não é inicializado e, portanto, deve ser configurado antes das interrupções serem habilitadas.

### *3.4.7 Registradores de bandeiras de interrupções*

Todas a bandeiras de interrupção estão distribuídas nesses registradores de 16 bits, IFS0<15:0>, IFS1<15:0> e IFS2<15:0>. Quando algum desse bits apresentar nível lógico alto que ocorreu uma interrupção, interna ou externa, no correspondente módulo. Para interrupções externas, de acordo com o definido nos tópicos anteriores, o (bit=1) significa que ocorreu uma mudança no nível lógico do respectivo pino. Uma interrupção modifica a sequencia de execução da pilha levando o ponteiro para uma rotina de serviço de interrupção sempre que permanece até que a rotina esteja concluída e o respectivo bit de bandeira seja apagado. Esse registrador é de leitura e gravação. Assim, pode-se gerar uma interrupção via software gravando o valor alto no respectivo bit em que se deseja ativar a interrupção.

### *3.4.8 Registradores de prioridade de interrupções IPC0<15:0> e IPC11<7:0>*

O nível de prioridade atribuível a cada uma das fontes de interrupção é realizada de forma centralizada nestes doze registros. Dessa forma tem-se a garantia de que se algumas bandeiras ocorrerem no mesmo momento, o módulo com interrupção escolhido com maior prioridade terá diretamente sua rotina de interrupção executada e as outras serão executadas sequencialmente de forma análoga.

## 3.5 Temporizadores

Esses módulos constituem de registradores de contagem de 16 bits que podem ser associados para um configuração de 32 bits. Eles são controlados pelos registradores TxCON. Um temporizado é apenas um contador e não apresenta necessariamente um conceito direto de tempo. É trabalho do programador estabelecer uma relação entre o valor no contador e e a grandeza de tempo, quando necessário.

### *3.5.1 Módulos temporizadores*

De forma resumida, o temporizador consiste de um comparador e um contador. Via software o programador define um valor de 16 bits no registrador PRx, esse será o valor limite da contagem, ao iniciar a contagem o registrador TMRx é incrementado sequencialmente até atingir o valor de PRx, nesse momento é ativada a bandeira do respectivo temporizador e é

zerado o valor do TMRx. Assim, periodicamente a bandeira do temporizador é ativada e, por consequência, a execução da CPU é desviada para a rotina de interrupção. A relação entre a periodicidade da interrupção e o tempo está na forma com que o incremento é realizado no TMRx, esse incremento depende da frequência interna do relógio do microcontrolador $FCY_{micro}$ e do *Loop* de bloqueio de fase $PLL_{TMRx}$ configurada no TxCON, e assim o valor da frequência de ocorrencia de interrupção $F_{inter}$ pode ser obtido por

$$F_{inter} = \left( \frac{FCY_{micro} PLL_{TMRx}}{PRx - TMRx_{inicial}} \right). \tag{3.1}$$

### *3.5.2 Módulos de captura/comparação*

Os *Timer 2* e *Timer 3* do dsPIC30F4011 tem quatro canais de captura/comparação. Cada um deles é controlado por um registrador ICxCON, onde é realizada a configuração. O registrador principal de cada canal de captura/comparação é o ICxBUF. No modo de captura, ele armazena o valor do contador TMRx no momento que uma transição ocorre na sua entrada correspondente. No modo de comparação, ele armazena o valor da contagem em que sua respectiva saída deve mudar de valor e uma interrupção é solicitada.

O modo captura não é usado para essa aplicação, pois o sensor de posição é um *encoder* que fornece diretamente a informação de maneira digital, uma alternativa é o tacogerador que gera um valor de tensão diretamente proporcional a velocidade do seu eixo. Um valor tensão no tacogerador é uma grandeza analógica, e a partir do modo captura é possível coletar parte dessa informação de maneira digital, para que dessa forma possa ser usada pelo microprocessador.

No modo comparação é possível gerar o sinal de PWM. O *timer* associado pode ser o *timer2* ou *timer3*, a escolha deve ser feita através do registrador OCTSEL e o *timer2* é a configuração padrão. Existem até seis modos de configuração para esse módulo, e que são escolhidos a partir dos bits OCxCON<2:0>. No entanto o PWM usado nesse trabalho foi gerado por um outro módulo, o módulo de controle do PWM do motor, especifico para aplicação desse dsPIC.

## 3.6 Módulos de comunicação

Estão disponíveis diversas formas de estabelecer e implementar uma comunicação via microcontrolador. No presente trabalho, será tratado apenas o modo utilizado, o qual é a comunicação do tipo UART.

Para habilitar e realizar a comunicação via UART deve-se configurar alguns registradores, esse microcontrolador apresenta dois módulos UART independentes. O registrador UxMODE permite definir o protocolo de comunicação, tal como paridade, número de bits, definir quais os pinos referentes ao transmissor $TX$ e ao receptor $RX$ e, por fim, habilitar o módulo. Tem-se que definir a direção dos pinos de comunicação utilizando o módulo de porta via registrador TRISx. O registrador trata de como o *buffer* do microcontrolador deve se comportar UxSTA e quantas e como serão as interrupções geradas pela recepção ou transmissão realizadas. Por fim, a taxa de transmissão $BR$ é obtida configurando o registrador UxBRG de acordo com

$$BR = \left( \frac{FCY_{micro}}{16(UxBRG+1)} \right), \tag{3.2}$$

atentando para o fato de que os registradores só permitem números representáveis em 16 bits. Como habilitar as interrupções e tratar as bandeiras foi descrito nos tópicos 3.4.5 e 3.4.7 configurando os registradores IEC0 e IFS0, respectivamente.

## 3.7 Módulo de controle do PWM do motor

Esse módulo é especifico para a função de controle de motores, ele simplifica a tarefa de gera múltiplas saídas sincronizadas de PWM. Com esse hardware dedicado tem-se a possibilidade de controlar motores de indução trifásica, motor de relutância variável, fontes de alimentação ininterrupta, motores CC e entre outros. São 6 saídas de PWM com 3 geradores de ciclo de trabalho, resolução de 16 bits, avaliador de falhas e a possibilidade de gerar interrupções do próprio módulo. Os seis pinos de entrada/saída são agrupados em alto/baixo numerados pares, denotados pelos sufixos alto $H$ ou baixo $L$, respectivamente. Para cargas complementares, os pinos PWM baixos são sempre o complemento do pino de entrada/saída alto correspondente. O módulo PWM permite vários modos de operação que são benéficas para o controle de potência específica aplicações citadas.

Diferente do módulo anterior, o contador deste é dedicado. Assim, todos os cinco *timers* ainda estão disponíveis para uso simultâneo. A configuração do módulo é realizada através do registrador PTCON, o valor da contagem está no registrador PTMR, o teto da contagem é estabelecido pelo registrador PTPER, o registrador PWMCON1 define o modo de operação do PWM e o PWMCON2 trata das interrupções. Os geradores de ciclo de trabalho são independentes e são controlados pelos registradores PDC1, PDC2 e PDC3. Os modos de operação definidos pelo PWMCON1 definem se os pares $H$ e $L$ serão duplicados ou complementares, se os ciclos

de trabalho devem operar em conjunto e outras funcionalizabilidades do tipo.

## 3.8 Módulo da interface de quadratura de *encoder*

O módulo a interface de quadratura de *encoder* (do inglês *Quadrature Encoder Interface - Quadrature Encoder Interface* (QEI)) fornece a interface para os codificadores atuando na obtenção de dados de posição mecânica. Em resumo esse módulo também é um contador, que nesse caso conta os pulsos gerados pelo *encoder* e reconhecidos pela QEI armazenando no registrador de 16 bits POSCNT. Parte da configuração é feita pelo registrador ADPCFG definindo os respectivos pinos para entrada do *encoder* e declarando-os como digitais, o registrador TRISx define a direção como entrada, o registrador DFLTCON trata de configurar o filtro de entrada, como será a interrupção e da avaliação de erros. Para definir o tipo de *encoder* utilizado e garantir o funcionamento do sistema, deve-se definir via software o registrador QEICON no mesmo padrão de funcionamento que o *encoder* utilizado no projeto, outro registrador importante é o MAXCNT que limita o limite de contagem para o registrador contador POSCNT que é fundamental para definir limites de movimento. Por fim, para habilitar as interrupções e tratar as bandeiras utiliza-se do que foi definido nos tópicos 3.4.5 e 3.4.7 desse capítulo.

# 4 SINTONIA AUTOMÁTICA DE CONTROLADOR PID

Um comando de sintonia automática de controlador é uma das funções mais desejadas pelos operadores de um sistema de controle industrial. Tal função facilita e acelera o processo de comissionamento de novas plantas.

A sintonia automática de um controlador pode ser realizada em duas etapas. Primeiro, realiza-se um experimento e, usando um método de identificação, obtém-se um modelo. Depois disso, este modelo é usado juntamente com um método de sintonia para se obter os parâmetros do controlador. Todo esse processo é realizado de forma autônoma por algorítimo executado em um dispositivo microprocessado.

Existem algumas soluções comerciais que permitem realizar todo o processo de sintonia automática, não necessitando de conhecimentos prévios da planta pelo operador. O trabalho de (BERNER *et al.*, 2018) compara três soluções comerciais de sintonia automática com um método proposto e, além disso, serve como exemplo do quanto essa tecnologia está consolidada na industria.

Um método que pode ser usado em plantas estáveis e integradoras de uma entrada e uma saída (do inglês, *Single Input Single Output* (SISO)), caracterizado por sua simplicidade na identificação, foi desenvolvido nos trabalhos de (OHTA *et al.*, 1980; NISHIKAWA *et al.*, 1984). Portanto, seus métodos de identificação por resposta ao degrau serão adotados.

As etapas e os métodos de identificação e sintonia do controlador adotados são apresentados a seguir.

## 4.1 Identificação em malha aberta

Inicialmente, é necessário realizar a identificação da planta de forma autônoma. Este processo pode ser realizado observando a resposta do sistema à uma entrada conhecida, como um impulso, um degrau, uma rampa, uma senoide ou uma Sequencia Binária Pseudo-Aleatória Sequencia Binária Pseudo-Aleatória (SBPA).

Neste trabalho foi utilizado o método proposto em (OHTA *et al.*, 1980; NISHIKAWA *et al.*, 1984), baseado em resposta de entrada tipo degrau para plantas estáveis e a resposta da planta ao aplicar um pulso na entrada para plantas integradoras. O método de identificação será embarcado em um microcontrolador e, assim, são necessários baixo custo computacional e robustez a ruído de medição. O método de (OHTA *et al.*, 1980; NISHIKAWA *et al.*, 1984)

utiliza apenas operações algébricas simples e integrais de sinais, que são resolvidas rapidamente pelo microcontrolador. Além disso, por ser baseado no cálculo de integrais, ele é robusto aos ruídos de média nula, pois a integral de um sinal com esse tipo de ruido é aproximadamente igual à integral de um sinal sem o ruido.

No Capitulo 2 foi obtido um modelo para o motor CC em (2.26) e (2.27) relacionando tensão de entrada e velocidade como também a tensão de entrada com a posição, esses modelos são baseados nas características físicas da máquina e apresentam complexidade considerável. Encontrar o valor os parâmetros desse modelo é uma tarefa bastante difícil e para muitas aplicações não é necessário, pois por exemplo, em uma aplicação de controle esse modelos podem ser simplificados para inclusão apenas do polo dominante, uma vez que o polo de natureza elétrica é bem mais rápido que o natureza mecânica. Assim, para esse aplicação, é vantajoso utilizar um modelo mais simples e obter seus parâmetros por métodos de identificação baseados em ensaios ao invés das características físicas da máquina.

O Modelo a ser identificado para o controle de velocidade é FOPDT, que pode ser descrito por três parâmetros: ganho estático $K_v$, constante de tempo $T_v$ e atraso de transporte $L_v$. Sua função de transferência é dada por

$$P(s) = \frac{K_v}{T_v s + 1} e^{-L_v s}. \tag{4.1}$$

O Modelo a ser identificado para controle de posição é do tipo IFOPDT, que pode ser descrito por três parâmetros: ganho estático $K_p$, constante de tempo $T_p$ e atraso de transporte $L_p$. Sua função de transferência é dada por

$$P(s) = \frac{K_p}{s\,(T_p s + 1)} e^{-L_p s}. \tag{4.2}$$

Inicialmente será definido o processo realizado para a planta estável. Após a aplicação do degrau na entrada $u_v(t)$ e a saída $y_v(t)$ atingir o regime permanente, inicia-se a identificação do modelo.

O primeiro parâmetro a ser obtido é o ganho estático, calculado por

$$K_v = \frac{y_v(\infty)}{A_{uv}}, \tag{4.3}$$

onde $y_v(\infty)$ é o valor final da saída e $A_{uv}$ é a amplitude do degrau.

Considera-se a soma da constante de tempo e do atraso de transporte como $T_{0v} = T_v + L_v$. Usando a integral $A_{0v}$:

$$A_{0v} = \int_0^\infty \{y_v(\infty) - y_v(t)\} dt, \tag{4.4}$$

é possível obter

$$T_{0v} = \frac{A_{0v}}{y_v(\infty)}. \tag{4.5}$$

A integral da saída de $t = 0$ até $t = T_{0v}$ é definida por:

$$A_{1v} = \int_0^{T_{0v}} y_v(t)dt. \tag{4.6}$$

A constante de tempo $T_v$ é obtida então por

$$T_v = \frac{A_{1v}}{e^{-1} \cdot y_v(\infty)} \tag{4.7}$$

e o atraso de transporte $L_v$ por

$$L_v = T_{0v} - T_v. \tag{4.8}$$

Para a planta do controle de posição, tem-se um desenvolvimento análogo atendando para as características próprias. Após a aplicação de um pulso na entrada $u_p(t)$ e a saída $y_p(t)$ atingir o regime permanente, inicia-se a identificação do modelo.

O primeiro parâmetro a ser obtido é o ganho estático, calculado por

$$K_p = \frac{y_p(\infty)}{t_p A_{up}}, \tag{4.9}$$

onde $y_p(\infty)$ é o valor final da saída considerando que o sistema partiu do repouso, $t_p$ é a largura do pulso no tempo e o $A_u$ é a amplitude do pulso.

Considera-se a soma da constante de tempo e do atraso de transporte como $T_{0p} = T_p + L_p$. Utilizando a integral $A_{0p}$:

$$A_{0p} = \int_0^{\infty} \{y_p(\infty) - y_p(t)\}dt, \tag{4.10}$$

é possível obter

$$T_{0p} = \frac{A_0}{y_p(\infty)} - \frac{t_p}{2}. \tag{4.11}$$

Usando $T_{0p}$, pode-se calcular a integral $A_{1p}$:

$$A_{1p} = \int_0^{T_{0p}} y_p(t)dt. \tag{4.12}$$

Assim, a constante de tempo $T_p$ pode ser obtida por

$$T_p = \frac{A_{1p}}{(0.5 - e^{-1})K_p A_{up} T_{0p}}, \tag{4.13}$$

e o atraso de transporte $L_p$ por

$$L_p = T_{0p} - T_p. \tag{4.14}$$

## 4.2 Sintonia do controlador

Existem vários métodos de sintonia de controladores baseados em modelos FOPDT e modelos IFOPDT na literatura de sistema de controle. No entanto, simplicidade e baixo custo computacional são características desejáveis para métodos de sintonia automática. Portanto, neste trabalho, usa-se uma regra de sintonia como o método de sintonia do controlador PID. No entanto, sempre existe sobressinal quando se usa um controlador PID para controlar uma planta integradora (VERONESI; VISIOLI, 2010a). Como é desejável que não haja sobressinal na saída, optou-se por substituir o controlador PID por um controlador I-PD. Este controlador tem uma variação do sinal de controle mais suave do que o controlador PID, resultando em uma resposta da saída menos agressiva.

Será utilizada a regra de sintonia proposta por (SKOGESTAD, 2003). Segundo esta regra, para uma planta FOPDT, é projetado um controlador Proporcional-Integral (PI), representado por

$$C(s) = K_c \left(1 + \frac{1}{T_i s}\right). \tag{4.15}$$

Os parâmetros do controlador $K_c$ e $T_i$ são obtidos por

$$K_c = \frac{T}{K(T_{cv} + L)}, \tag{4.16}$$

$$T_i = min\{T, 4(T_{cv} + L)\}, \tag{4.17}$$

onde $T_{cv}$ é a constante de tempo de malha fechada desejada, um parâmetro livre de sintonia.

A partir da sintonia do controlador PID descrita por (SKOGESTAD, 2003) para plantas do tipo IFOPDT, é possível obter um controlador I-PD. O controlador PID na forma série é representado por:

$$C_{PID}(s) = K_{cs} \left(\frac{T_{is} s + 1}{T_{is} s}\right) \left(\frac{T_{ds} s + 1}{T_{fs} s + 1}\right). \tag{4.18}$$

Os parâmetros do controlador PID na forma série, segundo (SKOGESTAD, 2003), são obtidos por:

$$K_{cs} = \frac{1}{K_p(T_{cp} + L_p)}, \tag{4.19}$$

$$T_{is} = 4(T_{cp} + L_p), \tag{4.20}$$

$$T_{ds} = T_p, \tag{4.21}$$

onde $T_{cp}$ é a constante de tempo de malha fechada desejada, um parâmetro livre de sintonia.

O controlador I-PD na forma ideal é dado por:

$$C_I(s) = \frac{K_c}{T_i s}, \tag{4.22}$$

$$C_{PD}(s) = K_c \left(1 + \frac{T_d s}{T_f s + 1}\right), \tag{4.23}$$

Sua função de transferência do sinal de controle é dada por

$$U(s) = C_I(s)E(s) + C_{PD}(s)Y(s), \tag{4.24}$$

onde $E(s)$ e $Y(s)$ são as funções de transferência dos sinais de erro e da saída, respectivamente.

Assim, os parâmetros do controlador PID série devem ser convertidos para a forma ideal de acordo com:

$$K_c = K_{cs}\left(1 + \frac{T_{ds}}{T_{is}}\right), \tag{4.25}$$

$$T_i = T_{is}\left(1 + \frac{T_{ds}}{T_{is}}\right). \tag{4.26}$$

$$T_d = \frac{T_{ds}}{\left(1 + \frac{T_{ds}}{T_{is}}\right)}, \tag{4.27}$$

A constante de tempo do filtro derivativo é calculada como $T_f = T_d/10$.

O compromisso entre agressividade da resposta e robustez depende da escolha do valor da constante de tempo de malha fechada $T_{cv}$ para o controle de velocidade e $T_{cp}$ para o controle de posição. Valores pequenos geram uma resposta agressiva, enquanto valores maiores geram maior robustez para o controlador. Em (SKOGESTAD, 2003), é recomendado usar $T_{cv} = L_v$ e $T_{cp} = L_p$ como o melhor compromisso entre agressividade e robustez. No entanto, existe um limite físico mínimo da planta para a constante de tempo de malha fechada desejada. Dessa forma, devido ao atraso de transporte real do motor poder ser dezenas de vezes inferior a constante de tempo em malha aberta, utilizar $T_{cv} = L_v$ e $T_{cp} = L_p$ pode forçar o sistema a uma resposta proporcionalmente muito agressiva. Por isso, quando se trata de motores CC, é melhor relacionar $T_{cv}$ e $T_{cp}$ diretamente a características físicas da máquina, como a respectiva constante de tempo de malha aberta $T_v$ e $T_p$.

# 5 AUTO-AJUSTE DE CONTROLADOR PID

Diferentemente da sintonia automática, o auto-ajuste de controladores ocorre quando o controlador já foi sintonizado e está em operação. Ele tem o objetivo de ressintonizar o controlador quando ocorrem mudanças na planta ou de pontos de operação em um processo não-linear.

Nos últimos anos, foi desenvolvida uma técnica de auto-ajuste baseada em resposta ao degrau e no teorema do valor final, caracterizado por sua simplicidade, baixo custo computacional e robustez a ruído de medição. Por essas características, de inicio, foi explorada sua capacidade de aplicação no campo de sintonia automática, om base em uma sintonia manual inicial. Tal metodologia foi aplicada aos controladores PID SISO para processos estáveis em (VERONESI; VISIOLI, 2009; VERONESI; VISIOLI, 2010b; VERONESI; VISIOLI, 2012), para processos instáveis em (PEREIRA *et al.*, 2018), para processos integradores em (VERONESI; VISIOLI, 2010c; PEREIRA *et al.*, 2018), usando controladores PID em cascata em (VERONESI; VISIOLI, 2011b), para controladores PID de duas entradas e duas saídas (do inglês, *Two Input Two Output* (TITO)) (VERONESI; VISIOLI, 2011a; PEREIRA; TORRICO, 2015; PEREIRA *et al.*, 2017) e para compensadores de atraso de transporte em processos estáveis, integradores e instáveis (NORMEY-RICO *et al.*, 2014; PEREIRA *et al.*, 2016). Desse modo, tal técnica será usada neste trabalho.

Semelhante ao procedimento realizado na sintonia automática, o auto-ajuste consiste de duas etapas: identificação do modelo e sintonia do controlador. Para o auto-ajuste, ao invés de utilizar a resposta do sistema a uma entrada típica, utiliza-se dados de malha fechada. Esse procedimento é descrito ao longo desta seção.

## 5.1 Identificação em malha fechada

Identificação de modelos em malha fechada pode acontecer por meio de lote de dados ou recursivamente a cada período de amostragem. Neste trabalho, usa-se por simplicidade um método baseado em lote de dados.

O método escolhido foi proposto em (VERONESI; VISIOLI, 2010a), é aplicável a controladores PID e é baseado em integrais de sinais e operações algébricas simples. Segundo este método, a identificação é realizada impondo uma mudança tipo degrau na referência $r(t)$ e avaliando a resposta do sistema até atingir o regime permanente. O modelo identificado é de

mesma natureza que os respectivo da sintonia automática, para controle de velocidade é um modelo FOPDT e para o controle de posição é um modelo IFOPDT.

De inicio será descrito o procedimento para o caso estável. O erro $e(t)$ do sistema em malha fechada e sua integral são definidos, respectivamente, por

$$e(t) = A_r - y(t), \tag{5.1}$$

$$I_e = \int_0^\infty e(t)dt, \tag{5.2}$$

onde $A_r$ é a amplitude do degrau na referência.

O ganho estático é então calculado por

$$K_v = \frac{T_i A_r}{K_c I_e}. \tag{5.3}$$

Considerando o sinal $e_u(t)$ e sua integral, respectivamente dados por

$$e_u(t) = Ku(t) - y(t), \tag{5.4}$$

$$I_{eu} = \int_0^\infty e_u(t)dt, \tag{5.5}$$

a soma da constante de tempo e do atraso de transporte é obtida por

$$T_0 = \frac{I_{eu}}{A_r}. \tag{5.6}$$

Segundo (VERONESI; VISIOLI, 2010a), o atraso de transporte $L_v$ pode ser obtido empiricamente como o tempo em que a saída atinge 2% do seu valor final ($y(L_v) = 0.02 \cdot A_r$). Uma outra alternativa mais prática é usar uma banda de ruído $NB$, sendo o atraso o tempo em que a saída se torna maior que a banda de ruído, desse modo $y(L_v) \geq NB$.

De posse do atraso de transporte, a constante de tempo pode ser calculada por

$$T_v = T_0 - L. \tag{5.7}$$

Para o caso IFOPDT desenvolvimento está apresentado a seguir:

A integral do sinal de controle $u(t)$ é calculada a partir de

$$I_u = \int_0^\infty u(t)dt, \tag{5.8}$$

e, assim, o ganho estático é obtido por:

$$K_p = \frac{A_r}{I_u}. \tag{5.9}$$

Considerando o sinal $e_u(t)$ e sua integral, respectivamente dados por

$$e_u(t) = K \int_0^t u(v)dv - y(t), \tag{5.10}$$

$$I_{eu} = \int_0^\infty e_u(t)dt, \tag{5.11}$$

a soma da constante de tempo e do atraso de transporte é obtida por

$$T_0 = \frac{I_{eu}}{A_r}. \tag{5.12}$$

O atraso de transporte $L_p$ é obtido de forma análoga ao experimento da planta estável, assim, pode utilizar uma das duas técnicas empíricas para obter o atraso de transporte $L_p$.

Após estimar o atraso de transporte $L$, pode-se calcular a constante de tempo por

$$T_p = T_0 - L. \tag{5.13}$$

## 5.2 Sintonia do controlador

O método de (SKOGESTAD, 2003) também foi utilizado para o auto-ajuste do controlador PID, para ambos os modos. Nessa seção foi realizado um procedimento análogo ao da seção 4.2, diferindo apenas no modelo identificado usado como base para a sintonia. Portanto, serão utilizadas (4.16) e (4.17) para sintonia de um controlador PI e o auto-ajuste do controlador I-PD,faz-se o uso de (4.19), (4.20), (4.21), (4.25), (4.26) e (4.27). O parâmetro livre $T_c$, constante de tempo equivalente da malha fechada, tem o mesmo valor que o escolhido para a sintonia automática. Utilizando o mesmo $T_c$ tem-se comparações mais justas entre os controladores e, assim, facilitando a avaliação dos desempenhos relativos.

# 6 SISTEMA DIDÁTICO DE IDENTIFICAÇÃO E CONTROLE DE VELOCIDADE E POSIÇÃO DE MOTOR CC

O sistema didático desenvolvido consiste de um hardware baseado em microcontrolador e uma interface gráfica em computador. Como todo sistema de controle, ele contém um sensor (*encoder*) para medição da variável controlada, um controlador embarcado em microcontrolador e comunicação para permitir definição da referência e obtenção dos sinais de entrada e saída do processo.

Para desenvolvimento do *firmware* do microcontrolador e da interface gráfica, utilizou-se um computador com sistema operacional Microsoft Windows 10, com acesso à tecnologia de comunicação sem fio *Bluetooth*®, com os softwares MPLAB X IDE®, PICkit 3 e MATLAB®, além de um gravador de microcontroladores PICkit 3 da Microchip. Para uso do sistema didático é necessário apenas um computador com sistema operacional Microsoft Windows®.

## 6.1 Hardware e firmware

O controlador foi implementado através de *firmware* embarcado em um microcontrolador dsPIC30F4011 da Microchip. A escolha do microcontrolador foi feita em concordância com os conhecimentos prévios dos alunos que irão utilizar o sistema. Além disso, este microcontrolador é voltado a aplicações de controle de motores, possuindo diversos módulos especiais para estas aplicações. Entre os módulos especiais, foram utilizados o módulo de codificador em quadratura de *encoder* diferencial na forma de contador de 16-bit e o módulo de PWM. Além destes, foram utilizados o módulo temporizador, o módulo de comunicação UART e o módulo controlador de portas.

Além do microcontrolador, outros componentes utilizados são descritos a seguir:

- *driver* de motor CC MC33926 *Motor Driver Carrier* da *Pololu Corporation*
- Motor CC: tensão nominal de 12 V, velocidade nominal de 80 rpm, caixa de transmissão 131,25:1, torque em rotor bloqueado de 1,059 Nm e *encoder* em quadratura com resolução de 64 pulsos por revolução;
- Bateria Chumbo Ácido Regulada por Válvula de 12 V e 7 Ah;
- Placa de circuito para uso do dsPIC30F4011 e periféricos
- Módulo *Bluetooth*® HC-06

No *firmware* estão incluídos os algoritmos de comunicação com a interface gráfica,

de identificação (malha aberta e malha fechada), de sintonia e dos controladores PI e I-PD discreto.

O controlador PI embarcado é uma equação de diferenças resultante da expressão (4.18) discretizada pelo método de Tustin. Foi utilizado o método de Tustin também em (4.22) e (4.23), obtendo-se as equações de diferenças do controlador I-PD implementadas no *firmware*. O tempo de amostragem escolhido foi $T_s = 10ms$, pois a partir do modelo obtido por (PEREIRA; REIS, 2014), tem-se que a constante de tempo da planta de controle de velocidade e de posição do motor CC é aproximadamente dez vezes o valor de $T_s$. O atraso de transporte original da planta é ainda menor que $T_s$. Assim, para também se avaliar a identificação deste parâmetro do modelo, foi incluído via *firmware* um acréscimo de atraso de quatro períodos de amostragem.

Foi usado um sistema de unidades internas ao *firmware* para reduzir o custo computacional das operações e evitar conversões de unidades desnecessárias. Assim, usou-se as próprias unidades de medida do sensor e de atuação do microcontrolador via registrador de PWM. Como por exemplo: tensão elétrica é diretamente o valor do registrador do módulo de PWM e a posição é baseada nos pulsos digitais fornecidos pelo *encoder* e obtidos pelo registrador associado. Essa conversão pode ser realizada facilmente por proporção, onde a tensão de 12V equivale a 1000 unidades de registrador do PWM, e para a posição, 8400 pulsos equivalem a 360 graus. Mesmo que para o *firmware* exista esse sistema de unidades, no caso da exibição de valores na interface gráfica, as unidades de posição são convertidas para graus. A unidade de tensão foi mantida como o valor do registrador do módulo PWM. Internamente no *firmware* nos algorítimos para o controle de velocidade utilizou-se como medida de velocidade a quantidade de pulsos de *encoder* por período de amostragem e como entrada do processo o valor do registrador do módulo de PWM. Desse modo, considerando-se valores nominais do motor CC, 80 rpm equivalem a 112 pulsos/$T_s$ e 12 V equivalem a 1000 unidades de registrador do módulo de PWM.

Para operações em baixas tensões com motor CC com caixa de redução no eixo, necessidade indissociável ao controle de posição, surge o efeito de uma não linearidade do tipo zona morta. Esse comportamento altera de forma expressiva a resposta do sistema de controle, e assim, influencia diretamente nos resultados do auto-ajuste do controlador. A zona morta reduz também o desempenho do controlador em ensaios, aumentando o tempo de estabilização e algumas vezes gerando até erro em regime permanente. Na literatura estão descritas diversas técnicas que permitem reduzir o efeito da zona morta. Assim, no *firmware* foi implementada uma técnica de cancelamento que consiste na aplicação de uma inversa da zona morta não-linear

adaptada de (TAO; KOKOTOVIC, 1995) e baseada em exponenciais.

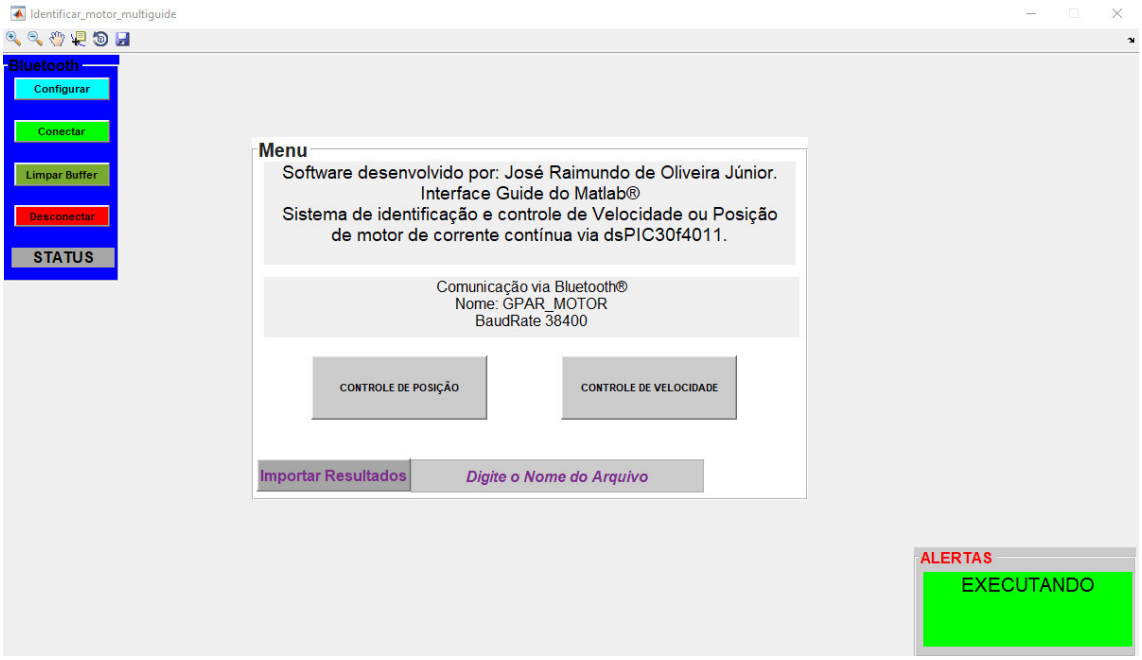
Ainda no *firmware*, foram desenvolvidos algoritmos contendo um padrão e uma estrutura de comunicação com a interface gráfica, permitindo interagir com as diferentes etapas de uso da interface, tais como: identificação (malha aberta e malha fechada), sintonia do controlador PI ou I-PD e ensaios de controle em tempo real. Onde tem-se menu inicial que permite escolher qual experimento será realizado: Controle de velocidade ou Controle de posição.

### *6.1.1 Interface gráfica*

A interface gráfica foi desenvolvida utilizando o Ambiente de Desenvolvimento de Interface Gráfica com Usuário (do inglês, *Graphical User Interface Development Environment* (GUIDE)) disponível no MATLAB®. Ela torna a interação do usuário com o sistema de controle bastante simples, pois permite a utilização completa do sistema sem precisar compreender o protocolo de comunicação, os algoritmos embarcados, a configuração do microcontrolador, o processamento dos dados e etc.

O programa inicia com apenas as ferramentas disponíveis na Fig.12. Trata-se do painel de comunicação Bluetooth, com seus respectivos botões habilitados, e um menu onde é possível definir o objetivo de controle, velocidade ou posição, respectivamente. Além do quadro de avisos, com mensagens de sucesso ou de erro.

Figura 12 – Menu de seleção dos ensaios



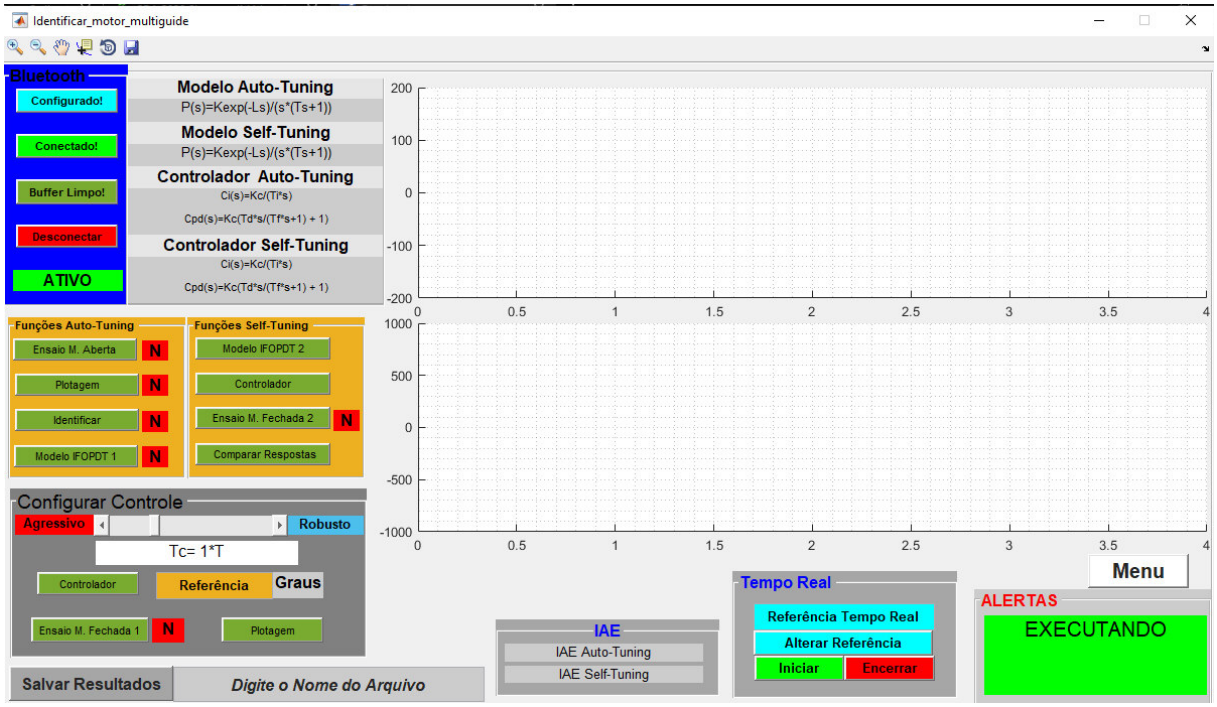

Fonte: Autoria própria.

A Figura 13 mostra o resultado na interface gráfica ao selecionar o botão “Controle de Posição” e tendo configurado o *Bluetooth*®. Ao pressionar o botão “Controle de Velocidade” nas mesmas circunstancias anteriores, tem-se como resultado na interface gráfica a Figura 14. A interface permite alterar o tipo de ensaio sem perder os resultados anteriores, assim, pode-se alternar entre os ensaios de velocidade e posição retornando ao mesmo passo antes da mudança.

Após estabelecida a comunicação com o microcontrolador, obtém-se acesso ao painel Funções Auto-Tuning. Para a opção de controle de velocidade ele comanda o ensaio de resposta ao degrau com o PWM configurado em 66,6%, salvando a resposta de velocidade na memória interna do microcontrolador. A partir da resposta salva são calculados os parâmetros do modelo FOPDT. Por fim, a resposta de velocidade e os parâmetros do modelo são enviados para a interface gráfica, onde é possível visualizar e plotar os gráficos em seus eixos.

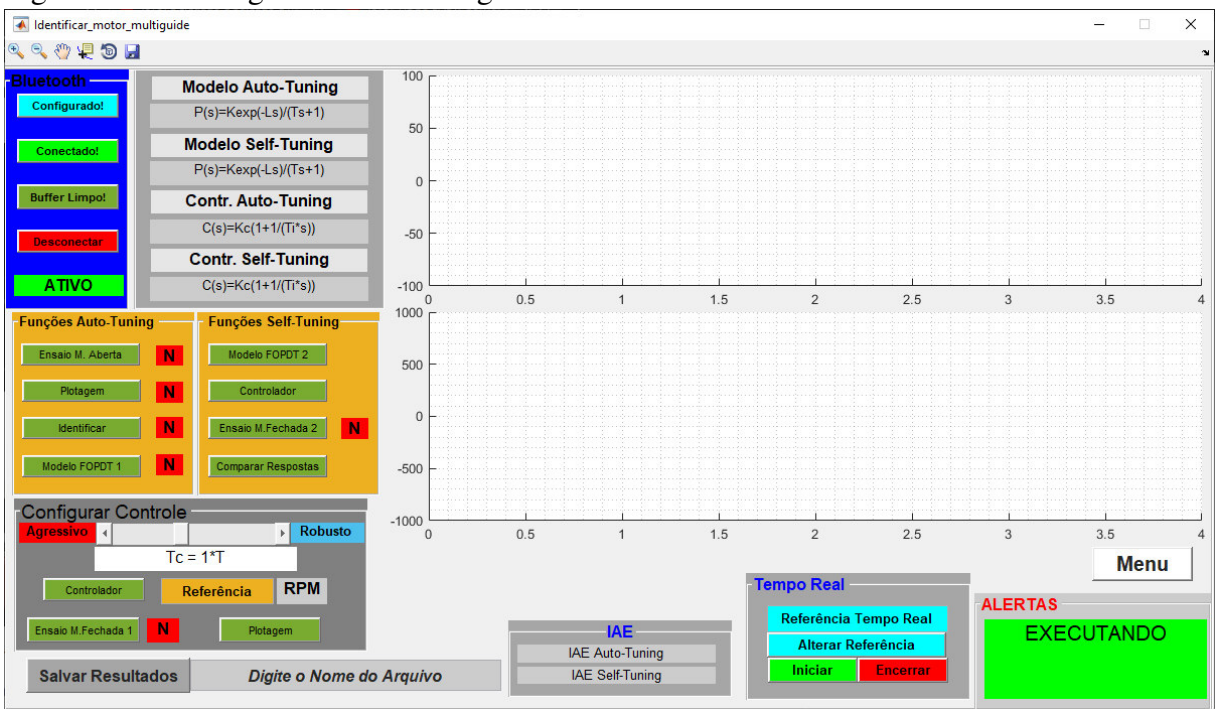
Para a opção de controle de posição o painel Funções Auto-Tuning, permite realizar o ensaio de malha aberta, plotar o resultado dos ensaios, realizar a identificação do modelo IFOPDT usado pelo auto-tuning. No ensaio, o pulso de tensão aplicado tem amplitude $\Delta u$ de 666 unidades de registrador do PWM e a duração $\Delta t_u = 50$ segundos. O microcontrolador sempre salva o ultimo ensaio realizado, tanto o vetor de leituras do *Encoder* quanto o valor de unidades de registrador do PWM aplicada. Os ensaios duram 4 segundos e, por consequência, são 400 amostras de cada vetor de dados.

Figura 13 – Visão geral da interface gráfica: modo posição




Fonte: Autoria própria.

Figura 14 – Visão geral da interface gráfica: modo velocidade




Fonte: Autoria própria.

A partir dos dados salvos, o microcontrolador calcula os parâmetros do modelo FOPDT ou do modelo IFOPDT. A interface gráfica pode solicitar o recebimento dos dados e do modelo, para assim exibi-los nas plotagens e no painel, respectivamente.

Ao concluir a identificação, a interface habilita o painel Configurar Controle. Nele, ajusta-se o valor da constante de tempo de malha fechada desejada $T_c$, que determina o compromisso entre agressividade e robustez da sintonia do controlador. Há um *Slider* que permite a variação entre o valor mínimo de $0,3T$ e máximo de $3T$ para o valor de $T_c$. Uma vez escolhido o valor de $T_c$, o *firmware* sintoniza o respectivo controlador. A referência é escolhida a partir de uma caixa texto e permite valores entre -80 e 80 rpm para a velocidade e -200 e 200 graus para a posição. Antes de enviar a referência para o microcontrolador, a interface gráfica converte seu valor para a unidade de pulsos/$T_s$, para velocidade, segundo a expressão

$$p = \frac{r_{tr} \cdot r_{enc} \cdot T_s}{60} v_{rpm} \tag{6.1}$$

onde $p$ é a quantidade de pulsos/$T_s$, $r_{tr}$ é a relação de transmissão, $r_{enc}$ é a resolução do *encoder*, $v_{rpm}$ é a velocidade em rpm. Para o caso da planta de posição é ainda mais simples e pode ser feita diretamente por proporção. Após a realização do ensaio em malha fechada com o controlador obtido pelo auto-tuning, os sinais de controle e a respectiva resposta são plotados na interface.

Logo após, o painel Funções Self-Tuning é habilitado. Este painel comanda a identificação de um novo modelo utilizando os sinais do ultimo ensaio em malha fechada realizado. A partir deste modelo, é sintonizado um novo controlador e realizado um novo ensaio de malha fechada. Por fim, o painel permite plotar o resultado deste ensaio de malha fechada junto com o ensaio anterior no mesmo gráfico. Além disso, calcula-se a integral do erro absoluto (do inglês, *Integrated Absolute Error* (IAE)) das respostas respectiva a velocidade ou posição, um índice de desempenho para comparação objetiva dos dois resultados para avaliação resultados da sintonia automática e do auto-ajuste.

Também é importante comentar que existe um painel que condensa as informações obtidas no decorrer da operação de toda a interface. Nele consta de forma visual os modelos de planta identificados pela sintonia automática e pelo auto-ajuste e os controladores sintonizados a partir de cada um desses modelos.

Na interface gráfica também consta um sistema de controle sem limite de tempo, o painel Tempo Real. Ele utiliza o último controlador sintonizado para sua operação e permite mudanças na referência a qualquer momento. O ensaio é acompanhado através de plotagens nos eixos da interface, tanto da resposta quando do sinal de controle. Esse sistema atualiza as plotagens a cada segundo, adicionando a cada um dos gráficos as última 100 amostras acumuladas. Esse sistema consistem em um supervisório simples e, no final desse ensaio, a resposta da variável controlada e do sinal de controle são salvos.

Por fim, a interface gráfica permite salvar todos os dados manipulados dentro da interface gráfica. Existe uma caixa de texto, que permite a escolha do nome do arquivo, associada à um botão que, ao pressionar, cria um arquivo de extensão ".mat" na pasta corrente do MATLAB® com os respectivos dados. A partir desse arquivo salvo também é possível carregar dados de um ensaio anterior na interface gráfica, permitindo importar um controlador sintonizado em um ensaio anterior, por exemplo. Foi feita a conversão da estrutura padrão de *handles* em variáveis comuns, para desassociar o arquivo da linguagem orientada a objetos, facilitando o uso por parte dos alunos.

## 6.2 Resultados Experimentais

Será apresentada uma sequência típica de experimentos usando a interface gráfica e com o motor operando sem carga. Como resultados serão apresentados os modelos identificados, os controladores obtidos e índices de desempenho das respectivas respostas de velocidade e

posição. No texto, variáveis com índice 1 referem-se ao auto-tuning e com o índice 2 ao self-tuning.

O primeiro ensaio foi realizado utilizando o painel Funções Auto-Tuning em cada uma das interfaces gráficas. Como pode ser visto para o ensaio de posição na Figura 15 e para o ensaio de velocidade na Figura 16. Os gráficos dos ensaios são, respectivamente, a resposta do sistema e o sinal de controle, ambos ao longo dos 4 segundos de ensaio. O modelo obtido pelo ensaio de resposta ao degrau em malha aberta foi destacado em verde e os parâmetros calculados foram inclusos. O modelo obtido pra o ensaio de posição está transcrito abaixo

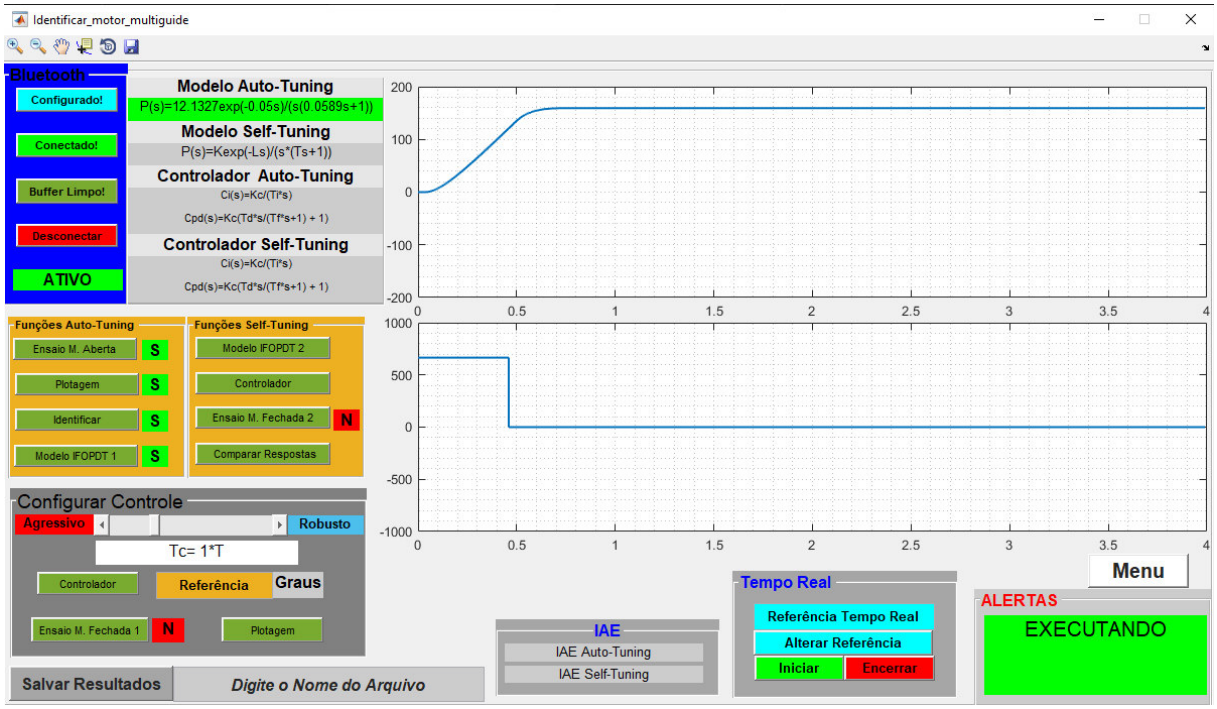
$$P_{1p}(s) = \frac{12,1327}{s\,(0,0589s+1)} e^{-0,05s}, \tag{6.2}$$

e para o ensaio de velocidade

$$P_{1v}(s) = \frac{0.1156}{0.0991s+1} e^{-0.05s}. \tag{6.3}$$

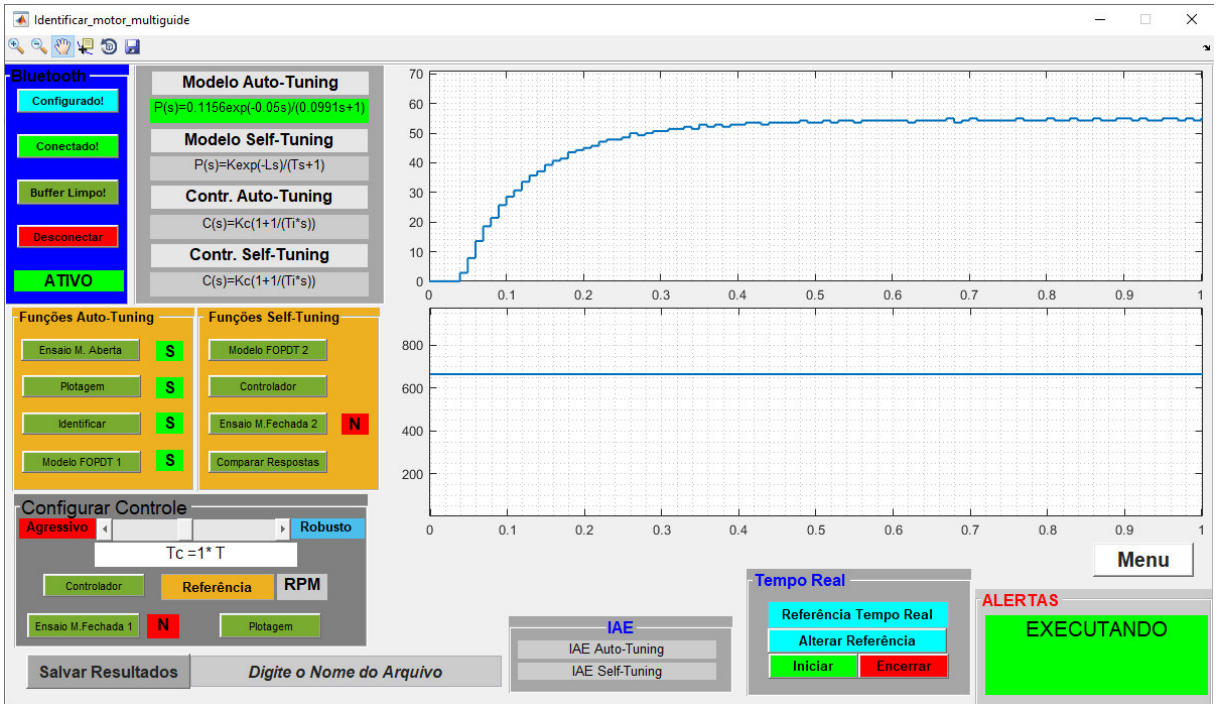
Observa-se também na Figura 15 a resposta de posição em graus e o pulso em unidades de registrador do módulo PWM. De forma análoga a Figura 16 mostra a reposta de velocidade em rpm e o degrau de entrada aplicado em unidades de registrador do módulo PWM. As unidades foram convertidas do padrão utilizado pelo microcontrolador para o padrão que está apresentado na interface gráfica através de operações próprias do código do GUIDE.

Figura 15 – Interface gráfica em ensaio de malha aberta: caso posição




Fonte: Autoria própria.

Figura 16 – Interface gráfica em ensaio de malha aberta: caso velocidade




Fonte: Autoria própria.

Posteriormente, foi utilizado o painel Configurar Controle, onde é definida a constante de tempo de malha fechada $T_c$ em função do valor da constante de tempo de malha aberta $T$. Foi escolhido $T_c = 2T$ para o ensaio de posição e $T_c = 0.8T$ para o ensaio de velocidade. O controlador é então sintonizado usando os parâmetros do respectivo modelo e o valor de $Tc$, sendo apresentado na interface. Os parâmetros calculados para o ensaio de posição, com controlador I-PD foram

$$K_{c1} = 0,5244,\ T_{i1} = 0.7417,\ T_{d1} = 0,0542. \tag{6.4}$$

e para o ensaio de velocidade, com o controlador PI obteve-se

$$K_{c1} = 6.9004,\ T_{i1} = 0.0991. \tag{6.5}$$

Na realização do ensaio em malha fechada, no ensaio de velocidade, foi escolhida a referência de 90 graus, utilizando o controlador recém sintonizado, originário do auto-tuning. Os dados resultantes do ensaio podem ser vistos na Fig. 17 em linhas de cor preta.

No ensaio de velocidade foi escolhida a referência de 40 rpm para o ensaio de malha fechada utilizando o controlador recém sintonizado pelo método de auto-tuning. Os sinais de controle e de velocidade deste ensaio podem ser vistos na Fig. 18 em linha de cor preta.

Dando prosseguimento ao processo, para o ensaio de posição, no painel Funções Self-Tuning, obtém-se um novo modelo IFOPDT baseado na resposta ao degrau de malha fechada anterior. Os resultados foram expostos no respectivo painel, e podem ser verificados na Fig. 17. O modelo encontrado está exposto abaixo:

$$P_{2p}(s) = \frac{11,9051}{s\,(0,0566s+1)} e^{-0,05s}. \tag{6.6}$$

Utilizando este modelo e o $T_c$ anterior, é realizada uma nova sintonia para o controlador I-PD, obtendo-se

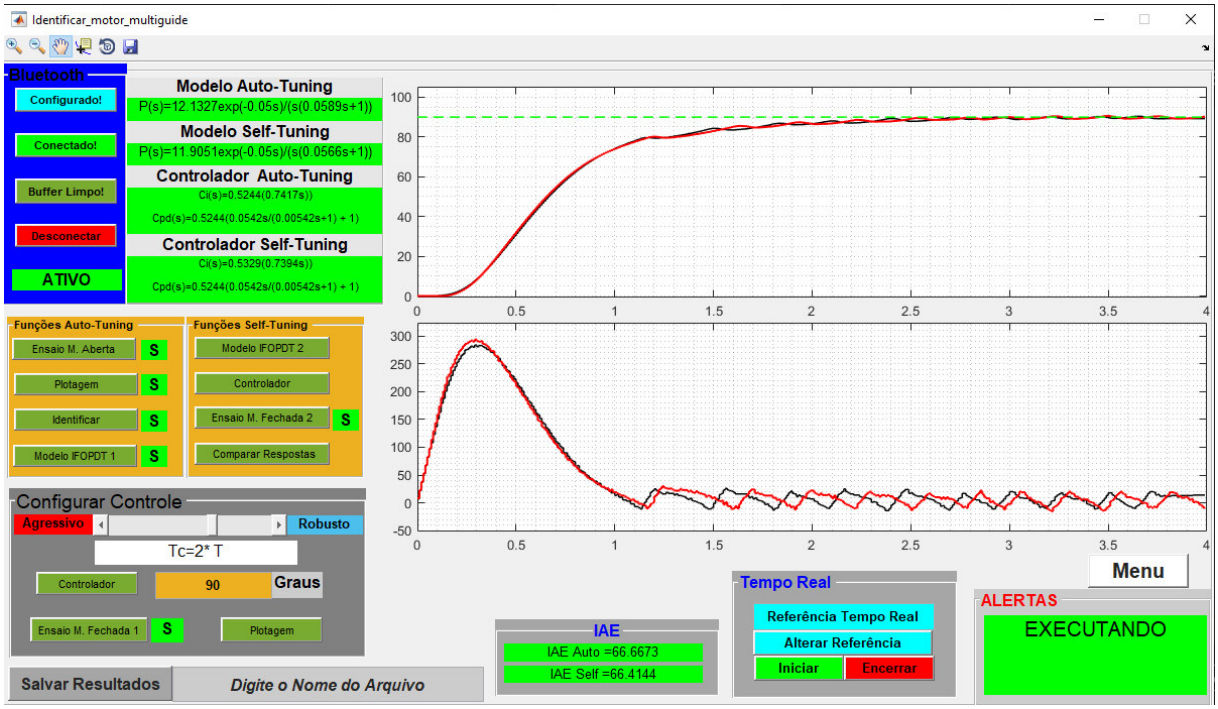
$$K_{c2} = 0,5329,\ T_{i2} = 0,7394,\ T_{d2} = 0,0542. \tag{6.7}$$

Na sequência, é realizado um outro ensaio de malha fechada usando os novos valores (6.7) dos parâmetros do controlador I-PD obtidos pelo método de self-tuning. O resultado foi apresentado na Fig. 17 através das plotagens em vermelho. Ambas respostas dos ensaios de controle de posição são plotadas sobrepostas, destacando a semelhança entre elas.

Por fim, são mostrados na interface gráfica os valores do índice de desempenho IAE de cada reposta em malha fechada:

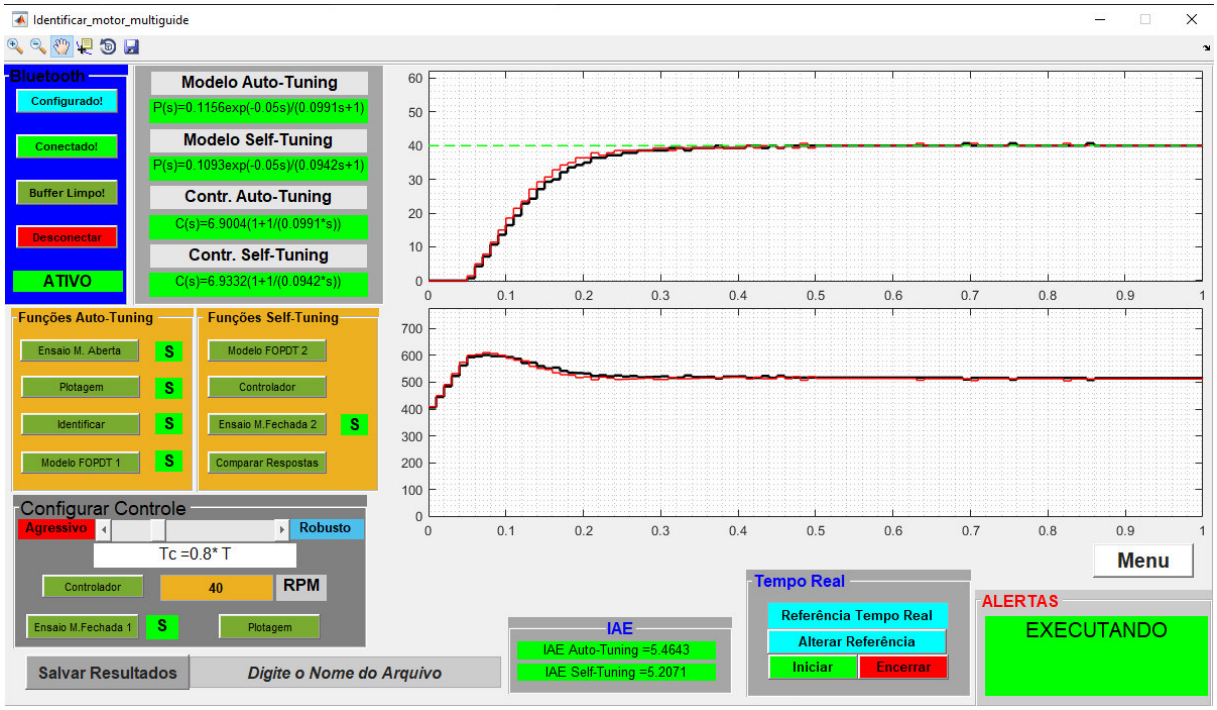
$$IAE_1 = 66,6673,\ IAE_2 = 66,4144. \tag{6.8}$$

Figura 17 – Interface gráfica com resultados de ensaios de auto-tuning e self-tuning: caso posição




Fonte: Autoria própria.

Figura 18 – Interface gráfica com resultados de ensaios de auto-tuning e self-tuning: caso velocidade




Fonte: Autoria própria.

De forma análoga para o ensaio de posição, no painel Funções Self-Tuning, obtém-se um novo modelo baseado na resposta ao degrau de malha fechada anterior. Este modelo pode ser visto na Fig. 18 e é dado por:

$$P_{2v}(s) = \frac{0.1093}{0.0942s+1}e^{-0.05s}. \tag{6.9}$$

Baseado no modelo acima, é realizada uma nova sintonia para o controlador PI, obtendo-se

$$K_{c2} = 6.9332,\ T_{i2} = 0.0942. \tag{6.10}$$

Assim, é realizado um outro ensaio de malha fechada usando os parâmetros (6.10) do controlador PI calculados pelo método de self-tuning. Seus resultados são mostrados na Fig. 18, onde as respostas de velocidade e sinal de controle estão em linha de cor vermelha.

Ao final dos experimentos, são mostrados na interface gráfica os valores de IAE de cada reposta em malha fechada:

$$IAE_1 = 5.4643,\ IAE_2 = 5.2071. \tag{6.11}$$

## 6.3 Discussão

Didaticamente, como se tratam de técnicas com abordagens diferentes de identificação (malha aberta e malha fechada), o objetivo do experimento não é determinar qual delas é a

melhor. O objetivo é mostrar aos alunos que as técnicas de auto-tuning e self-tuning utilizadas produzem resultados semelhantes. Isso pode ser constatado observando os resultados em termos dos parâmetros dos modelos identificados, dos controladores obtidos, das respostas dos ensaios e dos índices de desempenho, que são muito similares para ambos métodos. Portanto, educacionalmente, é importante ressaltar aos estudantes que estas técnicas podem, na verdade, ser utilizadas em conjunto, em diferente momentos para o sistema de controle. Como, por exemplo, o auto-tuning pode ser usado no momento de comissionamento de uma nova planta, enquanto o self-tuning durante a operação de uma planta já existente.

Mesmo utilizando mecanismos para reduzir a influência da zona morta na resposta, é perceptível que seu efeito ainda existe no ensaio de controle de posição. Um desses efeitos é a oscilação em torno da referência nos ensaios de malha fechada, porém tem-se que a amplitude das oscilações se reduzem ao longo do tempo e a resposta converge para referência e, assim, em um tempo suficientemente longo não há erro em regime permanente. O método de identificação que utiliza a resposta de malha fechada, obteve um modelo bastante semelhante ao método que utiliza a resposta de malha aberta. Desse modo, confirma-se a utilidade da técnica de correção da zona morta e a robustez do método do self-tuning, com relação à certos níveis de não linearidades nas respostas.

Didaticamente, o objetivo é apresentar aos alunos ambas técnicas de identificação e mostrar que seus resultados são semelhantes, mesmo consistindo de técnicas diferentes. A plotagem sobreposta, os parâmetros dos modelos obtidos e o cálculo dos índices de desempenho são fundamentais para justificar essa conjectura.

# 7 CONCLUSÕES E TRABALHOS FUTUROS

O sistema didático desenvolvido torna simples a realização de experimentos de sintonia automática e auto-ajuste de controlador PID em uma planta de controle de velocidade de motor CC. Ele condensa várias informações de um sistema de controle em uma única ferramenta. Além disso, o sistema didático dá aos alunos uma dimensão da interdisciplinaridade que envolvem as aplicações de controle.

O sistema didático desenvolvido mostra diretamente um sistema de controle constituído com todos os elementos fundamentais, permitindo ao aluno visualizar a interdisciplinariedade envolvida e as adaptações indissociáveis da implementação prática. Além disso, a partir da interface gráfica, tem-se a unificação das informações, do comando do sistema e da visualização dos resultados. Dessa forma, esse sistema proporciona aos alunos um ambiente que permite reforçar seus conceitos e comprovar a teoria apresentada em sala.

O conjunto de dados obtido pela interface gráfica pode ser salvo, permitindo a reutilização desses dados didaticamente. Assim, cada etapa dos processos de sintonia automática e auto-ajuste pode ser reproduzida pelo aluno por meio de simulações. Isso torna o sistema didático uma ferramenta eficaz no ensino e na consolidação de conceitos básicos da teoria de controle, como identificação de sistemas e sintonia de controladores.

Mesmo em uma planta com a presença de não linearidades, ao aplicar a técnica de inversão da zona morta a resposta obtida foi essencialmente linear. Ambos os métodos de identificação aplicados são sensíveis a respostas não lineares, e assim, as identificações realizadas com sucesso corroboram com a importância da correção da zona morta para os resultados alcançados.

Como trabalho futuro, sugere-se que seja construído um sistema de carga variável acoplado ao eixo do motor. Desse modo, permitindo aos alunos obter diferentes modelos tanto em malha aberta como em malha fechada. Além disso, torna possível perceber a deterioração das respostas de malha fechada após a mudança da planta e também as vantagens da aplicação de self-tuning como solução para o problema. Outra importante sugestão de trabalho é o desenvolvimento de uma técnica de inversa da zona morta adaptativa para, assim, permitir a correção automática em sistemas embarcados na presença dessa não linearidade.

Adicionalmente uma importante melhoria seria a implementação de uma nova interface gráfica usando ferramentas de software livre, como as baseadas em linguagem de programação Python. Dessa forma reduzindo os custos do projeto e permitindo aos alunos o uso e

o desenvolvimento de forma independente. Outro importante avanço está na implementações online desse sistema, permitindo que a interface gráfica esteja disponível online e que a planta permita um acesso remoto, e dessa forma, contribuindo para a acessibilidade e para o aprendizado em geral.

# REFERÊNCIAS

ÅSTRÖM, K. J.; WITTENMARK, B. On self tuning regulators. **Automatica**, v. 9, n. 2, p. 185 – 199, 1973.

BENCOMO, S. D. **Control learning: Present and future**. [S.l.]: IFAC, 2002. v. 15. 71–93 p.

BERNER, J.; SOLTESZ, K.; HÄGGLUND, T.; ÅSTRÖM, K. J. An experimental comparison of PID autotuners. **Control Engineering Practice**, Elsevier Ltd, v. 73, n. February, p. 124–133, 2018.

BISSELL, C. C. Control Education: Time for Radical Change? **IEEE Control Systems**, v. 19, n. 5, p. 44–49, 1999.

BRAATZ, R. D. Teaching Mathematics to Control Engineers [Focus on Education]. **IEEE Control Systems**, v. 33, n. 3, p. 66–67, 2013.

CHAPMAN, S. J. **Electric Machinery Fundamentals**. New York: he McGraw-Hill Companies, 2012. 698 p.

David Halliday, Robert Resnick, J. W. **Fundamentos de Física**. [S.l.: s.n.], 2016. 812 p.

GIERAS, J. F.; WING, M. **Permanent magnet motor technology: design and applications**. 2nd ed., r. ed. [S.l.]: Marcel Dekker, 2002. 611 p.

GUZMAN, J. L.; COSTA-CASTELLO, R.; DORMIDO, S.; BERENGUEL, M. An Interactivity-Based Methodology to Support Control Education: How to Teach and Learn Using Simple Interactive Tools [Lecture Notes]. **IEEE Control Systems**, v. 36, n. 1, p. 63–76, 2016.

HUGHES, A. **Electric Motors and Drives**. Burlington: Elsevier, 2006. 431 p.

KALMAN, R. E. Design of self-optimizing control system. **Trans. ASME**, v. 80, p. 468–478, 1958.

MICROCHIP TECHNOLOGY. **dsPIC30F4011/4012 Data Sheet**. [S.l.], 2005.

NISE, N. S. **CONTROL SYSTEMS ENGINEERING**. Pomona: Wiley, 2015.

NISHIKAWA, Y.; SANNOMIYA, N.; OHTA, T.; TANAKA, H. A Method for Auto-tuning of PID Control Parameters. **IFAC Proceedings Volumes**, Elsevier, v. 20, n. 8, p. 321–332, 1984.

NORMEY-RICO, J. E.; SARTORI, R.; VERONESI, M.; VISIOLI, A. An automatic tuning methodology for a unified dead-time compensator. **Control Engineering Practice**, Elsevier, v. 27, n. 1, p. 11–22, 2014.

OHTA, T.; TANAKA, H.; TANAKA, K.; SANNOMIYA, N.; NISHIKAWA, Y. A New Optimization Method of PID Control Parameters for Automatic Tuning by Process Computer. **IFAC Proceedings Volumes**, Elsevier, v. 12, n. 7, p. 133–138, 1980.

PEREIRA, R. D. O.; ANDRADE, F. V.; TORRICO, B. C.; CORREIA, W. B. Automatic tuning of a dead-time compensator for stable, integrative and unstable processes. In: **2016 IEEE Biennial Congress of Argentina**. Buenos Aires, ARG: [s.n.], 2016.

PEREIRA, R. D. O.; CORREIA, W. B.; NOGUEIRA, F. G.; TORRICO, B. C. Automatic tuning method for PID controllers applied to integrating and unstable processes. In: **2018 13th IEEE International Conference on Industry Applications (INDUSCON)**. [S.l.: s.n.], 2018. p. 744–749.

PEREIRA, R. D. O.; REIS, L. L. N. dos. Avaliação de desempenho de controladores PID 2-DOF em controle de velocidade de um motor CC. In: **Anais do XX Congresso Brasileiro de Automática**. Belo Horizonte: [s.n.], 2014. p. 4151–4158. Disponível em: <http://www.swge.inf.br/cba2014/anais/PDF/1569935307.pdf>.

PEREIRA, R. D. O.; TORRICO, B. C. New automatic tuning of multivariable PID controller applied to a neonatal incubator. In: **8th International Conference on Biomedical Engineering and Informatics (BMEI)**. Shenyang: [s.n.], 2015.

PEREIRA, R. D. O.; VERONESI, M.; VISIOLI, A.; NORMEY-RICO, J. E.; TORRICO, B. C. Implementation and test of a new autotuning method for PID controllers of TITO processes. **Control Engineering Practice**, v. 58, p. 171 – 185, 2017.

SADIKU, M. N. **Elementos de Eletromagnetismo**. [S.l.]: Bookman, 2013. 716 p.

SALCEDO, C. M. **Modelo de Controlador com Compensação de Zona Morta para uma Base Móvel de Robôs**. 119 p. Tese (Doutorado) — Universidade Federal da Bahia, 2010.

SKOGESTAD, S. Simple analytic rules for model reduction and PID controller tuning. **Journal of Process Control**, Elsevier, v. 13, n. 4, p. 291–309, jun 2003. Disponível em: <http://www.mic-journal.no/ABS/MIC-2004-2-2.asp>.

TAO, G.; KOKOTOVIC, P. V. Adaptive Control of Plants with Unknown Hystereses. **IEEE Transactions on Automatic Control**, v. 40, n. 2, p. 200–212, 1995.

UMANS, S. D. **Fitzgerald & Kingsley's Electric Machinery.** [S.l.: s.n.], 2014. 724 p.

VERONESI, M.; VISIOLI, A. Performance assessment and retuning of PID controllers. **Industrial and Engineering Chemistry Research**, v. 48, p. 2616–2623, 2009.

VERONESI, M.; VISIOLI, A. An industrial application of a performance assessment and retuning technique for PI controllers. **ISA Transactions**, Elsevier Ltd, v. 49, n. 2, p. 244–248, 2010.

VERONESI, M.; VISIOLI, A. An industrial application of a performance assessment and retuning technique for PI controllers. **ISA Transactions**, v. 49, p. 244–248, 2010.

VERONESI, M.; VISIOLI, A. Performance assessment and retuning of PID controllers for integral processes. **Journal of Process Control**, v. 20, p. 261–269, 2010.

VERONESI, M.; VISIOLI, A. An automatic tuning method for multiloop PID controllers. In: **Preprints 18th IFAC World Congress**. Milan, ITA: [s.n.], 2011.

VERONESI, M.; VISIOLI, A. A simultaneous closed-loop automatic tuning method for cascade controllers. **IET Control Theory and Applications**, v. 5, p. 263–270, Januar 2011.

VERONESI, M.; VISIOLI, A. Performance assessment and retuning of pid controllers for load disturbance rejection. In: **Preprints IFAC Conference on Advances in PID Control**. Brescia, ITA: [s.n.], 2012.

# ANEXO A – CÓDIGO DO *FIRMWARE* EMBARCADO

```
#include "config.h"
#include "reloj.h"
#include <stdio.h>
#include <libpic30.h>
#include <math.h>
#include <xc.h>
                    //Declarar variavel global
 unsigned int n;
 unsigned int i;
 unsigned int x=0;
 unsigned int y=0;
 unsigned char dado=2;
 unsigned char Operacao=0;
 unsigned char nr=0;
 unsigned char Modo_atual;
 float Entrada,Entrada_ant;
 int Saida_Int[400];
 int Enviar_int=0;;
 float u_control,v_control;
 int u_int[400];
 float Saida_R,u_R; //Variaveis do C.infinity
 float Reg_Morto;
 unsigned char Parar_ensaio=0;
 unsigned char Variador=10;
 unsigned long int Recebimento[4];
 unsigned long int Int_Recebimento;
          //Variaveis da função Float p/ int32
 float Conv_Float;
 double Conv_Double;
 long int Conv_Long_Int;
           //Variaveis para o controle
 float Ts=0.01;
 unsigned int uf;
 float ut=0;
 float yf;
 float K,L,T;
```

```
37   float K_MF,L_MF,T_MF,T0_MF,Soma_aux;
38   float m0,yT0,Sy,Su,di,error,T,g,test,uff;
39   float A0;
40   float T0;
41   float A1;
42   float Pd;
43   float Tc,Tc_MF;
44   float Ti;
45   float Kc;
46   float Kc_MF,Ti_MF;
47   float u_i,u_i_ant;
48   float u_pd,u_pd_ant;
49   float Saida_ant;
50   float Tf,Td;
51   float Kcs,Tis,Tds;
52   float nll,br2,nrr,br1,fll,bl2,frr,bl1,mf;
53                  //Fim da declaração de variaveis
54  //Declarando as funções
55  void config_PWM(void);
56  void IDENTIFICAR_V(void);
57  void IDENTIFICAR(void);
58  void IDENTIFICAR_SELF_V(void);
59  void IDENTIFICAR_SELF(void);
60  void ENCODER(void);
61  void ENSAIO_V(void);
62  void ENSAIO(void);
63  void ENVIAR_SAIDA(void);
64  void Banda_Morta(void);
65  void ENVIAR_U(void);
66  void CONTROLE_PI_V(void);
67  void CONTROLE_PI_T_Infinity_V(void);
68  void CONTROLE_PI(void);
69  void CONTROLE_PI_T_Infinity(void);
70  void config_UART(void);
71  void config_TIMER1(void);
72  void config_TIMER2(void);
73  void config_TIMER3(void);
74  void config_TIMER4(void);
75  void config_TIMER5(void);
```

```
76  void Enviar_Float (float Conv_Float);
77  void config_Output_Compare(void);
78
79  //CONDIGO PRINCIPAL
80  int main(void)
81  {
82                  //CONFIGURANDO OS MODULOS
83  config_PWM();
84  ENCODER();
85  config_UART();
86  config_TIMER1();
87  config_TIMER2();
88  config_TIMER3();
89  config_TIMER4();
90  config_TIMER5();
91  config_Output_Compare();
92  //----------------Ciclo infinito----------------------------------------
93      while(1)
94      {
95          if(dado==3)
96          {Modo_atual='V'; //Ciclo do controle de velocidade
97          dado=2;
98          }
99          if(dado==4)
100         {Modo_atual='P'; //Ciclo do controle de posição
101         dado=2;
102         }
103
104         while (Modo_atual=='P') //Ao escolher posição
105         {
106                 if(dado==3)
107                 {Modo_atual='V';
108                 dado=2;
109                 }
110                 if(dado==4)
111                 {Modo_atual='P';
112                 dado=2;
113                 }
114         if(dado==5)//Atualizando referência
```

```
115             {
116                 Recebimento[1]=Recebimento[1];
117                 Recebimento[2]=Recebimento[2]<<8;
118                 Recebimento[3]=Recebimento[3]<<16;
119                 Recebimento[4]=Recebimento[4]<<24;
120 Int_Recebimento=Recebimento[1]+Recebimento[2]+Recebimento[3]+Recebimento[4];
121                 Entrada=((float)((long int)Int_Recebimento));
122                 dado=2;
123             }
124         if(dado==6)//Atualizando Tc
125             {
126                 Recebimento[1]=Recebimento[1];
127                 Recebimento[2]=Recebimento[2]<<8;
128                 Recebimento[3]=Recebimento[3]<<16;
129                 Recebimento[4]=Recebimento[4]<<24;
130 Int_Recebimento=Recebimento[1]+Recebimento[2]+Recebimento[3]+Recebimento[4];
131                 Tc=((float)((long int)Int_Recebimento));
132                 Tc=Tc/10000;
133                 dado=2;
134             }
135         if(dado==10)//Direção de giro do eixo
136             {
137                 LATDbits.LATD0=1;
138                 dado=2;
139             }
140         if(dado==11)//Direção de giro do eixo
141             {
142                 LATDbits.LATD0=0;
143                 dado=2;
144             }
145         if(dado==12)//Ligar o driver
146             {
147                 LATDbits.LATD2=1;
148                 dado=2;
149             }
150         if(dado==13)//Desligar o driver
151             {
152                 LATDbits.LATD2=0;
153                 dado=2;
```

```
154             }
155             if(dado==51)//Ensaio auto-tuning
156             {
157                 POSCNT=0;
158                 y=0;
159                 Operacao=51;
160                 Saida_Int[1]=0;
161                 Saida_Int[2]=0;
162                 Saida_Int[3]=0;
163                 Saida_Int[4]=0;
164                 T1CONbits.TON = 1;
165                 //configurando banda morta
166                 frr=60;
167                 fll=-60;
168                 bl1=-170;//-130
169                 br1=170;//120
170                 bl2=-85;//-170
171                 br2=85;//160
172                 nrr=(br1-(frr+br2));
173                 nll=nrr;
174                 mf=10;
175                 //configuração termina
176                 uf=666;
177                 if (uf > frr)
178                 {PDC1=(int)(floor(uf+br2)+0.5);}
179                 if (uf <= frr)
180                 { PDC1= (int)(floor(-nr*uf+br1)+0.5);}
181                 //uf=(int)(Entrada);
182                 //PDC1=(Reg_Morto+uf*(1000-Reg_Morto)/100);
183                 dado=2;
184             }
185         if(dado==52)//Envia vetor de respostas
186             {
187                 Operacao=52;
188                 x=0;
189                 T1CONbits.TON = 1;
190                 dado=2;
191             }
192         if(dado==53)//Calcula e introduz correção de ZM
```

```
193             {
194                 POSCNT=0;
195                 PDC1=200;
196                 n=0;
197                 Operacao=53;
198                 T1CONbits.TON = 1;
199                 dado=2;
200             }
201         if(dado==61)//Realiza a identificação auto-tuning
202             {
203                 IDENTIFICAR();
204                 dado=2;
205             }
206         if(dado==62)//Envia o Resultado da identificação
207             {
208                 __delay_ms(100);
209                 Enviar_Float(K);
210                 __delay_ms(100);
211                 Enviar_Float(T);
212                 __delay_ms(100);
213                 Enviar_Float(L);
214                 dado=2;
215             }
216         if(dado==71)
217             {
218                 //-----------------INICIALIZAR-CONTROLE-------------------------
219                 Saida_ant=0;
220                 u_i_ant=0;
221                 u_pd_ant=0;
222                 Entrada_ant=0;
223                 POSCNT=0;
224                 n=0;
225                 Saida_Int[1]=0;
226                 Saida_Int[2]=0;
227                 Saida_Int[3]=0;
228                 Saida_Int[4]=0;
229             T3CONbits.TON = 1;
230                 dado=2;
231             }
```

```
232         if(dado==72)//Envia o sinal de controle
233             {
234                 Operacao=72;
235                 x=0;
236                 T1CONbits.TON = 1;
237                 dado=2;
238             }
239         if(dado==73)//Sintoniza o controlador
240             {
241                 //controlador serie
242                  //Definindo Ti=(4*(Tc+L))
243                 Tis=(4*(Tc+L));
244                 //Kc = 1/(K*(Tc+L))
245                 Kcs=(1/(K*(Tc+L)));
246                 Tds=T;
247                 //controlador paralelo
248                 //Kc = Kcs*(1+Tds/Tis)
249                 Kc=Kcs*(1+(Tds/Tis));
250                 //Ti = Tis*(1+Tds/Tis)
251                 Ti=Tis*(1+(Tds/Tis));
252                 //Td = Tds/(1+Tds/Tis)
253                 Td=Tds/(1+(Tds/Tis));
254                 Tf=Td/10;
255                 //enviar valores
256                 __delay_ms(100);
257                 Enviar_Float(Ti);
258                 __delay_ms(100);
259                 Enviar_Float(Kc);
260                 __delay_ms(100);
261                 Enviar_Float(Td);
262                 dado=2;
263             }
264         if(dado==81)
265             {
266                //---------------------------------CONTROLE----------------------
267                 //-----------------INICIALIZAR-CONTROLE-------------------------
268                 Saida_ant=0;
269                 u_i_ant=0;
270                 u_pd_ant=0;
```

```
271                 Entrada_ant=0;
272                 POSCNT=0;
273                 n=0;
274                 Saida_Int[1]=0;
275                 Saida_Int[2]=0;
276                 Saida_Int[3]=0;
277                 Saida_Int[4]=0;
278                 Parar_ensaio=0;
279             T2CONbits.TON = 1;
280             dado=2;
281             }
282         if(dado==82)//Comanda a pausa no ensaio
283             {
284             Parar_ensaio=1;
285             dado=2;
286             }
287
288         if(dado==161)//Resultado da identificação self-tuning
289         {
290             IDENTIFICAR_SELF();
291             __delay_ms(100);
292                 Enviar_Float(K_MF);
293             __delay_ms(100);
294             Enviar_Float(T_MF);
295             __delay_ms(100);
296             Enviar_Float(T0_MF);
297             __delay_ms(100);
298             Enviar_Float(L_MF);
299                 dado=2;
300         }
301         if(dado==171)
302             {
303                 //---------------------------------CONTROLE--------------------
304                 Saida_ant=0;
305                 u_i_ant=0;
306                 u_pd_ant=0;
307                 Entrada_ant=0;
308                 POSCNT=0;
309                 n=0;
```

```
310                 Saida_Int[1]=0;
311                 Saida_Int[2]=0;
312                 Saida_Int[3]=0;
313                 Saida_Int[4]=0;
314                 Kc=Kc_MF;
315                 Ti=Ti_MF;
316             T3CONbits.TON = 1;
317                 dado=2;
318             }
319         if(dado==173)
320         {
321                  //controlador I-PD
322                 //controlador serie
323                  //Definindo Ti=(4*(Tc+L))
324                 Tis=(4*(Tc+L_MF));
325                 //Kc = 1/(K*(Tc+L))
326                 Kcs=(1/(K_MF*(Tc+L_MF)));
327                 Tds=T_MF;
328                 //controlador paralelo
329                 //Kc = Kcs*(1+Tds/Tis)
330                 Kc_MF=Kcs*(1+(Tds/Tis));
331                 //Ti = Tis*(1+Tds/Tis)
332                 Ti_MF=Tis*(1+(Tds/Tis));
333                 //Td = Tds/(1+Tds/Tis)
334                 Td=Tds/(1+(Tds/Tis));
335                 Tf=Td/10;
336                 //enviar valores
337                 __delay_ms(100);
338                 Enviar_Float(Ti_MF);
339                 __delay_ms(100);
340                 Enviar_Float(Kc_MF);
341                 __delay_ms(100);
342                 Enviar_Float(Td);
343                 dado=2;
344         }
345     }
346     //Os comandos de posição são analogos aos de velocidade
347     //mantendo até a mesma sequência, assim basta consultar
348         while (Modo_atual=='V')//Escolheu velocidade
```

```
349         {
350                 if(dado==3)//permite alterar a escolha
351                 {Modo_atual='V';
352                 dado=2;
353                 }
354                 if(dado==4)//permite alterar a escolha
355                 {Modo_atual='P';
356                 dado=2;
357                 }
358         if(dado==5)
359             {
360                 Recebimento[1]=Recebimento[1];
361                 Recebimento[2]=Recebimento[2]<<8;
362                 Recebimento[3]=Recebimento[3]<<16;
363                 Recebimento[4]=Recebimento[4]<<24;
364 Int_Recebimento=Recebimento[1]+Recebimento[2]+Recebimento[3]+Recebimento[4];
365                 Entrada=((float)((long int)Int_Recebimento));
366                 dado=2;
367             }
368         if(dado==6)
369             {
370                 Recebimento[1]=Recebimento[1];
371                 Recebimento[2]=Recebimento[2]<<8;
372                 Recebimento[3]=Recebimento[3]<<16;
373                 Recebimento[4]=Recebimento[4]<<24;
374 Int_Recebimento=Recebimento[1]+Recebimento[2]+Recebimento[3]+Recebimento[4];
375                 Tc=((float)((long int)Int_Recebimento));
376                 Tc=Tc/10000;
377                 dado=2;
378             }
379         if(dado==10)
380             {
381                 LATDbits.LATD0=1;
382                 dado=2;
383             }
384         if(dado==11)
385             {
386                 LATDbits.LATD0=0;
387                 dado=2;
```

```
388             }
389         if(dado==12)
390             {
391                 LATDbits.LATD2=1;
392                 dado=2;
393             }
394         if(dado==13)
395             {
396                 LATDbits.LATD2=0;
397                 dado=2;
398             }
399              if(dado==51)
400             {
401                 POSCNT=0;
402                 y=0;
403                 Operacao=151;
404                 Saida_Int[1]=0;
405                 Saida_Int[2]=0;
406                 Saida_Int[3]=0;
407                 Saida_Int[4]=0;
408                 T1CONbits.TON = 1;
409                 uf=666;
410                 PDC1= (float)(uf);
411                 dado=2;
412             }
413         if(dado==52)
414             {
415                 Operacao=52;
416                 x=0;
417                 T1CONbits.TON = 1;
418                 dado=2;
419             }
420         if(dado==53)
421             {
422                 POSCNT=0;
423                 PDC1=0;
424                 Operacao=53;
425                 T1CONbits.TON = 1;
426                 dado=2;
```

```
427             }
428         if(dado==61)
429             {
430                 IDENTIFICAR_V();
431                 dado=2;
432             }
433         if(dado==62)
434             {
435                 __delay_ms(100);
436                 Enviar_Float(K);
437                 __delay_ms(100);
438                 Enviar_Float(T);
439                 __delay_ms(100);
440                 Enviar_Float(L);
441                 dado=2;
442             }
443         if(dado==71)
444             {
445                 //-----------------INICIALIZAR-CONTROLE-------------------------
446                 Saida_ant=0;
447                 u_i_ant=0;
448                 u_pd_ant=0;
449                 Entrada_ant=0;
450                 POSCNT=0;
451                 n=0;
452                 Saida_Int[1]=0;
453                 Saida_Int[2]=0;
454                 Saida_Int[3]=0;
455                 Saida_Int[4]=0;
456             T5CONbits.TON = 1;
457                 dado=2;
458             }
459         if(dado==72)
460             {
461                 Operacao=72;
462                 T1CONbits.TON = 1;
463                 dado=2;
464             }
465         if(dado==73)
```

```
466             {
467                  //Definindo Ti, o menor dos dois valores T ou (4*(Tc+L))
468                 if (T>=(4*(Tc+L)))
469                 {Ti=(4*(Tc+L));}
470                 else
471                 {Ti=T;}
472                 Kc=(T/(K*(Tc+L)));
473                 //enviar valores
474                 __delay_ms(100);
475                 Enviar_Float(Ti);
476                 __delay_ms(100);
477                 Enviar_Float(Kc);
478                 dado=2;
479             }
480         if(dado==81)
481             {
482                //----------------------------------CONTROLE----------------------
483                 //-----------------INICIALIZAR-CONTROLE-------------------------
484                 Saida_ant=0;
485                 u_i_ant=0;
486                 u_pd_ant=0;
487                 Entrada_ant=0;
488                 POSCNT=0;
489                 n=0;
490                 Saida_Int[1]=0;
491                 Saida_Int[2]=0;
492                 Saida_Int[3]=0;
493                 Saida_Int[4]=0;
494                 Parar_ensaio=0;
495             T4CONbits.TON = 1;
496             dado=2;
497             }
498         if(dado==82)
499             {
500             Parar_ensaio=1;
501             dado=2;
502             }
503
504         if(dado==161)
```

```
505         {
506             IDENTIFICAR_SELF_V();
507             __delay_ms(100);
508                 Enviar_Float(K_MF);
509             __delay_ms(100);
510             Enviar_Float(T_MF);
511             __delay_ms(100);
512             Enviar_Float(TO_MF);
513             __delay_ms(100);
514             Enviar_Float(L_MF);
515                 dado=2;
516         }
517
518         if(dado==171)
519             {
520                 //---------------------------------CONTROLE--------------------
521                 Saida_ant=0;
522                 u_i_ant=0;
523                 u_pd_ant=0;
524                 Entrada_ant=0;
525                 POSCNT=0;
526                 n=0;
527                 Saida_Int[1]=0;
528                 Saida_Int[2]=0;
529                 Saida_Int[3]=0;
530                 Saida_Int[4]=0;
531                 Kc=Kc_MF;
532                 Ti=Ti_MF;
533             T5CONbits.TON = 1;
534                 dado=2;
535             }
536
537         if(dado==173)
538         {
539                         //Definindo Ti, o menor dos dois valores T ou (4*(Tc+L))
540                 if (T_MF>=(4*(Tc+L_MF)))
541                 {Ti_MF=(4*(Tc+L_MF));}
542                 else
543                 {Ti_MF=T_MF;}
```

```
544                 Kc_MF=(T_MF/(K_MF*(Tc+L_MF)));
545                 __delay_ms(100);
546                 Enviar_Float(Ti_MF);
547                 __delay_ms(100);
548                 Enviar_Float(Kc_MF);
549                 dado=2;
550         }
551         }}
552     return 0;
553     }
554 //-----------------------------Definindo as funções----------------------------
555 //Função para enviar int32 4 casas
556 void Enviar_Float (float Conv_Float)
557  {
558     Conv_Double=(double)Conv_Float;
559     Conv_Double=floor(Conv_Double*10000);
560     Conv_Long_Int=(long int)Conv_Double;
561     U1TXREG=Conv_Long_Int;
562     U1TXREG=Conv_Long_Int>>8;
563     U1TXREG=Conv_Long_Int>>16;
564     U1TXREG=Conv_Long_Int>>24;
565  }
566 //---------------------------Declarando o PWM---------------------------------
567 void config_PWM(void)
568 {
569     PTCONbits.PTEN=1;          //PWM time base is ON
570     PTCONbits.PTSIDL=0;        //PWM time base runs in CPU Idle mode
571     PTCONbits.PTOPS=0b0000;//PWM Time Base Output Postscale Select bits 1:1 Postscale
572     PTCONbits.PTCKPS=0b00;//PWM Time Base Input Clock Prescale Select bits 1:1 prescale
573     PTCONbits.PTMOD=0b00;      //PWM time base operates in a free running mode
574
575     PTMRbits.PTDIR=0;         //PWM time base is counting up
576     PTMRbits.PTMR=0;         //PWM Timebase Register Count Value 15 bits
577
578     PTPERbits.PTPER=499;     //PWM Time Base Period Value bits   15 bits
579
580     SEVTCMP=0xFFFF;          //SEVTCMP: Special Event Compare Register
581
582     PWMCON1bits.PMOD1=1;     //PWM I/O pin pair is in the independent output mode
```

```
583     PWMCON1bits.PMOD2=1;    //PWM I/O pin pair is in the independent output mode
584     PWMCON1bits.PMOD3=1;    //PWM I/O pin pair is in the independent output mode
585     PWMCON1bits.PEN1H=0;     //PWM1H pin is enabled for PWM output
586     PWMCON1bits.PEN2H=0;//PWM2H pin disabled. I/O pin becomes general purpose I/O
587     PWMCON1bits.PEN3H=0;//PWM3H pin disabled. I/O pin becomes general purpose I/O
588     PWMCON1bits.PEN1L=1;//PWM1L pin disabled. I/O pin becomes general purpose I/O
589     PWMCON1bits.PEN2L=0;//PWM2L pin disabled. I/O pin becomes general purpose I/O
590     PWMCON1bits.PEN3L=0;//PWM3L pin disabled. I/O pin becomes general purpose I/O
591
592     PWMCON2bits.SEVOPS=0;//PWM Special Event Trigger Output Postscale Select bits 4 bits
593     PWMCON2bits.IUE=1;//Updates to the active PDC registers are immediate
594     PWMCON2bits.OSYNC=0; //Output overrides via the OVDCON register occur on next TCY boundary
595     PWMCON2bits.UDIS=0;  //Updates from duty cycle and period buffer registers are enabled
596
597     DTCON1bits.DTBPS0=0; //Clock period for Dead Time Unit B is TCY
598     DTCON1bits.DTBPS1=0; //Clock period for Dead Time Unit B is TCY
599     DTCON1bits.DTB0=0;  //Unsigned 6-bit Dead Time Value bits for Dead Time Unit B
600     DTCON1bits.DTB1=0;  //Unsigned 6-bit Dead Time Value bits for Dead Time Unit B
601     DTCON1bits.DTB2=0; //Unsigned 6-bit Dead Time Value bits for Dead Time Unit B
602     DTCON1bits.DTB3=0;  //Unsigned 6-bit Dead Time Value bits for Dead Time Unit B
603     DTCON1bits.DTB4=0; //Unsigned 6-bit Dead Time Value bits for Dead Time Unit B
604     DTCON1bits.DTB5=0; //Unsigned 6-bit Dead Time Value bits for Dead Time Unit B
605     DTCON1bits.DTAPS0=0; //Clock period for Dead Time Unit A is TCY 2 BITS
606     DTCON1bits.DTAPS1=0;//Clock period for Dead Time Unit A is TCY 2 BITS
607     DTCON1bits.DTA0=0; //Unsigned 6-bit Dead Time Value bits for Dead Time Unit A
608     DTCON1bits.DTA1=0;  //Unsigned 6-bit Dead Time Value bits for Dead Time Unit A
609     DTCON1bits.DTA2=0; //Unsigned 6-bit Dead Time Value bits for Dead Time Unit A
610     DTCON1bits.DTA3=0; //Unsigned 6-bit Dead Time Value bits for Dead Time Unit A
611     DTCON1bits.DTA4=0;//Unsigned 6-bit Dead Time Value bits for Dead Time Unit A
612     DTCON1bits.DTA5=0;//Unsigned 6-bit Dead Time Value bits for Dead Time Unit A
613     FLTACON=0;      //Fault A Control Register
614
615     OVDCONbits.POVD3H=0;//Output on PWMxx I/O pin is controlled
616     OVDCONbits.POVD2H=0;//Output on PWMxx I/O pin is controlled
617     OVDCONbits.POVD1H=0;//Output on PWM1H I/O pin is controlled by the PWM generator
618     OVDCONbits.POVD3L=0;//Output on PWMxx I/O pin is controlled
619     OVDCONbits.POVD2L=0;//Output on PWMxx I/O pin is controlled
620     OVDCONbits.POVD1L=1;//Output on PWMxx I/O pin is controlled
621
```

```
622     OVDCONbits.POUT3H=0;//PWMxx I/O pin is driven INACTIVE  POVDxx bit is cleared
623     OVDCONbits.POUT2H=0;//PWMxx I/O pin is driven INACTIVE  POVDxx bit is cleared
624     OVDCONbits.POUT1H=0; //PWMxx I/O pin is driven INACTIVE  POVDxx bit is cleared
625     OVDCONbits.POUT3L=0;//PWMxx I/O pin is driven INACTIVE POVDxx bit is cleared
626     OVDCONbits.POUT2L=0;//PWMxx I/O pin is driven INACTIVE  POVDxx bit is cleared
627     OVDCONbits.POUT1L=0; //PWMxx I/O pin is driven INACTIVE  POVDxx bit is cleared
628
629     PDC1=0;        //PWM Duty Cycle 1 indenpendente
630     PDC2=0;        //PWM Duty Cycle 2 indenpendente
631     PDC3=0;        //PWM Duty Cycle 3 indenpendente
632     }
633
634 void ENSAIO_V(void)//Função de ensaio de velocidade
635 {
636     y++;
637     Saida_Int[y+4]=((int)POSCNT);
638     POSCNT=0;
639     if(y==396)
640     {
641         T1CONbits.TON = 0;
642         PDC1=0;
643         Operacao=0;
644         y=0;
645         POSCNT=0;
646     }
647 }
648
649 void ENVIAR_SAIDA(void)//Função que envia o vetor de saída
650 {
651      x++;
652      Enviar_int=Saida_Int[x];
653     U1TXREG=Enviar_int;
654     U1TXREG=Enviar_int>>8;
655     if(x>=400)
656     {
657         x=0;
658         T1CONbits.TON = 0;
659         Operacao=0;
660     }
```

```
661 }
662
663 void Banda_Morta(void)//Calcula a correção da zona morta
664 {
665                     Reg_Morto=(float)((int)PDC1);
666                     PDC1=PDC1+1;
667                     if((int)POSCNT>=(1))//1Rpm de velocidade
668                     {
669                     Reg_Morto=Reg_Morto-50;//Margem de segurança;
670                     T1CONbits.TON=0;
671                     Operacao=0;
672                     POSCNT=0;
673                     PDC1=0;
674           Enviar_Float(Reg_Morto);
675                     }
676 }
677
678 void ENVIAR_U(void)//Função para envia o vetor de sinal de controle
679 {
680     x++;
681     Enviar_int=u_int[x];
682     U1TXREG=Enviar_int;
683     U1TXREG=Enviar_int>>8;
684     if(x>=400)
685     {
686         x=0;
687         T1CONbits.TON = 0;
688         Operacao=0;
689     }
690 }
691
692
693 void CONTROLE_PI_V(void) //Função que atualiza o sinal de controle
694 {
695      n++;
696      if(n<397)
697      {Saida_Int[n+4]=((int)POSCNT);}
698                     POSCNT=0;
699 //u[n]=((u_ant)+((Kc*(1+(Ts/Ti)))*(Entrada-Saida[n]))-((Kc)*(Entrada_ant-Saida_ant)));
```

```
700 u_control=((u_i_ant)+((Kc*(1+(Ts/(2*Ti))))*(Entrada-(float)(Saida_Int[n])))+
701 (((Kc*(-1+(Ts/(2*Ti)))))*(Entrada_ant-Saida_ant)));
702                         if(u_control>=0)
703                 {
704                 LATDbits.LATD2=0;
705 //PDC1=(unsigned long int)(floor((Reg_Morto+u_control*(1000-Reg_Morto)/100)+0.5));
706                 if (u_control>(1000-(100)))
707                 {u_control =(1000-(100));}
708                 PDC1=(unsigned int)(floor(u_control+0.5));
709                 u_int[n]=(int)(floor(u_control+0.5));
710                 }
711                 if (u_control<0)
712                 {
713                 LATDbits.LATD2=1;
714                 if(u_control<=-(1000-(100)))
715                 {u_control =-(1000-(100));}
716                 PDC1=(unsigned int)(floor(fabs(u_control+0.499)));
717                 u_int[n]=(int)(floor(u_control+0.499));
718                 }
719                 u_i_ant=u_control;
720                 u_pd_ant=0;
721                 Saida_ant=(float)(Saida_Int[n]);
722                 Entrada_ant=Entrada;
723                 if(n>=400)
724         {
725         n=0;
726         PDC1=0;
727         T5CONbits.TON = 0;
728         }
729
730 }
731
732
733 void CONTROLE_PI_T_Infinity_V(void) //Analogo ao anterior, mas sem limite de tempo
734 {
735                 Saida_Int[5]=((int)POSCNT);
736                 POSCNT=0;
737 //u_R=((u_ant)+((Kc*(1+(Ts/Ti)))*(Entrada-Saida_R))-((Kc)*(Entrada-Saida_ant)));
738 u_control=(((u_i_ant))+((Kc*(1+(Ts/(2*Ti))))*(Entrada-(float)(Saida_Int[1])))+
```

```
739 (((Kc*(-1+(Ts/(2*Ti)))))*(Entrada_ant-Saida_ant)));
740
741                                     if(u_control>0)
742                 {
743                 LATDbits.LATD2=0;
744 /*//PDC1=(unsigned long int)(floor((Reg_Morto+u_control*(1000-Reg_Morto)/100)+0.5));
745                 if (u_control > frr)
746                 {u_control=u_control+br2;}
747                 if (u_control <= frr)
748                 {u_control = (-nrr*(1-pow(2.7183,(-u_control/mf)))+br1);}*/
749                 if (v_control>(1000-(100)))
750                 {v_control =(1000-(100));}
751                 else
752                 {v_control=u_control;}
753                 PDC1=(unsigned int)(floor(v_control+0.5));
754                 }
755                 if (u_control<0)
756                 {
757                 LATDbits.LATD2=1;
758 /*PDC1=((unsigned long int)(floor(fabs((-Reg_Morto+u_control*(1000-Reg_Morto)/100))+0.5)));
759                 if (u_control >= fll)
760                 {u_control = (nll*(1-pow(2.7183,u_control/mf)))+br1;}
761                 if (u_control < fll)
762                     {u_control = u_control+bl2;}*/
763                 if(v_control<=-(1000-(100)))
764                 {v_control =-(1000-(100));}
765                 else
766                 {v_control=u_control;}
767                 PDC1=(unsigned int)(floor(fabs(v_control+0.499)));
768                 }
769                 if (u_control==0)
770                 {v_control=0;}
771                 u_i_ant=u_control;
772                 u_pd_ant=0;
773                 Saida_ant=(float)(Saida_Int[1]);
774                 Entrada_ant=Entrada;
775                 //atualização
776                 Saida_Int[1]=Saida_Int[2];
777                 Saida_Int[2]=Saida_Int[3];
```

```
778                 Saida_Int[3]=Saida_Int[4];
779                 Saida_Int[4]=Saida_Int[5];
780                 //Envio U depois Saida
781                 Enviar_int=u_control*30;
782                 U1TXREG=Enviar_int;
783                 U1TXREG=Enviar_int>>8;
784                 Enviar_int=Saida_Int[1];
785                 U1TXREG=Enviar_int;
786                 U1TXREG=Enviar_int>>8;
787 if(Parar_ensaio==1)
788         {
789         n=0;
790         PDC1=0;
791         T4CONbits.TON = 0;
792         u_i_ant=0;
793         u_pd_ant=0;
794         Saida_ant=0;
795         }
796
797 }
798
799
800
801
802
803 void IDENTIFICAR_V(void)//Função de auto-tuning para velocidade
804 {
805    //--------------------------------IDENTIFICAÇÃO----------------------------------
806                 //yf=mean(y(length(y)-10:length(y)));
807                 yf=0;
808                                 uf=666;
809                 for(i=1;i<=10;i++)
810                 {
811                     yf=yf+(float)(Saida_Int[i+(400-10)]);
812                 }
813                 yf=yf/10;
814                 //K=yf/u
815                 K=yf/((float)uf);
816                 //A0=trapz(yf-y)*Ts;
```

```
817                     A0=0;
818                     for(i=1;i<=(400-1);i++)
819                     {
820                         A0=A0+((yf-(float)(Saida_Int[i]))+(yf-(float)(Saida_Int[i+1])));
821                     }
822                     A0=A0*(Ts/2);
823                     //T0=A0/yf;
824                     T0=A0/yf;
825                     //A1=trapz(y(1:round(T0/Ts)))*Ts;
826                     A1=0;
827                     for(i=1;i<=((int)(floor((T0/Ts)+0.5))-1);i++)
828                     {
829                         A1=A1+((float)(Saida_Int[i])+(float)(Saida_Int[i+1]));
830                     }
831                     A1=A1*(Ts/2);
832                     //T=A1/(0.368*yf)
833                     T=A1/(0.368*yf);
834                     //L=T0-T
835                     L=((float)(floor(((T0-T)*100)+0.9999)))/100;
836 //--------------FINAL---------------IDENTIFICAÇÃO-------------------------------
837 }
838
839
840 void IDENTIFICAR_SELF_V(void)//Função de self-tuning para posição
841 {
842     //K_MF = mean(yr(141:end))/mean(ur(141:end))
843     uff=0;
844     K_MF=0;
845                     for(i=1;i<=10;i++)
846                     {
847                         uff=uff+(float)(u_int[i+(150-10)]);
848                     }
849                     uff=uff/10;
850                     K_MF=Entrada/uff;
851     //T0_MF=trapz(Kr*ur-yr)*Ts/uf;
852     T0_MF=0;
853                     for(i=1;i<=(400-1);i++)
854                     {
855                         T0_MF=T0_MF+(float)(u_int[i])+(float)(u_int[i+1]);
```

```
856                 }
857                 T0_MF=T0_MF*K_MF;
858                 for(i=1;i<=(400-1);i++)
859                 {
860                     T0_MF=T0_MF-((float)(Saida_Int[i])+(float)(Saida_Int[i+1]));
861                 }
862                 T0_MF=(T0_MF*Ts)/(Entrada*2);
863     //m0 = round(T0r/Ts);
864                 m0=(float)(floor((T0_MF/Ts)+0.5));
865     //yT0 = yr(m0+1);
866                 yT0=(float)(Saida_Int[((int)m0+1)]);
867     //Sy = sum(yr(1:m0));
868     //Su = sum(ur(1:m0));
869     Sy=0;
870     Su=0;
871                 for(i=1;i<=(int)m0;i++)
872                 {
873                     Sy=Sy+(float)(Saida_Int[i]);
874                     Su=Su+(float)(u_int[i]);
875                 }
876     error=99999;
877     //Calculo do L_MF, são testes de 1 a m0
878     g=0;
879     test=0;
880                 for(i=0;i<=((int)m0-1);i++)
881                 {
882                     g=pow(2.7183,(-1/((m0-i))));
883                     test=((1-g)*((K_MF*Su)-Sy));
884                     if(((error)>=(float)(fabs(yT0-test))))
885                     {
886                         error=(float)fabs(yT0-test);
887                         di=i+1;
888                     }
889                     Su=Su-(float)(u_int[(int)m0-i]);
890                 }
891                 L_MF=(di-1)*Ts;
892                 T_MF=T0_MF-L_MF;
893 }
894
```

```
895
896
897 //-------------------------Declarando o modulo do encoder-----------------------
898 void ENCODER(void)
899 {
900     //Contando os pulsos do encoder
901     ADPCFGbits.PCFG4=1;      //Configurando para entrada digital
902     ADPCFGbits.PCFG5=1;      //Configurando para entrada digital
903     TRISBbits.TRISB4=1;     // QEA RB4 IN
904     TRISBbits.TRISB5=1;      // QEB RB5 IN
905     DFLTCONbits.CEID=1;      //Interrupts due to count errors are disabled
906     DFLTCONbits.QEOUT=1;     //Digital filter outputs enabled
907     DFLTCONbits.QECK=0b111;  //QEA/QEB/INDX Digital Filter Clock Divide Select Bits 1:256
908 //Justificando a escolha anterior: A velocidade maxíma do motor é de 80 rpm (1/3 Volta/s)
909 //Cada canal apresenta 2100 pulsos por rotação do motor (Pulso/Volta)
910 //Assim, tem-se 2800 Pulsos/s na velocidade máxima da máquina
911 //O TCY é de 5 Mhz, com o Filter Clock Divide fica 5/256 Mhz que são: 19531.25 Pulsos/s
912 //Porém o filtro exige 3 pulsos do TCY, então a frequencia do TCY com o Clock Divide
913 //Tem que ser maior que a frequencia da velocidade maxima do canal do encoder
914 // Ftcy=19531.25 > 3*Fencoder= 3*2800 ->   19531.25 > 8400. O dobro é recomendado
915
916     QEICONbits.QEIM=0b111;
917     QEICONbits.SWPAB=0;  //The direction of quadrature counting
918     QEICONbits.PCDOUT=1; //Position Counter Direction Status Output Enable
919     //(QEI logic controls state of I/O pin)
920     QEICONbits.QEISIDL=0; //Not enter a power saving mode
921     //POSCNT=0; //Contador de pulsos do encoder max 112? por amostra
922     MAXCNT=0xFFFF;  //limite do contador de pulsos do encoder
923                     //Habilitar interrupção
924     IEC2bits.QEIIE=0;  //Enabling the QEI interrupt
925
926                     //Fim da contagem do encoder
927 }
928 //-------------------------------OutputCompare---------------------------------
929 void config_Output_Compare(void)
930 {
931 //-----------------Configurando para saida digital controlado por TRIS----------
932     OC1CONbits.OCM=0b000;
933     OC3CONbits.OCM=0b000;
```

```
934     TRISDbits.TRISD0=0;
935     TRISDbits.TRISD2=0;
936 }
937
938 //------------------------------FimOutputCompare---------------------------
939 //------------------------------Configurando UART--------------------------------
940 void config_UART(void)
941 {
942     U1MODEbits.UARTEN=1;    /*UART is enabled. UART pins are controlled by UART
943                               *  as defined by UEN<1:0> and UTXEN control bits.*/
944     U1MODEbits.USIDL=0;     //Continue operation in Idle mode
945     U1MODEbits.ALTIO=1;     //UART se comunica usando os pinos de E / S UxATX e UxARX
946     U1MODEbits.WAKE=1;      //Wake-up enabled
947
948     U1MODEbits.PDSEL=0b00;  //8-bit data, no parity
949     U1MODEbits.STSEL=0;     //1 Stop bit
950
951     TRISCbits.TRISC14 = 1;   // RX is input
952     TRISCbits.TRISC13 = 0;   // TX is output
953     U1STAbits.UTXBRK=0;      //UxTX pin operates normally
954     U1STAbits.URXISEL=0b00;  //Interrupt flag bit is set when a character is received
955     U1STAbits.ADDEN=0;       //Address Detect mode disabled
956
957     //CONFIGURANDO O BAUD RATE PARA 38400, erro de +1.7%
958     U1BRG=7;
959     //---------------------------------------------------------------------
960                     //RECEPÇÃO
961     //PRIORIDADE DE RECEPÇÃO
962     IPC2bits.U1RXIP=2;
963     //HABILITAR INTERRUPÇÃO RECEPÇÃO
964     IEC0bits.U1RXIE=1;
965     //APAGAR FLAG
966     IFS0bits.U1RXIF=0;
967     //---------------------------------------------------------------------
968                     //ENVIO
969     //PRIORIDADE DE ENVIO
970     IPC2bits.U1TXIP=2;
971     //HABILITAR INTERRUPÇÃO TRANSMISSÃO
972     IEC0bits.U1TXIE=0;
```

```
973     //APAGAR FLAG
974     IFS0bits.U1TXIF=0;
975     //---------------------------------------------------------------------
976                             //HABILITANDO A PORTA SERIAL
977     U1STAbits.UTXEN=1;
978 }
979
980 /*-------------------------Configurando TIMER 1-------------------------------*/
981 void config_TIMER1(void)
982 {
983     // Period = PR1 * prescaler * Tcy  / para estouro
984     T1CON = 0;              // Clear Timer 1 configuration
985     T1CONbits.TCKPS = 0;    // Set timer 1 prescaler (0=1:1, 1=1:8, 2=1:64, 3=1:256)
986     PR1 = 50000;            // Set Timer 1 period (max value is 65535)
987     _T1IP = 1;              // Set Timer 1 interrupt priority
988     _T1IF = 0;              // Clear Timer 1 interrupt flag
989     _T1IE = 1;              // Enable Timer 1 interrupt
990     T1CONbits.TSIDL=0;      //Continue timer operation in Idle mode
991     T1CONbits.TON = 0;      // Turn on Timer 1
992 }
993
994 /*-------------------------Configurando TIMER 2-------------------------------*/
995 void config_TIMER2(void)
996 {
997     // Period = PR2 * prescaler * Tcy  / para estouro
998     T2CON = 0;              // Clear Timer 2 configuration
999     T2CONbits.TCKPS = 0;    // Set timer 2 prescaler (0=1:1, 1=1:8, 2=1:64, 3=1:256)
1000    PR2 = 50000;            // Set Timer 2 period (max value is 65535)
1001    _T2IP = 1;              // Set Timer 2 interrupt priority
1002    _T2IF = 0;              // Clear Timer 2 interrupt flag
1003    _T2IE = 1;              // Enable Timer 2 interrupt
1004    T2CONbits.TSIDL=0;      //Continue timer operation in Idle mode
1005    T2CONbits.T32=0;        //TMRx and TMRy form separate 16-bit timer
1006    T2CONbits.TON = 0;      // Turn on Timer 2
1007 }
1008 /*-------------------------Configurando TIMER 3-------------------------------*/
1009 void config_TIMER3(void)
1010 {
1011     // Period = PR3 * prescaler * Tcy  / para estouro
```

```
1012      T3CON = 0;              // Clear Timer 3 configuration
1013      T3CONbits.TCKPS = 0;    // Set timer 3 prescaler (0=1:1, 1=1:8, 2=1:64, 3=1:256)
1014      PR3 = 50000;            // Set Timer 3 period (max value is 65535)
1015      _T3IP = 1;              // Set Timer 3 interrupt priority
1016      _T3IF = 0;              // Clear Timer 3 interrupt flag
1017      _T3IE = 1;              // Enable Timer 3 interrupt
1018      T3CONbits.TSIDL=0;      //Continue timer operation in Idle mode
1019      T3CONbits.TON = 0;      // Turn on Timer 3
1020  }
1021
1022  /*-------------------------Configurando TIMER 4-------------------------------*/
1023  void config_TIMER4(void)
1024  {
1025      // Period = PR4 * prescaler * Tcy  / para estouro
1026      T4CON = 0;              // Clear Timer 4 configuration
1027      T4CONbits.TCKPS = 0;    // Set timer 4 prescaler (0=1:1, 1=1:8, 2=1:64, 3=1:256)
1028      PR4 = 50000;            // Set Timer 4 period (max value is 65535)
1029      _T4IP = 1;              // Set Timer 4 interrupt priority
1030      _T4IF = 0;              // Clear Timer 4 interrupt flag
1031      _T4IE = 1;              // Enable Timer 4 interrupt
1032      T4CONbits.TSIDL=0;      //Continue timer operation in Idle mode
1033      T4CONbits.T32=0;        //TMRx and TMRy form separate 16-bit timer
1034      T4CONbits.TON = 0;      // Turn on Timer 4
1035  }
1036  /*-------------------------Configurando TIMER 5-------------------------------*/
1037  void config_TIMER5(void)
1038  {
1039      // Period = PR4 * prescaler * Tcy  / para estouro
1040      T5CON = 0;              // Clear Timer 5 configuration
1041      T5CONbits.TCKPS = 0;    // Set timer 5 prescaler (0=1:1, 1=1:8, 2=1:64, 3=1:256)
1042      PR5 = 50000;            // Set Timer 5 period (max value is 65535)
1043      _T5IP = 1;              // Set Timer 5 interrupt priority
1044      _T5IF = 0;              // Clear Timer 5 interrupt flag
1045      _T5IE = 1;              // Enable Timer 5 interrupt
1046      T5CONbits.TSIDL=0;      //Continue timer operation in Idle mode
1047      T5CONbits.TON = 0;      // Turn on Timer 5
1048  }
1049
1050  //Funções relacionadas ao controle de posição
```

```
1051 void IDENTIFICAR(void)//Função auto-tuning para posição
1052 {
1053    //-------------------------------IDENTIFICAÇÃO----------------------------------
1054                  //yf=mean(y(length(y)-10:length(y)));
1055                  yf=0;
1056                  uf=666;
1057                  for(i=1;i<=10;i++)
1058                  {
1059                      yf=yf+(float)(Saida_Int[i+(400-10)]);
1060                  }
1061                  yf=yf/10;
1062                  //tu=46*Ts;
1063                  ut=46*Ts;
1064                  //K=yf/(uf*ut)  %yf=delta y; uf= delta u; ut= largura do pulso no tempo;
1065                  K=yf/((float)uf*ut);
1066                  //A0=trapz(yf-y)*Ts;
1067                  A0=0;
1068                  for(i=1;i<=(400-1);i++)
1069                  {
1070                      A0=A0+((yf-(float)(Saida_Int[i]))+(yf-(float)(Saida_Int[i+1])));
1071                  }
1072                  A0=A0*(Ts/2);
1073                  //T0 = A0/yf - ut/2
1074                  T0=(A0/yf)-ut/2;
1075                  //A1=trapz(y(1:(ceil(T0/Ts)+1)))*Ts;
1076                  A1=0;
1077                  for(i=1;i<=(int)((floor((T0/Ts)+0.9999)));i++)
1078                  {
1079                      A1=A1+((float)(Saida_Int[i])+(float)(Saida_Int[i+1]));
1080                  }
1081                  A1=A1*(Ts/2);
1082                  //T = A1/(0.132*K*uf*T0)
1083                  T=A1/(0.132121*K*uf*T0);
1084                  //L=T0-T
1085                  L=((float)(floor(((T0-T)*100)+0.9999)))/100;
1086 //--------------FINAL---------------IDENTIFICAÇÃO-------------------------------
1087 }
1088
1089 void IDENTIFICAR_SELF(void)//Função self-tuning para posição
```

```
{
    uff=0;
    K_MF=0;
                for(i=1;i<=(400-1);i++)
                {
                    uff=uff+((float)(u_int[i])+(float)(u_int[i+1]));
                }
                uff=uff*(Ts/2);
                K_MF=Entrada/uff;

                    //Calculando T0_MF
    T0_MF=0;
                for(i=3;i<=(400-1);i++)
                {
                    Soma_aux=0;
                    for (x=2;x<=(i-1);x++)
                    {
                        Soma_aux=Soma_aux+((float)(u_int[x]));
                    }
T0_MF=((((Soma_aux*2+((((float)(u_int[1]))+((float)(u_int[i])))))*(Ts/2))*K_MF)-
((float)(Saida_Int[i]))+T0_MF);
                }

T0_MF=(((((float)(u_int[1]))+((float)(u_int[2])))*(Ts/2)*K_MF)-
((float)(Saida_Int[2]))+T0_MF)*2;
                Soma_aux=0;
                for (x=2;x<=(400-1);x++)
                    {
                    Soma_aux=Soma_aux+((float)(u_int[x]));
                    }

T0_MF=(T0_MF+(((Soma_aux)*2)+((float)(u_int[1]))+((float)(u_int[400])))*((Ts*K_MF)/2)-
((float)(Saida_Int[400])))*(Ts/2);
                T0_MF=T0_MF/Entrada;
    //m0 = round(T0r/Ts);
                m0=(floor((T0_MF/Ts)+0.5));
    //yT0 = yr(m0+1);
                yT0=(float)(Saida_Int[((int)m0+1)]);
    //Sy = sum(yr(1:m0));
```

```
1129        //Su = sum(ur(1:m0));
1130        Sy=0;
1131        Su=0;
1132                    for(i=1;i<=((int)m0-1);i++)
1133                    {
1134                        Sy=Sy+(float)(Saida_Int[i]);
1135                        Su=Su+(float)(u_int[i]);
1136                    }
1137        error=99999;
1138        //Calculo do L_MF, são testes de 1 a m0
1139        g=0;
1140        test=0;
1141        //(K*(m0-d)*Ts*((1/((m0-d)))-1+g))*u((m0-d))
1142                    for(i=0;i<=((int)m0-2);i++)
1143                    {
1144                        g=pow(2.7183,(-1/(m0-i)));
1145    test=(g)*((float)(Saida_Int[(int)(m0)]))+(K_MF*(m0-i)*Ts*(1/((m0-i)))*(1-g))*Su;
1146    test=test+(K_MF*(m0-i)*Ts*((1/((m0-i)))-1+g))*((float)(u_int[(int)(m0-i)]));
1147                        if(((error)>=(float)(fabs(yT0-test))))
1148                        {
1149                            error=(float)fabs(yT0-test);
1150                            di=i+1;
1151                        }
1152                        Su=Su-u_int[(int)m0-i];
1153                    }
1154                    L_MF=(di-1)*Ts;
1155                    T_MF=T0_MF-L_MF;
1156
1157    }
1158    void ENSAIO(void)//Função do ensaio do auto-tuning
1159    {
1160        y++;
1161        Saida_Int[y+4]=((int)POSCNT)+Saida_Int[y+3];
1162        POSCNT=0;
1163        if(y==46)
1164        {
1165        //PDC1=(Reg_Morto+uf*(1000-Reg_Morto)/100);
1166        PDC1=0;
1167        }
```

```
1168      if(y==396)
1169      {
1170          T1CONbits.TON = 0;
1171          PDC1=0;
1172          Operacao=0;
1173          y=0;
1174          POSCNT=0;
1175      }
1176  }
1177
1178
1179  void CONTROLE_PI(void)//Atualizando o sinal de controle
1180  {
1181       n++;
1182       if(n<397)
1183       {Saida_Int[n+4]=((int)POSCNT)+Saida_Int[n+3];}
1184                    POSCNT=0;
1185  u_i=((u_i_ant)+((Kc*Ts)/(2*Ti))*(Entrada-(float)(Saida_Int[n])))+
1186  (((Kc*Ts)/(2*Ti))*(Entrada_ant-Saida_ant));
1187  //u_pd=((10*(2*T-Ts))/(2*T+10*Ts))*(u_pd_ant)+
1188  ////((10*Kc*(2*T+Ts))/(2*T+10*Ts))*(Saida[n])+((10*Kc*(-2*T+Ts))/(2*T+10*Ts))*(Saida_ant);
1189  u_pd=(((2*Tf-Ts))/(2*Tf+Ts))*(u_pd_ant)+((Kc*(2*Td+Ts))/(2*Tf+Ts))*((float)(Saida_Int[n]))+
1190  ((Kc*(-2*Td+Ts))/(2*Tf+Ts))*(Saida_ant);
1191
1192                    u_control=u_i-u_pd;
1193                    if(u_control>0)
1194                    {
1195                    LATDbits.LATD2=0;
1196  //PDC1=(unsigned long int)(floor((Reg_Morto+u_control*(1000-Reg_Morto)/100)+0.5));
1197                    if (u_control > frr)
1198                    {v_control=u_control+br2;}
1199                    if (u_control <= frr)
1200                    {v_control = (-nrr*(1-pow(2.7183,(-u_control/mf)))+br1);}
1201                    if (v_control>(1000-(100)))
1202                    {v_control =(1000-(100));}
1203                    PDC1=(unsigned int)(floor(v_control+0.5));
1204                    u_int[n]=(int)(floor(u_control+0.5));
1205                    }
1206                    if (u_control<0)
```

```
1207                    {
1208                    LATDbits.LATD2=1;
1209    //PDC1=((unsigned long int)(floor(fabs((-Reg_Morto+u_control*(1000-Reg_Morto)/100))+0.5)));
1210                    if (u_control >= fll)
1211                    {v_control = (nll*(1-pow(2.7183,u_control/mf)))+br1;}
1212                    if (u_control < fll)
1213                        {v_control = u_control+bl2;}
1214                    if(v_control<=-(1000-(100)))
1215                    {v_control =-(1000-(100));}
1216                    PDC1=(unsigned int)(floor(fabs(v_control+0.499)));
1217                    u_int[n]=(int)(floor(u_control+0.499));
1218                    }
1219                    if (u_control==0)
1220                    {v_control=0;}
1221                    u_i_ant=u_i;
1222                    u_pd_ant=u_pd;
1223                    Saida_ant=(float)(Saida_Int[n]);
1224                    Entrada_ant=Entrada;
1225                    if(n>=400)
1226            {
1227            n=0;
1228            PDC1=0;
1229            T3CONbits.TON = 0;
1230            }
1231    }
1232
1233    void CONTROLE_PI_T_Infinity(void)//analogo ao anterior
1234    {
1235         Saida_Int[5]=((int)POSCNT);
1236    u_i=((u_i_ant)+((Kc*Ts)/(2*Ti))*(Entrada-(float)(Saida_Int[1])))+
1237    (((Kc*Ts)/(2*Ti))*(Entrada_ant-Saida_ant));
1238    //u_pd=((10*(2*T-Ts))/(2*T+10*Ts))*(u_pd_ant)+((10*Kc*(2*T+Ts))/(2*T+10*Ts))*(Saida[n])+
1239    ////((10*Kc*(-2*T+Ts))/(2*T+10*Ts))*(Saida_ant);
1240    u_pd=(((2*Tf-Ts))/(2*Tf+Ts))*(u_pd_ant)+((Kc*(2*Td+Ts))/(2*Tf+Ts))*((float)(Saida_Int[1]))+
1241    ((Kc*(-2*Td+Ts))/(2*Tf+Ts))*(Saida_ant);
1242
1243                    u_control=u_i-u_pd;
1244                    if(u_control>0)
1245                    {
```

```
1246                    LATDbits.LATD2=0;
1247    //PDC1=(unsigned long int)(floor((Reg_Morto+u_control*(1000-Reg_Morto)/100)+0.5));
1248                    if (u_control > frr)
1249                    {v_control=u_control+br2;}
1250                    if (u_control <= frr)
1251                    {v_control = (-nrr*(1-pow(2.7183,(-u_control/mf)))+br1);}
1252                    if (v_control>(1000-(100)))
1253                    {v_control =(1000-(100));}
1254                    PDC1=(unsigned int)(floor(v_control+0.5));
1255                    }
1256                    if (u_control<0)
1257                    {
1258                    LATDbits.LATD2=1;
1259    //PDC1=((unsigned long int)(floor(fabs((-Reg_Morto+u_control*(1000-Reg_Morto)/100))+0.5)));
1260                    if (u_control >= fll)
1261                    {v_control = (nll*(1-pow(2.7183,u_control/mf)))+br1;}
1262                    if (u_control < fll)
1263                        {v_control = u_control+bl2;}
1264                    if(v_control<=-(1000-(100)))
1265                    {v_control =-(1000-(100));}
1266                    PDC1=(unsigned int)(floor(fabs(v_control+0.499)));
1267                    }
1268                    if (u_control==0)
1269                    {v_control=0;}
1270                    u_i_ant=u_i;
1271                    u_pd_ant=u_pd;
1272                    Saida_ant=(float)(Saida_Int[1]);
1273                    Entrada_ant=Entrada;
1274                    //atualização
1275                    Saida_Int[1]=Saida_Int[2];
1276                    Saida_Int[2]=Saida_Int[3];
1277                    Saida_Int[3]=Saida_Int[4];
1278                    Saida_Int[4]=Saida_Int[5];
1279                    //Envio U depois Saida
1280                    Enviar_int=u_control*30;
1281                    U1TXREG=Enviar_int;
1282                    U1TXREG=Enviar_int>>8;
1283                    Enviar_int=Saida_Int[1];
1284                    U1TXREG=Enviar_int;
```

```
1285                    U1TXREG=Enviar_int>>8;
1286  if(Parar_ensaio==1)
1287          {
1288          n=0;
1289          PDC1=0;
1290          T2CONbits.TON = 0;
1291          u_i_ant=0;
1292          u_pd_ant=0;
1293          Saida_ant=0;
1294          }
1295  }
1296  //-----------------------Tratanto interrupções---------------------------------
1297  //TRATANDO INTERRUPÇÃO ENCODER
1298  void __attribute__((__interrupt__, no_auto_psv)) _QEIInterrupt(void)
1299  {
1300          /* Clear QEI interrupt flag */
1301              IFS2bits.QEIIF = 0;
1302  }
1303  //TRATANDO INTERRUPÇÃO RECEPÇÃO
1304  void __attribute__((__interrupt__,no_auto_psv)) _U1RXInterrupt(void)
1305  {
1306      /* Clear RX interrupt flag */
1307      IFS0bits.U1RXIF=0;//APAGA FLAG
1308      if (dado==201)
1309      {
1310          Recebimento[1]=(unsigned char)U1RXREG;
1311          dado=2;
1312      }
1313      else if (dado==202)
1314      {
1315          Recebimento[2]=(unsigned char)U1RXREG;
1316          dado=2;
1317      }
1318       else if (dado==203)
1319      {
1320          Recebimento[3]=(unsigned char)U1RXREG;
1321          dado=2;
1322      }
1323       else if (dado==204)
```

```
1324        {
1325            Recebimento[4]=(unsigned char)U1RXREG;
1326            dado=2;
1327        }
1328        else
1329        {dado=(unsigned char)U1RXREG;}
1330    }
1331
1332    void __attribute__((__interrupt__,no_auto_psv)) _U1TXInterrupt(void)
1333    {
1334        IFS0bits.U1TXIF=0;
1335    }
1336    //TRATANDO INTERRUPÇÃO TIMER1
1337    void __attribute__((__interrupt__, __auto_psv__)) _T1Interrupt(void)
1338    {
1339        _T1IF = 0;//LIMPA FLAG
1340        if (Operacao==151)
1341        {ENSAIO_V();}
1342        if (Operacao==51)
1343        {ENSAIO();}
1344        if (Operacao==52)
1345        {ENVIAR_SAIDA();}
1346        if (Operacao==53)
1347        {Banda_Morta();}
1348        if (Operacao==72)
1349        {ENVIAR_U();}
1350
1351    }
1352    //TRATANDO INTERRUPÇÃO TIMER2
1353    void __attribute__((__interrupt__, __auto_psv__)) _T2Interrupt(void)
1354    {
1355        _T2IF = 0;//LIMPA FLAG
1356        CONTROLE_PI_T_Infinity();
1357    }
1358    //TRATANDO INTERRUPÇÃO TIMER3
1359    void __attribute__((__interrupt__, __auto_psv__)) _T3Interrupt(void)
1360    {
1361        _T3IF = 0;//LIMPA FLAG
1362        CONTROLE_PI();
```

```
1363 }
1364
1365 //TRATANDO INTERRUPÇÃO TIMER4
1366 void __attribute__((__interrupt__, __auto_psv__)) _T4Interrupt(void)
1367 {
1368     CONTROLE_PI_T_Infinity_V();
1369     _T4IF = 0;//LIMPA FLAG
1370 }
1371
1372 //TRATANDO INTERRUPÇÃO TIMER5
1373 void __attribute__((__interrupt__, __auto_psv__)) _T5Interrupt(void)
1374 {
1375     CONTROLE_PI_V();
1376     _T5IF = 0;//LIMPA FLAG
1377 }
```