SHAMIR KATSUDI AFUSO

CONTROLADOR DIGITAL APLICADO A ROBÓTICA MÓVEL EMBARCADA

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

Curso de Engenharia Elétrica com ênfase em
Eletrônica

ORIENTADOR: Manoel Luís de Aguiar

São Carlos
2012

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

Afuso, Shamir Katsudi
A258c Controlador digital aplicado a robótica móvel embarcada. / Shamir Katdudi Afuso ; orientador Manoel Luis de Aguiar -- São Carlos, 2012.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São de São Paulo, 2012.

1. Controladores clássicos. 2. *Fuzzy*. 3. Microcontroladores. 4. Mini motor CC. I. Titulo.

Dedicatória

A Deus por tudo que me proporciona na vida.

À minha mãe e meu pai, os quais amo muito, pelo exemplo de vida e família.

A meus irmãos por tudo que me ajudarem até hoje.

À minha namorada Lenita, pelo carinho, compreensão e companheirismo.

Agradecimentos

Agradeço aos meus pais, Paulo e Celia, pela determinação e luta na minha formação e dos meus irmãos, fazendo amparar os ensinamentos de meus ancestrais.

Aos meus irmãos, Sheila e Sander, que por mais difícil que fossem as circunstâncias, sempre tiveram paciência e confiança.

Agradeço em especial à minha namorada pelo seu companheirismo, amizade, carinho e felicidade, que mesmo à distância, seja por chat ou por telefone, incluía mensagens de incentivo e força a mim. Obrigado por tudo amor!

Ao Profº Dr. Manoel, orientador, professor, amigo, um muito obrigado pela ajuda por esse trabalho.

Agradeço a todos os meus amigos, companheiros, integrantes do grupo de robótica da Usp de São Carlos - Warthog Robotics, em especial ao Leandro G., que graças a ele pude participar desse grupo e assim desenvolver meu conhecimento nessa área.

Quero também agradecer a todos os colegas de turma, especialmente ao Fernando, José, Marcio, João J., Gabiel H., João C., Gabriel Stein e Gabriel Silva. Foi bom conviver com vocês, muito obrigado.

Sumário

Lista de Figuras:

Resumo

Neste trabalho, descrevem-se todos os componentes mecânicos e eletrônicos do robô e a forma como se interliga para formar um robô móvel terrestre para o qual será projetado controladores de velocidade de suas rodas acionadas por dois mini-motores CC. O que viabiliza o robô ser programado com um conjunto de comandos através dos quais um usuário possa fazer com que o robô se movimente de uma maneira desejada.

Palavras-chaves: Controladores clássicos, *Fuzzy*, Microcontroladores, mini-motor CC.

Abstract:

In this paper, we describe all mechanical and electronic components of the robot and how it connects to form a terrestrial mobile robot which is designed for speed controllers of its wheels driven by two mini DC motors. What enables the robot to be programmed with a set of commands by which a user can make the robot move in a desired way.

Key-Words: Classic controller, *Fuzzy*, microcontrollers, mini DC motor.

1 Introdução:

Uma partida de futebol entre robôs autônomos foi o desafio lançado em 1996 por um grupo internacional de pesquisadores em Inteligência Artificial e Robótica Inteligente. O futebol de robôs reúne grande parte dos desafios presentes em problemas eminentemente distribuídos do mundo real, tais como, veículos autônomos, busca de informação em bases de dados distribuídos, planejamento da geração de energia elétrica, recomposição de linhas de transmissão, controle de tráfego aéreo e urbano, etc.

A robótica alcançou sucesso através da indústria, onde se aplica predominantemente robôs manipuladores fixos ao solo, os quais desempenham tarefas de precisão em aplicações determinísticas e limitadas por um espaço operacional definido pela soma de seus prolongamentos físicos e restrições físicas de movimento. E entre as tarefas mais comumente desempenhadas por estes sistemas destacam-se, a pintura, a soldagem (fortemente aplicada na indústria automobilística e de tecnologia da informação - TI), manipulação de objetos em ambientes biologicamente controlados e serviços de reparação em ambientes insalubres.

Um ambiente automatizado, onde uma mesma ferramenta tenha que ser utilizada em vários pontos da linha de produção, possui várias soluções, tais como: a inserção de equipamentos redundantes espalhados no ambiente de trabalho (soldadores em uma linha de produção) e o aumento da magnitude dos equipamentos de atuação, para que apresentem um espaço operacional maior (gruas atendendo toda a área de uma fábrica). Porém, essa redundância ou aumento de magnitude dos sistemas robóticos implica em ações inviáveis econômica ou logisticamente.

A robótica móvel torna-se uma solução plausível à indústria no momento em que estas opções acima citadas se tornam impossíveis ou não atrativas na prática, valendo-se de sua capacidade de se deslocar e de fácil adaptar-se a ambientes, embora controlados, que apresentam constantes mudanças de configuração física. Desenvolvendo tarefas como o transporte de peças e pessoas, reuso de ferramentas, etc. Sendo aplicados até em ambientes não determinísticos.

Alguns robôs de fábrica modernos são "autônomos" com as limitações de seu ambiente normal. Talvez não existam todos os níveis de liberdade ao seu redor, mas o ambiente de trabalho de uma fábrica é complexo e pode ser imprevisível e até mesmo

caótico. A orientação e posição exata do próximo objeto a trabalhar e até mesmo o tipo do objeto e o trabalho requerido devem ser determinados.

Um robô totalmente autônomo no mundo real tem a habilidade de:

- Receber informações do seu ambiente.
- Trabalhar sem nenhuma interferência humana.
- Se deslocar do ponto A ao ponto B, sem assistência de navegação humana.
- Evitar situações que são perigosas para as pessoas.
- Reparar-se sem ajuda externa.

Sendo assim, o futebol de robôs apresenta-se como um laboratório para pesquisa e ensino em automação e informática industrial.

Um time de futebol de robôs consiste em uma coleção de veículos autônomos capazes de reconhecer o ambiente onde estão inseridos, no caso o campo de futebol e seus pontos de referência, e os objetos pertencentes a este ambiente, a bola, os outros veículos que compõem o time e os adversários. Estes dispositivos devem ainda ser capazes de representar o ambiente, estabelecer metas, planejar e executar ações para atingir tais metas. Em se tratando de um jogo de equipe, além do caráter autônomo, os jogadores de um time de futebol de robôs devem ser capazes de interagir com os outros jogadores do time para estabelecer objetivos coletivos, metas globais, planejar, alocar tarefas aos demais integrantes do time, sincronizar as ações de forma a imprimir ao time um perfil cooperativo. A construção de um time de futebol de robôs envolve a integração de diversas tecnologias, como projeto de agentes autônomos, cooperação em sistemas multiagentes, estratégias de aquisição de conhecimento, sistemas de tempo real, sistemas distribuídos, reconhecimento de padrões, integração de sensores, aprendizado, robótica móvel, etc.

Nesse trabalho será implementado o controle de velocidade dos motores CC do robô cuja função de transferência será obtida através de ensaios nos quais será aplicado um degrau de tensão no motor, ou seja, o PWM (modulação por largura de pulso) será máximo habilitando o *driver* (ponte H) a fornecer a máxima potência ao motor, no qual o *encoder* embutido ao motor *Faulhaber* gerará um sinal proporcional a velocidade das rodas que é lido a cada ciclo pelo microcontrolador e por fim esse dispositivo realiza uma comunicação serial com o rádio que enviará os dados (amostrados em taxa adequada permitindo uma boa visualização da resposta transitória dos motores) a um computador

que possuirá um rádio receptor. A manipulação dessas informações será realizada no *toolbox Ident* do *Matlab* que através de métodos numéricos determina a função de transferência do sistema para o tempo contínuo.

A próxima etapa é o projeto de controladores que conduzam o sistema a um desempenho satisfatório. No domínio do tempo, as especificações desejadas podem ser definidas em termos do máximo sobresinal ou coeficiente de amortecimento, do tempo de acomodação para uma entrada degrau e do erro de regime, as quais podem ser descritas em termos de pólos e zeros do sistema a malha fechada. Pode-se assim utilizar o método do lugar das raízes para determinar a função de transferência do controlador para obter a conFiguração de pólos e zeros desejada.

Pretende-se testar os controladores proporcinal (P), proporcional-integral (PI) e PIncremental *Fuzzy* utilizando realimentação como forma de alterar as funções originais existentes no sistema, ou seja, estes controladores irão verificar se os dados de entrada (recebidos pelo sistema) estão de acordo com os dados reais. No nosso caso (controle de motor), o controlador verificará se a potência fornecida ao motor é a ideal para a situação em que ele se encontra ou se é necessário aumentar ou diminuir a velocidade do motor.

No Capítulo 2 desse trabalho serão apresentadas partes integrantes de um robô divididas em seu módulo mecânico e em seu módulo eletrônico.

No Capítulo 3 serão abordados conceitos de controle, critérios de estabilidade e alguns controladores digitais. Também explicará como utilizar o toolbox *RLTool* do *Software Matlab* para projetos de controladores através do método lugar das raízes.

No Capítulo 4 serão expostos e avaliados todos os resultados desse trabalho que compreendem a função de transferência do mini-motor CC, simulações e comparações com os dados reais.

2 Fundamentos e constituição de robótica móvel

Um robô é um dispositivo, ou grupo de dispositivos, eletromecânicos capazes de realizar trabalhos de maneira autônoma, pré-programada, ou através de controle humano. Composto por uma parte física (carcaça, rodas, engrenagens, etc) e um módulo eletrônico responsável pelo tratamento das informações recebidas via rádio e alguns sensores, e controle dos motores.

2.1 Mecânica

O módulo da mecânica tem por objetivo servir de base para o módulo eletrônico, sendo esta a parte física do modelo robótico. Foi projetado para ser uma estrutura modular, simples de ser ampliada, otimizando o espaço delimitado (cubo de 75 mm) pelas regras da categoria *IEEE Very-Small Size Soccer*.

A mecânica é, basicamente, composta por uma estrutura externa, representa na Figura 1, que serve de proteção contra colisões e, uma interna representada na Figura 2, que suporta os motores, as rodas, a redução, as baterias e placa eletrônica.

Figura 1 - Vista da estrutura externa do robô.

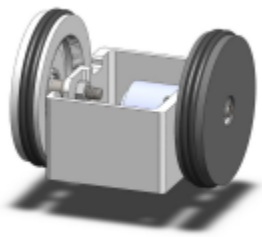

Figura 2 - Vista da estrutura interna do robô.

Os robôs foram construídos em ABS (Acrilonitrila Butadieno Estireno) é um copolímero composto pela combinação de acrilonitrila, butadieno e estireno cuja proporção de cada componente produz um material termoplástico rígido e leve, com alguma flexibilidade e resistência na absorção de impacto e Náilon, utilizando-se de técnicas de prototipagem rápida (ou impressão 3D), para grande parte de suas peças, principalmente para as estruturas acima citadas. O baixo peso específico e a facilidade de produção de peças com formatos não convencionais foram os fatores determinantes para escolha deste processo.

Foram utilizados dois motores *Faulhaber* 2224 006 SR(DC – Micromotors, 2004), acoplados a uma caixa de redução de 10:1, para diminuir a velocidade de rotação dos motores, que podem atingir 8000rpm. Seguindo a filosofia de otimização de espaço, utilizou-se uma engrenagem interna à própria roda. Os dados do motor com e sem redução estão na Tabela 1:

Tabela 1 - Dados dos motores e da redução.

Voltagem [V]	6
Torque [mM.m]	5
Velocidade Máx. [rpm]	8000
Potência [W]	4.55
Rendimento [%]	82
Torque Ampliado [mNm]	50
Vel. Reduzida [rpm]	800

As estruturas possuem paredes entre 2 mm e 2.5 mm, empiricamente testadas para resistir a colisões decorrentes de partidas de futebol de Robôs. A roda possui 50 mm de diâmetro e 10 mm de espessura, sendo 4 destes ocupados pela coroa da redução. A roda foi colocada diretamente sobre o externo do rolamento, e o interno ligado à um eixo fixo, evitando assim problemas de fadiga ou erro de batida no eixo.

2.2 Motor de corrente contínua (CC)

A designação de motores “CC” decorre de serem motores de corrente contínua (ou *Direct Current*). Uma corrente contínua passa por espiras de cobre, e cria um campo

magnético que interage com magnetos fixos. As forças geradas fazem girar o eixo do motor. É tipicamente usado em automóveis de pistas elétricas, por exemplo.

Nestes motores, alimentados a uma tensão constante, a intensidade da corrente elétrica é proporcional ao trabalho que o motor está a realizar. Isto significa que, quando a resistência é demasiado grande, o motor pára e consome o máximo de corrente. Esta característica é usada em robótica, para determinar situações em que um atuador se encontra bloqueado (ou impedido de completar o seu movimento). Por outro lado, este tipo de motores é pouco usado isoladamente (sem controlo auxiliar), precisamente por não garantir velocidade constante.

Nesse trabalho, mini-motores *Faulhaber* em combinação com *encoders* incremental, como os representados na Figura 3, foram utilizados para a indicação de controle da velocidade do eixo e sentido de rotação bem como para posicionamento o que contribuiu para escolha do mesmo nesse projeto.

Esses mini-motores possuem como características:

- Comutador e escovas de lâminas de metal precioso;
- Relações de tensão-velocidade e corrente-torque lineares;
- Baixa inércia - partida muito rápida;
- Baixa tensão de partida, mesmo após longos períodos de repouso;
- Alta relação tamanho-potência;
- Alta eficiência;
- Grande precisão, que garante longa vida.

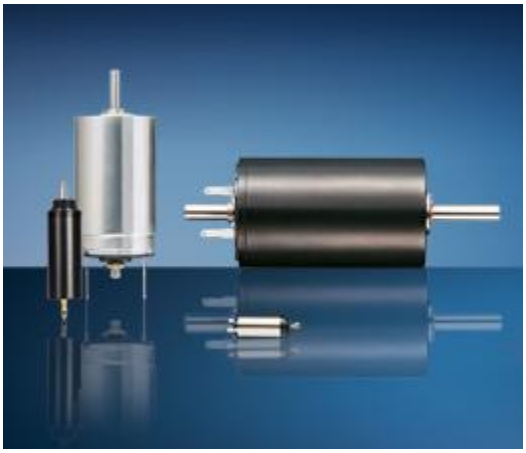

Figura 3 - Mini-motor CC Faulhaber

2.3 Eletrônica

A eletrônica embarcada deste robô realiza basicamente três tarefas. Recepção dos comandos enviados pelo computador remoto, acionamento elétrico dos motores e controle de velocidade das rodas através de *encoders*.

Um diagrama esquemático dos principais componentes necessários para o controle do robô pode ser visto na Figura 4. O rádio utilizado para recepção dos comandos enviados pelo computador remoto é um módulo Xbee que realiza uma conexão UART com o microcontrolador. Para o acionamento elétrico dos motores foi utilizado duas pontes H encapsuladas em um único chip (L298), que recebe sinais de PWM para controle de velocidade e sinais digitais para inversão de sentido dos motores. Um controlador digital foi implementado para que o robô desenvolva velocidades bem especificas durante o jogo, portando, para realimentar o controlador foram necessários dois *encoderes* digitais para realizar medidas das velocidades instantâneas dos motores. Todo o processamento digital embarcado (leitura do rádio, controle PID, PWM dos motores, leitura dos *encoderes*) é realizado por um único microcontrolador (PIC18F4431) com *clock* de 40Mhz, programado via linguagem C através do *software* (MPLAB) do próprio fabricante (Microchip).

A alimentação de todo o circuito elétrico/eletrônico embarcado é realizado através de duas células de baterias do tipo íon-lítio (Li-ion), que são baterias que oferecem boa relação peso/rendimento e menor risco de “explosão” em relação às baterias polímero de lítion (LiPo).

A maior parte dos componentes utilizados na montagem da placa eletrônica é do tipo SMD (*Surface Mounting Devices* ou dipositivos de montagem de superfície), assim foi possível reduzir drasticamente as dimensões da placa eletrônica como se pode visualizar na Figura 5, tornando possível a utilização de baterias maiores promovendo um maior tempo útil de uso do robô.

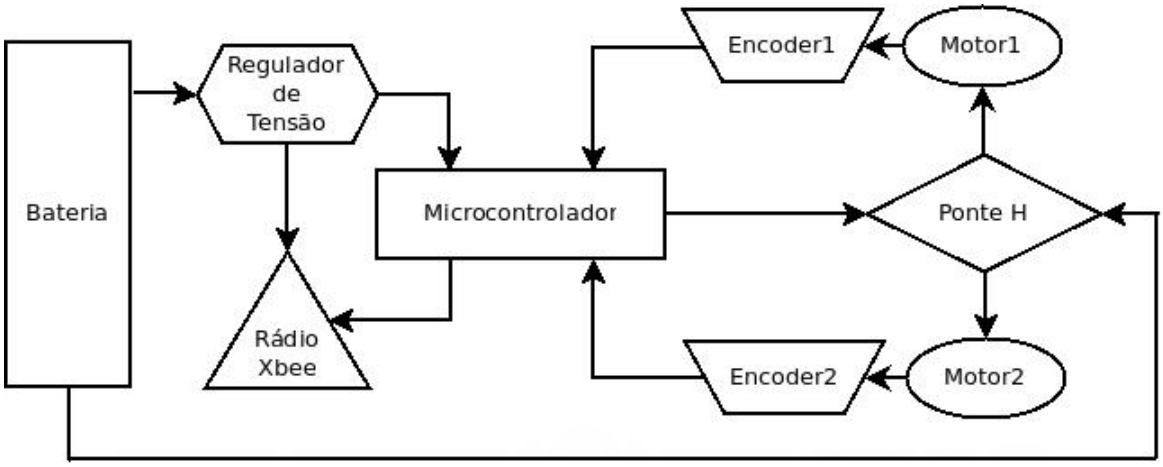

Figura 4 - Blocos das funções executadas pela eletrônica embarcada.

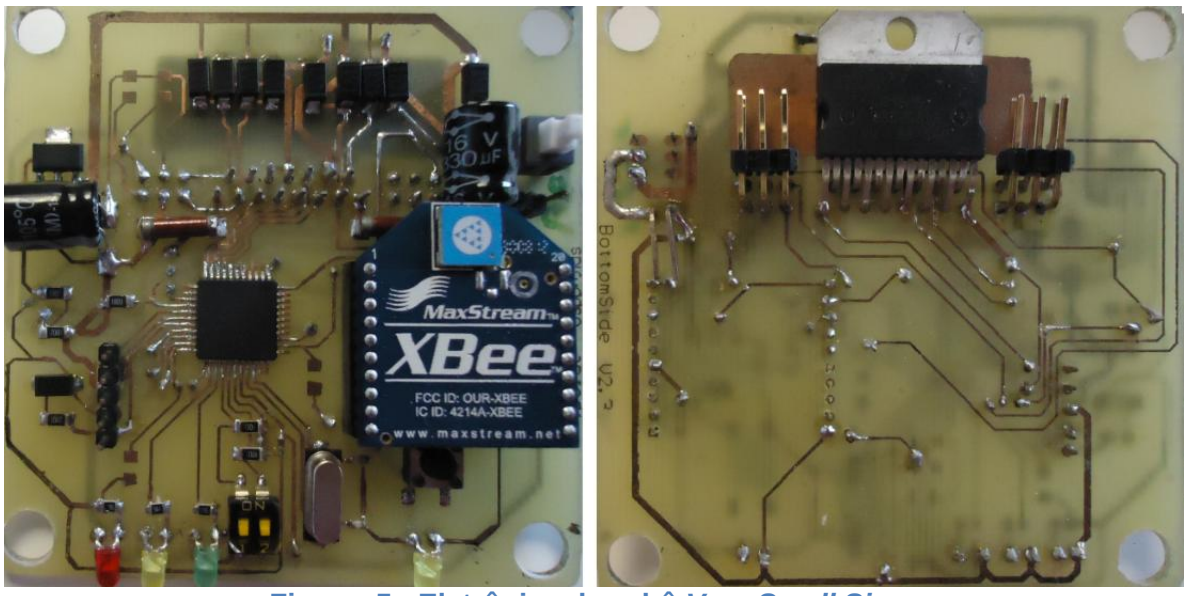

Figura 5 - Eletrônica do robô *Very Small Size.*

2.3.1 Microcontrolador

O controle de equipamentos para acionamentos industriais é comumente realizado por intermédio de um microprocessador embarcado no produto, o qual carrega um programa armazenado composto de algoritmos dedicados à aplicação em questão. Muitas vezes um único processador acumula, além das funções de controle, também a função de diálogo com o operador e comunicações com outros dispositivos, através de redes industriais (também conhecidas como barramento de campo).

Para que seja possível integrar em *software* as sofisticadas técnicas de controle utilizadas atualmente no acionamento de máquinas elétricas, juntamente com outros módulos de programas que assegurem conectividade em rede e interface amigável com o operador, tudo isso a custo competitivo, é necessário escolher adequadamente o processador e ser utilizado.

O tipo de microprocessador que melhor se presta a esta categoria de aplicações costuma ser aquele que integra na mesma pastinha de silício, além da unidade central de processamento (CPU), também circuitos de memória e uma diversidade de circuitos auxiliares (periféricos) dedicados a funções de entradas e saídos (E/S) específicas, tais como conversão analógica-digital (A/D) e saídas digitais moduladas por largura de pulso (PWM). Tal tipo de processador é usualmente chamado de microcontrolador. Com relação à arquitetura interna da CPU, os microcontroladores atualmente disponíveis no mercado podem ser classificados em três grupos principais:

- Os que possuem arquitetura de Von Neuman;
- Os de arquitetura de Harvard;
- Os de arquitetura RISC (*Reduced Instruction Set Computer*).

Devido à simplicidade dos seus circuitos internos, os microcontroladores com CPU do tipo RISC tendem a operar com maior eficiência (menor consumo de energia) com freqüências de *clock* mais elevadas. Também por este motivo, é possível integrar quantidades muito maiores de memória junto com a CPU e os circuitos periféricos na pastilha de silício (chip) que constitui o microcontrolador. Com isso, torna-se viável a incorporação de novas facilidades ao *software* de controle do produto, com impacto nos custos de produção.

Outra importante diferença encontrada entre famílias distintas de microcontroladores reside no comprimento da palavra da CPU, que normalmente vai de 8 a 32bits. De um modo geral, os microcontroladores com palavras maiores são mais eficientes na execução de algoritmos matemáticos. Isto pode acabar se refletindo na precisão e no desempenho dinâmico dos controles efetuados pelo microcontrolador.

Em aplicações típicas de controle digital, a execução dos algoritmos de controle precisa ocorrer a intervalos de tempo regulares. No caso particular do controle de dispositivos eletromecânicos, é comum que estes intervalos de tempo sejam muito

reduzidos, além de não serem toleráveis grandes variações nos mesmos. Estes aspectos caracterizam o *software* a ser utilizado como sendo de tempo real crítico. Em programas assim, a sincronização da execução dos algoritmos de controle é freqüentemente obtida através de mecanismos de interrupções produzidas por circuitos temporizadores internos ao microcontrolador.

Interrupção é um mecanismo de *hardware* disponível na memória dos microprocessadores, cuja finalidade é desencadear a execução de uma rotina de *software* em reposta a um evento ocorrido em circuitos internos ou externos à CPU. As interrupções externas são usualmente disparadas por transições de nível lógico em determinados pinos do circuito integrado (CI) que contém a CPU, ou em determinados bits de registradores associados a circuitos periféricos internos ao CI.

Ocorre, porém, que atrasos inerentes ao próprio sistema de interrupções (também chamados latências) e atrasos devidos à execução de determinadas operações podem prejudicar o desempenho do *software* em aplicações de tempo real crítico. Por isso o microcontrolador e o *software* devem ser cuidadosamente especificados para programas desse tipo e, mais uma vez, a arquitetura RISC com comprimento de palavra de 32 bits oferece vantagens para aplicações como essa.

Em uma aplicação de controle de servoacionamento, o microcontrolador é responsável pelas seguintes tarefas de tempo real:

- Aquisição de sinais de posição e velocidade para fins de controle, através de interfaces digitais para sensores do tipo *resolver* ou *encoders* (geradores de pulsos).
- Execução do algoritmo de controle de velocidade ou posição.
- Aquisição de sinais de corrente para fins de controle e proteção (conversão A/D)
- Execução do algoritmo de controle em coordenadas síncronas (d-q)
- Cálculo de valores de referência para modulação PWM das tensões produzidas pelo conversor (PWM senoidal ou vetorial).

Algumas dessas tarefas são executadas por circuitos periféricos específicos integrados no próprio CI do microcontrolador, enquanto que outras são feitas por sub-rotinas ativadas por interrupção, que são chamadas rotinas de serviço de interrupção (RSI).

❖ **PIC**

O PIC é um componente eletrônico pertencente à classe dos microcontroladores programáveis de arquitetura *Harvard* e conjunto reduzido de instruções (RISC). Em síntese, é um microcomputador completo, consistindo de uma memória RAM, memória não-volátil EEPROM, memória de programa, controladores de E/S digital e analógica (opcional) em torno de uma CPU com um conjunto reduzido de instruções, dentro de um único chip.

O PIC pode ser programado para executar diversas tarefas, como controlar um dispositivo eletro-mecânico, realizar medições, exibir informações em um display, ou simplesmente piscar luzes. A simplicidade, disponibilidade e o baixo custo são os principais atrativos do PIC.

A programação pode ser feita em um *Personal Computer* (PC) com ferramentas disponíveis gratuitamente em *Assembler* ou C, mas requer o uso de um dispositivo programador para transferir o código do programa para dentro do PIC. Normalmente isto pode ser feito de duas maneiras:

- Inserindo-se o PIC no soquete apropriado do programador. Posteriormente, o PIC é transferido para o circuito definitivo, onde pode ser soldado diretamente à placa de circuito impresso ou inserido em um soquete.
- Conectando-se um programador serial em um soquete previamente soldado à placa de circuito impresso onde o PIC já reside. Isto é conhecido como *In-Circuit Serial Programming* ou ICSP. Esta modalidade é bastante prática e causa menos estresse ao PIC, por evitar seu constante manuseio durante repetidos testes.

Há vários tipos diferentes de programadores cujo projeto pode ser encontrado na Internet ou adquirido pronto de determinados fornecedores. A maioria deles é de fácil construção e podem ser conectados ou à porta paralela ou à porta serial do PC. De fato, qualquer computador que disponha de uma dessas interfaces pode ser usado para programar o PIC.

A escolha do Pic18f4431, em destaque na Figura 6, deve-se a suas características e seus parâmetros destacados a seguir:

- *Power Control PWM Module* (8 channels)
- *Motion Feedback Module with 2 Quadrature Encoder Interface*
- 200Ksps ADC *Module*
- 8MHz *Internal Oscillator*
- *Self-Programming*
- 40 MHz *Max Speed*
- 34 I/O *Pins*
- *Enhanced Flash Program Memory Type*
- 16384 *bytes Program Memory Size*
- 768 *bytes RAM Size*
- 256 *bytes Data EEPROM*

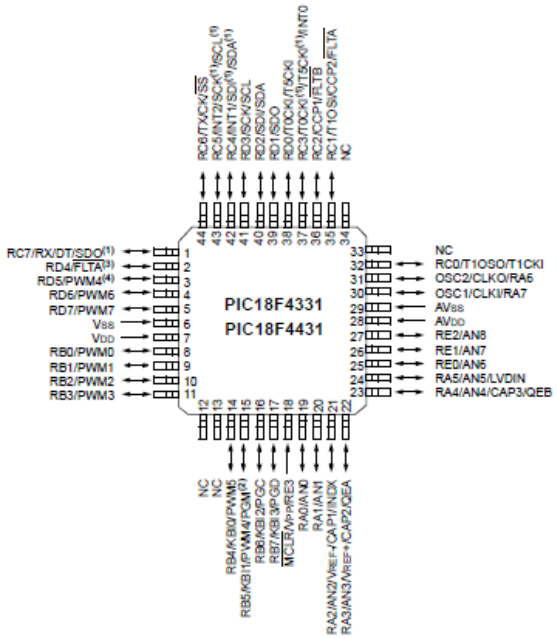

Figura 6 - Pinagem do microcontrolador Pic18F4431.

2.3.2 Sistema de entrada e saída de dados

Os circuitos de entrada e saída (E/S) que costumam ser integrados nos CI's de microcontroladores compreendem funções bastante diversificadas tais como:

- Entrada de sinais analógicos
- Entrada e saída digital paralela (controle de bits individuais)
- Comunicação serial síncrona e assíncrona

- Entradas para contagem e captura de eventos
- Interface para *encoder* incremental (gerador de pulsos)
- Saídas temporizadas
- Saídas com modulação por largura de pulso (PWM)

No controle de motores CC, os dispositivos de E/S que se associam mais diretamente aos circuitos eletrônicos de potência responsáveis pelo comando do desse motor são as entradas analógicas e as saídas PWM. Como características típicas das entradas analógicas de microcontroladores têm-se: resolução de 10bits, tempos de conversão da ordem de 10^{-6}s, disparo por *software* ou hardware, sincronizado ou não, diversos canais de entradas multiplexadas e circuito de amostragem e retenção (*sample & hold*) integrado. Alguns microcontroladores permitem a aquisição simultânea de pares de sinais.

É através das entradas analógicas que as correntes nas bobinas das fases do motor, depois de serem processadas por transdutores e circuitos de condicionamento de sinais, são convertidas em dados numéricos para serem utilizados como valores medidos no algoritmo de controle realimentado de corrente. Como resultado dos cálculos desses algoritmos, os níveis de modulação das saídas PWM são variados como valores medidos nos algoritmos de controle realimentado de corrente. Como resultado dos cálculos desses algoritmos, os níveis de modulação das saídas PWM são variados em tempo real, a cada intervalo de amostragem do sistema de controle digital da corrente.

A produção dos sinais de saída PWM é feita a partir de um contador/ temporizador dedicado, ao qual são associados circuitos internos de comparação digital. Para a geração de sinais PWM trifásicos são necessários três registradores de comparação. O contador associado a eles deve operar em modo crescente/decrescente, isto é, a contagem vai de zero até um valor máximo, correspondente a metade do período de modulação, e retorna em seguida a zero com a mesma taxa de variação que na subida. Como resultado pode-se imaginar a variação do conteúdo do contador como um sinal triangular quantizado.

Quando o valor da contagem (conteúdo do contador) ultrapassa o valor armazenado em um registrador de comparação, produz-se automaticamente uma mudança de estado nos pinos de saída correspondentes.

Comparando-se então a operação do circuito gerador de PWM de um microcontrolador com o método tradicional de geração de PWM por comparação seno/triângulo (portadora triangular e sinal modulante senoidal), tem-se que o conteúdo do contador é análogo ao papel da portadora, enquanto que o papel do sinal modulante é desempenhado pela variação do valor armazenado no registrador de comparação.

Outras características encontradas em microcontroladores incluem facilidades especiais em capturar rapidamente sinais nas linhas de entrada e gerar sinais de disparo para ativar periféricos externos. Estas são características visadas na temporização de máquinas e no sensoriamento através de transdutores.

O formato dos dados série utilizados neste projeto incluem um bit de início (*start bit*), oito bits de dados e 1bit de fim (*stop bit*). A Figura 7 ilustra o formato dos dados série.

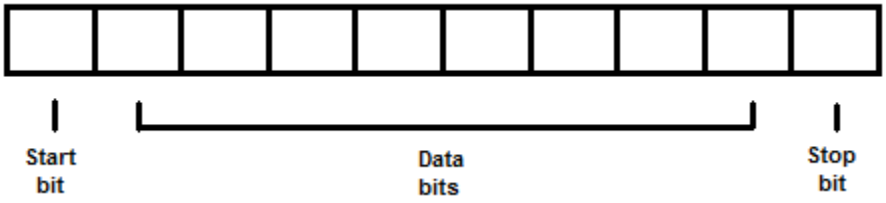

Figura 7 - Formato dos dados série

Por definição, os dados série são transmitidos um bit de cada vez. A ordem pela qual cada bit é enviado é feita de acordo com as considerações seguintes:

- O *start* bit é transmitido com o valor 0;
- Os bits de dados são transmitidos. O primeiro bit de dado corresponde ao *least significant bit* (LSB), enquanto que o último bit de dados corresponde ao *most significant bit* (MSB);
- O bit de paridade (se definido) é transmitido;
- Um ou vários *sto*p bits são transmitidos, com um valor 1.

Sendo que o número de bits transferidos por segundo é designado *baud rate*. Os bits transferidos incluem, como já visto, o *start bit*, os bits de dados, o bit de paridade (se definido), e o(s) *stop bits*. Por exemplo, 300 *baud* correspondem a 300 bits/segundo. Quando é referido um ciclo de relógio, fala-se da *baud rate*.

Como a comunicação série utilizada é assíncrona, significa que o *byte* transmitido deve ser identificado pelo *start bit* e *stop bits*. O *start bit* indica quando é que o *byte* de dado está para começar; o(s) *stop bit*(s) indica(m) quando o *byte* foi transferido. O processo de identificação de *bytes* com o formato série obedece aos seguintes passos:

- Quando um pino da porta série está *idle* (não transmitindo dados), então está num estado *on*;
- Quando os dados estão prestes a ser transmitidos, o pino da porta série alterna para um estado *off* devido ao *start bit*;
- O pino da porta série alterna novamente para o estado *on* devido ao(s) *stop bit*(s), indicando o fim do *byte* transmitido.

O(s) *stop bit*(s) são usados então para sinalizar o fim de um pacote de dados. Valores típicos correspondem a 1, 1.5, e 2 bits. Uma vez que os dados são sincronizados ao longo das linhas e que cada dispositivo possui o seu próprio relógio (na comunicação assíncrona), é possível que os dois dispositivos apareçam ligeiramente dessincronizados. Assim, o(s) *stop bit*(s) permitem alguma "folga" ao computador para compensar desvios nas velocidades dos relógios. Quanto mais *stop bits* forem usados, melhor será feita a sincronização entre os diferentes relógios, mas mais lenta será a taxa de transmissão.

2.3.3 Sistema de comunicação

Uma rede sem fio refere-se a uma rede de computadores sem a necessidade do uso de cabos – sejam eles telefônicos, coaxiais ou ópticos – por meio de equipamentos que usam radiofreqüência (comunicação via ondas de rádio) ou comunicação via infravermelho. É conhecido também pelo anglicismo *wireless*.

O uso da tecnologia vai desde transceptores de rádio como *walkie-talkies* até satélites artificiais no espaço. Seu uso mais comum é em redes de computadores, servindo como meio de acesso à Internet através de locais remotos como um escritório, um bar, um aeroporto, um parque, ou até mesmo em casa, entre outros.

Sua classificação é baseada na área de abrangência: redes pessoais ou curta distância (WPAN: *Wireless Personal Area Network* ou rede pessoal sem fio), redes locais (WLAN: (*Wireless Local Area Network*), redes metropolitanas (WMAN: *Wireless Metropolitan Area Network*) e redes geograficamente distribuídas ou de longa distância WWAN: (*Wireless Wide Area Network* - Rede de longa distância sem-fio).

Através da utilização de portadoras de rádio ou infravermelho, as WLANs estabelecem a comunicação de dados entre os pontos da rede. Os dados são modulados na portadora de rádio e transmitidos através de ondas eletromagnéticas.

Múltiplas portadoras de rádio podem coexistir num mesmo meio, sem que uma interfira na outra. Para extrair os dados, o receptor sintoniza numa freqüência específica e rejeita as outras portadoras de freqüências diferentes.

A comunicação sem fio já está inclusa na sociedade há anos todas voltadas para usuários finais de pequenas, médias e grandes empresas, onde o objetivo é a transferência de grandes volumes de dados e voz em altas velocidades. São poucas as redes wireless destinadas exclusivamente ao controle de dispositivos como relês, trancas eletromagnéticas, ventilação, aquecimento, motores, eletrodomésticos, brinquedos, aquisição de dados de sensores, como temperatura, luminosidade, umidade, pressão etc. Dentre as Redes WPAN existentes, a mais recente e promissora é a que usa o padrão ZigBee IEEE 802.15.4

- **Xbee/ZigBee (IEE 802.15.4)**

A ZigBee Alliane é a empresa que desenvolve o padrão ZigBee junto ao IEEE (*Institute of Electrical and Eletronics Engineers*), através da associação de várias empresas, que juntas, trabalham em conjunto para proporcionar e desenvolver tecnologias para criar um padrão de baixo consumo de energia, baixo custo, segurança, confiabilidade, e com funcionamento em rede sem fios baseado em uma norma aberta global.

A ZigBee permite comunicações robustas e opera na freqüência **ISM** (*Industrial, Scientific and Medical*), sendo na Europa de 868 MHz (1 canal), 915 MHz (10 canais) nos Estados Unidos e 2,4 GHz (16 canais) em outras partes do mundo, e não requerem licença para funcionamento. As Redes ZigBee oferecem uma excelente imunidade contra interferências, e a capacidade de hospedar milhares de dispositivos numa Rede (mais que 65.000), com taxas de transferências de dados variando entre 20Kbps a 250Kbps. O Protocolo ZigBee é destinado a aplicações industriais, portanto, o fator velocidade não é crítico numa implementação ZigBee.

Os módulos de rádio freqüência (RF) padrão ZigBee foram criados para economizar ao máximo energia. Com isso, é possível criar dispositivos sensores remotos alimentados com pilhas ou baterias comuns, que durarão meses ou mesmo anos sem

precisarem ser substituídas. Isso porque, os módulos ZigBee quando não estão transmitindo/recebendo dados, entram num estado de dormência ou em "*Sleep*", consumindo o mínimo de energia.

Os módulos XBee/XBee-Pro™ operam em dois modos diferentes:

- **Modo Transparente**

Os dados recebidos da UART (*Universal Asynchronous Receiver/ Transmitter*) pelo pino **DI** (RX) são colocados na fila para transmissão via RF. Já os dados recebidos do canal de RF, são transmitidos através do pino **DO** (Tx).

No modo transparente os dados são transmitidos e recebidos da mesma forma que uma comunicação Serial RS232 (padrão para troca serial de dados binários entre um terminal e um comunicador de dados), representado na Figura 8. Os módulos dispõem de *buffers* de transmissão e recepção para uma melhor performance na comunicação serial.

Esse modo é geralmente utilizado quando se deseja testar a funcionalidade das placas eletrônicas dos robôs, que recebe um pacote contendo *flag's* e *bytes* que irão ativar as operações da mesma, ou para fazer aquisição de dados como, por exemplo, foi realizado nesse trabalho para se obter valores da velocidade motor na sua resposta a um degrau de tensão.

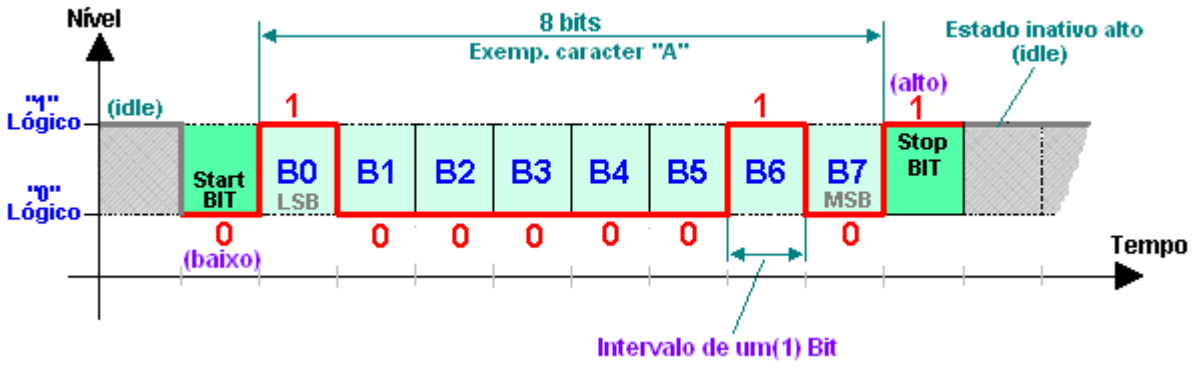

Figura 8 - Frame de dado padrão RS232 (8-N-1)

- **Modo API** (*Application Programming Interface*)

Esse modo de operação é uma alternativa ao modo de operação Transparente padrão. O modo API é baseado em frame e assim estende o nível para o qual uma aplicação de *Host* pode interagir com as capacidades de Rede do módulo.

No modo API os dados transmitidos e recebidos estão contidos em *frames*, que definem operações ou eventos dentro do módulo. Através desse modo de operação é possível um determinado módulo enviar endereço fonte, endereço destino, nome de um determinado nó, estado, e muito mais, como representado na Figura 9.

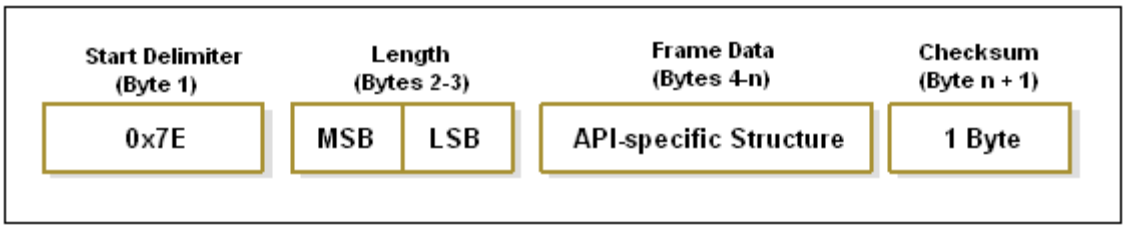

Figura 9 - Estrutura do frame de dados no modo API

No projeto do time de robôs *Very Small Size*, o modo API é habilitado para que um rádio conectado a um PC possa controlar mais de um robô ao mesmo tempo, possibilitando assim enviar comandos para todos esses a partir de um *Host*.

Características importantes do rádio XBee:

- Pode transmitir dados para múltiplos destinos sem entrar em Modo de Comandos;
- Recebe estados de sucesso/falha de cada pacote de RF transmitido;
- Identifica o endereço fonte de cada pacote recebido.

Basicamente os módulos XBee/XBee-Pro já vem de fábrica conFigurados para serem usados, o mínimo que precisamos fazer para estabelecer um *link* de comunicação é alimentar os módulos corretamente com uma tensão de 3,3V. Nem sempre temos uma fonte de alimentação de 3,3v, sendo assim, segue um esquema elétrico de um regulador de tensão que converte uma tensão de entrada de 5 a 9V em 3,3V, ideal para alimentar um módulo XBee/XBee-Pro.

NaFigura 10 abaixo, tem-se um exemplo de um microcontrolador alimentado com 5V conectado com um módulo XBee-Pro. Observe os resistores de 10K e 18K, seu objetivo é reduzir a tensão de entrada no pino 3-RX do módulo XBee-Pro. Nesse esquema, a tensão de entrada no pino 3-RX fica em torno de 3V.

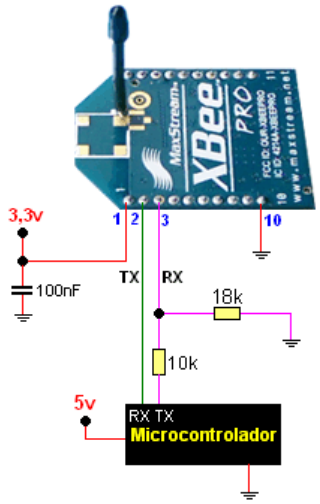

Figura 10 - XBee-Pro conectado a um microcontrolador (5V)

- **Adaptador/Conversor USB - XBee/XBee-Pro™**

Para facilitar a conexão do módulo Base XBee/XBee-Pro™ ao computador, seja para atualização do *firmware* ou mesmo para fazer coleta de dados ou controle, através dos módulos remotos, um fornecedor-Rogercom desenvolveu uma placa CON-USBBEE, com facilidade de conexão USB. Visto na Figura 11 abaixo:

Figura 11 - Placa CON-USBBEE (visão superior)

A placa CON-USBBE usa um chip conversor USB/Serial; regulador de tensão LDO (baixa queda de tensão), comparador de tensão conectado aos LEDs (RSSI) que

simulam a força do sinal de RF; LEDs indicadores de TX, RX, módulo ligado (ASS), e um micro-botão para reiniciar o módulo XBee/XBee-Pro™ em destaque na Figura 12.

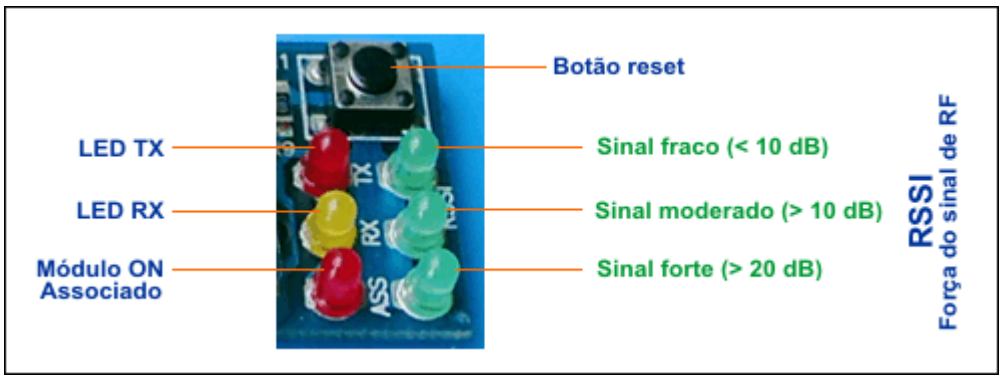

Figura 12 - Botão *Reset* e LEDs indicadores da placa CON-USBBEE

Ao instalar no computador o *driver* USB para (*Windows* 98, ME, 2000, XP,Vista, *Windows7* e também para *Linux* e *Mac*) que acompanha a placa, o *Windows* cria uma porta COMx virtual quando a placa CON-USBBEE é plugada. Assim, é possível através de um programa (escrito em C/C++Builder, *Delphi*, VB, *Java*, C#, entre outros), se comunicar com a placa como se fosse uma comunicação serial padrão RS232. Também é possível acessar a placa através de uma DLL, que oferece mais recursos na programação.

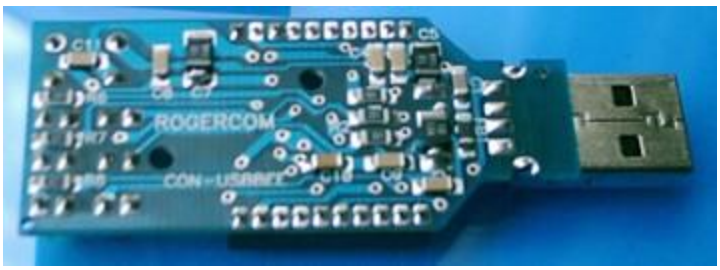

Figura 13 - Placa CON-USBBEE (visão inferior)

Sendo necessário para essa comunicação definir a taxa de transferência de dados, o tamanho do pacote de *bits*, se possui *bit* de paridade, se possui *bit stop* de acordo com foi programado a UART do microcontrolador. E também necessita da conFiguração do endereçado de cada rádio, e selecionar um modo no qual um rádio transmissor tenha apenas um rádio receptor.

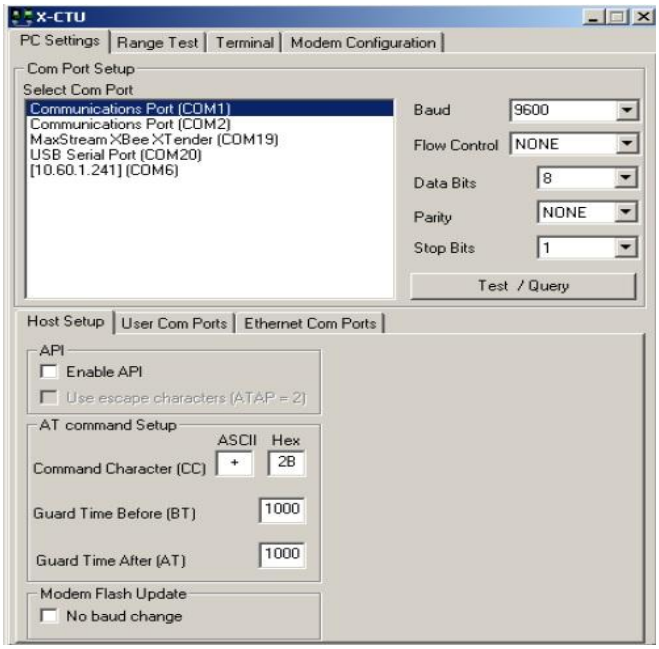

Figura 14 - Interface do *software* X-CTU.

Apesar de no projeto do futebol de robôs utilizar apenas a função de transmissão e recepção desse módulo, na Tabela 2 está descrito o significado de cada pino dos módulos XBee/XBee-pro™, como se pode ver, há pinos que podem exercer diferentes funções como, entrada analógica, entrada/saída digital, controle de fluxo e PWM. A maneira mais fácil para conFigurar a função de um determinado pino do módulo ou mesmo outros parâmetros, é através do programa X-CTU, disponível no site da MaxStream, o download e uso do programa são gratuitos.

Tabela 2 - Descrição dos pinos dos módulos XBee/XBee-Pro™

Pino #	Nome	Direção	Descrição
1	VCC	-	Alimentação 3,3v
2	DOUT	Saída	Saída de dados da UART
3	DIN / $\overline{\text{CONFIG}}$	Entrada	Entrada de dados da UART
4	DO8*	Saída	Saída digital 8
5	$\overline{\text{RESET}}$	Entrada	Inicializa módulo (um pulso nível 0 de pelo menos 200ms)
6	PWM0 / RSSI	Saída	Saída do PWM 0 / Indicador de Força do sinal de RF (RX)
7	PWM1	Saída	Saída do PWM 1
8	(Reservado)	-	Ainda não tem uma função definida (futura implementação)
9	$\overline{\text{DTR}}$ / SLEEP_IRQ / DI8	Entrada	Linha de Controle da Função Sleep ou Entrada digital 8
10	GND	-	Terra

11	AD4 / DIO4	Entrada/Saída	Só Entrada Analógica 4 ou Entrada/Saída Digital 4
12	$\overline{\text{CTS}}$ / DIO7	Entrada/Saída	Controle de Fluxo CTS ou Entrada/Saída Digital 7
13	ON / SLEEP	Saída	Indicador de Estado do Módulo
14	VREF	Entrada	Voltagem de Referência para as Entradas A/D
15	Associação / AD5 / DIO5	Entrada/Saída	Indicador de Associação, só Entrada Analógica 5 ou Entrada/Saída Digital 5
16	$\overline{\text{RTS}}$ / AD6 / DIO6	Entrada/Saída	Controle de Fluxo RTS, só Entrada Analógica 6 ou Entrada/Saída Digital 6
17	AD3 / DIO3	Entrada/Saída	Só Entrada Analógica 3 ou Entrada/Saída Digital 3
18	AD2 / DIO2	Entrada/Saída	Só Entrada Analógica 2 ou Entrada/Saída Digital 2
19	AD1 / DIO1	Entrada/Saída	Só Entrada Analógica 1 ou Entrada/Saída Digital 1
20	AD0 /DIO0	Entrada/Saída	Só Entrada Analógica 0 ou Entrada/Saída Digital 0

2.3.4 Driver

O *Driver* LM298 (Figura 15 eFigura 16) é um circuito integrado monolítico que possui um excitador duplo de alta tensão. Ele permite aceitar níveis padrão para a lógica TTL (*Transistor transistor logic*) que utiliza apenas transistores bipolares permitem maior rapidez (maior frequência), mas à custa de maior consumo para\ comandar cargas indutivas tais relés, solenóides e motores CC. Este dispositivo é fornecido com duas entradas independentes o que permite habilitar ou incapacitar o dispositivo dos sinais de entrada.

É uma unidade que permite o controle dos motores a partir de sinais, normalmente gerados por um microcontrolador. Porém os sinais de saída desse dispositivo não têm níveis de tensão adequados nem a capacidade de fornecer a corrente necessária para dar partida em um motor que esteja ligado diretamente aos seus terminais. Assim há a necessidade de uma unidade de potência que possa fornecer os níveis necessários de forma que os sinais PWM gerados pelo microcontrolador “excitem” as entradas do módulo de potência.

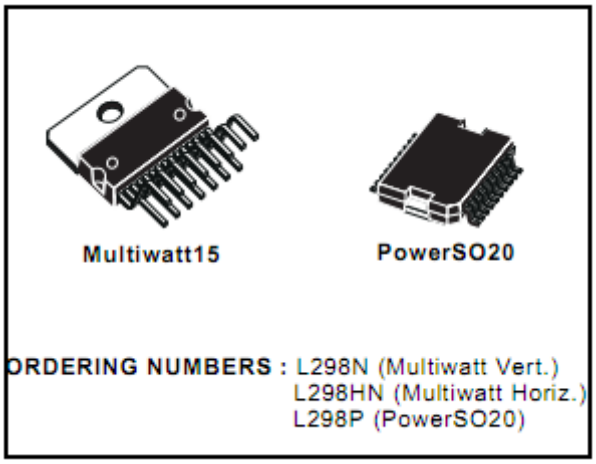

Figura 15 - Formas de encapisulamento do *driver* LM298.

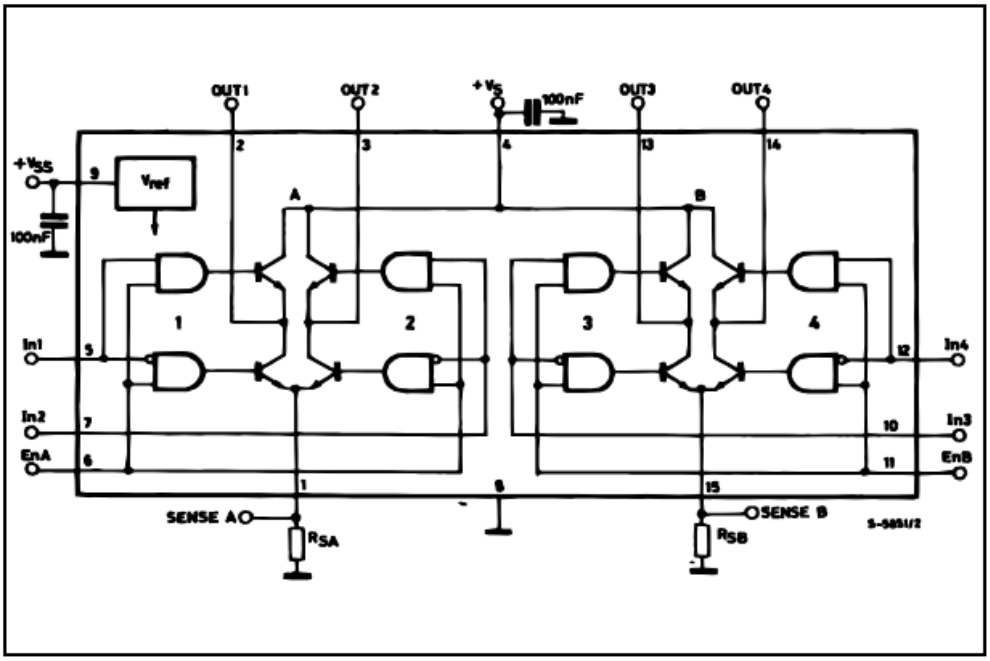

Figura 16 - Diagrama de blocos do *driver* LM298.

2.3.5 Sensores

Um sensor é um dispositivo que responde a um estímulo físico de maneira específica e mensurável. Alguns sensores respondem com sinal elétrico a um estímulo, isto é, convertem a energia recebida em um sinal elétrico. Nesse caso, podem ser chamados de transdutores. O transdutor converte um tipo de energia em outro. É geralmente composto por um elemento sensor e uma parte que converte a energia proveniente dele em sinal elétrico.

❖ ***Encoder***

Os *encoders* são transdutores de movimento capazes de converter movimentos lineares ou angulares em sinais elétricas que podem ser transformados em código binários e trabalhados por um programa que converta as informações passadas em algo que possa ser entendido como distância, velocidade, entre outros. Em nossa aplicação, o *encoder* é uma unidade de realimentação que informa sobre velocidades atuais de forma que possam ser comparadas com velocidades de referência.

Os *encoders* possuem internamente um ou mais discos (máscaras) perfurado, que permite, ou não, a passagem de um feixe de luz infravermelha, gerado por um emissor que se encontra de um dos lados do disco e captado por um receptor que se encontra do outro lado do disco, este, com o apoio de um circuito eletrônico gera um pulso, como pode ser visualizado na Figura 17. Dessa forma a velocidade ou posicionamento é registrado contando-se o número de pulsos gerados.

A quantidade de pulsos em uma volta, nos *encoders* rotativos, demonstra a relação impulso/volta do mesmo. Quanto maior for esta relação maior a precisão obtida. Sendo que esse motor *Faulhaber* possui acoplado um *encoder* que gera 32 ou 64 pulsos por volta tendo a seguinte relação angular: $360^{o}/32(64)$ pulsos. Pode-se então determinar o sentido da rotação utilizando duas fileiras de furos uma defasada em 90^{o} uma em relação à outra, sendo assim em um sentido a fileira mais próxima do centro estará adiantada em relação à outra e no sentido inverso ocorre também o inverso.

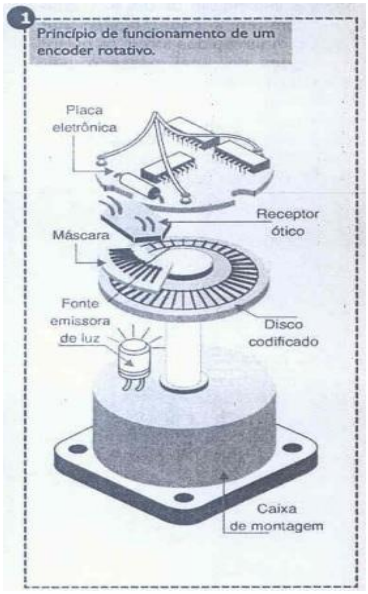

Figura 17 - Desenho mecânico de um *encoder*.

2.4 Características da robótica móvel

Um robô móvel é um dispositivo automático que é capaz de se movimentar e interagir em um ambiente definido. A robótica móvel tem como objetivo principal dotar os robôs de um nível de autonomia suficiente a torná-los capaz de interagir com o mundo no qual realizarão suas tarefas.

Caracteriza-se por ser uma área de pesquisa multidisciplinar em diversos campos da engenharia e também um excelente veículo para o aprendizado de conceitos relacionados a esta área, como a microeletrônica, a informática, a mecânica e sistemas de controle. Robôs móveis possuem a capacidade de se moverem ao redor seus ambientes e não estão fixados a uma localização física. Em contraste, os robôs industriais geralmente consistem de um braço articulado e um dispositivo de atuação, presos a uma superfície fixa.

Robôs móveis podem ser classificados sobre o ambiente em que eles se movem

- Robôs de terra: Ele normalmente tem rodas, mas alguns possuem duas pernas (humanóides), ou mais, se assemelhando a animais ou insetos.
- Robôs aéreos: Estão usualmente referidos como veículos aéreos não-tripulados (UAVs).
- Robôs subaquáticos: São chamados de veículos subaquáticos autônomos (AUVs).

Os robôs móveis são o foco de muitas pesquisas atuais, sendo que a maior parte das grandes universidades possui um ou mais laboratórios voltados à pesquisa de robôs móveis. Os robôs móveis também são encontrados na indústria, em instalação militares, em ambientes de segurança, e como produtos de consumo, seja para o entretenimento ou para realizar alguns trabalhos, como o cortador de grama da *Friendly Robotics*, limpeza de chão da *IRobot*, limpeza de piscinas com o XT5 da *Sibrape*, de vigilância da *MobileRobots* e etc.

Em robótica móvel, há diversas soluções adotadas para a movimentação em superfícies sólidas, tais como rodas, trilhos e pernas. As rodas são as mais utilizadas, pois oferecem mecânica, construção e controle simples, além de elevada velocidade. As outras soluções requerem uma mecânica bem mais complexa e pesada, mas apresentam como vantagem a possibilidade de se movimentarem em superfícies irregulares.

No que concerne às rodas para robótica móvel, os robôs movimentarem-se em trajetórias retilíneas a partir de qualquer ponto do plano, sendo que os robôs omnidirecionais fazem esse caminho sem precisar girar antes. Além do mais, em sistemas mais complexos, é possível combinar movimento rotacional e translacional do robô, de modo que ele chegue ao seu destino no ângulo correto. Para a movimentação da base, é necessário indicar um vetor para velocidade linear e outro vetor para angular para que os robôs descrevam uma trajetória e cheguem à posição final.

É uma área de pesquisa que possui ênfase no estudo de técnicas de controle de robôs autônomos e semi-autônomos na sua locomoção em diversos ambientes onde podem existir obstáculos móveis e fixos. Por isso, deve haver um mecanismo de sensoriamento para sua localização e mapeamento do ambiente em que o objeto está inserido e com essas informações processadas gerar uma trajetória ao robô.

Durante muito tempo, a tecnologia analógica dominou os acionamentos elétricos. Nas últimas duas décadas, com o desenvolvimento dos microprocessadores e circuitos periféricos, a tecnologia digital vem substituindo gradualmente a analógica nas funções convencionais de controle, e é hoje considerada a abordagem privilegiada para sistemas de alto desempenho, por causa das possibilidades únicas que ela oferece.

As desvantagens e limitações dos sistemas digitais são devidas principalmente a características inerentes de sistemas discretos que resultam em amostragem, quantização e erros de truncamento. Esses erros podem afetar seriamente os limites de rejeição de distúrbios de carga. Os atrasos de computação podem também limitar a largura de banda do sistema e a estabilidade do controle.

Um acionador elétrico controlado digitalmente é composto basicamente por três componentes: o motor elétrico, o conversor de potência, e o sistema de controle digital. A carga mecânica é acionada, diretamente ou através de engrenagens redutoras, pelo motor elétrico que é alimentado pelo conversor de potência. O conversor controla a potência vinda da fonte para o motor, ativando as chaves de acordo com os sinais de acionamento gerados pelo controlador, cujo esquemático pode ser visto na Figura 18 abaixo:

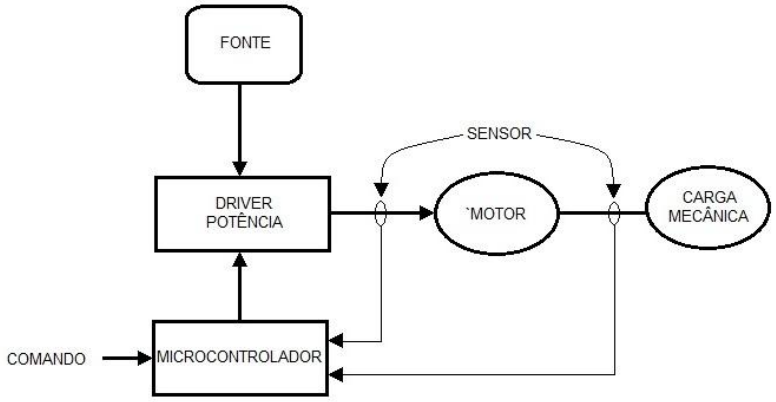

Figura 18 - Acionamento elétrico controlado digitalmente.

Devido à crescente disponibilidade de melhores dispositivos eletrônicos de potência e processadores digitais, existe uma tendência em obter-se alto desempenho de sistemas acionados por máquinas elétricas através da concepção de *software*s de controle mais sofisticados. Existem, no entanto, desafios significativos neste contexto, já que a dinâmica de máquinas elétricas exibe, em geral, não-linearidades importantes; nem todas as variáveis de estado são necessariamente medidas; os parâmetros do sistema podem variar significativamente de sues valores nominais.

2.5 Plataforma digital

As características fundamentais do sistema de controle digital são:

- Receber os sinais de comando do computador de coordenação;
- Ler e estimar as variáveis mecânicas e elétricas;
- Implementar os algoritmos de controle e a lógica do sistema.

Dependendo da aplicação específica, o controle pode ser efetuado sobre o torque, aceleração, velocidade e posição.

O sistema de controle digital, que pode envolver um ou mais processadores, processa os dados e programa o controle sob forma digital, seguindo a estrutura da Figura 19. Sua estrutura pode ser apresentada por dois aspectos: *hardware* e *software*. A conFiguração do *software* depende principalmente das funções realizadas pelo sistema de controle digital. As funções básicas podem ser classificadas em cinco grupos:

- Aquisição e processamento de dados
- Comunicação

- Sistema lógico e algoritmos de controle
- Interface do circuito de potência
- Funções auxiliares (Armazenamento, monitoração e proteção, teste e diagnóstico, e interface gráfica)

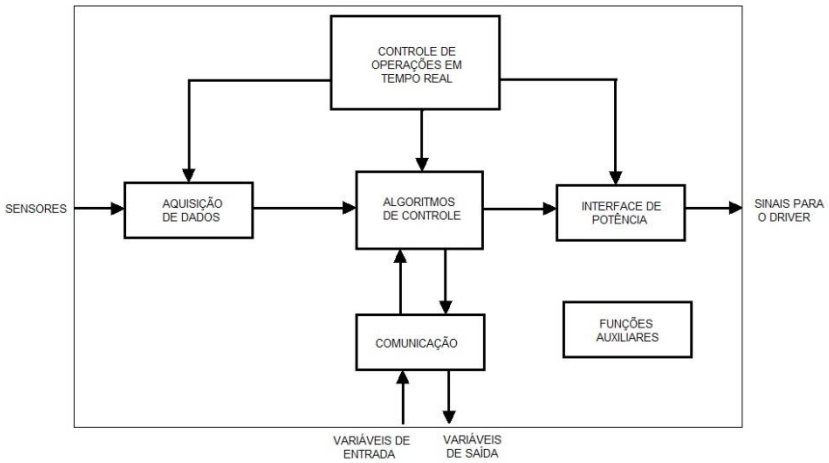

Figura 19 - Sistema de controle digital

A escolha do processador para aplicações em sistemas de controle do motor é crítica e devem-se considerar fatores tais como: tamanho de palavra e tipo de dados; velocidade de processamento; velocidade de processamento; recursos matemáticos, recursos de temporização e interrupção. Nestas aplicações, impõe-se aos processadores características especiais em vários aspectos incluindo:

- Operações matemáticas. Na implementação de filtros e algoritmos de controle, as operações mais utilizadas são as aritméticas e trigonométricas. Em estratégias mais complexas envolvendo, por exemplo, observadores de estado e controladores adaptativos, necessitam-se em alguns casos de operações matriciais.
- Operação em tempo real. Devem-se empregar os recursos de interrupção de forma a sincronizar o programa de controle com eventos externos. O tempo de latência da interrupção (intervalo entre a sua requisição e o início do atendimento) deve ser o menor possível comparado com o período de amostragem. Temporizadores são imprescindíveis para operações como a geração de sinais para o conversor de potência, medição periódica, modulação PWM, entre outros.
- Chaveamento de contexto. Esta é uma operação importante em controle motor multitarefa. O contexto de uma tarefa é caracterizado pelo estado dos registradores do processador bem como o endereço de partições especiais da

memória (tabelas, dados privados, etc.). O processador deve ser capaz de manipular a mudança de contexto num tempo mínimo evitando a degradação do desempenho.

- Recursos de comunicação. Isto é essencial na maioria dos sistemas de controle motor, para coordenar a operação de vários processadores.

2.5.1 Programa do microcontrolador no ambiente *MPLAB*

Para criar um código executável por um PIC MCU, o código fonte precisa estar inserido em um projeto, que é um caminho que os arquivos são organizados para serem compilados e estruturados.

Um projeto é criado usando o *Project Wizard* seguindo os seguintes passos:

- Seleciona-se o dispositivo, no nosso caso o Pic18f4431.
- Seleciona-se o conjunto de linguagem usados no projeto – Microchip C18 *toolsuite* (*plugin* para estruturar o programa em linguagem C).
- Nomeia-se o novo projeto e gravá-lo em uma pasta.
- Permite a seleção de arquivos ao projeto. Repare que o arquivo fonte não foi selecionado ou criado ainda. Há arquivos moldes que são simples arquivos usados para começar um projeto. Esses têm sessões essenciais para cada arquivo fonte e contem informações que irão ajudar a escrever e organizar o código.
- *Finish* - A última sessão do *Project Wizard* é um resumo mostrando o dispositivo selecionado, o conjunto de linguagem e o nome do arquivo do novo projeto.

Na janela do projeto, arquivos podem ser adicionados clicando com o botão direito do mouse nas pastas de arquivos.

Para gerar um programa executável no MCU criamos um novo arquivo fonte em *File →New* e salve este arquivo na mesma pasta onde foi criado o projeto (por exemplo: nomedoprojeto.c). Em seguida, clique com o botão direito do mouse em *Source Files* seleciona-se *add files* e seleciona-se o arquivo fonte salvo anteriormente. Adicione os arquivos correspondente ao Pic18F4431 clicando com o botão direito do mouse em *Header Files* (pic18f4431.h) e *Linker Script* (pic18f4431.gld) localizados dentro de pastas onde foi instalado o *MPlab* C18.

Para verificar erros na construção do projeto selecione *Project>Build All* para montar e conectar o código. Se o código está sem erros, na janela de saída (*Output*) a mensagem *BUILD SUCCEEDED*, caso contrário, aparecerá à mensagem *BUILD FAILED* e os erros serão apontados.

A programação ou para *debbugar* o PIC utiliza-se somente o *PICkit3* (*Programmer> Select Programmer> PICkit 3*) ou *PICkit2* dependendo do programador disponível.

Figura 20 - Robô *Very Small Size.*

3 Técnicas de controle investigadas para o robô

O controle de um processo só se dá definitivamente após as etapas de modelagem e análise. Esta seqüência está esquematizada na Figura 21 e apresenta o procedimento regular adotado em projetos de engenharia.

Quando a análise não fornece resultados compatíveis com a realidade, o modelo precisa ser aprimorado através de métodos de identificação. Quando o comando ou controle implementado não funciona, deve-se suspeitar de uma análise superficial ou de um modelo inadequado. Estas reavaliações estão sugeridas na Figura através das linhas de retorno.

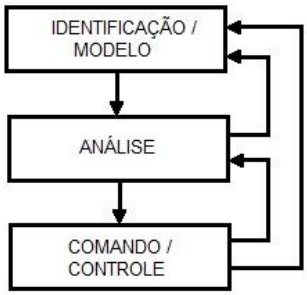

Figura 21 - procedimento de projetos em engenharia.

Os sistemas de comando ou controle à malha aberta exigem um conhecimento muito preciso do processo em estudo. Graças à realimentação, os sistemas de controle à malha fechada apresentam como vantagens:

- Rejeição de perturbações externas.
- Compensação de variações dos parâmetros do processo.

Como ponto negativo, os sistemas à malha fechada podem ser mais caros uma vez que, para sua implementação, são necessários:

- Sensores (transdutores)
- Controladores
- Atuadores, que convertem os sinais de baixa potência dos controladores em entradas do processo.

O projeto de controlador também exige técnicas especiais. Intuitivamente, percebe-se que o sinal de erro obtido pela diferença entre um sinal de referência desejado

e a atual saída do processo permitirá que se tomem as ações adequadas para obter os sinais de entrada do processo. No entanto, o processamento do sinal de erro, se não for corretamente escolhido, pode ser catastrófico para o desempenho do sistema realimentado. Vários exemplos da vida cotidiana servem para ilustras estes inter-relacionamentos. A forma como este desvio é processado e as ações daí resultantes podem fazer com que o sistema se recupere ou se perca totalmente.

Além da condição de estabilidade que se impõe como pré-requisito de qualquer projeto, outras características permitem definir o comportamento dinâmico de um sistema linear. Usualmente, para uma entrada em degrau, quantifica-se a resposta dinâmica através do tempo de subida (t_r), do tempo de assentamento (t_s), do tempo de pico(t_p) e do sobresinal (M_p), apresentados na Figura 22 para um sistema com erro de regime zero.

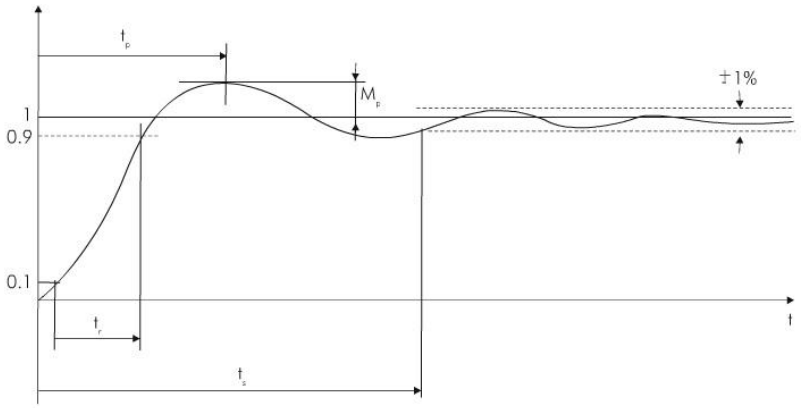

Figura 22 - Características de desempenho para uma entrada em degrau unitário e sistema com erro de regime zero.

Para os sistemas lineares, que é o caso do nosso processo, pode-se utilizar uma técnica clássica de projeto de controladores chamado lugar das raízes, tal técnica permite que, a partir do conhecimento do processo a ser controlado, se escolha um controlador de tal forma que o sistema realimentado apresente um comportamento pré-estabelecido, como será repassada no próximo item:

3.1 Lugar das raízes

O método do Lugar das Raízes foi proposto por W.R. Evans em 1948 e permite determinar a posição dos pólos de um sistema realimentado a partir do conhecimento dos pólos e zeros do sistema à malha aberta e em função do ganho da malha.
Para o entendimento do método, considere-se o sistema da Figura 23:

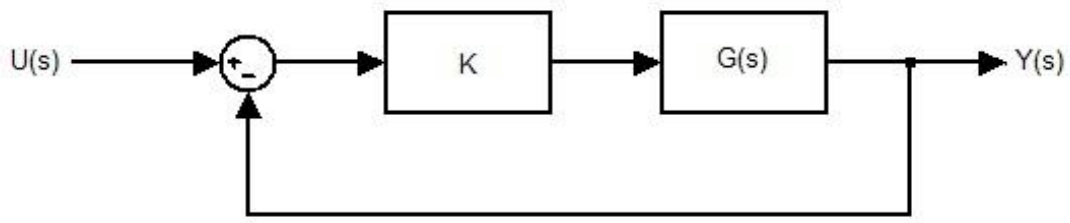

Figura 23 - Estrutura básica para o entendimento do lugar das raízes

A função de transferência do sistema à malha fechada vale:

$$\frac{Y(s)}{U(s)} = \frac{K \cdot G(s)}{1 + K \cdot G(s)}$$

O método permite determinar a posição das raízes de 1+KG(s) = 0 a partir do conhecimento de G(s).

Um breve exemplo servirá para ilustrar esta ferramenta. Tomando $G(s) = \frac{1}{s \cdot (s+2)}$, a função de transferência à malha fechada vale: $G_{MF}(s) = \frac{K}{s^2 + 2s + K}$.

Os pólos desta função de transferência encontram-se em:

$$S_{1,2} = -1 \pm \sqrt{1 - K}.$$

Para 0 ≤ K ≤ 1, as raízes são $S_{1,2} = -1 \pm \sqrt{1 - K}$.

Para K>1, as raízes são $S_{1,2} = -1 \pm j\sqrt{1 - K}$.

Este resultado está apresentado graficamente na Figura 24:

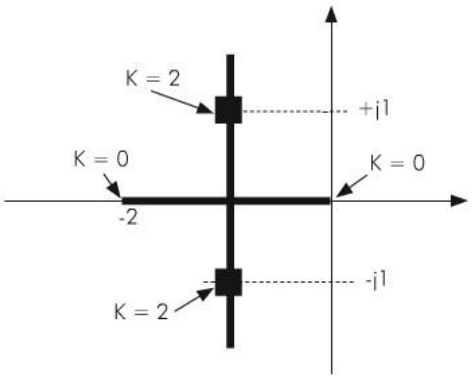

Figura 24 - Exemplo lugar das raízes

As raízes do sistema à malha fechada assumem posições diferentes do plano complexo em função do valor de K. Chama-se Lugar das Raízes (LR) o diagrama que apresenta o lugar que as raízes do sistema realimentado acupam no plano complexo em função de K.

Evans estabeleceu uma série de regras para o traçado deste lugar geométrico sem a necessidade do cálculo das raízes, como foi feito no exemplo anterior. Atualmente, existem vários programas de computador que fazem este cálculo. No entanto, é útil conhecer as regras mais simples, uma vez que a partir delas já se torna possível esboçar algumas curvas.

Considerando $G(s) = N(s) / D(s)$, as raízes de $1+KG(S)$ são as raízes de $D(s) + KN(s) = 0$. Assim, para K=0, esta igualdade reduz-se a D(s) = 0. Os valores de "s" que atendem esta condição são os pólos do sistema à malha aberta (regra1).

Para K $\rightarrow \infty$, a igualdade será satisfeita se N(s) = 0. Os valores de "s" que satisfazem esta igualdade são os zeros do sistema à malha aberta (regra2). Concluí-se, assim, que o LR inicia nos pólos do sistema à malha aberta e termina nos zeros e que existem tantos ramos quantos são os pólos.

Como usualmente o número de pólos de um sistema é o maior do que o número de zeros, as regras 1 e 2 sugerem que alguns ramos devem tender a infinito quando K $\rightarrow \infty$, pois esta seria uma forma de também atender à equação D(s) + KN(s) = 0.

Demonstra-se que estes ramos se encontram em um ponto do eixo real dado por:

$$\alpha = \frac{\sum_1^{n_p} p_i - \sum_1^{n_z} z_i}{n_p - n_z} \text{ (regra3),}$$

em que P_i são os pólos do sistema à malha aberta e z_i seus zeros , N_p o número de pólos e n_z o número de zeros.

Estas $n_p - n_z$ assíntotas formam com o eixo real ângulos dados por:

$$\phi_i = \frac{180^o + 360^o(i-1)}{n_p - n_z}, \text{ j = 1,2,...,}(n_p - n_z) \text{ (regra4)}$$

Para os demais valores de K, considerando K > 0, verifica-se que:

$$\angle G(s) = \angle -1 = -180^o.$$

Esta simples relação permite concluir que existirão raízes sobre o eixo real sempre que existir um número ímpar de pólos mais zeros à direita do ponto considerado (regra5). Por outro lado, se $\angle G(s) = -180^o$ então $\angle G(s^*) = 360^o - \angle G(s) = -180^o$.

Portanto, o LR é simétrico em relação ao eixo real (regra6).

Quando dois ramos do LR se encontram em um ponto do eixo real, os ramos explodem para o plano complexo com ângulos de ± 90°. O exemplo anterior ilustrou este fato (regra7).

Numerosos processos podem ser aproximados como possuindo dois pólos dominantes. A função de transferência parametrizada em termos do coeficiente de amortecimento (ς) e da freqüência natural não amortecida (ω_n) permite o estabelecimento de critérios de projeto com base no LR. Assim, para H(s) dado por:

$$H(s) = \frac{\omega_n^2}{s^2 + 2\varsigma\omega_n s + \omega_n^2}$$

Pode-se obter a resposta ao degrau unitário e apresentá-la com o tempo normalizado ($\omega_n t$) e parametrizado em função de ς (Figura 25).

Os pólos deste sistema são dados por: $S_{1,2} = -\varsigma\omega_n \pm \omega_n\sqrt{(\varsigma^2 - 1)}$.

E apresentados no plano complexo na Figura 26 para $\varsigma \leq 1$.

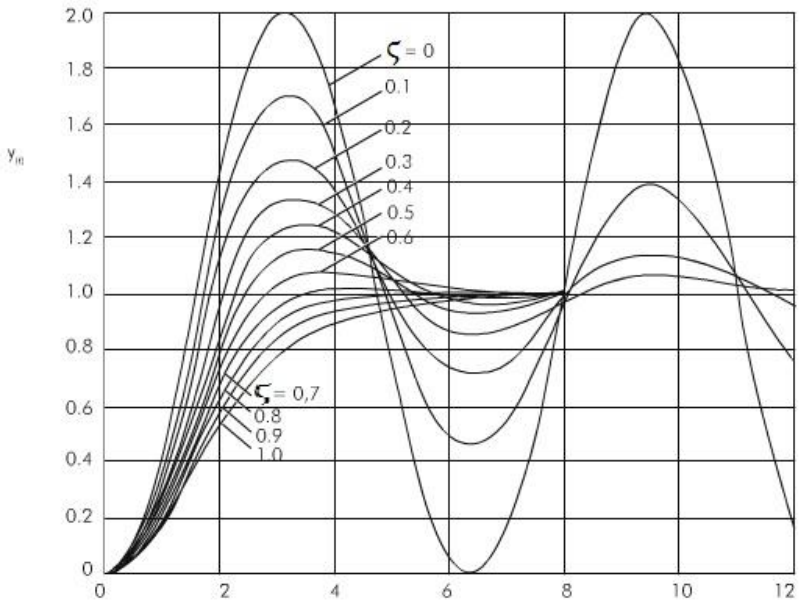

Figura 25 - Respostas ao degrau para um sistema de segunda ordem

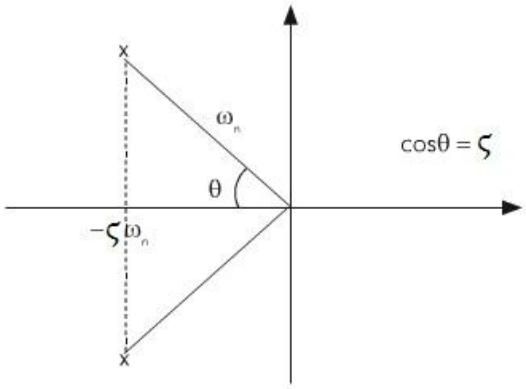

Figura 26 - Posição dos pólos do sistema de segunda ordem.

De um modo geral, a posição dos pólos e a resposta no tempo podem ser qualitativamente relacionadas como na Figura 27:

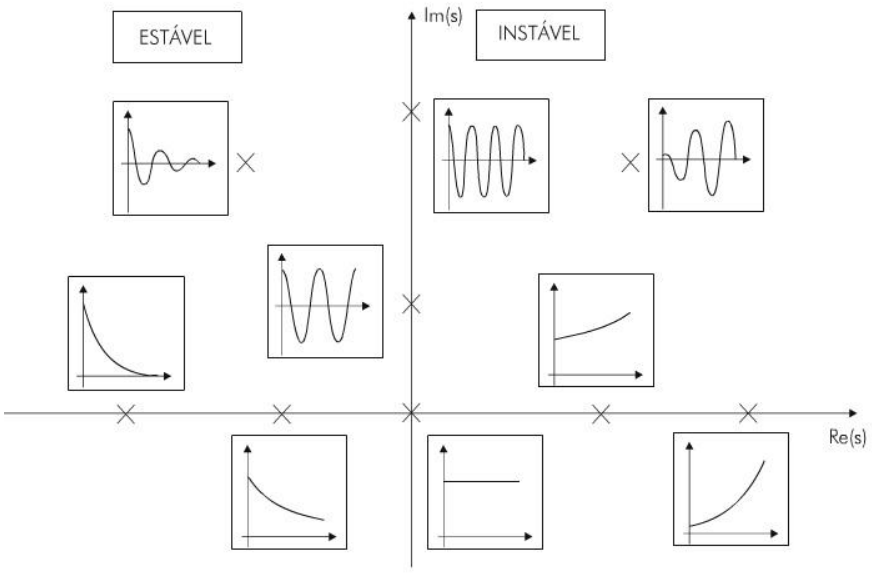

Figura 27 - Posição de pólos e resposta no tempo.

Deste conjunto de observações, percebe-se que a resposta no tempo pode ser inferida a partir do posicionamento dos pólos dominantes.

A partir de determinada especificação dada em termos de sobrepasso ou tempo de assentamento, pode-se delimitar uma região do plano complexo onde devem se situar as raízes dominantes do sistema realimentado.

Uma vez delimitada esta região, cabe ao projetista, engenhosamente, encontrar o compensador e o ganho da malha de controle de modo que as raízes fornecidas pelo traçado do LR se encontrem na região pré-estabelecida.

Para esta tarefa, o auxílio propiciado por programas de computador facilita extremamente o trabalho. Por exemplo, no *MATLAB*, existe disponível a ferramenta *RLTOOL*. Diferentes tipos de compensadores podem ser testados, o valor do ganho variado e a resposta no tempo observada.

O projetista, no entanto, precisa de uma boa noção do que está sendo calculado. Assim, o conhecimento das regras básicas do LR ajuda bastante. Por exemplo, se for necessário trazer as raízes do sistema realimentado para a esquerda do plano complexo, a regra2 ensina que se deve introduzir um zero na malha aberta. Em outras palavras, isto significa um compensador PD, cuja função de transferência é dada por:

$$G_r(s) = K \cdot (1 + T_o \cdot s)$$

3.2 Controle de velocidade

Primeiramente, precisa-se conhecer a planta do minimotor DC da *Faulhaber* (2342006 CR) o que pode ser obtida a partir dos parâmetros elétricos e mecânicos do motor fornecidos pelo fabricante no *datasheet* ou então usar outro método alternativo que leva em consideração a mecânica da base robótica (eixos, redução e rodas) que consiste em aplicar um degrau de tensão no motor e transmitir a leitura que o *encoder* faz da velocidade correspondente via rádio para o terminal do receptor do rádio plugado a um computador.

Com esses dados enfileirados em um arquivo, pode-se utilizar uma ferramenta de identificação de sistemas do *software Matlab - Ident* para identificação da função de transferência do motor, representada na Figura 28. Para isso, devem-se seguir os seguintes passos:

- Gerar um vetor coluna com os dados da planta do motor;
  Se o arquivo já tiver sido salvo em um arquivo de texto (exemplo: plantamotor.txt) com os dados seqüenciais, basta gerar o vetor na Janela de Comando ou no *Editor* da seguinte forma motor=load('plantamotor.txt') e no *Workspace* será armazenado um vetor com o nome 'motor' contendo os dados do motor, outra maneira, mas que deixaria o *Editor* com muitas linhas de código é fazer um vetor com os dados do motor da seguinte forma motor=[dado1 ;dado2 ;dado3 ; ... ;dadon] muitas vezes inviável.
- Gerar um vetor que represente o valor do nível de tensão utilizado no degrau.
  Uma alternativa, utilizando a seguinte linha de comando depois que gerado o vetor anterior:

```
for i=1:length(motor)
    tensaoentrada(i) =6.0;
end
```

- Na Janela de Comando digite *ident* e uma janela desse *toolbox* irá aparecer;
- Na janela *Import data>Time Domain Signals* para carregar os vetores alocados no *Workspace*;
- Em *Workspace Variable*, na janela *Input* coloca-se o nome do vetor que representa o nível de tensão (tensaoentrada) e em *OutPut* coloca-se o vetor dos dados do motor (motor);

- Em *Data Information*, na janela *Data name* atribui-se um novo nome para os dados do motor, em *Starting Time* coloca-se '0' (valor zero numérico sem aspas) e em *Sampling Interval* coloca-se o tempo que o programa gravado no Pic demora a fazer uma iteração ou um ciclo de máquina, no nosso caso, o tempo estimado foi 3.96ms.
- Clique em *Importe*; se ocorrer algum erro uma mensagem irá aparecer descrevendo qual foi o erro caso contrário clique em *Close*.
- Selecione a opção *Time Plot* e uma janela com os valores dos sinais de entrada e saída irão ser plotados em gráficos diferentes.
- Na janela *Estimate>Process Models* e uma janela aparecerá onde será definido o modelo da função de transferência do motor DC, que possui uma planta praticamente de primeira ordem, então seleciona-se a opção para que a planta tenha um pólo real e retira-se o *Delay* selecionado.
- Clique em *Estimate* e depois *Close* para fechar essa janela.
- Selecione a opção *Model Output* para visualizar o modelo calculado e simulado.
- Selecione o quadro onde foi gerado o modelo do processo e as informações da função de transferência da planta irá a aparecer da seguinte forma: $G(s) = \frac{Kp}{1+Tp1 \cdot s}$ representando um sistema de primeira ordem.

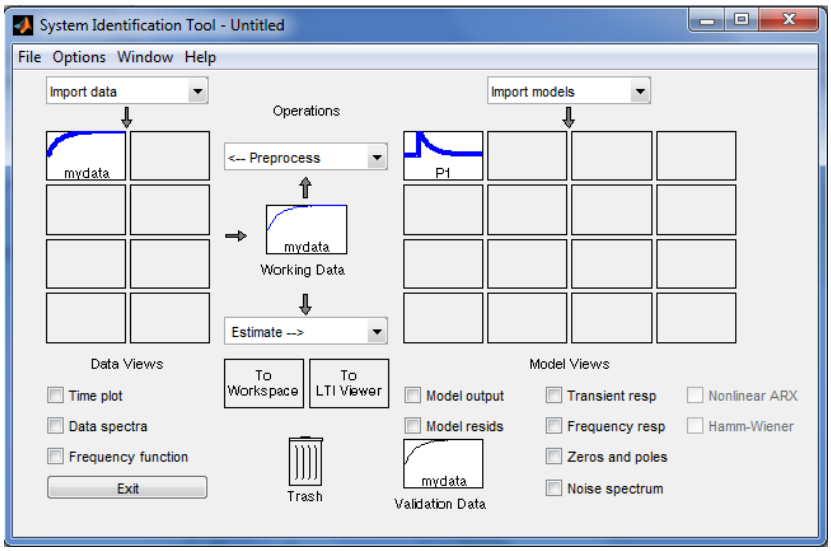

Figura 28 - Janela *System Identification* do *toolbox IDENT* do *Matlab*.

Determinada a função de transferência do motor DC, passa-se a modelar o acionamento PWM do motor DC para controle de velocidade que são os sinais enviados a

placa de *driver's*. O PWM é utilizado para variar o valor da transferência de potência entregue a uma carga sem as perdas ocorridas normalmente devido à queda de tensão por recursos resistivos. Nesse sistema PWM, a chave de estado sólido (IGBT, MOSFET ou transistor bipolar) usada para controlar o fluxo de corrente: ora conduzindo corrente, mas provocando uma queda de tensão muito baixa.

Como se sabe, uma variação na tensão do motor DC acarreta uma variação na velocidade do mesmo. E através de uma estratégia PWM pode-se conseguir o mesmo efeito através da variação do valor médio V_m, atuando-se na variação da largura de pulso T_{on}. Supondo que a chave dos dispositivos semicondutores opera periodicamente com um período T, de modo que seja mantida fechada por um período T_{on}, e aberto durante um período T_{off}, e $T = T_{on} + T_{off}$, a forma de onda da tensão instantânea $v_0(t)$ nos terminais do motor é $V_m = V \cdot \frac{T_{on}}{T}$.

Passa-se então ao projeto do controlador digital que conduza o sistema a um desempenho satisfatório. No domínio do tempo, as especificações desejadas podem ser definidas em termos do máximo sobresinal ou coeficiente de amortecimento, do tempo de acomodação para uma entrada degrau e do erro de regime, as quais podem ser descritas em termos de pólos e zeros do sistema a malha fechada. Pode-se assim utilizar o método do lugar das raízes para determinar a função de transferência do controlador para obter a configuração desejada de pólos e zeros.

Ressaltamos que o controlador digital pode ser obtido a partir de um controlador analógico (contínuo) já pré-calculado. Em seguida, discretizado para a obtenção da lei de controle discreta ou digital. Sendo que a transcrição da formulação de um controlador (ou processo) do domínio de *Laplace* para domínio do plano-Z deve seguir a definição da transformada-Z:

$$s = \frac{1}{T_o} * ln\,(z) \qquad \text{ou} \qquad z = e^{T_o * s}$$

No outro método, procede-se o mapeamento direto do plano 's' no plano 'z' pela definição da transformado –Z. Portanto um sistema assintoticamente estável no modo contínuo só será também assintoticamente estável se os pólos discretos situarem no interior do círculo unitário do plano-z, como representado na Figura 29.

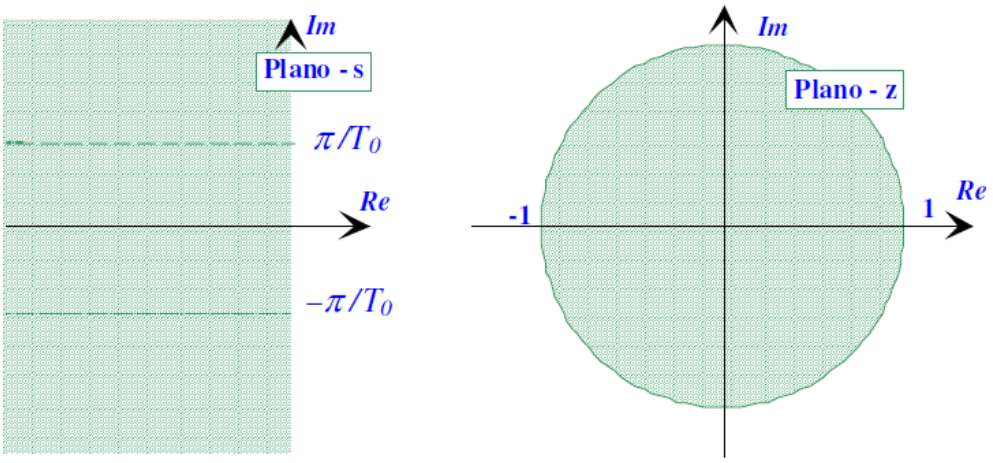

Figura 29 - Representação do mapeamento (s → z) pela definição.

3.2.1 Controlador PI digital convencional

No caso contínuo, uma ação de controle PI é representada como:

$$U(t) = K\left[e(t) + \frac{1}{T_1} * \int e(\tau)d\tau\right]$$

No caso discreto, admitindo-se que o tempo de amostragem é pequeno, a ação de controle pode ser descrita em forma de uma equação diferença, aproximando-se a integral pelo método retangular, tal como:

$$u(k) = K\left[e(k) + \frac{T_0}{T_1} * \sum_{v=0}^{k-1} e(v)\right]$$

e uma forma recursiva pode ser encontrada como sendo dada por:

$$u(k) - u(k\text{-}1) = q_0 * e(k) + q_1 * e(k\text{-}1)$$

com $q_0 = K$ e $q_1 = -K*(1 - T_0/T_1)$

Cuja respectiva função de transferência discreta para o controlador PI será:

$$G_{PI}(z) = \frac{U(z)}{E(z)} = \frac{q_0 + q_1 * z^{-1}}{1 - z^{-1}}$$

A ferramenta do *Matlab – Rltool*, cuja janela de comando pode ser vista na Figura 30, proporciona projetar o controlador PI alterando os pólos e ganho da função de

transferência no plano-z e visualizar o comportamento do sistema com uma entrada degrau a malha fechada.

Procedimento de ajuste de compensadores com o recurso *RLTOOL*:

- Designar o processo contínuo ou discreto a ser estudado;
- Inicializar o recurso digitando *rltool* na janela de comandos;
- Na janela *Control and Estimationa* escolha *Edit>SISO Tool Preferences* e mude o modo padrão de *Options* para ZERO/Pole/Gain;
- Nessa mesma janela selecione a Aba *Architecture* e clique no campo *System Data* para carregar a função de transferência do motor do *Matlab Workspace.*
- Nessa mesma janela selecione a Aba *Analysis Plots* e escolha as opções *Plot1→Step* e selecione o *Close Loop R to Y.* O que fará aparecer uma nova janela com a resposta ao degrau.
- Nessa mesma janela selecione a Aba *Compensator Edit* e no campo *Dynamics* use o botão direito do mouse para adicionar Pólos e/ou Zeros do Compensador. Nesta aba "C" representa o controlador a ser inserido em série com o processo. Imediatamente a frente de "C" é possível ajustar ou visualizar o Ganho da FT do Controle.
- Usa-se a Janela *SISO Design* para a posição dos zeros e ganho do controlador apropriado a cada caso, visualizando simultaneamente na Janela *LTI Viewer* a resposta ao degrau do sistema realimentado.
- Depois de pronto, na Aba *Architecture* da janela *Control and Estimation* selecione *Store Design* e verifique no diretório à esquerda o resultado geral do projeto;
- Pode-se ainda salvar no *Workspare* o controlador projetado escolhendo-se em *File>Export* cada um dos blocos do projeto.

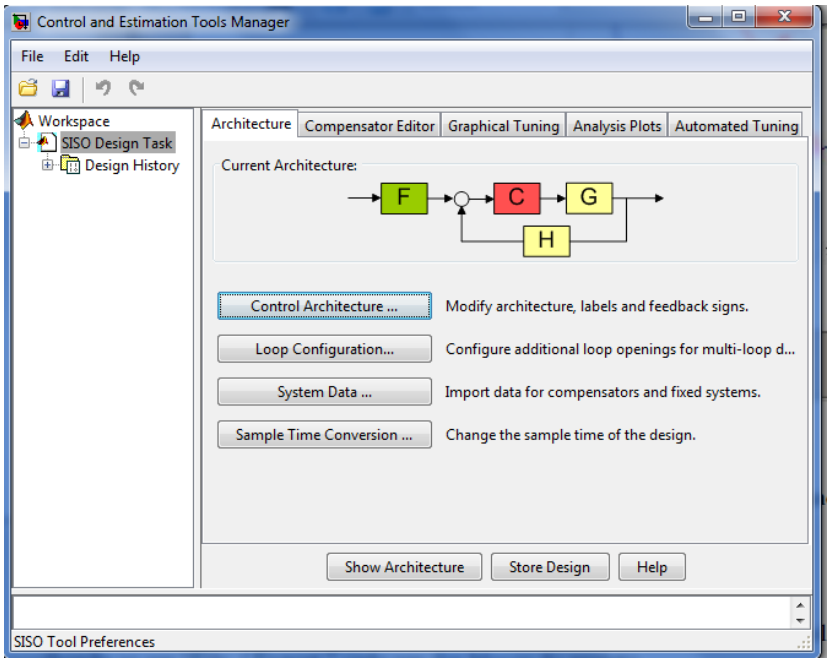

Figura 30 - Janela *Control and Estimationa* do *toolbox RLTOOL* do *Matlab*.

Com todos os blocos determinados, podemos simular o sistema no *Simulink* do *Matlab*, que nos permite uma melhor visualização do sistema e melhor análise dos sinais para evitar uma saturação indesejada ou mesmo a simulação de ruídos.

Digitando *simulink* na janela de comandos ou clicando no ícone do *Simulink* na barra de ferramentas do *Matlab*, irá abrir uma janela: *Simulink Library Browser*. Para criar um arquivo desse toolbox *File>New>Model* e lembrando-se de salvar esse arquivo na mesma pasta atual onde estavam salvos os arquivos.m. Assim um esquemático em blocos será montados com componentes localizados na biblioteca na janela *Simulink Library Browser,* como representado na Figura 31.

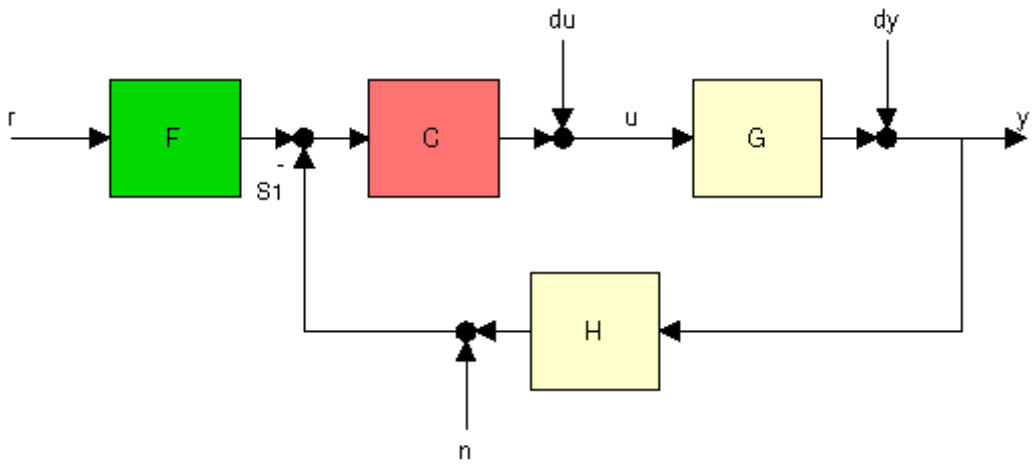

Figura 31 - Representação do controle de velocidade do motor DC.

3.2.2 Controlador PIncremental *fuzzy*

- **Lógica *fuzzy***

A teoria de conjuntos *Fuzzy* tem sido empregada com sucesso para exprimir conhecimento impreciso e resolver problemas em muitas áreas, onde o modelamento convencional é difícil, ineficiente ou muito oneroso. A possibilidade de descrição lingüística do modelo, ao invés de utilização das equações diferenciais, possibilita o aproveitamento do conhecimento heurístico dos operadores e facilita o desenvolvimento de soluções.

A estrutura de universo de discurso, variáveis lingüísticas, fuzzificação, banco de regras, máquina de inferência e sistema de defuzzificação proposta por Mandani, quando bem assimilada, é uma arma poderosa de simplificação e aumento da velocidade de processamento e robustez do controlador, possibilitando decisões rápidas e coerentes num ambiente de incertezas. A lógica *Fuzzy* é uma técnica comparativamente simples e de vasto espectro de aplicabilidade, em particular a problemas de controle e de decisão. A Figura 32 ilustra em blocos o controlador *Fuzzy* proposto por Mandani:

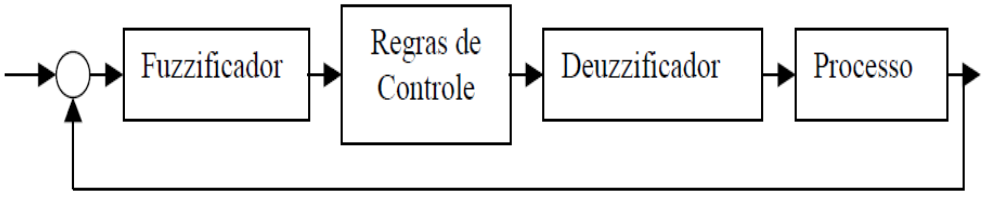

Figura 32 - Lógica *fuzzy*

Portanto, o projeto de um controlador *fuzzy* consiste em:

1. Definir os universos de discurso das variáveis lingüísticas do sistema, erro, variação do erro e variação da saída do controlador (discretização ou níveis de quantização);

2. Definição do número de termos primários e graus de pertinência dos conjuntos difusos que representam cada termo;

3. Determinação das regras que formam o algoritmo de controle;

4. Definição de parâmetros de projeto, como método de inferência, lógica a ser empregada, forma de defuzzificação e atuação do controlador

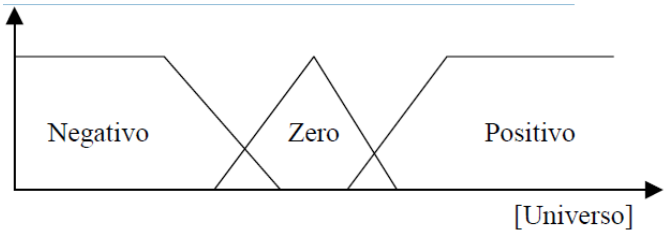

Figura 33 - Funções de pertinência

- ***O processo de fuzzificação***

O método escolhido para criar as variáveis lingüísticas foi o proposto em. O grau de participação de uma determinada grandeza de entrada é dado em função dos termos primários definidos para o universo de discurso da entrada. Os termos com formas simples tais como trapézios e triângulos são geralmente utilizados para representar as funções de grau de participação, porém, qualquer tipo de função pode ser utilizado. O número de termos primários a serem utilizados e suas formas depende da precisão requerida, tipo de resposta e estabilidade do sistema, facilidade de implementação, manuseio e manutenção, entre outros. A Figura 34 apresenta uma representação típica de um termo primário.

Esta função pode ser representada por apenas cinco variáveis, ao invés da tradicional função de participação mostrada acima. Sabemos que para os valores do universo de discurso abaixo de p1 e acima de p2, o grau de participação é zero. Por outro lado, a partir dos valores de Delta1*alfa1, Delta2*alfa2, e do limite superior, determinando qual destas três grandezas apresenta o menor grau de participação, chegamos ao valor fuzzificado da função. Se chamarmos a função acima de *índice de jovialidade*, representaremos o valor fuzzificado como "*índice de jovialidade* = f(x)", onde f(x) é o grau de participação e x é o valor inteiro não fuzzificado (*crisp*) no universo de discurso.

Regra para fuzzificação:

Se (x – p1) ou (p2 – x) ≤ 0
Então f(x) = 0;
*Senão f(x) = min(Delta1*alfa1, Delta2*alfa2, limite superior)*
Fim se;

Desta forma o processo de fuzzificação se torna:

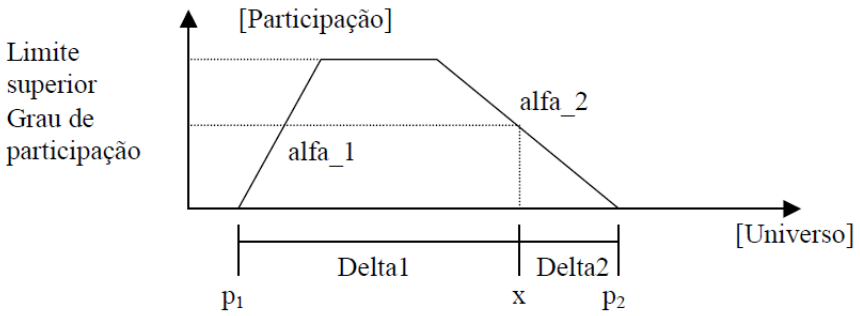

Figura 34 - Processo de fuzzificação.

1. A definição dos termos primários e dos cinco valores associados a cada termo primário no início do programa. Por exemplo:

   Negativo: p1, p2, inclinação1, inclinação2, limite superior;
   Zero: p1, p2, inclinação1, inclinação2, limite superior;
   Positivo: p1, p2, inclinação1, inclinação2, limite superior;

2. Durante a execução do programa, cada valor de entrada deve ter seu grau de participação determinado para cada termo primário definido, segundo a regra de fuzzificação dada acima. Por exemplo, sejam os valores de entrada *erro* e *taxa_do_erro*, para os termos primários definidos acima,

   er_N = Negativo (erro);
   er_Z = Zero (erro);
   er_P = Positivo (erro);
   taxa_er_N = Negativo (taxa_do_erro);
   taxa_er_Z = Zero (taxa_do_erro);
   taxa_er_P = Positivo (taxa_do_erro);

onde er_N significa o grau de participação do valor de erro na função Negativo segundo a regra acima, e assim sucessivamente.

3. Posteriormente estes valores serão combinados na máquina de inferência para produzir uma saída *fuzzy* que deverá ser convertida num valor *crisp* na saída.

❖ **Máquina de inferência.**

Para controlarmos adequadamente um processo, precisamos codificar o conhecimento que temos sobre o mesmo na forma de regras "Se *antecedente* então *conseqüente*" e de um mecanismo que avalie quais regras são pertinentes, e as aplique produzindo uma saída (*conseqüente*). No caso do controlador, o antecedente é função do erro (erro) e da taxa de variação do erro (delta_er). O valor a ser obtido na saída será dado pelo valor do menor grau de participação entre os antecedentes (erro e delta_er), ou seja, pela interseção dos termos lingüísticos correspondentes ao erro e a variação do erro. O menor grau de participação será o valor do grau de participação na saída. Exemplo:

Se erro = Negativo e delta_er = Negativo então Saída = Positivo.

Sendo o grau de participação de erro = 0,7 (Negativo) e delta_er = 0,5 (Negativo) então a saída será Positiva = 0,5.

Se tivermos mais do que uma regra com o mesmo termo lingüístico na saída (exemplo: Positivo), a prática comum é usar o maior grau de participação (união de conjuntos *fuzzy*). Exemplo:

R1: Se erro = Negativo (0,7) e delta_er = Negativo (0,5) então saída = Positivo

R2: Se erro = Negativo (0,7) e delta_er = Zero (1,0) então saída = Positivo

A regra R1 produzirá saída=Positivo (0,5) e a regra R2 produzirá saída=Positivo (0,7).

A combinação das duas regras nos dá saída = Positivo (0,7).

Resumindo: a operação "e" lógico (interseção) de fatos "a" e "b" é resolvida através do valor mínimo entre "a" e "b"; a operação "ou" lógico (união) de regras "a" e "b" é resolvida através do valor máximo entre "a" e "b". Na prática, as operações para se obter o algoritmo de controle consistem nos seguintes procedimentos:

- Para cada fator da parte “Se” da regra: obter um grau de participação do valor de entrada para cada função de associação ou termo lingüístico;
- O mínimo valor dos graus de participação obtidos em 1 é o grau de participação da parte “Se” (antecedente);
- Aplicar um limitador na função de associação da parte “então” através do fator obtido;
- Repetir de 1 a 3 para todas as regras e obter a soma lógica “OU” das funções de associação dadas pelas partes “então” (conseqüente) de cada regra.
- Calcular a defuzzificação da função de associação resultante e obter o valor de saída.

❖ **O processo de defuzzificação**

No estágio do defuzzificador, a variável difusa produzida pela máquina de inferência é transformada em variável numérica (determinística) que atuará no processo de forma a regulá-lo.

O termo defuzzificador equivale à transformação *fuzzy* - escalar, correspondendo a um mapeamento do espaço de ações de controle *fuzzy* e definido sobre o universo de discurso para o espaço de ações não *fuzzy* ou escalares.

De forma a produzir um valor numérico para aplicação ao processo, as variáveis lingüísticas (conjunto) produzidas pela máquina de inferência devem ser defuzzificadas. Basicamente são mais utilizados os seguintes métodos:

- Método do critério máximo: esse método produz como ação de controle, o valor numérico da saída correspondente ao índice da variável lingüística de saída, produzida pela máquina de inferência, de maior grau de pertinência.
- Método da média dos máximos: O valor numérico da saída corresponde ao índice referente à medida dos máximos locais da variável lingüística de saída produzida pela máquina de inferência.

$$D = (X1 + X2) / 2$$

- Método do centro de gravidade: é o método mais utilizado, e se baseia no cálculo do centro de gravidade da função de associação. No método do centro de gravidade, calcula-se a área da curva da variável lingüística de saída produzida

pela máquina de inferência, e acha-se o índice correspondente que divide esta área a metade.

$$D = \frac{\sum_{i=1}^{n} \mu.A(x_i).x_i}{\sum_{i=1}^{n} \mu.A(x_i)}$$

onde ‘n’ ‘e o número de níveis de quantização.

- Método do *Singleton*: este método é usado algumas vezes para simplificar o processo de defuzzificação. Um *Singleton* é uma função de saída com um grau de participação representada por uma única linha vertical. Esta linha passa pelo centro de gravidade do termo lingüístico para a participação máxima. Uma vez que um *Singleton* intercepta o eixo x em um único ponto, o cálculo do centro de gravidade se reduz apenas ao cálculo da média ponderada dos valores de x para cada *Singleton* e de seu grau de participação.

$$Saída = \frac{Singleton\_termo1 * Vlr\_\max\ \_termo1 + Singleton\_termo2 * Vlr\_\max\ \_termo2}{Vlr\_\max\ \_termo1 + Vlr\_\max\ \_termo2}$$

A literatura disponível tem mostrado que os controladores *Fuzzy* vem sendo aplicados com sucesso no controle de processos onde os controladores convencionais falham ou não exibem um bom desempenho devido, principalmente, a um conhecimento impreciso do comportamento dinâmico e dos parâmetros do processo.

4 Implementação e resultados

4.1 Obtenção do modelo do motor CC:

No processo de levantamento da função de transferência do mini-motor CC, optou-se por utilizar a própria placa eletrônica de controle do robô *Very Small Size* para aquisição de dados. Sendo que tal placa possui comunicação UART com um rádio operante tanto em modo transmissor como receptor de módulo XBee (Zigbee IEEE 802.15.4), pelo qual se enviou informação da velocidade do motor a cada ciclo de máquina (tempo de amostragem utilizado t=3,96ms) na resposta do motor a um degrau de tensão de 7,6V (apesar de ser um motor de 6V nominal). Esse valor da velocidade é obtido através um de *encoder* acoplado ao motor *Faulhaber*.

Um adaptador USB para o módulo XBee (funcionando agora como receptor) é conectado a um PC que através do *software* de interface X-CTU recebe os dados em seu terminal de forma serial, e um arquivo texto contendo esses valores é gerado para depois serem acessados e manipulados em um *software* que calculará uma função de transferência no plano 's' (Transformada de *Laplace*) representando a dinâmica do motor. Para serem acessados pelo *software Matlab* pode-se: atribuir diretamente a uma variável de uma ou duas dimensões ou acessando os dados do arquivo texto utilizando a função '*load'* do *Matlab*, lembrando que para isso, o arquivo deve ser salvo no mesmo diretório de um programa principal obtendo assim um vetor ou matriz dos dados contidos nesse arquivo.

Como exemplo de acesso: para um arquivo salvo como motor.txt, no editor do *Matlab* pode-se acessar os dados com o seguinte comando: dadosmotor=load('motor.txt'). Esses dados são tratados através do *toolbox* '*Ident'* do *Matlab*, pelo qual é gerada uma modelagem aproximada da função de transferência do sistema. A seguinte parte do código abaixo exemplifica como os dados são carregados no editor a partir de um arquivo de texto para se obter a Figura 35 que representa as funções de transferência dos dois motores do robô como resposta ao degrau.

```
dadosmotor=load('motor.txt');          %carrega os valores de velocidade
t=3.96*10^-3;                          %tempo de amostragem
x=0:t:t*(length(dadosmotor)-1);        %vetor representando o degrau de tensão
figure(1)
%representa a resposta de um dos motores do robô, lado esquerdo da Figura 35
subplot(1,2,1),plot(x,dadosmotor(:,1)*pi/(30));
title(['motor faulhaber']);
xlabel('t [s]'); ylabel('Velocidade [rad/s]');
%representa a resposta do outro motor do robô, lado direito da Figura 35.
subplot(1,2,2),plot(x,dadosmotor(:,2)*pi/(30));
title(['motor direita']);
xlabel('t [s]'); ylabel('Velocidade [rad/s]');
```

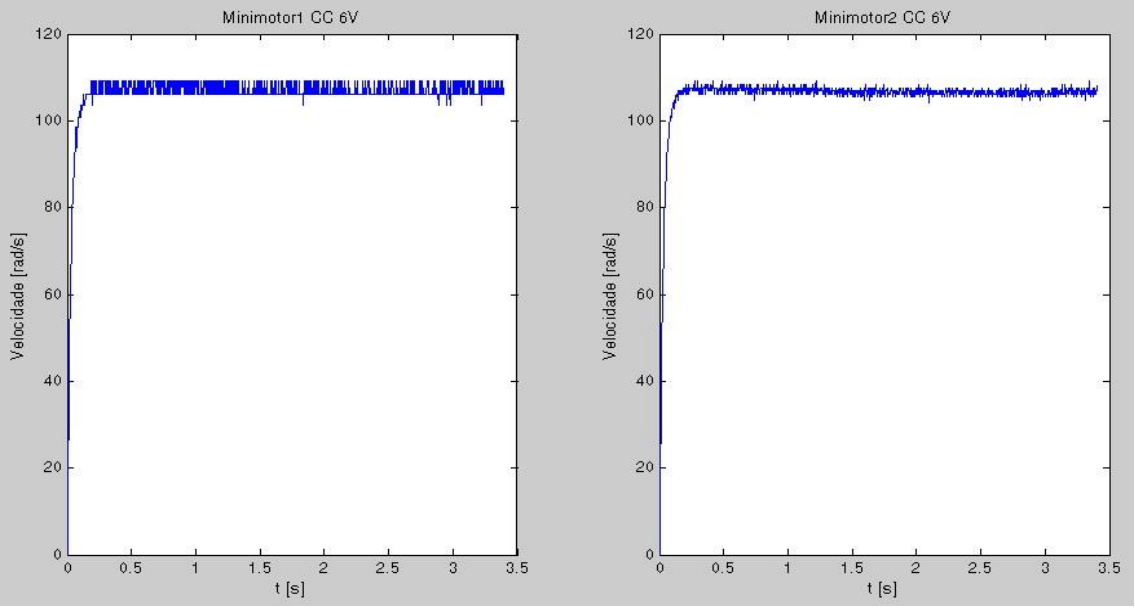

Figura 35 - Dados do valor da velocidade dos dois motores do robô ao aplicar um degrau de 1V.

Observa-se visualmente através dessa Figura 35 que as respostas dos mini-motores CC são bem parecidas e suas velocidades em regime iguais.

Já no ambiente do *toolbox Ident* do *Matlab*, os dados de entradas e saídas são carregados para um dos motores como representado na Figura 36:

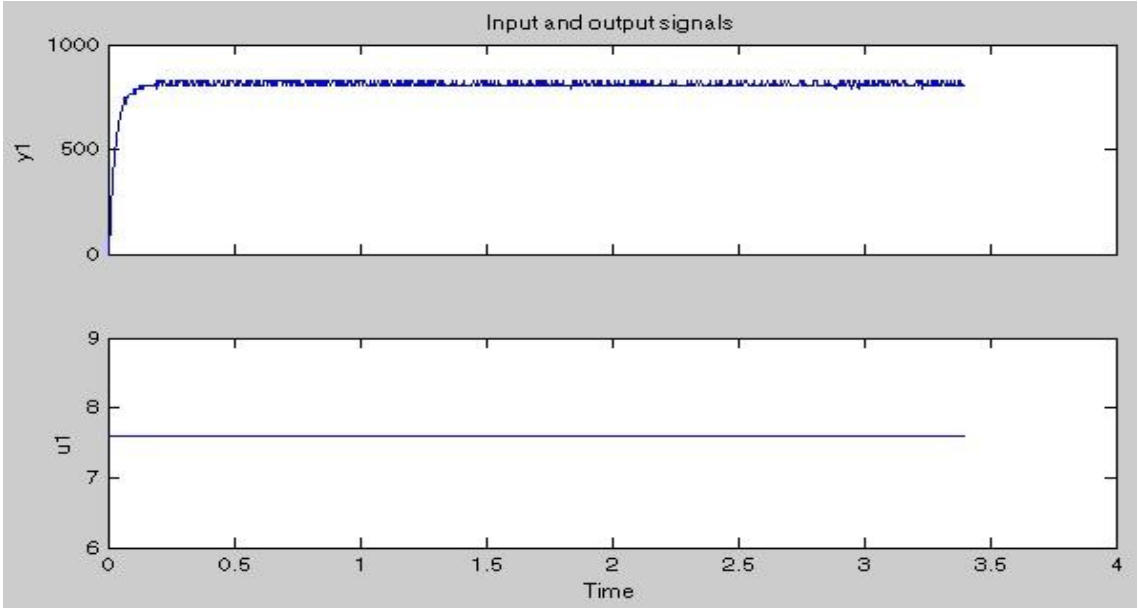

Figura 36 - Representação dos dados de saída(velocidade) e entrada(tesão) no tempo.

Apesar de não estarem representadas no gráfico, as escalas são:

- u1: [V] representando a entrada de tensão na planta.
- y1: [rad/s] representando a saída em velocidade da planta.
- time: [s] representando o tempo.

Em seguida, selecionou-se um método de identificação do sistema como descrito passo a passo na parte teoria e assim a seguinte curva foi traçada representada na Figura 37 e com valores de sua função de transferência mostrados na Figura 38:

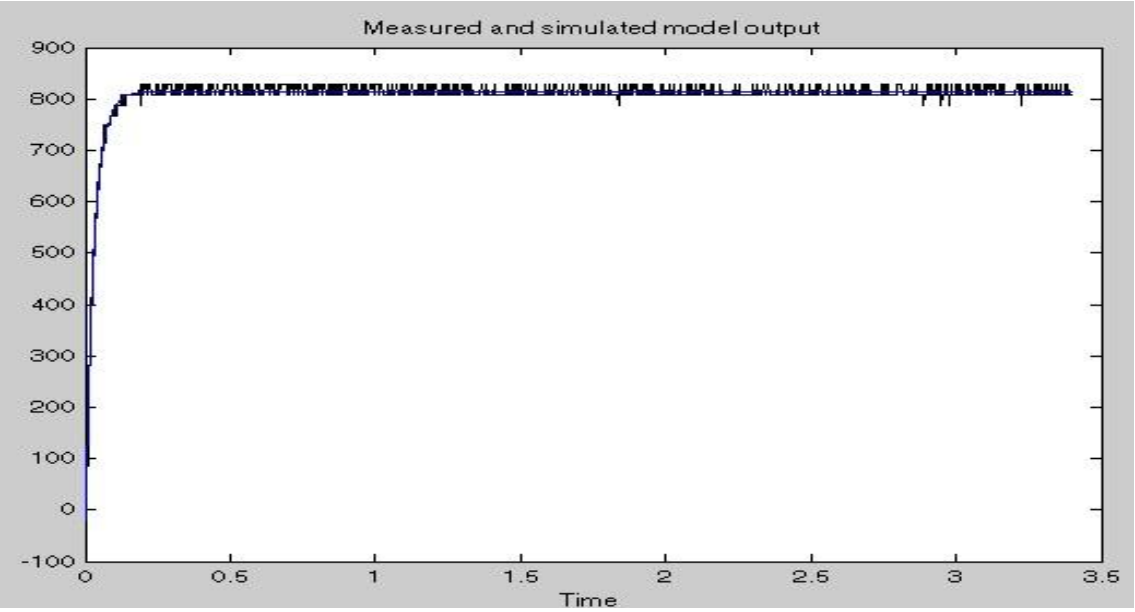

Figura 37 - Modelo da função de transferência obtido no domínio de *Laplace*.

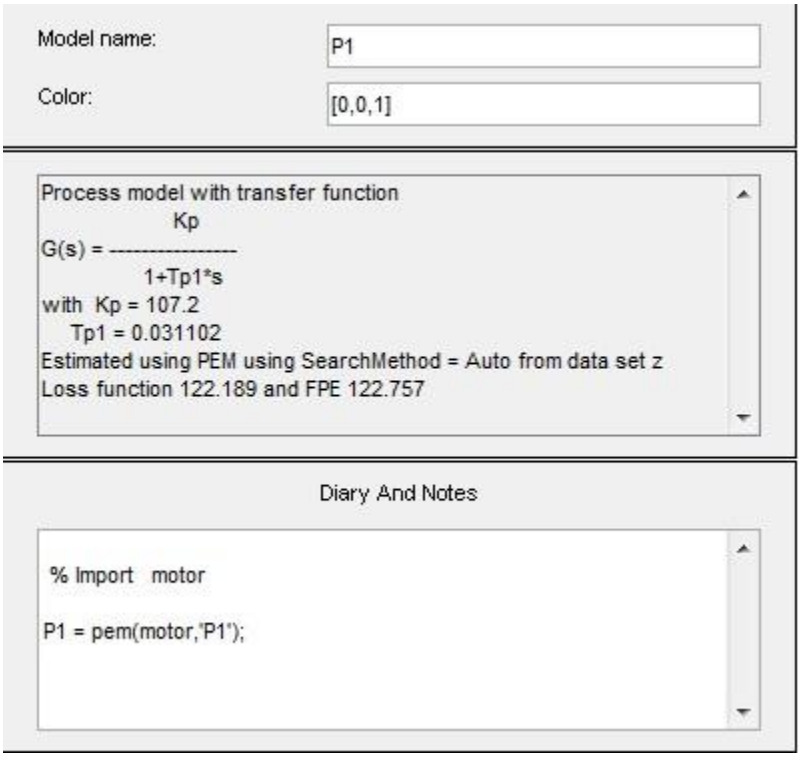

Figura 38 - Dados da função de transferência obtidos no domínio de *Laplace*.

A função de transferência do robô VSS obtida foi à seguinte:

$$G(s) = \frac{107.2}{0.031102s+1}$$

E para obter sua forma discretizada a partir do modelo contínuo, utilizou-se a função “c2d” (converte sistemas contínuos no tempo para discreto usando um *Holder* de ordem zero) do *Matlab* para uma taxa de amostragem de 3.96ms e chegou-se a seguinte resposta como ser visto na Figura 39:

$$G_{motordisc}(z) = \frac{12.82}{z - 0.8804}$$

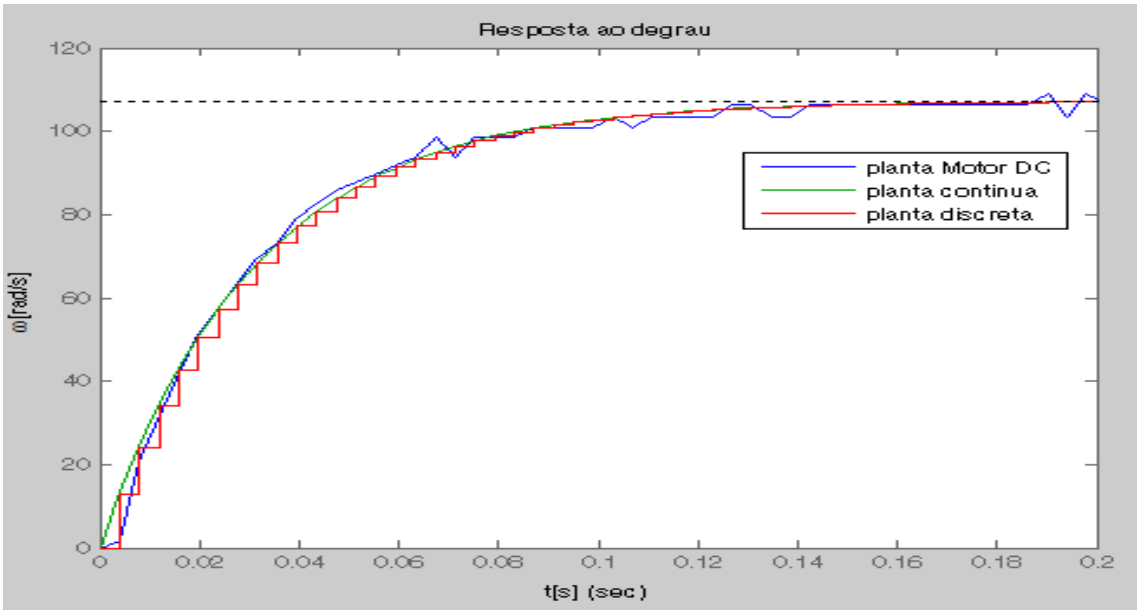

Figura 39 - Representações da resposta ao degrau do motor CC.

Obtida a função de transferência que descreve a resposta do mini-motor CC, passa-se ao projeto com controladores.

4.2 Investigação de controlador P:

Na primeira simulação via *simulink*, cujo diagrama de blocos pode ser visto na Figura 40, utilizou-se um ganho proporcional Kp=1 e obteve-se a seguinte resposta em velocidade da saída para um degrau de tensão de 7,6V mostrado na Figura 41. Sendo que o limitador representa a saída do microcontrolador que pode variar entre 0 e 5V.

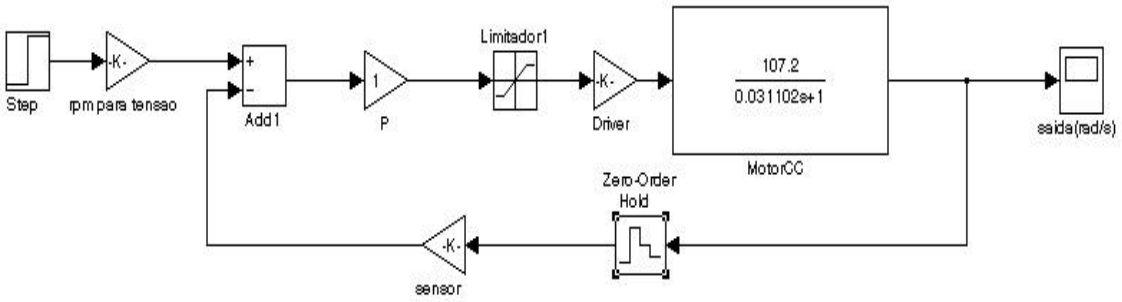

Figura 40 - Diagrama de blocos do controlador proporcional.

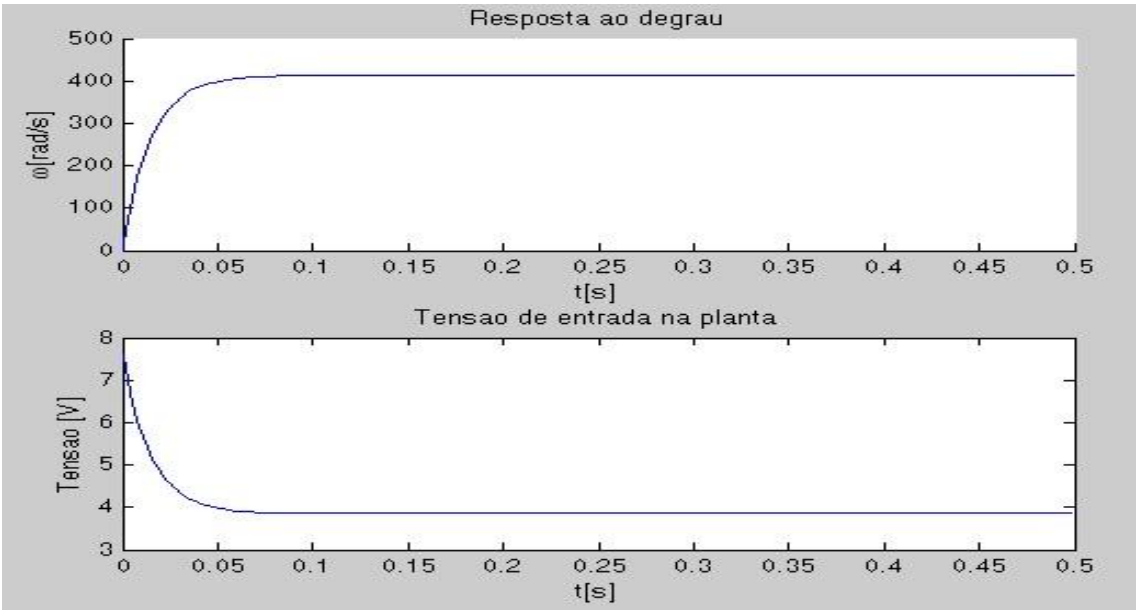

Figura 41 - Resposta ao degrau do motor CC e tensão na entrada da planta.

Observou que a máxima velocidade que o motor pode chegar é de 420 rad/s, sem saturar o sinal de entrada da planta.

Para o controlador proporcional com ganho um (Kp=1), obteve-se a seguinte resposta real comparada com a simulada vista na Figura 42:

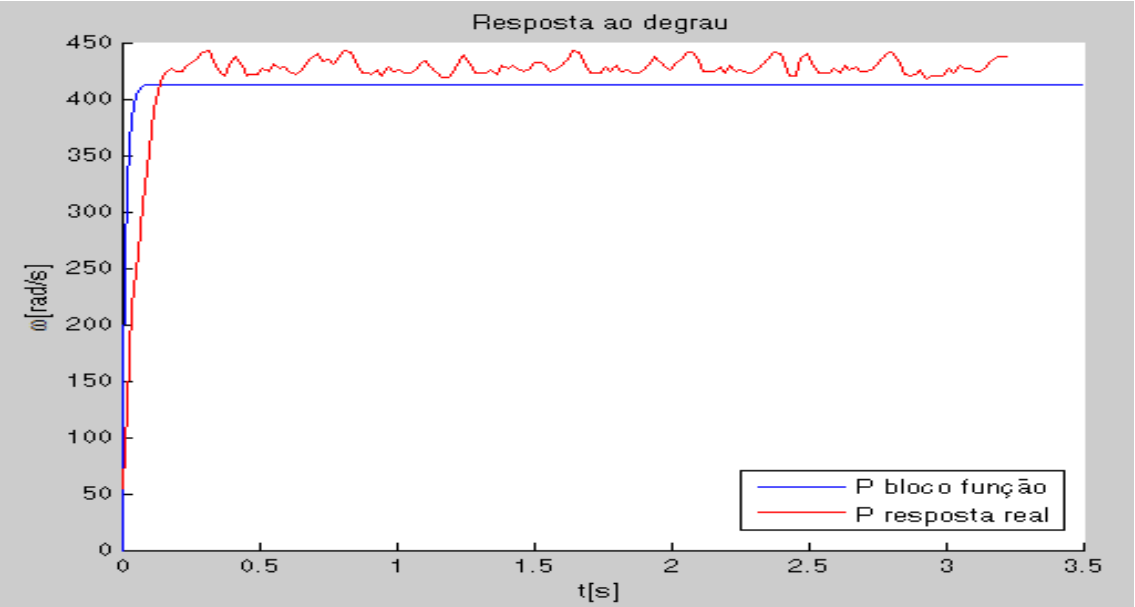

Figura 42 - Resposta ao degrau do motor CC com dados reais e simulados.

Em seguida, mudou-se o ganho proporcional para Kp=10 e obteve-se o seguinte desempenho como se pode visualizar Figura 43:

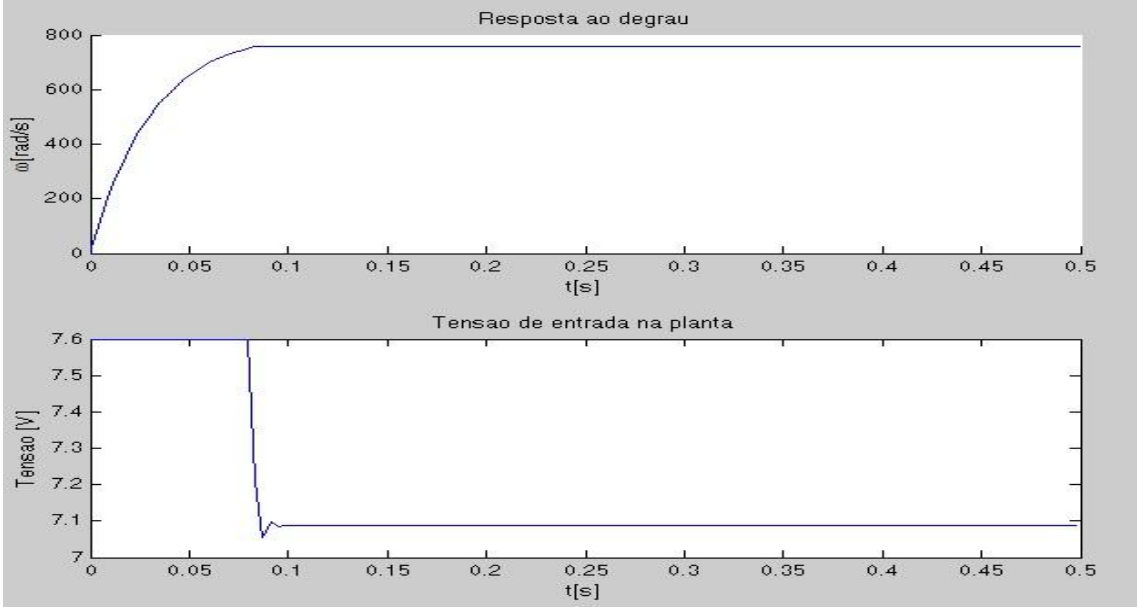

Figura 43 - Resposta ao degrau do motor CC e tensão na entrada da planta.

Nessa situação, a máxima velocidade que o motor atinge como resposta ao degrau foi de 760 rad/s, porém observou-se a saturação do sinal de entrada da planta durante todo o regime transitório. Essa situação se mostra indesejada para um projeto de

controlador, pois para uma faixa de tensão na entrada na planta a ação de controle irá ser saturada.

Sabendo que a máxima velocidade de rotação do motor é de 8000rpm (≈838rad/s), o controlador proporcional, para Kp=1, atinge apenas metade dessa velocidade máxima e quando esse ganho é aumentado para tentar se atingir a máxima velocidade o sinal de controle se mostra saturado para um faixa de tensão de entrada.

4.3 Investigação do controlador PI:

O projeto do controlador PI, cujo diagrama de blocos pode ser representado na Figura 44, foi realizado no *toolbox Rltool* do *Matlab* pelo qual se pode variar o ganho e zero do sistema e ao menos tempo tendo a ação de controle e resposta da planta monitorada na forma vista na Figura 45.

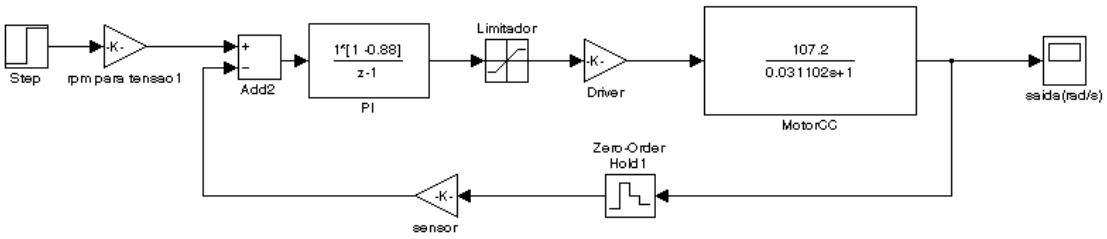

Figura 44 - Diagrama de blocos do controlador PI

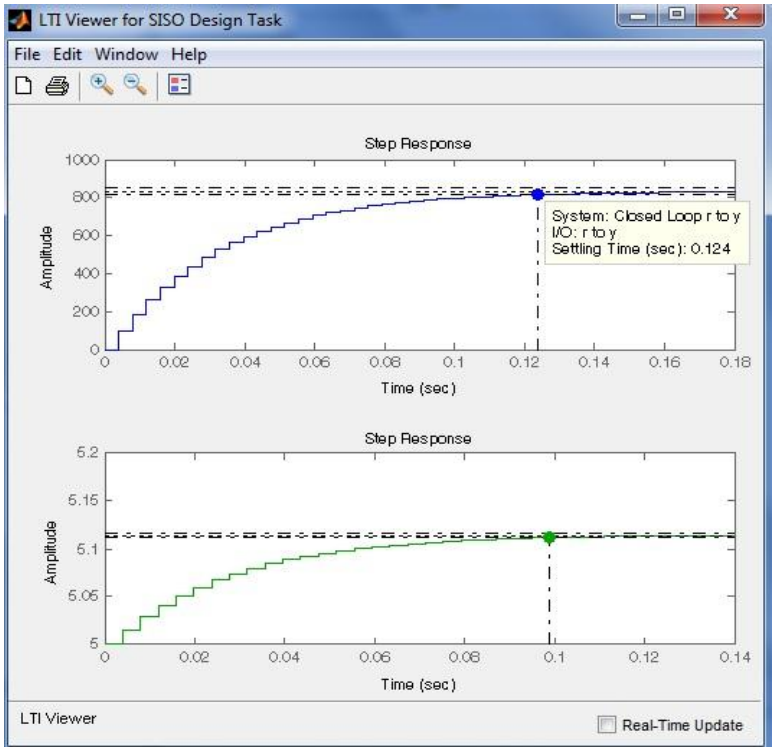

Figura 45 - Resposta ao degrau do motor CC e tensão de entrada na planta.

Através do editor *Rlocus* como o visualizado na Figura 46, podem-se delimitar algumas regiões para o lugar das raízes que atendam as especificações do projeto.

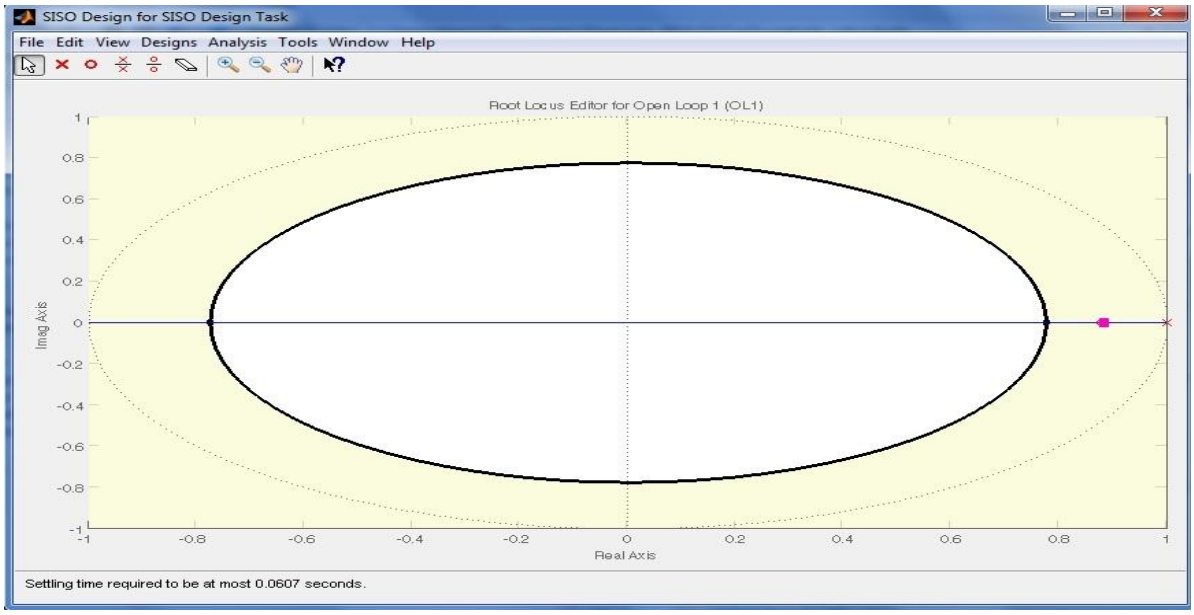

Figura 46 - Editor do lugar das raízes, cuja região fora da elipse delimitando uma região de tempo de acomodação menor que 60,7ms.

A função de transferência do controlador PI na sua forma discretizada obtida foi à seguinte:

$$G(s) = \frac{1 \cdot (z - 0.88)}{z - 1}$$

Na forma recursiva representada por:

$$u(k) = u(k\text{-}1) + 1 \cdot e(k) - 0{,}88 \cdot e(k\text{-}1)$$

E para obter sua forma contínua a partir do modelo discreto se utilizou a função "d2c" (transforma modelos lineares discretos para o tempo contínuo) do *Matlab* para uma taxa de amostragem de 3,96ms:

$$G_{motordisc}(z) = \frac{s + 30}{s}$$

Na simulação feita via *simulink*, obteve-se a seguinte resposta que pode ser visualizada na Figura 47:

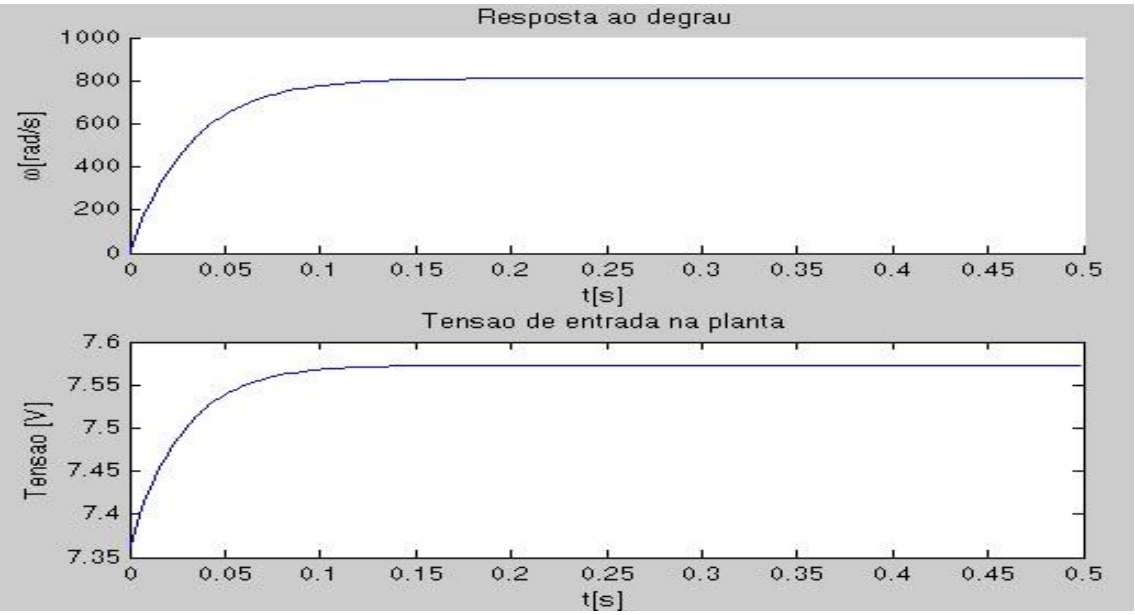

Figura 47 - Resposta ao degrau do motor CC e ação de controle na planta via *simulink*.

Na simulação via *simulink*, esquematizado na Figura 48, mas agora chamando uma função do *Matlab* (estruturada a partir de equações diferenças) que irá realizar a mesma ação de controle do bloco do controlador anterior. Já que dessa forma aproximados ainda mais um controlador simulado com aquele que seja implementado no microcontrolador em C. Cuja parte do código fonte *m-file* que representa o controlador PI na forma de equação recursiva se encontra abaixo.

```
vel=a(1);          % velocidade atual do motor
sp=a(2);           % velocidade de referencia: set point
t=a(3);            % tempo
ek=sp-vel;         % erro entre a velocidade atual do motor e a de referencia
Kp=1;              % constante Kp do controlador PI
Ti=0.88;           % constante Ti do controlador PI
Tt=1/sqrt(Ti);     % constante Tt caso use Wind up
%calcula o valor de velocidade a partir de equação diferença
 vk  = vk_1 + Kp*ek - (Kp*Ti)*ek_1 + (1/Tt)*es_1;
%guarda a valores das variáveis para a próxima iteração
 vk_1=vk;
 ek_1=ek;
 uk  =vk;
```

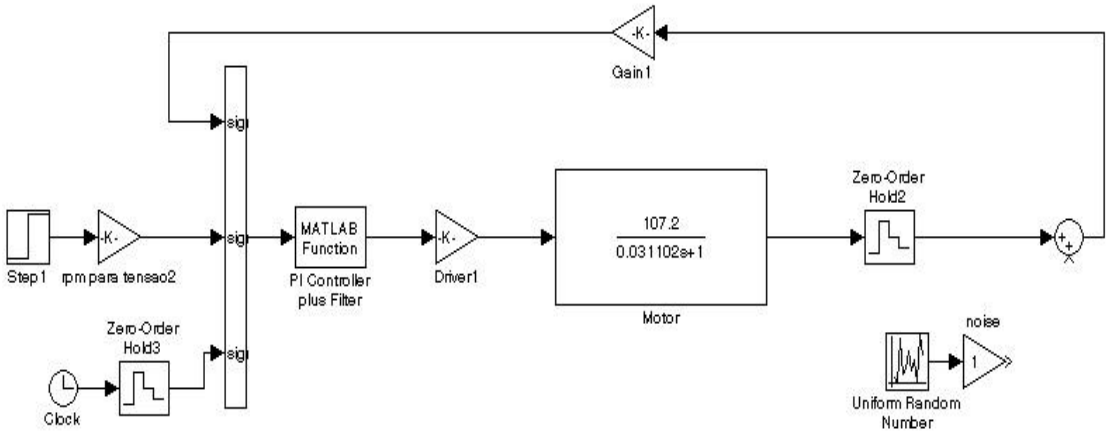

Figura 48 - Diagrama de blocos do controlador PI utilizando o bloco *Matlab function*.

A Figura 49 apresenta a resposta em velocidade do robô VSS com a respectiva ação de controle representada na mesma figura.

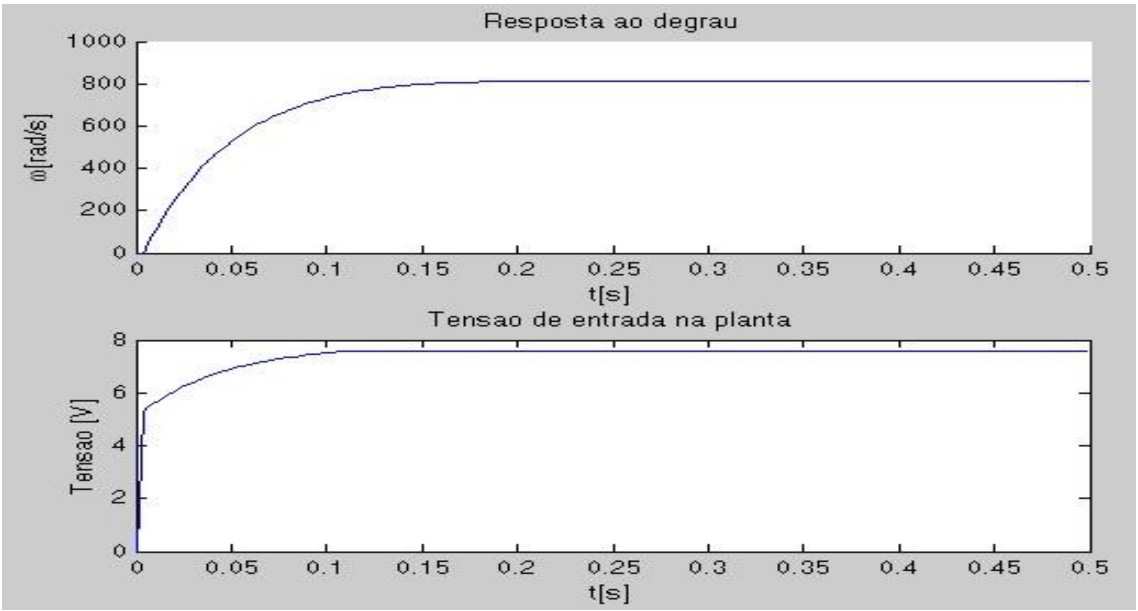

Figura 49 - Resposta ao degrau do motor CC e ação de controle na planta.

A comparação entre os métodos de representar o controlador PI com um bloco discreto e utilizando um bloco que carrega a forma recursiva do controlado pode ser vista na seguinte Figura 50:

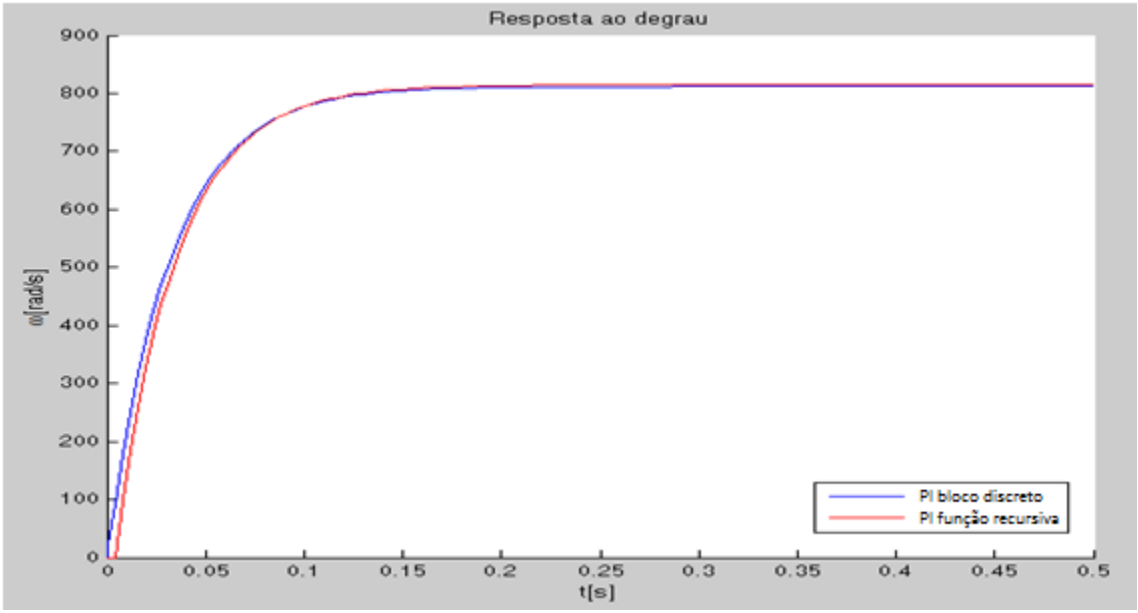

Figura 50 – Comparação de resposta ao degrau do motor CC.

A resposta real do motor com valores obtidos com o degrau de tensão aplicado ao robô, ou seja, quando o PWM é máximo e o *driver* fornece a tensão máxima aos motores comparada com a situação simulada mostrou-se na seguinte forma visualizada na Figura 51:

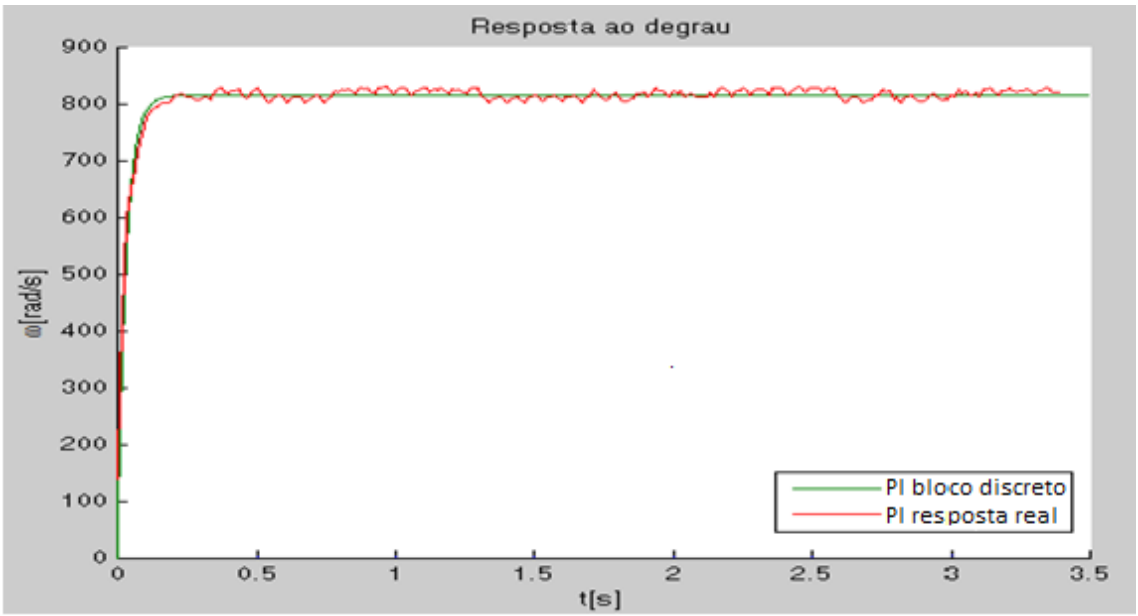

Figura 51 – Comparação de resposta ao degrau do motor CC.

Nessa investigação do controlador PI, observa-se que se pode atingir a velocidade máxima do mini-motor CC em regime com um tempo de acomodação próximo daquele obtido para o controlador proporcional com Kp=1 de aproximadamente 150ms, mas nesse caso é atingido a velocidade máxima próximo de 838 rad/s ou 8000 rpm.

4.4 Controlador PIncremental *fuzzy*:

Primeiramente, estruturou-se a lógica *fuzzy* utilizando o *toolbox* do *Matlab fuzzy.* Na fuzzyficação dos sinais de entrada: erro e delta_erro têm-se funções de pertinências triangulares (Figura 52) e na saída funções *singleton* (Figura 53):

As funções de pertinência utilizadas foram geradas através do módulo *Fuzzy Matlab*, utilizando os comandos “trimf”, ou através do seguinte trecho de código *m-file*:

```
%O primeiro argumento é o erro ou a derivado do erro e o segundo se é o caso negativo,
%zero ou positivo.
function f=pertinencia(e_de,nzp)
global N Z P
y=0;

switch nzp  %verifica qual função de pertinência foi escolhida
   case N,   %para o caso de pertinencia negativa
      if ( e_de <= 0 & e_de>=-5 )      %para uma entrada limitada entre -5 e 0
         y = - (1/5)*e_de;             %a saída obedece essa função
      else
         y=0;                          %fora dessa área vale zero
      end
      …
```

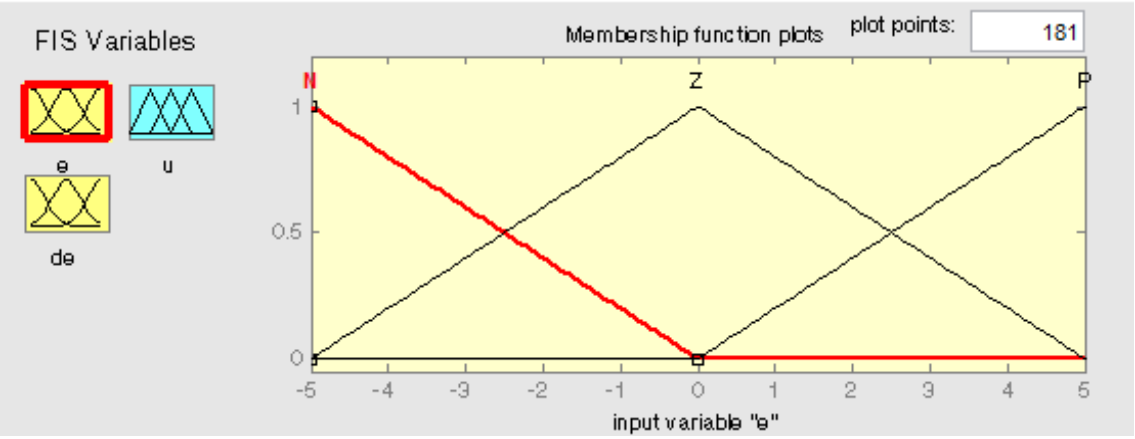

Figura 52 - Entrada e suas funções de pertinência.

Legenda das variáveis lingüísticas:

N(Negativo): função triangular centrada em -5, com x_{min} igual a -5 e com x_{max} igual a 0.

Z(Zero): função triangular centrada em 0, com x_{min} igual a -5 e x_{max} igual a 5.

P(Positivo): função triangular centrada em 5, com x_{min} igual a 0 e x_{max} igual a 0.

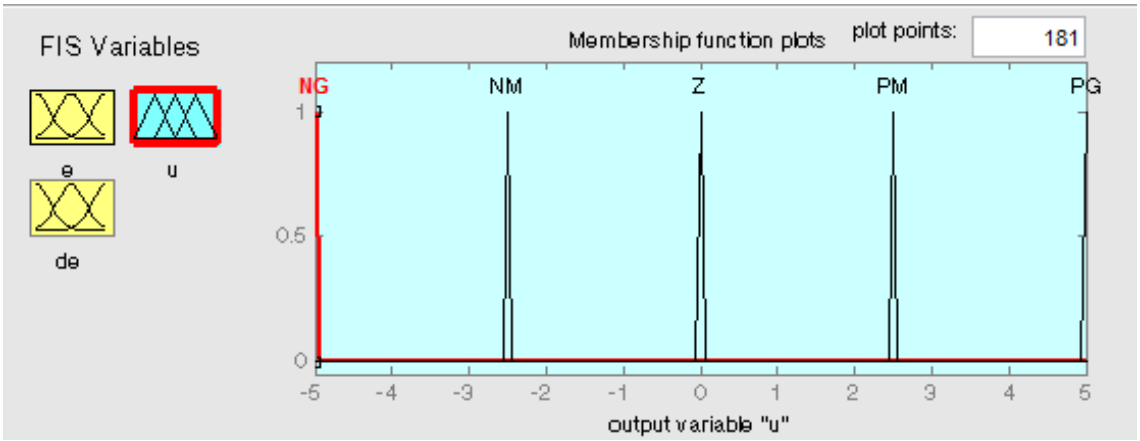

Figura 53 - Saída e suas funções de pertinência.

Legenda das variáveis lingüísticas:

NG(Negativo Grande): função singleton centrada em -5.
NM (Negativo Médio): função singleton centrada em -2,5.
Z(Zero): função singleton centrada em 0.
PM(Positivo Médio): função singleton centrada em 2,5.
PG(Positivo Grande): função singleton centrada em 5.

As regras *fuzzy* relacionam variáveis *fuzzy*, cada uma delas associada a um dos seus predicados lingüístico sendo o conjunto de regras Fuzzy dada a seguir:

		delta_erro		
		N	Z	P
erro	N	NG	NM	Z
	Z	NM	Z	PM
	P	Z	PM	PG

Dispõe-se a seguinte superfície de regras como se pode observar na Figura 54 e de um sistema de previsão das regras exemplificado na Figura 55.

E por fim na interface de defuzzificação o conjunto fuzzy lingüístico é convetido em sinal de controle, no nosso caso utilizou-se o centro de massa ou centróide, pela opção de comando "*centroid*", ou através do seguinte trecho de código *m-file*:

```
for k=1:length(t); %% saída defuzzificada
   for j=1:9
      num(k,j)=u(k,j)*(t(k));
      den1(k,j)=u(k,j);
   end
end
   saida=sum(num)/sum(den1);
```

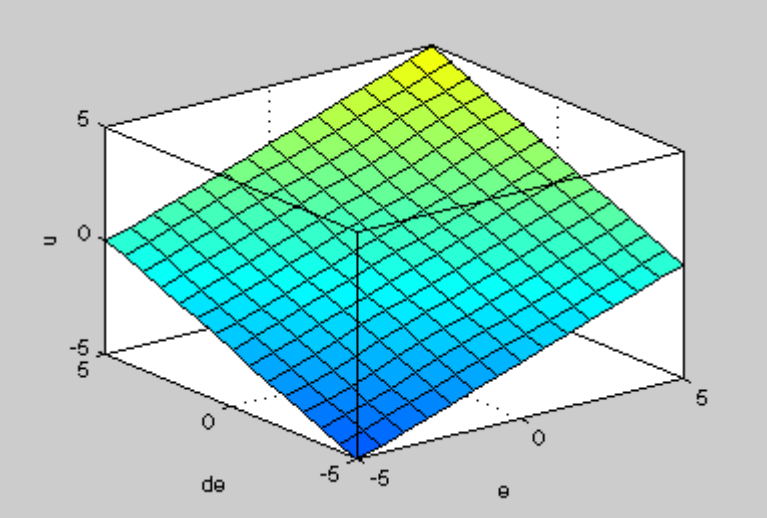

Figura 54 - Superfície de regras das entradas pela saída.

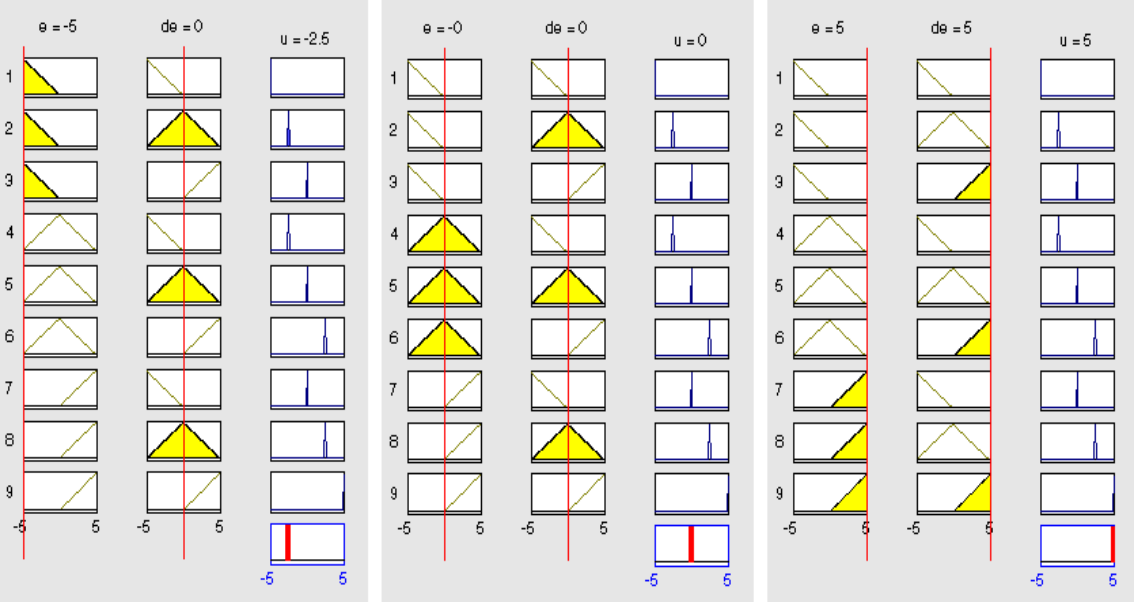

Figura 55 - Sistema de previsão.

Toda essa lógica *fuzzy* foi também implementada em *m-files* e obtiveram-se resultados semelhantes como se podem comparar os resultados obtidos no sistema de previsão da Figura 55 com os resultados obtidos na Figura 56, para os mesmos valores de entrada. Facilitando para que futuramente se realize a programação da mesma em microcontroladores. Sendo que o arquivo *m-file* que representa o sistema de previsão se encontra no apêndice B intitulado mfuzzy.m.

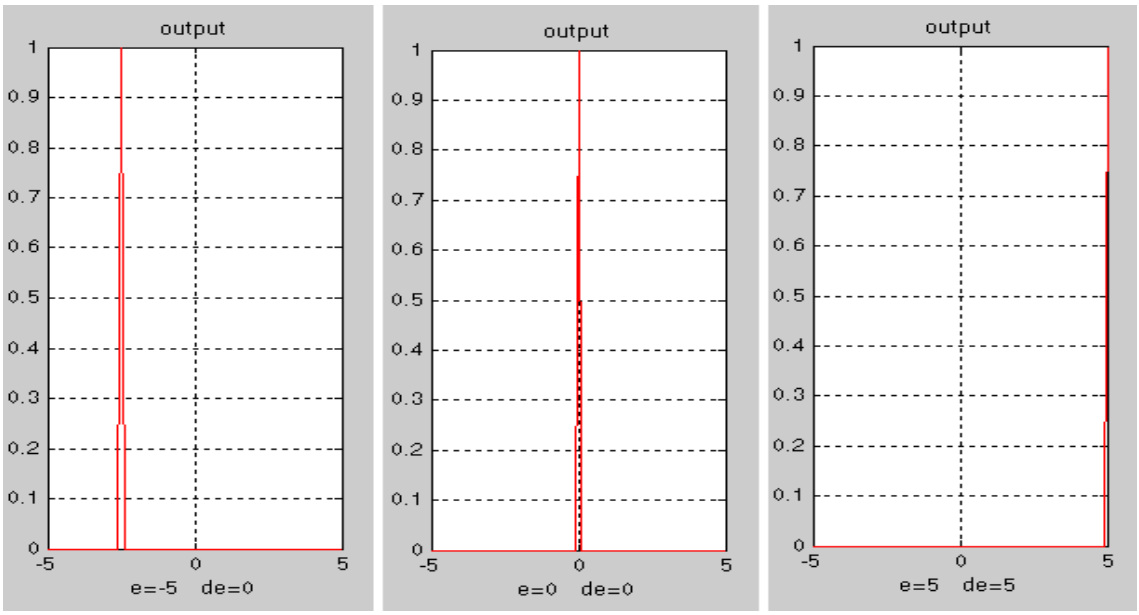

Figura 56 - Saída do sistema de previsão pelo *m-file*.

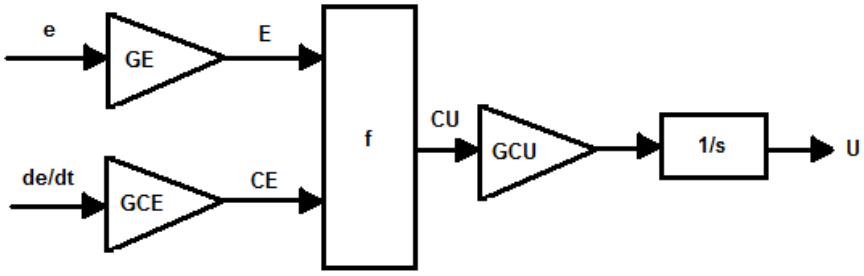

Figura 57 - Representação de um controlador PIncremental *fuzzy*.

Um controlador PIncremental *fuzzy* é análogo ao PI projetado anteriormente seguinte o diagrama da Figura 57, obedecendo as seguintes correlações:

$$u(k) = GCE \cdot GCU \cdot \left[\frac{GE}{GCE} \cdot \sum_{j=1}^{k} e(j) \cdot T_s + e(k)\right]$$

Sabendo que:

$$u(k) = K_p \cdot \left[\frac{1}{T_i} \cdot \sum_{j=1}^{k} e(j) \cdot T_s + e(k)\right]$$

Assim, tem-se a seguinte relação entre os controladores:

$$GCU \cdot GCE = K_p$$

$$GE/GCE = 1/T_i$$

Em que GE é o ganho do erro, GCE o ganho da derivada do erro e GCU o ganho da saída da lógica *fuzzy*. A Figura 58 representa o controlador PIncremental *fuzzy* com a parte mais acima representando um sistema que utiliza *m-file* e na parte mais abaixo representando o sistema que utiliza o *fuzzy-File*.

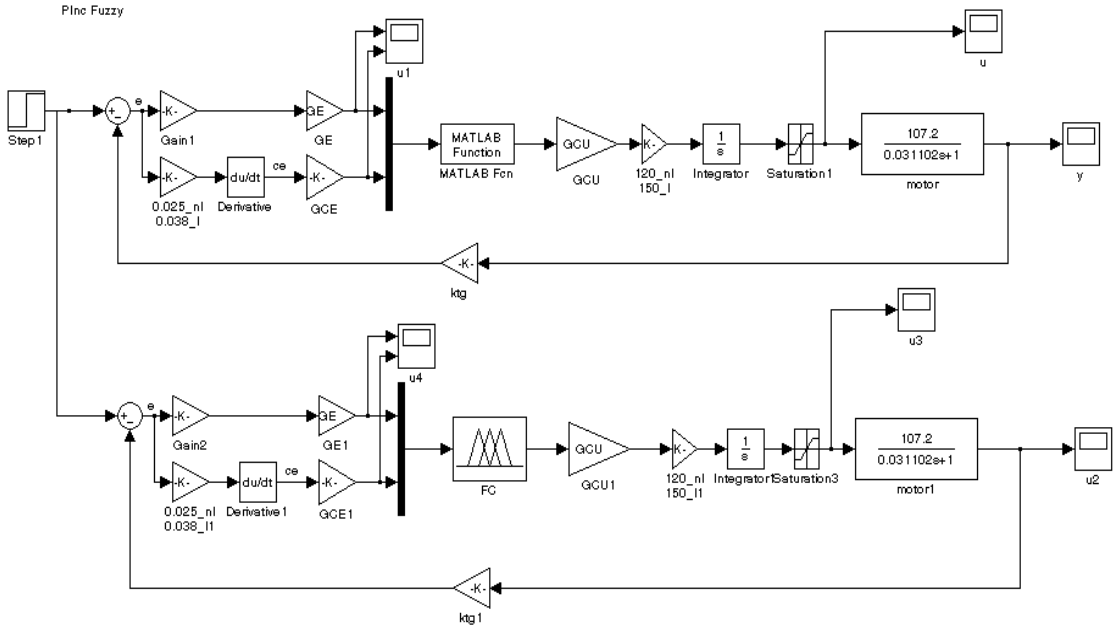

Figura 58 - Diagrama de blocos do controlador PIncremental *fuzzy*.

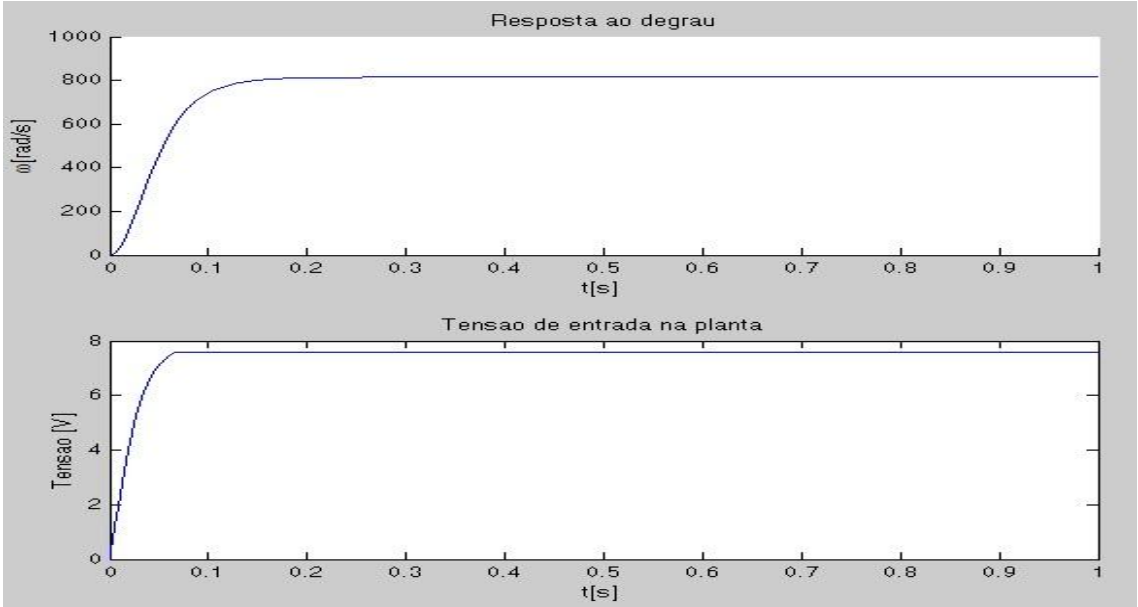

Figura 59 - Resposta ao degrau do motor CC e tensão de entrada utilizando *fis-file*.

No segundo método, utiliza-se um bloco que carrega uma função mfuzzy.m (*Matlab Function*) que tem resultado similar ao dados vindo do *toolbox fuzzy*. Cujo resultado pode ser visualizado na Figura 60 e feita a comparação na Figura 61 desse resultado com o bloco que carrega o arquivo *fis-file* (do *toolbox fuzzy*).

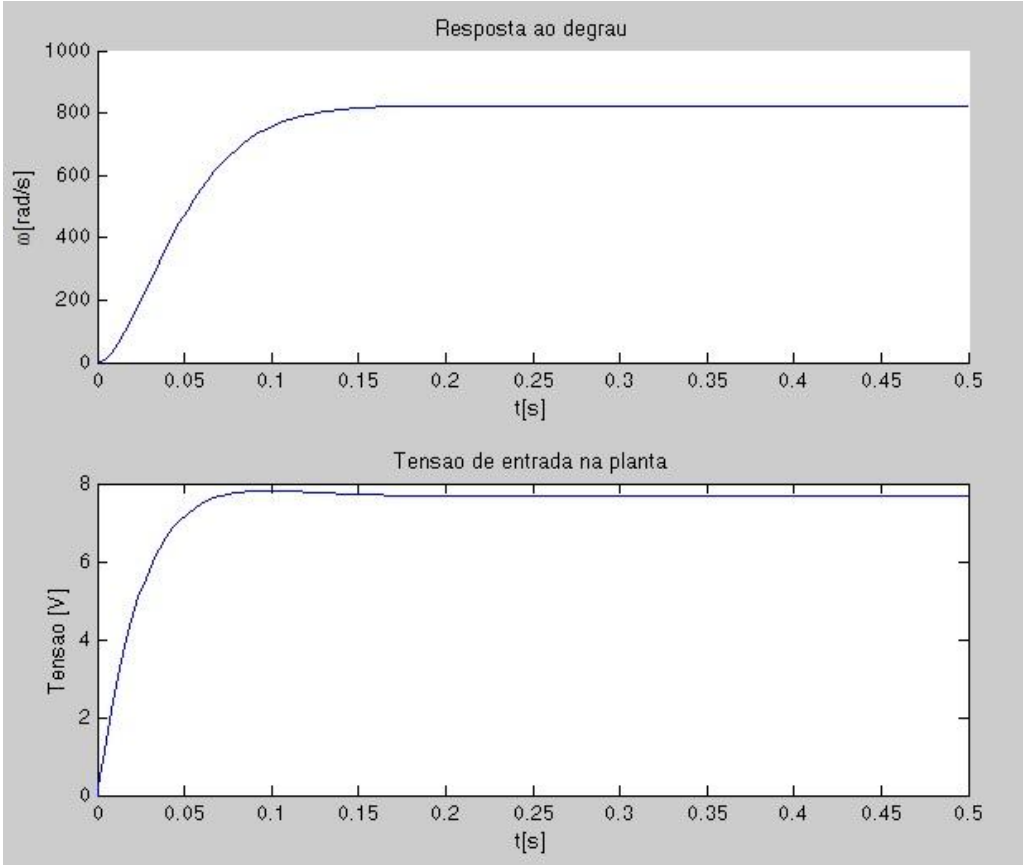

Figura 60 - Resposta ao degrau do motor CC e tensão de entrada quando utilizando *m-file*.

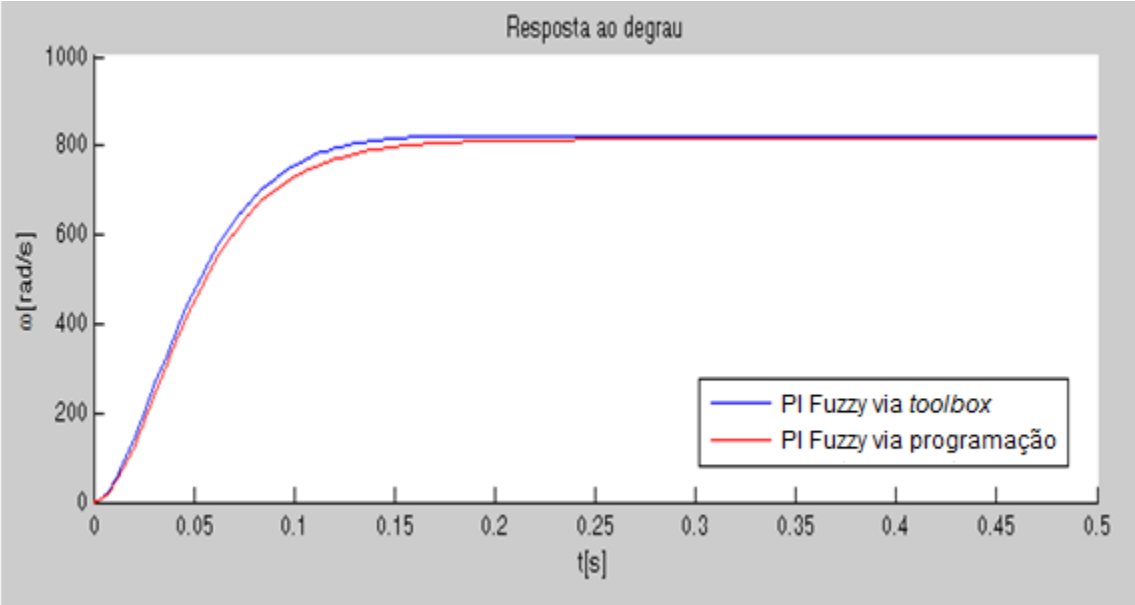

Figura 61 - Compara a resposta ao degrau do motor quando se utiliza *fis-file* e *m-file*.

Observa-se que a resposta quando se utiliza *m-file* está em pouca atrasada com relação à resposta utilizando *fis-file*, mas as duas atingem o regime com o mesmo valor de velocidade e com um tempo de acomodação próximo de 150ms parecido com o controlador PI.

Apesar desse trabalho ter simulado e obtido um resultado bom para o controlador PIncremental fuzzy não houve tempo de converter o código *m-file* para um código *c-file* e implementar o mesmo no microcontrolador.

5 Conclusão

Esse trabalho englobou duas grandes áreas do curso de engenharia elétrica (eletrônica) que são: sistemas digitais na programação do microcontrolador e controle no projeto do controlador utilizando o *software Matlab*, permitindo ao aluno aplicar os conhecimentos teóricos em algo prático que tem o retorno visual do comportamento do robô muito rápido, pois é possível programá-lo quantas vezes se for necessário e fazer os ajustes para que o robô tenha um comportamento adequado.

Desenvolveu-se uma programação do controle digital em linguagem de programação em C a partir de um código do *Matlab* (*m-files*) no qual foram simulados os controlados de velocidade proporcional(P), proporticonal-integral(PI) e PIncremental *fuzzy*. Sua implementação foi considerada um sucesso permitindo com que o robô desenvolvesse as velocidades de referência de cada uma das rodas que o compõe da forma desejada por um usuário ou programa de estratégia, ficando a cargo de um controle de posição a navegação correta no ambiente em que se encontra o robô. Assim foi possível comparar as respostas em termos de velocidade do robô simuladas via *Simulink* com as respostas reais.

Outros fatores também exercem influência significante sobre o comportamento do robô real que podem ser observados com os robôs ligados (jogando futebol na maioria das vezes) por um longo período são:

- Ocorrência de distúrbios incidentes sobre os motores;
- Distribuição desuniforme de carga sobre a plataforma, impondo torques resistentes distintos a cada motor;
- Diferença entre os raios das rodas motrizes (pode ocorrer como conseqüência do fator citado acima, a depender do tipo de roda);
- Desalinhamento das rodas;
- Existência de uma área e não um ponto de contato entre a roda e a superfície de navegação, o que implica em presença de atrito de rolamento;
- Irregularidades da superfície de navegação;
- Ocorrência de derrapagens na realização de manobras.

A não ponderação desses fatores pode tornar qualquer estratégia de controle, que a princípio mostra-se funcional, ineficiente. Porém isto não foi objeto de estudo nesse trabalho.

Projetos futuros que complementariam esse trabalho poderiam ser realizados no estudo e implementação de códigos em C para ser compilados diretamente no *Matlab* com o uso de um *plug-in* desse *software*, o que facilitaria exportar projetos de controladores do *Matlab Simulink* para os microcontroladores com uma melhor dinâmica.

6 Bibliografias

D. F. Wolf, E. V. Simões, F. S. Osório, O. T. Junior. (2009). “Robótica Móvel Inteligente: Da Simulação às Aplicações no Mundo Real”. Bento Gonçalves-RS, CSBC 2009.

FAULHABER.DC-Micromotors Series 2224 006 SR. Disponível em <http://www.faulhaber.com/uploadpk/EN_2224_SR_DFF.pdf>. Acesso em: 1 Out. 2011

Franklin, G.; Powwmm,J.; Naeini,A., (2002).Feedback Control of Dynamic System, Prentice Hall.

H. Duane, L. Bruce. (2003). Mastering MATLAB 6 – A Comprehensize Tutorial and Reference, Prentice Hall.

MATH WORKS. Disponível em: <www.mathworks.com.br>. Acesso em: 20 Set. 2011.

MATLAB 6 Curso Completo, São Paulo, 2003.

MICROCHIP. PIC18F2331/2431/4331/4431 Data Sheet. Disponível em <http://ww1.microchip.com/downloads/en/DeviceDoc/39616b.pdf>. Acesso em: 1 Out. 2011

O. A. Vilma, A. L. Manoel, V. B. Jerson. (2005). Sistemas de Controle: Aulas de Laboratório, EESC-USP.

REZNIK, L. Fuzzy Controllers. Newnes, Reino Unido, 1997. Disponível em: <http://www.ime.usp.br/~tonelli/verao- Fuzzy /links.php>. Acesso em: 01 nov. 2011.

ROGERCOM. (2011). “Manual da placa CON-USBBEE Rogercom”. Acesso em: Outubro de 2011, Brasil

SIMÕES, Marcelo G.; SHAW, Lan. (1999). Controle e modelagem FUZZY. São Paulo: Edgard & Blucher.

STMicroelectronics. L298 - Dual Full-Bridge Driver. Disponível em <http://www.cse.dmu.ac.uk/~mgongora/Resources/L298N.pdf>. Acesso em: 1 Out. 2011

7 Apêndices

7.1 Apêndice A - controladorpi.m

```
function u=controladorpi(a)
global sp ek ek_1 vk vk_1 uk vel t es es_1

vel=a(1);    % velocidade do motor
sp=a(2);     % velocidade de referencia: set point
t=a(3);      % tempo
ek=sp-vel;   % erro entre a velocidade do motor e a de referencia
Kp=1;        % constante Kp do controlador PI
Ti=0.88;     % constante Ti do controlador PI
Tt=1/sqrt(Ti);  %constante Tt caso use Wind up
limitador=5;    %maxima tensao de saida do microcontrolador: 5V

if t==0     % vefifica condicoes iniciais
    vk_1=0;
    vk  =0;
    uk=0;
    ek  =0;
    ek_1=0;
    es = 0;
    es_1 = 0;
else        %calcula o valor de velocidade a aparitr de equacao diferenca
    vk  = vk_1 + Kp*ek - (Kp*Ti)*ek_1 + (1/Tt)*es_1;
    vk_1=vk;
    ek_1=ek;
    uk  =vk;
end

%saturador superior
if vk>limitador
   uk=limitador;
end % uk must be <= upper limit

%saturador inferior
if vk<-limitador
   uk=-limitador;
end % uk must be >= lower limit

es=uk - vk;
es_1=es;
u=uk;
```

7.2 Apêndice B - mfuzzy.m

```
function f = mfuzzy(data)

%% parâmetros
erro=data(1);      % entrada do erro
delta_erro=data(2); % entrada do valor da variação do erro

global N Z P

%% fuzzyficacao
t=-5:0.1:5;   %entrada na faixa de -5V a 5V.

%função de pertinência no apêndice C
en=pertinencia(erro,N);
ez=pertinencia(erro,Z);
ep=pertinencia(erro,P);
den=pertinencia(delta_erro,N);
dez=pertinencia(delta_erro,Z);
dep=pertinencia(delta_erro,P);

%regras
regras=[min(en,den)
        min(en,dez)
        min(en,dep)
        min(ez,den)
        min(ez,dez)
        min(ez,dep)
        min(ep,den)
        min(ep,dez)
        min(ep,dep)];
k=0;
for i=1:length(t)
    k=k+1;
    u(k,:)=saidau(t(i),regras);
end;
% umax=max(u,[],2);
% figure,plot(t,umax,'r')
% title('output')
% axis([-5 5 0 1])
% grid
%% saída defuzzificada
for k=1:length(t);
    for j=1:9
        num(k,j)=u(k,j)*(t(k));
        den1(k,j)=u(k,j);
    end
end
    saida=sum(num)/sum(den1);
f=saida;
end
```

7.3 Apêndice C - pertinência.m

```
function f=pertinencia(e_de,nzp)
global N Z P
y=0;

switch nzp
    case N,   %pertinencia negativa
        if ( e_de <= 0 & e_de>=-5 )
            y = - (1/5)*e_de;
        else
            y=0;
        end
    case Z,   %pertinencia zero
        if ( e_de <= 0 & e_de>=-5)
            y = 1 + (1/5)*e_de;
        elseif (e_de>0 & e_de<=5)
            y = 1 - (1/5)*e_de;
        else
            y=0;
        end
    case P,   %pertinencia positiva
        if (e_de>=0 & e_de<=5)
            y = (1/5)*e_de;
        else
            y=0;
        end
    otherwise
        y=0;
end

f=y;
```