Universidade de São Paulo - Escola Politécnica
Departamento de Engenharia Mecânica

9,5 | nove e cinco

Desenvolvimento de linguagem de Programação para Robôs Móveis

Autor :
Valdir Grassi Junior

Trabalho de Gradução
Engenharia Mecânica – Automação e Sistemas

Orientador :
Jun Okamoto Jr.

EPUSP - 1998

Índice

1. Introdução

No ano de 1994 foi iniciado no Departamento de Engenharia Mecânica da Universidade de São Paulo um projeto denominado Base Móvel. Este projeto foi realizado por alunos de graduação sob a orientação do Prof. Dr. Jun Okamoto Jr., e constituiu-se na elaboração e construção de um veículo de pequeno porte do tipo AGV destinado a transportar sensores em um laboratório para a realização de pesquisas na área de visão.

Este projeto teve continuidade nos anos seguintes, onde se deu a elaboração de um hardware eletrônico para controle deste robô móvel, e também um software constituídos de comandos básicos para interface do usuário com o hardware eletrônico do robô.

A Base Móvel foi projetada para se deslocar no plano bidimensional, sendo que possui um formato cilíndrico, e é dotada de três rodas motoras que esterçam ao mesmo tempo. Desta forma possui a capacidade de fazer curvas em canto vivo.

O acionamento das rodas é feito por dois motores, um motor de passo que realiza o esterçamento das rodas, e um motor DC responsável pela translação da base móvel no plano. Entre cada motor e as rodas existe um sistema de redução composto de polias de correia dentada e engrenagens.

O hardware eletrônico da Base Móvel possui os seguintes componentes principais : microcontrolador 8032, driver PWM, driver de motor de passo, comunicação serial com PC, interface para joystick e sistema para contagem dos pulsos vindos do encoder do motor DC. O programa de controle da Base Móvel é armazenado em uma EPROM, sendo que o sistema possui uma memória RAM para armazenar dados temporários. Todo este hardware eletrônico foi projetado de maneira a ser compacto para que seja embarcado na Base Móvel.

Além do hardware eletrônico, se encontrava desenvolvido um software dotado de comandos e funções básicas que funcionam em um sistema de tempo real. Estes comandos e funções são simples e permitem, por exemplo, escrever uma palavra no driver de PWM ou de motor de passo, permitem uma contagem de 32 bits dos pulsos de encoder, visualizar continuamente os valores de encoder e posição da alavanca do joystick, já havia disponível também uma estrutura de programa em tempo real, no entanto, estes comandos não eram suficientes para controle adequado do movimento do robô.

2. Objetivo

Para que se tenha o controle adequado do movimento de um robô móvel, especificamente a Base Móvel, é necessário possuir um conjunto de comandos que permitam a um usuário fazer com que o robô se movimente da maneira desejada. Também é interessante que se possa construir programas que descrevam a determinada trajetória que o robô deve seguir. Para tanto é necessário ter uma linguagem de programação para o robô. Como esta linguagem e os comandos pertencentes a ela ainda não tinham sido desenvolvidos e implementados para a Base Móvel, ela não era capaz ainda de se movimentar por trajetórias pré-definidas.

O objetivo deste projeto é justamente desenvolver uma linguagem de programação para um robô móvel, sendo que esta linguagem seja composta de comandos de alto nível que devem ser executados em uma estrutura de software de tempo real.

Os comandos pertencentes à linguagem de programação serão implementados e testados na Base Móvel. Os comandos de movimento devem ser de fácil uso pelo usuário, sendo que representem de maneira adequada a trajetória que o robô deve fazer. Estes comandos devem usar um sistema de controle de posição em malha fechada com realimentação dada pelo encoder. Também é desejável que além da posiibilidade de se descrever trajetórias através de programas, se use o joystick para movimentar o robô, e também que seja possível a captura de pontos para formação de uma tabela que possa ser reproduzida posteriormente, fazendo com que o robô passe pelos pontos capturados.

3. Robôs Móveis

Existe uma variedade muito grande de robôs móveis, e eles se diferenciam entre si pela construção mecânica e pelo hardware eletrônico. A parte mecânica do robô é fundamental para definir onde ele poderá se locomover e o que poderá ser capaz de fazer ou não. Por exemplo, existem robôs móveis que se deslocam através de "patas", tentando reproduzir o movimento que insetos como aranha ou formiga fazem. Estes robôs são construídos para que possam se deslocar em solos irregulares, e são capazes de se transpor obstáculos como pequenas pedras. Existem robôs móveis submarinos, que precisam se locomover em três dimensões. Estes robôs são usados para atingir profundidades elevadas onde dificilmente o homem pode ir. Podemos encontrar ainda robôs móveis que possuem esteiras, como tanques de guerra, e robôs móveis que possuem quatro rodas, sendo que as duas rodas dianteiras possuem tração independente e a diferença de velocidade entre elas faz com que o robô faça curvas. Também existem robôs móveis com três rodas, como um triciclo, sendo que a da frente é motora e pode mudar a direção do robô.

Como se pode ver, podem ser feitas muitas variações nas características mecânicas dos robôs, e assim construir um que se adequa melhor a aplicação desejada.

Para cada tipo de robô, a implementação dos comandos e da linguagem proposta neste trabalho será diferente, pois ela depende das características construtivas, e também do hardware eletrônico do robô. Desta forma, será desenvolvido e implementados comandos para a Base Móvel. No entanto, o algoritmo de alguns comandos, a estrutura do software de controle, a estrutura da linguagem de programação, e as rotinas para reconhecimento desta linguagem podem ser aproveitadas e usadas em outros tipos de robôs móveis.

Os robôs móveis podem possuir uma série de sensores, e a partir dos sinais recebidos por eles, usando um sistema de inteligência artificial, o robô pode tomar decisões e então se locomover. Isto é usado quando se deseja explorar lugares desconhecidos que o homem não pode ir, como regiões próximas a vulcões, ou superfícies de outros planetas como Marte. Também é usado para que permita ao robô desviar de obstáculos que eventualmente podem haver na sua trajetória, evitando assim acidentes. Em relação à metodologia usada para mover tais robôs usando sistemas inteligentes, existem duas teorias principais : o método tradicional, e o método baseado em comportamento proposto por Rodney Brooks.

O método tradicional se baseia em coletar informações através dos sensores, construir uma representação do modelo do mundo em volta do robô através da fusão dos dados coletados pelos sensores, e então planejar as ações baseado no modelo adquirido. Por exemplo: o robô através de uma camera de vídeo capta a imagem do que está a sua frente, então é feito uma análise deste sinal, daí percebe-se que existe um obstáculo na sua frente e que este obstáculo têm uma largura 'x' , a partir deste dado é feito uma mudança na trajetória para que o robô desvie deste obstáculo.

O método proposto por Brooks não faz a fusão dos dados coletados pelos sensores, e nem um modelo do mundo exterior ao robô. Este método se baseia em programar vários comportamentos que o robô deve ter em resposta a algo, estes comportamentos possuem um nível hierárquico, e comportamentos que possuem um nível hierárquico alto, quando ativados, são executados em preferência àqueles de nível mais baixo. Isto não quer dizer que os comportamentos de nível hierárquico inferior não deixem de ser gerados, mas eles não são executados quando um comportamento de nível alto está ativado. Como ilustração, suponha que o comportamento de nível mais baixo habilite uma ação de andar para qualquer lugar e o de nível mais alto inicia um comportamento de seguir a luz, então normalmente, o robô irá andar sem rumo, movendo-se aleatoriamente. No entanto, quando alguém apontar uma lanterna para o robô, o comportamento de andar aleatoriamente será ignorado em detrimento do comportamento de seguir a luz que acaba de ser ativado e possui nível maior. O robô irá então seguir em direção á luz até que a lanterna seja apagada, quando o comportamento de andar sem rumo irá voltar a ter preferência. Uma observação que deve ser feita é que mesmo quando o comportamento de seguir a luz estava ativado, o outro comportamento também estava sendo gerado mas não estava sendo executado pois não tem a preferência.

O projeto que será desenvolvido este ano, não possui nenhuma característica de inteligência artificial, embora posteriormente a este trabalho podem ser feitas pesquisas nesta área, podendo utilizá-lo como ponto de partida. No entanto, neste trabalho, deseja-se que a trajetória do robô móvel que estiver sendo controlado seja bem definida, e esta seja especificada pelo usuário. Desta forma, este trabalho se enquadra com maior facilidade no método tradicional de controle utilizando inteligência artificial, já que será desenvolvido os comando para que se possa programar uma trajetória definida, conhecido o caminho.

4. Definição dos Comandos da Linguagem de Programação

A primeira tarefa a ser realizada é a de definir o projeto. Para isto deve-se definir como será a interface do usuário com o robô móvel, para que possa haver a programação do robô, e a execução de comandos associados ao movimento do robô e captura de dados. Esta interface pode ser determinada definindo-se os comandos que farão parte tanto da linguagem de programação como do software de controle, e também definindo-se a sintaxe destes comandos.

Como neste trabalho há um grande interesse em implementar a linguagem de programação na Base Móvel, optou-se por não se aprofundar no estudo de sintaxes de linguagens de programação, sendo que ao se definir a sintaxe, utilizou-se o conhecimento sobre linguagens de programação existentes como basic, C, e pascal. Também baseou-se nos comandos disponíveis em programas de desenho como AutoCAD para definição dos comandos. Como se quer descrever trajetórias do robô, e o que se faz em programas como o AutoCAD é descrever desenhos, ou trajetórias, através de vários comandos, optou-se por se basear no conjunto de funções e parâmetros exigidos por elas disponíveis em tal software.

Um outro fator importante que entrou na decisão de como seria a sintaxe foi a facilidade ou não de se reconhecer o comando e os parâmetros a partir da linha de comando digitada, já que a rotina de reconhecimento dos comandos segundo a sintaxe especificada deve ser implementada pois o reconhecimento deles não é imediato. O usuário entra através do teclado a linha de comando, esta fica armazenada em uma string de caracteres e depois esta string deve ser interpretada e retirado os parâmetros e o comando.

Baseado nos fatores descritos acima, definiu-se os comandos que farão parte da linguagem de programação, e do software de controle do robô móvel. Juntamente com a definição destes comandos, já se determinou como seria a interface do usuário com o programa de controle e programação da Base Móvel.

Os comandos definidos podem ser divididos por tipos de comandos, estes podem ser :

- Comandos de Set de Parâmetros : Estes comandos definem o valor de constantes usadas nos demais comandos, ou então habilitam e desabilitam o funcionamento de alguns dos modos de operação do robô ou do software de controle. Os comandos são os seguintes :

- SET MODE <0,1,2> : Seleciona o modo de operação do software de controle do robô. Os modos de operação possíveis são : modo normal (0), modo de programação (1), e modo de manutenção (2). Cada comando pode ser definido para funcionar ou não em cada modo de operação. O modo default é o modo normal, sendo que o modo de programação é usado para editar programas, e o modo de manutenção para ter acesso à valores de constantes internas ao sistema de controle, tais como valor do encoder, ou valor do eixo do joystick.
- SET JOYSTICK <0,1> : Este comando desabilita (0) ou habilita (1) o modo de movimento do robô através do joystick.
- SET TABLE <0,1> : Habilita (1) ou desabilita (0) a capturação de pontos para formar uma tabela que poderá posteriormente ser seguida. Os pontos podem ser capturados pressionando um botão do joystick, por exemplo. A tabela é armazenada na memória e depois pode ser salva em arquivo.
- SET TRANSMAXVEL <value> : Define o valor da velocidade máxima de translação do robô móvel em mm/s. Esta velocidade servirá como base para os comandos de movimento, já que o parâmetro que define a velocidade nestes comandos, é expresso em porcentagem da velocidade máxima.
- SET TRANSVEL <value> : Define a velocidade *default* em porcentagem da velocidade máxima a ser usada nos comandos de movimento.
- SET ROTMAXVEL <value> : Define a velocidade máxima de esterçamento das rodas. Assim como a velocidade máxima de translação, este parâmetro serve de base para os comandos de movimento.
- SET ROTVEL <value> : Define a velocidade de esterçamento das rodas expresso em porcentagem de velocidade máxima.
- SET POSITION <value1> <value2> : Define a posição absoluta atual do robô. Os parâmetros 1 e 2 representam as coordenadas da posição, respectivamente 'x' e 'y' se o sistema atual for cartesiano, e 'r' e 'θ' se o sistema for o de coordenadas polares.

- SET COORDENATE <0,1> : Define o sistema de coordenadas atual como sendo cartesiano (0), ou polar (1).

- Comandos de Movimento : São comandos que fazem com que o robô móvel se movimente, ou deixe de se movimentar até que uma condição seja satisfeita. Estes comandos são :
  - MOVEABS <valor1> <valor2> <valor3> : Este comando faz com que o robô se desloque para uma posição absoluta dada pelos dois primeiros parâmetros, com uma velocidade dada pelo terceiro parâmetro que é expressa em porcentagem da velocidade máxima pré-definida. O terceiro parâmetro é opcional, e quando omitido é substituído pelo valor da velocidade *default* também pré-definida. Os parâmetros 1 e 2 são respectivamente as coordenadas 'x' e 'y' da posição final quando o sistema de coordenadas ativo é o sistema cartesiano, e são as coordenadas 'r' e 'θ' quando o sistema ativo é o de coordenadas polares;
  - MOVEREL <valor1> <valor2> <valor3> : Este comando faz com que o robô se desloque para uma coordenada relativa. Os parâmetros seguem a mesma regra do comando anterior.
  - MOVE <valor1> <valor2> : Este comando realiza apenas uma translação simples de uma distância dada pelo parâmetro 1 com uma velocidade dada pelo parâmetro 2 expressa em porcentagem da velocidade máxima.
  - ROTATE <valor1> <valor2> : Este comando realiza uma mudança na direção das rodas de um valor dado pelo parâmetro 1 expresso em graus com uma velocidade dada pelo parâmetro 2 expressa em porcentagem da velocidade máxima de rotação das rodas.
  - MOVETAB <value1> : Faz com que o robô descreva uma trajetória que passe pelos pontos definidos em uma tabela previamente carregada pelo usuário. A trajetória descrita consciste em fazer uma interpolação linear entre os pontos, para isso estes pontos são armazenados de forma que representem distancias relativas em x e y. A velocidade utilizada é a velocidade definida no parâmetro 1.
  - RUN : Executa programa carregado na memória.

- WAITTIME <value> : Faz com que o robô espere na posição atual durante um determinado tempo definido pelo parâmetro do comando.
- WAITBUTTON : Faz com que o robô espere na posição atual até que seja pressionado um botão do joystick.
- WAITKEY : Faz com que o robô espere na posição atual até ser pressionado uma tecla do *keyboard.*
- OBS: Nos comandos de movimento, todos os parâmetros de velocidade se omitidos ou iguais a zero, são substituidos pela velocidade *default.* Os parâmetros de posição são sempre expresssos em milímetros, e os de velocidade em porcentagem de velocidade máxima.

- Comandos de Exibição : São comandos que mostram parâmetros e informações sobre o robô ao usuário. Estes comandos são :
  - SHOW VELOCITY : Mostra velocidade atual do robô
  - SHOW CVELOCITY : Mostra continuamente na tela a velocidade do robô até que seja pressionado ctrl-x.
  - SHOW POSITION : Mostra posição atual do robô
  - SHOW CPOSITION : Mostra continuamente na tela a posição do robô até que seja pressionado ctrl-x.
  - SHOW PARAMETERS : Mostra os parâmetros definidos pelo usuário, como velocidades máximas, velocidades default, mostra se modo joystick ativo ou inativo, etc.
  - ? (HELP) : Mostra um texto explicativo dos comandos disponíveis no modo atual de operação.
- Comandos de Manutenção : São comandos que funcionam apenas no modo de manutenção. Eles são :
  - CENC : Comando do modo de manutenção que mostra continuamente o valor do encoder até que seja pressionado ctrl-x.
  - READENC : Mostra o valor atual do encoder.
  - CJOY : Mostra continuamente o valor da posição do eixo do joystick até que seja pressionado ctrl-x.
  - READJOY <0,1> : Mostra o valor atual do eixo 'x'(0) ou 'y'(1) do

joystick

  - PWM <value1> <value2> : Escreve uma palavra dada pelo parâmetro 1 no driver de PWM definido pelo número dado pelo parâmetro 2.

- Comandos de Load e Upload de arquivos : São comandos que gravam e lêem arquivos de programas e tabelas. Os comandos são :
  - LOAD : Carrega arquivo de programa.
  - LOADTAB : Carrega arquivo de tabela. Este tipo de arquivo possui o seguinte formato : Cada valor deve ser colocado em uma linha diferente sendo que a primeira linha deve conter a coordenada X, e a segunda a coordenada Y, e assim por diante. Desta forma a cada duas linhas do arquivo, obtenho uma posição relativa diferente.
  - UPLOAD : Salva arquivo de programa.
  - UPLOADTAB : Salva arquivo de tabela no formato definido acima.
- Comandos de Edição de programas : São comandos que funcionam no modo de programação e servem para editar programas carregados na memória. Os comandos são :
  - <value> <string_de_comando> : Quando se entra no modo de programação e se começa a linha de comando com um número, a string digitada posteriormente ao número é inserida na linha dada por este número. Conforme o usuário for inserindo linhas de programa, estas vão sendo interpretadas e armazenadas na memória do robô. O usuário pode inseri-las em qualquer ordem, sendo que elas são internamente ordenadas pelo valor da linha. A execução normal do programa segue a sequência numérica das linhas, da de menor valor para a de maior valor. Exemplo de inserção de comando : 20 MOVEABS 10 15 - aqui o comando MOVEABS é colocado na linha de número 20 do programa. Obs: definiu-se que o valor de linha não pode ser zero.
  - CLEAR <value> : Apaga a linha cujo número se entrou como parâmetro. Caso o parâmetro seja nulo, ou omitido, todo o programa é apagado.
  - LIST <value> : Mostra a linha de programa desejada. Se o parâmetro for omitido ou nulo, o programa todo é mostrado.

- Comandos de Controle de programas : Até agora foram definidos os comandos do software de controle, e os comandos que poderão ser inseridos no corpo de um programa para executar movimentos, ou setar parâmetros. No entanto existem comandos que fazem a tarefa de controlar a execução do programa, e estes comandos estão definidos a seguir :
  - SETVAR <var_number> <number> : O programa permite utilizar 10 variáveis internas que são usadas como contadores. Estas variáveis podem ser incrementadas, decrementadas, e pode-se atribuir valores a elas. Esta função atribui um valor dado pelo parametro 2 ("number") a um contador referenciado pelo parametro 1 ("var_number"). Os contadores são referenciados por números de 0 a 9 (cada contador possui um número). Exemplo : SET VAR 0 10 - Atribui valor 10 ao contador 0.
  - INC <var_number> : Incrementa o contador "var_number".
  - DEC <var_number> : Decrementa o contador "var_number".
  - GOTO <value> : Muda a sequência de execução dos comandos pulando para a linha definida pelo parâmetro dado.
  - IFEQ <var_number> <number> <line_number> : Este comando faz uma comparação do valor contido na variável interna de índice "var_number" com o valor entrado no campo "number". Se os dois valores forem iguais, então a sequência de execução é desviada para a linha de número "line_number".
  - IFGT <var_number> <number> <line_number> : Se valor em "var_number" maior que "number" então vai para linha "line_number".
  - IFGE <var_number> <number> <line_number> : Se valor em "var_number" maior ou igual a "number" então vai para linha "line_number".
  - IFLT <var_number> <number> <line_number> : Se valor em "var_number" menor que "number" então vai para linha "line_number".
  - IFLE <var_number> <number> <line_number> : Se valor em "var_number" menor ou igual a "number" então vai para linha

"line_number".

- IFNE <var_number> <number> <line_number> : Se valor em "var_number" diferente de "number" então vai para linha "line_number".
- Observação : Todos os comandos apresentados aqui são usados apenas em programas, portanto devem ser precedidos do número da linha.

- Observação final quanto aos comandos : Todas os parâmetros de posição devem estar em milímetros, e os parâmetros de velocidade em porcentagem de velocidade, com exceção do comando se ajuste da velocidade máxima de translação, onde a velocidade é dada por mm/s. É importante observar que para todos os comandos aqui definidos existe um modo em que ele pode operar. Na tabela 1 se encontra a relação de todos os comandos definidos, e em que modo de operação estes comandos podem ser executados.
- Consideração em relação a funcionalidade do sistema : Na tabela 2 se encontra quando determinados comandos não podem ser acionados em função de outros comandos. Exemplo : o comando MOVE não pode ser acionado quando não terminou o movimento de um outro comando MOVE. Assim como esta restrição, a tabela 2 mostra as restrições existentes. Também é importante informar que pressionando a tecla ctrl-z, o sistema é resetado, e inicializado novamente.

A interface do usuário com o software de controle é feita através de interface serial. O esquema a seguir mostra o exemplo de como é a interface do sistema com o usuário. Observe que para cada modo de operação, é usado um prompt diferente.

Pode-se perceber pelo exemplo dado, de como é a interface com o usuário. Como resultado de trabalhos anteriores, já havia sido implementado grande parte dos elementos da interface como prompt, backspace, cabeçalho, etc. No entanto o sistema não apresentava todos os componentes de interface necessários, sendo que algumas rotinas que não estavam disponíveis foram desenvolvidas. Estas rotinas vão desde reconhecimento de comandos segundo a sintaxe definida, até impressão de números inteiros em decimal, o que não é imediato para o sistema. Grande parte do sistema de controle de execução de comandos também teve de ser implementada neste projeto.

Alguns detalhes de implementação serão mostrados posteriormente.

Exemplo de Interface

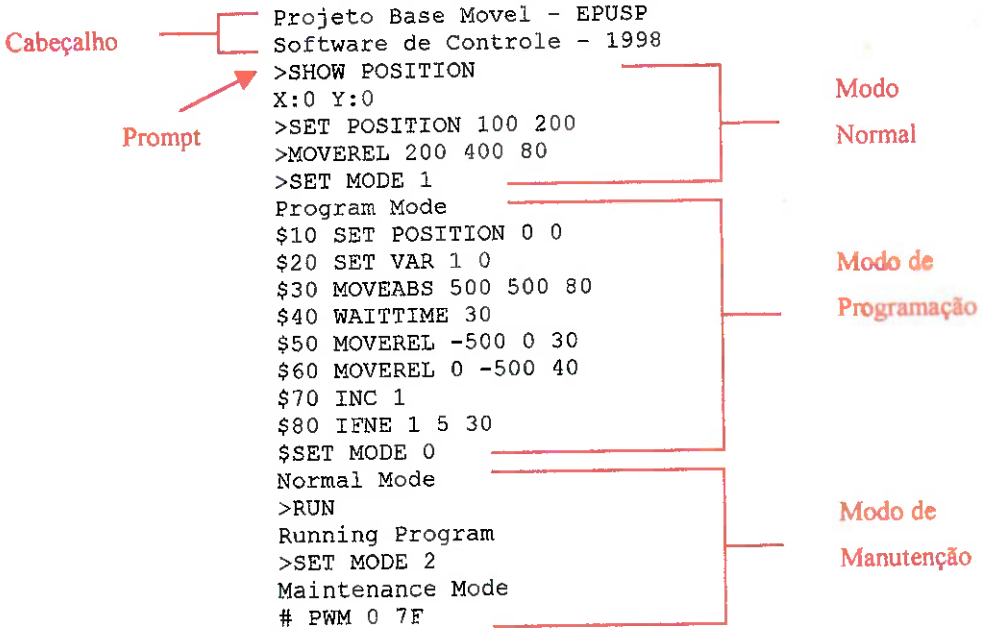


Tabela 1 - Relação dos comandos, e modo de execução em que são aceitos.

Comandos	Modo de Execução		
	Normal (>)	Programação ($)	Manutenção ($)
Comandos de Set de Parâmetros			
SET MODE	✓	✓	✓
SET JOYSTICK	✓	---	---
SET TABLE	✓	---	---
SET TRANSMAXVEL	✓	COM	---
SET TRANSVEL	✓	COM	---
SET ROTMAXVEL	✓	COM	---
SET ROTVEL	✓	COM	---
SET POSITION	✓	COM	---
SET COORDENATE	✓	COM	---

Tabela 1 (cont.) - Relação dos comandos, e modo de execução em que são aceitos.

Comandos	Modo de Execução		
	Normal (>)	Programação ($)	Manutenção ($)
Comandos de Movimento			
MOVEABS	✓	COM	---
MOVEREL	✓	COM	---
MOVE	✓	COM	✓
ROTATE	✓	COM	✓
MOVETAB	✓	COM	---
RUN	✓	---	---
WAITTIME	---	COM	---
WAITBUTTON	---	COM	---
WAITKEY	---	COM	---
Comandos de Exibição			
SHOW VELOCITY	✓	---	---
SHOW CVELOCITY	✓	---	---
SHOW POSITION	✓	---	---
SHOW CPOSITION	✓	---	---
SHOW PARAMETERS	✓	---	---
? (HELP)	✓	✓	✓
Comandos de Load e Upload			
LOAD	✓	---	---
LOADTAB	✓	---	---
UPLOAD	✓	---	---
UPLOADTAB	✓	---	---
Comandos de Edição			
LIST	---	✓	---
CLEAR	---	✓	---
DEL	---	✓	---
<value> <comando>	---	✓	---
Comandos de Manutenção			
CJOY	---	---	✓
CENC	---	---	✓
READENC	---	---	✓
READJOY	---	---	✓
PWM	---	---	✓

Tabela 1 (cont.) - Relação dos comandos, e modo de execução em que são aceitos.

Comandos	Modo de Execução		
	Normal (>)	Programação ($)	Manutenção (#)
Comandos de Controle de Programa			
SET VAR	---	COM	---
INC	---	COM	---
DEC	---	COM	---
IFNE, IFEQ, IFGE, ...	---	COM	---
GOTO	---	COM	---
END	---	COM	---

Obs: Na coluna de Modo de programação, quando está marcado 'COM' significa que o comando só pode ser usado no corpo de um programa como linha de comando. '✓' significa que o comando é executado.

Tabela 2 – Restrição quanto ao uso de alguns comandos em certas situações do sistema.

Comando que se deseja executar	Ação que está sendo acontecendo no robô			
	Modo Joystick ligado	Executando Programa	Executando Tabela	Robô em movimento
SET JOYSTICK 1	---	---	---	---
RUN	---	---	---	---
MOVE	---	✓	---	---
MOVEREL	---	✓	---	---
MOVEABS	---	✓	---	---
MOVETAB	---	✓	---	---
SET POSITION	✓	✓	---	---
SET TRANSMAXVEL	---	✓	---	---
SET MODE	---	---	---	---
LOAD, LOADTAB	---	---	---	---
PWM	---	---	---	---

OBS: ✓ : indica que comando pode ser executado

--- : indica que não pode ser executado

5. Projeto do Sistema de Controle de Posição e Velocidade

O movimento de translação da Base Móvel é efetuado por um motor DC. Desta forma, é necessário que haja um controle em malha fechada da posição do robô. Este sistema de controle de posição será utilizado por todos os comandos de movimento que realizam uma translação do robô.

Para obter o sistema de controle de posição, primeiramente modelou-se a Base Móvel com o motor DC, feito isto, projetou-se um controlador de posição com o auxílio do programa Matlab. Foram feitos então simulações do modelo, e então discretizou-se o controlador para se obter a equação de diferenças dele. A partir desta equação é possível a implementação do controle de posição na Base Móvel.

O detalhamento das etapas feitas se encontra a seguir.

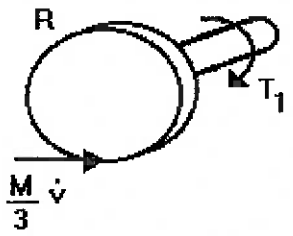
5.1 Modelo dinâmico da planta

Para se chegar ao modelo dinâmico da Base Móvel, modelou-se o motor DC e também a parte mecânica do robô.

5.1.1 Modelo dinâmico mecânico do robô

O modelo dinâmico do robô quando este acelera, pode ser determinado pela seguinte sequência de cálculos :

Sabendo-se que o torque resultante em uma roda é igual a inércia angular vezes a variação de velocidade angular dela; temos que em uma das rodas do robô :



$$T_1 = J_R \cdot \frac{d\omega_R}{dt} + \frac{M \cdot R^2}{3} \frac{d\omega_R}{dt}$$

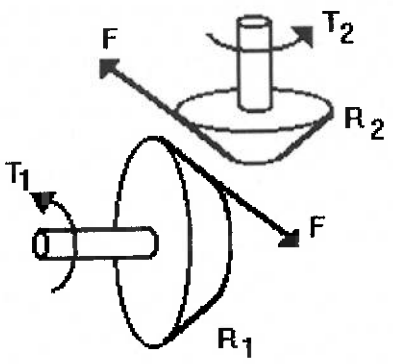
onde, J_R é a inércia da roda (kg.m^2)

M é a massa do robô (kg)

R é o raio da roda (m)

ω_R é a velocidade angular da roda (rad/s)

No par de engrenagens que liga a roda às polias, temos :



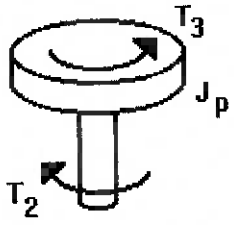
$$\frac{T_1}{T_2} = \frac{R_1}{R_2} \Rightarrow T_2 = \left(\frac{R_2}{R_1}\right) \cdot \left(\frac{M \cdot R^2}{3} + J_R\right) \frac{d\omega_R}{dt}$$

onde, R_1 é o raio da engrenagem maior

R_2 é o raio da engrenagem menor

Nas expressões acima, foram desprezadas as inércias das engrenagens

Em uma das polias movidas :

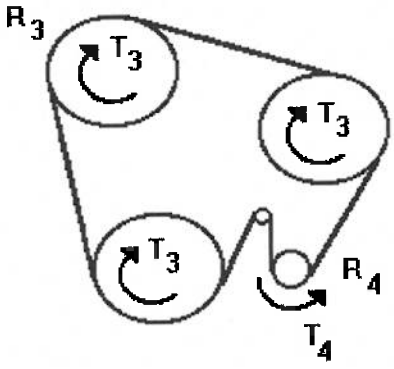


$$T_3 - T_2 = J_P \cdot \frac{d\omega_P}{dt} \Rightarrow T_3 = \left[\left(\frac{R_2}{R_1}\right)^2 \left(\frac{M \cdot R^2}{3} + J_R\right) + J_P\right] \frac{d\omega_P}{dt}$$

onde, J_P é a inércia da polia movida ($kg.m^2$)

ω_P é a velocidade angular na polia movida (rad/s)

Devido a relação de transmissão nas polias dentadas, o torque na polia motora é :



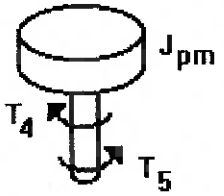
$$\frac{T_4}{3 \cdot T_3} = \frac{R_4}{R_3}$$

$$T_4 = \left[\left(\frac{R_2}{R_1}\right)^2 M \cdot R^2 + 3\left(\frac{R_2}{R_1}\right)^2 J_R + 3J_P\right] \frac{R_4}{R_3} \cdot \frac{d\omega_P}{dt}$$

onde, R_4 é o raio da polia menor (motora)

R_3 é o raio da polia maior (movida)

Na polia motora, têm-se :



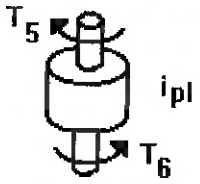
$$T_5 - T_4 = J_{PM} \frac{d\omega_{PM}}{dt}$$

$$T_5 = \left[\left(\frac{R_2}{R_1}\right)^2 \left(\frac{R_4}{R_3}\right)^2 M \cdot R^2 + 3\left(\frac{R_2}{R_1}\right)^2 \left(\frac{R_4}{R_3}\right)^2 J_R + 3J_P \left(\frac{R_4}{R_3}\right)^2 + J_{PM}\right] \frac{d\omega_{PM}}{dt}$$

onde, J_{PM} é a inércia da polia motora ($kg.m^2$)

ω_{PM} é a velocidade angular na polia motora (rad/s)

Antes de chegar ao motor, existe um planetário. Então o torque fica :



$$T_6 = \frac{1}{i_{PL}} T_5$$

$$T_6 = \left[\left(\frac{R_2}{R_1}\right)^2 \left(\frac{R_4}{R_3}\right)^2 M \cdot R^2 + 3\left(\frac{R_2}{R_1}\right)^2 \left(\frac{R_4}{R_3}\right)^2 J_R + 3J_P \left(\frac{R_4}{R_3}\right)^2 + J_{PM} \right] \left(\frac{1}{i_{PL}}\right)^2 \frac{d\omega}{dt}$$

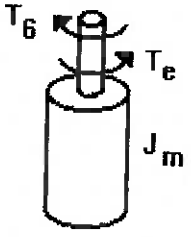
onde, i_{PL} é a relação de transmissão do planetário ($i_{PL} > 1$)

ω é a velocidade angular do motor

Pode-se escrever a expressão acima da seguinte forma :

$$T_6 = J_{EQ} \frac{d\omega}{dt}$$

Finalmente, a equação de torque no motor levando-se em conta a inércia do robô fica :



$$T_e - T_6 = J_M \frac{d\omega}{dt} \Rightarrow T_e = \left(J_M + J_{EQ}\right) \frac{d\omega}{dt}$$

onde, J_M é a inércia do motor ($kg.m^2$)

T_e é o torque elétrico

Portanto a inércia equivalente do robô, pode ser dada pela seguinte expressão :

$$J_{EQ} = \left(\frac{R_2}{R_1}\right)^2 \left(\frac{R_4}{R_3}\right)^2 \left(\frac{1}{i_{PL}}\right)^2 M \cdot R^2 + 3\left(\frac{R_2}{R_1}\right)^2 \left(\frac{R_4}{R_3}\right)^2 \left(\frac{1}{i_{PL}}\right)^2 J_R + 3J_P \left(\frac{R_4}{R_3}\right)^2 \left(\frac{1}{i_{PL}}\right)^2 + J_{PM} \left(\frac{1}{i_{PL}}\right)^2$$

Como em relação à massa do robô, as inércias das polias são de ordem de grandeza inferior, pode-se desprezá-las para efeitos de projeto do controlador de posição. Portanto nesta expressão, considera-se apenas o termo envolvendo a massa :

$$J_{EQ} = \left(\frac{R_2}{R_1}\right)^2 \left(\frac{R_4}{R_3}\right)^2 \left(\frac{1}{i_{PL}}\right)^2 M \cdot R^2$$

5.1.2 Modelo Dinâmico do Motor DC

Pode-se representar uma máquina de corrente contínua da seguinte forma :

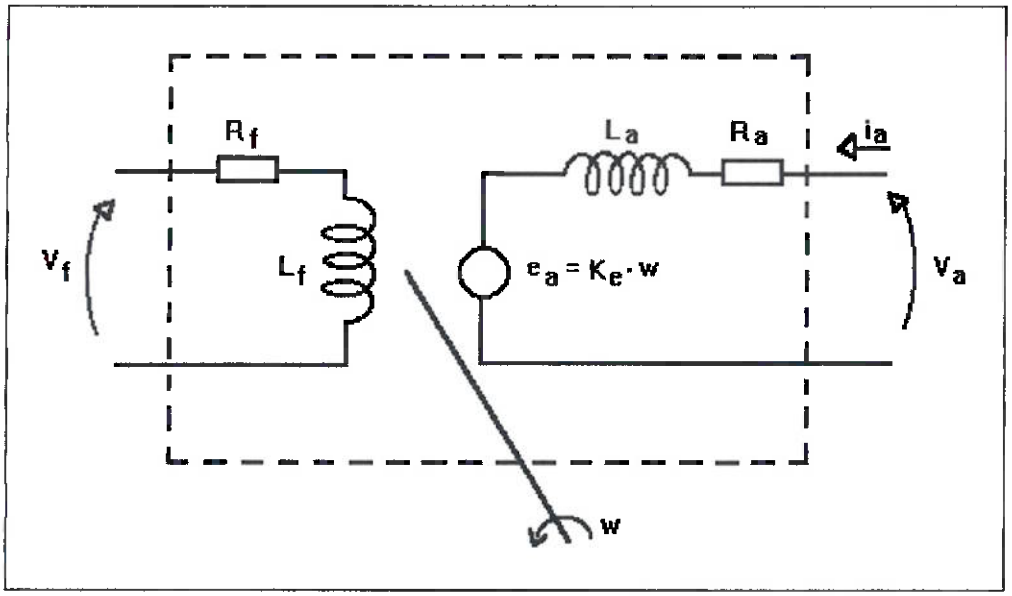


Figura 1 - Esquema de motor de corrente contínua

Nomenclatura :

V_a - Tensão da armadura

i_a - Corrente de armadura ou de torque

e_a - Força contra-eletromotriz

V_f - Tensão de excitação ou de campo

i_f - Corrente de excitação ou de campo

L_a - indutância de armadura

R_a - Resistência de armadura

L_f - Indutância de campo

R_f - resistência de campo

As equações dinâmicas são :

$$V_f = R_f \cdot i_f + L_f \frac{di_f}{dt}$$

$$V_a = R_a \cdot i_a + L_a \frac{di_a}{dt} + e_a$$

$$T_e - T_L = J \frac{d\omega}{dt} + B \cdot \omega$$

onde : T_e é o torque elétrico (Nm)

T_L é o torque de carga (Nm)

$J = J_M + J_{EQ}$ (Inércia do motor + Inércia equivalente do robô) (Nm)

B é um tipo de atrito expresso em Nm/(rad/s)

ω é a rotação do motor em rad/s

Para motores de corrente contínua com imãs permanentes, $V_f = R_f \cdot i_f$, e o fluxo é constante. Portanto pode-se dizer que :

$$T_e = K_t \cdot i_a$$

$$e_a = K_e \cdot \omega$$

Fazendo a Transformada de La Place nestas equações, tem-se :

$$V_a(s) = R_a \cdot I_a(s) + L_a \cdot I_a(s) \cdot s + e_a(s)$$
$$T_e(s) - T_L(s) = J \cdot W(s) \cdot s + B \cdot W(s)$$
$$e_a(s) = K_e \cdot W(s)$$
$$T_e(s) = K_t \cdot I_a(s)$$

Manipulando as equações acima, chega-se à :

$$I_a(s) = \frac{V_a(s) - T_L(s)}{(R_a + L_a \cdot s)}$$

$$W(s) = \frac{T_e(s) - T_L(s)}{(B + J \cdot s)}$$

Podemos expressar as equações na forma do seguinte grafo :

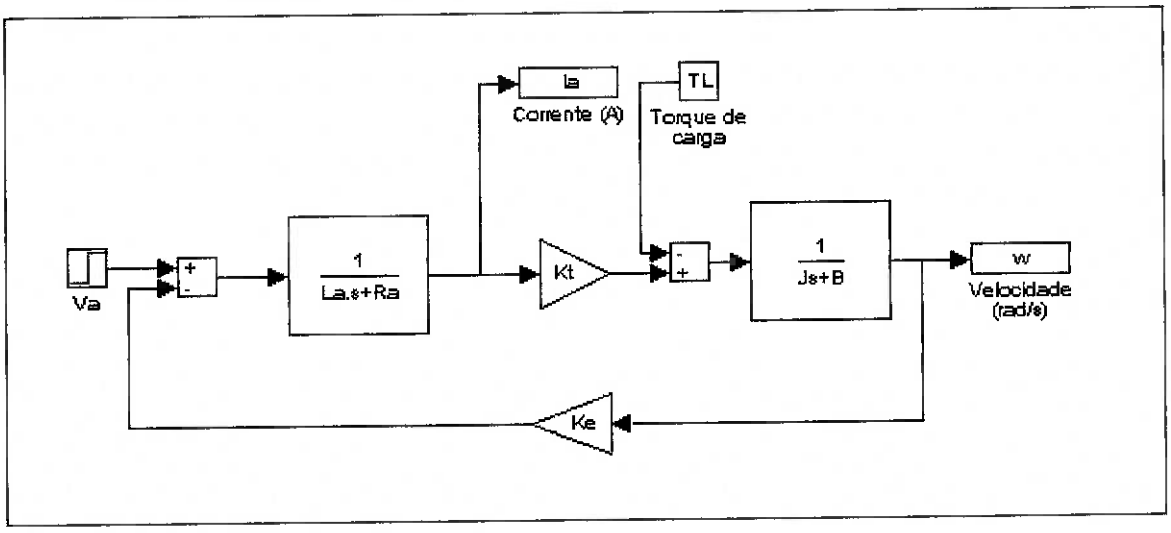


Figura 2 - Diagrama de blocos do Modelo da Máquina de Corrente Contínua

5.1.3 Características do Motor DC e do Robô Móvel

O motor DC usado na Base Móvel é um motor de 15 Watt da Maxon Motor. O número deste motor é o 966, e segundo o catálogo do fabricante ele possui as seguintes características :

- Voltagem nominal : 12 V
- Resistência de terminal (R_a) : 3.18 Ohm
- Constante de torque (K_T) : $23x10^{-3}$ Nm/A
- Constante elétrica (K_E) : $23x10^{-3}$ V/(rad/s)
- Inércia do rotor (J_M) : $2,43x10^{-6}$ kg.m^2
- Indutância do terminal (L_a) : $0.53x10^{-3}$ H

A Base Móvel possui as seguintes características :

- Massa (M) : 7,2 kg
- Raio das rodas (R) : $41x10^{-3}$ m
- R_2/R_1 (relação de transmissão das engrenagens) : 4
- R_4/R_3 (relação de transmissão das polias) : 1.25
- i_{PL} (relação de transmissão no planetário) : 5.2
- $J_{EQ} = 1,8x10^{-5}$

5.1.4 Torque de Carga (T_L)

Admitindo-se que B = 0, quando não há variação de velocidade : $T_L = T_E = K_T.i_a$

Portanto, mediu-se a corrente quando a velocidade é máxima no motor, e a partir daí calculou-se o torque de carga. A corrente observada foi da ordem de 0.4 A. Então, como $K_T = 23.0x10^{-3}$ N/A , conclui-se que o torque de carga é da ordem de 0.0092 N. Admitiu-se que este torque é constante.

5.2 Projeto do Controle de Posição

Simplificando o diagrama de blocos do motor, chega-se a seguinte expressão que descreve a planta :

$$G(s) = \frac{K_T}{(J \cdot L_a) \cdot s^2 + (R_a \cdot J + L_a \cdot B) \cdot s + (R_a \cdot B + K_T \cdot K_e)}$$

onde, J é a inércia do motor somado com a inércia do robô

Pode-se admitir para este motor que sua indutância é zero, já que o valor desta é muito inferior em comparação com o valor do termo envolvendo resistência e inércia. Também se considerou B = 0 para este problema.

Portanto, pode-se simplificar a expressão para :

$$G(s) = \frac{K_T}{R_a \cdot J \cdot s + K_T \cdot K_e}$$

O sistema em malha fechada de posição fica sendo :

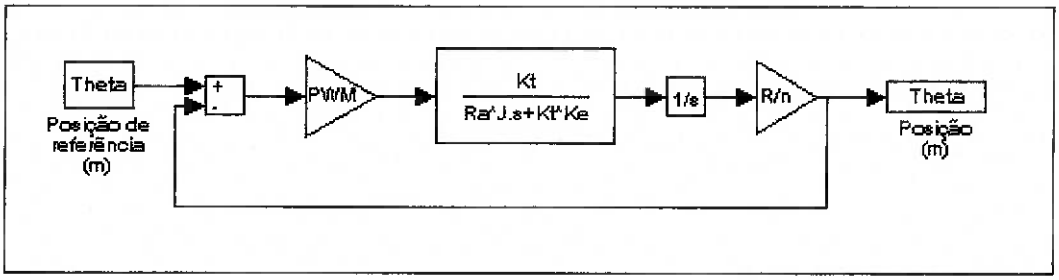


Figura 3 - **Planta simplificada com ganhos de PWM e relação de transmissão**

Onde PWM significa um ganho que transforma valores entre -128 e 128 em valores entre -12 e 12, que é a tensão de alimentação do motor. R/n é o ganho que transforma angulo em rad do eixo do motor em distância em metro percorrida pela roda.

A função de transferência em malha aberta da planta, se adicionarmos um ganho K antes do PWM, fica :

$$\frac{X(s)}{Xr(s)-X(s)} = \frac{K \cdot K_{PWM} \cdot K_T \cdot R}{n(R_a \cdot J \cdot s^2 + K_T \cdot K_e \cdot s)}$$

Para os valores dados anteriormente, com n = 4x1,25x5,2 ; e $K_{PWM} = 9{,}375x10^{-2}$:

$$\frac{X(s)}{X_r(s)-X(s)} = \frac{0{,}0523 \cdot K}{s \cdot (s+8{,}14)}$$

É desejável que a resposta em regime deste sistema com um controlador proporcional K, para uma entrada rampa, seja menor que 1%, então pode-se tirar que :

$$\varepsilon = \lim_{s\to 0} s \cdot (Xr(s)-X(s)) = \lim_{s\to 0} s \cdot \frac{Xr(s)}{1+\left[\frac{X(s)}{(Xr(s)-X(s))}\right]} = 0{,}01$$

Para $X_r(s) = 1/s^2$, temos que :

$$\varepsilon = \lim_{s\to 0} \frac{(s+8{,}14)}{s^2+8{,}14 \cdot s+0{,}0523K} = 0{,}01$$

Portanto : K > 15557

Rescrevendo a função de transferência em malha aberta da planta com este controlador proporcional, fica :

$$G(s) \cdot K = \frac{815}{s \cdot (s+8{,}14)}$$

No entanto, a resposta a rampa unitária apresenta uma oscilação inicial e final, como verifica-se na figura 4 .

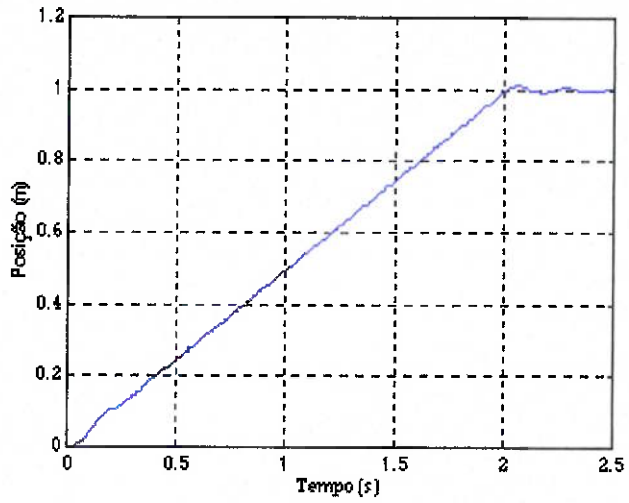


Figura 4 - Gráfico da planta com controlador proporcional à resposta à rampa

Para diminuir esta oscilação, escolheu-se projetar um controlador do tipo phase-lead, que irá funcionar juntamente com o ganho K projetado acima. Este controlador possui a seguinte forma :

$$G_C(s) = \frac{(1+\alpha \cdot T \cdot s)}{(1+T \cdot s)}$$

A função de malha aberta fica sendo :

$$G(s) \cdot G_c(s) = \frac{815}{s \cdot (s+8{,}14)} \cdot \frac{(1+\alpha \cdot T \cdot s)}{(1+T \cdot s)}$$

A equação característica de malha fechada, ou seja, o denominador da função de malha fechada $\frac{G(s) \cdot G_C(s)}{1+G(s) \cdot G_C(s)}$ é :

$s.(s+8{,}14)(1+T.s) + 815.(1+\alpha.T.s) = 0$

Dividindo por $s^2 + 8{,}14.s + 815$, fica :

$$1 + \frac{815 \cdot \alpha \cdot T \cdot s}{s \cdot (s+8{,}14) \cdot (1+T \cdot s) + 815} = 0$$

Escolhe-se valores de T, para plotar o root-locus da expressão :

$$\frac{815 \cdot \alpha \cdot T \cdot s}{s \cdot (s+8{,}14) \cdot (1+T \cdot s) + 815}$$

Chegou-se a um valor de T = 0,005. Que apresentou um root-locus bom para que seja determinado um α que resultasse em um baixo sobresinal.

O gráfico do root locus obtido, está na figura 5 .

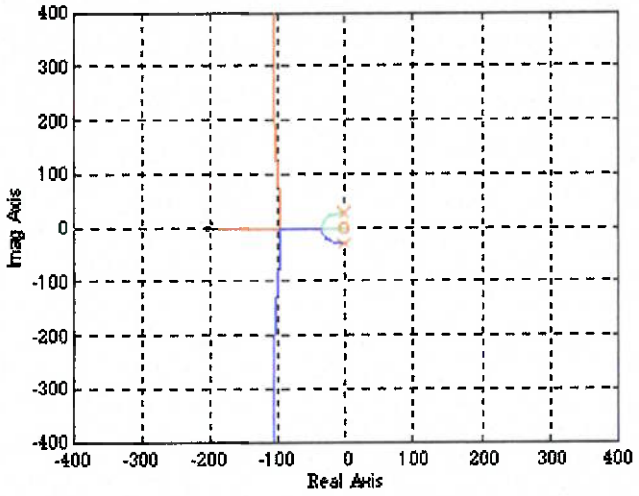


Figura 5 - Root locus para T=0,005

A região que se deseja que o sistema trabalhe, está ampliada na figura abaixo :

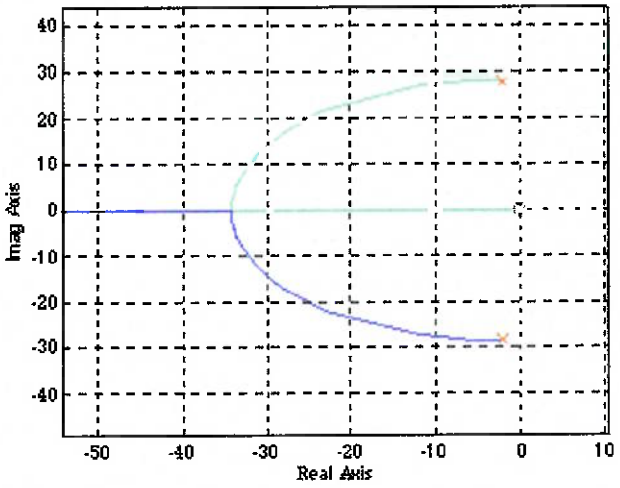


Figura 6 - Região do root locus que se deseja que os polos se localizem

A partir desta figura determinou-se o valor de α como sendo ao valor em que ocorre o cruzamento das linhas. Portanto :

$\alpha = 11,4$

Com estes valores, a resposta a degrau unitário pode ser vista na figura 7 .

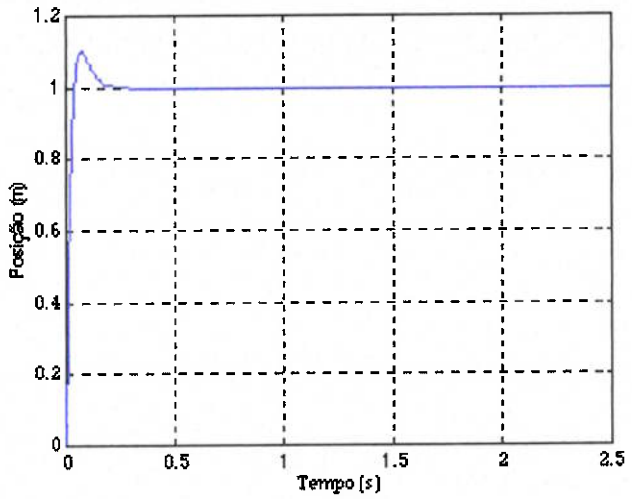


Figura 7 - Resposta à degrau unitário do sistema controlado

Observa-se que o sistema possui um alto tempo de resposta, com um sobresinal de 10%.

Para uma entrada rampa unitária, com velocidade de 0,5 m/s. A resposta obtida foi a seguinte :

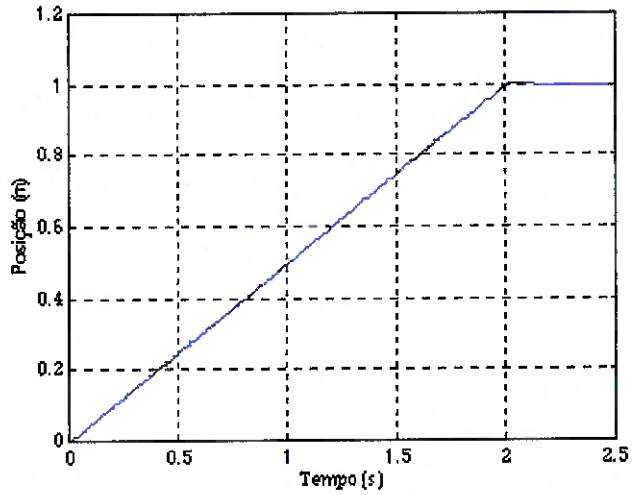


Figura 8 - Resposta à entrada rampa do sistema controlado

Observa-se que a resposta teve a característica desejada, com sobresinal dentro dos 1% que havia sido especificado.

Portanto, planta do controlador é a seguinte :

$$G_C(s) = \frac{1 + 0{,}057 \cdot s}{1 + 0{,}005 \cdot s}$$

E existe um ganho proporcional de K = 15557

5.3 Resultados da Simulação do Controle de Posição

Para simular como seria o comportamento do robô com este controlador, utilizou-se o simulink como ferramenta. No modelo, foi considerado o torque de carga T_L, também foi considerado a saturação no PWM, e também uma região, que foi verificada na prática, onde para um valor de entrada não há resposta de movimento no motor.

O modelo criado para simulação se encontra na figura 9. Nesta, o ganho K1 é o ganho do controlador, K2 é o ganho do PWM, e K3 é o ganho da relação de transmissão que transforma em velocidade do motor em rad/s para velocidade na roda em m/s.

O bloco denominado PWM contém um saturador, e blocos que simulam a região onde não existe resposta. O bloco aberto pode ser visto na figura 10.

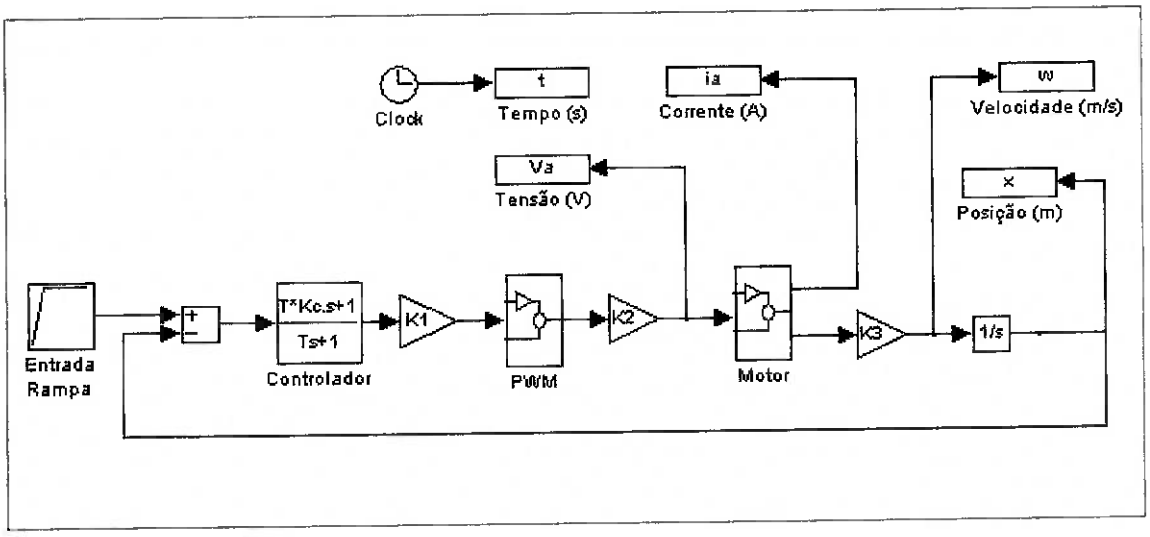


Figura 9 - Modelo usado na simulação

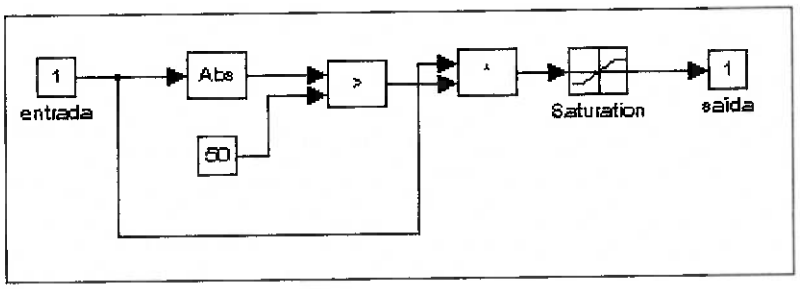


Figura 10 - Bloco do PWM com saturação

O bloco de motor corresponde ao modelo dinâmico apresentado na figura 3.

Após as simulações, verificou-se que o resultadose encontrava dentro do esperado, sendo assim, os valores dos ganhos são :

K1 = 15557;

K2 = $9.375x10^{-2}$;

K3 = $1.577x10^{-3}$;

O sinal de entrada utilizado foi uma rampa de posição que corresponde a uma velocidade de 0,5 m/s. O gráfico do sinal de entrada é :

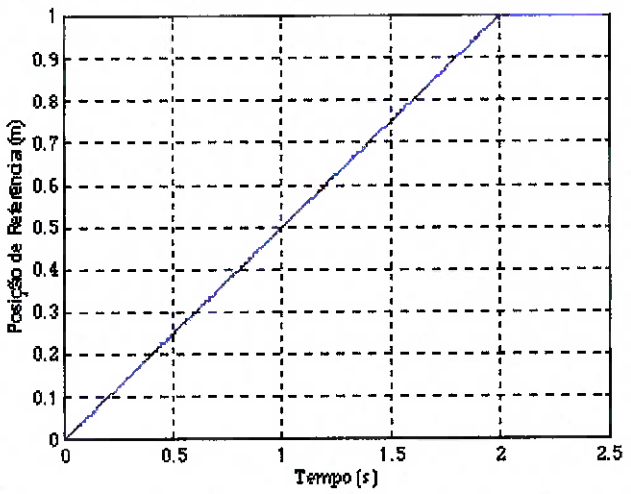


Figura 11 - Entrada usada na simulação

Foram plotados gráficos de tensão de alimentação do motor (V_a), velocidade do robô (w), posição do robô (x), e corrente no motor (I_a).

Os resultados obtidos se encontram nas figuras abaixo :

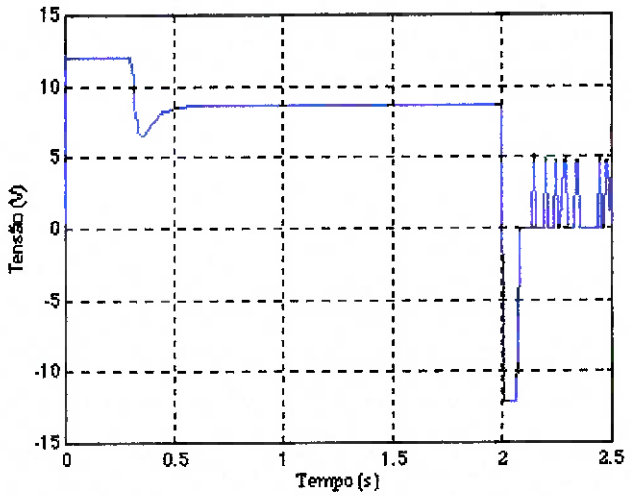


Figura 12 - Gráfico da tensão com saturação

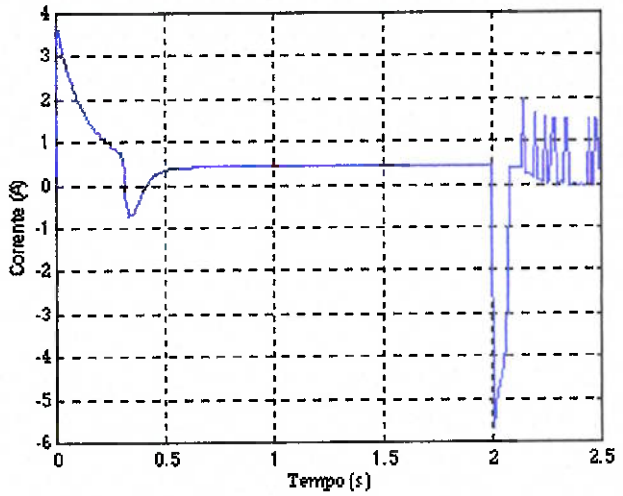


Figura 13 - Gráfico da corrente

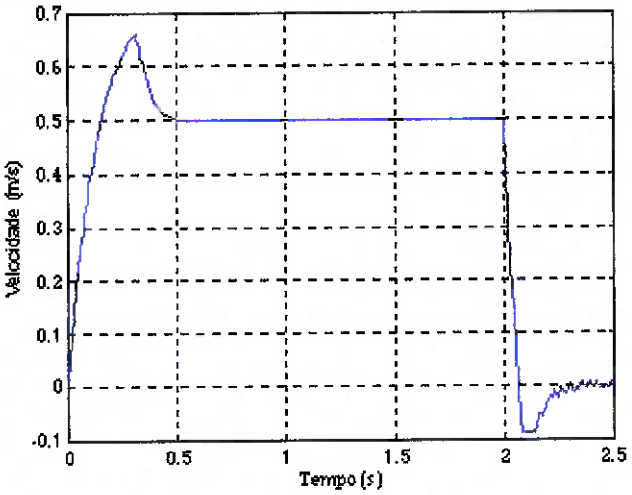


Figura 14 - Gráfico da velocidade

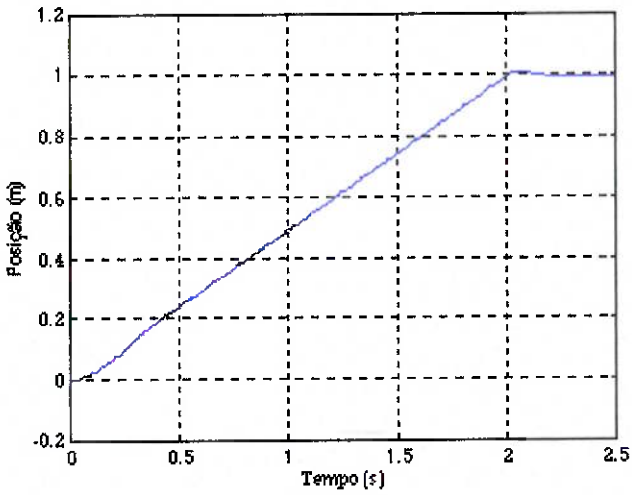


Figura 15 - Gráfico da posição

Se não houvesse saturação nos 12 V, os resultados seriam :

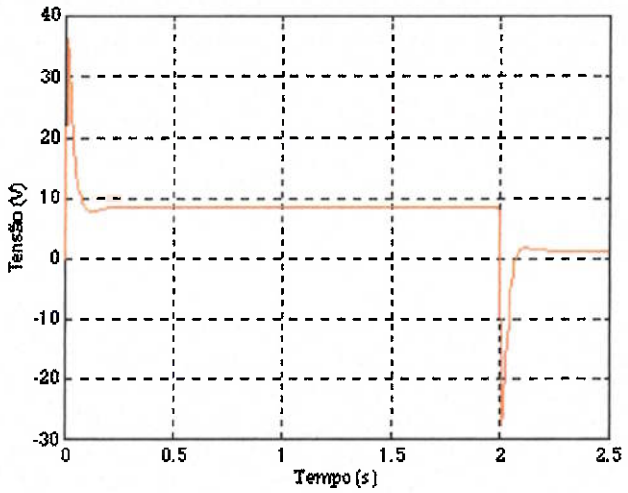


Figura 16 - Gráfico da tensão sem a saturação

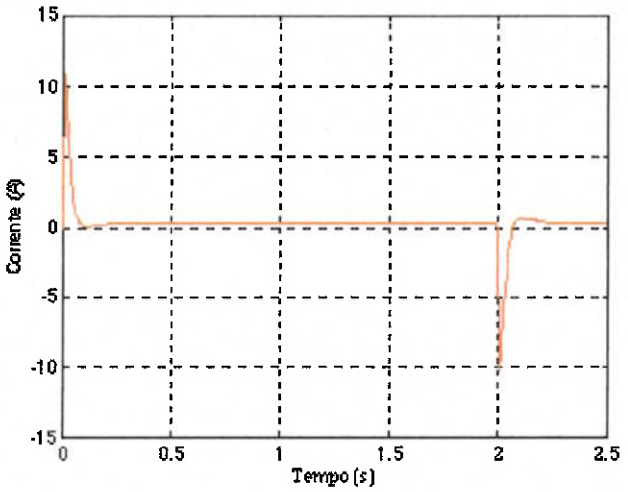


Figura 17 - Gráfico da corrente

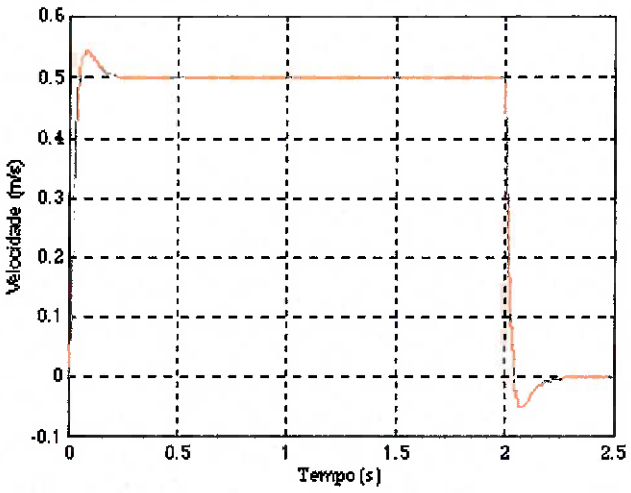


Figura 18 - Gráfico da velocidade

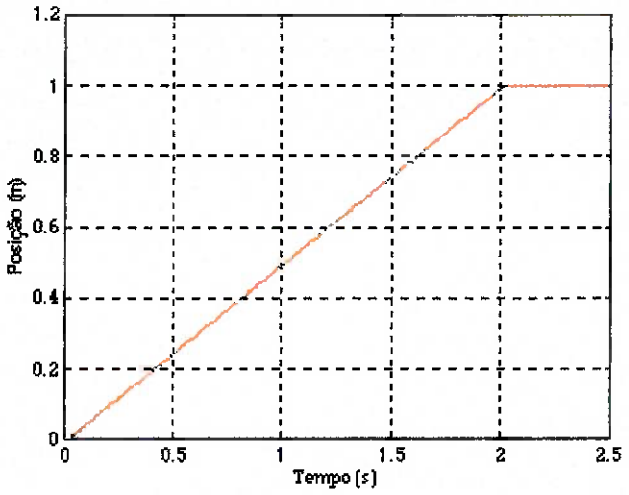


Figura 19 - Gráfico da posição

5.4 Discretização do Controlador de Posição

O controlador possui a seguinte função de transferência :

$$G_c(s) = K\frac{(1+\alpha \cdot T \cdot s)}{(1+T \cdot s)}$$

Utilizando-se o método de transformação bilinear, ou seja, fazendo-se a seguinte substituição : $s = \frac{2}{T_a}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)$, onde T_a é o período de amostragem; pode-se obter a expressão do controlador discretizado. Fazendo esta substituição chega-se a :

$$G_c(z) = K \cdot \left[\frac{(T_a + 2\cdot\alpha\cdot T) + (T_a - 2\cdot\alpha\cdot T)\cdot z^{-1}}{(T_a + 2\cdot T) + (T_a - 2\cdot T)\cdot z^{-1}}\right]$$

Esta expressão pode ser escrita como :

$$K\cdot[(T_a + 2\cdot\alpha\cdot T)\cdot U(z) + (T_a - 2\cdot\alpha\cdot T)\cdot z^{-1}\cdot U(z)] = (T_a + 2\cdot T)\cdot Y(z) + (T_a - 2\cdot T)\cdot z^{-1}\cdot Y(z)$$

Transformando para equação de diferenças, a expressão acima fica :

$$y_{atual} = -\frac{(T_a - 2\cdot T)}{(T_a + 2\cdot T)}\cdot y_{anterior} + K\cdot\frac{(T_a + 2\cdot\alpha\cdot T)}{(T_a + 2\cdot T)}\cdot u_{atual} + K\cdot\frac{(T_a - 2\cdot\alpha\cdot T)}{(T_a + 2\cdot T)}\cdot u_{anterior}$$

Para $T = 0{,}005$, $\alpha = 11{,}4$:

$$y_{atual} = -\frac{(T_a - 0{,}01)}{(T_a + 0{,}01)}\cdot y_{anterior} + K\cdot\frac{(T_a + 0{,}114)}{(T_a + 0{,}01)}\cdot u_{atual} + K\cdot\frac{(T_a - 0{,}114)}{(T_a + 0{,}01)}\cdot u_{anterior}$$

Para $T_a = 5$ ms :

$$y_{atual} = 0{,}33\cdot y_{anterior} + K\cdot 7{,}93\cdot u_{atual} - K\cdot 7{,}27\cdot u_{anterior}$$

Para $T_a = 7{,}5$ ms :

$$y_{atual} = 0{,}143\cdot y_{anterior} + K\cdot 6{,}94\cdot u_{atual} - K\cdot 6{,}09\cdot u_{anterior}$$

Sendo que y é a saída do controlador, u é a entrada do controlador, e K é o ganho do controlador que foi determinado como sendo K=15557.

A expressão determinada anteriormente pode ser usada para a implementação do controlador no robô móvel, no entanto na expressão a entrada u deve ser dada em metro.

No robô móvel, o que é utilizado para determinar a posição são pulsos de encoder, e a cada rotação do motor, são gerados 500 pulsos de encoder.

Então, um deslocamento de 1 metro do robô equivale a 50464 pulsos.

Uma outra relação importante, é que para um período de amostragem T_a, uma velocidade do robô de 1 m/s equivale a medir uma variação de $50465T_a$.

Para converter a expressão que dá o resultado da saída do controlador a fim de que a entrada, que é o erro entre a referência e a posição real atual, possa ser dada em pulsos de encoder, deve-se transformar este valor de pulsos de encoder para metro. Isto equivale a dividir os termos da expressão que são dependentes da entrada (u) por 50464. Portanto, basta usar ao invés de K, um K' = 15557/50464.

Então, para a expressão fica :

Para 5 ms :

$$y_{atual} = 0{,}33 \cdot y_{anterior} + 2{,}44 \cdot u_{atual} - 2{,}24 \cdot u_{anterior}$$

Para 7,5 ms:

$$y_{atual} = 0{,}14 \cdot y_{anterior} + 2{,}14 \cdot u_{atual} - 1{,}88 \cdot u_{anterior}$$

Sendo que y é a saída para o PMW, e u é a entrada dada em pulsos de encoder.

Observe que ao implementar o algorítmo deve-se saturar o valor da saída calculado em 127 e –127.

Caso haja necessidade de calcular a expressão para outro período de amostragem, ou até mesmo outro valor para os coeficientes da planta, basta seguir o roteiro de calculo deixado nesta seção e estes valores podem ser mudados sem grandes dificuldades.

O controle determinado aqui, foi implemtado sendo que se fez testes com período de amostragem de 5 ms e de 7,5 ms, comparando os resultados obtidos com o resultado de simulação. Os resultados obtidos para o movimento de 1 metro com velocidade de 320 mm/s se encontram dos gráficos seguintes, sendo que em vermelho está o resultado de simulação, e em azul o resultado obtido com o robô.

Para 5 ms :

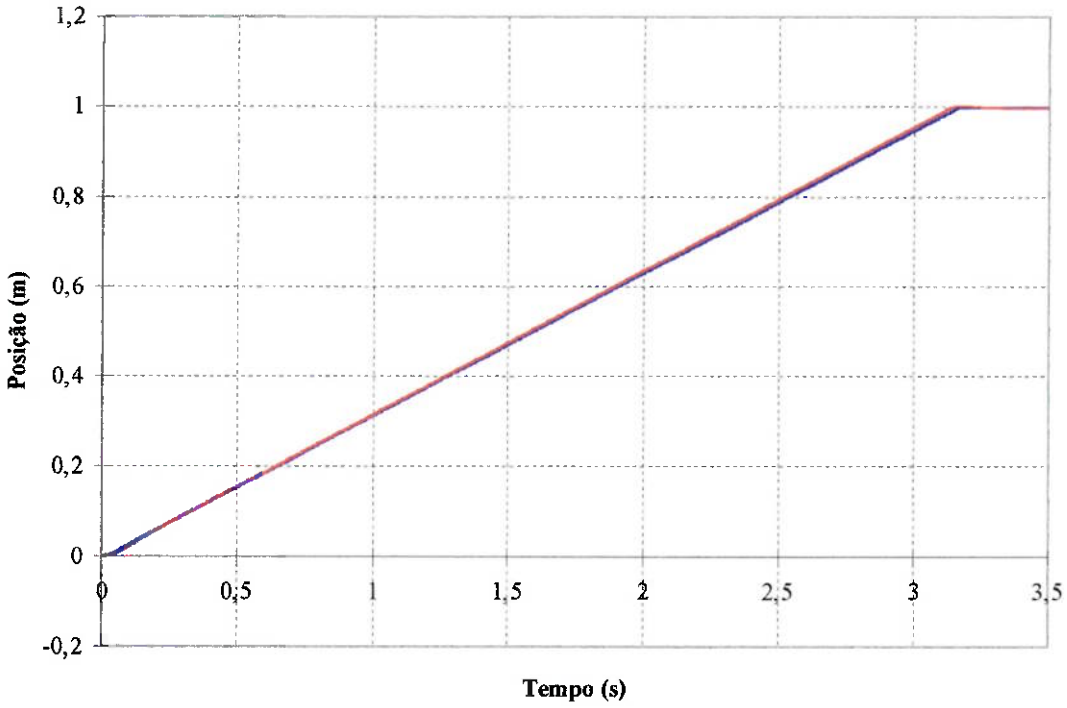


Figura 20 - Gráfico de posição observado (5 ms)

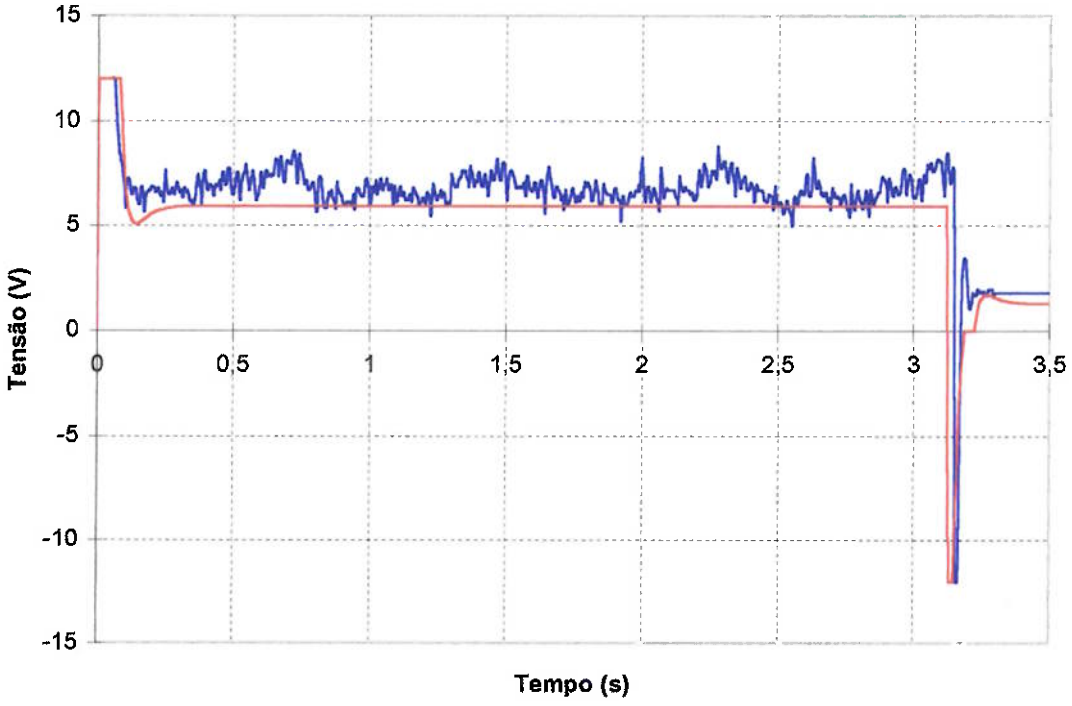


Figura 21 - Gráfico de tensão observado (5 ms)

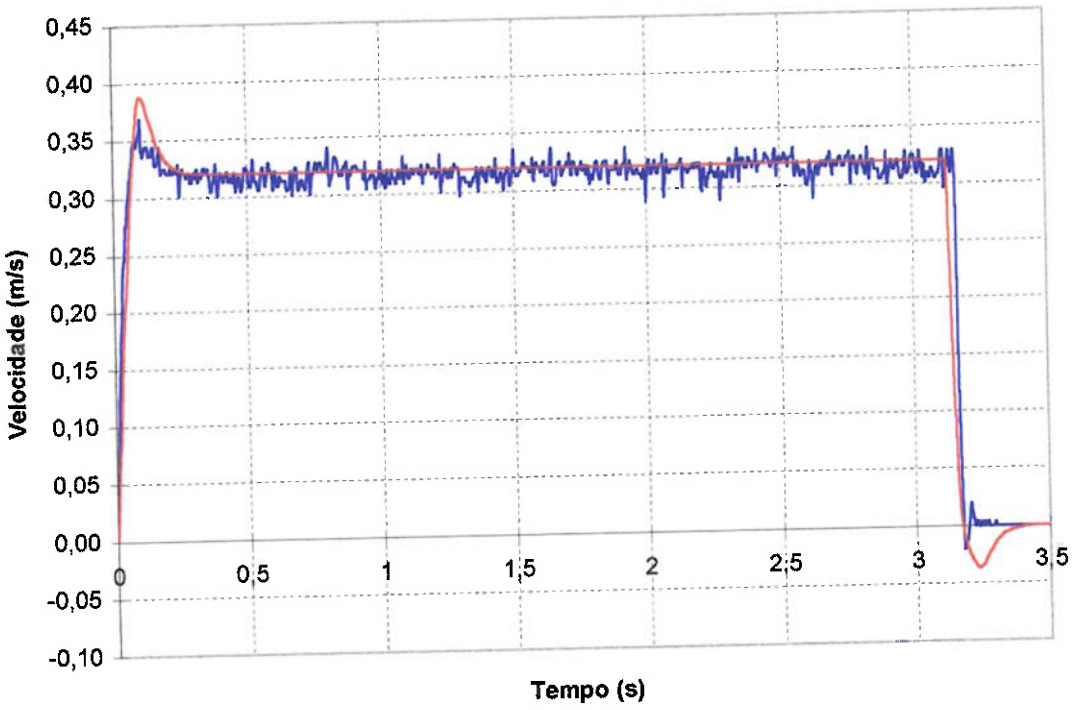


Figura 22 - Gráfico de velocidade observado (5 ms)

Para 7,5 ms :

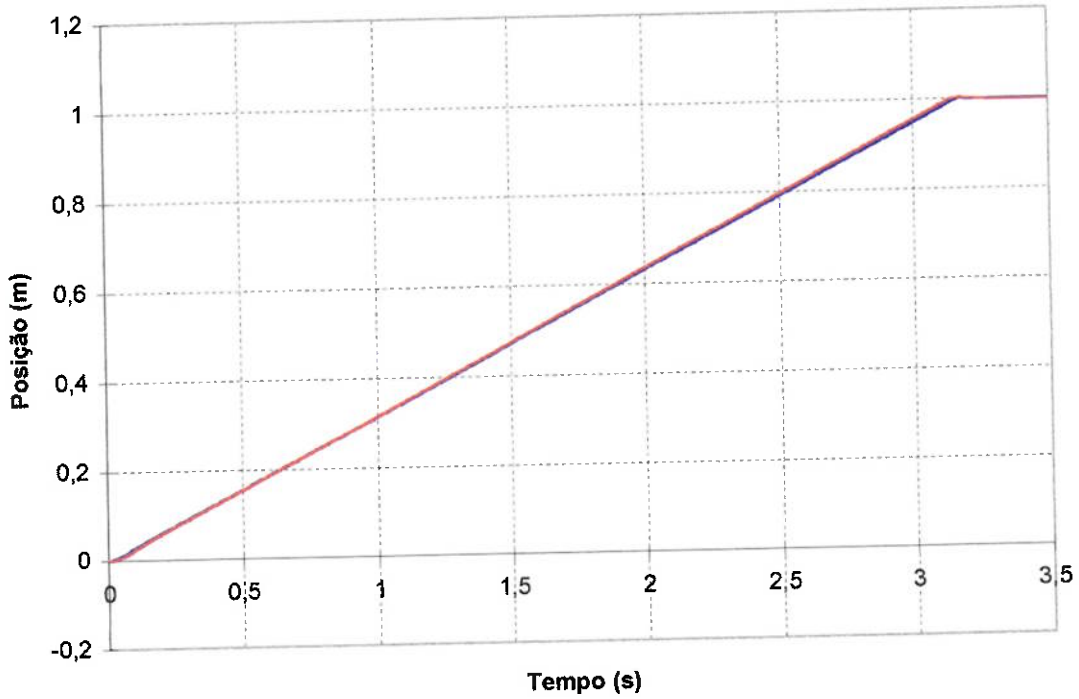


Figura 23 - Gráfico de posição observado (7,5 ms)

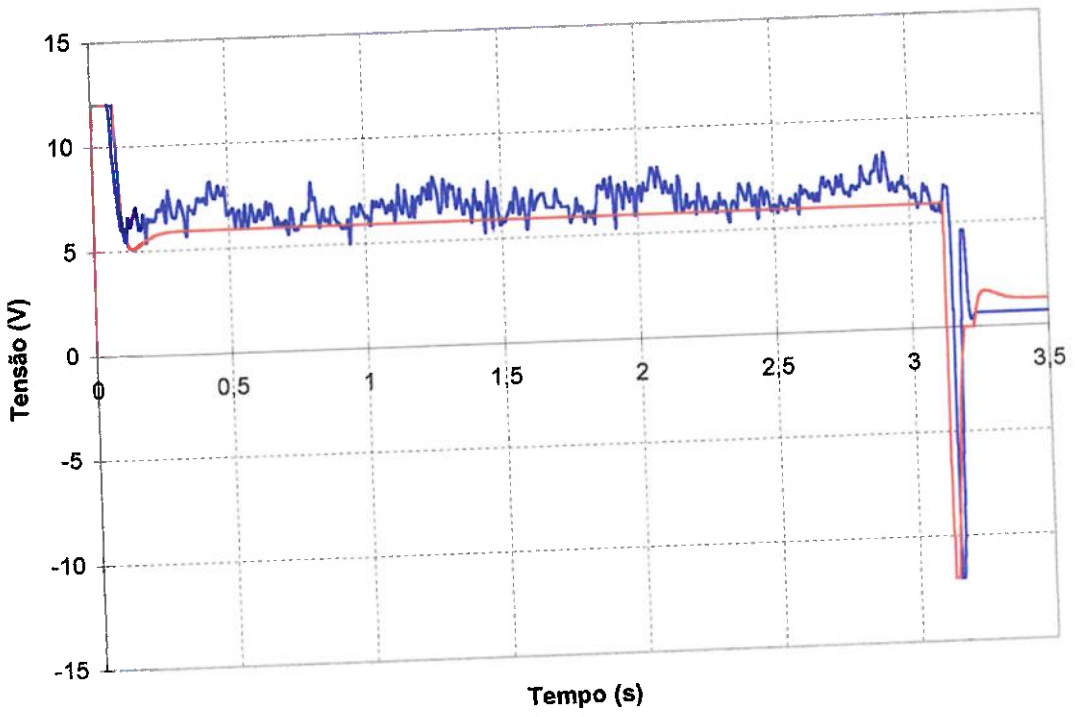


Figura 24 - Gráfico de tensão observado (7,5 ms)

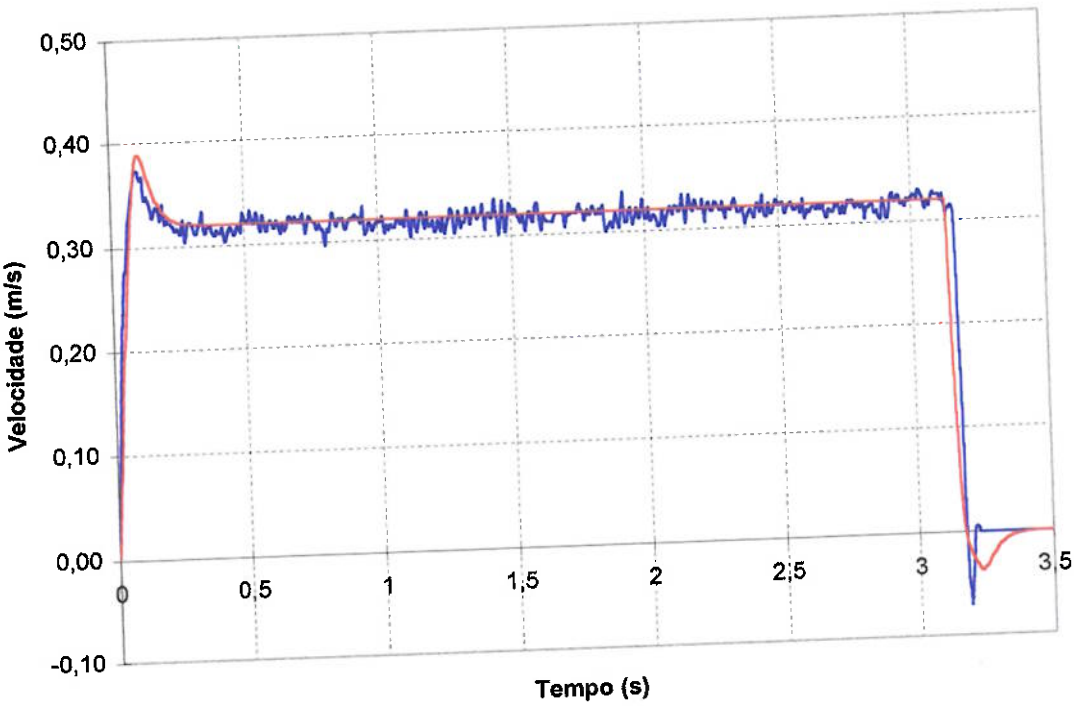


Figura 25 - Gráfico de velocidade observado (7,5 ms)

Também foi implementado um algorítmo de aceleração, as respostas obtidas para T_a = 7,5 ms com aceleração foram as seguintes :

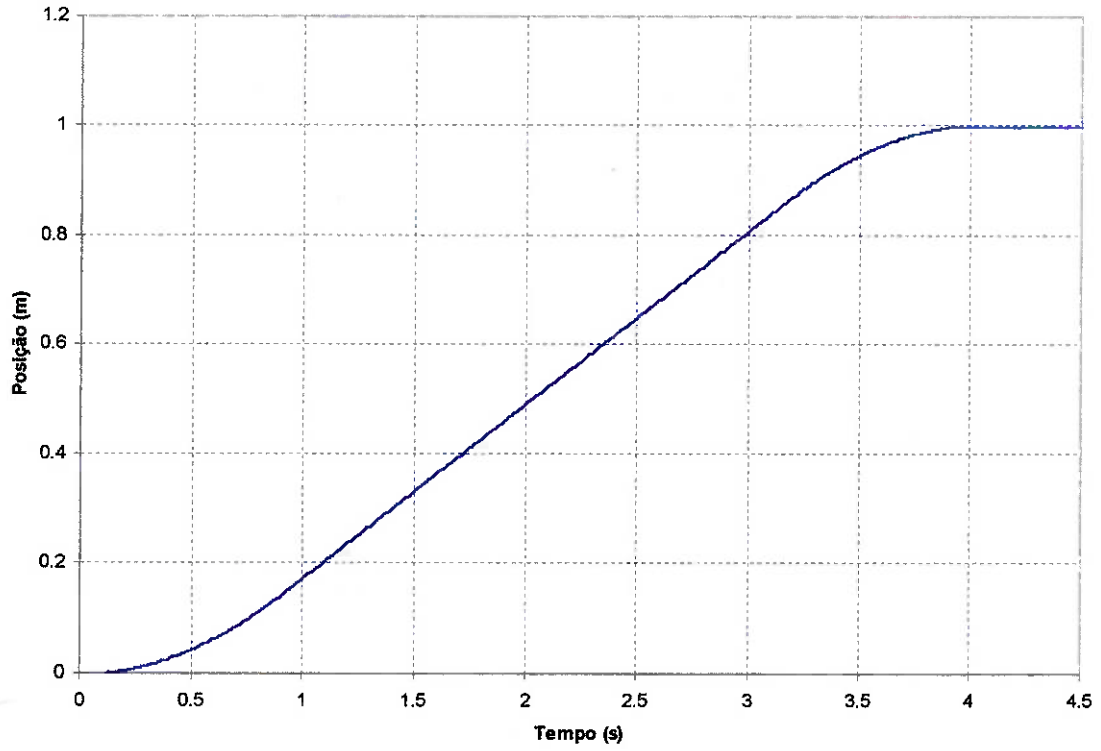


Figura 26 - Gráfico de posição observado (7,5 ms – com aceleração)

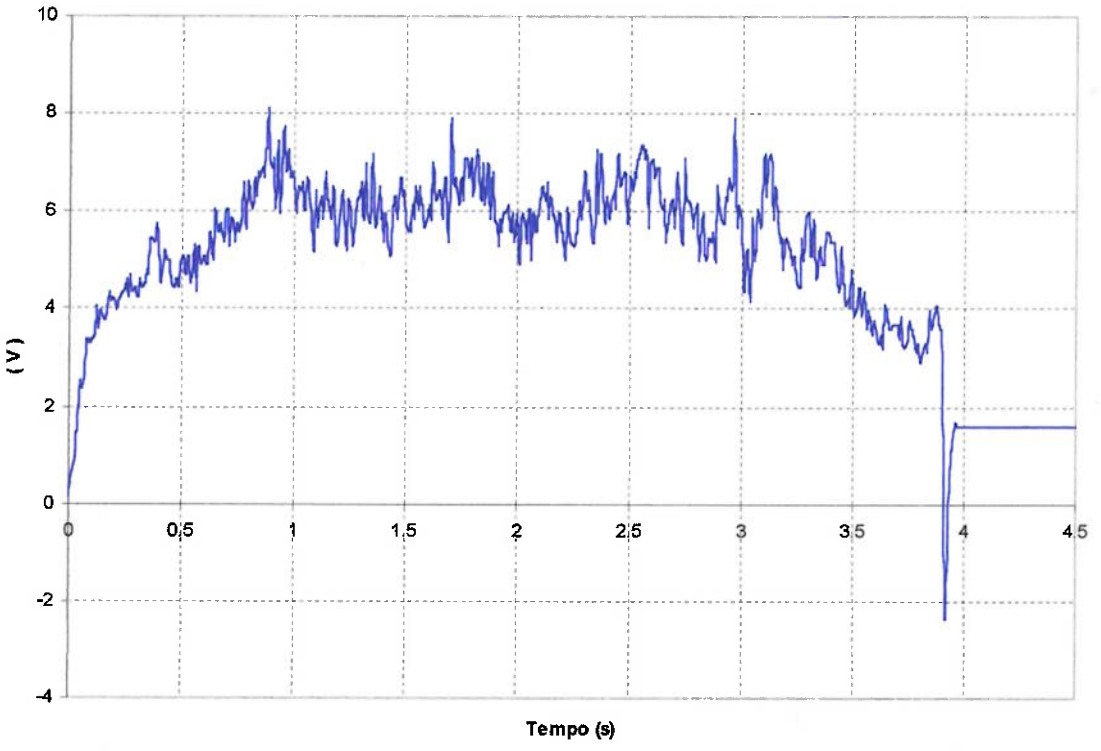


Figura 27 - Gráfico de tensão observado (7,5 ms – com aceleração)

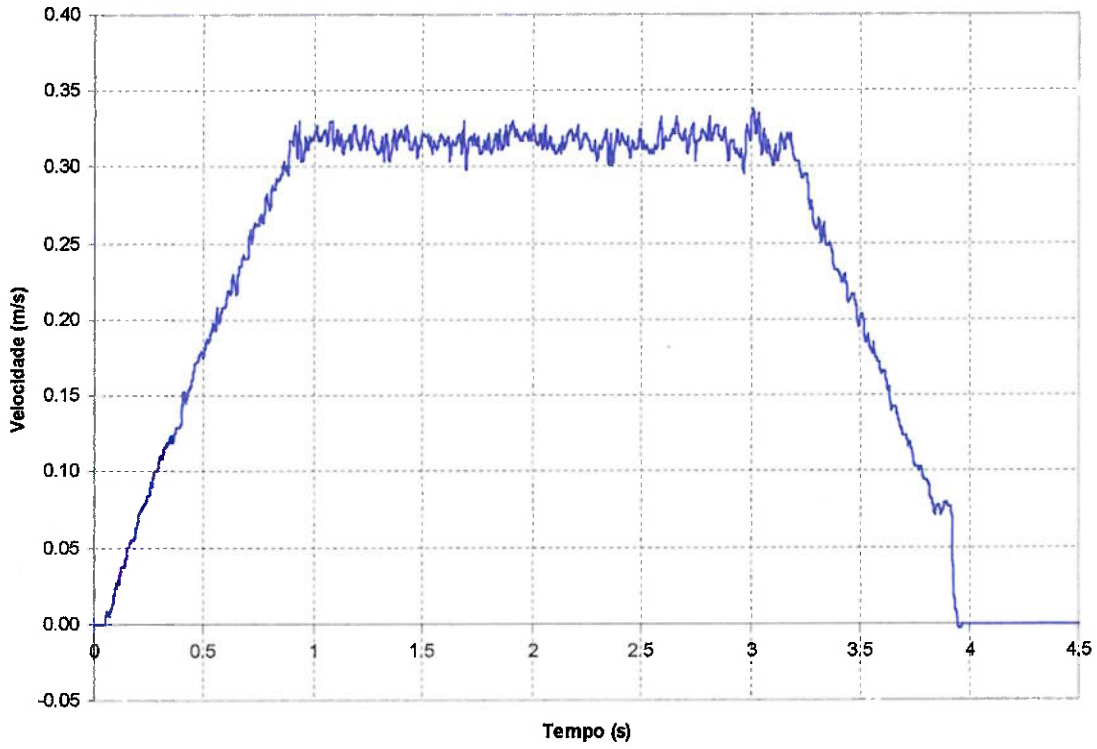


Figura 28 - Gráfico de velocidade observado (7,5 ms – com aceleração)

5.5 Controle de Velocidade

Foi projetado também um controlador de velocidade para ser usado no movimento por joystick. O controlador utilizado foi :

$$G_C(s) = K \cdot \frac{(T+s)}{s}$$

Com o auxílio do Matlab, projetou-se este controlador, chegando aos seguintes valores : T=10 e K=20.

Utilizou-se o modelo na figura 29 para simulação, e determinação dos valores de T e K.

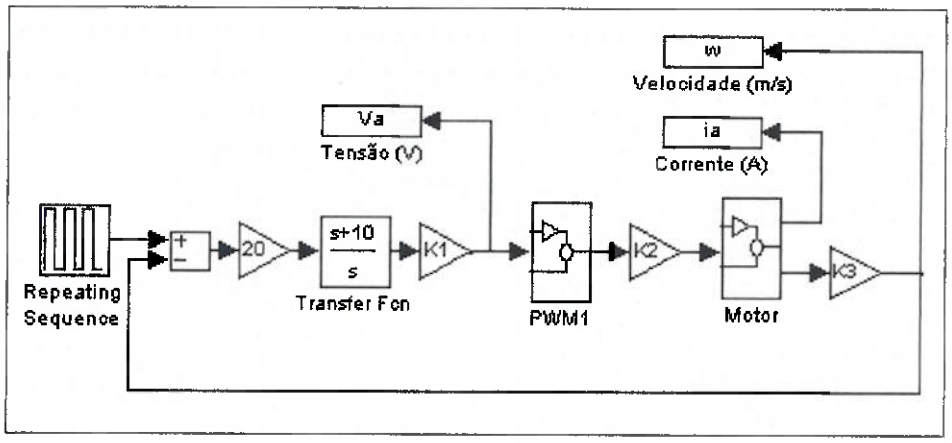


Figura 29 – Modelo usado para simular controle de velocidade

No modelo, K1 representa a transformação da saída do controlador que é um valor de tensão, em valor que represente um valor de PWM, ou seja que obedeça a relação de 12 V equivale à 128. Assim, é possível simular a saturação de tensão e outras características do PWM. K2 = 1/K1, e K3 é a conversão de velocidade do motor em rad/s para velocidade do robô em m/s.

Os resultados obtidos da simulação foram os seguintes :

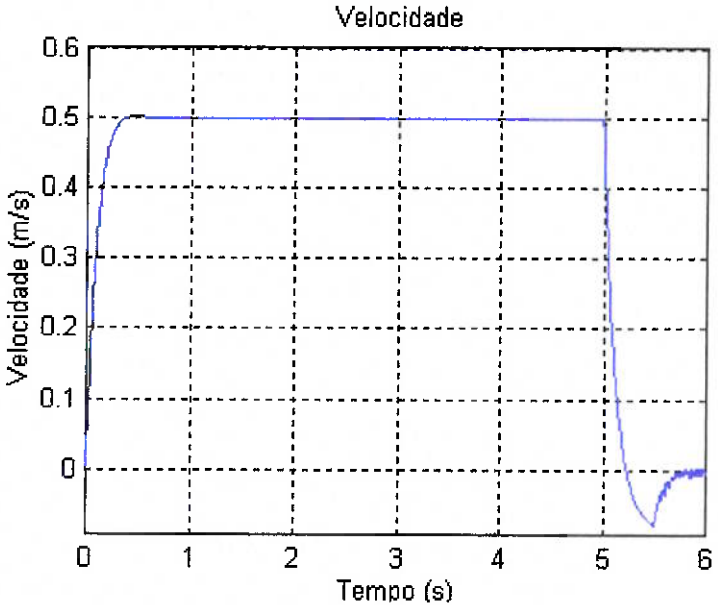


Figura 30 - Gráfico de velocidade

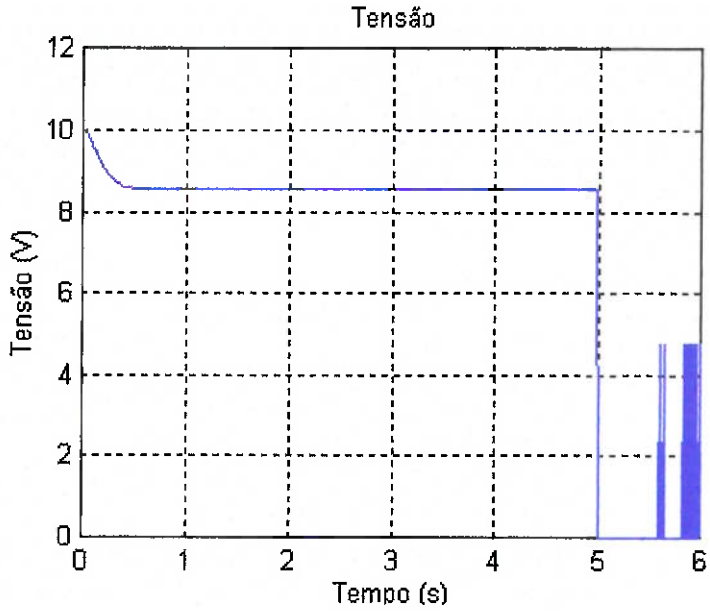


Figura 31 - Gráfico de tensão

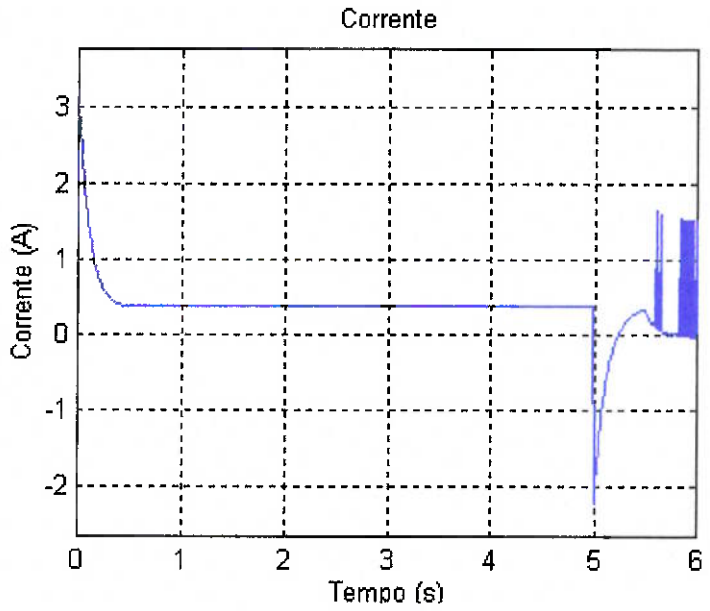


Figura 32 - Gráfico de Corrente

Os resultados obtidos foram satisfatórios, sendo que a planta possui um tempo de resposta bom para uma entrada degrau.

O controlador obtido foi discretizado pelo método de tranformação bilinear, e chegou-se a seguinte equação de diferenças :

$$y_{atual} = y_{anterior} + \frac{K}{2} \cdot (2 + T \cdot T_a) \cdot u_{atual} + \frac{K}{2} \cdot (T \cdot T_a - 2) \cdot u_{anterior}$$

No entanto, do modo como foi projetado, a saída é dada em tensão, e a entrada é dada em valor de velocidade em m/s. Por isso, deve-se transformar a saída em valor de

PWM, considerando o ganho K1 = 10,67 e além disto, transformar a entrada em diferença de pulsos de encoder. Como já foi dito, 1 m/s equivale a $50465T_a$, sendo assim basta dividir a entrada por este valor. Fazendo estas duas correções, a expressão fica :

$$y_{atual} = y_{anterior} + \left[\frac{K}{2}\cdot(2 + T\cdot T_a)\cdot u_{atual} + \frac{K}{2}\cdot(T\cdot T_a - 2)\cdot u_{anterior}\right]\cdot\frac{10,67}{50465\cdot T_a}$$

Para $T_a = 5$ ms :

$$y_{atual} = y_{anterior} + 0,867\cdot u_{atual} - 0,825\cdot u_{anterior}$$

Para $T_a = 7,5$ ms :

$$y_{atual} = y_{anterior} + 0,585\cdot u_{atual} - 0,545\cdot u_{anterior}$$

6. Implementação de Comandos

Como já foi dito anteriormente, já existe um software com um sistema de controle em tempo real que possui rotinas básicas. Este software já se encontrava pronto anteriormente ao início deste trabalho, e os comandos e a linguagem de programação definidas neste projeto serão implementados aproveitando a estrutura e as rotinas básicas que este software possui, fazendo-se as adaptações e melhorias necessárias.

A linguagem de programação que foi utilizada é o C, e o assembler do microcontrolador da Base Móvel, ou seja, assembler da família 51 de microcontroladores da Intel. Foi utilizado um software denominado "Dunfield Development System" para compilar o programa em C gerando o código em assembler do microcontrolador, e assim ser possível a gravação do software de controle em EPROM.

A seguir serão abordados alguns aspectos importantes sobre a implemtação do sistema de controle, comandos, e funções necessários.

6.1 Sistema de Controle em Tempo Real

O software de controle da Base Móvel existente, já possuia característica de tempo real. No entanto, o sistema sofreu algumas melhorias, sendo que a sua estrutura final para execução dos comandos pode ser vista na figura 33.

O sistema pode ser dividido em duas partes principais : o programa principal, e a rotina de tempo real.

No programa principal, ocorre a inicialização do sistema, e então o programa

entra em um loop. Neste loop ocorre o reconhecimento de caracteres e de comandos, caso um comando válido seja entrado pelo usuário, o programa realiza a execução do comando teclado, este por sua vez pode ser do tipo comando contínuo (exemplo : Mostra continuamente a posição do robô), ou de execução imediata (exemplo : mostra a posicao atual do robô).

Caso o comando seja do tipo contínuo, ocorre apenas a habilitação do comando, que por sua vez é executado de tempos em tempos na área do programa que se destina a execução de comandos contínuos habilitados. Estes comandos contínuos não podem fazer com que o fluxo do programa pare enquanto ele está sendo executado, pois pode ocorrer que mais de um comando contínuo esteja acionado. Por isso existe a necessidade de executá-los somente uma vez, caso habilitados, todas as vezes que ao percorrer o loop principal se chegar a posição de execução de comandos contínuos habilitados. Por exemplo : Ao invéz de fazer um loop que mostrasse continuamente a posição do robô, o que se faz é mostrar a posição atual do robô toda vez que o programa passar pela área de execução de comando contínuo, e encontrar o comando contínuo que mostra posição do robô habilitado.

Quando um comando contínuo (SHOW CPOSITION) é entrado pelo usuário, a habilitação deste comando ocorre ao se colocar o código deste comando em uma pilha de comandos contínuos habilitados. Quando o comando é cancelado, o que ocorre pressionando o crtl-x, o código é retirado da pilha.

Da mesma forma que existe o comando contínuo, também se implementou os mdos de execução, cujo taratamento se encotra na mesma área e do mesmo modo que os comandos contínuos. No entanto estes modos de execução são habilitados por flags, e não podem ser cancelados com o ctrl-x. Como exemplo desta ultima categoria está o MOVETAB, o RUN, a verificação se um comando MOVE, MOVEREL, ou MOVEABS já foi executado.

O bloco de reconhecimento de comandos faz a verificação se algum caracter foi teclado, e se foi, ocorre a formação de uma linha de comando. Caso esta linha seja completada, pelo pressionamento de um "return" então o comando é reconhecido segundo a sintaxe estabelecida.

Caso não ocorra o pressionamento de nenhuma tecla, a execução do programa passa por este bloco sem executá-lo, e neste caso o fluxo cai na execução de comandos contínuos habilitados, já que não há comando imediato para ser executado, e nem chamada de comando contínuo para habilitar flags. Depois da execução dos comandos

contínuos habilitados, o programa volta para o reconhecedor de comandos.

Paralelo ao programa principal, ocorre a execução da rotina de tempo real. Esta rotina têm início a cada período fixo de tempo. Assim têm-se o controle preciso do tempo em que ela está sendo executada. Nesta rotina ocorre a leitura dos sensores como encoder e joystick, ocorre o controle de posição e velocidade quando habilitados, ocorre o cálculo de posição atual e velocidade atual, e também ocorre o incremento de contadores de interrupção. Estes contadores possuem a utilidade de fazer com que o programa principal tenha uma noção de quanto tempo se passou. Através deles pode-se ter um controle de tempo grosseiro no programa principal. As variáveis de posição, velocidade, joystick e encoder ficam disponíveis para uso no programa principal. Nesta rotina de tempo real também ocorre a captura de pontos para formar a tabela.

Ambas as partes do sistema de controle pode atuar no robô móvel de alguma forma.

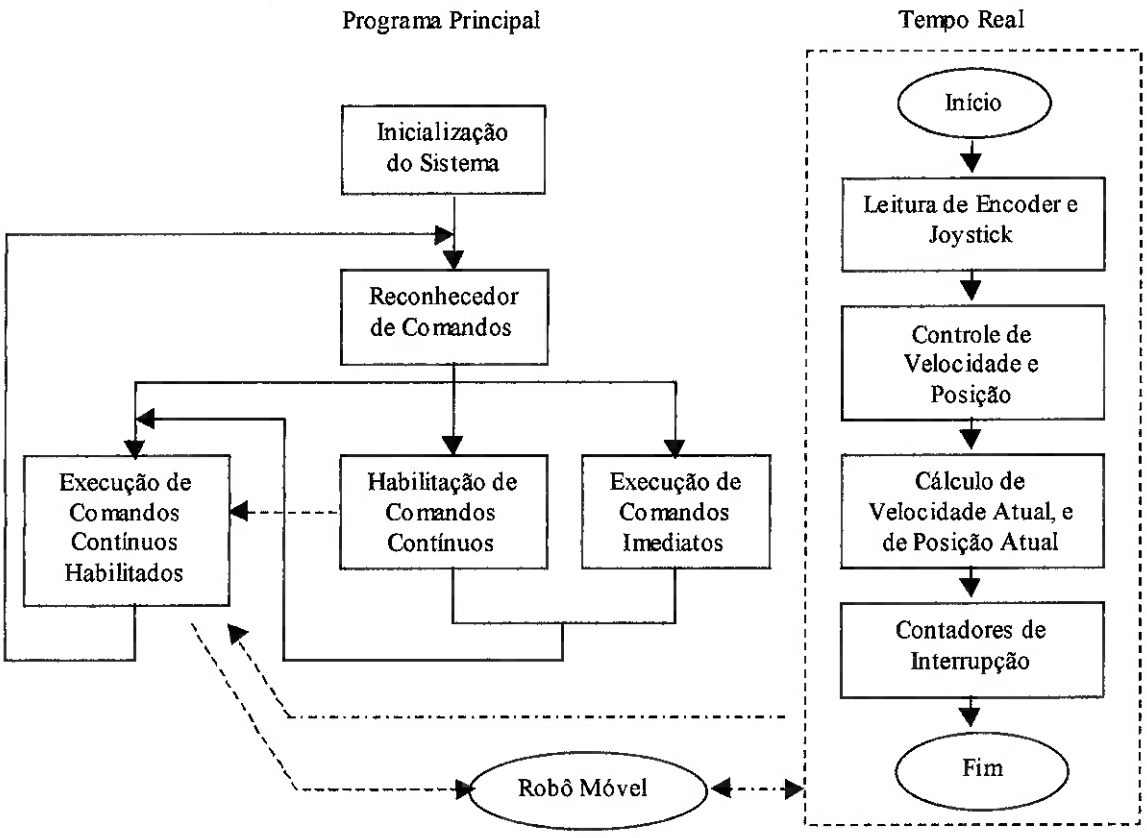


Figura 33 – Estrutura de Sistema de Tempo Real

6.2 Rotinas de Interface com o Usuário

Como já foi dito, muitos dos elementos de interface com o usuário, já haviam sido implementados como prompt, backspace, cabeçalho, algumas rotinas de impressão, conversão de caracteres ASCII para hexadecimais e vice-versa, controle da transmissão e recepção de caracteres pela serial, etc. No entanto o sistema não apresentava todos os componentes de interface necessários, principalmente a rotina de reconhecimento de comandos, já que só neste trabalho foi definida uma sintaxe de comandos, e de programação mais elaborada que deve ser reconhecida e interpretada pelo sistema. Também foi necessário desenvolver algumas rotinas de conversão de caracteres, e de impressão que não havia no sistema antigo, já que este trabalhava somente com impressões em hexadecimal. A seguir será explicado sobre as rotinas de interface que foi necessário desenvolver.

6.2.1 Reconhecimento de Comandos

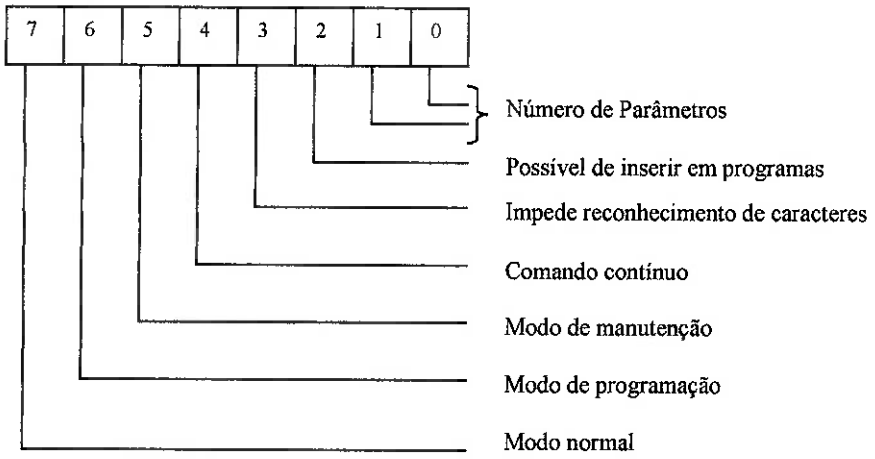
Como visto no ítem anterior, a função de reconhecimento dos comandos depende da sua sintaxe. O sistema deve possuir informações sobre os comandos para saber se o modo de execução atual é compatível com o comando, e para isso no desenvolvimento da rotina de reconhecimento de comandos foi proposta uma estrutura que guarda no sistema de controle, todos os comandos definidos, bem como informações sobre estes comandos para que se possa saber se o comando entrado é compatível com o modo atual, quantos parâmetros ele possui, se é comando contínuo ou de execução imediata, enfim, poder ter maior controle sobre o sistema e ser possível de identificar para o sistema, o comando teclado.

A estrutura proposta é formada de dois campos : uma string que guarda o nome do comando, e um campo de bits que guarda informações sobre o comando. Para guardar todos os comandos disponíveis, faz-se um vetor cujos elementos são definidos como sendo do tipo desta estrutura. Assim, pode-se ter cada comando associado ao índice do elemento no vetor onde ele foi definido.

A estrutura proposta é a seguinte :

```
struct comdef {
        char *name;
        char typebits; };
```

Onde 'typebits' é o campo de bits que contém as informações sobre o comando, e cada bit possui o seguinte significado :



Um exemplo de como esta estrutura pode ser :

```
struct comdef Comandos[2];
Comandos[0].name = "SET MODE";
Comandos[0].type = 0xe1;  /* Esta função pode ser usada em todos os níveis, é
                             de execução imediata, e possui 1 parâmetro */
Comandos[1].name = "CENC";
Comandos[1].type = 0x38; /* Esta função funciona apenas no modo de
                    manutenção, é de execução contínua, e não possui parâmetros,
                    e impede reconhecimento de caracteres que não sejam ctrl-x e ctrl-z */
```

A função de reconhecimento de comando implementada pega o buffer de linha de comando, e separa o nome do comando dos parâmetros. Então o nome de comando digitado é comparado com os nomes dos comandos existentes. A partir desta comparação, identifica-se o comando digitado, e uma variável recebe o código deste comando (índice no vetor de comandos). Também são reconhecidos os parâmetros, transformando a string de parâmetros em valores numéricos. Com o código do comando, pode-se executá-lo no loop da rotina principal (ver diagrama da rotina do sistema de controle em tempo real), sendo que ele pode ser executado imediatamente se não for contínuo, ou pode ocorrer a habilitação dele, caso seja um comando contínuo.

Esta rotina de reconhecimento de comandos altera a variável global usda no loop principal denominada "comando" que guarda o código do comando que deve ser executado, a função de reconhecimento também altera um vetor denominado "parametros" para guardar os valores dos parâmetros do comando reconhecidos que deve ser executado. Estas duas variáveis são responsáveis pela execução dos comandos.

Na mesma rotina de reconhecimento de caracteres pode ocorre a inserção de linhas em um programa. Esta inserção ocorre quando o sistema está no modo de programação e o primeiro caracter digitado é um número, que corresponde a linha do programa. Quando as condições anteriores são satisfeitas, esta rotina insere a linha digitada no programa, e não retorna o código do comando para o programa principal (variavel "comando"), já que não é desejado que se execute imediatamente a linha de programa digitada.

Para armazenar um programa na memória, optou-se por uma lista ligada onde os elementos representam cada linha do programa. Maiores informações serão dadas na seção de programação.

6.2.2 Manipulação de Valores ASCII, Hexadecimal, e Decimal

Foram implementadas rotinas auxiliares no sistema de controle que fazem a conversão de caracteres ASCII para valores hexadecimais e vice-versa. A conversão de uma string em ASCII para valor hexadecimal é utilizada no reconhecimento de parâmetros entrados em hexadecimal no modo de manutênção. O inverso , ou seja, conversão de valores hexadecimais em ASCII que pssa os representar é usado para a impressão de valores de encoder e joystick também no modo de manutenção.

Também foram implementadas rotinas que convertem valores (em hexadecimal) para caracteres ASCII que representem o valor convertido em decimal. Existem duas rotinas que foram implemtadas com esta finalidade, uma delas converte e imprime na tela, a outra guarda o resultado em uma string que pode ser usada para outras finalidades.

No modo normal e de programação, foi usado para reconhecer parâmetros digitados na forma decimal, uma função já existente na bibliotéca do compilador que converte uma string de caracteres ASCII em um valor inteiro (atoi). Nestes modos, são usados para a impressão de alguns valores, a função que converte hexadecimal em ASCII representando números decimais citada acima.

6.2.3 Modos de Operação - SET MODE <0,1,2>

Para implementação do comando SET MODE foi preciso de uma variável global que pudesse guardar o valor do modo de operação atual, ou seja, que indicasse através dos seus três últimos bits (da mesma forma que é indicado na variável de tipo de

comando) qual dos modos está ativado. Guardando o modo de execução atual desta forma, pode-se fazer uma simples operação lógica de AND entre a variável que indica o modo atual, e a variável que indica o modo que o comando digitado pode operar, e assim saber se o comando entrado pode ser executado ou não.

Para diferenciar na interface os três modos de operação, optou-se por mostrar um tipo de prompt diferente para cada modo, sendo assim exeiste um variável que guarda o prompt atual, e esta variável é alterada quando se muda de modo.

Esta função também desliga o controle de posição e velocidade quando se entra em modo de manutaenção e modo de programação, e religa o controle quando o modo é normal. Devido a esta funcionalidade, o comando não pode permitir mudanção de modo quando um movimento que requer controle de posição ou velociadde está sendo executado.

6.2.4 Help (?)

Como parte de um dos comandos de interface, está disponível um help que depende do modo de operação atual. Basta pressionar "?" para visualizar os comandos que estão disponíveis no modo corrente.

6.2.5 Tecla Ctrl-X e Ctrl-Z

Esta combinação de teclas especiais, juntamente com a tecla de help (?) são tratadas de maneira diferente. Ou seja, elas não são armazenadas na linha de comando como as teclas numéricas e alfanuméricas, e elas possuem funções especiais. No caso do ctrl-x, ele serve para desabilitar comandos contínuos habilitados, e o ctrl-z serve para resetar o sistema do robô. Estas duas teclas sempre se mantém funcionando mesmo que o sistema esteja impedido de reconhecer caracteres pela execução de algum comando contínuo. Já a tecla "?", também não é colocada na linha de comando, e no instante do seu pressionamento, o help é mostrado.

6.3 Controle de Posição – Comandos de Movimento

O controlador de posição projetado, como visto anteriormente, obteve um bom resultado. Este sistema de controle de posição se mantém sempre acionado quando se está no modo normal. No modo de manutenção e no modo de programação, como já foi dito, o sistema de controle é desligado, sendo que só funciona enquanto um comando de movimento está sendo executado.

A seguir serão abordados alguns aspectos relevantes sobre o controle, sobre as variáveis e comandos envolvidos, e sobre os problemas enfrentados na implementação dele.

6.3.1 Aceleração

Para implementar a aceleração, dividiu-se basicamente em dois casos :

- O robô acelera, atinge a posição intermediária sem ter atingido a velocidade de entrada e começa adesacelerar;
- O robô acelera, atinge a velocidade definida, fica com velocidade constante, e desacelera;

Foi definido uma flag que indica quando foi atingido a posicao intermediária do movimento. Foi definido uma variável que corresponde ao quanto deve-se somar na referência de posição. Esta variável age como se fosse a velocidade pois a cada período de interrupção, é somado na referência de posição o valor desta variável. Esta velocidade começa em zero e vai sendo incrementada de uma parcela fixa.

Também foram definidas flags que dizem quando o robô deve desacelerar, quando está na primeira metade do movimento, quando está com velocidade constante, e um contador para indicar quanto o robô andou em velocidade constante até a posição intermediária.

O algorítimo funciona assim : A variável de velocidade começa em zero e vai sendo incrementada a cada periodo de amostragem. Se o robô atingiu a posição intermediária sem ter atingido a velocidade constante, comeca a desacelerar. Se o robô estiver acelerando e atingir a velocidade constante definida, seta a flag de velocidade constante e começa a incrementar o contador para saber se quanto deslocou até a metade com velocidade constante. Atingindo a posição intemediária, começa a decrementar o contador até chegar ao zero. Então pode desacelerar.

Com este agorítmo implementou-se a aceleração do robô, e os resultados podem ser vistos nos gráficos apresentados na seção de controle de posição.

6.3.2 Cálculo de Posição Absoluta Atual

Para determinação da posição absoluta atual do robô, optou-se por fazer um sistema que fosse possível de determinar a posição de maneira rápida, sem perder muita precisão, e que pudesse calcular a posição sempre que houvesse movimento do robô, seja este movimento provocado por joystick, por atualção direta no motor, ou por

comandos como MOVE, ou MOVEABS.

Sendo assim, foi implementado na rotina de tempo real um algorítmo que calcula a variação da leitura do encoder entre duas interrupções (períodos de amostragem onde se entra na rotina de tempo real) consecutivas. Caso esta variação for diferente de zero, o sistema lê a posição angular das rodas, e atualiza em variáveis de 32 bits que guardam as posições absolutas, o quanto o robô se deslocou em X e em Y.

As posições absolutas são guardadas nestas variáveis de 32 bits, e estão em unidade de pulsos de encoder. Isto é feito para que não se perca tempo e precisão na conversão de unidades, já que a leitura natural de posição é feita em pulsos de encoder. Desta maneira, as conversões para milímetros são feitas no programa principal quando se deseja mostrar ao usuário, ou fazer alguma operação interna com os valores em milímetros.

Assim resolveu-se o problema do calculo da posição absoluta atual.

6.3.3 Comando MOVE, MOVEREL, e MOVEABS

A rotina de comando de movimento mais simples que se utiliza do controle de posição é a do comando MOVE. Este comando faz com que o robô se translade de uma distância dada pelo parâmetro deste comando. Nesta rotina, a distância entrada como parâmetro é convertida de milímetros para número de pulsos de encoder, e então guardada em uma variável que indica a posição relativa final em pulsos de encoder. O valor do encoder no início do movimento também é guardado em outra variável. Uma referência de posição é gerada a cada interrupção, com base na porcentagem de velocidade máxima que foi entrada como parâmetro.

Sabendo-se o valor do encoder no início do movimento, o valor do encoder no instante atual, o valor da referência, e o valor da posição relativa final em pulsos de encoder, pode-se calcular o erro do controlador e assim fazer com que o sistema de controle de posição funcione corretamente.

Para este comando, ainda se implementou uma flag que indica que o robô está em movimento e não chegou à sua posição final. A flag é setada sempre que se inicia o deslocamento, e é resetada quando o robô atinge a posição final desejada (com uma determinada precisão), e para de se mover. Esta flag é muito útil quando este comando é usado como um comando de programa, já que ela indica quando se pode comecar a execução de um novo comando do programa, ela também é util pois indica ao sistema que o robô está se movendo e o usuário não pode executar outro movimento enquanto

este não acabar.

Para implementar o comando MOVEABS, utilizou-se as variáveis que indicam a posição absoluta atual do robô, e também uma variável que indica a direção das rodas. Assim, quando uma coordenada é entrada como parâmetro do comando, e o sistema de coordenadas é o cartesiano, determina-se a partir da coordenada absoluta atual, qual é o deslocamento em X e em Y que o robô deve fazer. A partir destes valores chega-se a um valor de ângulo que as rodas devem rotacionar, e um valor de distância que o robô deve se deslocar. Após, usa-se duas funções básicas de movimento que são ROTATE, e MOVE. Quando a coordenada de entrada está no sistema polar, esta coordenada é transformada para cartesiana, e então realiza-se o procedimento descrito acima. Da forma com que foi implementado, o robô realiza a rotação das rodas de ângulos entre -180 e +180, faz o deslocamento sempre na direção positiva, no entanto com uma pequena alteração é possível fazer uma rotação somente entre –90 e +90 e então ajustar a direção do movimento, que seria negativa em alguns casos. Todos os cálculos de posição já estão preparados para este tipo de movimento.

O comando MOVEREL é simples já que basta pegar o ângulo e a distância fornecidos como parâmetros para o comando quando o sistema de coordenadas é o polar, ou determinar o ângulo e o deslocamento que o robô deve fazer a partir do deslocamento em X e em Y passados como parâmetros quando o sistema de coordenadas é cartesianos.

É importante observar que os parâmetros de entrada são inteiros em milimetros, portanto não é possível movimentos superiores a 30 metros. Também ocorre esta limitação quando se mostra a posição atual, não consegue-se mostrar posições maiores que 30 metros.

6.3.4 Funções Seno, Coseno, Arco Tangente, e Teorema de Pitágoras

É importante observar que os cálculos envolvidos nas operações de cáculo de posições descritas nos dois itens acima, usam funções como seno, cosseno, arco tangente e teorema de Pitágoras. Isto nos causa um problema que é o de operações com valores não inteiros, ou até mesmo menores que zero.

O microcontrolador, e até mesmo o compilador em C utilizado não possui funções que usam ponto variáveis de ponto flutuante, assim precisou-se trabalhar sempre com valores inteiros.

Para cálculo da hipotenusa usando os catetos, expandiu-se em série até um termo

que obtive-se uma precisão de cálculo razoável a expressão do teorema de pitágoras, já que não havia disponível a função raiz quadrada. Trabalhou-se nestas funções, com valores de entrada e saída em inteiros de 16 bits, no entanto no interior da função utilizou-se variáveis de 32 bits, já que os valores precisavam ser elevados a potencia de ordem dois ou três.

As funções seno, cosseno foram implementadas usando uma tabela de coseno, onde o índice da tabela representa o angulo em grau, e o valor na tabela é o cosseno multiplicado por um valor que é potência de 2, no caso implementado, multiplicou-se por 64. Usa-se potência de 2 para facilitar as operações de divisões pois assim elas podem ser feitas mais rapidamente. A tabela vai de 0 à 90 graus, variando de grau em grau.

Para implementação da função que retorna o angulo cujo o cateto oposto, cateto adjacente e a hipotenusa são conhecidos,função que se entitulou arco tangente, utilizou-se a tabela de cosenos para determinar o angulo. Calculou-se o coseno, ou o seno, dependendo de qual dos catetos era maior, multiplicou-se por 64, e procurou-se na tabela, qual era o índice que corresponde-se ao valor mais próximo, determinando assim o angulo.

Observe que em todas as contas envolvendo estas funções, mesmo procurando na sua implementação diminuir os erros de truncamento, este tipo de erro continua existindo.

6.3.5 Cálculo de Posição – Comando MOVE

Como pode ser visto anteriormente, no cálculo da posição atual absoluta a cada interrupção de tempo real, ocorre muitos erros de arredondamento, pois ela se utiliza de funções de seno e coseno para calcular o valor do incremento da posição em X e em Y, além do mais, este cálculo é feito com valores pequenos de pulsos de encoder (variação de pulsos entre uma interrupção e outra) e assim acumula os erros ao longo do seu movimento.

No entanto, quando se têm certeza de que o movimento será uma reta, o que só pode ser afirmado para os comandos MOVE, MOVEREL, e MOVEABS, então pode-se fazer o cálculo da posição baseado na posição absoluta inicial, e no deslocamento em pulsos de encoder durante o movimento. Este calculo de posição não impede que o outro calculo seja realizado, mas quando é executado um movimento do tipo citado, a posição atual é calculada com base em todo o movimento, e o valor da posição absoluta

final é atualizada ao término deste cálculo. Observe que não há duplicidade de variáveis de posição, e sim tudo é atualizado em uma única variável.

Este algorítmo faz com que o erro de truncamento diminua quando se têm certeza de que o movimento ocorreu em linha reta, sem variação do angulo das rodas.

6.4 Set de Parâmetros

Para guardar as velocidade de translação máxima e default, bem como as velocidades de rotação máxima e default foram usadas variáveis globais, que são ajustadas através dos comandos: SET TRANSMAXVEL, SET TRANSVEL, SET ROTMAXVEL, e SET ROTVEL respectivamente. O único cuidado com estas funções se dá em relação a não deixar que os valores entrados pelo usuário sejem maiores que os permitidos, e também não pode ser permitido setar a velocidade máxima quando o robô está em movimento.

Os comandos SET JOYSTICK e SET TABLE serão discutidos posteriormente.

O comando SET COORDENATE apenas altera o valor de uma flag que indica se o sistema está usando coordenadas cartesianas ou polares.

O comando SET MODE já foi discutido anteriormente.

6.5 Comandos de Exibição

Nesta categoria se encontra basicamente os comandos que permitem visualizar a posição, a velocidade e o valor doss parâmetros do sistema apresentados anteriormente.

Os comandos SHOW POSITION e SHOW CPOSITION convertem as variáveis de posição atual absoluta, de pulsos de encoder para milímetros, e imprimem na tela.

Os comandos SHOW VELOCITY e SHOW CVELOCITY fazem o mesmo com a variável que guarda a diferença na posição do encoder entre duas interrupções de tempo real.

As conversões são feitas seguindo a seguinte relação : 1m equivale a 50464 pulsos de encoder, e 1m/s equivale a uma variação de $50465T_a$, onde T_a é o período de amostrage em segundos.

Quando a função SHOW CVELOCITY ou SHOW CPOSITION é acionada, mostra-se continuamente a velocidade em valores decimais, e o sistema não reconhece nenhum caracter digitado exceto 'ctrl-x' (cancela ultimo comando contínuo ativado) e 'crtl-z' (reseta robô).

6.5 Movimento através do Joystick – SET JOYSTICK

O movimento do robô através do joystick é habilitado e desabilitado através do comando SET JOYSTICK. Este comando não é um comando contínuo, pois ele não pode ser cancelado por ctrl-x. Seu código não vai para a pilha de comandos contínuos sendo executados. No entanto, embora não seja um coando contínuo, seu funcionamento é semelhante, pois é setada uma flag que ativa este comando, e ele então é executado na mesma área de programa que os comandos contínuos são executados. Optou-se por fazer assim, para distinguir os comandos que podem ser cancelados por ctrl-x (comandos contínuos) e os que têm carater contínuo mas não podem ser cancelados, estes últimos habilitados por flags. Uma vantagem das flags é que elas são rápidas de serem comparadas, e este tipo de comando normalmente coloca restrições para a execução de outros comandos.

Quando o robô está neste modo, é acionado um controle de velocidade, e desligado o controle de posição, sendo que a referência para o controlador é obtida da seguinte forma : o valor atual do eixo do joystick é lido, e então através de uma tabela é pego o valor de referência, em porcentagem de velocidade máxima, para o controle de velocidade. Este valor de referência é proporcional ao eixo do joystick. Desta forma se obtém uma velocidade proporcional à posição do eixo do joystick.

Como o controle de posição é desligado, não é possível usar joystick quando se está usando este controle, ou seja, quando robô em movimento, quando executando programa, etc.

Para a rotação das rodas também é feito uma rotina semelhante, no entanto não se faz um controle de velocidade já que o motor de passo opera em malha aberta.

6.6 Tabela de Pontos

6.6.1 SET TABLE

Através do comando SET TABLE pode-se habilitar ou desabilitar o modo de captura de posições para a construção de uma tabela. Como feito para o SET JOYSTICK, utilizou-se uma flag que indicasse quando a captura está habilitada ou não. Sendo assim, este comando não é comando contínuo.

A captura dos pontos é feita sempre que se aperta o botão, e é preciso guardar o ponto no instante que foi apertado o botão, por isso colocou-se a rotina de captura de pontos na interrupção de tempo real. A captura dos pontos seria impossível se não

houvesse o cálculo da posição absoluta atual a cada período de amostragem, mesmo quando o comando sendo executado descreve uma linha reta (MOVE).

Quando SET TABLE é ligado, a posição absoluta atual em milímetros é calculada, e guardada em uma variável auxiliar. Quando o botão é pressionado, calcula-se a posição atual em milímetros, e armazena-se em dois vetores o deslocamento em X e em Y entre as duas posições. Então o indice do vetor é incrementado, e a posição absoluta guardada novamente para ser possível o calculo do deslocamento relativo no próximo click de botão. Deve-se tomar cuidado para que não se passe do limite físico de memória, estourando o espaço dos vetores de pontos.

Quando SET TABLE é desligado, guarda-se o indice do final da tabela, para que seja possível controlar sua posterior execução.

6.6.2 MOVETAB

Composta a tabela, esta pode ser reproduzida através do comando MOVETAB.

Como a tabela de pontos é formada por posições relativas, já que o importante é a trajetória realizada, e não a posição absoluta desta trajetória, para a execução da tabela basta realizar um comando semelhante ao MOVEREL para coordenadas cartesianas para cada conjunto de pontos X e Y capturados.

Quando MOVETAB é executado, aciona-se uma flag a semelhança de SET TABLE e de SET JOYSTICK, assim, não pode ser cancelado por ctrl-x. Entretanto, este comando poderia ser implementado de maneira diferente como um comando contínuo, e então poder ser cancelado.

Após ser setada aflag, o comando é executado na área de execução de comandos contínuos, sendo que é executado o movimento relativo do robô para cada par de pontos, não passando para o par seguinte enquanto o robô não executar todo o movimento. Ao final da tabela indicada pela variável de fim de tabela, a flag é resetada.

É importante observar que este comando pode ser usado dentro de um programa, e se usado for a, o usuário possui o prompt para realizar comandos que sejam permitidos enquanto se executa o MOVETAB.

6.6.3 LOADTAB e UPLOADTAB

Como os valores são armazenados em um espaço da memória, estes podem ser salvos em arquivo através do comando UPLOADTAB, enviando os pontos no formato especificado. Também podem ser carregados de arquivo através do comando LOAD. É importante observar que da forma que foi implementado, estes dois comandos não podem ser usados quando o robô estiver se movendo.

6.7 Programação

Para armazenar o programa na memória, optou-se por uma lista ligada onde os elementos representam as linhas do programa. Cada elemento constitui de uma estrutura que é formada por uma variável que guarda o valor da linha, outra variável que guarda o código do comando, e um vetor para armazenar os parâmetros do comando, e de um ponteiro para o próximo elemento. Fazendo a armazenagem desta forma, pode-se utilizar o algorítmo de reconhecimento de comandos descrito anteriormente para interpretar os comandos e armazená-los de forma interpretada na lista ligada que forma o programa. Para utilizar o mesmo algorítmo basta adicionar o reconhecimento do valor da linha de programa antes do nome do comando, e fazendo com que os valores interpretados sejam armazenados em um novo elemento da lista criado e que será inserido no programa posteriormente.

Neste algorítmo de inserção, verifica-se no tipo do comando reconhecido, se ele pode ser inserido como comando em um programa, se não puder, então não ocorre a inserção na lista ligada, e o elemento criado dinamicamente é apagado.

Para administrar a lista ligada, foram implementadas rotinas que inserem, buscam, e apagam elementos em uma lista ligada, sendo que a entrada destas rotinas é sempre o valor da linha, e a lista ligada se encontra sempre organizada por ordem crescente desse valor. Tomou-se o devido cuidado para nunca inserir um elemento na lista, se ela já possuir um elemento com o mesmo número de linha.

A estrutura que foi usada para a lista foi a seguinte :

```
struct ProgElement {
        int Linha;
        char Comando;
        int Parametros[3];
        struct ProgElemento *NextElemento };
```

onde :

Linha é o valor da linha de programa do elemento;

Comando é o código do comando desta linha;

Parametros guarda os parametros digitados para o comando;

NextElemento é o ponteiro para próximo elemento na lista.

É importante observar que esta estrutura foi escolhida por vários fatores. Um deles porque ela é semelhante à maneira com que qualquer comando é tratado, ou seja, encontrando-se o código correspondente ao comando, e guardando os parâmetros em variáveis que são utilizadas no programa principal quando a rotina do comando está sendo executada.

Outro fator está relacionado com o fato de que é necessário a execução do programa em tempo real, sendo que se tornaria complicado ter que interpretar o programa no momento de sua execução. Assim decidiu-se elaborar um linguagem que fosse possível de ser interpretada na medida que as linhas são inseridas, e que fosse compatível com a estrutura de comandos definida. Isto influenciou na escolha da estrutura da linguagem, e nos comandos de controle de execução, na medida que não poderiam ser usados estruturas de "if" e "while" parecidas com a de C e Pascal, já que este tipo de estrutura requer que após a edição do programa, se faça uma interpretação e compilação do programa para que se determine onde estão o fim do "while" e do "if". A solução para a estrutura de linguagem foi apresentada anteriormente, na seção onde se descreve a definição de comandos. Observe que os comandos de "if" definidos apresentam como um dos parâmetros, o número da linha para onde o programa deve ir caso a comparação esteja correta.

Portanto, a forma como se definiu a estrutura da linguagem, o método com que o programa é armazenado, e o interpretador de comandos, permite a execução do programa de maneira rápida e natural para o sistema de controle em tempo real apresentado.

6.7.2 RUN

Para implementar o comando RUN, que é responsável pela execução do programa, foi necessário definir um ponteiro que aponta para a linha do programa que deve ser executada ("pLinhaAtual"), e uma flag que indica que o programa está sendo executado. Uma vez iniciado a execução do programa, o tratamento do RUN é feito como se fosse um comando contínuo, desabilitando o reconhecimento de caracteres, entretanto do modo que foi implementado (por flag) ele não cancela a execução com o ctrl-x, mas pode ser feita uma alteração simples para que o comando RUN se torne um

comando contínuo. Sempre quando não há comando sendo executado, ou seja, já acabou a execução do comando anterior, a variável "comando" que guarda o código do comando que deve ser executado recebe o código do comando que se encontra na linha de programa apontada pelo ponteiro "pLinhaAtual"; a variável "parametros" também recebe os valores dos parâmetros da linha de programa apontada pelo mesmo ponteiro. Feito isto, o ponteiro "pLinhaAtual" é atualizado para o próximo ponteiro, e então o programa volta para o início do loop, passando a executar o comando com o código que acabou de ser dado para a variável "comando.

É interessante observar que o sistema sabe que um comando acabou de ser executado pois se este comando for de natureza imediata e não for de movimento, a execução dele acaba quando o programa chega na área de execução de comandos contínuos habilitados, que é onde o comando RUN se encontra. Caso o comomando seja de movimento, o comando RUN só manda executar o próximo comando quando o robô estiver parado, ou seja, quando acabar o movimento (flag que indica movimento do robô resetada). O RUN também não manda executar um novo comando caso a flag de MOVETAB estiver setada, isto impede que seja executado um novo comando ao término do movimento do primeiro ponto da tabela (já que o MOVETAB possui vários comandos MOVEREL implícitos). Com isso, resolveu-se o problema de execução do programa.

6.7.3 IFEQ, IFNE, IFGT, etc, e GOTO

Os comandos do tipo IFEQ, IFNE, IFGT, etc, foram de simples implementação, fazendo com que o valor de pLinhaAtual fosse alterado conforme o resultado da comparação. Usou-se nestas rotinas a função de busca na lista ligada.

O comando GOTO também altera o valor de pLinhaAtual como nos comandos anteriores, com exceção da comparação que não existe.

6.7.4 SET VAR, INC, e DEC

Os comandos de SETVAR, INC, e DEC também são rotinas simples que alteram o valor de elementos de um vetor usado para implementar os contadores internos do programa.

Com os comandos de controle de execução descritos neste ítem e nos ítens anteriores pôde-se definir a linguagem, e implementar a execução de programas no sistema de controle do robô.

6.7.5 CLEAR, LIST e EDIT

Na função CLEAR foi usada basicamente a função de apagar elemento da lista, é claro que primeiro é verificado se o elemento existe.

O comando LIST utiliza o comando de encontrar elemento na lista, e também um comando de imprimir comando na tela, nesta hora o código do comando é decodificado, eseu nome impresso.

Para o comando EDIT, utilizou-se de funções que convertessem valores inteiros em strings ASCII que representassem o valor em decimal, além disso foi necessário copiar esta string para o buffer de linha de comando. Depois que o comando foi jogado novamente para a linha de comando no formato de string, basta apagá-lo da lista, pois no caso do usuário fazer as mudanças necessárias e pressionar enter, sem ter apagado a linha da lista, o sistema tentará inserir um elemento com um número de linha igual a um existente. Por estes motivos, deve-se apagar o comando para que ele possa ser inserido novamente depois de editado.

6.7.6 Comandos de Espera

Para a implementação do comando WAIT TIME usou-se um contador de interrupções. Assim, o programa fica parado em um ponto até que o contador de interrupções atinja o valor correspondente ao tempo desejado.

O comando WAIT KEY, e WAIT BUTTON não apresentaram problemas na implementação, sendo que eles fazem com que o programa fique parado até que uma tecla ou um botão respectivamente seja apertado.

6.7.7 LOAD e UPLOAD

Os comandos LOAD e UPLOAD são parecidos com os comandos LOADTAB e UPLOADTAB, sendo assim, apresentam os mesmos problemas e os mesmos algorítmos.

6.8 Sugestões para Implementação Futura

Existem alguns comandos que podem ser modificados e outros comandos novos que podem ser implementados, sem grandes alterações no sistema. Alguns deles são :

- SET ACCELERATION <1,0> : Liga ou desliga a celeração do robô;
- Pode ser implementado um limite para o movimento do robô, de modo que não permita que ele atinja a fronteira de 30 metros.

- Pode-se fazer uma versão do MOVETAB para trabalhar com tabela em coordenadas polares, o que facilitaria a execução da tabela, pois os parâmetros seriam deslocamento e angulo de rotação. Não foi feito isso na captura de pontos, pois usaria as funções de arco tangente e a função de Teorema de Pitágoras que apresentam dentre as funções maiores erros devido arredondamento, no entanto poderia ser feito e realizado testes para verificar a precisão.
- SHOW POSITION pode ser implementado para mostrar tanto coordenadas cartesianas quanto coordenadas polares. Usa-se também as funções de arco tangente e o teorema de Pitágoras.
- Na implementação das rotinas de movimento (MOVE, MOVEABS, MOVEREL) o robô realiza a rotação das rodas de angulos entre –180 e +180, fazendo o deslocamento sempre na direção positiva. Poderia ser feito uma rotação somente entre –90 e +90 e então haveria o ajuste da direção do movimento, que seria negativa em alguns casos. Todos os cálculos de posição já estão preparados para este tipo de movimento.

Estas são algumas das modificações simples de serem feitas e que acrescentariam alguns detalhes de funcionalidade ao sistema.

7. Conclusão

Foi possível desenvolver e implementar um software para sistema de controle de movimento em tempo real de um robô móvel, sendo que para isso foram definidos comandos de alto nível suportados pelo sistema. Todos os comandos de alto nível propostos puderam ser implementados.

Desenvolveu-se uma linguagem de programação adequada para o sistema de controle real do robô, de forma que é possível usá-la para escrever programas que descrevam trajetórias específicas.

Implementou-se com sucesso o controle de posição do robô, que é usado nos comandos de movimento. Também foi implementado rotinas que permitem o movimento do robô através do joystick, sendo que é possível criar uma tabela de posições à medida que o usuário movimenta o robô e aperta o botão do joystick. Esta tabela pode ser reproduzida posteriormente.

O controle de posição apresentou um bom resultado, sendo que possuindo as equações de diferença em função do período de amostragem, fica fácil recalcular as constantes das eqações caso seja preciso alterar o período de amostragem.

O período de amostragem atualmente implementado foi o de 7,5 ms que apresentou um resultado muito próximo ao de 5 ms, não perdendo muito desempenho do controlador.

Portanto, com o software implementado obteve-se um sistema com uma linguagem de programação no qual é possível fazer com que o robô se movimente através de trajetórias definidas pelo usuário, bastando usar os comandos implementados na programação.

8. Referências Bibliográficas

[1] Kuo, Benjamin C., *Automatic Control System*, 7th ed., New Jersey, Prentice Hall, 1995

[2] Jones, Joseph L.; Flynn Anita M.; *Mobile Robots - Inspiration to Implementation*, Wellesley, A. K. Peters, 1993

[3] Tashibana, Luís S.; Fujiwara, Renato; *Trabalho de Formatura : Projeto Base Móvel - 2a. Fase de Desenvolvimento do Hardware Eletrônico*, Departamento de Eng. Mecânica, EPUSP, 1995.

[4] Mak, Ronald; Writing Compilers & Interpreters - an Applied Approach, New York, John Wiley & Sons, Inc, 1991.