Rodrigo Fernandes Vernini

Desenvolvimento do Controle de Velocidade e Posição para um Robô Móvel

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

Curso de Engenharia Elétrica com ênfase em
Sistemas de Energia e Automação

ORIENTADOR: Prof. Valdir Grassi Júnior

São Carlos
2010

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

Vernini, Rodrigo Fernandes
V537d Desenvolvimento do controle de velocidade e posição para um robô móvel / Rodrigo Fernandes Vernini ; orientador Valdir Grassi Júnior. -- São Carlos, 2010.

Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2010.

1. Robôs. 2. Controladores digitais. 3. Avanço de fase. I. Título.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Neste trabalho de conclusão de curso foi desenvolvido um controlador por avanço de fase para um robô móvel de pequeno porte que possui o módulo RCM4500W RabbitCore e um microprocessador Rabbit 4000. A programação, realizada através do software Dynamic C, permite o controle de velocidade e posição em malha fechada de dois motores, cada um responsável pelo acionamento de uma das duas rodas do robô. O modelo do motor foi identificado a partir de pontos gerados pelos encoders acoplados nas rodas do robô em resposta a uma entrada degrau de tensão aplicada nos motores. O acionamento de cada motor é realizado através de sinais de PWM e ponte H. Simulações do modelo e do controlador foram realizadas e os resultados obtidos com os controladores de velocidade e posição foram satisfatórios.

Palavras-chaves: robô, avanço de fase, controlador, RabbitCore, Dynamic C.

Abstract

In this completion of course work was developed a lead-phase controller software for a small mobile robot that has the RCM4500W RabbitCore module and a microprocessor Rabbit 4000. The program, made through Dynamic C software, allows you to control speed and position closed loop of two motors, each driving one of the two wheels of the robot. The engine model was identified from points generated by encoders coupled to the wheels of the robot in response to a input of a voltage step applied to the engines. The driving of each motor is done through PWM signals and an H-bridge. Simulations of the model and the controller were performed and the results obtained with de speed and the position controllers were satisfactory.

Keywords: robot, lead-phase, controller, RabbitCore, Dynamic C.

1. Introdução

Em 1939, no início da segunda guerra mundial, com avanços em pesquisas em novas áreas como ciência da computação e cibernética, surgiram os primeiros robôs móveis [1]. Assim como muitas tecnologias criadas pelo exército americano como o primeiro computador, o ENIAC, e as primeiras redes de computadores, que utilizavam protocolo TPC/IP, foram disponibilizados para os civis e assim criados diversos outros tipos de aplicações.

O robô móvel, em contraste com robôs industriais que possuem uma base fixa, tem a capacidade de se mover no ambiente para qual foi projetado. Essa liberdade gera uma enorme gama de possibilidades para as funcionalidades desses robôs. Através de sensores que podem ser adicionados em sua arquitetura são capazes de interagir com diversos objetos, dispositivos elétricos ou mecânicos, e até com pessoas em seu ambiente de aplicação.

No Laboratório de Sistemas Inteligentes, LASI, do campus da USP em São Carlos foi realizado, no ano de 2009, a construção do hardware de um robô móvel, e em 2010, apresentado nesse trabalho, foi realizado o controle e a programação desse robô. Este robô é composto pelo módulo RCM4500W RabbitCore, que possui o microprocessador Rabbit 4000, além de portas I/O configuráveis, conversores AD, timers, e diversos outros dispositivos que serão abordados nos próximos capítulos. O robô possui também dois motores ativados por PWM cada um conectado a uma roda de 5,5 cm de diâmetro, quatro sonares, dois sensores infravermelhos e dois encoders. O módulo também permite fazer a comunicação sem fio embarcada através de um modem ZigBee/802.15.4.

Os sonares e sensores infravermelhos foram adicionados ao módulo para ser possível a navegação autônoma do robô. O sonar consegue detectar obstáculos a longas distancias, porém possui baixa precisão, já os sensores infravermelhos complementam os sonares, pois apesar de possuírem pouco alcance, possuem alta precisão.

O encoder envia pulsos ao microcontrolador a partir da rotação da roda, e através de um decodificador de quadratura foi possível realizar a modelagem do motor do robô e seu controle de posição e velocidade.

Após a obtenção do modelo do motor, foi verificado através de simulações, uma resposta a rampa de posição muito oscilatória, e por isso foi decidido a implementação de um controlador por avanço de fase para aumentar a estabilidade da resposta. Esse controlador foi discretizado pelo método de transformação bilinear e implementado no código do programa.

O programa utilizado para escrever, compilar e depurar é o Dynamic C, usado em microprocessadores Rabbit, utiliza uma linguagem C estendida, com algumas funções adicionais para facilitar a programação do microcontrolador. É possível também a utilização da linguagem em assembly para uma resposta mais rápida. O fabricante fornece uma plataforma com um ambiente bem amigável e muitas funções para facilitar a depuração do programa.

Nos próximos tópicos será exposto com detalhes cada elemento do robô e os procedimentos para determinar o modelo completo do robô e seu controlador.

2. Robô móvel

O robô móvel é composto basicamente pelo módulo RCM4500W RabbitCore, sonares, sensores infravermelhos, indicador de carga na bateria, encoders e motores. Este módulo possui o microprocessador Rabbit 4000, que realiza desde a leitura dos sensores até o acionamento dos motores. O módulo também permite fazer a comunicação sem fio embarcada através de um modem ZigBee/802.15.4.

A seguir são apresentadas as principais características do hardware do robô: módulo RCM4500W, sonar, sensor infravermelho, encoder, motor PWM, ponte H, fornecimento de energia e estrutura mecânica.

2.1 Módulo RCM4500W RabbitCore

O módulo RCM4500W, Figura 1, possui todos os requerimentos para a implementação dos recursos desejados nesse projeto. Ele opera a uma tensão de 3,3 Volts, possui um microprocessador Rabbit 4000 de 29,2MHz, tem 40 terminais I/O configuráveis para diversas funções, 512K de memória de programa, mais 512K de RAM, 4 conversores analógico/digital, PWM, decodificador para quadratura incremental, supervisor watchdog, modem ZigBee/802.15.4 embarcado para comunicação sem fio, dentre algumas outras funções [2].

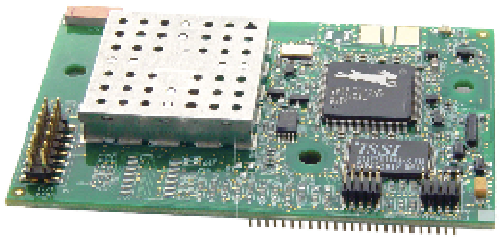

Figura 1 – Módulo RCM4500W

O hardware do robô foi montado com a ajuda do Kit de Desenvolvimento que acompanha o RCM4500W. No projeto os pinos mais importantes do módulo estão nos conectores J1 e J4, mostrados na Figura 2, dentre os pinos temos portas I/O que trabalham com tensão de 3,3V, e os conversores analógico/digital que trabalham com no máximo 1,2V.

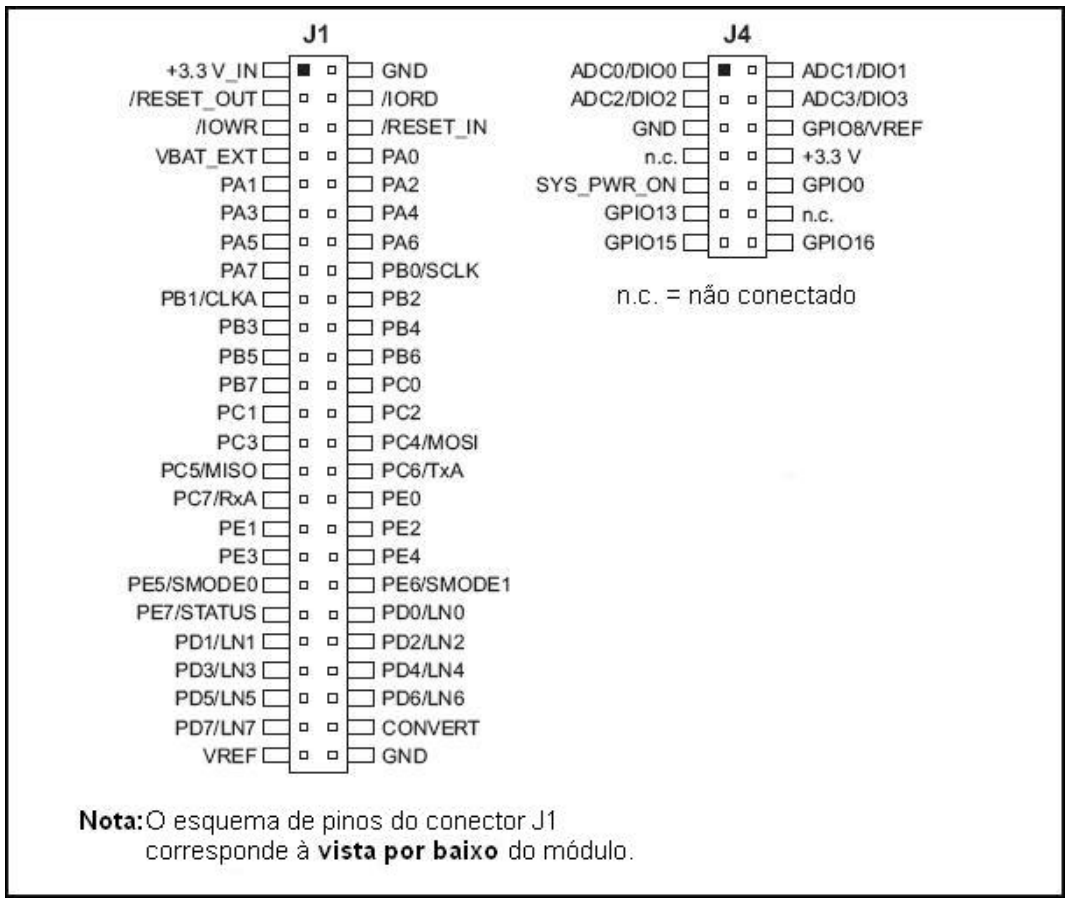

Figura 2 – Esquema de pinos do Módulo RCM4500W

Nos pinos também se pode notar a presença da alimentação e dos I/O que usam as nomenclaturas PA, PB, PC, PD e PE do conector J1, além dos conversores analógicos/digitais nomeados ADC nos pinos do conector J4. Algumas das funções que os pinos I/O podem exercer são de PWM, quadratura para encoder incremental e Input Capture [3].

No módulo também se tem o modem ZigBee/802.15.4 que permite a transferência de dados sem fio a uma velocidade de 250 Kbps com alcance de até 100m, com baixíssimo consumo de energia. Com esse módulo também é possível formar uma rede de conexões e conectar outros dispositivos que utilizam essa tecnologia, por exemplo, outros robôs móveis podem se intercomunicar, e realizar tarefas sincronizadas.

2.2 Sonar

No projeto foram utilizados 4 sensores de ultra-som Devantech SRF04, mostrado na Figura 3, comercializado pela Acroname Robotics. Este sonar é capaz de detectar objetos a distância. Os sensores estão localizados para otimizar o campo de detecção sendo 2 na parte frontal e 2 na parte traseira.

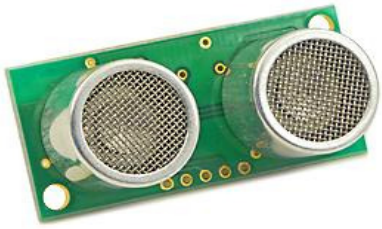

Figura 3 – Sonar Devantech SRF04

Este sensor opera a +5V e consome uma corrente típica de 30mA e máxima de 50mA. Os sinais ultrassônicos são emitidos numa freqüência de 40kHz. A Figura 4 mostra como devem ser feitas as ligações dos pinos.

Figura 4 – Pinos do sonar Devantech SRF04

O SRF04 é controlado por um impulso de nível lógico alto, que deve ter uma duração de no mínimo 10µs. Após detectar a borda de descida deste impulso de disparo, o sensor envia um sinal acústico ultrassônico. Em seguida, muda o sinal de saída para o nível lógico alto, ficando em

seguida à espera pelo sinal acústico do eco. Assim que o eco é detectado, o SRF04 coloca o sinal de saída novamente a nível lógico baixo. Caso nenhum objeto seja detectado, o sinal de saída é colocado a nível lógico baixo a aproximadamente 36ms após o sinal ultrassônico. É importante citar que o sensor somente estará apto a detectar o eco após um intervalo de 100µs para evitar ruídos. Além disso, é necessário um intervalo mínimo de 10ms após a detecção do eco até um novo impulso de disparo. A Figura 5 ilustra o que foi explicado.

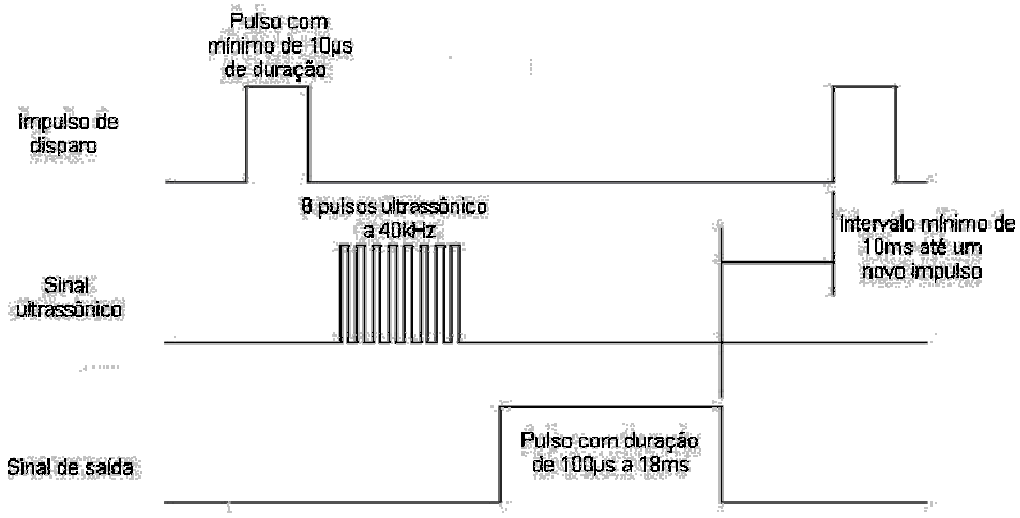

Figura 5 – Diagrama explicativo do funcionamento do sonar.

O módulo RCM4500W é responsável por medir o tempo em que o eco demorou para ser detectado, isto é, o tempo em que o sinal de saída esteve a nível lógico alto, e dessa forma calcular a distância a que se encontra o obstáculo que provocou o eco. O valor da distância até o obstáculo pode ser calculado por:

$$d = \frac{t}{2} \cdot v_{som} \qquad \textbf{(2.1)}$$

onde: d= distância em metros.

t= tempo de duração do pulso na saída do sonar.

V_{som}= velocidade do som no ar ≈ 343m/s

No RCM4500W existe a função input capture que trabalha juntamente com contadores internos do microprocessador e é capaz de determinar o tempo em que o sinal de saída fica em nível lógico alto, assim é possível determinar a distância em que o robô está do obstáculo ou se não foi encontrado obstáculo à frente.

O módulo tem a capacidade de trabalhar com dois canais de *input capture* por vez, assim para serem utilizados os quatro sensores foi decidido alternar a leitura dos sonares, com tempo suficiente de espera para não gerar interferência entre sonares.

2.3 Sensor Infravermelho

Sabe-se que o sonar possui pouca sensibilidade, por isso foi adicionado ao módulo dois sensores infravermelhos Sharp GP2D120, fabricado pela Acroname Roboticsmostrado, o sensor

está ilustrado na Figura 6.

Figura 6 – Sensor Infravermelho

O sensor é alimentado por uma faixa de tensão de 4,5 a 5,5 volts, consome uma corrente de 33mA e é capaz de detectar objetos a uma distância de 4 a 30cm. Ele possui uma saída analógica de tensão que varia com a posição do objeto detectado.

A saída possui uma curva de tensão pela distância, que, para a implementação do algoritmo de leitura, é necessário realizar sua linearização, para isso foram feitas as seguintes transformações:

$$R = \frac{1}{L + 0{,}42} \tag{2.2}$$

$$V = 0{,}037 + 12{,}837 \cdot R \tag{2.3}$$

Sendo:

L=valor da distância em centímetros;

R=variável linearizada, dada em cm^{-1};

V=tensão de saída do sensor.

A curva linearizada é data pelo datasheet do fabricante como mostrado na Figura 7.

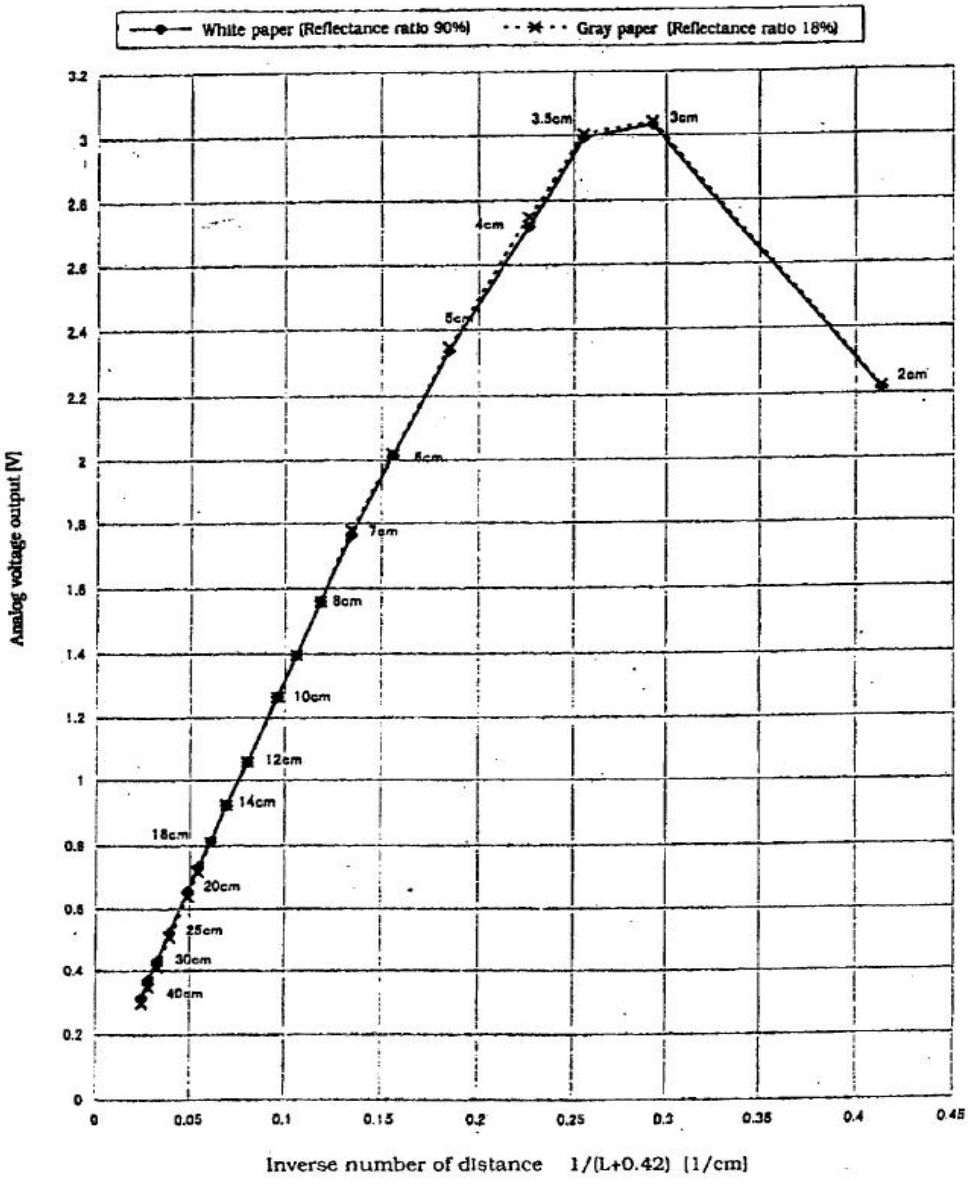

Figura 7 – Curva linearizada do sensor Sharp GP2D120 dada pelo fabricante.

2.4 Encoder

Os encoders são sensores capazes de determinar a posição angular dos eixos dos motores aos quais estão acoplados. No projeto são utilizados dois encoders incrementais do modelo HEDS-5500, tipo A, mostrado na Figura 8 abaixo, fabricado pela Agilent Technologies.

Figura 8 – Foto ilustrativa do encoder HEDS-5500.

O encoder é alimentado por uma faixa de tensão de +4,5V a +5,5V, consome uma corrente típica de 17mA. Possui uma resolução de 500 ciclos por revolução e 2 pinos de saída referidos como canal A e canal B, ilustrados na Figura 9, juntamente com os pinos de alimentação [6].

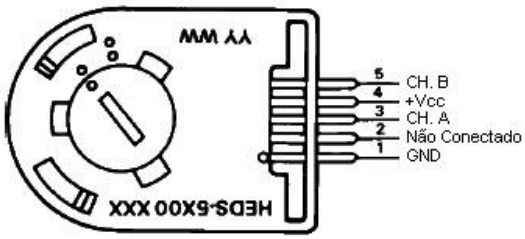

Figura 9 – Esquema de pinos do encoder HEDS-5500.

Os canais A e B são ativados conforme a posição do encoder com a roda em que está acoplada. Internamente o encoder possui um LED próximo a uma lente de policarbonato, que direciona um feixe de luz a fotodiodos como mostrado na Figura 10. Entre estes existe um disco que rotaciona juntamente com o eixo. Esse disco possui furos que em certa posição permite a passagem do feixe de luz do LED para os diodos. Assim, devido a disposição dos diodos e dos furos temos na saída uma seqüência de valores como mostrado na Figura 11.

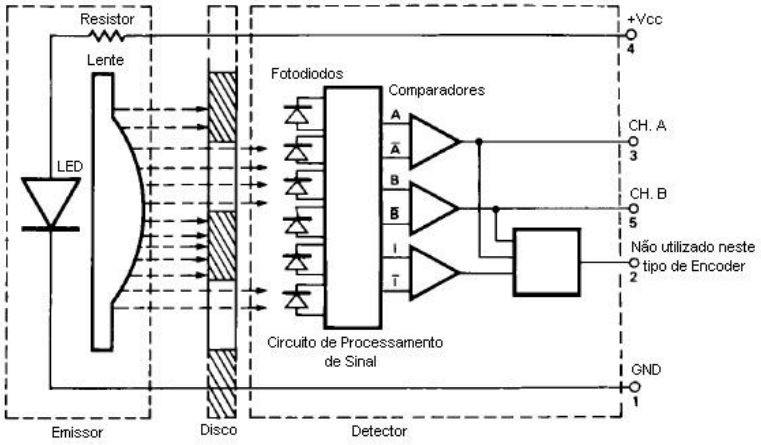

Figura 10 – Diagrama de blocos do circuito interno do encoder HEDS-5500.

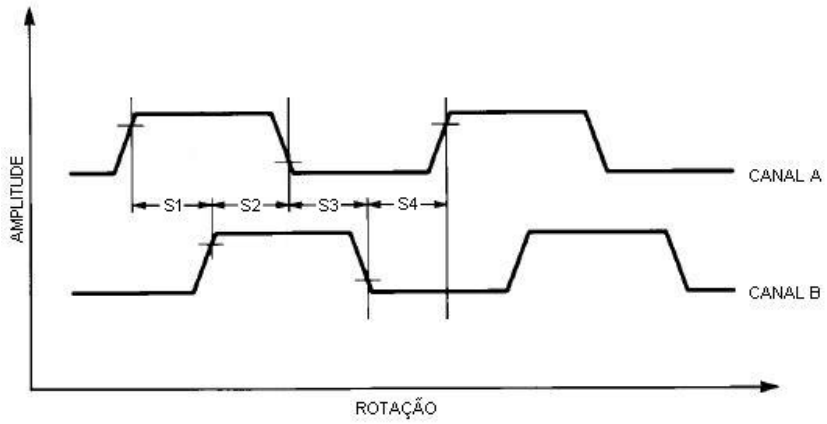

Figura 11 – Seqüência de pulsos nos canais A e B

Nota-se que a seqüência de pulsos gera 4 posições diferentes de saída, na qual é possível identificar o sentido de rotação do motor.

Na decodificação, o decoder incremental faz a contagem das diferentes posições de saída, assim, sendo o encoder de 500 ciclos por revolução, e 4 posições de saída por ciclo, terá então 2000 pulsos para cada rotação completa, essa contagem de pulsos está ilustrada na Figura 12. O contador também deve incrementar ou decrementar o número de pulsos dependendo do sentido de rotação do motor.

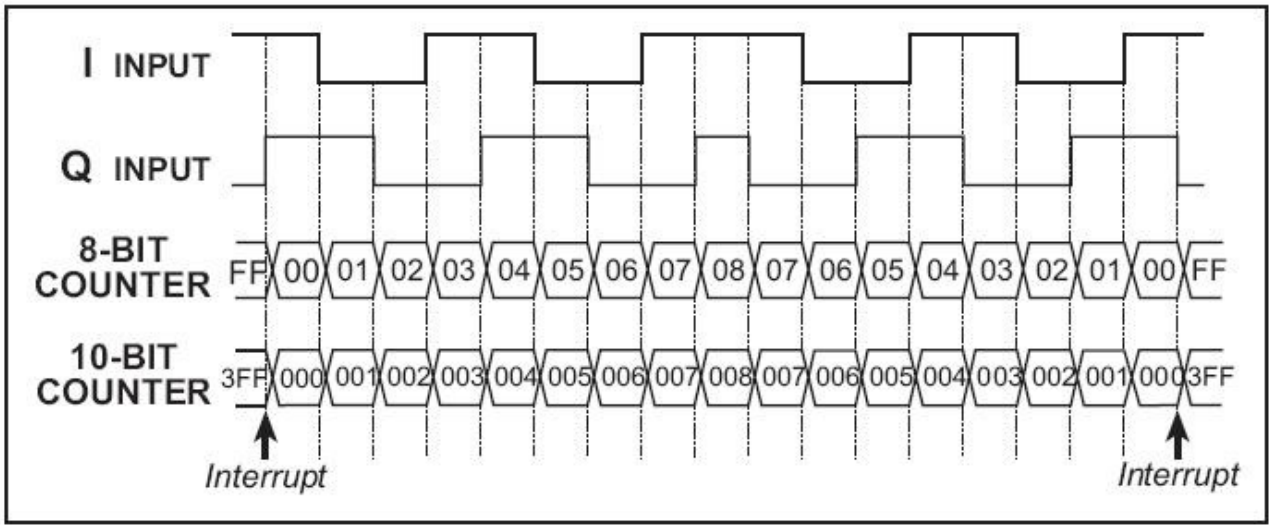

Figura 12 – Ilustração da contagem de pulsos do decoder incremental.

O módulo RCM4500W já possui uma função para encoders na qual realiza todos esses procedimentos citados acima. A função denominada Quadrature Decoder é capaz de operar com 2 encoders ao mesmo tempo. Os comandos envolvendo essa função estão listados abaixo:

- *void qd_init(int iplevel);* inicia o decodificador
- *long qd_read(int channel);* le o valor do encoder
- *void qd_zero(int channel);* zera a contagem do decodificador

2.4 Motor

O robô é constituído por duas rodas, cada uma ligada a um servo motor do modelo CS-60, fabricados pela Hobbico. O motor está ilustrado na Figura 13. Os motores são ligados diretamente na bateria, e seu acionamento é feito através de PWM, junto com a utilização de pontes H.

Figura 13 – Foto ilustrativa do servo motor CS-60.

O fabricante divulga poucas informações sobre esse servo motor, sendo por esse motivo a escolha do método de modelagem explicada no capítulo 3. As únicas informações que temos estão mostradas na Tabela 1 abaixo:

Tabela 1 – Características do servo motor CS-60 para diferentes valores de tensão

+4,8V		+6,0V	
Velocidade (segundos/60º)	**Torque (kg.cm)**	**Velocidade (segundos/60º)**	**Torque (kg.cm)**
0,19	3	0,16	3,5

2.5 PWM

A modulação por largura de pulso - mais conhecida pela sua sigla em inglês "PWM" (Pulse-Width Modulation) - de fontes de alimentação envolve a modulação de sua razão cíclica para controlar o valor da alimentação entregue a carga. Assim, pelo chaveamento de tensão na carga, com a razão cíclica apropriada, temos na saída um nível de tensão desejado [7]. A razão cíclica é definida pela razão entre o tempo em que a chave fica ligada em um período, sobre o período de cada ciclo.

O módulo RCM4500W possui 4 canais configuráveis para uso do PWM. Cada um desses canais consiste em um contador de 10 bits e, portanto sua sensibilidade de valores é de 1/1024.

A utilização do PWM pode gerar pausas no chaveamento que podem causar uma oscilação significativa do motor, para isso o módulo oferece uma configuração especial, o modo Spread, na qual permite subdivisão de cada pulso do sinal do PWM em 4 pulsos menores de 256 contagens. Dessa forma tal oscilação é minimizada, e assim é garantida uma rotação mais estável do motor. Em caso de valores não divisíveis por 4, a subdivisão do PWM fica como mostrado na Tabela 2.

Tabela 2 – Valores de PWM no modo Spread.

Pulse-Width LSBs	1st	2nd	3rd	4th
00	$n/4 + 1$	$n/4$	$n/4$	$n/4$
01	$n/4 + 1$	$n/4$	$n/4 + 1$	$n/4$
10	$n/4 + 1$	$n/4 + 1$	$n/4 + 1$	$n/4$
11	$n/4 + 1$	$n/4 + 1$	$n/4 + 1$	$n/4 + 1$

Sendo o “n”, usado na Tabela 2, o próximo valor do PWM divisível por 4.

Além do modo Spread também é possível usar o modo Supress em que é emitido um pulso de PWM a cada 2, 4 ou 8 ciclos, porém no projeto será usado somente o modo Spread.

O módulo oferece algumas funções para ajudar na configuração do PWM, sendo elas:

unsigned long pwm_init(unsigned long frequency);habilita os canais do PWM

int pwm_set(int channel, int duty_cycle, int options);especifica o canal, o valor do PWM e configure as portas I/O.

O modulo também oferece alguns macros que facilitam a programação:

-Definição da porta a ser utilizada:

PWM_USEPORTC (portas PC4, PC5, PC6 e PC7)
PWM_USEPORTD (portas PD4, PD5, PD6 e PD7)
PWM_USEPORTE (portas PE4, PE5, PE6 e PE7)

-Utilização do modo Spread:

PWM_SPREAD

2.6 Ponte H

O nome ponte H é dado pela forma que assume o circuito quando montado. É um circuito eletrônico que permite alternar a rotação de um motor DC através de um microcontrolador, sendo possível assim controlar a direção das rodas do robô.

. É constituído de quatro chaves, S1-S4, como mostrado na Figura 14, de modo que são acionadas de forma alternada, S1 e S4 ou S2 e S3. Para cada configuração das chaves o motor gira em um sentido [8].

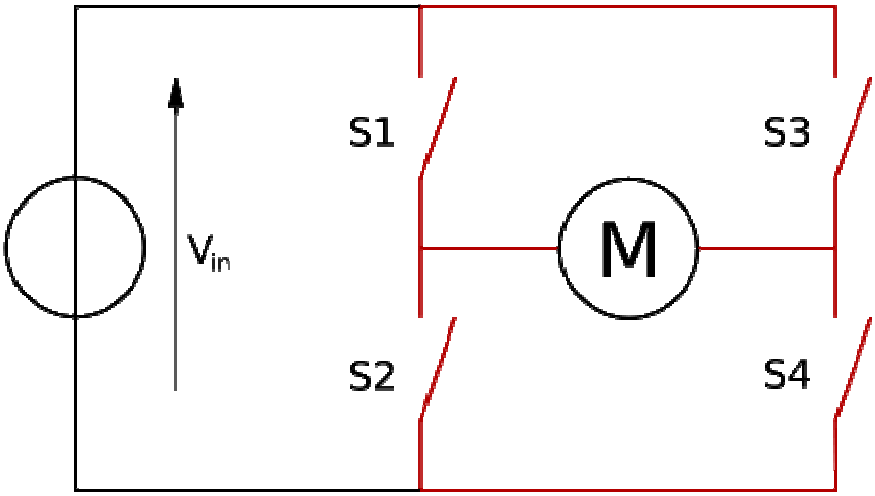

Figura 14 – Esquemático simples de uma ponte H

Em nenhum momento as chaves S1 e S2 ou S3 e S4 podem ser ligadas simultaneamente, pois geraria assim um curto circuito. Para evitar isso e facilitar o controle foi adicionado na montagem o CI L29 3D, que gera comportamentos do motor como mostra a Tabela 3.

Tabela 3– Comportamento do motor DC para diferentes combinações de entradas do CI L293D

EN	1A	2A	Função
H	L	H	Vira Direita
H	H	H	Vira Esquerda
H	L	L	Parada Rápida
H	H	H	Parada Rápida
L	X	X	Parada Rápida

L=low, H=high, X=don`t care

Na conexão de uma ponte H também é aconselhável a adição de diodos que permitem, na abertura das chaves, o retorno da corrente para a bateria, aumentando assim a autonomia da bateria. Na arquitetura do CI utilizado já existe internamente a conexão desses diodos.

2.7 Fornecimento de energia

Para a alimentação do robô foi utilizada uma bateria de Lítio Polímero (Li-Po) de 2 células, com tensão de saída nominal de +7,4V e capacidade de 1700mAh. A tensão de saída da bateria é utilizada diretamente para alimentar os motores devido ao alto consumo de corrente. Os componentes restantes do robô operam em tensão entre +3,3V e +5V, para isso foram utilizados reguladores de tensão para a tensão desejada.

O módulo RCM4500W também possui um indicador de carga na qual um LED é aceso quando é atingido um valor menor do que +4,55V na bateria.

2.8 Estrutura mecânica do robô

Foi possível estimar o tamanho e o peso do robô através das dimensões de cada um de seus componentes que o compõe. Tais medidas são relevantes para a dinâmica do robô, pois são fatores limitantes da velocidade de rotação dos motores, devido à influência no torque exigido pelo motor, e na velocidade de rotação em torno do eixo vertical, devido à distância entre os pontos de contato das rodas. A seguir são apresentadas as relações necessárias para estimar as características dinâmicas do robô.

- Velocidade linear do robô:

$$V_l = \omega_{motor} \frac{d_{roda}}{2} \qquad \textbf{(2.4)}$$

onde:

V_l = velocidade linear do robô em cm/s;

ω_{motor} = velocidade angular do motor em rad/s;

d_{roda} = diâmetro da roda em cm.

- Velocidade de rotação do robô em relação ao eixo vertical:

$$V_{rot} = \frac{2V_l}{D} \quad \textbf{(2.5)}$$

onde:

V_{rot} = velocidade de rotação do robô em rad/s;

D = distância entre as rodas em cm.

- Massa que o robô suporta:

$$m = 2\frac{\tau}{\mu_{din} d_{roda}} \quad \textbf{(2.6)}$$

onde:

m = massa que o robô suporta, em kg;

τ = torque do motor em kg.cm;

μ_{din} = coeficiente de atrito dinâmico;

Como já foi mostrado anteriormente, o motor operando a uma tensão de 6V teria velocidade angular de 6,54rad/s. Considerando que o robô possui rodas com 5,5cm de diâmetro, sua velocidade linear será de aproximadamente 18cm/s, que é um valor razoável. Utilizando o mesmo motor e considerando que a distância entre as rodas é de 9,2cm, obtém-se que a velocidade de rotação do robô em relação ao eixo vertical é de aproximadamente 3,9rad/s. É preferível que esta velocidade não seja muito alta para garantir um controle melhor do robô.

Com relação à massa suportada pelo robô, levando-se em consideração um coeficiente de atrito dinâmico igual a 0,8 (entre borracha e piso seco) e sabendo-se que o torque do motor à tensão de 6V é de 3,5kg.cm, tem-se que o robô suporta cargas de até aproximadamente 1,6kg.

Como a alimentação dos motores é feita com uma bateria de +7,4V, tem-se que o comportamento dos motores é um pouco diferente do que o dos valores estimados acima, atingindo velocidades maiores e suportando cargas maiores. A Figura 15 apresenta a montagem final do robô desenvolvido no LASI [9].

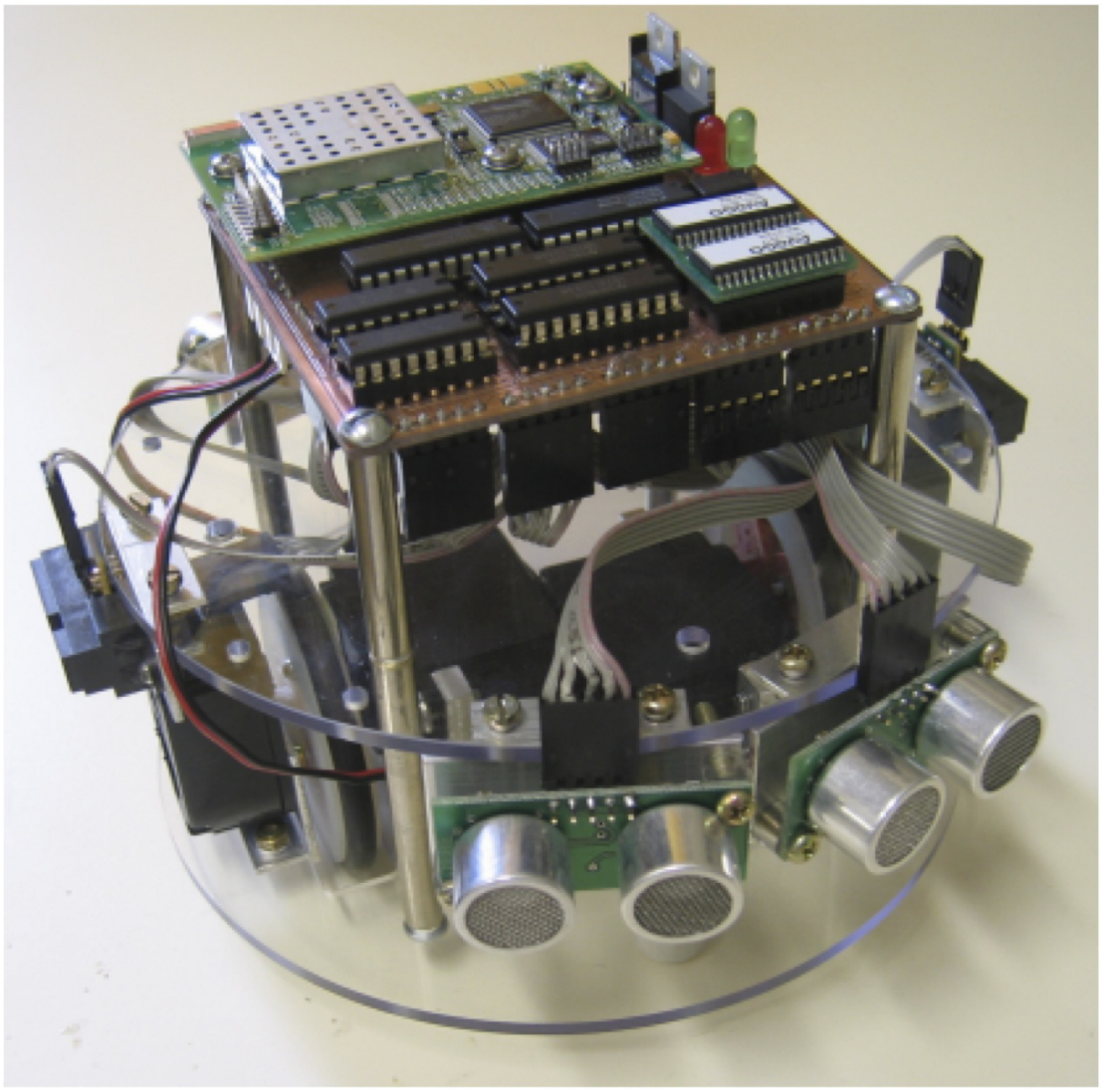

Figura 15 – Estrutura mecânica do robô.

3. Modelagem do Robô Móvel

A Hobbico, fabricante do motor, divulga poucas informações sobre o produto utilizado, e precisamos gerar um modelo matemático para esse motor. Para isso será utilizado o programa MATLAB. Esse programa da empresa MathWorks possui alta performance e um ambiente fácil de se utilizar. Ele possui várias ferramentas, dentre as quais, serão utilizadas principalmente duas delas, o Simulink e o System Identification Toolbox.

O simulink é uma ferramenta para modelagem, simulação e análise de sistemas dinâmicos. Sua interface primária é uma ferramenta de diagramação gráfica por blocos e bibliotecas customizáveis de blocos. O software oferece alta integração com o resto do ambiente do MATLAB, e é amplamente usado em teoria de controle e processamento digital de sinais para projeto e simulação multi-domínios [10].

O System Indentification Toolbox pode ser acessado digitando o termo Ident na janela de comando do MATLAB. É um software que permite a construção de modelos matemáticos para sistemas dinâmicos a partir de informações de entrada e saída do sistema. Essa informação ajuda a modelar sistemas que não são facilmente modelados por métodos convencionais ou por especificações, como processos químicos e dinâmicas de motores mais complexos [11].

É possível a determinação do modelo do motor a partir de contas através das características do gráfico de resposta a rampa do motor. Porém pela disponibilidade de se utilizar tal ferramenta do MATLAB se optou pelo método computacional.

Inicialmente usando o Ident foi observado a necessidade de se obter vários pontos de posição, que seria a saída do sistema, a partir de certa tensão, sendo essa a entrada. Foi utilizada então uma entrada de metade da tensão nominal, portanto, metade do valor de PWM. Como o PWM possui valores entre 1 e 1024, será utilizado então 512. Para facilitar, a variável será um valor de 0 até 1, multiplicada por 1024 na programação, assim, o valor de entrada usado na modelagem será 0,5.

Para adquirir os pontos necessários foi usado um encoder incremental acoplado a roda. O decoder do módulo RCM4500W gera valores que aumentam ou diminuem dependendo do sentido de rotação. Observou-se no datasheet que o encoder gera 2000 pulsos por revolução, o robô possui

rodas de 5,5cm de diâmetro. Assim, pela seguinte fórmula pode-se transformar os valores do encoder em velocidade do robô.

$$V = \frac{|E_{atual} - E_{anterior}| 2\pi r}{2000Ta}(cm/s) \quad \textbf{(3.1)}$$

Sendo:

Eatual=Valor atual do encoder

Eanterior=Valor anterior do encoder

r=Raio da roda do robô

Ta=Período de amostragem do encoder

Para coletar os dados do encoder e fazer a modelagem do motor foi necessário construir um programa que colocava as informações do encoder na tela de I/O do compilador, passa isso foi utilizado o comando "printf" a cada 10ms.

Primeiramente a aquisição dos pontos foi feita através do comando delay, realizando uma espera de 10us entre amostras, e observou-se uma resposta muito instável, como mostrado na Figura16.

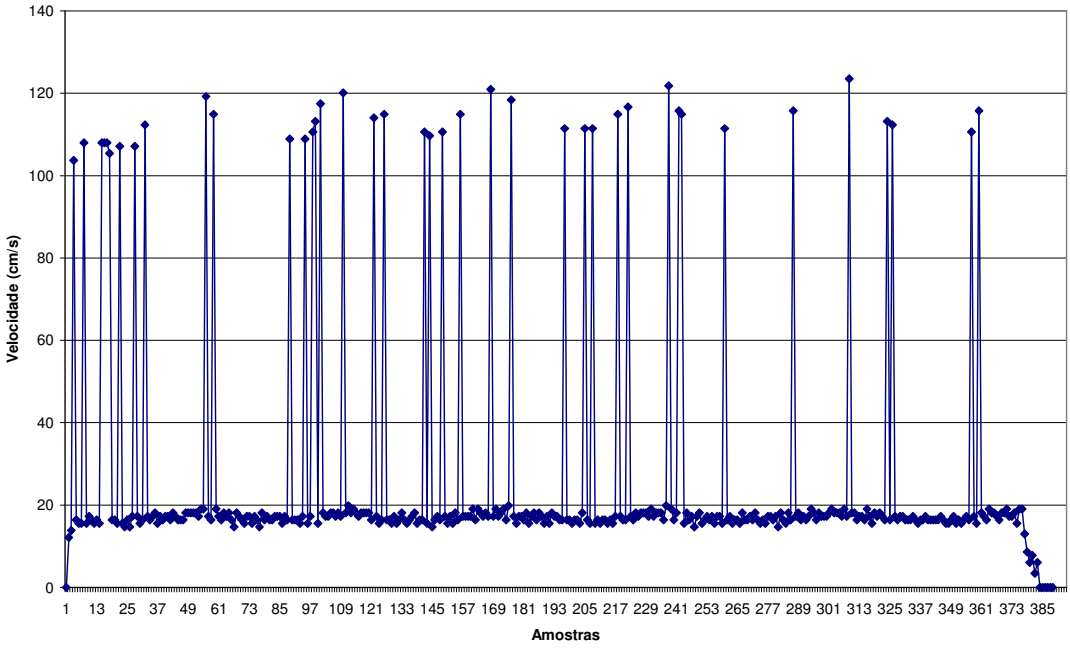

Figura 16 – Gráfico mostrando instabilidade das amostras.

Após muita pesquisa foi encontrado no manual do Dynamic C, o compilador usado, uma informação que levou a reformulação do programa. No manual está documentado que o comando delay pode gerar um erro de até 25ms, que está muito acima do valor que está sendo utilizado, portanto os dados obtidos não possuem nenhum valor. O comando delay deve ser utilizado apenas em casos que não é necessário precisão na espera.

Assim foi estudado o funcionamento dos timers para a utilização de interrupções de 10 ms. O Dynamic C fornece exemplos de programas utilizando interrupção, porém, programas extensos para um funcionamento genérico, como era desejado minimizar o programa pesquisou-se a fundo o funcionamento das interrupções.

A interrupção é ativada a partir do overflow do contador, a contagem é iniciada em 0 e irá até um valor inserido pelo programador. A velocidade de cada incremento é determinada pela freqüência do clock, que pode ser configurada para operar em freqüências de 2, 8 ou 16 vezes menores que a freqüência do microcontrolador. O limite superior da contagem é de 16 bits, portando será necessário inserir esse valor em 2 registradores de 8 bits, TCDHR e TCDLR sendo o primeiro o valor mais significativo e o segundo o menos significativo.

O microcontrolador trabalha com o clock de 29,4912 MHz, e foi observado que se for configurado um valor de clock/2, mesmo utilizando a contagem máxima dos registradores ainda iria gerar interrupções mais rápidas que 10 ms. Por isso no projeto será utilizada a freqüência de clock/16. Para isso deverá ser configurado o registrador TCCR, responsável pela configuração do clock, e escrito o valor binário de 00001101 ou valor hexadecimal de 0D, como observado no manual do Rabbit 4000. Assim obtém-se uma freqüência de 1,8432 MHz. Para isso foi utilizada a seguinte função:

```
WrPortI(TCCR, &TCCRShadow, 0x0D);
```

Com a freqüência de clock fixada é possível determinar os valores de overflow dos registradores. Sendo essa freqüência de 1,8432 MHz deverá ser gerado 18432 incrementos para se obter uma interrupção de 10ms. Assim será necessário escrever nos registradores o valor de 18432d

ou, em hexadecimal, 4800h. Abaixo está a função que realiza a escrita dos registradores TCDLR e TCDHR.

```
WrPortI(TCDLR, NULL, 0x00);
WrPortI(TCDHR, NULL, 0x48);
```

A partir desses valores foi rodado o programa novamente e adquirido novos pontos com uma entrada de 0,5 do PWM, e observou-se uma estabilidade muito maior, o gráfico da velocidade em função do tempo está mostrado abaixo na Figura 17.

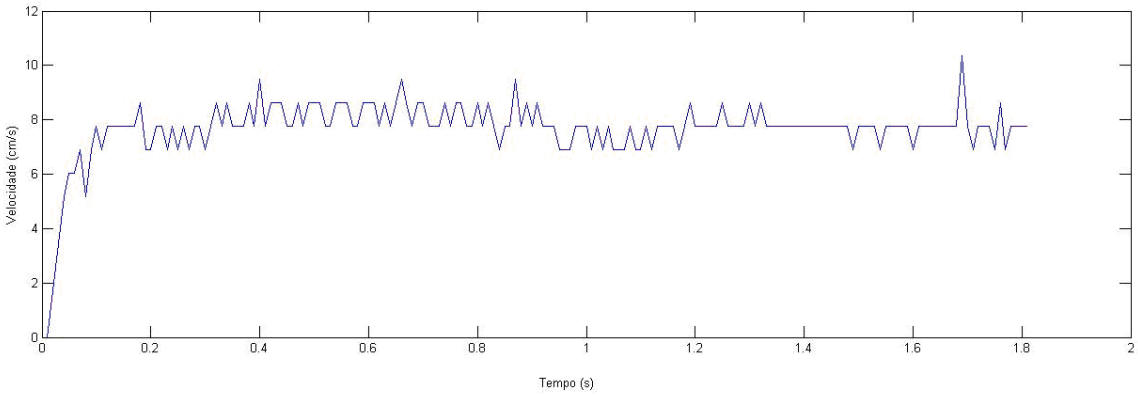

Figura 17 – Resposta da velocidade utilizando interrupção.

O Ident possui uma ferramenta que estima um modelo matemático com os dados de entrada e saída do sistema, vamos assim criar um modelo de segunda ordem da forma:

$$G(S) = \frac{K}{(1+Tp1S)(1+Tp2S)} \qquad \textbf{(3.2)}$$

Após algumas iterações chegou-se a um resultado satisfatório, onde:

Tp1 = 0,34291
Tp2 = 0,0093834
K = 15,742

Nota-se nos pólos uma grande dominância de Tp1, com tempo de resposta cerca de 38 vezes maior que Tp1, essa análise permite aproximar o modelo para um de primeira ordem, essa aproximação não foi realizada nesse trabalho. Assim pode-se criar um diagrama de blocos simples desse sistema, usando a ferramenta Simulink. A Figura 18 ilustra esse diagrama.

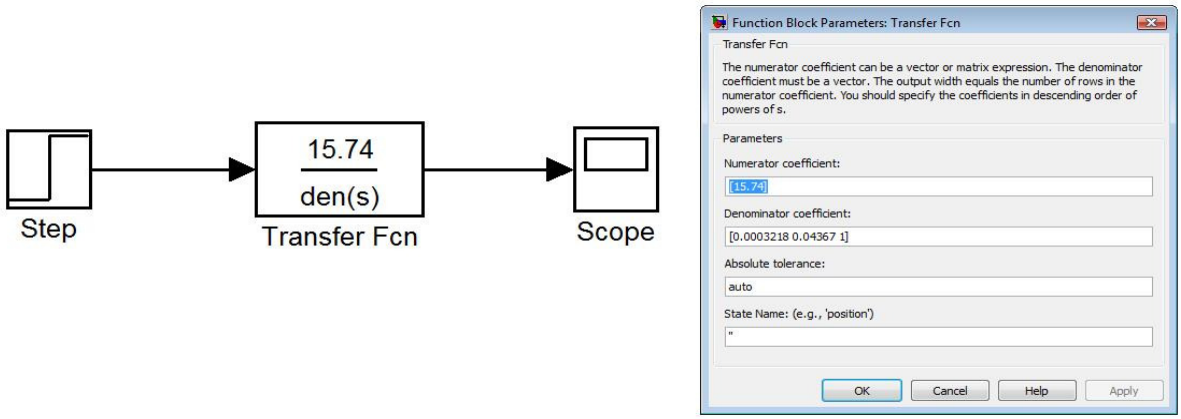

Figura 18 – Diagrama de blocos simples no simulink

Ao lado do diagrama está o bloco do sistema aberto, com os valores gerados pelo Ident. Na Figura 19, gerada pelo Scope do sistema acima, está ilustrada a resposta do sistema a um degrau de 0,5, no mesmo gráfico também está sobreposto o gráfico dos pontos reais obtidos do sistema.

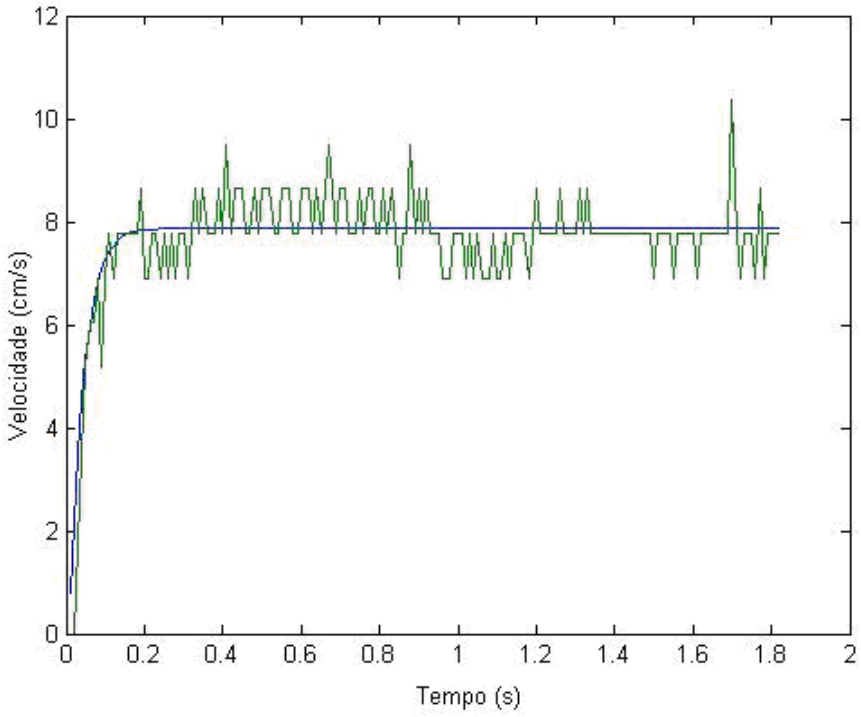

Figura 19– Comparação da resposta ao degrau do modelo do motor com os pontos reais.

Nota-se que a resposta do modelo possui uma característica bem parecida com a dada pelo encoder, portando foi validado o modelo para representar nosso motor.

4. Controlador Proporcional

Um sistema de controle com realimentação permite ajustar a resposta transitória, além de reduzir a sensibilidade do sistema e o efeito de perturbações. Para ser possível realizar a realimentação é necessário a implementação de um sensor no sistema para gerar as informações da saída. No caso o sensor é um encoder que gera dados de posição do robô móvel. Com o modelo determinado pode-ser realizar a realimentação do sistema, como mostrado na Figura 20.

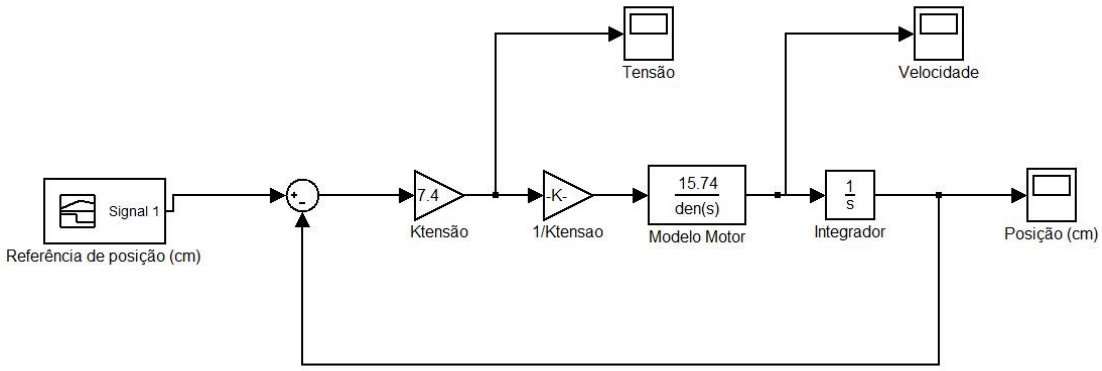

Figura 20 – Diagrama de blocos com realimentação.

No sinal de velocidade real que se adquiriu, observou-se que para uma entrada com razão cíclica de 0,5, é gerado saída de aproximadamente 7,5 cm/s. Assim, utilizou-se na referência uma rampa de posição gerando uma velocidade de 7,5 cm/s e devido ao uso de metade do valor de PWM é esperada uma tensão de aproximadamente metade de seu valor nominal. Após a realimentação devemos ter um sinal entre 0 e 1, como a tensão máxima é de 7,4V foi adicionada uma constante Ktensão de 7,4, para poder analisar a variação de tensão. Na entrada da planta deve-se ter também um valor entre 0 e 1, portanto foi adicionada outra constante no valor de 1/Ktensao.

Como no modelo tem-se um sinal de saída de velocidade deve-se utilizar um integrador para transformar o sinal de velocidade em um sinal de posição.

Nesse sistema foi obtido um erro entre a entrada e a saída, por isso deve-se adicionar um controlador proporcional, Kv, no sistema, assim,

$$G(S) = Kv\frac{15{,}74}{0{,}0003218S^2 + 0{,}04367S + 1}\frac{1}{S} \quad \textbf{(4.1)}$$

Agora é necessário ajustar o ganho de malha fechada para minimizar o erro de estado estacionário. Para calcular o erro em regime devemos utilizar o teorema do valor final, ou seja,

$$\lim_{t\to\infty} e(t) = \lim_{S\to 0} SE(S) \quad \textbf{(4.2)}$$

Portando utilizando uma entrada rampa como entrada de comparação obtém-se para o sistema de malha fechada

$$e_{SS}(\infty) = \lim_{S\to 0} S\frac{1}{1+G(S)}\frac{1}{S^2} \quad \textbf{(4.3)}$$

Assim, substituindo a equação 4.1 na equação 4.3 tem-se:

$$e_{SS}(\infty) = \frac{1}{15{,}75Kv} \quad \textbf{(4.4)}$$

No sistema é desejado um erro estacionário máximo de 1%, portanto deve-se ajustar o ganho Kv para obter tal especificação, de modo que,

$$e_{SS}(\infty) = \frac{1}{15{,}75Kv} = 0{,}01 \quad \textbf{(4.5)}$$

Assim, o controlador proporcional resultante é:

$$Kv = 6{,}35 \quad \textbf{(4.6)}$$

Agora implementando o ganho Kv no sistema, é esperado uma redução do erro entre a saída e a entrada. Para efeitos de comparação, foram montados dois diagramas de blocos, com e

sem essa constante, e adicionado uma perturbação degrau de amplitude 0,1 em ambos os casos. Os sistemas estão mostrados na Figura 21 abaixo.

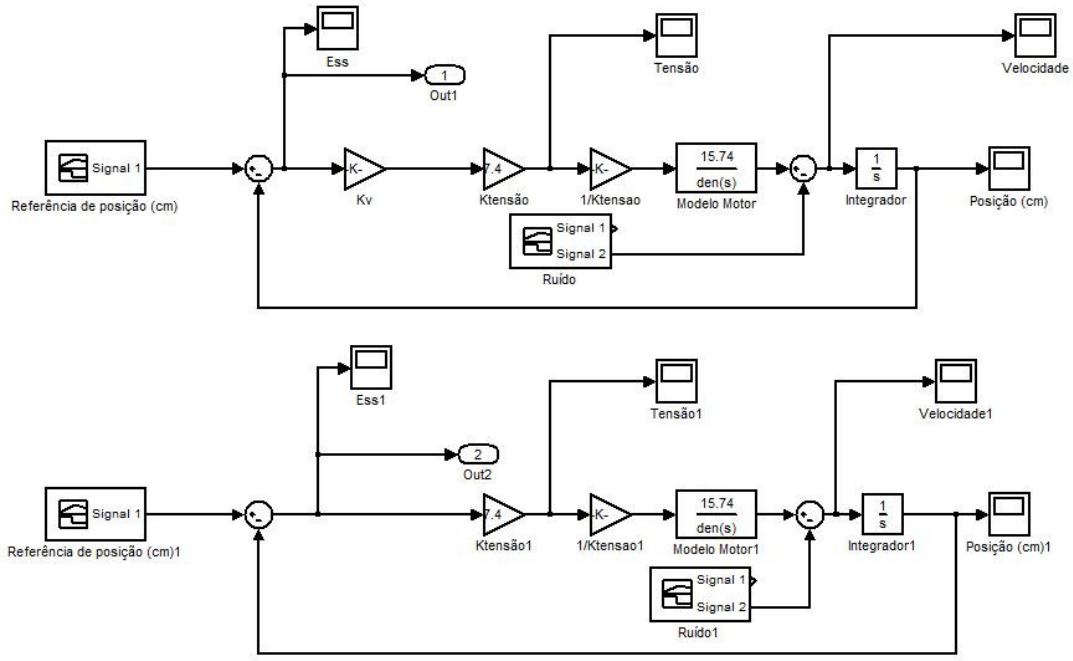

Figura 21 – Sistemas com Kv=1 e Kv=6,35, com perturbação adicionada.

A comparação, Figura 22, foi feita com o erro entre a entrada e a saída, mostrado na Figura 21 por Ess e Ess1, sendo o primeiro com o controlador e o segundo sem o controlador.

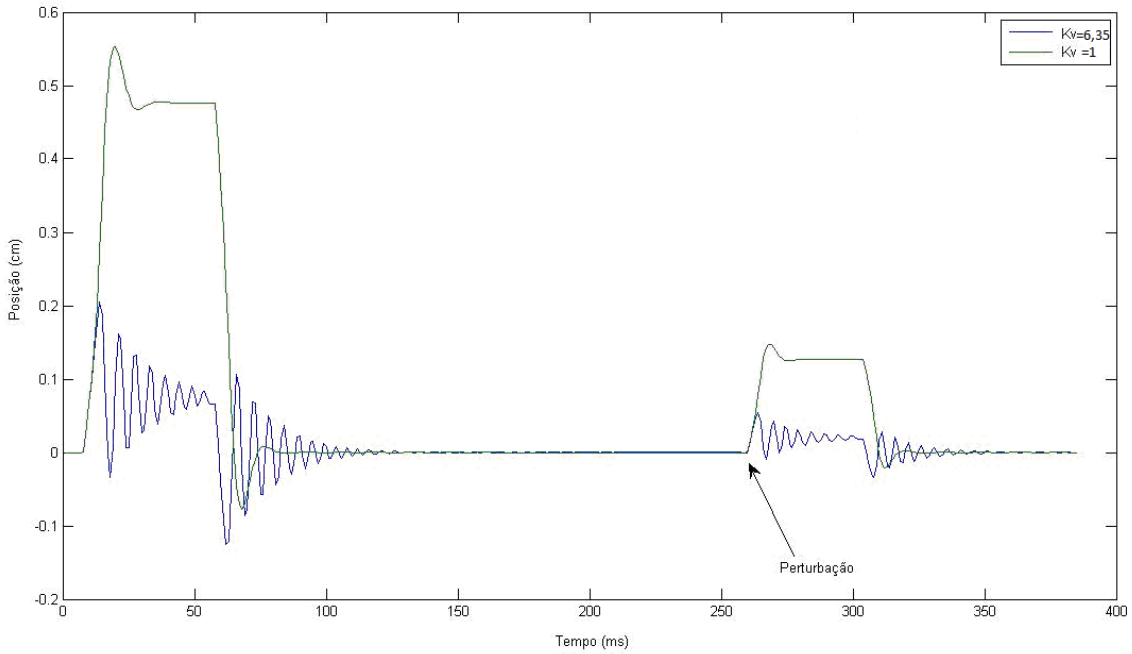

Figura 22 – Comparação do erro do sistema com Kv=6,35 e Kv=1.

Na Figura 22 se pode observar a resposta à perturbação muito menor no sistema com a constante Kv=6,35, isso se deve a diminuição da sensibilidade do sistema, é possível realizar uma comparação numérica para os dois casos mostrados acima, a comparação está ilustrada na Tabela 4.

Tabela 4 – Comparação de sensibilidade com Kv=1 e Kv=6,35.

	Kv=1	Kv=6,35
$e_{SS}(\infty)$	$\frac{1}{15,75 \cdot Kv}=0,063$	$\frac{1}{15,75 \cdot Kv}=0,01$
Variação positiva de 10%	Kv=1.1	Kv=6,985
$e_{SSnovo}(\infty)$	0,058	0,009
$\Delta e_{SS}(\infty)$	(0,063-0,058)=0,005	(0,01-0,009)=0,001
$\frac{\Delta e_{SS}(\infty)}{\lvert r(t) \rvert}$	$\frac{0,005}{1}=0,5\%$	$\frac{0,001}{1}=0,1\%$

Como se pode notar, o sistema sem a constante, representada por Kv=1, tem uma sensibilidade em relação a uma referência r(t)=1 de 0,5% enquanto o sistema com a constante possui uma sensibilidade de 0,1%, portanto o sistema ficou 5 vezes menos sensível o que já é uma grande melhoria [12].

Nota-se nos gráficos picos do sistema na entrada do modelo do motor acima de seu valor nominal, portanto deve-se adicionar um bloco saturador para criar uma situação no modelo mais parecida com o real. Devido a ponte H, o motor gira a roda para ambos os lados, portanto deve-se saturar a entrada da planta com valores entre -1 e 1, que equivalem a -1024 até 1024 valores de PWM.Assim a entrada do motor é mostrada na Figura 23.

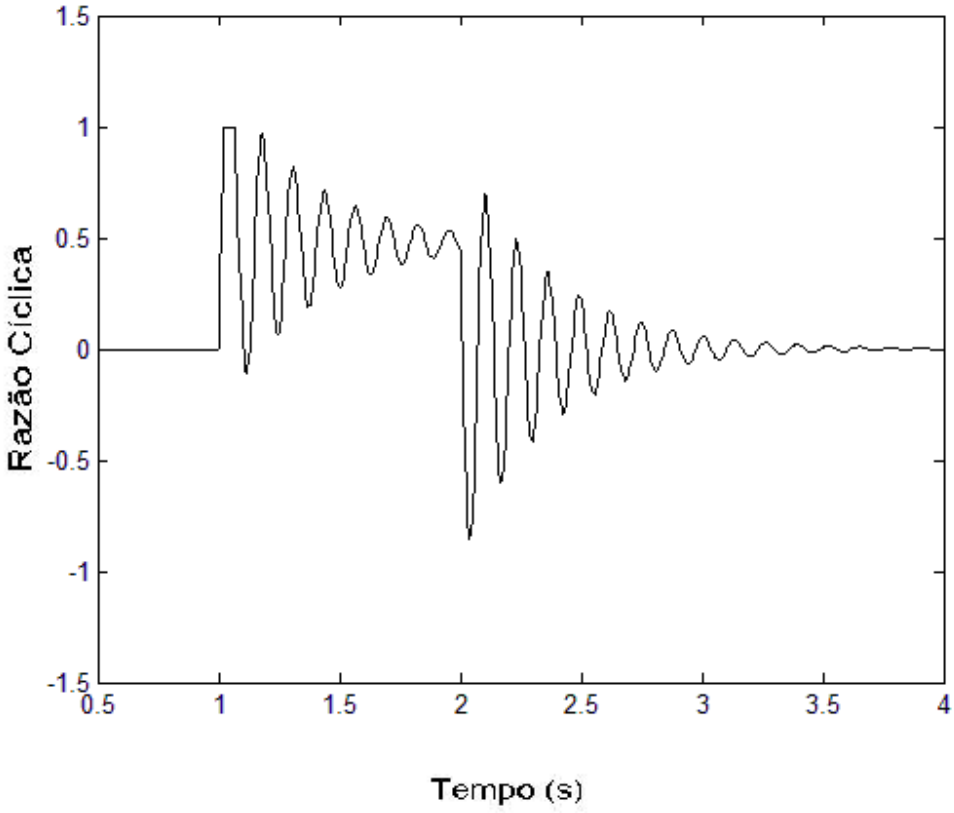

Figura 23 – Razão cíclica com saturação.

5. Controlador por Avanço de Fase

Na saída do sistema pode-se observar uma oscilação na resposta a uma rampa, como mostrado na Figura 24.

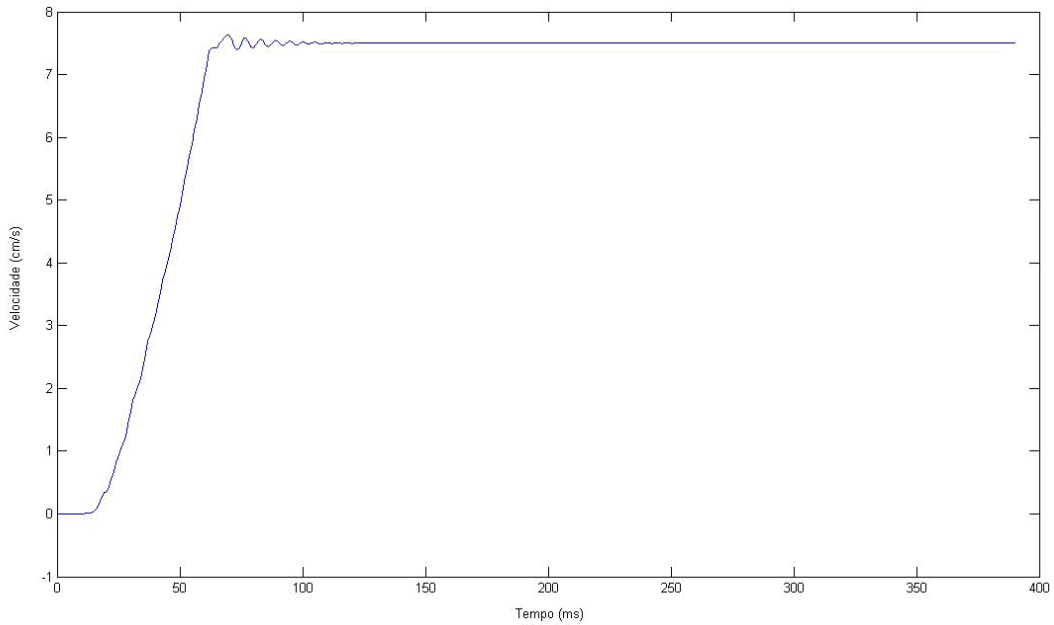

Figura 24 – Resposta a uma rampa de 7,5cm/s

Para minimizar essa oscilação foi decidido adicionar um controlador por avanço de fase no sistema. Esse controlador irá permitir remodelar o lugar das raízes de maneira a obter pólos dominantes desejados em malha fechada podendo assim gerar um menor tempo de subida e acomodação. Além disso a margem de ganho e de fase são melhoradas, e o erro estacionário não é afetado.

Para determinar esse controlador será utilizada outra ferramenta do MATLAB chamada Control System Toolbox, onde se poderá visualizar, adicionar e modificar a posição dos pólos e zeros do sistema. Assim será possível alcançar facilmente os requisitos do sistema. Esta ferramenta é acionada digitando o comando rltool na janela de comando do MATLAB.

Nos requisitos do controlador se tentará alcançar um sobresinal de no máximo 2%. Inicialmente foi adicionado o modelo do robô e observou-se que o sistema está muito distante dos

2% desejados, como mostrado a Figura 25. Na ilustração adicionou-se duas retas que mostram o local dos pólos pretendentes para obter tal requisito.

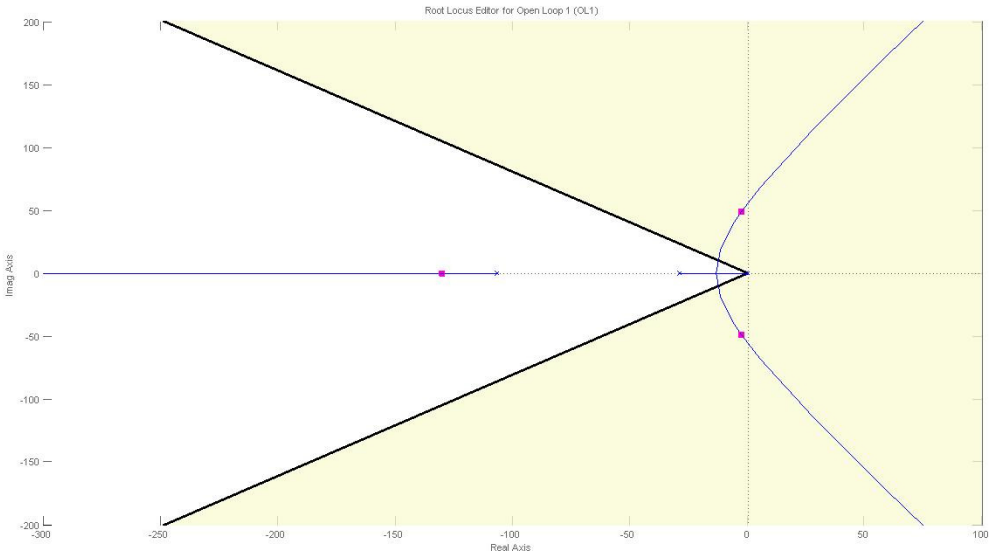

Figura 25 – Local dos pólos do sistema realimentado com controlador proporcional.

Agora adicionando no sistema um controlador por avanço de fase pode-se modificar seus pólos até que a resposta do sistema fique dentro da área de requisito.

Após um tempo movendo os termos do controlador para atingir uma resposta satisfatória foram encontradas as posições dos pólos mostrados na Figura 26.

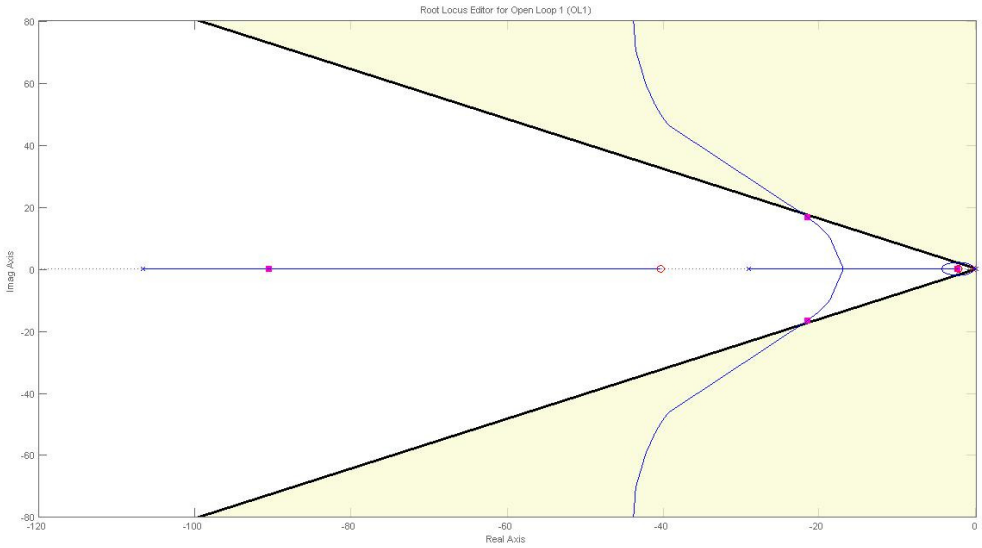

Figura 26 – Local dos pólos do sistema com controlador por avanço de fase.

Nota-se que o objetivo foi atingido, com os pólos dentro da área delimitada pelas retas, portanto tem-se um sobresinal menor que 2%. A planta do controlador está mostrada na equação 5.1.

$$G_C(S) = \frac{(1+0{,}031S)}{(1+0{,}0021S)} \qquad \textbf{(5.1)}$$

Assim foi adicionada a planta no Simulink, como mostrado na Figura 27.

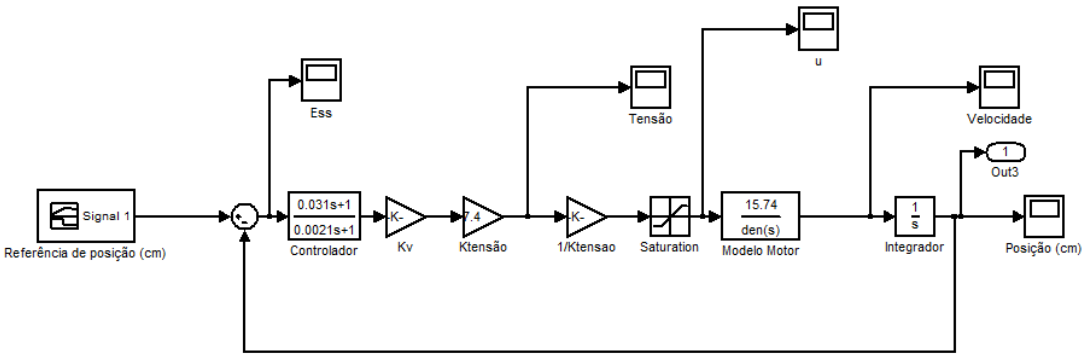

Figura 27 – Diagrama do simulink do sistema com controlador.

Foi gerado um gráfico comparativo entre o sistema com o controlador e sem o controlador por avanço de fase, mostrado na Figura 28.

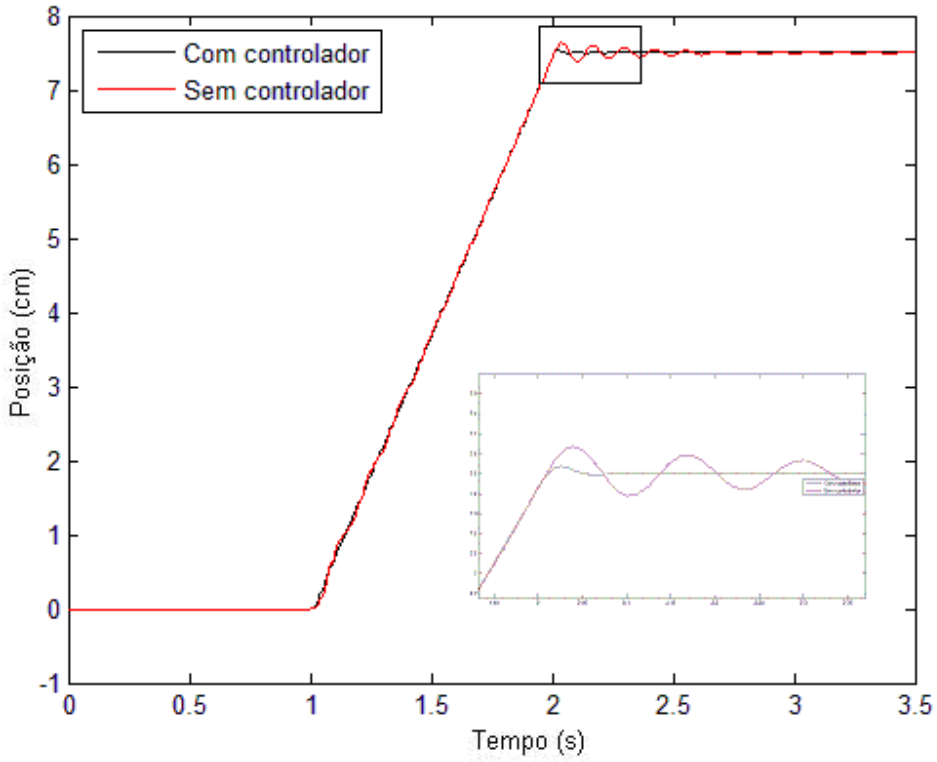

Figura 28 – Gráfico comparativo da posição do sistema com e sem o controlador por avanço de fase para uma entrada rampa.

Pode-se observar uma grande melhoria do sistema com uma oscilação quase imperceptível.

Também foi plotado a saída de posição a partir de um degrau unitário, ilustrado na Figura 29.

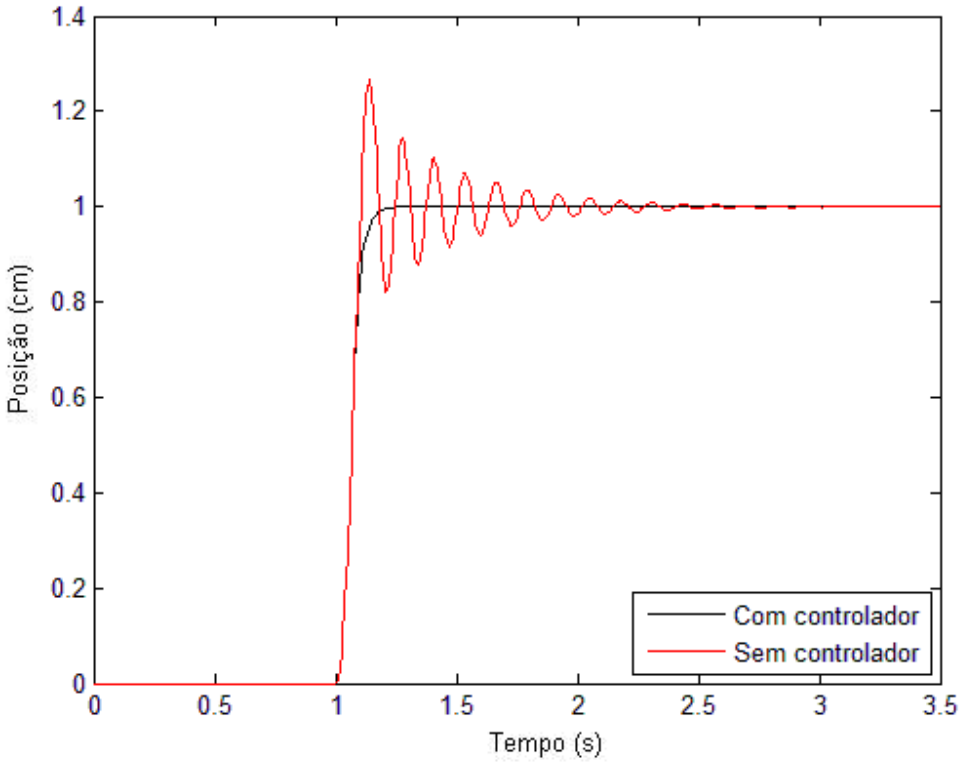

Figura 29- Gráfico comparativo da posição do sistema com e sem controlador o controlador por avanço de fase para uma entrada degrau unitário.

Os gráficos de velocidade e da tensão do motor para uma rampa de 7,5cm/s estão mostrados nas Figuras 30 e 31.

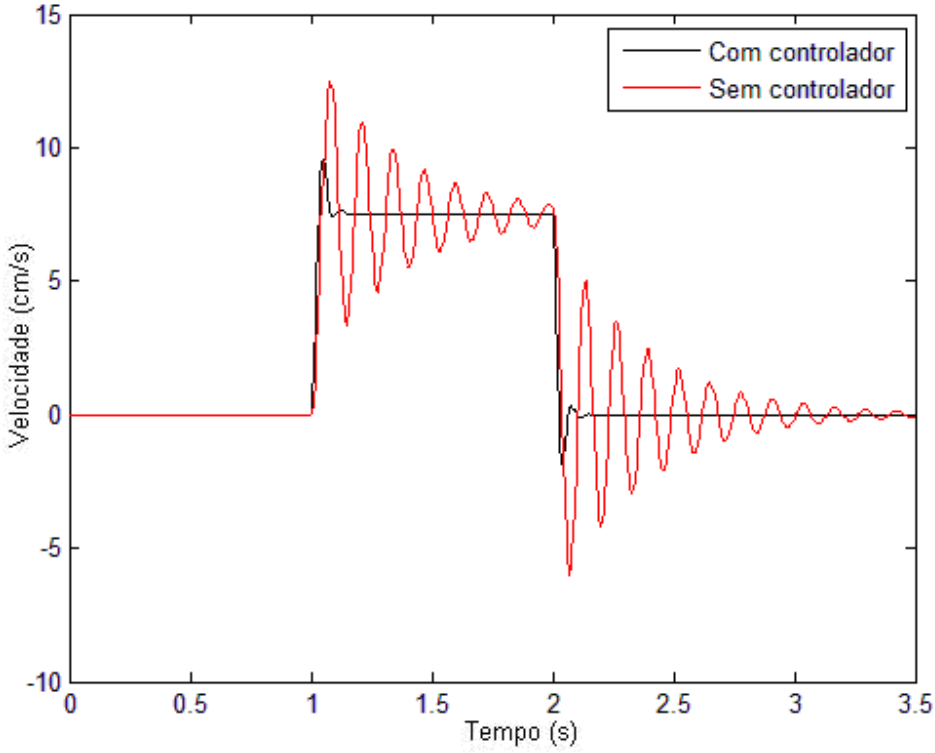

Figura 30- Gráfico comparativo da velocidade do sistema com e sem controlador por avanço de fase para uma entrada rampa.

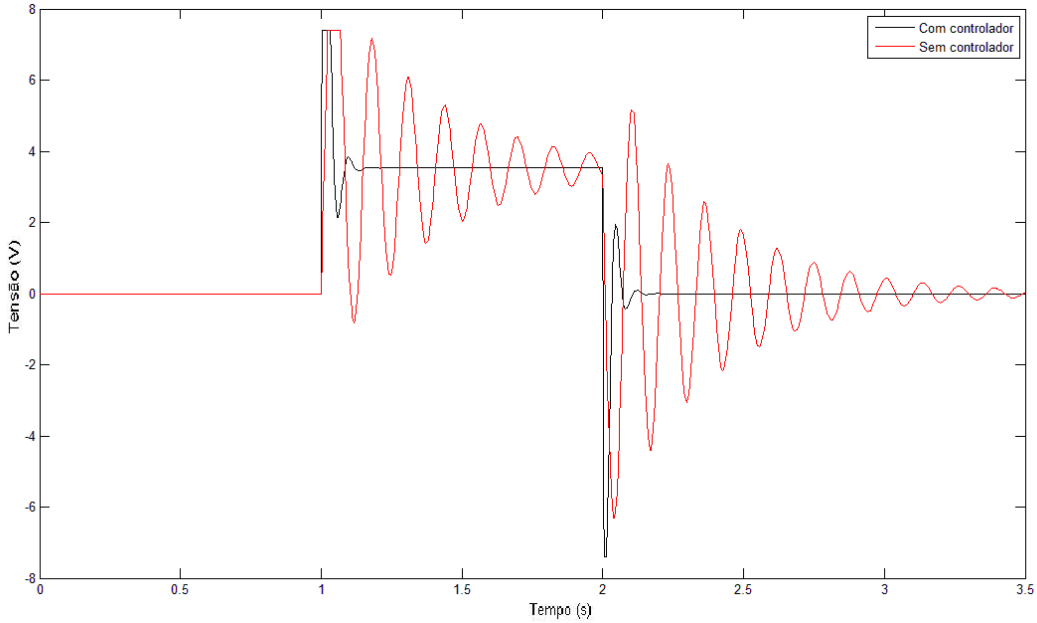

Figura 31 – Gráfico comparativo da tensão do motor com e sem controlador o controlador por avanço de fase para uma entrada rampa.

Claramente se vê uma resposta muito mais estável do sistema com o controlador, nota-se também o efeito do bloco saturador, limitando a tensão do motor entre +7,4 e -7,4.

6. Discretização do Controlador

A transformação bilinear, também chamada de método de Tustin, é usada em processamento digital de sinais e na teoria de controle para tempo discreto para transformar tempo contínuo em tempo discreto e vice-versa.

A transformação bilinear é um caso especial de um mapeamento conformal. A transformação preserva a estabilidade e mapeia todo ponto de resposta de freqüência do sistema contínuo do eixo jw do plano S para um ponto correspondente do sistema discreto em um círculo unitário no plano z [13].

O processo de transformação é realizado fazendo a seguinte substituição na função de transferência:

$$S = \frac{2}{Ta}\left(\frac{1-z^{-1}}{1+z^{-1}}\right) \quad \textbf{(6.1)}$$

onde Ta é o período de amostragem.

Foi escolhido um período de amostragem de 10ms. Assim, fazendo a substituição no controlador chega-se à seguinte expressão:

$$G_C(z) = 6{,}35\left(\frac{0{,}072 - 0{,}052z^{-1}}{0{,}0142 + 0{,}0058z^{-1}}\right) \quad \textbf{(62)}$$

Essa expressão pode ser reescrita como:

$$6{,}35[(0{,}072 - 0{,}052z^{-1})U(z)] = (0{,}0142 + 0{,}0058z^{-1})Y(z) \quad \textbf{(6.3)}$$

Transformando para a equação de diferenças:

$$y_{atual} = -0{,}485\,y_{anterior} + 32{,}197\,u_{atual} - 23{,}254\,u_{anterior} \quad \textbf{(6.4)}$$

Para simular o controlador adicionou-se o bloco de controlador por avanço de fase discretizado no simulink como mostra a Figura 32.

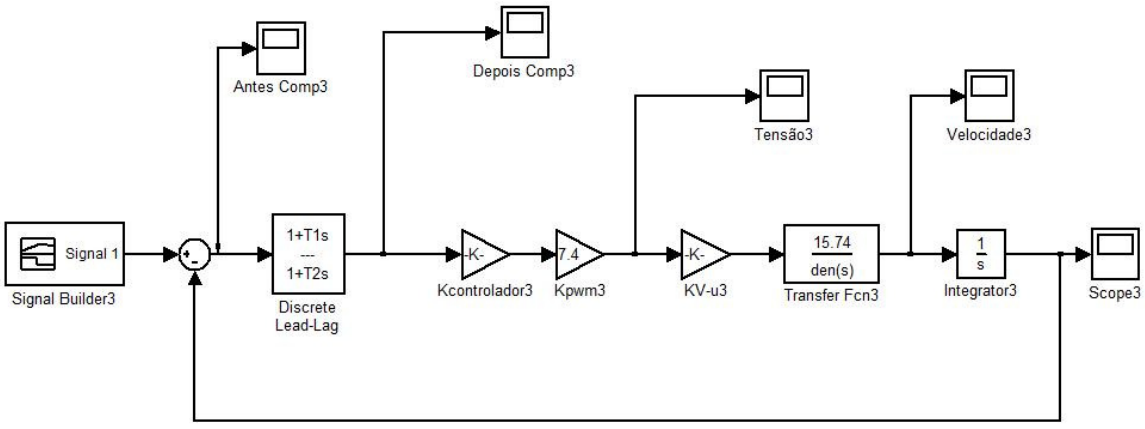

Figura 32 – Diagrama de blocos com o controlador discretizado.

E nossa saída de posição nesse caso está mostrada na Figura 33.

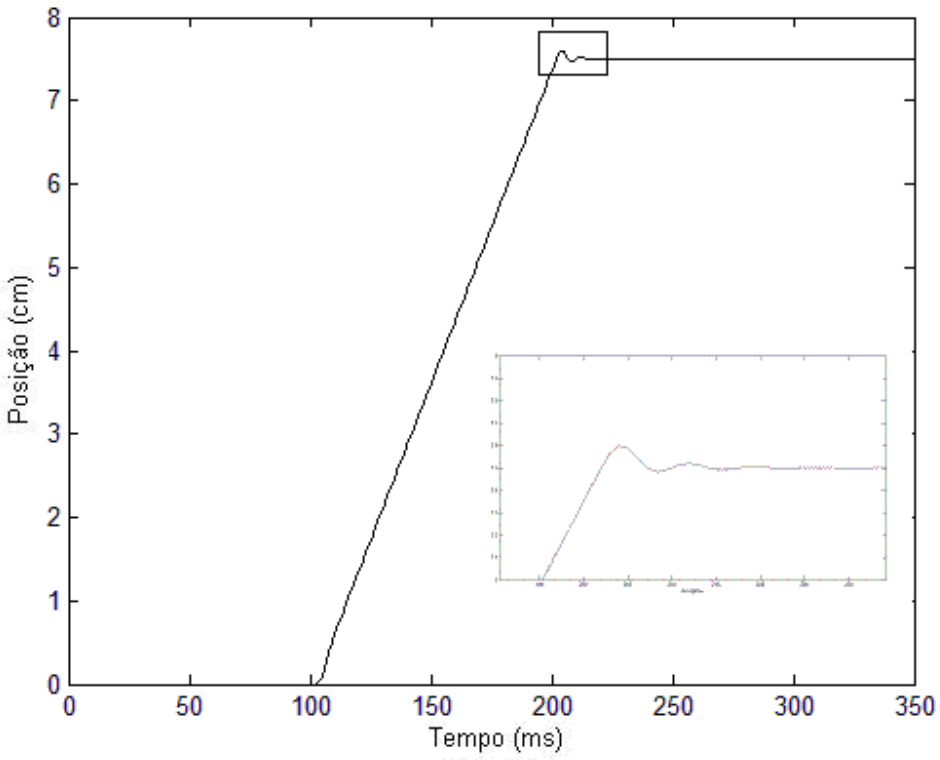

Figura 33 – Posição do robô com o bloco de controlador discretizado

Pode-se observar uma pequena oscilação no final da subida da curva. Essa oscilação poderia ser contornada com um aumento da freqüência de amostragem do nosso programa, porém devido a algumas dificuldades que já foram encontradas usando uma interrupção de maior freqüência foi decidido manter $Ta = 10ms$.

7. Controle de Velocidade

Deseja-se na entrada valores de posição, em cm, que será incrementado a cada interrupção para gerar uma velocidade, e na saída valores entre 0 e 1 que são referentes ao valor máximo e mínimo do PWM, ou seja, de 0 a 1024. Será necessário realizar uma transformação dos valores da equação de diferenças.

A entrada do controlador vem do erro entre a referência e o valor de saída do decoder, como mostrado na Figura 34. A referência será programada em centímetros, portanto é necessária a transformação dos valores do encoder em centímetros.

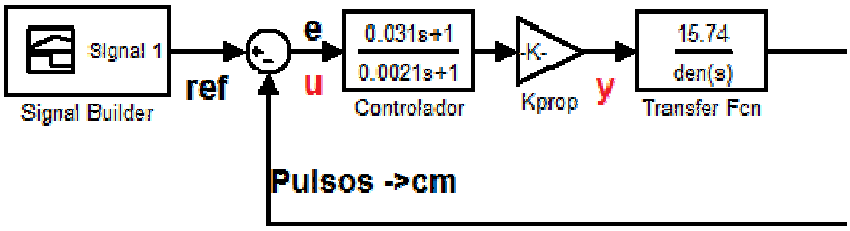

Figura 34 – Ilustração da composição do erro do sistema.

A transformação dos valores do decoder está mostrada na equação 7.1. Com o raio da roda sendo de 5,5cm e 2000 pulsos por revolução, foi determinado que 1 cm equivale a 57,87 pulsos do decoder. Assim o erro do sistema, que equivale à entrada da equação da transformação bilinear, está mostrado na equação 7.2.

$$u = ref - \frac{encoder}{57{,}87} \qquad \textbf{(7.1)}$$

A referência é gerada através de incrementos a cada período de interrupção. E como tem-se uma interrupção a cada 10ms, para uma velocidade de 1cm/s é necessário incrementar a cada interrupção:

$$Inc = \frac{1cm}{s}\frac{57{,}87\,pulsos}{cm}\frac{10^{-2}\,s}{amostra} = 0{,}5787\,\frac{pulsos}{amostra} \qquad \textbf{(7.2)}$$

O programa possui uma variável de entrada que será a velocidade em cm/s que é desejada, que deve ser multiplicada pela variável "Inc" acima, obtendo assim o incremento a cada interrupção.

O trecho de programa abaixo representa essas transformações:

//Trecho do programa principal:

rampa=0;
PWMatual=0;
Uatual=0;
Conversao= 0.5789;
Velocidade=7.5;
*Incremento=Conversao*Velocidade;* Incremento recebe posição a incrementar a cada 10ms

//Trecho da interrupção a cada 10ms:

Decoder=qd_read(motor);
PWManterior=PWMatual ; PWManterior recebe PWMatual
Uanterior=Uatual; Uanterior recebe Uatual
rampa=rampa+1; Variável criada para ajudar no incremento
*Uatual= rampa*Incremento-Decoder;* a cada interrupção Uatual é incrementado de Incremento
*PWMatual=-0.485*PWManterior + 32.197*Uatual – 23.254*Uanterior;*

A cada interrupção o valor de PWMatual deve ser inserido no valor de PWM do motor, o código completo está em anexo, no apêndice B, ao final do trabalho.

8. Controle de posição

Para o controle de posição será necessário simplesmente monitorar o valor da rampa, e a partir de certo valor é possível identificar que o robô chegou na posição desejada, pois se tem a velocidade do robô juntamente com o tempo, que a cada incremento da rampa vale 10ms.

Para transformar os valores de rampa e de velocidade valores de posição (cm), foi utilizada e equação 8.1.

$$Posicao = Velocidade \frac{rampa}{100} \quad \textbf{(8.1)}$$

Assim, comparando o valor de posição de entrada, inserida pelo usuário, com o valor de posição acima, no momento que o valor de posição for maior que o inserido, será escrito um valor nulo de PWM no motor, fazendo o robô parar.

9. Implementação dos Controladores

O software utilizado para escrever, compilar e depurar o código foi o Dynamic C, desenvolvido pela Digi International. É um software com um ambiente amigável com diversas ferramentas que auxiliam o programador.

Foi encontrado um enorme problema com a comunicação do computador com o robô quando adicionado interrupção ao código e isso gerou um grande atraso no cronograma do projeto. Para se obter os pontos iniciais da modelagem foi realizada toda a interrupção em assembly para deixar o código mais simples possível, e também algumas modificações foram feitas nas configurações do programa Dynamic C, como mostrado na Tabela 5.

Tabela 5 – Mudança nas configurações do programa

Função	Valor Padrão	Valor Usado
Bios Baud	115200	57600
Max Negotiated Baud	460800	115200
Serial Response Timeout	400	9000
Serial Retry Count	9	5
Serial Poll Interval	3000	9000
Verify DSR	1	0
Disable Baud Negotiation	0	1

Essas modificações diminuem a fidelidade de conexão entre o robô e o computador, aumentando as tentativas de comunicação, e diminuindo a taxa de transmissão (baud rate), além de outros parâmetros que estabilizam a conexão. Mesmo após essas modificações a comunicação não ficou totalmente estável, porém já foi possível a aquisição dos dados necessários para a modelagem do motor.

Na implementação dos controladores, o microcontrolador Rabbit4000 apresentou muita instabilidade em sua interrupção. Raramente foi mantida a comunicação do microcontrolador com o Dynamic C, sendo que as poucas vezes que foi possível manter a comunicação o código rodava com algumas falhas.

Com isso era possível apenas implementar códigos sem a utilização das interrupções. Códigos como a programação dos sonares e infra-vermelhos, mudanças nos valores de PWM e direção das rodas, foram implementados com sucesso.

10. Conclusão

No início do projeto foi realizado um vasto estudo, tanto na programação do microprocessador, nos vários recursos que o módulo disponibilizava, e no hardware utilizado. Porém através de um material completo de estudos como o "Dynamic C User`s Manual", "Rabbit 4000 User`s Manual", "RabbitCore RCM4500W" e alguns livros de programação em C, o aprendizado foi relativamente rápido e em torno de dois meses já possuía os conhecimentos para dar início ao projeto.

O estudo inicial de timers, registradores, macros, linguagem, compilador, sensores, motor, controladores e o MATLAB não foi o maior atraso do projeto. O maior atraso do cronograma do trabalho foi devido aos problemas de interrupção encontrados no microcontrolador. Tal atraso gerou um corte em alguns objetivos do projeto como a configuração do ZigBee para comunicação wireless.

Utilizando o System Identification Tool, foi determinado o modelo do motor, e verificado que o modelo corresponde muito bem com o real. Ainda utilizando ferramentas do MATLAB foi determinado seu controlador através do RLTOOL e simulado através do SIMULINK. Todas as respostas foram satisfatórias. A partir disso foi feito a transformação bilinear para a discretização do controlador e assim permitir a implementação computacional do controlador.

De um modo geral o projeto gerou resultados bem satisfatórios, foi adquirido um grande conhecimento na área de programação, hardware e microprocessadores. Em uma futura continuação do projeto seria interessante a programação da conexão wireless entre o módulo e um computador externo através do modem ZigBee que o RCM4500W possui.

Bibliografia

[1] WIKIPEDIA. Robô Móvel. Disponível em: http://pt.wikipedia.org/wiki/Rob%C3%B4_m%C3%B3vel Acesso em: 21/09/2010.

[2] Datasheet, RCM4500W.

[3] User`s Manual, RCM4500W.

[4] Dataheet, Devantech SRF04.

[5] Datasheet, Sharp GP2D120.

[6] Datasheet, HEDS-5500.

[7] WIKIPEDIA. Modulação por largura de pulso. Disponível em: http://pt.wikipedia.org/wiki/Modula%C3%A7%C3%A3o_por_largura_de_pulso. Acesso em:29/09/2010.

[8] WIKIPEDIA. Ponte H. Disponível em: http://pt.wikipedia.org/wiki/Ponte_H. Acesso em: 10/10/2010

[9] UEDA, Gilberto. Deselvolvimento de Robô Móvel para Navegação Autônoma. Relatório de Iniciação Científica, 2009.

[10] WIKIPEDIA. Simulink. Disponível em: http://pt.wikipedia.org/wiki/Simulink Acesso em: 20/09/2010.

[11] MATHWORKS. System Identification Tool. Disponível em: http://www.mathworks.com/products/sysid/. Acesso em: 21/09/2010.

[12] DORF, Richard. BISHOP, Ribert. Sistems de Controle Moderno. 8ª edição. Rio de Janeiro, RJ: LTC Editora, 2001.

[13] WKIPEDIA. Bilinear Transform. Disponível em: http://en.wikipedia.org/wiki/Bilinear_transform. Acesso em: 10/10/2010

Apêndice A

Código do programa: Programa de Aquisição.C

```
long timerc_count;

#asm ; Inicia codificao em assembly
   timerC_isr::
   push af            ; salva registradores usados
   push bcde
   push jkhl
   ioi ld a, (TCCSR)    ; clear the interrupt request and get status
   C   printf("%d\n", timerc_count);  Mostra na tela I/O a contagem do encoder
   ld bcde, (timerc_count)
    ld jkhl, 1
   add jkhl, bcde
   ld (timerc_count), jkhl
   pop jkhl
   pop bcde
   pop af             ; recupera registradores usados
   ipres
   ret
#endasm ; Termina codificacao em assembly

 main()

  {
  int waitFlag;
  timerc_count = 0;
  WrPortI(TCCR, &TCCRShadow, 0x0D);//  Seta clock/16
  WrPortI(TCDLR, NULL, 0x00);
```

```
WrPortI(TCDHR, NULL, 0x48);// Limitante superior do contador
SetVectIntern(TIMERC_OFS/0x10, timerC_isr);
 WrPortI(TCCSR, &TCCSRShadow, 0x01);  // Ativa timer C
  while(1)
   {
costate
waitfor( DelayMs(10000))   ; //Roda o programa por 10s
   }
```

Apêndice B

Código do programa: Programa Final.C

```
#define QD_10BIT_OPERATION      //uso de 10 bits na quadratura
#define QD1_USEPORTD            // usa portas D pinos 1 e 0 - Motor 1
#define QD2_USEPORTD            // usa portas D pinos 3 e 2  -  Motor 2
#defins FREQ 10UL               //Frequencia do PWM
#use XBEE_API.LIB

ENDPOINT_TABLE_BEGIN

ENDPOINT_TABLE_END

float PWMatual, PWManterior,Conversao, Velocidade, Incremento,Uatual, Uanterior;
long timerc_count, reading, rampa ;
int motor, pwm_options;

#asm
   timerC_isr::
   push af          ; save used registers
   push bcde
   push jkhl
   ioi ld a, (TCCSR)    ; clear the interrupt request and get status
C  PWManterior=PWMatual ;
C  Uanterior=Uatual;
C  rampa=rampa+1;
C  Decoder = abs(qd_read(motor));
C  printf("%10ld\n", Decoder);     Insere valores do encoder na tela I/O
C  BitWrPortI(PEDR, &PEDRShadow, 1, 0);
C  BitWrPortI(PEDR, &PEDRShadow, 0, 1);
C  Uatual= rampa*Incremento-Decoder;
```

```
 C   PWMatual=-0.485*PWManterior + 32.197*Uatual – 23.254*Uanterior;
 C   pwm_set(0, (int)(PWMatual * 1024.0), pwm_options);
 C   pwm_set(1, (int)(PWMatual * 1024.0), pwm_options);
    ld bcde, (timerc_count)
            ld jkhl, 1
    add jkhl, bcde
    ld (timerc_count), jkhl
    pop jkhl
    pop bcde
    pop af              ; restore used registers
    ipres
    ret
#endasm



//----------------Fim da interrupção do timer

            main()
{
   unsigned long  freq, pulse_width;
   int    vel, leu1, leu2, caracter;
   char capture_status;
   float Pvel;
   int waitFlag; timerc_count = 0;
   WrPortI(TCCR, &TCCRShadow, 0x0D);
   WrPortI(TCDLR, NULL, 0x00);
   WrPortI(TCDHR, NULL, 0x48);
   SetVectIntern(TIMERC_OFS/0x10, timerC_isr);
   WrPortI(TCCSR, &TCCSRShadow, 0x01);     // Enable timer C
            qd_init(1);                    //inicializa a quadratura com prioridade de interrupt 1,
            qd_zero(motor);                //Zera o contator de canal (motor)= 1 ou 2
  //------------inicialização dos sonares
   brdInit();
```

```
  zb_io_init(); // Função necessária para utilizar o ADC
  WrPortI(PCDDR, &PCDDRShadow, PCDDRShadow | 0x05);  // 00000101 Configura os Bits 0 e
2 da Porta Paralela C como Saídas
  WrPortI(PEDDR, &PEDDRShadow, 0x03); // 00000011 Configura os Bits 0 e 1 da Porta Paralela
E como Saídas
  WrPortI(ICS1R, NULL, 0x00);  // Start Porta C Bit 1, Stop Porta C Bit 1
   WrPortI(ICS2R, NULL, 0x11);  // 00010001 Start Porta C Bit 7, Stop Porta C Bit 7
        // use freq_divider for input capture prescaler
        // (freq_divider-1) runs at 19200*16Hz, multiply by 4 for better range
  WrPortI(TAT8R, NULL, freq_divider*4 - 1); // (TA8 prescaler)
  WrPortI(ICCSR, NULL, 0x0C);         // 00001100 Reseta os contadores
  WrPortI(ICCR, NULL, 0x00);          // Operação Normal; Sem a utilização de interrupções
  WrPortI(ICT1R, NULL, 0x56); // 01010110 Start Borda de Subida; Stop Borda de Descida
  WrPortI(ICT2R, NULL, 0x56);  // 01010110 Start Borda de Subida; Stop Borda de Descida
  pulse_width = 0;        //só está inicializando com 0, depois vai atribuir valores
  freq = pwm_init(FREQ);       // Define a frequencia do PWM
  pwm_options = PWM_USEPORTC;  // Seleciona a Porta que será utilizada para a função PWM
  pwm_options |= PWM_SPREAD;  // Seleciona o modo do PWM

 dir = 0;
 direcao = 0;
 conjunto1 = 1;
 conjunto2 = 0;
 escrita = 1;
 leitura = 0;
 Pvel=0    ;
 motor=2   ; //seleciona o motor 1 ou 2
 reading = 0;
 PWMatual=0;
 Uatual=0;
 Conversao= 0.5789;
 Velocidade=7.5;
 Incremento=Conversao*Velocidade;
 Uatual=0 ;
```

```
//*****Programação dos Sonares*****

        while(1)
  {
  costate{
   waitfor(DelayMs(10000));

          }
   }

 }

    costate {
                        if ( !kbhit() ) abort; // key been pressed?
        { caracter = getchar();
        if(isspace(caracter)) abort;
            {
            waitfor( DelayMs(100))     ;
                                                    Pvel=1              ;
            waitfor( DelayMs(2000))     ;
             Pvel=0                 ;
            waitfor( DelayMs(1000000))     ;
            }
                }




        }

   //Seta a direcao

    costate
    { //Envia pulsos para os sonares
        if(conjunto1 && escrita)
```

```
    {
       BitWrPortI(PCDR, &PCDRShadow, 0, 2);
      BitWrPortI(PCDR, &PCDRShadow, 1, 0);
      waitfor(DelayMs(1));
      BitWrPortI(PCDR, &PCDRShadow, 0, 0);
      leitura = 1;
      escrita = 0;
    }
    if(conjunto2 && escrita)
    {
       BitWrPortI(PCDR, &PCDRShadow, 0, 0);
      BitWrPortI(PCDR, &PCDRShadow, 1, 2);
      waitfor(DelayMs(1));
      BitWrPortI(PCDR, &PCDRShadow, 0, 2);
      leitura = 1;
      escrita = 0;
    }
    yield;
  }
  costate
  { // Realiza a leitura dos sonares
       if(conjunto1 && leitura)
    {
       capture_status = RdPortI(ICCSR);
      if(capture_status & 0x10)     // Detectou uma borda de descida no pulso
                                    {
                                          // Leitura do registrador, começando pelo LSB
                                          pulse_width += RdPortI(ICL1R);
                                          pulse_width += RdPortI(ICM1R) * 256L;

                                          // 76.8 = 19200*16*1000/4 (see TA8 setting above)
                                          sonar1 = (pulse_width)/76.8;
                                          pulse_width = 0;
                                          WrPortI(ICCSR, NULL, 0x04); // Zera a contagem
```

```
    leu1 = 1;
                        }
            if(capture_status & 0x40)    // Detectou uma borda de descida no pulso
                        {
                                // Leitura do registrador, começando pelo LSB
                                pulse_width += RdPortI(ICL2R);
                                pulse_width += RdPortI(ICM2R) * 256L;

                                // 76.8 = 19200*16*1000/4 (see TA8 setting above)
                                sonar2 = (pulse_width)/76.8;
                                pulse_width = 0;
                                WrPortI(ICCSR, NULL, 0x08); // Zera a contagem
    leu2 = 1;
                        }
  if(leu1 && leu2)
  {
   leitura = 0;
    escrita = 1;
    conjunto2 = 1;
    conjunto1 = 0;
    leu1 = 0;
    leu2 = 0;
  }
}
if(conjunto2 && leitura)
{
   capture_status = RdPortI(ICCSR);
  if(capture_status & 0x10)    // Detectou uma borda de descida no pulso
                        {
                                // Leitura do registrador, começando pelo LSB
                                pulse_width += RdPortI(ICL1R);
                                pulse_width += RdPortI(ICM1R) * 256L;

                                // 76.8 = 19200*16*1000/4 (see TA8 setting above)
```

```
                                        sonar3 = (pulse_width)/76.8;
                                        pulse_width = 0;
                                        WrPortI(ICCSR, NULL, 0x04); // Zera a contagem
        leu1 = 1;
                                }
                if(capture_status & 0x40)     // Detectou uma borda de descida no pulso
                                {
                                        // Leitura do registrador, começando pelo LSB
                                        pulse_width += RdPortI(ICL2R);
                                        pulse_width += RdPortI(ICM2R) * 256L;

                                        // 76.8 = 19200*16*1000/4 (see TA8 setting above)
                                        sonar4 = (pulse_width)/76.8;
                                        pulse_width = 0;
                                        WrPortI(ICCSR, NULL, 0x08); // Zera a contagem
        leu2 = 1;
                                }
     if(leu1 && leu2)
     {
      leitura = 0;
       escrita = 1;
       conjunto2 = 0;
       conjunto1 = 1;
       leu1 = 0;
       leu2 = 0;
     }
   }
   yield;
}
      costate
{  // Determinando a velocidade e o sentido de rotação dos motores
   if(vel==0)
      {
                pwm_set(0, (int)(0), pwm_options);
```

```
      pwm_set(1, (int)(0), pwm_options);
    }
if(vel==1)
   {
            pwm_set(0, (int)(PWMatual * 1024.0), pwm_options);
     pwm_set(1, (int)(PWMatual * 1024.0), pwm_options);
      }
if(vel==2)
   {
            pwm_set(0, (int)(PWMatual * 1024.0), pwm_options);
     pwm_set(1, (int)(PWMatual * 1024.0), pwm_options);
      }
if(dir==0)
{
  BitWrPortI(PEDR, &PEDRShadow, 0, 0);
  BitWrPortI(PEDR, &PEDRShadow, 1, 1);

}
if(dir==1)
{
  BitWrPortI(PEDR, &PEDRShadow, 1, 0);
  BitWrPortI(PEDR, &PEDRShadow, 0, 1);
}
if(dir==2)
{
  BitWrPortI(PEDR, &PEDRShadow, 0, 0);
  BitWrPortI(PEDR, &PEDRShadow, 0, 1);
}
if(dir==3)
{
  BitWrPortI(PEDR, &PEDRShadow, 1, 0);
  BitWrPortI(PEDR, &PEDRShadow, 1, 1);
}
yield;
```

```
}
costate
{ // Definindo o movimento do robô para diferentes respostas dos sonares
    if(direcao==0)
  {
            if(sonar1 < 1 && sonar3 > 1)
    {
            vel=1;
                    dir=3;
                    waitfor(DelayMs(100));
    }
     if(sonar1 > 1 && sonar3 < 1)
    {
            vel=1;
                    dir=2;
                    waitfor(DelayMs(100));
    }
     if(sonar1 < 1 && sonar3 < 1)
    {
            vel=1;
       dir=2;
       waitfor(DelayMs(100));
       direcao=1;
    }
     if(sonar1 > 1 && sonar3 > 1)
    {
            vel=1;
                    dir=1;
    }
  }
  if(direcao==1)
  {
            if(sonar2 < 1 && sonar4 > 1)
    {
```

```
                vel=Pvel;
                        dir=3;
                        waitfor(DelayMs(100));
        }
         if(sonar2 > 1 && sonar4 < 1)
        {
                vel=Pvel;
                        dir=2;
                        waitfor(DelayMs(100));
        }
         if(sonar2 < 1 && sonar4 < 1)
        {
                vel=Pvel;
           dir=2;
           waitfor(DelayMs(100));
           direcao=0;
        }
         if(sonar2 > 1 && sonar4 > 1)
        {
                vel=Pvel;
                        dir=0;
        }
      }
      yield;
    }
  }
}
```