Igor Matheus Simonelli Bermudes

Interface de Hardware e Software para uma Perna Robótica

Brasil

2015

Igor Matheus Simonelli Bermudes

Interface de Hardware e Software para uma Perna Robótica

Trabalho de Conclusão de Curso apresentado à Coordenadoria do Curso de Engenharia Mecânica da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do título de Bacharel em Engenharia Mecânica.

Universidade Federal do Espirito Santo – UFES

Centro Tecnológico

Departamento de Engenharia Mecânica

Orientador: Antônio Bento Filho

Brasil

2015

Igor Matheus Simonelli Bermudes

Interface de Hardware e Software para uma Perna Robótica

Trabalho de Conclusão de Curso apresentado à Coordenadoria do Curso de Engenharia Mecânica da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do título de Bacharel em Engenharia Mecânica.

Trabalho aprovado. Brasil, 14 de julho de 2015:

Antônio Bento Filho
Orientador

Rafhael Milanezi de Andrade
labGuará/UFES

Flávio Morais de Souza
CVRD/UFES

Victor Hugo Brito Fernandes
labGuará/UFES

Brasil
2015

Este trabalho é dedicado a todos que me ajudaram,
aos que quiseram me ajudar, e aos que pensam que ajudaram,
mas enfim, dedico a todos que eu gosto
e que estão no fundo do meu coração.

Agradecimentos

Quero agradecer, em primeiro lugar, a Deus, pela força e coragem durante toda esta longa caminhada.

À minha família, por sua capacidade de acreditar e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada. Irmã, pelos conselhos e a sabedoria que vc passou para sue irmão mais novo.

A todos os brothers que fizeram dessa jornada mais interessante e divertida. Victor Hugo, Matheus, Brunorio, André, Marcos agradeço a vocês especialmente por levar o nome Zorro a um nível que jamais eu pensaria que fosse chegar.

Agradeço a meu Professor Antônio Bento por me passar um pouco da sua grande sabedoria tanto de engenharia como de vida, por me cobrar sempre o meu melhor e pelos elogios que me fizeram eu gostar ainda mais do que eu faço.

Agradeço a minha namorada pela a paciência e compreensão nessa jornada. E a todos aqueles que de alguma forma estiveram e estão próximos de mim, fazendo esta vida cada vez valer mais apena.

A menos que modifiquemos a nossa maneira de pensar,
não seremos capazes de resolver os problemas causados
pela forma como nos acostumamos a ver o mundo".
(Albert Einstein)

"Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito.
(Bíblia Sagrada, Romanos 12, 2)

Resumo

O presente trabalho apresenta a implementação de uma plataforma de hardware e software para o acionamento de uma perna robótica com quatro graus de liberdade, acionada por moterredutores DC. A eletrônica é um Arduino interligado aos motores DC através de dois circuitos de ponte H para acionamento e aquisição de dados de corrente dos motores e aos potenciômetros acoplados aos redutores para aquisição de sinal de posição angular. Foi implementado o software de controle, aquisição de dados e comunicação no Arduino e as interfaces gráfica e de entrada de dados em MATLAB. Resultados obtidos mostram boa reação do controlador PID de posição a um impulso e a uma entrada intermitente em degrau.

Palavras-chaves: perna robótica, junta robótica, robô móvel, Arduino, motor DC, MATLAB, Simulink, aquisição de dados.

Abstract

This work presents the implementation of a hardware and software test bench to drive a four degrees of freedom robotic leg driven by DC gear motors. The eletronic is an Arduino board connected to the DC motors through two double channel H-bridge circuits for driving and current sensing of motors and to pots coupled to the gear for angular position acquisition. A control, data acquisition and communication software runs in Arduino and the graphical interfaces and data entry runs in MATLAB. The results shown the good reaction of the PID position controller to a step input and to a constant amplitude dutty cycle inputs.

Key-words: leg robotics, joint robotics, mobile robot, Arduino, Motor DC, MATLAB, Simulink, data acquisition.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

PWM	Pulse with modulation
M1	Motoredutor 1
M2	Motoredutor 2
I/O	Input / Output
DC	Direct Corrent
AC	Alternating Corrente
e_a	tensão aplicada na armadura
e_b	força contra motriz
θ	deslocamento angular do eixo do motor
T	torque fornecido pelo motor
J	momento de inercia equivalente do motor
b	coeficiente de fricção viscosa
K	constante de torque do motor
K_b	constante de força contra eletromotriz

Sumário

1 Introdução

Os robôs a pernas ágeis inspirados em animais e insetos são uma área multidisciplinar da robótica, que envolve temas de biomecânica, de mecânica, eletrônica, computação, entre outras. E a sua evolução e disseminação para aplicação em atividades militares e civis requer uma intensa interação com outros campos da ciência e tecnologia.

Com esta visão, o presente trabalho foca a implementação de uma interface capaz de viabilizar a prototipagem de pernas robóticas bio-inspiradas, sendo então uma ferramenta para a implementação rápida de gigas de testes para pernas robóticas, novos atuadores e sensores, diferentes estratégias de controle, etc.

2 Fundamentos

2.1 Cinemática

A Figura 1 mostra a configuração dos quatro graus de liberdade da perna e os respectivos sistemas de referência, de acordo com a convenção de Denavit-Hartenberg. A perna foi construída de maneira que os graus de liberdade 0 e 1 tenham a mesma origem, com o comprimento do primeiro elo nulo. Os graus de liberdade são: rolagem e mergulho da junta 0, e rolagem das juntas 1, 2 e 3.

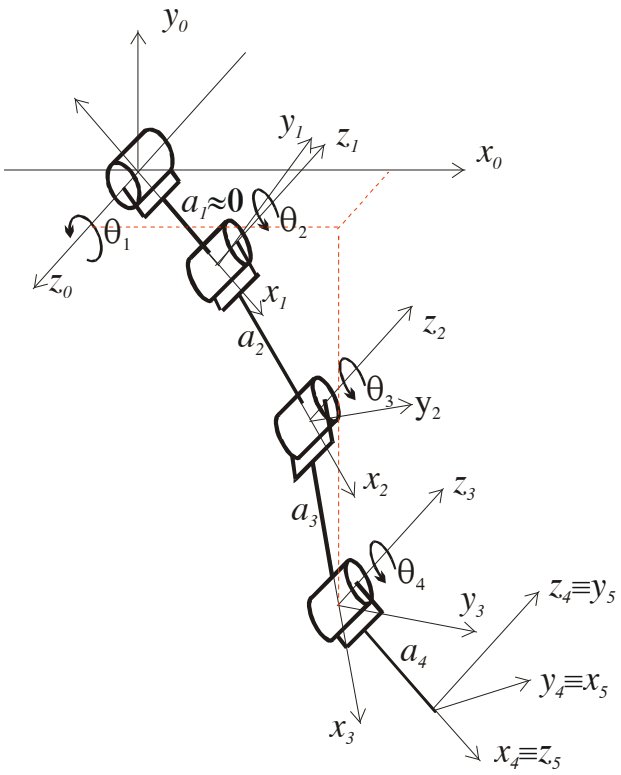


Figura 1 – Perna com referenciais e variáveis de juntas (BENTO; ANDRADE, 2014)

A configuração com um grau de liberdade de rolagem e um de mergulho na junta 0, possibilitam o jogo lateral do quadril observado nos animais quadrúpedes, realizando o balanço que desloca o CG na direção da perna que está apoiada.

2.1.1 Cinemática direta

Utilizando-se a convenção de Denavit Hartenberg, pode-se obter as matrizes das transformacções homogêneas ${A_i}^{i-1}$, para uma perna com 4 graus de liberdade, mostrada na Figura 1, conforme a seguir:

Tabela 1 – Parâmetros de Denavit-Hatenberg (SPONG; HUTCHINSON; VIDYASAGAR, 2006) para a perna da Figura 1

$link$	a_i	d_i	α_i	θ_i
1	0	0	$\frac{\pi}{2}$	θ_1
2	a_2	0	0	θ_2
3	a_3	0	0	θ_3
4	a_4	0	0	θ_4
5	0	0	$\frac{\pi}{2}$	$-\frac{\pi}{2}$

$$\mathbf{A_1^0} = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

$$\mathbf{A_2^1} = \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$$\mathbf{A_3^2} = \begin{bmatrix} c_3 & -s_3 & 0 & a_3c_3 \\ s_3 & c_3 & 0 & a_3s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

$$\mathbf{A_4^3} = \begin{bmatrix} c_4 & -s_4 & 0 & a_3c_4 \\ s_4 & c_4 & 0 & a_4s_4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

$$\mathbf{A_5^4} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

sendo que a matriz A_5^4, Equação 2.5 representa apenas uma rotação fixa entre os sistemas de referência 5 e 4 Figura 1 e $s_i = sen(\theta_i)$, $c_i = cos(\theta_i)$.

2.2 Motor DC

O Motor DC é usado em situações onde o sistema de controle é exigida uma quantidade moderada de potencia no eixo. Os motores DC possuem campos exitados

separadamente. São controlados por armadura com campo fixo ou com controlados por campo com corrente de armadura fixa (OGATA, 2003). No trabalho foi empregado um motor DC controlado por amadura e com o campo fixo de imã permanente indicado na Figura 2

2.2.1 Modelagem

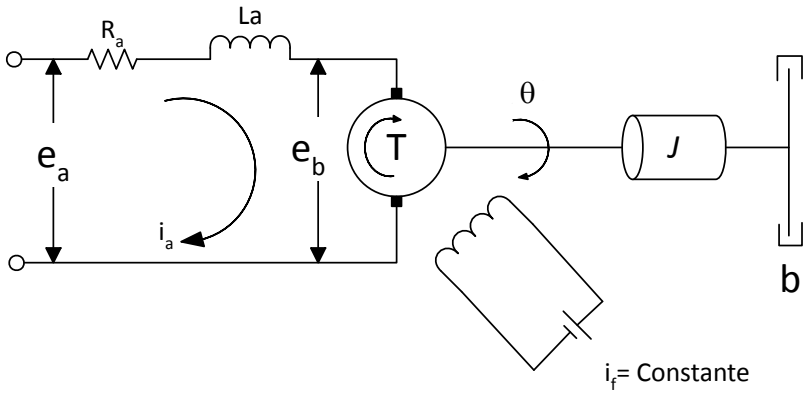


Figura 2 – Esquemático de um Motor DC controlado por armadura

R_a= resistência do enrolamento da armadura, ohms
L_a= indutância do enrolamento da armadura, henrys
i_a= corrente de campo, ampéres
e_a= tensão aplicada na armadura, volts
e_b= força contra motriz, volts
θ= deslocamento angular do eixo do motor, radianos
T= torque fornecido pelo motor, N.m
J= momento de inercia equivalente do motor, $kg \times m^2$
b= coeficiente de fricção viscosa, $kg \times /rad/s$
K= constante de torque do motor
K_b= constante de força contra eletromotriz

A partir do controle de entrada de tensão e_a as equações de causa-efeito do motor:

$$L_a \frac{di_a}{dt} + R_a i_a + e_b = e_a \tag{2.6a}$$

$$T = K i_a \tag{2.6b}$$

$$e_b = K_b \frac{d\theta}{dt} \tag{2.6c}$$

$$J \frac{d^2\theta}{dt^2} + b \frac{d\theta}{dt} = T \tag{2.6d}$$

Supondo todas as condições iniciais nulas e considerando as transformadas de Laplace das Equação 2.6 e $\Theta(S)$ como saída, obtemos a função de transferência do sistema

que é dada por

$$\frac{\Theta(S)}{\dot{E}_a(s)} = \frac{K}{s[L_aJs^2 + (L_ab + R_aJ)s - R_ab + KK_b]} \tag{2.7}$$

Montando o diagrama de blocos do sistema obtemos a Figura 3

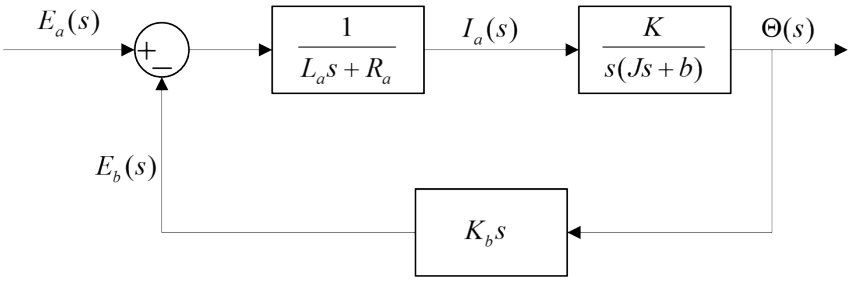


Figura 3 – Diagrama de blocos motor DC

A indutância L_a no circuito de armadura normalmente é pequena e pode ser desprezada. Se L_a for desprezada a função de transferência dada pela Equação 2.7 reduz a

$$\frac{\Theta(S)}{\dot{E}_a(s)} = \frac{K_m}{s[T_ms + 1]} \tag{2.8}$$

onde

$K_m = K/(R_ab - KK_b)$ = constante de ganho do motor

$T_m = R_aJ/(R_ab + KK_b)$ = constante de tempo do motor

Nas Eqs. 2.7 e 2.8 pode ser visualizado que contem o termo 1/s multiplicando a equação. Portanto este sistema possui uma propriedade de integração. E na Equação 2.8 note que a constante de tempo do motor é menor para um R_a menor e J menor. Então com J pequeno, conforme a resistência R_a é reduzida, a constante de tempo do motor se aproxima de zero, e o motor passa atuar como um integrador ideal.(OGATA, 2003)

2.2.2 PWM

Pulse-width modulation (PWM) ou Modulação de Largura de Pulso é usado principalmente para permitir o controle de potência fornecido a dispositivos elétricos. O PWM comuta entre desligado e ligado em forma de onda quadrada a uma frequência fixa(dependendo da aplicação pode exigir frequências muito altas), e o tempo ligado divido pelo tempo total do ciclo denominada-se *duty cicle* ou ciclo de trabalho que varia entre 0 a 100%, no caso do Arduino o ciclo de trabalho simula uma tesão proporcional que varia entre 0 e 5 V, como mostra a Figura 4

Então em um ciclo de trabalho de 50% temos que o Arduino manda 5V na metade do tempo e fica desligado a outra metade, como o a frequência que isso ocorre

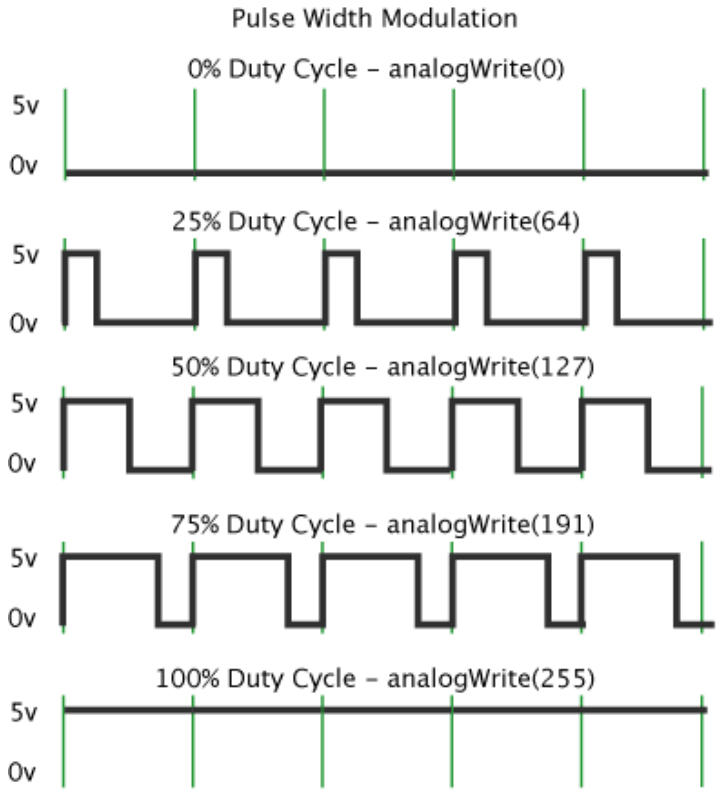


Figura 4 – Ciclo de trabalho PWM Fonte: <http://www.arduino.cc/en/Tutorial/PWM>

é alta(no caso do Arduino 500Hz) pode-se dizer que na saída adquirimos uma tensão média de 2,5V e assim proporcionalmente para o resto dos ciclos. Este principio é utilizado justamente para controles digitais que não conseguem variar a potência de trabalho analogicamente, modulando-se o sinal podemos controlar esses sistemas,permitindo a estes emular características analógicas, conforme exibido na Figura 5.

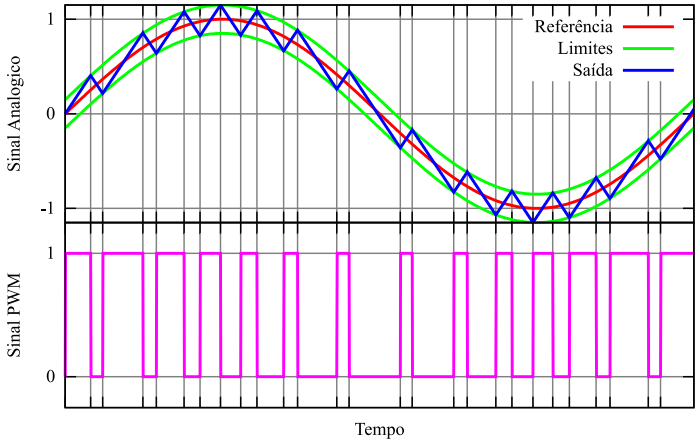


Figura 5 – Sinal analógico gerado por PWM

3 Especificações

3.1 Perna Guara

3.1.1 Motor e redução

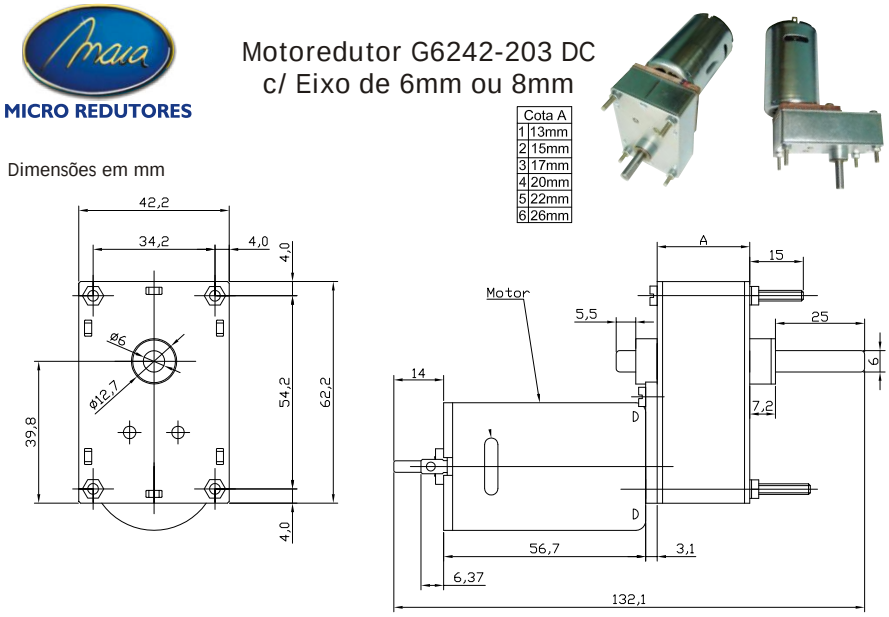


Figura 6 – Motor DC com cota A de 20 mm <http://www.maia.ind.br/>

O motorredutor usado nas pernas do GUARÁ é o G6242-203, que tem as dimensões mostradas na figura 6, e as características são retiradas da tabela no Anexo B. Que tem como propriedade a redução de 57,244:1, e faixa de operação nominal de 24V. Neste trabalho foi utilizado a tensão de 12V pelo limite da fontes existentes no laboratório.

3.1.2 Características

O objeto de estudo é uma perna antropomórfica, com 3 segmentos coxa, canela e pé. É capaz de realizar os movimentos de flexão-extensão e adução-abdução do quadril e de flexão-extensão do joelho e tornozelo, totalizando 4 graus de liberdade acionados por motorredutores DC, cujas características estão mostradas nas Tabelas 2 e 3.

Para o controle da perna foram usados somente as juntas 1 (Coxa) e 2(Canela) da tabela 3 por causa do maior curso que elas proporcionam, favorecendo a aquisição e controle dos dados obtidos.

Tabela 2 – Dimensões e peso (BENTO, 2007)

Link superiror da perna	150 mm
Link inferiror da perna	150 mm
Altura do tornozelo	50 mm
Massa de uma perna	4 kg

Tabela 3 – Relações de transmissão mecânica (BENTO, 2007)

Junta	Engrenagem		Relação de transmissão mecânica	Relação de transmissão final
	Motriz	Acionada		
0	19	120	0,158333	0,154773
1	19	120	0,158333	0,154773
2	19	108	0,175925	0,171970
3	60	120	0,500000	0,488758

3.2 Hardware e Software de Acionamento

O hardware de controle da perna mostrado na Figura 7 foi implementado com um Arduino Mega, que dispões de saídas PWM e entradas analógicas suficientes para o acionamento e aquisição de dados da perna.

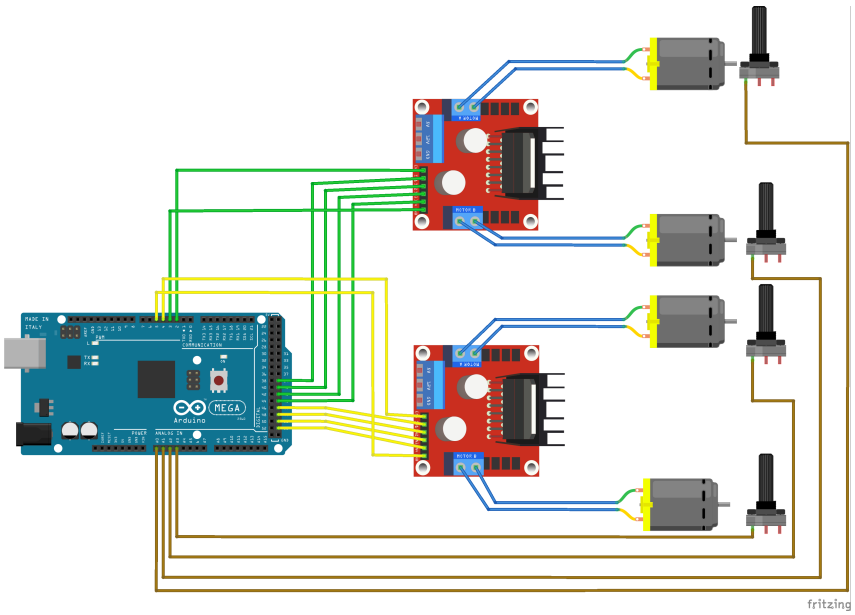


Figura 7 – Montagem do circuito do GUARÁ

Para o fim deste trabalho, foi utilizado somente o circuito verde na montagem da figura 7. Como a segunda parte deste circuito não influencia o controle e modelagem do robô, teve como prioridade a avaliação do Arduino e MATLAB no desenvolvimento da perna robótica. Assim podendo ser adicionado posteriormente o restante do circuito para a simulação de toda a perna.

3.2.1 Arduino

Arduino é uma plataforma open-source de prototipagem eletrônica, projetada com um microcontrolador Atmel AVR com suporte de I/O (entrada/saída) embutido, em um ambiente de desenvolvimento simples e padrão, que é essencialmente C/C++.

Arduino pode ser utilizado para desenvolver objetos interativos, tendo entradas a partir de uma variedade de sensores ou interruptores, e controle de uma variedade de luzes, motores e outras saídas físicas. Projetos Arduino podem ser stand-alone, ou podem se comunicar-se com o software em execução no computador (por exemplo, Flash, Processing, Max MSP, MATLAB). As placas podem ser montadas à mão ou compradas pré montadas, o IDE(Ambiente Integrado de Desenvolvimento) de código aberto pode ser baixado gratuitamente.(ARDUINO. . . , a)

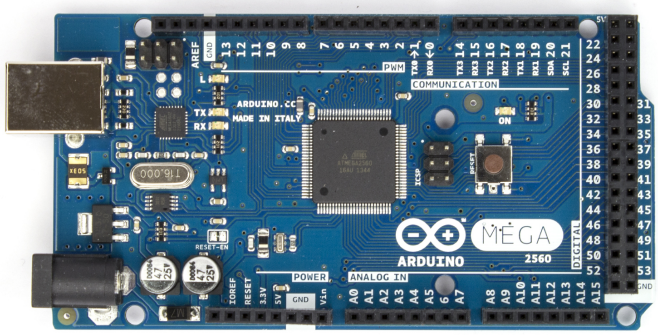


Figura 8 – Arduino Mega2560 <http://www.arduino.cc/en/Main/ArduinoBoardMega2560>

A placa usada foi um Arduino mega 2560 (Figura 8) baseado no ATmega2560. Esta possui 54 pinos digitais de entrada/saída (dos quais 15 podem ser usados como saídas PWM), 16 entradas analógicas, 4 UARTs (portas seriais de hardware), um oscilador de cristal(16 MHz), uma conexão USB, um conector de energia, um cabeçalho ICSP, e um botão de reset. Esta placa contém tudo o que é necessário para apoiar o microcontrolador. Basta conectá-lo a um computador com um cabo USB ou ligá-lo com um adaptador AC-to-DC ou bateria para começar.(ARDUINO. . . , b)

3.2.2 Ponte H

Para realizar o chaveamento liga-desliga na frequência elevada do PWM, normalmente utiliza-se um circuito em ponte H, implementado em uma PCI (Placa de Circuito Impresso) de baixo custo para aplicações em Arduíno.

Foi usado a ponte H L298n com o circuito (Figura 9) para controle dos motores. Onde a ponte H é um circuito de Eletrônica de potência do tipo chopper de classe E (um

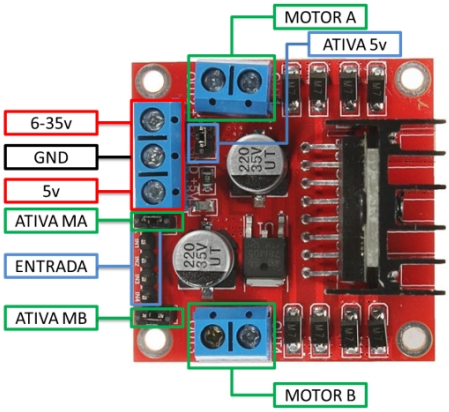


Figura 9 – Placa L298n

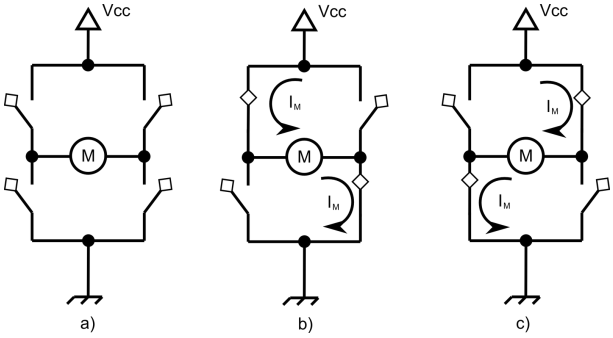


Figura 10 – Funcionamento da ponte H - *a)Motor desligado; b)Motor ligado no sentido horário; b)Motor ligado sentido anti-horário*

chopper classe E converte uma fonte fixa de corrente continua em uma tensão de corrente continua variável abrindo e fechando diversas vezes), e, portanto, pode determinar o sentido da corrente, a polaridade da tensão e a tensão em um dado sistema ou componente.

Seu funcionamento dá-se pelo chaveamento de componentes eletrônicos usualmente utilizando do método de PWM para determinar além da polaridade, o módulo da tensão em um dado ponto de um circuito. Tem como principal função o controle de velocidade e sentido de motores DC escovados, podendo também ser usado para controle da saída de um gerador DC ou como inversor monofásico. O termo Ponte H, é derivado da representação gráfica típica deste circuito (Figura 10).

3.2.3 Interface

A interface utilizado foi o MATALB, que é um software interativo de alta performance voltado para o cálculo numérico. O MATLAB integra análise numérica, cálculo com matrizes, processamento de sinais e construção de gráficos em ambiente fácil de usar onde problemas e soluções são expressos somente como eles são escritos matematicamente, ao contrário da programação tradicional. O MATLAB é um sistema interativo cujo elemento

básico de informação é uma matriz que não requer dimensionamento. Esse sistema permite a resolução de muitos problemas numéricos em apenas uma fração do tempo que se gastaria para escrever um programa semelhante em linguagem Fortran, Basic ou C. Além disso, as soluções dos problemas são expressas no MATLAB quase exatamente como elas são escritas matematicamente.

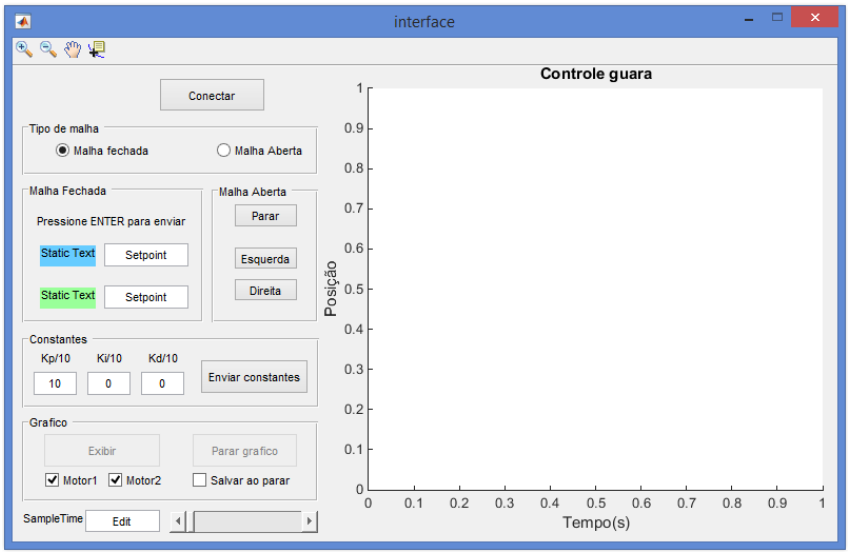


Figura 11 – Interface de controle e aquisição de dados

A Figura 11 mostra alguns parâmetros que foram ajustados para ajudar no controle e aquisição de dados do Arduino.

Conectar

Ao pressionar ele gera uma comunicação serial com o Arduino para envio e recebimentos de dados via serial(porta USB).

Tipo de malha

Configura se a planta tem realimentação (Malha fechada) ou fica sem realimentação (Malha Aberta).

Malha Fechada

Envia o valor do setpoint para o Arduino para que possa controlar o robô; os campos coloridos informam em qual posição o robô está.

Malha Aberta

Envia o sinal PWM (0 a 255) para o motor; esse valores podem ser modificados colocando um valor na caixa de setpoint e apertando esquerda ou direita.

Constantes

Envia um sinal para modificar as contantes do PID dentro do Arduino.

Gráfico

Inicia a aquisição de dados mostrando a posição e o PWM do Motor1 e Motor2; Ao pressionar *Parar gráfico* ele para a aquisição podendo ou não salvar os valores para posteriores comparações.

4 Desenvolvimento

4.1 Hardware

Para o controle da perna foram usados os motores do link 0 (Coxa) e do link 2 (Canela) chamados neste trabalho de M1 e M2 respectivamente. Mostrados na Figura 12.

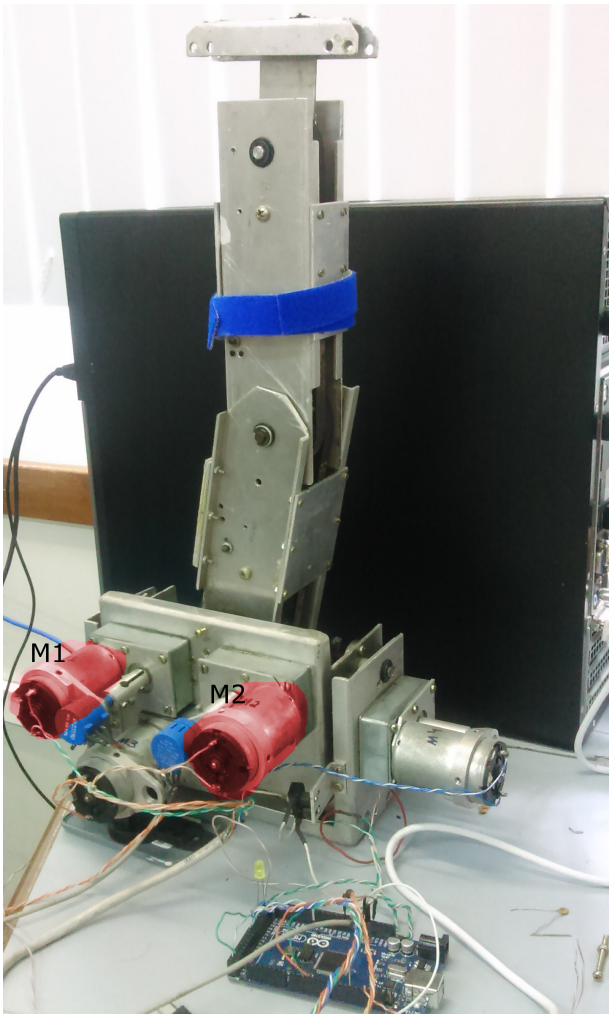


Figura 12 – Perna GURRÀ desacoplada do robô e montado no circuito do Arduino com a identificação dos motores usados

Foram feitos ensaios com o motorredutor desconectado da perna e utilizando mesmo circuito da Figura 7 porém com as ligações somente para um motor, podendo realizar uma maior quantidade de voltas e podendo fazer melhores ajustes no controlador.

Então para o controle do motor é necessário colocar a planta em malha fechada e adicionar um PID como mostrado na Figura 13 . A Equação 4.1 mostra um PID paralelo em função do tempo.

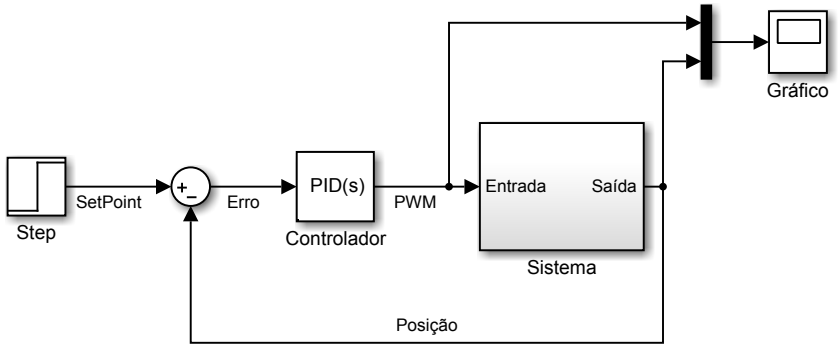


Figura 13 – Planta em malha fechada

$$Output = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt} \tag{4.1}$$
$$onde : e = Setpoint - Input$$

Fazendo transformações na Equação 4.1 para implementação em controladores digitais C/C++, resultou no código mostrado na Figura 14.

O código da Figura 14 precisa de algumas modificações para ser implementado no Arduino, pois este não leva em conta alguns fatores como um tempo fixo de controle, limites de saída, limite do termo integral e ajustes de reset Windup. Assim, foi utilizada a biblioteca do PID proposta no próprio site do Arduino que leva em conta todas essa modificações e com explicações de cada uma delas feita no código. (PIDLIBRARY, 2015)

4.2 Aquisição de dados

Com o motor funcionando e sendo controlado foi feito a aquisição de dados para a visualização. Em um primeiro momento fora escolhida a melhor maneira para adquirir os dados sem que atrasa-se o controle do motor, foi utilizando a função Serial.write do Arduino que teve o menor tempo de envio (tempo médio de envio de 4 valores foi de 8 milissegundos), e no MATLAB para receber esse valores usou a função fread(), com isso criou-se um loop onde a aquisição passou ser no mesmo tempo em que o Arduino envia os valores para a porta serial.

Os tempos tiveram que ser medidos para ter certeza que eles estariam contantes para não prejudicar na hora de avaliar os valores. A Figura 15 foi retirada de um osciloscópio para provar que os valores de sample time e o tempo de aquisição colocados no Arduino estariam corretos. Também foi adquirido o tempo de loop e tempo gasto para a função PID ser executada mostrados na Figura 16, mostrando que o PID tem grande influência no tempo de cada loop, e para o caso de dois motores poderia ter um *sample time* de até 700 microssegundos sem a aquisição de dados.

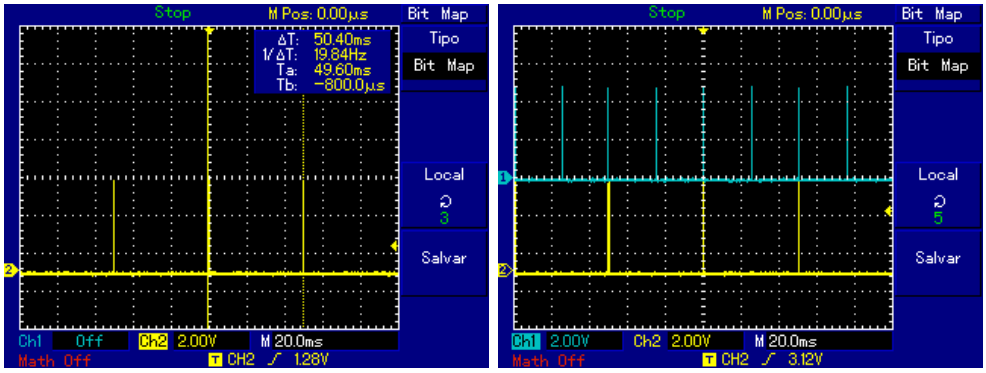
```
/*working variables*/
unsigned long lastTime;
double Input, Output, Setpoint;
double errSum, lastErr;
double kp, ki, kd;
void Compute()
{
/*How long since we last calculated*/
unsigned long now = millis();
double timeChange = (double)(now - lastTime);
/*Compute all the working error variables*/
double error = Setpoint - Input;
errSum += (error * timeChange);
double dErr = (error - lastErr) / timeChange;
/*Compute PID Output*/
Output = kp * error + ki * errSum + kd * dErr;
/*Remember some variables for next time*/
lastErr = error;
lastTime = now;
}
void SetTunings(double Kp, double Ki, double Kd)
{
kp = Kp;
ki = Ki;
kd = Kd;
}
```

Figura 14 – Código em C do PID.

Para que fosse possível enviar valores enquanto acontecesse a aquisição foi necessário criar uma interface gráfica. Com essa interface percebeu-se que poderia controlar diversos outros parâmetros sem a necessidade de compilar e reenviar um novo código a cada mudança neste. A interface final com a aquisição de dados dos motores M1, M2 e ajustes de vários parâmetros foi mostrada na Figura 11

4.3 Interface de comunicação e aquisição de dados

Depois da interface pronta o motorredutor foi conectado na perna robótica e vários testes foram executados para verificar problemas e erros de programação. Foi notado que existia uma região que do PWM não movia a perna(Região morta) pela adição de inércia ao sistema que não existia com o motor desconectado, para correção desse erro foi feito um



(a) (b)

Figura 15 – figura (a) mostrando que o tempo de sample time do PID é constante, a figura (b) mostrando que aquisição de dados é a metade do tempo do sample time.

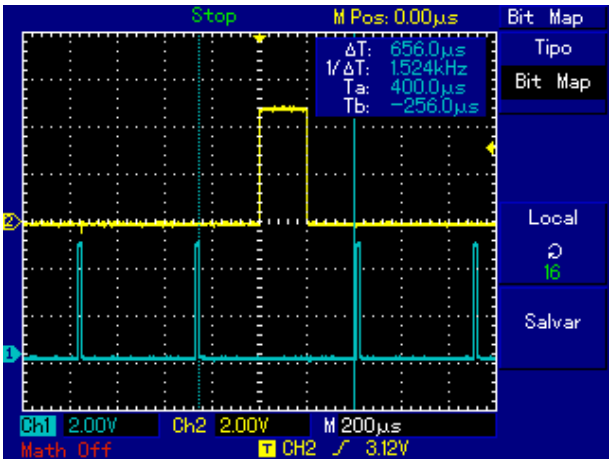


Figura 16 – o grafico azul é o tempo de execução do loop do arduino e o grafico amarelo é o tempo de execução dos PIDs M1 e M2

mapeamento desse valores fazendo com que o valor inicial do PWM estivesse fora dessa região tornando o sistema muito mais responsivo, o gráfico na figura 17 mostra como foi feito essa modificação.

Com o motor M1 configurado também foram feitas modificações no código do Arduino, transformando tudo em funções para que fosse fácil a adição de novos motores ao controle, assim foi adicionado o motor M2 ao sistema e o código do controle destes dois motores são mostrados no *Anexo A*.

Tendo M1 e M2 adicionado ao código foi preciso encontrar os ganhos do PID. Uma das formas é encontrar a função de transferência que define a planta real. Para isso a planta(GUARÁ) tem que ser simulado em malha aberta Figura 18, modificando-se mais uma vez o código para que ele pudesse alternar entre malha aberta e malha fechada, também foi alterado a interface para salvar os valores de saída (posição) e de entrada (PWM).

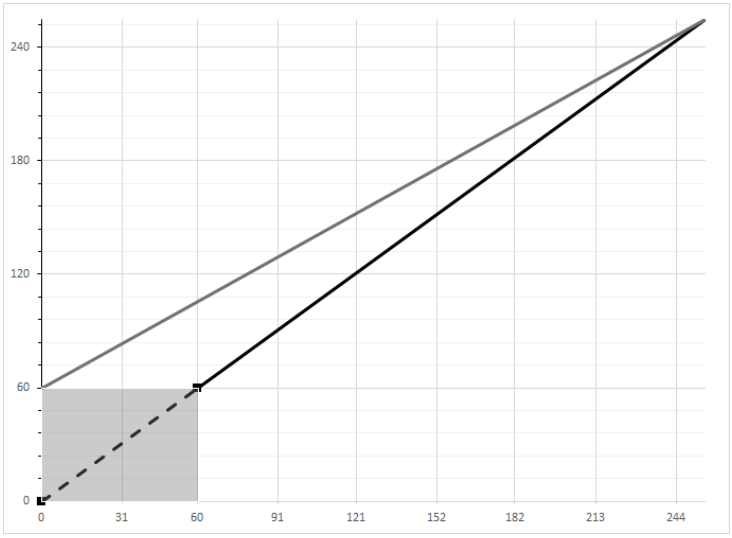


Figura 17 – Correção da região morta do motor d.c.; Região morta em cinza, valores de PWM(0 a 255) na saida do PID no eixo X, valores de PWM (0 a 255) enviados para o motor no eixo Y

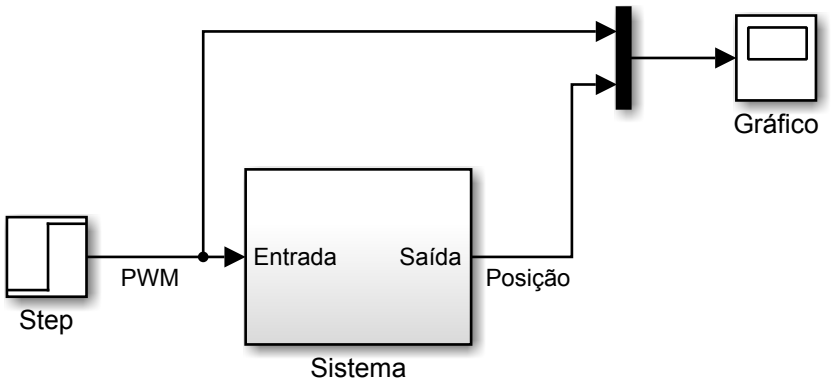
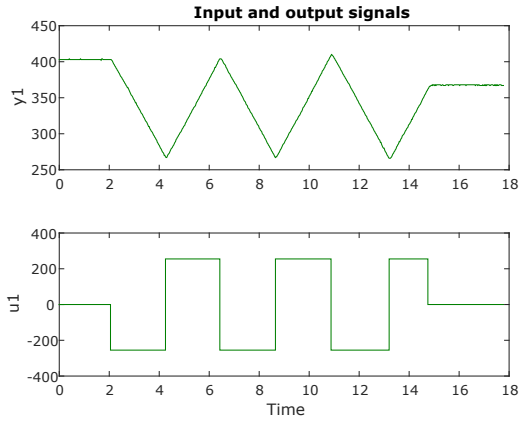


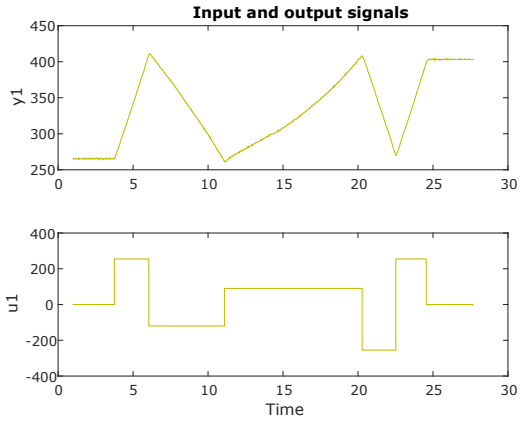
Figura 18 – Planta em malha aberta

5 Resultados

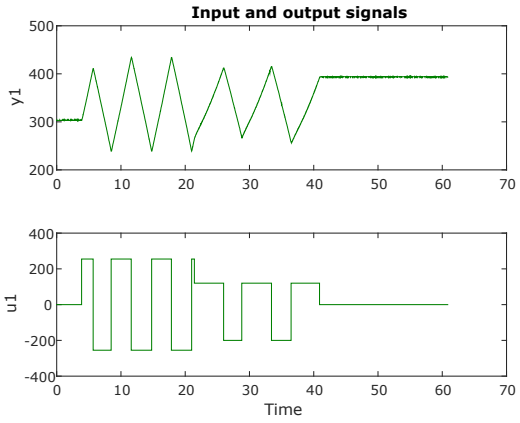
Os resultados obtidos em malha aberta permitiram fazer algumas comparações e testes. Para encontrar a função de transferência foram feitos 3 ensaios variando a entrada(PWM) da malha aberta e obtendo a saída(Posição), mostrados na Figura 19.



(a) Ensaio 1



(b) Ensaio 2



(c) Ensaio 3

Figura 19 – Ensaios em malha aberta da perna do GUARÁ onde y1 é a saída(posição) e u1 é a entrada (PWM) com a taxa de amostragem de 25 milissegundos

Agora utilizando os dados em conjunto com a função de *system identification* Figura 20 do MATLAB pode-se estimar uma função de transferência que mais se aproxime da planta (Perna do GUARÁ).

Estimando a função de transferência com o *"Process Model"* Figura 21, é necessário fazer diversas combinações de ensaios e diversos modelos de funções de transferência para obter a melhor aproximação da função, em porcentagem, em relação à planta(GUARÁ). Alguns dos melhores resultados obtidos estão mostrados na Figura 22

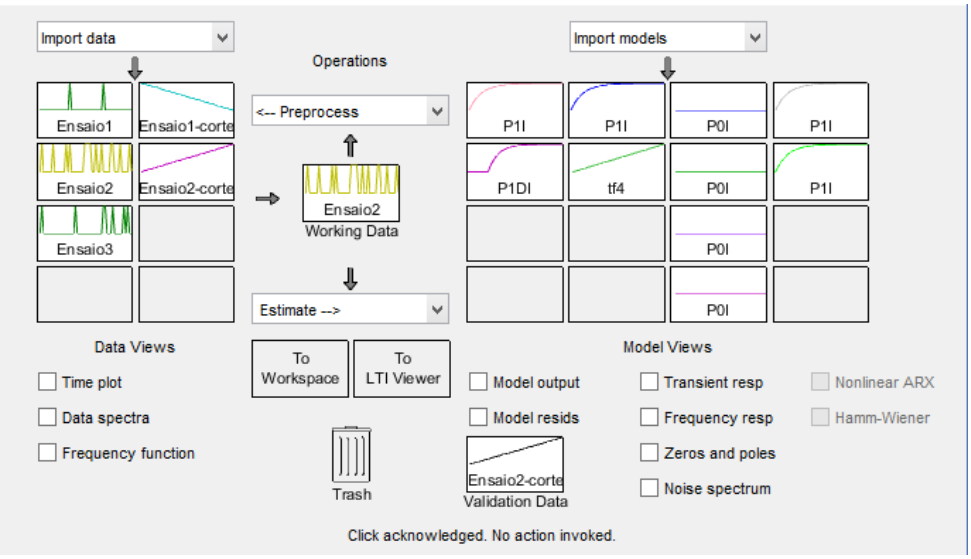


Figura 20 – Janela da função System Identification

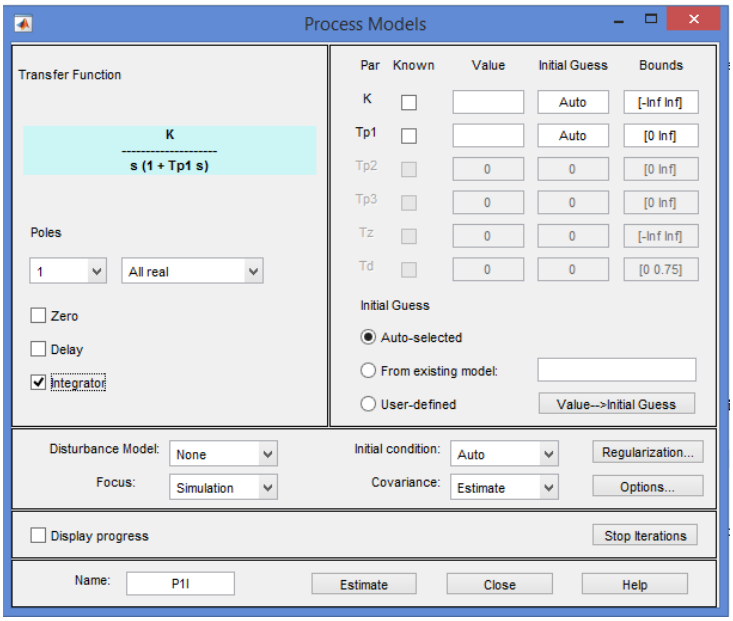


Figura 21 – Janela para estimativa de função de transferência

Percebe-se que todas as funções possuem uma característica muito parecida com a planta real, pois, todas elas foram estimadas utilizando um integrador multiplicando a função como foi mostrado na modelagem do motor DC. Então isso afirma que a função de transferência estimada, esta de acordo com o que foi mostrado na modelagem do motor do sistema. A função que teve melhor resultado foi utilizando o integrador com 1 polo real, que esta mostrada na equação 5.1.

$$\frac{\theta(s)}{PWM(s)} = \frac{0,24275}{s(1+0,026256s)} \tag{5.1}$$

Com a função de transferência em mãos é possível fazer os testes para a escolha dos

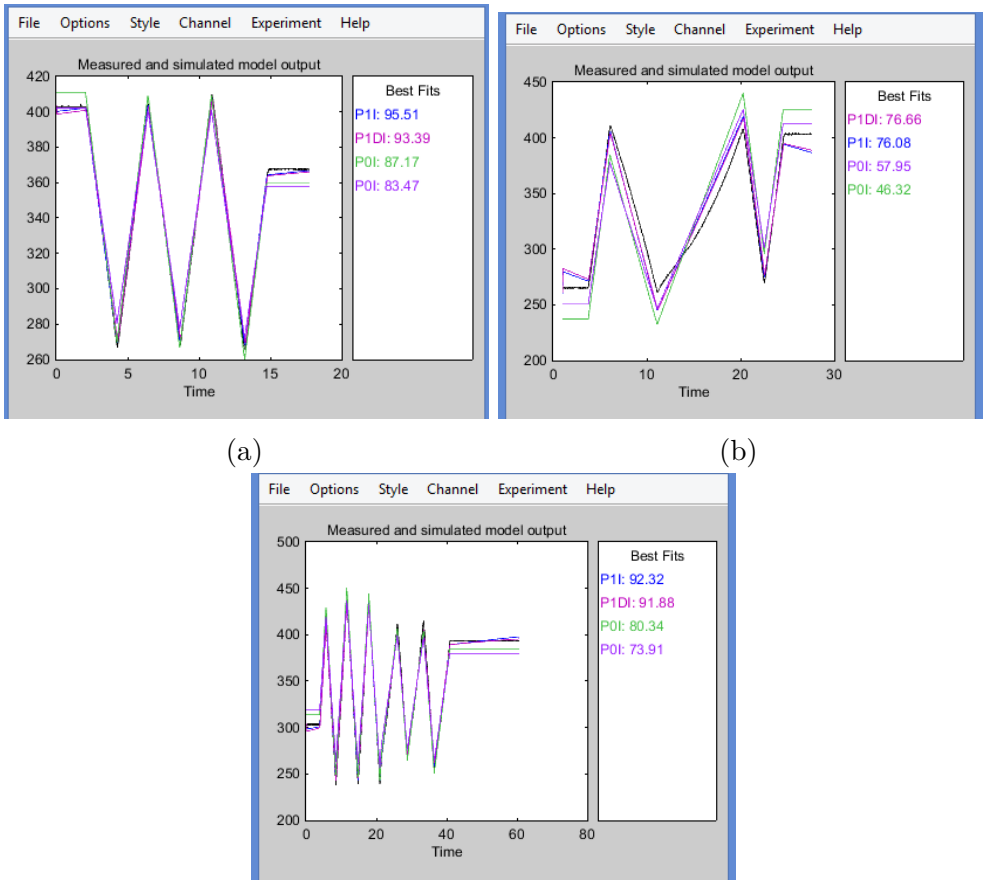


Figura 22 – Aproximações de funções de transferência para cada ensaio em malha aberta

ganhos do controlador. Foi usada a ferramenta PID tunner do MATLAB para procurar encontrar os ganhos, mas notou-se que essa ferramenta não leva em conta os limites de saída do Arduino assim jogando ganhos muito altos. Foi preciso utilizar o Simulink para ter uma aproximação mais real do experimento, também podendo validar a função de transferência encontrada no *System Identification*.

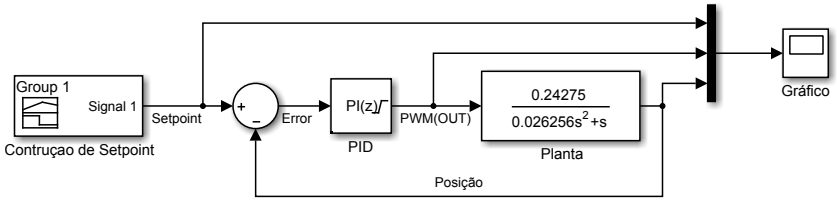


Figura 23 – Planta construída em Simulink

Com a planta (Figura 23) aonde foram colocados os limites do PWM, e utilizando os ganhos adquiridos inicialmente no PID tunner e ajustados de acordo com o Simulink obtemos um PI com $K_P = 27$ e um $K_I = 34$. O gráfico Figura 24 mostra a resposta para mudanças de Setpoint de 30, 20 e 5

Para comprovar o uso do Simulink para a escolha dos ganhos do controlador,

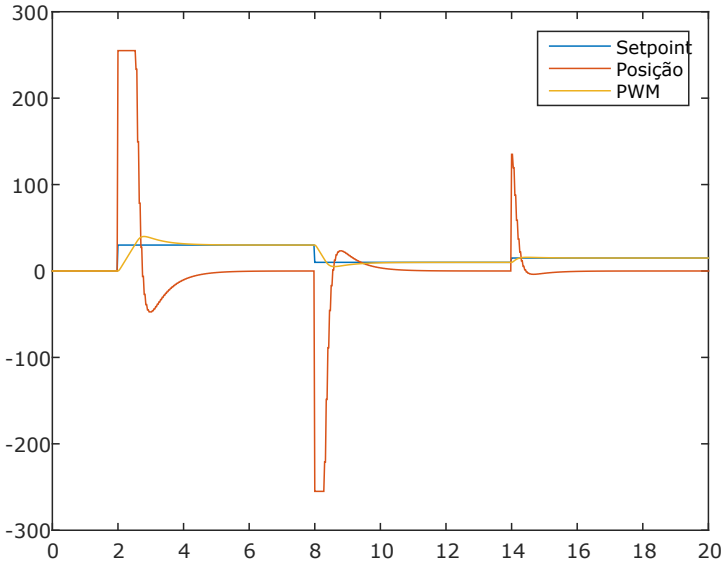
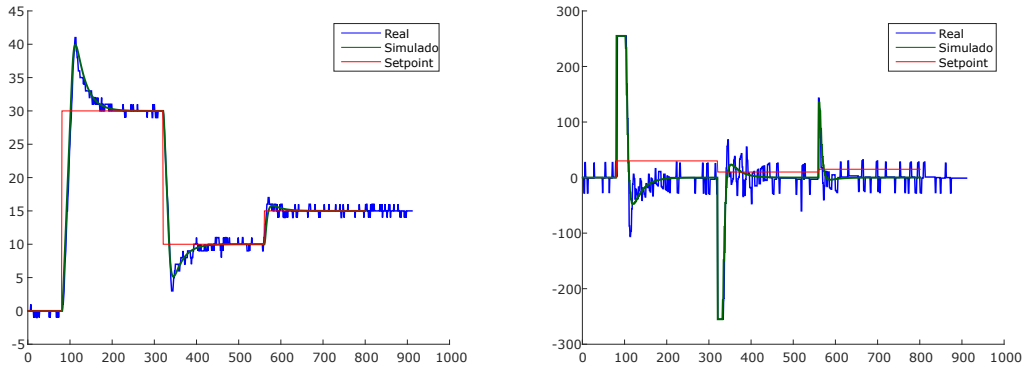


Figura 24 – Simulação da função de transferência no ambiente Simulink

coloca-se as mesmas contantes e os mesmos setpoints no modelo real para comparação dos valores na planta em funcionamento.

Feitas as simulações podemos observar nos gráficos da Figura 25 que os valores gerados no Simulink chegam a ser quase idênticos a aquisição de dados da planta real, a não ser pelo pequeno ruido que é gerado nos potenciômetros, que como podemos perceber esse ruido é multiplicado pelo ganho proporcional do controlador gerando a curva azul de PWM na Figura 25b.

Provado que a função de transferência que encontramos simula o que está acontecendo no real, poderá fazer simulações mais rapidamente e ainda determinar os tempos de andadura do robô não precisando de uma eletrônica montada, alem disso fazer ajustes nos ganhos do motor para melhor resposta do GUARÁ.



(a) Posição (b) PWM

Figura 25 – Gráficos mostrando comparações entre a planta real e a função de transferência estimada

6 Conclusões

Este trabalho mostra que o controle de uma perna robótica com 2 graus de liberdade utilizando o Arduino é possível e com bons resultados, mesmo que a frequência de controle não seja alta.

Que a implementação de uma interface de alto nível é possível aquisitar os dados em tempo real permitindo gigas de testes, emissões de gráficos e manipulações de dados. Alem de permitir o controle e o envio de algumas variáveis e setpoints para o Arduino mesmo com o programa em execução.

Para os resultados obtidos foi possível verificar que ao encontrar um função de transferência via manipulações dos dados pode-se assumir que ela representará o modelo real. Permitindo então, fazer simulações para melhoria de desempenho.

E ao se fazer envios dos ângulos dos graus de liberdade automaticamente via software pode-se simular a andadura de uma perna, sendo possível adicionar e replicar a programação para os outros motores e pernas, fazendo um robô completo andar.

Podem haver uma serie de melhorias para esta trabalho como usar 24V no motor DC para obter uma melhoria na resposta, velocidade e torque deste. Aumentar a tensão no potenciômetro, melhorando assim a precisão no conversor A/D e o uso de baterias para redução de ruídos no sistema.

7 Sugestões para trabalhos futuros

Aplicar um Arduino mega para o controle de duas pernas (8 motores), verificando o tempo de controle. Para certificar que é possível fazer o controle da perna.

Fazer com que a interface envie os setpoints da andadura do robô, e aplicar nas quatro pernas

Implementar sensores de corrente para avaliação de torque para saber quando o robô encontra um obstaculo ou pisa no chão.

Com isso usar a cinemática inversa e o jacobiano fazendo robô andar sozinho.

Referências

ARDUINO - Introduction. <http://www.arduino.cc/en/Guide/Introduction>. (Visitado em 25/05/2015). Citado na página 21.

ARDUINO Board Mega 2560. <http://www.arduino.cc/en/Main/ArduinoBoardMega2560>. (Visitado em 25/05/2015). Citado na página 21.

BENTO, A. F. *Modelagem, Controle de Andadura e Transposição de Obstáculos de Um Robô Quadrúpede com Quatro Graus de liberdade em Cada Perna*. Tese (Doutorado) — Universidade Federal do Espírito Santo, 2007. Citado 2 vezes nas páginas 9 e 20.

BENTO, A. F.; ANDRADE, R. M. A divide and conquer approach for obstacles overcoming of a legged robot. *International Journal of Emerging Technology and Advanced Engineering*, v. 5, n. 4, p. 71–97, 2014. Citado 2 vezes nas páginas 8 e 13.

OGATA, K. *Engenharia de Controle Moderno*. 4. ed. São Paulo: Prentice Hall, 2003. ISBN 85-7054-019-1. Citado 2 vezes nas páginas 15 e 16.

PIDLIBRARY. 2015. <http://playground.arduino.cc/Code/PIDLibrary>. (Visitado on 29/06/2015). Citado na página 26.

SPONG, M. W.; HUTCHINSON, S.; VIDYASAGAR, M. *Robot modeling and control*. [S.l.: s.n.], 2006. ISBN 0471649902. Citado 2 vezes nas páginas 9 e 14.

Anexos

ANEXO A – Programa Arduino

```
#include "PID_v1.h"
double Setpoint,Input,Output;
double SetpointM2,InputM2,OutputM2;
//double SetpointM3,InputM3,OutputM3;
//double SetpointM4,InputM4,OutputM4;
double Mkp=2,Mki=0,Mkd=0;
int Out,OutM2;

PID M1(&Input,&Output,&Setpoint,Mkp,Mki,Mkd,DIRECT);
PID M2(&InputM2,&OutputM2,&SetpointM2,Mkp,Mki,Mkd,DIRECT);
//PID M3(&InputM3,&OutputM3,&SetpointM3,Mkp,Mki,Mkd,DIRECT);
//PID M4(&InputM4,&OutputM4,&SetpointM4,Mkp,Mki,Mkd,DIRECT);
//-----------------------------------------------
//Motor1
#define M1A 39
#define M1B 41
#define M1PWM 2
#define M1P A0
//Motor2
#define M2A 43
#define M2B 45
#define M2PWM 3
#define M2P A1

int media,mediaM2;
boolean enviar;
int botao1 = 22, botao2 = 23;
int ini=70, iniM2 = 80;//60  75
double cont=0,contM2=0;//variavel para tirar media

int sample = 50;
unsigned long now = millis();
unsigned long lastTime;
unsigned long timeChange;

void setup(){
```

```
enviar = false;
pinMode(botao1,INPUT_PULLUP);//botao que fica na perna
pinMode(botao2,INPUT_PULLUP);//botao que fica na perna

//Pontos de controle- verificalçao em ociloscopio
pinMode(11,OUTPUT);
pinMode(9,OUTPUT);

//Motor 1
M1.SetMode(AUTOMATIC);
M1.SetOutputLimits(-255,255);
Setpoint = 350;
pinMode(M1A,OUTPUT);
pinMode(M1B,OUTPUT);

//Motor 2
M2.SetMode(AUTOMATIC);
M2.SetOutputLimits(-255,255);
SetpointM2 = 500;
pinMode(M2A,OUTPUT);
pinMode(M2B,OUTPUT);

Serial.begin(9600);
}

void loop() {
digitalWrite(9,LOW);

Input = analogRead(M1P);
InputM2 = analogRead(M2P);

bool a = M1.Compute();
M2.Compute();

if (!a){
if (digitalRead(botao1) == HIGH) Output = 255;
if (digitalRead(botao2) == HIGH) Output = -255;
}

Direction(M1A,M1B,Output);
Direction(M2A,M2B,OutputM2);
```

```
Out = map(abs(Output), 4, 255, ini, 255); //ini=60
OutM2 = map(abs(OutputM2), 4, 255, iniM2, 255); //ini=75

if (Output > -4 & Output < 4) Out = 0;
if (OutputM2 > -4 & OutputM2 < 4) OutM2 = 0;

analogWrite(M1PWM,Out);
analogWrite(M2PWM,OutM2);

now=millis();
timeChange = (now - lastTime);
if (enviar && timeChange>=sample){
EnviarDados(Input);
EnviarDados(Output);
EnviarDados(InputM2);
EnviarDados(OutputM2);
lastTime = now;
}

digitalWrite(9,HIGH);
} //--------END OF LOOP-----------------------------------------

void EnviarDados(int matlab){ //ENVIA 2 BYTES PELA PORTA SERIAL
Serial.write(matlab»8);
Serial.write(matlab);
}//--------END OD ENVIAR DADOS-------------

void Direction(int MA,int MB, double MOut){
if (MOut>0){
digitalWrite(MA,LOW);
digitalWrite(MB,HIGH);
} else {
digitalWrite(MA,HIGH);
digitalWrite(MB,LOW);
}
}//-------END OF DIRECTION ---------------

void serialEvent() {
int  condicao = Serial.read();

switch (condicao){
```

```
case 1: // LER SETPOINT DO MOTOR M1
byte lerSet[1];
Serial.readBytes(lerSet,2);
Setpoint = lerSet[1]*256+lerSet[0];
break;

case 2: //ENVIA VALORES OU NAO PARA PLOTAR O GRAFICO
if (enviar) enviar = false;
else enviar = true;
break;

case 3: // MUDA OS VALORES DAS CONTANTES M1 e M2
byte constantes[2];
Serial.readBytes(constantes,3);
Mkp = double(constantes[0])/10.0;
Mki = double(constantes[1])/10.0;
Mkd = double(constantes[2])/10.0;
//verificar se preciso dessas dua linhas abaixo
M1.SetTunings(Mkp, Mki, Mkd);
M2.SetTunings(Mkp, Mki, Mkd);
break;

case 4: //LER SETPOIN DO MOTOR M2
byte lerSet2[1];
Serial.readBytes(lerSet2,2);
SetpointM2 = lerSet2[1]*256+lerSet2[0];
break;

case 5: //inicializa a malha fechada
ini = 60;
iniM2 = 75;
Setpoint = 350;
SetpointM2 = 500;
M1.SetMode(AUTOMATIC);
M2.SetMode(AUTOMATIC);
break;

case 6: //inicializa a malha aberta
ini = 0;
iniM2 = 0;
M1.SetMode(MANUAL);
M2.SetMode(MANUAL);
```

```
Output = 0;
OutputM2 =0;
break;

case 7:  //planta em malha aberta
byte pwm[1];
Serial.readBytes(pwm,2);
if (pwm[0]==1) Output = pwm[1];
else if(pwm[0]==2)  Output = -1 * pwm[1];
else if (pwm[0]==3){
Output = 0;
OutputM2 =0;
}
break;

case 8:// configura sampletime / Sample time padrao é 100
byte samp[0];
Serial.readBytes(samp,1);
M1.SetSampleTime(samp[0]);
M2.SetSampleTime(samp[0]);
sample = samp[0]/2;
break;

default:
Serial.flush();
}
}//-----------------END OF SERIAL EVENT---------------------------------
```

ANEXO B – Tabela motor DC

Motoredutor G6242-203 DC
c/ Eixo de 6mm ou 8mm

Relações de Transmissão Redutor
G6242-203 DC - 12 VOLTS - Z14

		MOTORES CORRENTE DC					
		MODELO 203 - (CC)					
Redução	N.	6870 12 Vcc		9735 12 Vcc		13575 12 Vcc	
I = X:1	Pares	3600 rpm		7400 rpm		13600 rpm	
		Veloc.	Torque	Veloc.	Torque	Veloc.	Torque
		RPM	Kgf.cm	RPM	Kgf.cm	RPM	Kgf.cm
11,785	2	305	1,1	628	2,0	1154	2,7
20,035	3	180	1,9	369	3,4	679	4,6
33,672	3	107	3,2	220	5,7	404	8
57,244	4	63	5,4	129	9,7	238	
97,315	5	37	9,1	76		140	
163,555	5	22		45		83	
274,884	5	13		27		49	
472,679	7	7,6		16		29	
794,417	7	4,5	EX	9,3	EX	17	EX
1135,16	7	3,2		6,5		12	
2243,96	7	1,6		3,3		6	

Relações de Transmissão Redutor
G6242-203 DC - 24 VOLTS - Z14

MOTORES DE CORRENTE CONTINUA 24 VOLTS - MODELOS 203 E 204															
Redução	N.	37200 24 VDC		42600 24VDC		48540 24 VDC		54450 24 VDC		54315 24 VDC		8010 24 VDC		204/ 24 VDC	
I = X:1	Pares	1600 rpm		3100 rpm		3600 rpm		4400 rpm		6600 rpm		9800 rpm		4100 rpm	
		Veloc.	Torque	Veloc.	Torque	Veloc.	Torque	Veloc.	Torque	Veloc.	Torque	Veloc.	Torque	Veloc.	Torque
		RPM	Kgf.cm	RPM	Kgf.cm	RPM	Kgf.cm	RPM	Kgf.cm	RPM	Kgf.cm	RPM	Kgf.cm	RPM	Kgf.cm
11,785	2	135,8	0,8	263,0	1,5	305,5	1,1	373,4	2,0	560,0	1,6	831,6	2,5	347,9	4,2
12,14	2	131,8	0,8	255,4	1,5	296,5	1,1	362,4	2,0	543,7	1,7	807,2	2,5	337,7	4,3
20,035	3	79,9	1,3	154,7	2,5	179,7	1,9	219,6	3,4	329,4	2,7	489,1	4,2	204,6	7,1
33,672	3	47,5	2,1	92,1	4,2	106,9	3,2	130,7	5,7	196,0	4,6	291,0	7,0	121,8	11,9
57,244	4	28,0	3,6	54,2	7,1	62,9	5,4	76,9	9,6	115,3	7,8	171,2	11,9	71,6	
97,315	5	16,4	6,2	31,9	12,0	37,0	9,2	45,2		67,8	13,3	100,7		42,1	
163,55	5	9,8	10,4	19,0		22,0		26,9		40,4		59,9		25,1	
274,884	5	5,8		11,3		13,1		16,0		24,0		35,7		14,9	
467,304	6	3,4		6,6		7,7		9,4		14,1		21,0		8,8	
785,385	6	2,0		3,9		4,6		5,6		8,4		12,5		5,2	
1335,16	7	1,2	EX	2,3	EX	2,7	EX	3,3	EX	4,9	EX	7,3	EX	3,1	EX
2243,96	7	0,7		1,4		1,6		2,0		2,9		4,4		1,8	

Obs. As velocidades podem ser influenciadas pela carga aplicada em -20%
Ex: Excede o máximo de torque permitido 10kgf.cm

ANEXO C – Código parcial da interface Matlab

Um dos códigos de envio de valores para o Arduino

```
function setpointsend_Callback(hObject, eventdata, handles)

TxText = get(handles.setpointsend,'String');
TxTextNum = str2double(TxText);

if isnan(TxTextNum);
    errordlg('So aceita numeros');
else
if TxTextNum<1024;

        fwrite(handles.serConn, 1);
        fwrite(handles.serConn, TxTextNum, 'uint16'); %envia ate 65535/2

    else
        errordlg('escolha um numero de 0 a 1023');

    end
end
```

Função para a aquisição de dados e plotagem do gráfico.

```
function pushbutton6_Callback(hObject, eventdata, handles)

global grafico

set(hObject, 'Enable', 'off');
set(handles.pushbutton7, 'Enable', 'on');
i=1;
cont=0;
grafico = true;
clear B C A D E
cla(handles.axes1,'reset');%,'reset');
```

```
title(handles.axes1,'Controle GUARÁ');
ylabel(handles.axes1,'Posição');
xlabel(handles.axes1,'Tempo(s)');
hold(handles.axes1,'on');
set(handles.axes1,'xminortick','on')
%xlim(handles.axes1,[1 500]);
byt=8;
%set(handles.axes1,'Ylim',[0,1050]);
pause(2);
fwrite(handles.serConn, 2);
Npontosx =24;
pontos = 500;
tic;
while grafico

  A = fread(handles.serConn,byt); %Recebe o valor do Buffer da Serial
  %Converte 2 valore de 8 bits em um valor inteiro de 16bits
  B(i,1)=bitshift(A(1,1),8,'int16')+A(2,1);
  C(i,1)=bitshift(A(3,1),8,'int16')+A(4,1);
  D(i,1)=bitshift(A(5,1),8,'int16')+A(6,1);
  E(i,1)=bitshift(A(7,1),8,'int16')+A(8,1);

  set(handles.text15,'string',B(i,1));
  set(handles.text16,'string',D(i,1));

  cont = cont + 1;
  tempo(i,1) = toc; % inicia contagem de tempo
  % condiçoes para a plotagem de graficos para aumetar o desempenho do matlab
  if cont == 25
      pontos = int16(get(handles.slider3,'value'));
      limx = linspace(tempo(i-Npontosx,1),tempo(i,1),Npontosx+1);
      cla(handles.axes1);
      if i >= pontos%500

         xlim(handles.axes1,[tempo(i-Npontosx,1) tempo(i,1)]);

         if get(handles.checkbox2,'value')
            plot(handles.axes1,limx,B(i-Npontosx:i),'b');
            plot(handles.axes1,limx,C(i-Npontosx:i),'r');
         end
         if get(handles.checkbox3,'value')
           plot(handles.axes1,limx,D(i-Npontosx:i),'g');
```

```
            plot(handles.axes1,limx,E(i-Npontosx:i),'m');
            end
         Npontosx=pontos-1;
      else
         Npontosx=25+i-1;
         plot(handles.axes1,limx,B,'b');
         plot(handles.axes1,limx,C,'r');
         plot(handles.axes1,limx,D,'g');
         plot(handles.axes1,limx,E,'m');
      end
      cont = 0;
  end

  i = i+1;
  drawnow;
end
% Imprime o grafico inteiro ao parar o gráfico
cla(handles.axes1);
asdf = linspace(tempo(1,1),tempo(i-1,1),i-1);
plot(handles.axes1,asdf,B,'b');
plot(handles.axes1,asdf,C,'r');
plot(handles.axes1,asdf,D,'g');
plot(handles.axes1,asdf,E,'m');


legend(handles.axes1,'ImputM1','OutputM1','ImputM2','OutputM2');

if get(handles.checkbox1,'value') == 1
%-------SALVAR OS ARQUIVOS COM DATA E HORA
t = strrep(datestr(clock,0),':','_');
name=strcat('Pgzorro--',t);
save(name,'B','C','D','E','tempo');
%-------FIM DO PROGRAMA DE SALVAR
end
```