BRUNO PIGARI RATEIRO
DANILO SIQUEIRA CESAR

IMPLEMENTAÇÃO DE UM SOFTWARE DE CONTROLE DE IMPEDÂNCIAS PARA UM EXOESQUELETO ROBÓTICO

São Paulo
2013

BRUNO PIGARI RATEIRO
DANILO SIQUEIRA CESAR

IMPLEMENTAÇÃO DE UM SOFTWARE DE CONTROLE DE IMPEDÂNCIAS PARA UM EXOESQUELETO ROBÓTICO

Relatório apresentado à disciplina PMR2500 – PROJETO DE CONCLUSÃO DE CURSO II.

São Paulo
2013

BRUNO PIGARI RATEIRO
DANILO SIQUEIRA CESAR

IMPLEMENTAÇÃO DE UM SOFTWARE DE CONTROLE DE IMPEDÂNCIAS PARA UM EXOESQUELETO ROBÓTICO

Relatório apresentado à disciplina PMR 2500 – PROJETO DE CONCLUSÃO DE CURSO II.

Orientador(a):
Prof. Dr. Arturo Forner-Cordero

São Paulo
2013

FICHA CATALOGRÁFICA

Rateiro, Bruno Pigari
Implementação de um software de controle de impedâncias para um exoesqueleto robótico / B.P. Rateiro, D.S. Cesar. -- São Paulo, 2013.
p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Robótica (Reabilitação) 2.Arquitetura de software I.Cesar, Danilo Siqueira II.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos III.t.

Às nossas famílias, que nos deram suporte através da sua solicitude durante nossas vidas inclusive nesse momento decisivo para nossas carreiras.

AGRADECIMENTOS

Agradecemos às nossas famílias, nosso eterno suporte, por todo o apoio durante todos os dias de nossas vidas.

Agradecemos ao nosso colega Luís Rossi pelas sugestões e debates realizados ao longo de todo o projeto.

Agradecemos ao professor João Alcino de Andrade Martins pelas orientações com relação à instrumentação da célula de carga.

Agradecemos ao Prof. Dr. Arturo Forner-Cordero pela oportunidade de nos envolvermos em um projeto diferenciado e inspirador.

RESUMO

O trabalho se trata da concepção e implementação de uma arquitetura de programação que visa controlar o comportamento de um exoesqueleto robótico para o braço, atuado na região do cotovelo, valendo-se da teoria de controle de impedâncias.

É utilizada linguagem de programação de baixo nível (linguagem C), com o objetivo de reduzir custo computacional e permitir menores tempos de amostragem, controle e atuação.

Usa-se um controlador PC/104 cuja programação é feita em Linux Real-Time através do projeto Xenomai, para garantir que a aplicação tenha tempos de resposta exatos e definidos.

A utilização de controle de impedâncias tem aplicação em mecanismos de reabilitação e fisioterapia, de amplificação da capacidade humana, dispositivos de teleoperação, entre outros.

O programa foi desenvolvido e verificado. Funciona conforme o esperado e respeita os tempos de execução planejados.

Palavras-chave: robótica de reabilitação, controle de impedâncias, exoesqueleto.

ABSTRACT

This project focuses on the design and implementation of a programming architecture aiming at controlling a robotic exoskeleton for the arm, actuated in the elbow, making use of impedance control theory.

Low level programming language is used (C language) in order to reduce computational cost and allow lower sampling, controlling and acting times.

A PC/104 controller is used, running Linux Real-Time as operative system with real time programming implemented with a Xenomai project, to ensure that the application follows the pre-defined response times accurately.

Impedance control may be applied in rehabilitation and physiotherapy mechanisms, amplification of human capacity, teleoperation devices, among others.

The feasibility of practical implementation of discrete linear control theory applied to impedance control of mechanical systems could be verified, even on the existence of mechanical and instrumentation issues.

Keywords: rehabilitation robotics, impedance control, exoskeleton.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

As pesquisas na área de robótica são tradicionalmente focadas em aplicações industriais. No entanto, com a evolução da tecnologia, aplicações na área tem se estendido, não apenas com interesse em auxiliar tarefas humanas repetitivas, mas também para auxiliar as pessoas em tarefas diárias (AREVALO e GARCIA, 2012). Para que robôs possam executar tarefas que envolvam contato direto com pessoas de forma segura e eficiente é necessário conhecimento e capacidade de modelagem do sistema motor humano.

1.1. Motivação

Neste trabalho busca-se projetar e utilizar uma arquitetura de controle para um exoesqueleto robótico do braço já construído, atuado em um grau de liberdade na região do cotovelo. Com isso, pretende-se desenvolver o controle de um exoesqueleto que possa ser utilizado para obter informações sobre o controle motor humano e entender como a interação entre seres humanos e robôs pode ser efetuada de forma segura e eficiente. Os resultados de tais estudos podem fornecer informações úteis aplicáveis a mecanismos de reabilitação e fisioterapia, de amplificação da capacidade humana, dispositivos de teleoperação, entre outros.

Esse projeto é uma continuação do trabalho desenvolvido em dois trabalhos de conclusão de curso, de 2011 e 2012.

O primeiro, realizado por Yasutomi e Miranda (2011), envolve a primeira etapa do projeto geral, que consistiu em projetar e construir a estrutura mecânica do exoesqueleto. Nele estão descritas as etapas de construção e aspectos relevantes em relação à modelagem matemática do mecanismo.

O segundo, realizado por Abduch e Ruivo (2012), visou complementar o trabalho mecânico e concluir o projeto ao propor a instrumentação e a malha de controle para o mesmo. Inclui tomadas de decisão na configuração mecânica, que resultaram na adição de uma célula de carga não comercial para o sensoriamento do sistema e na adição de

um novo cotovelo mecânico. Também inclui o projeto eletrônico de todo o sistema e a implementação de uma malha de controle em plataforma computacional de alto nível, que permite a criação da malha a partir da construção de diagramas de bloco utilizando o software Simulink.

Apesar de esse último ter sido importante devido à implantação do sensoriamento e realização de alguns testes que permitiram a verificação da funcionalidade do sistema mecânico, alguns de seus resultados justificam a origem do projeto atual:

- A implantação da malha de controle em software de alto nível apresentava alto custo computacional.
- A utilização de linguagem de alto nível dificultava a compreensão, controle e manutenção do código que de fato era compilado para ser executado pelo controlador.
- Não era possível garantir que a leitura de sensores, o processamento e a atuação fossem realizadas em tempo real, ou seja, que os tempos de cada uma dessas tarefas fossem fixos e bem determinados.

Todos esses fatores afetaram a estabilidade e eficiência do sistema e a aplicabilidade para o estudo do sistema motor humano.

1.2. Proposta

O presente trabalho tem como objetivo solucionar os problemas destacados ao propor uma nova malha de controle que será implementada através de uma plataforma que executa o controle em tempo real (*real time*), utilizando linguagem de programação de mais baixo nível (linguagem C) em relação ao trabalho desenvolvido anteriormente.

A necessidade de o controle ser em tempo real deriva do fato da teoria de controle discreto ser desenvolvida considerando períodos de amostragem, controle e

atuação como constantes, não sendo válida, portanto, quando estes se comportam como uma variável aleatória.

O projeto foi realizado no Laboratório de Biomecatrônica da Escola Politécnica da Universidade de São Paulo e faz parte de um projeto de pesquisa, Estudo do Controle Motor do Membro Superior, desenvolvido por uma equipe multidisciplinar. Este projeto, atualmente, pretende também realizar a construção de um exoesqueleto atuado em três graus de liberdade (ombro, cotovelo e punho), onde os resultados do presente trabalho (exoesqueleto atuado apenas no cotovelo) podem ser reaproveitados, especificamente aqueles relativos ao controle.

Assim como nos trabalhos anteriores, serão utilizados os conceitos da teoria de controle de impedâncias mecânicas. Com essa nova malha de controle, espera-se atingir um maior controle sobre a impedância mecânica aparente do exoesqueleto.

Esta monografia encontra-se organizada da seguinte maneira: no item 2 é apresentada uma revisão conceitual e bibliográfica de conceitos teóricos relevantes, além de uma revisão breve do estado da arte sobre robôs de auxílio a atividades humanas. No item 3 é apresentado como o projeto foi desenvolvido: inicialmente apresenta-se quais são os equipamentos e recursos disponíveis utilizados durante todas as etapas do projeto e, em seguida, é apresentada a metodologia de projeto adotada. No item 4, os resultados obtidos são apresentados e no item 5, discutidos. No item 6 são apresentadas as conclusões que puderam ser realizadas a partir dos resultados.

2. REVISÃO

2.1. Exoesqueleto

Na área de robótica, exoesqueleto pode ser definido como um mecanismo que pode ser vestido, usualmente com uma configuração antropomórfica, capaz de acompanhar os movimentos das extremidades do usuário (AGUIRRE-OLLINGER, *et al*., 2012).

Exoesqueletos robóticos podem apresentar diversas aplicações, como amplificadores das capacidades mecânicas humanas, na reabilitação e fisioterapia de pacientes, como dispositivo de controle remoto (incluindo a utilização de *feedback* de força/posição) e na pesquisa biomédica e biomecatrônica (YASUTOMI e MIRANDA, 2011).

2.2. Estado da Arte

Foram selecionados para esse relatório dois trabalhos do estado da arte. Ambos tiveram seus trabalhos publicados recentemente.

2.2.1. HAL 5

Modelo de exoesqueleto desenvolvido pela Cyberdine, que é a evolução de modelos anteriores. HAL significa Hybrid Assistive Limbs. O exoesqueleto vestido encontra-se ilustrado na figura 1. O modelo é um exoesqueleto completo para o corpo humano, incluindo pernas, torso e braços.

O objetivo do robô é auxiliar as pessoas com algum tipo de deficiência a desenvolver tarefas do cotidiano.

O controle do exoesqueleto se vale da teoria do controle de impedâncias, porém difere do presente trabalho por utilizar o sensoriamento por sinais lidos diretamente da musculatura humana (sinais eletromiográficos).

Figura 1 - Exoesqueleto para membros superiores e inferiores.
Disponível em: <http://www.cyberdyne.jp/english/robotsuithal/index.html**>.**
Acesso em: maio. 2013.

2.2.2. Esbirro

O Esbirro é um exoesqueleto dos membros inferiores que visa auxiliar pessoas com deficiência de locomoção. O trabalho de programação é muito semelhante ao do atual projeto, pois utiliza o mesmo controlador que será utilizado no projeto (PC/104), a mesma estratégia de programação (Linux Real-Time) e teoria de controle de impedâncias. Além disso, a estratégia de sensoriamento é similar, e utiliza feedback a partir de Strain-Gauges em ponte de Wheatstone completa. Apesar da finalidade do conjunto ser diferente da do atual projeto, a estratégia de controle, arquitetura de programação e sensoriamento é similar. Informações disponíveis em: <http://www.car.upm-csic.es/bioingenieria/projects.htm>.

2.3. Controle de impedâncias

Com o surgimento do interesse no desenvolvimento de robôs para aplicações que não envolvem tarefas repetitivas e tarefas onde as forças de interação devem ser controladas mesmo sem o conhecimento exato de qual o ambiente de interação do robô, o controle de força ou posição apenas torna-se insuficiente. Particularmente em tarefas de contato com seres humanos, o controle de posição apenas pode oferecer riscos devido à possibilidade de aplicação de altas forças, porém o controle de força apenas não garantiria um posicionamento coerente do robô.

O objetivo do controle de impedâncias não é controlar diretamente posição ou força, mas como elas se relacionam (AREVALO e GARCIA, 2012). Para isso, é possível se valer do conceito de impedância mecânica, a partir da analogia matemática entre a modelagem de grandezas mecânicas e elétricas. Mais especificamente, existe equivalência matemática na modelagem do comportamento entre força e velocidade e no comportamento entre tensão elétrica e corrente (AREVALO e GARCIA, 2012). A partir disso, é definida a impedância mecânica Z e a admitância mecânica Y como:

$$Z = \frac{F}{\dot{x}} \qquad Y = \frac{\dot{x}}{F}$$

onde F é força e $\dot{x}$ é a velocidade. A equação de Z acima define a viscosidade de um sistema mecânico em analogia matemática a resistência de um sistema elétrico. Porém, estendendo a analogia aos conceitos de indutância e capacitância elétricas, têm-se, de forma equivalente, massa e rigidez, respectivamente, no campo mecânico. As equações abaixo permitem visualizar a equivalência matemática:

$$M \cdot \frac{\mathrm{d}v}{\mathrm{dt}} + B \cdot v + K \cdot \int v \,\mathrm{dt} = T$$

$$L \cdot \frac{\mathrm{d}i}{\mathrm{dt}} + R \cdot i + \frac{1}{C} \cdot \int i \,\mathrm{dt} = V$$

Onde:

M: massa;

B: viscosidade;

K: rigidez;

L: indutância;

R: resistência;

C: capacitância.

Para isso, é necessário medir a força e a posição do mecanismo em cada instante e calcular a sua impedância efetiva naquele instante. A atuação é feita de forma a alterar essa impedância para que ela se aproxime da impedância de referência previamente definida.

A partir disso, o controle de impedâncias tem como objetivo alterar a impedância mecânica aparente de um sistema, de forma a atender determinado requisito de controle, pois, como comentado, para os casos descritos anteriormente, o controle de força ou posição apenas não é o desejável.

Outra forma de definir o controle de impedâncias seria como um sistema massa-mola-amortecedor com parâmetros (massa, viscosidade e rigidez) ajustáveis para valores determinados. O ajuste destes parâmetros depende das características do ambiente e do tipo de interação que se deseja atingir com ele. Para aplicações de contato com seres humanos, por exemplo, é necessário garantir que a massa e a rigidez aparente do robô sejam baixas evitando o surgimento de grandes forças de interação. Entretanto para a manipulação de objetos pesados, é necessário garantir alta rigidez.

Segundo Hogan (1987), um robô pode ser considerado como um sistema inercial dirigido pelas forças exercidas pelo atuador e pelo ambiente. O exemplo a seguir mostra o comportamento de um robô de massa M_{real} exposto à força externa $F_{externa}$, provinda da interação com o ambiente e a força de atuação $F_{atuador}$:

$$M_{real}\frac{d^2x}{dt^2} = F_{atuador} - F_{externa}$$

Se a força de atuação for adotada como uma função da força externa, do tipo:

$$F_{atuador} = -KF_{externa}$$

onde K é um ganho de valor arbitrário positivo não nulo, o comportamento dinâmico do sistema poderia ser escrito como:

$$\frac{M_{real}}{(K+1)}\frac{d^2x}{dt^2} = -F_{externa}$$

A partir dessa equação, é possível observar que o comportamento dinâmico do sistema equivale ao de um sistema cuja massa é inferior à massa M_{real}. Ou seja, é possível afirmar que o controlador por feedback de força reduziu a inércia aparente do sistema. O controle de impedâncias parte da premissa que o controlador pode mudar o comportamento dinâmico do sistema à partir da alteração dos valores da impedância aparente.

Existem duas abordagens práticas a partir das quais pode ser implementado tal controle: controle de impedâncias baseado em força e controle de impedâncias baseado em posição.

No controle de impedâncias baseado em posição, a força de interação entre o robô e o ambiente serve como referência para se determinar qual a posição necessária do sistema naquele instante tal que satisfaça a sua impedância mecânica desejada. Esta posição necessária, por sua vez, serve como referência para uma malha interna de controle de posição.

Analogamente, em um controle de impedâncias baseado em força, a leitura da posição, velocidade e aceleração servem como parâmetro para se determinar a força necessária que satisfaça a impedância mecânica desejada, a qual serve de referência de força para uma malha interna de controle de força.

Este projeto utilizará controle de impedância baseado em posição e um de seus objetivos é desenvolver um controle que possibilite ajustar diversas configurações de teste, variando a impedância mecânica do exoesqueleto em função da forma de interação entre robô e usuário e fornecer assim, uma ferramenta de trabalho que possa ser utilizada para diversas aplicações, em especial aquelas que envolvam interação direta com pessoas.

O controle de impedâncias baseado em posição foi escolhido, pois é mais adequado considerando-se os equipamentos e recursos com os quais se pretende desenvolver o trabalho: o driver utilizado já possui um controlador de posição e a célula de carga desenvolvida no trabalho de conclusão de curso anterior fornece a força medida entre exoesqueleto e usuário.

Por exemplo, o comportamento dinâmico do sistema perna humana e exoesqueleto foi proposta por Araújo, Tannuri e Forner-Cordero (2012) como sendo um sistema massa mola e amortecedor. De forma similar, a dinâmica do braço acoplado com o exoesqueleto pode ser expressa pela seguinte relação:

$$\mathrm{J} \cdot \ddot{\theta} + \mathrm{B} \cdot \dot{\theta} + \mathrm{K} \cdot \theta = \tau$$

Onde J é o momento de inércia do conjunto, B é o amortecimento viscoso e K é a rigidez, θ é o ângulo formado pela abertura da articulação do cotovelo e τ é o torque existente entre o exoesqueleto e o braço do usuário.

Como apresentado em trabalhos anteriores de Araújo, Tannuri e Forner-Cordero (2012), Hogan (1985) e Aguirre-Ollinger (2012), a estrutura de controle de impedâncias programada é ilustrada na figura 2. O controlador recebe o torque medido entre o exoesqueleto e o usuário e o sinal é então tratado no bloco que implementa a impedância desejada (“Desired Impedance”). O sinal de saída do bloco de controle de impedâncias é então enviado como referência de posição para o controlador de posição atuante no exoesqueleto.

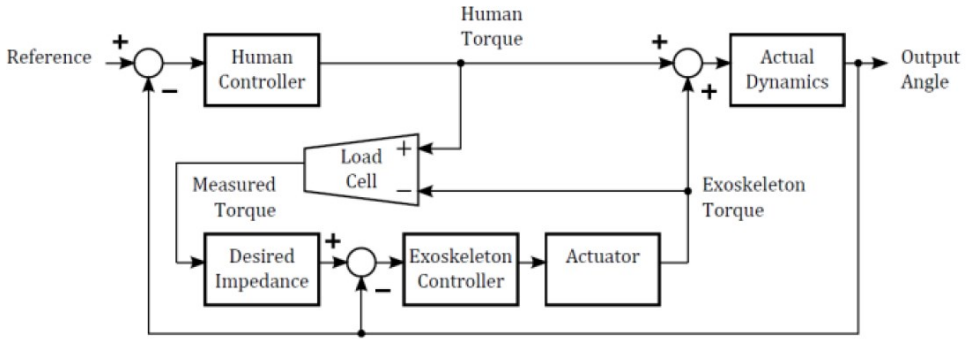


Figura 2 - Malha de controle de impedâncias. Extraído de Araújo, Tannuri e Forner-Cordero (2012).

O bloco de impedância modela o mesmo comportamento físico de um sistema massa, mola e amortecedor. A implementação corresponde à seguinte função de transferência:

$$\frac{\theta}{\tau} = \frac{1}{J_d \cdot s^2 + B_d \cdot s + K_d}$$

Os índices *d* presentes nos termos de inércia, amortecimento e rigidez representam os parâmetros desejados do sistema e são configurados pelo programa.

Segundo Araújo, Tannuri e Forner-Cordero (2012), caso a impedância desejada do sistema se iguale à impedância real do braço do usuário, este não sentiria resistência ao movimento por parte do exoesqueleto. Dessa forma, o exoesqueleto se comportaria como um elemento seguidor de movimento.

Impedâncias superiores à impedância de um elemento seguidor caracterizam um sistema que resiste ao movimento. De forma contrária, impedâncias abaixo do elemento seguidor caracterizam sistemas que amplificam o movimento.

A estrutura de controle deve ser tal que a impedância mecânica do mecanismo seja alterada de forma a atender as necessidades de controle em função do estado do sistema (usuário e exoesqueleto).

3. DESENVOLVIMENTO

3.1. Recursos, equipamentos e ambiente de trabalho.

O trabalho é realizado no Laboratório de Biomecatrônica da Escola Politécnica da Universidade de São Paulo. Durante o projeto serão utilizados os seguintes equipamentos e recursos.

3.1.1. Exoesqueleto

O primeiro protótipo do exoesqueleto utilizado foi desenvolvido por Yasutomi e Miranda (2011) e seu modelo em CAD e foto ilustrativa estão apresentados na figura 3.

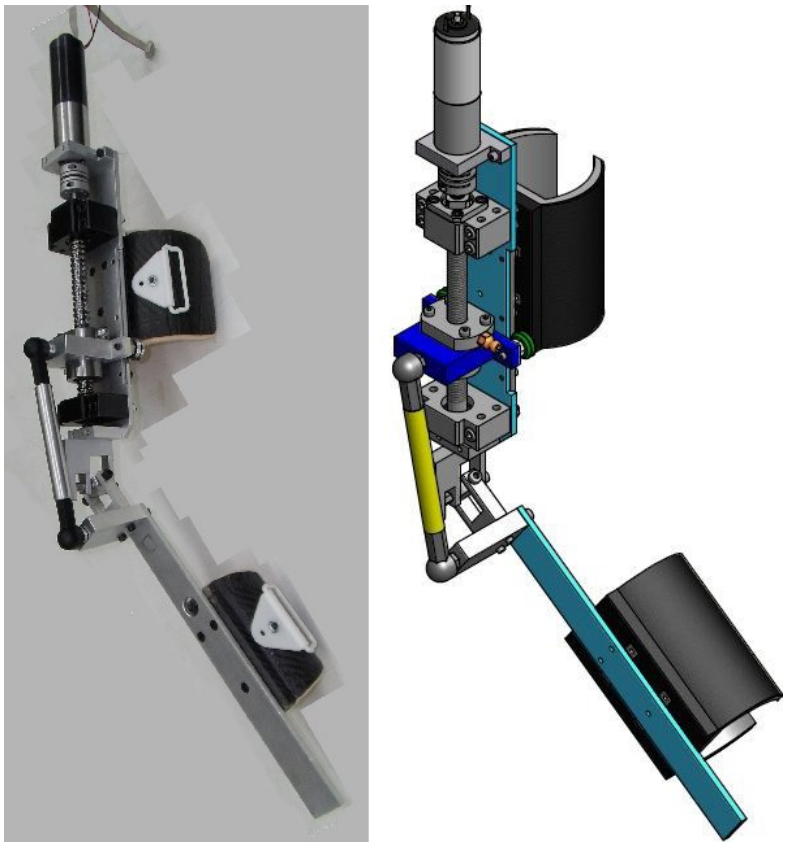


Figura 3 - Exoesqueleto à esquerda e modelo em CAD a direta. Extraído de Yasutomi e Miranda (2011)

O exoesqueleto com as modificações realizadas por Abduch e Ruivo (2012) é apresentado na figura 4 e inclui uma nova célula de carga e um novo cotovelo mecânico.

Figura 4 - Exoesqueleto de membro superior (versão atual)

O mecanismo dispõe de um motor acoplado rigidamente ao braço do exoesqueleto que é acionado pelo sistema de controle. Através da rotação do motor, um fuso é rotacionado movimentando linearmente uma guia. A guia está acoplada através de uma junta esférica a extremidade de uma haste, cuja extremidade oposta está acoplada novamente por uma junta esférica ao antebraço do exoesqueleto. Desta forma, o movimento da guia impõe um movimento a haste e, consequentemente, um movimento relativo entre braço e antebraço.

Percebeu-se a existência de folgas em vários componentes mecânicos do exoesqueleto. Outro problema observado foi a deflexão sob esforços de algumas uniões modeladas como rígidas, principalmente a união entre cotovelo e braço. Alguns desses problemas puderam ser significativamente reduzidos através da refixação de alguns dos componentes do exoesqueleto, porém outros permaneceram até o fim do projeto e não foram eliminados, pois exigiriam um retrabalho incluindo eventual reprojeto e/ou refabricação de partes mecânicas o que foge do escopo do projeto.

Apesar disso, pôde-se executar o desenvolvimento do controle do braço de forma considerada satisfatória, como apresentado nas seções finais desta monografia.

Maiores detalhes quanto à modelagem cinemática do mecanismo estão apresentados na seção 3.2.3. Todos os detalhes sobre o projeto e construção do primeiro protótipo do mecanismo do exoesqueleto podem ser encontrados em Yasutomi e Miranda (2011).

3.1.2. PC/104, modelo PCM3362

O dispositivo encarregado de realizar todo controle é a placa (PC/104, modelo PCM-3362, Advantech Inc) apresentada na figura 5, que na verdade possui a arquitetura de um PC, porém com um processador sem *cooler*. O PC/104 apresenta todas as funcionalidade de um PC comum, mas com baixo consumo de energia, o que o torna bastante recomendado para aplicações embarcadas. Possui portas seriais, portas USB, entrada IDE e interface LCD e CRT. Suporta Windows XP, Windows CE e Linux.

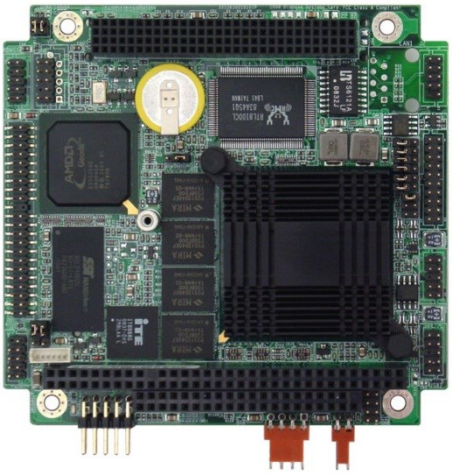


Figura 5 - Power PC/104-Plus SBC

Foi escolhido o PC104 porque ele permite a implementação de um controlador, porém com todos os recursos de um computador com arquitetura PC. Dentre esses recursos, são úteis ao projeto: a possibilidade de utilização de sistemas operacionais diferentes, a facilidade de impressão de resultados em um monitor, a possibilidade de modificação da linguagem de programação utilizada, a facilidade de entrada de dados durante a execução do programa com um teclado, entre outros. Além disso, o mesmo possui capacidade de processamento superior à requisitada pelo projeto.

O laboratório dispõe de uma bancada onde é possível conectar o PC104 a um monitor e a teclado e mouse, facilitando os testes e a programação. Instalou-se um sistema operacional Linux, modificado de forma a poder operar em tempo real, conforme descrito no item 3.1.8. Especificações técnicas relativas ao PC104 se encontram no Anexo A.

3.1.3. Placa Diamond-MM-16-AT

Para realizar a leitura e o envio de sinais analógicos pelo controlador foi utilizada a placa Diamond-MM-16-AT, ilustrada na figura 6. Ela é especificamente projetada para funcionar com o PC104, descrito anteriormente, e conta com um elevado número de entradas e saídas analógicas e digitais, superior às necessidades do projeto.

Figura 6 - Diamond-MM-16-AT Analog I/O Module

A calibração e utilização das portas pode ser feita através de uma interface desenvolvida em C, fornecida pelo próprio fabricante. Mais informações a respeito da placa Diamond-MM-16-AT podem ser encontradas no Anexo B.

3.1.4. Motor Maxon EC 32 - 80 watts (modelo 118889)

O motor utilizado na atuação é do tipo EC, fabricado pela Maxon Ag, Switzerland, cujo modelo e imagem são apresentados na figura 7. Mais informações técnicas relacionadas ao motor estão apresentadas no Anexo C.

A escolha do motor mencionado e sua aplicação em exoesqueletos de membros superiores é citada no trabalho de Yasutomi e Miranda, 2011 e detalhada no trabalho de Forner-Cordero, et al. 2011. Nesse último trabalho são explorados os aspectos da transferência de potência entre o exoesqueleto e o usuário que definem as características desejadas do atuador. O intervalo de operação é então estimado através dos parâmetros de torque e velocidade exigidos pela interação usuário exoesqueleto.

Figura 7 - Motor Maxon EC 32 - 80 watts- modelo 118889

3.1.5. Driver Maxon EPOS2 24/5

Para controlar o motor acima, foi escolhido o driver Maxon EPOS2 24/5, ilustrado na figura 8. Informações técnicas a respeito do driver escolhido podem ser encontradas no Anexo D. Analogamente ao motor, o driver fora mencionado nos trabalhos de Forner-Cordero, et al.

Figura 8 - Driver Maxon EPOS2 24/5

3.1.6. Potenciômetro

Utilizou-se como referência auxiliar para medição da posição por osciloscópio um potenciômetro rotativo acoplado ao cotovelo, ilustrado na figura 9.

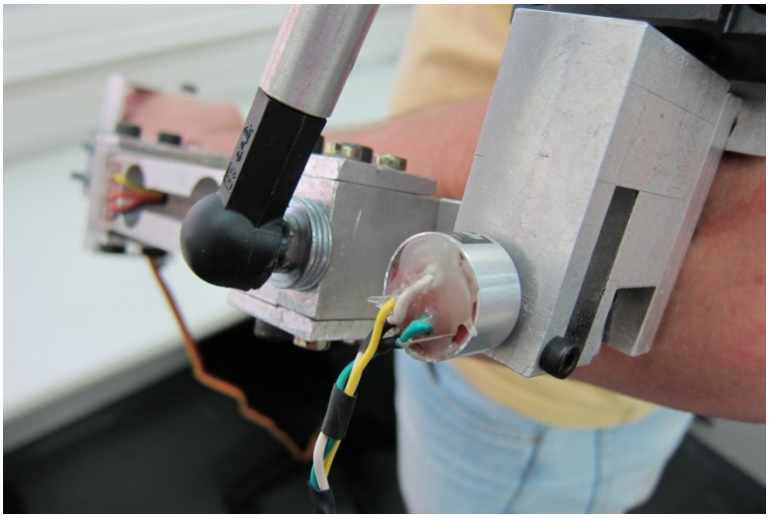


Figura 9 - Potenciômetro acoplado ao cotovelo do exoesqueleto

3.1.7. Circuito de condicionamento de sinais

Para a leitura e tratamento dos sinais provenientes da célula de carga e do potenciômetro utilizou-se as placas eletrônicas das figuras 10 e 11. A primeira, desenvolvida no projeto do ano anterior e a segunda projetada por um dos membros do laboratório e que realiza a mesma função relacionada ao tratamento dos sinais dos *strain gauges* (célula de carga) na primeira placa. Ambas possuem amplificação e filtros de alta frequência, porém a segunda com uma menor amplificação de sinal.

Testou-se ambas as placas para o tratamento do sinal do *strain gauge*, mas a segunda foi escolhida por apresentar menor possibilidade de saturação após a amplificação do sinal, devido a sua menor amplificação. Porém a primeira continuou a ser utilizada na leitura do sinal do potenciômetro.

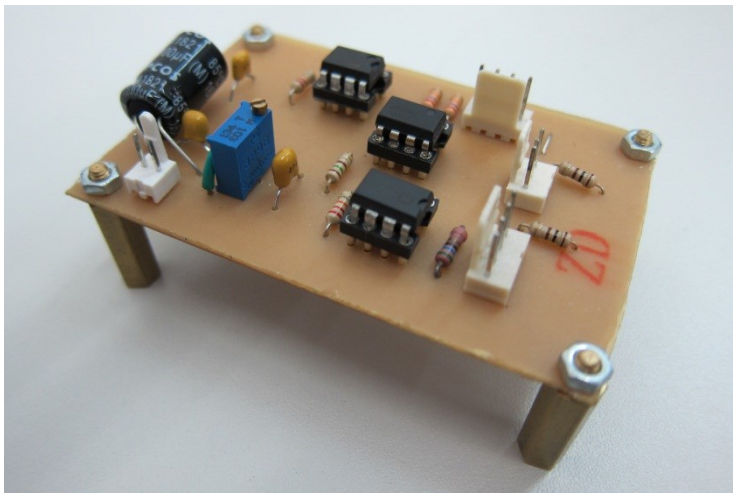


Figura 10 - Placa de tratamento de sinais (célula de carga e potenciômetro)

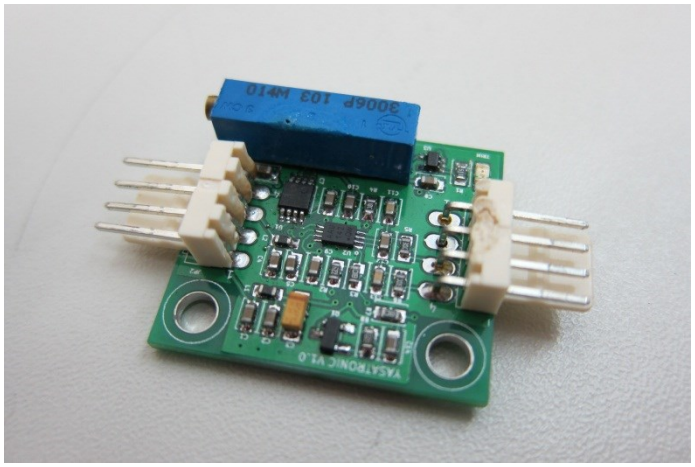


Figura 11 - Placa de tratamento de sinais (célula de carga)

O esquemático do circuito da segunda placa se encontra no Anexo F e o da primeira placa encontra-se no Anexo E.

3.1.8. Linux Real Time

Um controle de impedâncias eficiente depende não somente da malha de controle em si, mas também da rapidez com a qual o sistema é capaz de ler os sensores, realizar o controle e enviar o sinal de controle para os atuadores. Portanto,

um dos aspectos de preocupação do projeto é a latência de execução das etapas de controle.

Uma maneira de contornar o problema é fazer com que o código contenha funções que são priorizadas pelo sistema operacional, com a finalidade de reduzir o tempo de espera para o processamento dos comandos. O conceito de funções priorizadas pelo sistema operacional é o principal conceito de um Sistema Operacional. Em se tratando desse projeto, o objetivo é atingido satisfatoriamente através da configuração do *kernel* (núcleo) de qualquer uma das distribuições do Linux com projetos de Linux Real Time compatíveis.

Foram levados em conta outros sistemas operacionais de tempo real dedicados a aplicações embarcadas, porém devido ao acesso e suporte relativamente inferiores em relação às distribuições Linux, não foram escolhidos como resposta ao problema.

As aplicações Linux tem vantagem por serem estáveis, possuírem uma comunidade de programadores que colaboram e conhecem projetos envolvidos, gratuidade em sua distribuição, por serem customizáveis e pelo grande número de rotinas de código aberto.

Dentre as distribuições Linux mais utilizadas, podemos destacar as seguintes distribuições: Ubuntu, Fedora e Debian. Dentre os projetos que implementam o Linux Real Time, temos: RTLinux, Xenomai e RTAI. Esses projetos são semelhantes entre si e todas as funções neles implementadas satisfazem a necessidade do projeto, pois todos possuem uma biblioteca de funções similares.

Segundo Brown (2012), quando se trata de eficiência ao controlar a execução das *tasks*, o Xenomai, dentre os projetos citados, é o que melhor garante aplicações Real Time.

Para este projeto escolheu-se utilizar a distribuição Debian, modificada a partir do projeto Xenomai (disponível em: <http://www.xenomai.org/>). Escolhemos esse par de soluções pelo fato de que essa distribuição já vem pré-configurada com alguns parâmetros do projeto Xenomai, e este, por sua vez, é um projeto que é constantemente atualizado pela comunidade de programadores.

Inicialmente, as devidas instalações foram feitas em uma máquina virtual. Optou-se pela utilização de uma máquina virtual, pois alterar o *kernel* de qualquer software é uma tarefa que requer experiência e altamente susceptível a erros que só são reversíveis caso a instalação seja reiniciada, o que seria altamente impactante no tempo de projeto. Com a máquina virtual, é possível simular as características do controlador e gerar cópias da situação do mesmo em cada etapa da instalação. Dessa forma, caso alguma etapa passasse por uma não conformidade que viria a comprometer o trabalho, a etapa anterior seria restaurada.

O código de controle é desenvolvido também dentro de uma máquina virtual e posteriormente importado diretamente para o controlador via rede, eliminando a necessidade de integrar o código e aumentando o poder de divisão de tarefas entre os programadores.

3.1.9. RedMine

Com a finalidade de criar um ambiente unificador de informações, de gerenciamento de projeto e de um repositório de código, a dupla propôs como solução a criação e manutenção de um servidor. Essa ferramenta visa dar acesso a toda a documentação, servir de meio de comunicação entre os membros do laboratório e gerenciar e armazenar as versões de código desenvolvidas.

O RedMine é uma interface instalada em um servidor, onde se têm um ambiente apropriado para o gerenciamento de tarefas, desenvolvimento de código (devido ao repositório de código) e compartilhamento de documentação de projeto.

A configuração de um servidor envolveu a aplicação de conceitos de programação Ruby on Rails, linguagem na qual o RedMine foi concebido. Com essa linguagem, foi possível customizar a interface para melhor atender as nossas necessidades. O servidor se encontra em nuvem e o acesso para o controle do mesmo é feito com protocolo SSH. O sistema operacional utilizado pelo servidor é Ubuntu 12.04 devido ao seu baixo custo operacional.

Um dos ganhos imediatos é a comunicação entre membros do laboratório, podendo observar os trabalhos desenvolvidos em paralelo, facilitada pelo envio de e-mails gerados automaticamente para cada atualização dos status das tarefas, fóruns de perguntas e respostas, agenda do projeto e gráfico de Gantt. Têm-se valido dessa ferramenta para documentar o andamento das tarefas.

Como o RedMine está vinculado ao repositório, é possível relacionar tarefas a serem desenvolvidas com trechos de código específicos. O código de versões diferentes também pode ser comparado através da internet apenas com o ambiente RedMine, facilitando os trabalhos de colaboração no laboratório. O servidor criado está disponível em: <http://biomecatronica.bitnamiapp.com/redmine/>.

3.2. Metodologia

3.2.1. Máquina de estados funcional

A aplicação desenvolvida deve operar como uma máquina de estados (implementada em *software*) conforme figura 12.

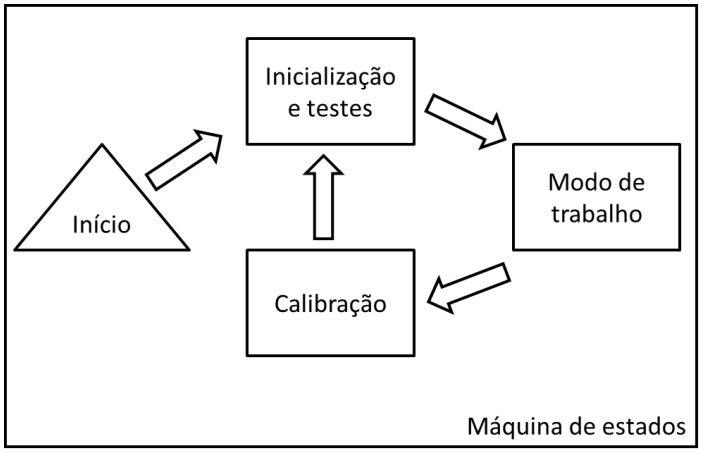


Figura 12 – Máquina de estados funcional da aplicação

Após o início da aplicação, o sistema entra no estado de inicialização e testes. Primeiramente, um menu é oferecido ao usuário, de forma que ele possa modificar alguns parâmetros discutidos na seção 3.2.6. Nesse estado são inicializadas todas as estruturas de software e os atuadores e se estes estão funcionando corretamente. É necessário verificar a posição mecânica atual do sistema para que ele seja levado a uma posição inicial segura e operável. Configura-se o driver do motor para aceitar a determinada posição como *home*. Dessa forma é garantida a segurança do usuário e o correto funcionamento dos estados seguintes. Evidentemente, por segurança, é necessário verificar manualmente a posição mecânica antes da inicialização da aplicação.

Terminada essa etapa, o sistema passa para o próximo estado, que consiste na calibração de parâmetros do controlador. A obtenção de tais parâmetros é realizada a

partir de testes mecanismos com e sem usuário. Estes testes visam estimar as características do exoesqueleto e características associadas ao sistema, considerando o usuário.

Após o término dos testes, o sistema entra em um estado de modo de trabalho, onde de fato executa as aplicações propostas. Inicialmente, tinha-se como objetivo implementar três modos de trabalho, os quais teriam aplicações distintas. Os três modos de trabalho se distinguiam no sentido de auxiliar, seguir ou atrapalhar o movimento do usuário. Porém, as definições partiam do mesmo modelamento matemático. Portanto, alternativamente, preferiu-se deixar o controle parametrizado de acordo com o comportamento físico desejado. Ou seja, rigidez, viscosidade e inércia poderiam ser alterados no início do programa, podendo-se obter uma gama de impedâncias desejadas.

3.2.2. Segurança em software

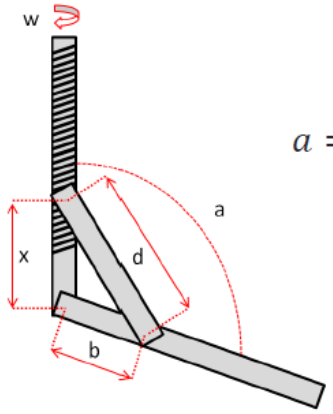
O curso de operação do exoesqueleto possui duas camadas de software que impedem a hiperextensão do braço. A primeira, pertencente à parte de controle do programa, filtra os valores de posição a serem enviados como *setpoint* para o motor. Caso o valor calculado de *setpoint* através da impedância desejada supere os limites superior ou inferior, a posição assume o valor de *setpoint* correspondente ao limite mais próximo.

A segunda pertence à própria programação do driver, cuja relação entre a voltagem recebida na porta analógica e número de rotações possui uma limitação máxima. Essa limitação é calibrada em torno da posição de equilíbrio tida como o ângulo de 90º da articulação do cotovelo, de tal forma que é possível obter um mesmo deslocamento angular em ambos os sentidos.

Ambas as proteções visam garantir a segurança do usuário. O curso completo devido às limitações impostas, levando em conta a redução do exoesqueleto e do motor, fica restrito à 60º e 120º de abertura.

3.2.3. Curso de operação

Uma preocupação inicial quanto ao controle desenvolvido, envolvia a necessidade de se considerar as não linearidades envolvidas na cinemática do mecanismo, no que diz respeito à relação entre o ângulo do antebraço e deslocamento linear da guia no fuso, proporcional a rotação do motor, ilustrada na figura 13.



$$a = \cos^{-1}\left(\frac{b^2 + x^2 - d^2}{2xb}\right)$$

$$x = w \times passo$$

Figura 13 - Relação entre ângulo entre braço e antebraço e deslocamento linear da guia

Onde:

- a: Ângulo de abertura;
- b: Dimensão compreendida entre a junta do cotovelo e a junta esférica do antebraço.
- x: Dimensão compreendida entre a junta do cotovelo e a junta esférica do braço. Dimensão variável pelo número de rotações do motor.
- d: Comprimento do braço que conecta as duas juntas esféricas.

Percebeu-se que a relação era aproximadamente linear para a região trabalhada de fato (60° a 120º ou aproximadamente 120mm a 170mm de deslocamento da guia), conforme figura 14.

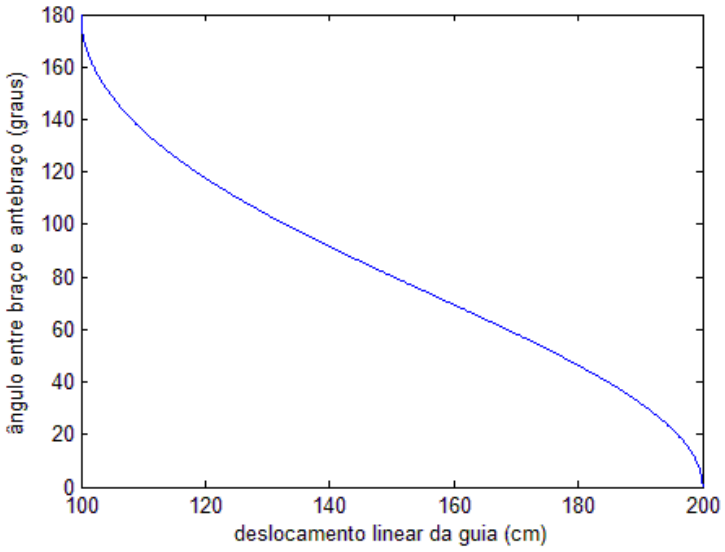


Figura 14 - Gráfico Relação entre ângulo entre braço e antebraço e deslocamento linear da guia

Para se verificar a validade dessa hipótese, analisou-se o gráfico de sua derivada, apresentado na figura 15, onde é possível verificar que a derivada é aproximadamente constante nessa mesma região. A partir disso, considerou-se que o controle linear era suficiente.

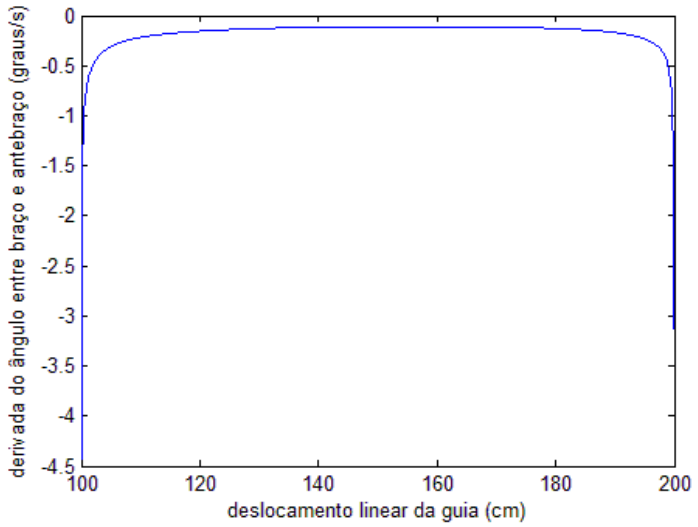


Figura 15 - Derivada do ângulo em relação ao deslocamento linear da guia

3.2.4. Estrutura de Controle

A estrutura de controle adotada segue o mesmo esquema apresentado na figura 2, na seção 2.3, onde o bloco de impedância desejada (*Desired Impedance*) calcula a posição desejada a partir do torque medido entre usuário e exoesqueleto e da impedância desejada, previamente escolhida. Porém, como ferramenta de obtenção da equação exata executada pelo controlador para implementação prática desse bloco, utilizou-se o software MATLAB, onde simulações permitiram analisar o comportamento teórico do sistema que se deseja controlar.

Inicialmente foi criado um modelo contínuo do bloco de impedância desejada, apresentado na figura 16.

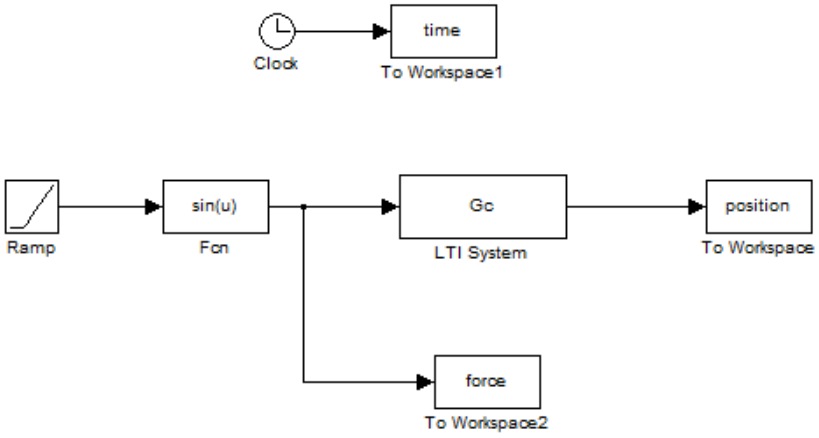


Figura 16 - Modelo contínuo do bloco de controle de impedância desejada

Onde Gc é o bloco de impedância desejada e implementa uma função de transferência de segunda ordem do tipo

$$\frac{\theta(s)}{\tau(s)} = \frac{1}{J_d \cdot s^2 + B_d \cdot s + K_d}$$

e Fcn é uma função qualquer ajustável.

Com esse modelo é possível analisar o comportamento do sistema para diversos tipos de sinais de torque. Além disso, esse modelo contínuo serviu como base para a simulação de outros modelos discretos, já que a conversão de uma função de transferência contínua para discreta no MATLAB é imediata a partir da utilização de uma função conversora.

O segundo modelo criado é semelhante ao primeiro, porém é utilizado um bloco de impedância discreto, obtido a partir da discretização do bloco contínuo do modelo anterior, utilizando a função “c2d”, disponível no MATLAB. O segundo modelo é apresentado na figura 17.

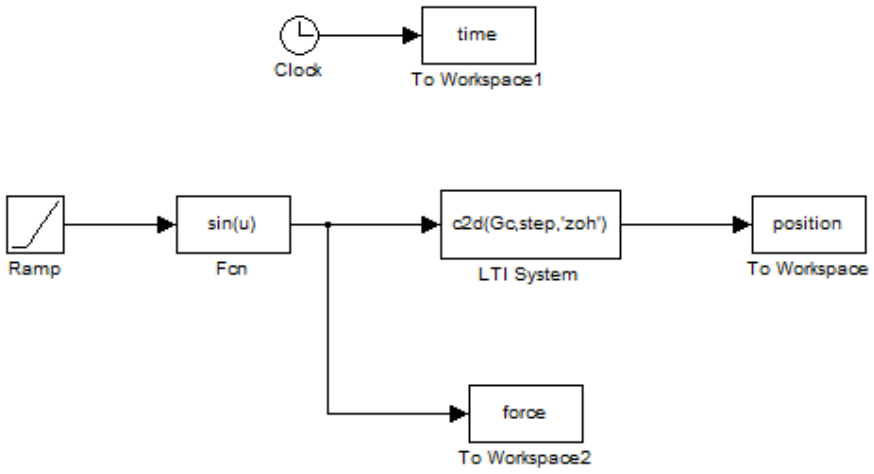


Figura 17 - Modelo do bloco de impedância (discreto)

A partir desse modelo foi criado um *script* que implementa uma equação de diferenças calculada a partir da função de transferência discreta do bloco de impedâncias desejada. Esse *script* plota a resposta a degrau de torque o que possibilitou a verificação da eficácia da equação de diferenças em relação ao modelo discreto já que as respostas eram matematicamente idênticas.

A função de transferência discreta do bloco de impedância desejada é da forma:

$$\frac{\theta(z)}{\tau(z)} = \frac{a \cdot z + b}{c \cdot z^2 + d \cdot z + e}$$

Transcrevendo essa função para uma equação de diferenças equivalente, obtêm-se:

$$\theta(k) = \frac{a \cdot \tau(k-1) + b \cdot \tau(k-2) - d \cdot \theta(k-1) - e \cdot \theta(k-2)}{c}$$

Essa equação é implementada então no projeto em C, onde os parâmetros *a, b, c, d* e *e* são obtidos automaticamente durante a conversão da função de transferência contínua para a discreta.

Inicialmente, para testar a validade prática da equação obtida que implementa o bloco de controle de impedâncias, escolhem-se parâmetros de inércia, rigidez e viscosidade correspondentes a um determinado comportamento dinâmico.

O comportamento dinâmico é então analisado através de uma excitação a degrau de força. O comportamento obtido pelo modelo do MATLAB é então comparado com o comportamento calculado pelo programa em C e com o comportamento dinâmico real do sistema, medido através do potenciômetro localizado na junta do cotovelo.

Para reproduzir com fidelidade um degrau de força, o vetor de força do programa em C foi inicializado com valor nulo e, após o início do programa, forçou-se um valor fixo.

O controle de posição implementado pelo *driver* teve sua aceleração e velocidade máximas configuradas, de forma a permitir que a posição real do mecanismo consiga atingir a posição desejada calculada pelo bloco de controle impedância.

Uma vez que provado que é possível configurar o comportamento do sistema através da comparação dos valores calculados e medidos, efetua-se o projeto de controle.

3.2.5. Projeto de Controle

O projeto de controle se baseia na escolha de parâmetros do comportamento dinâmico, tais como sobressinal e tempo de assentamento dada uma excitação a degrau. Para este projeto, convencionou-se utilizar o tempo de assentamento com valor de 4%. A equação do bloco de impedância desejada pode ser escrita na forma:

$$\frac{\theta(s)}{\tau(s)} = \frac{\omega_n^2}{s^2 + 2 \cdot \zeta \cdot \omega_n \cdot s + \omega_n^2}$$

As fórmulas para o sobressinal (Mp) e tempo de assentamento (Ts) utilizadas são as que seguem:

$$M_p = \varepsilon^{-\left(\frac{\zeta \cdot \pi}{\sqrt{1-\zeta^2}}\right)}$$

$$T_s = \frac{4}{\zeta \cdot \omega_n}$$

Pode-se escrever os valores de viscosidade (B_d) e inércia (J_d) em função da rigidez (K_d), tempo de assentamento e máximo sobressinal:

$$B = \cos^2\left(\tan^{-1}\left(\frac{-\pi}{\ln M_p}\right)\right) \cdot \frac{K \cdot T_s}{2}$$

$$J = \cos^2\left(\tan^{-1}\left(\frac{-\pi}{\ln M_p}\right)\right) \cdot \frac{K \cdot T_s^2}{16}$$

Portanto, define-se para a situação desejada quais o máximo sobressinal, rigidez e tempo de assentamento que devam caracterizar o sistema.

Na seção 4, Resultados, são explorados alguns exemplos práticos implementando essa estratégia.

3.2.6. Estrutura de Programação

Primeiramente, define-se no contexto de programação o que é uma *task*. Uma *task* é uma função ou conjunto de funções a serem agendadas de acordo com um pedido de interrupção do sistema. A ordem de execução é controlada pelo *kernel*. Um sistema *real time* se diferencia dos demais pelo fato de garantir que as *tasks* ocorrerão no tempo determinado.

Programas de tempo real precisam garantir que o tempo de processamento de cada *task* a ser executada não comprometa o agendamento do processamento de outras *tasks*. Por isso, optou-se por elaborar um programa em linguagem C, o mais enxuto possível, valendo-se da estrutura de leitura e escrita da memória e segmentação das *tasks* por similaridade funcional. Nessa estrutura, procura-se controlar o tempo máximo em que cada *task* deve ser realizada, levando em conta que a soma dos tempos de um ciclo de operação não comprometa o pleno funcionamento do sistema.

O diagrama contido na figura 18 ilustra uma representação dos períodos de agendamento e repetibilidade de uma *task* segundo o programa.

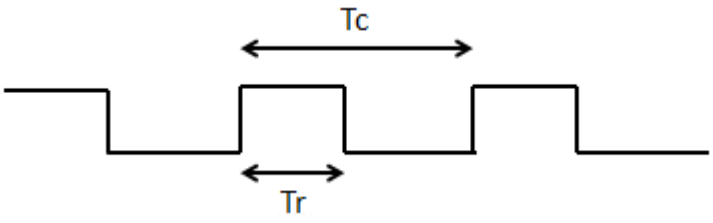


Figura 18 – Períodos de agendamento e repetibilidade

Onde Tc é o período controlado entre cada chamada de uma *task* e Tr é o tempo real de execução de cada *task*. O sistema operacional modificado com o *kernel real time* garante precisão dos intervalos Tc. Ainda assim, é necessário garantir que Tr<Tc, pois isso acarretaria em *overrun*, ou seja, a *task* seria interrompida em algum ponto não conhecido, podendo, por exemplo, afetar a qualidade do controle ou atuação do conjunto.

Além disso, é necessário que Tc seja o mínimo possível, para uma melhor performance, como por exemplo, uma frequência de amostragem maior para o caso do sensor.

Os testes de intervalo de *tasks* foram baseados nas impressões dos intervalos de interesse e, dessa forma, ajustados de maneira iterativa. Chegou-se a um intervalo de 2ms para taxas de amostragem, controle e atuação, que se mostra satisfatório, uma vez que o período dos movimentos humanos é superior a esse valor.

O diagrama contido na figura 19 ilustra a relação entre as classes do programa. A relação destacada em traço pontilhado indica que de fato não há acesso de dados entre cada um dos elementos.

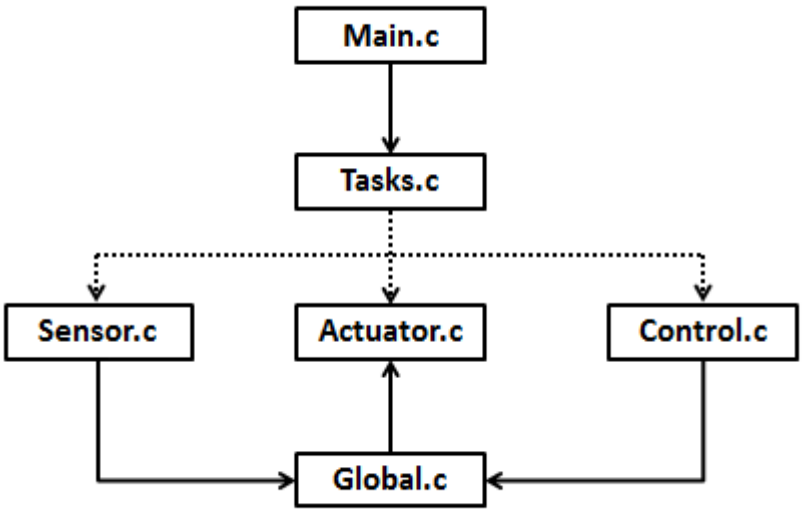


Figura 19 – Relação entre as classes do programa

O programa Main.c contém o menu que permite ao usuário configurar alguns parâmetros desejáveis, tais como habilitar ou desabilitar determinada funcionalidade, alterar o período de execução entre as tarefas ou prioridade.

O programa Global.c contém parâmetros e definições que são utilizadas por um ou mais programas. O intuito é fazer com que estes possam ser acessíveis sem a necessidade de enviá-los como parâmetros entre as *tasks*.

O programa Tasks.c contém todos os parâmetros necessários para inicializar argumentos pertinentes ao agendamento das *tasks*. Tais parâmetros são os mesmos que são configuráveis através do menu implementado em Main.c. É responsável pelo agendamento das *tasks* contidas nos programas Sensor.c, Actuator.c e Control.c, utilizando a API Native do Xenomai.

O programa Sensor.c realiza uma conversão analógica digital, sinal proveniente dos *strain gauges* do exoesqueleto. O sinal é armazenado em um vetor contido no programa Global.h e depois utilizado pelo controle.

Para melhorar a qualidade do sinal recebido pelos sensores, procurou-se implementar um filtro digital e um *dead zone* em torno da posição de equilíbrio da mola, tida como 90º em relação ao antebraço.

O programa Actuator.c realiza uma conversão digital analógica que alimentará o *setpoint* da entrada analógica do driver EPOS2. Tal conversão pode gerar resultados na faixa de 0-5V o que é interpretado pelo driver como um deslocamento angular correspondente, previamente configurado. Uma alternativa para o envio do *setpoint* de posição seria desenvolver um protocolo RS232 ou um protocolo CAN. Optou-se pelo envio do sinal analógico pela maior facilidade de utilização deste, visto que o desenvolvimento de uma estrutura de comunicação usando um protocolo demandaria mais trabalho e, possivelmente, adicionaria mais latência na execução do controle.

O programa Control.c implementa a malha de controle de impedâncias desejada. Para isso, utiliza-se a equação de diferenças, obtida da função de transferência discreta descrita na seção 3.2.4. Utiliza os valores lidos do sensor de força e define os valores de posição a serem enviados pela task actuator.

A listagem do *software* desenvolvido é apresentada no Apêndice A.

3.3. Tratamento do sinal do *Strain Gauge*

3.3.1. Problemas com a instrumentação

A posição do exoesqueleto que satisfaz a impedância desejada é calculada a partir do sinal de força medido. A medição da força entre o braço do usuário e o exoesqueleto é medida a partir da célula de carga instalada no antebraço do exoesqueleto, cujo princípio de funcionamento baseia-se na instalação de *strain gauges* em sua superfície dispostos em Ponte de Wheatstone (ABDUCH e RUIVO, 2012).

Os testes realizados para verificar a medição da força mostraram que existe um ruído de leitura de amplitude considerável. Outro problema observado foi que a Ponte de Wheatstone estava descalibrada, pois o sinal de tensão não era nulo mesmo com o exoesqueleto em repouso em qualquer orientação. Além disso, percebeu-se que a saída do amplificador saturava muito mais rapidamente quando a força era aplicada em determinado sentido em relação ao outro sentido.

Uma hipótese a respeito da razão da descalibração é uma alteração da resistência elétrica de pelo menos um dos resistores da ponte, devido à deterioração ou dano mecânico da colagem dos *strain gauges*, visto que essa se apresenta sem qualquer tipo de proteção a agentes externos.

Outra hipótese é que o comprimento dos fios entre os strain gauges e os pontos de união dos ramos foi realizado de forma a deixar alguns fios significativamente mais longos que outros, alterando a resistência de cada ramo e, portanto a tensão de saída da ponte.

3.3.2. Soluções encontradas

A solução adotada para resolver o desbalanceamento foi proposta pelo professor João Alcino de Andrade Martins do Departamento de Engenharia Naval e Oceânica da Escola Politécnica da USP e consiste em acoplar um divisor de tensão ajustável em paralelo a um dos polos da Ponte de Wheatstone, conforme figura 20. A adição do divisor de tensão permite a regulagem das resistências equivalentes dos ramos AC e BC, dadas, respectivamente, por $\frac{R3R1}{R3+R1}$ e $\frac{R4R2}{R4+R2}$.

A análise matemática das resistências equivalentes permite afirmar que quanto maiores os valores de R3 e R4 escolhidos, mais próximas as resistências equivalentes

dos ramos AC e BC estarão de seus valores originais, anteriores a adição do divisor de tensão, ou seja, R3 e R4, respectivamente. Além disso, quanto maiores R3 e R4, menor é a variação das resistências equivalentes em função da variação de R3 e R4.

Dessa forma, a adição do divisor resistivo com resistências suficientemente grandes em relação aos resistores da ponte pode ser utilizada como calibrador da tensão de saída da ponte, com precisão tão boa quanto maiores os valores das resistências do divisor resistivo.

Para este caso, a resistência nominal do *strain gauge* é de 350Ω e como divisor resistivo, utilizou-se um *trimpot* de resistência de 50KΩ.

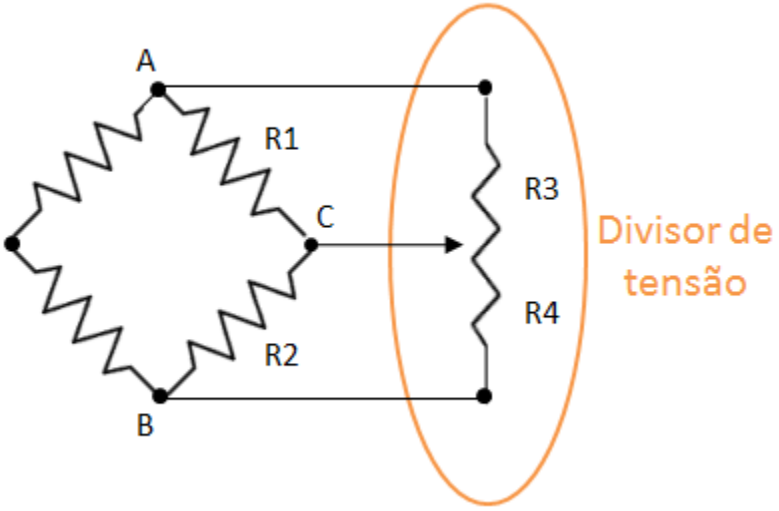


Figura 20 - Circuito de calibração da Ponte de Wheatstone

Uma tentativa de minimizar os efeitos do ruído foi através da inserção de um *deadzone* de leitura implementado por software no programa executado pelo controlador. O ajuste do melhor valor de *deadzone* foi encontrado experimentalmente, analisando a resposta do controle em situações variadas. Chegou-se ao valor de 0,07V, tido como aceitável, partindo do objetivo de minimizar a influência do ruído, porém sem atrapalhar o controle de forma consideravelmente perceptível ao usuário.

4. RESULTADOS

4.1. Tempo Real

Como foi discutido na seção 3.2.6, o tempo de chamada de cada *task* e o tempo real de execução das mesmas é monitorado habilitando-se a variável *TIMEMONITORING*.

O teste consiste na minimização dos tempos de chamada de cada *task*, fazendo com que o tempo em que o programa fique ocioso seja o menor possível, sem que seja observado nenhuma interrupção em alguma *task* que viria a levar o sistema a um estado desconhecido.

A tarefa de controle deve ser realizada na ordem que segue: leitura dos sensores, cálculo da posição desejada pelo controle e envio dos sinais de atuação ao atuador. O resultado impresso na tabela abaixo exemplifica a execução do monitoramento da duração de execução efetiva e erro de chamada das *tasks* para um período de 50ms.

A coluna "*task"* apresenta qual etapa do controle ela executa. A coluna "Duração efetiva" apresenta quanto tempo a *task* efetivamente permaneceu realizando processamento de dados. A coluna "Erro de período" apresenta a diferença entre o tempo de início da execução entre dois ciclos consecutivos da *task* e o período esperado, que para este caso vale 2ms. A coluna *"overruns"* apresenta quantos ciclos de execução foram abortados devido à necessidade de se esperar o término de outra *task* de prioridade maior ou igual terminar a execução.

Assim como no caso da tabela abaixo, foram realizados testes com longos períodos de execução e não foram observados *overruns* para o período utilizado de 2ms.

Task	Duração efetiva	Erro de período	Número de *overruns*
Atuador :	0.134022 ms	0.555 us	"0"
Sensor :	0.130291 ms	0.259 us	"0"
Controle :	0.008993 ms	0.584 us	"0"

Atuador :	0.085227 ms	0.504 us	"0"
Sensor :	0.130671 ms	0.246 us	"0"
Controle :	0.008976 ms	0.332 us	"0"
Atuador :	0.084511 ms	0.062 us	"0"
Sensor :	0.128240 ms	0.386 us	"0"
Controle :	0.008790 ms	0.162 us	"0"
Atuador :	0.082898 ms	0.590 us	"0"
Sensor :	0.129708 ms	0.300 us	"0"
Controle :	0.008825 ms	0.554 us	"0"
Atuador :	0.084336 ms	0.059 us	"0"
Sensor :	0.128246 ms	0.229 us	"0"
Controle :	0.008717 ms	0.041 us	"0"
Atuador :	0.082790 ms	0.053 us	"0"
Sensor :	0.130081 ms	0.348 us	"0"
Controle :	0.008898 ms	0.549 us	"0"
Atuador :	0.084276 ms	0.495 us	"0"
Sensor :	0.128090 ms	0.380 us	"0"
Controle :	0.008699 ms	0.173 us	"0"
Atuador :	0.082567 ms	0.714 us	"0"
Sensor :	0.130280 ms	0.258 us	"0"
Controle :	0.008916 ms	0.434 us	"0"

A tarefa de impressão em tela ou em arquivo é custosa e tende a aumentar o tempo de execução das *tasks*. É possível obter períodos menores, porém com o custo de não se obterem os dados de monitoramento. No entanto, 2ms foi considerado um período de amostragem aceitável visto que o período para os movimentos do braço humano é superior a esse valor.

Afim de verificar o comportamento ou interação com o *hardware*, foi verificado o período de execução das *tasks* com um osciloscópio, para ver se o período atingido era de fato o apresentado pelo controlador. Nesse caso, o cálculo de controle foi alterado para gerar uma tensão de atuação que varia entre dois valores a cada ciclo de execução. A figura 21 foi obtida, onde se percebe que o período atingido para execução de dois ciclos, 4ms, é realmente o dobro do período de 2ms programado.

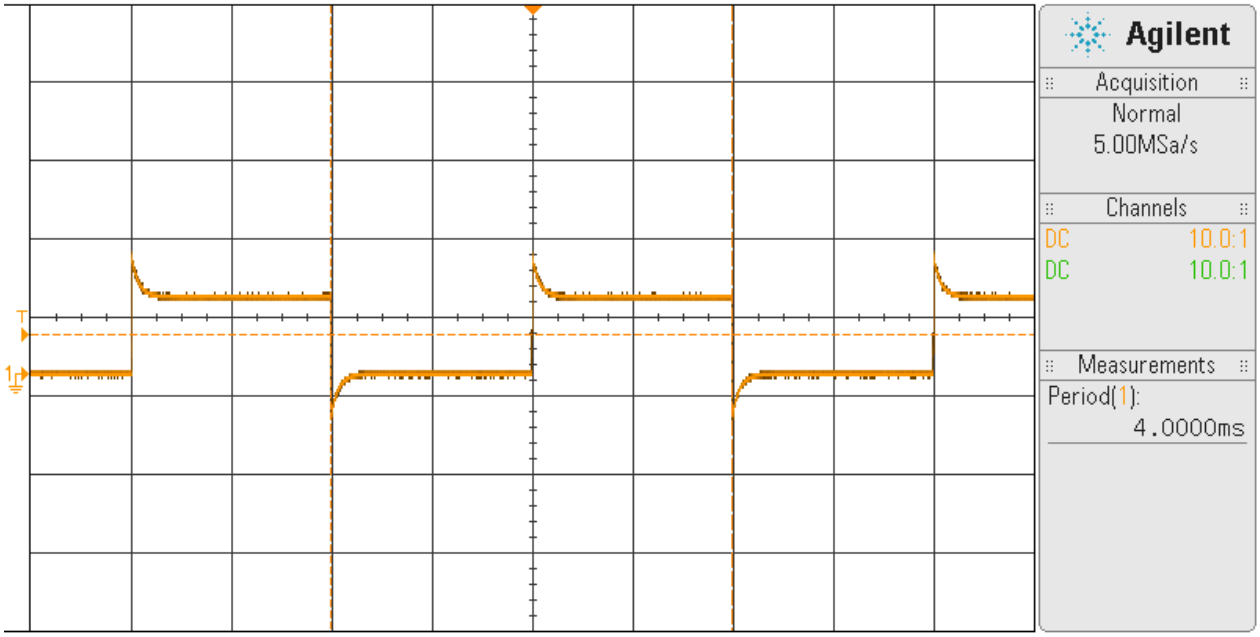


Figura 21 - Sinal gerado para verificação do período de execução das *tasks*

4.2. Variação dos parâmetros do controlador

4.2.1. Malha 1

Inicialmente, foram escolhidos parâmetros para a rigidez, inércia e amortecimento para um comportamento dinâmico arbitrário, excitado por um sinal de força a degrau. A intenção era poder observar se o modelo matemático implementado em MATLAB condizia com o controle discreto implementado em C, ou seja, se a saída do bloco de controle de impedâncias era condizente com o projeto.

Como mencionado na seção 3.2.4, para se reproduzir com fidelidade um sinal de força a degrau, decidiu-se não utilizar a leitura dos *strain gauges* nesse teste e, em substituição, impor o valor de força constante a partir do início do experimento.

Os parâmetros utilizados no teste que caracterizam essa primeira malha foram:

$$K = 0{,}01333\ Nm\ ;\quad J = 4.053 \times 10^{-4}\ Kg \cdot m^2;\quad B = 5 \times 10^{-3} Nm \cdot s\ ;\quad \tau = 0{,}2\ Nm$$

Utilizando-se o modelo do MATLAB, o comportamento dinâmico observado é ilustrado na figura 22.

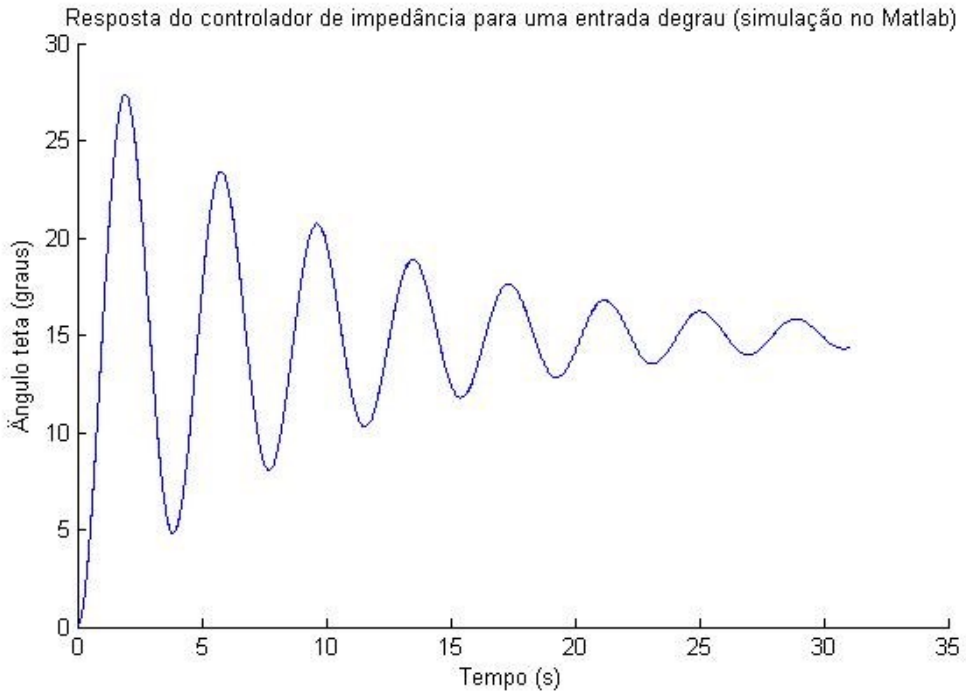


Figura 22 - Comportamento dinâmico, modelo MATLAB, Malha 1

Os valores das constantes do controle discreto relativos ao comportamento do sistema a serem programados no controle implementado em C, discutidos na seção 3.2.4 são os seguintes:

$$a = 3.999463165010274 \times 10^{-4}$$
$$b = 3.998929938615196 \times 10^{-4}$$
$$c = 1$$
$$d = 1.999589415465196$$
$$e = 0.999600079989334$$

Os dados obtidos da posição desejada calculada através do controle implementado no programa em C estão ilustrados na Figura 23.

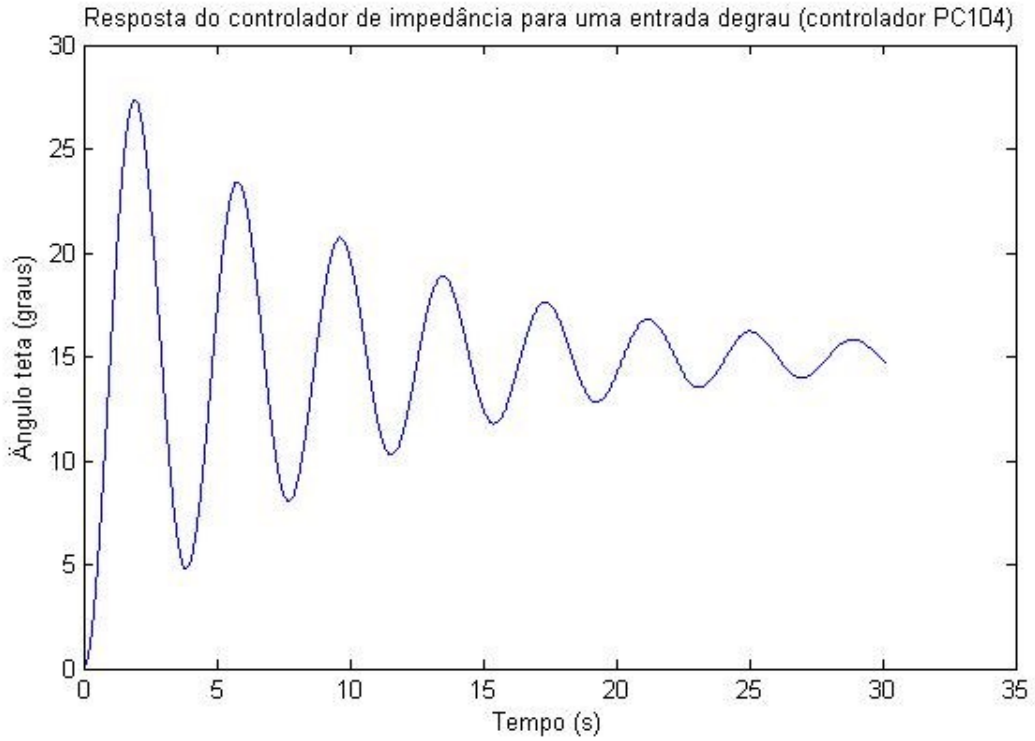


Figura 23 - Comportamento dinâmico, controlador em C, Malha 1.

As figuras 24 e 25 ilustram o gráfico obtido através da leitura do potenciômetro encontrado na junta do cotovelo do exoesqueleto. A figura 24 ilustra o sobressinal obtido através do potenciômetro e a figura 25 ilustra o período do movimento observado.

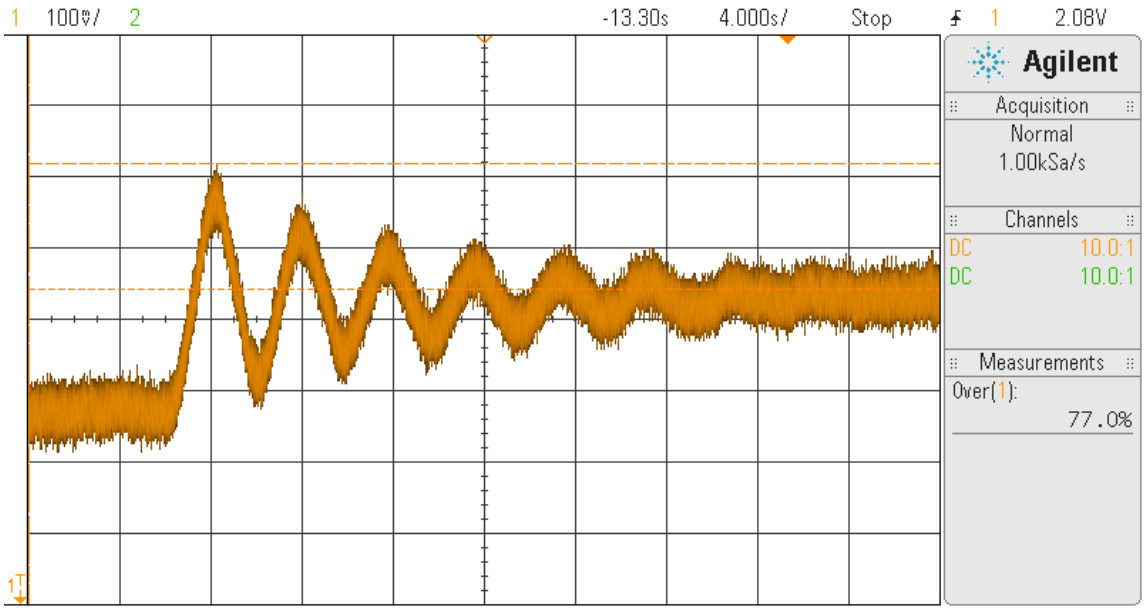


Figura 24 - Comportamento dinâmico, potenciômetro, sobressinal, Malha 1.

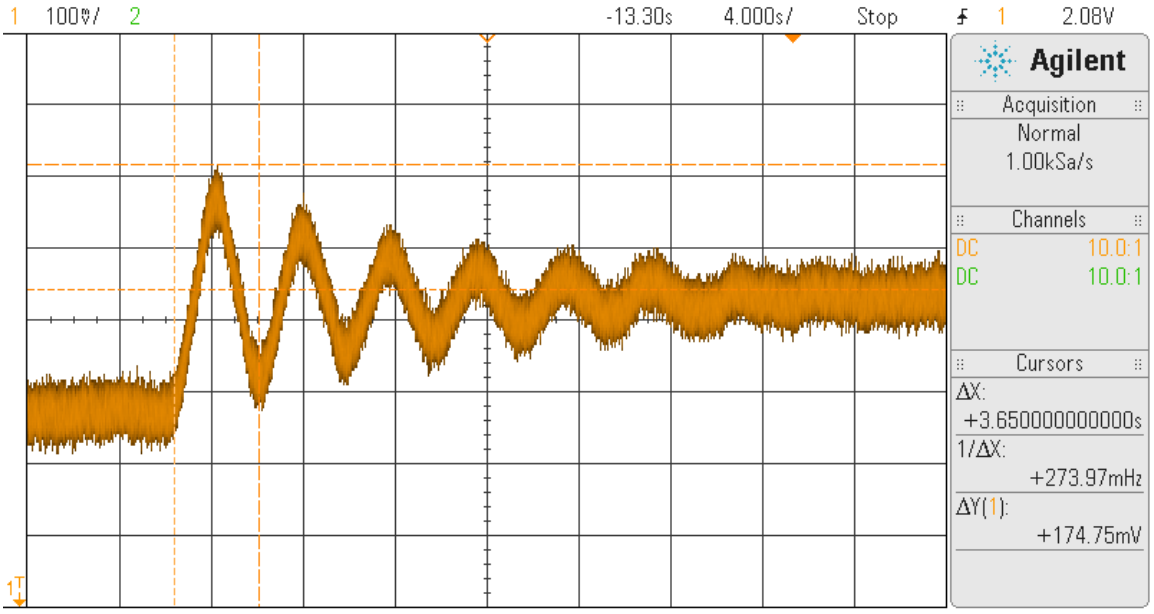


Figura 25 - Comportamento dinâmico, potenciômetro, período, Malha 1

A resposta do sistema simulada no Matlab apresentou *overshoot* de 82% e período de 3,85 segundos. As medidas apresentadas acima, obtidas com o osciloscópio a partir da posição real do exoesqueleto, forneceram *overshoot* de 77% e período de 3,65 segundos, aproximadamente. Supõe-se que as discrepâncias estejam associadas

às imprecisões de fabricação mecânica do exoesqueleto (folgas e atrito) e a imprecisão de leitura do osciloscópio, inclusive devido ao ruído presente no sinal.

O teste proposto com a Malha 1 visa mostrar que é possível configurar o comportamento dinâmico do sistema através da alteração dos parâmetros implementados no bloco de controle de impedâncias.

4.2.2. Malha 2

Levando-se em conta o procedimento descrito na 3.2.5, definem-se os parâmetros de rigidez, amortecimento e inércia baseados nos requisitos de projeto (máximo sobressinal e tempo de assentamento).

Procura-se para este projeto as especificações de sobressinal (Mp) e tempo de assentamento (Ts) abaixo:

$$M_p = 0.02$$

$$T_s = 0.2\ s$$

Seguindo as equações expostas na seção 3.2.5 e adotando-se a mesma rigidez da Malha 1, encontra-se os seguintes parâmetros de viscosidade e inércia:

$$K = 1.333 \times 10^{-2} Nm\ ; \qquad J = 2.026 \times 10^{-5}\ Kg \cdot m^2; \quad B = 8.106 \times 10^{-4} Nm \cdot s\ ;$$
$$\tau = 0.2\ Nm$$

Utilizando-se o modelo do MATLAB, o comportamento dinâmico observado é ilustrado na figura 26.

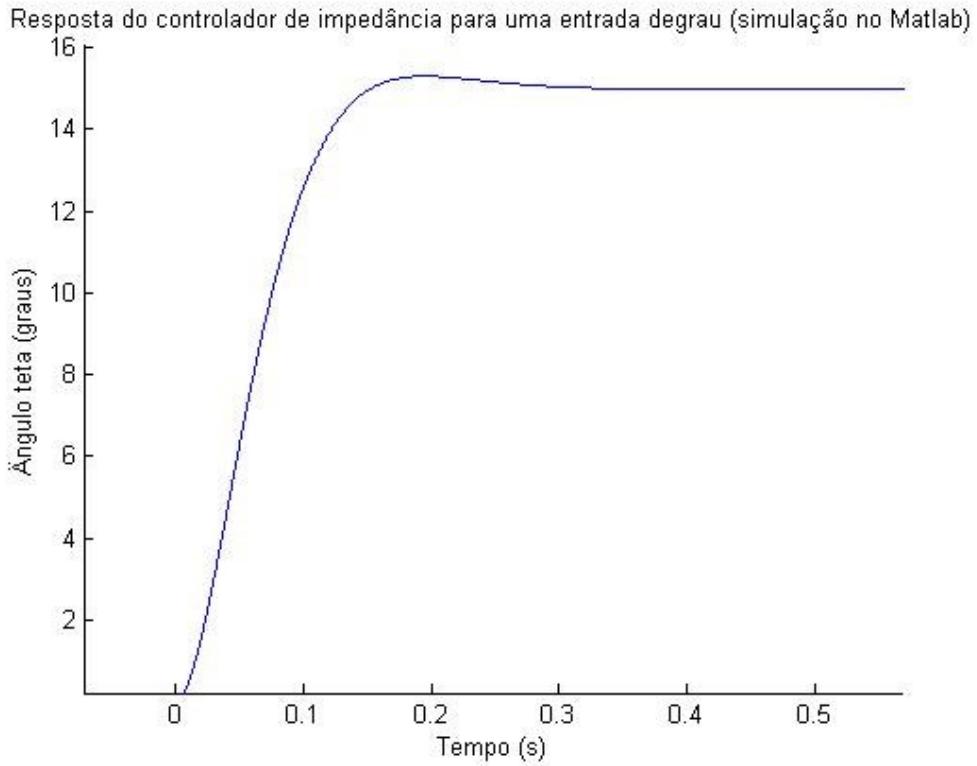


Figura 26 - Comportamento dinâmico, modelo MATLAB, Malha 2.

Os valores das constantes do controle discreto relativos ao comportamento do sistema a serem programados no controle implementado em C, discutidos na seção 3.2.5, são os seguintes:

$$a = 0.096093397200609$$
$$b = 0.093564670856204$$
$$c = 1$$
$$d = -1.920587572145879$$
$$e = 0.923116346386636$$

Os dados obtidos da posição desejada calculada através do controle implementado no programa em C estão ilustrados na Figura 27.

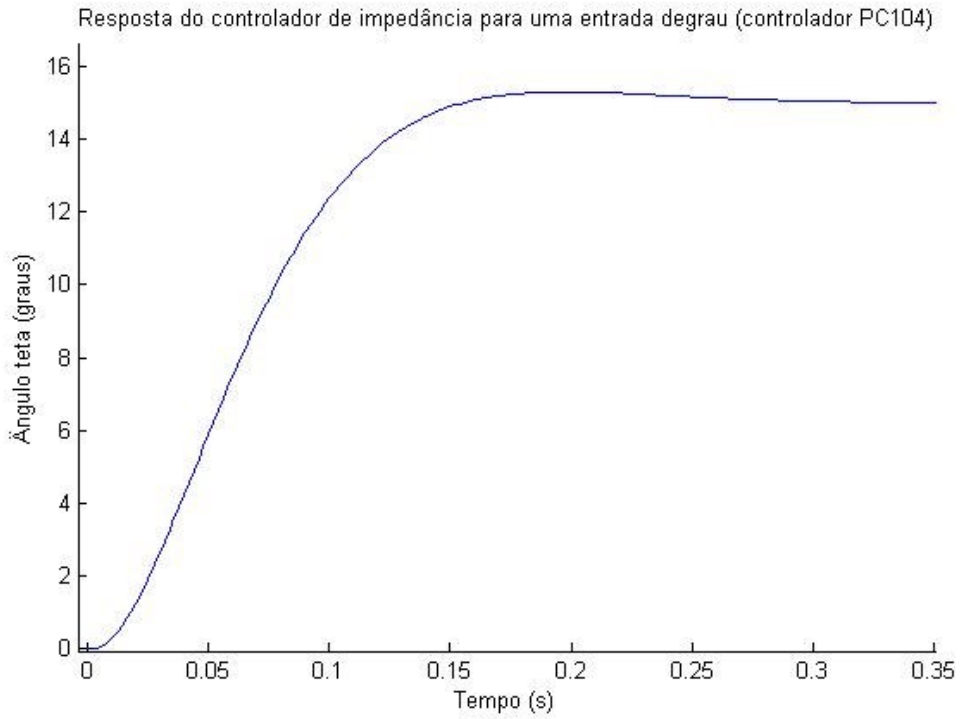


Figura 27 - Comportamento dinâmico, controlador em C, Malha 2.

A figura 28 ilustra o gráfico obtido através da leitura do potenciômetro encontrado na junta do cotovelo do exoesqueleto. Não foi possível observar o sobressinal, pois a amplitude do ruído está na mesma ordem de grandeza deste. Porém, é possível perceber que o tempo de assentamento obtido de aproximadamente 198ms está condizente com o esperado de 200 ms.

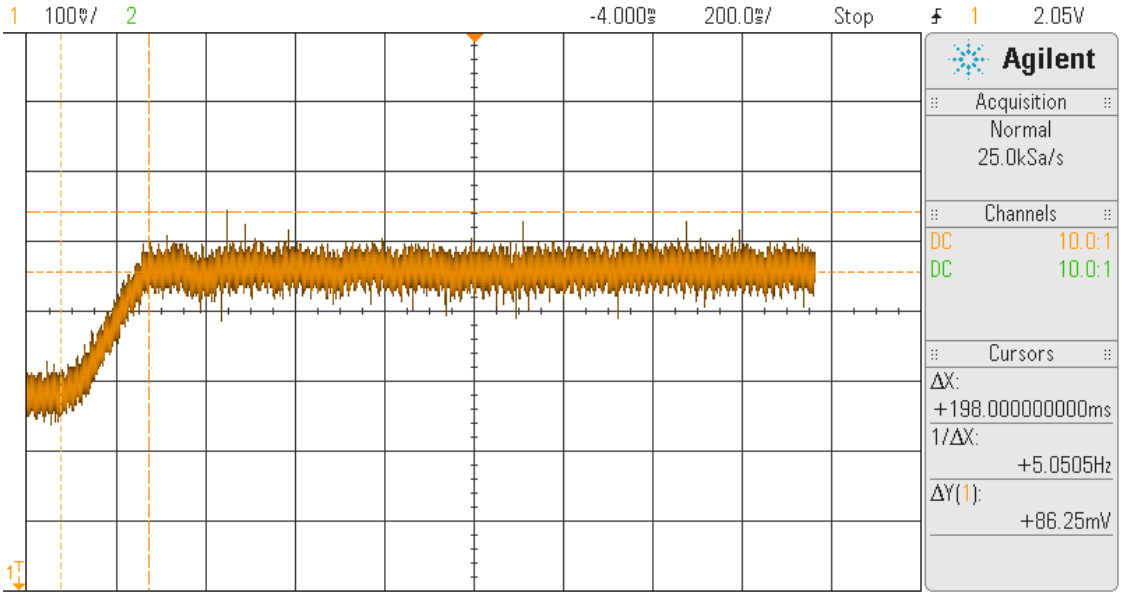


Figura 28 - Comportamento dinâmico, potenciômetro, Malha 2.

O teste proposto com a Malha 2 visa mostrar que é possível configurar o comportamento dinâmico do sistema para se adequar aos parâmetros de projeto de controle.

4.3. Controle Seguidor

Procura-se através do controle seguidor, minimizar a força de interação entre o usuário e o exoesqueleto (ARAÚJO, TANNURI e FORNER-CORDERO, 2012). Para tal, levando em conta o modelo adotado, massa-mola-amortecedor, deve-se considerar as possíveis causas de força entre o usuário e o exoesqueleto.

Primeiramente, uma força relativa à mola retornando à posição de equilíbrio. Essa força é proporcional ao valor de rigidez escolhido, portanto, escolhe-se o menor valor de rigidez possível.

Outra força surge devido à incapacidade do exoesqueleto seguir a posição do braço do usuário. Por exemplo, se a força que o usuário exerce puder ser aproximada por um degrau, o exoesqueleto deve atingir a posição relacionada no menor tempo

possível e com baixas oscilações. Ou seja, procuram-se parâmetros de máximo sobressinal e tempo de assentamento pequenos.

Juntamente com o modelo do MATLAB utilizado para desenvolver as malhas I e II e com experimentos com usuários, encontrou-se iterativamente uma solução para o controle seguidor proposta a seguir:

$$K = 1.00 \times 10^{-7} Nm\,; \qquad J = 8.29 \times 10^{-14}\, Kg \cdot m^2; \quad B = 1.66 \times 10^{-10} Nm \cdot s\,;$$

Com esses valores foi possível movimentar o exoesqueleto com facilidade, este não impunha muita resistência ao movimento, como exemplificado na figura 29.

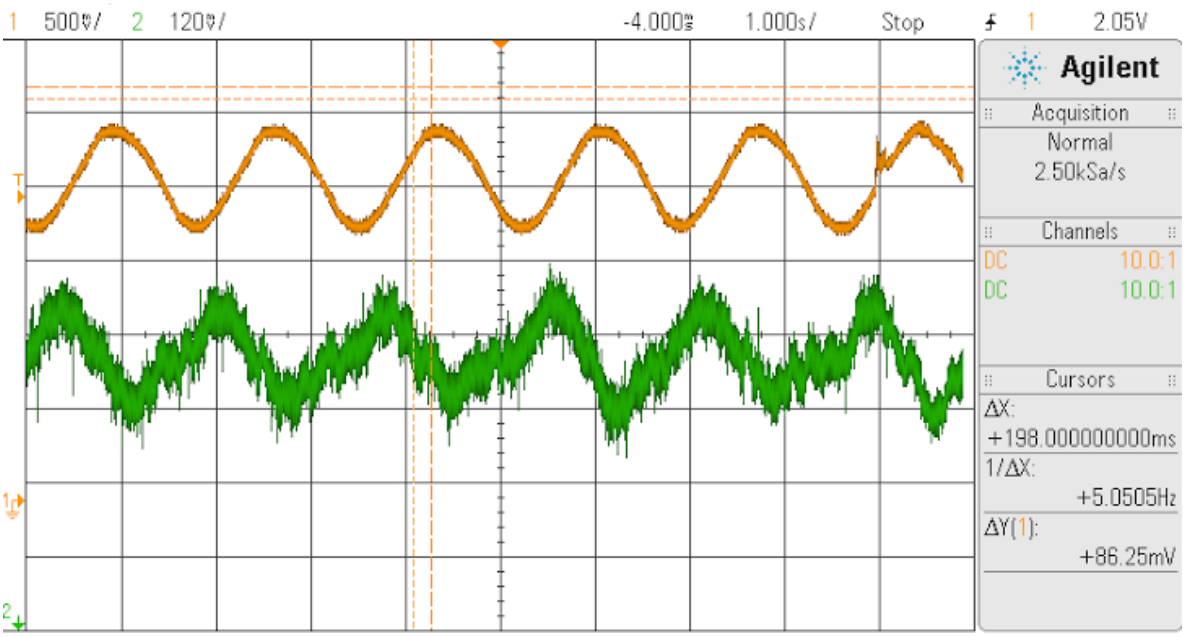


Figura 29 - Posição (laranja) e torque (verde) para o controle seguidor

O primeiro sinal foi medido no potenciômetro acoplado ao cotovelo do exoesqueleto e, o segundo é o sinal de leitura do *strain gauge* amplificado.

É possível afirmar que foi possível ao usuário movimentar o braço através do curso inteiro do exoesqueleto ao mesmo tempo em que o mesmo efetuava forças comparáveis à amplitude do ruído do *strain gauge*. Fisicamente, pode-se observar que o usuário é capaz de movimentar o exoesqueleto com baixo esforço.

5. DISCUSSÕES

Os testes analisando o tempo de execução das *tasks* e a não existência de *overruns* permitiram a verificação do correto agendamento de execução de cada etapa de controle (sensoriamento, cálculo do controle e atuação) e da execução em tempo real.

O teste proposto com a Malha 1 mostrou a concordância entre a resposta simulada no MATLAB e a calculada pelo controlador. Finalmente, a resposta do sistema real controlado se apresentou condizente com a simulada, dentro das limitações envolvidas ao exoesqueleto e ao seu sensoriamento, a partir da análise do período de oscilação e do sobressinal. Verificou-se, portanto, que é possível configurar o comportamento dinâmico do sistema através da alteração dos parâmetros implementados no bloco de controle de impedâncias.

O teste proposto com a Malha 2 mostrou a adequação do sistema real aos parâmetros de projeto de controle, novamente, dentro das limitações mecânicas e de sensoriamento.

A implementação do controle seguidor proposto visou verificar a eficácia do controle, incluindo o *feedback* de força. O sistema obtido para o controle seguidor contava com baixos valores de rigidez, o que amplificava o ruído observado na leitura dos *strain gauges*, introduzindo características dinâmicas indesejáveis, dentre elas, a introdução de ruídos de alta frequência na atuação.

Muito dos ruídos de alta frequência na atuação foram reduzidos regulando a rigidez desejada, a aceleração máxima produzida pelo motor através do driver e pela utilização de um *deadzone* na leitura do sinal de força. Porém uma componente de baixa frequência ainda era percebida na oscilação do exoesqueleto. Em parte, essa oscilação se dá pelos acoplamentos mecânicos do próprio dispositivo, que contam com jogo e alguns apresentam desgaste devido aos testes realizados ao longo de três anos.

Esses problemas puderam ser amenizados com a calibração dos *strain gauges* e com a refixação de algumas partes mecânicas. Porém, durante os testes,

frequentemente a amplitude da oscilação aumentava, sendo necessária recalibração da célula carga.

Apesar dos problemas mencionados anteriormente, o controle em si foi considerado satisfatório, uma vez que os testes práticos levaram a obtensão de parâmetros aceitáveis para o controle seguidor, levando em conta os critérios de mínimo ruído e critério de mínima rigidez. Observou-se que a posição do exoesqueleto era influenciada unicamente pela força aplicada, dentro das limitações físicas apresentadas pelo exoesqueleto. Com isso, concluiu-se que a estrutura de *software* e controle adotadas realizam seu objetivo proposto de forma satisfatória, porém o funcionamento do sistema global é limitado principalmente pelas deficiências mecânicas e de sensoriamento.

6. CONCLUSÕES

A implementação de um controle de impedâncias é uma tarefa que envolve diversas áreas da engenharia, como computação, eletrônica, mecânica e controle. Quanto à parte que envolve computação, percebeu-se que a seleção e implementação do ambiente de trabalho - que compreende o sistema operacional, a estrutura de controle e a forma de programação – não são uma tarefa simples, visto que envolveram a maior parte do tempo gasto.

A inexperiência dos alunos fez com que o trabalho de implementação do ferramental de *software* fosse prolongado. Não havia experiência prévia em relação ao uso de sistema operacional Linux e seus comandos, tão pouco quanto à modificação de um *kernel*, que é uma etapa crucial que exige atenção a detalhes e tempo. Dentre os textos encontrados pelos alunos, considerou-se que a teoria de controle de impedâncias possui pouco material com foco em nível didático, principalmente em relação a livros. Muito do tempo do projeto foi gasto com a adaptação dos alunos ao o conjunto de *softwares* que serão utilizados e a compreensão, ainda que incompleta, da teoria de controle de impedâncias.

A existência de problemas na estrutura mecânica, como folgas, e na instrumentação, como ruídos, se apresentou como um empecilho real ao desenvolvimento do controle, evidenciando que a execução de um projeto de engenharia mecatrônica completo envolve a resolução de problemas de diversas naturezas, sendo que todos os componentes influenciam a resposta final de forma significativa. A busca na solução rápida e aceitável desses problemas, como, por exemplo, com a utilização de *deadzone* e calibração do sinal da célula de carga, permitiu a continuidade do desenvolvimento do controle, embora a resposta mecânica do sistema possa ser melhorada consideravelmente.

Apesar disso, verificou-se a factibilidade da aplicação prática da teoria de controle linear discreto aplicada ao controle de impedâncias em sistemas mecânicos,

mesmo na existência dos empecilhos citados, além de outros efeitos de natureza não linear, inerentes ao mecanismo utilizado.

Como sugestão para trabalhos futuros a partir do mesmo projeto, apresenta-se:

- Melhoria da instrumentação de medição de força, de forma a reduzir o ruído produzido e diminuir ainda mais a possibilidade de saturação de leitura.
- Aumentar o espaço de trabalho do exoesqueleto e passar a considerar a cinemática não linear nas regiões extremas de movimentação.
- Reconstrução ou reprojeto de alguns componentes mecânicos com o objetivo de reduzir folgas e proporcionar menores deflexões.
- Extensão da estrutura de controle e programação adotadas para mais graus de liberdade.
- Implantação de uma bancada de testes para avaliação mais precisa da eficiência do controle.
- Realização de mais testes com usuários reais em diferentes situações para análise do sistema motor humano, visto que esse era um dos objetivos desse projeto, que não foi atingido.

7. REFERÊNCIAS BIBLIOGRÁFICAS

ABDUCH, I. F.; RUIVO, J. P. P., 2012. *Controle de Impedâncias para exoesqueleto robótico de membro superior para estudo de controle motor humano*. Monografia, Trabalho de conclusão de curso em Engenharia Mecatrônica, Escola Politécnica da Universidade de São Paulo.

AGUIRRE-OLLINGER, G,; COLGATE, J.; PESHKIN, M.; GOSWAMI, A., 2012. Inertia Compensation Control of a One-Degree-of-Freedom Exoskeleton for Lower-Limb Assistance: Initial Experiments, IEEE Transactions on Neural Systems And Rehabilitation Engineering, Vol. 20, No. 1, p. 1534-4320.

ARAUJO A.A. M., TANNURI, E.A.; FORNER-CORDERO, A. 2012. Simulation of model-based impedance control applied to a biomechatronic exoskeleton with shape memory alloy actuators. The Fourth IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics Roma, Italy. June 24-27.

AREVALO, J.C.; GARCIA, E., 2012. Impedance Control for Legged Robots: An Insight Into the Concepts Involved, *IEEE Transactions on Systems, Man, and Cybernetics*, *Part C: Applications and Reviews*, vol. 42, no. 6, p. 1400-1411.

BROWN, J. H.; MARTIN, BRAD, 2012. How fast is fast enough? Choosing between Xenomai and Linux for real-time applications, Rep Invariant Systems inc., USA.

N. HOGAN, 1985. Impedance control: an approach to manipulation, Journal of Dynamic Systems, Measurement and Control,.107 , pp. 1-24.

HOGAN, N., 1987. Stable Execution of contact Tasks Using Impedance Control. Department of Mechanical Engineering and Laboratory for Manufacturing and Productivity, Massachussetts Institute of Technology.

YASUTOMI, Y. A.; MIRANDA, W. B. A., 2011. *Exoesqueleto Robótico de membro superior para estudo de controle motor humano*. Monografia, Trabalho de conclusão de curso em Engenharia Mecatrônica, Escola Politécnica da Universidade de São Paulo.

FORNER-CORDERO,A..; MIRANDA, A.B.; YASUTOMI,A.Y.; ROSSI,L.F.; OLIVEIRA,A.L.L.; HESS-COELHO; T.A., 2011. *Upper Limb exoesqueleton for motor control,* 21st Brazilian Congress of Mechanical Engineering.

APÊNDICE A – LISTAGEM DO SOFTWARE E SCRIPTS DO MATLAB

PROGRAMA EM C

actuator.c:

```
#include "actuator.h"

int A_Init()
{
        channel = 0;
        board_type = DSC_DMM16AT;

   //Ensure that all the members of the structure are initialized to 0.
   memset(&dsccb, 0, sizeof(DSCCB));
   dsccb.base_address = 0x300;
   dsccb.int_level = 7;

   A_Core();

   return 0;
}

int A_Calibration()
{
        rt_printf( "Calibração dos conversores DA... \n\n" );

   memset(&dsccb, 0, sizeof(DSCCB));
   dsccb.base_address = 0x300;
   dsccb.int_level = 3;

        dsccb.boardtype = DSC_DMM16AT;
        dsccb.dma_level = 1;
        dsccb.clock_freq = 10000000;

   if( dscInit( DSC_VERSION ) != DE_NONE )
   {
         dscGetLastError(&errorParams);
         rt_printf("dscInit error: %s %s\n", dscGetErrorString(errorParams.ErrCode), er-
rorParams.errstring );
         return 0;
   }

        //Inicializa a Placa
   if(dscInitBoard(DSC_DMM16AT, &dsccb, &dscb)!= DE_NONE)
   {
         dscGetLastError(&errorParams);
```

```
        rt_printf("dscInitBoard error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
errorParams.errstring );
         return 0;
   }

        memset(&dscdacalparams, 0, sizeof(DSCDACALPARAMS));
        dscdacalparams.darange = 65535;

        if( ( result = dscDAAutoCal( dscb, &dscdacalparams ) ) != DE_NONE )
        {
                dscGetLastError(&errorParams);
                rt_printf("dscDAAutoCal error: %s %s\n", dscGetError-
String(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }

        if( ( result = dscDACalVerify( dscb, &dscdacalparams ) ) != DE_NONE )
        {
                dscGetLastError(&errorParams);
                rt_printf("dscDACalVerify error: %s %s\n", dscGetError-
String(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }

        printf( "Offset Error: %f, Gain Error: %f\n", dscdacalparams.offset, dscdacal-
params.gain );

        if ( fabs( dscdacalparams.offset ) > 2.0 ||
                 fabs( dscdacalparams.gain ) > 2.0 )
                rt_printf( "Value for offset or gain exceeded specified tolerance\n" );
        else rt_printf( "Values for offset and gain met specified tolerance\n" );

        dscFree();

        rt_printf("Calibração dos conversores DA... COMPLETA\n\n" );

        return 0;
}

int A_Kill()
{
        dscFree();
        return 0;
}
```

```
int A_Core()
{
   if( dscInit( DSC_VERSION ) != DE_NONE )
   {
          dscGetLastError(&errorParams);
          rt_printf("dscInit error: %s %s\n", dscGetErrorString(errorParams.ErrCode), er-
rorParams.errstring );
          return 0;
   }

         //Inicializa a Placa
   if(dscInitBoard(board_type, &dsccb, &dscb)!= DE_NONE)
   {
          dscGetLastError(&errorParams);
         rt_printf("dscInitBoard error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
errorParams.errstring );
          return 0;
   }

   // send DACODE of 4095 - Full scale to channel 0.
   dscDAConvert(dscb, channel , (int)4095*((SETPOINT[2])+30)/60);

   dscFree();

         return 0;
}
```

actuator.h:

```
/*
 * actuator.h
 *
 *  Created on: Oct 9, 2013
 *      Author: root
 */

#ifndef ACTUATOR_H_
#define ACTUATOR_H_

#include "dscud.h"
#include "global.h"

DSCB dscb;
BYTE channel;
DSCDACODE output_code;
```

```
BYTE result;
DSCCB dsccb;
ERRPARAMS errorParams;
BYTE board_type;

DSCADSETTINGS dscadsettings;    // structure containing A/D conversion settings
DSCDACALPARAMS dscdacalparams;  // structure containing auto-calibration settings

int A_Init();
int A_Core();
int A_Kill();

#endif /* ACTUATOR_H_ */
```

control.c:

```
#include "control.h"

int C_Init()
{
        SETPOINT[0] = 0;
        SETPOINT[1] = 0;
        SETPOINT[2] = 0;

        a = 6.820326068784110e+06;
        b = 1.700712897992126e+06;
        c = 1;
        d = -0.166211742211110;
        e = 0.018315638888734;

        i=0;

        return 0;
}

int C_Kill()
{
        return 0;
}

int C_Core()
{
        //MALHA COMPLETA
        double position;
```

```
        SETPOINT[0] = SETPOINT[1];
        SETPOINT[1] = SETPOINT[2];

        position = a*TORQUE[1]+b*TORQUE[0]-d*SETPOINT[1]-e*SETPOINT[0];
        position = position/c;

        SETPOINT[2]=position;

    if (SETPOINT[2] >= 30) SETPOINT[2] = 30;
    if (SETPOINT[2] <= -30) SETPOINT[2] = -30;

    rt_printf("\t%5.8f \t %5.8f\n ",SETPOINT[2], TORQUE[2]);

        return 0;
}
```

control.h:

```
#ifndef CONTROL_H_
#define CONTROL_H_

#include "global.h"

//Parâmetros de controle discreto
double a;
double b;
double c;
double d;
double e;

int i;

int C_Init();
int C_Kill();
int C_Core();

#endif /* CONTROL_H_ */
```

global.h:

```
#ifndef GLOBAL_H_
#define GLOBAL_H_

#define NUMBEROFTASKS 4
```

```
//ID das Tasks
#define SENSOR          1
#define CONTROL         2
#define ACTUATOR        3
#define TEST            4

//Conversão do Timer
#define NS2MS 1000000 //Ns para Ms
#define NS2US 1000    //Ns para Us

//Opções iniciais
int MONITORING;
int SENSORING;

//Valores a ser enviado do Controle para o Atuador [k-2,k-1,k]
double SETPOINT[3];

//Valores lido do Strain Gage [k-2,k-1,k]
double TORQUE[3];

//DeadZone
double DEADZONE;

//Referência de calibração
double VOLTAGEREFERENCE;

#endif /* GLOBAL_H_ */
```

include.h:

```
#ifndef DEFINES_H_
#define DEFINES_H_

#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/io.h>
#include <native/task.h>
#include <native/timer.h>
#include <native/heap.h>
#include <rtdk.h>
```

```
#endif /* DEFINES_H_ */
```

main.c:

```
#include "tasks.h"

//Declarações
void calibracao();
void priority();
void period();
void enable();
void menu();
void config();
void ParametrosDasTasks();

//Implementações
void calibracao()
{
	int choice = 0;

	while (choice!=5){
		printf("Calibração\n");
		printf("1. Calibração dos conversores DA\n");
		printf("2. Calibração dos conversores AD\n");
		printf("3. Voltar\n\n");

		scanf ("%d",&choice);

		if(choice==1) A_Calibration();
		else if(choice==2) S_Calibration();
		else if(choice==3) break;
		else printf("Opção Inválida\n\n");
	}
}

void priority()
{
	int choice = 0;

	while (choice!=5){
		printf("Prioridade das Tasks [0-100]\n");
		printf("1. PRIORITY[SENSOR]   =  %d\n",PRIORITY[SENSOR]);
		printf("2. PRIORITY[CONTROL]  =  %d\n",PRIORITY[CONTROL]);
	    printf("3. PRIORITY[ACTUATOR] =  %d\n",PRIORITY[ACTUATOR]);
		printf("4. PRIORITY[TEST]     =  %d\n",PRIORITY[TEST]);
```

```
		printf("5. Voltar\n\n");

		scanf ("%d",&choice);

		if(choice==1) scanf ("%d",&PRIORITY[SENSOR]);
		else if(choice==2) scanf ("%d",&PRIORITY[CONTROL]);
		else if(choice==3) scanf ("%d",&PRIORITY[ACTUATOR]);
		else if(choice==4) scanf ("%d",&PRIORITY[TEST]);
		else if(choice==5) break;
		else printf("Opção Inválida\n\n");
	}
}

void period(){
	int choice = 0;

	while (choice!=5){
		printf("Período das Tasks [ms]\n");
		printf("1. PERIOD[SENSOR]   =  %d\n",PERIOD[SENSOR]);
		printf("2. PERIOD[CONTROL]  =  %d\n",PERIOD[CONTROL]);
		printf("3. PERIOD[ACTUATOR] =  %d\n",PERIOD[ACTUATOR]);
		printf("4. PERIOD[TEST]     =  %d\n",PERIOD[TEST]);
		printf("5. Voltar\n\n");

		scanf ("%d",&choice);

		if(choice==1) scanf ("%d",&PERIOD[SENSOR]);
		else if(choice==2) scanf ("%d",&PERIOD[CONTROL]);
		else if(choice==3) scanf ("%d",&PERIOD[ACTUATOR]);
		else if(choice==4) scanf ("%d",&PERIOD[TEST]);
		else if(choice==5) break;
		else printf("Opção Inválida\n\n");
	}
}

void enable(){
	int choice = 0;

	while (choice!=5){
		printf("Habilita/Desabilita Task\n");
		printf("1. ENABLETASK[SENSOR]   =  %d\n",ENABLETASK[SENSOR]);
		printf("2. ENABLETASK[CONTROL]  =  %d\n",ENABLETASK[CONTROL]);
		printf("3. ENABLETASK[ACTUATOR] =
%d\n",ENABLETASK[ACTUATOR]);
		printf("4. ENABLETASK[TEST]     =  %d\n",ENABLETASK[TEST]);
```

```
		printf("5. Voltar\n\n");

		scanf ("%d",&choice);

		if(choice==1) scanf ("%d",&ENABLETASK[SENSOR]);
		else if(choice==2) scanf ("%d",&ENABLETASK[CONTROL]);
		else if(choice==3) scanf ("%d",&ENABLETASK[ACTUATOR]);
		else if(choice==4) scanf ("%d",&ENABLETASK[TEST]);
		else if(choice==5) break;
		else printf("Opção Inválida\n\n");
	}
}

void menu(){
	int choice = 0;

	while (choice!=2){
		//printf("Controle de Impedâncias\n");
		//printf("1. Configurações\n");
		//printf("2. Executar\n\n");

		scanf ("%d",&choice);

		if(choice==1) config();
		if(choice!=1 && choice!=2)	printf("Opção Inválida\n\n");
	}
}

void config(){
	int choice = 0;

	while (choice!=3){
		printf("Configurações\n");
		printf("1. Parâmetros das Tasks\n");
		printf("2. Calibração dos conversores AD/DA\n");
		printf("3. MONITORING = %d\n",MONITORING);
		printf("3. MONITORING = %d\n",SENSORING);
		printf("5. Voltar\n\n");

		scanf ("%d",&choice);

		if(choice==1) ParametrosDasTasks();
		else if(choice==2) calibracao();
		else if(choice==3) scanf ("%d",&MONITORING);
		else if(choice==4) scanf ("%d",&SENSORING);
```

```
                else if(choice==5) break;
                else printf("Opção Inválida\n\n");
        }
}

void ParametrosDasTasks(){
        int choice = 0;

        while (choice!=4){
                printf("Parâmetros das Tasks\n");
                printf("1. Habilitar/Desabilitar\n");
                printf("2. Período [ms]\n");
                printf("3. Prioridade [0-100]\n");
                printf("4. Voltar\n\n");

                scanf ("%d",&choice);

                if(choice==1) enable();
                else if(choice==2) period();
                else if(choice==3) priority();
                else if(choice==4) {}
                else printf("Opção Inválida\n\n");
        }
}

int main(int argc, char* argv[]) {

        A_Init();

        MONITORING = 0;
        SENSORING = 0;

        T_Xenomai();
        T_Init();

        T_Create();
        menu();

        T_Start();
        T_Kill();

//      sleep(1);
        pause();

        return 0;
```

```
}
```

sensor.c:

```
#include "sensor.h"

int S_Init()
{
	dsccb.base_address=0x300;
	dsccb.int_level = 7;

	memset(&dscadsettings, 0, sizeof(DSCADSETTINGS));

	dscadsettings.range = RANGE_5;
	dscadsettings.gain = GAIN_1;
	dscadsettings.load_cal = (BYTE)FALSE;
	dscadsettings.current_channel = 0;

	dscadscan.low_channel = 0;
	dscadscan.high_channel = 0;
	dscadscan.gain = dscadsettings.gain;

	samples = (DSCSAMPLE*)malloc( sizeof(DSCSAMPLE) * (
dscadscan.high_channel - dscadscan.low_channel + 1 ) );

	VOLTAGEREFERENCE = 2.45;

	TORQUE[0] = 0;
	TORQUE[1] = 0;
	TORQUE[2] = 0;

	return 0;
}

int S_Calibration(){
	int i;

	dsccb.base_address=0x300;
	dsccb.int_level = 3;

	rt_printf( "Calibração dos conversores AD...\n\n" );

	if( dscInit( DSC_VERSION ) != DE_NONE )
	{
		dscGetLastError(&errorParams);
```

```
                rt_printf( "dscInit error: %s %s\n", dscGetError-
String(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }

        if(dscInitBoard(DSC_DMM16AT, &dsccb, &dscb)!= DE_NONE)
        {
                dscGetLastError(&errorParams);
                rt_printf( "dscInitBoard error: %s %s\n", dscGetError-
String(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }

        dscautocal.adrange = 0xFF;
        dscautocal.use_eeprom = (BYTE)TRUE;
        dscautocal.boot_adrange = 0;

        if( (result = dscADAutoCal( dscb, &dscautocal )) != DE_NONE )
        {
                dscGetLastError(&errorParams);
                rt_printf("dscADAutoCal error: %s %s\n", dscGetError-
String(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }

        for( i = 0; i < 16; i++ )
        {
                if ( ( i > 3 && i < 8 )  )
                        continue;

                dscautocal.adrange = i;
                dscautocal.ad_gain = 0;
                dscautocal.ad_offset = 0;

                if( ( result = dscADCalVerify( dscb, &dscautocal ) ) != DE_NONE )
                {
                        dscGetLastError(&errorParams);
                        rt_fprintf( "dscADCalVerify error: %s %s\n", dscGetError-
String(errorParams.ErrCode), errorParams.errstring );
                        return 0;
                }

                rt_printf( "Configuration Mode: %d, Offset Error: %9.3f, Gain Error:
%9.3f\n", i, dscautocal.ad_offset, dscautocal.ad_gain );
```

```
                if ( fabs( dscautocal.ad_offset ) > 2.0f ||
                        fabs( dscautocal.ad_gain ) > 2.0f )
                        rt_printf( "Value for offset or gain exceeded specified tolerance\n" );
                else rt_printf( "Values for offset and gain met specified tolerance\n" );
        }

        dscFree();

        rt_printf( "Calibração dos conversores AD... COMPLETA\n\n" );
}

int S_Kill()
{
        dscFree();
        return 0;
}

int S_Core()
{
        int i;
        double torque;

        if( dscInit( DSC_VERSION ) != DE_NONE )
        {
                dscGetLastError(&errorParams);
                rt_printf( stderr, "dscInit error: %s %s\n", dscGetError-
String(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }

        if(dscInitBoard(DSC_DMM16AT, &dsccb, &dscb)!= DE_NONE)
        {
                dscGetLastError(&errorParams);
                rt_printf( stderr, "dscInitBoard error: %s %s\n", dscGetError-
String(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }

        if( ( result = dscADSetSettings( dscb, &dscadsettings ) ) != DE_NONE )
        {
                dscGetLastError(&errorParams);
                rt_printf( stderr, "dscADSetSettings error: %s %s\n", dscGetError-
String(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }
```

```
        if( ( result = dscADScan( dscb, &dscadscan, samples ) ) != DE_NONE )
                        {
                                dscGetLastError(&errorParams);
                                fprintf( stderr, "dscADScan error: %s %s\n", dscGetError-
String(errorParams.ErrCode), errorParams.errstring );
                                free( samples ); // remember to deallocate malloc() memory
                                return 0;
                        }

                        if (SENSORING) rt_printf( "\nActual voltages:" );

                        for( i = 0; i < (dscadscan.high_channel - dscadscan.low_channel)+
1; i++)
                        {
                                if( dscADCodeToVoltage(dscb, dscadsettings,
dscadscan.sample_values[i], &voltage) != DE_NONE)
                                {
                                        dscGetLastError(&errorParams);
                                        fprintf( stderr, "dscInit error: %s %s\n", dscGetError-
String(errorParams.ErrCode), errorParams.errstring );
                                        free(samples);
                                        return 0;
                                }

                                if (SENSORING) rt_printf(" %5.3lfV", voltage);
                                if (i==0)
                                {
                                        DEADZONE = .04;

                                        TORQUE[0] = TORQUE[1];
                                        TORQUE[1] = TORQUE[2];

                                        torque=(voltage-VOLTAGEREFERENCE);

                                        //DEADZONE
                                        torque = (voltage-VOLTAGEREFERENCE);
                                        if              (TORQUE[1]-torque<=-DEADZONE)
                                                TORQUE[2]=torque-DEADZONE;
                                        else if (TORQUE[1]-torque>=DEADZONE)
                                                TORQUE[2]=torque;
                                        else TORQUE[2]=TORQUE[1]+DEADZONE;

                                        if (SENSORING) rt_printf("\n Torque: %5.3lfNm
%5.3lfNm %5.3lfNm", TORQUE[0], TORQUE[1], TORQUE[2]);
```

```
                        }

                    }

        dscFree();

        return 0;
}
```

sensor.h:

```
#ifndef SENSOR_H_
#define SENSOR_H_

#include "dscud.h"
#include "global.h"
#include "include.h"

BYTE result; // returned error code
DSCB dscb;   // handle used to refer to the board
DSCCB dsccb; // structure containing board settings
DSCSAMPLE sample; // sample reading
DSCADSETTINGS dscadsettings; // structure containing A/D conversion settings
ERRPARAMS errorParams;       // structure for returning error code and error string
DFLOAT voltage;
DSCAUTOCAL dscautocal; // structure containing auto-calibration settings

DSCSAMPLE* samples;          // sample readings
DSCADSCAN dscadscan;         // structure containing A/D scan settings

int S_Init();
int S_Calibration();
int S_Kill();
int S_Core();

#endif /* SENSOR_H_ */
```

tasks.c:

```
#include "tasks.h"

void T_Init()
{
        //Declaração do ponteiro de funções
        pfunction[SENSOR]    = S_Core;
```

```
        pfunction[CONTROL]   = C_Core;
        pfunction[ACTUATOR]  = A_Core;
        pfunction[TEST]      = T_Test;

        //Habilita/Desabilita Task
        ENABLETASK[SENSOR]   = TRUE;
        ENABLETASK[CONTROL]  = TRUE;
        ENABLETASK[ACTUATOR] = TRUE;
        ENABLETASK[TEST]     = FALSE;

        //Período das Tasks [ms]
        PERIOD[SENSOR]   = 2;
        PERIOD[CONTROL]  = 2;
        PERIOD[ACTUATOR] = 2;
        PERIOD[TEST]     = 0;

        //Prioridade das Tasks [0-100]
        PRIORITY[SENSOR]   = 50;
        PRIORITY[CONTROL]  = 50;
        PRIORITY[ACTUATOR] = 50;
        PRIORITY[TEST]     = 0;

}

void T_Kill()
{
        A_Kill();
        C_Kill();
        S_Kill();
}

void T_Xenomai()
{
        //      Avoids memory swapping for this program
        mlockall(MCL_CURRENT|MCL_FUTURE);
        //      Perform auto-init of rt_print buffers if the task doesn't do so
        rt_print_auto_init(1);
        rt_timer_set_mode(1000000); //1ms -not working for periodic mode
}

void T_Create()
{
        rt_task_create(&TaskSensor,   "Sensor  ", 0, PRIORITY[SENSOR], 0);
        rt_task_create(&TaskControl,  "Controle", 0, PRIORITY[CONTROL], 0);
        rt_task_create(&TaskActuator, "Atuador ", 0, PRIORITY[ACTUATOR], 0);
```

```
        rt_task_create(&TaskTest,     "Teste   ", 0, PRIORITY[TEST], 0);
}

void T_Start()
{
        if (ENABLETASK[SENSOR]) {
                S_Init();
                rt_task_start(&TaskSensor, &T_Monitor, SENSOR);
        }
        if (ENABLETASK[CONTROL]) {
                C_Init();
                rt_task_start(&TaskControl, &T_Monitor, CONTROL);
        }
        if (ENABLETASK[ACTUATOR]) {
                A_Init();
                rt_task_start(&TaskActuator, &T_Monitor, ACTUATOR);
        }
        if (ENABLETASK[TEST])
                rt_task_start(&TaskTest, &T_Monitor, TEST);
}

void T_CleanUp (void)
{
        rt_task_delete(&TaskControl);
        rt_task_delete(&TaskSensor);
        rt_task_delete(&TaskActuator);
        rt_task_delete(&TaskTest);
}

void T_PrintTaskName()
{
        RT_TASK *curtask;
        RT_TASK_INFO curtaskinfo;

        curtask=rt_task_self();

        rt_task_inquire(curtask,&curtaskinfo);
        rt_printf("%s ",curtaskinfo.name);
}

void T_Monitor(int functionID)
{
        //RTIME é um unsigned long long
        RTIME start, end, duration, idle, period;
```

```
	//Indicador de overuns
	unsigned long overs;

	//Valores precisam ser em Ns!
	period = PERIOD[functionID]*NS2MS;

	rt_task_set_periodic(NULL, TM_NOW, period);
	end = rt_timer_read();
	rt_task_wait_period(&overs);

	while (1) {
		if (MONITORING) {
			start = rt_timer_read();
			idle = start-end;
			T_PrintTaskName();
			rt_printf(":\t%ld.%06ld ms\t%6lld.%03lld us\t\"%d\"\n",
					(long) duration / NS2MS,
					(long) duration % NS2MS,
					(signed long long)(duration+idle-period)/ NS2US,
//conversion to us
					(duration+idle-period) % NS2US,
					overs);
		}
		pfunction[functionID]();
		if (MONITORING) {
			end = rt_timer_read();
			duration = end-start;
		}
		rt_task_wait_period(&overs);
		if(overs>0) rt_printf("!\n");
	}
	return;
}

int T_Test()
{
	//Inserir funções teste...
	return 0;
}
```

tasks.h:

```
#ifndef TASKS_H_
#define TASKS_H_
```

```
#include "global.h"
#include "include.h"

#include "control.h"
#include "sensor.h"
#include "actuator.h"

//Declaração das Tasks
RT_TASK TaskSensor, TaskControl, TaskActuator, TaskTest;

//Período das Tasks [ms]
int PERIOD[NUMBEROFTASKS];
int PRIORITY[NUMBEROFTASKS];
int ENABLETASK[NUMBEROFTASKS];

//Array de Funções
int (*pfunction[NUMBEROFTASKS])();

void T_Init();
void T_Kill();
void T_Xenomai();
void T_Create();
void T_Start();
void T_Monitor(int);
void T_CleanUp ();
void T_PrintTaskName();
int T_Test();

#endif /* TASKS_H_ */
```

SCRIPTS DO MATLAB

Main.m:

```
clearall;

step = 0.002;
torque = 0.2;

k = 1/10000000;
% b = 1e-3;
% m = 0.005;
```

```
Mp = 0.001;
Ts = .004;
beta = atan(-pi/(log(Mp)));
ba = (cos(beta))^2*k*Ts/2;
m = (cos(beta))^2*k*Ts^2/16;

zeta = cos(beta)
wn = 4/(zeta*Ts)

Simulink;
Simulink_discreto;
calculado;

% malha = textread('malha2.txt','%f','delimiter',' ');
%
% figure;
% time_malha(:,1)=0:step:step*(length(malha)-1);
% createfigure(time_malha,malha);
```

Simulink.m:

```
% torque = 0.2;

hold on;
Gc = tf(1, [m ba k]);
sim modelo;
createfigure(time,position);
% plot(time,position);
% hold on;
% plot(time,force,'r');
```

Simulink_discreto.m:

```
% step=0.002;
sim discreto;
plot(time,position);
% hold on;
% plot(time,position_ref,'y');
% hold on;
% plot(time,force,'r');
```

Calculado.m:

```
timec = 0:step:30;
l = length(timec);
```

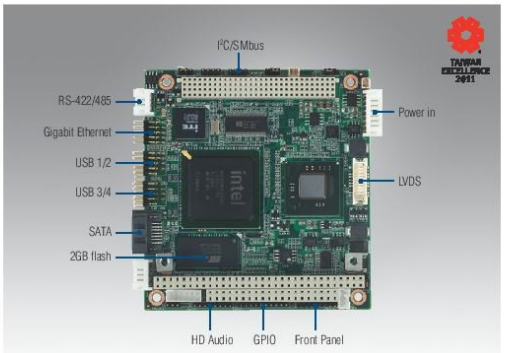
```
force=zeros(l,1);
force(:)=torque;
timec=timec';
Gd = c2d(Gc,step,'zoh')
y = zeros (l,1);
r = zeros (l,1);
a=0;b=0;c=0;d=0;e=0;
a = Gd.num{1,1}(2);
b = Gd.num{1,1}(3);
c = Gd.den{1,1}(1);
d = Gd.den{1,1}(2);
e = Gd.den{1,1}(3);
display([a b c d e]);
for i=3:l
    y(i) = a*force(i-1) + b*force(i-2) -d*y(i-1) -e*y(i-2);
    y(i) = y(i)/c;
%     display([a*force(i-1) b*force(i-2) y(i) -d*y(i-1) -e*y(i-
2)]);
%     pause;
end

plot(timec,y);
% hold on;
% plot(timec,force,'r');
```

ANEXO A – ESPECIFICAÇÕES TÉCNICAS – PC/104 -

PCM-3362

Intel® Atom™ N450 PC/104-Plus SBC, VGA, LVDS, Ethernet, USB, COM, SATA, Onboard Flash



VxWorks® Windows Embedded QNX SUSIAccess CE FCC

Features

- Intel® Atom™ N450 1.66 GHz Processor and DDR2 667 MHz SDRAM up to 2 GB
- Supports extended temperature -40 ~ 85° C
- Standard 96 x 90 mm dimensions and PC/104-Plus expansion connector
- Onboard 2 GB flash (4 GB optional)
- Supports SUSIAccess and Embedded Software APIs

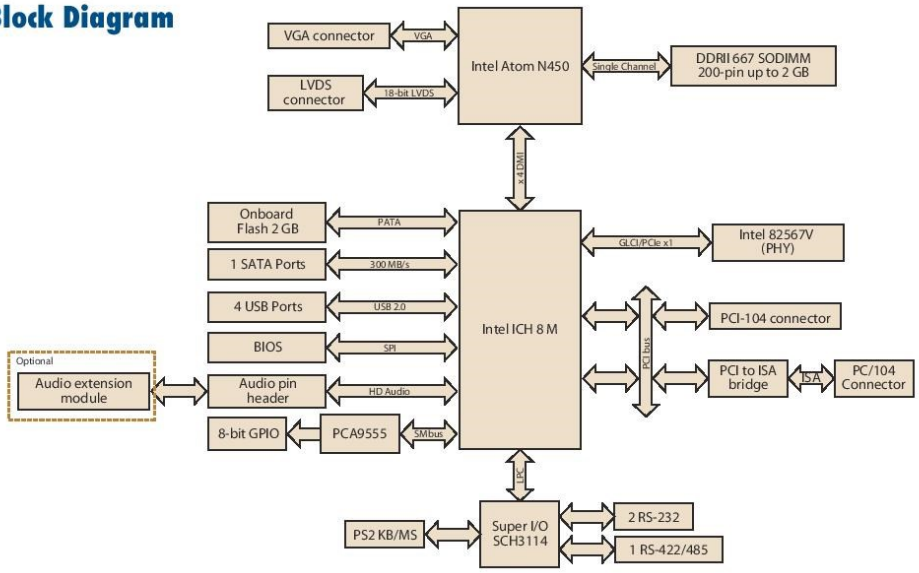
Software APIs: Watchdog, I²C, Brightness, H/W Monitor, GPIO, Backlight On/Off

Utilities: Monitoring, Embedded Security ID

Specifications

Processor System	CPU	Intel Atom N450 1.66 GHz
	Frequency	1.66 GHz
	L2 Cache	512 KB
	System Chipset	Intel Atom N450 + ICH8M
	BIOS	AMI 16 Mbit
Memory	Technology	DDR2 667 MHz
	Max. Capacity	2 GB
	Socket	1 x 200-pin SO-DIMM
Display	Chipset	Intel Atom N450 1.66 GHz
	VRAM	Shared system memory up to 224 MB
	Graphics Engine	Intel Gen 3.5 graphic core, DX9 compliant, MPEG2 Hardware Acceleration
	LVDS	Single channel 18-bit LVDS up to WXGA 1366 x 768
	VGA	Supports up to SXGA 1400 x 1060 at 60 Hz
	Dual Display	VGA+LVDS
Ethernet	Speed	10/100/1000 Mbps
	Controller	ICH8M + Intel 82567V (PHY), supports Wake-on-LAN
	Connector	Pin Header
Watchdog Timer		Output System Reset Programmable counter from 1 ~ 255 minutes/ seconds
Storage	SATA	1 SATAII, up to 3.0 GB/s (300 MB/s)
	Onboard Flash	2 GB (Up to 4 GB)
Internal I/O	USB	4 x USB 2.0
	Serial	2 RS-232 from COM1/2, 1 RS-422/485 from COM3 (ESD protection for RS-232: Air gap ±15 kV, Contact ±8 kV)
	Keyboard/Mouse	1
	GPIO	8-bit general purpose input/output
	I²C	1
	Audio	Intel High Definition audio interface
Expansion	PC/104-Plus Slot	1
Power	Power Type	AT/ATX
	Power Supply Voltage	5 V ± 5% only to boot up (12 V is optional for LCD inverter and add-on card)
	Power Consumption (Typical)	2 A @ +5 V, 5 mA @ +12 V (10.06 Watt)
	Power Consumption (Max, test in HCT)	2.37 A @ +5 V, 7 mA @ +12 V (11.93 Watt)
	Battery	Lithium 3 V / 210 mAH
	Power Management	ACPI/ APM 1.2
Environment	Operational	0 ~ 60° C (32 ~ 140° F) (Operational humidity: 40° C @ 85% RH non-condensing)
	Non-Operational	-40° C ~ 85° C and 60° C @ 95% RH non-condensing
Physical Characteristics	Dimensions (L x W)	96 x 90 mm (3.8" x 3.5")
	Weight	0.664 kg (1.46 lb) (with heat-sink)
	Height	Top Side: 14.4 mm, 19.4 mm (Z & Z2); Bottom Side: 10.6 mm

Block Diagram



Ordering Information

Part No.	CPU	Memory	On board Flash	VGA	LVDS	Gigabit Ethernet	USB 2.0	RS-232	RS-422/485	Expansion	Thermal Solution	Operating Temp.
PCM-3362N-S6A1E	Atom N450	SODIMM	2 GB	Yes	18-bit	1	4	2	1	PC/104+	Passive	0 ~ 60° C
PCM-3362N-S6F4A1E	Atom N450	SODIMM	4 GB	Yes	18-bit	1	4	2	1	PC/104+	Passive	0 ~ 60° C
PCM-3362Z-1GS6A1E	Atom N450	1 GB bundle	2 GB	Yes	18-bit	1	4	2	1	PC/104+	Passive	-20 ~ 80° C
PCM-3362Z2-1GS6A1E	Atom N450	1 GB bundle	2 GB	Yes	18-bit	1	4	2	1	PC/104+	Passive	-40 ~ 85° C

Note: Wide temperature version is bundled with extended temperature grade memory module
Note: Passive = fanless; Active = with fan

Packing List

Part No.	Description	Quantity
	PCM-3362 SBC	
	Startup Manual	
	Utility CD	
1700000898	VGA cable D-SUB 15P(F)/12P-1.25 mm15 cm	1
1700003491	AT power cable 1 x 8P-2.0/B4P-5.08 x 2 15 cm	1
1700060202	Cable 6P-6P-6P PS/2 KB & Mouse 20 cm	1
1703040157	RS-422/485 W/D-SUB COM 4P 15 cm	1
1703060053	PS2 cable 6P (MINI-DIN)-6P (Wafer 2.0 mm) 6 cm	1
1700002332	ATX power cable 20P-13P/8P/3P/3P 13 cm	1
1703100260	USB cable 2 ports 2.0 mm pitch w/ bracket 26 cm	1
1700071000	SATA data cable 7p 100 cm	1
1703150102	SATA power cable B4P-5.08/SATA 15P 10 cm	1
1701200220	RS-232 x 2 ports 2.0 mm 22 cm	1
1700017863	LAN cable RJ-45/2 x 5P-2.0 15 cm	1
9660104000	PC/104 screw and copper post package	1
1960045487T001	Heatsink for PCM-3362 (79.66 x 77.97 x 12.22 mm)	1
1960046618T001	Heatsink for PCM-3362Z&Z2 series only (79.66 x 77.97 x 17.22 mm)	1

Optional Accessories

Part No.	Description
1960047106T001	Heat spreader (79.66 x 77.98 x 10.32 mm) of PCM-3362
1653130421	PCI-104 connector 120-pin (Long pin)
165313222B	PC/104 connector 64-pin (Long pin)
165312022B	PC/104 connector 40-pin (Long pin)
PCA-AUDIO-HDA1E	Audio extension module with bracket
1700018427	Audio cable connecting PCM-3362 and PCA-AUDIO-HDA1E

Embedded OS/API

Embedded OS/API	Part No.	Description
WinCE	2070009692	CE 6.0 Pro PCM-3362 V1.3 ENG
	2070011081	WinCE 7.0 Pro PCM-3362 V1.0 ENG
Win XPE	2070009030	XPE WES2009 Luna Pier V4.0 ENG
	2070009031	XPE WES2009 Luna Pier V4.0 MUI24
QNX		6.5
Linux		Ubuntu 10.04
VxWorks		6.8
Software API	205E362000	SUSI 3.0 SW API for PCM-3362 B:20091015 XP

ANEXO B – ESPECIFICAÇÕES TÉCNICAS - PLACA DIAMOND-MM-16-AT

DIAMOND SYSTEMS CORPORATION

DIAMOND-MM-16-AT

16-channel, 16-bit Analog I/O with Autocalibration

- 16 16-bit A/D with 100KHz sample rate, programmable input ranges and a 512 sample FIFO
- Autocalibration of A/D and D/A for high accuracy
- 4 12-bit D/A
- 8 digital inputs and 8 digital outputs
- Counter / timers for A/D control and general use

DESCRIPTION

The Diamond-MM-16-AT features top performance and flexibility for a mid-range price. It has 16 single-ended / 8 differential analog inputs with both unipolar and bipolar input ranges and programmable gain. It has a maximum sampling rate of 100KHz, supported by a 512-sample FIFO with a 256-sample threshold for gap-free A/D sampling. Both single-channel and multi-channel scan sampling modes are supported. The A/D can be triggered with a software command, the on-board programmable timer, or an external signal. These feature give you maximum flexibility to configure the board to your application.

ANALOG INPUTS

The 16 16-bit analog input channels on Diamond-MM-16-AT feature programmable gains of 1, 2, 4, and 8, as well as programmable unipolar/bipolar range, for a total of 7 different input ranges. Maximum sampling rate is 100KHz (total for all channels), and a new 512-sample FIFO enables the board to operate at full speed in Windows operating systems using interrupts. DMA is no longer required to attain full speed.

SPECIFICATIONS

Analog Inputs	
Number of inputs	16 16-bit
Input Modes	Single-ended, Differential
Input Ranges	±10V, ±5V, ±2.5V, ±1.25V, ±0-.625V, 0-10V, 0-5V, 1.25V, 0-.625V
Max Sample Rate	100KHz
Nonlinearity	±3LSB, no missing codes
On-board FIFO	512, prog. threshold
Calibration	Software initiated autocalibration
Analog Outputs	4, 12-bit resolution
Output Ranges	±5V, 0-5V
Output Current	±5mA max per channel
Settling Time	6µS max to 0.01%
Analog Outputs	
Relative Accuracy	±1 LSB
Digital I/O Lines	8 In, 8 Out
DIO Input Voltage	Logic 0: 0.0V min, 0.8V max Logic 1: 2.0V min, 5.0V max
DIO Output Voltage	Logic 0: 0.0V min, 0.33V max Logic 1: 3.8V min, 5.0V max
Counter / Timers	1 - 32-bit; 1 - 16-bit
Clock Source	10MHz clock or external signal
Power Supply	+5VDC±10%@350mA
Operating Temp	-40°C to +85°C
Weight	3.3oz / 93g

ANALOG OUTPUTS

The board also has 4 12-bit D/A channels with multiple unipolar and bipolar output ranges. The DACs feature simultaneous update capability. A new programmable output range feature lets you set the output range via software anywhere between 0V and 10V with 1mV precision in both unipolar and bipolar modes.

COUNTERS AND DIGITAL I/O

Diamond-MM-16-AT has an on-board counter/timer to control A/D sampling or rate generator functions, 8 digital inputs, and 8 digital outputs. New features enable you to generate hardware interrupts from the counter/timer as well as an external digital signal. And in keeping with our real-world-friendly design, Diamond-MM-16-AT requires only +5V power supply and operates over the full industrial temperature range of -40 to +85oC.

ORDERING INFORMATION

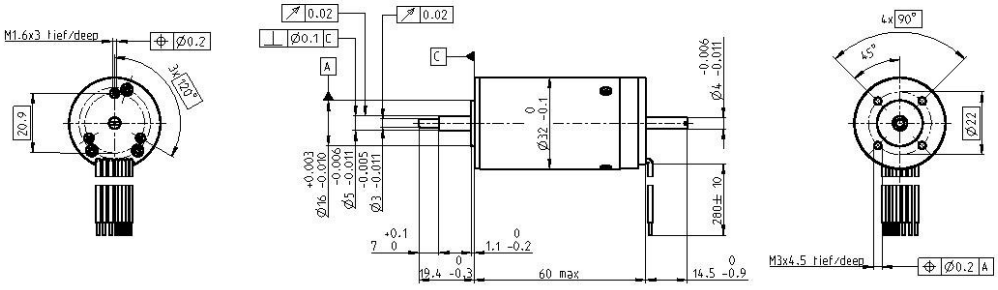
Part No.	Description
DMM-16-AT	Diamond-MM Autocalibrating 16-ch 16-bit A/D + 4-ch 12-bit D/A Extended Temperature
DMM-16-NA-AT	Diamond-MM Autocalibrating 16-ch 16-bit A/D only Extended Temperature

FOR MORE INFORMATION

Diamond Systems Corporation
1255 Terra Bella Avenue
Mountain View, CA 94043
Tel: 650-810-2500
Fax: 650-810-2525
techinfo@diamondsystems.com

ANEXO C – ESPECIFICAÇÕES TÉCNICAS - MOTOR MAXON - MODELO 118889

EC 32 ∅32 mm, bürstenlos, 80 Watt



M 1:2

Lagerprogramm
Standardprogramm
Sonderprogramm (auf Anfrage)

Motordaten		Artikelnummern							
		118891	118892	118888	118889	118893	118890		
Werte bei Nennspannung									
1 Nennspannung	V	12	18	18	24	36	48		
2 Leerlaufdrehzahl	min⁻¹	15100	14300	13100	11000	14700	11300		
3 Leerlaufstrom	mA	662	404	349	199	211	104		
4 Nenndrehzahl	min⁻¹	13400	12700	11500	9450	13200	9740		
5 Nennmoment (max. Dauerdrehmoment)	mNm	44.6	45.2	45.9	47.2	43.8	45.9		
6 Nennstrom (max. Dauerbelastungsstrom)	A	6.51	4.15	3.82	2.46	2.07	1.23		
7 Anhaltemoment	mNm	428	443	407	355	454	353		
8 Anlaufstrom	A	57.2	37.4	31.4	17.3	19.7	8.84		
9 Max. Wirkungsgrad	%	80	81	81	80	81	80		
Kenndaten									
10 Anschlusswiderstand Phase-Phase	Ω	0.21	0.481	0.573	1.39	1.83	5.43		
11 Anschlussinduktivität Phase-Phase	mH	0.03	0.0752	0.09	0.226	0.285	0.856		
12 Drehmomentkonstante	mNm A⁻¹	7.48	11.8	13	20.5	23.1	40		
13 Drehzahlkonstante	min⁻¹ V⁻¹	1280	806	737	465	414	239		
14 Kennliniensteigung	min⁻¹ mNm⁻¹	35.8	32.7	32.6	31.5	32.8	32.5		
15 Mechanische Anlaufzeitkonstante	ms	7.49	6.86	6.82	6.59	6.87	6.8		
16 Rotorträgheitsmoment	gcm²	20	20	20	20	20	20		

Spezifikationen

Thermische Daten

17	Therm. Widerstand Gehäuse-Luft	5.4 KW⁻¹
18	Therm. Widerstand Wicklung-Gehäuse	2.5 KW⁻¹
19	Therm. Zeitkonstante der Wicklung	14.8 s
20	Therm. Zeitkonstante des Motors	1180 s
21	Umgebungstemperatur	-20…+100°C
22	Max. Wicklungstemperatur	+125°C

Mechanische Daten (vorgespannte Kugellager)

23	Grenzdrehzahl[1]		25000 min⁻¹
24	Axialspiel bei Axiallast	< 8 N	0 mm
		> 8 N	max. 0.14 mm
25	Radialspiel		vorgespannt
26	Max. axiale Belastung (dynamisch)		5.6 N
27	Max. axiale Aufpresskraft (statisch)		98 N
	(statisch, Welle abgestützt)		1200 N
28	Max. radiale Belastung, 5 mm ab Flansch		28 N

Weitere Spezifikationen

29	Polpaarzahl	1
30	Anzahl Phasen	3
31	Motorgewicht	270 g

Motordaten gemäss Tabelle sind Nenndaten.

Anschlüsse Motor (Kabel AWG 22)

rot	Motorwicklung 1
schwarz	Motorwicklung 2
weiss	Motorwicklung 3

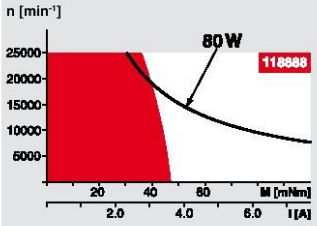
Anschlüsse Sensoren (Kabel AWG 26)[1]

grün	V_{Hall} 4.5…24 VDC
blau	GND
rot/grau	Hall-Sensor 1
schwarz/grau	Hall-Sensor 2
weiss/grau	Hall-Sensor 3

Schaltbild für Hall-Sensoren siehe S. 35

[1] In Kombination mit Resolver nicht herausgeführt.

Betriebsbereiche



Legende

Dauerbetriebsbereich
Unter Berücksichtigung der angegebenen thermischen Widerstände (Ziffer 17 und 18) und einer Umgebungstemperatur von 25°C wird bei dauernder Belastung die maximal zulässige Rotortemperatur erreicht = thermische Grenze.

Kurzzeitbetrieb
Der Motor darf kurzzeitig und wiederkehrend überlastet werden.

Typenleistung

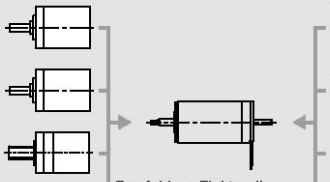
maxon-Baukastensystem

Übersicht Seite 20 - 25

Planetengetriebe
∅32 mm
0.75 - 4.5 Nm
Seite 259

Planetengetriebe
∅32 mm
0.75 - 6.0 Nm
Seite 261–265

Spindelgetriebe
∅32 mm
Seite 286–288



Encoder HED_5540
500 Imp.,
3 Kanal
Seite 306/308

Resolver Res 26
∅26 mm
10 V
Seite 316

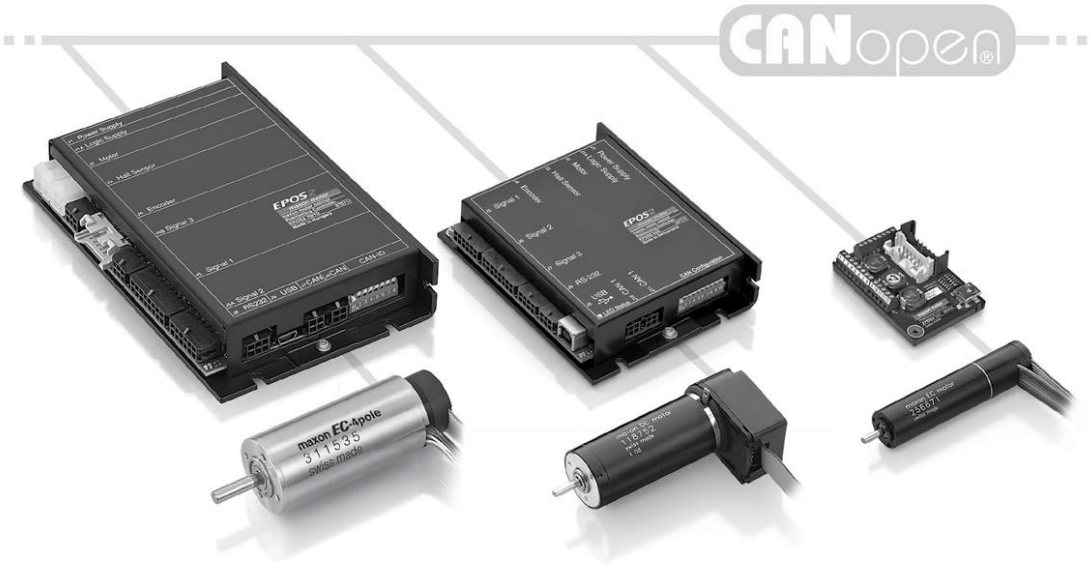
Empfohlene Elektronik:

ESCON 50/5	Seite 321
ESCON Module 50/5	321
ESCON 70/10	321
DECS 50/5	324
DEC Module 50/5	325
EPOS2 24/5, 50/5, 70/10	331
EPOS2 P 24/5	334
EPOS3 70/10 EtherCAT	337
Hinweise	**24**

Ausgabe März 2013 / Änderungen vorbehalten

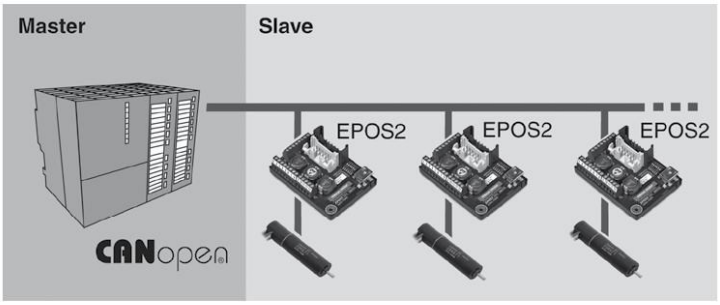
ANEXO D – ESPECIFICAÇÕES TÉCNICAS - DRIVER MAXON EPOS2 24/5

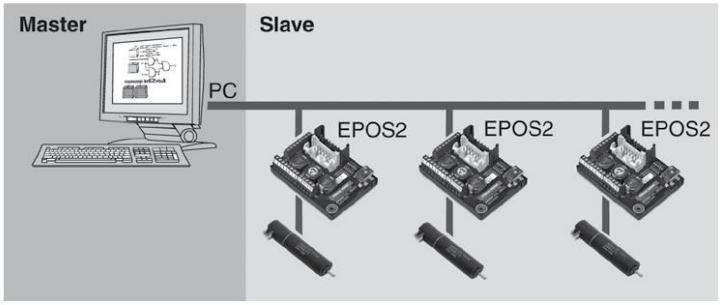
EPOS2 Positioning Control Units



Slave version (online commanded)

Single motion and I/O commands from the process control are transmitted to the positioning control unit by a superior system (Master). For that purpose product specific commands are available.





EPOS2 is a modular constructed digital positioning controller. It is suitable for DC and EC motors with incremental encoder with a power range from 1 - 700 watts.
A number of operating modes provides flexible application in a wide range of drive systems in automation technology and mechatronics.

Point to point
The "CANopen Profile Position Mode" move the position of the motor axis from point A to point B. Positioning is in relation to the axis Home position (absolute) or the actual axis position (relative).

Interpolated Position Mode (PVT)
Thanks to Interpolated Position Mode, the EPOS2 is able to synchronously run a path specified by interpolating points. With a suitable master, coordinated multi-axis movements as well as any profile in a 1-axis system can be carried out. (PVT = Position and Velocity versus Time)

Position and Speed control with Feed Forward
The combination of feedback and feed forward control provides ideal motion behaviour. Feed forward control reduces control error. EPOS2 supports feed forward acceleration and speed control.

Speed control
In "CANopen Profile Velocity Mode", the motor axis is moved with a set speed. The motor axis retains speed until a new speed is set.

Torque control
In "Current Mode", a controlled torque can be produced on the motor shaft. The sinusoidal commutation used produces minimum torque ripple.

Homing
The "CANopen Homing Mode" is for referencing to a special mechanical position. There are more than 30 methods available for finding the reference position.

Electronic gearhead
In "Master Encoder Mode", the motor follows a reference input produced by an external encoder. A gearing factor can also be defined using software parameters. Two motors can be very easily synchronised using this method.

Step/Direction
In "Step/Direction Mode" the motor axis follows a digital signal step-by-step. This mode can replace stepper motors. It can also be used to control the EPOS2 by a PLC without CAN interface.

Analog Commands
In the position, speed and current mode it is possible to give commands via an external analog set value. This function offers further possibilities to operate the EPOS2 without serial on-line commanding.

Capture inputs (Position Marker)
Digital inputs can be configured so that the actual position value is saved when a positive and/or negative edge of an input appears.

Trigger output (Position Compare)
Digital outputs can be configured so that a digital signal is emitted at a set position value.

Dual Loop Position and Speed Control
With an additional sensor the load can be controlled directly and with high precision; the motor control is subordinated. The mechanical play and the elasticity can be compensated.
Wide range of sensors can be handled: digital incremental encoder, SSI absolute encoder, analog incremental encoder (sin/cos). (Only in use with EPOS2 50/5 and EPOS2 70/10.)

Control of Holding Brakes
The control of the holding brake can be implemented in the device state management. There the delay times can be individually configured for switching on and off.

Additional information for technical data of page 330/331

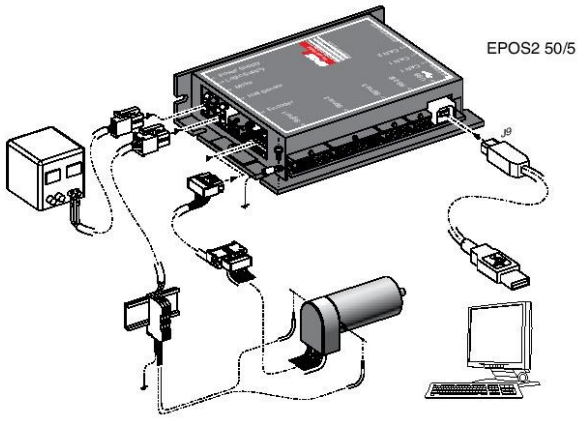
Standardised, extendable
CANopen standard CiA DS-301, DSP-402 and DSP-305. Can easily be integrated into existing CANopen systems. Networks with other CANopen modules. Alternatively controllable by serial interface (USB and RS232).

Flexible, modular
The same technology for DC and EC motors. Configurable inputs and outputs for limit switches, reference switches, brakes and for other sensors and indicators near the drive.

Easy start-up procedure
Graphic user interface with many functions and wizards for start-up procedure, automatic control settings, I/O configuration, tests.

Easy programming
Numerous IEC 61131-3 libraries free available for CAN-Master units of several PLC manufacturers providers (Beckhoff, Siemens/Helmholz, VIPA) and 32-/64-bit Windows-DLLs for PC Master (IXXAT, Vector and National Instruments). Various programming examples free available for MS Visual C#, MS Visual C++, MS Visual Basic, Borland C++, Borland Delphi, National Instruments LabVIEW and National Instruments LabWindows/CVI.
Also available is the 32-bit Linux Shared Object Library with the programming example for Eclipse C++/QT. In addition, the integration of the EPOS2 into the National Instruments Compact Rio System is easy to handle thanks to the available maxon library for NI SoftMotion.

State-of-the-art
Digital position, speed and current/torque control. Sinusoidal commutation for smooth operation of EC motors.



EPOS2 50/5

Operating modes
CANopen Profile Position-, Profile Velocity- and Homing Mode
Position, Velocity and Current Mode
Alternative set value setting via Step/Direction, Master Encoder or external analog commanding
Path generating with trapezoidal or sinusoidal profiles
Feed forward for velocity and acceleration
Interpolated Position Mode (PVT)
Sinusoidal or block commutation for EC motors
Dual loop position and speed controller
Communication
Communication via CANopen and/or USB 2.0 and/or RS232
Gateway function USB-to-CAN and RS232-to-CAN

Inputs/Outputs
Free configurable digital inputs e.g. for limit switches and reference switches
Free configurable digital outputs e.g. for holding brakes
Free analog inputs
Available software
EPOS Studio
Windows DLL
IEC 61131-3 Libraries
Firmware
Available documentation
Getting Started
Cable Starting Set
Hardware Reference
Firmware Specification
Communication Guide
Application Notes
Cable
A comprehensive range of cables is available as an option. Details can be found on page 339.

EPOS2 24/5
Matched with DC brush motors with encoder or brushless EC motors with Hall sensors and encoder, from 5 to 120 watts.

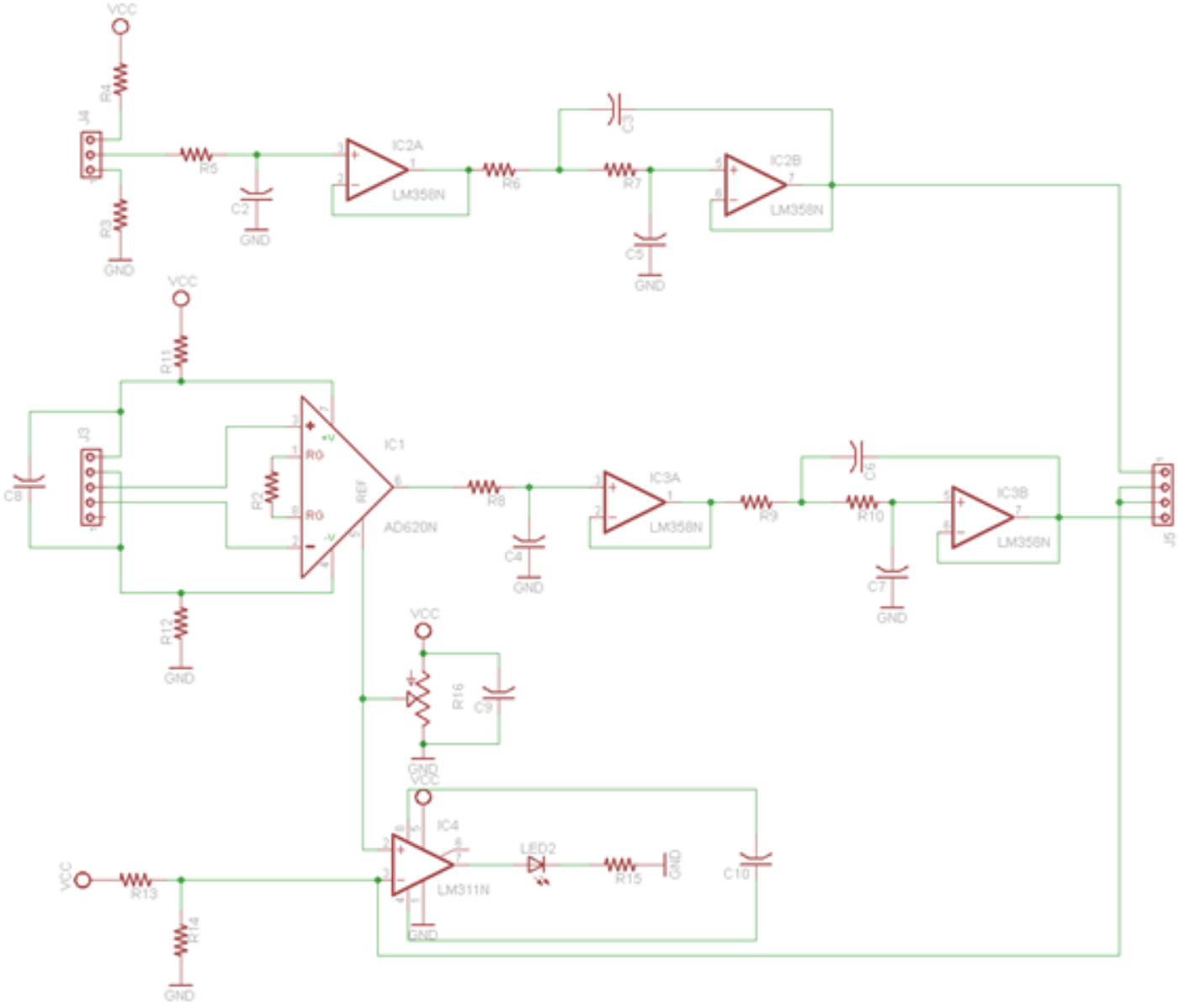
EPOS2 50/5
Matched with DC brush motors with encoder or brushless EC motors with Hall sensors and encoder, from 5 to 250 watts.

EPOS2 70/10
Matched with DC brush motors with encoder or brushless EC motors with Hall sensors or encoder, from 80 to 700 watts.

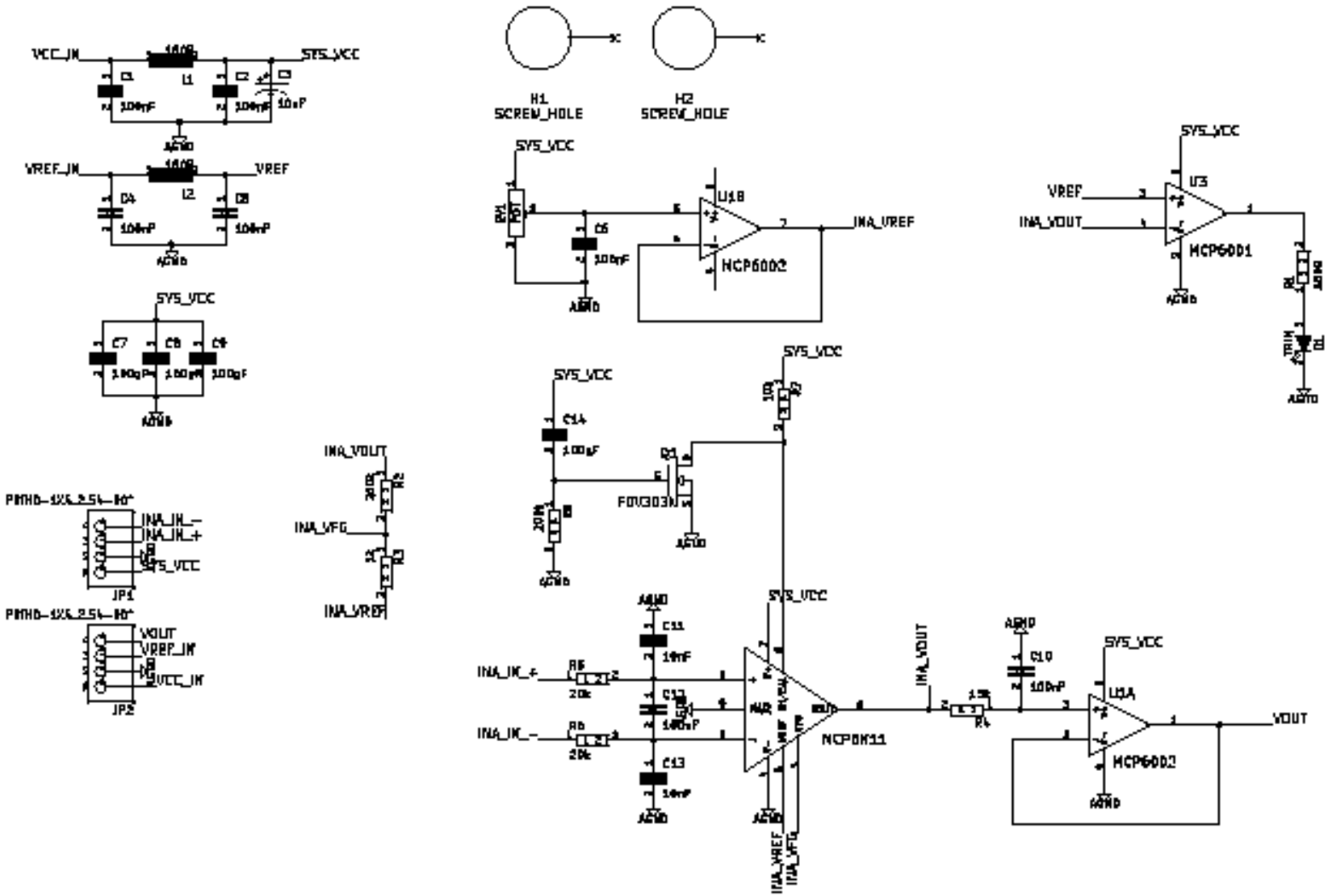
Controller versions		
Slave version	**Slave version**	**Slave version**
Electrical Data		
11 - 24 VDC	11 - 50 VDC	11 - 70 VDC
11 - 24 VDC	11 - 50 VDC	11 - 70 VDC
0.9 x V_{CC}	0.9 x V_{CC}	0.9 x V_{CC}
10 A	10 A	25 A
5 A	5 A	10 A
50 kHz	50 kHz	50 kHz
10 kHz	10 kHz	10 kHz
1 kHz	1 kHz	1 kHz
1 kHz	1 kHz	1 kHz
25 000 rpm (sinusoidal); 100 000 rpm (block)	25 000 rpm (sinusoidal); 100 000 rpm (block)	25 000 rpm (sinusoidal); 100 000 rpm (block)
15 µH / 5 A	22 µH / 5 A	25 µH / 10 A
Input		
H1, H2, H3	H1, H2, H3	H1, H2, H3
A, A\, B, B\, I, I\ (max. 5 MHz)	A, A\, B, B\, I, I\ (max. 5 MHz)	A, A\, B, B\, I, I\ (max. 5 MHz)
6 (TTL and PLC level)	11 (7 optically isolated, 4 differential)	10 (7 optically isolated, 3 differential)
2 12-bit resolution, 0…+5 V	2 (differential) 12-bit resolution, ±10 V	2 (differential) 12-bit resolution, 0…+5 V
configurable with DIP switch 1…7	configurable with DIP switch 1…7	configurable with DIP switch 1…7
Output		
4	5 (4 optically isolated, 1 differential)	5 (4 optically isolated, 1 differential)
	1 (12-bit, 0…10 V)	
+5 VDC, max 100 mA	+5 VDC, max. 100 mA	+5 VDC, max. 100 mA
+5 VDC, max. 30 mA	+5 VDC, max. 30 mA	+5 VDC, max. 30 mA
V_{CC}, max. 1300 mA	+5 VDC, max. 150 mA	+5 VDC, max. 150 mA; +5 VDC (R_i = 1 kΩ)
Interface		
RxD; TxD (max. 115 200 bit/s)	RxD; TxD (max. 115 200 bit/s)	RxD; TxD (max. 115 200 bit/s)
high; low (max. 1 Mbit/s)	high; low (max. 1 Mbit/s)	high; low (max. 1 Mbit/s)
Data+; Data- (max. 12 Mbit/s)	Data+; Data- (max. 12 Mbit/s)	Data+; Data- (max.12 Mbit/s)
Indicator		
green LED, red LED	green LED, red LED	green LED, red LED
Ambient temperature and humidity range		
-10…+45°C	-10…+45°C	-10…+45°C
-40…+85°C	-40…+85°C	-40…+85°C
20…80%	20…80%	20…80%
Mechanical data		
Approx. 170 g	Approx. 240 g	Approx. 330 g
105 x 83 x 24 mm	120 x 93.5 x 27 mm	150 x 93 x 27 mm
Flange for M3-screws	Flange for M3-screws	Flange for M3-screws
Part Numbers		
367676 EPOS2 24/5	**347717** EPOS2 50/5	**375711** EPOS2 70/10

Accessories		
309687 DSR 50/5 Shunt regulator	**309687** DSR 50/5 Shunt regulator	**235811** DSR 70/30 Shunt regulator
Order accessories separately, see page 339	Order accessories separately, see page 339	Order accessories separately, see page 339

ANEXO E – CIRCUITO DE CONDICIONAMENTO DE SINAL DA CÉLULA DE CARGA (PRIMEIRA PLACA)


VCC
R4
J4
R5
C2
GND
R3
GND
IC2A
LM358N
R6
C3
R7
C5
GND
IC2B
LM358N
VCC
R11
J3
C8
+V
R0
R2
R0
-V
REF
IC1
AD620N
R12
GND
R8
C4
GND
IC3A
LM358N
R9
C6
R10
C7
GND
IC3B
LM358N
J5
VCC
R16
C9
GND
VCC
IC4
LM311N
LED2
R15
GND
C10
VCC
R13
R14
GND

ANEXO F – CIRCUITO DE CONDICIONAMENTO DE SINAL DA CÉLULA DE CARGA (SEGUNDA PLACA)



File:		
Sheet: /		
Title:		
Size: A4	Date: 20 aug 2012	Rev:
KiCad E.D.A. eeschema (2013-02-27 BZR 3978)-stable		Id: 1/1