Carolina Pachler
Diego Ken Yabuki

IMPLEMENTAÇÃO DE UM CONTROLE DE IMPEDÂNCIAS MODULAR PARA UM EXOESQUELETO ROBÓTICO DE MEMBRO SUPERIOR

São Paulo
2014

Carolina Pachler
Diego Ken Yabuki

IMPLEMENTAÇÃO DE UM CONTROLE DE IMPEDÂNCIAS MODULAR PARA UM EXOESQUELETO ROBÓTICO DE MEMBRO SUPERIOR

Área de Concentração:

Engenharia Mecatrônica

Orientador:

Prof. Dr. Arturo Forner-Cordero

São Paulo
2014

Catalogação-na-publicação

Pachler, Carolina
Implementação de um controle de impedâncias modular para um exoesqueleto robótico de membro superior / C. Pachler; Yabuki, Diego Ken. – São Paulo, 2014.
204 p.

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

1.Controle de impedâncias 2.Controle motor 3.Exoesqueleto 4.Robótica 5.Tempo-real (Controle)I. Yabuki, Diego Ken II.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos III.t.

AGRADECIMENTOS

Agradecemos ao Prof. Dr. Arturo Forner Cordero, pelo apoio, atenção dedicada e orientação do projeto.

Agradecemos aos membros do Laboratório de Biomecatrônica da Escola Politécnica da USP, pela disposição em prestar auxílio quando necessário.

Agradecemos às nossas famílias, pela compreensão e pelo suporte oferecido em todos os momentos até o presente.

RESUMO

Este trabalho consiste no projeto e implementação de um controle de impedâncias modular para um exoesqueleto robótico de membro superior de um grau de liberdade. O algoritmo foi estruturado de forma a permitir a inclusão de outros graus de liberdade, que podem ser integrados por meio de um controle hierárquico. A implementação foi feita em linguagem C/C++, em conjunto com um *kernel* de Linux com uma extensão que permite executar tarefas em tempo real (*Xenomai*). A estratégia de controle utilizada permite a obtenção de diferentes modos de interação entre exoesqueleto e usuário através da variação dos parâmetros de impedância desejados para o conjunto, de tal forma que diferentes níveis de atividade muscular do usuário são obtidos. Esse resultado foi verificado através de testes com auxílio de equipamento de eletromiografia. Além disso, o trabalho também inclui o início do desenvolvimento de uma interface gráfica de suporte aos testes com o exoesqueleto que permite a visualização de medidas de torque e posição conforme recebimento do controlador e facilita a manipulação do software de controle por pessoas não familiarizadas com a estrutura do algoritmo.

Palavras-Chave: Controle de impedâncias, controle motor, eletromiografia (EMG), exoesqueleto, robótica, tempo-real.

ABSTRACT

The objective of the present work was the design and implementation of a modular impedance control for an upper limb robotic exoskeleton. The control was developed and tested on a 1-degree-of-freedom (DoF) exoskeleton which corresponds to the human elbow joint. The software was structured modularly, in a manner that it allows more degrees of freedom to be added to the system. The implementation of the control algorithms were done in the C/C++ language, and the resulting software is run on an embedded computer running a Linux OS with a framework for real time tasks (Xenomai). The chosen control strategy allows the operator to get different muscular activity levels from the user by varying the desired impedance parameters. On this aspect, an EMG acquisition system was used to evaluate the performance of the controller. Also, this work includes the development of a graphical user interface for test support, graphical visualization of torque and position measurements, and the simplification of the controller software manipulation by people who are not familiarized with its source code.

Keywords: Impedance control, motor control, electromyography (EMG), exoskeleton, robotics, real-time.

SUMÁRIO

Lista de Ilustrações

1 Introdução **15**

1.1 Projetos anteriores 16

1.2 Estado da arte 16

1.3 Objetivos 19

1.4 Estrutura do relatório 20

2 Descrição do sistema **22**

2.1 Estrutura mecânica 22

2.2 Sensoriamento 24

2.2.1 Potenciômetro rotativo 24

2.2.2 Encoder 25

2.2.3 Célula de carga com extensômetros 25

2.3 Componentes de *hardware* 26

2.3.1 Computador embarcado 26

2.3.2 Placa de entrada-saída 26

2.3.3 Circuitos para condicionamendo de sinais 27

2.3.4 Motor 28

2.3.5 Driver do motor 29

3 Metodologia **30**

3.1 Requisitos do projeto 30

3.2 Programação 31

3.2.1 Real time - Xenomai 31

3.2.2 Maxon Epos Studio 32

3.2.3 Ferramentas de programação 32

3.3 Projeto de controle 32

4 Modelo do sistema **35**

4.1 Conversões do sistema 36

4.1.1 Mecanismo da articulação do exoesqueleto 36

4.1.2 Conversão de *setpoint* 39

4.2 Modelo do motor 40

4.2.1 Modelagem do sistema elétrico 41

4.2.2 Modelagem do sistema mecânico 43

4.2.3 Modelo eletromecânico 44

5 Controle do sistema **46**

5.1 Controle de impedâncias 47

5.1.1 Estratégia de controle de impedâncias implementada . . 48

5.2 Discretização do modelo do sistema 55

5.2.1 Discretização Bilinear 55

5.2.2 Método das diferenças - *backward difference* 57

5.3 Identificação dos parâmetros de impedância 58
5.3.1 Modelo para identificação do sistema 59
5.3.1.1 Modelo obtido por discretização bilinear 60
5.3.1.2 Modelo obtido por discretização segundo método das diferenças *backward difference* 62
5.3.2 Protocolo de testes para captura de sinais 63
5.3.3 Conversão dos sinais de tensão em variáveis mecânicas 64
5.3.3.1 Conversão do sinal de tensão do potenciômetro em sinal de posição angular 64
5.3.3.2 Conversão do sinal de tensão do *strain gage* em sinal de torque 65
5.3.4 Critérios para avaliação dos resultados 65
5.3.5 Resultados e análise 67
5.3.5.1 Resultados da identificação do modelo obtido por discretização bilinear 67
5.3.5.2 Resultados da identificação do modelo obtido por discretização segundo o método das diferenças (*backward difference*) 70
5.3.5.3 Análise dos resultados 73
5.3.6 Modificações no procedimento de identificação 74
5.3.6.1 Processamento de sinais 74
5.3.6.2 Consideração da não linearidade do sistema . . 76

5.3.7 Considerações finais sobre identificação dos parâmetros de impedância 77

5.3.8 Desempenho do controlador 78

5.4 Estrutura do software de controle de impedâncias 80

5.4.1 Ferramentas 81

5.4.2 Estrutura do software 81

6 Estudo de sinais de eletromiografia 84

6.1 Equipamento e protocolo de testes para a captura de sinais EMG 85

6.1.1 Instrumentação do usuário 85

6.1.2 Descrição dos testes 87

6.2 Análise dos resultados 88

6.2.1 Relação entre sinais de eletromiografia e atividade muscular do indivíduo 88

6.2.2 Avaliação do desempenho do controlador com sinais de EMG 90

6.2.3 Relação entre sinal de EMG e torque 95

7 Interface gráfica para o usuário 98

7.1 Ferramentas de desenvolvimento 98

7.2 Layout da interface 99

7.3 Estrutura do programa 102

7.3.1 Comunicação serial 103

8 Conclusão **105**

Referências **107**

Apêndice A – Placa de condicionamento de sinais 1 **111**

Apêndice B – Placa de condicionamento de sinais 2 **113**

Apêndice C – Códigos do Controle **115**

Apêndice D – Códigos da Interface **137**

Anexo A – Controle do driver Epos 2 **191**

LISTA DE ILUSTRAÇÕES

1 Estrutura do exoesqueleto de membro superior, vista 1. 23

2 Estrutura do exoesqueleto de membro superior, vista 2. 23

3 Célula de carga com *strain gauges*. 25

4 Módulo PCM3362 de PC/104. 27

5 Placa Diamond-MM-16-AT. 27

6 Motor Maxon EC 32, sem escovas - 80 Watt. 28

7 Driver Maxon EPOS2 24/5. 29

8 Metodologia do projeto de controle. 33

9 Modelo do sistema. Fonte: autoria própria. 35

10 Esquema do mecanismo do exoesqueleto. Fonte: autoria própria. 37

11 Relação entre α e x. Fonte: autoria própria. 38

12 Modelo do motor EC. Fonte: autoria própria. 45

13 Diagrama de blocos da arquitetura de controle de impedâncias. 48

14 Modelo em Matlab para validação da estratégia de controle. . . 51

15 Malha de controle para representação do comportamento do ser humano sem exoesqueleto. 52

16 Resultado da simulação para fator dos parâmetros de impedância desejada $c = 0.25$. 52

17 Resultado da simulação para fator dos parâmetros de impedância desejada $c = 2$. 53

18 Resultado da simulação para fator dos parâmetros de impedância desejada $c = 1$. 54

19 Resultado da simulação para fator dos parâmetros de impedância desejada $c = 0.6$. 54

20 Avaliação da qualidade do modelo estimado: modelo obtido por discretização bilinear, exoesqueleto sem usuário. 68

21 Avaliação da qualidade do modelo estimado: modelo obtido por discretização bilinear, exoesqueleto com usuário. 69

22 Avaliação da qualidade do modelo estimado: modelo obtido por discretização segundo método das diferenças, exoesqueleto sem usuário. 71

23 Avaliação da qualidade do modelo estimado: modelo obtido por discretização segundo método das diferenças, exoesqueleto com usuário. 72

24 Comparação entre os torques atuantes na célula de carga para diferentes valores do fator c dos parâmetros de impedância. . . 79

25 Pontos de fixação ideais de eletrodos para captura sinais de EMG do bíceps braquial e do tríceps braquial. Fonte: (HERMENS et al., 2000) 86

26 Pontos de fixação dos eletrodos utilizados nos testes de captura sinais de EMG. 86

27 Sinais de eletromiografia do bíceps e do tríceps e correspondente trajetória de movimento. 89

28 Máxima contração voluntária do bíceps braquial e do tríceps braquial do indivíduo 1. 90

29 Sinais de eletromiografia do bíceps braquial para usuário com exoesqueleto (diferentes coeficientes dos parâmetros de impedância desejada) e para usuário sem exoesqueleto. 92

30 Sinais de eletromiografia do tríceps braquial para usuário com exoesqueleto (diferentes coeficientes dos parâmetros de impedância desejada) e para usuário sem exoesqueleto. 93

31 Sinais de EMG do bíceps e do tríceps e correspondente variação do torque aplicado sobre a célula de carga. Teste com coeficiente $c = 5.7\ 10^{-4}$ e metrônomo a 80 bpm. 97

32 Sinais de EMG do bíceps e do tríceps e correspondente variação do torque aplicado sobre a célula de carga. Teste com coeficiente $c = 5.7\ 10^{-6}$ e metrônomo a 80 bpm. 97

33 Janela principal da interface gráfica para o usuário 100

1 INTRODUÇÃO

Uma definição simples para o conceito de "exoesqueleto", dada em (PERRY; ROSEN; BURNS, 2007), define-o como um mecanismo externo com configuração de articulações e ligações antropomórfica. Quando vestido, ele é capaz de transmitir torques de um atuador presente em sua estrutura para as articulações do usuário. Em termos de campos de aplicação, a maior parte dos exoesqueletos existentes pode ser dividida em dois grupos principais (MIKULSKI, 2011):

1. para reabilitação: a participação do exoesqueleto tem caráter auxiliar em tratamentos baseados em fisioterapia. Desta forma, o exoesqueleto pode facilitar ou dificultar o movimento, conforme necessário. Exemplos de exoesqueletos nessa categoria podem ser vistos em (MIHELJ; NEF; RIENER, 2007; VITIELLO et al., 2013; HU et al., 2012; TSAI et al., 2010; MIKULSKI, 2011; KIGUCHI; HAYASHI, 2012; PERRY; ROSEN; BURNS, 2007).

2. para amplificação de capacidade humana: o exoesqueleto amplifica o torque exercido pelo usuário, de forma que o carregamento sentido por ele é reduzido. Um exemplo conhecido deste grupo é o BLEEX ((ZOSS; KAZEROONI; CHU, 2006)).

O presente projeto não se enquadra totalmente nos grupos anteriores, sendo parte de um terceiro, que tem por objetivo desenvolver o exoesque-

leto como uma ferramenta de estudo do sistema motor humano (como em (FORNER-CORDERO et al., 2011)). O trabalho é a continuação de projetos desenvolvidos desde 2011, de forma que o protótipo mecânico e a estrutura eletrônica existentes são reutilizados, podendo ser aplicadas adaptações conforme as necessidades.

1.1 Projetos anteriores

Em (YASUTOMI; MIRANDA, 2011), foi desenvolvido o projeto do protótipo do exoesqueleto de membro superior. As etapas referentes à montagem mecânica, tais como a modelagem matemática e os desenhos em CAD, foram executadas e descritas nesse trabalho.

No ano seguinte, em (ABDUCH; RUIVO, 2012), foram aplicadas modificações no protótipo mecânico, sendo elaborada uma nova peça para a região do antebraço, o que possibilitou a instalação de uma célula de carga para a medição do torque exercido pelo usuário. Nesse trabalho foi desenvolvida a estrutura eletrônica do sistema e foi proposta uma malha de controle de alto nível por meio do *software* Simulink.

Finalmente em (RATEIRO; CESAR, 2013), foi proposta uma implementação de controle de impedâncias de baixo nível em tempo real, sendo utilizada para tal a linguagem C/C++ em Linux com extensão para suporte a tempo real.

1.2 Estado da arte

Recentemente, tem-se notado um número crescente de indivíduos atingidos por deficiência motora, fato causado por diversos fatores, tais como o

envelhecimento da população e a incidência de Acidentes Vasculares Cerebrais (AVCs). O tratamento para esses casos é normalmente baseado em exercícios mecânicos repetitivos (MIRANDA et al., 2012). Seguindo essa realidade, cada vez mais a sociedade científica e médica desperta interesse na robótica de reabilitação (RUIZ et al., 2006).

Nesse contexto, múltiplos protótipos de exoesqueletos robóticos voltados para a reabilitação podem ser encontrados na literatura. São exemplos o ARMin (existente nas versões I, II e comercial), o CADEN-7 e o NEUROExos.

O ARMin II, com 7 graus de liberdade, conta com sistema de segurança ativo e passivo. As dimensões ajustáveis das seções do braço e antebraço permitem que seja vestido por indivíduos com diferentes tipos de estruturas corporais (MIHELJ; NEF; RIENER, 2007). Também com 7 graus de liberdade, O CADEN-7 apresenta estrutura robusta e é construído sobre uma base fixa. Pode ser utilizado em reabilitação e amplificação da capacidade humana (PERRY; ROSEN; BURNS, 2007). O NEUROExos, por sua vez, possui um mecanismo passivo com 4 graus de liberdade, permitindo a adaptação da estrutura às restrições dadas pela cinemática do membro superior humano (VITIELLO et al., 2013).

Em termos de controle, exoesqueletos robóticos necessitam que a força aplicada durante a movimentação seja controlada, de forma a evitar possíveis danos ao usuário (especialmente no caso de exoesqueletos de reabilitação) e ao ambiente. No entanto, também é necessário controlar a posição do dispositivo para permitir movimentos precisos. Dadas essas restrições, o controle de impedâncias é amplamente utilizado nesse campo, uma vez que permite controlar a relação entre diferentes parâmetros. Seu princípio de ação é controlar a dureza aparente, amortecimento ou massa, de acordo com o modo de operação desejado (AREVALO; GARCIA, 2012).

Com relação aos sinais de entrada, os métodos de controle para exoesqueletos robóticos podem ser divididos em três tipos (GUNASEKARA et al., 2012):

1. baseados em sinais biológicos;
2. baseados em sinais não biológicos;
3. de plataforma independente, que utilizam ambos os tipos de sinal.

Os métodos de controle baseados em sinais biológicos têm a vantagem de apresentar desempenho satisfatório, mesmo sob a influência de distúrbios externos (GUNASEKARA et al., 2012). Nesse grupo, as abordagens de mais frequentes na literatura utilizam sinais provenientes de: (i) eletromiografia (EMG), aplicada, por exemplo, em (KIGUCHI; HAYASHI, 2012; LALITHARATNE et al., 2013); (ii) eletroencefalografia (EEG), como empregado em (LALITHARATNE et al., 2012). O primeiro consiste em obter sinais musculares por meio de eletrodos superficiais ou intramusculares, enquanto o segundo é caracterizado pela captura de sinais cerebrais de forma não invasiva (LALITHARATNE et al., 2012). Ambos agem sob o mesmo princípio de predição de intenção de movimento do usuário, porém o método por EMG tem a vantagem de detectar reflexos condicionados ou movimentos repetitivos (TAO et al., 2012).

Por meio de sinais provenientes de 16 canais de EMG, em (KIGUCHI; HAYASHI, 2012) foi proposto um método de controle de impedâncias que estima com relativa precisão a intenção de movimento do usuário. Seu diferencial é a utilização de uma matriz de peso *neurofuzzy* adaptável, de forma a reduzir variações do modelo causadas por diferenças individuais entre usuários e mudanças de postura do membro superior. Em (LALITHARATNE et al., 2013) uma estratégia de controle baseada em *fuzzy* foi apresentada com

o objetivo de compensar efeitos de fadiga muscular em controladores baseados em sinais EMG. O resultado da compensação foi considerado satisfatório, representando um dos passos iniciais no sentido de contornar os efeitos de inconsistência dos sinais EMG no contexto de controle de exoesqueletos robóticos.

No contexto dos métodos de controle baseados em sinais EEG, em (LALITHARATNE et al., 2012) foi proposto um método baseado em algoritmos genéticos para decodificação e reconstrução de perfis de velocidade da articulação do cotovelo usando sinais EEG. Resultados positivos foram obtidos em uma análise *off-line* e a adaptação deste método para análise em tempo real foi proposta como trabalho futuro, com o objetivo de possibilitar seu uso em controladores de exoesqueleto de membro superior.

No artigo de revisão de métodos de controle para exoesqueletos robóticos (GUNASEKARA et al., 2012), que envolveu os anos de 2005 a 2012, foi observado que a maior parte dos exoesqueletos robóticos tem sido implementada com sinais não biológicos e necessitam buscar formas de aumentar a precisão na detecção de intenção de movimento do usuário. Também foi notada uma tendência em se combinar sinais de EMG com estratégias de controle baseadas em sensores.

1.3 Objetivos

Este projeto tem como objetivo a implementação de um algoritmo de controle de impedâncias modular para um exoesqueleto robótico de membro superior voltado para o estudo de controle motor. Considerando-se a complexidade do sistema, a estrutura de controle do protótipo deve envolver três diferentes níveis: um controle de baixo nível, correspondente ao controle dos

atuadores das articulações através da configuração dos *drivers* correspondentes; um controle de nível intermediário, no qual a estratégia do controle de impedâncias será implementada para controlar a interação entre exoesqueleto e usuário; e, finalmente, um controle de alto nível, no qual testes de movimento podem ser programados para a realização de experimentos voltados ao estudo de controle motor.

Para permitir a operação entre diferentes níveis, o controle deve ser hierárquico. Além disso, visando uma possível expansão futura do projeto, o controle deve apresentar estrutura modular, facilitando a adição de outros graus de liberdade. Também faz parte dos objetivos do trabalho o desenvolvimento de interfaces, as quais podem ser utilizadas nos testes ao longo do desenvolvimento dos algoritmos de controle e, posteriormente, podem também auxiliar a execução e análise de testes para o estudo de controle motor.

O trabalho também deverá englobar o uso de equipamento de eletromiografia no desenvolvimento de um controlador para o exoesqueleto. Nesta etapa estão inclusos a captura dos sinais, a análise da correspondência entre o comportamento dos sinais e as modificações no sistema sensoreado, o condicionamento dos sinais para possibilitar o seu uso nas estratégias de controle e avaliação do desempenho do controlador desenvolvido.

Os requisitos do projeto em termos de controle são decorridos em maiores detalhes na seção 3.1.

1.4 Estrutura do relatório

A presente monografia encontra-se segmentado em 7 capítulos.

No capítulo 2 são listados e descritos os componentes físicos do sistema. Para fins de clareza, foi feita a divisão dos componentes nos grupos de mecâ-

nica e hardware.

No capítulo 3, os requisitos de projeto são explicados em detalhes. Também é explicada a metodologia a ser seguida no decorrer do projeto. Ainda nesse capítulo as ferramentas utilizadas no desenvolvimento do programa de controle são listadas e brevemente descritas.

O capítulo 4 contém a modelagem do sistema, a descrição de seus componentes, como conversores do sinal de controle, e a modelagem do motor de corrente contínua sem escovas.

O capítulo 5 discorre sobre o controle de impedâncias, iniciando com uma fundamentação teórica e prosseguindo com o detalhamento das etapas envolvidas no desenvolvimento do controlador.

No capítulo 6 é descrito o protocolo de testes adotado para os testes de desempenho do controlador. São mostrados os resultados obtidos, além de análises feitas sobre eles.

O sétimo capítulo apresenta informações sobre a Interface Gráfica para Usuário. São explicados o layout da tela principal, a estrutura do programa e a comunicação utilizada para interagir com o PC104.

Por fim, o capitulo 8 traz a conclusão do trabalho, junto a sugestões para trabalhos futuros.

2 DESCRIÇÃO DO SISTEMA

Este capítulo contém informações sobre os componentes do exoesqueleto de membro superior disponível no Laboratório de Biomecatrônica da Escola Politécnica da Universidade de São Paulo. Além de uma visão geral da estrutura do exoesqueleto, são apresentados detalhes dos seus componentes de *hardware* e utilizados no seu sensoriamento, controle e acionamento.

2.1 Estrutura mecânica

A atual configuração do exoesqueleto está representada na Figura 2 e é resultado do protótipo inicial desenvolvido em (YASUTOMI; MIRANDA, 2011) com as modificações introduzidas nos trabalhos de (ABDUCH; RUIVO, 2012) e (RATEIRO; CESAR, 2013).

Pode-se considerar que mecanismo possui duas partes principais, uma correspondente ao braço e a outra ao antebraço do exoesqueleto, conectadas através de uma junta rotacional correspondente ao cotovelo robótico. Ao braço do exoesqueleto está acoplado rigidamente um motor, cuja ação é regulada pelo sistema de controle.

A rotação do motor implica na rotação do fuso e, consequentemente, no movimento linear da castanha, cujo movimento de rotação é restringido por uma guia. Esta guia está acoplada ao antebraço robótico através de uma

Figura 1: Estrutura do exoesqueleto de membro superior, vista 1.

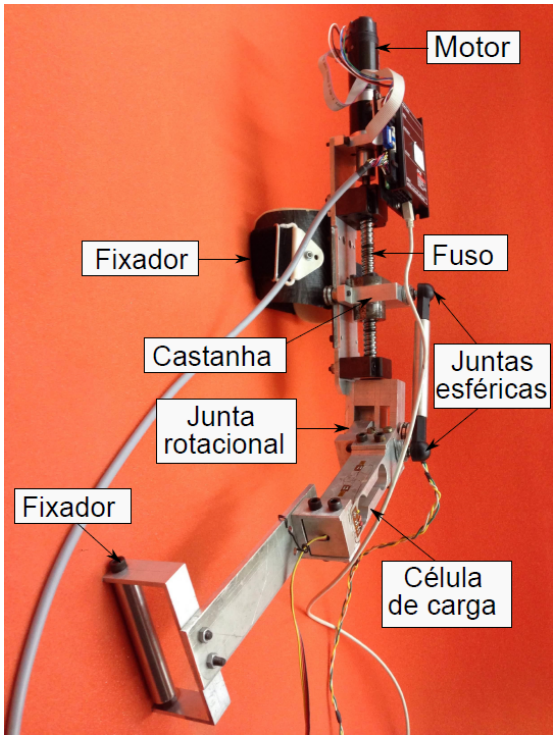
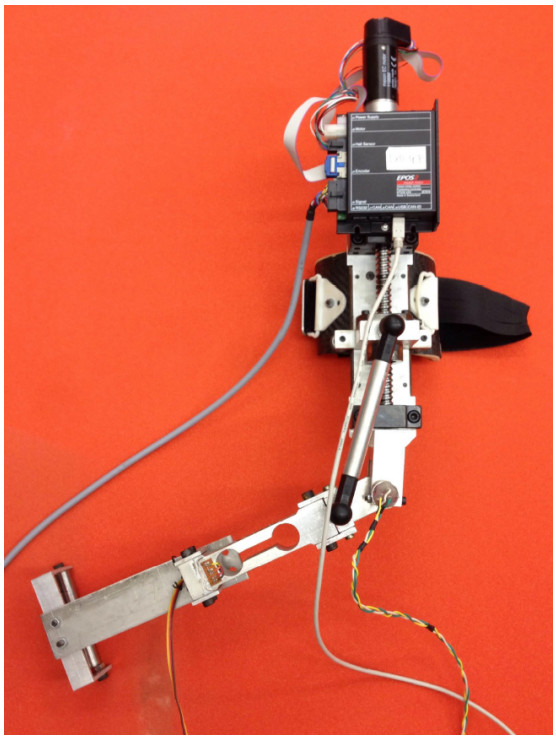


Figura 2: Estrutura do exoesqueleto de membro superior, vista 2.



haste, a qual é fixada à cada uma das partes com auxílio de juntas esféricas. Devido a este acoplamento, o movimento linear da guia impõe uma rotação

do antebraço robótico em torno do eixo da junta rotacional.

Deve ser observado que a estrutura do antebraço foi projetada de forma a conter uma célula de carga, que auxilia nas medições de torque no antebraço robótico, como apresentado em (ABDUCH; RUIVO, 2012). Mais informações sobre o sensoriamento são discutidas na seção 2.2.

Também fazem parte da estrutura os elementos para a fixação do exoesqueleto ao usuário: um superior, composto por fibra de carbono, que pode ser ajustado ao braço e um inferior, que consiste em um arco de alumínio que deve ser segurado pela mão do usuário.

2.2 Sensoriamento

O conhecimento dos valores de algumas variáveis do sistema é necessário tanto para a realização do controle do exoesqueleto robótico, como também para a avaliação do seu comportamento. Por esse motivo, o sistema conta com três sensores, os quais estão descritos nos subitens a seguir.

2.2.1 Potenciômetro rotativo

Um potenciômetro de alta precisão, acoplado à articulação do cotovelo do exoesqueleto, funciona como sensor absoluto de posição angular do antebraço. O componente permite não só a medição da variação da posição angular do antebraço robótico com relação a uma referência pré-definida, como também conhecer a posição exata deste no espaço, permitindo executar o programa sempre na mesma posição inicial.

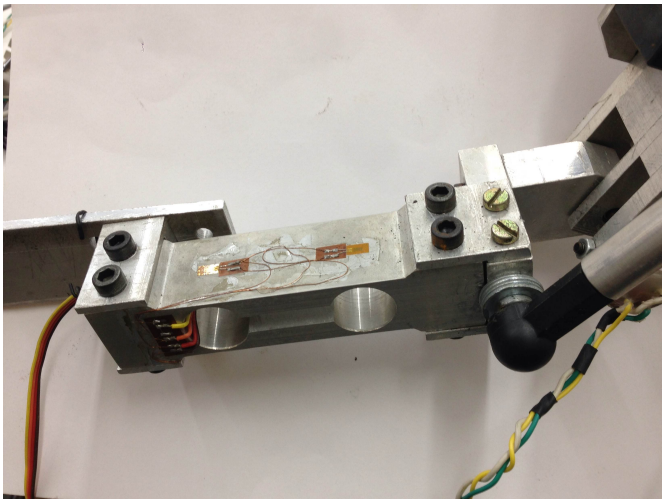
2.2.2 Encoder

O motor possui um encoder acoplado, o qual realiza medições da posição angular do eixo do motor. Estas medidas são utilizadas na realimentação da malha de controle do driver do motor, que visa assegurar precisão de posição.

2.2.3 Célula de carga com extensômetros

As medições de torque são realizadas com o auxílio de uma célula de carga não comercial acoplada ao antebraço do exoesqueleto (ABDUCH; RUIVO, 2012), como ilustrado na Figura 3.

Figura 3: Célula de carga com *strain gauges*.



Na superfície desta célula de carga estão fixados *strain gauges*, os quais apresentam uma variação de resistência em caso de deformação. Neste trabalho essa variação de resistência é medida com o auxílio de uma Ponte de Wheatstone com um divisor de tensão ajustável acoplado a um de seus polos, o qual é detalhado em (RATEIRO; CESAR, 2013). Segundo este último trabalho, o divisor resistivo corresponde a um calibrador da tensão de saída da ponte e a precisão deste é maior quanto maiores os valores de suas resistências.

2.3 Componentes de hardware

2.3.1 Computador embarcado

O computador embarcado utilizado no projeto é um módulo PCM3362 (Advantech Co., Ltd.), que segue o padrão PC/104 plus. PC/104 é um padrão de fator de forma e de barramento para computadores embarcados. Sua principal vantagem é a estrutura de barramento empilhável. Uma variação desta, o PC/104 plus, inclui suporte a barramentos PCI[1].

O PCM3362 ilustrado na Figura 4, apresenta em sua configuração uma CPU Intel Atom N450 com um núcleo de 1.66GHz, e 2 gigabytes de memória flash, além de baixo consumo energético, sendo apropriado para sistemas embarcados. Apresenta suporte a Windows, Linux e SUSIAccess, que é um software próprio da Advantech.

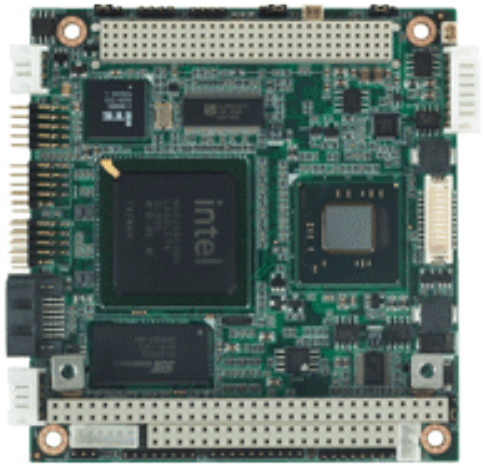
No projeto, o sistema operacional instalado é o Linux Debian. Com a assistência do Diamond-MM-16-AT, o módulo é responsável por executar o programa do controle de impedâncias.

2.3.2 Placa de entrada-saída

A Diamond-MM-16-AT, exibida na Figura 5, consiste em uma placa de extensão de entradas e saídas analógicas para computadores PC/104. Contém 16 pinos de entrada analógica, de 16 bits cada, e 4 pinos de saída, de 12 bits. Por software, é possível ajustar os níveis de tensão de entrada e saída em intervalos unipolares (0 a +Vmax) ou bipolares (-Vmax a +Vmax). Além disso, a placa permite a auto-calibração dos conversores digital-analógico e analógico-digital, por meio de referências de precisão on-board, via software.

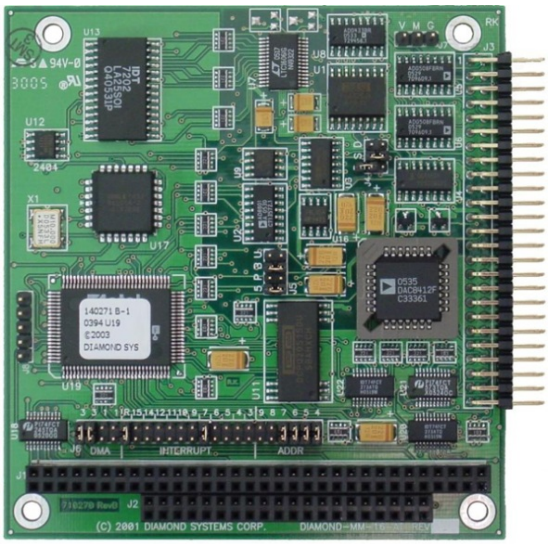
[1] http://www.pc104.org/

Figura 4: Módulo PCM3362 de PC/104.



Fonte: <http://www.advantech.com/products/1-2JKLTU/PCM-3362/mod_2b69da4c-d506-4aea-8cc3-f6e58e3cbc87.aspx>. Acesso em: 06 de Abril, 2014.

Figura 5: Placa Diamond-MM-16-AT.



Fonte: <http://www.diamondsystems.com/products/diamondmm16at>. Acesso em: 06 de Abril, 2014.

2.3.3 Circuitos para condicionamendo de sinais

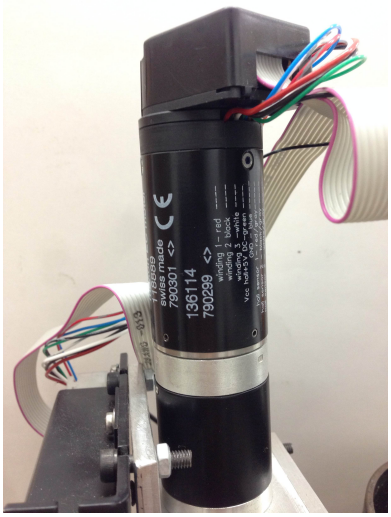
Visando o condicionamento adequado dos sinais de leitura do potenciômetro e do extensômetro, são utilizados dois circuitos de fabricação própria do laboratório, desenvolvidos em projetos anteriores. As duas placas existen-

tes incluem filtros de alta frequência e amplificação de sinal. A princípio, os sinais de ambos os sensores eram tratados na placa cujo esquemático é apresentado no Apêndice A. Posteriormente foi optado por se alterar o circuito de condicionamento do sinal dos extensômetros, de forma que o sinal de posição proveniente do potenciômetro é ainda tratado pela placa do Apêndice B, enquanto o do *strain gauge* é tratado pela representada no esquemático do Apêndice A.

2.3.4 Motor

O motor é responsável pelo acionamento do exoesqueleto rotacionando o fuso, cujo movimento implica um movimento linear da guia. Neste projeto é utilizado um motor eletronicamente comutado (EC) produzido pela Maxon Motor AG, o qual está apresentado na Figura 6. Ele pode ser encontrado em catálogos com os seguintes dados: Motor Maxon EC 32, 80 Watt, modelo 118889. Algumas das características desse motor relevantes para este trabalho encontram-se na Tabela 1.

Figura 6: Motor Maxon EC 32, sem escovas - 80 Watt.



A escolha deste motor foi baseada nas diretrizes apresentadas em (FOR-

Tensão nominal	36.0 V
Torque nominal	39.7 mNm
Corrente nominal	1.98 A

Tabela 1: Características do motor Maxon EC 32, sem escovas, 80 Watt, modelo 118889.

NER-CORDERO et al., 2011), segundo as quais a determinação do atuador é feita de acordo com o torque e a potência necessários para o movimento da articulação em questão, os quais estão relacionados à máxima velocidade angular da articulação nos pontos de operação considerados.

2.3.5 Driver do motor

O *driver* é o responsável pelo controle do motor elétrico, transformando os sinais de referência de posição enviados pelo processador em sinais adequados para uma movimentação correspondente do motor (ABDUCH; RUIVO, 2012). No sistema em estudo é utilizado o *driver* Maxon EPOS2 24/5, representado na Figura 7.

Figura 7: Driver Maxon EPOS2 24/5.

Fonte: <http://www.maxonmotor.com/medias/sys_master/8805647908894/367676_Hardware_Reference_En.pdf>. Acesso em: 07 de Abril, 2014.

3 METODOLOGIA

3.1 Requisitos do projeto

Este trabalho tem como objetivo a implementação de um controle de impedâncias para o exoesqueleto robótico apresentado no capítulo 2. Considerando os objetivos do projeto apresentados na seção 1.3, o algoritmo de controle deve ser estruturado de forma modular, permitindo sua expansão para envolver o controle de mais graus de liberdade em projetos futuros do laboratório. A estrutura de controle também deve ser hierárquica, regulando a relação entre os diferentes níveis de controle.

Além da estrutura, também devem ser feitas considerações sobre o desempenho desejado do sistema. Segundo (RATEIRO; CESAR, 2013), a eficiência de um sistema de controle depende tanto da malha de controle, como também da rapidez com que as etapas de controle são executadas. Desta forma, a implementação do controle deve possibilitar uma execução rápida das suas tarefas, as quais correspondem à leitura dos sinais dos sensores, à execução dos algoritmos de controle e ao envio dos sinais resultantes para o atuador. Neste contexto, deve-se assegurar também que não haja conflito entre o agendamento das tarefas.

Outro aspecto relevante está relacionado à segurança. Dado que o exoesqueleto robótico deve ser utilizado por uma pessoa, é fundamental que uma

hiperextensão do braço robótico seja evitada para garantir a integridade do usuário. A isso podem ser relacionadas estratégias de controle e limitações dos próprios componentes utilizados.

3.2 Programação

3.2.1 Real time - Xenomai

A eficiência de um controle depende tanto da malha de controle, como também da rapidez com que as etapas de controle são executadas (RATEIRO; CESAR, 2013). Essas etapas correspondem à leitura dos sinais dos sensores, à execução dos algoritmos de controle e ao envio dos sinais resultantes para os atuadores.

Sistemas *real time* são caracterizados por garantir um controle preciso sobre o período em que duas chamadas consecutivas da mesma tarefa são executadas (RATEIRO; CESAR, 2013). Estas tarefas, por sua vez, são funções ou conjuntos de funções que devem ser agendadas através de um pedido de interrupção do sistema, cujas ordens de execução são reguladas pelo *kernel*, núcleo da distribuição (RATEIRO; CESAR, 2013). No contexto deste trabalho, as tarefas correspondem às etapas de controle mencionadas anteriormente.

Assim, em (RATEIRO; CESAR, 2013) foi proposto o desenvolvimento dos algoritmos de controle do exoesqueleto em estudo utilizando a distribuição Debian de Linux com a extensão Xenomai para *real time*.

O Xenomai é um *framework* de tempo real para plataformas Linux. Ele disponibiliza interfaces de programação de aplicativos de sistemas operacionais de tempo real para o Linux. Mesmo quando o *kernel* do Linux não apresenta os requerimentos necessários em relação a tempos de resposta, o Xenomai os fornece por meio de sua tecnologia *co-kernel*.

Desde a versão 2.1, o Xenomai desacopla as funcionalidades de suporte ao *kernel* das bibliotecas usadas para acessá-las. Assim, torna-se possível compilá-las separadamente. No caso específico do Debian, também é possível instalar e compilar o Xenomai como um conjunto de pacotes Debian.

3.2.2 Maxon Epos Studio

O software Epos Studio da Maxon pode ser utilizado para configurar o driver Maxon EPOS2 24/5 presente no protótipo. Consiste em um programa de gerenciamento de projetos que utilizam dispositivos Maxon, possibilitando inclusive calibrar atuadores e selecionar o tipo de referência que o driver deve receber.

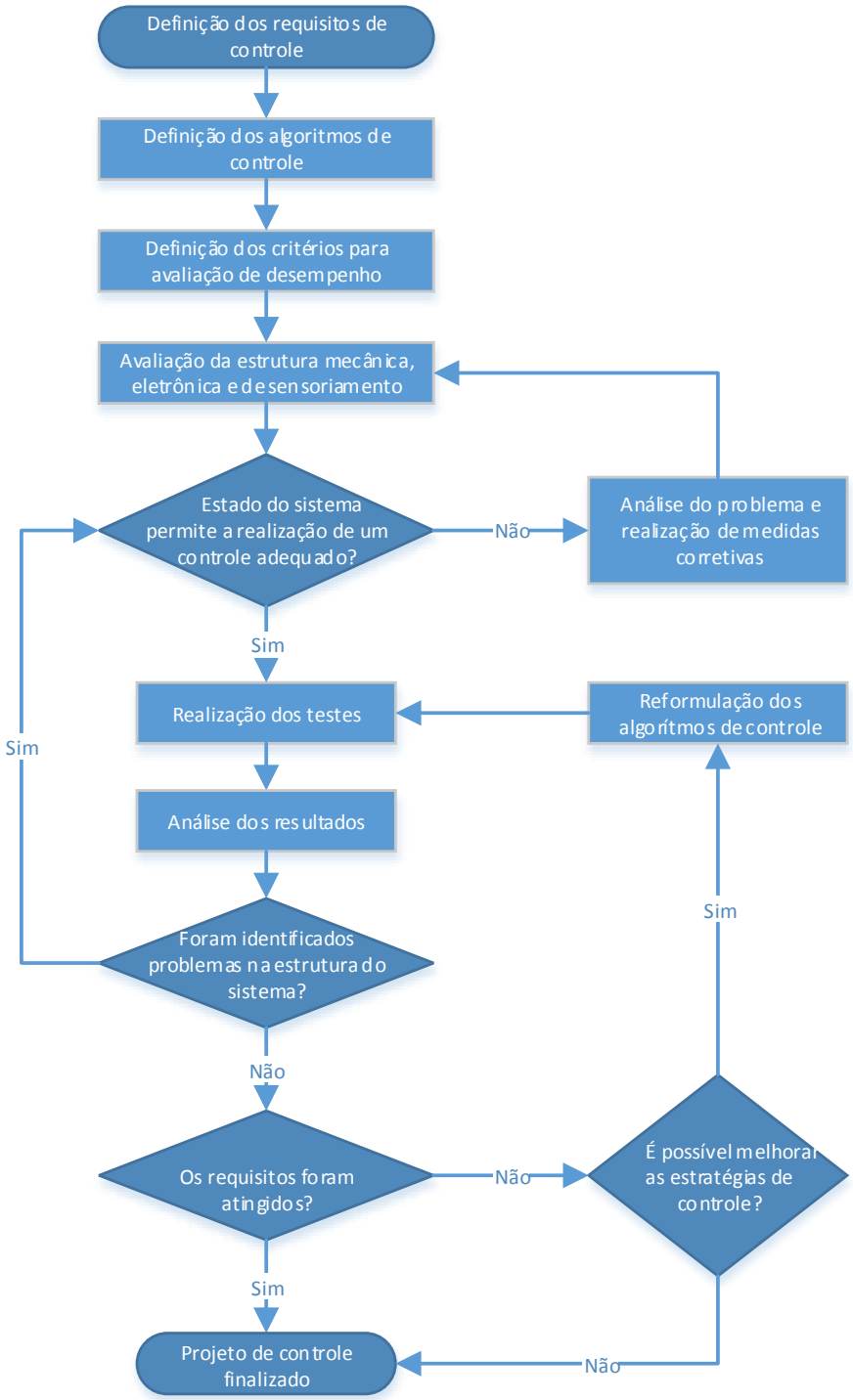
3.2.3 Ferramentas de programação

Para o desenvolvimento e compilação do programa do controle de impedâncias, é utilizado o ambiente integrado de desenvolvimento Eclipse CDT para programação em C/C++. Instalado em uma máquina virtual com sistema operacional Linux Debian, o Eclipse CDT também é utilizado para compilar o programa, gerando o executável a ser transferido para o controlador. A máquina virtual é emulada por meio do software Virtual Box, da Oracle.

3.3 Projeto de controle

Para a execução do projeto de controle foi proposta a metodologia apresentada na Figura 8. De acordo com o diagrama, primeiramente se definem os requisitos de controle, ou seja, o que o controle desenvolvido deve ser capaz de realizar. Completada essa etapa inicial, uma primeira versão dos algoritmos de controle pode ser desenvolvida e os critérios de avaliação devem

Figura 8: Metodologia do projeto de controle.



ser definidos. Estes podem corresponder a variáveis que podem ser diretamente medidas ou, então, à variáveis que podem ser determinadas a partir

dos valores medidos.

Como a realização adequada do controle depende das condições de funcionamento da parte mecânica, eletrônica e de sensoriamento, essas devem ser avaliadas no início do projeto. Caso algum problema seja identificado, este deve ser analisado e medidas corretivas devem ser executadas até a solução do problema. Apenas quando o sistema apresentar um estado adequado pode-se seguir com o projeto de controle.

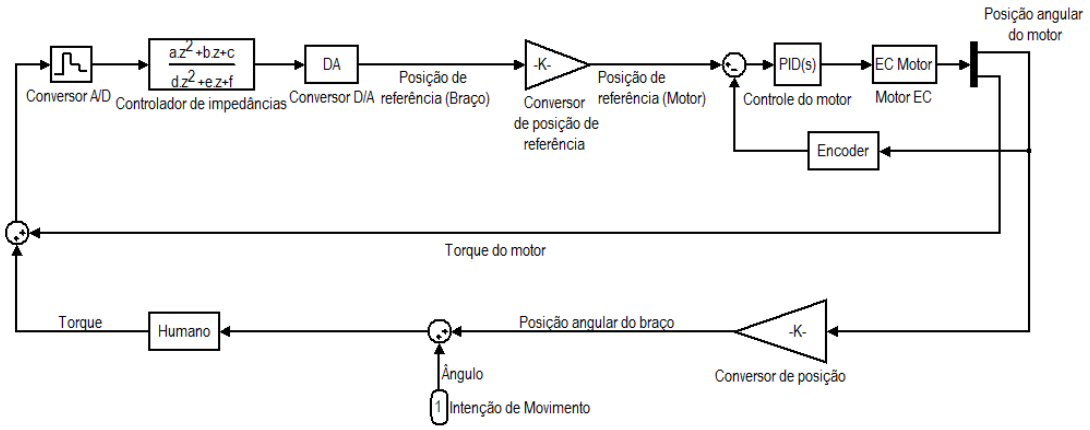
Os algoritmos de controle devem então ser avaliados através da execução de testes e análise dos resultados obtidos. Se algum problema da estrutura for identificado durante essa etapa, deve-se voltar ao laço superior de medidas corretivas dos problemas estruturais. Do contrário, segue-se normalmente a verificação do atendimento aos requisitos de controle. Caso todos os requisitos de controle tenham sido atingidos, o projeto de controle é finalizado. Em caso negativo, deve-se avaliar se é possível melhorar as estratégias de controle considerando os recursos disponíveis e os prazos do cronograma.

Se não for possível realizar melhorias o projeto de controle é finalizado. Mas se for possível, as estratégias de controle devem ser reformuladas e reavaliadas como apresentado anteriormente, dando origem a um ciclo.

4 MODELO DO SISTEMA

O modelo do sistema usuário-exoesqueleto é representado abaixo, na figura 9.

Figura 9: Modelo do sistema. Fonte: autoria própria.



O controle de impedâncias, implementado no PCM3362, recebe o sinal digitalizado de torque do *strain gauge* e gera o sinal analógico de referência de ângulo do braço. A tensão gerada, de 0 a 5 volts, já recebida pelo *driver*, é convertida para contagens de quadratura e recebe realimentação negativa do sinal do encoder. Essa conversão é explicada em maiores detalhes na subseção 4.1.2. O erro é então aplicado no controlador do *driver*, que controla a posição do motor enviando-lhe as correntes das três fases. O controle do *driver* Epos 2 é explicado no anexo A. O modelo do motor é explicado na seção 4.2. Posteriormente, a rotação do motor é transformada em movimento linear pelo mecanismo de fuso de esferas e castanha e finalmente convertido

em movimento de rotação do braço por meio da barra com juntas esféricas. O mecanismo envolvido nesta etapa é detalhado na subseção 4.1.1.

4.1 Conversões do sistema

Pela observação da modelagem do sistema no Capítulo 4, nota-se a presença de duas conversões do sinal de controle. Uma converte o sinal de *setpoint* de ângulo do braço, enviado pelo controle de impedâncias, em um sinal de referência para o motor, sendo implementada totalmente pelo *driver* Epos 2. A outra, consistindo em múltiplos componentes mecânicos, realiza a conversão inversa, transformando a rotação do motor em rotação do braço.

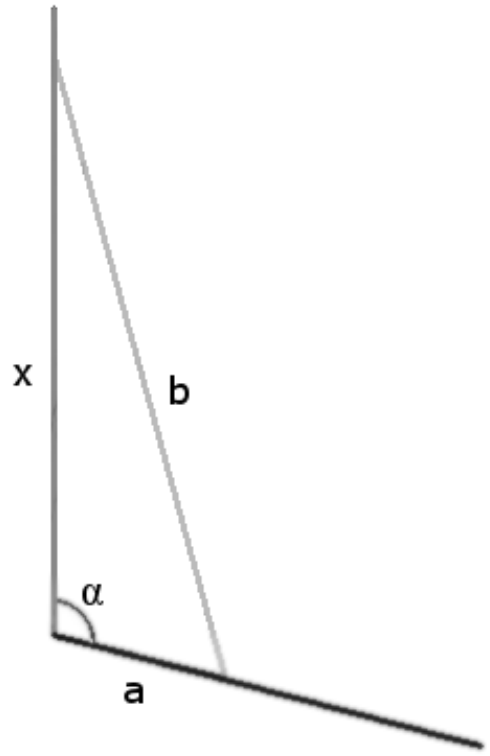
4.1.1 Mecanismo da articulação do exoesqueleto

A transmissão da rotação do motor para o braço é composta por múltiplos componentes: um redutor de rotação (redução 14:1), um fuso de passo $p = 5mm$, e o mecanismo do exoesqueleto. Dado que o redutor e o fuso podem ser considerados simples ganhos no modelo do sistema, o componente mais relevante nesta planta é o mecanismo do exoesqueleto, que converte o movimento linear da castanha em rotação do braço.

Sabe-se que esse mecanismo apresenta comportamento não linear. Porém, dentro de um dado intervalo ao redor da posição horizontal do antebraço, é possível considerá-lo aproximadamente linear. No trabalho de (RATEIRO; CESAR, 2013), tal intervalo foi considerado como sendo de $\pm 30°$, porém foi decidido reduzir esse espaço de trabalho para $\pm 20°$ devido a restrições impostas pelo *driver*.

O mecanismo é exibido acima, na Figura 10, onde o trecho x tem comprimento variável e representa a distância da castanha à articulação do cotovelo.

Figura 10: Esquema do mecanismo do exoesqueleto. Fonte: autoria própria.



As dimensões a e b possuem comprimento fixo, correspondendo à distância da ligação da junta esférica até a articulação, e ao comprimento da barra transmissora de movimento, respectivamente.

Assim, para calcular a relação entre o ângulo α e o deslocamento x da castanha, parte-se de duas equações para o cálculo da área do triângulo, a Fórmula de Heron e o produto vetorial:

$$s = \frac{x + a + b}{2} \tag{4.1}$$

$$Area = \sqrt{s \times (s - a) \times (s - b) \times (s - x)}; \tag{4.2}$$

$$Area = \frac{\left\| \vec{a} \times \vec{x} \times sin(\alpha) \right\|}{2} \tag{4.3}$$

$$Area = \frac{axsin(\alpha)}{2}. \tag{4.4}$$

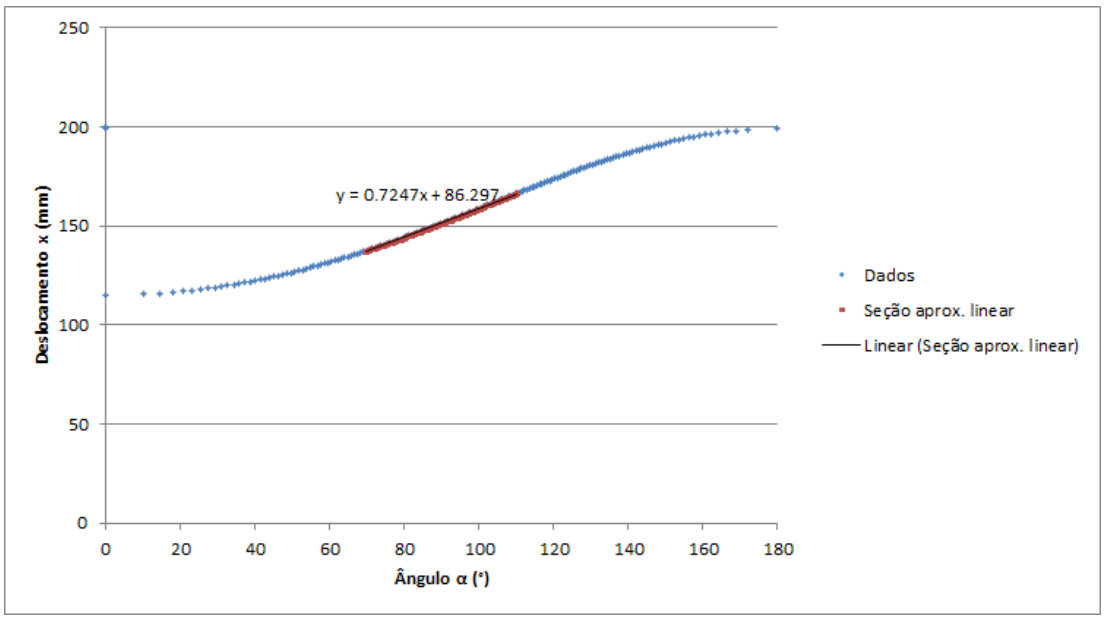
Unindo-se as Equações (4.2) e (4.4), tem-se:

$$\frac{a^2x^2sin^2(\alpha)}{4} = s(s-a)(s-b)(s-x). \tag{4.5}$$

Por meio de medições diretas, sabe-se que $a = 42mm$ e $b = 157mm$.

Iterando-se os valores de x em variações de $0,5mm$ e montando-se uma tabela relacionando tais valores com os ângulos resultantes, nota-se que a equação acima apresenta resultados válidos no intervalo de $x = 115mm$ a $x = 199mm$

Figura 11: Relação entre α e x. Fonte: autoria própria.



Tomando os pontos internos ao intervalo de interesse $(90° \pm 20°)$ e executando uma regressão linear, chegou-se à relação:

$$x = 0,7247\alpha + 86,297, \tag{4.6}$$

ou

$$\Delta x = 0,7247\Delta\alpha. \tag{4.7}$$

Incluindo o ganho do redutor e do fuso:

$$\Delta\theta_{braco} = \frac{1}{14} \times \frac{5}{360} \times \frac{1}{0,7247} \times \Delta\theta_{motor}. \tag{4.8}$$

4.1.2 Conversão de setpoint

A conversão de *setpoints* de rotação do braço em referência para o motor, é implementada totalmente pelo *driver*, por meio da configuração de um parâmetro do software Epos Studio.

O sinal de referência emitido pelo Diamond-MM-16-AT é calculado como rotação da articulação, e reescalado para um sinal analógico, unipolar e com limite de 0 a 5 volts.

Dado o conjunto de motor e encoder utilizado nas configurações do driver, o usuário pode selecionar o valor de contagens de quadratura do encoder a ser incrementado por volt recebido no canal de input do *driver*.

Dos componentes mecânicos e eletrônicos, tira-se as seguintes informações:

$$\text{Resolução do encoder} = 2000\frac{qc}{volta(motor)} = 28000\frac{qc}{\text{volta (fuso)}}; \tag{4.9}$$

$$V_{\text{máxima admissível no driver}} = 5V; \tag{4.10}$$

$$\text{passo do fuso} = \frac{5mm}{\text{volta (fuso)}}. \tag{4.11}$$

Utilizando as informações da resolução do encoder em conjunto com o passo do fuso, tem-se o deslocamento linear da castanha em função da contagem

de quadraturas do encoder (1 pulso corresponde a 4 quadraturas):

$$\frac{\text{quadraturas}}{\Delta x} = \frac{28000}{5} = 5600\frac{qc}{mm}. \tag{4.12}$$

Para converter o deslocamento linear da castanha em rotação angular do braço, utiliza-se a Equação (4.1.1), encontrada na seção anterior:

$$\frac{\text{quadraturas}}{\text{rotação do braço}} = 5600 \times 0,7247 = 4058,32\frac{qc}{^\circ}. \tag{4.13}$$

Para um intervalo de $\pm 20^\circ$, é necessário que cada volt fornecido ao *driver* corresponda a uma variação de 8° do braço. Finalmente, a relação entre quadraturas do encoder e a tensão recebida como input é dada por:

$$\frac{quadraturas}{V_{ref}} = 4058,32 \times 8 = 32466,56\frac{qc}{V}. \tag{4.14}$$

Assim, basta configurar esse parâmetro no software do *driver* para garantir que a conversão seja feita corretamente. O principal motivo que levou à alteração do espaço de trabalho do exoesqueleto é o limite imposto, pelo Epos Studio, no valor máximo aceito para este parâmetro. O valor inserido pelo usuário para o parâmetro é armazenado em uma variável de 15 bits e possui valor máximo de $32767\frac{qc}{V}$, insuficiente para garantir movimentos de $\pm 30^\circ$ ao redor da posição inicial do braço.

Desta forma, no lugar da alteração do intervalo de trabalho, outras possíveis alternativas, porém menos viáveis, incluem a troca do redutor, do encoder ou do *driver*.

4.2 Modelo do motor

O motor utilizado para o acionamento do exoesqueleto foi previamente descrito na Subseção 2.3.4. Uma vez que a dinâmica do motor é introduzida no sistema, sua correta modelagem é essencial para o desenvolvimento do

controle do exoesqueleto.

Motores eletronicamente comutados (EC), também chamados "'*direct current brushless motors*"' ou "'motores de corrente contínua sem escovas"', funcionam por meio da comutação de correntes em sua armadura fixa. Ao contrário de motores de escova, motores EC apresentam em seus rotores imãs permanentes, e a comutação das correntes na armadura é feita por um controlador, ou seja, eletronicamente.

O princípio de funcionamendo de um motor EC difere dos motores DC comuns por apresentar o campo variável na armadura, enquanto o rotor possui um campo magnético fixo e tenta manter-se alinhado com o campo fornecido pelo estator.

A seguinte modelagem do motor EC teve como base o trabalho de (RAMBABU, 2007).

4.2.1 Modelagem do sistema elétrico

O motor apresenta três enrolamentos no estator, defasados de 120° entre si. A corrente induzida no rotor pode ser desprezada devido às elevadas resistividades dos imãs e do aço.

Cada enrolamento apresenta em seu circuito uma resistência constante e e uma indutância constante em série. Também está presente em cada circuito uma força contra eletromotriz, induzida na armadura pela rotação do rotor. Assim, tem-se, para os três enrolamentos:

$$\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \begin{bmatrix} R_s & 0 & 0 \\ 0 & R_s & 0 \\ 0 & 0 & R_s \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \frac{d}{dt} \begin{bmatrix} L_{aa} & L_{ab} & L_{ac} \\ L_{ba} & L_{bb} & L_{bc} \\ L_{ca} & L_{cbs} & L_{cc} \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \begin{bmatrix} e_a \\ e_b \\ e_c \end{bmatrix}, \qquad (4.15)$$

onde v_i são as tensões dos enrolamentos do estator; R_s é a resistência de cada enrolamento do estator; L_{ii} são as auto-indutâncias; L_{ij} são as indutâncias mútuas entre os enrolamentos e i_i são as correntes que passam em cada enrolamento.

Assumindo que os enrolamentos são idênticos, temos:

$$L_{aa} = L_{bb} = L_{cc} = L \tag{4.16}$$

$$L_{ab} = L_{ac} = L_{ba} = L_{bc} = L_{ca} = L_{cb} = M. \tag{4.17}$$

Com as equações (4.16) e (4.17), a Equação 4.15 ganha a forma:

$$\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \begin{bmatrix} R_s & 0 & 0 \\ 0 & R_s & 0 \\ 0 & 0 & R_s \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \frac{d}{dt} \begin{bmatrix} L & M & M \\ M & L & M \\ M & M & L \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \begin{bmatrix} e_a \\ e_b \\ e_c \end{bmatrix}. \tag{4.18}$$

As correntes das fases tem valor restrito para que sua soma dê zero:

$$i_a + i_b + i_c = 0. \tag{4.19}$$

Portanto, torna-se possível utilizar a simplificação:

$$Mi_b + Mi_c = -Mi_a, \tag{4.20}$$

de forma que o sistema fica desacoplado:

$$\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \begin{bmatrix} R_s & 0 & 0 \\ 0 & R_s & 0 \\ 0 & 0 & R_s \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \frac{d}{dt} \begin{bmatrix} L-M & 0 & 0 \\ 0 & L-M & 0 \\ 0 & 0 & L-M \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \begin{bmatrix} e_a \\ e_b \\ e_c \end{bmatrix}. \quad (4.21)$$

$$v_i = R_s i_i(t) + (L-M)\frac{di_i(t)}{dt} + e_i(t) \quad (4.22)$$

4.2.2 Modelagem do sistema mecânico

Primeiramente, para maior clareza, listamos e denominamos os diferentes torques envolvidos. São eles o torque gerado pelo motor, T_g; o torque de inércia do motor, T_J; o torque de carga, T_L, e o torque de atrito viscoso, T_B.

O torque de atrito viscoso é proporcional à velocidade de rotação do rotor, e é dado por:

$$T_B = B\omega(t), \quad (4.23)$$

enquanto o torque de inércia, proporcional à aceleração angular, é considerado como sendo:

$$T_J = J\frac{d\omega(t)}{dt}, \quad (4.24)$$

Assim, a relação entre os torques envolvidos e a velocidade de rotação do eixo do motor é dada por:

$$T_g(t) = T_L(t) + T_J(t) + T_B(t). \quad (4.25)$$

Inserindo as equações (4.23) e (4.24), temos:

$$T_g(t) - T_L(t) = J\frac{d\omega(t)}{dt} + B\omega(t). \tag{4.26}$$

4.2.3 Modelo eletromecânico

Introduzindo-se a equação do torque eletromagnético:

$$T_g(t) = \frac{e_a i_a(t) + e_b i_b(t) + e_c i_c(t)}{\omega}, \tag{4.27}$$

é possível acoplar os modelos elétrico e mecânico. Aplicando a Transformada de Laplace com condição inicial zero nas equações (4.21), (4.26) e (4.27) temos:

$$\begin{bmatrix} e_a(s) \\ e_b(s) \\ e_c(s) \end{bmatrix} = -\begin{bmatrix} R_s & 0 & 0 \\ 0 & R_s & 0 \\ 0 & 0 & R_s \end{bmatrix}\begin{bmatrix} i_a(s) \\ i_b(s) \\ i_c(s) \end{bmatrix} - s\begin{bmatrix} L-M & 0 & 0 \\ 0 & L-M & 0 \\ 0 & 0 & L-M \end{bmatrix}\begin{bmatrix} i_a(s) \\ i_b(s) \\ i_c(s) \end{bmatrix} + \begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix}. \tag{4.28}$$

$$\frac{T_g(s) - T_L(s)}{\omega(s)} = \frac{1}{Js + B}. \tag{4.29}$$

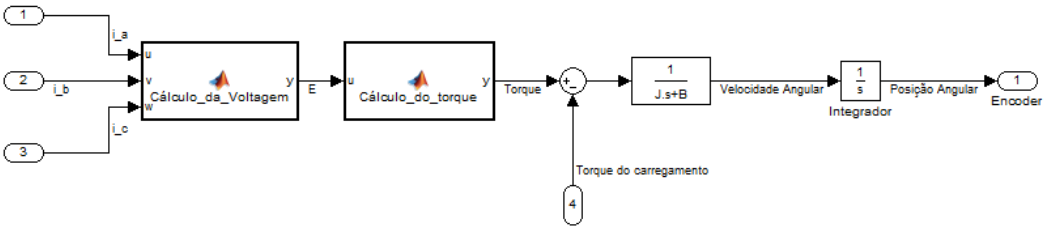
$$T_g(s) = \frac{e_a(s) i_a(s) + e_b(s) i_b(s) + e_c(s) i_c(s)}{\omega}, \tag{4.30}$$

Desta forma, obtemos o seguinte modelo:

onde:

- *Cálculo da Voltagem* implementa a equação (4.28);

Figura 12: Modelo do motor EC. Fonte: autoria própria.



- *E* carrega as informações de $i_i(s)$ e $e_i(s)$, $i = a, b, c$;
- *Cálculo do Torque* implementa a equação (4.30).

5 CONTROLE DO SISTEMA

Conforme mencionado anteriormente, o objetivo deste trabalho é fazer uma implementação de controle geral do protótipo em três níveis de controle diferentes.

O nível mais baixo consiste no controle do motor EC por meio do driver. O controle é feito pelo driver, e pode ser configurado pelo usuário por meio da definição dos ganhos do controle PID, como feito em (DOBRIANSKYJ; COUTINHO, 2013). Alternativamente, o software Epos Studio possui um método de auto-calibração, que pode ser utilizado para obter tais parâmetros de forma automática.

O nível médio de controle corresponde ao controle de impedâncias. Neste nível, o objetivo é controlar a relação entre posição e força no acionamento da articulação do cotovelo do exoesqueleto, considerando o controle de nível baixo uma caixa preta.

O nível alto corresponde à utilização do protótipo para quaisquer fins pretendidos. No caso de atividades de reabilitação, o alto nível consiste na determinação da sequência de movimentos e tarefas a serem seguidas pelo paciente. No caso do projeto, futuras aplicações terão como objetivo principal o estudo do controle motor humano.

Este projeto tem foco no nível médio de controle, com possibilidade de avanço até o nível alto. Os passos para projetar o controle de impedâncias

incluem a discretização do modelo do sistema, a identificação dos parâmetros do modelo discretizado e, por fim, o cálculo dos parâmetros do controlador.

5.1 Controle de impedâncias

O controle de impedâncias pode ser definido como uma estratégia de controle que determina o compromisso entre exatidão de posição e o controle de força (OLAYA, 2008). Segundo (HOGAN, 1987), a principal ideia por trás do controle de impedâncias é a compreensão de que a tarefa do robô não pode ser descrita unicamente em termos de força ou de movimento, mas sim em função da interação entre eles. Neste contexto, a dinâmica do comportamento interativo do robô com o ambiente é considerada parte integrante da tarefa e não apenas um distúrbio (HOGAN, 1987).

Assim, neste trabalho, o controle de impedâncias deve garantir o compromisso de controle de precisão e de força considerando a dinâmica de interação entre exoesqueleto e usuário. Para isso, primeiramente uma descrição do comportamento mecânico do sistema foi realizada. No sistema em estudo a impedância pode ser definida em termos de torque e posição angular, ou seja,

$$Z(t) = \frac{\tau(t)}{\theta(t)}. \tag{5.1}$$

O comportamento dinâmico das articulações do membro superior humano é definido neste trabalho por um modelo linear de segunda ordem, como apresentado em (5.2).

$$\frac{\theta_{\mathrm{h}}(s)}{\tau_{\mathrm{h}}(s)} = \frac{1}{J_{\mathrm{h}}s^2 + B_{\mathrm{h}}s + K_{\mathrm{h}}}. \tag{5.2}$$

Este modelo relaciona a posição angular da articulação com o torque aplicado pelo humano através dos parâmetros do usuário: a inércia J_{h}, a viscosidade B_{h} e a rigidez K_{h}. Este modelo é amplamente utilizado na literatura, uma

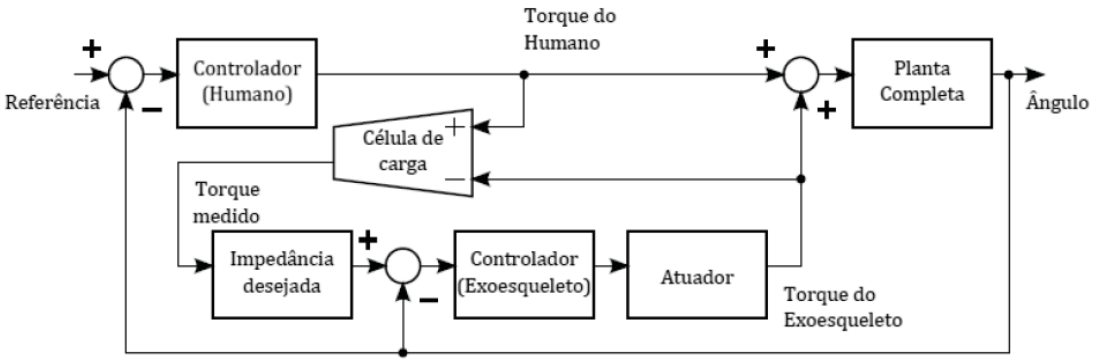
vez que aproxima razoavelmente as propriedades mecânicas dominantes da articulação (OLAYA, 2008). Comparando-se (5.1) e (5.2), pode-se identificar que J_{h}, B_{h} e K_{h} correspondem aos parâmetros de impedância do humano. Será considerado que o exoesqueleto apresenta uma dinâmica semelhante a do humano, com alteração apenas dos parâmetros:

$$\frac{\theta_{\mathrm{exo}}(s)}{\tau_{\mathrm{exo}}(s)} = \frac{1}{J_{\mathrm{exo}}s^2 + B_{\mathrm{exo}}s + K_{\mathrm{exo}}}. \tag{5.3}$$

5.1.1 Estratégia de controle de impedâncias implementada

A estratégia de controle de impedâncias utilizada neste trabalho corresponde à apresentada em (ARAUJO; TANNURI; FORNER-CORDERO, 2012), a qual foi desenvolvida com base nos resultados de (HOGAN, 1987), (AGUIRRE-OLLINGER et al., 2007) e (AGUIRRE-OLLINGER, 2009). A arquitetura de controle correspondente encontra-se representada no diagrama de blocos da Figura 13.

Figura 13: Diagrama de blocos da arquitetura de controle de impedâncias.



Fonte: (ARAUJO; TANNURI; FORNER-CORDERO, 2012).

Do diagrama pode-se observar que a entrada do controlador de impedâncias é o torque de interação entre humano e exoesqueleto, dado por

$$\tau_{\mathrm{m}} = \tau_{\mathrm{h}} - \tau_{\mathrm{exo}}. \tag{5.4}$$

Esta diferença corresponde ao torque medido pela célula de carga, a qual neste trabalho é dada como uma função do sinal de tensão medido pelo *strain gage*. A posição desejada do exoesqueleto é, então, determinada através da multiplicação desse sinal pela função de transferência da impedância desejada, ou seja,

$$\frac{\theta_{\mathrm{d}}(s)}{\tau_{\mathrm{m}}(s)} = \frac{1}{J_{\mathrm{d}}s^2 + B_{\mathrm{d}}s + K_{\mathrm{d}}}. \tag{5.5}$$

Este sinal de referência de posição é, por sua vez, enviado ao controlador do *driver* do motor. No caso do *driver* MAXON EPOS2 24/5, tem-se uma malha de controle com realimentação negativa da posição medida do motor e um PID que fornece a referência de torque do motor.

No contexto de exoesqueletos robóticos, de acordo com (ARAUJO; TANNURI; FORNER-CORDERO, 2012), o controle de impedâncias pode apresentar três modos de atuação: neutralizar a própria impedância do exoesqueleto, de tal forma que a presença do exoesqueleto não interfira no movimento do usuário; amplificar o desempenho dinâmico em termos de força ou velocidade de movimento; e, por fim, opor-se ao movimento do usuário, implicando um esforço físico maior, o que pode ser utilizado para a realização de exercícios físicos. A seleção entre esses modos de atuação é dada pela a escolha adequada dos parâmetros de impedância desejada, conforme descrito a seguir.

Segundo (SICILIANO; SCIAVICCO; VILLANI, 2009) quando a força de interação é gerada pelo contato do manipulador, o exoesqueleto, com um ambiente com massa, amortecimento e rigidez próprias, como é o caso do humano, o conjunto formado por manipulador e ambiente pode ser tratado como um sistema de duas impedâncias em paralelo, cujo comportamento dinâmico é condicionado pelo peso relativo entre as mesmas. Este conceito foi explorado no trabalho de Araujo et al (ARAUJO; TANNURI; FORNER-CORDERO, 2012), no qual a seleção da funcionalidade do controle de impedâncias foi

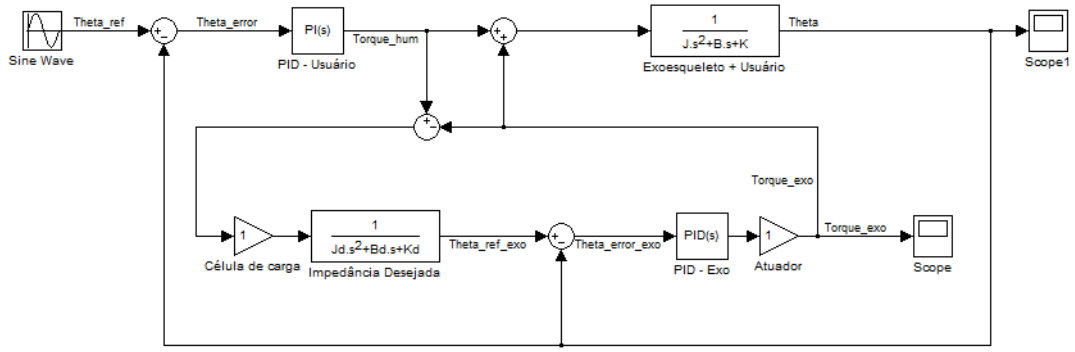
determinada pela relação entre os parâmetros da impedância desejada e os parâmetros do humano. Segundo a hipótese apresentada no trabalho mencionado, a seguinte relação entre os parâmetros de impedância e os modos do controle pode ser definida:

- Parâmetros da impedância desejada **menores** que os parâmetros do humano: amplificação do desempenho dinâmico;
- Parâmetros da impedância desejada **iguais** aos parâmetros do humano: neutralização da impedância do exoesqueleto (controle seguidor);
- Parâmetros da impedância desejada **maiores** que os parâmetros do humano: oposição ao movimento do humano.

Essas relações entre os parâmetros de impedância desejada e o comportamento do sistema foram verificadas através de simulações do modelo apresentado na figura 14, seguindo-se parcialmente o modelo apresentado em (ARAUJO, 2014). Esse modelo apresenta todos os elementos da malha de controle apresentada na figura 13. O controlador humano foi aproximado por um por um controlador PI com ganho proporcional $P = 20$ e ganho integral $I = 1$. Essa simplificação pode ser considerada razoável uma vez que o objetivo do modelo é a avaliação do desempenho do controlador do exoesqueleto na estratégia apresentada (ARAUJO, 2014). Este controlador, por sua vez, consiste em um PID na situação real e os parâmetros do modelo correspondem aos parâmetros reais: $P = 254$, $i = 1161$ e $D = 296$. O ganho do sensor e a dinâmica do atuador foram desconsiderados, o que foi considerado aceitável para a avaliação do controle de impedâncias.

Os parâmetros impedância do ser humano foram escolhidos entre alguns exemplos de parâmetros reais apresentados em (OLAYA, 2008) e são iguais a: $J_{hum} = 2.0\ 10^{-3}\ Nms^2/^\circ$, $B_{hum} = 4.4\ 10^{-2}\ Nms/^\circ$ e $K_{hum} = 1.5\ 10^{-1}\ Nm/^\circ$. Os

Figura 14: Modelo em Matlab para validação da estratégia de controle.



parâmetros de impedância do exoesqueleto foram considerados iguais aos parâmetros do ser humano multiplicados por um fator de 0.4. Finalmente, cada parâmetro de impedância do conjunto exoesqueleto-usuário foi calculado como a soma dos parâmetros correspondentes do exoesqueleto e do ser humano.

A malha de controle foi simulada para diferentes valores de um fator c que define os parâmetros da impedância desejada em função dos parâmetros do humano de acordo com as equações 5.6, 5.7 e 5.8.

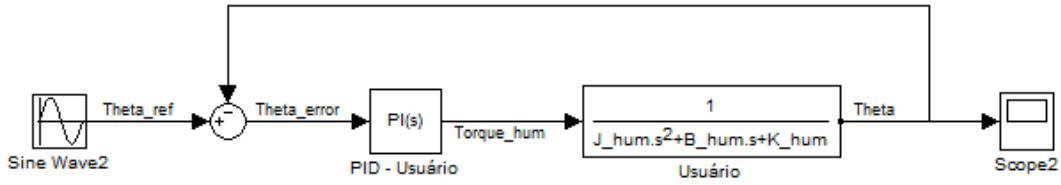
$$J_d = c * J_{hum} \tag{5.6}$$

$$B_d = c * B_{hum} \tag{5.7}$$

$$K_d = c * K_{hum} \tag{5.8}$$

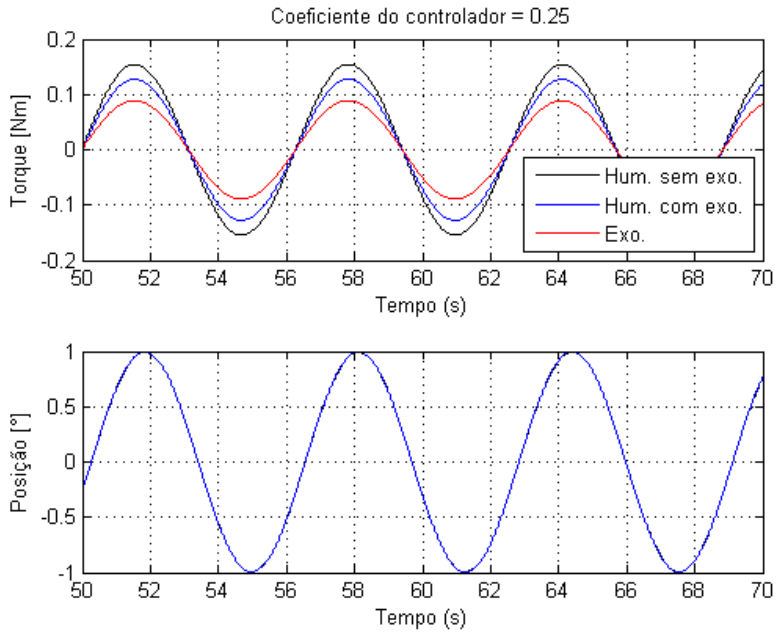
Visando a verificação do correto funcionamento da estratégia de controle, as medidas de posição e torque do modelo em questão foram comparadas às medidas de uma malha que representa o comportamento do ser humano sem exoesqueleto, a qual se encontra na figura 15.

Figura 15: Malha de controle para representação do comportamento do ser humano sem exoesqueleto.



A figura 16 mostra o resultado da simulação para $c = 0.25$. Do gráfico pode-se observar que, neste caso, o torque do humano com exoesqueleto é menor que o torque do humano sem exoesqueleto. Isso é consequência da contribuição do exoesqueleto, cujo torque é aplicado no mesmo sentido do torque do humano com uma magnitude grande o suficiente para diminuir o esforço do humano na execução do movimento. Esse comportamento está de acordo com o esperado para parâmetros da impedância desejada menores que os parâmetros do humano.

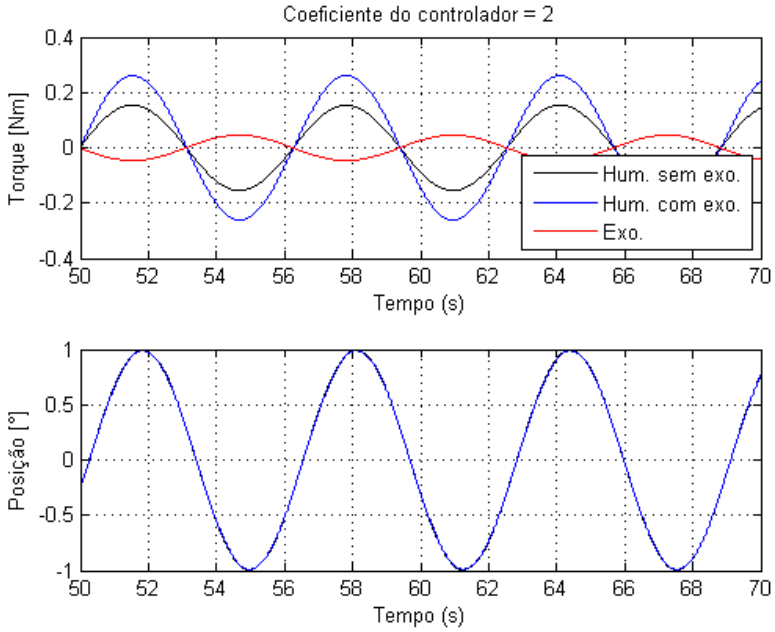
Figura 16: Resultado da simulação para fator dos parâmetros de impedância desejada $c = 0.25$.



Na figura 17 é apresentado o resultado da simulação para $c = 2$. Neste

caso, o torque do exoesqueleto atua no sentido contrário ao do torque do ser humano, de tal forma que, para realizar o mesmo movimento, o usuário com exoesqueleto precisa realizar um esforço maior do que sem o exoesqueleto. Isso caracteriza o modo de controle de oposição ao movimento para parâmetros da impedância desejada maiores que os parâmetros do humano e, portanto, é um resultado coerente.

Figura 17: Resultado da simulação para fator dos parâmetros de impedância desejada $c = 2$.



Finalmente, os resultados da simulação para $c = 1$ encontram-se na figura 18. Ao contrário do esperado, apesar de apresentarem valores próximos, os torques do usuário com e sem exoesqueleto para o fator $c = 1$não são iguais. Esse comportamento é resultado de uma pequena diferença entre as trajetórias seguidas pelas duas malhas, as quais são também são bem próximas entre si mas não são exatamente coincidentes. O modo de controle seguidor é, contudo, implementável com a estratégia proposta, mas para o modelo em questão o fator correspondente teve que ser reajustado para $c = 0.6$ devido a essa pequena diferença nas trajetórias. O resultado pode ser

encontrado na figura 19.

Figura 18: Resultado da simulação para fator dos parâmetros de impedância desejada $c = 1$.

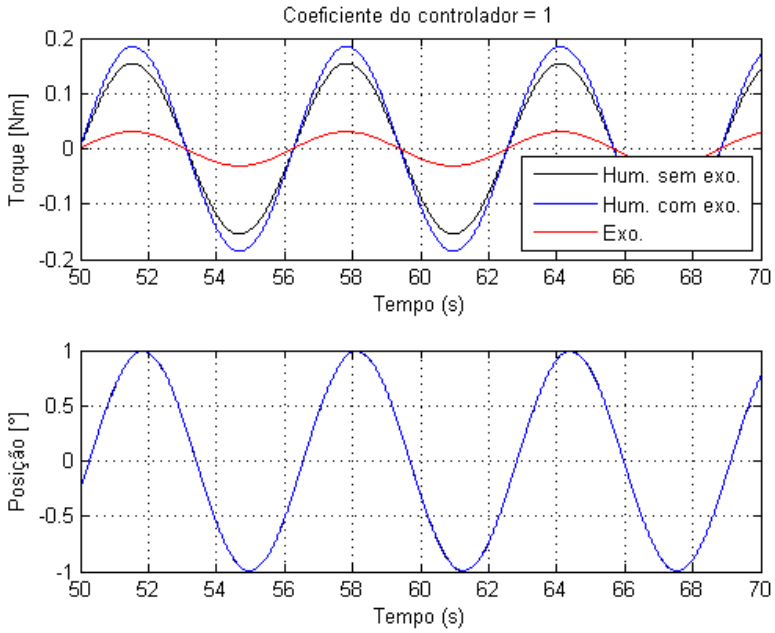
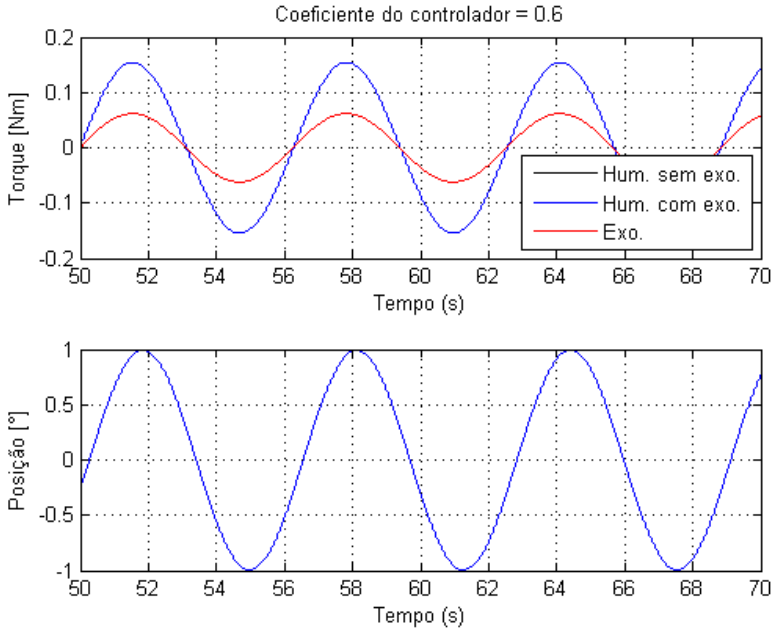


Figura 19: Resultado da simulação para fator dos parâmetros de impedância desejada $c = 0.6$.



Com isso, pode-se concluir que a estratégia de controle escolhida realmente é capaz de realizar os modos de controle propostos baseando-se na

relação entre os parâmetros de impedância desejada e os parâmetros de impedância do usuário do exoesqueleto. As seções subsequentes descrevem o que foi feito para implementar essa estratégia de controle no sistema real.

5.2 Discretização do modelo do sistema

Para realizar a implementação da estratégia de controle de impedâncias no PC/104 é necessária a discretização do controlador acima apresentado. Diferentes métodos de discretização estão disponíveis na literatura. Neste trabalho, a escolha do método de discretização envolveu os seguintes critérios:

- deve ser possível automatizar o procedimento de discretização no programa em C que roda no PC/104;
- a estabilidade do sistema deve ser mantida após a discretização;
- os coeficientes da função de transferência discreta devem ser função dos parâmetros de impedância de tal forma que esses parâmetros físicos possam ser recuperados no procedimento de identificação apresentado na seção 5.3;

Até o presente momento duas técnicas de discretização foram implementadas. A adequação de cada uma dessas técnicas no presente trabalho continua sendo avaliada juntamente com o procedimento de identificação descrito na seção 5.3. Uma descrição desses métodos se encontra nos itens a seguir.

5.2.1 Discretização Bilinear

Também conhecido como método de discretização de Tustin ou trapezoidal, esse método de discretização consiste na substituição da variável s do

domínio da transformada de Laplace pela seguinte função de z:

$$s = \frac{2(z-1)}{T(z+1)}, \tag{5.9}$$

onde T representa o tempo de amostragem do sistema discreto (KONIGORSKI, 2012). Esta relação entre os domínios s e z corresponde a um simplificação da relação exata dada por

$$z = e^{sT} \tag{5.10}$$

e pode ser utilizada quando o tempo de amostragem T é suficientemente pequeno (KONIGORSKI, 2012). Aplicando-se essa transformação à função de transferência contínua do controle de impedâncias representada em (5.5), obtém-se a seguinte função de transferência no domínio z:

$$\frac{\theta_{\mathrm{d}}(z)}{\tau_{\mathrm{m}}(z)} = \frac{1 + 2z^{-1} + 1z^{-2}}{\left(\frac{4J}{T^2} + \frac{2B}{T} + K\right) + \left(-\frac{8J}{T^2} + 2K\right)z^{-1} + \left(\frac{4J}{T^2} - \frac{2B}{T} + K\right)z^{-2}}. \tag{5.11}$$

A sequência numérica correspondente pode ser escrita como

$$\theta_{\mathrm{d}}(k) = \frac{\tau_{\mathrm{m}}(k) + 2\tau_{\mathrm{m}}(k-1) + \tau_{\mathrm{m}}(k-2) - \left(-\frac{8J}{T^2} + 2K\right)\theta_{\mathrm{d}}(k-1) - \left(\frac{4J}{T^2} - \frac{2B}{T} + K\right)\theta_{\mathrm{d}}(k-2)}{\left(\frac{4J}{T^2} + \frac{2B}{T} + K\right)}. \tag{5.12}$$

Essa última equação mostra que os que o método de discretização de Tustin pode ser facilmente automatizado tendo estrutura e parâmetros definidos em função dos parâmetros de impedância da malha contínua. Para uma malha contínua estável, a transformação de Tustin assegura a estabilidade no domínio z (HARNEFORS, 2009). Desta forma, todos os requisitos definidos para o método de discretização são atendidos pela transformação de Tustin.

5.2.2 Método das diferenças - backward difference

Este método de discretização também consiste em uma aproximação da relação entre os domínios s e z, a qual é dada por

$$s = \frac{(1 - z^{-1})}{T}, \tag{5.13}$$

sendo T o tempo de amostragem. Neste método, contudo, a conversão entre o domínio s e a sequência numérica pode ser realizada diretamente através das seguintes equações:

$$\frac{\mathrm{d}\theta_\mathrm{d}(t)}{\mathrm{d}t} = \frac{\theta_\mathrm{d}[k] - \theta_\mathrm{d}[k-1]}{T}, \tag{5.14}$$

$$\frac{\mathrm{d}^2\theta_\mathrm{d}(t)}{\mathrm{d}t^2} = \frac{\theta_\mathrm{d}[k] - 2\theta_\mathrm{d}[k-1] + \theta_\mathrm{d}[k-2]}{T^2}. \tag{5.15}$$

Substituindo as equações (5.14) e (5.15) em (5.5), obtém-se a seguinte equação:

$$\tau_\mathrm{m}[k] = \left(\frac{J}{T^2} + \frac{B}{T} + K\right)\theta_\mathrm{d}[k] + \left(-\frac{2J}{T^2} - \frac{B}{T^2}\right)\theta_\mathrm{d}[k-1] + \left(\frac{J}{T^2}\right)\theta_\mathrm{d}[k-2]. \tag{5.16}$$

Da mesma forma que o método bilinear de discretização, este procedimento também atende a todos os requisitos apresentados no início da seção. Deve-se notar, porém, que o método das diferenças elimina a dependência do torque medido em relação aos valores anteriores do mesmo. Além disso, neste caso há um único coeficiente independente dos parâmetros de impedância, o que difere do resultado da discretização bilinear, a qual conta com quatro coeficientes independentes.

5.3 Identificação dos parâmetros de impedância

A aplicação da estratégia de controle definida em (ARAUJO; TANNURI; FORNER-CORDERO, 2012) apresentada na subseção 5.1.1 requer o conhecimento do parâmetros de impedância do usuário do exoesqueleto. Como estes parâmetros variam de uma pessoa para outra e, portanto, não apresentam um valor padrão, há a necessidade de realizar um procedimento para sua identificação. Além disso, a identificação dos parâmetros de impedância do humano, do exoesqueleto e/ou do conjunto pode ser também utilizada nos estudos de controle motor.

Como já explicado anteriormente o sistema formado por humano e exoesqueleto corresponde a dois sistemas de impedância em paralelo, de tal forma que, de acordo com (ARAUJO; TANNURI; FORNER-CORDERO, 2012), os parâmetros do conjunto são dados por

$$J = J_{\mathrm{h}} + J_{\mathrm{exo}}, \tag{5.17}$$

$$B = B_{\mathrm{h}} + B_{\mathrm{exo}}, \tag{5.18}$$

$$K = K_{\mathrm{h}} + K_{\mathrm{exo}}. \tag{5.19}$$

Considerando-se as equações acima, a seguinte procedimento foi adotado para a identificação dos parâmetros de impedância do humano:

1. Identificação dos parâmetros de impedância do exoesqueleto através da avaliação do comportamento do sistema em um teste definido de movimento sem a presença de usuário.

2. Identificação dos parâmetros de impedância do sistema humano-exoes-

queleto através da avaliação do comportamento do sistema através do mesmo teste de movimento utilizado na etapa anterior, desta vez com a presença de um usuário.

3. Cálculo dos parâmetros do usuário aplicando-se os resultados das etapas 1 e 2 em (5.17), (5.18) e (5.19).

5.3.1 Modelo para identificação do sistema

Para a identificação dos parâmetros de impedância o modelo ARX(*Auto-Regressive with eXogenous inputs*) foi escolhido. O modelo ARX é um dos modelos paramétricos lineares mais conhecidos na área de identificação, tendo sido utilizado em números trabalhos na identificação de parâmetros de impedância, como por exemplo em (OLAYA, 2008), (LOVE; BOOK, 1995), (EROL et al., 2005) e (EROL et al., 2006).

A estrutura do modelo ARX está representada na equação (5.20) (OLAYA, 2008),

$$\frac{Y(q)}{U(q)} = q^{-n_k}\frac{B(q)}{A(q)}, \tag{5.20}$$

onde $U(q)$ e $Y(q)$ são respectivamente os dados de entrada e saída e os polinômios $A(q)$ e $B(q)$ são dados por

$$A(q) = 1 + a_1 q^{-1} + ... + a_{n_a} q^{n_a} \tag{5.21}$$

e

$$B(q) = b_1 + b_2 q^{-1} + ... + b_{n_b} q^{-n_b+1}. \tag{5.22}$$

Substituindo-se os polinômios (5.21) e (5.22), a equação (5.20) pode ser reescrita na forma de diferença linear como

$$y[k] + a_1 y[k-1] + ... + a_{n_a} y[k-n_a] = b_1 u[k-n_k] + ... + b_{n_b} u[k-n_k-n_b+1]. \tag{5.23}$$

Neste trabalho, a identificação dos parâmetros de impedância segundo o modelo ARX foi realizada utilizando-se a *toolbox* de identificação do programa MATLAB. Dois modelos ARX diferentes foram definidos a partir das equações que relacionam posição angular e torque no domínio discreto apresentadas em (5.12) e (5.16). Para a utilização da ferramenta mencionada devem ser definidos o vetor de dados de entrada $u[k]$, o vetor de dados de saída $y[k]$, o número de polos do sistema n_{a}, o número de zeros do sistema dado por $n_{\mathrm{b}} - 1$ e o atraso entre entrada e saída n_{k}. Abaixo seguem algumas considerações sobre ambos os modelos.

5.3.1.1 Modelo obtido por discretização bilinear

Através do rearranjo da equação (5.12) e considerando-se os valores medidos das variáveis, a relação entre torque e posição angular deste modelo pode ser apresentada como

$$\tau_{\mathrm{m}}[k] + 2\tau_{\mathrm{m}}[k-1] + \tau_{\mathrm{m}}[k-2] = \left(\frac{4J}{T^2} + \frac{2B}{T} + K\right)\theta_{\mathrm{m}}[k] + \left(-\frac{8J}{T^2} + 2K\right)\theta_{\mathrm{m}}[k-1] + \left(\frac{4J}{T^2} - \frac{2B}{T} + K\right)\theta_{\mathrm{m}}[k-2]. \quad (5.24)$$

Como os coeficientes associados aos valores de torque já estão definidos, na identificação essas variáveis foram agrupadas como uma única variável

$$T[k] = \tau_{\mathrm{m}}[k] + 2\tau_{\mathrm{m}}[k-1] + \tau_{\mathrm{m}}[k-2], \quad (5.25)$$

a exemplo das soluções apresentadas em (LOVE; BOOK, 1995) e (EROL et al., 2006).

Definindo-se o vetor de entrada $u[k]$ como a variável que agrupa as contribuições de torque medido $T[k]$ e o vetor de saída $y[k]$ como os valores medidos de posição angular $\theta_{\mathrm{m}}[k]$, o modelo pode ser escrito segundo a estrutura

definida em (5.23) como

$$\theta_{\mathrm{m}}[k] + \frac{\frac{-8J}{T^2} + 2K}{\frac{4J}{T^2} + \frac{2B}{T} + K}\theta_{\mathrm{m}}[k-1] + \frac{\frac{4J}{T^2} - \frac{2B}{T} + K}{\frac{4J}{T^2} + \frac{2B}{T} + K}\theta_{\mathrm{m}}[k-2] = \frac{1}{\frac{4J}{T^2} + \frac{2B}{T} + K}T[k]. \quad (5.26)$$

Simplicando-se a notação dos coeficientes obtém-se

$$\theta_{\mathrm{m}}[k] + a_1\theta_{\mathrm{m}}[k-1] + a_2\theta_{\mathrm{m}}[k-2] = b_1 T[k]. \quad (5.27)$$

Da equação (5.27), tem-se $n_{\mathrm{a}} = 2$, $n_{\mathrm{b}} = 1$ e $n_{\mathrm{k}} = 0$ (sistema sem atraso). Com isso, todos os parâmetros necessários para a utilização da ferramenta de identificação do MATLAB estão definidas. Como o resultado da identificação fornece os valores dos coeficientes a_1, a_2 e b_1, os parâmetros de impedância devem ser obtidos em função destes. De (5.26) e (5.27), os parâmetros de impedância podem ser calculados através das seguintes equações:

$$J = \frac{1 - a_1 + a_2}{b_1} \cdot \frac{T^2}{16}, \quad (5.28)$$

$$B = \frac{1 - a_2}{b_1} \cdot \frac{T}{4}, \quad (5.29)$$

$$K = \frac{1 + a_1 + a_2}{4b_1}. \quad (5.30)$$

Uma observação deve ser feita quanto a escolha de se agrupar as variáveis de torque em uma única variável $T[k]$. Essa etapa foi realizada como uma forma de adequar os objetivo da análise, obtenção dos parâmetros físicos de impedância, com as limitações da ferramenta de identificação de modelos ARX utilizada. Nesta, a estrutura do modelo a ser identificado é definida através dos parâmetros n_{a}, n_{b} e n_{k}, e não foi encontrada uma forma de fixar alguns dos coeficientes da equação. Assim, ao se utilizar como entrada o vetor de torque $\tau_{\mathrm{m}}[k]$ no lugar de $T[k]$, a relação dos pesos relativos entre as componentes $\tau_{\mathrm{m}}[k]$, $\tau_{\mathrm{m}}[k-1]$ e $\tau_{\mathrm{m}}[k-2]$ não é mantida e, com isso, perde-se o

significado físico dos coeficientes identificados.

5.3.1.2 Modelo obtido por discretização segundo método das diferenças backward difference

Da mesma forma que o modelo anterior, o modelo obtido pelo método de discretização *backward difference* definido na equação (5.16) pode escrito no formato de um modelo ARX em função do torque e da posição angular medidos:

$$\theta_{\mathrm{m}}[k] + \frac{-\frac{2J}{T^2} - \frac{B}{T}}{\frac{J}{T^2} + \frac{B}{T} + K}\theta_{\mathrm{m}}[k-1] + \frac{\frac{J}{T^2}}{\frac{J}{T^2} + \frac{B}{T} + K}\theta_{\mathrm{m}}[k-2] = \frac{1}{\frac{J}{T^2} + \frac{B}{T} + K}\tau_{\mathrm{m}}[k]. \quad (5.31)$$

Com coeficientes reduzidos, (5.31) pode ser representada por

$$\theta_{\mathrm{m}}[k] + a_1\theta_{\mathrm{m}}[k-1] + a_2\theta_{\mathrm{m}}[k-2] = b_1\tau_{\mathrm{m}}[k]. \quad (5.32)$$

No caso deste modelo, não há o problema dos coeficientes fixos e o sinal de torque $\tau_{\mathrm{m}}[k]$ pode ser utilizado diretamente como entrada nos cálculos de identificação. O vetor de saída é novamente dado pela posição angular $\theta_{\mathrm{m}}[k]$. Os demais parâmetros também são os mesmos: $n_{\mathrm{a}} = 2$, $n_{\mathrm{b}} = 1$ e $n_{\mathrm{k}} = 0$. Finalmente, os parâmetros de impedância são associados aos coeficientes determinados na identificação através das seguintes equações:

$$J = \frac{a_2}{b_1} \cdot T^2, \quad (5.33)$$

$$B = \frac{-a_1}{b1} \cdot T - 2\frac{J}{T}, \quad (5.34)$$

$$K = \frac{1}{b_1} - \frac{J}{T^2} - \frac{B}{T}. \quad (5.35)$$

5.3.2 Protocolo de testes para captura de sinais

A estratégia definida no início da seção para determinação dos parâmetros de impedância do humano implica a dependência desses parâmetros em relação ao resultado de dois testes de identificação diferentes: um para a identificação dos parâmetros do conjunto e outro para a identificação dos parâmetros do exoesqueleto. Além disso, dois modelos diferentes devem ser avaliados e comparados quanto sua adequação na representação do sistema em estudo. Há, portanto, a necessidade de se definir um procedimento padrão para os testes de captura de sinais dos sensores para sua aplicação nos cálculos de identificação. Os seguintes itens definem procedimentos dos testes realizados:

- O exoesqueleto é posicionado na horizontal.
- No caso dos testes com o usuário, deve-se garantir o alinhamento do eixo do cotovelo do humano com o eixo de rotação do exoesqueleto.
- O teste consiste no movimento do exoesqueleto seguindo uma trajetória senoidal com as seguintes características: período $T_{\text{sen}} = 8\ s$ e amplitude $A = 20°$;
- O teste é iniciado sempre a partir da mesma posição angular absoluta;
- Os sinais dos *strain gage* e do potenciômetro são amostrados com um período $T = 0.002\ s$ e salvos em um arquivo para posterior análise;

Optou-se por utilizar no procedimento de identificação os sinais medidos sem qualquer filtração. Ao longo das análises, o número de amostras foi avaliado e foi estabelecido como padrão de 16000 amostras por teste.

5.3.3 Conversão dos sinais de tensão em variáveis mecânicas

O *strain gage* e o potenciômetro fornecem as medidas de torque e posição angular em termos de tensão elétrica. Para que os parâmetros do modelo identificado tenham significado físico e permitam a determinação dos parâmetros de impedância, esses sinais foram convertidos nas variáveis correspondentes antes da sua utilização nos cálculos do procedimento de identificação. Os detalhes dessas conversões estão apresentados abaixo.

5.3.3.1 Conversão do sinal de tensão do potenciômetro em sinal de posição angular

A posição angular em graus do exoesqueleto $\theta[k]$, considerando-se a posição inicial padrão dos testes correspondente a $0°$, pode ser relacionada ao sinal de tensão em volts $V[k]$ medido pelo potenciômetro pela seguinte equação:

$$\theta[k] = aV[k] + b, \tag{5.36}$$

onde $a = -62,9774$ e $b = 164,6252$. Estes coeficientes foram determinados com dados experimentais resolvendo-se o seguinte sistema de equações:

$$\begin{aligned} 20 &= a\overline{V}_{20} + b, \\ -20 &= a\overline{V}_{-20} + b, \end{aligned} \tag{5.37}$$

onde $\overline{V}_{20}$ e $\overline{V}_{-20}$ correspondem às tensões médias de uma série de 1000 pontos da tensão medida nas posições $20°$ e $-20°$, respectivamente.

5.3.3.2 Conversão do sinal de tensão do strain gage em sinal de torque

O torque em $N.m$ $\tau\,[k]$ atuante sobre o braço do exoesqueleto relaciona-se ao sinal de tensão em volts $V\,[k]$ do *strain gage* pela seguinte equação:

$$\tau\,[k] = cV\,[k] + d. \tag{5.38}$$

onde os coeficientes $c = -14,4255$ e $d = -0,0830$ foram calculados através da resolução do seguinte sistema de equações:

$$\begin{aligned} \tau_1 &= c\overline{V}_1 + d, \\ \tau_2 &= c\overline{V}_2 + d, \end{aligned} \tag{5.39}$$

onde $\overline{V}_1$ e $\overline{V}_2$ correspondem às tensões médias de uma série de 1000 pontos da tensão medida pelo *strain gage* para os pesos de $1\ Kg$ e $1,5\ Kg$, respectivamente. Esses pesos foram fixados na extremidade do antebraço correspondente ao punho de tal forma que a distância massa ao eixo de rotação do antebraço mede $30,5\ cm$.

5.3.4 Critérios para avaliação dos resultados

Existe uma série de coeficientes para avaliação da qualidade do modelo identificado disponíveis na literatura. A *toolbox* de identificação do matlab conta com alguns deles, os quais estão descritos a seguir:

- Erro de predição final ou FPE(*Final Prediction Error*)

$$FPE = V\frac{(1 + d/N)}{1 - d/N}, \tag{5.40}$$

  onde d é o número de parâmetros estimados, N o número de amostras e V a função de perda. Quanto menor o valor de FPE, melhor é a adequação do modelo para predição.

- Erro médio quadrático ou MSE(*Mean Square Error*)

$$MSE = \frac{||y_{\text{calc}} - y_{\text{real}}||}{N-1} \tag{5.41}$$

Quanto mais próximo de zero o valor de MSE, melhor é a estimativa dos parâmetros do modelo.

- Porcentagem de ajuste ou Fit

$$Fit = 1 - \frac{||y_{\text{calc}} - y_{\text{real}}||}{||N - mean(y_{\text{real}}||} \tag{5.42}$$

Quanto maior a porcentagem de ajuste, melhor é o resultado da identificação.

Além desses critérios, uma poderosa ferramenta para análise de qualidade do modelo identificado é o GEE, do inglês *Generalized Equation Error*. Essa ferramenta é indicada para a avaliação de modelos de diferenças lineares (ISERMANN; MÜNCHHOF, 2010) e, portanto, é adequada para a análise dos modelos identificados neste trabalho. Considerando-se um sistema definido pela função de transferência

$$\frac{u[k]}{y[k]} = \frac{B[z]}{A[z]} \tag{5.43}$$

e um modelo estimado definido por

$$\frac{u[k]}{y_{\text{m}}[k]} = \frac{\hat{B}[z]}{\hat{A}[z]}, \tag{5.44}$$

onde y_{m} é a saída medida, considerando-se efeitos de distúrbio, o erro de equação generalizado pode ser calculado como

$$e[k] = y_{\text{m}}\hat{A}[z] - [k]\,u[k]\,\hat{B}[z]\,. \tag{5.45}$$

5.3.5 Resultados e análise

Aqui se encontram os resultados da identificação dos modelos ARX definidos na subseção 5.3.1, os quais estão subdivididos de acordo com o método de discretização correspondente.

5.3.5.1 Resultados da identificação do modelo obtido por discretização bilinear

Teste 1: Identificação dos parâmetros de impedância do exoesqueleto

- Modelo identificado:

$$\theta_{\mathrm{m}}[k] - 0,7036\theta_{\mathrm{m}}[k-1] - 0,2963\theta_{\mathrm{m}}[k-2] = -0.00005251T[k]. \quad (5.46)$$

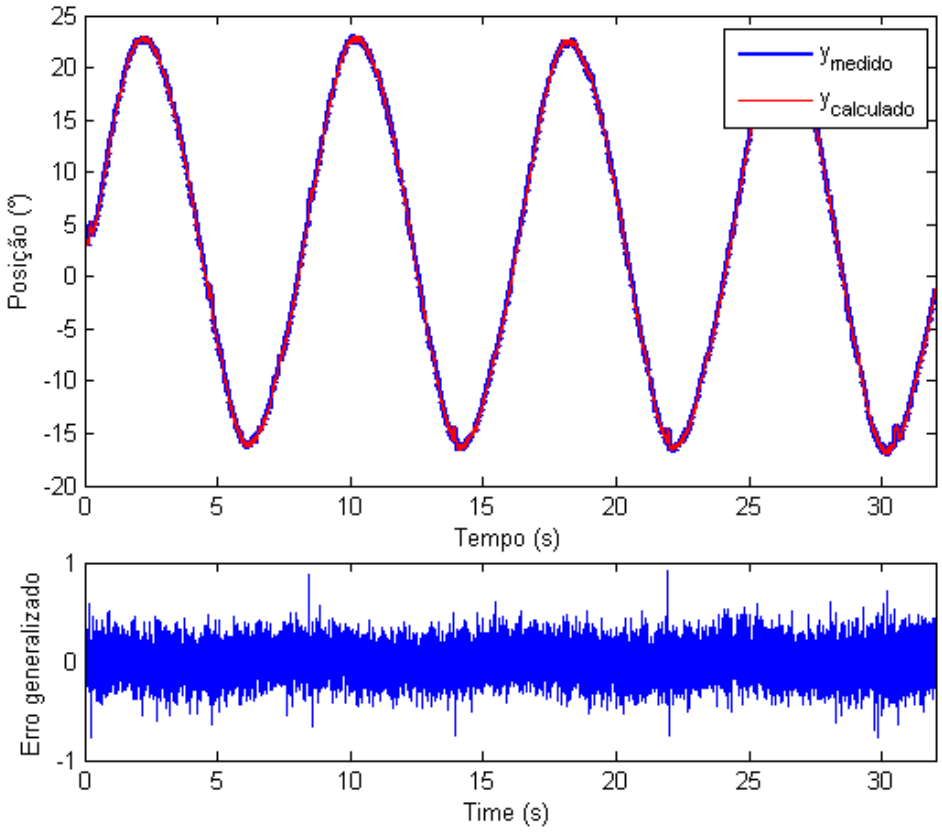
- Coeficientes de avaliação da qualidade do modelo

$$FPE = 0,02705$$
$$MSE = 0,02705$$
$$Fit = 98,78\%$$

- Parâmetros de impedância

$$\begin{aligned} J_{\mathrm{exo,bilinear}} &= -6,7 \times 10^{-3}\ Kg \cdot m^2 \\ B_{\mathrm{exo,bilinear}} &= -1,23 \times 10^{1}\ Kg \cdot m^2/s \\ K_{\mathrm{exo,bilinear}} &= -2,61 \times 10^{-1}\ N \cdot m \end{aligned} \quad (5.47)$$

Figura 20: Avaliação da qualidade do modelo estimado: modelo obtido por discretização bilinear, exoesqueleto sem usuário.



Teste 2: Identificação dos parâmetros de impedância do conjunto humano-exoesqueleto

- Modelo identificado:

$$\theta_{\mathrm{m}}[k] - 0,4629\theta_{\mathrm{m}}[k-1] - 0,5363\theta_{\mathrm{m}}[k-2] = 0,01663T[k]. \qquad (5.48)$$

- Coeficientes de avaliação da qualidade do modelo

$$FPE = 0,04198$$

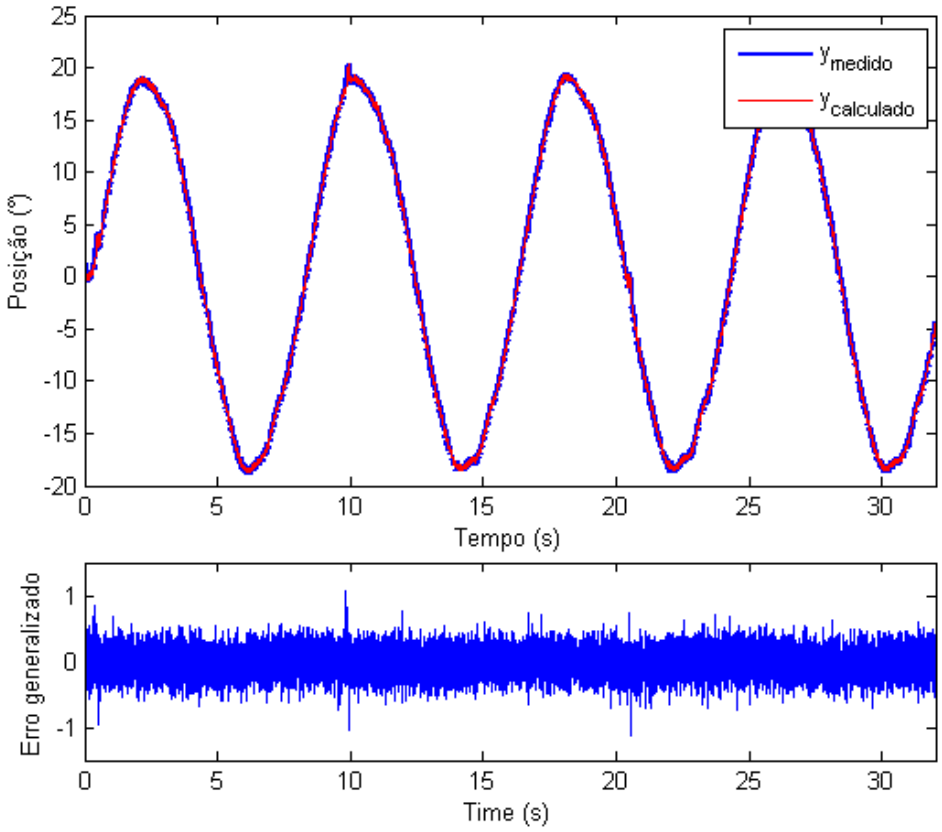
$$MSE = 0,04197$$

$$Fit = 98,44\%$$

- Parâmetros de impedância

$$J_{\text{conj,bilinear}} = 1,39 \times 10^{-5}\ Kg \cdot m^2$$
$$B_{\text{conj,bilinear}} = 4,62 \times 10^{-2}\ Kg \cdot m^2/s \qquad (5.49)$$
$$K_{\text{conj,bilinear}} = 1,20 \times 10^{-2}\ N \cdot m$$

Figura 21: Avaliação da qualidade do modelo estimado: modelo obtido por discretização bilinear, exoesqueleto com usuário.



Determinação dos parâmetros de impedância do humano.

Os parâmetros de impedância foram calculados substituindo-se os valores dos parâmetros de (5.47) e (5.49) em (5.17), (5.18) e (5.19). O resultado

encontra-se abaixo.

$$\begin{aligned} J_{\text{h,bilinear}} &= 6,7 \times 10^{-3}\ Kg \cdot m^2 \\ B_{\text{h,bilinear}} &= 1,24 \times 10^{1}\ Kg \cdot m^2/s \\ K_{\text{h,bilinear}} &= 2,73 \times 10^{-1}\ N \cdot m \end{aligned} \tag{5.50}$$

5.3.5.2 Resultados da identificação do modelo obtido por discretização segundo o método das diferenças (backward difference)

Teste 1: Identificação dos parâmetros de impedância do exoesqueleto

- Modelo identificado:

$$\theta_{\text{m}}[k] - 0,7036\theta_{\text{m}}[k-1] - 0,2964\theta_{\text{m}}[k-2] = -0,002811T[k]. \tag{5.51}$$

- Coeficientes de avaliação da qualidade do modelo

$$\begin{aligned} FPE &= 0,02705 \\ MSE &= 0,02704 \\ Fit &= 98,78\% \end{aligned}$$

- Parâmetros de impedância

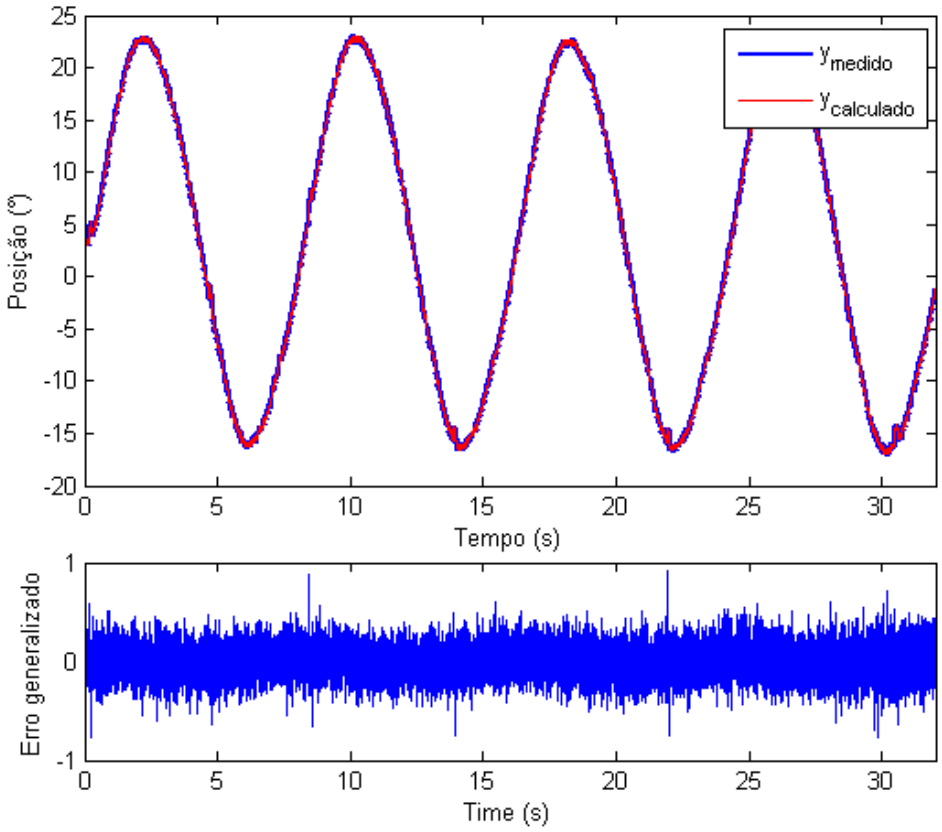
$$\begin{aligned} J_{\text{exo,dif}} &= 4,22 \times 10^{-4}\ Kg \cdot m^2 \\ B_{\text{exo,dif}} &= -7,89 \times 10^{-2}\ Kg \cdot m^2/s \\ K_{\text{exo,dif}} &= -4,22 \times 10^{2}\ N \cdot m \end{aligned} \tag{5.52}$$

Teste 2: Identificação dos parâmetros de impedância do conjunto humano-exoesqueleto

- Modelo identificado:

$$\theta_{\text{m}}[k] - 0,4626\theta_{\text{m}}[k-1] - 0,5366\theta_{\text{m}}[k-2] = 0,06516T[k]. \tag{5.53}$$

Figura 22: Avaliação da qualidade do modelo estimado: modelo obtido por discretização segundo método das diferenças, exoesqueleto sem usuário.



- Coeficientes de avaliação da qualidade do modelo

$$FPE = 0,042$$

$$MSE = 0,04199$$

$$Fit = 98,44\%$$

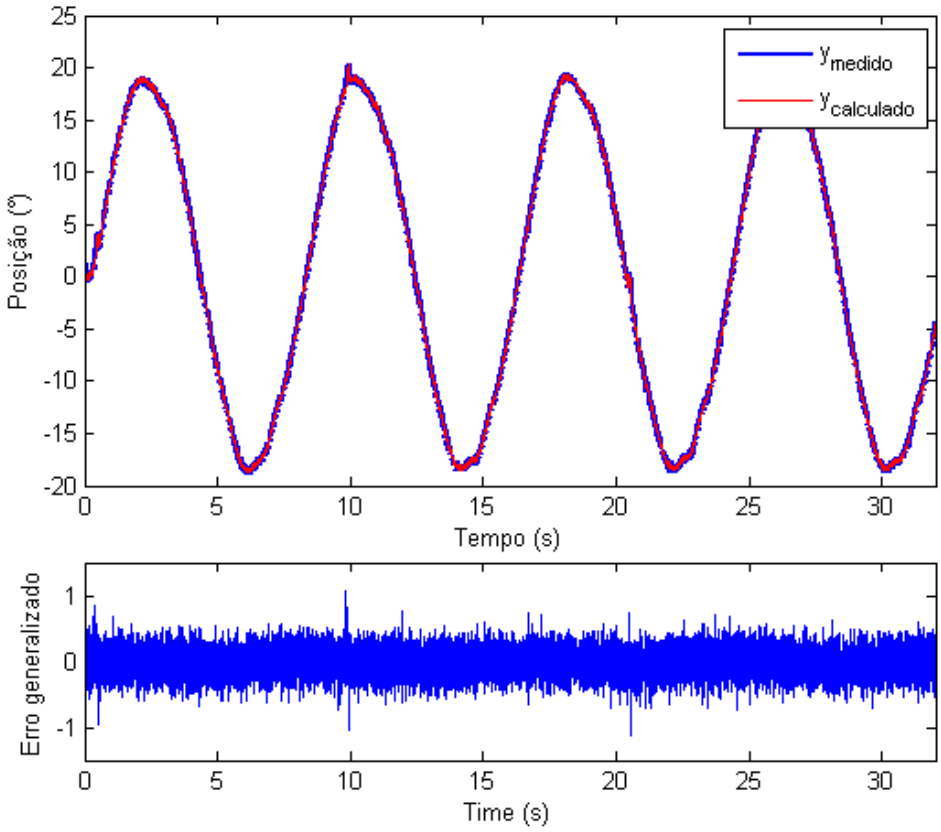
- Parâmetros de impedância

$$J_{\mathrm{h,dif}} = -3,29 \times 10^{-5}\ Kg \cdot m^2$$

$$B_{\mathrm{h,dif}} = 4,36 \times 10^{-1}\ Kg \cdot m^2/s \qquad (5.54)$$

$$K_{\mathrm{h,dif}} = -5,07 \times 10^{1}\ N \cdot m$$

Figura 23: Avaliação da qualidade do modelo estimado: modelo obtido por discretização segundo método das diferenças, exoesqueleto com usuário.



Determinação dos parâmetros de impedância do humano.

Os parâmetros de impedância foram calculados substituindo-se os valores dos parâmetros de (5.52) e (5.54) em (5.17), (5.18) e (5.19). O resultado encontra-se abaixo.

$$\begin{aligned} J_{\mathrm{h,dif}} &= -4,55 \times 10^{-5}\ Kg \cdot m^2 \\ B_{\mathrm{h,dif}} &= 5,15 \times 10^{-1}\ Kg \cdot m^2/s \\ K_{\mathrm{h,dif}} &= 3,71 \times 10^{2}\ N \cdot m \end{aligned} \tag{5.55}$$

5.3.5.3 Análise dos resultados

Os coeficientes FPE, MSE e Fit fornecem uma medida da qualidade do modelo estimado considerando-se o erro entre a saída medida e a saída estimada, as quais, nesta análise, correspondem à posição angular medida e estimada respectivamente. Nas quatro identificações realizadas os valores de FPE e MSE aproximam-se do zero, sendo menores que $0,005$; o valor de Fit também é muito semelhante entre os quatro testes e sempre maior que 98%. Isso indica que o erro de saída entre planta e modelo é, nos quatro casos, bastante baixo. Esse resultado pode ser também facilmente identificado observando-se o primeiro gráfico das figuras 20, 21, 22 e 23. Estas mostram adicionalmente o erro generalizado a cada amostra de dados de posição, o qual também é próximo de zero.

Esses resultados indicam que o modelo estimado aproxima bem o comportamento real do sistema em condições semelhantes às que caracterizaram os testes. O cálculo dos parâmetros de impedância do exoesqueleto e do sistema humano-exoesqueleto indicam, contudo, a presença de problemas na identificação. Com exceção do teste para identificação dos parâmetros do sistema humano-exoesqueleto calculados a partir do modelo obtido por discretização bilinear, nos demais testes um ou mais parâmetros de impedância calculados são negativos, incluindo valores de inércia. A impossibilidade física deste resultado mostra a presença de uma ou mais falhas no procedimento de identificação realizado. Assim, como os parâmetros de impedância do humano são calculados a partir dos parâmetros identificados, os valores apresentados em (5.50) e (5.55) foram considerados inválidos. Uma comparação concreta entre os dois modelos utilizados também ficou prejudicada.

5.3.6 Modificações no procedimento de identificação

Dados os resultados insatisfatórios apresentados na seção anterior surgiu a necessidade de revisão dos métodos, hipóteses e aproximações utilizadas na etapa anterior do projeto. O ponto de partida foi uma avaliação da adequação do procedimento utilizado para a identificação de um sistema em malha fechada. Neste contexto, procurou-se em um primeiro momento eliminar a influência das características dinâmicas do PID do *driver* do motor. Para isso, foram alterados os parâmetros desse controlador de tal forma que ele apresentasse o comportamento de um controlador proporcional com ganho unitário. Essa modificação provocou, contudo, a geração de picos de corrente acima dos valores tolerados pelos equipamentos utilizados. Essa estratégia foi, portanto, abandonada e procurou-se focar em métodos de processamento de sinais voltados especificamente para sistemas em malha fechada. Para tanto, utilizou-se como referência o trabalho de Urban Forssell (FORSSELL, 1999), o qual engloba diferentes tópicos voltados para identificação de sistemas em malha fechada.

5.3.6.1 Processamento de sinais

Optou-se pelo processamento dos sinais medidos antes dos cálculos de identificação, tentando, assim, minimizar a influência do ruído no resultado final. Neste contexto, foi utilizada uma estratégia voltada para procedimentos de identificação que utilizam sinais periódicos de entrada. Dois processamentos foram implementados e analisados: utilização dos sinais médios de entrada e saída e filtragem dos sinais. Para verificar a correta implementação dos métodos utilizou-se um modelo exemplo definido em (FORSSELL; GUSTAFSSON; MCKELVEY, 1998).

O primeiro processamento, como o próprio nome já diz, consiste no cálculo da média de cada um dos sinais de entrada e saída, considerando-se os diversos períodos dos sinais medidos. Sendo u_{m} e y_{m} respectivamente os sinais medidos de entrada e saída, P a quantidade de amostras em um período e M o número de períodos do sinal medido, os sinais médios de entrada e saída são dados por (FORSSELL; GUSTAFSSON; MCKELVEY, 1998):

$$\bar{u} = \frac{1}{M}\sum_{k=0}^{M-1} u_{\mathrm{m}}(t+kP), \quad t \in [1, P]\,, \tag{5.56}$$

$$\bar{y} = \frac{1}{M}\sum_{k=0}^{M-1} y_{\mathrm{m}}(t+kP), \quad t \in [1, P]\,. \tag{5.57}$$

Embora este procedimento seja bastante simples, quando M tende ao infinito o sinais médios obtidos de entrada e saída são livres de ruído (FORSSELL; GUSTAFSSON; MCKELVEY, 1998). Para valores menores de M a influência do ruído não é eliminada, mas minimizada. Nestes casos, pode-se melhorar a qualidade do sinal a ser utilizado na identificação através da aplicação da filtragem dos sinais médios (FORSSELL; GUSTAFSSON; MCKELVEY, 1998). Para isso foi utilizada uma ferramenta de Matlab chamada LTPDA Toolbox, na qual o filtro é determinado com base na amplitude do espectro de frequências do ruído.

A tabela 2 reúne os parâmetros do modelo utilizado na validação dos métodos e os resultados das identificações realizadas com os sinais sem processamento $(CasoA)$, com os sinais médios $(CasoB)$ e com os sinais médios filtrados $(CasoC)$.

Pode-se observar uma melhoria significativa entre o resultado dos parâmetros estimados com sinais sem processamento e aqueles estimados com os sinais médios. Os sinais filtrados apresentaram, contudo, um resultado pior que os sinais médios. Portanto, apenas o processamento dos sinais médios

Parâmetros	a_1	a_2	a_3	a_4
Valores reais	-1.5000	0.7000	1.0000	0.5000
Caso A	-0.9007	0.1221	0.8929	0.7922
Caso B	-1.4912	0.6929	0.9778	0.5049
Caso C	-1.4641	0.6744	0.9668	0.5759

Tabela 2: Comparação entre parâmetros reais do modelo exemplo e parâmetros identificados com sinais de diferentes características. Caso A – sem processamento, Caso B – sinais médios e Caso C – sinais médios filtrados.

foi utilizado na avaliação do sistema em estudo. Os resultados dessa análise para a estrutura do modelo resultante da discretização por *backward differences* encontram-se na tabela 3.

Parâmetros	a_1	a_2	b_1
Exo. - Padrão	-0.7036	-0.2964	-0.002811
Exo. - Sinal médio	-0.7427	-0.2573	-0.001725
Hum. com exo. - Padrão	-0.4626	-0.5366	-0.06516
Hum. com exo. - Sinal médio	-0.4823	-0.5169	-0.06782

Tabela 3: Comparação entre parâmetros identificados com sinais sem processamento e parâmetros identificados com sinais médios do exoesqueleto em vazio e do conjunto humano-exoesqueleto.

Pode-se notar que não houve mudança significativa nos parâmetros identificados com e sem processamento dos sinais, apontando que esta não é a principal causa do problema de identificação.

5.3.6.2 Consideração da não linearidade do sistema

Embora a literatura apresente exemplos de identificação de sistemas semelhantes ao deste trabalho utilizando uma abordagem linear, devido aos resultados insatisfatórios obtidos anteriormente, optou-se por uma reconsideração dessa estratégia. A solução encontrada foi aproximar o sistema com um modelo quasi-linear de segunda ordem, cujos parâmetros variam de acordo com o ponto de trabalho. Uma representação desse modelo pode ser encon-

trada na equação 5.58, onde ρ representa o ponto de trabalho.

$$\frac{\theta(s)}{\tau(s)} = \frac{1}{J(\rho)\, s^2 + B(\rho)\, s + K(\rho)}. \tag{5.58}$$

Numa aproximação inicial escolheu-se realizar múltiplas identificações, uma para cada região de trabalho. Para isso, foi estruturado um pequeno algoritmo que divide os sinais de entrada e saída em janelas de mesmo tamanho não sobrepostas. Os resultados mostraram uma pequena variação entre os parâmetros de cada janela, mas continuaram não solucionando os problemas de identificação apontados anteriormente.

5.3.7 Considerações finais sobre identificação dos parâmetros de impedância

A identificação de parâmetros de um sistema é um procedimento complexo influenciado por um grande conjunto de fatores, como por exemplo: relação sinal ruído; testes de excitação do sistema; sensibilidade dos sensores utilizados; qualidade dos modelos utilizados para representar o sistema; método de cálculo dos parâmetros, etc. Cada um desses fatores tem uma maior ou menor contribuição no resultado final da identificação, sendo, portanto, difícil identificar qual ou quais foram os pontos responsáveis pelos resultados insatisfatórios mostrados anteriormente. Suspeita-se, contudo, que a folga do cotovelo, o elevado nível de ruído dos sinais e a baixa sensibilidade da célula de carga tenham sido fatores decisivos neste projeto.

Apesar da grande diferença entre os parâmetros de impedância de diferentes indivíduos, a qual foi a motivação para a realização de um procedimento de identificação, devido à restrição de tempo para a execução do projeto como um todo optou-se por dar continuidade ao trabalho utilizando-se os parâmetros apresentados em (OLAYA, 2008).

5.3.8 Desempenho do controlador

Conforme já mencionado, devido aos problemas encontrados no procedimento de identificação, foram utilizados como parâmetros de impedância do usuário os mesmos parâmetros do ser humano utilizados na simulação da malha de controle. Esses parâmetros estão disponíveis em (OLAYA, 2008) e são dados por $J_{hum} = 2.0\ 10^{-3}\ Nms^2/^\circ$, $B_{hum} = 4.4\ 10^{-2}\ Nms/^\circ$ e $K_{hum} = 1.5\ 10^{-1}\ Nm/^\circ$. Para verificar a relação entre os parâmetros de impedância e os diferentes modos de controle, os parâmetros de impedância desejados foram definidos como a multiplicação dos parâmetros correspondentes do ser humano por um fator c, conforme apresentado em 5.6, 5.7 e 5.8.

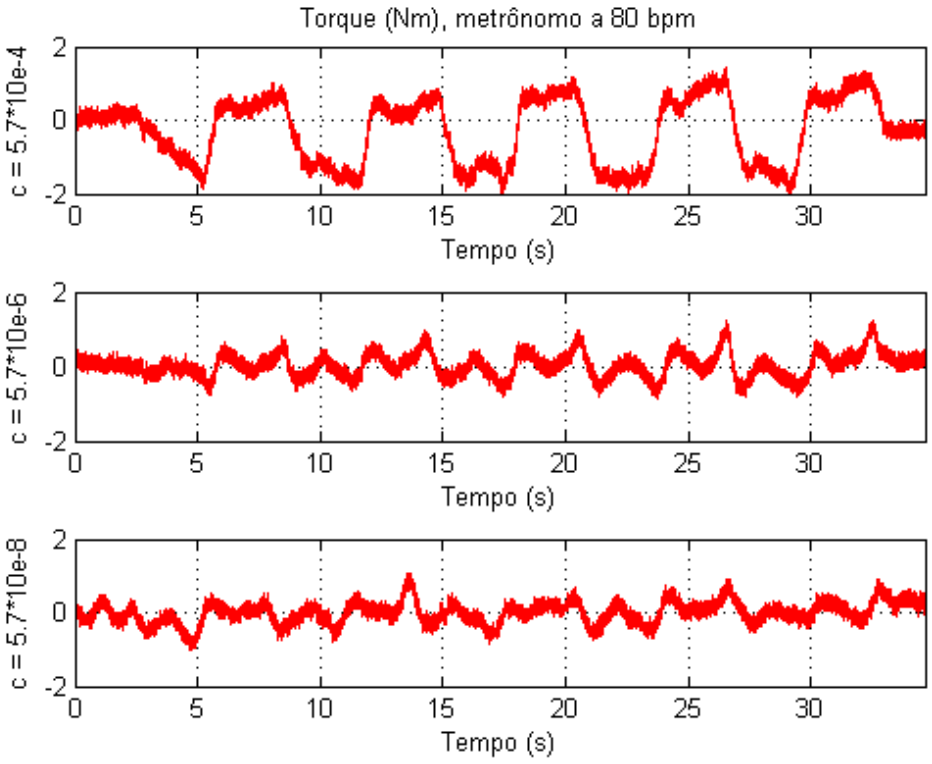
Inicialmente foi verificado o comportamento do sistema para $c = 1$. Nesta situação o exoesqueleto apresentou um comportamento rígido, de tal forma que não era possível movê-lo. Realizou-se então a determinação empírica de uma faixa de valores do fator c para a qual é possível mover o exoesqueleto e a estabilidade do sistema é assegurada. A faixa aproximada de valores do fator c encontrada através dos testes está representada em 5.59.

$$5.7\ 10^{-6} \leq c \leq 5.7\ 10^{-2} \tag{5.59}$$

O gráfico da figura 24 mostra a variação dos níveis de torque para diferentes valores do fator c. Os testes seguiram um procedimento semelhante aos dos testes de identificação, com a diferença de que as oscilações entre 20° e -20° foram limitadas a cinco ciclos e o movimento do usuário foi auxiliado por um metrônomo de 80 bpm. Dos gráficos, pode-se observar uma grande queda no torque do teste com $c = 5.7\ 10^{-4}$ para os testes com $c = 5.7\ 10^{-6}$ e $c = 5.7\ 10^{-8}$. Esse comportamento está de acordo com o esperado, uma vez que coeficientes menores significam uma menor impedância desejada e, portanto, uma menor resistência ao movimento do usuário. Por outro lado, não houve dife-

rença significativa no torque entre os testes para $c = 5.7\ 10^{-6}$ e $c = 5.7\ 10^{-8}$. Isso ocorre porque para $c = 5.7\,10^{-8}$ o sistema apresenta características de instabilidade. Para esse fator o sistema reage a qualquer leve estímulo aplicado sobre o braço do exoesqueleto e também às variações de sinal de torque e posição que ocorrem em função dos ruídos desses sinais. Desta forma, mesmo quando o usuário está com o braço em repouso o exoesqueleto tenta se movimentar. Portanto, para fins de análise de desempenho do controlador os valores abaixo de $c = 5.7\ 10^{-6}$ foram desconsiderados. Caso esse controlador seja utilizado futuramente em estudos de controle motor, esse limite inferior deve ser respeitado para assegurar a segurança do usuário do exoesqueleto.

Figura 24: Comparação entre os torques atuantes na célula de carga para diferentes valores do fator c dos parâmetros de impedância.



O limite superior da faixa de valores do fator c, por outro lado, é definido pela limitação da mobilidade do exoesqueleto para uma impedância desejada

muito elevada. Para $c = 5.7\ 10^{-2}$ o exoesqueleto se move, mas há uma limitação da velocidade com que os movimentos podem ser realizados. Com esse fator, por exemplo, já não foi possível executar o teste de movimento seguindo o referencial do metrônomo a 80 bpm; o limite foi uma taxa de 50 bpm. Esse resultado é coerente, uma vez que uma viscosidade elevada implica a geração de uma força resistente maior baseada na velocidade do movimento.

Esses resultados mostram que há uma relação entre torque e a variação nos parâmetros de impedância desejada definidos em função dos parâmetros de impedância do humano. Contudo, para verificar se houve realmente um auxílio ou oposição ao movimento do usuário uma verificação mais adequada é a análise da variação da atividade muscular do usuário entre os diversos testes. Esse resultado está disponível no capítulo 6.

5.4 Estrutura do software de controle de impedâncias

O software desenvolvido neste trabalho utiliza como estrutura base o software de controle de impedâncias em C desenvolvido no trabalho de (RATEIRO; CESAR, 2013), que fizeram a primeira implementação de um controlador em tempo real para o protótipo. Alguns trechos do código apresentado no apêndice C são, portanto, de autoria desses autores anteriormente citados.

O software é executado no computador embarcado PCM3362, e tem como objetivo primário empregar os dados lidos dos sensores do protótipo em conjunto com os parâmetros de impedância selecionados pelo usuário na execução de um controle de impedâncias. Uma função adicional é o armazenamento dos dados lidos pelos sensores, durante um intervalo desejado.

5.4.1 Ferramentas

O desenvolvimento e a execução do software de controle são feitos em um sistema operacional Linux Debian 32 bits, com a framework Xenomai e a biblioteca específica da placa de extensão de entradas e saídas Diamond--mm-16-at para o acesso às funções de leitura dos conversores analógico-digitais. O desenvolvimento do código é feito em uma máquina virtual, por meio do ambiente de desenvolvimento Eclipse Kepler CDT.

5.4.2 Estrutura do software

O funcionamento do software de controle baseia-se na configuração, inicialização e execução de tarefas de tempo real, cada qual sendo responsável por um conjunto de etapas do controle.

Quanto aos parâmetros de configuração das tarefas de tempo real, dois devem ser determinados previamente à sua ativação: a frequência de execução da função núcleo da tarefa e sua prioridade. Em sistemas de tempo real, o sistema busca manter a frequência de execução das tarefas o mais próximo possível do configurado. Para tal, caso haja a sobreposição de tarefas de prioridades diferentes, aquela com prioridade maior é executada antes, interrompendo-se a execução da de menor prioridade. O software permite ainda habilitar ou desabilitar determinadas tarefas, caso o usuário assim deseje.

O controle desenvolvido no trabalho de (RATEIRO; CESAR, 2013) empregava três tarefas de tempo real, cada uma implementada em um arquivo diferente:

- Sensor.c: Agrupa as funções referentes aos conversores analógico-digitais da placa, tais como calibração, leitura de valores de tensão dos sensores e a função núcleo da tarefa de tempo real. A função núcleo

é encarregada de fazer a leitura, processamento e armazenamento das leituras atuais dos sensores em variáveis globais, para posterior uso no controle.

- Control.c: Apresenta as funções de inicialização e execução do controle, sendo esta última a função núcleo da tarefa de controle.

- Actuator.c: Engloba as funções referentes aos conversores digital-analógicos, como a calibração, envio de setpoint arbitrário para o atuador e a função núcleo, que envia o *setpoint* calculado na etapa do controle.

Foi feita uma reestruturação do código, de forma a minimizar qualquer tipo de processamento na tarefa de sensoriamento, de forma a poder aumentar a frequência de amostragem do controle. Adicionalmente, em contraste com os sinais utilizados no controle anterior, optou-se por incluir uma conversão dos sinais de posição e torque para valores físicos (grau e Newtons.metro, respectivamente), na etapa de controle. Tal medida tem como finalidade permitir o uso dos parâmetros de inércia, viscosidade e elasticidade, devidamente discretizados, diretamente na configuração do controle discreto.

Outra correção em relação ao controle anterior é o armazenamento e uso das medidas de posição anteriores no controle, o que era feito utilizando se apenas a medida atual e os valores de setpoint calculados nas etapas anteriores.

Por fim, de forma a permitir a interação do controlador com a interface gráfica e, futuramente, com um controlador mestre que implemente o alto-nível do controle, foi criada uma quarta tarefa de tempo real, responsável pela comunicação do dispositivo por meio de uma conexão serial. Em Linux, a operação da porta serial é simples, dado que existem portas destinadas a comunicações por RS232 e RS422/485 no computador embarcado. A abertura da

porta para escrita e/ou leitura de dados é feita de forma similar à abertura de um arquivo, sendo que a real dificuldade recai sobre a montagem de mensagens a serem enviadas e à interpretação de comandos recebidos da máquina mestra. Dado que a taxa de envio de dados por meio da comunicação serial não acompanha a frequência com que as etapas de controle acontecem, o armazenamento dos valores lidos pelos sensores é uma funcionalidade extra implementada no software de controle, também nesta quarta tarefa de tempo real.

Mais detalhes em relação à comunicação são dados na Seção 7.3.

Os códigos-fonte podem ser vistos no Apêndice C.

6 ESTUDO DE SINAIS DE ELETROMIOGRAFIA

Os sinais elétricos gerados nas fibras musculares como resultado da requisição de um neurônio são chamados *potenciais de ação da unidade motora*, em inglês *motor unit action potential* (m.u.a.p.) (WINTER,). O sinal de eletromiografia ou sinais de EMG corresponde à soma algébrica de todos m.u.a.p.'s sendo transmitidos através das fibras musculares em um determinado ponto e instante tempo (WINTER,). Desta forma, os sinais de EMG são a manifestação elétrica da atividade neuromuscular associada à contração dos músculos.

Neste trabalho os sinais de eletromiografia apresentam dois usos distintos. O primeiro consiste na avaliação do desempenho do controlador desenvolvido através da análise das variações do nível de atividade muscular de acordo com o modo de controle selecionado. O segundo corresponde a uma análise da viabilidade do uso desses sinais no desenvolvimento de uma interface exo-esqueleto-humano capaz de realizar um controle de impedâncias a partir da intenção de movimento detectada pelos sinais EMG.

Esses sinais podem ser registrados tanto através de eletrodos de superfície, os quais são fixados sobre a pele através de adesivos, como através de eletrodos intramusculares. Neste trabalho a aquisição de sinais de EMG foi realizada com eletrodos de superfície, os quais são amplamente utilizados em estudos voltados para o controle motor humano.

6.1 Equipamento e protocolo de testes para a captura de sinais EMG

A aquisição dos sinais EMG foi realizada com os equipamentos da marca BTS Bioengineering. Os sinais EMG são medidos com eletrodos diferenciais de superfície, que são fixados sobre a pele do indivíduo nos pontos de interesse. Esses sinais são então amplificados com o auxílio de sondas, os quais enviam o sinal amplificado para uma central móvel de recebimento de dados de através de comunicação *wireless*. Este dispositivo comunica-se com um computador, também via *wi-fi*, onde o programa Myolab permite a visualização, o armazenamento e o processamento dos sinais medidos.

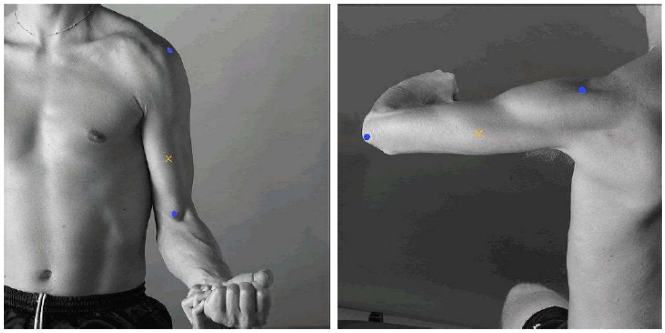
De acordo com (ABDUCH; RUIVO, 2012), os movimentos de flexão e extensão do braço humano em torno da junta do cotovelo envolvem dois grupos musculares distintos: o grupo primário, composto por bíceps braquial e tríceps braquial; e o grupo secundário, formado por trapézio inferior e eretor da espinha. Os músculos primários são os principais responsáveis pelo movimento, apresentando atividade muscular mais elevada. Por esse motivo, a análise feita neste capítulo baseou-se na atividade muscular desses dois músculos.

6.1.1 Instrumentação do usuário

O posicionamento dos eletrodos sobre a pele do usuário foi realizado seguindo as recomendações do projeto SENIAM (Surface ElectroMyography for the Non-Invasive Assessment of Muscles), o qual define o ponto ideal de localização dos eletrodos e apresenta um procedimento para sua localização (HERMENS et al., 2000). Os pontos correspondentes do bíceps braquial e do tríceps braquial estão indicados em amarelo na figura 25. Esses pontos correspondem aproximadamente ao ponto médio do músculo ao longo do seu

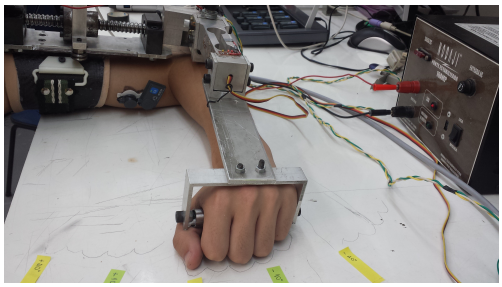
eixo.

Figura 25: Pontos de fixação ideais de eletrodos para captura sinais de EMG do bíceps braquial e do tríceps braquial. Fonte: (HERMENS et al., 2000)

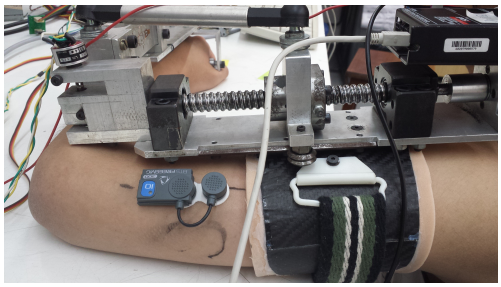


Antes da fixação dos eletrodos, a superfície da pele do indivíduo foi primeiramente higienizada com álcool. Os pontos de interesse foram localizados através da contração voluntária dos músculos sem a presença do exoesqueleto. Esse procedimento forneceu o posicionamento final do eletrodo a ser colocado sobre o tríceps braquial. Para bíceps, contudo, foi necessário ajustar a posição do eletrodo um pouco abaixo do ponto desejado para evitar que o eletrodo ficasse pressionado pelo fixador do exoesqueleto. A configuração final dos eletrodos está apresentada na figura 26.

Figura 26: Pontos de fixação dos eletrodos utilizados nos testes de captura sinais de EMG.



(a) Bíceps braquial.



(b) Tríceps braquial.

6.1.2 Descrição dos testes

Foram realizados dois tipos principais de testes:

1. Teste de máxima contração voluntária: o indivíduo é orientado a fazer a maior força possível contra uma força aplicada sobre seu braço pela pessoa que conduz o teste. A aplicação de forças pelo instrutor segue as mesmas orientações utilizadas para localizar os pontos de fixação dos eletrodos. Esse teste é realizado sem a presença do exoesqueleto.

2. Testes de movimento: o indivíduo permanece sentado com o braço apoiado sobre uma bancada plana de superfície lisa e é orientado a realizar movimentos oscilatórios repetitivos seguindo uma trajetória padrão. O braço e o cotovelo ficam em uma posição fixa e apenas o antebraço se move. O movimento é iniciado com o antebraço na posição $20°$, considerando-se como zero a posição para a qual o braço e o antebraço tem um ângulo de $90°$ e deslocamento positivo no sentido em que o antebraço se aproxima do corpo do usuário. Parte-se do repouso e são realizadas cinco oscilações completas entre $20°$ e $-20°$. O indivíduo é orientado a manter velocidade constante dentro do possível, sendo a trajetória guiada por marcações intermediárias em $10°$, $0°$ e $-10°$.

   De acordo com as análises a serem feitas os testes de movimento foram realizados sob diferentes condições, as quais foram definidas pelas seguintes variáveis:

   - Presença ou ausência do exoesqueleto;
   - Presença ou ausência de sinal sonoro de referência produzido por um metrônomo;
   - Parâmetros do controlador;

Nos testes de movimento sem exoesqueleto foram registrados apenas os sinais de EMG. Nos demais, além dos sinais de EMG capturados com auxílio do equipamento anteriormente descrito da BTS Bioengineering, também foram registradas medidas de posição e torque com o equipamento do exoesqueleto. O sincronismo no registro das medidas pelos dois sistemas (equipamento de EMG e equipamento do exoesqueleto) foi assegurado através da utilização de um sinal de *trigger*, o qual definia o início e o fim do registro dos sinais de EMG através de um sinal enviado pelo PC 104 no correspondente início e fim dos registros de posição e torque.

6.2 Análise dos resultados

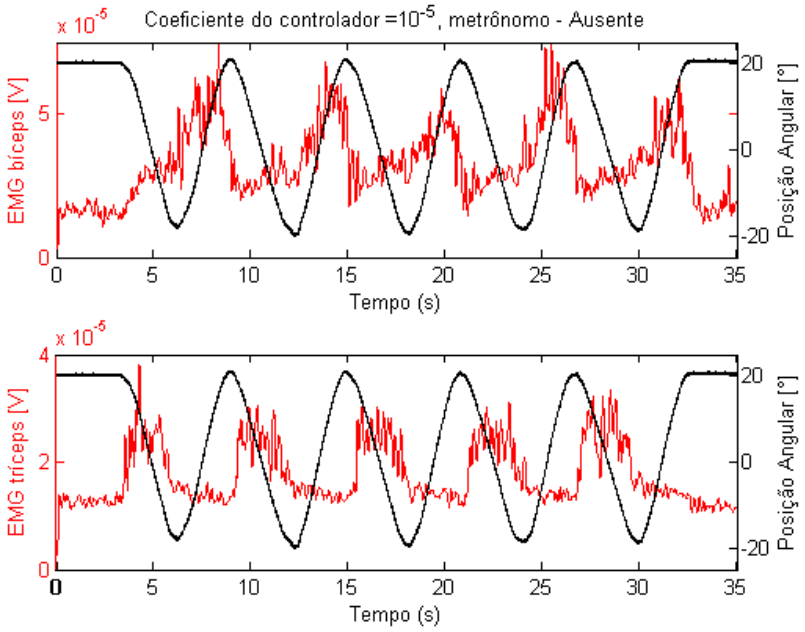
6.2.1 Relação entre sinais de eletromiografia e atividade muscular do indivíduo

O primeiro passo da análise é a verificação da relação entre os sinais EMG e o correspondente estado muscular do indivíduo. A figura 27 mostra os sinais EMG do bíceps braquial e do tríceps braquial de um dos indivíduos acompanhados das medidas de posição angular em um dos testes de movimento, com coeficiente dos parâmetros do controlador igual a $5.7\ 10^{-4}$ e metrônomo inativo. Conforme será detalhado na subseção 6.2.2, para esta configuração do controlador o exoesqueleto oferece uma resistência ao movimento do usuário e atividade muscular é maior, facilitando a visualização das variações dos sinais EMG.

Da literatura, sabe-se que o bíceps braquial é responsável pelo movimento de flexão do antebraço em torno do cotovelo, enquanto o tríceps é responsável pelo movimento oposto, o de extensão. No gráfico, os movimentos de extensão do antebraço em torno do cotovelo ocorrem de 20° para -20°, en-

quanto os movimentos de flexão estão ocorrem no sentido oposto. Os dados mostram uma clara relação entre os sinais de EMG e os movimentos de extensão e flexão do antebraço. Durante os movimentos de extensão, o sinal de EMG do tríceps braquial apresentou amplitude consideravelmente maior que a amplitude medida durante o repouso. Uma variação similar pode ser observada nos sinais de EMG do bíceps braquial durante os movimentos de flexão. Este comportamento mostra uma correspondência entre variações nos sinais de EMG e variações no estado de contração muscular do indivíduo.

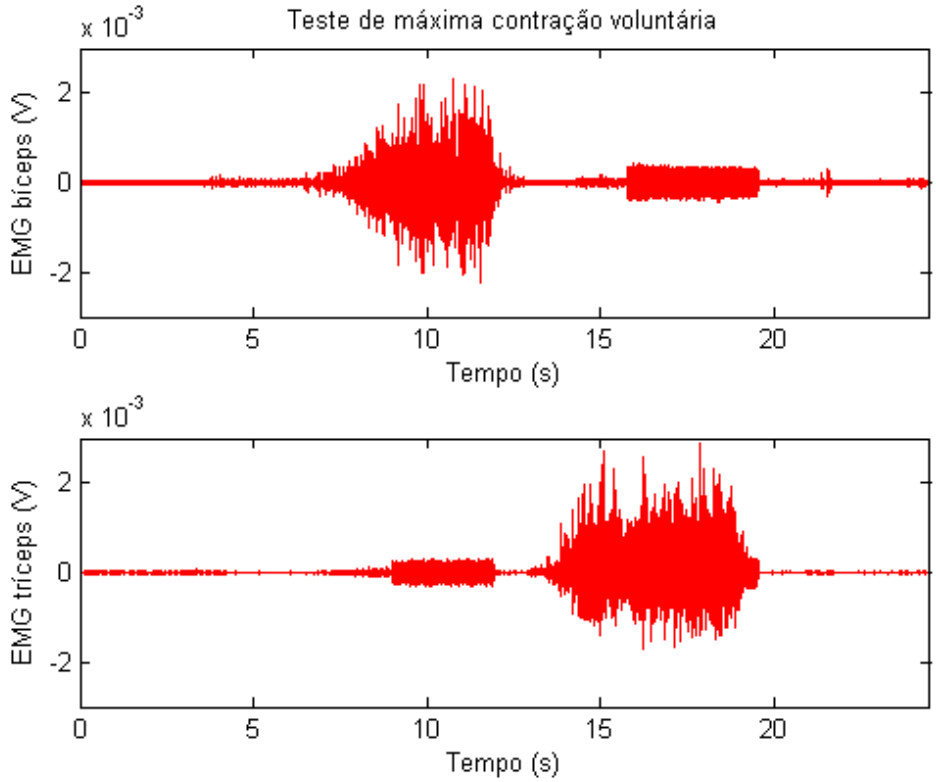
Figura 27: Sinais de eletromiografia do bíceps e do tríceps e correspondente trajetória de movimento.



Essa relação também pode ser verificada nos testes de máxima contração voluntária representados na figura 28, no qual uma contração máxima do bíceps braquial foi seguida por uma pequena pausa e, então, pela máxima contração do tríceps braquial. Neste teste, como o próprio nome já indica, o indivíduo é induzido a provocar a máxima contração de seus músculos. Deve-se notar que a amplitude dos sinais de EMG nesses casos (ordem de 10^{-3}) é consideravelmente mais elevada que a amplitude dos sinais do teste de

movimento padrão apresentado na figura 27 (ordem de 10^{-5}). Isso mostra a existência de uma relação entre o nível de contração muscular e a amplitude do sinal EMG.

Figura 28: Máxima contração voluntária do bíceps braquial e do tríceps braquial do indivíduo 1.



Esses resultados são coerentes com as informações sobre sinais de EMG encontradas na literatura e, portanto, a utilização dos sinais de EMG obtidos nos testes é válida nas análises subsequentes.

6.2.2 Avaliação do desempenho do controlador com sinais de EMG

A proposta da estratégia do controle de impedâncias desenvolvido neste trabalho é proporcionar diferentes modos de interação entre indivíduo e exoesqueleto de acordo com a variação dos parâmetros de impedância desejados

para o conjunto. Como os modos de controle diferem quanto a resistência imposta pelo exoesqueleto ao movimento do usuário, é esperado que diferentes modos de controle impliquem uma diferença entre os níveis de atividade muscular necessários para a execução de uma mesma tarefa. Nesta seção, essa relação é verificada com auxílio dos sinais de EMG.

As figuras 29 e 30 mostram, respectivamente, os sinais eletromiográficos processados do bíceps e do tríceps para testes de movimento com metrônomo ativo a 80 bpm em quatro situações diferentes:

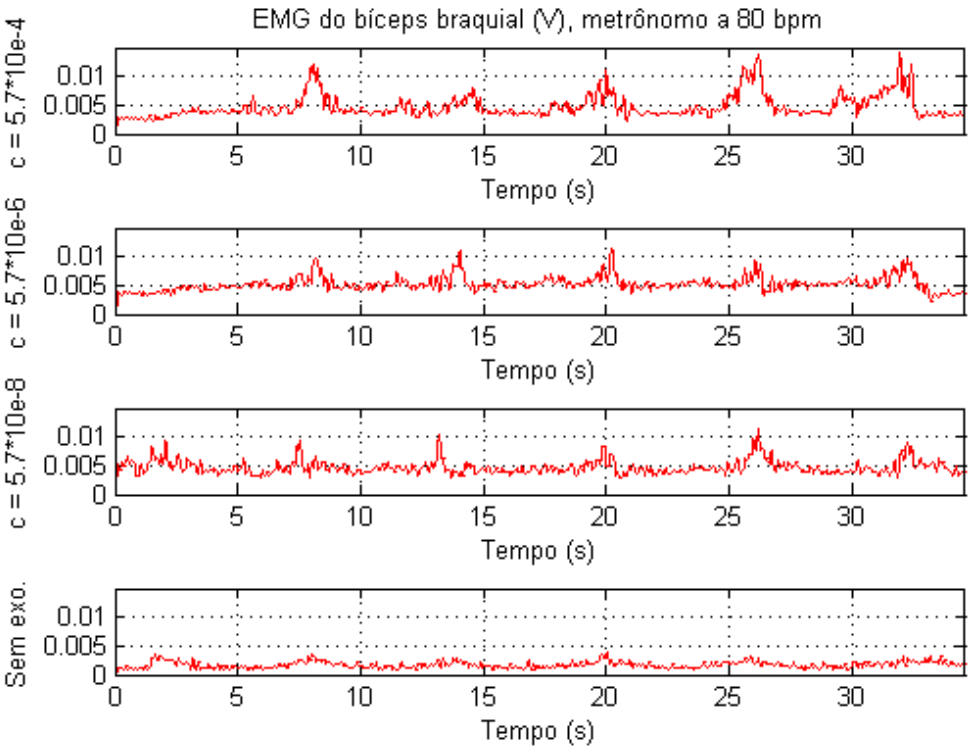
- Usuário com exoesqueleto, coeficiente $c = 5.7\ 10^{-4}$;
- Usuário com exoesqueleto, coeficiente $c = 5.7\ 10^{-6}$;
- Usuário com exoesqueleto, coeficiente $c = 5.7\ 10^{-8}$;
- Usuário sem exoesqueleto.

Esses testes foram realizados com metrônomo a fim de permitir uma comparação válida dos níveis de ativação muscular entre testes, considerando não apenas o deslocamento mas também a velocidade do braço do usuário.

O processamento dos sinais de EMG registrados nos testes seguiu os seguintes passos:

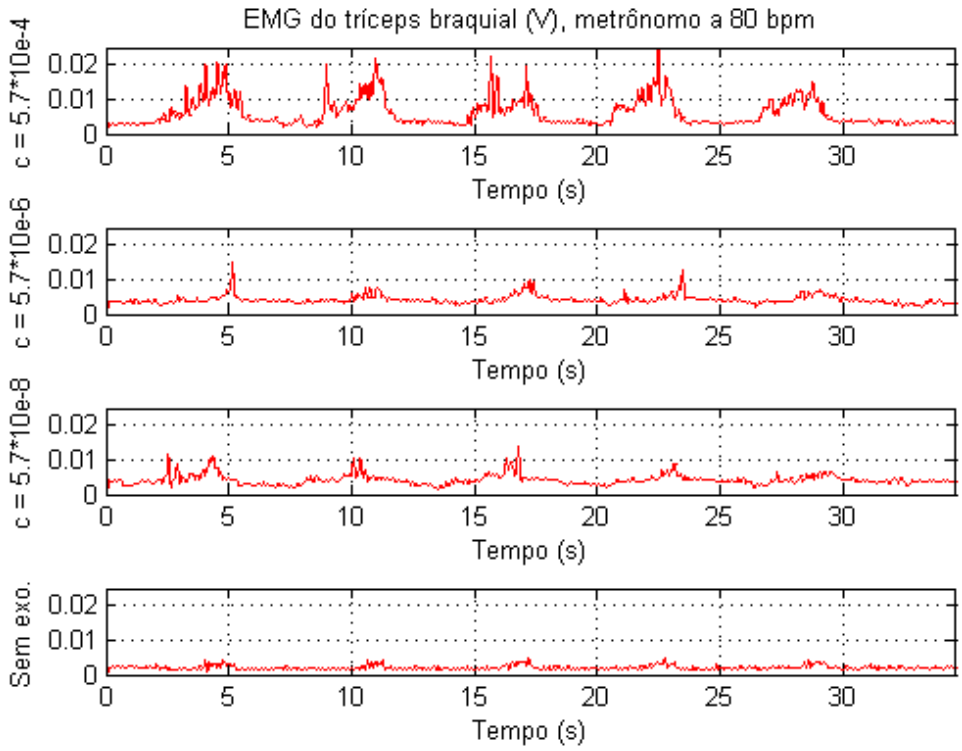
1. Retificação;
2. Normalização em relação ao pico do sinal de máxima contração voluntária (MCV) do músculo correspondente. Foi utilizado o pico do sinal MCV retificado.
3. Filtragem com filtro passa-baixas de quarta ordem do tipo *Butterworth*, com frequência de corte 5Hz.

Figura 29: Sinais de eletromiografia do bíceps braquial para usuário com exoesqueleto (diferentes coeficientes dos parâmetros de impedância desejada) e para usuário sem exoesqueleto.



Um primeiro ponto a ser observado é a diferença de *offset* entre os sinais capturados com o exoesqueleto, os quais para um mesmo músculo apresentam *offset* muito semelhantes entre si (valor próximo a $3.0\ 10^{-3}$), e o sinal capturado sem o exoesqueleto, cujo *offset* é menor que os demais (valor próximo a $1.3\ 10^{-3}$). Esse *offset* representa o nível de ativação muscular na condição de repouso e o fato constatado mostra que o simples ato de vestir o exoesqueleto aumenta a ativação muscular do usuário. Esse resultado é coerente uma vez que o peso do exoesqueleto e a pressão exercida pelo seu fixador em torno braço agem diretamente sobre os músculos avaliados. Além disso, parte dessa alteração no *offset* dos sinais resulta na interferência produzida pelo equipamento do exoesqueleto, em especial o motor, sobre os sinais capturados pelas sondas do equipamento de EMG.

Figura 30: Sinais de eletromiografia do tríceps braquial para usuário com exoesqueleto (diferentes coeficientes dos parâmetros de impedância desejada) e para usuário sem exoesqueleto.



Para auxiliar a análise, o nível de ativação muscular foi quantificado através do cálculo da média das áreas abaixo do envelope do sinal EMG de cada ciclo de ativação (cinco por teste). Como o sinal de EMG não é nulo no repouso e seu valor médio nesta condição é diferente para os casos usuário sem exoesqueleto e usuário com exoesqueleto, foi considerada a área entre o envelope do sinal de EMG e o valor médio do sinal de EMG para a situação de repouso correspondente às condições do teste. Esse cálculo foi realizado utilizando-se o sinal de EMG retificado, normalizado em relação ao pico do sinal de MVC e filtrado. O filtro aplicado foi um passa-baixas do tipo *Butterworth* com frequência de corte 30Hz. Essa frequência foi escolhida de tal forma que a determinação dos pontos de início e fim de cada ciclo de ativação muscular pudesse ser calculado com maior precisão. Além disso, o filtro

foi aplicado com auxílio da função *filtfilt* do Matlab de tal forma a se obter um filtro com deslocamento de fase zero. O cálculo da área foi aproximado pela regra dos trapézios através da função *trapz* do Matlab. O resultado encontra-se na tabela 4. Da tabela 4 nota-se que a atividade muscular do usuário

Condição do teste	Bíceps	Tríceps	% Bíceps	% Tríceps
$c = 5.7\ 10^{-4}$	18.41	26.02	538.05%	1076.46%
$c = 5.7\ 10^{-6}$	11.87	7.73	346.92%	319.77%
$c = 5.7\ 10^{-8}$	9.92	8.81	289.87%	364.52%
Sem exo.	3.42	2.94	100.00%	100.00%

Tabela 4: Média das das áreas entre o envelope do sinal EMG e o correspondente valor médio do sinal na condição de repouso. Testes de movimento com metrônomo a 80 bpm com cinco ciclos de ativação muscular.

para $c = 5.7\ 10^{-6}$ é consideravelmente menor que a atividade muscular para $c = 5.7\ 10^{-4}$. Esse resultado é coerente com o comportamento esperado para a estratégia de controle implementada, uma vez que um coeficiente menor implica uma impedância desejada menor, ou seja, menor resistência do exoesqueleto em relação ao movimento imposto pelo usuário. Deve-se observar, que essa diferença entre os níveis de ativação muscular é consideravelmente mais nítida para tríceps do que para o bíceps. Isso pode ser tanto resultado das características individuais do usuário, como também do posicionamento do eletrodo, o qual foi fixado com um certo deslocamento em relação à posição ideal para evitar o contato com o exoesqueleto.

A diferença no nível ativação muscular para $c = 5.7\ 10^{-6}$ e $c = 5.7\ 10^{-8}$, contudo, já não é tão clara. Para o bíceps a ativação muscular diminui de $c = 5.7\ 10^{-6}$ para $c = 5.7\ 10^{-8}$, mas os valores absolutos são relativamente próximos. Para o tríceps a ativação muscular nos dois casos também pode ser considerada semelhante, mas a variação é oposta a do bíceps: a ativação muscular aumenta de $c = 5.7\ 10^{-6}$ para $c = 5.7\ 10^{-8}$. Como já mencionado anteriormente, a manipulação do exoesqueleto para $c = 5.7\ 10^{-8}$ ao longo do

desenvolvimento do controlador mostrou que para esse coeficiente o controle do exoesqueleto está próximo da instabilidade. Desta forma, para esse coeficiente o controlador parcialmente auxilia o movimento do usuário e parcialmente impõe uma resistência ao mesmo de forma não padronizada ao longo do teste. Esse é o motivo pelo qual não houve uma grande variação na atividade muscular com a mudança de $c = 5.7\ 10^{-6}$ para $c = 5.7\ 10^{-8}$.

Finalmente, comparando-se os sinais de EMG para cada coeficiente c do controlador com os resultados do teste sem exoesqueleto é possível observar que em todos os casos a presença do exoesqueleto implicou uma maior atividade muscular do usuário. Como para valores do coeficiente c menores ou iguais $5.7\ 10^{-8}$ o sistema é instável, não é possível obter uma atividade muscular menor que a do indivíduo saudável sem exoesqueleto com a estratégia de controle implementada na configuração atual do sistema do exoesqueleto. Deve-se enfatizar, contudo, que foi verificada a existência de uma correspondência entre o coeficiente c dos parâmetros de impedância desejada e os níveis de ativação muscular do indivíduo, de tal forma que dentro dos limites de estabilidade do controlador a variação desse coeficiente produz as mudanças esperadas no comportamento do indivíduo: quanto menor o coeficiente c, menor a ativação muscular.

6.2.3 Relação entre sinal de EMG e torque

A relação entre os sinais de EMG e o torque exercido pelo músculo em questão é um objeto de estudo frequente em trabalhos na literatura (LENZI et al., 2012). Por meio dessa relação, torna-se possível desenvolver controladores que empregam sinais de EMG como sinais de controle, aproveitando o fato de que os sinais de EMG representam diretamente a intenção de movimento do usuário (LALITHARATNE et al., 2012). São utilizadas abordagens bastante

distintas. Em (LENZI et al., 2012) são feitos testes para avaliar a reação de 10 indivíduos ao auxílio de movimento fornecido por um exoesqueleto controlado por EMG, partindo do pressuposto que o sistema nervoso central é capaz de compensar a falta de precisão do controlador. Em (OLAYA, 2008), por sua vez, o sinal de EMG não é relacionado diretamente com o torque aplicado, e sim com os parâmetros de rigidez e viscosidade do sistema.

Sabe-se, no entanto, que os sinais de EMG são não-lineares, variantes no tempo, variantes por indivíduo, dependem do posicionamento dos eletrodos e têm baixa repetibilidade (KIGUCHI; HAYASHI, 2012). Esses fatores contribuem para aumentar a complexidade do estudo e da determinação da relação entre a ativação muscular e o comportamento dinâmico final do membro do humano.

Na figura 31 podemos visualizar os sinais de EMG (em vermelho) e do torque medido no exoesqueleto (em preto) para o teste com parâmetro $c = 5.7\ 10^{-4}$ e metrônomo a 80 bpm.

Nota-se que de fato há uma relação entre o sinal de EMG de cada músculo e o torque de interação medido pelo exoesqueleto. Os instantes em que há ativação do bíceps coincidem com os de torque elevado, e os do tríceps com o torque reduzido (em sentido contrário). A relação entre os sinais, no entanto, como afirmado anteriormente, não é clara.

Para o caso da figura 32, onde a rigidez do exoesqueleto é menor e, consequentemente, a ativação muscular é menos contrastante, a relação entre os dois sinais fica menos visível.

Por fim, uma dificuldade adicional quanto à implementação de um controle que utiliza o sinal de EMG é a aquisição e tratamento do sinal (o sinal não pode ser utilizado diretamente da forma como é adquirido) em tempo real sem

Figura 31: Sinais de EMG do bíceps e do tríceps e correspondente variação do torque aplicado sobre a célula de carga. Teste com coeficiente $c = 5.7\ 10^{-4}$ e metrônomo a 80 bpm.

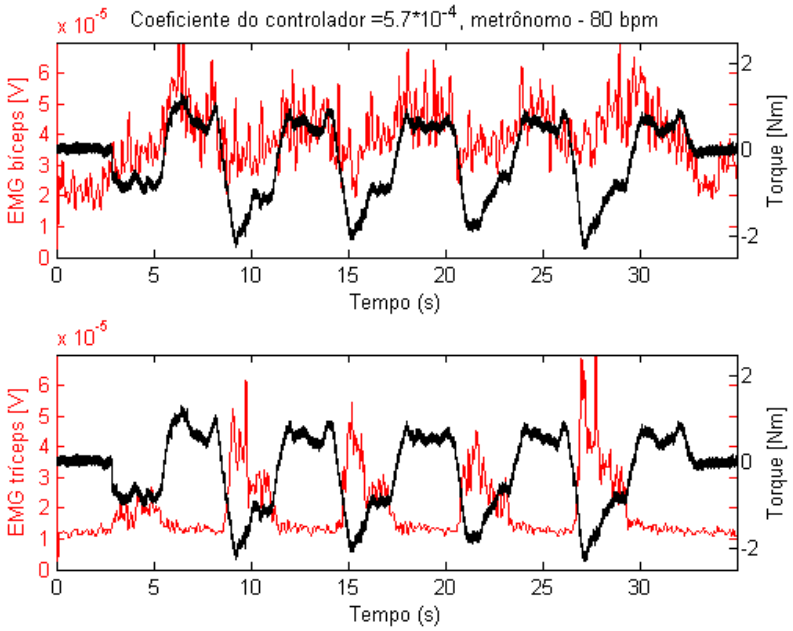
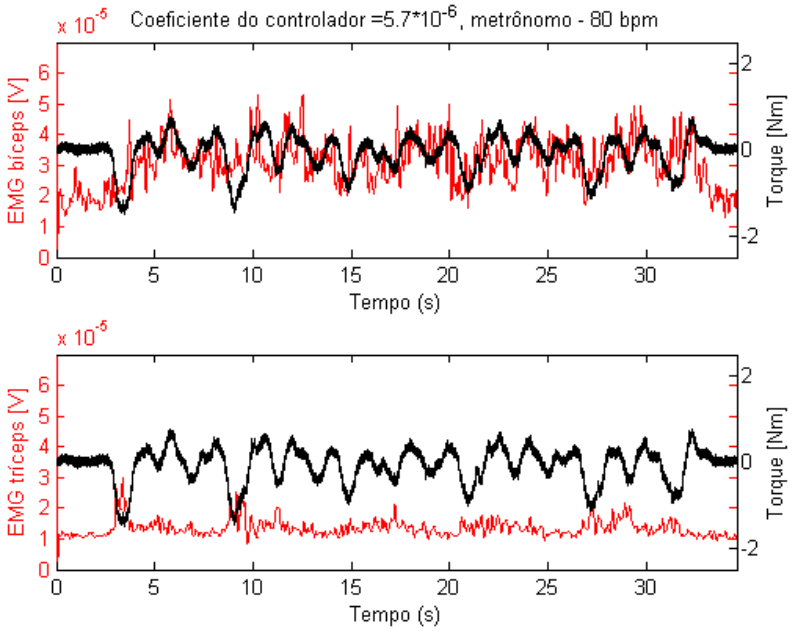


Figura 32: Sinais de EMG do bíceps e do tríceps e correspondente variação do torque aplicado sobre a célula de carga. Teste com coeficiente $c = 5.7\ 10^{-6}$ e metrônomo a 80 bpm.



comprometer a eficiência do controlador.

7 INTERFACE GRÁFICA PARA O USUÁRIO

Com a finalidade de facilitar e tornar mais intuitivo o procedimento de configuração e uso do software de controle implementado no PCM3362, foi desenvolvida uma interface gráfica a ser executada em um computador, o qual em trabalhos futuros deve implementar o controle de alto nível, coordenando múltiplas articulações para uma atividade comum. Com a interface, é necessário apenas iniciar o programa de controle no computador embarcado, podendo o restante das operações ser feito por meio da interface. A interface gráfica também oferece a funcionalidade de plotagem de gráficos das leituras dos sensores do protótipo, imprimindo os pontos conforme os recebe do controlador.

O código desenvolvido pode ser visto em sua íntegra no Apêndice D.

7.1 Ferramentas de desenvolvimento

A programação da interface gráfica foi feita na linguagem C++ em ambiente Windows 7 x64, utilizando o ambiente de desenvolvimento Qt Creator, que facilita o posicionamento de elementos da interface no layout e a programação de suas funções.

O Qt Creator é fortemente ligado à framework Qt, também utilizada no desenvolvimento desse programa. O Qt apresenta uma vasta gama de ele-

Tabela 5: Versão das ferramentas utilizadas

Ferramenta	Versão
Qt Creator	3.1.0
Qt	5.2.1
Qwt	1.6.0

mentos de interface de fácil implementação, e também permite que os desenvolvedores criem suas próprias variações utilizando-os como base.

Por fim, também foi utilizada a biblioteca Qwt, que contém componentes e classes de C++ voltadas para programas técnicos, tais como gráficos para plotagem.

As versões de cada programa utilizado seguem na Tabela 5

Todas as ferramentas acima citadas foram compiladas com Mingw (GCC) 4.8 32 bits.

7.2 Layout da interface

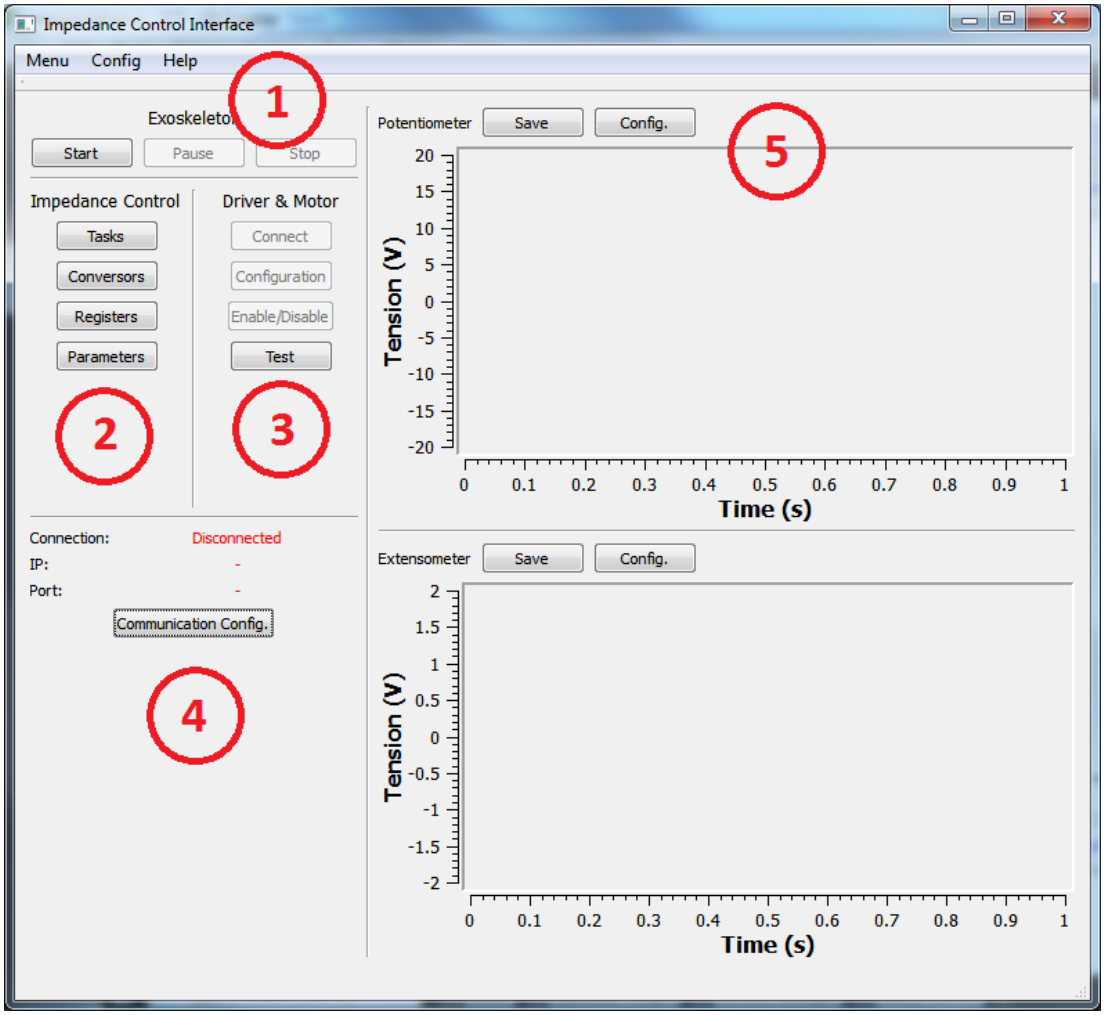
A janela principal da interface gráfica é exibida na Figura 33.

Nela, dividimos o Layout da interface em 5 partes principais, que consistem em:

1. Ações para a ativação, pausa ou parada do controle do exoesqueleto. Uma vez acionado o controle, o botão "‘Start"’ é desabilitado, e os botões "‘Pause"’ e "‘Stop"’ são habilitados. Quando "‘Pause"’ é pressionado, apenas a plotagem dos pontos é interrompida, mas o controle do exoesqueleto permanece em estado ativo. Por fim, o botão "‘Stop"’ difere do "‘Pause"’ por desativar o controlador.

2. Botões de configuração do controlador. São os botões que permitem ao usuário atribuir o comportamento desejado ao controle de impedâncias.

Figura 33: Janela principal da interface gráfica para o usuário



- É possível, através do botão "'Tasks"', configurar os parâmetros das tarefas de tempo real, indicando quais devem estar ativas, com que frequência devem ser chamadas e com que prioridade. Dado que a tarefa de comunicação é essencial para manter a troca de dados entre a interface e o controlador, o usuário não pode desabilitar essa tarefa. No entanto, lhe é permitido alterar seu período de chamada e prioridade.
- O botão "'Conversors"' abre uma janela de diálogo com dois botões, ambas para calibração dos conversores da placa PCM3362. Cada botão corresponde a um conversor, Analógico-digital ou Digital-ana-

lógico.

- O botão "‘Registers"’ permite configurar variáveis que selecionam a forma como o controlador se comporta, tais como o tipo de discretização escolhido, e se o tempo das tarefas é monitorado ou não.
- Clicando em "‘Parameters"’, abre-se uma janela para que o usuário selecione os parâmetros do controlador (inércia, elasticidade e viscosidade) desejados.

3. Os botões dessa seção não foram implementados até o momento da entrega desta monografia, sendo sugerido que próximos trabalhos se encarreguem desta tarefa. O objetivo planejado para esses botões é configurar o driver Epos2 que controla o atuador remotamente, minimizando a complexidade para ligar o controlador (diminuir a quantidade de softwares que deve ser utilizada). Apenas o último botão, "‘Test"’, encontra-se ativado, e é utilizado apenas para fins de testes de desenvolvimento da interface.

4. Configuração da comunicação utilizada. Apresenta duas opções: comunicação serial ou por UDP. A comunicação por UDP ainda não foi completamente implementada e, portanto, deve-se por ora utilizar apenas a serial. É possível configurar a taxa de Baud para a comunicação e a porta serial utilizada.

5. Áreas de plotagem. A área superior recebe medidas do potenciômetro e a inferior do extensômetro. É possível configurar a área de plotagem (limites inferior e superior dos eixos dos gráficos) e armazenar os dados recebidos do controlador em arquivo *.txt.

O último elemento relevante da janela principal é o menu superior "‘Config"’, por meio do qual se pode salvar a configuração atual do programa para

facilitar a configuração no próximo uso.

Inicialmente, ao se abrir o programa, todos os botões de configuração, ou que requerem enviar ou receber dados do controlador, aparecem desativados. Eles são ativados somente quando uma comunicação é configurada com sucesso.

7.3 Estrutura do programa

O funcionamento do programa de interface gráfica consiste, basicamente em duas linhas de comando (threads). A linha de comando principal é iniciada junto com o programa, e tem como função principal lidar com os eventos da interface. Ela também é responsável por plotar os pontos novos nas respectivas áreas de plotagem. Desta forma, em casos de sobrecarga de requisições sobre a interface (tais como redimensionamento ou movimento da janela dentro da tela do computador) podem haver falhas na continuidade dos gráficos plotados.

A thread secundária é iniciada pela primeira vez quando a comunicação é configurada (através do botão "‘Communication Config."’). Nos acionamentos consequentes desse botão, a thread secundária antiga é interrompida e apagada, e uma nova instância é criada. No Qt, a comunicação entre as threads é feita por um sistema de sinais e slots. Dada uma determinada situação, tal como a recepção de uma nova mensagem pela comunicação serial, pode-se emitir um sinal, que pode ou não carregar consigo uma variável de parâmetro. O sinal, por sua vez, pode ser conectado a um slot (uma função) de um outro objeto.

A thread secundária é responsável pela comunicação, incluindo a montagem das mensagens conforme o protocolo utilizado, envio, recepção e inter-

pretação das mensagens recebidas.

7.3.1 Comunicação serial

Para configurar a comunicação serial, é necessário definir alguns parâmetros, a começar pela ligação física dos componentes que devem interagir entre si.

O módulo de computador embarcado PCM3362 conta com três portas seriais, sendo duas RS232 e uma RS422/RS485. Para o computador que executa a interface gráfica, dada a dificuldade em se encontrar portas seriais com conector DB9 em notebooks atuais, é utilizado um conversor USB/Serial.

Outra definição importante, ainda referente às ligações, é o protocolo físico utilizado. Dada a necessidade de comunicação entre múltiplos dispositivos, considerando-se a modularidade do controle, deve ser utilizado o protocolo RS485, por permitir a ligação entre diversos dispositivos por um mesmo barramento.

Em seguida, em termos de padronização das mensagens trocadas, empregamos uma variação do Modbus ASCII, aplicando alterações para a conveniência do projeto:

Tabela 6: Protocolo de comunicação utilizado

Segmento	Modbus ASCII	Protocolo utilizado
Início de linha	':'	':'
Endereço	2 caracteres	1 caractere
Função	2 caracteres	2 caracteres
Tamanho dos dados	não tem	1 caractere
Dados	0 a 2 x 252 caracteres	0 a 247 caracteres
Checksum	LRC 2 caracteres	LRC 2 caracteres
Caracteres de fim de mensagem	0x0D e 0x0A	0x0D e 0x0A

Para fins de teste e desenvolvimento da interface, foi utilizada uma conexão por RS232 com modem nulo (Inversão do Rx e Tx) entre o computador

embarcado e o da interface.

8 CONCLUSÃO

Ao longo deste projeto foi seguida uma metodologia de pesquisa, partindo da revisão bibliográfica para a localizar o trabalho no contexto das pesquisas contemporâneas, passando pela definição dos requisitos de projeto e determinando a metodologia a ser seguida. Dentro dessa metodologia, tivemos a etapa de preparo, que consistiu na familiarização com os equipamentos existentes e o estudado dos conceitos teóricos do controle de impedâncias. Foi então feita a modelagem do sistema e selecionada a estratégia de controle a ser adotada. Na etapa seguinte, de identificação dos parâmetros do modelo do sistema, foram utilizadas diversas abordagens buscando-se os parâmetros essenciais para o desenvolvimento do controlador. Utilizando o dispositivo de eletromiografia, foi avaliado o comportamento do controlador para parâmetros de diferentes ordens de grandeza, utilizando como base valores encontrados na literatura.

Por meio dos resultados obtidos foi possível observar um comportamento coerente com o esperado. Idealmente, deveriam ser utilizados os parâmetros de impedância adequados para o usuário do exoesqueleto e para o sistema, o que forneceria resultados mais , o que não foi possível para o exoesqueleto em questão. Tanto os ruídos nos sinais de controle, bem como a excessiva folga na articulação controlada contribuíram para a complexidade do procedimento de identificação do sistema. O ruído ainda entra como fator limitador da sensibilidade atribuída ao controlador, levando o sistema à instabilidade

quando os parâmetros de impedância selecionados são muito reduzidos.

Apesar das complicações geradas em grande parte pelo sistema mecânico-eletrônico, foi feito um avanço considerável nessa linha de pesquisa, principalmente do ponto de vista do laboratório, uma vez que outros darão continuidade ao projeto. Em termos de aprendizado e aplicação de metodologias de pesquisa científica, este trabalho mostrou trazer consigo uma grande carga de conhecimento, que servirá de referência para projetos futuros.

Como sugestões para trabalhos futuros nessa mesma linha de pesquisa, listamos:

- Adição do sinal de eletromiografia à interface gráfica;
- Implementação de filtro para o sinal de EMG em tempo real, possivelmente seguindo para um controle baseado no sinal tratado;
- Reprojetar a articulação do exoesqueleto, minimizando a folga (backlash) existente no protótipo;
- Reprojetar o circuito do sensor de torque, buscando minimizar o ruído na leitura.

REFERÊNCIAS

ABDUCH, I. F.; RUIVO, J. a. P. P. *Controle de impedâncias para exoesqueleto robótico de membro superior para estudo de controle motor humano*. Tese (Doutorado) — Monografia, Trabalho de Conclusão de Curso em Engenharia Mecatrônica. Escola Politécnica da Universidade de São Paulo, 2012.

AGUIRRE-OLLINGER, G. *Active impedance control of a lower limb assistive exoskeleton*. Tese (Doutorado) — Northwestern University, Evaston, USA, 2009.

AGUIRRE-OLLINGER, G. et al. A 1-dof assistive exoskeleton with virtual negative damping: effects on the kinematic response of the lower limbs. In: *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*. [S.l.: s.n.], 2007. p. 1938–1944.

ARAUJO, A. *Desenvolvimento de controle de impedância aplicado a exoesqueleto biomecatrônico atuado por liga de memória de forma*. Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo, 2014.

ARAUJO, A.; TANNURI, E.; FORNER-CORDERO, A. Simulation of model-based impedance control applied to a biomechatronic exoskeleton with shape memory alloy actuators. In: *Biomedical Robotics and Biomechatronics (BioRob), 2012 4th IEEE RAS EMBS International Conference on*. [S.l.: s.n.], 2012. p. 1567–1572. ISSN 2155-1774.

AREVALO, J.; GARCIA, E. Impedance control for legged robots: An insight into the concepts involved. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, v. 42, n. 6, p. 1400–1411, Nov 2012. ISSN 1094-6977.

DOBRIANSKYJ, G. M.; COUTINHO, A. G. *Plataforma Robótica para Reabilitação do Membro Superior Humano*. Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo, 2013.

EROL, D. et al. A new control approach to robot assisted rehabilitation. In: *Rehabilitation Robotics, 2005. ICORR 2005. 9th International Conference on*. [S.l.: s.n.], 2005. p. 323–328.

EROL, D. et al. Autonomously adapting robotic assistance for rehabilitation therapy. In: *Biomedical Robotics and Biomechatronics, 2006. BioRob 2006. The First IEEE/RAS-EMBS International Conference on*. [S.l.: s.n.], 2006. p. 567–572.

FORNER-CORDERO, A. et al. Upper limb exoskeleton for motor control. *Proceedings of the 21th international congress of mechanical engineering, Natal, Brazil*, 2011.

FORSSELL, U. *Closed-loop Identification. Methods,Theory and Applications.* Tese (Doutorado) — Linköping Studies in Science and Technology. Dissertations No. 566., 1999.

FORSSELL, U.; GUSTAFSSON, F.; MCKELVEY, T. Time-domain identification of dynamic errors-in-variables systems using periodic excitation signals. In: *In Submitted to IFAC Congress*. [S.l.: s.n.], 1998. p. 6.

GUNASEKARA, J. et al. Control methodologies for upper limb exoskeleton robots. In: *System Integration (SII), 2012 IEEE/SICE International Symposium on*. [S.l.: s.n.], 2012. p. 19–24.

HARNEFORS, L. Implementation of resonant controllers and filters in fixed-point arithmetic. *Industrial Electronics, IEEE Transactions on*, v. 56, n. 4, p. 1273–1281, April 2009. ISSN 0278-0046.

HERMENS, H. J. et al. Development of recommendations for SEMG sensors and sensor placement procedures. *J Electromyogr Kinesiol*, v. 10, n. 5, p. 361–374, Oct 2000.

HOGAN, N. Stable execution of contact tasks using impedance control. In: *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*. [S.l.: s.n.], 1987. v. 4, p. 1047–1054.

HU, J. et al. Training strategies for a lower limb rehabilitation robot based on impedance control. In: *Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE*. [S.l.: s.n.], 2012. p. 6032–6035. ISSN 1557-170X.

ISERMANN, R.; MÜNCHHOF, M. *Identification of Dynamic Systems: An Introduction with Applications*. Springer, 2010. (Advanced Textbooks in Control and Signal Processing Series). ISBN 9783540788799. Disponível em: <http://books.google.com.br/books?id=tTwYA6ph9L0C>.

KIGUCHI, K.; HAYASHI, Y. An EMG-based control for an upper-limb power-assist exoskeleton robot. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, v. 42, n. 4, p. 1064–1071, Aug 2012. ISSN 1083-4419.

KONIGORSKI, U. *Digitale Regelungssysteme*. [S.l.]: Institut für Automatisierungstechnik und Mechatronik. Technische Universität Darmstadt. Skript zur Vorlesung, 2012.

LALITHARATNE, T. et al. A study on effects of muscle fatigue on EMG-based control for human upper-limb power-assist. In: *Information and Automation for Sustainability (ICIAfS), 2012 IEEE 6th International Conference on*. [S.l.: s.n.], 2012. p. 124–128.

LALITHARATNE, T. et al. Compensation of the effects of muscle fatigue on EMG-based control using fuzzy rules based scheme. In: *Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE*. [S.l.: s.n.], 2013. p. 6949–6952. ISSN 1557-170X.

LALITHARATNE, T. et al. Toward EEG control of upper limb power-assist exoskeletons: A preliminary study of decoding elbow joint velocities using EEG signals. In: *Micro-NanoMechatronics and Human Science (MHS), 2012 International Symposium on*. [S.l.: s.n.], 2012. p. 421–424.

LENZI, T. et al. Intention-based EMG control for powered exoskeletons. *Biomedical Engineering, IEEE Transactions on*, v. 59, n. 8, p. 2180–2190, Aug 2012. ISSN 0018-9294.

LOVE, L.; BOOK, W. Environment estimation for enhanced impedance control. In: *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*. [S.l.: s.n.], 1995. v. 2, p. 1854–1859 vol.2. ISSN 1050-4729.

MIHELJ, M.; NEF, T.; RIENER, R. Armin ii - 7 dof rehabilitation robot: mechanics and kinematics. In: *Robotics and Automation, 2007 IEEE International Conference on*. [S.l.: s.n.], 2007. p. 4120–4125. ISSN 1050-4729.

MIKULSKI, M. Electromyogram control algorithms for the upper limb single-dof powered exoskeleton. In: *Human System Interactions (HSI), 2011 4th International Conference on*. [S.l.: s.n.], 2011. p. 117–122. ISSN 2158-2246.

MIRANDA, A. et al. Bioinspired mechanical design of an upper limb exoskeleton for rehabilitation and motor control assessment. In: *Biomedical Robotics and Biomechatronics (BioRob), 2012 4th IEEE RAS EMBS International Conference on*. [S.l.: s.n.], 2012. p. 1776–1781. ISSN 2155-1774.

OLAYA, A. F. R. *Sistema robótico multimodal para análisis y estudios en biomecánica, movimiento humano y control neuromotor*. Tese (Doutorado) — Universidad Carlos III, 2008.

PERRY, J.; ROSEN, J.; BURNS, S. Upper-limb powered exoskeleton design. *Mechatronics, IEEE/ASME Transactions on*, v. 12, n. 4, p. 408–417, Aug 2007. ISSN 1083-4435.

RAMBABU, S. *Modeling and Control of a Brushless DC Motor*. Dissertação (Mestrado) — National Institute of Technology Rourkela, 2007.

RATEIRO, B. P.; CESAR, D. S. *Implementação de um software de controle de impedâncias para um exoesqueleto robótico*. Tese (Doutorado) — Monografia, Trabalho de Conclusão de Curso em Engenharia Mecatrônica. Escola Politécnica da Universidade de São Paulo, 2013.

RUIZ, A. et al. Exoskeletons for rehabilitation and motor control. In: *Biomedical Robotics and Biomechatronics, 2006. BioRob 2006. The First IEEE/RAS-EMBS International Conference on*. [S.l.: s.n.], 2006. p. 601–606.

SICILIANO, B.; SCIAVICCO, L.; VILLANI, L. *Robotics: Modelling, Planning and Control*. Springer, 2009. (Advanced Textbooks in Control and Signal Processing). ISBN 9781846286414. Disponível em: <http://books.google.com.br/books?id=jPCAFmE-logC>.

TAO, R. et al. Review of EMG-based neuromuscular modeling for the use of upper limb control. In: *Mechatronics and Machine Vision in Practice (M2VIP), 2012 19th International Conference*. [S.l.: s.n.], 2012. p. 375–380.

TSAI, B.-C. et al. An articulated rehabilitation robot for upper limb physiotherapy and training. In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. [S.l.: s.n.], 2010. p. 1470–1475. ISSN 2153-0858.

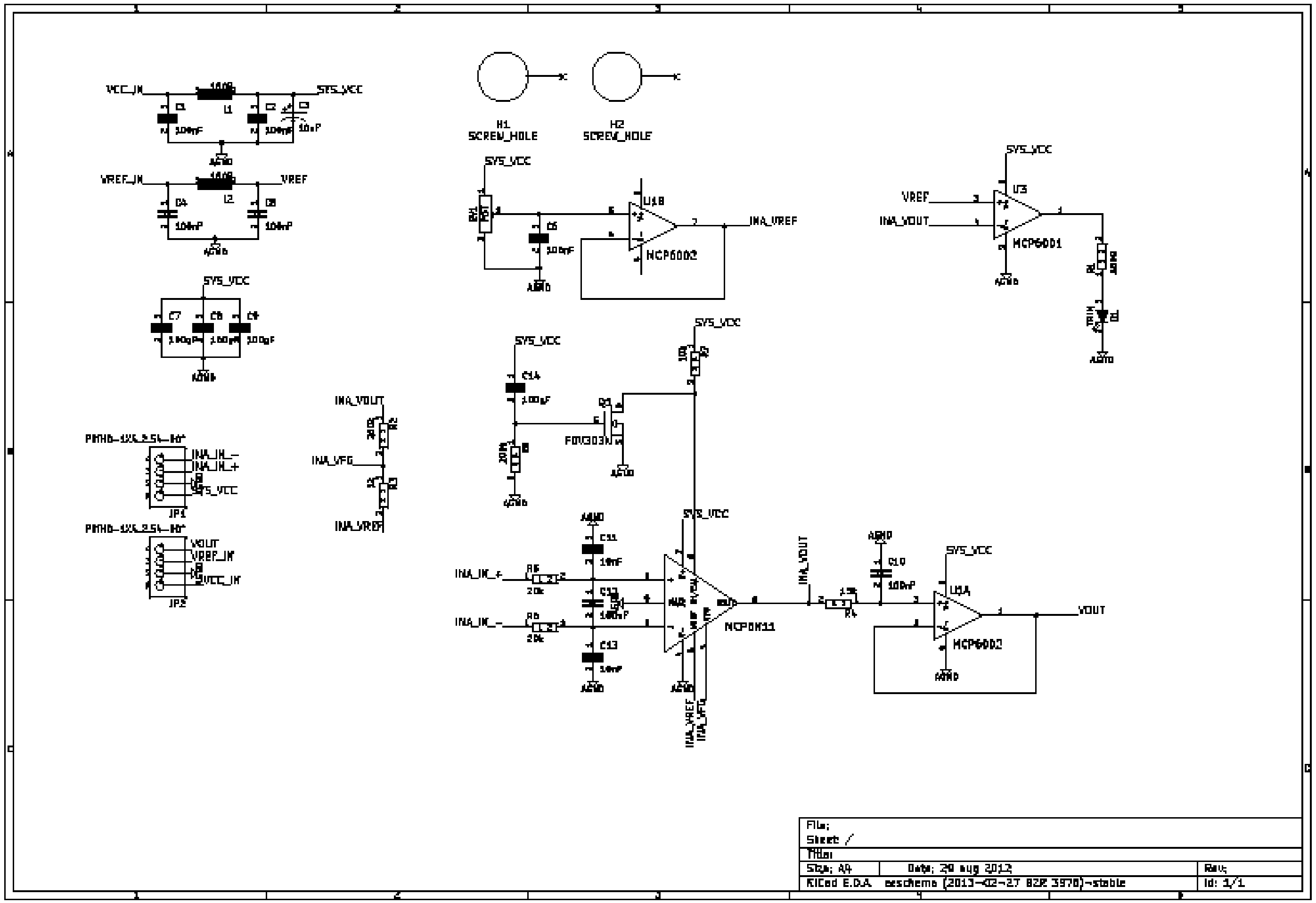
VITIELLO, N. et al. Neuroexos: A powered elbow exoskeleton for physical rehabilitation. *Robotics, IEEE Transactions on*, v. 29, n. 1, p. 220–235, Feb 2013. ISSN 1552-3098.

WINTER, D. A. *Biomechanics and Motor Control of Human Movement*. [S.l.]: University of Waterloo.

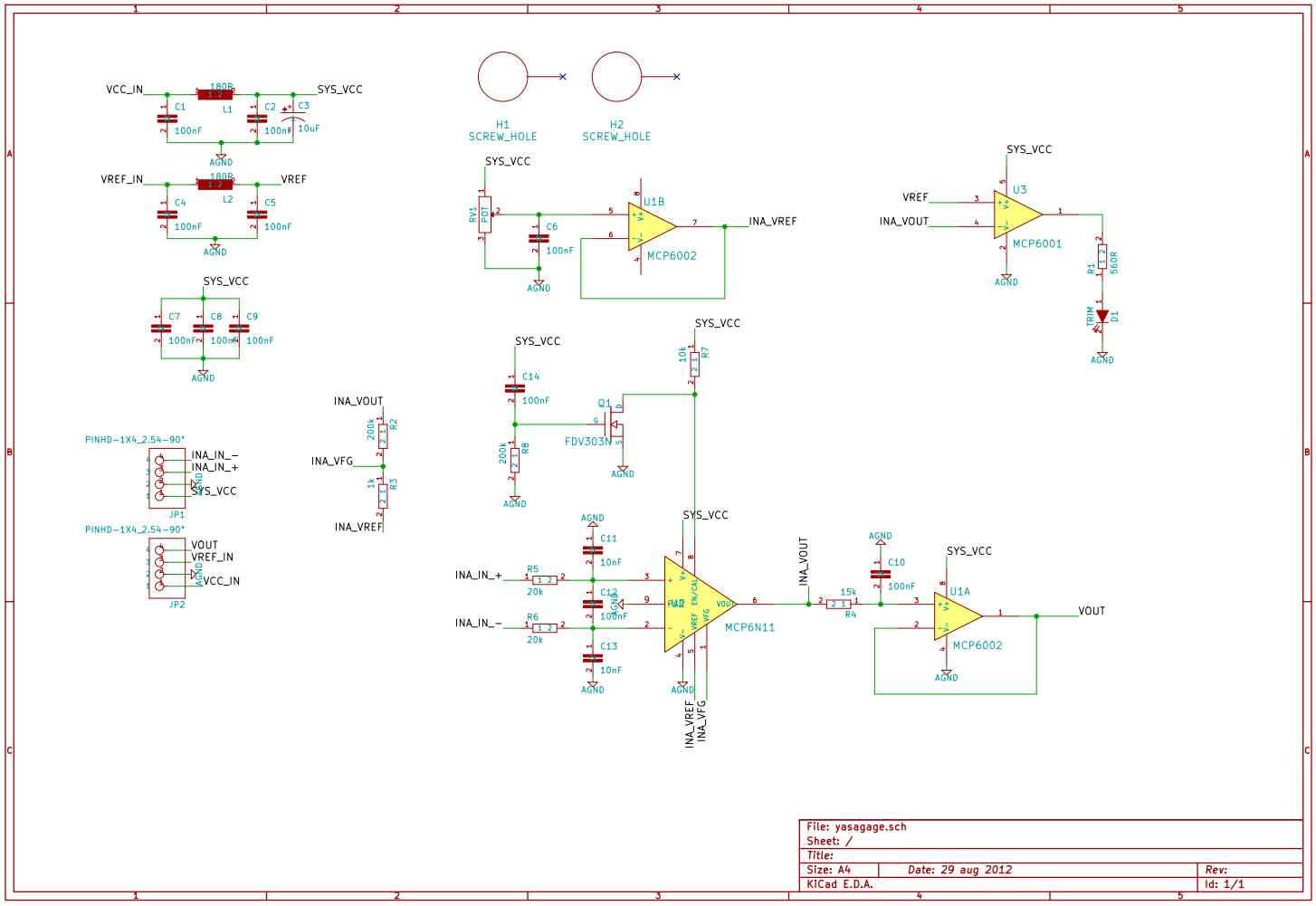
YASUTOMI, A. Y.; MIRANDA, A. B. W. *Exoesqueleto Robótico de membro superior para estudo de controle motor humano*. Tese (Doutorado) — Monografia, Trabalho de Conclusão de Curso em Engenharia Mecatrônica. Escola Politécnica da Universidade de São Paulo, 2011.

ZOSS, A.; KAZEROONI, H.; CHU, A. Biomechanical design of the Berkeley Lower Extremity Exoskeleton (BLEEX). *Mechatronics, IEEE/ASME Transactions on*, v. 11, n. 2, p. 128–138, April 2006. ISSN 1083-4435.

APÊNDICE A – PLACA DE CONDICIONAMENTO DE SINAIS 1


VCC_IN
SYS_VCC
VREF_IN
VREF
H1
SCREW_HOLE
H2
SCREW_HOLE
U1B
MCP6002
INA_VREF
U3
MCP6001
INA_VOUT
INA_VFG
FDV303N
INA_IN_+
INA_IN_-
MCP6N11
U1A
VOUT
AGND
JP1
JP2
VREF_IN
VCC_IN
File:
Sheet: /
Title:
Size: A4
Date: 29 aug 2012
Rev:
KiCad E.D.A. eeschema (2013-02-27 BZR 3970)-stable
Id: 1/1

APÊNDICE B – PLACA DE CONDICIONAMENTO DE SINAIS 2


VCC_IN
180R
L1
SYS_VCC
C1
100nF
C2
100nF
C3
10uF
AGND
VREF_IN
180R
L2
VREF
C4
100nF
C5
100nF
AGND
SYS_VCC
C7
100nF
C8
100nF
C9
100nF
AGND
PINHD-1X4_2.54-90°
INA_IN_−
INA_IN_+
SYS_VCC
AGND
JP1
PINHD-1X4_2.54-90°
VOUT
VREF_IN
AGND
VCC_IN
JP2
INA_VOUT
200k
R2
INA_VFG
1k
R3
INA_VREF
H1
SCREW_HOLE
H2
SCREW_HOLE
SYS_VCC
RV1
POT
U1B
C6
100nF
MCP6002
INA_VREF
AGND
SYS_VCC
VREF
INA_VOUT
U3
MCP6001
R1
560R
TRIM
D1
AGND
SYS_VCC
C14
100nF
200k
R8
AGND
Q1
FDV303N
10k
R7
SYS_VCC
AGND
C11
10nF
INA_IN_+
R5
20k
C12
10nF
INA_IN_−
R6
20k
C13
10nF
AGND
SYS_VCC
MCP6N11
INA_VOUT
INA_VREF
INA_VFG
AGND
C10
100nF
15k
R4
SYS_VCC
U1A
MCP6002
VOUT
AGND
File: yasagage.sch
Sheet: /
Title:
Size: A4
Date: 29 aug 2012
Rev:
KiCad E.D.A.
Id: 1/1

APÊNDICE C – CÓDIGOS DO CONTROLE

Software de controle em C utilizado na implementação do controle de impedâncias em tempo real do exoesqueleto. O código aqui apresentado contém trechos do código desenvolvido em (RATEIRO; CESAR, 2013).

main.c:

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

//Declarações
void initBoard();
void menu();
void config();
void task_config();
void calibration();
void set_task_priority();
void set_task_period();
void enable_task();
int waitForUser ();

int main(int argc, char* argv[])
{
        initBoard();
        MONITORING = 0;
        SENSORING = 0;
        TRAJECTORY = CONTROL_ACTIVE;    //1-SINUSOIDAL_POS 2-CONSTANT 3-SINUSOIDAL_TOR 4-CONTROL_ACTIVE
        DISCRETIZATION = BACKWARD_DIFF;
        printf("Digite o valor do multiplicador c: ");
        scanf ("%lf", &coef);
        T_Xenomai();
        T_Init();
        T_Create();
        send_setpoint(SET_CTE);
        printf("Calculando offsets...\n");
        sleep(5);
        S_CalcOffset();
        printf("Offsets calculados: \n  Torque: %5.8f \t Posição: %5.8f \n", potentiometerOffset,
              extensometerOffset);
        menu();
        T_Start();
        sleep(1);
        waitForUser();
        waitForUser();
        startSaving();
        waitForUser();
        finishSaving();
        pause();
        T_Kill();
        return 0;
}

void initBoard()
{
        dsccb.boardtype = DSC_DMM16AT;
        dsccb.base_address=0x300;               //endereço de base da placa
        dsccb.int_level = 7;                    //nível de interrupção
        dsccb.clock_freq = 10000000;

        //Inicializa a biblioteca dscud
        if( dscInit( DSC_VERSION ) != DE_NONE )
```

```
55          {
56                  dscGetLastError(&errorParams);
57                  rt_printf("dscInit error: %s %s\n", dscGetErrorString(errorParams.ErrCode), errorParams.errstring
                        );
58                  return;
59          }
60
61          //Inicializa a Placa
62          if(dscInitBoard(DSC_DMM16AT, &dsccb, &dscb)!= DE_NONE)
63          {
64                  dscGetLastError(&errorParams);
65                  rt_printf("dscInitBoard error: %s %s\n", dscGetErrorString(errorParams.ErrCode), errorParams.
                        errstring );
66                  return;
67          }
68  }
69
70  void menu()
71  {
72          int choice = 0;
73          while (1)
74          {
75                  printf("Controle de Impedâncias\n");
76                  printf("1. Configurações\n");
77                  printf("2. Executar\n\n");
78
79                  scanf ("%d",&choice);
80
81                  if(choice == 1)
82                          config();
83
84                  else if(choice == 2)
85                          break;
86
87                  else
88                          printf("Opção inválida!\n\n");
89          }
90  }
91
92  void config()
93  {
94          int choice = 0;
95          while (1)
96          {
97                  printf("Configurações\n");
98                  printf("1. Configurar Tasks\n");
99                  printf("2. Calibrar conversores\n");
100                 printf("3. MONITORING = %hd\n",MONITORING);
101                 printf("4. SENSORING = %hd\n",SENSORING);
102                 printf("5. TRAJECTORY = %hd\n", TRAJECTORY);
103                 printf("6. Voltar\n\n");
104                 scanf ("%d",&choice);
105                 if(choice == 1) task_config();
106                 else if(choice == 2)
107                         calibration();
108                 else if(choice == 3)
109                         scanf ("%hd",&MONITORING);
```

```
110                 else if(choice == 4)
111                         scanf ("%hd",&SENSORING);
112                 else if(choice == 5)
113                         scanf ("%hd",&TRAJECTORY);
114                 else if(choice == 6)
115                         break;
116                 else
117                         printf("Opção␣inválida!\n\n");
118         }
119 }
120
121 void task_config()
122 {
123         int choice = 0;
124         while (1)
125         {
126                 printf("Configurações␣das␣Tasks\n");
127                 printf("1.␣Habilitar/Desabilitar␣Task\n");
128                 printf("2.␣Configurar␣período␣das␣Tasks␣[ms]\n");
129                 printf("3.␣Configurar␣prioridade␣das␣Tasks␣[0-100]\n");
130                 printf("4.␣Voltar\n\n");
131                 scanf ("%d",&choice);
132                 if(choice == 1)
133                         enable_task();
134                 else if(choice == 2)
135                         set_task_period();
136                 else if(choice == 3)
137                         set_task_priority();
138                 else if(choice == 4)
139                         break;
140                 else
141                         printf("Opção␣inválida!\n\n");
142         }
143 }
144
145 void enable_task()
146 {
147         int choice = 0;
148         while (1)
149     {
150                 printf("Habilita/Desabilita␣Task\n");
151                 printf("1.␣ENABLETASK[SENSOR]␣␣␣=␣␣%d\n",ENABLETASK[SENSOR]);
152                 printf("2.␣ENABLETASK[CONTROL]␣␣=␣␣%d\n",ENABLETASK[CONTROL]);
153                 printf("3.␣ENABLETASK[ACTUATOR]␣=␣␣%d\n",ENABLETASK[ACTUATOR]);
154                 printf("4.␣ENABLETASK[TEST]␣␣␣␣␣=␣␣%d\n",ENABLETASK[TEST]);
155                 printf("5.␣Voltar\n\n");
156                 scanf ("%d",&choice);
157
158                 if(choice == 1)
159                         scanf ("%d",&ENABLETASK[SENSOR]);
160
161                 else if(choice == 2)
162                         scanf ("%d",&ENABLETASK[CONTROL]);
163
164                 else if(choice == 3)
165                         scanf ("%d",&ENABLETASK[ACTUATOR]);
166
```

```
167                 else if(choice == 4)
168                         scanf ("%d",&ENABLETASK[TEST]);
169
170                 else if(choice == 5)
171                         break;
172
173                 else
174                         printf("Opção inválida!\n\n");
175         }
176 }
177
178 void set_task_period()
179 {
180         int choice = 0;
181         while (1)
182         {
183                 printf("Período das Tasks [ms]\n");
184                 printf("1. PERIOD[SENSOR]    = %d\n",PERIOD[SENSOR]);
185                 printf("2. PERIOD[CONTROL]   = %d\n",PERIOD[CONTROL]);
186                 printf("3. PERIOD[ACTUATOR]  = %d\n",PERIOD[ACTUATOR]);
187                 printf("4. PERIOD[TEST]      = %d\n",PERIOD[TEST]);
188                 printf("5. Voltar\n\n");
189                 scanf ("%d",&choice);
190
191                 if(choice == 1)
192                         scanf ("%d",&PERIOD[SENSOR]);
193
194                 else if(choice == 2)
195                         scanf ("%d",&PERIOD[CONTROL]);
196
197                 else if(choice == 3)
198                         scanf ("%d",&PERIOD[ACTUATOR]);
199
200                 else if(choice == 4)
201                         scanf ("%d",&PERIOD[TEST]);
202
203                 else if(choice == 5)
204                         break;
205
206                 else
207                         printf("Opção inválida!\n\n");
208         }
209 }
210
211 void set_task_priority()
212 {
213         int choice = 0;
214         while (1)
215         {
216                 printf("Prioridade das Tasks [0-100]\n");
217                 printf("1. Prioridade do Sensor    = %d\n",PRIORITY[SENSOR]);
218                 printf("2. Prioridade do Controle  = %d\n",PRIORITY[CONTROL]);
219             printf("3. Prioridade do Atuador   = %d\n",PRIORITY[ACTUATOR]);
220                 printf("4. Prioridade do Teste     = %d\n",PRIORITY[TEST]);
221                 printf("5. Voltar\n\n");
222                 scanf ("%d",&choice);
223                 if(choice == 1)
```

```
224                         scanf ("%d",&PRIORITY[SENSOR]);
225                 else if(choice == 2)
226                         scanf ("%d",&PRIORITY[CONTROL]);
227                 else if(choice == 3)
228                         scanf ("%d",&PRIORITY[ACTUATOR]);
229                 else if(choice == 4)
230                         scanf ("%d",&PRIORITY[TEST]);
231                 else if(choice == 5)
232                         break;
233                 else
234                         printf("Opção inválida!\n\n");
235         }
236 }
237
238 void calibration()
239 {
240         int choice = 0;
241         while (1)
242         {
243                 printf("Calibração dos conversores\n");
244                 printf("1. Conversores Digital-Analógicos\n");
245                 printf("2. Conversores Analógico-Digitais\n");
246                 printf("3. Voltar\n\n");
247                 scanf ("%d",&choice);
248                 if(choice==1)
249                         A_Calibration();
250                 else if(choice==2)
251                         S_Calibration();
252                 else if(choice==3)
253                         break;
254                 else
255                         printf("Opção inválida!\n\n");
256         }
257 }
258
259 int waitForUser ()
260 {
261   int ch;
262   struct termios oldt, newt;
263
264   tcgetattr ( STDIN_FILENO, &oldt );
265   newt = oldt;
266   newt.c_lflag &= ~( ICANON | ECHO );
267   tcsetattr ( STDIN_FILENO, TCSANOW, &newt );
268   ch = getchar();
269   tcsetattr ( STDIN_FILENO, TCSANOW, &oldt );
270
271   return ch;
272 }
```

actuator.c:

```
1  #include "actuator.h"
2
3  int A_Init()
4  {
```

```
5         channel = 0;
6     return 0;
7 }
8
9 int A_Calibration()
10 {
11         DSCDACALPARAMS dscdacalparams;
12         memset(&dscdacalparams, 0, sizeof(DSCDACALPARAMS));
13         dscdacalparams.darange = 65535;
14         if( ( dscDAAutoCal( dscb, &dscdacalparams ) ) != DE_NONE )
15         {
16                 dscGetLastError(&errorParams);
17                 rt_printf("dscDAAutoCal error: %s %s\n", dscGetErrorString(errorParams.ErrCode), errorParams.
                        errstring );
18                 return 0;
19         }
20         if( ( dscDACalVerify( dscb, &dscdacalparams ) ) != DE_NONE )
21         {
22                 dscGetLastError(&errorParams);
23                 rt_printf("dscDACalVerify error: %s %s\n", dscGetErrorString(errorParams.ErrCode), errorParams.
                        errstring );
24                 return 0;
25         }
26         printf( "Offset Error: %f, Gain Error: %f\n", dscdacalparams.offset, dscdacalparams.gain );
27         if ( fabs( dscdacalparams.offset ) > 2.0 ||
28                  fabs( dscdacalparams.gain ) > 2.0 )
29                 rt_printf( "Value for offset or gain exceeded specified tolerance\n" );
30         else rt_printf( "Values for offset and gain met specified tolerance\n" );
31         rt_printf("Calibração dos conversores DA... COMPLETA\n\n" );
32         return 0;
33 }
34
35 int A_Kill()
36 {
37         dscFree();
38         return 0;
39 }
40
41 int A_Core()
42 {
43     dscDAConvert(dscb, channel , (int)4095*((setpoint+20)/40.0));
44         return 0;
45 }
46
47 void send_setpoint(double value)
48 {
49         dscDAConvert(dscb, channel , (int)4095*((value+20)/40));
50         sleep(2);
51 }
```

actuator.h:

```
1 #ifndef ACTUATOR_H_
2 #define ACTUATOR_H_
3
4 #include "global.h"
```

```
5  #include "include.h"
6
7  BYTE channel;
8  DSCDACODE output_code;
9  int A_Init();
10 int A_Calibration();
11 int A_Core();
12 int A_Kill();
13 void send_setpoint(double value);
14
15 #endif /* ACTUATOR_H_ */
```

comm.c:

```
1  #include "comm.h"
2
3  #define BAUDRATE B19200
4  #define MODEMDEVICE "/dev/ttyS0"
5  #define _POSIX_SOURCE 1
6
7  int Comm_Init()
8  {
9          printf("Inicializando porta serial para comunicação...\n");
10         fd = open_port();
11         if(fd == -1)
12         {
13                 printf("Não foi possível abrir a porta serial!\n");
14                 return 0;
15         }
16
17         tcgetattr(fd, &options);
18         old = options;
19         cfsetispeed(&options, BAUDRATE);
20         cfsetospeed(&options, BAUDRATE);
21         options.c_cflag |= (CLOCAL | CREAD);
22         options.c_cflag &= ~PARENB;
23         options.c_cflag &= ~CSTOPB;
24         options.c_cflag &= ~CSIZE;
25         options.c_cflag |= CS8;
26         options.c_cflag &= ~CRTSCTS;
27         options.c_iflag = IGNPAR | ICRNL;
28         options.c_iflag &= ~(IXON | IXOFF | IXANY);
29         options.c_oflag = 0;
30         options.c_lflag = (ICANON | ECHO | ECHOE);
31         memset(options.c_cc, 0, sizeof(options.c_cc));
32         options.c_cc[VEOF] = 4;
33         options.c_cc[VMIN] = 1;
34         tcflush(fd, TCIFLUSH);
35         tcsetattr(fd, TCSANOW, &options);
36         printf("A configuração da porta serial foi concluída!\n");
37         printf("Digite o nome do arquivo para salvar os dados (com o formato): ");
38         char filename[20];
39         scanf("%s", filename);
40         printf("Abrindo stream de arquivo para armazenamento de dados...\n");
41         if((file = fopen("startend.txt","w")) == NULL)
42         {
```

```
43                 printf("Não foi possível abrir o arquivo!\n");
44                 return 0;
45         }
46         printf("Concluído!\n");
47         savingData = FALSE;
48         lineCount = 0;
49         return 0;
50         running = false;
51 }
52
53 int Comm_Kill()
54 {
55         free(message);
56         tcsetattr(fd, TCSANOW, &old);
57         return 0;
58 }
59
60 int Comm_Core()
61 {
62         if(running == true)
63         {
64                 if(savingData)
65                 {
66                         saveToFile (file);
67                         lineCount++;
68                 }
69         }
70         else
71         {
72                 memset(buff, 0, sizeof(buff));
73                 int res;
74                 char* aux;
75                 aux = buff;
76                 while((res = read(fd, aux, buff + sizeof(buff) - aux -1)) > 0)
77                 {
78                         aux += res;
79                         if(aux[-1] == '\0')
80                                 break;
81                 }
82                 printf("mensagem recebida: ");
83                 message = (struct commdata *)buff;
84                 if(message->start == ':')
85                         if(message->id == 'A')
86                                 switch(message->command[0])
87                                 {
88                                 case ('S'):
89                                         printf("mensagem de config. - ");
90                                         switch(message->command[1])
91                                         {
92                                         case('T'):
93                                                         printf("tarefas\n");
94                                                         break;
95                                         }
96                                         break;
97                                 }
98         }
99         return 0;
```

```
100 }
101
102 void saveToFile (FILE *f){
103         if (file != NULL){
104                 fprintf(f, "%5.8f, %5.8f, %5.8f, %5.8f, %5.8f,\n", torque_reading, TORQUE[2], position_reading,
                        POSITION[2], setpoint);
105         }
106 }
107
108 void closeFile (FILE *f){
109         if (file != NULL){
110                 fclose(f);
111         }
112 }
113
114 void startSaving(){
115         rt_printf("Começou!\n");
116         dscDIOOutputBit(dscb, 1, 0, 1);
117         savingData = TRUE;
118 }
119
120 void finishSaving(){
121         rt_printf("Terminou!\n");
122         dscDIOOutputBit(dscb, 1, 0, 0);
123         savingData = FALSE;
124         closeFile(file);
125 }
126
127 int open_port()
128 {
129         int fd;
130         fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NDELAY);
131         if (fd == -1)
132         {
133                 printf("open_port: Unable to open port\n");
134         }
135         else
136                 fcntl(fd, F_SETFL, FNDELAY);
137         return (fd);
138 }
```

comm.h:

```
1  #ifndef COMM_H_
2  #define COMM_H_
3
4  #include "global.h"
5  #include "include.h"
6
7  int fd;
8  short int savingData;
9  char buff[247];
10 bool running;
11
12 //Structs
13 #pragma pack(push, back)
```

```
#pragma pack(1)
struct commdata
{
        char                            start;
        char                            id;
        char                            command[2];
        unsigned char           size;
        char                       *data;
        char                    checksum[2];
} *message;
#pragma pack(pop, back)

//struct sockaddr_in address;
struct termios options, old;

//File management
FILE *file;
int lineCount;

//Task functions
int Comm_Init();
int Comm_Kill();
int Comm_Core();
int open_port();

//File management functions
void saveToFile (FILE *f);
void closeFile (FILE *f);
void startSaving();
void finishSaving();

#endif /* COMM_H_ */
```

control.c:

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

int C_Init()
{
        POSITION[0] = 0;
        POSITION[1] = 0;
        setpoint = 0;

        Tsample = 0.002;
        J = coef*0.112;
        B = coef*2.5365;
        K = coef*8.6590;

        if(DISCRETIZATION == TUSTIN)
        {
                a = 1;
                b = 2;
                c = 1;
                d = 4*J/pow(Tsample, 2) + 2*B/Tsample + K;
                e = -8*J/pow(Tsample, 2) + 2*K;
```

```
22                  f = 4*J/pow(Tsample, 2) - 2*B/Tsample + K;
23          }
24          else if(DISCRETIZATION == BACKWARD_DIFF)
25          {
26                  a = 1;
27                  d = J/pow(Tsample, 2) + B/Tsample + K;
28                  e = -2*J/pow(Tsample, 2) -B/Tsample;
29                  f = J/pow(Tsample, 2);
30          }
31          trajectory_time = 0;
32          return 0;
33  }
34
35  int C_Kill()
36  {
37          return 0;
38  }
39
40  int C_Core()
41  {
42          if(TRAJECTORY == SINUSOIDAL_POS){                       //senoidal
43                  trajectory_time += 2;
44                  setpoint = 20*sin(2*M_PI*trajectory_time/TRAJ_PERIOD);
45          }
46          else if(TRAJECTORY == CONSTANT)                         //cte
47          {
48                  setpoint = SET_CTE;
49          }
50          else if(TRAJECTORY == SINUSOIDAL_TOR || TRAJECTORY == CONTROL_ACTIVE)                           //controle
                ativado
51          {
52                  TORQUE[0] = TORQUE[1];
53                  TORQUE[1] = TORQUE[2];
54                  TORQUE[2] = (torque_reading-extensometerOffset-3.198)/(-0.0865);
55                  POSITION[0] = POSITION[1];
56                  POSITION[1] = POSITION[2];
57                  POSITION[2] = 62.9774*position_reading - potentiometerOffset;
58                  if(DISCRETIZATION == TUSTIN)
59                  {
60                          setpoint = c*TORQUE[2]+b*TORQUE[1]+a*TORQUE[0]-e*POSITION[1]-f*POSITION[0];
61                          setpoint = setpoint/d;
62                  }
63                  else if (DISCRETIZATION == BACKWARD_DIFF)
64                  {
65                          setpoint = a*TORQUE[2]-e*POSITION[1]-f*POSITION[0];
66                          setpoint = setpoint/d;
67                  }
68                  if (setpoint > 20) setpoint = 20;
69                  if (setpoint < -20) setpoint = -20;
70          }
71          else if(TRAJECTORY == RANDOM_EXCITATION)
72          {
73                  POSITION[0] = POSITION[1];
74                  POSITION[1] = setpoint;
75                  if(random()%10 >= 5)
76                  {
77                          setpoint += 3;
```

```
78                  }
79                  else
80                  {
81                          setpoint -= 3;
82                  }
83                  if(setpoint > 10)
84                  {
85                          setpoint -= 4;
86                  }
87                  else if(setpoint < -10)
88                  {
89                          setpoint += 4;
90                  }
91          }
92          return 0;
93  }
```

control.h:

```
1   #ifndef CONTROL_H_
2   #define CONTROL_H_
3
4   #include "global.h"
5   #include "include.h"
6   //Setpoint enviado quando no modo de setpoint constante
7   #define SET_CTE 0
8   //Periodo do movimento na trajetória senoidal
9   #define TRAJ_PERIOD 8000 //ms
10  //Parâmetros de inicialização do controle
11  double Tsample;
12  //Parâmetros de controle discreto
13  double a;
14  double b;
15  double c;
16  double d;
17  double e;
18  double f;
19  //double a0;
20  //double a1;
21  //double a2;
22  //double b0;
23  //double b1;
24  //double b2;
25  double trajectory_time;
26  int C_Init();
27  int C_Kill();
28  int C_Core();
29
30  #endif /* CONTROL_H_ */
```

global.c:

```
1   #include "global.h"
2
3   short int MONITORING;
```

```
4  short int SENSORING;
5  short int TRAJECTORY;
6  short int DISCRETIZATION;
7  double J;
8  double K;
9  double B;
10 double coef;
11 //saída do controle
12 double setpoint;
13 //entrada do controle
14 double position_reading;
15 double torque_reading;
16 double potentiometerOffset;
17 double extensometerOffset;
18 DSCB dscb;
19 DSCCB dsccb;
```

global.h:

```
1  #ifndef GLOBAL_H_
2  #define GLOBAL_H_
3
4  #include "include.h"
5
6  #define NUMBEROFTASKS 5
7  #define bool    char
8  #define false   '0'
9  #define true    '1'
10
11 //ID das Tasks
12 #define SENSOR          0
13 #define CONTROL         1
14 #define ACTUATOR        2
15 #define COMM            3
16 #define TEST            4
17
18 //Valores para a variável DISCRETIZATION
19 #define TUSTIN                          1
20 #define BACKWARD_DIFF           2
21
22 //Valores para a variável TRAJECTORY
23 #define SINUSOIDAL_POS          1
24 #define CONSTANT                        2
25 #define SINUSOIDAL_TOR          3
26 #define CONTROL_ACTIVE          4
27 #define RANDOM_EXCITATION       5
28
29 //Conversão do Timer
30 #define NS2MS 1000000 //Ns para Ms
31 #define NS2US 1000    //Ns para Us
32
33 //Configuração da placa
34 extern DSCB dscb;
35 extern DSCCB dsccb;
36 ERRPARAMS errorParams;
37
```

```
38 //Opções iniciais
39 extern short int MONITORING;
40 extern short int SENSORING;
41 extern short int TRAJECTORY;
42 extern short int DISCRETIZATION;
43
44 //saida do controle
45 extern double setpoint;
46
47 //entrada do controle
48 extern double torque_reading;
49 extern double position_reading;
50
51 //Parâmetros do controlador
52 extern double J;
53 extern double K;
54 extern double B;
55 extern double coef;
56
57 //Offsets dos sensores
58 extern double extensometerOffset;
59 extern double potentiometerOffset;
60
61 //Valores armazenados para o controle
62 double POSITION[3];
63 double TORQUE[3];
64
65 #endif /* GLOBAL_H_ */
```

include.h:

```
1  #ifndef DEFINES_H_
2  #define DEFINES_H_
3
4  #include <stdio.h>
5  #include <signal.h>
6  #include <string.h>
7  #include <unistd.h>
8  #include <termios.h>
9  #include <fcntl.h>
10 #include <stdlib.h>
11 #include <math.h>
12 #include <sys/mman.h>
13 #include <sys/io.h>
14 #include <sys/types.h>
15 #include <sys/stat.h>
16 #include <native/task.h>
17 #include <native/timer.h>
18 #include <native/heap.h>
19 #include <rtdk.h>
20 #include <time.h>
21
22 #include "dscud.h"
23
24 #endif /* DEFINES_H_ */
```

sensor.c:

```
#include "sensor.h"


int S_Init()
{
        //inicializar variaveis globais do sensor
        osc_time = 0;
        TORQUE[0] = 0;
        TORQUE[1] = 0;
        TORQUE[2] = 0;
        POSITION[0] = 0;
        POSITION[1] = 0;
        POSITION[2] = 0;
        extensometerOffset = 0;
    potentiometerOffset = 0
        //inicializa as configuracoes do conversor AD
        memset(&dscadsettings, 0, sizeof(DSCADSETTINGS)); //zera valores de dscadsettings
        samples = (DSCSAMPLE*)malloc( sizeof(DSCSAMPLE) * ( dscadscan.high_channel - dscadscan.low_channel + 1 ) )
             ;
        dscadsettings.range = RANGE_5;
        dscadsettings.gain = GAIN_1;
        dscadsettings.load_cal = (BYTE)FALSE;
        dscadsettings.current_channel = 0;
        dscadscan.low_channel = 0;
        dscadscan.high_channel = 2;
        dscadscan.gain = dscadsettings.gain;
        if( ( result = dscADSetSettings( dscb, &dscadsettings ) ) != DE_NONE )
        {
                dscGetLastError(&errorParams);
                rt_printf( "dscADSetSettings error: %s %s\n", dscGetErrorString(errorParams.ErrCode), errorParams.
                     errstring );
                return 0;
        }
        return 0;
}

void S_CalcOffset()
{
        int i;
        double reading_ext, reading_pot;
        double sumtor = 0;
        double sumpos = 0;
        for (i = 0; i<10000; i++)
        {
                if( ( result = dscADScan( dscb, &dscadscan, samples ) ) != DE_NONE )
                {
                        dscGetLastError(&errorParams);
                        fprintf( stderr, "dscADScan error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
                             errorParams.errstring );
                        return;
                }
                if( dscADCodeToVoltage(dscb, dscadsettings, dscadscan.sample_values[0], &reading_ext) != DE_NONE)
                {
                        dscGetLastError(&errorParams);
                        fprintf( stderr, "dscInit error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
```

```
                                errorParams.errstring );
53                      return;
54              }
55              sumtor += reading_ext;
56              if( dscADCodeToVoltage(dscb, dscadsettings, dscadscan.sample_values[2], &reading_pot) != DE_NONE)
57              {
58                      dscGetLastError(&errorParams);
59                      fprintf( stderr, "dscInit error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
                                errorParams.errstring );
60                      return;
61              }
62              sumpos += reading_pot;
63              usleep(100);
64      }
65      extensometerOffset = sumtor/10000.0 - 3.198; //3.24
66      potentiometerOffset = 62.9774*sumpos/10000.0;
67  }
68
69  double S_getTorque()
70  {
71      double reading_ext;
72      if( ( result = dscADScan( dscb, &dscadscan, samples ) ) != DE_NONE )
73      {
74              dscGetLastError(&errorParams);
75              fprintf( stderr, "dscADScan error: %s %s\n", dscGetErrorString(errorParams.ErrCode), errorParams.
                        errstring );
76              return -1;
77      }
78      if( dscADCodeToVoltage(dscb, dscadsettings, dscadscan.sample_values[0], &reading_ext) != DE_NONE)
79      {
80              dscGetLastError(&errorParams);
81              fprintf( stderr, "dscInit error: %s %s\n", dscGetErrorString(errorParams.ErrCode), errorParams.
                        errstring );
82              return -1;
83      }
84      return (reading_ext);
85  }
86
87  int S_Calibration(){
88      int i;
89      rt_printf( "Calibração dos conversores AD...\n\n" );
90      dscadcalparams.adrange = 0xFF;
91      dscadcalparams.boot_adrange = 0;
92      dscautocal.adrange = 0xFF;
93      dscautocal.boot_adrange = 0;
94      if( (result = dscADAutoCal( dscb, &dscautocal )) != DE_NONE )
95      {
96              dscGetLastError(&errorParams);
97              rt_printf("dscADAutoCal error: %s %s\n", dscGetErrorString(errorParams.ErrCode), errorParams.
                        errstring );
98              return 0;
99      }
100     for( i = 0; i < 16; i++ )
101     {
102             if ( ( i > 3 && i < 8 )  )
103                     continue;
104             dscautocal.adrange = i;
```

```
105                 dscautocal.ad_gain = 0;
106                 dscautocal.ad_offset = 0;
107                 if( ( result = dscADCalVerify( dscb, &dscautocal ) ) != DE_NONE )
108                 {
109                         dscGetLastError(&errorParams);
110                         rt_printf( "dscADCalVerify error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
                                errorParams.errstring );
111                         return 0;
112                 }
113                 rt_printf( "Configuration Mode: %d, Offset Error: %9.3f, Gain Error: %9.3f\n", i, dscautocal.
                        ad_offset, dscautocal.ad_gain );
114                 if ( fabs( dscautocal.ad_offset ) > 2.0f ||
115                          fabs( dscautocal.ad_gain ) > 2.0f )
116                         rt_printf( "Value for offset or gain exceeded specified tolerance\n" );
117                 else rt_printf( "Values for offset and gain met specified tolerance\n" );
118         }
119         rt_printf( "Calibração dos conversores AD... COMPLETA\n\n" );
120         return 0;
121 }
122
123 int S_Kill()
124 {
125         free(samples);
126         return 0;
127 }
128
129 int S_Core()
130 {
131     if( ( result = dscADScan( dscb, &dscadscan, samples ) ) != DE_NONE )
132         {
133                 dscGetLastError(&errorParams);
134                 fprintf( stderr, "dscADScan error: %s %s\n", dscGetErrorString(errorParams.ErrCode), errorParams.
                        errstring );
135                 free( samples );
136                 return 0;
137         }
138         int i;
139         for( i = 0; i < (dscadscan.high_channel - dscadscan.low_channel)+ 1; i++)
140         {
141                 if (i == 0)                             //TORQUE
142                 {
143                         if( dscADCodeToVoltage(dscb, dscadsettings, dscadscan.sample_values[i], &torque_reading)
                                != DE_NONE)
144                         {
145                           dscGetLastError(&errorParams);
146                           fprintf( stderr, "dscInit error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
                                  errorParams.errstring );
147                           return 0;
148                         }
149                 } else if (i == 1)              //POSITION
150                 {
151                         if( dscADCodeToVoltage(dscb, dscadsettings, dscadscan.sample_values[i+1], &
                                position_reading) != DE_NONE)
152                         {
153                           dscGetLastError(&errorParams);
154                           fprintf( stderr, "dscInit error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
                                  errorParams.errstring );
```

```
155                                 free(samples);
156                                 return 0;
157                         }
158                 } else
159                         break;
160         }
161         return 0;
162 }
```

sensor.h:

```
1  #ifndef SENSOR_H_
2  #define SENSOR_H_
3
4  #include "global.h"
5  #include "include.h"
6  #define BILLION 1E9
7  #define OSC_PERIOD 8000
8
9  BYTE result;                                    // returned0f error code
10 DSCSAMPLE sample;                               // sample reading
11 DSCADCALPARAMS dscadcalparams;
12 DSCADSETTINGS dscadsettings;    // structure containing A/D conversion settings
13 DSCAUTOCAL dscautocal;                 // structure containing auto-calibration settings
14 DSCSAMPLE* samples;             // sample readings
15 DSCADSCAN dscadscan;            // structure containing A/D scan settings
16
17 //Time tracking
18 struct timespec start;
19 struct timespec end;
20
21 //Sensor reading
22 double osc_time;
23
24 //Task functions
25 int S_Init();
26 int S_Calibration();
27 int S_Kill();
28 int S_Core();
29
30 //Tool functions
31 //void set_reference_voltage();
32 void S_CalcOffset();
33 double S_getTorque();
34
35 #endif /* SENSOR_H_ */
```

tasks.c:

```
1  #include "tasks.h"
2
3  void T_Init()
4  {
5          //Declaração do ponteiro de funções
6          pfunction[SENSOR]    = S_Core;
```

```
7           pfunction[CONTROL]   = C_Core;
8           pfunction[ACTUATOR]  = A_Core;
9           pfunction[COMM]          = Comm_Core;
10          pfunction[TEST]      = T_Test;
11
12          //Habilita/Desabilita Task
13          ENABLETASK[SENSOR]   = FALSE;
14          ENABLETASK[CONTROL]  = FALSE;
15          ENABLETASK[ACTUATOR] = FALSE;
16          ENABLETASK[COMM]         = TRUE;
17          ENABLETASK[TEST]         = FALSE;
18
19          //Período das Tasks [ms]
20          PERIOD[SENSOR]   = 2;
21          PERIOD[CONTROL]  = 2;
22          PERIOD[ACTUATOR] = 2;
23          PERIOD[COMM]     = 2;
24          PERIOD[TEST]     = 2;
25
26          //Prioridade das Tasks [0-100]
27          PRIORITY[SENSOR]   = 50;
28          PRIORITY[CONTROL]  = 50;
29          PRIORITY[ACTUATOR] = 50;
30          PRIORITY[COMM]     = 50;
31          PRIORITY[TEST]     = 50;
32
33          A_Init();
34          C_Init();
35          S_Init();
36          Comm_Init();
37  }
38
39  void T_Kill()
40  {
41  //      A_Kill();
42  //      C_Kill();
43  //      S_Kill();
44          dscFree();
45  }
46
47  void T_Xenomai()
48  {
49          mlockall(MCL_CURRENT|MCL_FUTURE);
50          rt_print_auto_init(1);
51          rt_timer_set_mode(1000000); //1ms -not working for periodic mode
52  }
53
54  void T_Create()
55  {
56          rt_task_create(&TaskSensor, "Sensor", 0, PRIORITY[SENSOR], 0);
57          rt_task_create(&TaskControl, "Controle", 0, PRIORITY[CONTROL], 0);
58          rt_task_create(&TaskActuator, "Atuador", 0, PRIORITY[ACTUATOR], 0);
59          rt_task_create(&TaskComm, "Comunicação", 0, PRIORITY[COMM], 0);
60          rt_task_create(&TaskTest, "Teste", 0, PRIORITY[TEST], 0);
61  }
62
63  void T_Start()
```

```
64  {
65          if (ENABLETASK[SENSOR]) {
66                  rt_task_start(&TaskSensor, &T_Monitor, SENSOR);
67          }
68          if (ENABLETASK[CONTROL]) {
69                  rt_task_start(&TaskControl, &T_Monitor, CONTROL);
70          }
71          if (ENABLETASK[ACTUATOR]) {
72                  rt_task_start(&TaskActuator, &T_Monitor, ACTUATOR);
73          }
74          if (ENABLETASK[COMM]) {
75                  rt_task_start(&TaskComm, &T_Monitor, COMM);
76          }
77          if (ENABLETASK[TEST])
78                  rt_task_start(&TaskTest, &T_Monitor, TEST);
79  }
80
81  void Start_Comm()
82  {
83          if (ENABLETASK[COMM]) {
84  //              Comm_Init();
85                  rt_task_start(&TaskComm, &T_Monitor, COMM);
86          }
87  }
88
89  void T_CleanUp (void)
90  {
91          rt_task_delete(&TaskControl);
92          rt_task_delete(&TaskSensor);
93          rt_task_delete(&TaskActuator);
94          rt_task_delete(&TaskComm);
95          rt_task_delete(&TaskTest);
96  }
97
98  void T_PrintTaskName()
99  {
100         RT_TASK *curtask;
101         RT_TASK_INFO curtaskinfo;
102         curtask=rt_task_self();
103         rt_task_inquire(curtask,&curtaskinfo);
104         rt_printf("%s␣",curtaskinfo.name);
105 }
106
107 void T_Monitor(int functionID)
108 {
109         RTIME start, end, duration, idle, period;
110         unsigned long overs;
111         period = PERIOD[functionID]*NS2MS;
112         rt_task_set_periodic(NULL, TM_NOW, period);
113         end = rt_timer_read();
114         rt_task_wait_period(&overs);
115         while (1) {
116                 if (MONITORING) {
117                         start = rt_timer_read();
118                         idle = start-end;
119                         T_PrintTaskName();
120                         rt_printf(":\t%ld.%06ld␣ms\t%6lld.%03lld␣us\t\"%d\"\n",
```

```
121                                     (long) duration / NS2MS,
122                                     (long) duration % NS2MS,
123                                     (signed long long)(duration+idle-period)/ NS2US, //conversion to us
124                                     (duration+idle-period) % NS2US,
125                                     overs);
126                 }
127                 pfunction[functionID]();
128                 if (MONITORING) {
129                         end = rt_timer_read();
130                         duration = end-start;
131                 }
132                 rt_task_wait_period(&overs);
133                 if(overs>0) rt_printf("!\n");
134         }
135         return;
136 }
137
138 int T_Test(){
139         return 0;
140 }
```

tasks.h:

```
1  #ifndef TASKS_H_
2  #define TASKS_H_
3
4  #include "global.h"
5  #include "include.h"
6  #include "control.h"
7  #include "sensor.h"
8  #include "actuator.h"
9  #include "comm.h"
10
11 //Declaração das Tasks
12 RT_TASK TaskSensor, TaskControl, TaskActuator, TaskComm, TaskTest;
13
14 //Período das Tasks [ms]
15 int PERIOD[NUMBEROFTASKS];
16 int PRIORITY[NUMBEROFTASKS];
17 int ENABLETASK[NUMBEROFTASKS];
18
19 //Array de Funções
20 int (*pfunction[NUMBEROFTASKS])();
21
22 void T_Init();
23 void T_Kill();
24 void T_Xenomai();
25 void T_Create();
26 void T_Start();
27 void Start_Comm();
28 void T_Monitor(int);
29 void T_CleanUp ();
30 void T_PrintTaskName();
31 int T_Test();
32
33 #endif /* TASKS_H_ */
```

APÊNDICE D – CÓDIGOS DA INTERFACE

- Main.cpp

```
#include "mainwindow.h"
#include <QApplication>
#include "config.h"

Registers Config::reg;

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

- MainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "config.h"
#include "dialt.h"
#include "dialr.h"
#include "dialogc.h"
#include "dialtorplot.h"
#include "dialposplot.h"
#include "dialquit.h"
#include "sequenceplot.h"
#include "threadStarter.h"
#include "dialcomm.h"
#include <QFileDialog>
#include <QFile>
#include <QTextStream>
#include <QAction>
#include <QString>
#include <QStringList>
#include <QMessageBox>
#include <qwt_plot_marker.h>
#include <qwt_symbol.h>
#include <qwt_plot_item.h>
#include <QErrorMessage>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

```

```
39  private slots:
40
41      //Utilities
42      void close_program();
43      void save_config();
44      void load_config();
45
46      //Plot
47      void update_plot_canvas();
48      void append_pos(const QPointF &point);
49      void append_ext(const QPointF &point);
50
51      //GUI related
52      void control_signal_emitted(bool on);
53      void control_pause_signal_emitted(bool on);
54      void show_error(QString str);
55      void show_status(QString str);
56      void update_connection(QStringList str);
57
58      //Thread related
59      void connect_thread_signals();
60
61      //Button slots
62      void on_taskButton_clicked();
63      void on_convButton_clicked();
64      void on_regButton_clicked();
65      void on_posSaveButton_clicked();
66      void on_torSaveButton_clicked();
67      void on_torConfigButton_clicked();
68      void on_posConfigButton_clicked();
69      void on_pushButton_clicked();
70      void on_controlStartButton_clicked();
71      void on_controlPauseButton_clicked();
72      void on_controlStopButton_clicked();
73      void on_commConfButton_clicked();
74
75      void commConfSeq();
76
77  signals:
78      void on_controlStartup(bool on);
79      void on_controlPause(bool on);
80      void comm_config(QHostAddress ipconf, quint16 portconf);
81
82      //SINAIS PARA A COMM.
83      void config();
84      void sendRequest(int msg);
85      void startComm();
86      void stopComm();
87      void pauseComm(bool pause);
88      void finishComm();
89
90  private:
91      threadStarter *thread;
92      double timepot, timetor;
93      QPalette red;
94      QPalette black;
95      Ui::MainWindow *ui;
```

```
96      bool running;
97  };
98
99  #endif // MAINWINDOW_H
```

- MainWindow.cpp

```
1
2
3   typedef short int (Registers::*sigetter)(void);
4   sigetter sigetters[NSI];
5
6   typedef bool (Registers::*bgetter)(void);
7   bgetter bgetters[NB];
8
9   typedef double (Registers::*dgetter)(void);
10  dgetter dgetters[ND];
11
12  typedef int (Registers::*igetter)(void);
13  igetter igetters[NI];
14
15  typedef QString (Registers::*sgetter)(void);
16  sgetter sgetters[NST];
17
18  typedef void (Registers::*sisetter)(short int);
19  sisetter sisetters[NSI];
20
21  typedef void (Registers::*bsetter)(bool);
22  bsetter bsetters[NB];
23
24  typedef void (Registers::*dsetter)(double);
25  dsetter dsetters[ND];
26
27  typedef void (Registers::*isetter)(int);
28  isetter isetters[NI];
29
30  typedef void (Registers::*ssetter)(QString);
31  ssetter ssetters[NST];
32
33  MainWindow::MainWindow(QWidget *parent) :
34      QMainWindow(parent),
35      ui(new Ui::MainWindow)
36  {
37      ui->setupUi(this);
38      //INICIALIZACAO DE MEMBROS
39      sigetters[0] = &Registers::getMonitoring;
40      sigetters[1] = &Registers::getSensoring;
41      sigetters[2] = &Registers::getTrajectory;
42      sigetters[3] = &Registers::getSensorPriority;
43      sigetters[4] = &Registers::getSensorPeriod;
44      sigetters[5] = &Registers::getControlPriority;
45      sigetters[6] = &Registers::getControlPeriod;
46      sigetters[7] = &Registers::getActuatorPriority;
47      sigetters[8] = &Registers::getActuatorPeriod;
48      sigetters[9] = &Registers::getSerialPriority;
49      sigetters[10] = &Registers::getSerialPeriod;
```

```
50      sigetters[11] = &Registers::getPosPlotUnit;
51      sigetters[12] = &Registers::getTorPlotUnit;
52      sigetters[13] = &Registers::getMachineId;
53
54      bgetters[0] = &Registers::getSensorEnable;
55      bgetters[1] = &Registers::getControlEnable;
56      bgetters[2] = &Registers::getActuatorEnable;
57      bgetters[3] = &Registers::getSerialOn;
58      bgetters[4] = &Registers::getUdpOn;
59
60      dgetters[0] = &Registers::getPosXMax;
61      dgetters[1] = &Registers::getPosYMax;
62      dgetters[2] = &Registers::getTorXMax;
63      dgetters[3] = &Registers::getTorYMax;
64      dgetters[4] = &Registers::getPosXMin;
65      dgetters[5] = &Registers::getPosYMin;
66      dgetters[6] = &Registers::getTorXMin;
67      dgetters[7] = &Registers::getTorXMin;
68      dgetters[8] = &Registers::getPosXStep;
69      dgetters[9] = &Registers::getTorXStep;
70      dgetters[10] = &Registers::getPosYStep;
71      dgetters[11] = &Registers::getPosYStep;
72      dgetters[12] = &Registers::getBaud;
73
74      igetters[0] = &Registers::getPort;
75
76      sgetters[0] = &Registers::getIp;
77      sgetters[1] = &Registers::getSerialPort;
78
79      sisetters[0] = &Registers::setMonitoring;
80      sisetters[1] = &Registers::setSensoring;
81      sisetters[2] = &Registers::setTrajectory;
82      sisetters[3] = &Registers::setSensorPriority;
83      sisetters[4] = &Registers::setSensorPeriod;
84      sisetters[5] = &Registers::setControlPriority;
85      sisetters[6] = &Registers::setControlPeriod;
86      sisetters[7] = &Registers::setActuatorPriority;
87      sisetters[8] = &Registers::setActuatorPeriod;
88      sisetters[9] = &Registers::setSerialPriority;
89      sisetters[10] = &Registers::setSerialPeriod;
90      sisetters[11] = &Registers::setPosPlotUnit;
91      sisetters[12] = &Registers::setTorPlotUnit;
92      sisetters[13] = &Registers::setMachineId;
93
94      bsetters[0] = &Registers::setSensorEnable;
95      bsetters[1] = &Registers::setControlEnable;
96      bsetters[2] = &Registers::setActuatorEnable;
97      bsetters[3] = &Registers::setSerialOn;
98      bsetters[4] = &Registers::setUdpOn;
99
100     dsetters[0] = &Registers::setPosXMax;
101     dsetters[1] = &Registers::setPosYMax;
102     dsetters[2] = &Registers::setTorXMax;
103     dsetters[3] = &Registers::setTorYMax;
104     dsetters[4] = &Registers::setPosXMin;
105     dsetters[5] = &Registers::setPosYMin;
106     dsetters[6] = &Registers::setTorXMin;
```

```
107     dsetters[7] = &Registers::setTorXMin;
108     dsetters[8] = &Registers::setPosXStep;
109     dsetters[9] = &Registers::setTorXStep;
110     dsetters[10] = &Registers::setPosYStep;
111     dsetters[11] = &Registers::setPosYStep;
112     dsetters[12] = &Registers::setBaud;
113 
114     isetters[0] = &Registers::setPort;
115 
116     ssetters[0] = &Registers::setIp;
117     ssetters[1] = &Registers::setSerialPort;
118 
119     //Inicializa variaveis globais
120     running = false;
121     timepot = 0;
122     timetor = 0;
123     red.setColor(QPalette::WindowText,Qt::red);
124     black.setColor(QPalette::WindowText,Qt::black);
125 
126     //Inicializa a GUI
127     ui->potPlot->setAxisScale(0,-20,20,5);
128     ui->potPlot->setAxisScale(2,0,1,0.1);
129     ui->torPlot->setAxisScale(0,-2,2,0.5);
130     ui->torPlot->setAxisScale(2,0,1,0.1);
131     ui->potPlot->setAxisTitle(0,"Tension␣(V)");
132     ui->potPlot->setAxisTitle(2,"Time␣(s)");
133     ui->torPlot->setAxisTitle(0,"Tension␣(V)");
134     ui->torPlot->setAxisTitle(2,"Time␣(s)");
135     ui->connButton->setEnabled(false);
136     ui->driverConfigButton->setEnabled(false);
137     ui->driverPlayPauseButton->setEnabled(false);
138 
139     //Conecta sinais internos
140     connect(ui->actionClose, SIGNAL(triggered()), this, SLOT(close_program()));
141     connect(ui->actionSave_Current_Config, SIGNAL(triggered()), this, SLOT(save_config()));
142     connect(ui->actionLoad_Config, SIGNAL(triggered()), this, SLOT(load_config()));
143     connect(this, SIGNAL(on_controlStartup(bool)), this, SLOT(control_signal_emitted(bool)),Qt::
            DirectConnection);
144     connect(this, SIGNAL(on_controlPause(bool)), this, SLOT(control_pause_signal_emitted(bool)),Qt::
            DirectConnection);
145 
146     thread = NULL;
147     emit on_controlStartup(false);
148 }
149 
150 MainWindow::~MainWindow()
151 {
152 //    thread->quit();
153     delete ui;
154 }
155 
156 void MainWindow::update_plot_canvas(){
157     ui->potPlot->setAxisScale(0,Config::reg.getPosYMin(),Config::reg.getPosYMax(),Config::reg.getPosYStep
            ());
158     ui->potPlot->setAxisScale(2,Config::reg.getPosXMin(),Config::reg.getPosXMax(),Config::reg.getPosXStep
            ());
159     ui->torPlot->setAxisScale(0,Config::reg.getTorYMin(),Config::reg.getTorYMax(),Config::reg.getTorYStep
```

```
        ());
160     ui->torPlot->setAxisScale(2,Config::reg.getTorXMin(),Config::reg.getTorXMax(),Config::reg.getTorXStep
            ());
161     switch (Config::reg.getPosPlotUnit()){
162     case 0:
163         ui->potPlot->setAxisTitle(0, "Tension (V)");
164         break;
165     case 1:
166         ui->potPlot->setAxisTitle(0, "Tension (mV)");
167         break;
168     case 2:
169         ui->potPlot->setAxisTitle(0, "Angular Position (Deg)");
170         break;
171     case 3:
172         ui->potPlot->setAxisTitle(0, "Angular Position (Rad)");
173         break;
174     }
175     switch (Config::reg.getTorPlotUnit()){
176     case 0:
177         ui->torPlot->setAxisTitle(0, "Tension (V)");
178         break;
179     case 1:
180         ui->torPlot->setAxisTitle(0, "Tension (mV)");
181         break;
182     case 2:
183         ui->torPlot->setAxisTitle(0, "Torque (N.m)");
184         break;
185     case 3:
186         ui->torPlot->setAxisTitle(0, "Torque (N.mm)");
187         break;
188     }
189     ui->potPlot->updateAxes();
190     ui->torPlot->updateAxes();
191     ui->potPlot->replot();
192     ui->torPlot->replot();
193 }
194
195 void MainWindow::append_pos(const QPointF &point)
196 {
197     ui->potPlot->appendPoint(point);
198     if((double)point.x()>ui->potPlot->axisInterval(2).maxValue())
199     {
200         double min = ui->potPlot->axisInterval(2).maxValue();
201         double max = ui->potPlot->axisInterval(2).maxValue()*2 - ui->potPlot->axisInterval(2).minValue();
202         ui->potPlot->setAxisScale(2, min, max, Config::reg.getPosXStep());
203         ui->potPlot->replot();
204     }
205 }
206
207 void MainWindow::append_ext(const QPointF &point)
208 {
209     ui->torPlot->appendPoint(point);
210     if((double)point.x()>ui->torPlot->axisInterval(2).maxValue())
211     {
212         double min = ui->torPlot->axisInterval(2).maxValue();
213         double max = ui->torPlot->axisInterval(2).maxValue()*2 - ui->torPlot->axisInterval(2).minValue();
214         ui->torPlot->setAxisScale(2, min, max, Config::reg.getTorXStep());
```

```
215         ui->torPlot->replot();
216     }
217 }
218 
219 void MainWindow::show_error(QString str)
220 {
221     QErrorMessage err;
222     err.showMessage(str);
223     err.setModal(true);
224     err.exec();
225 }
226 
227 void MainWindow::show_status(QString str)
228 {
229     ui->statusBar->showMessage(str);
230 }
231 
232 void MainWindow::update_connection(QStringList str)
233 {
234     if(str.size() != 3){
235         show_error("Error in GUI update - Connection: size of the config stringlist is larger than 3");
236         return;
237     }
238 
239     for(int i = 0; i < str.size(); i++)
240     {
241         switch(i)
242         {
243         case 0:
244             ui->connectionLabel->setText((str)[i]);
245             if(QString::compare((str)[i],"Disconnected",Qt::CaseInsensitive) != 0)
246                 ui->connectionLabel->setPalette(black);
247             else
248                 ui->connectionLabel->setPalette(red);
249             break;
250         case 1:
251             ui->ipLabel->setText((str)[i]);
252             if(QString::compare((str)[i],"-",Qt::CaseInsensitive) != 0)
253                 ui->ipLabel->setPalette(black);
254             else
255                 ui->ipLabel->setPalette(red);
256             break;
257         case 2:
258             ui->portLabel->setText((str)[i]);
259             if(QString::compare((str)[i],"-",Qt::CaseInsensitive) != 0)
260                 ui->portLabel->setPalette(black);
261             else
262                 ui->portLabel->setPalette(red);
263         }
264     }
265 }
266 
267 void MainWindow::control_signal_emitted(bool on)
268 {
269 //    ui->connButton->setEnabled(!on);
270     ui->controlStartButton->setEnabled(!on);
271     ui->convButton->setEnabled(!on);
```

```
272 //    ui->driverConfigButton->setEnabled(!on);
273 //    ui->driverPlayPauseButton->setEnabled(!on);
274     ui->posConfigButton->setEnabled(!on);
275     ui->torConfigButton->setEnabled(!on);
276     ui->posSaveButton->setEnabled(!on);
277     ui->torSaveButton->setEnabled(!on);
278     ui->pushButton->setEnabled(!on);
279     ui->regButton->setEnabled(!on);
280     ui->taskButton->setEnabled(!on);
281     ui->controlPauseButton->setEnabled(on);
282     ui->controlStopButton->setEnabled(on);
283     ui->menuBar->setEnabled(!on);
284     ui->commConfButton->setEnabled(!on);
285     QStringList *connectioncfg = new QStringList();
286     if(on)
287     {
288         connectioncfg->append("Listening to");
289         if(Config::reg.getSerialOn())
290         {
291             connectioncfg->append(QString::number(Config::reg.getBaud()));
292             connectioncfg->append(QString::number(Config::reg.getMachineId()));
293         }
294         else
295         {
296             connectioncfg->append(Config::reg.getIp());
297             connectioncfg->append(QString::number(Config::reg.getPort()));
298         }
299     }
300     else
301     {
302         connectioncfg->append("Disconnected");
303         connectioncfg->append("-");
304         connectioncfg->append("-");
305     }
306     update_connection(*connectioncfg);
307     running = on;
308 }
309
310 void MainWindow::control_pause_signal_emitted(bool on)
311 {
312     ui->posConfigButton->setEnabled(on);
313     ui->torConfigButton->setEnabled(on);
314     ui->posSaveButton->setEnabled(on);
315     ui->torSaveButton->setEnabled(on);
316     ui->controlStopButton->setEnabled(!on);
317     ui->controlPauseButton->setText(on? "Continue":"Pause");
318     QStringList *connectioncfg = new QStringList();
319     if (Config::reg.getSerialOn())
320     {
321         if(!on)
322         {
323             connectioncfg->append("Listening to");
324             connectioncfg->append(QString::number(Config::reg.getBaud()));
325             connectioncfg->append(QString::number(Config::reg.getMachineId()));
326         }
327         else
328         {
```

```
329             connectioncfg->append("Paused");
330             connectioncfg->append(QString::number(Config::reg.getBaud()));
331             connectioncfg->append(QString::number(Config::reg.getMachineId()));
332         }
333     }
334     else if (Config::reg.getUdpOn())
335     {
336         if(!on)
337         {
338             connectioncfg->append("Listening to");
339             connectioncfg->append(Config::reg.getIp());
340             connectioncfg->append(QString::number(Config::reg.getPort()));
341         }
342         else
343         {
344             connectioncfg->append("Paused");
345             connectioncfg->append(Config::reg.getIp());
346             connectioncfg->append(QString::number(Config::reg.getPort()));
347         }
348     }
349     update_connection(*connectioncfg);
350     running = !on;
351 }
352
353 void MainWindow::connect_thread_signals()
354 {
355     if(thread != NULL)
356     {
357         connect(thread, SIGNAL(showError(QString)), this, SLOT(show_error(QString)));
358         connect(thread, SIGNAL(showMsg(QString)), this, SLOT(show_status(QString)));
359         connect(this, SIGNAL(pauseComm(bool)), thread, SLOT(emitPause(bool)));
360         connect(this, SIGNAL(startComm()), thread, SLOT(emitStart()));
361         connect(this, SIGNAL(stopComm()), thread, SLOT(emitStop()));
362         connect(this, SIGNAL(config()), thread, SLOT(emitUpdate()));
363         connect(this, SIGNAL(sendRequest(int)), thread, SLOT(emitSendRequest(int)));
364         connect(this, SIGNAL(finishComm()), thread, SLOT(emitFinish()));
365     }
366 }
367
368 void MainWindow::close_program(){
369     DialQuit dialq;
370     dialq.setModal(true);
371     dialq.exec();
372     if(dialq.getAnswer())
373         this->close();
374 }
375
376 void MainWindow::save_config()
377 {
378     QString fileName = QFileDialog::getSaveFileName(this,"","","Impedance Control Config File (*.iccf)");
379     QFile file (fileName);
380     if(!file.open(QIODevice::WriteOnly | QIODevice::Text))
381     {
382         statusBar()->showMessage("Operation Cancelled or could not open file to save");
383         return;
384     }
385     QTextStream out(&file);
```

```
386
387     for (int i = 0; i < NSI; i++){
388         out<<"00 "<<i<<" "<<QString::number((Config::reg.*sigetters[i])())<<"\n";
389     }
390
391     for (int i = 0; i < NB; i++){
392         out<<"01 "<<i<<" "<<QString::number((Config::reg.*bgetters[i])())<<"\n";
393     }
394
395     for (int i = 0; i < ND; i++){
396         out<<"02 "<<i<<" "<<QString::number((Config::reg.*dgetters[i])())<<"\n";
397     }
398
399     for (int i = 0; i < NI; i++){
400         out<<"03 "<<i<<" "<<QString::number((Config::reg.*igetters[i])())<<"\n";
401     }
402
403     for (int i = 0; i < NST; i++){
404         if (i != 1) //nao salva a porta serial
405             out<<"04 "<<i<<" "<<(Config::reg.*sgetters[i])()<<"\n";
406     }
407     file.close();
408 }
409
410 void MainWindow::load_config()
411 {
412     int j;
413     QString fileName = QFileDialog::getOpenFileName(this,"","","Impedance Control Config File (*.iccf)");
414     QFile file (fileName);
415     if(!file.open(QIODevice::ReadOnly | QIODevice::Text))
416     {
417         statusBar()->showMessage("Operation cancelled or could not open file to load");
418         return;
419     }
420     QTextStream in(&file);
421     QString line = in.readLine();
422     while(!line.isNull())
423     {
424         QStringList list = line.split(" ");
425         if (list.empty())
426         {
427             ui->statusBar->showMessage("Empty line in selected file");
428             return;
429         }
430         switch(list.first().toInt())
431         {
432         case 00:
433             list.pop_front();
434             if (list.empty())
435             {
436                 ui->statusBar->showMessage("Incomplete line in selected file");
437                 return;
438             }
439             j = list.first().toInt();
440             list.pop_front();
441             (Config::reg.*sisetters[j])(list.first().toInt());
442             break;
```

```
443         case 01:
444             list.pop_front();
445             if (list.empty())
446             {
447                 ui->statusBar->showMessage("Incomplete␣line␣in␣selected␣file");
448                 return;
449             }
450             j = list.first().toInt();
451             list.pop_front();
452             (Config::reg.*bsetters[j])(list.first().toInt());
453             break;
454         case 02:
455             list.pop_front();
456             if (list.empty())
457             {
458                 ui->statusBar->showMessage("Incomplete␣line␣in␣selected␣file");
459                 return;
460             }
461             j = list.first().toInt();
462             list.pop_front();
463             (Config::reg.*dsetters[j])(list.first().toDouble());
464             break;
465         case 03:
466             list.pop_front();
467             if (list.empty())
468             {
469                 ui->statusBar->showMessage("Incomplete␣line␣in␣selected␣file");
470                 return;
471             }
472             j = list.first().toInt();
473             list.pop_front();
474             (Config::reg.*isetters[j])(list.first().toInt());
475             break;
476         case 04:
477             list.pop_front();
478             if (list.empty())
479             {
480                 ui->statusBar->showMessage("Incomplete␣line␣in␣selected␣file");
481                 return;
482             }
483             j = list.first().toInt();
484             list.pop_front();
485             (Config::reg.*ssetters[j])(list.first());
486             break;
487         }
488         line = in.readLine();
489     }
490     file.close();
491     update_plot_canvas();
492 }
493
494 void MainWindow::on_taskButton_clicked()
495 {
496     Dialt taskdialog;
497     taskdialog.setModal(true);
498     taskdialog.exec();
499 }
```

```
500
501 void MainWindow::on_convButton_clicked()
502 {
503     Dialogc conversordialog;
504     conversordialog.setModal(true);
505     conversordialog.exec();
506 }
507
508 void MainWindow::on_regButton_clicked()
509 {
510     Dialr registerdialog;
511     registerdialog.setModal(true);
512     registerdialog.exec();
513 }
514
515 void MainWindow::on_posSaveButton_clicked()
516 {
517     QString fileName = QFileDialog::getSaveFileName(this,"","","Text␣File␣(*.txt)");
518     QFile file (fileName);
519     file.open(QIODevice::WriteOnly | QIODevice::Text);
520     QTextStream out(&file);
521     for(int i = 0; i < ui->potPlot->size(); i++)
522     {
523         out<<i<<"\t"<<ui->potPlot->dataSample(i).x()<<"\t"<<ui->potPlot->dataSample(i).y()<<"\n";
524     }
525     file.close();
526 }
527
528 void MainWindow::on_torSaveButton_clicked()
529 {
530     QString fileName = QFileDialog::getSaveFileName(this,"","","Text␣File␣(*.txt)");
531     QFile file (fileName);
532     file.open(QIODevice::WriteOnly | QIODevice::Text);
533     QTextStream out(&file);
534     for(int i = 0; i < ui->torPlot->size(); i++)
535     {
536         out<<i<<"\t"<<ui->torPlot->dataSample(i).x()<<"\t"<<ui->torPlot->dataSample(i).y()<<"\n";
537     }
538     file.close();
539 }
540
541 void MainWindow::on_torConfigButton_clicked()
542 {
543     DialTorPlot torplotdialog;
544     torplotdialog.setModal(true);
545     torplotdialog.exec();
546     update_plot_canvas();
547 }
548
549 void MainWindow::on_posConfigButton_clicked()
550 {
551     DialPosPlot posplotdialog;
552     posplotdialog.setModal(true);
553     posplotdialog.exec();
554     update_plot_canvas();
555 }
556
```

```
557 void MainWindow::on_commConfButton_clicked()
558 {
559     DialComm *commdialog = new DialComm(this);
560     commdialog->setModal(true);
561     commdialog->exec();
562     if(Config::reg.getSerialOn())
563     {
564         ui->comm1label->setText("Baud␣Rate:");
565         ui->comm2label->setText("Machine␣ID:");
566     }
567     else
568     {
569         ui->comm1label->setText("IP:");
570         ui->comm2label->setText("Port:");
571     }
572     if(thread == NULL)
573     {
574         thread = new threadStarter(this);
575         connect_thread_signals();
576         thread->start();
577     }
578     QThread::msleep(100);
579     emit(config());
580     emit(sendRequest(0));
581 }
582
583 void MainWindow::commConfSeq()
584 {
585 }
586
587 void MainWindow::on_pushButton_clicked()
588 {
589
590     double ypos, ytor;
591     ypos = ((Config::reg.getPosYMax() - Config::reg.getPosYMin())/2)*qSin(timepot*25)+(Config::reg.
          getPosYMax() + Config::reg.getPosYMin())/2;
592     ytor = ((Config::reg.getTorYMax() - Config::reg.getTorYMin())/2)*qSin(timetor*25)+(Config::reg.
          getTorYMax() + Config::reg.getTorYMin())/2;
593     timepot += 0.002;
594     timetor += 0.002;
595     ui->potPlot->appendPoint(QPointF(timepot, ypos));
596     ui->torPlot->appendPoint(QPointF(timetor, ytor));
597     if (timepot > Config::reg.getPosXMax())
598     {
599         timepot -= 0.002;
600         timepot -= Config::reg.getPosXMax();
601         ui->potPlot->clearPoints();
602     }
603     if (timetor > Config::reg.getTorXMax())
604     {
605         timetor -= 0.002;
606         timetor -= Config::reg.getTorXMax();
607         ui->torPlot->clearPoints();
608     }
609     CommMessage handshake;
610     handshake.start = ':';
611     handshake.id = 'A';
```

```
612     handshake.command[0] = 'H';
613     handshake.command[1] = '0';
614     handshake.data = (new QString("Hello, device!"))->toLocal8Bit().data();
615     handshake.datasize = (char)(strlen(handshake.data));
616     handshake.checksum[0] = '0';
617     handshake.checksum[1] = '0';
618     emit(sendRequest(0));
619 
620 }
621 
622 void MainWindow::on_controlStartButton_clicked()
623 {
624     if((ui->potPlot->size()>0) | (ui->torPlot->size()>0))
625     {
626         ui->potPlot->clearPoints();
627         ui->torPlot->clearPoints();
628     }
629     update_plot_canvas();
630     emit on_controlStartup(true);
631 }
632 
633 void MainWindow::on_controlPauseButton_clicked()
634 {
635     emit on_controlPause(running);
636 }
637 
638 void MainWindow::on_controlStopButton_clicked()
639 {
640     emit on_controlStartup(false);
641 }
```

- Config.h

```
1  #ifndef CONFIG_H
2  #define CONFIG_H
3  #include "registers.h"
4  
5  class Config
6  {
7  public:
8      static Registers reg;
9  };
10 
11 #endif // CONFIG_H
```

- DialComm.h

```
1  #ifndef DIALCOMM_H
2  #define DIALCOMM_H
3  
4  #include <QDialog>
5  #include <QtSerialPort/QSerialPortInfo>
6  
7  namespace Ui {
8  class DialComm;
```

```
}

class DialComm : public QDialog
{
    Q_OBJECT

public:
    explicit DialComm(QWidget *parent = 0);
    ~DialComm();

private slots:
    void on_buttonBox_accepted();

    void on_buttonBox_rejected();

    void on_serialButton_clicked(bool checked);

    void on_udpButton_clicked(bool checked);

    void on_udpButton_toggled(bool checked);

    void on_serialButton_toggled(bool checked);

    void on_serialCombo_currentIndexChanged(int index);

private:
    void updateSerial();

    Ui::DialComm *ui;
};

#endif // DIALCOMM_H
```

- DialComm.cpp

```
#include "dialcomm.h"
#include "ui_dialcomm.h"
#include "config.h"

DialComm::DialComm(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::DialComm)
{
    ui->setupUi(this);
    ui->baudBox->setCurrentIndex(ui->baudBox->findText(QString::number(Config::reg.getBaud())));
    ui->ipLineEdit->setText(Config::reg.getIp());
    ui->machineIDLineEdit->setText(QString::number(Config::reg.getMachineId()));
    ui->portLineEdit->setText(QString::number(Config::reg.getPort()));
    updateSerial();
    if(Config::reg.getSerialOn())
    {
        ui->serialButton->setChecked(true);
        ui->baudBox->setEnabled(true);
        ui->machineIDLineEdit->setEnabled(true);
        ui->ipLineEdit->setEnabled(false);
        ui->portLineEdit->setEnabled(false);
```

```
22          ui->serialCombo->setEnabled(true);
23          int index = 0;
24          if (QString::compare(Config::reg.getSerialPort(), "") != 0)
25              foreach(const QSerialPortInfo &info, QSerialPortInfo::availablePorts())
26              {
27                  index++;
28                  if (QString::compare(info.portName(), Config::reg.getSerialPort()) == 0)
29                  {
30                      ui->serialCombo->setCurrentIndex(index);
31                      break;
32                  }
33              }
34          else
35              ui->serialCombo->setCurrentIndex(0);
36
37          if(ui->serialCombo->currentIndex() == 0)
38              ui->buttonBox->buttons()[0]->setEnabled(false);
39      }
40      else if (Config::reg.getUdpOn())
41      {
42          ui->udpButton->setChecked(true);
43          ui->baudBox->setEnabled(false);
44          ui->machineIDLineEdit->setEnabled(false);
45          ui->ipLineEdit->setEnabled(true);
46          ui->portLineEdit->setEnabled(true);
47      }
48      else
49      {
50          ui->baudBox->setEnabled(false);
51          ui->machineIDLineEdit->setEnabled(false);
52          ui->ipLineEdit->setEnabled(true);
53          ui->portLineEdit->setEnabled(true);
54      }
55  }
56
57  DialComm::~DialComm()
58  {
59      delete ui;
60  }
61
62  void DialComm::on_buttonBox_accepted()
63  {
64      if(ui->serialButton->isChecked())
65      {
66          Config::reg.setBaud(ui->baudBox->currentText().toDouble());
67          Config::reg.setMachineId(ui->machineIDLineEdit->text().toShort());
68          Config::reg.setSerialOn(true);
69          Config::reg.setUdpOn(false);
70          Config::reg.setSerialPort(ui->serialCombo->currentText());
71      }
72      else if(ui->udpButton->isChecked())
73      {
74          Config::reg.setIp(ui->ipLineEdit->text());
75          Config::reg.setPort(ui->portLineEdit->text().toInt());
76          Config::reg.setSerialOn(false);
77          Config::reg.setUdpOn(true);
78      }
```

```
79      else
80      {
81          Config::reg.setIp(ui->ipLineEdit->text());
82          Config::reg.setPort(ui->portLineEdit->text().toInt());
83          Config::reg.setSerialOn(false);
84          Config::reg.setUdpOn(false);
85      }
86  }
87
88  void DialComm::on_buttonBox_rejected()
89  {
90      this->close();
91  }
92
93  void DialComm::on_serialButton_clicked(bool checked)
94  {
95      ui->baudBox->setEnabled(checked);
96      ui->machineIDLineEdit->setEnabled(checked);
97      ui->ipLineEdit->setEnabled(!checked);
98      ui->portLineEdit->setEnabled(!checked);
99      ui->serialCombo->setEnabled(checked);
100 }
101
102 void DialComm::on_udpButton_clicked(bool checked)
103 {
104     ui->baudBox->setEnabled(!checked);
105     ui->machineIDLineEdit->setEnabled(!checked);
106     ui->ipLineEdit->setEnabled(checked);
107     ui->portLineEdit->setEnabled(checked);
108     ui->serialCombo->setEnabled(!checked);
109 }
110
111 void DialComm::updateSerial()
112 {
113     ui->serialCombo->addItem("");
114     foreach(const QSerialPortInfo &info, QSerialPortInfo::availablePorts())
115         ui->serialCombo->addItem(info.portName());
116 }
117
118 void DialComm::on_udpButton_toggled(bool checked)
119 {
120     if(checked)
121         ui->buttonBox->buttons()[0]->setEnabled(true);
122 }
123
124 void DialComm::on_serialButton_toggled(bool checked)
125 {
126     bool isSelected;
127     if(ui->serialCombo->currentIndex() == 0)
128         isSelected = false;
129     else
130         isSelected = true;
131     if(checked)
132         ui->buttonBox->buttons()[0]->setEnabled(isSelected);
133 }
134
135 void DialComm::on_serialCombo_currentIndexChanged(int index)
```

```
136 {
137     if(index == 0)
138         ui->buttonBox->buttons()[0]->setEnabled(false);
139     else
140         ui->buttonBox->buttons()[0]->setEnabled(true);
141 }
```

- DialogC.h

```
1  #ifndef DIALOGC_H
2  #define DIALOGC_H
3
4  #include <QDialog>
5
6  namespace Ui {
7  class Dialogc;
8  }
9
10 class Dialogc : public QDialog
11 {
12     Q_OBJECT
13
14 public:
15     explicit Dialogc(QWidget *parent = 0);
16     ~Dialogc();
17
18 private:
19     Ui::Dialogc *ui;
20 };
21
22 #endif // DIALOGC_H
```

- DialogC.cpp

```
1  #include "dialogc.h"
2  #include "ui_dialogc.h"
3
4  Dialogc::Dialogc(QWidget *parent) :
5      QDialog(parent),
6      ui(new Ui::Dialogc)
7  {
8      ui->setupUi(this);
9      this->setWindowFlags(this->windowFlags() & ~Qt::WindowContextHelpButtonHint);
10 }
11
12 Dialogc::~Dialogc()
13 {
14     delete ui;
15 }
```

- DialPosPlot.h

```
1  #ifndef DIALPOSPLOT_H
```

```
#define DIALPOSPLOT_H

#include <QDialog>

namespace Ui {
class DialPosPlot;
}

class DialPosPlot : public QDialog
{
    Q_OBJECT

public:
    explicit DialPosPlot(QWidget *parent = 0);
    ~DialPosPlot();

private slots:
    void on_physButton_clicked();

    void on_vButton_clicked();

    void on_vButton_toggled(bool checked);

    void on_physButton_toggled(bool checked);

    void on_buttonBox_rejected();

    void on_buttonBox_accepted();

private:
    Ui::DialPosPlot *ui;
};

#endif // DIALPOSPLOT_H
```

- DialPosPlot.cpp

```
#include "dialposplot.h"
#include "ui_dialposplot.h"
#include "config.h"
#include <qstring.h>

DialPosPlot::DialPosPlot(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::DialPosPlot)
{
    Qt::WindowFlags flags(Qt::WindowTitleHint);
    this->setWindowFlags(flags);
    ui->setupUi(this);
    ui->xmaxLineEdit->setText(QString::number(Config::reg.getPosXMax()));
    ui->xminLineEdit->setText(QString::number(Config::reg.getPosXMin()));
    ui->ymaxLineEdit->setText(QString::number(Config::reg.getPosYMax()));
    ui->yminLineEdit->setText(QString::number(Config::reg.getPosYMin()));
    ui->stepSizeXLineEdit->setText(QString::number(Config::reg.getPosXStep()));
    ui->stepSizeYLineEdit->setText(QString::number(Config::reg.getPosYStep()));
    int unit = Config::reg.getPosPlotUnit();
```

```
20      if(unit <= 1){
21          ui->vButton->setChecked(true);
22          if(unit == 0){
23              ui->vCheck->setChecked(true);
24          } else {
25              ui->mvCheck->setChecked(true);
26          }
27      } else {
28          ui->physButton->setChecked(true);
29          if(unit == 2){
30              ui->degCheck->setChecked(true);
31          } else {
32              ui->radCheck->setChecked(true);
33          }
34      }
35  }
36
37  DialPosPlot::~DialPosPlot()
38  {
39      delete ui;
40  }
41
42  void DialPosPlot::on_physButton_clicked()
43  {
44      ui->degCheck->setEnabled(true);
45      ui->radCheck->setEnabled(true);
46      ui->vCheck->setEnabled(false);
47      ui->mvCheck->setEnabled(false);
48  }
49
50  void DialPosPlot::on_vButton_clicked()
51  {
52      ui->degCheck->setEnabled(false);
53      ui->radCheck->setEnabled(false);
54      ui->vCheck->setEnabled(true);
55      ui->mvCheck->setEnabled(true);
56  }
57
58  void DialPosPlot::on_vButton_toggled(bool checked)
59  {
60      if(checked == true){
61          ui->degCheck->setEnabled(false);
62          ui->radCheck->setEnabled(false);
63          ui->vCheck->setEnabled(true);
64          ui->mvCheck->setEnabled(true);
65      }
66  }
67
68  void DialPosPlot::on_physButton_toggled(bool checked)
69  {
70      if(checked == true){
71          ui->degCheck->setEnabled(true);
72          ui->radCheck->setEnabled(true);
73          ui->vCheck->setEnabled(false);
74          ui->mvCheck->setEnabled(false);
75      }
76  }
```

```
77
78  void DialPosPlot::on_buttonBox_rejected()
79  {
80      this->close();
81  }
82
83  void DialPosPlot::on_buttonBox_accepted()
84  {
85      if(ui->vButton->isChecked()){
86          if(ui->vCheck->isChecked()){
87              Config::reg.setPosPlotUnit(0);
88          } else {
89              Config::reg.setPosPlotUnit(1);
90          }
91      } else {
92          if(ui->degCheck->isChecked()){
93              Config::reg.setPosPlotUnit(2);
94          } else {
95              Config::reg.setPosPlotUnit(3);
96          }
97      }
98      Config::reg.setPosXMax(ui->xmaxLineEdit->text().toDouble());
99      Config::reg.setPosYMax(ui->ymaxLineEdit->text().toDouble());
100     Config::reg.setPosXMin(ui->xminLineEdit->text().toDouble());
101     Config::reg.setPosYMin(ui->yminLineEdit->text().toDouble());
102     Config::reg.setPosXStep(ui->stepSizeXLineEdit->text().toDouble());
103     Config::reg.setPosYStep(ui->stepSizeYLineEdit->text().toDouble());
104 }
```

- DialQuit.h

```
1   #ifndef DIALQUIT_H
2   #define DIALQUIT_H
3
4   #include <QDialog>
5
6   namespace Ui {
7   class DialQuit;
8   }
9
10  class DialQuit : public QDialog
11  {
12      Q_OBJECT
13
14  public:
15      explicit DialQuit(QWidget *parent = 0);
16      ~DialQuit();
17      bool getAnswer();
18
19  private slots:
20      void on_buttonBox_accepted();
21
22      void on_buttonBox_rejected();
23
24  private:
25      Ui::DialQuit *ui;
```

```
26      bool quitProgram;
27  };
28
29  #endif // DIALQUIT_H
```

- DialQuit.cpp

```
1   #include "dialquit.h"
2   #include "ui_dialquit.h"
3   #include <QMessageBox>
4
5   DialQuit::DialQuit(QWidget *parent) :
6       QDialog(parent),
7       ui(new Ui::DialQuit)
8   {
9       ui->setupUi(this);
10      Qt::WindowFlags flags(Qt::WindowTitleHint);
11      this->setWindowFlags(flags);
12  }
13
14  DialQuit::~DialQuit()
15  {
16      delete ui;
17  }
18
19  void DialQuit::on_buttonBox_accepted()
20  {
21      quitProgram = true;
22  }
23
24  void DialQuit::on_buttonBox_rejected()
25  {
26      quitProgram = false;
27  }
28
29  bool DialQuit::getAnswer(){
30      return (quitProgram);
31  }
```

- DialR.h

```
1   #ifndef DIALR_H
2   #define DIALR_H
3
4   #include <QDialog>
5   #include "config.h"
6   #include <stdlib.h>
7   #include <qstring.h>
8
9   namespace Ui {
10  class Dialr;
11  }
12
13  class Dialr : public QDialog
14  {
```

```
15      Q_OBJECT
16
17  public:
18      explicit Dialr(QWidget *parent = 0);
19      ~Dialr();
20
21  private slots:
22      void on_buttonBox_accepted();
23
24  private:
25      Ui::Dialr *ui;
26  };
27
28  #endif // DIALR_H
```

- DialR.cpp

```
1   #include "dialr.h"
2   #include "ui_dialr.h"
3
4   Dialr::Dialr(QWidget *parent) :
5       QDialog(parent),
6       ui(new Ui::Dialr)
7   {
8       ui->setupUi(this);
9       Config general_config;
10      ui->monitoringLineEdit->setText(QString::number(general_config.reg.getMonitoring()));
11      ui->sensoringLineEdit->setText(QString::number(general_config.reg.getSensoring()));
12      ui->trajectoryLineEdit->setText(QString::number(general_config.reg.getTrajectory()));
13      this->setWindowFlags(this->windowFlags() & ~Qt::WindowContextHelpButtonHint);
14  }
15
16  Dialr::~Dialr()
17  {
18      delete ui;
19  }
20
21  void Dialr::on_buttonBox_accepted()
22  {
23      Config general_config;
24      general_config.reg.setMonitoring(ui->monitoringLineEdit->text().toInt());
25      general_config.reg.setSensoring(ui->sensoringLineEdit->text().toInt());
26      general_config.reg.setTrajectory(ui->trajectoryLineEdit->text().toInt());
27  }
```

- DialS.h

```
1   #ifndef DIALS_H
2   #define DIALS_H
3
4   #include <QDialog>
5
6   namespace Ui {
7   class DialS;
8   }
```

```
9
10  class DialS : public QDialog
11  {
12      Q_OBJECT
13
14  public:
15      explicit DialS(QWidget *parent = 0);
16      ~DialS();
17
18  private:
19      Ui::DialS *ui;
20  };
21
22  #endif // DIALS_H
```

- DialS.cpp

```
1   #include "dials.h"
2   #include "ui_dials.h"
3
4   DialS::DialS(QWidget *parent) :
5       QDialog(parent),
6       ui(new Ui::DialS)
7   {
8       ui->setupUi(this);
9       this->setWindowFlags(this->windowFlags() & ~Qt::WindowContextHelpButtonHint);
10  }
11
12  DialS::~DialS()
13  {
14      delete ui;
15  }
```

- DialT.h

```
1   #ifndef DIALT_H
2   #define DIALT_H
3
4   #include <QDialog>
5   #include "config.h"
6
7   namespace Ui {
8   class Dialt;
9   }
10
11  class Dialt : public QDialog
12  {
13      Q_OBJECT
14
15  public:
16      explicit Dialt(QWidget *parent = 0);
17      ~Dialt();
18
19  private slots:
20      void on_sensorCheck_stateChanged(int arg1);
```

```
21      void on_controlCheck_stateChanged(int arg1);
22      void on_actuatorCheck_stateChanged(int arg1);
23      void on_serialCheck_stateChanged(int arg1);
24
25      void on_sensorEnable_stateChanged(int arg1);
26      void on_controlEnable_stateChanged(int arg1);
27      void on_actuatorEnable_stateChanged(int arg1);
28      void on_serialEnable_stateChanged(int arg1);
29
30      void on_buttonBox_accepted();
31
32      void on_buttonBox_rejected();
33
34  private:
35      Ui::Dialt *ui;
36  };
37
38  #endif // DIALT_H
```

- DialT.cpp

```
1   #include "dialt.h"
2   #include "ui_dialt.h"
3   #include <qstring.h>
4
5   Dialt::Dialt(QWidget *parent) :
6       QDialog(parent),
7       ui(new Ui::Dialt)
8   {
9       ui->setupUi(this);
10      ui->sensorPriorityLineEdit->setText(QString::number(Config::reg.getSensorPriority()));
11      ui->controlPriorityLineEdit->setText(QString::number(Config::reg.getControlPriority()));
12      ui->actuatorPriorityLineEdit->setText(QString::number(Config::reg.getActuatorPriority()));
13      ui->serialPriorityLineEdit->setText(QString::number(Config::reg.getSerialPriority()));
14      ui->sensorPeriodLineEdit->setText(QString::number(Config::reg.getSensorPeriod()));
15      ui->controlPeriodLineEdit->setText(QString::number(Config::reg.getControlPeriod()));
16      ui->actuatorPeriodLineEdit->setText(QString::number(Config::reg.getActuatorPeriod()));
17      ui->serialPeriodLineEdit->setText(QString::number(Config::reg.getSerialPeriod()));
18      if (Config::reg.getSensorEnable())
19      {
20          ui->sensorEnable->setChecked(true);
21          ui->sensorPeriodLineEdit->setDisabled(true);
22          ui->sensorPriorityLineEdit->setDisabled(true);
23      }
24      if (Config::reg.getControlEnable())
25      {
26          ui->controlEnable->setChecked(true);
27          ui->controlPeriodLineEdit->setDisabled(true);
28          ui->controlPriorityLineEdit->setDisabled(true);
29      }
30      if (Config::reg.getActuatorEnable())
31      {
32          ui->actuatorEnable->setChecked(true);
33          ui->actuatorPeriodLineEdit->setDisabled(true);
34          ui->actuatorPriorityLineEdit->setDisabled(true);
35      }
```

```
36      this->setWindowFlags(this->windowFlags() & ~Qt::WindowContextHelpButtonHint);
37  }
38
39  Dialt::~Dialt()
40  {
41      delete ui;
42  }
43
44  void Dialt::on_sensorCheck_stateChanged(int arg1)
45  {
46      if (arg1!=0){
47          ui->sensorEnable->setDisabled(false);
48          if(ui->sensorEnable->isChecked())
49          {
50              ui->sensorPeriodLineEdit->setDisabled(false);
51              ui->sensorPriorityLineEdit->setDisabled(false);
52          }
53          else
54          {
55              ui->sensorPeriodLineEdit->setDisabled(true);
56              ui->sensorPriorityLineEdit->setDisabled(true);
57          }
58      } else {
59          ui->sensorEnable->setDisabled(true);
60          ui->sensorPeriodLineEdit->setDisabled(true);
61          ui->sensorPriorityLineEdit->setDisabled(true);
62      }
63  }
64
65  void Dialt::on_controlCheck_stateChanged(int arg1)
66  {
67      if (arg1!=0){
68          ui->controlEnable->setDisabled(false);
69          if(ui->controlEnable->isChecked())
70          {
71              ui->controlPeriodLineEdit->setDisabled(false);
72              ui->controlPriorityLineEdit->setDisabled(false);
73          }
74          else
75          {
76              ui->controlPeriodLineEdit->setDisabled(true);
77              ui->controlPriorityLineEdit->setDisabled(true);
78          }
79      }
80      else
81      {
82          ui->controlEnable->setDisabled(true);
83          ui->controlPeriodLineEdit->setDisabled(true);
84          ui->controlPriorityLineEdit->setDisabled(true);
85      }
86  }
87
88  void Dialt::on_actuatorCheck_stateChanged(int arg1)
89  {
90      if (arg1!=0){
91          ui->actuatorEnable->setDisabled(false);
92          if(ui->actuatorEnable->isChecked())
```

```
93          {
94              ui->actuatorPeriodLineEdit->setDisabled(false);
95              ui->actuatorPriorityLineEdit->setDisabled(false);
96          }
97          else
98          {
99              ui->actuatorPeriodLineEdit->setDisabled(true);
100             ui->actuatorPriorityLineEdit->setDisabled(true);
101         }
102     } else {
103         ui->actuatorEnable->setDisabled(true);
104         ui->actuatorPeriodLineEdit->setDisabled(true);
105         ui->actuatorPriorityLineEdit->setDisabled(true);
106     }
107 }
108
109 void Dialt::on_serialCheck_stateChanged(int arg1)
110 {
111     if (arg1!=0){
112         ui->serialEnable->setDisabled(false);
113         if(ui->serialEnable->isChecked())
114         {
115             ui->serialPeriodLineEdit->setDisabled(false);
116             ui->serialPriorityLineEdit->setDisabled(false);
117         }
118         else
119         {
120             ui->serialPeriodLineEdit->setDisabled(true);
121             ui->serialPriorityLineEdit->setDisabled(true);
122         }
123     } else {
124         ui->serialEnable->setDisabled(true);
125         ui->serialEnable->setChecked(false);
126         ui->serialPeriodLineEdit->setDisabled(true);
127         ui->serialPriorityLineEdit->setDisabled(true);
128     }
129 }
130
131 void Dialt::on_sensorEnable_stateChanged(int arg1)
132 {
133     if (arg1!=0 && ui->sensorCheck){
134         ui->sensorPeriodLineEdit->setDisabled(false);
135         ui->sensorPriorityLineEdit->setDisabled(false);
136     } else {
137         ui->sensorPeriodLineEdit->setDisabled(true);
138         ui->sensorPriorityLineEdit->setDisabled(true);
139     }
140 }
141
142 void Dialt::on_controlEnable_stateChanged(int arg1)
143 {
144     if (arg1!=0 && ui->controlCheck){
145         ui->controlPeriodLineEdit->setDisabled(false);
146         ui->controlPriorityLineEdit->setDisabled(false);
147     } else {
148         ui->controlPeriodLineEdit->setDisabled(true);
149         ui->controlPriorityLineEdit->setDisabled(true);
```

```
150     }
151 }
152
153 void Dialt::on_actuatorEnable_stateChanged(int arg1)
154 {
155     if (arg1!=0 && ui->actuatorCheck){
156         ui->actuatorPeriodLineEdit->setDisabled(false);
157         ui->actuatorPriorityLineEdit->setDisabled(false);
158     } else {
159         ui->actuatorPeriodLineEdit->setDisabled(true);
160         ui->actuatorPriorityLineEdit->setDisabled(true);
161     }
162 }
163
164 void Dialt::on_serialEnable_stateChanged(int arg1)
165 {
166     if (arg1!=0){
167         ui->serialPeriodLineEdit->setDisabled(false);
168         ui->serialPriorityLineEdit->setDisabled(false);
169     } else {
170         ui->serialPeriodLineEdit->setDisabled(true);
171         ui->serialPriorityLineEdit->setDisabled(true);
172     }
173 }
174
175 void Dialt::on_buttonBox_accepted()
176 {
177     if(ui->sensorCheck->isChecked())
178     {
179         Config::reg.setSensorEnable(ui->sensorEnable->isChecked());
180         Config::reg.setSensorPeriod(ui->sensorPeriodLineEdit->text().toInt());
181         Config::reg.setSensorPriority(ui->sensorPriorityLineEdit->text().toInt());
182     }
183     if(ui->actuatorCheck->isChecked())
184     {
185         Config::reg.setActuatorEnable(ui->actuatorEnable->isChecked());
186         Config::reg.setActuatorPeriod(ui->actuatorPeriodLineEdit->text().toInt());
187         Config::reg.setActuatorPriority(ui->actuatorPriorityLineEdit->text().toInt());
188     }
189     if(ui->controlCheck->isChecked())
190     {
191         Config::reg.setControlEnable(ui->controlEnable->isChecked());
192         Config::reg.setControlPeriod(ui->controlPeriodLineEdit->text().toInt());
193         Config::reg.setControlPriority(ui->controlPriorityLineEdit->text().toInt());
194     }
195     Config::reg.setSerialPeriod(ui->serialPeriodLineEdit->text().toInt());
196     Config::reg.setSerialPriority(ui->serialPriorityLineEdit->text().toInt());
197     this->destroy();
198 }
199
200 void Dialt::on_buttonBox_rejected()
201 {
202     this->destroy();
203 }
```

- DialTorPlot.h

```
#ifndef DIALTORPLOT_H
#define DIALTORPLOT_H

#include <QDialog>

namespace Ui {
class DialTorPlot;
}

class DialTorPlot : public QDialog
{
    Q_OBJECT

public:
    explicit DialTorPlot(QWidget *parent = 0);
    ~DialTorPlot();

private slots:
    void on_vButton_clicked();

    void on_physButton_clicked();

    void on_vButton_toggled(bool checked);

    void on_physButton_toggled(bool checked);

    void on_buttonBox_rejected();

    void on_buttonBox_accepted();

private:
    Ui::DialTorPlot *ui;
};

#endif // DIALTORPLOT_H
```

- DialTorPlot.cpp

```
#include "dialtorplot.h"
#include "ui_dialtorplot.h"
#include "config.h"
#include "qstring.h"

DialTorPlot::DialTorPlot(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::DialTorPlot)
{
    Qt::WindowFlags flags(Qt::WindowTitleHint);
    this->setWindowFlags(flags);
    ui->setupUi(this);
    ui->xmaxLineEdit->setText(QString::number(Config::reg.getTorXMax()));
    ui->xminLineEdit->setText(QString::number(Config::reg.getTorXMin()));
    ui->ymaxLineEdit->setText(QString::number(Config::reg.getTorYMax()));
    ui->yminLineEdit->setText(QString::number(Config::reg.getTorYMin()));
```

```
17      ui->stepSizeXLineEdit->setText(QString::number(Config::reg.getTorXStep()));
18      ui->stepSizeYLineEdit->setText(QString::number(Config::reg.getTorYStep()));
19      int unit = Config::reg.getTorPlotUnit();
20      if(unit <= 1){
21          ui->vButton->setChecked(true);
22          if(unit == 0){
23              ui->vCheck->setChecked(true);
24          } else {
25              ui->mvCheck->setChecked(true);
26          }
27      } else {
28          ui->physButton->setChecked(true);
29          if(unit == 2){
30              ui->nmCheck->setChecked(true);
31          } else {
32              ui->nmmCheck->setChecked(true);
33          }
34      }
35  }
36
37  DialTorPlot::~DialTorPlot()
38  {
39      delete ui;
40  }
41
42  void DialTorPlot::on_vButton_clicked()
43  {
44      ui->nmCheck->setEnabled(false);
45      ui->nmmCheck->setEnabled(false);
46      ui->vCheck->setEnabled(true);
47      ui->mvCheck->setEnabled(true);
48  }
49
50  void DialTorPlot::on_physButton_clicked()
51  {
52      ui->nmCheck->setEnabled(true);
53      ui->nmmCheck->setEnabled(true);
54      ui->vCheck->setEnabled(false);
55      ui->mvCheck->setEnabled(false);
56  }
57
58  void DialTorPlot::on_vButton_toggled(bool checked)
59  {
60      if(checked == true){
61          ui->nmCheck->setEnabled(false);
62          ui->nmmCheck->setEnabled(false);
63          ui->vCheck->setEnabled(true);
64          ui->mvCheck->setEnabled(true);
65      }
66  }
67
68  void DialTorPlot::on_physButton_toggled(bool checked)
69  {
70      if(checked == true){
71          ui->nmCheck->setEnabled(true);
72          ui->nmmCheck->setEnabled(true);
73          ui->vCheck->setEnabled(false);
```

```
74          ui->mvCheck->setEnabled(false);
75      }
76  }
77
78  void DialTorPlot::on_buttonBox_rejected()
79  {
80      this->close();
81  }
82
83  void DialTorPlot::on_buttonBox_accepted()
84  {
85      if(ui->vButton->isChecked()){
86          if(ui->vCheck->isChecked()){
87              Config::reg.setTorPlotUnit(0);
88          } else {
89              Config::reg.setTorPlotUnit(1);
90          }
91      } else {
92          if(ui->nmCheck->isChecked()){
93              Config::reg.setTorPlotUnit(2);
94          } else {
95              Config::reg.setTorPlotUnit(3);
96          }
97      }
98      Config::reg.setTorXMax(ui->xmaxLineEdit->text().toDouble());
99      Config::reg.setTorYMax(ui->ymaxLineEdit->text().toDouble());
100     Config::reg.setTorXMin(ui->xminLineEdit->text().toDouble());
101     Config::reg.setTorYMin(ui->yminLineEdit->text().toDouble());
102     Config::reg.setTorXStep(ui->stepSizeXLineEdit->text().toDouble());
103     Config::reg.setTorYStep(ui->stepSizeYLineEdit->text().toDouble());
104     this->close();
105 }
```

- PlotData.h

```
1   #ifndef PLOTDATA_H
2   #define PLOTDATA_H
3   #include <qwt_series_data.h>
4
5   class PlotData : public QwtArraySeriesData<QPointF>
6   {
7   public:
8       PlotData();
9       static PlotData &instancePot();
10      static PlotData &instanceTor();
11
12      virtual QRectF boundingRect() const;
13
14      void append( const QPointF &point );
15
16      void clear();
17  };
18
19  #endif // PLOTDATA_H
```

- PlotData.cpp

```
#include "plotdata.h"

PlotData::PlotData()
{
}

QRectF PlotData::boundingRect() const
{
    if ( d_boundingRect.width() < 0.0 )
    d_boundingRect = qwtBoundingRect( *this );
    return d_boundingRect;
}

void PlotData::append( const QPointF &point )
{
    d_samples += point;
}

void PlotData::clear()
{
    d_samples.clear();
    d_samples.squeeze();
    d_boundingRect = QRectF( 0.0, 0.0, -1.0, -1.0 );
}
```

- Registers.h

```
#ifndef REGISTERS_H
#define REGISTERS_H

#define NSI     14
#define NB      5
#define ND      13
#define NI      1
#define NST     2

#include <QString>
#include <QtNetwork>

class Registers
{
public:

    Registers();
    ~Registers();

    short int getMonitoring ();
    short int getSensoring ();
    short int getTrajectory ();

    short int getSensorPriority();
    short int getSensorPeriod();
    bool getSensorEnable();

```

```
28      short int getControlPriority();
29      short int getControlPeriod();
30      bool getControlEnable();
31
32      short int getActuatorPriority();
33      short int getActuatorPeriod();
34      bool getActuatorEnable();
35
36      short int getSerialPriority();
37      short int getSerialPeriod();
38
39      short int getPosPlotUnit();
40      short int getTorPlotUnit();
41      double getPosXMax();
42      double getPosYMax();
43      double getTorXMax();
44      double getTorYMax();
45      double getPosXMin();
46      double getPosYMin();
47      double getTorXMin();
48      double getTorYMin();
49      double getPosXStep();
50      double getTorXStep();
51      double getPosYStep();
52      double getTorYStep();
53
54      short int getMachineId();
55      double getBaud();
56
57      bool getSerialOn();
58      bool getUdpOn();
59
60      int getPort();
61
62      QString getIp();
63      QString getSerialPort();
64
65      void setMonitoring (short int v);
66      void setSensoring (short int v);
67      void setTrajectory (short int v);
68
69      void setSensorPriority(short int v);
70      void setSensorPeriod(short int v);
71      void setSensorEnable(bool v);
72
73      void setControlPriority(short int v);
74      void setControlPeriod(short int v);
75      void setControlEnable(bool v);
76
77      void setActuatorPriority(short int v);
78      void setActuatorPeriod(short int v);
79      void setActuatorEnable(bool v);
80
81      void setSerialPriority(short int v);
82      void setSerialPeriod(short int v);
83
84      void setPosPlotUnit(short int v);
```

```
85      void setTorPlotUnit(short int v);
86      void setPosXMax(double v);
87      void setPosYMax(double v);
88      void setTorXMax(double v);
89      void setTorYMax(double v);
90      void setPosXMin(double v);
91      void setPosYMin(double v);
92      void setTorXMin(double v);
93      void setTorYMin(double v);
94      void setPosXStep(double v);
95      void setTorXStep(double v);
96      void setPosYStep(double v);
97      void setTorYStep(double v);
98
99      void setMachineId(short int v);
100     void setBaud(double v);
101
102     void setSerialOn(bool v);
103     void setUdpOn(bool v);
104
105     void setPort(int v);
106
107     void setIp(QString v);
108     void setSerialPort(QString v);
109
110 private:
111
112     //Short Int = 15
113     short int MONITORING;        //0
114     short int SENSORING;         //1
115     short int TRAJECTORY;        //2
116     short int SENSOR_PRIORITY;   //3
117     short int SENSOR_PERIOD;     //4
118     short int CONTROL_PRIORITY;  //5
119     short int CONTROL_PERIOD;    //6
120     short int ACTUATOR_PRIORITY; //7
121     short int ACTUATOR_PERIOD;   //8
122     short int SERIAL_PRIORITY;   //9
123     short int SERIAL_PERIOD;     //10
124     short int POS_PLOT_UNIT;     //11 (0 = v, 1 = mv, 2 = deg, 3 = rad)
125     short int TOR_PLOT_UNIT;     //12 (0 = v, 1 = mv, 2 = N.m, 3 = N.mm)
126     short int MACHINE_ID;        //13
127
128     //Bool = 3
129     bool ACTUATOR_ENABLE;    //0
130     bool CONTROL_ENABLE;     //1
131     bool SENSOR_ENABLE;      //2
132     bool SERIAL_ON;          //3
133     bool UDP_ON;             //4
134
135     //Double = 13
136     double POS_X_MAX;    //0
137     double POS_Y_MAX;    //1
138     double TOR_X_MAX;    //2
139     double TOR_Y_MAX;    //3
140     double POS_X_MIN;    //4
141     double POS_Y_MIN;    //5
```

```
142     double TOR_X_MIN;    //6
143     double TOR_Y_MIN;    //7
144     double POS_X_STEP;   //8
145     double TOR_X_STEP;   //9
146     double POS_Y_STEP;   //10
147     double TOR_Y_STEP;   //11
148     double BAUD;         //12
149 
150     //Int = 1
151     int PORT;           //0
152 
153     //String = 1
154     QString IP;             //0
155     QString SERIAL_PORT;    //1
156 };
157 
158 #endif // REGISTERS_H
```

- Registers.cpp

```
1   #include "registers.h"
2   
3   Registers::Registers() //inicializa registros na configuraÃ§Ã£o padrÃ£o
4   {
5       MONITORING = 0;
6       SENSORING = 0;
7       TRAJECTORY = 0;
8       SENSOR_PRIORITY = 100;
9       SENSOR_PERIOD = 2;
10      CONTROL_PRIORITY = 100;
11      CONTROL_PERIOD = 2;
12      ACTUATOR_PRIORITY = 100;
13      ACTUATOR_PERIOD = 2;
14      SERIAL_PRIORITY = 100;
15      SERIAL_PERIOD = 2;
16      POS_PLOT_UNIT = 0;
17      TOR_PLOT_UNIT = 0;
18  
19      SENSOR_ENABLE = 1;
20      CONTROL_ENABLE = 1;
21      ACTUATOR_ENABLE = 1;
22  
23      POS_X_MAX = 1;
24      POS_Y_MAX = 20;
25      TOR_X_MAX = 1;
26      TOR_Y_MAX = 2;
27      POS_X_MIN = 0;
28      POS_Y_MIN = -20;
29      TOR_X_MIN = 0;
30      TOR_Y_MIN = -2;
31      POS_X_STEP = 0.1;
32      TOR_X_STEP = 0.1;
33      POS_Y_STEP = 5;
34      TOR_Y_STEP = 0.5;
35  
36      MACHINE_ID = 1;
```

```
37      BAUD = 115200;
38
39      SERIAL_ON = true;
40      UDP_ON = false;
41
42      PORT = 45454;
43      IP = "127.0.0.1";
44      SERIAL_PORT = "";
45  }
46
47  Registers::~Registers()
48  {
49  }
50
51  /***********************************/
52  /*              GETTERS              */
53  /***********************************/
54
55  short int Registers::getMonitoring ()
56  {
57      return (MONITORING);
58  }
59
60  short int Registers::getSensoring ()
61  {
62      return (SENSORING);
63  }
64
65  short int Registers::getTrajectory ()
66  {
67      return (TRAJECTORY);
68  }
69
70  short int Registers::getSensorPriority()
71  {
72      return (SENSOR_PRIORITY);
73  }
74
75  short int Registers::getSensorPeriod()
76  {
77      return (SENSOR_PERIOD);
78  }
79
80  bool Registers::getSensorEnable()
81  {
82      return (SENSOR_ENABLE);
83  }
84
85  short int Registers::getControlPriority()
86  {
87      return (CONTROL_PRIORITY);
88  }
89
90  short int Registers::getControlPeriod()
91  {
92      return (CONTROL_PERIOD);
93  }
```

```
94
95  bool Registers::getControlEnable()
96  {
97      return (CONTROL_ENABLE);
98  }
99
100 short int Registers::getActuatorPriority()
101 {
102     return (ACTUATOR_PRIORITY);
103 }
104
105 short int Registers::getActuatorPeriod()
106 {
107     return (ACTUATOR_PERIOD);
108 }
109
110 bool Registers::getActuatorEnable()
111 {
112     return (ACTUATOR_ENABLE);
113 }
114
115 short int Registers::getSerialPriority()
116 {
117     return (SERIAL_PRIORITY);
118 }
119
120 short int Registers::getSerialPeriod()
121 {
122     return (SERIAL_PERIOD);
123 }
124
125 short int Registers::getPosPlotUnit(){
126     return(POS_PLOT_UNIT);
127 }
128
129 short int Registers::getTorPlotUnit(){
130     return(TOR_PLOT_UNIT);
131 }
132
133 double Registers::getPosXMax(){
134     return(POS_X_MAX);
135 }
136
137 double Registers::getPosYMax(){
138     return(POS_Y_MAX);
139 }
140
141 double Registers::getTorXMax(){
142     return(TOR_X_MAX);
143 }
144
145 double Registers::getTorYMax(){
146     return(TOR_Y_MAX);
147 }
148
149 double Registers::getPosXMin(){
150     return(POS_X_MIN);
```

```
151 }
152
153 double Registers::getPosYMin(){
154     return(POS_Y_MIN);
155 }
156
157 double Registers::getTorXMin(){
158     return(TOR_X_MIN);
159 }
160
161 double Registers::getTorYMin(){
162     return(TOR_Y_MIN);
163 }
164
165 double Registers::getPosXStep(){
166     return(POS_X_STEP);
167 }
168
169 double Registers::getTorXStep(){
170     return(TOR_X_STEP);
171 }
172
173 double Registers::getPosYStep(){
174     return(POS_Y_STEP);
175 }
176
177 double Registers::getTorYStep(){
178     return(TOR_Y_STEP);
179 }
180
181 short int Registers::getMachineId(){
182     return(MACHINE_ID);
183 }
184
185 double Registers::getBaud(){
186     return(BAUD);
187 }
188
189 bool Registers::getSerialOn(){
190     return(SERIAL_ON);
191 }
192
193 bool Registers::getUdpOn(){
194     return(UDP_ON);
195 }
196
197 int Registers::getPort(){
198     return(PORT);
199 }
200
201 QString Registers::getIp(){
202     return(IP);
203 }
204
205 QString Registers::getSerialPort(){
206     return(SERIAL_PORT);
207 }
```

```
208
209 /***********************************/
210 /*              SETTERS              */
211 /***********************************/
212
213 void Registers::setMonitoring (short int v){
214     MONITORING = v;
215 }
216
217 void Registers::setSensoring (short int v)
218 {
219     SENSORING = v;
220 }
221
222 void Registers::setTrajectory (short int v)
223 {
224     TRAJECTORY = v;
225 }
226
227 void Registers::setSensorPriority(short int v)
228 {
229     SENSOR_PRIORITY = v;
230 }
231
232 void Registers::setSensorPeriod(short int v)
233 {
234     SENSOR_PERIOD = v;
235 }
236
237 void Registers::setSensorEnable(bool v)
238 {
239     SENSOR_ENABLE = v;
240 }
241
242 void Registers::setControlPriority(short int v)
243 {
244     CONTROL_PRIORITY = v;
245 }
246
247 void Registers::setControlPeriod(short int v)
248 {
249     CONTROL_PERIOD = v;
250 }
251
252 void Registers::setControlEnable(bool v)
253 {
254     CONTROL_ENABLE = v;
255 }
256
257 void Registers::setActuatorPriority(short int v)
258 {
259     ACTUATOR_PRIORITY = v;
260 }
261
262 void Registers::setActuatorPeriod(short int v)
263 {
264     ACTUATOR_PERIOD = v;
```

```
265 }
266
267 void Registers::setActuatorEnable(bool v)
268 {
269     ACTUATOR_ENABLE = v;
270 }
271
272 void Registers::setSerialPriority(short int v)
273 {
274     SERIAL_PRIORITY = v;
275 }
276
277 void Registers::setSerialPeriod(short int v)
278 {
279     SERIAL_PERIOD = v;
280 }
281
282 void Registers::setPosPlotUnit(short int v){
283     POS_PLOT_UNIT = v;
284 }
285
286 void Registers::setTorPlotUnit(short int v){
287     TOR_PLOT_UNIT = v;
288 }
289
290 void Registers::setPosXMax(double v){
291     POS_X_MAX = v;
292 }
293
294 void Registers::setPosYMax(double v){
295     POS_Y_MAX = v;
296 }
297
298 void Registers::setTorXMax(double v){
299     TOR_X_MAX = v;
300 }
301
302 void Registers::setTorYMax(double v){
303     TOR_Y_MAX = v;
304 }
305
306 void Registers::setPosXMin(double v){
307     POS_X_MIN = v;
308 }
309
310 void Registers::setPosYMin(double v){
311     POS_Y_MIN = v;
312 }
313
314 void Registers::setTorXMin(double v){
315     TOR_X_MIN = v;
316 }
317
318 void Registers::setTorYMin(double v){
319     TOR_Y_MIN = v;
320 }
321
```

```
322 void Registers::setPosXStep(double v){
323     POS_X_STEP = v;
324 }
325 
326 void Registers::setTorXStep(double v){
327     TOR_X_STEP = v;
328 }
329 
330 void Registers::setPosYStep(double v){
331     POS_Y_STEP = v;
332 }
333 
334 void Registers::setTorYStep(double v){
335     TOR_Y_STEP = v;
336 }
337 
338 void Registers::setMachineId(short int v){
339     MACHINE_ID = v;
340 }
341 
342 void Registers::setBaud(double v){
343     BAUD = v;
344 }
345 
346 void Registers::setSerialOn(bool v){
347     SERIAL_ON = v;
348 }
349 
350 void Registers::setUdpOn(bool v){
351     UDP_ON = v;
352 }
353 
354 void Registers::setPort(int v){
355     PORT = v;
356 }
357 
358 void Registers::setIp(QString v){
359     IP = v;
360 }
361 
362 void Registers::setSerialPort(QString v){
363     SERIAL_PORT = v;
364 }
```

- SequencePlot.h

```
1  #ifndef SEQUENCEPLOT_H
2  #define SEQUENCEPLOT_H
3  
4  #include <qwt_plot.h>
5  #include <QWidget>
6  
7  class QwtPlotCurve;
8  class QwtPlotDirectPainter;
9  
10 namespace Ui{
```

```
11      class SequencePlot;
12  }
13
14  class SequencePlot : public QwtPlot
15  {
16      Q_OBJECT
17
18  public:
19      SequencePlot(QWidget *parent = NULL);
20      virtual ~SequencePlot();
21      QPointF dataSample(int i);
22      void appendPoint(const QPointF &);
23      void clearPoints();
24      int size();
25
26  public Q_SLOTS:
27      void showSymbols(bool);
28
29  private:
30      QwtPlotCurve *curve;
31      QwtPlotDirectPainter *dPainter;
32  };
33
34  #endif // SEQUENCEPLOT_H
```

- SequencePlot.cpp

```
1   #include "sequenceplot.h"
2   #include <qwt_plot.h>
3   #include <qwt_plot_canvas.h>
4   #include <qwt_plot_curve.h>
5   #include <qwt_symbol.h>
6   #include <qwt_plot_directpainter.h>
7   #include <qwt_painter.h>
8   #include <qpaintengine.h>
9   #include "plotdata.h"
10
11  SequencePlot::SequencePlot(QWidget *parent):QwtPlot(parent),curve(NULL)
12  {
13      dPainter = new QwtPlotDirectPainter(this);
14
15      if ( QwtPainter::isX11GraphicsSystem() )
16      {
17  #if QT_VERSION < 0x050000
18          canvas()->setAttribute( Qt::WA_PaintOutsidePaintEvent, true );
19  #endif
20          canvas()->setAttribute( Qt::WA_PaintOnScreen, true );
21      }
22
23      curve = new QwtPlotCurve( "Test␣Curve" );
24      curve->setData( new PlotData() );
25      showSymbols( true );
26
27      curve->attach( this );
28
29      setAutoReplot( false );
```

```
30  }
31
32  SequencePlot::~SequencePlot()
33  {
34      delete curve;
35  }
36
37  QPointF SequencePlot::dataSample(int i)
38  {
39      return(curve->data()->sample(i));
40  }
41
42  void SequencePlot::appendPoint(const QPointF &point)
43  {
44      PlotData *data = static_cast<PlotData *>( curve->data() );
45      data->append( point );
46
47      const bool doClip = !canvas()->testAttribute( Qt::WA_PaintOnScreen );
48      if ( doClip )
49      {
50          const QwtScaleMap xMap = canvasMap( curve->xAxis() );
51          const QwtScaleMap yMap = canvasMap( curve->yAxis() );
52
53          QRegion clipRegion;
54
55          const QSize symbolSize = curve->symbol()->size();
56          QRect r( 0, 0, symbolSize.width() + 2, symbolSize.height() + 2 );
57
58          const QPointF center =
59              QwtScaleMap::transform( xMap, yMap, point );
60          r.moveCenter( center.toPoint() );
61          clipRegion += r;
62
63          dPainter->setClipRegion( clipRegion );
64      }
65      dPainter->drawSeries( curve, data->size() - 1, data->size() - 1 );
66  }
67
68  void SequencePlot::clearPoints()
69  {
70      PlotData *data = static_cast<PlotData *>(curve->data());
71      data->clear();
72      replot();
73  }
74
75  void SequencePlot::showSymbols( bool on )
76  {
77      if ( on )
78      {
79          curve->setStyle( QwtPlotCurve::NoCurve );
80          curve->setSymbol( new QwtSymbol( QwtSymbol::Star1,
81              Qt::NoBrush, QPen( Qt::black ), QSize( 2, 2 ) ) );
82      }
83      else
84      {
85          curve->setPen( Qt::white );
86          curve->setStyle( QwtPlotCurve::Dots );
```

```
87          curve->setSymbol( NULL );
88      }
89
90      replot();
91  }
92
93  int SequencePlot::size()
94  {
95      return(curve->dataSize());
96  }
```

- SerialComm.h

```
1   #ifndef SERIALCOMM_H
2   #define SERIALCOMM_H
3
4   #include <QObject>
5   #include <sstream>
6   #include <string.h>
7   #include <QtSerialPort>
8   #include <QTimer>
9   #include <QtSerialPort/QSerialPortInfo>
10  #include "config.h"
11
12  #pragma pack(push, before)
13  #pragma pack(1)
14  struct CommMessage
15  {
16      char                        start;          //1 byte
17      char                        id;             //1 byte
18      char                        command[2];     //2 bytes
19      unsigned char               datasize;       //1 byte
20      char                        *data;          //atÃ© 255 bytes
21      char                        checksum[2];    //2 bytes
22  };
23  #pragma pack(pop, before)
24
25  class SerialComm : public QObject
26  {
27      Q_OBJECT
28
29  public:
30      SerialComm(QObject *parent = NULL);
31
32  signals:
33      void show_message(QString msg);
34      void show_error(QString error);
35      void finished();
36
37  public slots:
38      void process();
39      CommMessage* readMsg();
40      QString test();
41
42  private slots:
43      void sendQuery(int msgCode);
```

```
44      void config();
45      void start();
46      void stop();
47      void pause(bool paused);
48      void finishComm();
49
50      void hasPendingMessage();
51      int validateMessage(CommMessage *msg);
52  private:
53      bool active;
54      bool ready;
55      QSerialPort *portNo;
56      char* IdList;
57      char* thisId;
58      QTimer* timer;
59  };
60
61  #endif // SERIALCOMM_H
```

- SerialComm.cpp

```
1   #include "serialcomm.h"
2
3   SerialComm::SerialComm(QObject *parent) :
4       QObject(parent)
5   {
6       active = false;
7       ready = false;
8       portNo = NULL;
9   }
10
11  CommMessage* SerialComm::readMsg()
12  {
13      CommMessage *msg;
14      msg = (CommMessage *)portNo->readAll().data_ptr();
15      return (msg);
16  }
17
18  QString SerialComm::test()
19  {
20      std::stringstream ss;
21      ss<<QThread::currentThreadId();
22      QString str;
23      str = QString::fromStdString(ss.str());
24      return("thread␣ID␣" + str);
25  }
26
27  void SerialComm::process()
28  {
29      if(active)
30      {
31          if(!portNo->isRequestToSend() && !timer->isActive())
32          {
33              if(ready)
34              {
35                  CommMessage *incoming = (CommMessage *)portNo->readAll().data_ptr();
```

```
36                  if(validateMessage(incoming) == 0)
37                  {
38                      //TRATA SINAL RECEBIDO
39                  }
40              }
41          }
42      }
43  //    std::stringstream ss;
44  //    ss<<QThread::currentThreadId();
45  //    QString str;
46  //    str = QString::fromStdString(ss.str());
47  //    emit show_message("thread ID " + str);
48  }
49
50  void SerialComm::sendQuery(int msgCode)
51  {
52      char buf[5];
53      char *msg;
54      msg = (char *)malloc(262);
55      char *datasize;
56      char *aux = msg;
57      *aux = ':';
58      aux++;
59      *aux = 'A';
60      aux++;
61      switch(msgCode)
62      {
63      case(0):
64          *aux = 'S';
65          aux++;
66          *aux = 'T';
67          aux++;
68          datasize = aux;
69          aux += sizeof(CommMessage::datasize);
70          *aux = '|';
71          aux++;
72          strcpy(aux, itoa(Config::reg.getSensorEnable(), buf, 10));
73          aux += strlen(buf);
74          *aux = '|';
75          aux++;
76          strcpy(aux, itoa(Config::reg.getSensorPeriod(), buf, 10));
77          aux += strlen(buf);
78          *aux = '|';
79          aux++;
80          strcpy(aux, itoa(Config::reg.getSensorPriority(), buf, 10));
81          aux += strlen(buf);
82          *aux = '|';
83          aux++;
84          strcpy(aux, itoa(Config::reg.getControlEnable(), buf, 10));
85          aux += strlen(buf);
86          *aux = '|';
87          aux++;
88          strcpy(aux, itoa(Config::reg.getControlPeriod(), buf, 10));
89          aux += strlen(buf);
90          *aux = '|';
91          aux++;
92          strcpy(aux, itoa(Config::reg.getControlPriority(), buf, 10));
```

```
93          aux += strlen(buf);
94          *aux = '|';
95          aux++;
96          strcpy(aux, itoa(Config::reg.getActuatorEnable(), buf, 10));
97          aux += strlen(buf);
98          *aux = '|';
99          aux++;
100         strcpy(aux, itoa(Config::reg.getActuatorPeriod(), buf, 10));
101         aux += strlen(buf);
102         *aux = '|';
103         aux++;
104         strcpy(aux, itoa(Config::reg.getActuatorPriority(), buf, 10));
105         aux += strlen(buf);
106         *aux = '|';
107         aux++;
108         strcpy(aux, itoa(Config::reg.getSerialPeriod(), buf, 10));
109         aux += strlen(buf);
110         *aux = '|';
111         aux++;
112         strcpy(aux, itoa(Config::reg.getSerialPriority(), buf, 10));
113         aux += strlen(buf);
114         *aux = '|';
115         aux++;
116         *datasize = (char)(aux - (datasize + sizeof(CommMessage::datasize)));
117         break;
118     case(1):
119         break;
120     case(2):
121         break;
122     case(3):
123         break;
124     case(4):
125         break;
126     case(5):
127         break;
128     case(6):
129         break;
130     case(7):
131         break;
132     case(8):
133         break;
134     case(9):
135         break;
136     case(10):
137         break;
138     }
139     *aux = '0';
140     aux++;
141     *aux = '0';
142     aux++;
143     *aux = '\0';
144 //    emit(show_error((QString)(msg)));
145     portNo->write((char *)&msg, strlen(msg) + 1);
146     if(!portNo->waitForBytesWritten(10))
147         emit(show_error("Falhou em escrever a mensagem!"));
148 }
149
```

```
150 void SerialComm::start()
151 {
152     sendQuery(3);
153     active = true;
154 }
155 
156 void SerialComm::stop()
157 {
158     sendQuery(4);
159     active = false;
160 }
161 
162 void SerialComm::finishComm()
163 {
164 //    if(portNo->isOpen())
165 //        portNo->close();
166 //    portNo->deleteLater();
167 //    free(IdList);
168 //    free(thisId);
169     emit(show_message("Emitiu␣sinal␣2"));
170 //    emit(finished());
171 //    this->deleteLater();
172 }
173 
174 void SerialComm::pause(bool paused)
175 {
176     active = false;
177     if(paused)
178     {
179         if(portNo->isOpen())
180         {
181             portNo->clear();
182             portNo->close();
183         }
184     }
185     else
186         if(!portNo->isOpen())
187             portNo->open(QIODevice::ReadWrite);
188 }
189 
190 void SerialComm::config()
191 {
192     emit(show_message("Iniciando␣configuraÃ§Ãfo␣da␣porta␣serial..."));
193     if (portNo != NULL)
194     {
195         if (portNo->isOpen())
196             portNo->close();
197         free(portNo);
198     }
199     foreach(const QSerialPortInfo info, QSerialPortInfo::availablePorts())
200     {
201         if(QString::compare(info.portName(), Config::reg.getSerialPort()) == 0)
202             portNo = new QSerialPort(info.portName());
203     }
204     if (portNo != NULL)
205     {
206         if(!portNo->open(QIODevice::ReadWrite))
```

```
207         {
208             emit(show_error("NÃ£o␣foi␣possÃvel␣abrir␣a␣porta␣serial"));
209             return;
210         }
211         if(!portNo->setBaudRate((qint32)Config::reg.getBaud()))
212         {
213             emit(show_error("NÃ£o␣foi␣possÃvel␣ajustar␣o␣Baud"));
214             return;
215         }
216         if(!portNo->setDataBits(QSerialPort::Data8))
217         {
218             emit(show_error("NÃ£o␣foi␣possÃvel␣ajustar␣o␣tamanho␣dos␣dados"));
219             return;
220         }
221         if(!portNo->setParity(QSerialPort::NoParity))
222         {
223             emit(show_error("NÃ£o␣foi␣possÃvel␣ajustar␣a␣paridade␣dos␣bits"));
224             return;
225         }
226         if(!portNo->setStopBits(QSerialPort::OneStop))
227         {
228             emit(show_error("NÃ£o␣foi␣possÃvel␣ajustar␣o␣stop␣bit"));
229             return;
230         }
231         if(!portNo->setFlowControl(QSerialPort::NoFlowControl))
232         {
233             emit(show_error("NÃ£o␣foi␣possÃvel␣ajustar␣o␣controle␣de␣fluxo"));
234             return;
235         }
236         QByteArray temp = QString::number(Config::reg.getMachineId()).toLocal8Bit();
237         thisId = temp.data();
238     }
239     else
240         emit(show_error("NÃ¡o␣foi␣possÃvel␣encontrar␣a␣porta␣serial"));
241     emit(show_message("Porta␣serial␣configurada␣com␣sucesso!"));
242     return;
243 }
244 
245 void SerialComm::hasPendingMessage()
246 {
247     ready = true;
248 }
249 
250 int SerialComm::validateMessage(CommMessage *msg)
251 {
252     if(msg->start == ':')
253         if(msg->id == *thisId)
254             switch(msg->command[0])
255             {
256             case ('S'):
257             case ('O'):
258             case ('H'):
259             case ('C'):
260                 return 0;
261             default:
262                 return -1;
263             }
```

```
264     return -1;
265 }
```

- ThreadStarter.h

```
1   #ifndef SAMPLINGTHREAD_H
2   #define SAMPLINGTHREAD_H
3   #include "qwt_sampling_thread.h"
4   #include "sequenceplot.h"
5   #include "udpcomm.h"
6   #include "serialcomm.h"
7   #include <QThread>
8   #include <QTimer>
9   #include "config.h"
10
11  class threadStarter : public QThread
12  {
13      Q_OBJECT
14
15  public:
16      threadStarter(QObject *parent = NULL);
17      ~threadStarter();
18      void run();
19
20  private slots:
21      //da comm para a GUI
22      void emitError(QString error);
23      void emitMessage(QString msg);
24
25      //da GUI para a comm
26      void emitUpdate();
27      void emitStart();
28      void emitStop();
29      void emitPause(bool paused);
30      void emitSendRequest(int msgCode);
31      void emitPlotPoint(double time, double tor, double pos);
32      void emitFinish();
33
34  signals:
35      void showError(QString error);
36      void showMsg(QString msg);
37      void updateComm();
38      void startComm();
39      void stopComm();
40      void pauseComm(bool paused);
41      void finishComm();
42      void sendRequest(int msgCode);
43      void plotPoint(double time, double tor, double pos);
44  };
45
46  #endif // SAMPLINGTHREAD_H
```

- ThreadStarter.cpp

```
1   #include "threadStarter.h"
```

```
2  #include <qmath.h>
3
4  threadStarter::threadStarter(QObject *parent):QThread(parent)
5  {
6  }
7
8  threadStarter::~threadStarter()
9  {
10     this->deleteLater();
11 }
12
13 void threadStarter::run()
14 {
15     QTimer *timer = new QTimer();
16     if (Config::reg.getSerialOn())
17     {
18         SerialComm *serialmgr = new SerialComm();
19         connect(timer, SIGNAL(timeout()), serialmgr, SLOT(process()));
20
21         //sinais da comm. para a GUI
22         connect(serialmgr, SIGNAL(finished()), this, SLOT(quit()));
23 //        connect(serialmgr, SIGNAL(finished()), serialmgr, SLOT(quit()));
24         connect(serialmgr, SIGNAL(finished()), this, SLOT(deleteLater()));
25         connect(serialmgr, SIGNAL(show_error(QString)), this, SLOT(emitError(QString)));
26         connect(serialmgr, SIGNAL(show_message(QString)), this, SLOT(emitMessage(QString)));
27
28         //sinais da GUI para a comm.
29         connect(this, SIGNAL(updateComm()), serialmgr, SLOT(config()));
30         connect(this, SIGNAL(startComm()), serialmgr, SLOT(start()));
31         connect(this, SIGNAL(pauseComm(bool)), serialmgr, SLOT(pause(bool)));
32         connect(this, SIGNAL(stopComm()), serialmgr, SLOT(stop()));
33         connect(this, SIGNAL(sendRequest(int)), serialmgr, SLOT(sendQuery(int)));
34         connect(this, SIGNAL(finishComm()), serialmgr, SLOT(finishComm()));
35     }
36     else if (Config::reg.getUdpOn())
37     {
38         UdpComm udpmgr;
39         connect(timer, SIGNAL(timeout()), &udpmgr, SLOT(process()));
40         connect(&udpmgr, SIGNAL(finished()), this, SLOT(quit()));
41         connect(&udpmgr, SIGNAL(finished()), &udpmgr, SLOT(quit()));
42         connect(&udpmgr, SIGNAL(finished()), this, SLOT(deleteLater()));
43     }
44     timer->start(100);
45     exec();
46 }
47
48 void threadStarter::emitPlotPoint(double time, double tor, double pos)
49 {
50     emit(plotPoint(time, tor, pos));
51 }
52
53 void threadStarter::emitError(QString error)
54 {
55     emit(showError(error));
56 }
57
58 void threadStarter::emitMessage(QString msg)
```

```
59  {
60      emit(showMsg(msg));
61  }
62
63  void threadStarter::emitUpdate()
64  {
65      emit(updateComm());
66  }
67
68  void threadStarter::emitStart()
69  {
70      emit(startComm());
71  }
72
73  void threadStarter::emitStop()
74  {
75      emit(stopComm());
76  }
77
78  void threadStarter::emitPause(bool paused)
79  {
80      emit(pauseComm(paused));
81  }
82
83  void threadStarter::emitSendRequest(int msgCode)
84  {
85      emit(sendRequest(msgCode));
86  }
87
88  void threadStarter::emitFinish()
89  {
90      emit(finishComm());
91  }
92
93  //void threadStarter::sample( double elapsed )
94  //{
95  //    Q_UNUSED(elapsed);
96  //    if(active)
97  //    {
98  //        /************************/
99  //        /*   Conexao por SERIAL  */
100 //        /************************/
101 //        else if(Config::reg.getSerialOn())
102 //        {
103
104 //            emit(showMsg(serialmgr->test()));
105 //        }
106 //    }
107
108 //}
109
110 //void threadStarter::pause(bool running)
111 //{
112 //    active = !running;
113 //    else if (Config::reg.getSerialOn())
114 //    {
115
```

```
116 //    }
117 //}
118
119 //void threadStarter::plot(QStringList query)
120 //{
121 //    x = time->elapsed();
122 //    const QPointF p(x/1000.00, query[1].toDouble()/1000.00);
123 //    const QPointF t(x/1000.00, query[2].toDouble()/1000.00);
124 //    emit pointAppendedPot(p);
125 //    emit pointAppendedExt(t);
126 //}
```

ANEXO A – CONTROLE DO DRIVER EPOS 2

EPOS

Positioning Controller

Application Note
"Position Regulation with Feed Forward"

Edition May 2008

EPOS 24/1, EPOS 24/5, EPOS 70/10, MCD EPOS
Firmware version 2020h or higher

Introduction

EPOS is a modular-designed digital positioning system suitable for DC and EC motors with incremental encoder. The performance range of these compact positioning controllers starts at a few Watt and goes up to 700 Watt (1750 Watt peak).

A variety of operating modes allows all kinds of drive and automation systems to be flexibly assembled using positioning, speed and current regulation. The built-in CANopen interface allows networking to multiple axis drives and online commanding by CAN bus master units.

In addition to the standard EPOS PID position control feed forward compensation is available.
This feed forward compensation provides good results in application with higher load inertia and accelerations and/or in applications with considerable speed dependent load (example: friction).

Objectives

This application note explains the functionality of the built-in acceleration and velocity feed forward. Advantages compared to simple PID control are shown.

References and Required Tool

The latest editions of maxon motor documents and tools are freely available at http://www.maxonmotor.com category «Service & Downloads».

Document	Suitable order number for EPOS Positioning Controller
EPOS Firmware Specification	280937, 302267, 302287, 317270, 275512, 300583

Tool	
EPOS Studio Version 1.30 or higher	280937, 302267, 302287, 317270, 275512, 347717, 300583

Controller architecture

For the position control a discrete PID controller with anti windup, acceleration feed forward and velocity feed forward was implemented using a digital signal processor (DSP) where the sample time (T_S) was taken at 1ms, much smaller than the mechanical time constant of a typical drive system. The structure is shown in the following figure.

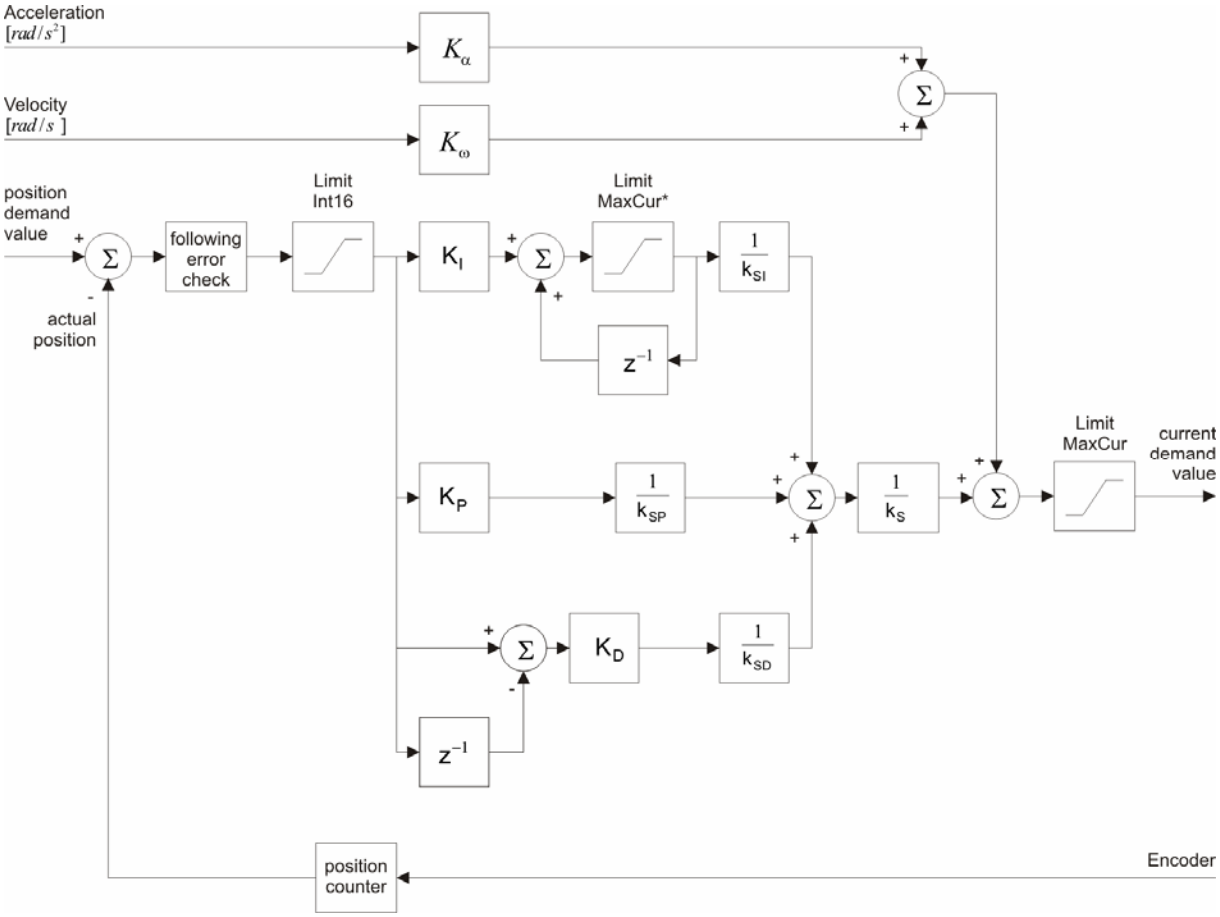


Figure 1: Controller architecture

Constants

Constant	EPOS 24/1	EPOS 24/5	EPOS 70/10
k_{SP}	$2^5 = 32$	$2^3 = 8$	$2^2 = 4$
k_{SI}	$2^8 = 256$	$2^6 = 64$	$2^5 = 32$
k_{SD}	$2^2 = 4$	$2^0 = 1$	$2^0 = 1$

Encoder pulse number	k_s
0 - 200	$2^0 = 1$
201 - 1000	$2^1 = 2$
1001 - 5000	$2^2 = 4$
5001 - 7500	$2^3 = 8$

Object dictionary entries

Symbol	Name	Index	Sub-Index
K_P	Position Regulator P-Gain	0x60FB	0x01
K_I	Position Regulator I-Gain	0x60FB	0x02
K_D	Position Regulator D-Gain	0x60FB	0x03
K_ω	Velocity Feedforward Factor	0x60FB	0x04
K_α	Acceleration Feedforward Factor	0x60FB	0x05

Units

Acceleration: $\left[\frac{rad}{s^2}\right]$

Velocity: $\left[\frac{rad}{s}\right]$

Position: $[qc] = \left[\frac{1 \cdot (turn)}{Encoder\_pulse\_number \cdot 4}\right]$ (quadrature counts)

Current: $[mA]$

K_P: $\left[\frac{mA}{k_{SP} \cdot k_S \cdot qc}\right]$

K_I: $\left[\frac{mA}{k_{SI} \cdot k_S \cdot qc \cdot T_S}\right]$

K_D: $\left[\frac{mA \cdot T_S}{k_{SD} \cdot k_S \cdot qc}\right]$

K_α: $\left[\frac{A}{rad/s^2} \cdot 10^{-6}\right]$ (EPOS 24/5, EPOS 70/10 and MCD EPOS 60W)

K_α: $\left[\frac{A}{rad/s^2} \cdot 10^{-7}\right]$ (EPOS 24/1)

K_ω: $\left[\frac{\mu A}{rad/s}\right]$ (EPOS 24/1, EPOS 24/5, EPOS 70/10 and MCD EPOS 60W)

Operation Modes

The acceleration and velocity feed forward take effect in Profile Position Mode and Homing Mode. There is no influence to all the other operation modes like Position Mode, Profile Velocity Mode, Velocity Mode and Current Mode.

Purpose of K_α (Acceleration Feedforward Factor)

K_α provides additional current in cases of high acceleration and/or high load inertias.

Purpose of K_ω (Velocity Feedforward Factor)

K_ω provides additional current in cases, where the load increases with speed, e.g. speed dependent friction. The load is assumed to increase linear with speed.

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 resolution
Input Modes	Single-ended, Differential
Input Ranges	±10V, ±5V, ±2.5V, ±1.25V, ±0-.625V, 0-10V, 0-5V, 0-2.5V, 0-1.25V
Max Sample Rate	100KHz
Nonlinearity	±3LSB with no missing codes
On-board FIFO	512 samples with programmable threshold
Calibration	Software initiated autocalibration
Analog Outputs	
Number of Outputs	4 12-bit resolution
Output Ranges	±5V, 0-5V
Output Current	±5mA max per channel
Settling Time	6µS max to 0.01%
Relative Accuracy	±1 LSB
Digital I/O	
Number of 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
General	
Counter / Timers	1 - 32-bit & 1 - 16-bit
Clock Source	10MHz clock or external signal
Power Supply	+5VDC±10% at 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°C to +85°C.

ORDERING INFORMATION

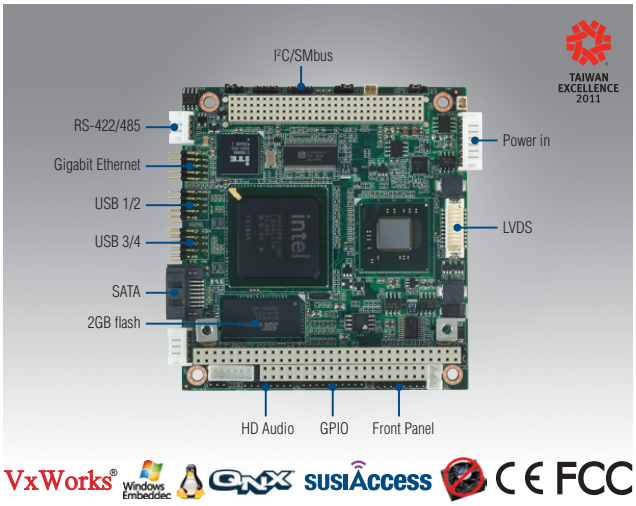
Part No.	Description
DMM-16-AT	Diamond-MM Autocalibrating 16-ch 16-bit A/D + 4-ch 12-bit D/A Extended Temperature
DMM-16-NA-AT	Diamond-MM Autocalibrating 16-ch 16-bit A/D only Extended Temperature

FOR MORE INFORMATION

Diamond Systems Corporation
555 Ellis Street
Mountain View, CA 94043
Tel: 650-810-2500
Fax: 650-810-2525
techinfo@diamondsystems.com

PCM-3362

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



VxWorks Windows Embedded Linux QNX SUSIAccess CE FC

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

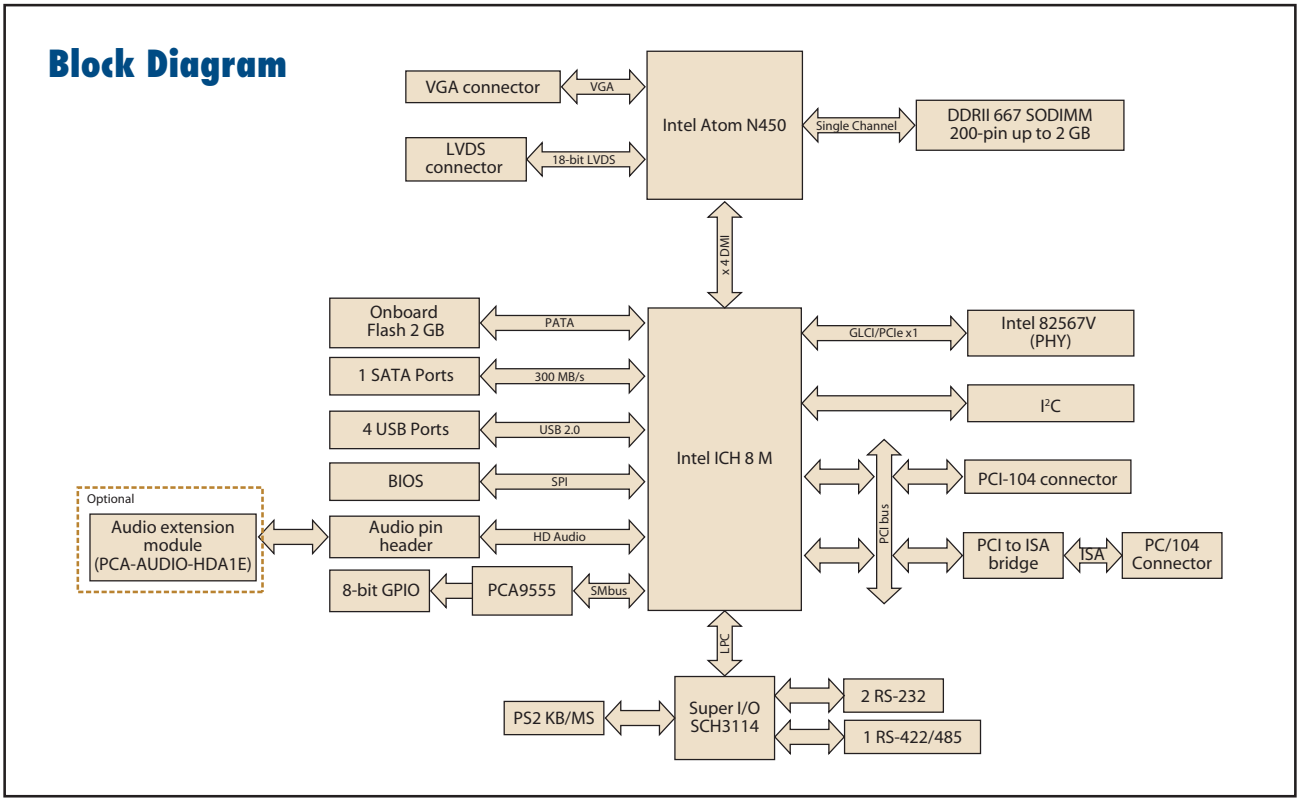
Software APIs: Watchdog, I²C, Brightness, H/W Monitor, GPIO, Backlight On/Off

Utilities: Monitoring

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 SODIMM
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 Watts)
	Power Consumption (Max, test in HCT)	2.37 A @ +5 V, 7 mA @ +12 V (11.93 Watts)
	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	2070010323	XPE WES2009 MUI24
QNX		6.5
Linux		Ubuntu 10.04.1
VxWorks		6.8
Software API	205E362000	SUSI 3.0 SW API for PCM-3362 B:20091015 XP

maxon motor control **EPOS2 Positioning Controller**

Hardware Reference **Edition December 2012**

EPOS2 24/5

Positioning Controller

Hardware Reference

Document ID: rel3346

maxon motor ag Brünigstrasse 220 P.O.Box 263 CH-6072 Sachseln Phone +41 41 666 15 00 Fax +41 41 666 16 50 www.maxonmotor.com

3 Technical Data

3.1 Electrical Data

Rating	
Nominal power supply voltage V_{CC}	11…24 VDC
Nominal logic supply voltage V_C (optional)	11…24 VDC
Absolute minimum supply voltage	10 VDC
Absolute max. supply voltage	28 VDC
Max. output voltage	0.9 • V_{CC}
Max. output current I_{max} (<1sec)	10 A
Continuous output current I_{cont}	5 A
Switching frequency	50 kHz
Max. efficiency	92%
Sample rate PI – current controller	10 kHz
Sample rate PI – speed controller	1 kHz
Sample rate PID – positioning controller	1 kHz
Max. speed @ sinusoidal commutation (motors with 1 pole pair)	25 000 rpm
Max. speed @ block commutation (motors with 1 pole pair)	100 000 rpm
Built-in motor choke per phase	15 µH / 5 A

Table 3-3 Electrical Data – Rating

Inputs	
Hall sensor signals	Hall sensor 1, Hall sensor 2 and Hall sensor 3 for Hall effect sensor ICs (Schmitt trigger with open collector output)
Encoder signals	A, A\, B, B\, I, I\ (max. 5 MHz) internal line receiver EIA RS422 Standard
Digital Input 1 (“General Purpose”)	+3…+24 VDC (Ri = 8 kΩ)
Digital Input 2 (“General Purpose”)	+3…+24 VDC (Ri = 8 kΩ)
Digital Input 3 (“General Purpose”)	+3…+24 VDC (Ri = 8 kΩ)
Digital Input 4 (“Home Switch”)	+9…+24 VDC (Ri = 4 kΩ)
Digital Input 5 (“Positive Limit Switch”)	+9…+24 VDC (Ri = 4 kΩ)
Digital Input 6 (“Negative Limit Switch”)	+9…+24 VDC (Ri = 4 kΩ)
Analog Input 1	resolution 12-bit 0…+5 V (Ri = 47 kΩ)
Analog Input 2	resolution 12-bit 0…+5 V (Ri = 47 kΩ)
CAN ID (CAN identification)	ID 1…127 configurable via DIP switch or software

Table 3-4 Electrical Data – Inputs

Outputs	
Digital Output 1 (“General Purpose”), open drain	max. 24 VDC (I_L <100 mA)
Digital Output 2 (“General Purpose”), open drain	max. 24 VDC (I_L <100 mA)
Digital Output 3 (“General Purpose”), open drain	max. 24 VDC (I_L <100 mA)
Digital Output 4 (“Brake”) open drain	max. 24 VDC (I_L <1000 mA)

Table 3-5 Electrical Data – Outputs

Voltage Outputs	
Encoder supply voltage	+5 VDC (I_L <100 mA)
Hall sensors supply voltage	+5 VDC (I_L <30 mA)
Auxiliary output voltage	V_{cc} (I_L <1300 mA)

Table 3-6 Electrical Data – Voltage Outputs

Motor Connections	
maxon EC motor	**maxon DC motor**
Motor winding 1	+ Motor
Motor winding 2	- Motor
Motor winding 3	

Table 3-7 Electrical Data – Motor Connections

Interfaces		
RS232	RxD; TxD	max. 115 200 bit/s
USB 2.0	Data+; Data-	max.12 Mbit/s
CAN 1	CAN_H (high); CAN_L (low)	max.1 Mbit/s
CAN 2	CAN_H (high); CAN_L (low)	max.1 Mbit/s

Table 3-8 Electrical Data – Interfaces

Status Indicators	
Operation	green LED
Error	red LED

Table 3-9 Electrical Data – LEDs

HEDL-550x/554x, HEDL-560x/564x, HEDL-9000/9100, HEDL-9040/9140/ 92xx

Encoder Line Drivers

Data Sheet

Description

Line Drivers are available for the HEDS-55xx/56xx series and the HEDS-9000/9100/9200/9040/9140 series encoders. The line driver offers enhanced performance when the encoder is used in noisy environments, or when it is required to drive long distances.

The encoder line driver utilizes an industry standard line driver IC, AM26C31Q, which provides complementary outputs for each encoder channel. Thus, the output of the line driver encoder is A, $\overline{A}$, B, $\overline{B}$ and I/$\overline{I}$ for three channel versions. Suggested line receivers are 26LS32 and 26LS33.

For additional information, please refer to:

HEDS-5500/5540/5600/5640 data sheet,

HEDS-90x0/91x0/92x0 data sheets,

HEDS-9000 series extended resolution data sheet, and AM26C31Q data sheet.

Features

- Available on Both Encoder Modules (HEDS-9000 Series) and Encoder Kit Housing (HEDS-5500 Series)
- Complementary Outputs
- Industry Standard Line Driver IC
- Single 5 V Supply
- Onboard Bypass Capacitor

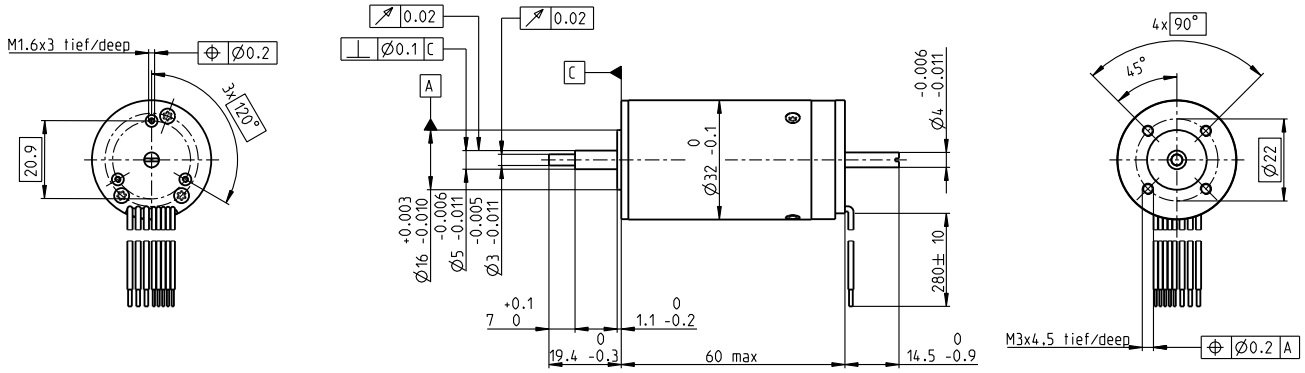
Note: Avago Technologies encoders are not recommended for use in safety critical applications. Eg. ABS braking systems, power steering, life support systems and critical care medical equipment. Please contact sales representative if more clarification is needed.

Device Characteristics

Parameter	Characteristic	Notes
Termination	10 conductor ribbon cable with10 position IDC connector	See pinout
Electrical Outputs	Complementary outputs: A, $\overline{A}$, B, $\overline{B}$, I, $\overline{I}$	I and $\overline{I}$ available only on three channel encoders
Line Driver Components	AM26C31Q line driver IC, decoupling capacitor on PC board.	
Operating Temperature Range	-40°C to 100° C	100° C Series
Storage Temperature	-40°C to 100° C	100° C Series

ESD WARNING: *NORMAL HANDLING PRECAUTIONS SHOULD BE TAKEN TO AVOID STATIC DISCHARGE.*

EC 32 ∅32 mm, brushless, 80 Watt, CE approved



M 1:2

Stock program
Standard program
Special program (on request)

Order Number

		118891	118892	118888	118889	118893	118890
Motor Data							
	Values at nominal voltage						
1	Nominal voltage	V	12.0	18.0	18.0	24.0	36.0	48.0
2	No load speed	rpm	15000	14300	13000	11000	14700	11300
3	No load current	mA	901	555	487	286	289	148
4	Nominal speed	rpm	13600	12800	11600	9510	13200	9790
5	Nominal torque (max. continuous torque)	mNm	37.5	40.1	41.2	43.6	39.7	42.6
6	Nominal current (max. continuous current)	A	5.82	3.88	3.61	2.37	1.98	1.19
7	Stall torque	mNm	428	443	407	355	454	353
8	Starting current	A	57.2	37.4	31.4	17.3	19.7	8.84
9	Max. efficiency	%	77	78	77	76	78	76
	Characteristics							
10	Terminal resistance phase to phase	Ω	0.21	0.481	0.573	1.39	1.83	5.43
11	Terminal inductance phase to phase	mH	0.03	0.0752	0.09	0.226	0.285	0.856
12	Torque constant	mNm / A	7.48	11.8	13.0	20.5	23.1	40.0
13	Speed constant	rpm / V	1280	806	737	465	414	239
14	Speed / torque gradient	rpm / mNm	35.8	32.7	32.6	31.5	32.8	32.5
15	Mechanical time constant	ms	7.49	6.86	6.82	6.59	6.87	6.8
16	Rotor inertia	gcm²	20.0	20.0	20.0	20.0	20.0	20.0

Specifications

Thermal data

17	Thermal resistance housing-ambient	5.4 K / W
18	Thermal resistance winding-housing	2.5 K / W
19	Thermal time constant winding	15.4 s
20	Thermal time constant motor	1180 s
21	Ambient temperature	-20 ... +100°C
22	Max. permissible winding temperature	+125°C

Mechanical data (preloaded ball bearings)

23	Max. permissible speed[1]	25000 rpm
24	Axial play at axial load < 8 N	0 mm
	> 8 N	max. 0.14 mm
25	Radial play	preloaded
26	Max. axial load (dynamic)	5.6 N
27	Max. force for press fits (static)	98 N
	(static, shaft supported)	1200 N
28	Max. radial loading, 5 mm from flange	28 N

Other specifications

29	Number of pole pairs	1
30	Number of phases	3
31	Weight of motor	270 g

Values listed in the table are nominal.

Connection motor (Cable AWG 22)

red	Motor winding 1
black	Motor winding 2
white	Motor winding 3

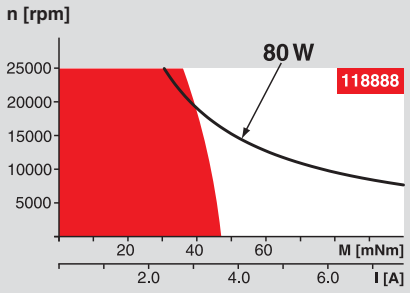
Connection sensors (Cable AWG 26)[1]

green	V_{Hall} 4.5 ... 24 VDC
blue	GND
red / grey	Hall sensor 1
black / grey	Hall sensor 2
white / grey	Hall sensor 3

Wiring diagram for Hall sensors see p. 27

[1] Not lead through in combination with resolver.

Operating Range



Comments

Continuous operation
In observation of above listed thermal resistance (lines 17 and 18) the maximum permissible winding temperature will be reached during continuous operation at 25°C ambient.
= Thermal limit.

Short term operation
The motor may be briefly overloaded (recurring).

Assigned power rating

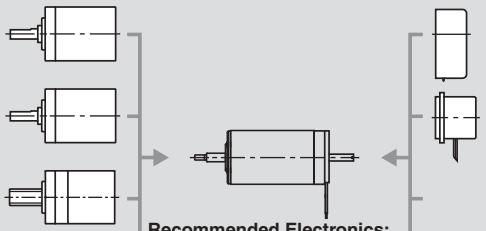
maxon Modular System

Overview on page 16 - 21

Planetary Gearhead
∅32 mm
0.75 - 4.5 Nm
Page 229

Planetary Gearhead
∅32 mm
0.75 - 6.0 Nm
Page 231 / 233

Spindle Drive
∅32 mm
Page 249 / 250 / 251



Encoder HED_5540
500 CPT,
3 channels
Page 267 / 269

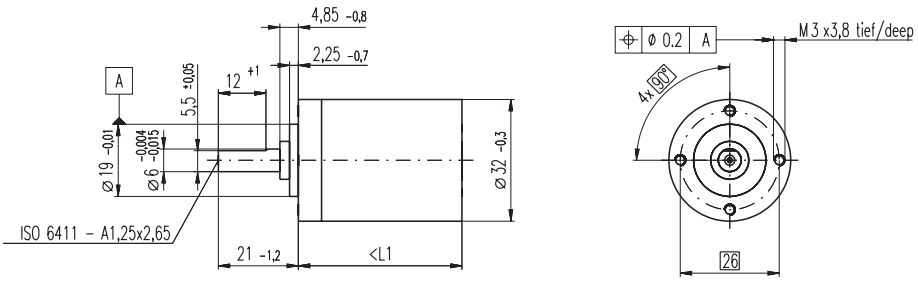
Resolver Res 26
∅26 mm
10 V
Page 277

Recommended Electronics:

DECS 50/5	Page 289
DEC 50/5	291
DEC Module 50/5	291
DECV 50/5, DEC 70/10	297
DES 50/5	298
EPOS2 24/5, EPOS2 50/5	305
EPOS2 70/10	305
EPOS2 P 24/5	308
Notes	**20**

Planetary Gearhead GP 32 A ∅32 mm, 0.75 - 4.5 Nm

Metal Version



M 1:2

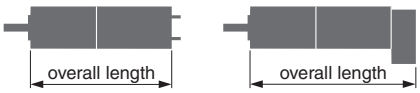
Technical Data

Planetary Gearhead	straight teeth
Output shaft	stainless steel
Shaft diameter as option	8 mm
Bearing at output	ball bearing
Radial play, 5 mm from flange	max. 0.14 mm
Axial play	max. 0.4 mm
Max. radial load, 10 mm from flange	140 N
Max. permissible axial load	120 N
Max. permissible force for press fits	120 N
Sense of rotation, drive to output	=
Recommended input speed	< 6000 rpm
Recommended temperature range	-20 ... +100°C
Extended area as option	-35 ... +100°C

Option: Low-noise version

Stock program
Standard program
Special program (on request)

			Order Number											
			166155	166158	166163	166164	166169	166174	166179	166184	166187	166192	166197	166202
Gearhead Data														
1	Reduction		3.7 : 1	14 : 1	33 : 1	51 : 1	111 : 1	246 : 1	492 : 1	762 : 1	1181 : 1	1972 : 1	2829 : 1	4380 : 1
2	Reduction absolute		$^{26}/_{7}$	$^{676}/_{49}$	$^{529}/_{16}$	$^{17576}/_{343}$	$^{13824}/_{125}$	$^{421824}/_{1715}$	$^{86112}/_{175}$	$^{19044}/_{25}$	$^{1012376}/_{8575}$	$^{8626176}/_{4375}$	$^{495144}/_{175}$	$^{109503}/_{25}$
3	Max. motor shaft diameter	mm	6	6	3	6	4	4	3	3	4	4	3	3
	Order Number		166156	166159		166165	166170	166175	166180	166185	166188	166193	166198	166203
1	Reduction		4.8 : 1	18 : 1		66 : 1	123 : 1	295 : 1	531 : 1	913 : 1	1414 : 1	2189 : 1	3052 : 1	5247 : 1
2	Reduction absolute		$^{24}/_{5}$	$^{624}/_{35}$		$^{16224}/_{245}$	$^{6877}/_{56}$	$^{101062}/_{343}$	$^{331776}/_{625}$	$^{36501}/_{40}$	$^{2425488}/_{1715}$	$^{536406}/_{245}$	$^{1907712}/_{625}$	$^{839523}/_{160}$
3	Max. motor shaft diameter	mm	4	4		4	3	3	4	3	3	3	3	3
	Order Number		166157	166160		166166	166171	166176	166181	166186	166189	166194	166199	166204
1	Reduction		5.8 : 1	21 : 1		79 : 1	132 : 1	318 : 1	589 : 1	1093 : 1	1526 : 1	2362 : 1	3389 : 1	6285 : 1
2	Reduction absolute		$^{23}/_{4}$	$^{299}/_{14}$		$^{3887}/_{49}$	$^{3312}/_{25}$	$^{389376}/_{1225}$	$^{20631}/_{35}$	$^{279841}/_{256}$	$^{9345024}/_{6125}$	$^{2066688}/_{875}$	$^{474513}/_{140}$	$^{6436343}/_{1024}$
3	Max. motor shaft diameter	mm	3	3		3	3	4	3	3	4	3	3	3
	Order Number			166161		166167	166172	166177	166182		166190	166195	166200	
1	Reduction			23 : 1		86 : 1	159 : 1	411 : 1	636 : 1		1694 : 1	2548 : 1	3656 : 1	
2	Reduction absolute			$^{576}/_{25}$		$^{14976}/_{175}$	$^{1587}/_{10}$	$^{359424}/_{875}$	$^{79488}/_{125}$		$^{1162213}/_{686}$	$^{7962624}/_{3125}$	$^{457056}/_{125}$	
3	Max. motor shaft diameter	mm		4		4	3	4	3		3	4	3	
	Order Number			166162		166168	166173	166178	166183		166191	166196	166201	
1	Reduction			28 : 1		103 : 1	190 : 1	456 : 1	706 : 1		1828 : 1	2623 : 1	4060 : 1	
2	Reduction absolute			$^{138}/_{5}$		$^{3588}/_{35}$	$^{12167}/_{64}$	$^{89401}/_{196}$	$^{158171}/_{224}$		$^{2238912}/_{1225}$	$^{2056223}/_{784}$	$^{3637933}/_{896}$	
3	Max. motor shaft diameter	mm		3		3	3	3	3		3	3	3	
4	Number of stages		1	2	2	3	3	4	4	4	5	5	5	5
5	Max. continuous torque	Nm	0.75	2.25	2.25	4.50	4.50	4.50	4.50	4.50	4.50	4.50	4.50	4.50
6	Intermittently permissible torque at gear output	Nm	1.1	3.4	3.4	6.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5
7	Max. efficiency	%	80	75	75	70	70	60	60	60	50	50	50	50
8	Weight	g	118	162	162	194	194	226	226	226	258	258	258	258
9	Average backlash no load	°	0.7	0.8	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10	Mass inertia	gcm²	1.5	0.8	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
11	Gearhead length L1	mm	26.4	36.3	36.3	43.0	43.0	49.7	49.7	49.7	56.4	56.4	56.4	56.4



Combination

+ Motor	Page	+ Tacho / Brake	Page	Overall length [mm] = Motor length + gearhead length + (tacho / brake) + assembly parts										
RE 25, 10 W	77			81.0	90.9	90.9	97.6	97.6	104.3	104.3	104.3	111.0	111.0	111.0	111.0
RE 25, 10 W	77	MR	258	92.0	101.9	101.9	108.6	108.6	115.3	115.3	115.3	122.0	122.0	122.0	122.0
RE 25, 10 W	77	Enc 22	260	95.1	105.0	105.0	111.7	111.7	118.4	118.4	118.4	125.1	125.1	125.1	125.1
RE 25, 10 W	77	HED_ 5540	262/264	101.8	111.7	111.7	118.4	118.4	125.1	125.1	125.1	131.8	131.8	131.8	131.8
RE 25, 10 W	77	DCT 22	271	103.3	113.2	113.2	119.9	119.9	126.6	126.6	126.6	133.3	133.3	133.3	133.3
RE 25, 20 W	78			69.5	79.4	79.4	86.1	86.1	92.8	92.8	92.8	99.5	99.5	99.5	99.5
RE 25, 20 W	79			81.0	90.9	90.9	97.6	97.6	104.3	104.3	104.3	111.0	111.0	111.0	111.0
RE 25, 20 W	79	MR	258	92.0	101.9	101.9	108.6	108.6	115.3	115.3	115.3	122.0	122.0	122.0	122.0
RE 25, 20 W	79	Enc 22	260	95.1	105.0	105.0	111.7	111.7	118.4	118.4	118.4	125.1	125.1	125.1	125.1
RE 25, 20 W	79	HED_ 5540	262/264	101.8	111.7	111.7	118.4	118.4	125.1	125.1	125.1	131.8	131.8	131.8	131.8
RE 25, 20 W	79	DCT 22	271	103.3	113.2	113.2	119.9	119.9	126.6	126.6	126.6	133.3	133.3	133.3	133.3
RE 25, 20 W	79	AB 28	308	115.1	125.0	125.0	131.7	131.7	138.4	138.4	138.4	145.1	145.1	145.1	145.1
RE 25, 20 W	79	HED_ 5540/AB 28	262/308	132.2	142.1	142.1	148.8	148.8	155.5	155.5	155.5	162.2	162.2	162.2	162.2
RE 26, 18 W	80			85.3	95.2	95.2	101.9	101.9	108.6	108.6	108.6	115.3	115.3	115.3	115.3
RE 26, 18 W	80	MR	258	96.3	106.2	106.2	112.9	112.9	119.6	119.6	119.6	126.3	126.3	126.3	126.3
RE 26, 18 W	80	Enc 22	260	102.7	112.6	112.6	119.3	119.3	126.0	126.0	126.0	132.7	132.7	132.7	132.7
RE 26, 18 W	80	HED_ 5540	262/264	103.7	113.6	113.6	120.3	120.3	127.0	127.0	127.0	133.7	133.7	133.7	133.7
RE 26, 18 W	80	DCT 22	271	106.3	116.2	116.2	122.9	122.9	129.6	129.6	129.6	136.3	136.3	136.3	136.3
A-max 26	115-122			71.2	81.1	81.1	87.8	87.8	94.5	94.5	94.5	101.2	101.2	101.2	101.2
A-max 26	115-121	MEnc 13	270	78.3	88.2	88.2	94.9	94.9	101.6	101.6	101.6	108.3	108.3	108.3	108.3
A-max 26	116-122	MR	258	80.0	89.9	89.9	96.6	96.6	103.3	103.3	103.3	110.0	110.0	110.0	110.0
A-max 26	116-122	Enc 22	261	85.6	95.5	95.5	102.2	102.2	108.9	108.9	108.9	115.6	115.6	115.6	115.6
A-max 26	116-122	HED_ 5540	263/265	90.0	99.9	99.9	106.6	106.6	113.3	113.3	113.3	120.0	120.0	120.0	120.0
RE-max 29	145-148			71.2	81.1	81.1	87.8	87.8	94.5	94.5	94.5	101.2	101.2	101.2	101.2
RE-max 29	146/148	MR	258	80.0	89.9	89.9	96.6	96.6	103.3	103.3	103.3	110.0	110.0	110.0	110.0