ANDRÉ RIBEIRO LINS DE ALBUQUERQUE

**APLICAÇÕES DE *HARDWARE-IN-THE-LOOP* NO
DESENVOLVIMENTO DE UMA MÃO ROBÓTICA**

Tese apresentada à Escola de Engenharia de São Carlos
da Universidade de São Paulo para obtenção do título
de Doutor em Engenharia Mecânica.

Área de Concentração: Mecatrônica
Orientador: Prof. Assoc. Glauco Augusto de Paula
Caurin

São Carlos
2007

à querida família, pai, mãe, irmã que sempre me incentivaram e torceram por mim e, especialmente a minha mais nova e encantadora família: Juliana e Enrique.

AGRADECIMENTO

Ao Prof. Glauco, pela oportunidade, orientação e paciência neste trabalho.

À Fapesp, pelo suporte financeiro para o desenvolvimento deste trabalho.

À Universidade de São Paulo, seus funcionários e docentes que colaboram de forma expressiva e eficiente com o desenvolvimento de pesquisas e de novas tecnologias no país.

A todos os membros da equipe BRAHMA, incluindo os técnicos do laboratório de dinâmica, pela colaboração e empenho no desenvolvimento do projeto.

Ao Dr. Ronald Vuillemin, chefe do Instituto de Eletrônica da Universidade Central de Ciências Aplicadas da Suíça e, à sua equipe, pelo acolhimento, hospitalidade e apoio a este trabalho.

Aos amigos, Leonardo Marquez, Marcio Montezuma, Alan Ettlin e, Cláudio Policastro pela grata oportunidade de convivência, trabalho e descontração.

À grande amiga Luciana Abdo, pelo seu companheirismo e apoio neste período.

“O conhecimento amplia a vida. Conhecer é viver uma realidade que a ignorância impede desfrutar”.

Carlos Bernardes Gonzáles Pecoche

RESUMO

ALBUQUERQUE, A.R.L.A. **Aplicações de *Hardware-in-the-Loop* no desenvolvimento de uma mão robótica**. 2007. 173 f. Tese (Doutorado) - Escola de Engenharia de São Carlos da Universidade de São Paulo, São Carlos, 2007.

O trabalho tem como objetivo o estudo e a aplicação da técnica de *Hardware-in-the-Loop* como uma ferramenta de suporte no processo de desenvolvimento de uma mão artificial robótica. Os esforços se concentram no desenvolvimento de um ambiente computacional e um ambiente experimental para trabalharem em conjunto e simultaneamente. No ambiente computacional foi desenvolvido o modelo do sistema simulado em tempo real. No ambiente experimental, partes do protótipo da mão robótica foram implementadas. Em ambos os casos, foram desenvolvidos e empregados um controlador seguidor multivariável. Adotando este tipo de abordagem, partes do sistema simulado em tempo real poderão ser substituídas - à medida de suas necessidades - por partes físicas, como por exemplo: sensores, atuadores e novos *hardwares* de controle, possibilitando uma considerável redução de investimento em *hardware* e de tempo de projeto.

ABSTRACT

ALBUQUERQUE, A.R.L.A. **Hardware-in-the-loop applications in the robotic hand development.** 2007. 173 p. Thesis (Doctoral) - Escola de Engenharia de São Carlos da Universidade de São Paulo, São Carlos, 2007.

The purpose of this work is the study and the application of the Hardware-in-the-Loop technique as a support tool in the development process of an artificial robotic hand. The efforts concentrate on the development of a computational and experimental environment to work together and simultaneously. In the computational environment, the simulated system model was developed in real-time. In the experimental environment, prototype parts of the robotic hand were implemented. In both cases, a multivariable controller was developed and utilized. By adopting this approach, parts of the system simulated in real time can be substituted – according to the needs - by physical parts, such as: sensors, actuators, and new control hardware, allowing a considerable investment reduction in hardware and in time of project.

LISTA DE FIGURAS

Figura 1.1: Combinações possíveis entre estruturas físicas e simuladas em uma abordagem híbrida de <i>Hardware-in-the-Loop</i>	18
Figura 1.2: Estrutura mecânica da BRAHMA	22
Figura 1.3: Metodologia simplificada do desenvolvimento do projeto	24
Figura 2.1: Vista dorsal da junta interfalangeal e metacarpo-falangeal retirada de NEUMANN, (2002)	31
Figura 2.2: Representação de um dedo genérico.....	32
Figura 2.3: Diferentes vistas da versão final do protótipo virtual da BRAHMA.....	33
Figura 2.4: Representação dos três últimos corpos do dedo, sendo: ab = falange proximal; ac = falange medial e ad = falange distal.....	34
Figura 2.5: Representação geométrica do ângulo ϕ_m	36
Figura 2.6: Modelagem da parte referente à “palma da mão” - do metacarpo ao punho	36
Figura 2.7: Representação geométrica ângulo ϕ_p no dedo polegar.	37
Figura 2.8: Notação adotada para cada dedo.....	38
Figura 2.9: Velocidades angulares relativas.....	40
Figura 2.10: Simulação do movimento nas articulações do dedo indicador.....	45
Figura 2.11: Resultado das Simulações. O gráfico a esquerda mostra o deslocamento angular imposto às articulações do dedo indicador. O gráfico à direita é apresentado o resultado dos deslocamentos lineares da ponta do dedo.	46
Figura 2.12: Resultado das simulações da cinemática inversa. O primeiro gráfico apresenta uma imposição de um movimento linear à ponta do dedo. O gráfico do meio apresenta a orientação deste movimento linear. O último gráfico mostra como resposta os deslocamentos angulares nas articulações.....	47
Figura 3.1: Esquema do sistema de acionamento –motor, acoplamento e carga.	50

Figura 3.2: Representação do sistema na forma de diagrama de blocos	51
Figure 3.3: Circuitos Elétrico de um Motor DC	54
Figura 3.4: Representação do sistema completo na forma de diagrama de blocos (Eq. 3.17).	56
Figura 4.1: Planta do sistema representada na forma de espaço de estados	63
Figura 4.2: Sistema de Controle Seguidor	65
Figura 4.3: Sinais de Entrada Degrau em cada uma das articulações.....	68
Figura 4.4: Deslocamentos angulares das articulações dos dedos.....	68
Figura 4.5: Medição dos Erros (diferença entre as figuras 4.3 e 4.4)	69
Figura 4.6: Velocidades angulares das articulações dos dedos	69
Figura 4.7: Torques das articulações dos dedos	70
Figura 4.8: Sinal do Controle	70
Figura 4.9: Sinais de Entrada Degrau	71
Figura 4.10: Deslocamentos angulares das articulações dos dedos.....	71
Figura 4.11: Medição do Erro (diferença entre as figuras 4.9 e 4.10)	72
Figura 4.12: Velocidades angulares das articulações dos dedos	72
Figura 4.13: Torques das articulações dos dedos	73
Figura 4.14: Sinal do Controle	73
Figura 4.15: Sinais de Entrada Senoidal com frequência de 5 rad/s	74
Figura 4.16: Deslocamentos angulares das articulações dos dedos.....	74
Figura 4.17: Medição do Erro (diferença entre as Figuras 4.15 e 4.16)	75
Figura 4.18: Velocidades angulares das articulações dos dedos	75
Figura 4.19: Torques das articulações dos dedos	76
Figura 4.20: Sinal do Controle	76
Figura 4.21: Sinal de erro para a primeira atribuição de autovalores.	77
Figura 4.22: Sinal de erro para a segunda atribuição de autovalores.	78

Figura 4.23: Sinal de erro para a terceira atribuição de autovalores.....	78
Figura 5.1: O <i>hardware</i> de controle posicionado no mesmo <i>loop</i> de uma simulação em tempo real do sistema.	82
Figura 5.2: O <i>hardware</i> de controle e um atuador físico no mesmo <i>loop</i> de uma simulação em tempo real de um dedo da BRAHMA.....	83
Figura 5.3: Sistemas de Tempo Real	89
Figura 5.4: Esquema da conexão de <i>hardware</i>	94
Figure 5.5: Estrutura geral da plataforma de testes em HIL	95
Figure 5.6: Plataforma MVME162 utilizada no trabalho	96
Figura 5.7: Controle seguidor aplicado ao modelo discreto do motor Maxon (modelo 14434)	100
Figura 5.8: Configuração do RTW para trabalhar com o Tornado [®]	102
Figura 5.9: Resultados experimentais do controle de posição aplicado a um modelo de motor em HIL. Em azul tem-se a trajetória desejada e em verde a resposta medida.....	103
Figura 5.10: Na ordem têm-se as seguintes respostas no tempo: 1) posição angular (em verde) e sinal de entrada (em azul); 2) medição do erro; 3) tensão de entrada do motor; 4) torque no eixo de entrada do motor.	104
Figura 5.11: <i>Device Drivers</i> desenvolvidos para a comunicação com os <i>Industrial Packs</i>	105
Figura 5.12: Ambiente experimental. Na sequência: 1) O protótipo da Brahma; 2) o <i>Target</i> ; 3) o <i>driver</i> de potência e motor da Maxon; 4) o <i>Host</i>	106
Figura 5.13: Aplicação do controle seguidor a uma planta simulada e a uma planta física para uma entrada referência de meia onda senoidal com amplitude de 15 radianos e frequência de 5 rad/s.....	107
Figura 5.14: Na ordem tem-se as seguintes respostas instantâneas em uma janela de 4 segundos para uma entrada de meia onda senoidal com amplitude de 15 radiano e	

frequência de 5 rad/s no eixo de entrada do motor: 1) trajetória desejada; 2) resposta da planta simulada em tempo real; 3) resposta da planta física.	107
Figura 5.15: Na ordem tem-se as seguintes respostas instantâneas em uma janela de 4 segundos dos sinais de controle em Volts para uma entrada de meia onda senoidal com amplitude de 15 radianos e frequência de 5 rad/s: 1) sinal de controle da planta simulada em tempo real; 2) sinal de controle da planta física.....	108
Figura 5.17: Aplicação do controle seguidor a uma planta simulada e a uma planta física para uma entrada pulsada de referência com amplitude de 15 radianos no eixo de entrada do motor e passo de 2 segundos.	108
Figura 5.18: Na ordem tem-se as seguintes respostas instantâneas em uma janela de 4 segundos para uma entrada pulsada: 1) trajetória desejada; 2) resposta da planta simulada em tempo real; 3) resposta da planta física.....	109
Figura 5.19: Na ordem tem-se as seguintes respostas instantâneas em uma janela de 4 segundos dos sinais de controle em Volts para uma entrada pulsada: 1) sinal de controle da planta simulada em tempo real; 2) sinal de controle da planta física.....	109
Figura 5.20: Instrumentação do <i>Target</i> utilizando ferramentas de monitoramento do RTOS	110
Figura 5.21: Representação em diagrama de blocos do modelo da BRAHMA levando-se em conta o acoplamento dinâmico entre os atuadores e as articulações.....	111
Figura 5.22: Sinais de teste aplicados no modelo de simulação em tempo real de um dos dedos da BRAHMA.	111
Figura 5.23: Resposta instantânea dos deslocamentos angulares de um dos dedos da BRAHMA em um período de tempo específico.....	112
Figura 5.24: Ilustração da abordagem híbrida de HIL adotada, onde o hardware de controle e um atuador físico estão no mesmo <i>loop</i> de simulação em tempo real de um dedo da BRAHMA.....	113

Figura 5.25: Resposta instantânea dos deslocamentos angulares de um dos dedos da BRAHMA em uma abordagem híbrida de HIL para as mesmas condições da abordagem clássica.....	113
Figura 5.26: Resposta instantânea do sinal de entrada versus a resposta do deslocamento angular no eixo de saída do motor versus a resposta do deslocamento angular na articulação θ_5	114
Figura 5.27: Comparação entre as Figuras 5.25 e 5.26. Sendo as curvas contínuas, as respostas instantâneas dos testes em HIL clássico e, as linhas tracejadas, as referentes às resposta instantâneas da abordagem híbrida de HIL.	115
Figura A1: Representação dos corpos de dedo indicador.	129
Figura A2: Representação do sistema de coordenadas do dedo indicador da BRAHMA.	131
Figura A3: Função sigmóide de entrada	133
Figura A4: Gráfico de resposta da Energia Cinética no Tempo.....	134
Figura A5: Gráfico de resposta da Energia Potencial no Tempo	134
Figura A6: Gráfico de resposta do Torque nas articulações no Tempo.....	135
Figura A7: Torque na articulação 6 e suas componentes devido a gravidade, inércia e as acelerações centrípeta.....	135
Figura A8: Torque na articulação 5 e suas componentes devido a gravidade, inércia e as acelerações centrípeta.....	136
Figura A9: Torque na articulação 4 e suas componentes devido a gravidade, inércia e as acelerações centrípeta.....	136
Figura A10: Função sigmóide de entrada	137
Figura A.11: Gráfico de resposta da Energia Cinética no Tempo.....	137
Figura A.12: Gráfico de resposta da Energia Potencial no Tempo	138
Figura A.13: Gráfico de resposta do Torque nas articulações no Tempo.....	138

Figura A.14: Torque na articulação 6 e suas componentes devido a gravidade, inércia e as acelerações centrípeta.....	139
Figura A.15: Torque na articulação 5 e suas componentes devido a gravidade, inércia e as acelerações centrípeta.....	139
Figura A.16: Torque na articulação 4 e suas componentes devido a gravidade, inércia e as acelerações centrípeta.....	139
Figura B1: Interface gráfica do Modelo BRAHMA (conjunto dedos + atuadores)	140

LISTA DE TABELA

Tabela 2.1: Nomenclatura de ângulos e dimensões para cada dedo.....	39
Tabela 3.1: Valores das raízes do sistema linearizado em duas situações extremas	53
Tabela A.1: Propriedades físicas do dedo indicador da BRAHMA.....	133

SUMÁRIO

1 INTRODUÇÃO E JUSTIFICATIVA.....	16
1.1 OBJETIVO.....	23
1.2 METODOLOGIA E ORGANIZAÇÃO DO TRABALHO	24
2. PROJETO DE UMA MÃO ARTIFICIAL ROBÓTICA	28
2.1 INTRODUÇÃO E JUSTIFICATIVA	28
2.2 DESENVOLVIMENTO DA ESTRUTURA DA BRAHMA	31
2.3 ANÁLISE CINEMÁTICA DOS DEDOS.....	34
3 DESENVOLVIMENTO DO MODELO DINÂMICO DA BRAHMA	48
3.1 MODELAGEM DO SISTEMA BRAHMA	49
3.1.1 MODELAGEM DA ARMADURA DO MOTOR DC	53
3.2 IMPLEMENTAÇÃO COMPUTACIONAL.....	57
3.2.1 INTEGRADOR	57
4 SISTEMA DE CONTROLE	62
4.1 MODELO EM ESPAÇO DE ESTADOS	62
4.2 SISTEMA DE CONTROLE SEGUIDOR	63
4.3 RESULTADOS PARA ENTRADA DEGRAU	67
4.4 RESULTADOS PARA ENTRADA PULSADA.....	71
4.5 RESULTADOS PARA ENTRADA DE MEIA ONDA SENOIDAL COM FREQUÊNCIA DE 5 RAD/S	74
4.6 RESULTADOS PARA ENTRADA DEGRAU COM DIFERENTES ATRIBUIÇÕES DE AUTOVALORES	77
4.7ANÁLISE DOS RESULTADOS	79
4.8 CONCLUSÕES SOBRE AS SIMULAÇÕES OFF-LINE DO SISTEMA CONTROLADO	81
5 HARDWARE-IN-THE-LOOP	82
5.1 INTRODUÇÃO.....	84
5.2 INTRODUÇÃO AO CONCEITO DE TEMPO REAL	88
5.3 DESCRIÇÃO DOS SOFTWARES E HARDWARES UTILIZADOS	91

5.3.1 SISTEMA OPERACIONAL TEMPO REAL – VxWORKS	91
5.3.2 TORNADO	92
5.3.3 <i>REAL-TIME-WORKSHOP</i> (RTW)	93
5.3.4 PLATAFORMA MVME162	94
5.3.5 <i>INDUSTRY PACK (IP)</i>	97
5.4 METODOLOGIA DE IMPLEMENTAÇÃO DE HIL	99
5.4.1 IMPLEMENTAÇÃO	99
5.5 RESULTADOS DAS APLICAÇÕES EM HIL	106
6 CONCLUSÕES E CONSIDERAÇÕES FINAIS.....	116
6.1 ANÁLISE DOS BENEFÍCIOS DO USO DE HIL NO TRABALHO	118
6.2 SUGESTÕES PARA TRABALHOS FUTUROS.....	120
7. REFERÊNCIAS BIBLIOGRÁFICAS	121
APÊNDICE A – MODELAGEM MATEMÁTICA.....	127
APÊNDICE B - TOPOLOGIA DO MODELO BRAHMA	140
APÊNDICE C – CÓDIGO FONTE DAS SIMULAÇÕES EM HIL.....	147

1 INTRODUÇÃO E JUSTIFICATIVA

A técnica de *Hardware-in-the-Loop* tem sido requerida cada vez mais em projetos, implementações, análises e testes de sistemas mecatrônicos. A proposta deste tipo de abordagem é o de minimizar as dificuldades envolvidas no desenvolvimento de sistemas mecatrônicos complexos.

O desenvolvimento completo de um projeto mecatrônico, assim como de uma mão artificial robótica, enfrenta uma grande variedade de problemas causada pela interação entre diferentes áreas. Esta interdisciplinaridade da mecatrônica impõe a utilização de ferramentas de engenharia que permitam o projeto simultâneo e concorrente de diferentes partes do sistema associados com as áreas específicas de conhecimento. Por conta do crescimento, da complexidade e da interdisciplinaridade envolvidas entre o projeto mecânico e o projeto do sistema de controle, novas abordagens e ferramentas computacionais têm sido, cada vez mais, aplicadas para auxiliar o projeto. Dentre elas, destacam-se a modelagem e as simulações *off-line* e em tempo real. A consequência do uso de tais abordagens e ferramentas tem sido a redução do tempo de desenvolvimento bem como o aumento na qualidade, confiabilidade e segurança do produto final (em fase de elaboração¹).

Durante o projeto de sistemas mecatrônicos, é importante que as modificações na mecânica e no controlador sejam feitas simultaneamente. Apesar de um controlador apropriado poder oferecer uma construção mecânica mais barata, um sistema mecânico mal projetado nunca poderá oferecer uma boa performance, mesmo com a adição de um controle sofisticado. Dessa forma, é importante que uma escolha apropriada com relação às propriedades mecânicas seja feita durante os estágios preliminares do projeto – levando em

¹ Albuquerque, A. R. L. et.al. *Hardware-in-the-Loop as Supporting Tool for Mechatronic Project Development*. A ser editado pela ABCM.

consideração o tipo de material e o seu projeto estrutural e dinâmico - necessárias para se obter um bom desempenho do sistema controlado.

A realização completa de muitos produtos e processos modernos abrange a integração dos sistemas de controle digitais que podem ser divididos em duas fases de integração: de *hardware* e de *software*. Na fase de integração de *hardware*, os atuadores, sensores, microcontroladores e a eletrônica de potência deverão, não só ser selecionados e/ou desenvolvidos, mas também testados, analisados e validados de forma sistemática. Na fase de integração de *software*, as variáveis medidas de entrada e saída devem abastecer os múltiplos níveis de um sistema de controle distribuído (VAN AMERONGEN; BREEDVELD, 2003). Tipicamente isto inclui: baixo nível de controladores, alto nível de controle (e.g. estratégias avançadas de controle realimentado e supervisão) e nível de otimização.

Atualmente, técnicas de simulação em tempo real oferecem benefícios significativos para minimizar as dificuldades associadas com as fases de integração de *hardware* e *software* de um processo de desenvolvimento de um sistema mecatrônico. Usualmente, existem três abordagens básicas mais comumente aplicadas. Estas podem ser distinguidas como:

1. *Software-in-the-loop*: A planta do sistema e a sua estrutura de controle são simuladas em tempo-real. Esta abordagem pode ser requerida quando o *hardware* não está disponível ou em estágios preliminares às simulações de *Hardware-in-the-Loop*.
2. *Hardware-in-the-Loop*: O *hardware* dedicado de controle é utilizado no experimento e a planta é substituída por um modelo de simulação em tempo real. Esta abordagem é clássica em testes e análises de comportamento e desempenho de sistemas dedicados de controle.
3. *Control Prototyping*: A planta do sistema é utilizada no experimento e o *hardware* dedicado de controle é substituído por um modelo simulado do controlador executado em um computador de propósito geral. Esta abordagem é utilizada com

freqüência quando se deseja testar novos algoritmos de controle diretamente na planta física.

Cada uma destas abordagens pode ser aplicada no processo de desenvolvimento de um produto mecatrônico (ISERMAN et al., 2003), dependendo da necessidade específica de cada projeto. Neste trabalho, a abordagem de *Hardware-in-the-Loop* (HIL) será explorada, apontando os benefícios, ganhos e riscos na utilização desta técnica.

Simulações em *Hardware-in-the-Loop* (HIL) são usualmente utilizadas em um nível de projeto em que seja possível a modelagem da dinâmica do sistema com alto grau de precisão e fidelidade. Frequentemente, a dinâmica da planta é simulada em tempo real porque o protótipo físico não está disponível, ou porque experimentos com partes físicas reais em determinadas fases do projeto implicariam em um custo desnecessário devido ao tempo despendido e ao investimento monetário do mesmo. Além de ser, em muitos casos, uma forma de se evitar riscos desnecessários em projetos que possam comprometer, de alguma forma, a vida humana (e.g. plantas nucleares, projetos de lançamentos de veículos espaciais, etc.).

A combinação de duas das mais usuais técnicas de simulação em tempo real - HIL e *Control Prototype* - é definida neste trabalho como uma abordagem híbrida de HIL. Na Figura 1.1, esta abordagem é esquematizada de forma geral, sendo um produto mecatrônico subdividido em quatro subsistemas básicos: sistema de controle, atuadores, planta e, sensores, podendo cada um deles ser total ou parcialmente simulados em tempo real.

Figura 1.1: Combinações possíveis entre estruturas físicas e simuladas em uma abordagem híbrida de *Hardware-in-the-Loop*.

Na abordagem híbrida de HIL, algumas partes do sistema são frequentemente protótipos ou componentes físicos reais, enquanto outras são modelos simulados em tempo real. Uma planta completa de um sistema mecatrônico complexo é tipicamente composta por múltiplos subsistemas que podem ser decompostos hierarquicamente até o nível de componente. Isso remete a uma consideração de que modelos simulados em tempo real em simulações HIL podem substituir não somente a planta inteira, mas também qualquer nível hierárquico do sistema que se deseja estudar em maior profundidade.

A técnica HIL tem se tornado uma abordagem cada vez mais interessante para diversos segmentos de atuação, sendo amplamente aplicada principalmente na indústria automotiva (HAGIWARA et al., 2002), (KENDALL; JONES, 1999), pelo fato de ser um dos segmentos de mercado mais competitivos na atualidade. Onde compradores de veículos esperam alto nível de qualidade, segurança e funcionabilidade, e isto é acompanhado, naturalmente, por uma pressão por diminuição de custos e tempo de desenvolvimento.

A construção de protótipos se faz necessária em diversas etapas do projeto de um novo veículo (BERTRAM et al., 2003), (HAGIWARA et al., 2002). Cada protótipo é cuidadosamente construído. Entretanto, quando se iniciam os testes, frequentemente surgem problemas que devem ser comunicados ao fornecedor das peças ou equipamentos que ocasionaram tais problemas, ocorrendo alterações no projeto de *hardware* e/ou de *software*. Isto acaba gerando uma constante reestruturação do projeto, o que acarreta na construção de uma frota de protótipos até que se atinja, com sucesso, o sistema desejado.

A fim de reduzir a dependência dos protótipos veiculares, principalmente nos estágios preliminares do projeto e, além disto, reduzir tempo e esforço, aplica-se a técnica de HIL. Esta permite que o protótipo completo do veículo seja construído cada vez mais tarde no programa de desenvolvimento, de forma que não sejam necessárias maiores modificações antes de iniciar a produção em escala (HAGIWARA et al., 2002).

Uma outra área que está começando a se interessar pela técnica HIL é a área de automação. Em Linjama et al.(2003) é analisada a possibilidade de se substituir servo-posicionadores hidráulicos por modelos simulados em tempo real, tendo como objetivo o teste do *hardware* de controle juntamente com o modelo simulado que inclui a dinâmica do servo bem como a retroalimentação dos sensores. Neste trabalho, a técnica HIL é apontada como um método promissor para diminuir custos de desenvolvimento em plantas industriais.

A área espacial também tem utilizado com frequência as técnicas de HIL. Um exemplo disso é o trabalho de Careful et al. (2000), onde o HIL é utilizado para simular o *hardware* do espaço e emular o contato dinâmico utilizando um robô rígido para executar tarefas. O objetivo desse trabalho é obter a manipulação habilidosa de manipuladores robóticos para uso em estações espaciais.

No trabalho de Necsulescu e Basic (2003) é realizada uma análise de caracterização do desempenho e dos limites de uma interface homem-máquina do tipo *Haptic* utilizando uma configuração experimental em *Hardware-in-the-Loop*, na qual é utilizado um modelo não linear de realimentação de força do ambiente virtual na implementação da interface.

No desenvolvimento de veículos autônomos subaquáticos (AUV) utiliza-se largamente a técnica HIL. O trabalho de David et al.(2001) apresenta uma implementação e considerações sobre simulação HIL para veículos subaquáticos. O uso de simulações HIL é muito proveitoso neste tipo de aplicação devido à complexidade do desenvolvimento da maioria dos veículos autônomos subaquáticos.

Técnicas de simulação HIL estão sendo utilizadas para agilizar os procedimentos de testes de uma nova configuração de elevador (BÜCHLER et al. 2005). Vários componentes de um sistema global de elevador são simulados em tempo real. Em uma primeira situação, o mecanismo de abertura das portas do elevador - incluindo as correias dentadas e polias - é simulado em tempo real e conectado a um *hardware* dedicado de controle (*target*). Barreiras

fotoelétricas também são simuladas no mesmo ambiente de simulação para testar as respostas do controle a determinadas entradas deste sensor. Em outra situação, um sistema completo - incluindo múltiplas portas e o cabo central do elevador - é simulado em tempo real e seu controle avaliado por meio da técnica de HIL.

Engenheiros de tráfego já utilizam a algum tempo *softwares* de simulação para desenvolver modelos e testar planos de seqüenciamento de sinais de trânsito, porém, antes de implementar estes planos em campo é necessário um fino ajuste em um controlador de tráfego atuante. Trabalhos como o de Zhen et al. (2004) mostram que estes planos de escalonamento podem ser testados e implementados em laboratório pelo uso de técnicas de simulação em HIL evitando assim, o desconforto de motoristas e pedestres de enfrentar atrasos e congestionamentos.

Muitos outros processos de desenvolvimento de produtos mecatrônicos utilizam atualmente a abordagem de HIL em alguma situação, tanto para obter uma melhora no processo de integração em diferentes fases do desenvolvimento ou para testar novos algoritmos de controle em *hardware* específicos. No presente trabalho, a técnica de HIL é demonstrada utilizando como exemplo o desenvolvimento de uma mão artificial robótica (Figura 1.2), denominada BRAHMA (*Brazilian Anthropomorphic Hand*). Este exemplo de aplicação será utilizado para ilustrar vários aspectos da técnica aqui proposta.

Figura 1.2: Estrutura mecânica da BRAHMA

Dentro do processo de desenvolvimento da BRAHMA, as estratégias clássica e híbrida de HIL são aplicadas quando um *hardware* dedicado de controle e algumas partes da planta são inseridos no *loop* de simulação em tempo real. Adotando este tipo de abordagem, partes do sistema simulado em tempo real são substituídas - a medida de suas necessidades - por partes físicas, como por exemplo: sensores, atuadores e, novos *hardwares* de controle. Dentre os possíveis benefícios, destacam-se:

1. Verificação do comportamento do sistema simulado em tempo real utilizando um controle seguidor multivariável sendo executado em um *hardware* dedicado. Com isto o sistema pode ser testado e analisado sob diversas condições.
2. Interação e simultaneidade entre modelos simulados em tempo real e protótipos físicos na execução das tarefas e testes.
3. Diminuição do risco do desenvolvimento relacionado à utilização de diferentes tipos de tecnologia.
4. Criação de uma estrutura para o desenvolvimento de novos algoritmos (gerador de trajetórias, controladores, etc.) de maneira mais ágil e direta.
5. Diminuição dos custos do projeto de desenvolvimento, uma vez que construir, instrumentar, acionar todo o protótipo físico (20 graus de liberdade) e testar diversos sistemas de controle e planejadores de trajetória teria um custo proibitivo para este projeto de pesquisa.

1.1 OBJETIVO

O trabalho tem como objetivo o estudo e a aplicação da técnica de *Hardware-in-the-Loop* como uma ferramenta de suporte dentro do processo de desenvolvimento de uma mão artificial robótica.

1.2 METODOLOGIA E ORGANIZAÇÃO DO TRABALHO

Todas as fases do processo de desenvolvimento do projeto BRAHMA (*Brazilian Anthropomorphic Hand*) são apresentadas nos presentes capítulos para assim iniciar a aplicação e avaliação da técnica de HIL. A sequência metodológica básica de desenvolvimento do projeto é mostrada na Figura 1.3.

Figura 1.3: Metodologia simplificada do desenvolvimento do projeto

O projeto se inicia com a existência clara de um objetivo a ser alcançado que, no caso deste trabalho, é reproduzir as funcionalidades da mão humana mimetizando inclusive as suas formas naturais e estéticas. A partir daí os projetistas envolvidos no desenvolvimento concebem e criam as primeiras formas e características do projeto. Um maior grau de detalhamento se faz, então, necessário, dando as características construtivas do mesmo, como por exemplo: o acionamento mecânico, o tipo de material, a união entre os corpos, a tolerância nos ajustes, as restrições de movimentos, etc. Uma vez concebido o protótipo virtual, o próximo passo é a obtenção de um modelo dinâmico que represente o comportamento físico deste protótipo, ou seja, a relação entre seus movimentos e os torques nas articulações e vice e versa. Na ocorrência de alguma inconsistência nesta fase de desenvolvimento, deve-se necessariamente voltar para a fase anterior e corrigir o projeto (protótipo virtual). A próxima etapa é o projeto do controlador, cujo objetivo é o

dimensionamento de um controle que cumpra com as necessidades do projeto com relação a velocidade e precisão do sistema, analisando e avaliando também a energia consumida pelo sistema sob ação do controle.

Em seguida vem o dimensionamento, seleção e/ou desenvolvimento dos atuadores e sensores. Nesta etapa, modificações no projeto se fazem necessárias para agregar estes novos componentes. O fim desta etapa é a obtenção de um protótipo virtual mais acabado. A partir daí, o modelo dinâmico para este novo protótipo virtual é atualizado, seguindo-se para a etapa de desenvolvimento do controlador final para que o sistema em desenvolvimento se comporte conforme as necessidades de suas aplicações.

Por fim, aplicando a abordagem clássica HIL, o comportamento do sistema controlado é simulado em tempo real utilizando, para tanto, um ambiente de desenvolvimento de controle apropriado. Além disso, partes do protótipo físico que necessitem de maiores análises e testes são integradas ao sistema de simulação acrescentando, nesta fase, a característica de interação entre o ambiente físico (experimental) e o modelo do sistema simulado em tempo real (computacional).

O trabalho é apresentado respeitando a sequência metodológica apresentada na Figura 1.3, sendo cada uma das etapas dividida em forma de capítulos, da seguinte forma:

- O Capítulo 2 apresenta o projeto da BRAHMA (*Brazilian Anthropomorphic Hand*). Inicialmente é feita uma introdução ao tema mãos humanas artificiais. Em seguida é descrito o processo de concepção do protótipo virtual que vai desde o projeto das articulações dos dedos até o seu sistema de acionamento. A cinemática de um dos dedos da mão, juntamente com o formalismo matemático adequado é também apresentada.
- No Capítulo 3, é apresentado de forma simplificada e direta o desenvolvimento, testes e análises do modelo completo do sistema BRAHMA. Uma análise mais aprofundada

da dinâmica do projeto BRAHMA pode ser encontrada em um dos anexos deste trabalho. Neste anexo é apresentada a modelagem e a dedução completa das equações dinâmicas de um dos dedos da BRAHMA, além de uma análise da dinâmica do sistema como um todo, observando o quanto cada componente da equação dinâmica (inércia, gravidade e aceleração Centrípeta) influência no resultado final.

- No Capítulo 4 é descrito o processo de desenvolvimento de um controle seguidor multivariável do modelo do sistema BRAHMA com apenas 3 dos dedos da BRAHMA (12 graus de liberdade) por meio de realimentação de estados. As respostas desse controlador são analisadas para diversos tipos de sinais de entrada por meio de simulações numéricas.
- No Capítulo 5 inicia-se a abordagem *Hardware-in-the-loop*. Logo após uma introdução sobre o tema, é apresentada inicialmente uma discussão sobre a técnica de HIL, apontando os conceitos principais, as formas de aplicação e as principais ferramentas e abordagens existentes. Além disso, o conceito sobre tempo real é mais aprofundado em uma seção específica. Em seguida, é apresentada tanto as ferramentas computacionais como a bancada experimental utilizada para a aplicação das simulações em HIL. Na seção de metodologia de implementação de HIL, todo o processo de implantação, juntamente com as ferramentas computacionais adotadas são descritas de forma seqüencial, utilizando sempre como exemplo o modelo da BRAHMA para contextualizar o processo de desenvolvimento e testes. Na etapa final, são apresentados e analisados alguns exemplos de aplicação da técnica de HIL tanto de forma clássica como na híbrida.
- No Capítulo 6 são apresentadas as conclusões gerais e as considerações finais do trabalho.

Este texto ainda inclui três anexos referentes:

1. a modelagem e a dedução completa das equações dinâmicas de um dos dedos da BRAHMA;
2. a topologia do modelo dinâmico desenvolvido no software ADAMS[®];
3. ao código fonte das simulações HIL.

2. PROJETO DE UMA MÃO ARTIFICIAL ROBÓTICA

2.1 INTRODUÇÃO E JUSTIFICATIVA

A mão humana representa um grande desafio para a robótica devido à sua flexibilidade, destreza e, por consequência, grande potencialidade de aplicações. Quando se pensa em termos de cooperação entre homem e máquina e no uso de mãos artificiais como próteses, surgem novos desafios como o conforto, facilidade de uso e integração.

Filósofos antigos como Anaxágoras (500?-428 ac) e Aristóteles (384-322 ac) já debatiam a respeito da relação entre a mão e a mente humana, pois tanto uma como outra fornecem características ao ser humano que o diferencia dos outros animais. As pautas dos debates eram, principalmente, filosofar se foi por causa da habilidade de manipulação da mão que o ser humano se tornou inteligente, ou ao contrário. Paleontologistas mais recentemente (BICCHI, 2000), mostraram que a destreza mecânica da mão humana foi um fator preponderante no desenvolvimento de um cérebro superior do homo sapiens.

Quando os seres humanos capturam um objeto, parte do procedimento adotado é preestabelecida e parte do procedimento é otimizada instantaneamente. Os dedos posicionam-se de forma coerente e forças são aplicadas de modo a não permitir que o objeto caia ou deslize. Muitos autores, segundo Valero (2000), afirmam ser a mão humana uma das partes mais evoluídas do corpo humano, capaz de interagir, de forma versátil, com o meio ambiente, através de movimentos, de sensações de tato, de controle de forças e outras habilidades que muitos pesquisadores nesta área gostariam de emular em garras robóticas e próteses de mão humana.

Trabalhos anteriores em mãos artificiais foram desenvolvidos para aplicações clínicas e não se adaptam bem ao problema da análise de movimentos e dos respectivos comandos e

acionamentos. Muitos destes trabalhos foram realizados para predizer as forças aplicadas em músculos e tendões (CHAO et al., 1976), (COONEY et al., 1977), (BERME et al.1977), ou, exploram considerações importantes para o *design* de órteses ou para cirurgias reconstrutivas.

Em nenhuma das referências foi possível encontrar uma análise apropriada da manipulação habilidosa de objetos ou mesmo da utilização de ferramentas genéricas. Dados existentes sobre antropometria e resistência (ARMSTRONG, 1982) são de grande utilidade para o *design* de ferramentas industriais. Poucos dados relacionados à cinemática antropométrica foram coletados. Em An et al.(1982) são apresentados dados sobre o comprimento médio das falanges proximais e distais para quatro dedos, mas estes dados não se relacionam com as articulações do punho nem definem as posições das juntas metacarpo-falangeanas entre si. Os dados são importantes para comparar atividades normais da mão com atividades anormais, mas, devido às interações entre as ações dos cinco dedos, a mão e o objeto que se deseja manipular, necessita ser tratado como uma estrutura cinemática única. Podendo inclusive, assumir diferentes topologias, dependendo do estabelecimento ou do desaparecimento de contato dos dedos com os objetos. O ambiente proposto neste trabalho trata a mão como um todo e utiliza métodos baseados na dinâmica de sistemas multicorpos (CRAIG, 1986), (GILLESPIE, 1996), (NGUYEN, 2002), para predizer, de maneira aproximada, o comportamento da mão humana.

Estudos mostram que, aproximadamente 70% das próteses e órteses de membros superiores são abandonadas após pouco tempo de uso (SCHERBINA, 2002), (FRASER, 1993), (KEJELLA, 1993). Os benefícios alcançados com o uso destes equipamentos ainda são pequenos, se comparados ao esforço de treinamento, adaptação e, principalmente, aos resultados obtidos.

O projeto BRAHMA (*Brazilian Anthropomorphic Hand*) compreende um novo conceito de mão artificial, cujo mecanismo se baseia nas características construtivas, funcionais e

motoras de uma mão humana. Não existe junta do tipo pino; as articulações atuam por contato, semelhante ao mecanismo biofísico. Além disso, toda a parte estrutural é constituída de material polimérico biocompatível.

Baseando-se em características anatômicas e biomecânicas da mão humana, o projeto BRAHMA inicia-se com a observação e análise do funcionamento de uma mão humana natural. Ao longo da concepção do projeto, algumas simplificações, em relação à anatomia real, se fazem necessárias, não simplesmente para facilitar, mas principalmente para viabilizar a construção de um protótipo.

Apesar de a mão humana apresentar um total de 23 graus de liberdade (g.d.l), a concepção atual da BRAHMA apresenta 20 g.d.l., pois os movimentos do punho são desconsiderados para focar única e exclusivamente no desenvolvimento da movimentação dos dedos. A movimentação do punho pode ser obtida posteriormente, acoplando a BRAHMA a um braço robótico.

2.2 DESENVOLVIMENTO DA ESTRUTURA DA BRAHMA

A Figura 2.1 mostra as juntas interfalangeais e metacarpo-falangeais da mão humana, de certa forma, simplificada. Da observação deste mecanismo em Neumann (2002), surgiu a proposta de simplificar ainda mais este perfil, utilizando um perfil de cela nas juntas da BRAHMA (Figura 2.2 e Figura 2.3).

Figura 2.1: Vista dorsal da junta interfalangial e metacarpo-falangial retirada de NEUMANN, (2002)

Os ossos metacarpais e do carpo, que compõe a palma da mão humana, apresentam movimentos relativos entre si. Contudo, por motivos de simplicidade neste projeto, as estruturas correspondentes a estes “ossos” estão fixas umas em relação às outras (Figura 2.2).

Figura 2.2: Representação de um dedo genérico

O projeto da estrutura passou por um processo de aprimoramento até atingir um nível de similaridade com os ossos da mão humana. Na última versão (Figura 2.3), o protótipo virtual atingiu o nível satisfatório desejado de similaridade com a mão humana, adotando superfícies côncavas nas extremidades distais dos ossos. Além disso, foram concebidos canais na superfície dos ossos para o alojamento dos cabos - que tem a mesma função de tendão e músculo, sendo assim os responsáveis por manter o equilíbrio dinâmico e movimentação relativa entre ossos – que foram criados para evitar a saliência dos cabos na superfície da mão, obtendo assim uma melhor estética no projeto.

Figura 2.3: Diferentes vistas da versão final do protótipo virtual da BRAHMA

2.3 ANÁLISE CINEMÁTICA DOS DEDOS

Com exceção do polegar, todos os dedos da mão artificial podem ser tratados de forma análoga. Inicialmente, desenvolveu-se o raciocínio para um dedo genérico e, posteriormente, o polegar foi tratado como um caso particular (CAURIN et al., 2003).

Nesta etapa serão considerados apenas os três últimos corpos do dedo, ou seja, a falange proximal, medial e distal. Com base na geometria da mão artificial verificamos que, independentemente do movimento executado, estes corpos se mantêm coplanares. O sistema de coordenadas associado a cada corpo i tem sua origem posicionada na articulação anterior do mesmo, encontrando-se o versor $\vec{x}_i$ alinhado com o eixo longitudinal do corpo. O versor $\vec{z}_i$ tem a direção do eixo de rotação da articulação correspondente e o versor $\vec{y}_i$ é definido pelo produto vetorial de $\vec{z}_i$ com $\vec{x}_i$.

Figura 2.4: Representação dos três últimos corpos do dedo, sendo: ab = falange proximal; ac = falange medial e ad = falange distal.

A posição da extremidade P em relação ao referencial 6 pode ser expressa diretamente pelo vetor:

$${}^6\vec{I}_P = \begin{bmatrix} a_D \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\text{em 4 componentes}} {}^6\vec{I}_P = \begin{bmatrix} a_D \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (2.1)$$

Observando a Figura 2.4, podem-se determinar as matrizes de transformação que nos interessam.

Matriz de transformação de 6 em 5:

$${}^5_6T = \begin{bmatrix} \cos\theta_6 & -\sin\theta_6 & 0 & a_C \\ \sin\theta_6 & \cos\theta_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.2)$$

Matriz de transformação de 5 em 4:

$${}^4_5T = \begin{bmatrix} \cos\theta_5 & -\sin\theta_5 & 0 & a_B \\ \sin\theta_5 & \cos\theta_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

Matriz de transformação de 4 em 3:

$${}^3_4T = \begin{bmatrix} \cos\theta_4 & -\sin\theta_4 & 0 & 0 \\ \sin\theta_4 & \cos\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.4)$$

Para fins de simplificação, os elementos que compõem a BRAHMA são tratados como corpos rígidos. Como as referenciais 2 e 3 são fixas, o ângulo φ_m descrito na Figura 2.5 resulta de uma característica geométrica da mão artificial, ou seja, φ_m não é uma variável.

Figura 2.5: Representação geométrica do ângulo φ_m .

Figura 2.6: Modelagem da parte referente à “palma da mão” - do metacarpo ao punho

Com base na Figura 2.6, a matriz de transformação que relaciona os referenciais 3 e 2 pode ser escrita como:

$${}^2_3\mathbf{T} = \begin{bmatrix} \cos(\theta_3 + \varphi_m) & 0 & \sin(\theta_3 + \varphi_m) & a_A \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_3 + \varphi_m) & 0 & \cos(\theta_3 + \varphi_m) & d_A \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.5)$$

O dedo polegar difere um pouco dos demais, por apresentar o eixo z do referencial fixo à sua falange rodado de φ_p em torno do eixo x preso a este mesmo corpo. Assim, o desenvolvimento do cálculo para o polegar sugere o uso de uma transformação adicional, que leve em consideração o ângulo φ_p . Realiza-se então uma transformação de rotação em torno de x_3 , do referencial 3 em um novo referencial 3':

$${}_{3'}^3T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \varphi_p & -\sin \varphi_p & 0 \\ 0 & \sin \varphi_p & \cos \varphi_p & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.6)$$

Figura 2.7: Representação geométrica ângulo φ_p no dedo polegar.

Para manter a coerência das notações, chamamos 2_3T de ${}^2_{3'}T$, ou seja:

$${}^2_{3'}T = \begin{bmatrix} \cos(\theta_3 + \varphi_m) & 0 & \sin(\theta_3 + \varphi_m) & a_A \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_3 + \varphi_m) & 0 & \cos(\theta_3 + \varphi_m) & d_A \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.7)$$

Como o ângulo φ_p , assim como o φ_m , resulta das características construtivas da mão artificial, ele não é variável.

Para o mecanismo de um dedo, a cinemática direta consiste na determinação das coordenadas retangulares da ponta do dedo a partir dos valores das coordenadas generalizadas $\theta_1(t)$, $\theta_2(t)$, $\theta_3(t)$, $\theta_4(t)$, $\theta_5(t)$ e $\theta_6(t)$.

Assim, por meio de uma composição das matrizes de transformação encontradas anteriormente, podemos expressar a posição da ponta P em relação ao referencial inercial 0:

$${}^0\vec{r}_P = \begin{bmatrix} {}^0x_p \\ {}^0y_p \\ {}^0z_p \\ 1 \end{bmatrix} = {}^0T_1 \cdot {}^1T_2 \cdot {}^2T_3 \cdot {}^3T_3' \cdot {}^3T_4 \cdot {}^4T_5 \cdot {}^5T_6 \cdot {}^6T_P \vec{r}_P \quad (2.8)$$

Figura 2.8: Notação adotada para cada dedo

Para se identificar cada dedo, é necessário definir uma notação, conforme mostrado na Figura 2.8:

- Dedo I: Mínimo
- Dedo II: Anular
- Dedo III: Médio
- Dedo IV: Indicador
- Dedo V: Polegar

Finalmente, para se aplicar o equacionamento do sistema a cada um dos dedos, basta substituir os valores usados pelos correspondentes a cada dedo, conforme mostra a tabela a seguir:

TABELA 2.1: NOMENCLATURA DE ÂNGULOS E DIMENSÕES PARA CADA DEDO.

	Dedo I	Dedo II	Dedo III	Dedo IV	Dedo V
d_A	$d_{A,I}$	$d_{A,II}$	$d_{A,III}$	$d_{A,IV}$	$d_{A,V}$
a_A	$a_{A,I}$	$a_{A,II}$	$a_{A,III}$	$a_{A,IV}$	$a_{A,V}$
a_B	$a_{B,I}$	$a_{B,II}$	$a_{B,III}$	$a_{B,IV}$	$a_{B,V}$
a_C	$a_{C,I}$	$a_{C,II}$	$a_{C,III}$	$a_{C,IV}$	$a_{C,V}$
a_D	$a_{D,I}$	$a_{D,II}$	$a_{D,III}$	$a_{D,IV}$	$a_{D,V}$
θ_6	$\theta_{6,I}$	$\theta_{6,II}$	$\theta_{6,III}$	$\theta_{6,IV}$	$\theta_{6,V}$
θ_5	$\theta_{5,I}$	$\theta_{5,II}$	$\theta_{5,III}$	$\theta_{5,IV}$	$\theta_{5,V}$
θ_4	$\theta_{4,I}$	$\theta_{4,II}$	$\theta_{4,III}$	$\theta_{4,IV}$	$\theta_{4,V}$
θ_3	$\theta_{3,I}$	$\theta_{3,II}$	$\theta_{3,III}$	$\theta_{3,IV}$	$\theta_{3,V}$
θ_2	θ_2	θ_2	θ_2	θ_2	θ_2
θ_1	θ_1	θ_1	θ_1	θ_1	θ_1
φ_m	φ_m, I	φ_m, II	φ_m, III	φ_m, IV	φ_m, V
φ_p	0	0	0	0	φ_p

Nota: É importante salientar que os ângulos e as dimensões acima devem respeitar o sentido dos referenciais adotados na dedução anterior. Note também que os ângulos θ_2 e θ_1 , por pertencerem à articulação do punho, apresentam valores iguais para todos os dedos da mão.

A cinemática inversa trata o processo inverso ao da cinemática direta. Ou seja, visa a determinação das coordenadas das articulações $\theta_1(t)$, $\theta_2(t)$, $\theta_3(t)$, $\theta_4(t)$, $\theta_5(t)$ e $\theta_6(t)$ - a partir das coordenadas retangulares da ponta do dedo.

2.3.1 Velocidades lineares

O procedimento para se determinar a velocidade linear da ponta do dedo consiste na diferenciação temporal do vetor posição ${}^0\vec{r}_p$:

$${}^0\vec{v}_p = \frac{d}{dt}({}^0\vec{r}_p) = \begin{bmatrix} {}^0\dot{x}_p \\ {}^0\dot{y}_p \\ {}^0\dot{z}_p \end{bmatrix} \quad (2.9)$$

2.3.2 Velocidades Angulares

A existência de velocidades relativas sugere a definição de uma nova notação que permita uma representação mais clara da relação entre os corpos ou referenciais envolvidos. Assim, adotemos:

$${}^A\vec{\omega}_{(B,C)u} \quad (2.10)$$

Onde:

- A é o referencial no qual o vetor velocidade angular está expresso.
- B, C indicam que o vetor em questão representa a velocidade angular do corpo (ou referencial) C em relação ao corpo (ou referencial) B .
- A direção da velocidade angular é especificada pelo índice u , que representa a direção do *versor* de A associado à ω .

Figura 2.9: Velocidades angulares relativas

De acordo com a Figura 2.9, podem-se definir as seguintes velocidades angulares relativas:

$${}^6\vec{\omega}_{(5,6)z} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_6 \end{bmatrix} \quad (2.11)$$

$${}^5\vec{\omega}_{(4,5)z} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_5 \end{bmatrix} \quad (2.12)$$

$${}^4\vec{\omega}_{(3,4)z} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_4 \end{bmatrix} \quad {}^4\vec{\omega}_{(3,4)y} = \begin{bmatrix} 0 \\ \dot{\theta}_3 \\ 0 \end{bmatrix} \quad (2.13)$$

$${}^1\vec{\omega}_{(0,1)z} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_2 \end{bmatrix} \quad {}^0\vec{\omega}_{(0,1)y} = \begin{bmatrix} 0 \\ \dot{\theta}_1 \\ 0 \end{bmatrix} \quad (2.14)$$

Para a determinação das velocidades angulares de cada corpo em relação ao referencial inercial, serão utilizadas as matrizes de rotação. Assim:

$${}^0\vec{\omega}_{(4,5)z} = {}^0C_{.2}^1 C_{.3'}^2 C_{.3}^3 C_{.4}^3 C_{.5}^4 \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_5 \end{bmatrix} \quad (2.15)$$

e

$${}^0\vec{\omega}_{(5,6)z} = {}^0C_{.2}^1 C_{.3'}^2 C_{.3}^3 C_{.4}^3 C_{.5}^4 C_{.6}^5 \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_6 \end{bmatrix} \quad (2.16)$$

A velocidade angular ${}^0\vec{\omega}_{(3,4)y}$ pode ser obtida fazendo-se o seguinte cálculo matricial:

$${}^0\vec{\omega}_{(3,4)y} = {}^0C_{.1}^1 C_{.2}^2 C_{.3'}^{3'} C_{.3}^3 \dot{\theta}_3 \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (2.17)$$

A velocidade angular ${}^0\vec{\omega}_{(0,1)y}$ não necessita de cálculo matricial, pois está já está expressa em relação ao referencial inercial 0.

Para se obter a velocidade angular ${}^0\vec{\omega}_{(0,1)z}$ basta multiplicar o vetor ${}^1\vec{\omega}_{(0,1)z}$ pela matriz de rotação 0C_1 :

$${}^0\vec{\omega}_{(0,1)z} = {}^0C_1 \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_2 \end{bmatrix} \quad (2.18)$$

Assim, a velocidade angular absoluta da falange distal em relação ao referencial inercial pode ser descrita como a soma vetorial das velocidades angulares determinadas anteriormente:

$${}^0\vec{\omega}_p = {}^0\vec{\omega}_{0,6} = {}^0\vec{\omega}_{(5,6)z} + {}^0\vec{\omega}_{(4,5)z} + {}^0\vec{\omega}_{(3,4)z} + {}^0\vec{\omega}_{(3,4)y} + {}^0\vec{\omega}_{(0,1)z} + {}^0\vec{\omega}_{(0,1)y} \quad (2.19)$$

Embora o desenvolvimento tenha sido feito para o dedo polegar, ele é também aplicado a todos os outros dedos da mão (casos em que $\varphi_p = 0$).

A fim de tornar o tratamento das grandezas de velocidade mais eficiente, será introduzido o uso da matriz Jacobiana, relacionando velocidades lineares de um ponto na extremidade de um dedo no espaço cartesiano com as velocidades angulares nas coordenadas das articulações (coordenadas generalizadas).

Fazendo uso das expressões de velocidade linear e velocidade angular apresentadas anteriormente, a relação com as coordenadas generalizadas pode ser expressa da seguinte forma:

$${}^0\vec{V} = \begin{bmatrix} {}^0V_{xP} \\ {}^0V_{yP} \\ {}^0V_{zP} \\ {}^0\omega_{x0,6} \\ {}^0\omega_{y0,6} \\ {}^0\omega_{z0,6} \end{bmatrix} = J \bullet \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \\ \dot{\theta}_5 \\ \dot{\theta}_6 \end{bmatrix} \quad (2.20)$$

Pela complexidade do sistema, cada um dos 36 elementos que compõe a matriz Jacobiana (J) representa uma expressão transcendental complexa. A Jacobiana é uma forma matricial de se representar os coeficientes das velocidades generalizadas, que expressam, neste caso, seis funções, cada qual com seis variáveis independentes. A matriz Jacobiana é um dos recursos mais importantes para a caracterização de um sistema robótico, sendo útil para encontrar posições singulares, determinar algoritmos de cinemática inversa, determinar a relação entre forças aplicadas na ponta dos dedos e os torques resultantes nas juntas e projetar estratégias de controle.

$$J = \begin{bmatrix} \frac{\partial {}^0r_{x_p}}{\partial \theta_1(t)} & \frac{\partial {}^0r_{x_p}}{\partial \theta_2(t)} & \frac{\partial {}^0r_{x_p}}{\partial \theta_3(t)} & \frac{\partial {}^0r_{x_p}}{\partial \theta_4(t)} & \frac{\partial {}^0r_{x_p}}{\partial \theta_5(t)} & \frac{\partial {}^0r_{x_p}}{\partial \theta_6(t)} \\ \frac{\partial {}^0r_{y_p}}{\partial \theta_1(t)} & \frac{\partial {}^0r_{y_p}}{\partial \theta_2(t)} & \frac{\partial {}^0r_{y_p}}{\partial \theta_3(t)} & \frac{\partial {}^0r_{y_p}}{\partial \theta_4(t)} & \frac{\partial {}^0r_{y_p}}{\partial \theta_5(t)} & \frac{\partial {}^0r_{y_p}}{\partial \theta_6(t)} \\ \frac{\partial {}^0r_{z_p}}{\partial \theta_1(t)} & \frac{\partial {}^0r_{z_p}}{\partial \theta_2(t)} & \frac{\partial {}^0r_{z_p}}{\partial \theta_3(t)} & \frac{\partial {}^0r_{z_p}}{\partial \theta_4(t)} & \frac{\partial {}^0r_{z_p}}{\partial \theta_5(t)} & \frac{\partial {}^0r_{z_p}}{\partial \theta_6(t)} \\ \frac{\partial {}^0\omega_{x_p}}{\partial \theta_1(t)} & \frac{\partial {}^0\omega_{x_p}}{\partial \theta_2(t)} & \frac{\partial {}^0\omega_{x_p}}{\partial \theta_3(t)} & \frac{\partial {}^0\omega_{x_p}}{\partial \theta_4(t)} & \frac{\partial {}^0\omega_{x_p}}{\partial \theta_5(t)} & \frac{\partial {}^0\omega_{x_p}}{\partial \theta_6(t)} \\ \frac{\partial {}^0\omega_{y_p}}{\partial \theta_1(t)} & \frac{\partial {}^0\omega_{y_p}}{\partial \theta_2(t)} & \frac{\partial {}^0\omega_{y_p}}{\partial \theta_3(t)} & \frac{\partial {}^0\omega_{y_p}}{\partial \theta_4(t)} & \frac{\partial {}^0\omega_{y_p}}{\partial \theta_5(t)} & \frac{\partial {}^0\omega_{y_p}}{\partial \theta_6(t)} \\ \frac{\partial {}^0\omega_{z_p}}{\partial \theta_1(t)} & \frac{\partial {}^0\omega_{z_p}}{\partial \theta_2(t)} & \frac{\partial {}^0\omega_{z_p}}{\partial \theta_3(t)} & \frac{\partial {}^0\omega_{z_p}}{\partial \theta_4(t)} & \frac{\partial {}^0\omega_{z_p}}{\partial \theta_5(t)} & \frac{\partial {}^0\omega_{z_p}}{\partial \theta_6(t)} \end{bmatrix} \quad (2.21)$$

2.3.3 Acelerações

A aceleração da ponta do dedo pode ser determinada pela diferenciação no tempo do vetor velocidade ${}^0\vec{v}_p$. Da mesma forma, a aceleração angular da falange distal pode também ser obtida pela diferenciação temporal do vetor velocidade ${}^0\omega_{0,6}$.

No entanto, para fins de implementação computacional, é interessante representarmos essas acelerações em um único vetor de dimensão seis, onde os três primeiros elementos correspondem às componentes da aceleração linear e os três últimos correspondem às componentes da aceleração angular, conforme mostrada abaixo:

$${}^0\vec{a} = \begin{bmatrix} {}^0\dot{v}_{xP} \\ {}^0\dot{v}_{yP} \\ {}^0\dot{v}_{zP} \\ {}^0\dot{\omega}_{x0,6} \\ {}^0\dot{\omega}_{y0,6} \\ {}^0\dot{\omega}_{z0,6} \end{bmatrix} \quad (2.22)$$

Podendo ser apresentada com relação às coordenadas das juntas da seguinte forma:

$${}^0\vec{a}_i = \mathbf{J}.\ddot{\theta}_i + \dot{\mathbf{J}}.\dot{\theta}_i \quad (2.23)$$

2.3.4 Simulações da Cinemática

A determinação analítica da cinemática inversa para cálculo de posição torna-se bastante complexa para um sistema com mais de três graus de liberdade e caracteriza-se, normalmente, pela existência de múltiplas soluções. Neste caso, a aplicação de métodos numéricos iterativos é a melhor alternativa para a resolução do problema (CAURIN et al., 2003).

As soluções encontradas, tanto para a cinemática direta como para a inversa, foram tratadas utilizando métodos numéricos iterativos, utilizando ferramentas como o Matlab® e ADAMS® para implementar as simulações.

É apresentado um exemplo em que os ângulos θ_1 , θ_2 , θ_3 do dedo indicador permaneçam constantes e os ângulos θ_4 , θ_5 , θ_6 variáveis. Como exemplo, foi imposto um movimento de 1 radiano no período de 1 segundo para as articulações variáveis do dedo indicador, como ilustrado na Figura 2.10. Para a segunda simulação, foi imposto um deslocamento linear na ponta do dedo indicador de -60 milímetros nas direções X e Y e uma variação em sua orientação de -0,785 radiano no período de 1 segundo. Os resultados de ambas as simulações encontram-se respectivamente nas Figuras 2.11 e 2.12.

Figura 2.10: Simulação do movimento nas articulações do dedo indicador.

Figura 2.11: Resultado das simulações. O gráfico à esquerda mostra o deslocamento angular imposto às articulações do dedo indicador. No gráfico à direita é apresentado o resultado dos deslocamentos lineares da ponta do dedo.

Figura 2.12: Resultado das simulações da cinemática inversa. O primeiro gráfico apresenta uma imposição de um movimento linear à ponta do dedo. No gráfico do meio apresenta a orientação deste movimento linear. O último gráfico mostra como resposta os deslocamentos angulares nas articulações.

3 DESENVOLVIMENTO DO MODELO DINÂMICO DA BRAHMA

Um Sistema Multicorpos (MBS) é definido como um sistema mecânico que possui dois ou mais corpos com vários graus de liberdade. Os movimentos de um MBS são governados por expressões matemáticas chamadas de equações dinâmicas. Estas equações são compostas por um conjunto de equações diferenciais eventualmente acrescido de algumas equações algébricas. As equações diferenciais descrevem os movimentos dos corpos rígidos e as equações e inequações algébricas levam em consideração restrições impostas pela geometria do sistema ou de seus movimentos, tais como a ligação de dois corpos por juntas, ou mesmo características peculiares de contato entre dois corpos.

A obtenção das equações dinâmicas para sistemas mecânicos era realizada manualmente no passado. Entretanto, devido ao aumento da complexidade com o qual se deseja estudar os novos sistemas, este processo tornou-se trabalhoso e passível de erros, por ser particularmente difícil acomodar modificações de projeto, ou mesmo pequenas variações no modelo. Atualmente conta-se com programas para geração automática das equações de movimento de MBS.

O modelo para a simulação (ALBUQUERQUE et al, 2005) deste trabalho foi obtido com o auxílio de um programa MBS numérico, chamado ADAMS[®] (MSC, 2007), que possui rotinas para a geração das equações e para a solução do sistema. Este programa possui aplicativos de pré e pós-processamento. Estas ferramentas facilitam a criação do modelo da BRAHMA e a posterior simulação com análise e apresentação gráfica dos resultados.

Neste capítulo são apresentados, de forma simplificada, o desenvolvimento, testes e análises do modelo completo do sistema BRAHMA. Uma análise mais aprofundada da dinâmica do projeto BRAHMA pode ser encontrada tanto em (ALBUQUERQUE et al, 2005) como no primeiro anexo deste trabalho. Neste anexo são apresentadas a modelagem e a

dedução completa das equações dinâmicas de um dos dedos da BRAHMA, além de uma análise da dinâmica do sistema como um todo, observando o quanto cada componente da equação dinâmica (inércia, gravidade e aceleração centrípeta) influencia no resultado final.

3.1 MODELAGEM DO SISTEMA BRAHMA

As equações de movimento da BRAHMA são obtidas pelo método de Lagrange, uma vez que as variáveis θ_n constituem um conjunto de coordenadas generalizadas, ou variáveis independentes que descrevem os movimentos das juntas de um dos dedos da BRAHMA. As equações de movimento segundo Lagrange podem ser escritas como:

$$\frac{d}{dt} \left[\frac{\partial L(\theta, \dot{\theta}, t)}{\partial \dot{\theta}_n} \right] - \frac{\partial L(\theta, \dot{\theta}, t)}{\partial \theta_n} = \tau_n - \beta_n \dot{q}_n \quad (3.1)$$

onde $n = 1, \dots, N$ juntas independentes, τ é o torque generalizado e β é o coeficiente de atrito nas articulações.

Deste modo, a função Lagrangeana, ou simplesmente o Lagrangeano, é determinada pela diferença entre a energia cinemática e a energia potencial do sistema, assumindo a seguinte forma:

$$L(\theta, \dot{\theta}, t) = K(\theta, \dot{\theta}, t) - P(\theta, t) \quad (3.2)$$

As equações de movimento são obtidas pela substituição da Eq. (3.2) na Eq. (3.1) para formar as derivadas apropriadas. O modelo dinâmico da BRAHMA pode ser expresso, resumidamente, por:

$$D(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) + G(\theta) = \tau \quad (3.3)$$

onde $\theta, \dot{\theta}, \ddot{\theta}$ são respectivamente os vetores de posição, velocidade e aceleração, D é a matriz de inércia generalizada, C representa os vetores das forças de Coriolis e Centrípeta, G representa o vetor das forças gravitacionais e, τ é vetor de torque aplicado nas articulações.

No modelo completo, deve-se levar em conta o acionamento das articulações da BRAHMA que é realizado por motores da MAXON. O modelo do motor é 118754, com 20 Watts de potência. O redutor, modelo 110394, é um planetário de três estágios e relação de transmissão de 19 : 1. Utiliza-se ainda um encoder HP (1998), modelo HEDS 5511, acoplado ao motor com três canais e 500 pulsos por ciclo.

O acoplamento entre um motor e uma articulação pode ser representado conforme a Figura 3.1, onde a carga é a representação simplificada da dinâmica de uma articulação do dedo. O ângulo de saída do motor é θ_m e θ_c o deslocamento angular de uma articulação do dedo. Caso o acoplamento seja considerado ideal, ou seja, com rigidez infinita (K_c), o valor de θ_c é igual ao valor de θ_m multiplicado pela relação de transmissão (η).

Figura 3.1: Esquema do sistema de acionamento –motor, acoplamento e carga.

Analisando o torque no motor DC, pode-se afirmar que:

$$\tau_m = D_m \ddot{\theta}_m + B_m \dot{\theta}_m + \frac{\tau_c}{\eta} \quad (3.4)$$

onde D_m é a inércia e B_m é o atrito viscoso no motor. O torque devido à carga pode ser representado conforme a equação abaixo:

$$\tau_c = D_c(\theta) \ddot{\theta} + C_c(\theta) \dot{\theta} + G_c(\theta) + K_c \left(\frac{\theta_m}{\eta} - \theta_c \right) + B_c \left(\frac{\dot{\theta}_m}{\eta} - \dot{\theta}_c \right) \quad (3.5)$$

Representando o sistema na forma de diagrama de blocos (Figura 3.2), cada elemento da dinâmica do sistema pode ser observado.

Figura 3.2: Representação do sistema na forma de diagrama de blocos

Desprezado o efeito da dinâmica do acoplamento no sistema, a equação principal pode ser escrita como:

$$\tau_m = D_m \ddot{\theta}_m + B_m \dot{\theta}_m + \left(\frac{D_c(\theta) \ddot{\theta} + C_c(\theta) \dot{\theta} + G_c(\theta)}{\eta} \right) \quad (3.6)$$

Substituindo as variáveis angulares das articulações θ por deslocamentos angulares nos motores θ_m , a equação fica:

$$\tau_m = D_m \ddot{\theta}_m + B_m \dot{\theta}_m + \left(\frac{D_c(\theta) \frac{\ddot{\theta}_m}{\eta} + C_c(\theta) \frac{\dot{\theta}_m}{\eta} + G_c(\theta)}{\eta} \right) \quad (3.7)$$

A análise da dinâmica não linear da BRAHMA realizada em Albuquerque et al. (2005) constatou que a influência dos componentes de inércia e de aceleração centrípeta é insignificante para este sistema dinâmico em uma faixa de velocidade de até 1 rad/s. Porém, com o aumento das velocidades, as respostas obtidas mostram que a influência destes componentes começa a se tornar significativa. Para velocidades angulares de 2 rad/s nas articulações dos dedos essa influência na dinâmica do sistema é de aproximadamente 5 %. Isso denota que, com o aumento da velocidade de operação da BRAHMA (acima de 2 rad/s), as influências dos componentes de inércia e de aceleração centrípeta começam a ter uma determinada significância na dinâmica do sistema, porém o efeito da gravidade no sistema permanece sendo o de maior influência na grande maioria das situações de operação.

Para analisar os efeitos das não linearidades na dinâmica completa do sistema (Eq. 3.6), o modelo foi linearizado em duas situações extremas. A primeira com os dedos totalmente verticalizados e a segunda com os dedos totalmente horizontalizados. A articulação $\theta_{4,III}$, foi a escolhida para se fazer a análise. Isso porque esta articulação tem uma maior influência de não linearidades por ser a última articulação do dedo de maior massa.

Os valores das raízes do sistema de segunda ordem para as duas situações são mostrados na tabela 3.1:

Tabela 3.1: Valores das raízes do sistema linearizado em duas situações extremas.

	Real (σ)	Imaginário ($j\omega$)
Mão na Vertical	- 0.096	0
	- 93,634	0
Mão da Horizontal	-0,002	0
	-93,535	0

Com esta análise, verifica-se que, mesmo em duas situações extremas, o sistema é praticamente o mesmo, ou seja, a não linearidade esperada não ficou evidenciada. Isso devido ao acoplamento dos motores com relação de transmissão ser razoavelmente alto (1:19).

3.1.1 MODELAGEM DA ARMADURA DO MOTOR DC

Os motores DC convertem a energia através da interação entre um campo magnético e um fluxo elétrico. O campo é produzido por um ímã, e o fluxo elétrico é induzido por uma corrente elétrica que passa pelas espiras da armadura. Como resultado, um torque é produzido e aplicado no eixo do rotor. Conforme o rotor gira, uma força contra eletromotriz é gerada, essa força é proporcional à velocidade do rotor, portanto, ao atingir o equilíbrio, a velocidade do rotor será proporcional à tensão gerada. Essa relação é dada por:

$$V_b = K_b \cdot \frac{d\theta_m}{dt} \quad (3.8)$$

Onde:

$\frac{d\theta_m}{dt}$ – Velocidade Angular do rotor [rad/s]

K_b – Constante contra-eletromotriz [rad/V/s]

V_b – Tensão Gerada [V]

Simplificando, o motor pode ser considerado uma associação em série de um resistor, um indutor e um gerador de força contra-eletromotriz conforme o esquema:

Figura 3.3: Circuito Elétrico de um Motor DC

Segundo a lei das malhas:

$$V_a = L \cdot \frac{dI_a}{dt} + R_a \cdot I_a + V_b \quad (3.9)$$

Onde:

V_a – Tensão aplicada [Volt]

R_a – Resistência da Armadura [Ohm]

L – Indutância da armadura [Henry]

V_b – Força contra eletromotriz [Volt]

I_a – Corrente Aplicada [Ampere]

Substituindo a Eq. 3.8 na Eq. 3.9:

$$V_a = L \cdot \frac{dI_a}{dt} + R_a \cdot I_a + K_b \cdot \frac{d\theta_m}{dt} \quad (3.10)$$

Nos motores DC utilizados na robótica (motores de pequeno porte), o campo magnético é gerado por um ímã permanente, logo, não é necessário criar um campo magnético. O fluxo magnético no estator permanece constante para todas as correntes aplicadas na armadura. Dessa forma a curva velocidade por torque do motor é linear segundo a Eq. 3.11.

Como resultado da interação entre os campos magnéticos, uma corrente elétrica flui através das espiras e um torque, proporcional a corrente aplicada, é gerado.

$$\tau_m = K_m \cdot I_a \quad (3.11)$$

Onde:

τ_m - Torque no motor [Nm]

K_m – Constante elétrica do motor [Nm/A]

A relação entre o torque aplicado e giro do eixo do rotor é dada pela substituição da Eq. 3.11 na Eq. 3.6, ficando:

$$K_m \cdot I_a = D_m \cdot \ddot{\theta}_m + B_m \cdot \dot{\theta}_m + \left(\frac{D_c}{\eta^2} \cdot \ddot{\theta}_m + \frac{C_c}{\eta^2} \cdot \dot{\theta}_m \right) + \frac{G_c}{\eta} \quad (3.12)$$

Com intuito de representar a interação de um motor DC em cada uma das articulações da mão artificial, optou-se por determinar a função transferência que relacionasse deslocamento angular de uma articulação com a tensão aplicada em um atuador. Para tanto, as equações são escritas no domínio da frequência por meio de transformações de Laplace, considerando as condições iniciais nulas.

$$K_m \cdot I_a(s) = \left(\left(D_m + \frac{D_c}{\eta^2} \right) s^2 + \left(B_m + \frac{C_c}{\eta^2} \right) s \right) \cdot \theta_m + \frac{G_c}{\eta} \quad (3.13)$$

Agora escrevendo a Eq. 3.10 no domínio da frequência:

$$V_a(s) = L \cdot I_a \cdot s + R_a \cdot I_a + K_b \cdot \theta_m \cdot s \quad (3.14)$$

Isolando-se a corrente I_a :

$$I_a(s) = \frac{V_a - K_b \cdot \theta_m \cdot s}{L \cdot s + R_a} \quad (3.15)$$

Substituindo 3.15 em 3.13:

$$K_m \cdot \left(\frac{V_a - K_b \cdot \theta_m \cdot s}{L \cdot s + R_a} \right) = \left(\left(D_m + \frac{D_c}{\eta^2} \right) \cdot s^2 + \left(B_m + \frac{C_c}{\eta^2} \right) \cdot s \right) \cdot \theta_m + \frac{G_c}{\eta} \quad (3.16)$$

A função transferência (FT) da saída (posição angular do eixo) pela entrada (tensão aplicada na armadura) é dada por:

$$\frac{\theta_m(s)}{V_a(s)} = \left(\frac{\tau_m(s) + \frac{G_c}{\eta}}{V_a(s)} \right) \cdot \left(\frac{\theta_m(s)}{\tau_m(s) + \frac{G_c}{\eta}} \right) \quad (3.17)$$

Na forma de diagrama de blocos representa-se o conjunto motor e articulação de um dedo da BRAHMA, da seguinte forma:

Figura 3.4: Representação do sistema completo na forma de diagrama de blocos (Eq. 3.17)

Na prática, a função de transferência do sistema elétrico (armadura) - que tem como variável de saída o torque no motor e a variável de entrada a tensão elétrica na armadura - exerce uma influência insignificante no sistema completo. Isso porque a constante de tempo

do modelo de primeira ordem para o motor utilizado no projeto (Eq. 3.15), que representa a função de transferência, é da ordem de 0,11 milissegundos, tendo o valor da raiz por volta de 9100 Hertz. Comparando as raízes relativas ao sistema mecânico (Tabela 3.1), verificam-se o afastamento entre as raízes de ambos os sistemas (mecânico e elétrico). Com isso, conclui-se que a função de transferência do sistema elétrico pode ser substituída, sem maiores prejuízos, por um simples ganho proporcional.

3.2 IMPLEMENTAÇÃO COMPUTACIONAL

As equações de movimento obtidas são equações diferenciais não-lineares de segunda ordem, cuja variável independente é o tempo. Para resolvê-las adota-se um integrador numérico denominado GSTIFF (NIKRAVESH, 1988). Sendo este um integrador de ordem e passo variado com ordem máxima de integração igual a seis.

3.2.1 INTEGRADOR

Existem diversos tipos de integradores para resolver as equações de movimento e de restrições dos sistemas modelados que podem ser classificados a grosso modo como: os rígidos (*stiff*) e os não rígidos (*non-stiff*). Um integrador rígido é aquele que pode manipular numericamente sistemas de corpos rígidos. Já o integrador não rígido é mais eficiente para a manipulação numérica de corpos flexíveis (NIKRAVESH, 1988).

Para integradores rígidos como é o caso do GSTIFF, usado no presente trabalho, o passo de integração é limitado ao inverso da maior frequência ativa do sistema. Para os não rígidos o passo de integração é limitado ao inverso da maior frequência, ativa ou não do sistema.

Devido ao fato de muitos sistemas mecânicos serem considerados corpos rígidos, o integrador GSTIFF é um dos mais utilizados, sendo este um integrador rígido baseado no trabalho de Gear (1971) que se baseia no BDF “*Backward Difference Formulate*” e integradores multi-passos. A solução para este integrador ocorre em duas fases: uma predição seguida de uma correção.

A solução para as equações dinâmicas geradas é obtida através de algoritmos que utilizam a técnica conhecida como Predição/Correção (NIKRAVESH, 1988). Esta técnica prediz a solução em um ponto futuro no tempo através de funções polinomiais que utilizam soluções já computadas de pontos anteriores no tempo e realizam a extrapolação. A solução é uma aproximação e geralmente deve ser melhorada ou corrigida de modo a satisfazer uma especificação desejada de tolerância.

A Predição é um processo explícito no qual somente valores passados são considerados e, é baseado na premissa de que os valores passados são bons indicadores para que valores correntes sejam computados. Não há garantia que o valor previsto satisfaça as equações de movimento e de restrições. É simplesmente um valor sugerido para se iniciar a correção, do qual se garante que as equações governantes sejam satisfeitas.

As fórmulas relacionadas ao Corretor são um ajuste implícito de relações de diferença (BDFs), isso relaciona a derivada dos estados no momento atual com os valores de seus estados futuros. Esta relação transforma as equações diferenciais não lineares em um ajuste não linear algébrico de equações de diferença dos estados do sistema. O integrador de Euler é um exemplo de um BDF de primeira-ordem. Dada as equações diferenciais ordinárias da forma $\frac{dx}{dt} = f(x, t)$, o integrador de Euler utiliza a relação de diferença:

$$x_{n+1} = x_n + h\dot{x}_{n+1} \quad (3.18)$$

Onde:

x_n é a solução calculada em $t = t_n$

h é o tamanho do passo sendo utilizado

x_{n+1} é a solução em t_{n+1} , sendo computada

Note que $n + 1$ está em ambos os lados da Eq. 3.18. Isto é um método implícito.

Utiliza-se um algoritmo iterativo de Newton-Raphson (NIKRAVESH, 1988) para resolver as equações de diferença e obter os valores de variáveis de estado. Este algoritmo assegura que os estados dos sistemas satisfaçam às equações de movimento e restrição. As iterações de Newton-Raphson requerem o uso de uma matriz Jacobiana a ser utilizada e recalculada a cada iteração para efetuar as correções dos estados.

Assume-se que as equações de movimento têm agora a seguinte forma:

$$F(x, \dot{x}, t) = 0 \quad (3.19)$$

Onde x representa todos os estados do sistema.

A linearização da Eq. 3.18 sobre um ponto de operação $x = x^k$ e $\dot{x} = \dot{x}^k$ dado:

$$F(x, \dot{x}, t) = F(x^k, \dot{x}^k, t) + \left. \frac{\partial F}{\partial x} \right|_{x^k, \dot{x}^k} (x - x^k) + \left. \frac{\partial F}{\partial \dot{x}} \right|_{x^k, \dot{x}^k} (\dot{x} - \dot{x}^k) = 0$$

substituindo $x - x^k$ com Δx , se obtém:

$$F(x^k, \dot{x}^k, t) + \left. \frac{\partial F}{\partial x} \right|_{x^k, \dot{x}^k} (\Delta x) + \left. \frac{\partial F}{\partial \dot{x}} \right|_{x^k, \dot{x}^k} (\Delta \dot{x}) = 0 \quad (3.20)$$

Da equação BDF de primeira ordem, pode-se obter a seguinte relação:

$$\Delta \dot{x} = \frac{1}{h} \Delta x \quad (3.21)$$

Substituindo a Eq. 3.20 na Eq. 3.19, se obtém:

$$\left[\frac{\partial F}{\partial x} \Big|_{x^k, \dot{x}^k} + \frac{1}{h} \frac{\partial F}{\partial \dot{x}} \Big|_{x^k, \dot{x}^k} \right] \Delta x = -F(x^k, \dot{x}^k, t) \quad (3.22)$$

Uma generalização da Eq. 3.21 para BDFs de ordem maior, é dado por:

$$\left[\frac{\partial F}{\partial x} \Big|_{x^k, \dot{x}^k} + \frac{1}{h\beta_0} \frac{\partial F}{\partial \dot{x}} \Big|_{x^k, \dot{x}^k} \right] \Delta x = -F(x^k, \dot{x}^k, t) \quad (3.23)$$

Onde:

β_0 é uma função de ordem de integração

A matriz do lado esquerdo da Eq. 3.23 é a matriz Jacobiana de F

Δx são as correções

F é o resíduo das equações

A convergência do corretor ocorre quando o resíduo F e as correções Δx vão se tornando cada vez menores.

Após o corretor ter convergido para uma solução, o integrador estima o erro de integração local na solução. Este é, normalmente, uma função da diferença entre o valor previsto e o valor corrigido, o tamanho do passo e a ordem do integrador. Se o erro calculado é maior que o erro de integração especificado, o integrador rejeita a solução e dá um passo de tempo menor. Se o erro calculado é menor que o erro de integração local especificado, o integrador aceita a solução e dá um passo de tempo novo. O integrador repete o processo estimação Predição/Correção do erro, até alcançar o tempo especificado dentro do comando de simulação imposto pelo usuário.

A premissa para o uso da Predição como uma suposição inicial para a Correção é violada severamente, quando descontinuidades acontecem ou quando grandes forças são impostas ao modelo. Exemplos destes fenômenos são: contatos, atrito e estados de transição. Nestes casos, é necessária certa experiência do usuário para minimizar os erros de integração, alterando a ordem e o passo de integração referente à ordem do polinômio usado para a solução.

4 SISTEMA DE CONTROLE

Neste capítulo é apresentado o desenvolvimento de um controle seguidor multivariável de posição de um modelo linear do sistema de três dedos da BRAHMA acrescidos de seus respectivos atuadores (com 12 graus de liberdade) por meio de realimentação de estados. A topologia deste modelo encontra-se no anexo 2 deste trabalho.

É utilizado um modelo linearizado de três dedos da BRAHMA, gerados a partir do *software* de modelagem dinâmica ADAMS[®], contendo doze entradas (torques nas articulações), e vinte e quatro saídas, das quais doze são os deslocamentos angulares nas articulações e as outras doze, a derivada desses deslocamentos (Albuquerque et al., 2004a).

4.1 MODELO EM ESPAÇO DE ESTADOS

Os métodos de controle desenvolvidos neste trabalho estão baseados no domínio do tempo e, por consequência da quantidade de entradas e saídas, o sistema será representado na forma de variáveis de estado.

A grande vantagem do uso da notação matricial é que as equações diferenciais de ordem n podem ser representadas por equações matriciais diferenciais de primeira ordem.

O modelo linearizado em variáveis de estados é utilizado para desenvolver o controle seguidor multivariável por realimentação de estados. A matriz A possui dimensões $n \times n$, onde n é o número de estados, sendo igual a duas vezes o número de graus de liberdade do sistema. Neste caso, a dimensão é 24. A matriz B possui dimensões $n \times m$ onde m é igual a 12, que é o número de entradas do sistema. A matriz C é $p \times n$, onde p é o número de saídas do sistema, sendo esta igual a 24, e a matriz D é: $p \times m$ com todos os seus elementos nulos.

O sistema linearizado de malha aberta é representado abaixo pelas equações de estado de n-ésima ordem e as equações de saída de p-ésima ordem.

$$\dot{x} = A.x + B.u \quad (4.1)$$

$$y = C.x \quad (4.2)$$

Foi atribuído que o vetor de entrada (u) no sistema são as tensões aplicadas nas armaduras dos motores e, o vetor de saída (y) são as posições e velocidades angulares nas articulações dos dedos.

Figura 4.1: Planta do sistema representada na forma de espaço de estados

Por definição, o estado de um sistema dinâmico é o menor conjunto de variáveis (variáveis de estado) tal que o conhecimento destas variáveis em $t = t_0$, juntamente com a entrada do sistema para $t \geq t_0$, determina completamente o comportamento do sistema para qualquer instante (Figura 4.1). Já as variáveis de estado (x) podem ser definidas como o menor conjunto de variáveis que determina o estado do sistema dinâmico.

4.2 SISTEMA DE CONTROLE SEGUIDOR

Nesta seção tratar-se-á de realimentação de estado, de modo a alterar a estrutura original do sistema, em termos de autovalores e autovetores.

O sistema linearizado de malha aberta é representado abaixo pelas equações de estado de n-ésima ordem (Eq. 4.1) e as equações de saída de p-ésima ordem (Eq. 4.3).

$$y = C.x = \begin{bmatrix} E \\ F \end{bmatrix}.x \quad (4.3)$$

onde y é um vetor $p \times 1$ e $w = Ex$ é um vetor $m \times 1$ representando as saídas que são requeridas para seguir o vetor de entrada θ .

O controle por realimentação de estados é projetado com o intuito de que o vetor w siga o comando de entrada θ , quando este for um comando de entrada constante por partes. Segundo D'Azzo e Houpis, (1995), o método do projeto consiste em adicionar um vetor comparador e integrador que satisfaça à seguinte equação:

$$\dot{z} = \theta - w = \theta - Ex \quad (4.4)$$

O sistema de malha aberta é, então, governado pelas equações aumentadas de estado e saída formadas a partir da Eq. 4.4 à Eq. 4.6.

$$\begin{aligned} \begin{bmatrix} \dot{x} \\ \dot{z} \end{bmatrix} &= \begin{bmatrix} A & 0 \\ -E & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix}.u + \begin{bmatrix} 0 \\ I \end{bmatrix}.\theta \\ &= \bar{A}x' + \bar{B}u + \bar{B}'\theta \end{aligned} \quad (4.5)$$

$$y = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} = \bar{C}x'$$

Em (D'AZZO; HOUPIS, 1995) é mostrado que a lei de controle a ser usada é:

$$u = K_1x + K_2z = \begin{bmatrix} K_1 & K_2 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \quad (4.6)$$

$$\bar{K} = \begin{bmatrix} K_1 & K_2 \end{bmatrix} \quad (4.7)$$

Satisfeita a condição de controlabilidade e observabilidade (D'AZZO E HOUPIS, 1995), o diagrama representando o sistema de controle por realimentação, consistindo das equações

de estado e de saída dada pela Eq. 4.5 e a lei de controle dada pela Eq. 4.6, é mostrado na Figura 4.2. Esta lei de controle atribui o espectro de autovalores e autovetores de malha fechada. A equação de malha fechada é:

$$\begin{aligned} \dot{\mathbf{x}}' &= \begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}} \end{bmatrix} = \begin{bmatrix} \mathbf{A} + \mathbf{BK}_1 & \mathbf{BK}_2 \\ -\mathbf{E} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{z} \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{I} \end{bmatrix} \boldsymbol{\theta} \\ &= \mathbf{A}'_{cl} \mathbf{x}' + \mathbf{B}'_r \mathbf{r} . \end{aligned} \quad (4.8)$$

Figura 4.2: Sistema de Controle Seguidor

A obtenção da matriz $\overline{\mathbf{K}}$ é realizada a partir da seleção dos autovalores a serem atribuídos à matriz da planta de malha fechada $\mathbf{A}'_{cl}$ na Eq. 4.6.

$$\sigma(\overline{\mathbf{A}} + \overline{\mathbf{B}}\overline{\mathbf{K}}) = \{\lambda_1, \lambda_2, \dots, \lambda_{n+m}\} \quad (4.9)$$

e um conjunto associado de autovetores

$$v(\overline{A} + \overline{B}\overline{K}) = \{v_1, v_2, \dots, v_{n+m}\} \quad (4.10)$$

que são selecionados afim de se obter as características desejadas de resposta no tempo. Os autovalores e autovetores são relacionados pela equação:

$$[\overline{A} + \overline{B}\overline{K}] v_i = \lambda_i v_i \quad (4.11)$$

que pode ser colocada na forma:

$$[\overline{A} - \lambda_i I \quad \overline{B}] \begin{bmatrix} v_i \\ g_i \end{bmatrix} = 0 \quad (4.12)$$

para $i = 1, 2, \dots, n + m$,

Para satisfazer à Eq. 4.12, o vetor $[v_i \ g_i]^T$ deve pertencer ao núcleo de:

$$\overline{S}(\lambda_i) = [\overline{A} - \lambda_i I \quad \overline{B}] \text{ para } i = 1, 2, \dots, n + m. \quad (4.13)$$

A notação $\ker \overline{S}(\lambda_i)$ é usada para definir o espaço nulo que contém todos os vetores $[v_i \ g_i]^T$ para que a Eq. 4.12 seja satisfeita (D'AZZO; HOUPIS, 1995). A Eq. 4.14 pode ser usada para formar a igualdade matricial

$$[g_1 \ g_2 \ \dots \ g_{n+m}] = [\overline{K}v_1 \ \overline{K}v_2 \ \dots \ \overline{K}v_{n+m}] \quad (4.14)$$

da onde se obtém a matriz $\overline{K}$ como segue:

$$\overline{K} = [g_1 \ g_2 \ \dots \ g_{n+m}] [v_1 \ v_2 \ \dots \ v_{n+m}]^{-1} = QV^{-1} \quad (4.15)$$

Os autovalores podem ter valores repetidos de número igual às entradas do sistema. Isso se deve ao fato do espaço nulo ter dimensão igual ao número de entradas. Assim associa-se

um autovalor repetido a um vetor da base do espaço nulo. Com isso, todas as colunas da matriz V continuam sendo linearmente independentes.

4.3 RESULTADOS PARA ENTRADA DEGRAU

Para a implementação do controle, adotou-se um conjunto de autovalores para a planta em malha fechada que se aproximasse do comportamento de uma mão natural com relação à velocidade de resposta a um estímulo:

$$\sigma(\overline{A} + \overline{B}\overline{K}) = \left\{ \underbrace{-40, \dots, -40}_{8 \text{ autovalores}}, \underbrace{-50, \dots, -50}_{8 \text{ autovalores}}, \underbrace{-60, \dots, -60}_{8 \text{ autovalores}} \right\} \quad (4.16)$$

Atribuindo autovalores reais negativos, significa que estamos impondo um comportamento criticamente amortecido ao sistema em malha fechada.

Os valores em radiano de entrada atribuídos para serem seguidos pelo sistema de controle foram:

$$\theta_d = \{0,1;0,2;0,3;0,4;0,13;0,5;0,6;0,7;0,16;0,8;0,9;1\} \quad (4.17)$$

Figura 4.3: Sinais de Entrada Degrau em cada uma das articulações.

A seguir alguns resultados para este conjunto de entradas de teste são apresentados:

Figura 4.4: Deslocamentos angulares das articulações dos dedos

Figura 4.5: Medição dos Erros (diferença entre as figuras 4.3 e 4.4)

Figura 4.6: Velocidades angulares das articulações dos dedos

Figura 4.7: Torques das articulações dos dedos

Figura 4.8: Sinal do Controle

4.4 RESULTADOS PARA ENTRADA PULSADA

Figura 4.9: Sinais de Entrada Degrau

Figura 4.10: Deslocamentos angulares das articulações dos dedos

Figura 4.11: Medição do Erro (diferença entre as figuras 4.9 e 4.10)

Figura 4.12: Velocidades angulares das articulações dos dedos

Figura 4.13: Torques das articulações dos dedos

Figura 4.14: Sinal do Controle

4.5 RESULTADOS PARA ENTRADA DE MEIA ONDA SENOIDAL COM FREQUÊNCIA DE 5 RAD/S

Sinais de Entrada

Figura 4.15: Sinais de Entrada Senoidal com frequência de 5 rad/s

Figura 4.16: Deslocamentos angulares das articulações dos dedos

Figura 4.17: Medição do Erro (diferença entre as Figuras 4.15 e 4.16)

Figura 4.18: Velocidades angulares das articulações dos dedos

Figura 4.19: Torques das articulações dos dedos

Figura 4.20: Sinal do Controle

4.6 RESULTADOS PARA ENTRADA DEGRAU COM DIFERENTES ATRIBUIÇÕES DE AUTOVALORES

Com intuito de verificar o tempo de resposta do sistema para diferentes atribuições de autovalores, adotou-se 3 conjuntos de autovalores para a planta em malha fechada:

$$\begin{aligned}
 (1) \quad \sigma(\bar{A} + \bar{B}\bar{K}) &= \left\{ \underbrace{-20, \dots, -20}_{8 \text{ autovalores}}, \underbrace{-30, \dots, -30}_{8 \text{ autovalores}}, \underbrace{-40, \dots, -40}_{8 \text{ autovalores}} \right\} \\
 (2) \quad \sigma(\bar{A} + \bar{B}\bar{K}) &= \left\{ \underbrace{-40, \dots, -40}_{8 \text{ autovalores}}, \underbrace{-50, \dots, -50}_{8 \text{ autovalores}}, \underbrace{-60, \dots, -60}_{8 \text{ autovalores}} \right\} \\
 (3) \quad \sigma(\bar{A} + \bar{B}\bar{K}) &= \left\{ \underbrace{-60, \dots, -60}_{8 \text{ autovalores}}, \underbrace{-70, \dots, -70}_{8 \text{ autovalores}}, \underbrace{-80, \dots, -80}_{8 \text{ autovalores}} \right\}
 \end{aligned}$$

As Figuras 4.21, 4.22, 4.23 apresentam os valores de erro para as três atribuições de autovalores. Nestas simulações os sinais de entrada são gerados em 0.5 segundos.

Figura 4.21: Sinal de erro para a primeira atribuição de autovalores.

Figura 4.22: Sinal de erro para a segunda atribuição de autovalores.

Figura 4.23: Sinal de erro para a terceira atribuição de autovalores.

4.7 ANÁLISE DOS RESULTADOS

Por meio das simulações, analisou-se o comportamento das grandezas físicas envolvidas na dinâmica do sistema controlado para diferentes sinais de entradas. Nestas simulações, não foi atribuída uma limitação de carga no sinal de controle, justamente para analisar o comportamento desta grandeza para as diferentes entradas. Para a entrada do tipo degrau, observa-se que, apesar do sistema de controle estar estável, o torque e a velocidade exigida pelo sistema de controle são relativamente altos, exigindo, na pior das hipóteses, 470 N.mm de torque e 11.5 rad/s de velocidade angular em uma das articulações da mão. Como neste trabalho são utilizados motores de 20 Watts com torque operacional nominal de 500 N.mm a 40 rad/s, não surgem restrições aparentes na malha de controle projetada. Na figura 4.5, observa-se que o tempo de estabilização do sistema de controle está abaixo de 0,30 segundo. Para a entrada do tipo pulsada, o comportamento assim como do degrau é estável e rápido, com tempo de estabilização por volta dos 0,2 segundos. Comparando os gráficos das Figuras 4.15 e 4.16, observa-se um erro de regime de aproximadamente de 0,1 segundo.

É importante salientar também que não foram consideradas nestes testes, as limitações de tensão na entrada dos motores de ± 10 Volts, sendo que, para os sinais de testes do tipo senoidal aplicados ao sistema controlado, este se mostrou capaz de segui-los sem se aproximar dos limites de saturação. O mesmo não ocorreu com sinais de testes mais agressivos, como sinais de entrada degrau e sinais de entrada pulsada onde os limites de saturação foram superados na maioria dos casos. Isso denota que o gerador de trajetória (Caurin et al. 2004) deverá ser suave o suficiente para que não ocorra a saturação do sinal de controle, ou então, o sistema de controle deve ser projetado novamente utilizando autovalores menores.

Analisando os gráficos de deslocamento angular e erro, constata-se que o sistema controlado apresenta um comportamento estável com amortecimento crítico. Ou seja, sem sobre-sinal, conforme projetado no sistema de controle (atribuição de autovalores reais com valores negativos).

Observando as Figuras 4.21, 4.22 e 4.23, verifica-se que o tempo de estabilização do sistema varia em função dos autovalores atribuídos, sendo que, quanto maior for os autovalores atribuídos, menor o tempo de estabilização. Na primeira atribuição, o tempo de convergência é por volta de 0,35 segundo. Na segunda atribuição, este tempo está na faixa de 0,19 segundo e, na terceira e mais alta atribuição de autovalores, o tempo que o sistema converge é de aproximadamente de 0,13 segundo.

É importante observar que, pela atribuição do conjunto de autovalores, o sistema deveria, na primeira atribuição, ter convergido para resposta desejada em pelo menos 0,315 segundo, devido ao menor autovalor atribuído, que, no caso, foi de 20 rad/s, ou 3,18 Hz. Na segunda atribuição, o sistema deveria ter respondido a 0,16 segundo, devido ao menor valor atribuído (40 rad/s). Já na ultima atribuição, o sistema deveria responder em 0,105 segundo. Isso denota uma diferença de 10 a 20 % entre o tempo de resposta imposto e o efetivamente obtido, sendo as diferenças tão maiores quanto maiores forem os autovalores atribuídos.

4.8 CONCLUSÕES SOBRE AS SIMULAÇÕES OFF-LINE DO SISTEMA CONTROLADO

O ambiente de modelagem Adams[®] mostrou-se prático e flexível na obtenção e integração das equações dinâmicas da BRAHMA. O controle foi desenvolvido no ambiente de desenvolvimento Matlab/Simulink[®]. A simulação foi realizada de maneira simples e direta.

O controle seguidor projetado nesse trabalho mostrou-se eficiente para controlar a posição das articulações dos dedos da BRAHMA para uma gama de autovalores em malha fechada escolhidos, apresentando boa acuracidade para diversos tipos de sinais de entrada, mesmo com a atribuição de grandes deslocamentos angulares.

A diferença entre os tempos de resposta impostos ao sistema por intermédio da atribuição de autovalores e as respostas efetivamente obtidas deve-se principalmente às limitações dos *solvers* utilizados, além das limitações de processamento de PC para simular uma malha de controle de um sistema com 36 graus de liberdade. Sendo estas deficiências maiores à medida que se exige um comportamento mais ágil do sistema. Esta limitação será tratada com maior profundidade no próximo capítulo.

Pelas respostas obtidas no controlador seguidor desenvolvido, conclui-se que o mesmo poderá ser utilizado no projeto BRAHMA para seguir qualquer trajetória gerada nos algoritmos de geração de trajetória.

5 *HARDWARE-IN-THE-LOOP*

De posse do modelo dinâmico do sistema BRAHMA e desenvolvido o controle seguidor do mesmo, a integração entre o sistema de controle digital e o sistema simulado é feita pela utilização da abordagem HIL. Nesta fase é particularmente importante simular o comportamento de todo o sistema, sob restrições temporais, para que seja possível conectar o *hardware* dedicado de controle no *loop* de controle (Figura 5.1). Isso permite que este *hardware* seja testado em uma faixa extensa de condições. Além disso, partes físicas da planta devem ser incluídas neste contexto, para que se possa estudar e aplicar o acoplamento entre partes físicas e simuladas trabalhando simultaneamente (Figura 5.2).

Figura 5.1: O *hardware* de controle posicionado no mesmo *loop* de uma simulação em tempo real do sistema.

Para a execução de experimentos individuais em HIL é necessário que todos os passos descritos nos capítulos anteriores sejam cumpridos. Entretanto, isso não significa que simulações em HIL possam ser somente conduzidas no final do ciclo de desenvolvimento do produto. Para projetos finalizados, dos quais se deseja testar novos subsistemas - como, por exemplo, o sistema de frenagem de um veículo, ou um novo servomotor em um sistema de transporte – é possível completar todas as etapas descritas na metodologia proposta neste trabalho para este subsistema e aplicar as simulações de HIL, simplesmente substituindo o modelo antigo do subsistema por um novo, mantendo todos os outros modelos que compõem o sistema completo intactos.

Neste trabalho, a dinâmica acoplada entre um atuador físico e um modelo simulado em tempo real de um dos dedos da BRAHMA é testada e analisada como exemplo de uso da abordagem de HIL híbrido (Figura 5.2). O objetivo é criar uma interação entre o atuador e o modelo simulado em tempo real a fim de obter um melhor entendimento dos parâmetros físicos envolvidos neste acoplamento para, em uma segunda etapa, poder otimizar e/ou identificar (Albuquerque et al., 2007) estes parâmetros provenientes deste estudo.

Figura 5.2: O *hardware* de controle e um atuador físico no mesmo *loop* de uma simulação em tempo real de um dedo da BRAHMA

Nesta seção, é apresentada inicialmente uma discussão sobre a técnica de HIL, apontando os conceitos principais, as formas de aplicação, as principais ferramentas e abordagens existentes. Além disso, o conceito sobre tempo real é mais aprofundado em um sub-capítulo específico (item 5.2). Em seguida, são apresentadas tanto as ferramentas computacionais como as da bancada experimental utilizadas para a aplicação das simulações em HIL. Na seção de metodologia de implementação de HIL, todo o processo de implantação, juntamente com as ferramentas computacionais adotadas, são descritas de forma sequencial, utilizando sempre como exemplo o modelo da BRAHMA para contextualizar o processo de desenvolvimento e testes. Na etapa final, são apresentados e analisados alguns exemplos de aplicação da técnica de HIL, tanto na forma clássica (Figura 5.1) como na híbrida (Figura 5.2).

5.1 INTRODUÇÃO

Uma simulação HIL refere-se a uma simulação que inclui ou é capaz de incluir qualquer parte de *hardware* do sistema simulado em tempo real. É considerado um método de teste efetivo para sistemas em fase de desenvolvimento.

O processo de projetar um sistema mecatrônico utilizando a abordagem HIL apresenta desafios adicionais, se comparado a projetos que utilizem somente simulações *off-line* para implementações futuras. Um exemplo disso é a exigência do controlador ser executado em um determinado tipo de computador de tempo-real, que possa garantir a programação I/O (entrada/saída), resolvendo as derivadas em passos fixos e executando todo algoritmo de controle dentro de uma faixa da frequência condizente com a frequência de resposta projetada para o sistema. No caso deste trabalho este valor está por volta de 10 Hz devido aos autovalores atribuídos no projeto do sistema de controle (vide sub-capítulo 4.6), ou seja, tempo de resposta de 100 milisegundos. Considerações adicionais devem ser endereçadas anteriormente ao projeto, como por exemplo, o *hardware* com a capacidade de processamento requerido e interface de sinal A/D (analógico/ digital), a exigência de *software* e necessidade de programação.

O projeto de um sistema HIL típico consiste dos seguintes componentes de *hardware*:

1. **Computador do *software* de desenvolvimento (*Host*):** Este computador hospeda o editor do *software*, o depurador, e ocasionalmente um simulador (ex. Microsoft Visual Studio, Watcom C/C++, ou Matlab e Simulink da Mathwork).
2. **Computador de execução dedicado (*Target*):** Este computador executa o código gerado pelo computador de desenvolvimento e está conectado diretamente com a planta. A complexidade dos sistemas de interesse define o porte deste equipamento,

que pode ser um computador multiprocessado, ou então um simples microcontrolador isolado.

3. **Circuito de condicionamento de sinais:** Este subsistema filtra, amplifica, protege, atenua ou formata sinais enviados entre a planta – podendo ser simulado em tempo real, ou mesmo, com partes físicas - e o computador de execução do *software*.

Em um projeto de sistemas de controle utilizando a abordagem HIL, o computador de desenvolvimento executa editores de alto nível, aplicativos de comunicação, compiladores ou até mesmo gerador de código automático (como o Real-Time-Workshop[®] da Mathwork). Em casos em que um mesmo equipamento assume as funções de *host* e *target*, exigências consideráveis são impostas ao processador devido ao gerenciamento do I/O em tempo real e interface com usuário.

A planta deve ser interfaceada com o *target*, frequentemente exigindo a conversão de sinais analógicos para digitais. A maior disponibilidade de *softwares* de suporte a placas de I/O (na forma de bibliotecas e de *device drivers*) gera uma redução no tempo de projeto e permite ao programador focar sua atenção no desenvolvimento do sistema de controle.

Uma vez que uma interface de *hardware* satisfatória é selecionada, um condicionador de sinais próprio é requerido. A filtragem analógica é feita preferencialmente em *hardware* para aliviar a carga computacional do processador digital de sinais ou unidade de processamento digital. Entretanto, condicionadores de sinais são menos flexíveis do que se feitos em *software*, devido à dependência de componentes físicos.

O desenvolvimento do *software* para executar o sistema de controle é frequentemente complexo e conseqüentemente demanda um grande tempo de projeto. A seleção do sistema operacional e ferramentas de desenvolvimento deve ser feita cuidadosamente, devendo o projetista estar ciente das ferramentas existentes que manipulam automaticamente

complicações associadas ao desenvolvimento do código de tempo real com taxa variável de amostragem.

O sistema operacional no qual o computador de execução do *software* pode trabalhar é dividido em dois tipos: *Real-Time* e *Event-Driven*. Sistemas operacionais *real-time*, como VxWorks® e QNX®, executam o código em períodos pré-determinados, onde a capacidade computacional para executar uma seção particular do código é organizada pelo sistema operacional tão logo o código seja completado, antes que a próxima seção seja programada para ser executada. Espera-se do sistema operacional *Event-Driven* (como Microsoft Windows 95/98/NT/2000/XP, Linux e BSD) que ele responda somente a entradas externas do usuário. Entretanto, é possível conseguir um comportamento “pseudo” tempo real, através do uso inteligente do tempo de *software* e interrupções para forçar o sistema a executar o código em tarefas com períodos fixos. Por outro lado, os resultados de tal procedimento são inconstantes e não podem ser garantidos, pelo fato que estes sistemas operacionais são tipicamente gráficos e multitarefas em sua natureza, implicando que várias aplicações ou tarefas gráficas podem demandar tempo de processamento durante pontos críticos na operação da planta.

As ferramentas de desenvolvimento estão amplamente disponíveis nos sistemas operacionais *Event-Driven*, sendo tentador selecionar estes sistemas operacionais para o computador de execução do *software*, pelo fato de trabalhar em um ambiente conhecido e utilizando ferramentas familiares. Porém, somente é recomendado para *hardware* que não necessite ser executado com altas taxas de amostragem. Se a escolha for escrever o código por completo, várias considerações devem ser levadas em conta, incluindo o tipo de sistema operacional (*Real-Time* x *Event-Driven*), linguagem de programação e disponibilidade de suporte em *hardware* para a plataforma escolhida.

Para desenvolver simulações HIL utilizando sistemas operacionais *Real-Time* é necessária uma familiarização com as linhas de código do programa para execução em *hardware*, requerendo entradas e saídas sincronizadas e, ao mesmo tempo, resolver as equações diferenciais em tempo real. Entender como um *solver* de equações diferenciais opera e implementar um *solver* para execução em tempo real são coisas distintas. Sendo assim, projetistas podem facilmente se perder nas linhas de código e negligenciar conceitos importantes no projeto de controle e nos sistemas operacionais *Real-Time*. Muitos pacotes de *softwares* têm buscado automatizar a geração do código de controle para a execução no *hardware*. Porém, estes pacotes ainda têm algumas limitações. Por exemplo, o Matlab caracterizou o *Real-Time Workshop* durante vários anos. Este *software* traz blocos de controle gráfico no Matlab/Simulink® e compila para execução dentro de vários ambientes, como o modo DOS protegido ou VxWorks®. Infelizmente, *Real-Time Workshop* não incluem muitas variedades de *drivers* de *hardwares* específicos de controle, exigindo do usuário o desenvolvimento dos seus próprios *drivers*. Um outro pacote também disponível para sistemas *Real-Time* da Mathworks que incluem numerosas bibliotecas de apoio ao *hardware* é o “xPC”. Este pacote permite aos usuários compilar modelos gráficos do Simulink® que inclui blocos para interface com I/O de *hardwares* específicos, e estes modelos são carregados para executar no *hardware* de destino executando o sistema operacional *Real-Time* proprietário, tornando o desenvolvimento de controladores mais rápidos e podendo ser usado para progredir a transição do controlador simulado para a implementação.

Uma interface gráfica intuitiva é interessante para mostrar os fluxos de sinais de forma clara e suportar simulações e ferramentas de análises. O pacote deve estar diretamente unido com o sistema operacional, de forma que os recursos administrativos diretos possam ser alcançados, como a criação de códigos embarcados que rodam no núcleo do sistema *Real-*

Time do qual se administra o uso do processador, a distribuição da memória e os sistemas I/O em tempo real.

Um conceito muito importante para simulações HIL, devido às exigências de execução sob rígidas restrições de tempo, é o de tempo real, que é introduzido na sequência desta seção.

5.2 INTRODUÇÃO AO CONCEITO DE TEMPO REAL

Aplicações com requisitos de tempo real tornam-se cada vez mais comuns na medida em que o uso de sistemas computacionais se prolifera no mundo. Nestas aplicações, existe uma grande variação em relação à complexidade e às necessidades de garantia no atendimento de restrições temporais. Como sistemas simples, podem-se apontar os eletrodomésticos e *videogames*, assim como lavadoras de roupa, de louça e aparelhos de DVD. Para sistemas mais complexos, aponta-se para os sistemas de controle de plantas industriais (químicas e nucleares), controle de tráfego aéreo, sistemas militares de defesa, robôs, etc. Todas essas aplicações, que apresentam de alguma forma características de estarem sujeitas a restrições temporais, são agrupadas no que se é, usualmente, chamado de sistemas de tempo real.

Um sistema de tempo real é um sistema computacional que deve reagir a estímulos oriundos do seu ambiente em prazos específicos (Buttazzo, 1997). O atendimento desses prazos resulta em requisitos de natureza temporal sobre o comportamento desses sistemas. Em consequência, em cada reação, o sistema de tempo real deve entregar um resultado correto dentro de um prazo específico. O comportamento correto de um sistema de tempo real, portanto, não depende só da integridade dos resultados dos dados obtidos (correção lógica), mas também dos valores de tempo em que são produzidos (correção temporal). Uma reação que ocorrer além do prazo especificado pode ser sem utilidade ou até mesmo representar uma ameaça (Buttazzo, 1997, Stankovic, 1990).

As aplicações de tempo real se comportam como sistemas reativos com restrições temporais. A reação desses sistemas aos eventos provenientes do ambiente externo ocorre em tempos compatíveis com as exigências do ambiente e mensuráveis na mesma escala de tempo. Sendo assim, a concepção do sistema de tempo real é diretamente relacionada com o ambiente no qual está relacionado e com o comportamento temporal do mesmo.

Na classe de sistemas de tempo real onde se encontram os sistemas embarcados, os sistemas de supervisão e controle, existe uma distinção entre o sistema a controlar, o sistema computacional de controle e o usuário. O sistema a controlar e o operador são considerados como o ambiente do sistema computacional (Figura 5.3). A interação entre os mesmos ocorre através de interfaces de instrumentação (composta por sensores e atuadores) e da interface com o usuário.

Figura 5.3: Sistemas de Tempo Real

Muitos são os que acreditam que o problema de tempo real se resolve pelo aumento da velocidade computacional. A rapidez de cálculo visa melhorar o desempenho de um sistema computacional, minimizando o tempo de resposta médio de um conjunto de tarefas, enquanto o objetivo de um cálculo em tempo real é o atendimento dos requisitos temporais de cada uma das atividades de processamento caracterizadas nesses sistemas (Stankovic, 1988). Ter um

tempo de resposta curto não significa ter a garantia de que os requisitos temporais de cada processamento no sistema serão atendidos.

Um sistema de tempo real é previsível no domínio lógico e no domínio temporal quando, independente de variações no nível de *hardware*, das falhas e da carga computacional, o comportamento do sistema pode ser previsto antes de sua execução. Porém, para se assumir a previsibilidade de um sistema de tempo real, precisa-se conhecer antecipadamente o comportamento deste sistema. Para tanto, é necessário levar em consideração, além da carga computacional e atribuição de hipóteses de falhas (Stankovic, 1988), um conjunto de fatores relacionados à arquitetura de *hardware*, aos sistemas operacionais e às linguagens de programação.

Diante deste contexto, é importante salientar que, em cada etapa do ciclo de desenvolvimento de um sistema de tempo real, torna-se necessário o uso de ferramentas e metodologias apropriadas que permitam verificar o comportamento do sistema e a sua implementação como previsíveis.

Comparado aos problemas clássicos de simulação *off-line*, simulações em tempo real demandam fundamentalmente diferentes propriedades de rotinas de integração para resolver as equações diferenciais ordinárias.

Integradores *off-line* buscam minimizar o tempo global de integração com uma dada precisão, utilizando tamanhos de passos de integração sofisticados, controle de ordem de mecanismo e métodos de mais alta ordem.

Em simulações em tempo real, o computador geralmente se comunica com outros componentes de *hardware* durante a simulação. Esta comunicação ocorre utilizando intervalos fixos e curtos de amostragem. A simulação deve, portanto, executar o passo de integração e prover seu resultado antes da conclusão deste intervalo. Excedendo este prazo, um erro poderá acontecer. Assim, o objetivo da simulação de tempo real não é reduzir o custo

computacional médio, mas, garantir que o tempo de cálculo para um passo nunca exceda o prazo estabelecido. Isso conduz a uma escolha apropriada de métodos de integração. Para se minimizar o custo computacional de um passo de integração, geralmente se utiliza métodos de integração simples, como é o caso do método explícito de Euler utilizado neste trabalho.

5.3 DESCRIÇÃO DOS SOFTWARES E HARDWARES UTILIZADOS

5.3.1 Sistema Operacional Tempo Real – VxWorks

Aplicações de tempo real são mais facilmente construídas, se puderem aproveitar os serviços de um sistema operacional. Desta forma, o programador da aplicação não precisa se preocupar com a gerência dos recursos básicos (processador, memória física, controlador de disco). Ele utiliza-se de abstrações de mais alto nível criadas por um sistema operacional (tarefas, segmentos, arquivos).

Um sistema operacional de tempo real (RTOS) procura tornar a utilização do computador mais eficiente e mais conveniente. Eficiente significa mais trabalho obtido a partir de um mesmo *hardware*. Isto é alcançado através da distribuição dos recursos do *hardware* entre as tarefas. Uma utilização mais conveniente diminui o tempo necessário para a construção dos programas.

Neste trabalho, o sistema RTOS é o VxWorks® que juntamente com as ferramentas de *software* (editor, compilador e depurador) forma o ambiente integrado de programação denominado Tornado.

O VxWorks® – fornecido pela WindRiver – é um RTOS assíncrono multitarefa de 32 bits (neste trabalho, utiliza-se o VxWorks 5.5). Este sistema operacional utiliza suporte POSIX que é um padrão para sistemas operacionais, baseados no Unix, criado pela IEEE (*Institute of Electrical and Electronic Engineers*). Esse sistema operacional inclui também um

conjunto de ferramentas e utilitários tanto para o *host* (computador de desenvolvimento) como para o *target* (computador alvo) e opções de comunicação (*Ethernet* e *serial*).

5.3.2 Tornado

Utiliza-se como IDE (*Integrated Development Environment*) o Tornado[®] para o desenvolvimento de sistemas embarcados em diferentes plataformas de *hardware*. O Tornado integra os elementos necessários para o desenvolvimento que incluem:

- Um ambiente para a escrita de programas e módulos de *software*.
- Ferramentas para a compilação e geração de uma imagem executável para ser embarcada no dispositivo de *hardware* utilizado.
- Um simulador de *hardware* para o teste de algoritmos e rotinas independentes de *hardware*.
- Um sistema de conectividade com o dispositivo de *hardware* (*target*).
- Um depurador integrado ao editor de código para facilitar o segmento dos fluxos de execução. Ele pode ser usado com um *target* simulado ou real.
- Ferramentas para análise de desempenho do *software* (*Scope Tools*).

O *target*, no caso a plataforma MVME162-522A, precisa ser carregado com um sistema operacional para realizar as funções e controle dos dispositivos ligados a ele (o sistema utilizado é o VxWorks 5.5 da WindRiver). O primeiro passo é gerar uma imagem executável para poder ser “embarcada” no *target*, podendo, em seguida, criar os módulos de *software* que serão responsáveis pela execução das tarefas de controle necessárias para o projeto. O segundo passo é a criação dos módulos para acessar as funcionalidades dos dispositivos e periféricos do *target*. A plataforma utilizada no trabalho possui 04 baias para a inserção de 4 *Industry Packs* com 50 pinos cada um.

5.3.3 Real-Time-Workshop (RTW)

O Real-Time-Workshop[®] (RTW) é uma extensão da capacidade encontrada no Simulink[®] e Matlab[®] de prover a prototipagem rápida de aplicações de *software* em tempo real em uma grande variedade de sistemas. O RTW junto com outras ferramentas e componentes da MathWorks, provê:

- Geração automática de código para uma grande variedade de plataformas alvo;
- Um caminho rápido e direto do projeto de sistemas para a implementação;
- Integração com Matlab[®] e Simulink[®];
- Uma interface gráfica simples;
- Uma arquitetura aberta e extensível.

O RTW constrói aplicações dos diagramas do Simulink[®] para prototipagem, testes e desenvolvimento de sistemas de tempo real em uma variedade de plataformas computacionais. O RTW pode gerar diretamente o código que hospeda os compiladores, dispositivos de entrada e saída, modelos de memórias, modos de comunicação e outras características que a aplicação pode exigir.

O RTW está preparado para trabalhar com o Tornado por meio de um conjunto de ferramentas para criação de aplicações de tempo real sob o sistema operacional VxWorks[®]. Dentro do RTW existe um conjunto de bibliotecas de suporte ao VxWorks[®] que vão desde atribuição de tarefas, controle de interrupção, definição de taxas de transmissão, sincronização de tarefas até a um conjunto de bibliotecas de dispositivos I/O (*device drivers*) que possibilitam a comunicação com a instrumentação de controle (atuadores, sensores e condicionadores de sinais).

Neste trabalho, onde as tarefas de tempo real são baseadas no sistema operacional VxWorks, o *hardware* utilizado consiste de um PC (*Host*) executando o Simulink[®] e o *Real-Time-Workshop* conectado via Ethernet a um *Target* sob gerenciamento do sistema

operacional VxWorks. Além disso, no chassi do *Target* (plataforma VME) existe um conjunto de *Industrial Packs* com conversores A/D e D/A que se comunicam com a bancada experimental. O código de tempo real é compilado no *Host* utilizando o compilador fornecido pelo pacote do VxWorks. É gerado então um arquivo objeto que é carregado no *Target* via conexão *Ethernet* (Figura 5.4).

Figura 5.4: Esquema da conexão de *hardware*

5.3.4 Plataforma MVME162

Para obter um comportamento em tempo real esperado para aplicar a técnica de HIL é necessário o uso, além de ferramentas de desenvolvimentos, uma arquitetura robusta de *hardware*. A plataforma computacional utilizada é baseada em uma placa microcontrolada MVME162FX montada em um padrão de barramento VME (*Versa Module Eurobus*). Esta estrutura de *hardware* é complementada por quatro *Industrial Packs* usados como interfaces de comunicação (digital e analógica) requerida pelos sensores e atuadores do projeto (Figura 5.5).

Figure 5.5: Estrutura geral da plataforma de testes em HIL

Esta plataforma é ideal para monitoramento e aplicações de controladores distribuídos, permitindo que um OEM (*Office of Emergency Management*) minimize gastos de engenharia enquanto integra de forma incremental aplicações de *hardware* e *software*. Para tanto, a plataforma MVME possui uma variedade de MPU (*Multiple Processor Unit*), memória e opções de interface como ponto flutuante, *Ethernet*, SCSI e VME. Isso resulta em um ambiente flexível, que pode se adaptar a um variado número de requisitos de aplicação.

Figure 5.6: Plataforma MVME162 utilizada no trabalho

O modelo da plataforma MVME162 (Figura 5.6) usado no projeto possui como principais características:

- Microprocessador de 32 MHz MC68040, 8MB de *Dynamic Random Access Memory* (DRAM), interface *Ethernet*, 512KB *Static Random Access Memory* (SRAM).
- 1MB de memória *Flash*: um pente Intel 28F008SA (para placas mais antigas) ou quatro pentes Intel 28F020s (para placas mais recentes).
- Quatro *timers* programáveis de 32 bits e um *Watchdog Timer* programável (MCchip).
- *Input/Output*
 - Duas portas seriais (uma EIA-232-D DCE; uma EIA-232-D ou EIA-530 DCE/DTE).
 - Controlador de porta serial (Zilog Z85230).
 - Interface de barramento *Small Computer Systems Interface* (SCSI) com acesso *Direct Memory Access* (DMA) de 32 bits em rajada (burst) (controlador NCR 53C710).
 - Interface LAN *Ethernet* com barramento local de 32-bits.

- DMA (controlador Intel 82596CA)
- Módulos *Industry Pack* (IP): até quatro interfaces MVIP Industry Pack podem ser instaladas no MVME162. A interface entre os IPs e o MVME162 é denominada *Industry Pack Interface Controller* (IPIC) ASIC. O acesso aos IPs é realizado através de quatro conectores 3M localizados atrás do painel frontal do MVME162.
- *Interface VMEbus* (VMEchip2)
 - Funções de controlador de sistema VMEbus
 - Interface VMEbus para barramento local (A24/A32, D8/D16/D32, D8/D16/D32/D64BLT, onde BLT = *Block Transfer*).
 - Barramento local para a interface VMEbus (A16/A24/A32, D8/D16/D32).
 - Interruptor VMEbus.
 - Manipulador de interrupção VMEbus.
 - DMA para memória local rápida – transferências VMEbus (A16/A24/A32, D16/D32[D16/D32/D64BLT])

5.3.5 *Industry Pack* (IP)

Cada *Industry Pack* (IP) possui um espaço de memória para uso individual, um espaço para acesso aos registradores de identificação do IP e um espaço para acesso aos registradores de operação, chamado também espaço de entrada/saída (I/O).

O espaço de memória de cada IP é definido no momento da inicialização do sistema, através da configuração do *IP Carrier*. A forma de programação e acesso aos IPs depende do número e função dos registradores definidos para cada um deles. A seguir, serão detalhados os registradores dos IPs usados no projeto.

IP220

O IP220 é um IP utilizado como dispositivo de saída analógica. Ele possui 16 canais analógicos através dos quais é possível fornecer uma tensão entre -10 e 10 Volts com uma resolução de 12 bits o que proporciona uma resolução de 4096 valores.

IP320

O *Industry Pack Acromag* IP320 é semelhante ao IP220 para entradas analógicas, ele possui 16 entradas analógicas de -10 a 10 Volts com uma resolução de 12 bits (4096 passos).

MVIP 341

O MVIP 341 é um IP que serve tanto para operações de entrada como de saída digital, através de 40 canais disponíveis. Ele está baseado em 2 microcontroladores IP-DIGITAL-482 de 20 canais cada. Os registradores de configuração definem a direção de modo de acesso aos canais digitais enquanto os registradores de dados armazenam ou transferem os dados entre os canais digitais e o barramento VME.

IP Quadrature

O IP quadrature é um *IndustryPackage* para leituras de *encoders* incrementais, possui 4 canais, cada um com contador de 24 bits. Cada canal possui três entradas (X,Y,Z) canal A, canal B e índice I.

5.4 METODOLOGIA DE IMPLEMENTAÇÃO DE HIL

Após obter as exigências necessárias para executar simulações em tempo real, uma simulação clássica em HIL (Figura 5.1) é inicialmente implementada. Nesta configuração, o comportamento de todo o sistema, sendo controlado por um controle multivariável executado em um *hardware* dedicado, é analisado em tempo real para um conjunto de sinais de teste.

Para tanto, o sistema deve ser discretizado em um determinado tempo de amostragem para, logo em seguida, ser transferido para o *target*, por meio das ferramentas do RTW. O passo seguinte é iniciar o processo de simulação do controle seguidor em tempo real, gerenciado por ferramentas do sistema operacional VxWorks®.

Após a implantação e análises da abordagem clássica de HIL, uma abordagem híbrida é adotada. Nesta fase, a configuração adotada foi a inserção de um dos motores físicos na mesma malha de controle do resto do sistema simulado em tempo real, ou seja, de forma sincronizada.

No decorrer da presente seção, todo o processo de implementação, juntamente com as ferramentas computacionais adotadas, é descrito de forma seqüencial, utilizando sempre como exemplo o modelo da BRAHMA para contextualizar o processo de desenvolvimento e testes.

5.4.1 Implementação

Para se iniciar os testes com simulações HIL, o modelo dos atuadores utilizados no projeto BRAHMA é validado utilizando as mesmas ferramentas de implementação de simulações e testes em HIL. A proposta aqui é comparar, de forma instantânea (*on-line*) e simultânea, dois sistemas - um sistema físico e um sistema simulado do atuador – sob influência de um mesmo controlador.

No desenvolvimento do controlador (Capítulo 5) foi discutido que a frequência de resposta do sistema em malha fechada não passaria de 10 Hz, ou seja, com um período de 100 ms. A fim de garantir que o intervalo entre as leituras seja suficientemente pequeno, de modo a não comprometer o sistema controlado - seguindo o Teorema de Shannon sobre a frequência mínima de amostragem necessária para reconstruir o sinal original a partir de um sinal amostrado - foi definido um período de amostragem vinte vezes menor, ou seja, de 5 ms.

Dessa forma, tanto o sistema de controle como o modelo de simulação, devem ser discretizados com um taxa de amostragem de 5 ms. Estes momentos especificam o instante em que é feita a medida física ou o instante em que é lida a memória de um computador digital. O intervalo de tempo entre dois instantes discretos deve ser suficientemente pequeno, de tal forma que os dados para os tempos entre dois instantes discretos podem ser aproximados por interpolação simples.

Inicialmente, propõe-se simular o mesmo controle desenvolvido no Capítulo 4 aplicado somente a um modelo discretizado do motor Maxon (Figura 5.7), utilizando para tanto, as mesmas plataformas de desenvolvimento descritas na Seção 5.3.

Figura 5.7: Controle seguidor aplicado ao modelo discreto do motor Maxon (modelo 14434)

São três as ferramentas utilizadas: o Simulink® para o desenvolvimento do controlador aplicado ao modelo do motor; o Real Time Workshop (RTW); e o Tornado/VxWorks utilizado para embarcar e executar o código gerado pelo RTW no *Target*.

Real-Time-Worshop® constrói aplicações dos diagramas do Simulink® para prototipagem, testes e desenvolvimento de sistemas de tempo real em uma variedade de plataformas computacionais. O RTW gera diretamente o código por meio de compiladores pré-instalados, além dos dispositivos de entrada e saída, modelos de memórias, modos de comunicação e outras características que a aplicação pode exigir.

O ambiente do Tornado® por sua vez, está preparado para trabalhar com o RTW. Quando instalados os serviços do sistema VxWorks®, o processo de desenvolvimento está pronto para ser realizado.

O próximo passo, então, é configurar o Simulink® e o RTW para gerar o código fonte (Figura 5.8), compilar e vincular as determinadas bibliotecas para, então, ser carregado ao *Target*. Desta forma o sistema estará pronto e poderá:

- Gerar código C por meio de diagramas de blocos do Simulink® para então ser automaticamente compilado (Figura 5.8), vinculado às bibliotecas necessárias (incluindo, quando for o caso, os *drivers* particulares de entrada e saída) e carregado dentro do computador alvo.
- Ajustar e monitorar, em tempo real, parâmetros do controle durante a simulação. Isto pode ser feito carregando o *Target* (computador alvo) com os parâmetros e processos desejados diretamente do Simulink® executando o modo externo (*External Mode*) no *Host* (computador de desenvolvimento).

Figura 5.8: Configuração do RTW para trabalhar com o Tornado®.

O Simulink®, executado no modo externo pode se comunicar com o processador utilizando conexão via TCP/IP. Essa comunicação é utilizada para ajustar parâmetros (ganhos, constante de tempo, sinal de entrada, etc.) rapidamente, sem a necessidade de o modelo ser recompilado.

Este ambiente provê condições necessárias para a criação de programas e testes, carregando aplicações dentro do *Target*, sendo este gerenciado pelo sistema operacional VxWorks®.

Na Figura 5.9 são apresentadas as respostas da simulação em tempo real de um sistema de controle seguidor aplicado a um modelo de motor executado na plataforma VME e gerenciado pelo Sistema Operacional VxWorks® para uma meia onda senoidal com frequência de 5 rad/s. O ambiente do Simulink® está sendo utilizado como um sistema supervisor para atualizar parâmetros e visualizar os resultados.

Figura 5.9: Resultados experimentais do controle de posição aplicado a um modelo de motor em HIL. Em azul tem-se a trajetória desejada e em verde a resposta medida.

Na Figura 5.10 é apresentado o monitoramento instantâneo de diversas respostas do sistema controlado em tempo real para uma dada entrada pulsada.

Figura 5.10: Na ordem têm-se as seguintes respostas no tempo: 1) posição angular (em verde) e sinal de entrada (em azul); 2) medição do erro; 3) tensão de entrada do motor; 4) torque no eixo de entrada do motor.

Neste ponto do trabalho, o procedimento para gerar e embarcar o código fonte e, monitorar de forma instantânea as respostas do sistema simulado está estabelecido para a realização de testes de simulação em HIL.

Agora, para poder implementar a abordagem híbrida de HIL, e assim inserir novos *hardwares* no mesmo *loop* do modelo simulado em tempo real o interfaceamento de instrumentação (comunicação com sensores e atuadores), placas de entrada e saída (I/O) devem ser testadas (Figure 5.5).

Nesta fase do trabalho foi necessário preparar os drivers de modo que houvesse uma comunicação entre o programa de tempo real e os dispositivos de I/O. Isso é obtido pela criação de *drivers* particulares dentro do ambiente do Simulink® através de S-Function. O S-Function é uma linguagem de computador descrita na forma de diagrama de blocos do Simulink. O S-Function permite que se crie um bloco específico para uma determinada tarefa diretamente no Simulink, esse bloco pode ser escrito em diferentes linguagens, tais como: Matlab, C++, C, Fortran, etc.

As tarefas suportadas por cada *driver* criado são:

- Obtenção dos valores de parâmetros do Simulink®;
- Inicialização da placa. Isso significa a disponibilização das portas, registros, modos de ajustes de operação, ganhos, etc;
- Leitura ou escrita no *hardware* (dependendo do bloco ser referente a entrada ou saída de um sinal);
- Disponibiliza a placa quando a tarefa estiver terminada.

Figura 5.11: *Device Drivers* desenvolvidos para a comunicação com os *Industrial Packs*

Caso os *drivers* já existam, não há a necessidade de serem refeitos, pois podem ser importados das bibliotecas onde se encontram para qualquer diagrama de bloco do Simulink®.

5.5 RESULTADOS DAS APLICAÇÕES EM HIL

O ambiente experimental utilizado neste trabalho encontra-se na Figura 5.12. Desta forma, todos os processos (simulação em tempo real, controle de motores, leitura de sensores, dentre outros) são executados na placa MOTOROLA MVME 162, através do ambiente operacional de tempo real VxWorks 5.5 como descrito na Seção 5.3.

Figura 5.12: Ambiente experimental. Na seqüência: 1) O protótipo da Brahma; 2) o *Target*; 3) o *driver* de potência e motor da Maxon; 4) o *Host*.

Nas Figuras de 5.13 até 5.19 é apresentada - por meio do monitoramento instantâneo de diversas respostas - uma comparação das respostas do controle aplicadas às plantas (real e simulada em tempo real) do motor para uma meia onda senoidal com frequência de 5 rad/s.

Figura 5.13: Aplicação do controle seguidor a uma planta simulada e a uma planta física para uma entrada referência de meia onda senoidal com amplitude de 15 radianos e frequência de 5 rad/s.

Figura 5.14: Na ordem tem-se as seguintes respostas instantâneas em uma janela de 4 segundos para uma entrada de meia onda senoidal com amplitude de 15 radiano e frequência de 5 rad/s no eixo de entrada do motor: 1) trajetória desejada; 2) resposta da planta simulada em tempo real; 3) resposta da planta física.

Figura 5.18: Na ordem tem-se as seguintes respostas instantâneas em uma janela de 4 segundos para uma entrada pulsada: 1) trajetória desejada; 2) resposta da planta simulada em tempo real; 3) resposta da planta física.

Figura 5.19: Na ordem tem-se as seguintes respostas instantâneas em uma janela de 4 segundos dos sinais de controle em Volts para uma entrada pulsada: 1) sinal de controle da planta simulada em tempo real; 2) sinal de controle da planta física.

A Figura 5.20 apresenta um monitoramento das execuções e interrupções das tarefas gerenciadas em *hardware* pelo sistema operacional VxWorks®. Uma das tarefas é evidenciada em um espaço determinado de tempo, onde seus estados são monitorados. Esta tarefa é relativa à comunicação com o *host*, onde novas atribuições do modelo podem ser feitas de forma instantânea, ou seja, sem a necessidade de interromper a simulação.

Figura 5.20: Instrumentação do *Target* utilizando ferramentas de monitoramento do RTOS

Nas Figuras de 5.24 até 5.27 são apresentadas - por meio do monitoramento instantâneo de diversas respostas – algumas respostas da investigação realizada sobre o efeito do acoplamento dinâmico entre os atuadores e articulações realizado em um modelo de simulação em tempo real de um dos dedos da BRAHMA. Na figura a seguir (Figura 5.21) é

ilustrado o modelo de simulação da BRAHMA considerando um acoplamento dinâmico entre os atuadores e as articulações.

Figura 5.21: Representação em diagrama de blocos do modelo da BRAHMA levando-se em conta o acoplamento dinâmico entre os atuadores e as articulações.

Um conjunto de sinais de teste foi aplicado no sistema, conforme é mostrado na Figura 5.22.

Figura 5.22: Sinais de teste aplicados no modelo de simulação em tempo real de um dos dedos da BRAHMA.

Figura 5.23: Resposta instantânea dos deslocamentos angulares de um dos dedos da BRAHMA em um período de tempo específico.

Os próximos resultados são apresentados para uma abordagem híbrida de HIL, onde uma interação flexível entre um protótipo físico (atuador) e o modelo simulado é estabelecida como mostrado na Figura 5.2 e também ilustrada na Figura 5.24.

A Figura 5.25 apresenta a resposta dos deslocamentos angulares nas articulações de um dedo para o mesmo conjunto de sinal de entrada (Figura 5.22). A resposta específica do deslocamento θ_5 é composta pela união e sincronismo entre uma de parte real e outra simulada. Esta união pode ser melhor observada na Figura 5.26, que mostra o sinal de entrada versus a resposta do deslocamento angular no eixo de saída do motor versus a resposta do deslocamento angular na articulação θ_5 .

Figura 5.24: Ilustração da abordagem híbrida de HIL adotada, onde o hardware de controle e um atuador físico estão no mesmo *loop* de simulação em tempo real de um dedo da BRAHMA.

Figura 5.25: Resposta instantânea dos deslocamentos angulares de um dos dedos da BRAHMA em uma abordagem híbrida de HIL para as mesmas condições da abordagem clássica.

Figura 5.26: Resposta instantânea do sinal de entrada versus a resposta do deslocamento angular no eixo de saída do motor versus a resposta do deslocamento angular na articulação θ_5 .

Por fim, é apresentada, na Figura 5.27, uma comparação entre as respostas da abordagem clássica de HIL - onde somente se encontra no *loop* o *hardware* de controle - e da abordagem híbrida de HIL, que além do *hardware* de controle, um dos atuadores simulados é substituído por um atuador físico para trabalhar no mesmo *loop* da simulação em tempo real de um dos dedos da BRAHMA.

Analisando a comparação entre as respostas das duas abordagens aplicadas a um modelo de um dedo da BRAHMA (Figura 5.27), verificou-se uma pequena defasagem entre as respostas dos deslocamentos angulares θ_5 , onde justamente foi inserido um atuador físico no *loop* de simulação em tempo real. As respostas dos deslocamentos angulares θ_4 também apresentaram uma defasagem - porém, em menor grau do que em θ_5 - causada pela influência

dinâmica de uma articulação na outra. Estas influências também ocorrem nas outras duas articulações (θ_3 e θ_6), porém, em menores proporções.

Figura 5.27: Comparação entre as Figuras 5.25 e 5.26. Sendo as curvas contínuas, as respostas instantâneas dos testes em HIL clássico e, as linhas tracejadas, as referentes às respostas instantâneas da abordagem híbrida de HIL.

6 CONCLUSÕES E CONSIDERAÇÕES FINAIS

Neste trabalho, foram estudados e criados os procedimentos e a estrutura necessária para a implementação dos testes e simulações relacionadas com a abordagem de *Hardware-in-the-Loop*. A abordagem adotada baseou-se na integração de ferramentas de desenvolvimento (*software*) comerciais distintos e na agregação das capacidades inerentes a cada um deles. Testes preliminares já indicam uma integração estável e confiável das ferramentas. Até o presente momento não foram encontradas dificuldades ou gargalos de comunicação entre elas.

Com a estratégia utilizada para embarcar o código, tanto do modelo da planta como do controle, dentro de uma mesma plataforma (*Target*) gerenciada por um sistema operacional de tempo real (VxWorks®), eliminou-se a necessidade de um maior gerenciamento de rotinas de chamadas de TCP/IP, uma vez que todo o sistema de HIL se encontra no *Target*. Sendo o *Host*, neste processo, simplesmente utilizado para o pós-processamento e atualização de parâmetros. Com o aumento das variáveis controladas do modelo mais completo da planta (modelo da BRAHMA), esta opção pode ser colocada em prova, uma vez que aumentará muito o custo computacional do processo.

Na abordagem de HIL híbrida, onde se inseriu um novo *hardware* no processo de simulação em tempo real, a causa da defasagem apresentada nas respostas em relação ao sistema puramente simulado foi devido à separação em diferentes tarefas, onde a execução da simulação do sistema foi considerada uma tarefa e a leitura do *encoder* e execução do controle no atuador uma outra tarefa distinta. Sendo assim, os diferentes tempos de execução das tarefas foram a causa da defasagem das respostas (Figura 5.27).

Este trabalho apresentou, além da metodologia para implantação da técnica de HIL, um grande leque de possibilidades de aplicação desta técnica como uma ferramenta de suporte no

processo de desenvolvimento de uma mão artificial robótica, o caso de estudo escolhido para contextualizar a aplicação.

Os esforços foram concentrados no desenvolvimento de um ambiente computacional e um ambiente experimental para trabalharem em conjunto e simultaneamente. No ambiente computacional foram desenvolvidos modelos do sistema BRAHMA simulados em tempo real. No ambiente experimental, tanto o *hardware* de controle como um atuador do protótipo BRAHMA, foram utilizados. Em ambos os casos foram desenvolvidos e empregados um controlador seguidor multivariável.

Adotando este tipo de abordagem, partes do sistema simulado em tempo real podem ser substituídas - à medida de suas necessidades - por partes físicas, como por exemplo: sensores, atuadores e novos *hardwares* de controle. Obtendo assim, uma ferramenta de suporte para integração tanto do *hardware* de controle como de novos *softwares* no processo de desenvolvimento do projeto BRAHMA.

Os itens destacados como possíveis benefícios na aplicação de técnicas de HIL apresentados no Capítulo Introdução e Justificativa, são analisados e discutidos individualmente.

6.1 ANÁLISE DOS BENEFÍCIOS DO USO DE HIL NO TRABALHO

6.1.1 Verificação do Comportamento do Sistema Simulado em Tempo Real

O comportamento do modelo de simulação em tempo real da BRAHMA, sob influência de um controle multivariável de posição, sendo executado em um *hardware* dedicado foi verificado. Com isto o sistema pôde ser testado para um conjunto de sinais de entrada, por meio da geração automática de código fonte e pelo monitoramento, de forma instantânea, das respostas simuladas.

6.1.2 Interação e simultaneidade entre modelos simulados em tempo real e protótipos físicos na execução das tarefas e testes

O interfaceamento de instrumentação (comunicação com sensores e atuadores), placas de entrada e saída (I/O) foi estabelecido por meio de preparação adequada de *drivers* para executar a comunicação entre o RTOS e os dispositivos de I/O. Com isto, a planta real e simulada em tempo real puderam ser comparadas e validadas por meio do monitoramento instantâneo das respostas de ambos os sistemas. Caso houvesse a necessidade de um refinamento em algum dos parâmetros da planta simulada, o mesmo poderia ser realizado de forma instantânea, ou seja, sem a necessidade de interromper o processo de simulação em tempo real.

Uma outra possibilidade que pode ser apontada é a inserção de novos *hardwares* de controle, quando se tratar de um sistema de controle distribuído, onde cada *hardware* de controle pode atuar em diferentes partes do modelo de simulação em tempo real.

6.1.3 Diminuição do risco do desenvolvimento relacionado à utilização de diferentes tipos de tecnologia.

O fato de não ficar refém a uma tecnologia é estratégico para qualquer equipe ou projeto tecnológico. Dessa forma, buscou-se fugir das soluções de “pacote” e optar pela integração de diferentes tecnologias e soluções como, por exemplo: o ADAMS[®] para as simulações off-line da dinâmica da BRAHMA, o Matlab/Simulink[®] para o desenvolvimento do controlador e monitoramento instantâneo de simulações em tempo real, o RTW para a geração automática dos códigos fonte e preparação dos *drivers*, o VxWorks[®] como sistema operacional em tempo real, o Tornado[®] como ambiente de desenvolvimento e gerenciamento de sistemas embarcados e a plataforma com arquitetura VME para a realização tanto das simulações como das ações de controle.

6.1.4 Criação de uma estrutura para o desenvolvimento de novos algoritmos (gerador de trajetórias, controladores, etc.) de maneira mais ágil e direta.

Qualquer novo algoritmo desenvolvido para o projeto BRAHMA pode ser implementado diretamente no modelo de simulação em tempo real e em um *hardware* dedicado de controle por meio da obtenção de uma estrutura de HIL bem estabelecida.

6.1.5 Diminuição dos custos do projeto de desenvolvimento

As simulações em HIL tanto clássicas como híbridas ofereceram e continuam oferecendo uma diminuição dos custos do projeto. Isso porque, retardam a construção de protótipos físicos e a aquisição de uma grande quantidade de sensores e atuadores em fases precoces do processo de desenvolvimento do projeto BRAHMA.

Somente com a aquisição dos motores, o custo seria algo em torno de R\$ 40.000,00 (aproximadamente USD 1.000,00 cada um). Estimando os custos básicos da produção do protótipo e as horas dos engenheiros e técnicos envolvidos em R\$ 30.000,00, deixou-se de investir nesta fase do projeto aproximadamente R\$ 70.000,00.

Com relação à diminuição de tempo do projeto, pode-se somente trabalhar com uma percepção qualitativa de um ganho efetivo, pois não se possui um histórico com relação ao tempo de desenvolvimento de um projeto como este.

6.2 SUGESTÕES PARA TRABALHOS FUTUROS

Com a estrutura desenvolvida no presente trabalho, novas explorações com relação à combinação entre novos *hardwares* e modelos simulados em tempo real podem e ainda devem ser feitas. Como sugestão inicial, coloco como prioridade a distribuição de algoritmos de controle em dois ou mais microcontroladores como uma abordagem de controle distribuído, utilizando as técnicas de *Hardware-in-the-Loop*. Além, é claro de aumentar o número de dedos simulados em tempo real da mão artificial.

Iniciou-se recentemente a utilizar a estrutura de HIL para trabalhar com identificação de parâmetros físicos da planta (Albuquerque et al, 2007), porém, ainda de forma *off-line*. Como sugestão, é proposto que se continue a avançar nas técnicas de identificação, buscando a evolução para uma identificação instantânea de parâmetros físicos, como por exemplo: a flexibilidades nos acoplamentos entre os atuadores e as articulações dos dedos, bem como possíveis variações destes parâmetros em diferentes condições de trabalhos (manipulando diferentes objetos) e em função do tempo.

7. REFERÊNCIAS BIBLIOGRÁFICAS

ALBUQUERQUE A.R.L et al. (2007). The Dynamic Coupling Identification between the Artificial Robot Hand and the Actuators using Genetic Algorithm. In: XII DINAME - The International Symposium on Dynamic Problems of Mechanics, Proceedings of The International Symposium on Dynamic Problems of Mechanics.

ALBUQUERQUE, A.R.L.; CAURIN, G.A.P.; MIRANDOLA, A.L.A. (2005). Dynamic Modelling of the BRAHMA – Brazilian Anthropomorphic Hand. In. International Symposium on Multibody Systems and Mechatronics – MUSME 2005, 2005, Uberlândia/MG. Proceedings International Symposium on Multibody Systems and Mechatronics.

ALBUQUERQUE A.R.L., et al. (2004a). Desenvolvimento de um Controle Multivariável de Posição Aplicado a um Projeto de Mão Artificial Robótica. In. XV Congresso Brasileiro de Automática - CBA 2004, Gramado/RS. Anais XV Congresso Brasileiro de Automática CBA 2004.

ALBUQUERQUE A.R.L., et al.(2004b). Controle Multivariável Aplicado a um Projeto de Mão Artificial Robótica. In. Congresso Nacional de Engenharia Mecânica – CONEM 2004, Belém/PA. Anais Congresso Brasileiro de Engenharia Mecânica - CONEM 2004.

ALBUQUERQUE, A.R.L; BRUEL, P. E. N, CAURIN, G.A.P (2003). Modal Analysis of an Anthropomorphic Gripper Finger in Different Working Configurations. In: X International Symposium on Dynamic Problems of Mechanics. P. 141-146.

ALBUQUERQUE, A.R.L (2003). **Modelagem e Controle de uma Garra Antropomôrfica de Robô**. 64 f. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2003.

AN, K.N.; CHAO, E. Y.; COONEY, W.P., LINSCHIED, R.L. (1982). Normative model of human hand for biomechanical analysis. Journal of Biomechanics vol.12, p.775-788.

ARMSTRONG, T. J. (1982). Development of a biomechanical hand model for study of manual activities. In Anthropometry and Biomechanics: Theory and Application (Edited by Easterby, R., Kromer, K.H. E. Chaffin, D.)

BENALI-KHOUDJA et al. (2004). Shape and Direction Perception Using VIT-AL: A Vibro-Tactile Interface. Touch and Haptics Workshop. IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS 2004.

BERME, N.; PAUL, J.P.; PURVES, W.K. (1977). A biomechanical analysis of the metacarpophalangeal joint. J. of Biomechanics, vol.10, p.409-412.

BERTRAM, F.T. et al. (2003). Modelling and simulation for mechatronic design in automotive systems. In. Control Engineering Practice, vol.11, p. 179-190.

BICCHI, A. (2000). Hand for Dexterous Manipulation and Robust Grasping: A Difficult Road Toward Simplicity. In: IEEE TRANSACTIONS ON ROBOTICS AND AUTOMOTION, vol.16, no.6, p.652-662.

BUCHHOLZ, B. ARMSTRONG, T. (1992). Kinematic Model of the human hand to evaluate its prehensile capabilities. Journal of Biomechanics, 25, No.2 49-162.

BÜCHLER, P.; ETTLIN, A.; NELSON, B.J. (2005). Ibex - A Framework for Hardware-in-the-Loop-Simulation", COBEM18th International Congress of Mechanical Engineering. Proceedings International Congress of Mechanical Engineering.

CAURIN, G. A. P.; ALBUQUERQUE, A. R. L.; RIBEIRO, A. (2005). Sistema de Manipulação e Planejamento de Trajetórias para Mãos Artificiais. In: 4o. Congresso Temático de Dinâmica, Controle e Aplicações - DINCON 2005, 2005, Bauru, SP. Anais 4o. Congresso Temático de Dinâmica, Controle e Aplicações - DINCON 2005,.

CAURIN, G. A. P.; ALBUQUERQUE, A. R. L.; MIRANDOLA, A. L. A. (2004). Manipulation Strategy for an Anthropomorphic Robotic Hand. In: IEEE/RSJ International Conference on Intelligent Robots and Systems IROS 2004, 2004, Sendai/Japão. Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems IROS 2004,

CAURIN, G. A. P. et al. (2003). Modelagem e Simulação de Prótese Mecatrônica Brahma 1. In: 2o. Congresso Temático de Dinâmica e Controle da SBMAC - DINCON 2003, 2003, São José dos Campos/SP. Anais do 2o. Congresso Temático de Dinâmica e Controle da SBMAC - DINCON 2003.

CAURIN, G.A.P.; AGUIAR, P.M.; VALENTE, C.M.O. (2002). “Planning and Control System for an Anthropomorphic Gripper”, IEEE/RSJ Int. Conf. on Intelligent Robotics and Systems– IROS 2002, Lausanne, Suíça.

CAREFUEL, J; MARTIN, E., PIEDBOEUF, J.C. (2000). “Control Strategies for Hardware-in-the-Loop Simulation of Flexible Robots”. In: IEE Proc.-Control Theory Appl., Vol. 147, No.6.

CHAO, E. Y.; OPGRANDE, J. D.; AXMER, F.E. (1976). Three-dimensional force analysis of finger joints in selected isometric hand functions, J. Biomechanics 9, 389-396,

CHI, Z.; SHIDA, K. (2004). A new multifunctional tactile sensor for three-dimensional force measurement - Sensors and Actuators (A 111), p. 172–179.

COONEY, W.P., LUCCA, M.J. CHAO, E. Y. S. (1977). Biomedical analysis of static forces in the thumb during hand function J. Bone Jt Surg. 59A 27-36.

CRAIG, J.J. (1986). Introduction to Robotics Mechanics & Control, Addison-Wesley Publishing Comp Inc. Boston, MA, USA.

CUNHA, F.L. (1999). **Obtenção e Uso dos Acoplamentos Cinemáticos Interfalangianos e Interdigitais no Projeto de Próteses Antropomórficas para Membros Superiores.** Dissertação de Mestrado em Eng. Elétrica, Universidade Federal do Espírito Sato.

D’AZZO, J. J.; HOUPIS, H. C. (1995). Linear control system analysis and desing: conventional and modern. New York, McGraw Hill.

DAVID M, et al.(2001). Interoperability and Synchronisation of Distributed Hardware-in-the-Loop Simulation for Underwater Robot Development: Issues and Experiments. IEEE International Conference on Robotics & Automation, p.909-914.

DE CARUFEL, J.; MARTIN, E.; PIEDBOEUF, J.C. (2000). Control Strategies for Hardware-in-the-loop Simulations of Flexible Space. IEEE Proceedings – Control Theory Applications. Vol.147, nº 6, p.569-579.

FARHAB AGHILI et. al. (1999). Hardware in the Loop Simulation of Robots Performing Contact Tasks. Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space, pp.583-588.

FRASER C. (1993). A survey of users of upper limb prostheses. British Journal of Occupational Therapy. Vol.56:5. p. 166-168.

GEAR, D.W. (1971). Numerical initial value problems in ordinary differential equations. Prentice-Hall. ISBN: 0136266061.

GILLESPIE, R.B. (1996). Haptic Display of Systems with Changing Kinematic Constraints: The Virtual Piano Action. PhD thesis, Standfor University.

GLESNER, M. et al.(2002). State-of-Art in Rapid Prototyping for Mechatronic System. Mechatronics. vol.12, p. 987-998.

HAND BONES. <http://www.pncl.co.uk/~belcher/handbone.htm>, (acessado em Maio de 2005).

HOOGEN, J.; RIENER, R.; SCHMIDT, G. (2002). Control aspects of a robotic haptic interface for kinesthetic knee joint simulation. Control Engineering Practice vol.10, p. 1301-1308.

ISERMANN, R.; MULLER, N. (2003). Design of Computer Controlled Combustion Engines. Mechatronics vol. 13. p. 1067-1089.

KEJELLA GH.(1993). Consumer concerns and the functional value of prostheses to upper limb amputees. Pros Orth Intl 1993. Vol.1. p.157-163.

KENDALL, I. R.; JONES, R.P. (1999), An investigation into the use of hardware-in-the-loop simulation testing for automotive electronic control systems. In. Control Engineering Practice vol.7, p. 1343-1356.

KRENN, R; SCHAFER, B. (1999). Limitation of Hardware in the Loop Simulation of Space Robotics Dynamics Using Industrial Robots. Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space, p.681-686.

KOIVO, A.J.J. (1989), Fundamentals for Control of Robotic Manipulators. New York, John Wiley and Sons, Inc. p.478.

HAGIWARA, K. et al. (2002). Development of automatic transmission control system using hardware in the loop simulation. In: Society of Automotive Engineers of Japan, p.55-59.

LINJAMA, M. et al (2000). Hardware-in-the-loop environment for servo system controller design, tuning and testing. In. Microprocessors and Microsystems, vol. 24, p.13-21.

GLESNER, M. et al. (2002). State of the art in rapid prototyping for mechatronic systems. In. Mechatronics. vol.12. p. 987-998.

MARTINELLI, R. C., NELSON, D. B., 1948, "Prediction of Pressure Drop During Forced Circulation Boiling Water", Trans. ASME, n° 70, p.695.

MIYAMOTO, H. (1988). Prosthetics. in: Dorf, R.C., (Ed.), International Encyclopedia of Robotics,2 , Wiley, p.1269-1282.

MOHAMED, E. et al. (1992). Hardware-In-the-Loop (HIL) simulation: an application of Colbert's object-oriented software development method. Annual International Conference on Ada TRI-Ada '92 November 16 - 20, 1992, Orlando, FL USA, p. 176-188.

MSC SOFTWARE (2007). <http://www.mscsoftware.com/products/adams.cfm> - acessado em Março de 2007.

NECSULESCU, D. BASIC, G. (2003). Haptic interface development using a hardware-in-the-loop experimental setup. In: Virtual Environments, Human-Computer Interfaces and Measurement Systems, 2003. VECIMS '03. 2003 IEEE International Symposium. P. 190 – 193.

NEUMANN, D. A. (2002); Kinesiology of Musculoskeletal System, Foundations For Physical Rehabilitation. St. Louis, Mosby.

NIKRAVESH, P.E. (1988). Computer-Aided Analysis of Mechanical Systems, Prentice-Hall International, ISBN 0-13-162702-3.

OMATA, S.; MURAYAMA, Y.; CONSTANTINOU, C.E. (2004). Real time robotic tactile sensor system for the determination of the physical properties of biomaterials. Sensors and Actuators. vol. 112, n°2-3, p. 278-285.

SCHERBINA, K. Prosthetics - Upper Limb - Patient Care Management. *IN*: 11th International Trade Show and World Congress. Workshop, May 8th to 11th, 2002.

SCOTT, R.N., LOVELY, D.F.,Hudgins, B.S. in: Proc. of the 2nd Int. Conference on Rehabilitatin Engineering, Ottawa, 235-238, 1984.

SILVESTRE FILHO, G. D. (2001). **Comportamento mecânico do poliuretano derivado de óleo de mamona reforçado por fibra de carbono: Contribuição para o Projeto de Hastes de Quadril.** 156 p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2001.

STANKOVIC, J.A. (1988). Misconception about real-time computing. In: IEEE Computer, vol. 21 (10).

STANKOVIC, J.A. RAMAMRITHAN, K (1990). What is predictability for Real-Time System? In: The Jornal or Real-Time Systems, vol. 2, p. 247-254.

TEMELTAS, H. et al. (2002). “Hardware in the Loop Simulation of Robot Manipulators through Internet in Mechatronics Education. IEEE Proceedings, p.2617-2622.

VALERO, F. J., (2000). “Applying principles of robotics to understand the biomechanics, neuromuscular control and clinical rehabilitation of human digits”. In: Proc. IEEE Int. Conf. Robotics and Automation. pp. 270–275.

VAN AMERONGEN, J.; BREEDVELD, P. (2003). Modelling of physical system for the design and control of mechatronics systems. Annual Reviews in Control vol.27, p.87-117.

ZHEN LI KYTE, M. JOHNSON, B. (2004). Hardware-in-the-loop real-time simulation interface software design. In: Intelligent Transportation Systems, 2004. Proceedings. The 7th International IEEE Conference. p. 1012 – 1017.

APÊNDICE A – MODELAGEM MATEMÁTICA

Neste anexo é apresentado um procedimento para a modelagem dinâmica de uma mão artificial de cinco dedos (BRAHMA). As expressões desenvolvidas aqui (ALBUQUERQUE et al., 2005) servirão de base para o desenvolvimento de algoritmos de controle para ser utilizado no movimento e interação com objetos pela BRAHMA.

As equações de movimento da BRAHMA são obtidas pelo método de Lagrange. Uma vez que as variáveis q constituem um conjunto de coordenadas generalizadas, ou variáveis independentes (θ_n) que descrevem os movimentos das juntas de um dos dedos da BRAHMA. As equações de movimento segundo Lagrange podem ser escritas como:

$$\frac{d}{dt} \left[\frac{\partial L(\underline{q}, \underline{\dot{q}}, t)}{\partial \underline{\dot{q}}_n} \right] - \frac{\partial L(\underline{q}, \underline{\dot{q}}, t)}{\partial \underline{q}_n} = \underline{F}_n - \beta_n \underline{\dot{q}}_n \quad (A1)$$

onde $n = 1, \dots, N$ juntas independentes, F é a força generalizada (torque) e β é o coeficiente de atrito nas articulações.

Deste modo, a Função Lagrangiana, ou simplesmente Lagrangeano, é determinada pela diferença de energia cinemática e energia potencial do sistema, assumindo a seguinte forma:

$$L(\underline{q}, \underline{\dot{q}}, t) = K(\underline{q}, \underline{\dot{q}}, t) - P(\underline{q}, t) \quad (A2)$$

As equações de movimento são obtidas pela substituição da Eq. A2 na Eq. A1 para formar as derivadas apropriadas. O modelo dinâmico pode ser expresso, resumidamente, como equações diferenciais não-lineares de segunda ordem (Koivo, 1989):

$$\sum_{k=1}^N D_{nk} \ddot{\underline{q}}_k + \sum_{k=1}^N \sum_{j=1}^N D_{nkj} \dot{\underline{q}}_k \dot{\underline{q}}_j + \underline{G}_n = \underline{F}_n + \beta_n \underline{\dot{q}}_n \quad (A3)$$

Do lado esquerdo da Eq. A3, o primeiro termo representa componentes inerciais, o segundo, as componentes de Coriolis quando $k \neq j$ e centrípeta quando $k = j$, e o terceiro, o efeito gravitacional sobre a junta n . A representação de D_{nk} , D_{nkj} e G_n são apresentadas como:

$$D_{nk} = \sum_{i=\max(n,k)}^N \text{tr} \left[\frac{\partial T_0^i}{\partial \underline{q}_n} J_i \left(\frac{\partial T_0^i}{\partial \underline{q}_k} \right)' \right] \quad (A4)$$

$$D_{nkj} = \sum_{i=\max(n,k,j)}^N \text{tr} \left[\frac{\partial T_0^i}{\partial \underline{q}_n} J_i \left(\frac{\partial^2 T_0^i}{\partial \underline{q}_k \partial \underline{q}_j} \right)' \right] \quad (A5)$$

$$\underline{G}_n = - \sum_{i=n}^N m_i \underline{g}' \frac{\partial T_0^i}{\partial \underline{q}_n} \underline{p}_i \quad (A6)$$

onde $\text{tr}[]$ significa a diagonal da matriz, T são as matrizes de transformadas homogêneas (CAURIN, 2003), $\underline{p}_i$ é o centro de gravidade do i -ésimo link, J são as matrizes de inércia e “ N ” é o número de articulação do manipulador.

Com a equação dinâmica BRAHMA, determina-se os torques necessários das articulações para seguir uma trajetória desejada. Essa equação completa pode ser expressa por:

$$\begin{bmatrix} D_{44} & \cdots & D_{4k} \\ \vdots & \ddots & \vdots \\ D_{n4} & \cdots & D_{nk} \end{bmatrix} \begin{bmatrix} \ddot{\underline{q}}_4 \\ \vdots \\ \ddot{\underline{q}}_N \end{bmatrix} + \begin{bmatrix} \dot{\underline{q}}' C_4(\underline{q}) \dot{\underline{q}} \\ \vdots \\ \dot{\underline{q}}' C_N(\underline{q}) \dot{\underline{q}} \end{bmatrix} + \begin{bmatrix} G_4(\underline{q}) \\ \vdots \\ G_N(\underline{q}) \end{bmatrix} = \begin{bmatrix} \underline{\tau}_4 - \beta_4 \dot{\underline{q}}_4 \\ \vdots \\ \underline{\tau}_N - \beta_N \dot{\underline{q}}_N \end{bmatrix} \quad (A7)$$

$$e \quad C_N(\underline{q}) = \begin{bmatrix} D_{n44}(\underline{q}) & \cdots & D_{n4j}(\underline{q}) \\ \vdots & \ddots & \vdots \\ D_{nk}(\underline{q}) & \cdots & D_{nkj}(\underline{q}) \end{bmatrix} \quad (A8)$$

onde a diagonal dessa matriz $CN(q)$ (Eq.A8), encontra-se os componentes de aceleração centrípeta e os outros termos estão associados a aceleração de Coriolis. A equação dinâmica da BRAHMA pode ser expressa resumidamente por:

$$D(\underline{q})\ddot{\underline{q}} + C(\underline{q}, \dot{\underline{q}}) + G(\underline{q}) = \underline{F} \quad (A9)$$

Como a BRAHMA é um sistema complexo por sua natureza, a modelagem dinâmica é apresentada para um dos dedos da BRAHMA. Este cálculo pode ser estendido de forma análoga para os outros dedos da mão. Assim, o dedo indicador será tomado como padrão para a modelagem. No caso do dedo polegar, que possui características singulares, é tratado como um caso particular.

Figura A1: Representação dos corpos de dedo indicador.

Todos os dedos da mão artificial podem ser tratados de forma análoga. Inicialmente, desenvolveu-se a modelagem dinâmica do dedo indicador da BRAHMA, o mesmo procedimento pode ser expandido para os outros dedos.

Desse modo, considerando-se as massas dos últimos três corpos do dedo indicador (Fig.5.1) da BRAHMA como m_4 , m_5 e m_6 ; as matrizes de transformadas homogêneas

definidas como $T_6^5, T_5^4 e T_4^3$; os torques aplicados em cada corpo como τ_4, τ_5 e τ_6 ; os coeficientes de atrito viscoso β_4, β_5 e β_6 ; e $N = 3$ que representa o número de corpos rígidos, ou seja, falante (corpo 4), falanginha (corpo 5), falangeta (corpo 6); $k = 4, 5, 6$ e $j = 4, 5, 6$. Onde k e j são os sistemas de coordenadas fixas às articulações. Finalmente, defini-se a dinâmica da BRAHMA por meio da Eq. A3:

Articulação 4:

$$D_{44} \ddot{q}_4 + D_{45} \ddot{q}_5 + D_{46} \ddot{q}_6 + \sum_{k=4}^6 (D_{4k4} \dot{q}_k \dot{q}_4 + D_{4k5} \dot{q}_k \dot{q}_5 + D_{4k6} \dot{q}_k \dot{q}_6) + G_4 = \tau_4 - \beta_4 \dot{q}_4$$

Articulação 5:

$$D_{54} \ddot{q}_4 + D_{55} \ddot{q}_5 + D_{56} \ddot{q}_6 + \sum_{k=4}^6 (D_{5k4} \dot{q}_k \dot{q}_4 + D_{5k5} \dot{q}_k \dot{q}_5 + D_{5k6} \dot{q}_k \dot{q}_6) + G_5 = \tau_5 - \beta_5 \dot{q}_5$$

Articulação 6:

$$D_{64} \ddot{q}_4 + D_{65} \ddot{q}_5 + D_{66} \ddot{q}_6 + \sum_{k=4}^6 (D_{6k4} \dot{q}_k \dot{q}_4 + D_{6k5} \dot{q}_k \dot{q}_5 + D_{6k6} \dot{q}_k \dot{q}_6) + G_6 = \tau_6 - \beta_6 \dot{q}_6$$

Com o cálculo dos termos D_{nk} , D_{nkj} e G_n da expressão acima, pode-se encontrar a função lagrangiana da BRAHMA baseada em equações de energia cinemática e potencial. Desse modo, quando é imposta uma trajetória nas articulações da BRAHMA, recorrem-se às equações lagrangianas para determinar os torques necessários da juntas e, assim, resolver eficientemente o problema da dinâmica inversa do sistema.

Assumindo que J_{Zi} é o momento de inércia no eixo z_i fixo na articulação i (Fig. A2) e l_i é o comprimento dos corpos, obtém-se a fórmula para o cálculo do momento de inércia de cada corpo em relação ao referencial 3:

$$J_{Zi} = J_{3i} + m_i \left(\frac{l_i}{2} \right)^2$$

Figura A2: Representação do sistema de coordenadas do dedo indicador da BRAHMA.

Para obter os termos da Eq. A3, é necessário calcular as derivadas parciais das matrizes de transformadas homogêneas T_i^{i-1} :

$$\frac{\partial T_0^i}{\partial q_k} = T_0^{k-1} Q_k T_{k-1}^k T_k^i \quad \text{para } 1 \leq k \leq i$$

$$\frac{\partial^2 T_0^i}{\partial q_k \partial q_j} = \begin{cases} T_0^{k-1} (Q_k T_{k-1}^k) \Gamma_k^{j-1} (Q_j T_{j-1}^j) \Gamma_j^i & \text{para } 1 \leq k < j \leq i \\ T_0^{k-1} (Q_k^2 T_{k-1}^k) \Gamma_k^i & \text{para } k = j \leq i \end{cases}$$

O próximo passo é calcular as componentes inerciais D_{nk} , os componentes de Coriolis e centrípeta D_{nkj} e, o efeito gravitacional sobre a junta n é o $\underline{G}_n$:

Resolvendo a Eq. A4 analiticamente, tem-se os seguintes componentes inerciais:

$$D_{44} = I_{36} + I_{35} + I_{34} + m_4 \left(\frac{a_B}{2} \right)^2 + m_5 \left(\frac{a_C}{2} \right)^2 + m_5 a_C a_B \cos(\theta_5) + m_5 a_B^2 + m_6 \left(\frac{a_D}{2} \right)^2$$

$$+ 2 m_6 a_C a_B \cos(\theta_5) + m_6 a_C^2 + m_6 a_B^2 + m_6 a_C a_D \cos(\theta_6) + m_6 a_D a_B \cos(\theta_5 + \theta_6)$$

$$D_{45} = I_{36} + I_{35} + m_5 \left(\frac{a_C}{2} \right)^2 + m_5 \frac{a_C}{2} a_B \cos(\theta_5) + m_6 \left(\frac{a_D}{2} \right)^2 + m_6 a_C a_B \cos(\theta_5)$$

$$+ m_6 a_C^2 + m_6 a_C a_D \cos(\theta_6) + m_6 \frac{a_D}{2} a_B \cos(\theta_5 + \theta_6)$$

$$D_{46} = I_{36}$$

$$D_{54} = I_{36} + I_{35} + m_5 \cdot \left(\frac{a_C}{2} \right)^2 + m_5 \cdot \frac{a_C}{2} \cdot a_B \cdot \cos(\theta_5) + m_6 \cdot \left(\frac{a_D}{2} \right)^2 + m_6 \cdot a_C \cdot a_D \cdot \cos(\theta_6)$$

$$+ m_6 \cdot a_C^2 + m_6 \cdot a_C \cdot a_B \cdot \cos(\theta_5) + m_6 \cdot \frac{a_D}{2} \cdot a_B \cdot \cos(\theta_5 + \theta_6)$$

$$D_{55} = I_{36} + I_{35} + m_5 \cdot \left(\frac{a_C}{2} \right)^2 + m_6 \cdot \left(\frac{a_D}{2} \right)^2 + m_6 \cdot a_C \cdot a_D \cdot \cos(\theta_6) + m_6 \cdot a_C^2$$

$$D_{56} = I_{36}$$

$$D_{64} = I_{36} + m_6 \cdot \left(\frac{a_D}{2} \right)^2 + m_6 \cdot \left(\frac{a_D}{2} \right) \cdot a_C \cdot \cos(\theta_6) + m_6 \cdot \left(\frac{a_D}{2} \right) \cdot a_B \cdot \cos(\theta_5 + \theta_6)$$

$$D_{65} = I_{36} + m_6 \cdot \left(\frac{a_D}{2} \right)^2 + m_6 \cdot \left(\frac{a_D}{2} \right) \cdot a_C \cdot \cos(\theta_6)$$

$$D_{66} = I_{36}$$

Resolvendo a Eq. A5 analiticamente, tem-se os seguintes componentes de centrípeta.

Nesse caso as componentes de Coriolis são nulas devido a ausência de juntas de translação:

$$\begin{aligned} \underline{\dot{q}}' C_4 \underline{\dot{q}} &= -m_5 \cdot a_C \cdot a_B \cdot \sin(\theta_5) \cdot \dot{\theta}_4 \cdot \dot{\theta}_5 - 2 \cdot m_6 \cdot a_C \cdot a_B \cdot \sin(\theta_5) \cdot \dot{\theta}_4 \cdot \dot{\theta}_5 - m_5 \cdot \frac{a_C}{2} \cdot a_B \cdot \sin(\theta_5) \cdot \dot{\theta}_5^2 \\ &- m_6 \cdot a_C \cdot a_B \cdot \sin(\theta_5) \cdot \dot{\theta}_5^2 - m_6 \cdot a_D \cdot a_B \cdot \sin(\theta_5 + \theta_6) \cdot \dot{\theta}_4 \cdot \dot{\theta}_5 - m_6 \cdot a_D \cdot a_B \cdot \sin(\theta_5 + \theta_6) \cdot \dot{\theta}_5^2 \\ \underline{\dot{q}}' C_5 \underline{\dot{q}} &= m_5 \cdot \frac{a_C}{2} \cdot a_B \cdot \sin(\theta_5) \cdot \dot{\theta}_4^2 + m_6 \cdot \frac{a_D}{2} \cdot a_B \cdot \sin(\theta_5 + \theta_6) \cdot \dot{\theta}_4^2 + m_6 \cdot a_C \cdot a_B \cdot \sin(\theta_5) \cdot \dot{\theta}_4^2 \\ \underline{\dot{q}}' C_6 \underline{\dot{q}} &= m_6 \cdot \left(\frac{a_D}{2} \right) \cdot a_B \cdot \sin(\theta_5 + \theta_6) \cdot \dot{\theta}_4^2 + m_6 \cdot \left(\frac{a_D}{2} \right) \cdot a_C \cdot \sin(\theta_6) \cdot (\dot{\theta}_4 + \dot{\theta}_5)^2 \end{aligned}$$

Resolvendo a Eq. A6 analiticamente, tem-se os seguintes componentes devido a gravidade:

$$\begin{aligned} G_4(\underline{q}) &= m_4 \cdot g \cdot \frac{a_B}{2} \cdot \sin(\theta_4) + m_5 \cdot g \cdot a_B \cdot \sin(\theta_4) + m_6 \cdot g \cdot a_B \cdot \sin(\theta_4) + m_5 \cdot g \cdot \frac{a_C}{2} \cdot \sin(\theta_4 + \theta_5) \\ &+ m_6 \cdot g \cdot a_C \cdot \sin(\theta_4 + \theta_5) + m_6 \cdot g \cdot \left(\frac{a_D}{2} \right) \cdot \sin(\theta_4 + \theta_5 + \theta_6) \\ G_5(\underline{q}) &= m_5 \cdot g \cdot \frac{a_C}{2} \cdot \sin(\theta_4 + \theta_5) + m_6 \cdot g \cdot a_C \cdot \sin(\theta_4 + \theta_5) + m_6 \cdot g \cdot \left(\frac{a_D}{2} \right) \cdot \sin(\theta_4 + \theta_5 + \theta_6) \end{aligned}$$

$$G_6(\underline{q}) = m_6 \cdot g \cdot \left(\frac{a_D}{2} \right) \cdot \sin(\theta_4 + \theta_5 + \theta_6)$$

TABELA A1: PROPRIEDADES FÍSICAS DO DEDO INDICADOR DA BRAHMA

	Falange	Falanginha	Falangeta
CM_x (mm)	-0,37	-0,37	-0,37
CM_y (mm)	0	0	-0,02
CM_z (mm)	76,09	142,23	114,28
Massa 10⁻³(kg)	7	3	5
Ixx (Kg.mm²)	0,28	0,041	0,13
Iyy (Kg.mm²)	2,783	0,616	1,203
Izz (Kg.mm²)	2,718	0,607	1,162

Impondo-se inicialmente um determinado movimento (Fig.A3) às articulações da BRAHMA, se obtém por meio de um integrador numérico a variação da energia cinética (Fig A4) no tempo, a variação da energia potencial no tempo (Fig.A5) e por fim, a variação do torque no tempo (Fig.A6).

Figura A3: Função sigmoide de entrada

Figura A4: Gráfico de resposta da Energia Cinética no Tempo

Figura A5: Gráfico de resposta da Energia Potencial no Tempo

Figura A6: Gráfico de resposta do Torque nas articulações no Tempo

Com intuito de avaliar o quanto cada componente da equação de movimento (Eq.8) influencia nos valores de torques nas articulações do dedo é apresentado graficamente e em uma faixa de tempo os valores dessas componentes.

Figura A7: Torque na articulação 6 e suas componentes devido a gravidade, inércia e as acelerações centrípeta

Figura A8: Torque na articulação 5 e suas componentes devido a gravidade, inércia e as acelerações centrípeta

Figura A9: Torque na articulação 4 e suas componentes devido a gravidade, inércia e as acelerações centrípeta

Impõem-se agora o mesmo movimento da Fig.A3, porém, com uma velocidade duas vezes maior (Fig.A10) às articulações da BRAHMA, obtendo-se a variação da energia cinética (Fig A11) no tempo, a variação da energia potencial no tempo (Fig.A12) e por fim, a variação do torque no tempo (Fig.A13).

Figura A10: Função sigmóide de entrada

Figura A.11: Gráfico de resposta da Energia Cinética no Tempo

Figura A.12: Gráfico de resposta da Energia Potencial no Tempo

Figura A.13: Gráfico de resposta do Torque nas articulações no Tempo

Avalia-se o quanto cada componente da equação de movimento (Eq.A8) influencia nos valores de torques nas articulações do dedo com a nova função de entrada de movimento por meio das seguintes respostas gráficas:

Figura A.14: Torque na articulação 6 e suas componentes devido a gravidade, inércia e as acelerações centrípeta

Figura A.15: Torque na articulação 5 e suas componentes devido a gravidade, inércia e as acelerações centrípeta

Figura A.16: Torque na articulação 4 e suas componentes devido a gravidade, inércia e as acelerações centrípeta

APÊNDICE B - TOPOLOGIA DO MODELO BRAHMA

O modelo para a simulação (ALBUQUERQUE et al., 2005) deste trabalho foi obtido com auxílio de um programa MBS numérico chamado ADAMS que possui rotinas para a geração das equações e para a solução do sistema. Este programa possui aplicativos de pré e pós-processamento. Estas ferramentas facilitaram a criação do modelo da BRAHMA e a posterior simulação com análise e apresentação gráfica dos resultados.

A seguir a topologia do modelo no ADAMS é apresentado como parte da documentação do modelo:

Figura B1: Interface gráfica do Modelo BRAHMA (conjunto dedos + atuadores)

Topology of model: model_1

Ground Part: ground

Part ground

Is connected to:

carpo	via JOINT_18 (Fixed Joint)
motor_C3	via SPRING_C3.sforce (Single_Component_Force)
motor_C2	via SPRING_C2.sforce (Single_Component_Force)
motor_C1_1	via SPRING_C1_1.sforce (Single_Component_Force)
motor_D3	via SPRING_D3.sforce (Single_Component_Force)
motor_D2	via SPRING_D2.sforce (Single_Component_Force)
motor_D1_1	via SPRING_D1_1.sforce (Single_Component_Force)
motor_E3	via SPRING_E3.sforce (Single_Component_Force)
motor_E2	via SPRING_E2.sforce (Single_Component_Force)
motor_E1	via SPRING_E1.sforce (Single_Component_Force)
motor_E1_1	via SPRING_E1_1.sforce (Single_Component_Force)
motor_C3	via JOINT_MOTOR_C3 (Revolute Joint)
motor_C3	via COUPLER_C3 (Coupler)
motor_C2	via JOINT_MOTOR_C2 (Revolute Joint)
motor_C2	via COUPLER_C2 (Coupler)
motor_C3	via Torque_Motor_C3 (Single_Component_Force)
motor_C1	via Torque_motor_C1 (Single_Component_Force)
motor_C1_1	via Torque_motor_C1_1 (Single_Component_Force)
motor_C2	via Torque_motor_C2 (Single_Component_Force)
motor_D2	via Torque_motor_D2 (Single_Component_Force)
motor_D1	via Torque_motor_D1 (Single_Component_Force)
motor_D1_1	via Torque_motor_D1_1 (Single_Component_Force)
motor_E3	via Torque_motor_E3 (Single_Component_Force)
motor_E2	via Torque_motor_E2 (Single_Component_Force)
motor_E1	via Torque_motor_E1 (Single_Component_Force)
motor_E1_1	via Torque_motor_E1_1 (Single_Component_Force)
motor_D3	via JOINT_Motor_D3 (Revolute Joint)
motor_D3	via COUPLER_D3 (Coupler)
motor_D2	via JOINT_MOTOR_D2 (Revolute Joint)
motor_D2	via COUPLER_D2 (Coupler)
motor_D1_1	via JOINT_MOTOR_D1_1 (Revolute Joint)
motor_D1_1	via COUPLER_D1_1 (Coupler)
motor_C1_1	via JOINT_MOTOR_C1_1 (Revolute Joint)
motor_C1_1	via COUPLER_C1_1 (Coupler)
motor_D3	via SFORCE_MOTOR_D3 (Single_Component_Force)
motor_E3	via JOINT_MOTOR_E3 (Revolute Joint)
motor_E3	via COUPLER_E3 (Coupler)
motor_E2	via JOINT_MOTOR_E2 (Revolute Joint)
motor_E2	via COUPLER_E2 (Coupler)
motor_E1	via JOINT_MOTOR_E1 (Revolute Joint)
motor_E1	via COUPLER_E1 (Coupler)
motor_E1_1	via JOINT_MOTOR_E1_1 (Revolute Joint)
motor_E1_1	via COUPLER_E1_1 (Coupler)
motor_C1	via JOINT_MOTOR_C1 (Revolute Joint)

motor_C1	via COUPLER_C1	(Coupler)
motor_D1	via JOINT_MOTOR_D1	(Revolute Joint)
motor_D1	via COUPLER_D1	(Coupler)

Part carpo

Is connected to:

metacarpo2	via JOINT_c0	(Fixed Joint)
metacarpo_1	via JOINT_d0	(Fixed Joint)
ground	via JOINT_18	(Fixed Joint)
PART_40	via JOINT_E1	(Revolute Joint)
PART_40	via COUPLER_E1	(Coupler)

Part falanginha0

Is connected to:

falange_0	via JOINT_e3	(Revolute Joint)
falange_0	via COUPLER_E3	(Coupler)

Part falange_0

Is connected to:

falanginha0	via JOINT_e3	(Revolute Joint)
falanginha0	via COUPLER_E3	(Coupler)
metarcapo_0	via JOINT_e2	(Revolute Joint)
metarcapo_0	via COUPLER_E2	(Coupler)

Part metarcapo_0

Is connected to:

falange_0	via JOINT_e2	(Revolute Joint)
falange_0	via COUPLER_E2	(Coupler)
PART_43	via JOINT_E1_1	(Revolute Joint)
PART_43	via COUPLER_E1_1	(Coupler)

Part metacarpo_1

Is connected to:

carpo	via JOINT_d0	(Fixed Joint)
PART_39	via JOINT_D1	(Revolute Joint)
PART_39	via COUPLER_D1	(Coupler)

Part falange1

Is connected to:

falangeta1	via JOINT_d2	(Revolute Joint)
falangeta1	via COUPLER_D2	(Coupler)
PART_42	via JOINT_D1_1	(Revolute Joint)
PART_42	via COUPLER_D1_1	(Coupler)

Part falangeta1

Is connected to:

falanginha1	via JOINT_d3	(Revolute Joint)
falanginha1	via COUPLER_D3	(Coupler)
falange1	via JOINT_d2	(Revolute Joint)
falange1	via COUPLER_D2	(Coupler)

Part falanginha1

Is connected to:

falangeta1	via JOINT_d3	(Revolute Joint)
falangeta1	via COUPLER_D3	(Coupler)

Part metacarpo2

Is connected to:

carpo	via JOINT_c0	(Fixed Joint)
PART_38	via JOINT_C1_1	(Revolute Joint)
PART_38	via COUPLER_C1_1	(Coupler)

Part falange2

Is connected to:

falangeta2	via JOINT_c2	(Revolute Joint)
falangeta2	via COUPLER_C2	(Coupler)
PART_41	via JOINT_C1	(Revolute Joint)
PART_41	via COUPLER_C1	(Coupler)

Part falangeta2

Is connected to:

falange2	via JOINT_c2	(Revolute Joint)
falange2	via COUPLER_C2	(Coupler)
falanginha2	via JOINT_c3	(Revolute Joint)
falanginha2	via COUPLER_C3	(Coupler)

Part falanginha2

Is connected to:

falangeta2	via JOINT_c3	(Revolute Joint)
falangeta2	via COUPLER_C3	(Coupler)

Part motor_C3

Is connected to:

ground	via SPRING_C3.sforce	(Single_Component_Force)
ground	via JOINT_MOTOR_C3	(Revolute Joint)
ground	via COUPLER_C3	(Coupler)
ground	via Torque_Motor_C3	(Single_Component_Force)

Part motor_C2

Is connected to:

ground	via SPRING_C2.sforce	(Single_Component_Force)
ground	via JOINT_MOTOR_C2	(Revolute Joint)
ground	via COUPLER_C2	(Coupler)
ground	via Torque_motor_C2	(Single_Component_Force)

Part motor_C1

Is connected to:

motor_C1_1	via SPRING_C1.sforce	(Single_Component_Force)
ground	via Torque_motor_C1	(Single_Component_Force)
ground	via JOINT_MOTOR_C1	(Revolute Joint)

ground via COUPLER_C1 (Coupler)

Part motor_C1_1

Is connected to:

motor_C1 via SPRING_C1.sforce (Single_Component_Force)
 ground via SPRING_C1_1.sforce (Single_Component_Force)
 ground via Torque_motor_C1_1 (Single_Component_Force)
 ground via JOINT_MOTOR_C1_1 (Revolute Joint)
 ground via COUPLER_C1_1 (Coupler)

Part motor_D3

Is connected to:

ground via SPRING_D3.sforce (Single_Component_Force)
 ground via JOINT_Motor_D3 (Revolute Joint)
 ground via COUPLER_D3 (Coupler)
 ground via SFORCE_MOTOR_D3 (Single_Component_Force)

Part motor_D2

Is connected to:

ground via SPRING_D2.sforce (Single_Component_Force)
 ground via Torque_motor_D2 (Single_Component_Force)
 ground via JOINT_MOTOR_D2 (Revolute Joint)
 ground via COUPLER_D2 (Coupler)

Part motor_D1

Is connected to:

motor_D1_1 via SPRING_D1.sforce (Single_Component_Force)
 ground via Torque_motor_D1 (Single_Component_Force)
 ground via JOINT_MOTOR_D1 (Revolute Joint)
 ground via COUPLER_D1 (Coupler)

Part motor_D1_1

Is connected to:

motor_D1 via SPRING_D1.sforce (Single_Component_Force)
 ground via SPRING_D1_1.sforce (Single_Component_Force)
 ground via Torque_motor_D1_1 (Single_Component_Force)
 ground via JOINT_MOTOR_D1_1 (Revolute Joint)
 ground via COUPLER_D1_1 (Coupler)

Part motor_E3

Is connected to:

ground via SPRING_E3.sforce (Single_Component_Force)
 ground via Torque_motor_E3 (Single_Component_Force)
 ground via JOINT_MOTOR_E3 (Revolute Joint)
 ground via COUPLER_E3 (Coupler)

Part motor_E2

Is connected to:

ground via SPRING_E2.sforce (Single_Component_Force)
 ground via Torque_motor_E2 (Single_Component_Force)

ground via JOINT_MOTOR_E2 (Revolute Joint)
ground via COUPLER_E2 (Coupler)

Part motor_E1

Is connected to:

ground via SPRING_E1.sforce (Single_Component_Force)
ground via Torque_motor_E1 (Single_Component_Force)
ground via JOINT_MOTOR_E1 (Revolute Joint)
ground via COUPLER_E1 (Coupler)

Part motor_E1_1

Is connected to:

ground via SPRING_E1_1.sforce (Single_Component_Force)
ground via Torque_motor_E1_1 (Single_Component_Force)
ground via JOINT_MOTOR_E1_1 (Revolute Joint)
ground via COUPLER_E1_1 (Coupler)

Part PART_38

Is connected to:

PART_41 via JOINT_fix_C1 (Fixed Joint)
metacarpo2 via JOINT_C1_1 (Revolute Joint)
metacarpo2 via COUPLER_C1_1 (Coupler)

Part PART_39

Is connected to:

PART_42 via JOINT_fix_D1 (Fixed Joint)
metacarpo_1 via JOINT_D1 (Revolute Joint)
metacarpo_1 via COUPLER_D1 (Coupler)

Part PART_40

Is connected to:

PART_43 via JOINT_fix_E1 (Fixed Joint)
carpo via JOINT_E1 (Revolute Joint)
carpo via COUPLER_E1 (Coupler)

Part PART_41

Is connected to:

PART_38 via JOINT_fix_C1 (Fixed Joint)
falange2 via JOINT_C1 (Revolute Joint)
falange2 via COUPLER_C1 (Coupler)

Part PART_42

Is connected to:

PART_39 via JOINT_fix_D1 (Fixed Joint)
falange1 via JOINT_D1_1 (Revolute Joint)
falange1 via COUPLER_D1_1 (Coupler)

Part PART_43

Is connected to:

PART_40 via JOINT_fix_E1 (Fixed Joint)

metarcarpo_0	via	JOINT_E1_1	(Revolute Joint)
metarcarpo_0	via	COUPLER_E1_1	(Coupler)

APÊNDICE C – CÓDIGO FONTE DAS SIMULAÇÕES EM HIL

Controle de Posição

```

#define S_FUNCTION_NAME controle_moderno
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"

extern real_T readEncoder(void);
extern void resetEncoder(void);
extern real_T motordrive(short int v);
extern void motorreset(void);

int controleReset=0;
float posicaoAnterior=0.0,integralErro=0.0,
      erroAnterior=0.0;

/*
 * mdlInitializeSizes - initialize the sizes array
 */
static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams( S, 0); /*number of input arguments*/
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 2);
    ssSetInputPortDirectFeedThrough(S, 0, 2);
    if (!ssSetNumOutputPorts(S,1)) return;
    ssSetOutputPortWidth(S, 0, 2);
    ssSetNumSampleTimes(S, 1);
}
/*
 * mdlInitializeSampleTimes - indicate that this S-function runs
 * at the rate of the source (driving block)
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}

/*
 * mdlOutputs - compute the outputs by calling my_alg, which
 * resides in another module, my_alg.c
 */

static void mdlOutputs(SimStruct *S, int_T tid)
{

```

```

float posicaoAtual=0.0,
    erro = 0.0,
    tensaoAplicada=0.0,
    matrizInter[2] = {0,0},
    referencia=0.0,
    velocidade=0.0,
    estados[2]={0.0,0.0},
    tensao2=0.0,
    tensao1=0.0;

/* declaracao das entradas e saidas*/

InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);

real_T *y = ssGetOutputPortRealSignal(S,0);

if (controleReset==0){

    resetEncoder();

    motorreset();

    controleReset=1;

    integralErro=0.0;

    posicaoAnterior=0.0;

    erroAnterior=0.0;
}

if (controleReset==1){

    referencia = *uPtrs[0];

    posicaoAtual = readEncoder();

    erro = referencia - posicaoAtual;

    velocidade = (posicaoAtual-posicaoAnterior)/0.004; /* 0.004 eh a taxa de amostragem */

    posicaoAnterior = posicaoAtual;

    integralErro = integralErro + ((erro+erroAnterior)/2)*0.004;
    erroAnterior= erro; /* modificacao feita por Andre Lins 19/10/05 */

    estados[0] = posicaoAtual;
    estados[1] = velocidade;

    matrizInter[0] = 0.0003*velocidade;

```

```

matrizInter[1] = 0.0003*posicaoAtual;

tensao2 = 1.34*matrizInter[0]-2834*matrizInter[1];

tensao1=integralErro*9.9;

tensaoAplicada=(tensao1+tensao2);

if (tensaoAplicada>10) tensaoAplicada=10;

if (tensaoAplicada<-10) tensaoAplicada=-10;

*(y+1)=tensaoAplicada;

tensaoAplicada = tensaoAplicada*3276.7*(*uPtrs[1]); /* 3276.7 eh um fator de correcao
do tipo de variavel entre as funcoes */

motordrive(tensaoAplicada);

*y = posicaoAtual;

if (posicaoAtual>500) controleReset=0;

if (posicaoAtual<-500) controleReset=0;
}

/*comentario*/

}

/*
* mdlTerminate - called when the simulation is terminated.
*/
static void mdlTerminate(SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE /* Is this file being compiled as a MEX-file? */
#include "simulink.c" /* MEX-file interface mechanism */
#else
#include "cg_sfun.h" /* Code generation registration function */
#endif

```

Hardware-in-the-Loop I

```

/*
 * Real-Time Workshop code generation for Simulink model "hil2.mdl".
 *
 * Model Version          : 1.28
 * Real-Time Workshop file version : 5.0 $Date: 2002/05/30 19:21:33 $
 * Real-Time Workshop file generated on : Wed Nov 02 15:38:29 2005
 * TLC version            : 5.0 (Jun 18 2002)
 * C source code generated on      : Wed Nov 02 15:38:30 2005
 */

#include <math.h>
#include <string.h>
#include "hil2.h"
#include "hil2_private.h"
#include "ext_work.h"

#include "hil2_dt.h"

/* Block signals (auto storage) */
BlockIO rtB;

/* Block states (auto storage) */
D_Work rtDWork;

/* Parent Simstruct */
static rtModel_hil2 model_S;
rtModel_hil2 *const rtM_hil2 = &model_S;

/* Initial conditions for root system: '<Root>' */
void MdlInitialize(void)
{
    /* Derivative Block: <Root>/Derivative */
    rtDWork.Derivative_RWORK.TimeStampA = rtInf;
    rtDWork.Derivative_RWORK.TimeStampB = rtInf;

    /* DiscreteIntegrator Block: <Root>/Integrador Discreto */
    rtDWork.Integrador_Discreto_DSTATE = rtP.Integrador_Discreto_IC;
}

/* Start for root system: '<Root>' */
void MdlStart(void)
{
    /* DiscretePulseGenerator Block: <Root>/Pulse Generator */
    {
        int_T Ns;
        real_T tFirst = rtmGetTStart(rtM_hil2);
    }
}

```

```

Ns = (int_T)floor(tFirst / 0.004 + 0.5);
if (Ns <= 0) {
    rtDWork.Pulse_Generator_IWORK.ClockTicksCounter = Ns;
} else {
    rtDWork.Pulse_Generator_IWORK.ClockTicksCounter = Ns -
        (int_T)(rtP.Pulse_Generator_Period*floor((real_T)Ns /
            rtP.Pulse_Generator_Period));
}
}

MdlInitialize();
}

/* Outputs for root system: '<Root>' */
void MdlOutputs(int_T tid)
{
    /* local block i/o variables */
    real_T rtb_TmpHiddenBuffer_Feeding_inv[2];
    real_T rtb_inv_C[2];
    real_T rtb_temp11;
    real_T rtb_temp12;

    /* tid is required for a uniform function interface. This system
     * is single rate, and in this case, tid is not accessed. */
    UNUSED_PARAMETER(tid);

    /* DiscretePulseGenerator: '<Root>/Pulse Generator' */
    rtb_temp11 =
        (rtDWork.Pulse_Generator_IWORK.ClockTicksCounter < rtP.Pulse_Generator_Duty
        &&
        rtDWork.Pulse_Generator_IWORK.ClockTicksCounter >= 0) ?
        rtP.Pulse_Generator_Amp :
        0.0;
    if (rtDWork.Pulse_Generator_IWORK.ClockTicksCounter >=
        rtP.Pulse_Generator_Period-1) {
        rtDWork.Pulse_Generator_IWORK.ClockTicksCounter = 0;
    } else {
        (rtDWork.Pulse_Generator_IWORK.ClockTicksCounter)++;
    }

    /* Saturate: '<Root>/Saturation'
     *
     * Regarding '<Root>/Saturation':
     * Lower limit: rtP.Saturation_LowerSat
     * Upper limit: rtP.Saturation_UpperSat
     */
    if (rtb_temp11 >= rtP.Saturation_UpperSat) {
        rtB.Saturation = rtP.Saturation_UpperSat;
    } else if (rtb_temp11 <= rtP.Saturation_LowerSat) {
        rtB.Saturation = rtP.Saturation_LowerSat;
    }
}

```

```

} else {
    rtB.Saturation = rtb_temp11;
}

/* DiscreteTransferFcn: '<Root>/TF Motor' */
rtB.TF_Motor = rtP.TF_Motor_C[0]*rtDWork.TF_Motor_DSTATE[0] +
    rtP.TF_Motor_C[1]*rtDWork.TF_Motor_DSTATE[1];

/* Level2 S-Function Block: <Root>/S-Function Driver (controle_moderno) */
{
    SimStruct *rts = rtM_hil2->childSfunctions[0];
    sfcnOutputs(rts, tid);
}

/* DiscreteTransferFcn: '<Root>/Discrete Transfer Fcn3' */
rtB.Discrete_Transfer_Fcn3 =
    rtP.Discrete_Transfer_Fcn3_C[0]*rtDWork.Discrete_Transfer_Fcn3_DSTATE[0]
    + rtP.Discrete_Transfer_Fcn3_C[1]*rtDWork.Discrete_Transfer_Fcn3_DSTATE[1]
    + rtP.Discrete_Transfer_Fcn3_C[2]*rtDWork.Discrete_Transfer_Fcn3_DSTATE[2];

/* Gain: '<Root>/K1'
 *
 * Regarding '<Root>/K1':
 *   Gain value: rtP.K1_Gain
 */
rtB.K1 = rtB.S_Function_Driver[1] * rtP.K1_Gain;

/* Derivative Block: <Root>/Derivative */
{
    real_T t = rtmGetT(rtM_hil2);
    real_T timeStampA = rtDWork.Derivative_RWORK.TimeStampA;
    real_T timeStampB = rtDWork.Derivative_RWORK.TimeStampB;

    if (timeStampA >= t && timeStampB >= t) {
        rtb_temp12 = 0.0;
    } else {
        real_T deltaT;
        real_T *lastBank = &rtDWork.Derivative_RWORK.TimeStampA;
        if (timeStampA < timeStampB) {
            if (timeStampB < t) {
                lastBank += 2;
            }
        } else if (timeStampA >= t) {
            lastBank += 2;
        }
        deltaT = t - *lastBank++;
        rtb_temp12 = (rtB.TF_Motor - *lastBank++) / deltaT;
    }
}

```



```

/* DiscreteIntegrator: '<Root>/Integrador Discreto'
*
* Regarding '<Root>/Integrador Discreto':
*   Unlimited, w/o Saturation Port
*/
rtb_temp11 = rtDWork.Integrador_Discreto_DSTATE;

/* Gain: '<Root>/K2'
*
* Regarding '<Root>/K2':
*   Gain value: rtP.K2_Gain
*/
rtB.K2 = rtb_temp11 * rtP.K2_Gain;

/* Sum: '<Root>/Sum' */
rtB.Sum = - rtB.TF_Motor + rtB.Saturation;

/* HiddenBuffer */
rtb_TmpHiddenBuffer_Feeding_inv[0] = rtB.TF_Motor;
rtb_TmpHiddenBuffer_Feeding_inv[1] = rtb_temp12;

/* Gain: '<Root>/inv(C)'
*
* Regarding '<Root>/inv(C)':
*   Gain value: [ 0 0.0003; 0.0003 0]
*/
{
    static const int dims[3] = { 2, 2, 1 };
    rt_MatMultRR_Dbl((real_T *)&rtb_inv_C[0], (real_T *)&rtP.inv_C_Gain[0],
        (real_T *)&rtb_TmpHiddenBuffer_Feeding_inv[0], &dims[0]);
}

/* Gain: '<Root>/k1'
*
* Regarding '<Root>/k1':
*   Gain value: [1.34 -2834]
*/
{
    static const int dims[3] = { 1, 2, 1 };
    rt_MatMultRR_Dbl((real_T *)&rtb_temp12, (real_T *)&rtP.k1_Gain[0],
        (real_T *)&rtb_inv_C[0], &dims[0]);
}

/* Sum: '<Root>/Sum1' */
rtB.Sum1 = rtB.K2 + rtb_temp12;

/* DiscreteTransferFcn: '<Root>/TF Armadura' */
rtB.TF_Armadura = rtP.TF_Armadura_C*rtDWork.TF_Armadura_DSTATE;

/* Sin: '<Root>/Sine Wave1' */

```

```

rtb_temp12 = rtP.Sine_Wave1_Amp *
    sin(rtP.Sine_Wave1_Freq * rtmGetT(rtM_hil2) + rtP.Sine_Wave1_Phase) +
    rtP.Sine_Wave1_Bias;
}

/* Update for root system: '<Root>' */
void MdlUpdate(int_T tid)
{

/* tid is required for a uniform function interface. This system
 * is single rate, and in this case, tid is not accessed. */
UNUSED_PARAMETER(tid);

/* DiscreteTransferFcn Block: <Root>/TF Motor */
{
    real_T xtmp = rtB.TF_Armadura;
    xtmp += (rtP.TF_Motor_A[0])*rtDWork.TF_Motor_DSTATE[0] +
        (rtP.TF_Motor_A[1])*rtDWork.TF_Motor_DSTATE[1];
    rtDWork.TF_Motor_DSTATE[1] = rtDWork.TF_Motor_DSTATE[0];
    rtDWork.TF_Motor_DSTATE[0] = xtmp;
}

/* DiscreteTransferFcn Block: <Root>/Discrete Transfer Fcn3 */
{
    real_T xtmp = rtB.Sum1;
    xtmp +=
        (rtP.Discrete_Transfer_Fcn3_A[0])*rtDWork.Discrete_Transfer_Fcn3_DSTATE[0]
        +
        (rtP.Discrete_Transfer_Fcn3_A[1])*rtDWork.Discrete_Transfer_Fcn3_DSTATE[1]
        +
        (rtP.Discrete_Transfer_Fcn3_A[2])*rtDWork.Discrete_Transfer_Fcn3_DSTATE[2];
    rtDWork.Discrete_Transfer_Fcn3_DSTATE[2] =
        rtDWork.Discrete_Transfer_Fcn3_DSTATE[1];
    rtDWork.Discrete_Transfer_Fcn3_DSTATE[1] =
        rtDWork.Discrete_Transfer_Fcn3_DSTATE[0];
    rtDWork.Discrete_Transfer_Fcn3_DSTATE[0] = xtmp;
}

/* Derivative Block: <Root>/Derivative */
{
    real_T timeStampA = rtDWork.Derivative_RWORK.TimeStampA;
    real_T timeStampB = rtDWork.Derivative_RWORK.TimeStampB;
    real_T *lastBank = &rtDWork.Derivative_RWORK.TimeStampA;

    if (timeStampA != rtInf) {
        if (timeStampB == rtInf) {
            lastBank += 2;
        } else if (timeStampA >= timeStampB) {
            lastBank += 2;
        }
    }
}

```

```

    }
}
*lastBank++ = rtmGetT(rtM_hil2);
*lastBank++ = rtB.TF_Motor;
}

/* DiscreteIntegrator Block: <Root>/Integrador Discreto */
rtDWork.Integrador_Discreto_DSTATE = rtDWork.Integrador_Discreto_DSTATE +
0.004 * rtB.Sum;

/* DiscreteTransferFcn Block: <Root>/TF Armadura */
{
    rtDWork.TF_Armadura_DSTATE = rtB.Discrete_Transfer_Fcn3 +
    rtP.TF_Armadura_A*rtDWork.TF_Armadura_DSTATE;
}
}

/* Terminate for root system: '<Root>' */
void MdlTerminate(void)
{
    if(rtM_hil2 != NULL) {

        /* Level2 S-Function Block: <Root>/S-Function Driver (controle_moderno) */
        {
            SimStruct *rts = rtM_hil2->childSfunctions[0];
            sfcnTerminate(rts);
        }
    }
}

/* Helper function to make function calls from non-inlined S-functions */
int_T rt_CallSys(SimStruct *S, int_T element, int_T tid)
{
    (*(S)->callSys.fcns[element])((S)->callSys.args1[element],
    (S)->callSys.args2[element], tid);

    if (ssGetErrorStatus(S) != NULL) {
        return 0;
    } else {
        return 1;
    }
}

/* Function to initialize sizes */
void MdlInitializeSizes(void)
{
    rtM_hil2->Sizes.numContStates = (0); /* Number of continuous states */
    rtM_hil2->Sizes.numY = (0);          /* Number of model outputs */
    rtM_hil2->Sizes.numU = (0);          /* Number of model inputs */
    rtM_hil2->Sizes.sysDirFeedThru = (0); /* The model is not direct feedthrough */
}

```

```

rtM_hil2->Sizes.numSampTimes = (2); /* Number of sample times */
rtM_hil2->Sizes.numBlocks = (21); /* Number of blocks */
rtM_hil2->Sizes.numBlockIO = (9); /* Number of block outputs */
rtM_hil2->Sizes.numBlockPrms = (31); /* Sum of parameter "widths" */
}

/* Function to initialize sample times */
void MdlInitializeSampleTimes(void)
{
    /* task periods */
    rtM_hil2->Timing.sampleTimes[0] = (0.0);
    rtM_hil2->Timing.sampleTimes[1] = (0.004);

    /* task offsets */
    rtM_hil2->Timing.offsetTimes[0] = (0.0);
    rtM_hil2->Timing.offsetTimes[1] = (0.0);
}

/* Function to register the model */
rtModel_hil2 *hil2(void)
{
    (void)memset((char *)rtM_hil2, 0, sizeof(rtModel_hil2));

    {
        /* Setup solver object */
        static RTWSolverInfo rt_SolverInfo;
        rtM_hil2->solverInfo = (&rt_SolverInfo);

        rtsiSetSimTimeStepPtr(rtM_hil2->solverInfo, &rtM_hil2->Timing.simTimeStep);
        rtsiSetTPtr(rtM_hil2->solverInfo, &rtmGetTPtr(rtM_hil2));
        rtsiSetStepSizePtr(rtM_hil2->solverInfo, &rtM_hil2->Timing.stepSize);
        rtsiSetdXPtr(rtM_hil2->solverInfo, &rtM_hil2->ModelData.derivs);
        rtsiSetContStatesPtr(rtM_hil2->solverInfo, &rtM_hil2->ModelData.contStates);
        rtsiSetNumContStatesPtr(rtM_hil2->solverInfo,
                                &rtM_hil2->Sizes.numContStates);
        rtsiSetErrorStatusPtr(rtM_hil2->solverInfo, &rtmGetErrorStatus(rtM_hil2));

        rtsiSetRTModelPtr(rtM_hil2->solverInfo, rtM_hil2);
    }

    /* timing info */
    {
        static time_T mdlPeriod[NSAMPLE_TIMES];
        static time_T mdlOffset[NSAMPLE_TIMES];
        static time_T mdlTaskTimes[NSAMPLE_TIMES];
        static int_T mdlTsMap[NSAMPLE_TIMES];
        static int_T mdlSampleHits[NSAMPLE_TIMES];

        {
            int_T i;

```

```

    for(i = 0; i < NSAMPLE_TIMES; i++) {
        mdlPeriod[i] = 0.0;
        mdlOffset[i] = 0.0;
        mdlTaskTimes[i] = 0.0;
    }
}
(void)memset((char_T *)&mdlTsMap[0], 0, 2 * sizeof(int_T));
(void)memset((char_T *)&mdlSampleHits[0], 0, 2 * sizeof(int_T));

rtM_hil2->Timing.sampleTimes = (&mdlPeriod[0]);
rtM_hil2->Timing.offsetTimes = (&mdlOffset[0]);
rtM_hil2->Timing.sampleTimeTaskIDPtr = (&mdlTsMap[0]);
rtMSetTPtr(rtM_hil2, &mdlTaskTimes[0]);
rtM_hil2->Timing.sampleHits = (&mdlSampleHits[0]);
}
rtsiSetSolverMode(rtM_hil2->solverInfo, SOLVER_MODE_SINGLETASKING);

/*
 * initialize model vectors and cache them in SimStruct
 */

/* block I/O */
{
    void *b = (void *) &rtB;
    rtM_hil2->ModelData.blockIO = (b);

    {
        int_T i;

        b = &rtB.Saturation;
        for (i = 0; i < 10; i++) {
            ((real_T*)b)[i] = 0.0;
        }
    }
}

/* parameters */
rtM_hil2->ModelData.defaultParam = ((real_T *) &rtP);

/* data type work */
{
    void *dwork = (void *) &rtDWork;
    rtM_hil2->Work.dwork = (dwork);
    (void)memset((char_T *) dwork, 0, sizeof(D_Work));
    {
        int_T i;
        real_T *dwork_ptr = (real_T *) &rtDWork.TF_Motor_DSTATE[0];

        for (i = 0; i < 11; i++) {

```

```

        dwork_ptr[i] = 0.0;
    }
}

/* data type transition information (for external mode) */
{
    static DataTypeTransInfo dtInfo;

    (void)memset((char_T *) &dtInfo, 0, sizeof(dtInfo));
    rtM_hil2->SpecialInfo.mappingInfo = (&dtInfo);

    dtInfo.numDataTypes = 13;
    dtInfo.dataTypeSizes = &rtDataTypeSizes[0];
    dtInfo.dataTypeNames = &rtDataTypeNames[0];

    /* Block I/O transition table */
    dtInfo.B = &rtBTransTable;

    /* Parameters transition table */
    dtInfo.P = &rtPTransTable;
}

/* Model specific registration */

rtM_hil2->modelName = ("hil2");
rtM_hil2->path = ("hil2");

rtmSetTStart(rtM_hil2, 0.0);
rtM_hil2->Timing.tFinal = (4.0);
rtM_hil2->Timing.stepSize = (0.004);
rtsiSetFixedStepSize(rtM_hil2->solverInfo, 0.004);

rtM_hil2->Sizes.checksums[0] = (1502964216U);
rtM_hil2->Sizes.checksums[1] = (1827190871U);
rtM_hil2->Sizes.checksums[2] = (510327778U);
rtM_hil2->Sizes.checksums[3] = (1102673188U);

{
    static const EnableStates rtAlwaysEnabled = SUBSYS_ENABLED;

    static RTWExtModeInfo rt_ExtModeInfo;
    static const void *sysModes[1];

    rtM_hil2->extModeInfo = (&rt_ExtModeInfo);
    rteiSetSubSystemModeVectorAddresses(&rt_ExtModeInfo, sysModes);

    sysModes[0] = &rtAlwaysEnabled;

    rteiSetModelMappingInfoPtr(&rt_ExtModeInfo,

```

```

    &rtM_hil2->SpecialInfo.mappingInfo);

    rteiSetChecksumsPtr(&rt_ExtModeInfo, rtM_hil2->Sizes.checksums);

    rteiSetTPtr(&rt_ExtModeInfo, rtmGetTPtr(rtM_hil2));
}

/* child S-Function registration */
{
    static RTWSFcnInfo _sfcnInfo;
    RTWSFcnInfo *sfcnInfo = &_sfcnInfo;

    rtM_hil2->sfcnInfo = (sfcnInfo);

    rtssSetErrorStatusPtr(sfcnInfo, &rtmGetErrorStatus(rtM_hil2));
    rtssSetNumRootSampTimesPtr(sfcnInfo, &rtM_hil2->Sizes.numSampTimes);
    rtssSetTPtrPtr(sfcnInfo, &rtmGetTPtr(rtM_hil2));
    rtssSetTStartPtr(sfcnInfo, &rtmGetTStart(rtM_hil2));
    rtssSetTFinalPtr(sfcnInfo, &rtM_hil2->Timing.tFinal);
    rtssSetTimeOfLastOutputPtr(sfcnInfo, &rtM_hil2->Timing.timeOfLastOutput);
    rtssSetStepSizePtr(sfcnInfo, &rtM_hil2->Timing.stepSize);
    rtssSetStopRequestedPtr(sfcnInfo, &rtM_hil2->Timing.stopRequestedFlag);
    rtssSetDerivCacheNeedsResetPtr(sfcnInfo,
        &rtM_hil2->ModelData.derivCacheNeedsReset);
    rtssSetZCCacheNeedsResetPtr(sfcnInfo,
        &rtM_hil2->ModelData.zCCacheNeedsReset);
    rtssSetBlkStateChangePtr(sfcnInfo, &rtM_hil2->ModelData.blkStateChange);
    rtssSetSampleHitsPtr(sfcnInfo, &rtM_hil2->Timing.sampleHits);
    rtssSetPerTaskSampleHitsPtr(sfcnInfo, &rtM_hil2->Timing.perTaskSampleHits);
    rtssSetSimModePtr(sfcnInfo, &rtM_hil2->simMode);
    rtssSetSolverInfoPtr(sfcnInfo, &rtM_hil2->solverInfo);
}

rtM_hil2->Sizes.numSFcns = (1);

/* register each child */
{
    static SimStruct childSFunctions[1];
    static SimStruct *childSFunctionPtrs[1];

    (void)memset((char_T *)&childSFunctions[0], 0, sizeof(childSFunctions));
    rtM_hil2->childSfunctions = (&childSFunctionPtrs[0]);
    {
        int_T i;

        for(i = 0; i < 1; i++) {
            rtM_hil2->childSfunctions[i] = (&childSFunctions[i]);
        }
    }
}

```

```

/* Level2 S-Function Block: hil2/<Root>/S-Function Driver (controle_moderno) */
{
    SimStruct *rts = rtM_hil2->childSfunctions[0];
    /* timing info */
    static time_T sfcnPeriod[1];
    static time_T sfcnOffset[1];
    static int_T sfcnTsMap[1];

    {
        int_T i;

        for(i = 0; i < 1; i++) {
            sfcnPeriod[i] = sfcnOffset[i] = 0.0;
        }
    }
    ssSetSampleTimePtr(rts, &sfcnPeriod[0]);
    ssSetOffsetTimePtr(rts, &sfcnOffset[0]);
    ssSetSampleTimeTaskIDPtr(rts, sfcnTsMap);

    /* Set up the mdlInfo pointer */
    {
        static struct _ssBlkInfo2 _blkInfo2;
        struct _ssBlkInfo2 *blkInfo2 = &_blkInfo2;
        ssSetBlkInfo2Ptr(rts, blkInfo2);
        ssSetRTWSfcnInfo(rts, rtM_hil2->sfcnInfo);
    }

    /* Allocate memory of model methods 2 */
    {
        static struct _ssSFcnModelMethods2 methods2;
        ssSetModelMethods2(rts, &methods2);
    }

    /* inputs */
    {
        static struct _ssPortInputs inputPortInfo[1];

        _ssSetNumInputPorts(rts, 1);
        ssSetPortInfoForInputs(rts, &inputPortInfo[0]);

        /* port 0 */
        {
            static real_T const *sfcnUPtrs[2];
            sfcnUPtrs[0] = &rtB.Saturation;
            sfcnUPtrs[1] = &rtP.on_off_Value;
            ssSetInputPortSignalPtrs(rts, 0, (InputPtrsType)&sfcnUPtrs[0]);
            _ssSetInputPortNumDimensions(rts, 0, 1);
            ssSetInputPortWidth(rts, 0, 2);
        }
    }
}

```



```

}

/* outputs */
{
    static struct _ssPortOutputs outputPortInfo[1];
    _ssSetNumOutputPorts(rts, 1);
    ssSetPortInfoForOutputs(rts, &outputPortInfo[0]);
    /* port 0 */
    {
        _ssSetOutputPortNumDimensions(rts, 0, 1);
        ssSetOutputPortWidth(rts, 0, 2);
        ssSetOutputPortSignal(rts, 0, ((real_T *) &rtB.S_Function_Driver[0]));
    }
}
/* path info */
ssSetModelName(rts, "S-Function\nDriver");
ssSetPath(rts, "hil2/S-Function Driver");
ssSetRTModel(rts, rtM_hil2);
ssSetParentSS(rts, NULL);
ssSetRootSS(rts, rts);
ssSetVersion(rts, SIMSTRUCT_VERSION_LEVEL2);

/* registration */
controle_moderno(rts);

sfcnInitializeSizes(rts);
sfcnInitializeSampleTimes(rts);

/* adjust sample time */
ssSetSampleTime(rts, 0, 0.004);
ssSetOffsetTime(rts, 0, 0.0);
sfcnTsMap[0] = 1;

/* set compiled values of dynamic vector attributes */

ssSetNumNonsampledZCs(rts, 0);
/* Update connectivity flags for each port */
_ssSetInputPortConnected(rts, 0, 1);
_ssSetOutputPortConnected(rts, 0, 1);
_ssSetOutputPortBeingMerged(rts, 0, 0);
/* Update the BufferDstPort flags for each input port */
ssSetInputPortBufferDstPort(rts, 0, -1);
}
}

return rtM_hil2;
}

```

Hardware-in-the-Loop I I

```

* Real-Time Workshop code generation for Simulink model "hil2.mdl".
*
* Model Version          : 1.28
* Real-Time Workshop file version : 5.0 $Date: 2002/05/30 19:21:33 $
* Real-Time Workshop file generated on : Wed Nov 02 15:38:29 2005
* TLC version           : 5.0 (Jun 18 2002)
* C source code generated on      : Wed Nov 02 15:38:30 2005
*/

#include <math.h>
#include <string.h>
#include "hil2.h"
#include "hil2_private.h"
#include "ext_work.h"

#include "hil2_dt.h"

/* Block signals (auto storage) */
BlockIO rtB;

/* Block states (auto storage) */
D_Work rtDWork;

/* Parent Simstruct */
static rtModel_hil2 model_S;
rtModel_hil2 *const rtM_hil2 = &model_S;

/* Initial conditions for root system: '<Root>' */
void MdlInitialize(void)
{
    /* Derivative Block: <Root>/Derivative */
    rtDWork.Derivative_RWORK.TimeStampA = rtInf;
    rtDWork.Derivative_RWORK.TimeStampB = rtInf;

    /* DiscreteIntegrator Block: <Root>/Integrador Discreto */
    rtDWork.Integrador_Discreto_DSTATE = rtP.Integrador_Discreto_IC;
}

/* Start for root system: '<Root>' */
void MdlStart(void)
{
    /* DiscretePulseGenerator Block: <Root>/Pulse Generator */
    {
        int_T Ns;
        real_T tFirst = rtmGetTStart(rtM_hil2);
        Ns = (int_T)floor(tFirst / 0.004 + 0.5);
    }
}

```

```

if (Ns <= 0) {
    rtDWork.Pulse_Generator_IWORK.ClockTicksCounter = Ns;
} else {
    rtDWork.Pulse_Generator_IWORK.ClockTicksCounter = Ns -
        (int_T)(rtP.Pulse_Generator_Period*floor((real_T)Ns /
            rtP.Pulse_Generator_Period));
}
}
MdlInitialize();
}
/* Outputs for root system: '<Root>' */
void MdlOutputs(int_T tid)
{
    /* local block i/o variables */
    real_T rtb_TmpHiddenBuffer_Feeding_inv[2];
    real_T rtb_inv_C[2];
    real_T rtb_temp11;
    real_T rtb_temp12;

    /* tid is required for a uniform function interface. This system
     * is single rate, and in this case, tid is not accessed. */
    UNUSED_PARAMETER(tid);

    /* DiscretePulseGenerator: '<Root>/Pulse Generator' */
    rtb_temp11 =
        (rtDWork.Pulse_Generator_IWORK.ClockTicksCounter < rtP.Pulse_Generator_Duty
            &&
            rtDWork.Pulse_Generator_IWORK.ClockTicksCounter >= 0) ?
        rtP.Pulse_Generator_Amp :
        0.0;
    if (rtDWork.Pulse_Generator_IWORK.ClockTicksCounter >=
        rtP.Pulse_Generator_Period-1) {
        rtDWork.Pulse_Generator_IWORK.ClockTicksCounter = 0;
    } else {
        (rtDWork.Pulse_Generator_IWORK.ClockTicksCounter)++;
    }
    /* Saturate: '<Root>/Saturation'
     *
     * Regarding '<Root>/Saturation':
     *   Lower limit: rtP.Saturation_LowerSat
     *   Upper limit: rtP.Saturation_UpperSat
     */
    if (rtb_temp11 >= rtP.Saturation_UpperSat) {
        rtB.Saturation = rtP.Saturation_UpperSat;
    } else if (rtb_temp11 <= rtP.Saturation_LowerSat) {
        rtB.Saturation = rtP.Saturation_LowerSat;
    } else {
        rtB.Saturation = rtb_temp11;
    }
}

```

```

/* DiscreteTransferFcn: '<Root>/TF Motor' */
rtB.TF_Motor = rtP.TF_Motor_C[0]*rtDWork.TF_Motor_DSTATE[0] +
  rtP.TF_Motor_C[1]*rtDWork.TF_Motor_DSTATE[1];

/* Level2 S-Function Block: <Root>/S-Function Driver (controle_moderno) */
{
  SimStruct *rts = rtM_hil2->childSfunctions[0];
  sfcnOutputs(rts, tid);
}

/* DiscreteTransferFcn: '<Root>/Discrete Transfer Fcn3' */
rtB.Discrete_Transfer_Fcn3 =
  rtP.Discrete_Transfer_Fcn3_C[0]*rtDWork.Discrete_Transfer_Fcn3_DSTATE[0]
  + rtP.Discrete_Transfer_Fcn3_C[1]*rtDWork.Discrete_Transfer_Fcn3_DSTATE[1]
  + rtP.Discrete_Transfer_Fcn3_C[2]*rtDWork.Discrete_Transfer_Fcn3_DSTATE[2];

/* Gain: '<Root>/K1'
*
* Regarding '<Root>/K1':
*   Gain value: rtP.K1_Gain
*/
rtB.K1 = rtB.S_Function_Driver[1] * rtP.K1_Gain;

/* Derivative Block: <Root>/Derivative */
{
  real_T t = rtmGetT(rtM_hil2);
  real_T timeStampA = rtDWork.Derivative_RWORK.TimeStampA;
  real_T timeStampB = rtDWork.Derivative_RWORK.TimeStampB;

  if (timeStampA >= t && timeStampB >= t) {
    rtb_temp12 = 0.0;
  } else {
    real_T deltaT;
    real_T *lastBank = &rtDWork.Derivative_RWORK.TimeStampA;
    if (timeStampA < timeStampB) {
      if (timeStampB < t) {
        lastBank += 2;
      }
    } else if (timeStampA >= t) {
      lastBank += 2;
    }
    deltaT = t - *lastBank++;
    rtb_temp12 = (rtB.TF_Motor - *lastBank++) / deltaT;
  }
}

/* DiscreteIntegrator: '<Root>/Integrador Discreto'
*
* Regarding '<Root>/Integrador Discreto':
*   Unlimited, w/o Saturation Port

```

```

*/
rtb_temp11 = rtDWork.Integrador_Discreto_DSTATE;

/* Gain: '<Root>/K2'
*
* Regarding '<Root>/K2':
*   Gain value: rtP.K2_Gain
*/
rtB.K2 = rtb_temp11 * rtP.K2_Gain;

/* Sum: '<Root>/Sum' */
rtB.Sum = - rtB.TF_Motor + rtB.Saturation;

/* HiddenBuffer */
rtb_TmpHiddenBuffer_Feeding_inv[0] = rtB.TF_Motor;
rtb_TmpHiddenBuffer_Feeding_inv[1] = rtb_temp12;

/* Gain: '<Root>/inv(C)'
*
* Regarding '<Root>/inv(C)':
*   Gain value: [ 0  0.0003;  0.0003  0]
*/
{
    static const int dims[3] = { 2, 2, 1 };
    rt_MatMultRR_Dbl((real_T *)&rtb_inv_C[0], (real_T *)&rtP.inv_C_Gain[0],
        (real_T *)&rtb_TmpHiddenBuffer_Feeding_inv[0], &dims[0]);
}

/* Gain: '<Root>/k1'
*
* Regarding '<Root>/k1':
*   Gain value: [1.34 -2834]
*/
{
    static const int dims[3] = { 1, 2, 1 };
    rt_MatMultRR_Dbl((real_T *)&rtb_temp12, (real_T *)&rtP.k1_Gain[0],
        (real_T *)&rtb_inv_C[0], &dims[0]);
}

/* Sum: '<Root>/Sum1' */
rtB.Sum1 = rtB.K2 + rtb_temp12;

/* DiscreteTransferFcn: '<Root>/TF Armadura' */
rtB.TF_Armadura = rtP.TF_Armadura_C*rtDWork.TF_Armadura_DSTATE;

/* Sin: '<Root>/Sine Wave1' */

rtb_temp12 = rtP.Sine_Wave1_Amp *
    sin(rtP.Sine_Wave1_Freq * rtmGetT(rtm_hil2) + rtP.Sine_Wave1_Phase) +
    rtP.Sine_Wave1_Bias;

```

```

}

/* Update for root system: '<Root>' */
void MdlUpdate(int_T tid)
{
    /* tid is required for a uniform function interface. This system
     * is single rate, and in this case, tid is not accessed. */
    UNUSED_PARAMETER(tid);

    /* DiscreteTransferFcn Block: <Root>/TF Motor */
    {
        real_T xtmp = rtB.TF_Armadura;
        xtmp += (rtP.TF_Motor_A[0])*rtDWork.TF_Motor_DSTATE[0] +
            (rtP.TF_Motor_A[1])*rtDWork.TF_Motor_DSTATE[1];
        rtDWork.TF_Motor_DSTATE[1] = rtDWork.TF_Motor_DSTATE[0];
        rtDWork.TF_Motor_DSTATE[0] = xtmp;
    }

    /* DiscreteTransferFcn Block: <Root>/Discrete Transfer Fcn3 */
    {
        real_T xtmp = rtB.Sum1;
        xtmp +=
            (rtP.Discrete_Transfer_Fcn3_A[0])*rtDWork.Discrete_Transfer_Fcn3_DSTATE[0]
            +
            (rtP.Discrete_Transfer_Fcn3_A[1])*rtDWork.Discrete_Transfer_Fcn3_DSTATE[1]
            +
            (rtP.Discrete_Transfer_Fcn3_A[2])*rtDWork.Discrete_Transfer_Fcn3_DSTATE[2];
        rtDWork.Discrete_Transfer_Fcn3_DSTATE[2] =
            rtDWork.Discrete_Transfer_Fcn3_DSTATE[1];
        rtDWork.Discrete_Transfer_Fcn3_DSTATE[1] =
            rtDWork.Discrete_Transfer_Fcn3_DSTATE[0];
        rtDWork.Discrete_Transfer_Fcn3_DSTATE[0] = xtmp;
    }

    /* Derivative Block: <Root>/Derivative */
    {
        real_T timeStampA = rtDWork.Derivative_RWORK.TimeStampA;
        real_T timeStampB = rtDWork.Derivative_RWORK.TimeStampB;
        real_T *lastBank = &rtDWork.Derivative_RWORK.TimeStampA;

        if (timeStampA != rtInf) {
            if (timeStampB == rtInf) {
                lastBank += 2;
            } else if (timeStampA >= timeStampB) {
                lastBank += 2;
            }
        }
        *lastBank++ = rtmGetT(rtM_hil2);
        *lastBank++ = rtB.TF_Motor;
    }
}

```

```

}

/* DiscreteIntegrator Block: <Root>/Integrador Discreto */
rtDWork.Integrador_Discreto_DSTATE = rtDWork.Integrador_Discreto_DSTATE +
    0.004 * rtB.Sum;

/* DiscreteTransferFcn Block: <Root>/TF Armadura */
{
    rtDWork.TF_Armadura_DSTATE = rtB.Discrete_Transfer_Fcn3 +
        rtP.TF_Armadura_A*rtDWork.TF_Armadura_DSTATE;
}
}

/* Terminate for root system: '<Root>' */
void MdlTerminate(void)
{
    if(rtM_hil2 != NULL) {

        /* Level2 S-Function Block: <Root>/S-Function Driver (controle_moderno) */
        {
            SimStruct *rts = rtM_hil2->childSfunctions[0];
            sfcnTerminate(rts);
        }
    }
}

/* Helper function to make function calls from non-inlined S-functions */
int_T rt_CallSys(SimStruct *S, int_T element, int_T tid)
{
    (*(S)->callSys.fcns[element])((S)->callSys.args1[element],
        (S)->callSys.args2[element], tid);

    if (ssGetErrorStatus(S) != NULL) {
        return 0;
    } else {
        return 1;
    }
}

/* Function to initialize sizes */
void MdlInitializeSizes(void)
{
    rtM_hil2->Sizes.numContStates = (0); /* Number of continuous states */
    rtM_hil2->Sizes.numY = (0);          /* Number of model outputs */
    rtM_hil2->Sizes.numU = (0);          /* Number of model inputs */
    rtM_hil2->Sizes.sysDirFeedThru = (0); /* The model is not direct feedthrough */
    rtM_hil2->Sizes.numSampTimes = (2);  /* Number of sample times */
    rtM_hil2->Sizes.numBlocks = (21);    /* Number of blocks */
    rtM_hil2->Sizes.numBlockIO = (9);     /* Number of block outputs */
    rtM_hil2->Sizes.numBlockPrms = (31); /* Sum of parameter "widths" */
}

```

```

}

/* Function to initialize sample times */
void MdlInitializeSampleTimes(void)
{
    /* task periods */
    rtM_hil2->Timing.sampleTimes[0] = (0.0);
    rtM_hil2->Timing.sampleTimes[1] = (0.004);

    /* task offsets */
    rtM_hil2->Timing.offsetTimes[0] = (0.0);
    rtM_hil2->Timing.offsetTimes[1] = (0.0);
}

/* Function to register the model */
rtModel_hil2 *hil2(void)
{
    (void)memset((char *)rtM_hil2, 0, sizeof(rtModel_hil2));

    {
        /* Setup solver object */
        static RTWSolverInfo rt_SolverInfo;
        rtM_hil2->solverInfo = (&rt_SolverInfo);

        rtsiSetSimTimeStepPtr(rtM_hil2->solverInfo, &rtM_hil2->Timing.simTimeStep);
        rtsiSetTPtr(rtM_hil2->solverInfo, &rtmGetTPtr(rtM_hil2));
        rtsiSetStepSizePtr(rtM_hil2->solverInfo, &rtM_hil2->Timing.stepSize);
        rtsiSetdXPtr(rtM_hil2->solverInfo, &rtM_hil2->ModelData.derivs);
        rtsiSetContStatesPtr(rtM_hil2->solverInfo, &rtM_hil2->ModelData.contStates);
        rtsiSetNumContStatesPtr(rtM_hil2->solverInfo,
            &rtM_hil2->Sizes.numContStates);
        rtsiSetErrorStatusPtr(rtM_hil2->solverInfo, &rtmGetErrorStatus(rtM_hil2));

        rtsiSetRTModelPtr(rtM_hil2->solverInfo, rtM_hil2);
    }

    /* timing info */
    {
        static time_T mdlPeriod[NSAMPLE_TIMES];
        static time_T mdlOffset[NSAMPLE_TIMES];
        static time_T mdlTaskTimes[NSAMPLE_TIMES];
        static int_T mdlTsMap[NSAMPLE_TIMES];
        static int_T mdlSampleHits[NSAMPLE_TIMES];

        {
            int_T i;

            for(i = 0; i < NSAMPLE_TIMES; i++) {
                mdlPeriod[i] = 0.0;
                mdlOffset[i] = 0.0;
            }
        }
    }
}

```



```

        mdlTaskTimes[i] = 0.0;
    }
}
(void)memset((char_T *)&mdlTsMap[0], 0, 2 * sizeof(int_T));
(void)memset((char_T *)&mdlSampleHits[0], 0, 2 * sizeof(int_T));

rtM_hil2->Timing.sampleTimes = (&mdlPeriod[0]);
rtM_hil2->Timing.offsetTimes = (&mdlOffset[0]);
rtM_hil2->Timing.sampleTimeTaskIDPtr = (&mdlTsMap[0]);
rtmSetTPtr(rtM_hil2, &mdlTaskTimes[0]);
rtM_hil2->Timing.sampleHits = (&mdlSampleHits[0]);
}
rtsiSetSolverMode(rtM_hil2->solverInfo, SOLVER_MODE_SINGLETASKING);

/*
 * initialize model vectors and cache them in SimStruct
 */

/* block I/O */
{
    void *b = (void *) &rtB;
    rtM_hil2->ModelData.blockIO = (b);

    {
        int_T i;

        b = &rtB.Saturation;
        for (i = 0; i < 10; i++) {
            ((real_T*)b)[i] = 0.0;
        }
    }
}

/* parameters */
rtM_hil2->ModelData.defaultParam = ((real_T *) &rtP);

/* data type work */
{
    void *dwork = (void *) &rtDWork;
    rtM_hil2->Work.dwork = (dwork);
    (void)memset((char_T *) dwork, 0, sizeof(D_Work));
    {
        int_T i;
        real_T *dwork_ptr = (real_T *) &rtDWork.TF_Motor_DSTATE[0];

        for (i = 0; i < 11; i++) {
            dwork_ptr[i] = 0.0;
        }
    }
}

```

```

/* data type transition information (for external mode) */
{
    static DataTypeTransInfo dtInfo;

    (void)memset((char_T *) &dtInfo, 0, sizeof(dtInfo));
    rtM_hil2->SpecialInfo.mappingInfo = (&dtInfo);

    dtInfo.numDataTypes = 13;
    dtInfo.dataTypeSizes = &rtDataTypeSizes[0];
    dtInfo.dataTypeNames = &rtDataTypeNames[0];

    /* Block I/O transition table */
    dtInfo.B = &rtBTransTable;

    /* Parameters transition table */
    dtInfo.P = &rtPTransTable;
}

/* Model specific registration */

rtM_hil2->modelName = ("hil2");
rtM_hil2->path = ("hil2");

rtmSetTStart(rtM_hil2, 0.0);
rtM_hil2->Timing.tFinal = (4.0);
rtM_hil2->Timing.stepSize = (0.004);
rtsiSetFixedStepSize(rtM_hil2->solverInfo, 0.004);

rtM_hil2->Sizes.checksums[0] = (1502964216U);
rtM_hil2->Sizes.checksums[1] = (1827190871U);
rtM_hil2->Sizes.checksums[2] = (510327778U);
rtM_hil2->Sizes.checksums[3] = (1102673188U);

{
    static const EnableStates rtAlwaysEnabled = SUBSYS_ENABLED;

    static RTWExtModeInfo rt_ExtModeInfo;
    static const void *sysModes[1];

    rtM_hil2->extModeInfo = (&rt_ExtModeInfo);
    rteiSetSubSystemModeVectorAddresses(&rt_ExtModeInfo, sysModes);

    sysModes[0] = &rtAlwaysEnabled;

    rteiSetModelMappingInfoPtr(&rt_ExtModeInfo,
        &rtM_hil2->SpecialInfo.mappingInfo);

    rteiSetChecksumsPtr(&rt_ExtModeInfo, rtM_hil2->Sizes.checksums);
}

```

```

    rteiSetTPtr(&rt_ExtModeInfo, rtmGetTPtr(rtM_hil2));
}

/* child S-Function registration */
{
    static RTWSfcnInfo _sfcnInfo;
    RTWSfcnInfo *sfcnInfo = &_sfcnInfo;

    rtM_hil2->sfcnInfo = (sfcnInfo);

    rtssSetErrorStatusPtr(sfcnInfo, &rtmGetErrorStatus(rtM_hil2));
    rtssSetNumRootSampTimesPtr(sfcnInfo, &rtM_hil2->Sizes.numSampTimes);
    rtssSetTPtrPtr(sfcnInfo, &rtmGetTPtr(rtM_hil2));
    rtssSetTStartPtr(sfcnInfo, &rtmGetTStart(rtM_hil2));
    rtssSetTFinalPtr(sfcnInfo, &rtM_hil2->Timing.tFinal);
    rtssSetTimeOfLastOutputPtr(sfcnInfo, &rtM_hil2->Timing.timeOfLastOutput);
    rtssSetStepSizePtr(sfcnInfo, &rtM_hil2->Timing.stepSize);
    rtssSetStopRequestedPtr(sfcnInfo, &rtM_hil2->Timing.stopRequestedFlag);
    rtssSetDerivCacheNeedsResetPtr(sfcnInfo,
        &rtM_hil2->ModelData.derivCacheNeedsReset);
    rtssSetZCCacheNeedsResetPtr(sfcnInfo,
        &rtM_hil2->ModelData.zCCacheNeedsReset);
    rtssSetBlkStateChangePtr(sfcnInfo, &rtM_hil2->ModelData.blkStateChange);
    rtssSetSampleHitsPtr(sfcnInfo, &rtM_hil2->Timing.sampleHits);
    rtssSetPerTaskSampleHitsPtr(sfcnInfo, &rtM_hil2->Timing.perTaskSampleHits);
    rtssSetSimModePtr(sfcnInfo, &rtM_hil2->simMode);
    rtssSetSolverInfoPtr(sfcnInfo, &rtM_hil2->solverInfo);
}

rtM_hil2->Sizes.numSFcns = (1);

/* register each child */
{
    static SimStruct childSFunctions[1];
    static SimStruct *childSFunctionPtrs[1];

    (void)memset((char_T *)&childSFunctions[0], 0, sizeof(childSFunctions));
    rtM_hil2->childSfunctions = (&childSFunctionPtrs[0]);
    {
        int_T i;

        for(i = 0; i < 1; i++) {
            rtM_hil2->childSfunctions[i] = (&childSFunctions[i]);
        }
    }
}

/* Level2 S-Function Block: hil2/<Root>/S-Function Driver (controle_moderno) */
{
    SimStruct *rts = rtM_hil2->childSfunctions[0];
    /* timing info */

```

```

static time_T sfcnPeriod[1];
static time_T sfcnOffset[1];
static int_T sfcnTsMap[1];

{
    int_T i;

    for(i = 0; i < 1; i++) {
        sfcnPeriod[i] = sfcnOffset[i] = 0.0;
    }
}
ssSetSampleTimePtr(rts, &sfcnPeriod[0]);
ssSetOffsetTimePtr(rts, &sfcnOffset[0]);
ssSetSampleTimeTaskIDPtr(rts, sfcnTsMap);

/* Set up the mdlInfo pointer */
{
    static struct _ssBlkInfo2 _blkInfo2;
    struct _ssBlkInfo2 *blkInfo2 = &_blkInfo2;
    ssSetBlkInfo2Ptr(rts, blkInfo2);
    ssSetRTWSfcnInfo(rts, rtM_hil2->sfcnInfo);
}

/* Allocate memory of model methods 2 */
{
    static struct _ssSFcnModelMethods2 methods2;
    ssSetModelMethods2(rts, &methods2);
}

/* inputs */
{
    static struct _ssPortInputs inputPortInfo[1];

    _ssSetNumInputPorts(rts, 1);
    ssSetPortInfoForInputs(rts, &inputPortInfo[0]);

    /* port 0 */
    {
        static real_T const *sfcnUPtrs[2];
        sfcnUPtrs[0] = &rtB.Saturation;
        sfcnUPtrs[1] = &rtP.on_off_Value;
        ssSetInputPortSignalPtrs(rts, 0, (InputPtrsType)&sfcnUPtrs[0]);
        _ssSetInputPortNumDimensions(rts, 0, 1);
        ssSetInputPortWidth(rts, 0, 2);
    }
}

/* outputs */
{

```

```

static struct _ssPortOutputs outputPortInfo[1];
_ssSetNumOutputPorts(rts, 1);
ssSetPortInfoForOutputs(rts, &outputPortInfo[0]);
/* port 0 */
{
    _ssSetOutputPortNumDimensions(rts, 0, 1);
    ssSetOutputPortWidth(rts, 0, 2);
    ssSetOutputPortSignal(rts, 0, ((real_T *) &rtB.S_Function_Driver[0]));
}
}
/* path info */
ssSetModelName(rts, "S-Function\nDriver");
ssSetPath(rts, "hil2/S-Function Driver");
ssSetRTModel(rts, rtM_hil2);
ssSetParentSS(rts, NULL);
ssSetRootSS(rts, rts);
ssSetVersion(rts, SIMSTRUCT_VERSION_LEVEL2);

/* registration */
controle_moderno(rts);

sfcnInitializeSizes(rts);
sfcnInitializeSampleTimes(rts);

/* adjust sample time */
ssSetSampleTime(rts, 0, 0.004);
ssSetOffsetTime(rts, 0, 0.0);
sfcnTsMap[0] = 1;

/* set compiled values of dynamic vector attributes */

ssSetNumNonsampledZCs(rts, 0);
/* Update connectivity flags for each port */
_ssSetInputPortConnected(rts, 0, 1);
_ssSetOutputPortConnected(rts, 0, 1);
_ssSetOutputPortBeingMerged(rts, 0, 0);
/* Update the BufferDstPort flags for each input port */
ssSetInputPortBufferDstPort(rts, 0, -1);
}
}

return rtM_hil2;
}

```