

ESTE EXEMPLAR CORRESPONDE A REDAÇÃO FINAL D
TESE DEFENDIDA POR Camilo Andres
Gordillo Carrillo E APROVAD
PELA COMISSÃO JULGADORA EM 28/02/2012


.....
ORIENTADOR

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA

Camilo Andres Gordillo Carrillo

**Estratégias para a correção
dos efeitos de atraso de sistemas
Hardware In the Loop (HIL)**

Campinas, 2012.

30/2012

Camilo Andres Gordillo Carrillo

Estratégias para a correção dos efeitos de atraso de sistemas Hardware In the Loop (HIL)

Tese apresentada ao Curso de Mestrado da
Faculdade de Engenharia Mecânica da
Universidade Estadual de Campinas, como
requisito para a obtenção do título de Mestre
em Engenharia Mecânica.

Área de Concentração: **Mecânica dos Sólidos e Projeto Mecânico**

Orientador: **Prof. Dr. Janito Vaqueiro Ferreira**

Campinas
2012

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

G652e Gordillo Carrillo, Camilo Andres
Estratégias para a correção dos efeitos de atraso de
sistemas Hardware In the Loop (HIL) / Camilo Andrés
Gordillo Carrillo. --Campinas, SP: [s.n.], 2012.

Orientador: Janito Vaqueiro Ferreira.
Dissertação de Mestrado - Universidade Estadual de
Campinas, Faculdade de Engenharia Mecânica.

1. Sistemas de tempo real. 2. Modelamento
matemático. I. Ferreira, Janito Vaqueiro. II.
Universidade Estadual de Campinas. Faculdade de
Engenharia Mecânica. III. Título.

Título em Inglês: Strategies to correct the effects of delay on the Hardware In the
Loop (HIL) systems

Palavras-chave em Inglês: Real-time systems, Mathematical modeling

Área de concentração: Mecânica dos Sólidos e Projeto Mecânico

Titulação: Mestre em Engenharia Mecânica

Banca examinadora: Pablo Siqueira Meirelles, Oscar Fernando Aviles Sanchez

Data da defesa: 28-02-2012


Programa de Pós Graduação: Engenharia Mecânica

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA
DEPARTAMENTO DE MECÂNICA COMPUTACIONAL

**Estratégias para a correção
dos efeitos de atraso de sistemas
*Hardware In the Loop (HIL)***

Autor: Camilo Andres Gordillo Carrillo
Orientador: Janito Vaqueiro Ferreira.

A Banca Examinadora composta pelos membros abaixo aprovou esta Tese:


Prof. Dr. Janito Vaqueiro Ferreira, Presidente.
Universidade Estadual de Campinas - FEM/DMC


Prof. Dr. Pablo Siqueira Meirelles
Universidade Estadual de Campinas – FEM/DPM


Prof. Dr. Oscar Fernando Avilés Sanchez
Universidad Militar Nueva Granada – UMNG/COLOMBIA

Campinas, 28 de Fevereiro de 2012.

Dedicatória

Dedico este trabalho e o processo de aprendizagem a minha mãe, meu pai e minha avó.

Agradecimentos

À Deus.

Aos meus pais e minha avó pelo apoio, amor e bons desejos, ainda estando longe nunca me abandonaram.

Ao meu orientador, Prof. Dr. Janito Vaqueiro Ferreira, pelo apoio, orientação e paciência neste processo de aprendizagem como pessoa e profissional.

A todos os professores e colegas do departamento, que ajudaram de forma direta e indireta na conclusão deste trabalho, especialmente a secretaria do departamento e amiga Elizabeth Viana.

A todas as pessoas que me acompanharam nestes dois anos, com as quais compartilhei e aprendi muitas coisas.

À CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pelo apoio financeiro.

*“Não tentes ser bem sucedido,
tenta ser um homem de valor”.*
Albert Einstein

Resumo

O conceito de *Hardware In the Loop* (HIL) é bastante útil em indústrias automotivas e em indústrias espaciais, já que sistemas complexos são difíceis de se modelar. Este conceito proporciona uma grande confiabilidade aos resultados, diminui o risco de avaria dos equipamentos e dos usuários em seu funcionamento, como também uma diminuição do tempo no desenvolvimento de projetos. Tudo isto sem precisar de um orçamento elevado ou protótipos elaborados para realização de testes.

Neste trabalho propõem-se duas estratégias para solucionar o problema do atraso (*delay*) apresentado pelo sinal de resposta nos sistemas HIL em tempo real, levando-se em conta a sequência de execução real dos processos, bem como também outros aspectos como dos sistemas de aquisição e atuação (inércia, limitações de *hardware* e *software*, tempo de amostragem).

Os resultados obtidos através das estratégias propostas foram analisados e comparados com resultados numéricos em uma bancada experimental obtendo uma boa concordância eliminando o atraso na resposta.

Palavras Chave: *Hardware In the Loop* (HIL); atraso; modelagem, Tempo Real;

Abstract

The Hardware In the Loop (HIL) concept is useful in automotive and spaceship industries, because of the difficulty of modeling complex systems. This concept provides great reliability at the results, decrease the risk of damage to the equipment and to the user operation, as well as decreasing the time of projects development. All of this without requiring a high budget or developing prototypes for testing.

This study propose a strategy to solve the delay problem presented by the response signal in real time HIL systems, considering a real execution sequence of the process, as well as other aspects such as in the acquisition and the actuation systems (inertia, hardware and software limitations, sample time).

The results obtained through the proposed strategies was analyzed and compared with numerical results in a testing platform with excellent concordance eliminating the delay in the response.

Key Words: Hardware In the Loop (HIL); delay; modeling, Real Time;

Lista de Figuras

Figura 2.1 Componentes do HIL

Figura 2.2 Estrutura da revisão Bibliográfica

Figura 2.3 Execução do Tempo Real

Figura 2.4 Metodologia de desenvolvimento adotada. (BORGES , 2002)

Figura 2.5 Sinal com atraso

Figura 2.6 Predição do deslocamento para compensação do atraso (HORIUCHI, INOUE et al., 1999)

Figura 2.7 Controle SkyHook para $\frac{1}{4}$ de veículo com HIL (BATTERBEE, SIMS, 2006)

Figura 2.8 Desenvolvimento de HIL (LINS, 2007)

Figura 3.1 Modelo esquemático proposto do HIL.

Figura 3.2 Modelo Proposto a) Real b) Esquemático c) Diagrama de Corpo livre

Figura 3.3 Modelo Proposto a) Esquemático b) Diagrama de Corpo livre

Figura 3.4 Aplicação do MATLAB para o HIL (PASSOS, 2008)

Figura 3.5 Placa DSPACE DS-1102

Figura 3.6 Instrumentação Virtual ControlDesk catálogo 2008

Figura 3.7 Sistema aquisição e atuação Instron/Schenck

Figura 3.8 Montagem física

Figura 3.9 Calibração deslocamento Simulink-Instron a) Respostas b) Parâmetros software Instron

Figura 3.10 Modelo Simulink Calibração Deslocamento

Figura 3.11 Respostas em força do Simulink-Instron

Figura 3.12 Respostas em força calibrada do Simulink-Instron

Figura 3.13 Modelo Simulink Calibração

Figura 3.14. a) Características gerais na mola (RILL,2011) b) Curva ajustada da mola

Figura 3.15 Modelo Simulink da Mola

Figura 3.16 Tempo de atraso do sistema Instron a) Respostas b) Detalhamento dos sinais

Figura 4.1 Sinal de “Clock” da placa de aquisição DSPACE

Figura 4.2 Operações da placa de aquisição DSPACE

Figura 4.3 Sequência de operações em tempo real da placa de aquisição DSPACE

Figura 4.4 Modelo HIL-Real no simulink

Figura 4.5 Modelo HIL-Simulado no simulink

Figura 4.6 Modelo HIL-Simulado - Sequência 1

Figura 4.7 Modelo HIL-Simulado - Sequência 2

Figura 4.8 Modelo com sistema de atuação ideal - Sequência 1

Figura 4.9 Modelo com sistema de atuação ideal - Sequência 2

Figura 4.10 Modelo com atraso obtido na sequência 1

Figura 4.11 Modelo com atraso obtido na sequência 2

Figura 4.12 Modelo simulado com atraso

Figura 4.13 Modelo sem atraso proposta 1

Figura 4.14 Sequência dos processos proposta 2

Figura 5.1 Diagrama de blocos do Modelo1 Hil-Simulado com atraso

Figura 5.2 Resposta do Modelo1 Hil-Simulado com atraso

Figura 5.3 Diagrama de blocos do Modelo1 Hil-Simulado sem atraso

Figura 5.4 Resposta do Modelo1 Hil-Simulado sem atraso

Figura 5.5 Diagrama de blocos do Modelo2 Hil simulado com atraso

Figura 5.6 Resposta do Modelo2 Hil simulado com atraso

Figura 5.7 Diagrama de blocos do Modelo2 Hil simulado sem atraso

Figura 5.8 Resposta do Modelo2 Hil simulado sem atraso

Figura 6.1 Diagrama de blocos do Experimento1 Modelo1 HIL com atraso

Figura 6.2 Resposta do Experimento1 Modelo1 Hil com atraso

Figura 6.3 Comparação teórica e experimental das Respostas do Modelo1 HIL com atraso

Figura 6.4 Diagrama de blocos Experimento2 Modelo1 HIL-Sem atraso – Proposta1

Figura 6.5 Resposta do Experimento2-Modelo1 HIL-Sem Atraso-Proposta1

Figura 6.6 Comparação teórica e experimental das Respostas do Modelo1 HIL Sem atraso – Proposta1

Figura 6.7 Diagrama de blocos Experimento3 Modelo1 HIL-Sem atraso – Proposta 2

Figura 6.8 Resposta do Experimento3-Modelo1 HIL-Sem Atraso-Proposta2

Figura 6.9 Comparação teórica e experimental das Respostas do Modelo1 HIL sem atraso – Proposta2

Figura 6.10 Diagrama de blocos do Experimento4 Modelo2 HIL com atraso

Figura 6.11 Resposta do Experimento4 Modelo2 HIL com atraso

Figura 6.12 Comparação teórica e experimental das Respostas do Modelo2 HIL com atraso

Figura 6.13 Diagrama de blocos Experimento5 Modelo2 Hil-Sem atraso – Proposta1

Figura 6.14 Resposta do Experimento5-Modelo2 HIL-Sem Atraso-Proposta1

Figura 6.15 Comparação teórica e experimental das Respostas do Modelo2 HIL Sem atraso – Proposta1

Figura 6.16Diagrama de blocos Experimento6 Modelo2 HIL-Sem atraso – Proposta 2

Figura 6.17 Resposta do Experimento6-Modelo2 HIL-Sem Atraso-Proposta2

Figura 6.18 Comparação teórica e experimental das Respostas do Modelo2 HIL sem atraso – Proposta2

Lista de Tabelas

Tabela 3.1 Variáveis utilizadas no modelo real

Tabela 3.2 Variáveis utilizadas no modelo HIL

Tabela 3.3. Coeficientes da mola

Tabela 4.1. Descrição dos processos

Lista de Abreviaturas e Siglas

Letras Latinas

m - Massa	[Kg]
k - Constante de Rigidez	[N/m]
F - Força	[N]
c - Constante de Amortecimento	[N*s/m]
F.Peso - Peso da Massa	[N]
Y - Deslocamento da Massa	[m]
\dot{Y} - Velocidade Da Massa	[m/s]
\ddot{Y} - Aceleração da Massa	[N/m ²]

.....

Abreviações e Siglas

RCP - *Rapid Control Prototype*
SIL - *Software In the Loop*
HIL - *Hardware In the Loop*
DOF - *Degree Of Freedom*
HWIL - *Hardware In the Loop*
HITL - *Hardware In The Loop*
HILS - *Hardware In the Loop Simulation*
HLS - *Hardware in the Loop Simulation*
CID - *Controller Interface Devic*
CAN - *Controller Area Network*
FPGA - *Field Programmable Gate Array*
ECU - *Electronic Control Unit*
PID - *Proporcional Integrador Derivador*

ECOV - *Error Coefficient of Variance*
DDE - *Delay Differential Equation*
AFP - *Adaptive Forward Prediction*
STR - *Sistema em Tempo Real*
A/D - *Análogo / Digital*
D/A - *Digital / Análogo*
NI - *National Instrument*
RTW - *Real Time Workshop*
RTI - *Real Time Interface*
ISA - *Industry Standard Architecture*
BNC - *Bayonet Neill – Concelman*
LVDT - *Linear Variable Differential Transformer*
ODE - *Ordinary Differential Equations*

.....

SUMÁRIO

Lista de Figuras	vi
Lista de Tabelas	ix
Lista de Abreviaturas e Siglas	x
CAPITULO 1	1
1. INTRODUÇÃO	1
1.1.Objetivo do Trabalho	2
1.2.Organização do trabalho	2
CAPITULO 2	4
2.REVISÃO BIBLIOGRÁFICA	4
2.1. <i>Hardware In the Loop</i> (HIL)	4
2.2.Atraso em <i>Hardware In the Loop</i> (HIL)	4
CAPITULO 3	15
3. HARDWARE IN THE LOOP (HIL)	15
3.1. Modelo Esquemático HIL	15
3.2. Modelagem Matemática do Sistema	16
3.2.1.Modelo real massa, mola e amortecedor 1 DOF discreto	16
3.2.2.Modelo massa, amortecedor e mola física de 1 DOF discreto	19
3.3.Ferramentas usadas no HIL	22
3.3.1.Software MATHWORKS	22
3.3.2.Placa DSPACE DS-1102	23
3.3.3.Software CONTROLDESK	24
3.3.4.Sistema INSTRON/SCHENCK	25
3.4. Montagem Bancada e Calibração	26
3.4.1.Calibração do deslocamento	27
3.4.2.Calibração da força	29
3.4.3.Calibração dos parâmetros da mola física	31
3.4.4.Resposta do Sistema Instron Schenck	34
CAPITULO 4	36

4.CORREÇÃO DO HIL PARA TEMPO REAL	36
4.1.Introdução	36
4.2.Modelo HIL - Real no Simulink	40
4.3.Modelo HIL - Simulado no Simulink	41
4.4.Operações do Modelo HIL - Simulado	42
4.5.Operações do HIL em Tempo Real (Ideal desejado)	44
4.6.Operações do HIL em Tempo Real (com atraso obtido)	48
4.7.Operação do HIL em Tempo Real (proposta 1 sem atraso)	51
4.8.Operação do HIL em Tempo Real (proposta 2 sem atraso)	53
 CAPITULO 5	 56
5.SIMULAÇÃO NUMÉRICA	56
5.1.Introdução	56
5.2.Modelo1 - HIL	56
5.2.1.Modelo1 HIL - Simulado - Com Atraso - Instável	57
5.2.2.Modelo1 HIL - Simulado - Sem Atraso - Estável	58
5.3.Modelo2 - HIL	60
5.3.1.Modelo2 HIL - Simulado - Com Atraso	60
5.3.2.Modelo2 HIL - Simulado - Sem Atraso	62
 CAPITULO 6	 64
6.RESULTADOS EXPERIMENTAIS	64
6.1.Introdução	64
6.2.Experimental - Modelo1 - HIL	65
6.2.1.Experimento1 - Modelo1 HIL - Com Atraso	66
6.2.2.Experimento2 - Modelo1 HIL - Sem Atraso-Proposta1	68
6.2.3.Experimento3 - Modelo1 HIL - Sem Atraso-Proposta2	70
6.3.Experimental – Modelo 2 - HIL	73
6.3.1.Experimento4 - Modelo2 HIL - Com Atraso	74
6.3.2.Experimento5 - Modelo2 HIL - Sem Atraso-Proposta1	76
6.3.3.Experimento6 - Modelo2 HIL - Sem Atraso-Proposta2	78
 CAPITULO 7	 82
7.CONCLUSÃO E SUGESTÕES PARA TRABALHOS FUTUROS	82
7.1.Conclusão	82

7.2.Trabalhos Futuros	83
REFERÊNCIAS	85
ANEXO A – CODIGO GERADO MODELO 1 PROPOSTA 1	
ANEXO B – CODIGO CORRIGIDO MODELO 1 PROPOSTA 1	
ANEXO C – CODIGO GERADO MODELO 1 PROPOSTA 2	
ANEXO D – CODIGO CORRIGIDO MODELO 1 PROPOSTA 2	
ANEXO E – CODIGO GERADO MODELO 2 PROPOSTA 1	
ANEXO F – CODIGO CORRIGIDO MODELO 2 PROPOSTA 1	
ANEXO G – CODIGO GERADO MODELO 2 PROPOSTA 2	
ANEXO H – CODIGO CORRIGIDO MODELO 2 PROPOSTA 2	

CAPITULO 1

INTRODUÇÃO

Na década de 80 tiveram-se grandes idéias e projetos ao redor do mundo em diferentes campos da engenharia graças à criação de novas ferramentas em aspectos de software e *hardware*, as quais tinham como objetivo a otimização de recursos. Esta otimização de recursos principalmente procurava uma diminuição no tempo de desenvolvimento e uma possibilidade de validar de modo seguro e sem ter custos maiores nos projetos desde os mais simples projetos até os projetos em grande escala. Passados os anos, na década de 90 começou-se a utilizar nas indústrias automotivas, aero espaciais, e empresas de componentes eletrônicos, técnicas para melhorar os processos tanto de produção, montagem, como de modelagem e simulação, levando a um aumento no conhecimento e no incentivo pela pesquisa com a elaboração de artigos científicos.

Nestes artigos científicos aplicaram-se tais ferramentas, elaborando conceitos, técnicas, metodologias e procedimentos em cada um dos campos das indústrias que precisavam suprir uma necessidade.

Tendo as ferramentas suficientes para se conseguir simular os processos, os modelos e suas características mais importantes, chegou-se ao ponto onde esses métodos teóricos foram aplicados em simulações através de um computador, conseguindo-se validar os modelos de um modo mais representativo em comparação com o ambiente real. Para isto criaram-se técnicas especiais como a *Rapid Control Prototype* (RCP), a *Software In the Loop* (SIL) e a *Hardware In the Loop* (HIL), tendo como característica principal nas três técnicas de serem feitas em tempo real ou de modo *Off-Line*, com o objetivo de se tentar obter uma validação entre as respostas matemáticas simuladas em um software e as respostas físicas obtidas no ambiente real.

Uma das técnicas bastante utilizadas em auxílio de projetos na engenharia é a técnica de *Hardware In the Loop* (HIL). Esta técnica vem sendo utilizada por pesquisadores em universidades e empresas em várias aplicações como se demonstra no artigo da revista Mecatrônica Atual(Lins de Albuquerque 2007).

Esta técnica apresenta vantagens como a redução do tempo de desenvolvimento bem como o aumento na qualidade, tendo a oportunidade de realizar testes sem o risco de causar um estrago no equipamento ou afetar o usuário, gerando assim uma maior confiabilidade no produto final. Como desvantagens ela possui a necessidade de recursos computacionais muito elevados, bem como um atraso (*delay*) no sinal de resposta, sendo esta última desvantagem a motivação e o objetivo neste trabalho.

Este atraso (*delay*) gerado pode ser proveniente a diferentes motivos devido às características dos diferentes componentes do sistema como sensores, atuadores, sistemas de aquisição, componentes físicos, entre outros HORIUCHI, INOUE e outros (1999), MISSELHORN (2004), LINS (2007). Por isto se propõe neste trabalho estudar estratégias para eliminar o atraso nesta técnica, para assim obter respostas iguais às obtidas em simulações, aumentando a confiabilidade na sua utilização.

1.1. Objetivo do Trabalho

Este trabalho tem como objetivo estudar e propor estratégias para eliminar o atraso que se apresenta entre as respostas de sistemas modelados matematicamente e sistemas onde se tem aplicado à técnica HIL (*Hardware In the Loop*).

1.2. Organização do trabalho

Este trabalho esta estruturado em sete capítulos descritos a seguir.

No capitulo 1, realiza-se uma breve introdução do problema apresentado pelos sistemas HIL, além da definição do objetivo deste trabalho de pesquisa e a organização deste trabalho.

No capítulo 2, mostra-se uma revisão bibliográfica dos principais conceitos do HIL onde é acompanhado do fenômeno do atraso, suas características mais importantes e trabalhos realizados nas áreas de maior uso deste conceito. Também se explica o conceito de tempo real, sendo este uma característica muito importante na estratégia proposta para este trabalho.

No capítulo 3, realiza-se a modelagem matemática de um sistema composto por uma massa, uma mola e um amortecedor de 1 grau de liberdade (DOF) no domínio discreto. Aplica-se a técnica de HIL no mesmo sistema modelado anteriormente substituindo um dos componentes matemáticos por um componente físico. Além disso, apresentam-se as ferramentas usadas para o desenvolvimento do projeto seguido das etapas de calibração.

Logo a seguir, no capítulo 4 são analisadas e apresentadas as considerações sobre a técnica de HIL modeladas com e sem o fenômeno do atraso, e expondo detalhadamente as propostas e seus procedimentos para eliminar este fenômeno.

No capítulo 5 são realizadas simulações utilizando a técnica HIL através de dois modelos e são apresentados os resultados obtidos.

O capítulo 6 mostra os resultados experimentais feitos na bancada experimental e os compara com os que foram feitos na simulação avaliando as soluções propostas.

Por fim, no capítulo 7 apresenta-se as conclusões e propostas para trabalhos futuros.

CAPITULO 2

REVISÃO BIBLIOGRÁFICA

2.1. *Hardware In the Loop (HIL)*

O conceito *Hardware In the Loop* (HIL), começou a aparecer nos textos científicos da engenharia ao final da década dos anos 80 e foi sendo ainda mais usado e conhecido na década dos anos 90, onde não era claro o seu nome e não se tinha uma definição concreta nas sociedades científicas. Algumas pessoas que trabalharam neste tema (mais comumente pessoas que tinham uma relação ou vínculo com a parte da pesquisa nas empresas ou laboratórios) escreveram artigos científicos e o chamaram de muitos modos como *Real Time Control Application* ou com variações nos acrônimos (HWIL¹-HITL²-HILS³-HLS⁴), segundo a aplicação. Nestes casos geralmente ou foram componentes desenvolvidos por empresas como a DSPACE⁵, ou aplicações feitas na FORD ou na KIA⁶, ou em laboratórios científicos espaciais como a CANADYAN SPACE AGENCY⁷.

¹ HWIL: *HardWare In the Loop*

² HITL: *Hardware In The Loop*

³ HILS: *Hardware In the Loop Simulation*

⁴ HLS: *Hardware in the Loop Simulation*

⁵ “*Hardware-in-the-Loop Simulation Testing and its Integration into a CACSD Toolset*”, por Hanselmann em 1996.

“*Automated Test of ECUs in a Hardware-in-the-Loop Simulation Environment*”, por Boot, Richert em 1999.

⁶ “*Real Time Hardware In The Loop Vehicle Simulation*”, por Alles, Swick em 1992.

“*The Versatile Hardware-in-the-Loop Laboratory: Beyond the Ad Hoc Fixture*”, por StaSko em 1998.

“*An Experimental Investigation of a CWKA System for Automobiles Using Hardware-in-the-Loop Simulations*”, por Kyongsu em 1999.

⁷ “*Space Technology Transition Using Hardware in the Loop Simulation*”, por Leitner em 1996.

“*Real-Time Spacecraft Simulation and Hardware-in-the-Loop Testing*”, por Ptak, Foundy em 1998.

“*Control strategies for hardware-in-the-loop simulation of flexible space robots*”, por Carufel, Martin em 2000.

O conceito de HIL mostra um sistema composto principalmente pelos seguintes componentes mostrados na figura 2.1.

1. Modelo Matemático: O modelo matemático da planta ou componente do ambiente.
2. Sensores e Atuadores: Componentes que têm a função de medir ou entregar algum sinal do processo, com a característica de necessitar de um condicionamento dos sinais.
3. Sistema de Aquisição: Placa ou sistema de aquisição de sinais, para testes *On-Line* ou *Off-Line*.
4. Unidade de Controle: Sistema para processar os sinais e comunicar a interface gráfica com o sistema de aquisição, através de diferentes tipos e protocolos de comunicação.
5. Interface Gráfica: É o modo de visualizar os diferentes sinais do processo, para obter um melhor controle do processo, e assim interagir com mudanças dos parâmetros e características em tempo real.
6. Componente Físico do sistema: Elemento do ambiente real que interage através dos outros componentes com o modelo simulado.

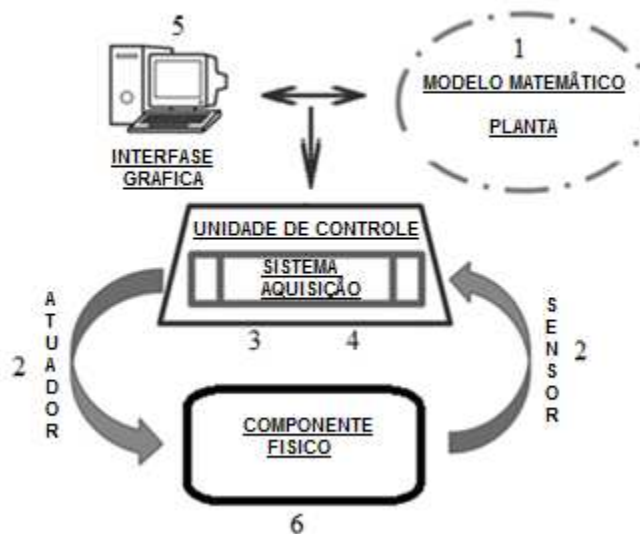


Figura 2.1 Componentes do HIL

De acordo com a definição de DEL SIGNORE E KROVI (2005), HIL é uma técnica que permite realizar uma representação rápida do modelo virtual de um sistema, em um protótipo físico através de sistemas embarcados, com o intuito de permitir a realização de testes mais próximos com a realidade; em um ponto de vista semelhante, para CRAVOTTA (2005), HIL é uma técnica que combina e interconecta o real e o virtual, dentro de uma configuração operacional, para simular e testar o comportamento dinâmico de sistemas complexos.

O HIL tem sido usado com resultados aceitáveis nas áreas da Indústria Automotiva, de Produção e da Robótica. Sendo assim, apresenta-se uma divisão nestes dois (2) campos conforme mostra a figura 2.2, procurando levar ao leitor uma ideia das subáreas que utilizam os avanços e desenvolvimentos desta técnica.

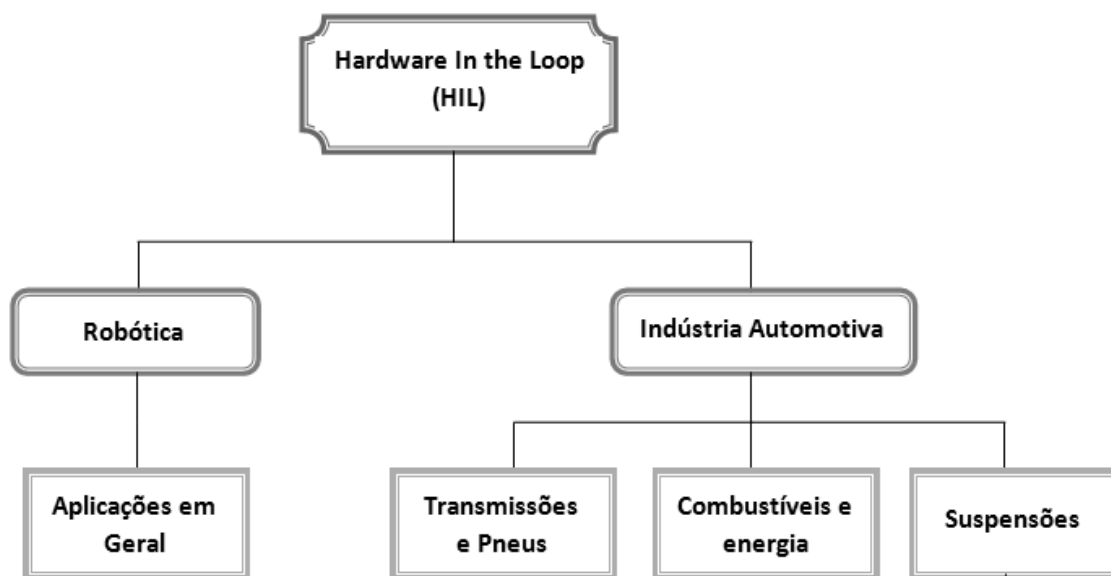


Figura 2.2 Estrutura da revisão Bibliográfica

Graças às pesquisas realizadas pelas empresas e laboratórios nos diferentes campos e áreas foram apontadas por SHIAKOLAS (2003) e PASSOS (2008) como vantagens a possibilidade de desenvolver produtos com uma maior rapidez com menor custo, de melhorar a

interação entre modelos simulados e protótipos físicos, de verificar o comportamento do produto com maior rigor e assim aumentar a credibilidade dos testes associando simulações de subsistemas modelados com componentes reais, para detectar e antecipar possíveis falhas no projeto antes que o produto seja industrializado; e uma desvantagem também foi apontada como a de estar limitado aos recursos computacionais, já que esta técnica exige recursos elevados e complexos para se obter uma grande precisão e resolução nos seus resultados.

Para aplicação desta técnica no desenvolvimento de diferentes processos, necessita-se conhecer um conceito importante que é aplicado, como o de que os sistemas devem ser executados em tempo real. A *Oxford Dictionary of computing* define um sistema em tempo real como sendo qualquer sistema para o qual é importante o instante em que a saída é produzida. Isto acontece por que geralmente a entrada corresponde a algum movimento no mundo físico e a saída esta relacionada com o mesmo movimento. Por tanto o intervalo de tempo entre a entrada e a saída deve ser suficientemente pequeno para se obter um desempenho aceitável. Para YOUNG (1982) o tempo real é qualquer atividade ou sistema de processo de informação que tem que responder a um estímulo de entrada gerado externamente, em um período finito e especificado. Consequentemente, RANDELL (1995), define um sistema em tempo real como aquele ao qual é solicitado que responda a estímulos do ambiente exterior (incluindo o passo do tempo físico) em intervalos de tempo definidos pelo ambiente exterior. Concluindo, segundo (BUTTAZZO, 1997) a definição de um sistema em tempo real é um sistema computacional que deve reagir a estímulos oriundos do seu ambiente em prazos específicos. Esta característica faz com que o sistema seja previsível, mas para isto se faz necessário conhecer seu comportamento antes de sua execução.

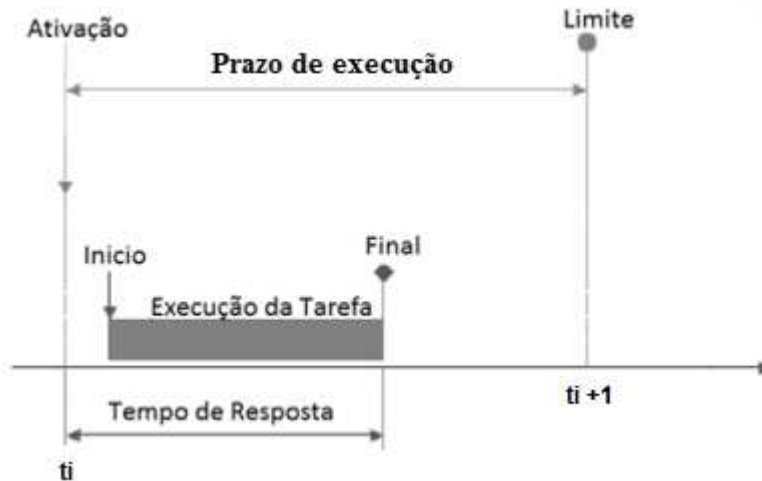


Figura 2.3 Execução do Tempo Real

Em resumo, um sistema precisa cumprir com três condições principais para ser chamado de Sistema em Tempo Real (STR):

- Interatuar com o mundo real.
- Entregar respostas corretas.
- Cumprir com as restrições temporais segundo a dinâmica do sistema.

Estes sistemas em tempo real STR, possuem características interessantes. Uma dessas características é relativa a seu modo de execução, onde através de um computador (moderno), apesar do seu processamento ser sequencial, do ponto de vista do sistema ele se comporta como se o processamento fosse simultâneo. Outra característica interessante é sua velocidade de execução, onde para ser considerado um STR todas as tarefas devem ser executadas em um tempo menor que o tempo de amostragem.

Para a criação e desenvolvimento deste tipo de sistemas em diferentes aplicações existem algumas empresas que se dedicam em cumprir essa necessidade. Uma destas empresas é a DSPACE que desenvolve softwares e *hardwares* dedicados. Por exemplo, através de suas soluções (PETIT e MANTILLA, 2011), conseguiram desenvolver um trabalho na área da engenharia elétrica onde o objetivo foi o de criar um algoritmo para a medição de parâmetros que afetam a qualidade da energia elétrica através de simulações pelo MATLAB (Simulink). Esta aplicação foi feita em tempo real com a placa de aquisição DS1104, apresentando resultados satisfatórios.

Outra destas empresas também representativa neste campo é a NATIONAL INSTRUMENTS (NI), onde (BORGES , 2002) através de seu software LABVIEW elaborou um trabalho aonde são apresentados os resultados de um sistema flexível de monitoração e controle de ensaios experimentais através de uma rede de computadores, para um laboratório virtual multiusuários em tempo real com acesso via Internet. Este sistema consiste na interligação de instrumentos de medida, placa de aquisição de dados e software que controla os instrumentos conforme mostrado mostra a figura 2.4.

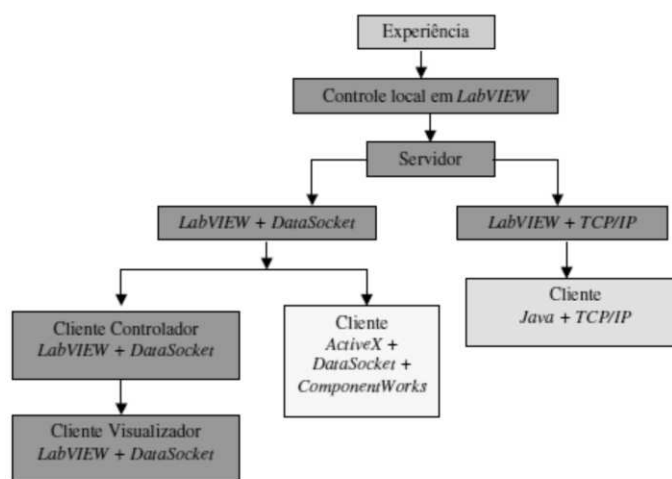


Figura 2.4 Metodologia de desenvolvimento adotada. (BORGES , 2002)

2.2. Atraso em *Hardware In the Loop* (HIL)

O fenômeno do atraso é um termo técnico usado para designar o atraso de um sinal com relação ao sinal original conforme mostra a Figura 2.5. Em um sistema HIL este atraso (*delay* ou *lag*) pode ser gerado por vários motivos seja pelo efeito de amortecimento negativo segundo HORIUCHI, INOUE e outros (1999), ou pela etapa de filtragem dos sinais segundo MISSELHORN (2004), ou pelos diferentes tempos de execução das tarefas segundo LINS (2007). Todos estes atrasos levam a erros nas respostas ou sinais de interesse e serão explicados a seguir.

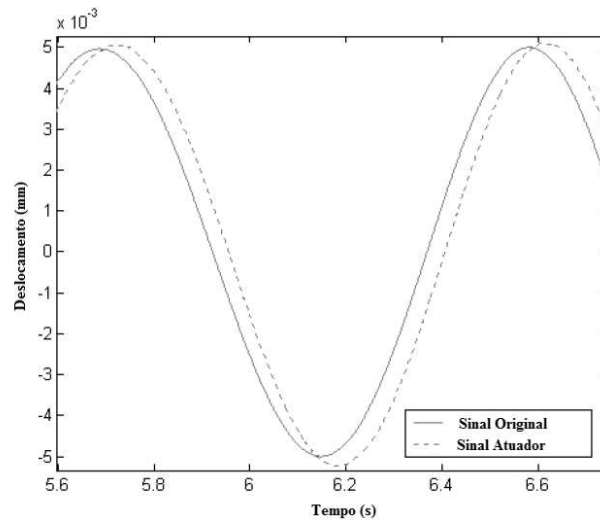


Figura 2.5 Sinal com atraso

Este fenômeno nos sistemas HIL, tem sido objeto de pesquisa em diferentes trabalhos, discutindo algumas das possíveis teorias, do porque é gerado o atraso, onde são gerados e em que condições eles são gerados.

BURNS E WELLINGS, (1997) e LIU (2000) mostraram uma análise da origem do atraso. Levando-se em conta que em simulações em tempo real, o computador geralmente se comunica com outros componentes através de *hardware* como sensores, atuadores, conversores Analógico/Digital (A/D) e Digital/Analógico (D/A) entre outros durante a simulação, e que o processamento de todas as suas tarefas devem ser executadas em tempo real, estas tarefas são executadas em intervalos fixos e curtos de amostragem diferentes do tempo escolhido de amostragem em tempo real, podendo gerar fenômenos como atrasos e erros na comunicação.

Como solução para se executar em tempo real devem-se otimizar as tarefas, utilizar processadores mais rápidos e também interatuar com o código modificando e otimizando a sequência do processo.

Tendo como objetivo diminuir ou eliminar o atraso através de diferentes métodos, segundo os trabalhos mencionados anteriormente, alguns autores tem estudado este fenômeno para encontrar uma solução. Uma solução matemática foi apresentada por HORIUCHI, INOUE et al (1999), onde se aplicou o conceito de *real time* em uma estrutura geradora de vibrações usando um atuador. O objetivo foi o de desenvolver matematicamente um sistema modificado HIL para corrigir o atraso, comparando os resultados (simulados e experimentais) e que estes sinais mostrados na figura 2.6 se aproximaram, demonstrando as vantagens da técnica HIL. Este método prediz o deslocamento \dot{x} do atuador depois do atraso gerado, sendo dependente do tempo do atraso e da frequência de excitação, através da equação 2.1, onde n é a ordem da predição, x_0 é o deslocamento calculado, x_i é o deslocamento calculado nos instantes t_i e a_i é uma constante. Depois em um trabalho posterior (DARBY, WILLIAMS et al., 2002) aplicaram a mesma técnica de compensação para um atuador hidráulico com resultados aceitáveis.

$$\dot{x} = \sum_{i=0}^n a_i x_i$$

Equação 2.1.

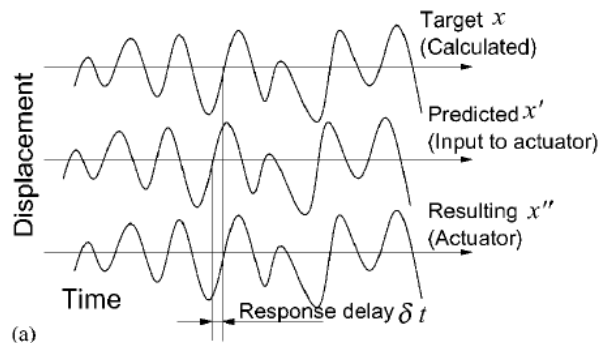


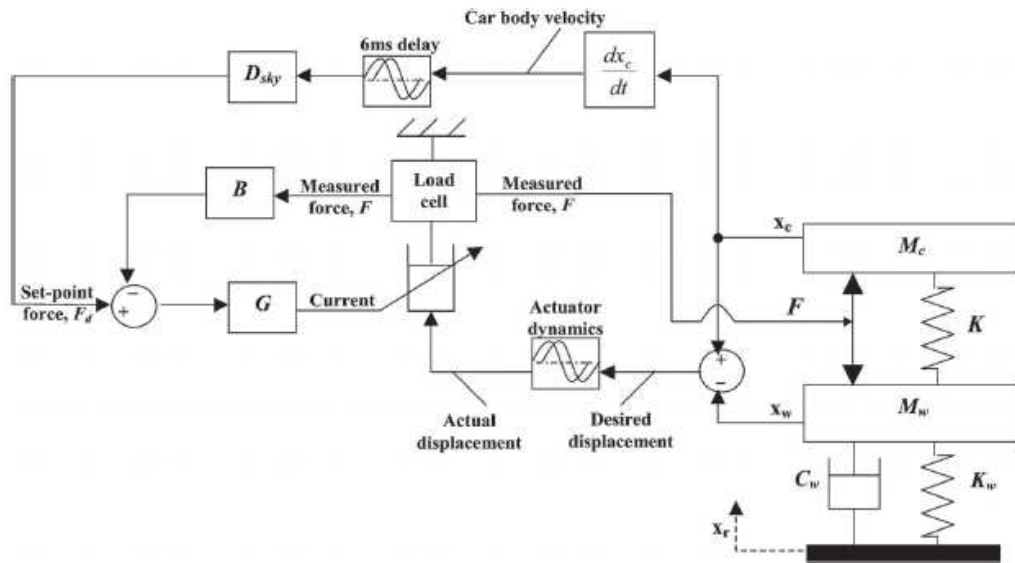
Figura 2.6 Predição do deslocamento para compensação do atraso (HORIUCHI, INOUE et al., 1999)

Outros trabalhos também apresentam o atraso com pontos de vista bastante semelhantes, como o de REINHORN e SIVASELVAN (2004), o qual propôs dois (2) algoritmos de controle de força, um deles baseado na convolução em tempo real e o outro em series de elasticidade com compensação no deslocamento. Foram feitos testes numa bancada experimental com resultados satisfatórios. WALLACE e SIEBER (2005) através de um exemplo linear propuseram uma solução para a compensação do atraso chamado de *Delay differential equation* (DDE), em dois métodos (analítico (AFP⁸) e simulado). Ele tinha como objetivo no seu trabalho estudar o fenômeno do atraso em sistemas dinâmicos, e obteve resultados satisfatórios.

MISSELHORN, (2004) propôs alterações no controle PID e na etapa de filtragem dos sinais do atuador, para minimizar o atraso, obtendo entretanto como conclusão que estas alterações realizadas não afetaram o resultado no sistema HIL. Seu trabalho consistia em um sistema massa, mola e amortecedor, para a modelagem dos sistemas de (1/4) de veículo com 1 (um) e 2 (dois) graus de liberdade, usando os softwares MATLAB (Simulink), além de uma bancada experimental não linear, fazendo uma comparação destes modelos através de *Error Coefficient of Variance* (ECOV).

BATTERBEE e SIMS (2006) implementaram o HIL em sistemas de amortecimento magneto reológico em suspensão de veículos. Utilizando as leis de controle de *SkyHook* para um modelo de (1/4) de veículo com 2 (dois) graus de liberdade, foi exibido um atraso de 6 ms nos testes experimentais. Este atraso de primeira ordem foi simulado e implementado através do software para obter a medição correta conforme mostra a Figura 2.7. Concluiu que as respostas foram aceitáveis em baixas frequências, mas afetava os resultados em altas frequências (7 -8 Hz).

⁸ AFP: *Adaptive Forward Prediction*



**Figura 2.7 Controle SkyHook para ¼ de veículo com HIL
(BATTERBEE, SIMS, 2006)**

Mudando o campo de aplicações, mais ainda apresentando trabalhos com o fenômeno de atraso, LINS (2007), aplicou o HIL em uma mão artificial robótica, elaborando dois (2) ambientes (virtual e experimental), reduzindo o investimento em hardware e tempo de trabalho, como também mostrando que existem muitas ferramentas de software comerciais para as aplicações. Neste trabalho observou-se uma defasagem nos resultados da comparação entre o HIL clássico como ele o chama ao comportamento de um sistema de motores que foi controlado e simulado em tempo real, e o HIL híbrido que é quando se tem um componente físico ou hardware interagindo com o ambiente simulado.

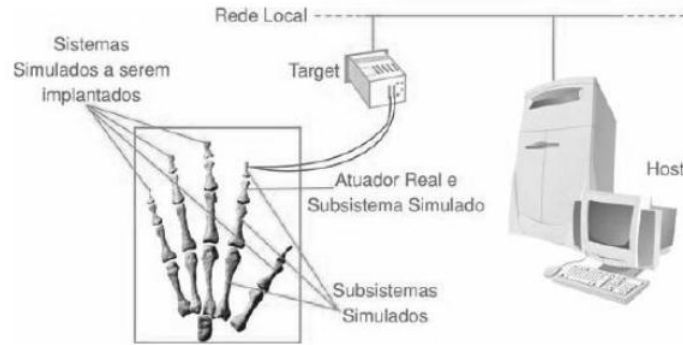


Figura 2.8 Desenvolvimento de HIL

(LINS, 2007)

No artigo de CHANG, CONG e HAN (2007), se avalia a efetividade de um compensador nas características dinâmicas de um sistema modelado com HIL, através de uma plataforma *Stewart* de 6 (seis) graus de liberdade que possui atuadores eletro-hidráulicos. Segundo Chang o atraso se manifesta por três motivos: o primeiro devido ao movimento do simulador, o segundo devido ao sensor de força e por ultimo, devido ao método de controle numérico. Um compensador foi inserido na malha do processo obtendo-se resultados possíveis através do software MATLAB (Simulink-Simmechanics). Finalizando esta revisão, CHEN (2007) apresenta uma revisão bibliográfica de trabalhos e métodos⁹ implementados para minimizar o fenômeno de atraso dos atuadores nas aplicações em tempo real. Ele propôs um novo método de compensação baseado no modelamento simples de um sistema servo-hidráulico usando o conceito de funções de transferência discretas.

Como conclusão segundo esta revisão bibliográfica existe ainda a necessidade de se propor estratégias para eliminar o fenômeno do atraso para sistemas HIL em condições de tempo real, usando ferramentas de *hardware* e software conhecidas no mercado.

⁹ *Polynomial Extrapolation, Linear Acceleration Extrapolation, Model-Based Prediction, Derivative Feedforward, Phase Lead Compensator.*

CAPITULO 3

HARDWARE IN THE LOOP (HIL)

3.1. Modelo Esquemático HIL

Para uma melhor análise da aplicação da técnica de HIL propôs-se um modelo geral e esquemático para a representação do sistema, lembrando-se que este modelo HIL esta composto por partes reais e partes virtuais, cada uma destas partes possuindo características especiais e que uma das partes escolhe-se por substituir pelo seu componente físico por apresentar um grau de complexidade na obtenção do seu respectivo modelo. Entre as partes geralmente existe um software de modelagem e geração automática de códigos que facilita seu desenvolvimento com características de se alterar parâmetros, adquirir sinais, disponibilizar e visualizar dados, e ainda simular processos e sistemas de uma forma geral.

Este modelo está composto por sete (7) itens conforme mostrado na figura 3.1, com um comportamento cíclico e dependente das outras partes. Cada uma destas partes será detalhada para o sistema escolhido nas seções 3.1, 3.2, 3.3.

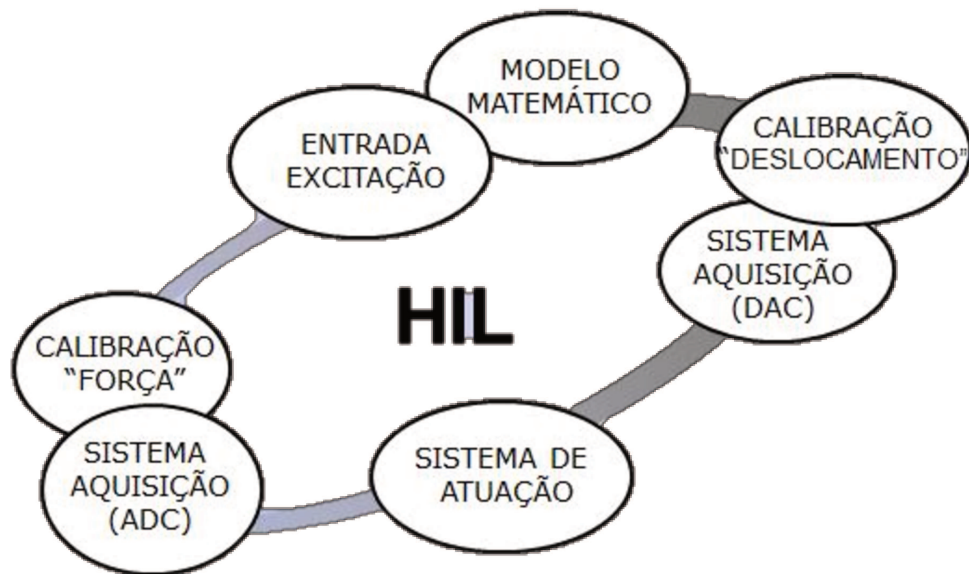


Figura 3.1 Modelo esquemático proposto do HIL.

3.2. Modelagem Matemática do Sistema

Para apresentar cada uma das partes do modelo esquemático do HIL, se faz necessário definir um sistema que se deseja aplicar a técnica de HIL. O sistema que será aplicado à técnica do HIL esta descrito na seção 3.2.1. Uma vez definido o modelo, escolhem-se quais componentes continuarão sendo físicos e quais componentes serão virtuais, ou seja, representados por modelos simulados. A seção 3.2.2 descreve o modelo matemático resultante devido à eliminação do componente físico e substituição deste componente pelas suas forças respectivas.

3.2.1. Modelo real massa, mola e amortecedor 1 DOF discreto

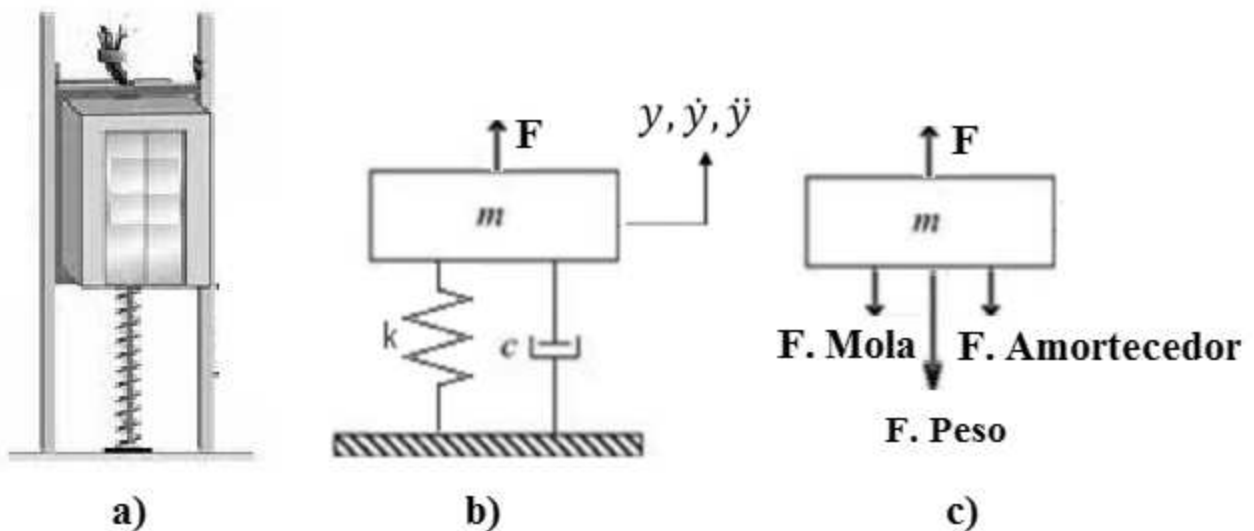


Figura 3.2 Modelo Proposto
a) Real b) Esquemático c) Diagrama de Corpo livre

O sistema que será aplicado à técnica do HIL é um sistema massa, mola e amortecedor de um grau de liberdade conforme mostra a figura 3.2.

Neste modelo o movimento vertical é permitido e suas variáveis estão listadas na tabela 3.1, levando-se em conta que o sistema está referenciado a partir da posição da mola indeformada e não a partir da posição de equilíbrio estático.

Tabela 3.1 Variáveis utilizadas no modelo real

Variável	Unidades	Descrição
m	Kg	Massa suspensa
k	N/m	Constante de rigidez da mola
F	N	Força de excitação
c	N*s/m	Constante de amortecimento do amortecedor
Fp	N	Força Peso da Massa suspensa
Y	m	Deslocamento vertical da massa suspensa
\dot{Y}	m/s	Velocidade vertical da massa suspensa
\ddot{Y}	m/s ²	Aceleração vertical da massa suspensa

O modelo da figura 3.2 pode ser descrito a partir da equação diferencial 3.1;

$$m\ddot{y} + c\dot{y} + ky = F - F_p \quad \text{Equação 3. 1}$$

Adotando como variáveis de estado o deslocamento y e sua velocidade \dot{y} , o sistema de segunda ordem é reduzido a um sistema de primeira ordem através do método das variáveis de estado. A equação de estado em tempo contínuo e invariante no tempo é dada pela equação 3.2.

$$\begin{aligned} \dot{x} &= ax + bu \\ y &= cx + du \end{aligned} \quad \text{Equação 3. 2}$$

onde $x = \{ y ; \dot{y} \}$, as matrizes a, b, c e d são dadas por:

$$a = \begin{bmatrix} 0 & 1 \\ k/m & c/m \end{bmatrix} \quad \text{Equação 3. 3}$$

$$\mathbf{b} = \begin{bmatrix} 0 \\ 1/m \end{bmatrix} \quad \text{Equação 3. 4}$$

$$\mathbf{c} = [1 \quad 0] \quad \text{Equação 3. 5}$$

$$\mathbf{d} = 0 \quad \text{Equação 3. 6}$$

e a entrada u do sistema esta dada portanto pela força de excitação e pela força peso constante da massa suspensa dado pela equação 3.7.

$$\mathbf{u} = \mathbf{F} - \mathbf{Fp} \quad \text{Equação 3. 7}$$

Discretizando o sistema da equação 3.2 pelo método degrau invariante utilizando um segurador de ordem zero e uma taxa de amostragem Ta , obtendo-se as matrizes A , B , C , D do sistema equivalente no espaço discreto resultando na equação 3.8.

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{Ax}(k) + \mathbf{Bu}(k) \\ \mathbf{y}(k) &= \mathbf{Cx}(k) + \mathbf{Du}(k) \end{aligned} \quad \text{Equação 3.8}$$

3.2.2. *Modelo HIL massa, mola física e amortecedor de 1 DOF discreto*

Vai se utilizar o modelo da figura 3.3, onde foi trocado o elemento físico mola, por uma entrada de força, para ser aplicada a técnica do HIL.

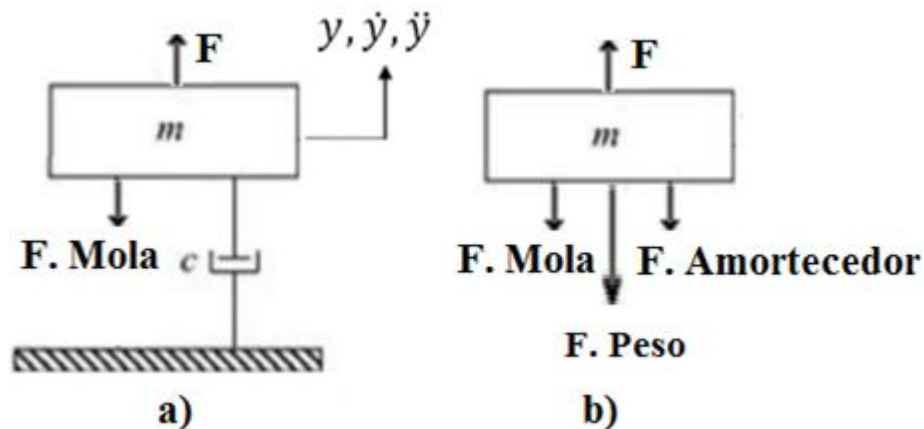


Figura 3.3 Modelo Proposto
a) Esquemático b) Diagrama de Corpo livre

O sistema resultante da eliminação da mola na modelagem e substituição pela força correspondente que será utilizada na técnica do HIL é um sistema massa, amortecedor de um grau de liberdade com a força de excitação F e a força da mola F_{Mola} conforme mostra a figura 3.3 a).

Neste modelo o movimento vertical é permitido e suas variáveis estão listadas na tabela 3.2. O sistema está referenciado a partir da posição inicial em que a mola não está deformada e não a partir do equilíbrio estático.

Tabela 3.2 Variáveis utilizadas no modelo HIL

Variável	Unidades	Descrição
m	Kg	Massa suspensa
F	N	Força de excitação
c	N*s/m	Constante de amortecimento do amortecedor
F. Mola	N	Força gerada pela mola física
Y	m	Deslocamento vertical da massa suspensa
\dot{y}	m/s	Velocidade vertical da massa suspensa
\ddot{Y}	m/s ²	Aceleração vertical da massa suspensa
Fp	N	Força Peso da Massa suspensa

O modelo da figura 3.3 pode ser descrito a partir da equação diferencial 3.9;

$$m\ddot{y} + c\dot{y} = +F - F.Mola - Fp \quad \text{Equação 3. 9}$$

Adotando como variáveis de estado o deslocamento y e sua velocidade \dot{y} , o sistema de segunda ordem é reduzido a um sistema de primeira ordem através do método das variáveis de estado. A equação de estado em tempo contínuo e invariante no tempo é dada pela equação 3.10.

$$\begin{aligned} \dot{x} &= ax + bu \\ y &= cx + du \end{aligned} \quad \text{Equação 3. 10}$$

onde $x = \{ y ; \dot{y} \}$, as matrizes a, b, c e d são dadas por:

$$a = \begin{bmatrix} 0 & 1 \\ 0 & c/m \end{bmatrix} \quad \text{Equação 3. 11}$$

$$b = \begin{bmatrix} 0 \\ 1/m \end{bmatrix} \quad \text{Equação 3. 12}$$

$$c = [1 \quad 0] \quad \text{Equação 3. 13}$$

$$d = 0 \quad \text{Equação 3. 14}$$

e a entrada u do sistema está dada portanto pela força de excitação, pela força gerada pela mola

física e pela força peso constante da massa suspensa dado pela equação 3.15.

$$u = F - F.mola - Fp \quad \text{Equação 3. 15}$$

Discretizando o sistema da equação 3.10 pelo método degrau invariante utilizando um segurador de ordem zero e uma taxa de amostragem Ta , obtendo-se as matrizes A , B , C , D do sistema equivalente no espaço discreto resultando na equação 3.16.

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned} \quad \text{Equação 3.16}$$

3.3. Ferramentas usadas no HIL

Nesta seção serão mostradas as ferramentas principais de desenvolvimento de HIL que serão usadas neste trabalho. No trabalho de PASSOS (2008) apresenta-se uma revisão mais detalhada das empresas, linhas e componentes que podem ser utilizadas para o uso da técnica do HIL.

3.3.1. Software MATHWORKS

A MathWorks desenvolveu uma ferramenta muito interessante chamada Simulink, a qual permite a modelagem de montagens mecânicas e seus controladores, modelagem de componentes mecânicos, obter a simulação dos movimentos e análise direta dos resultados dentro de um mesmo ambiente através de diagramas de blocos, sem ter que por exemplo programar todos os passos para se introduzir as equações de movimento e resolver o sistema de equações do sistema, proporcionando a concentração na criação de algoritmos de controle mais convenientes e específicos da pesquisa, apresentando ainda a capacidade de interagir com outras ferramentas de modelagem tais como ADAMS, PRO-ENGINEER, AMESIM, etc...

Outra ferramenta de desenvolvimento de projetos que utiliza processamento digital de sinais é a *DSP Builder* que permite a integração de blocos do Simulink com blocos de implementação de lógica em FPGA, permitindo assim a elaboração e simulação de sistemas no ambiente MATLAB. Outra opção interessante é a possibilidade de utilização destes blocos já embutidos em uma FPGA para acelerar um processo de simulação, ou de verificar o funcionamento de uma lógica utilizando o Simulink como ferramenta de geração e verificação de sinais.

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 incluem blocos para interface com I/O de *hardwares* específicos. Estes modelos são descarregados no *hardware* de destino e sendo executados pelo

sistema operacional *Real-Time* proprietário, desenvolvendo controladores mais rapidamente e acelerando a transição da implementação do controlador simulado na implementação real.

Uma ferramenta muito interessante e que é utilizada no HIL é a chamada *Real Time Workshop* RTW, o qual tem a tarefa de gerar e executar o código C para desenvolvimentos e testes de algoritmos modelados no Simulink. Estes códigos obtidos podem ser usados em aplicações em tempo real ou não.

A figura 3.4 apresenta um exemplo completo de implementação *Hardware in Loop* HIL para um sistema mecatrônico, onde partes do modelo físico são geradas no ambiente Simulink.

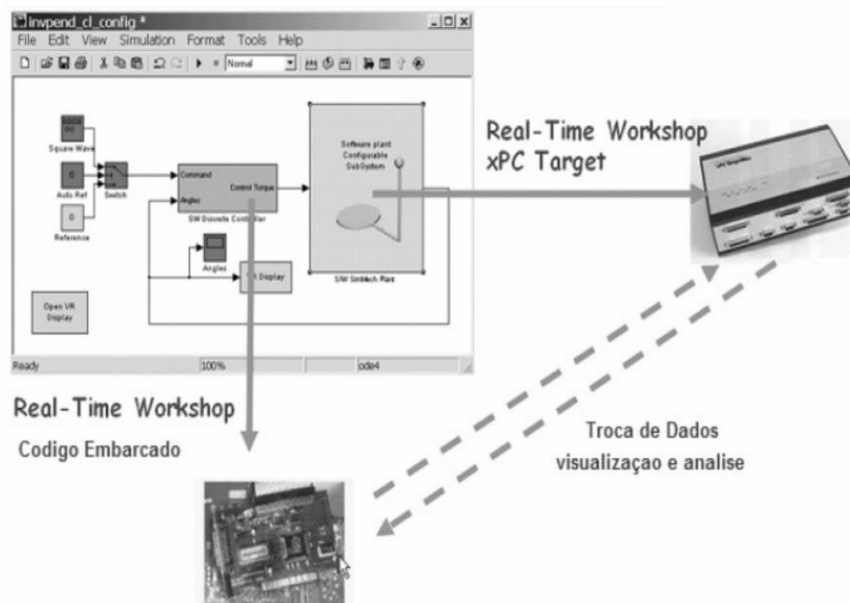


Figura 3.4 Aplicação do MATLAB para o HIL (PASSOS, 2008)

3.3.2. Placa DSPACE DS-1102

Conhecendo as capacidades que apresenta o software Simulink e o *Real Time Workshop* RTW do MATLAB, a DSPACE elaborou ferramentas para que seu *hardware* consiga interagir com estes softwares, no desenvolvimento de aplicações através do HIL. Uma destas ferramentas é o *Real Time Interface* RTI, representado por meio de blocos os quais estão compostos das

diversas funções do *hardware*, tais como sinais de aquisição, sensores, conversores entre outros, segundo o modelo do equipamento da DSPACE.

A placa 1102 da Dspace mostrada na figura 3.5 esta composta de um processador Texas Instrument TMS320C31 DSP de 60 MHz, 33.3 ns de tempo de ciclo, quatro (4) canais para entradas e saídas análogas de 12 e 16 bits, range de saída de ± 10 V, com a possibilidade de usar conversores DAC (*Digital Analog Converter*) ou ADC (*Analog Digital Converter*,) e entrada e saída digital. Esta placa para funcionar corretamente necessita de um computador que apresenta barramento ISA, sistema operacional Windows 2000 *Professional* e ainda Matlab 6.5 com as ferramentas Simulink, *Real Time Workshop* RTW e *Real Time Interface* RTI (GMBH, 2008).



Figura 3.5 Placa DSPACE DS-1102

3.3.3. **Software CONTROLDESK**

Este é um software usado para acompanhar a aquisição, visualização e monitoramento de dados, podendo-se alterar alguns dos parâmetros e variáveis de modo contínuo (GMBH, 2003), interagindo com os componentes e características do Simulink do MATLAB. Este software mostrado na figura 3.6 apresenta uma interface simples para iniciar, validar, parar e acompanhar o comportamento dos modelos de operação e controle, com ferramentas de automação, instrumentação virtual e com a vantagem de ser em tempo real.

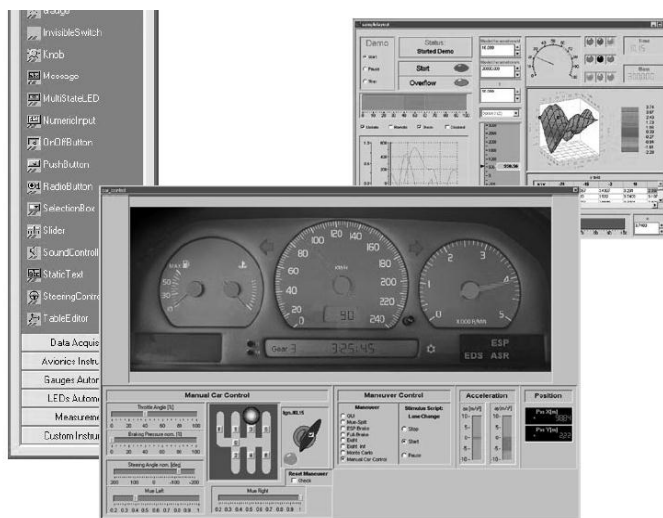


Figura 3.6 Instrumentação Virtual ControlDesk catálogo 2008

3.3.4. Sistema INSTRON/SCHENCK

O sistema Instron/Schenck é composto por dois componentes. O primeiro componente é o sistema de aquisição mostrado na figura 3.7 a) composto por dois (2) canais com (3) conectores BNC (duas saídas, uma entrada) de $\pm 10V$, com uma comunicação com o Host (computador com Windows NT) é IEEE488 (*GPIB*¹⁰), com identificação e calibração nos conectores e possui um completo software chamado Rs-LabSite com ferramentas variadas para aquisição, visualização e controle dos processos e tarefas que se deseja desenvolver. O segundo componente é o sistema de atuação mostrado na figura 3.7 b) composto por dois (2) cilindros hidráulicos com deslocamento linear, um deles com uma capacidade de geração de força de 10 KN e um deslocamento de 100 mm e o outro com uma capacidade de geração de força de 25 KN e um deslocamento de 100 mm. Estes atuadores estão equipados com células de carga resistivas, sensores de deslocamento LVDT, entre outros elementos que ajudam no desenvolvimento da aplicação.

¹⁰ *GPIB*: acrônimo para *General Purpose Interface Bus*, este foi criado pela *Hewlett-Packard* nos anos 70, para conectar dispositivos de testes e medições.

Estes dois sistemas trabalham juntos, através do *software* de monitoramento e controle das tarefas que podem ser operadas manualmente ou automaticamente.

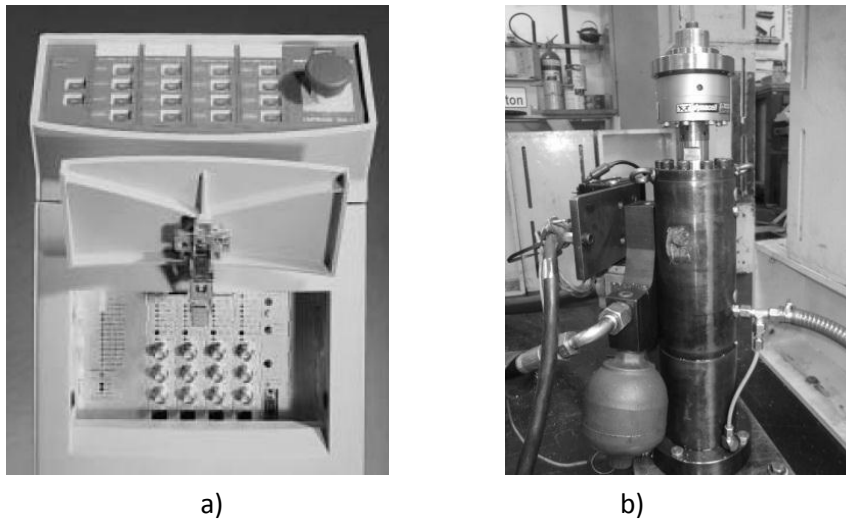


Figura 3.7 Sistema aquisição e atuação Instron/Schenck

3.4. Montagem Bancada e Calibração

Esta é uma etapa bastante importante no processo. Nesta etapa realizam-se testes simples de entrada e saída de sinais através dos conversores (A/D e D/A) da placa de aquisição, para se conhecer características do sistema como por exemplo o tempo mínimo de processamento, se os sinais de resposta apresentam algum atraso ou ainda os valores de sensibilidade da força e do deslocamento entre os sinais digitais antes dos conversores e depois dos conversores e vice-versa.

Para se realizar estes testes se faz necessário construir a montagem física do sistema com o componente de *Hardware* que será utilizado, neste caso a mola, como mostrado na figura 3.8. Observa-se que nos testes de calibração será imposto um deslocamento sobre a mola para se medir através da célula de carga a sua reação (força) e através do sensor LVDT do atuador o deslocamento resultante.

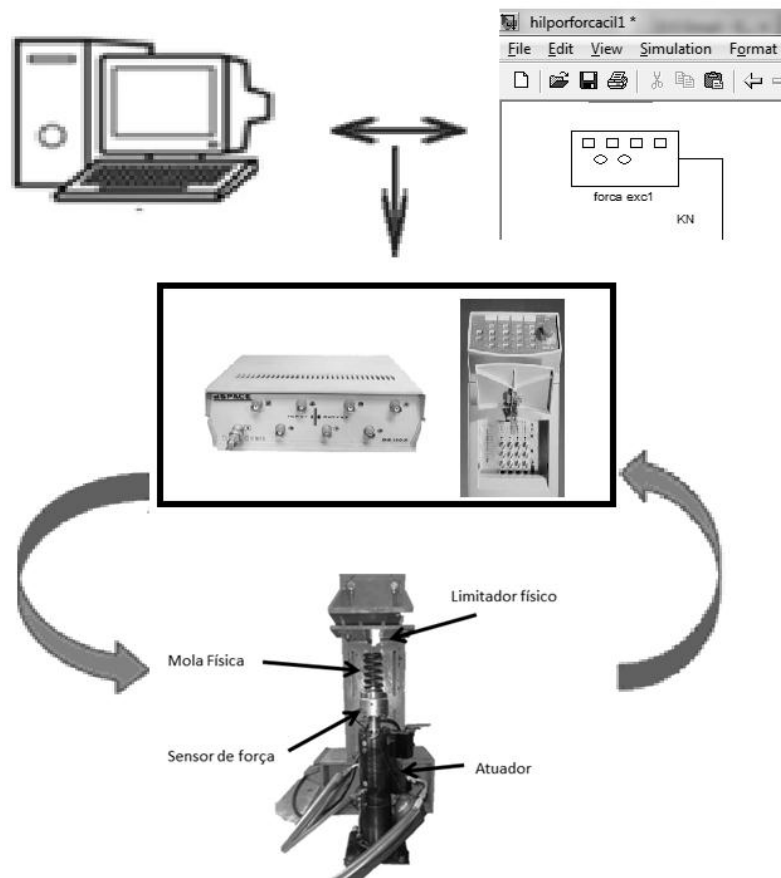


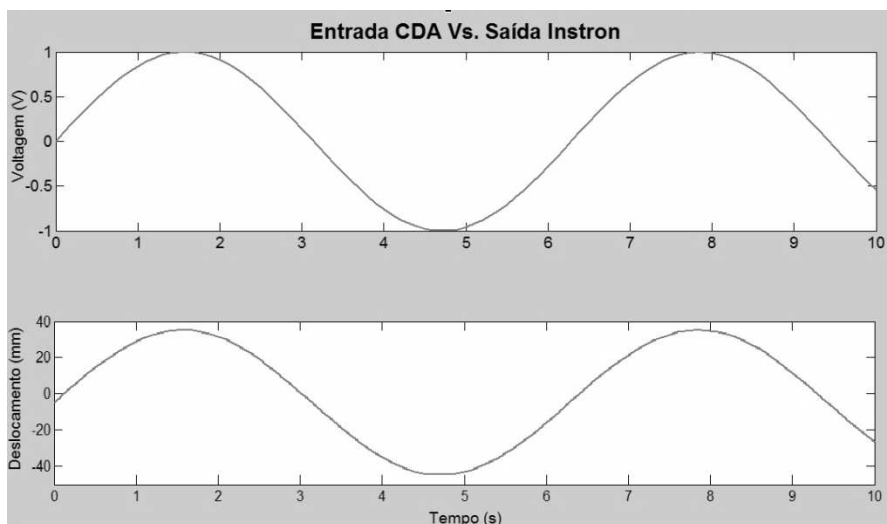
Figura 3.8 Montagem física

Algumas considerações nesta etapa são importantes. A primeira é que as calibrações devem ser feitas individualmente em cada atuador e os valores obtidos serão próprios do sistema composto pelo atuador, seus componentes mecânicos, hidráulicos e eletrônicos, ou seja, os valores das calibrações de cada atuador não serão iguais. A segunda observação é que cada atuador possui um controlador PID e, portanto, apresenta uma resposta dinâmica, ou seja, ao se impor um deslocamento no atuador a sua resposta não é instantânea devido a vários fatores como a inércia, filtragem dos sinais, resposta do sistema entre outros.

3.4.1. Calibração do deslocamento

Para esta calibração do deslocamento realizou-se um teste simples utilizando o software Simulink e o software Rs-LabSite e comparou-se as respostas enviadas com as obtidas. Neste

caso o teste consistiu primeiramente em gerar no Simulink um sinal senoidal com amplitude máxima de 1 Volt que entra no bloco do conversor Digital/Analógico. Em seguida se faz a aquisição da magnitude da resposta do sistema no software de visualização do Instron Schenck. Os valores obtidos estão mostrados na figura 3.9 a). Observa-se que o valor da amplitude da resposta no atuador depende ainda de parâmetros que são configurados no software RS-LabSite como o curso máximo permitido ao cilindro, bem como também da coordenada inicial “*SetPoint*” do atuador como mostrado nos parâmetros da figura 3.9 b).



a)



b)

Figura 3.9 Calibração deslocamento Simulink-Instron
a) Respostas b) Parâmetros software Instron

Observa-se na figura 3.9 b) que o “*SetPoint*” foi definido como o deslocamento inicial em que a mola começa a ser comprimida, ou seja, o deslocamento em que a força ainda é nula, neste caso com valor de -4.8mm.

Após a calibração do deslocamento chegou-se aos dados apresentados no diagrama de blocos da figura 3.10. Observa-se que se tem uma constante -4.8mm relativo ao *SetPoint*, com uma amplitude representada por um ganho de 40 indicando que o deslocamento máximo permitido ao cilindro é de 40mm, e um ganho de conversão devido a que o sistema de equações calcula uma resposta do deslocamento em metros (m) e o sistema Instron recebe valores em milímetros (mm).

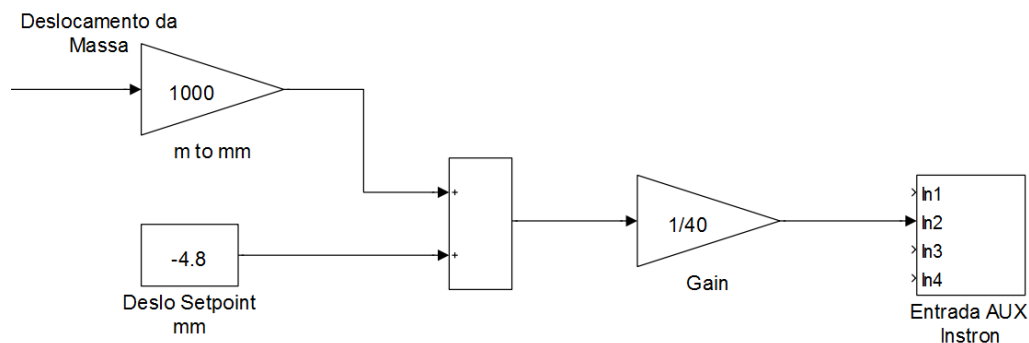


Figura 3.10 Modelo Simulink Calibração Deslocamento

3.4.2. Calibração da força

Seguindo a mesma metodologia utilizada na calibração do deslocamento foi feito um teste simples gerando uma resposta qualquer ao sistema e o sinal de força foi adquirido pelo software Rs-LabSite da Instron bem como pelo conversor analógico digital do sistema de aquisição Dspace. Os dados registrados pelo programa Controldesk foram salvos em arquivos. Ambos os dados podem ser vistos na figura 3.11, onde a linha tracejada corresponde aos dados obtidos do conversor analógico/digital (CAD) e a linha sólida corresponde aos dados do sistema Instron.

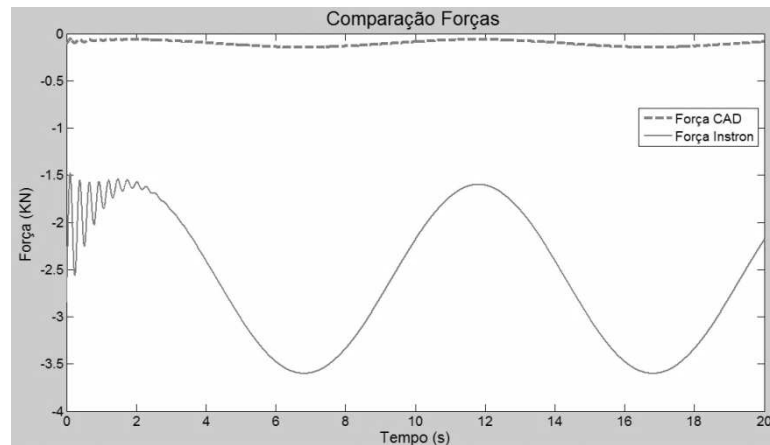


Figura 3.11 Respostas em força do Simulink-Instron

Pode-se observar que os dados não estão calibrados e que estes dados precisam ser ajustados. Portanto realizou-se a calibração ajustando os dados adquiridos do software da Control desk pelos dados obtidos no software da Rs-LabSite da Instron obtendo-se os valores da figura 3.12 e o diagrama de blocos da figura 3.13

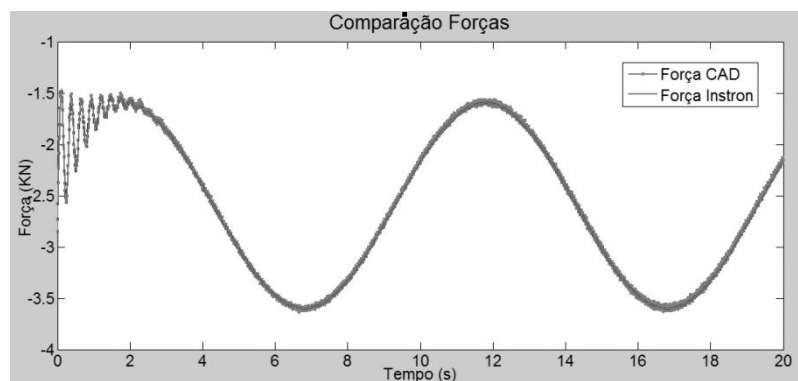


Figura 3.12 Respostas em força calibrada do Simulink-Instron

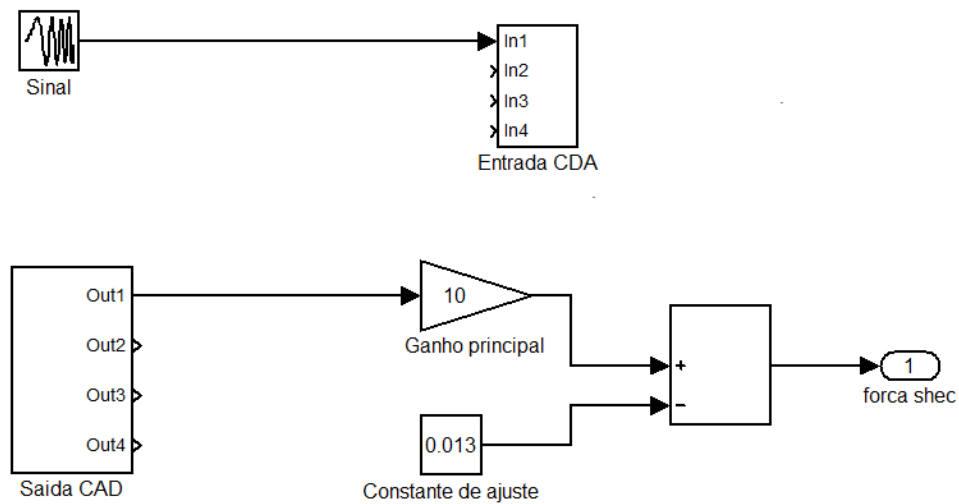
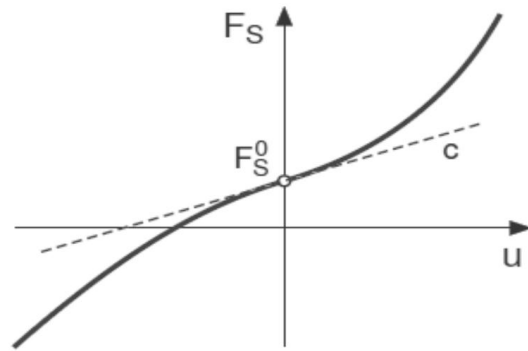
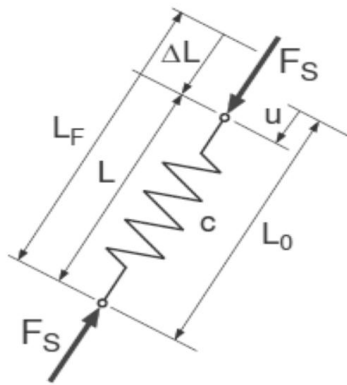


Figura 3.13 Modelo Simulink Calibração Força

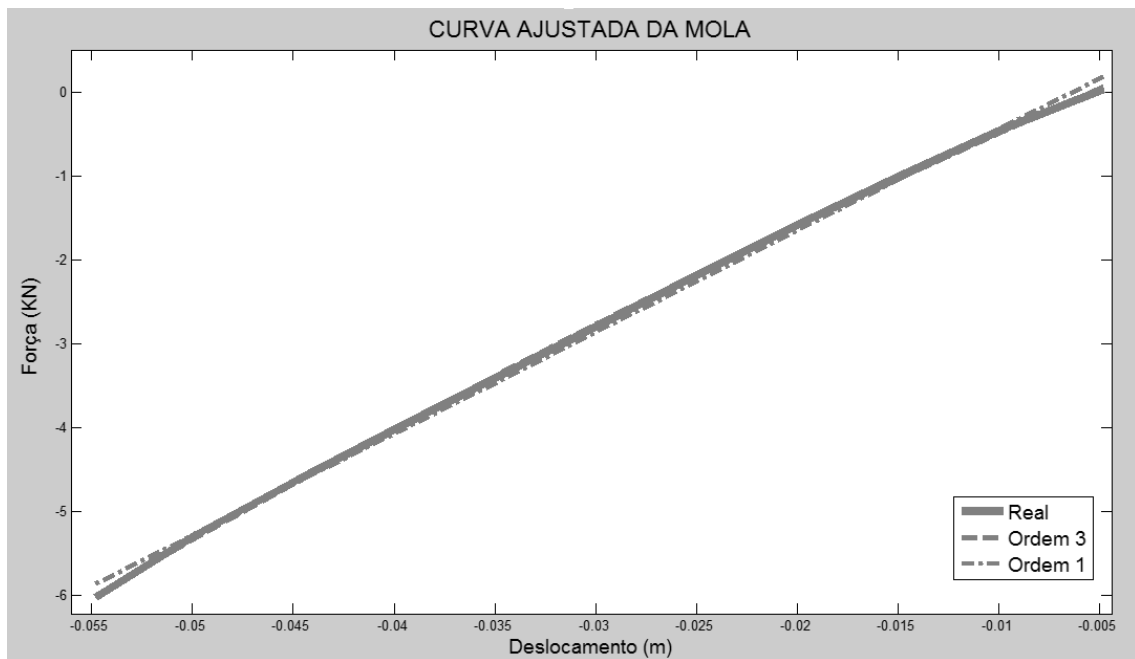
3.4.3. Calibração dos parâmetros da mola física

Decidiu-se em obter os parâmetros de calibração da mola física para auxiliar nos testes de validação das respostas simuladas com as respostas obtidas com o HIL.

O procedimento para se obter as características da mola foi o de impor deslocamentos conhecidos conforme mostrado na figura 3.14 a) e em seguida medir a força resultante através dos softwares de aquisição.



a)



b)

Figura 3.14. a) Características gerais na mola (RILL,2011)

b) Curva ajustada da mola

Após a aquisição de vários pontos da curva (força x deslocamento) mostrado na figura 3.14 (b) utilizou-se da função Polyfit e Polival¹¹ do MATLAB para se obter o comportamento

¹¹ Funções do MATLAB, que proporciona os coeficientes polinomiais através da técnica de mínimos quadrados e os avalia segundo um dado.

linear ajustado da mola dado pela equação 3.17. Para maior precisão segundo os testes experimentais decidiu-se em aumentar a ordem da equação para se obter melhores resultados. Neste caso adotou-se uma equação de ordem 3, a qual está representada na equação 3.18. O resultado obtido mostra que a curva desta ordem tem um comportamento mais próximo da curva real conforme pode ser visto na figura 3.14 (b).

$$Y = aX + b \quad \text{equação 3.17}$$

$$Y = aX^3 + bX^2 + cX + d \quad \text{equação 3.18}$$

Os resultados obtidos do ajuste estão na tabela 3.3, e sua representação no Simulink através de um ganho e uma constante que simulam a equação correspondente é mostrada na figura 3.15.

Tabela 3.3. Coeficientes da mola

PARÂMETRO	ORDEM UM	ORDEM TRÊS
FATOR DE AJUSTE "a"	121.1291	-122.7661
FATOR DE AJUSTE "b"	0.7680	-358.7142
FATOR DE AJUSTE "c"	-----	100.1314
FATOR DE AJUSTE "d"	-----	0.5445

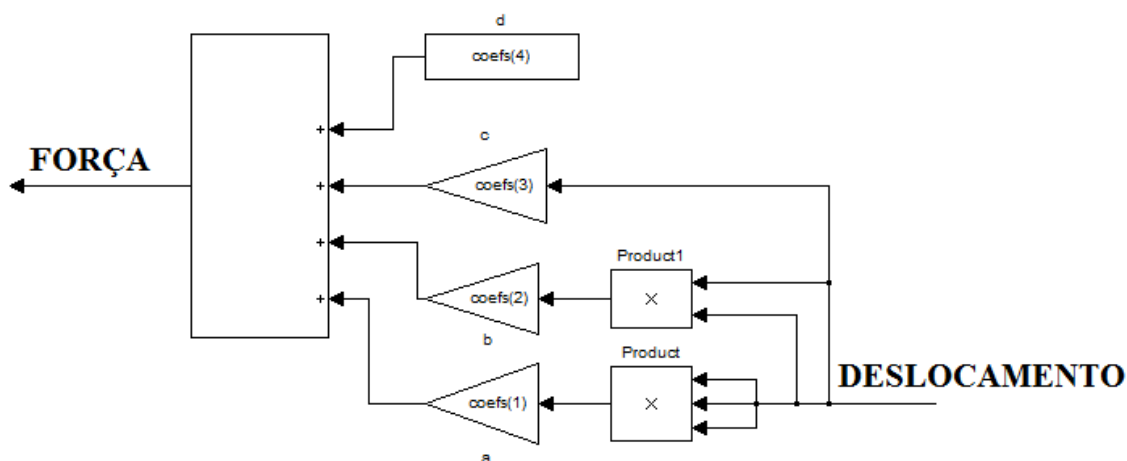


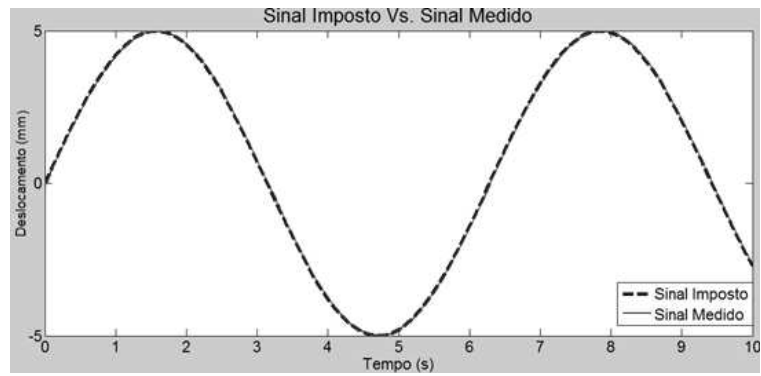
Figura 3.15 Modelo Simulink da Mola

3.4.4. *Resposta do Sistema Instron Schenck*

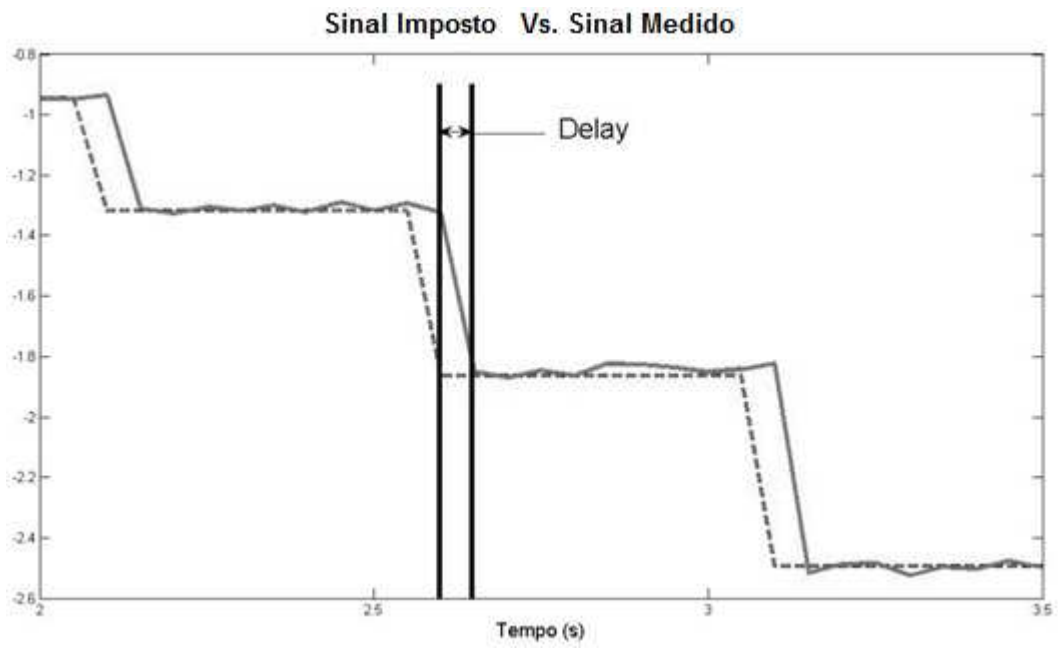
Este sistema de atuação tem como função impor deslocamento na mola e registrar a sua respectiva força. Para se verificar o tempo de atraso do atuador foi feito um teste onde foi imposto uma entrada senoidal discreta através do conversor digital D/A pelo Simulink, este sinal analógico gerado foi aplicado no atuador de modo físico por cabos tipo BNC e ao mesmo tempo foi adquirido este sinal pelo sistema de aquisição da Instron pela entrada auxiliar. O atuador respondeu ao sinal aplicado se movimentando e o seu sinal de deslocamento também foi adquirido pelo sistema de aquisição do Instron e ambos os sinais foram armazenados em um arquivo.

Os sinais salvos em um arquivo foram comparados e mostrados na figura 3.16 (a), Para se visualizar o atraso da resposta mostra-se uma ampliação dos sinais obtidos em um intervalo de tempo de amostragem conforme apresenta a figura 3.16(b), Observa-se que o sinal medido apresenta um atraso devido ao comportamento do atuador, cujo valor obtido foi de 2.6 ms.

Este atraso foi medido para se saber qual era o tempo mínimo de amostragem que poderia ser adotado nas simulações do sistema (massa, amortecedor e mola física) de modo discreto para se obter a resposta correta.



a)



b)

Figura 3.16 Tempo de atraso do sistema Instron
a) Respostas b) Detalhamento dos sinais

CAPITULO 4

CORREÇÃO DO HIL PARA TEMPO REAL

4.1. Introdução

Para se descrever o funcionamento em tempo real do sistema modelado através do conceito do HIL com atraso e sem atraso é preciso primeiramente entender o completo funcionamento da placa de aquisição DSPACE em tempo real que será descrito a seguir.

A placa DSPACE 1102 possui um sinal quadrado chamado de “*clock*” cuja frequência é escolhida pelo usuário conforme mostra a figura 4.1.

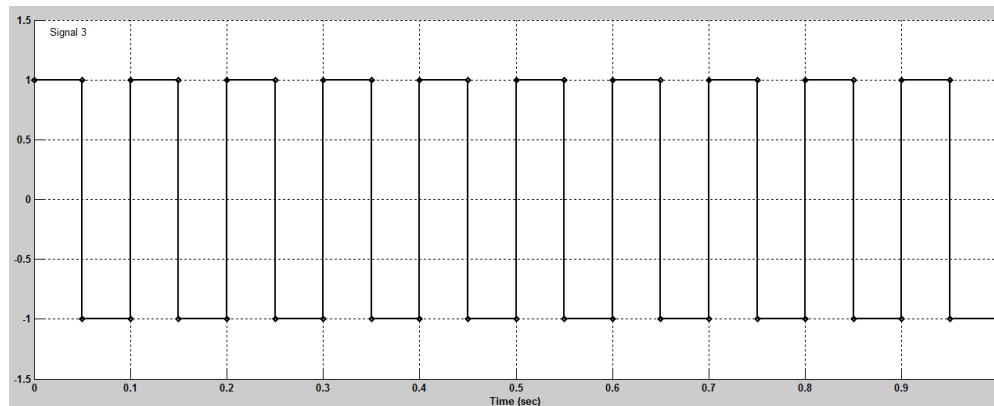


Figura 4.1 Sinal de “*Clock*” da placa de aquisição DSPACE

Cada transição de subida deste sinal é responsável em disparar a execução de várias tarefas e ao mesmo tempo de checar se antes de outra transição de subida do sinal todas as operações já foram concluídas. Se todas as operações não conseguiram ser concluídas antes da próxima transição de subida do sinal, uma mensagem será enviada ao usuário informando que não é possível realizar o processamento desejado em tempo real.

Este sinal de “*Clock*”, portanto, é responsável em disparar várias tarefas em *Hardware* e em *Software* que estão apresentadas na Tabela 4.1. Dentre as tarefas de *Hardware* podemos

destacar a tarefa de disparar os conversores A/D e D/A. Ao mesmo tempo em que a placa esta executando suas tarefas de *Hardware*, podem ser executadas tarefas de *Software* como a de alterar alguns de seus parâmetros ou mesmo de ler alguns dos seus parâmetros que estão disponíveis na placa. Por exemplo, pode-se alterar o valor da variável na memória do conversor D/A na placa de aquisição a qualquer instante, entretanto somente no próximo sinal de transição de subida do “*clock*” que virá após a alteração executada é que irá disparar o conversor e transforma-lo em uma saída de valor analógico correspondente. Outro exemplo é a possível leitura do valor digital disponível que foi convertido pelo conversor A/D e armazenado na memória da placa pelo último sinal de transição de subida do “*clock*” que disparou a conversão.

Tabela 4.1. Descrição dos processos

PROCESSO	ELEMENTO USADO	FUNÇÕES
A	Conversor A/D Dspace - Instron	<ul style="list-style-type: none"> • Ativação através de um disparo no início de um sinal de “clock” • Tarefa de <i>hardware</i>
B	Conversor D/A Dspace - Instron	<ul style="list-style-type: none"> • Ativação através de um disparo no início de um sinal de “clock” • Tarefa de <i>hardware</i>
C	Simulink – Matlab	<ul style="list-style-type: none"> • Operações matemáticas antes da escrita do CDA • Tarefa de software
D	Conversor D/A Dspace - Computador	<ul style="list-style-type: none"> • Escrita da variável do CDA • Tarefa de software
E	Conversor A/D Dspace - Computador	<ul style="list-style-type: none"> • Leitura da variável do CAD • Tarefa de software
F	Simulink – Matlab	<ul style="list-style-type: none"> • Operações matemáticas depois da leitura do CAD • Tarefa de software

Conforme mostrado na Tabela 4.1, o processo A corresponde a tarefa de *Hardware* de disparo da conversão A/D, o processo B corresponde à tarefa de *Hardware* de disparo da conversão D/A, o processo C corresponde às operações de Software que foram realizadas antes da operação de escrita do valor a ser utilizado pelo conversor D/A, o processo D corresponde à operação de Software de escrita na variável correspondente a ser utilizada pelo conversor D/A, o processo E corresponde à operação de Software de leitura da variável correspondente a utilizada pelo conversor A/D e o processo F corresponde às operações de Software que foram realizadas depois da leitura do valor convertido pelo conversor A/D no software.

Podemos ilustrar o funcionamento da placa DSPACE 1102 em tempo real sendo utilizada com a técnica HIL através das suas tarefas básicas mostradas na figura 4.2.

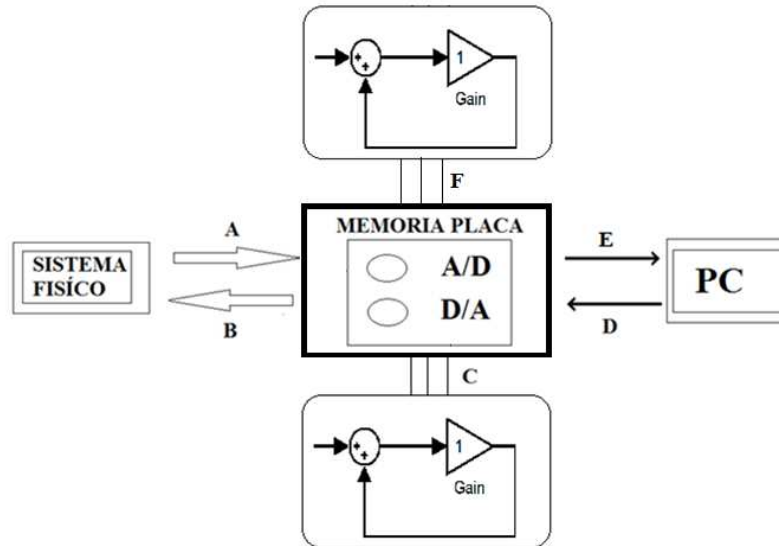


Figura 4.2 Operações da placa de aquisição DSPACE

Conforme descrito anteriormente, todos estes processos devem ser executados em um período de tempo do sinal de “clock” podendo ser visualizados em uma sequência continua de execução mostrada em tempo real na figura 4.3, onde os processos A e B ocorrem no início do “clock”, seguidos das operações antes do conversor D/A no processo C, continua com a escrita na memória do conversor D/A na placa no processo D, seguido da leitura da memória da placa do conversor A/D no processo E, e para terminar calculando as operações depois do conversor A/D no processo F, continuando assim sucessivamente e repetindo todo o procedimento descrito para os próximos sinais de transição de subida do “clock”.

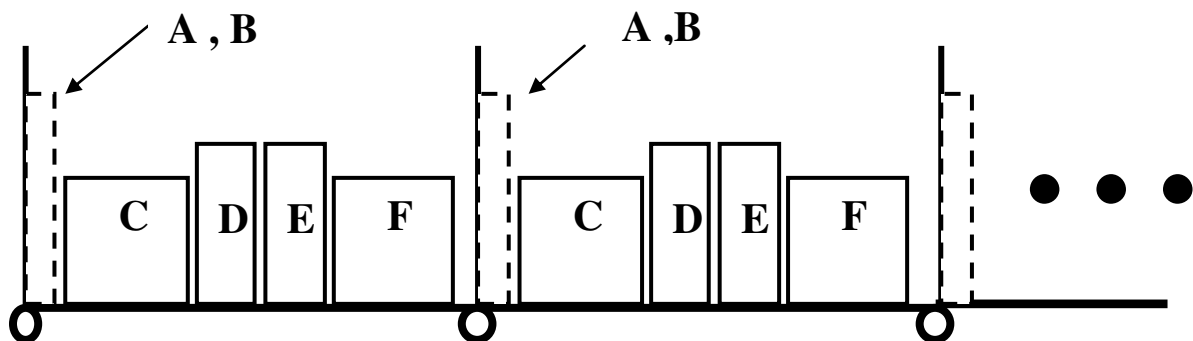


Figura 4.3 Sequência de operações em tempo real da placa de aquisição DSPACE

4.2. Modelo HIL - Real no Simulink

O diagrama de blocos básico do sistema modelado no Simulink através do conceito do HIL é mostrado na figura 4.4, onde se tem uma entrada “11” de excitação em força (KN) somada com um valor fixo “12” correspondente a uma força peso constante e com um valor de força “F” obtida inicialmente pela leitura “7” do conversor A/D e depois processada pelas operações “8”, “9” e “10” correspondente a sua etapa de calibração (KN). O valor de saída do somador “13” é multiplicado por um ganho para realizar a conversão de KiloNewton (KN) para Newton(N) obtendo-se a entrada “u” do sistema correspondente a modelagem de estado discreto. Este valor de “u” será utilizado para se calcular a tarefa de software “15” correspondente a próxima solução de estado “x” da equação dinâmica. A resposta em deslocamento “y” em metros obtida da equação de saída “1” é convertida para milímetros pelo ganho “2” e depois é subtraído de um valor constante “3” correspondente ao *SetPoint* (mm) da montagem física. O valor de saída do somador “4” em milímetros é convertido para o valor em volts pelo ganho “5” e finalmente este valor é escrito por “6” na memória do conversor D/A. Todos estes blocos ligados sequencialmente pelas suas setas correspondem ao modelo esquemático do HIL.

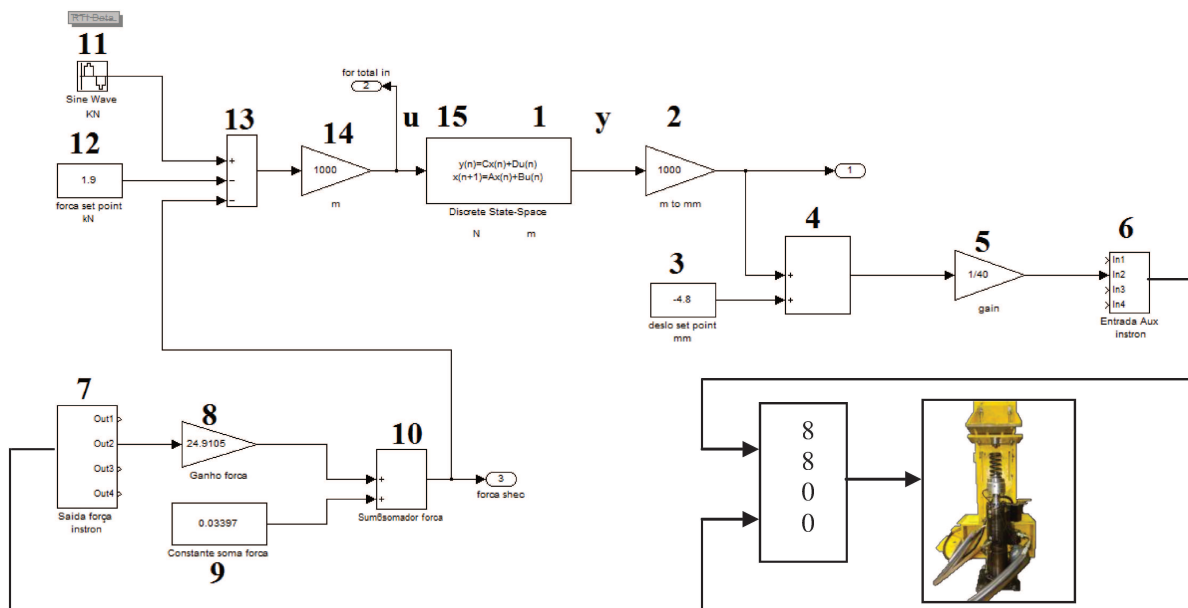


Figura 4.4 Modelo HIL-Real no simulink

4.3. Modelo HIL - Simulado no Simulink

Para poder simular e visualizar a resposta esperada do sistema completo ao se aplicar a técnica de HIL mostrado na figura 4.4, vamos substituir o *Hardware* da técnica de HIL por um modelo matemático cujas entradas e saídas são as mesmas.

O diagrama de blocos básico do sistema simulado modelado no Simulink através do conceito do HIL é mostrado na figura 4.5. Comparando as figuras 4.4 e 4.5, observa-se que as tarefas 6, 7, 8, 9 e 10 da figura 4.4 respectivo a parte de *Hardware* do HIL foram substituídas por uma modelagem em Software da força segundo os parâmetros de calibração da mola física na figura 4.5 cuja saída é em KN e a entrada em Volts dado pelas novas tarefas 6, 7, 8a, 8b, 9a, 9b, 9c, 9d e 10.

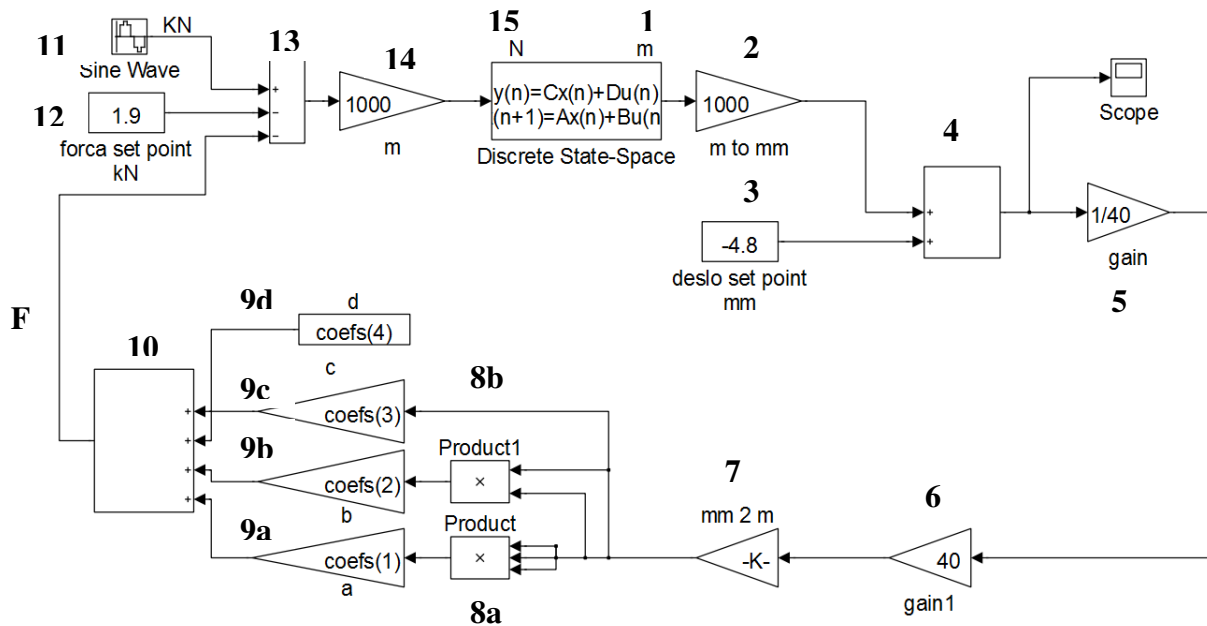


Figura 4.5 Modelo HIL-Simulado no simulink

4.4. Operações do Modelo HIL - Simulado

Conhecido o modelo matemático de estado discreto do sistema a ser modelado com a técnica do HIL representado pelas equações 4.1 e 4.2,

$$x(k+1) = Ax(k) + Bu(k) \quad \text{Equação 4.1}$$

$$y(k) = Cx(k) + Du(k) \quad \text{Equação 4.2}$$

pode-se descrever a sequência de funcionamento das operações do modelo HIL simulado da figura 4.5 em tempo real.

Existem duas sequências de implementações possíveis que descrevem o funcionamento das operações. Vamos descrever as duas sequências.

A primeira sequência apresentada na figura 4.6 mostra na parte inferior a sequência de tarefas executadas em cada intervalo de tempo em tempo real e na parte superior o resultados dos sinais obtidos, ambos serão descritos detalhadamente a seguir. No início do instante de tempo “k+1”, têm-se os dados correspondentes à entrada $u(k)$, deslocamento $y(k)$ e o vetor de estados $x(k)$. Em seguida com os valores conhecidos do passo anterior de $x(k)$ e de $u(k)$ se executa a tarefa de Software “15” correspondente ao cálculo do vetor de estados $x(k+1)$ dado pela equação 4.1. Com o valor do deslocamento do passo anterior $y(k)$ são realizadas operações correspondentes de “2” até “10” os quais transformam o sinal de saída de deslocamentos $y(k)$ em metros para a força $F(k+1)$ em KN. Com o valor obtido da força são executadas várias operações desde a tarefa “11” até a “14”, obtendo-se no final o dado de entrada do sistema que é a força total $u(k+1)$. Com este valor de $u(k+1)$ e com o valor calculado de $x(k+1)$ calcula-se a tarefa de software “1” correspondente à operação de saída do sistema discreto dado pela equação 4.2 obtendo-se o deslocamento $y(k+1)$. Terminado o passo “k+1” se inicia todo o processo para o próximo instante “k+2” nas mesmas condições descritas e assim sucessivamente.

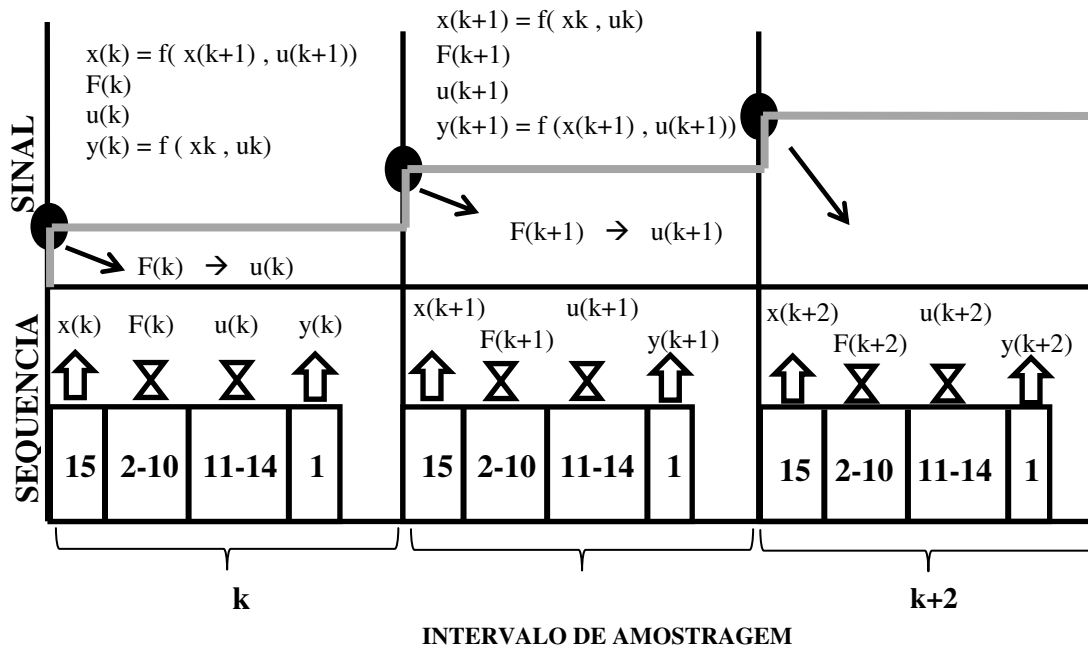


Figura 4.6 Modelo HIL-Simulado - Sequência 1

A segunda sequência apresentada na figura 4.7 mostra na parte inferior a sequência de tarefas executadas em cada intervalo de tempo em tempo real e na parte superior o resultados dos sinais obtidos, ambos serão descritos detalhadamente a seguir. No início do instante de tempo “k+1”, têm-se os dados correspondentes à entrada $u(k)$, o deslocamento $y(k)$ e o vetor de estados $x(k+1)$. Com o valor do deslocamento calculado no passo anterior $y(k)$ são realizadas operações correspondentes de “2” até “10” os quais transformam o sinal de saída de deslocamentos $y(k)$ em metros para a força $F(k+1)$ em KN. Com o valor obtido da força são executadas várias operações desde a tarefa “11” até a “14”, obtendo-se no final o dado de entrada do sistema que é a força total $u(k+1)$. Com este valor de $u(k+1)$ e com o valor calculado no passo anterior de $x(k+1)$ calcula-se a tarefa de software “1” correspondente à operação de saída do sistema discreto dado pela equação 4.2 obtendo-se o deslocamento $y(k+1)$. Em seguida com o valor conhecido do passo anterior de $x(k+1)$ e com o valor calculado de $u(k+1)$ se executa a tarefa de Software “15” correspondente ao cálculo do próximo vetor de estados $x(k+2)$ dado pela equação 4.1. Terminado o passo “k+1” se inicia todo o processo para o próximo instante “k+2” nas mesmas condições descritas e assim sucessivamente.

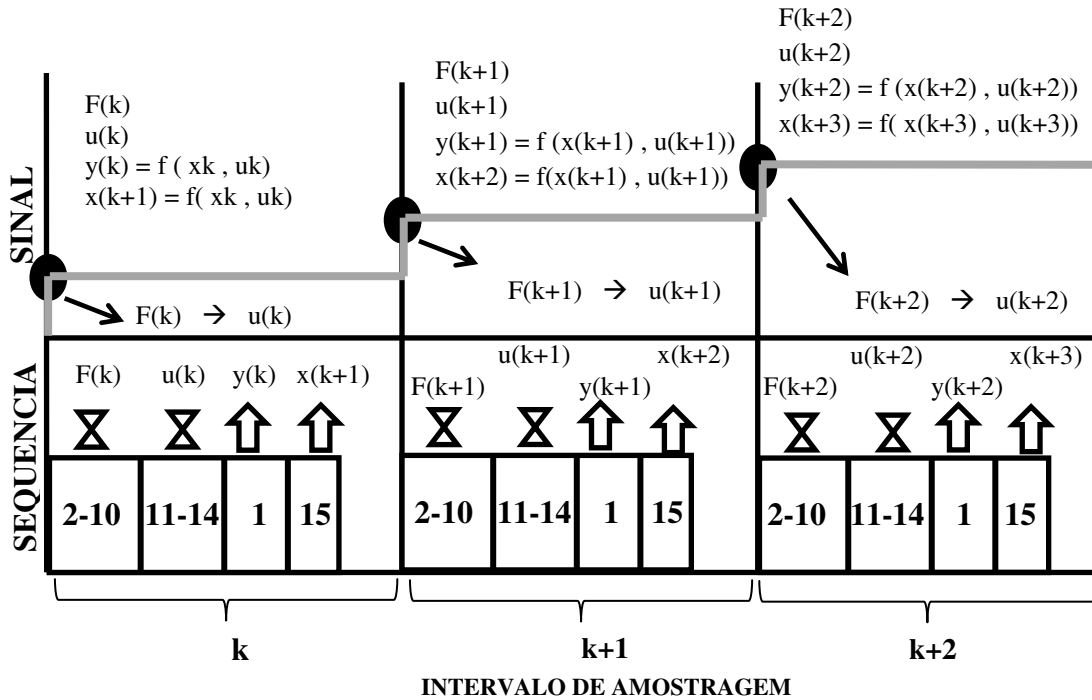


Figura 4.7 Modelo HIL-Simulado - Sequência 2

4.5. Operações do HIL em Tempo Real (Ideal desejado)

Conhecido o funcionamento da placa de aquisição, o modelo matemático de estado discreto do sistema a ser modelado com a técnica do HIL representado pelas equações 4.1 e 4.2 e as sequências de funcionamento do Modelo Hil Simulado em tempo real das figuras 4.6 e 4.7, pode-se descrever as duas sequências esperadas de funcionamento ideal do modelo HIL em tempo real.

A primeira sequência apresentada na figura 4.8 mostra na parte inferior a sequência de tarefas executadas em cada intervalo de amostragem em tempo real e na parte superior o resultados dos sinais obtidos, ambos serão descritos detalhadamente a seguir. Ao se receber uma subida do sinal de “clock” correspondente ao início do instante de tempo “ $k+1$ ”, os processos de *Hardware* “A e B” são disparados correspondentes às tarefas de conversão (D/A e A/D), ou seja, convertendo de D/A o valor do deslocamento $y(k)$ imposto ao sistema físico calculado no passo

anterior “k” e convertendo de A/D o valor lido da força devido ao deslocamento que acabou de ser imposto e que será a força $F(k+1)$. Em seguida com os valores conhecidos do passo anterior $x(k)$ e de $u(k)$ se executa a tarefa de Software “15” correspondente ao vetor de estados $x(k+1)$ dado pela equação 4.1. Na sequência se executa a tarefa de Software “7” sendo esta a leitura da memória do valor da força $F(k+1)$ convertido pelo conversor A/D. Com o valor obtido da força $F(k+1)$ são executadas várias operações desde a tarefa “8” até a “14”, obtendo-se no final o dado de entrada do sistema que é a força total $u(k+1)$. Em seguida com o valores atuais calculados de $x(k+1)$ e de $u(k+1)$ se executa a tarefa de Software “1” correspondente à operação de saída do sistema discreto dado pela equação 4.2 obtendo-se o deslocamento $y(k+1)$ a ser imposto no próximo instante de tempo $(k+2)$. Com o valor do deslocamento calculado $y(k+1)$ são realizadas operações correspondentes de “2” até o “5” os quais transformam o sinal de saída de deslocamentos $y(k+1)$ de metros para volts. Este deslocamento agora em volts passa pela tarefa de Software “6” correspondente ao salvamento deste dado na memória da placa do conversor D/A.

Na parte superior da figura 4.8, de acordo com a modelagem discreta do sistema no espaço de estados, no instante de amostragem “k+1”, portanto calculou-se o vetor de estados $x(k+1)$ e para isto foi necessário ter os valores de $x(k)$ e $u(k)$, sabendo que o valor de $u(k)$ depende do valor de $F(k)$ e que por sua vez depende do valor de deslocamento imposto $y(k-1)$.

Para o caso do sistema que responde idealmente, no início do instante “k+1” com o sinal de transição de subida do “clock” são disparados os conversores D/A e A/D, e por tanto com isso impõe-se o valor de deslocamento $y(k)$ através do conversor D/A. Instantaneamente o sistema responde com o valor de $F(k+1)$ e este valor é obtido através do conversor A/D. Na sequência calcula-se o valor de $x(k+1)$, $u(k+1)$ e finalmente valor de $y(k+1)$ e assim sucessivamente repetindo todo o processo no próximo instante “k+2” nas mesmas condições descritas.

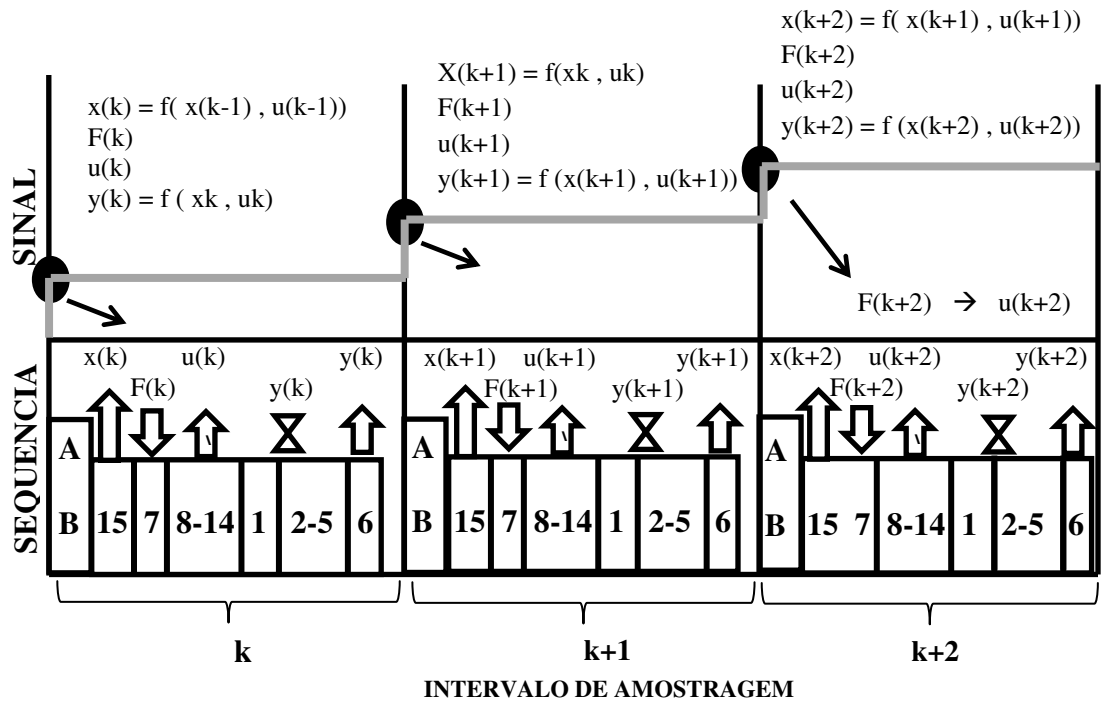


Figura 4.8 Modelo com sistema de atuação ideal - Sequência 1

A segunda sequência é apresentada na figura 4.9 que apresenta dois gráficos, na parte inferior a sequência de tarefas executadas em cada intervalo de amostragem em tempo real e na parte superior o resultados dos sinais obtidos, ambos serão descritos detalhadamente a seguir. Ao se receber uma subida do sinal de “clock” correspondente ao início do instante de tempo “k+1”, os processos de *Hardware* “A e B” são disparados correspondentes às tarefas de conversão (D/A e A/D), ou seja, convertendo de D/A o valor do deslocamento $y(k)$ imposto ao sistema físico calculado no passo anterior “k” e convertendo de A/D o valor lido da força devido ao deslocamento que acabou de ser imposto e que será a força $F(k+1)$. Na sequência se executa a tarefa de Software “7” sendo esta a leitura da memória do valor da força $F(k+1)$ convertido pelo conversor A/D. Com o valor obtido da força $F(k+1)$ são executadas várias operações desde a tarefa “8” até a “14”, obtendo-se no final o dado de entrada do sistema que é a força total $u(k+1)$. Em seguida com o valor conhecido do passo anterior $x(k+1)$ e com o valor atual de $u(k+1)$ se executa a tarefa de Software “1” correspondente à operação de saída do sistema discreto dado pela equação 4.2 obtendo-se o deslocamento $y(k+1)$ a ser imposto no próximo instante de tempo

(k+2). Com o valor do deslocamento imposto $y(k+1)$ são realizadas operações correspondentes de “2” até o “5” os quais transformam o sinal de saída de deslocamentos $y(k+1)$ de metros para volts. Este deslocamento agora em volts passa pela tarefa de Software “6” correspondente ao salvamento deste dado na memória da placa do conversor D/A. Na sequência também com os mesmos valores conhecidos $x(k+1)$ e $u(k+1)$ se executa a tarefa de Software “15” correspondente ao cálculo do vetor de estados $x(k+2)$ dado pela equação 4.1.

Na parte superior da figura 4.9, de acordo com a modelagem discreta do sistema no espaço de estados, no instante de amostragem “k+1”, portanto calculou-se o vetor de estados $x(k+2)$ e para isto foi necessário se ter os valores de $x(k+1)$ e $u(k+1)$, sabendo-se que o valor de $u(k+1)$ depende do valor de $F(k+1)$ e que por sua vez depende do valor de deslocamento imposto $y(k)$.

Para o caso do sistema de atuação que responde idealmente, no início do instante “k+1” com o sinal de transição de subida do “clock” são disparados os conversores D/A e A/D, e portanto com isso impõe-se o valor de deslocamento $y(k)$ através do conversor D/A. Instantaneamente o sistema de atuação responde com o valor de $F(k+1)$ e este valor é obtido através do conversor A/D. Na sequência calcula-se o valor de $u(k+1)$, $y(k+1)$ e finalmente valor de $x(k+2)$ e assim sucessivamente repetindo todo o processo no próximo instante “k+2” nas mesmas condições descritas.

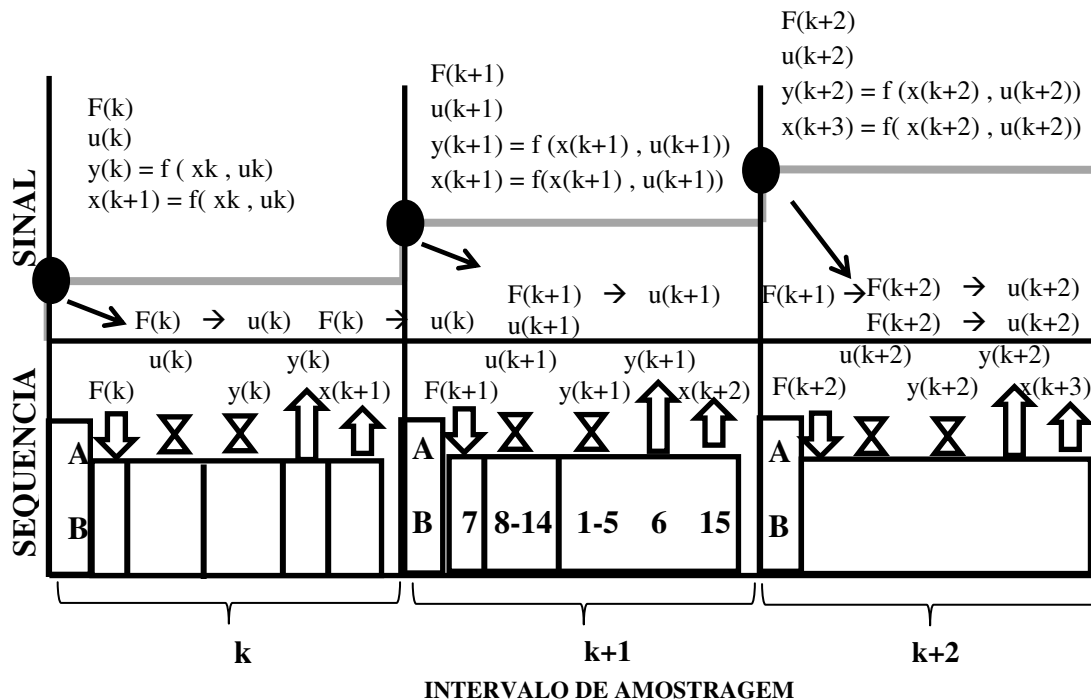


Figura 4.9 Modelo com sistema de atuação ideal - Sequência 2

Observa-se por tanto que ambas as sequências apresentam um ponto fundamental de necessitarem no início de cada instante de tempo do valor correto de $F(k+1)$ que é obtido através do conversor A/D.

4.6. Operações do HIL em Tempo Real (com atraso obtido)

Conhecido o funcionamento esperado do modelo HIL cujo sistema de atuação responde idealmente como mostrado nas sequências das figuras 4.8 e 4.9, vamos apresentar as diferenças para as duas sequências do sistema de atuação real respondendo em tempo real, e também qual sequência que está implementada pela DSPACE em seus geradores automáticos de códigos.

A primeira sequência é mostrada na figura 4.10 e a segunda sequência, que em particular também corresponde ao código gerado automaticamente pela DSPACE, é mostrada na figura 4.11. Ambas as figuras apresentam na parte inferior a sequência de tarefas executadas em cada

intervalo de amostragem em tempo real e na parte superior os resultados dos sinais obtidos os quais já foram descritas detalhadamente na seção 4.5.

Na parte superior da figura 4.10 correspondente a sequência 1, de acordo com a modelagem discreta do sistema no espaço de estados, no instante de amostragem “ $k+1$ ”, calculou-se o vetor de estados $x(k+1)$ e o valor de $y(k+1)$ e para isto foi necessário já se ter disponível os valores de $x(k)$, $u(k)$ e $u(k+1)$, sabendo que o valor de $u(k+1)$ depende do valor de $F(k+1)$ que por sua vez depende do valor de deslocamento imposto $y(k)$.

Na parte superior da figura 4.11 correspondente a sequência 2, de acordo com a modelagem discreta do sistema no espaço de estados, no instante de amostragem “ $k+1$ ”, calculou-se o próximo vetor de estados $x(k+2)$ e o valor de $y(k+1)$ e para isto foi necessário já se ter disponível os valores de $x(k+1)$ e $u(k+1)$, sabendo que o valor de $u(k+1)$ depende do valor de $F(k+1)$ que por sua vez depende do valor de deslocamento imposto $y(k)$.

Para o caso do sistema atual ser o real e que portanto não responde idealmente pode-se observar o resultado do sinal a ser obtido como mostrado na figura 4.10 e 4.11. Ambas, no início do instante “ $k+1$ ” com o sinal de transição de subida do “*clock*” são disparados os conversores D/A e A/D, e portanto com isso se impõe o valor de deslocamento $y(k)$ através do conversor D/A. Entretanto o sistema de atuação não responde instantaneamente com o valor de $F(k+1)$ desejado e o valor que é obtido através do conversor A/D é o valor atrasado de $F(k+1)$, ou seja, o valor de $F(k)$, calculando-se com isso um valor de $u(k+1)$ diferente do esperado, pois ao invés de se utilizar o valor correto $u(k+1)$ utilizou-se o valor atrasado $u(k)$ e assim sucessivamente repetindo todo o processo no instante “ $k+2$ ” nas mesmas condições descritas.

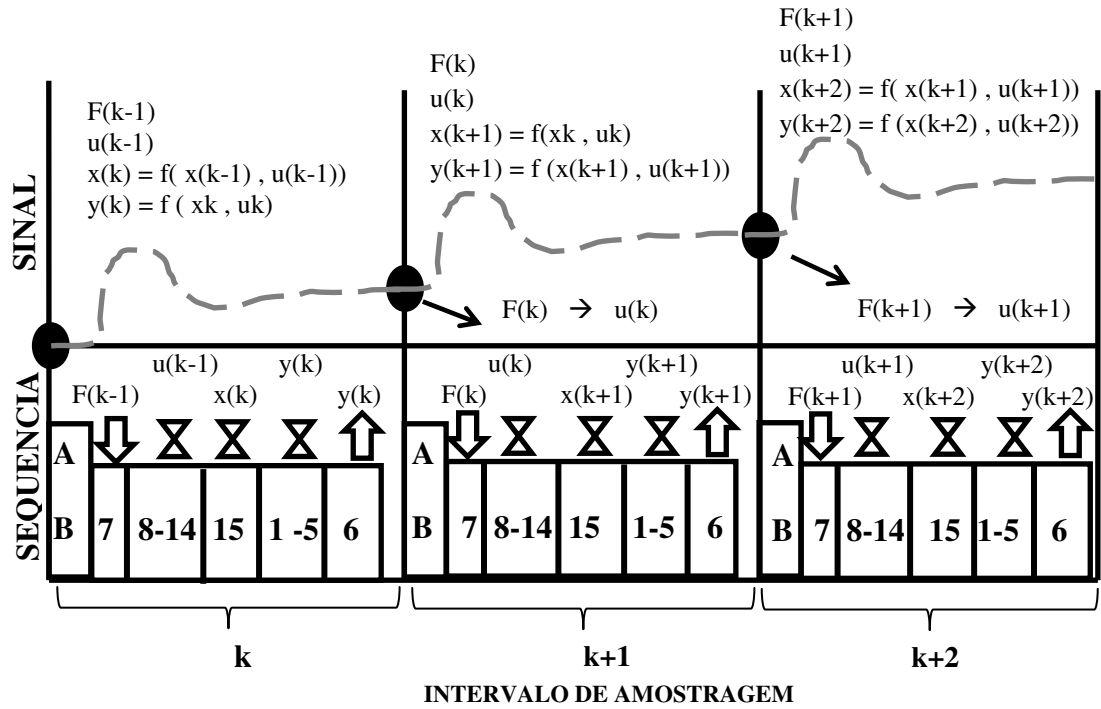


Figura 4.10 Modelo com atraso obtido na sequência 1

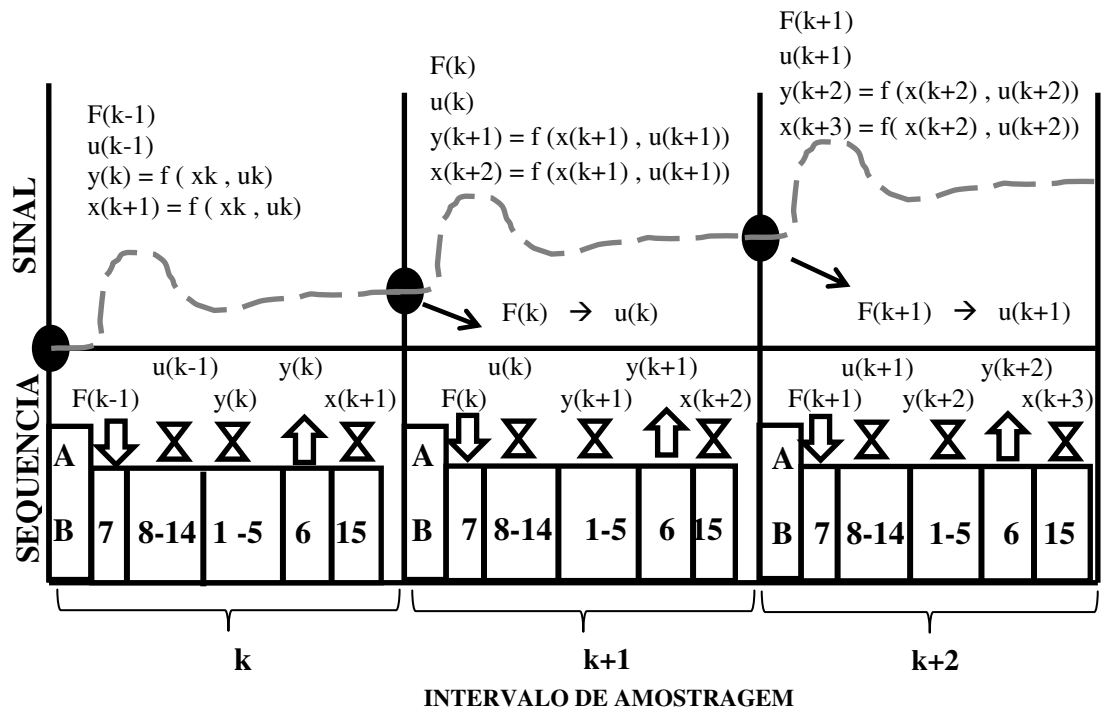


Figura 4.11 Modelo com atraso obtido na sequência 2

Com esta análise conclui-se que o sistema sendo analisado o qual foi modelado não é mais o apresentado na figura 4.5 e sim um novo modelo com um atraso na força dado pelo processo “16” conforme mostrado na figura 4.12.

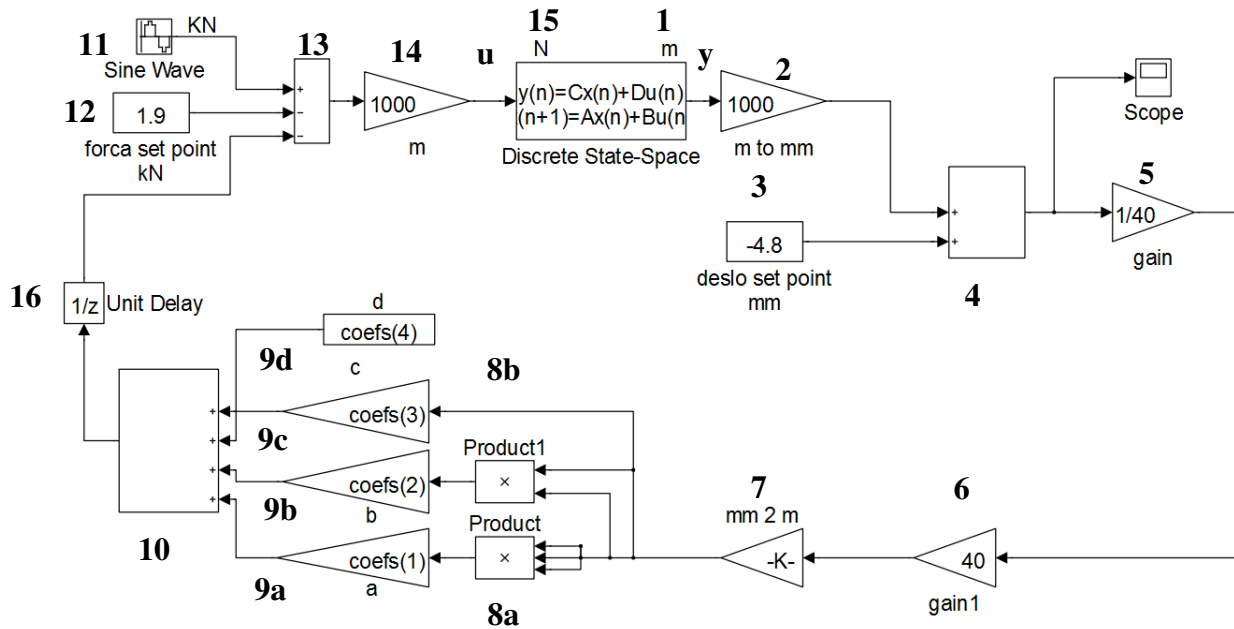


Figura 4.12 Modelo simulado com atraso

4.7. Operação do HIL em Tempo Real (proposta 1 sem atraso)

Estudando as sequências do modelo em atraso apresentado na seção 4.6, observa-se que o atraso na primeira sequência aparece devido ao fato de que no instante “k+1” o cálculo de $y(k+1)$, apesar deste calculo depender de $x(k+1)$ e $u(k+1)$, ele esta sendo executado com o valor $x(k+1)$ e com o valor atrasado de $u(k+1)$, e portanto com o valor de $u(k)$. Entretanto para os casos onde a matriz D do modelo de estado é zero, $y(k+1)$ vai depender apenas de $x(k+1)$ como pode ser visto na equação 4.2. Neste caso esta sequência com $D=0$ não apresentará defasagem.

Portanto a sequência de funcionamento desta proposta 1 está apresentado no gráfico inferior da figura 4.13. No instante de amostragem “k+1”, calcula-se inicialmente o vetor de estados desse instante de tempo $x(k+1)$ através primeiro da leitura do valor de $F(k)$ obtido através do disparo do conversor A/D no início do instante (k+1) realizado com o sinal de transição de subida do *clock*. Obtido o valor correto de $F(k)$ deve-se calcular o valor correto de $u(k)$ e consequentemente o valor desejado de $x(k+1)$. Com o valor calculado de $x(k+1)$ se executam as tarefas de Software de saída “1” do sistema discreto dado pela equação 4.1 obtendo-se o deslocamento $y(k+1)$ correto a ser imposto sem atraso.

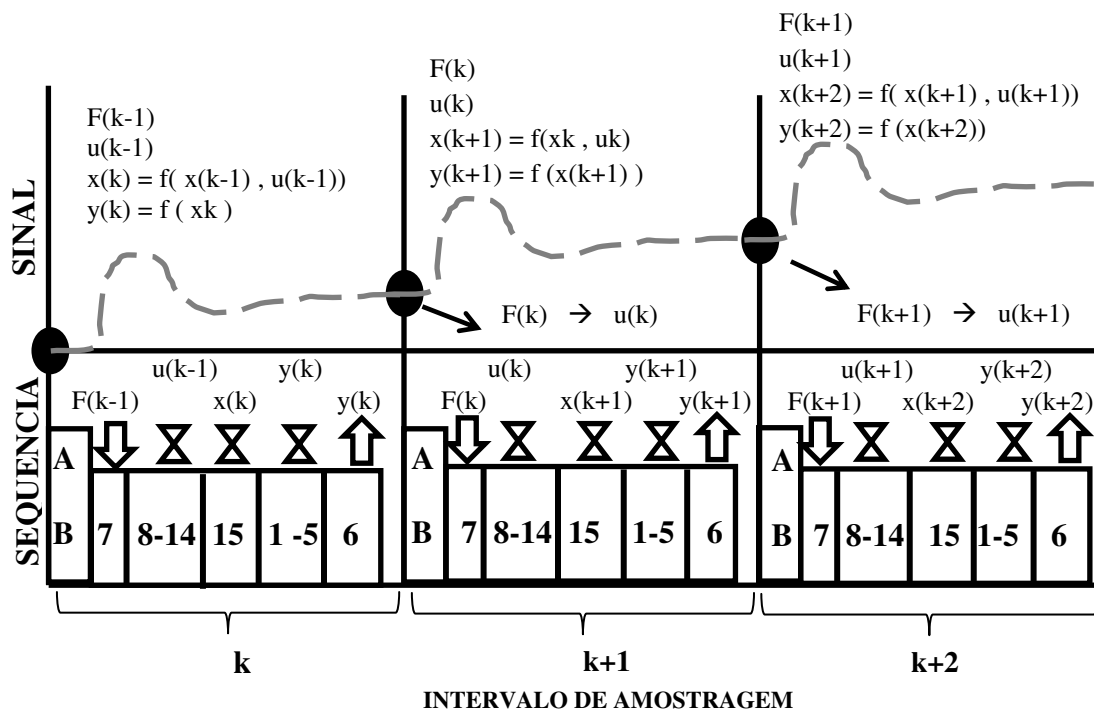


Figura 4.13 Modelo sem atraso proposta 1

Para se conseguir implementar a proposta 1 diretamente no ambiente Simulink realizaram-se inicialmente vários estudos de alterações dos parâmetros das ferramentas internas de cada bloco sendo utilizado no Simulink, bem como da possibilidade de se mudar a sequência de execução segundo uma prioridade desejada. Apesar destas tentativas não se conseguiu obter resultados satisfatórios, devido às limitações da versão do software sendo utilizado. Por ultimo para conseguir implementar a proposta e testa-la, decidiu-se em alterar diretamente o código C

gerado pelo MATLAB e posteriormente carrega-lo na placa de aquisição, obtendo-se assim o resultado desejado.

4.8. Operação do HIL em Tempo Real (proposta 2 sem atraso)

Outra proposta elaborada para eliminar o efeito de atraso está apresentada na figura 4.14 e será detalhada a seguir.

Esta nova proposta consiste em mudar a taxa de amostragem de algumas das sequências do código gerado automaticamente pela DSPACE, ou seja, trabalha-se com duas taxas de amostragem no sistema, uma delas dez vezes mais rápida que a outra taxa.

Para melhor descrever e visualizar o comportamento desta proposta apresentada na figura 4.14, considera-se duas taxas de amostragem no sistema, onde conforme mencionado somente para efeito de visualização adotou-se a primeira taxa de 0.1 s, e a segunda dez vezes menor, de 0.01s. Neste caso, após definidas as duas taxas de amostragem, escolhem-se as operações que deverão ser executadas somente com a primeira taxa de amostragem de 0.1s, e todas as outras serão executadas na segunda taxa de amostragem de 0.01s. Portanto foram escolhidas as tarefas (B, C e D) para serem executadas somente com a primeira taxa de amostragem de 0.1s e atribuiu-se às outras tarefas (A, E e F) a segunda taxa de amostragem, resultando na execução das operações (A, B, C, D, E e F) a cada 0.1s e das operações (A,E,F) a cada 0.01s conforme mostrado na sequência infinita da figura 4.14.

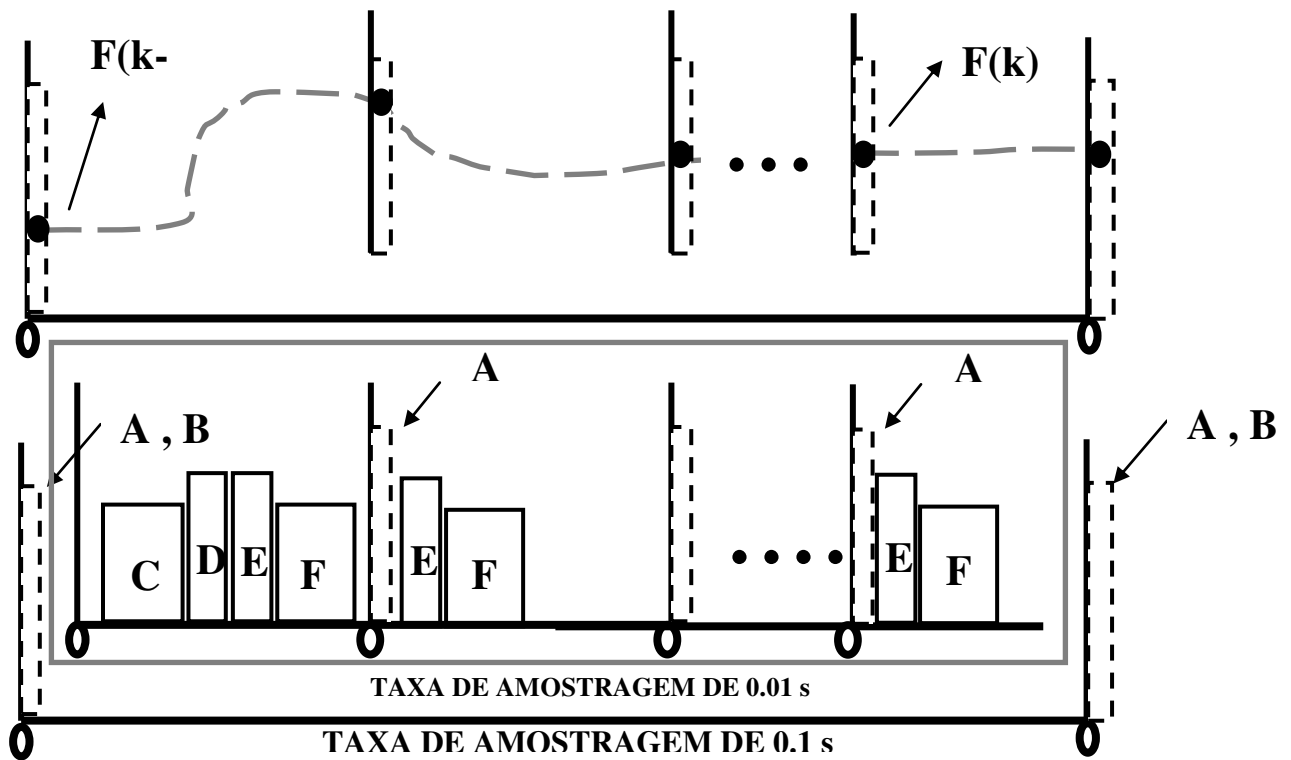


Figura 4.14 Sequência dos processos proposta 2

Observa-se da figura 4.14 que o instante k corresponde sempre a maior taxa de amostragem. Com esta sequência consegue-se obter o valor correto de $F(k)$ dentro do instante de tempo (k) e consequentemente consegue-se obter o valor correto de $x(k+1)$ também calculado dentro do instante (k) . Nesta sequência observa-se que os valores de $F(k)$ e $x(k+1)$ serão recalculados dez vezes e somente o último valor calculado é armazenado e corresponde ao valor correto esperado de $F(k)$ e consequentemente de $y(k)$ e $x(k+1)$. Observa-se também que neste caso não existe limitação nenhuma a matriz D na equação 4.1.

Novamente para se conseguir implementar a proposta 2 diretamente no ambiente Simulink realizaram-se os estudos de alterações dos parâmetros das ferramentas internas de cada bloco sendo utilizado no Simulink, bem como da possibilidade de se mudar a sequência de execução segundo a prioridade desejada. Novamente, apesar destas tentativas, não se conseguiu obter resultados satisfatórios, devido às mesmas limitações da versão do software. Novamente para conseguir implementar a proposta e testa-la, decidiu-se em alterar diretamente o código C gerado pelo MATLAB e posteriormente carrega-lo na placa de aquisição obtendo assim o resultado desejado.

CAPITULO 5

SIMULAÇÃO NUMÉRICA

5.1. Introdução

Neste capítulo serão apresentadas as simulações realizadas aplicando-se a técnica de HIL. Escolheram-se dois tipos de sistemas para demonstrar o efeito negativo do atraso no comportamento dinâmico do sistema. No primeiro sistema modelado os parâmetros do sistema foram escolhidos para mostrar o efeito do atraso onde na simulação com HIL e com atraso resulta em um sistema instável e por outro lado onde na simulação com HIL e sem atraso obtêm-se o comportamento estável esperado com a técnica. O segundo sistema modelado apesar dos parâmetros escolhidos resultarem em um sistema estável na simulação com e sem atraso na técnica de HIL, ele mostra que o comportamento da resposta transiente do sistema com HIL e com atraso é completamente diferente do comportamento da resposta transiente do sistema com HIL e sem atraso.

Na modelagem utilizada nestes dois sistemas foram incluídas todas as considerações apresentadas no capítulo 3 como, por exemplo, as respectivas modelagens de calibrações realizadas, o referencial adotado bem como a modelagem da mola que irá representar o comportamento da mola física para se antecipar as respostas a serem obtidas nos testes experimentais que foram realizados e apresentados no capítulo 6.

5.2. Modelo1 – HIL

Adotou-se neste modelo o mesmo referencial absoluto do atuador e a respectiva posição inicial onde se inicia a reação da mola e o curso máximo e mínimo do atuador, ou seja, o modelo apresenta um “*setpoint*” igual ao do atuador de -4.8 mm e uma amplitude de deslocamento máxima do cilindro de 40 mm, além de apresentar como parâmetros físicos as constantes da massa de 200 Kg, do amortecimento de 11.2 KN*s/m, da sua força peso constante de 1.0 KN e adotar a resposta em relação à posição inicial do “*setpoint*” correspondente a posição da mola não deformada e não do equilíbrio estático.

No software Simulink foram utilizados como tempo total de simulação o valor de 10 s, com uma taxa de amostragem tipo “single task” de 0.1 s e como método de integração o ODE 4 (Runge-Kutta).

Para o sinal de excitação foi adotado como um sinal tipo senoidal com amplitude 0.1 KN de zero a pico e com uma frequência de 2 rad/s.

Na simulação obtém-se a resposta do deslocamento da massa suspensa onde se espera que devido ao sistema ser referenciado a partir da posição inicial da mola não deformada, com condições iniciais nulas, com uma entrada harmônica e uma força constante, a resposta permanente da massa deverá oscilar na frequência da força harmônica de excitação em torno da posição de equilíbrio estático.

5.2.1. Modelo1 HIL –Simulado - Com Atraso - Instável

O diagrama de blocos desenvolvido para simular o modelo1 HIL com atraso é apresentado na figura 5.1. A sequência de simulação é a mesma apresentada na seção 4.6.

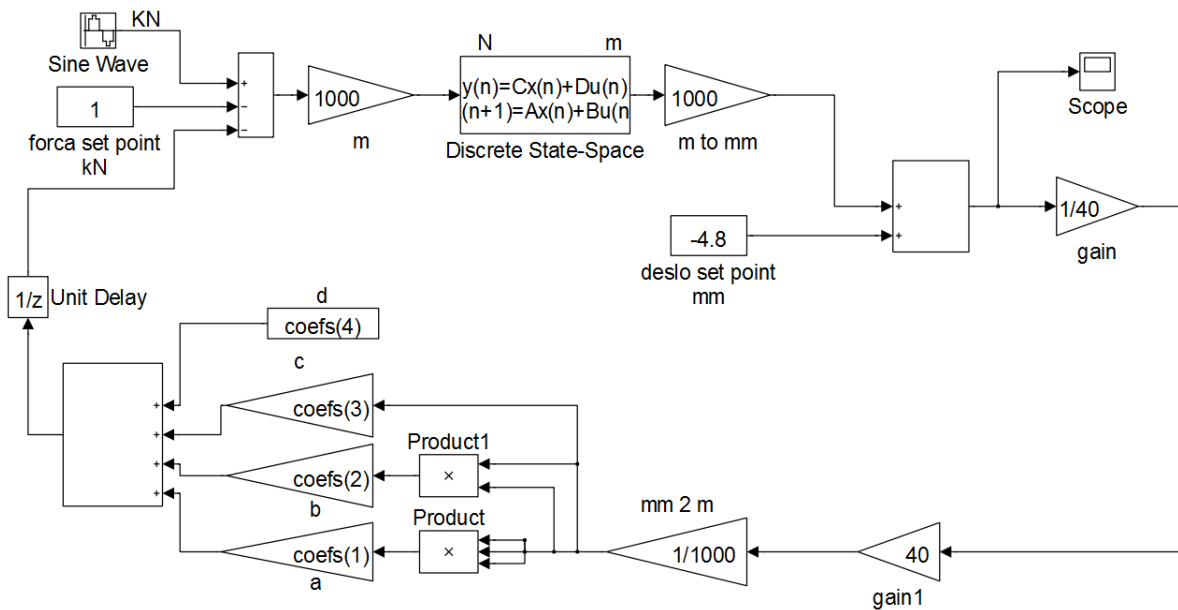
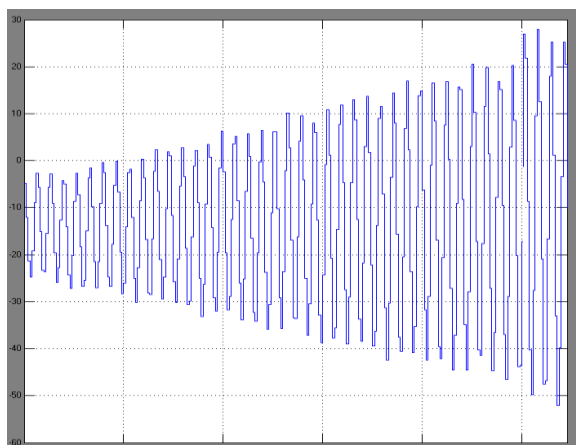
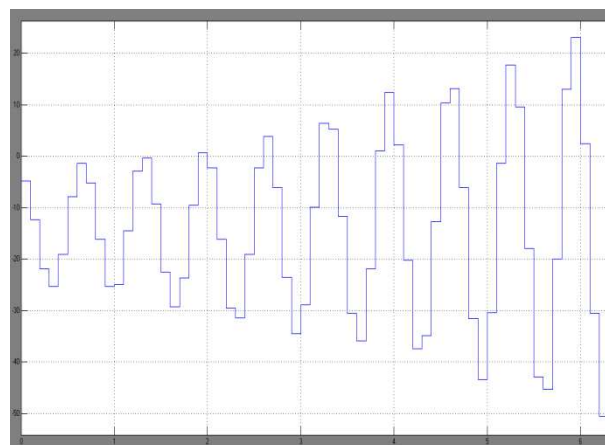


Figura 5.1 Diagrama de blocos do Modelo1 Hil-Simulado com atraso

Na figura 5.2, mostram-se as respostas obtidas do deslocamento da massa suspensa. A figura 5.2 (a) mostra a resposta do modelo tendendo ao infinito, e a figura 5.2 (b) mostra um detalhe no tempo da resposta para ilustrar o comportamento do sistema.



a)



b)

Figura 5.2 Resposta do Modelo1 Hil-Simulado com atraso

5.2.2. *Modelo1 HIL-Simulado - Sem Atraso - Estável*

O diagrama de blocos desenvolvido para simular o modelo1 HIL sem atraso é apresentado na figura 5.3. A sequência de simulação é a mesma apresentada na seção 4.7 e 4.8.

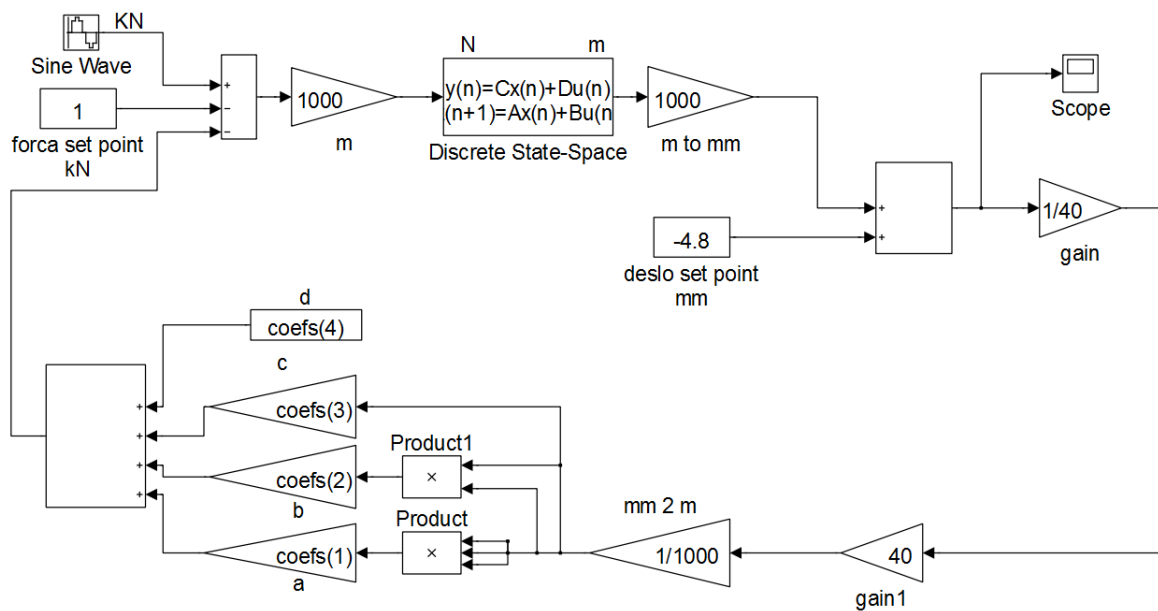


Figura 5.3 Diagrama de blocos do Modelo1 Hil-Simulado sem atraso

Na figura 5.4, mostra-se a resposta obtida do deslocamento da massa suspensa, onde pode se observar a resposta transiente e a resposta permanente do sistema.

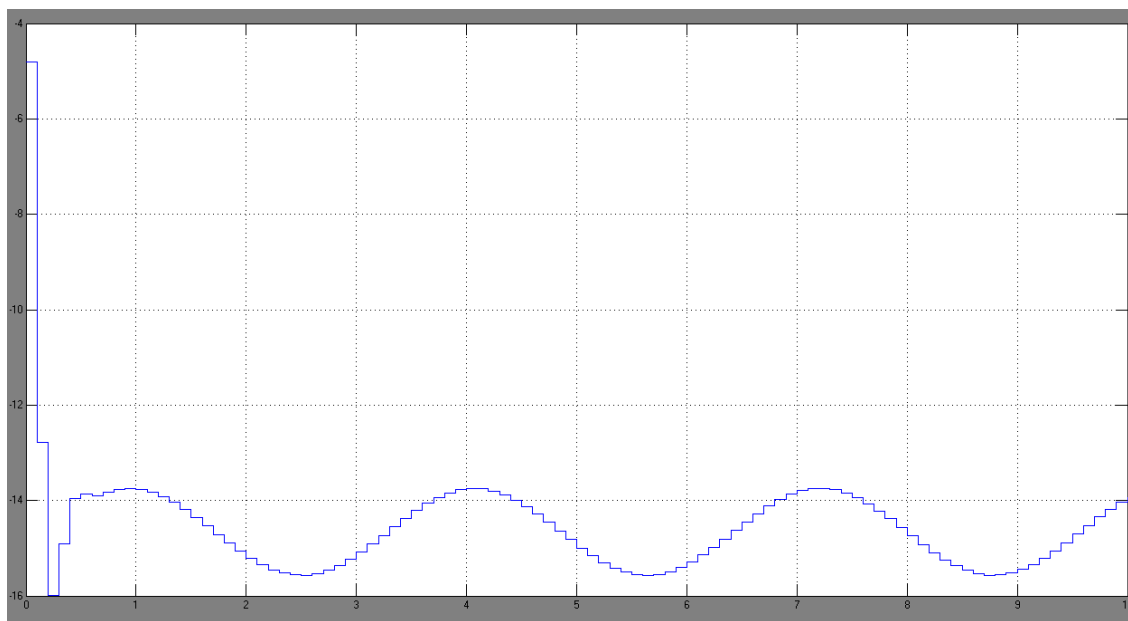


Figura 5.4 Resposta do Modelo1 Hil-Simulado sem atraso

5.3. Modelo2 - HIL

Adotou-se neste modelo o mesmo referencial absoluto do atuador e a respectiva posição inicial onde se inicia a reação da mola e o curso máximo e mínimo do atuador, ou seja, o modelo apresenta um “*setpoint*” igual a do atuador de -4.8 mm e uma amplitude de deslocamento máxima do cilindro de 40 mm, além de apresentar como parâmetros físicos as constantes da massa de 200 Kg, do amortecimento de 14 KN*s/m e da sua força peso constante de 1.9 KN. O referencial da resposta foi adotado em relação à posição inicial do “*setpoint*” correspondente a posição da mola não deformada e não em relação ao equilíbrio estático.

No software Simulink foram utilizados como tempo total de simulação o valor de 10 s, com uma taxa de amostragem tipo “*single task*” de 0.1 s e como método de integração ODE 4 (Runge-Kutta).

O sinal de excitação foi considerado como um sinal tipo senoidal com amplitude 1 KN de zero a pico e com uma frequência de 2 rad/s.

Na simulação obtém-se a resposta do deslocamento da massa suspensa onde novamente se espera que devido ao sistema ser referenciado a partir da posição inicial, com condições iniciais nulas, com uma entrada harmônica e uma força constante, a massa deverá oscilar na frequência da força harmônica de excitação em torno da posição de equilíbrio estático.

5.3.1. Modelo2 HIL- Simulado – Com Atraso

O diagrama de blocos desenvolvido para simular o modelo HIL com atraso é apresentado na figura 5.5. Observa-se que a simulação da resposta do deslocamento da massa suspensa será obtida com o atraso da força devido ao bloco de atraso inserido depois do cálculo da força. A sequência de simulação é a mesma apresentada na seção 4.6.

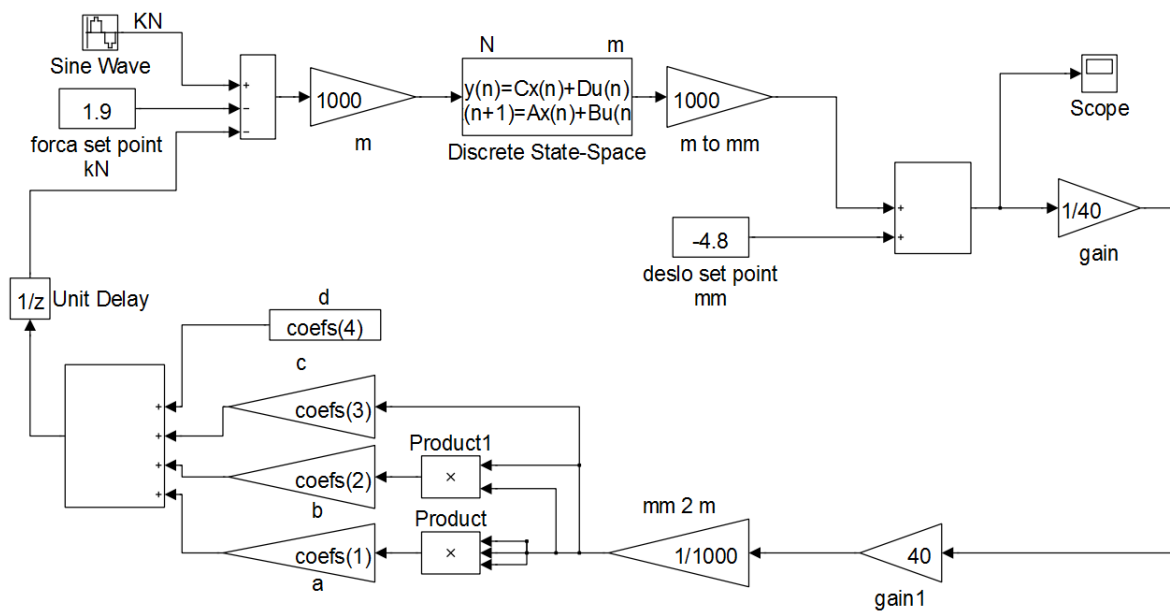


Figura 5.5 Diagrama de blocos do Modelo2 Hil simulado com atraso

Na figura 5.6, mostra-se a resposta obtida do deslocamento da massa suspensa, onde pode se observar a resposta transiente e a resposta permanente do sistema.

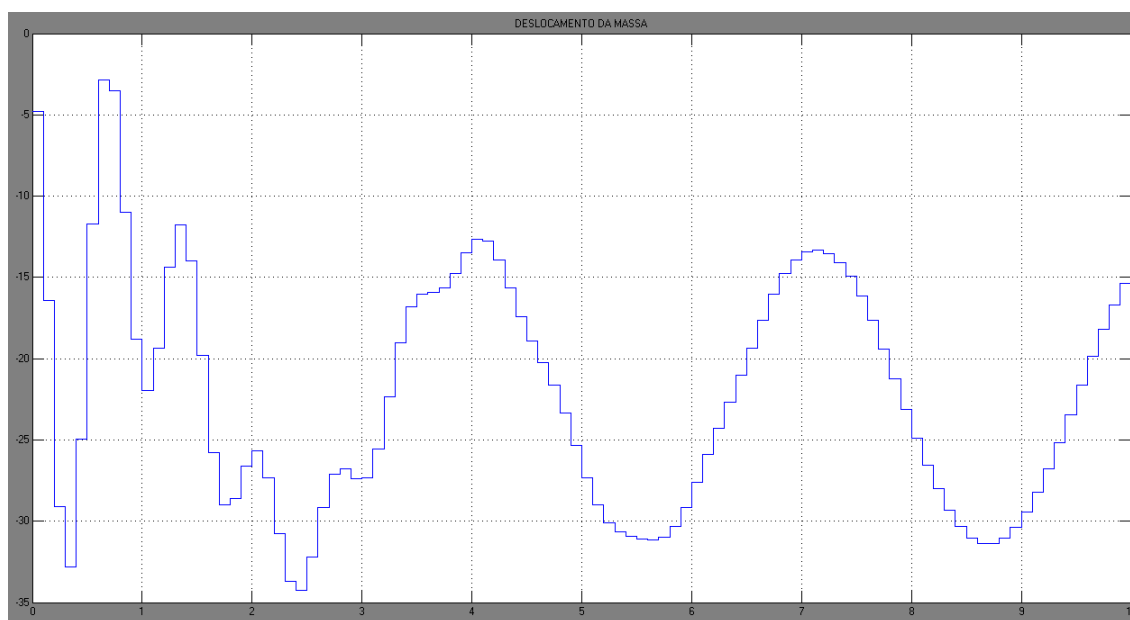


Figura 5.6 Resposta do Modelo2 Hil simulado com atraso

5.3.2. Modelo2 HIL- Simulado - sem Atraso

O diagrama de blocos desenvolvido para simular o modelo HIL sem atraso é apresentado na figura 5.7. A sequência de simulação é a mesma apresentada na seção 4.7 e 4.8.

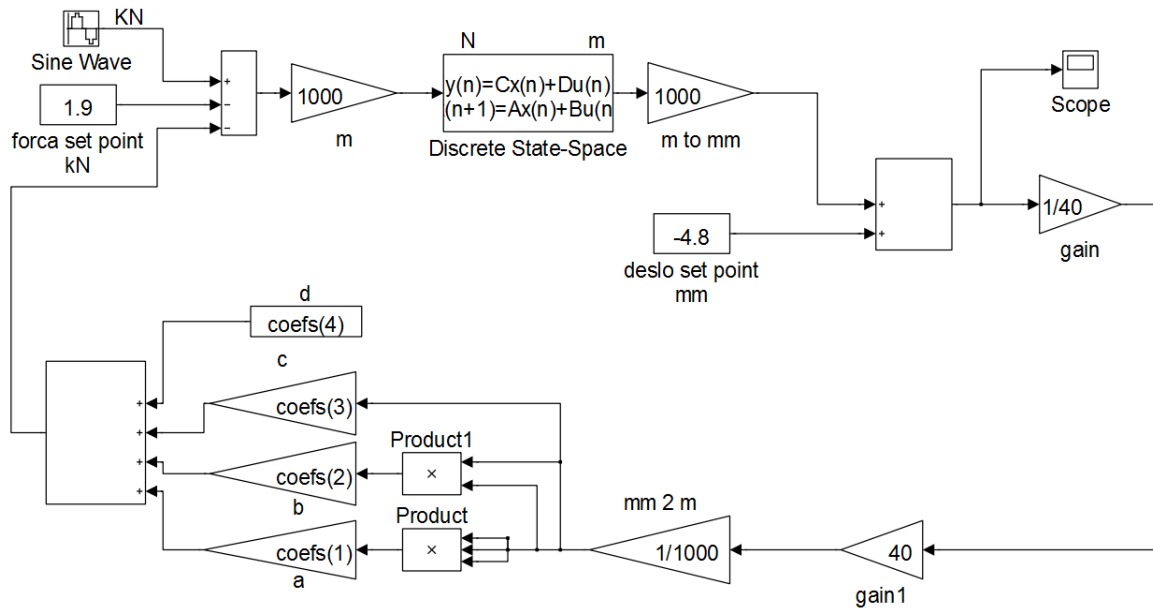


Figura 5.7 Diagrama de blocos do Modelo2 Hil simulado sem atraso

Na figura 5.8, mostra-se a resposta obtida do deslocamento da massa suspensa, onde pode se observar a resposta transiente e a resposta permanente do sistema.

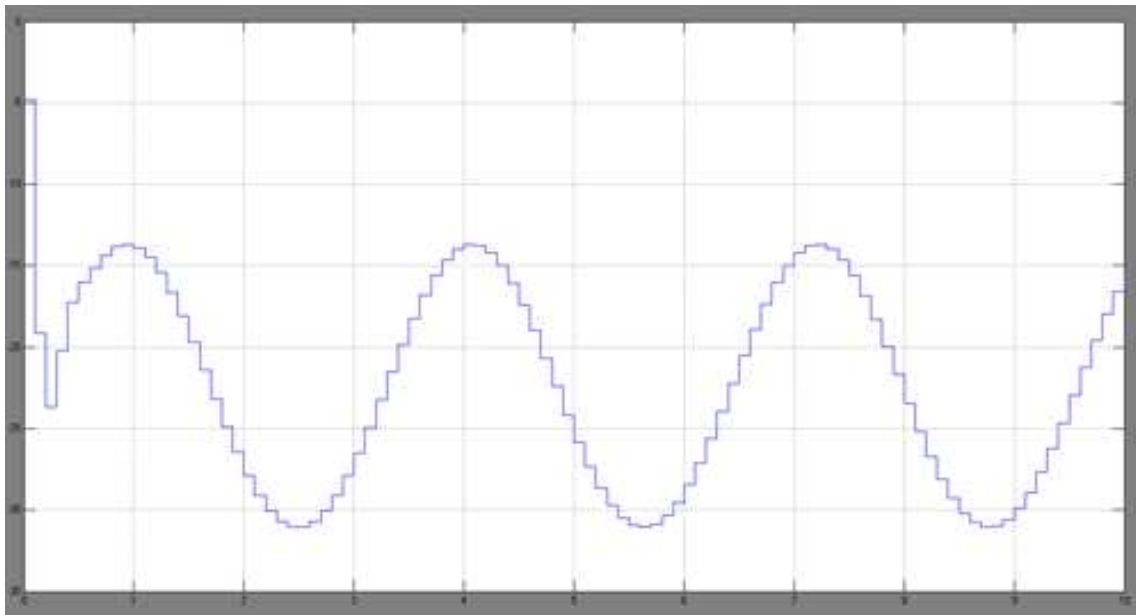


Figura 5.8 Resposta do Modelo2 Hil simulado sem atraso

Analisando o resultado do transiente das figuras 5.6 e 5.8 observam-se claramente as diferentes características da resposta do sistema devido à existência ou não do atraso na força medida.

CAPITULO 6

RESULTADOS EXPERIMENTAIS

6.1. Introdução

Neste capítulo serão apresentados os resultados experimentais realizados com a aplicação da técnica de HIL em dois tipos de sistemas, os mesmos utilizados nas simulações realizadas no capítulo 5. Estes resultados serão da técnica de HIL com atraso, da técnica de HIL com a metodologia de correção do atraso proposta 1 e da técnica de HIL com a metodologia de correção do atraso proposta 2.

No primeiro sistema modelado os parâmetros escolhidos para o sistema mostram o efeito negativo do atraso no sistema, pois com a aplicação da técnica HIL com atraso resultou em um sistema instável impossibilitando obter a resposta correta da influencia do componente físico. Por outro lado aplicando-se as técnicas do HIL sem atraso obtêm-se o comportamento estável esperado com a técnica.

No segundo sistema modelado os parâmetros escolhidos para o sistema mostram novamente o efeito negativo do atraso, pois com a aplicação da técnica HIL com atraso resulta uma resposta transiente completamente diferente da esperada. Por outro lado aplicando-se as técnicas do HIL sem atraso no sistema obtêm-se o comportamento transiente esperado com a técnica.

Para a correta modelagem e aplicação da técnica de HIL foram utilizadas nos experimentos todas as considerações apresentadas no capítulo 3 como, por exemplo, todas as respectivas modelagens de calibrações realizadas bem como o referencial adotado.

Outro ponto que deve ser observado é em relação à montagem com a mola física. Teoricamente se o valor da resposta é maior ou menor que o valor do “*setpoint*” adotado, a mola começaria a trabalhar em tração e compressão, desde que a modelagem adotada permitisse a mola trabalhar em tração e compressão. Entretanto a montagem do conjunto atuador e mola foi

realizada de forma que a mola só trabalhe em compressão, de modo que se o valor da resposta é menor que o valor do “*setpoint*” adotado, a mola fica sem ação.

Portanto para efeito de validação das respostas, só se pode comparar a resposta experimental com a resposta simulada desde a condição inicial do teste até o primeiro ponto onde a mola perde sua ação.

6.2. Experimental – Modelo1 - HIL

Adotou-se neste modelo o mesmo referencial absoluto do atuador e a respectiva posição inicial onde se inicia a reação da mola e o curso máximo e mínimo do atuador, ou seja, o modelo apresenta um “*setpoint*” igual a do atuador de -4.8 mm e uma amplitude de deslocamento máxima do cilindro de ± 40 mm, além de apresentar como parâmetros físicos as constantes da massa de 200 Kg, do amortecimento de 11.2 KN*s/m e da sua força peso constante de 1.0 KN. O comportamento da mola física é obtido através de dois sinais, o sinal de deslocamento imposto na mola (D/A) pelo atuador e o sinal da força medido (A/D) na célula de carga. O referencial da resposta foi adotado em relação à posição inicial do “*setpoint*” correspondente a posição da mola sem deformação e não em relação ao equilíbrio estático.

Como sinal de excitação foi adotado como um sinal tipo senoidal com amplitude 0.1 KN de zero a pico e com uma frequência de 2 rad/s.

Nos experimentos obtém-se a resposta do deslocamento da massa suspensa onde novamente se espera que devido ao sistema ser referenciado a partir da posição inicial da mola sem deformação, com condições iniciais nulas, com uma entrada harmônica e uma força constante, a resposta permanente da massa deverá oscilar na frequência da força harmônica de excitação em torno da posição de equilíbrio estático.

6.2.1. Experimento1 - Modelo1 HIL- Com Atraso

O diagrama de blocos desenvolvido para realizar o teste experimental 1 da técnica de HIL com atraso é apresentado na figura 6.1. A sequência de simulação é a mesma apresentada na seção 4.6. Nesta modelagem foi utilizado um tempo total de simulação de 10 s, uma taxa de amostragem tipo “single task” de 0.1 s e como método de integração o ODE 4 (Runge-Kutta).

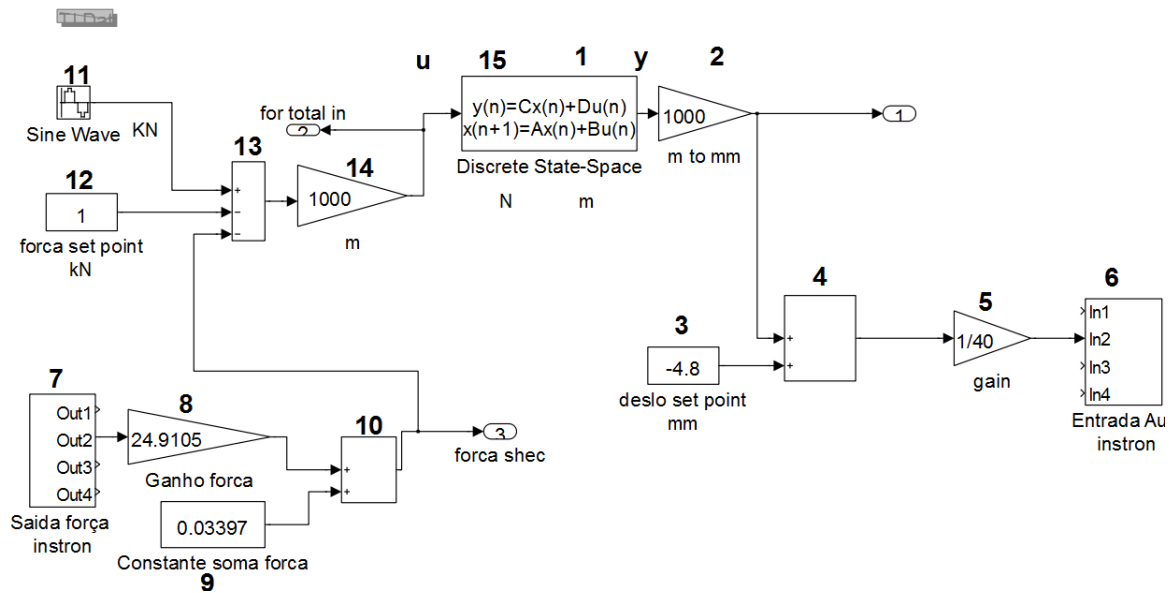


Figura 6.1 Diagrama de blocos do Experimento1 Modelo1 HIL com atraso

O código gerado pelo MATLAB/SIMULINK correspondente ao modelo da figura 6.1 que é carregado na placa de aquisição DSpace 1102 é apresentado no Anexo A.

Na figura 6.2, mostra-se a resposta obtida do deslocamento da massa suspensa, onde pode se observar a resposta transiente e a resposta permanente do sistema.

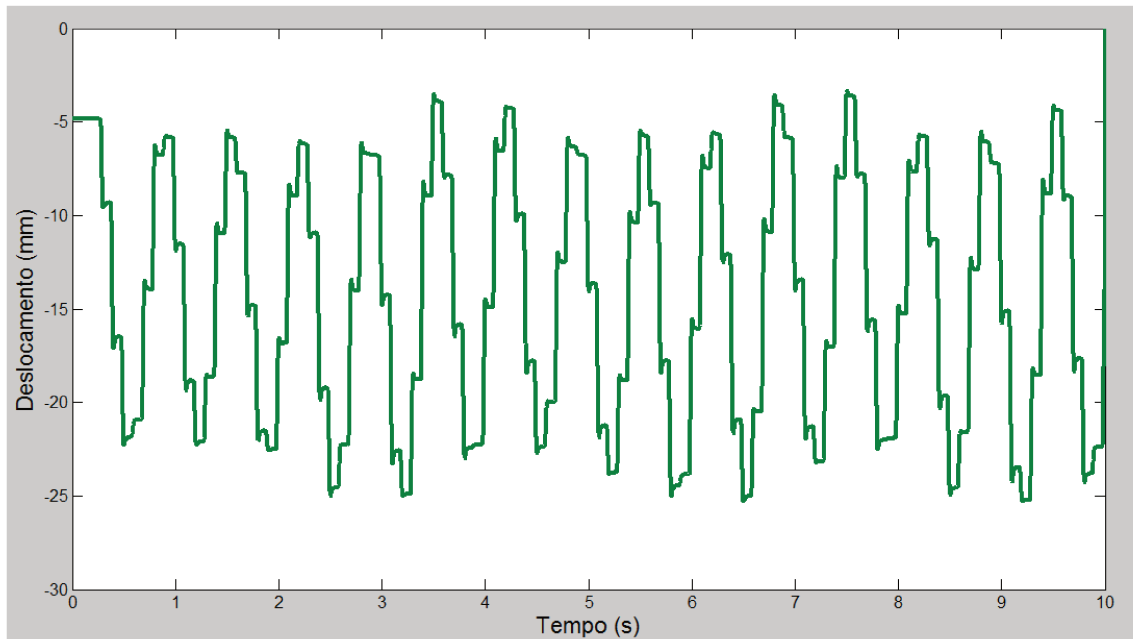


Figura 6.2 Resposta do Experimento1 Modelo1 HIL com atraso

A figura 6.3 mostra simultaneamente a resposta simulada da figura 5.2(b) e a resposta experimental da figura 6.2 obtidas do deslocamento da massa suspensa num intervalo de tempo de 0 a 1 s conforme explicado na seção 6.1. O resultado mostra a perfeita concordância das respostas, mas não tendendo ao infinito, devido aos limites de *Hardware* do sistema de atuação.

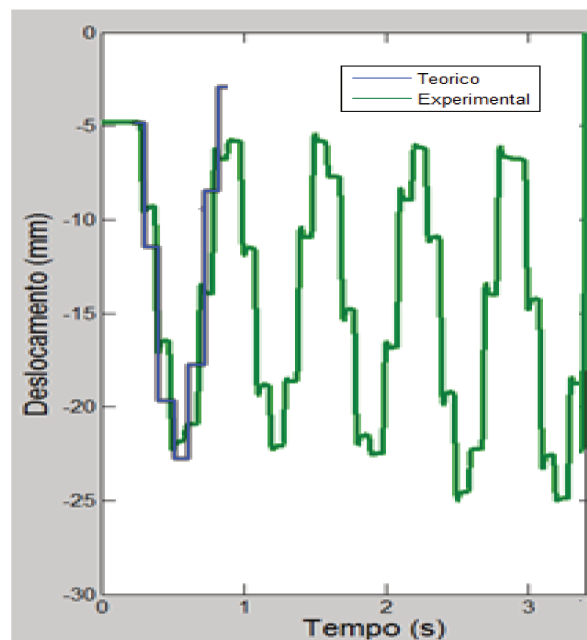


Figura 6.3 Comparação teórica e experimental das Respostas do Modelo1 HIL com atraso

6.2.2. Experimento2-Modelo1 HIL-Sem Atraso-Proposta1

O diagrama de blocos desenvolvido para realizar o teste experimental 2 da técnica de HIL sem atraso é apresentado na figura 6.4. A sequência de simulação é a mesma apresentada na seção 4.7. Nesta modelagem foi utilizado um tempo total de simulação de 10 s, uma taxa de amostragem tipo “*single task*” de 0.1 s e como método de integração o ODE 4 (Runge-Kutta).

Através do diagrama de blocos da figura 6.4, foi gerado primeiro o código C pelo MATLAB/SIMULINK apresentado no Anexo A e em seguida foi alterado o código segundo a proposta 1 apresentada na seção 4.7. Finalmente o código obtido foi carregado na placa de aquisição DSpace 1102. Este código final modificado é apresentado no Anexo B e apresenta as seguintes mudanças:

- Na função “void MdlOutputs(int_T tid)” foram acrescentadas as linhas sublinhadas com o objetivo de se calcular no inicio do instante de tempo (k) o vetor de estados $x(k)$ correspondente ao instante de tempo (k).
- Na função “void MdlUpdate(int_T tid)”, foram acrescentadas as linhas sublinhadas, com o objetivo de se armazenar do instante (k-1) as variáveis de estado $x(k-1)$ necessárias para se calcular no inicio do instante de tempo (k) o vetor de estados $x(k)$ correspondente ao instante de tempo (k).

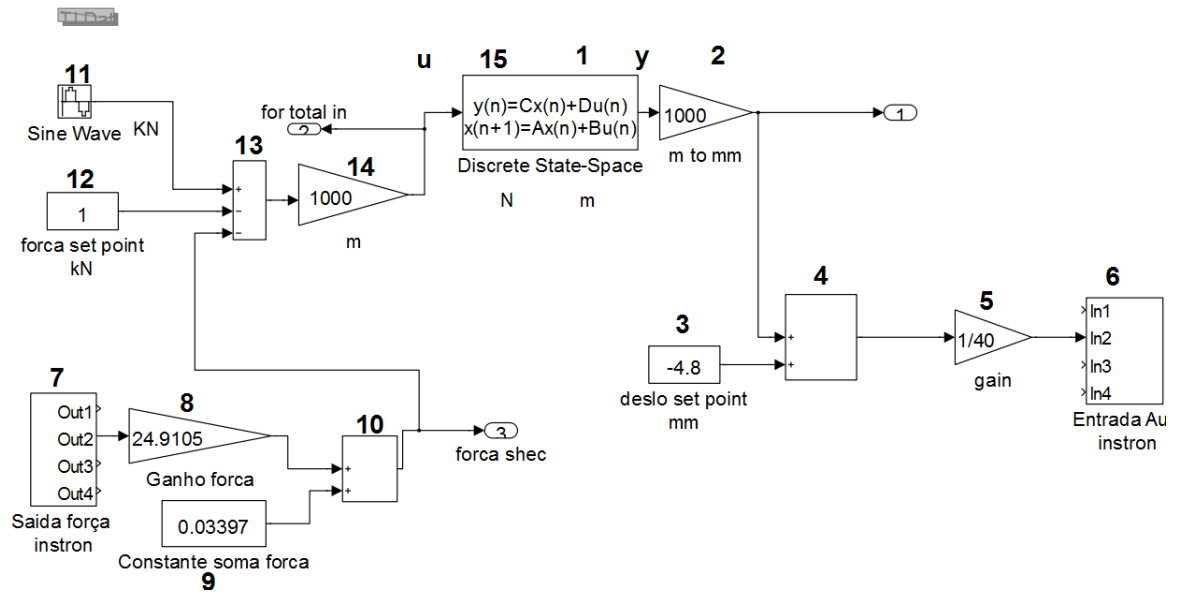


Figura 6.4 Diagrama de blocos Experimento2 Modelo1 HIL-Sem atraso – Proposta1

A figura 6.5 apresenta a resposta medida do deslocamento da massa suspensa através do sistema de aquisição Instron.

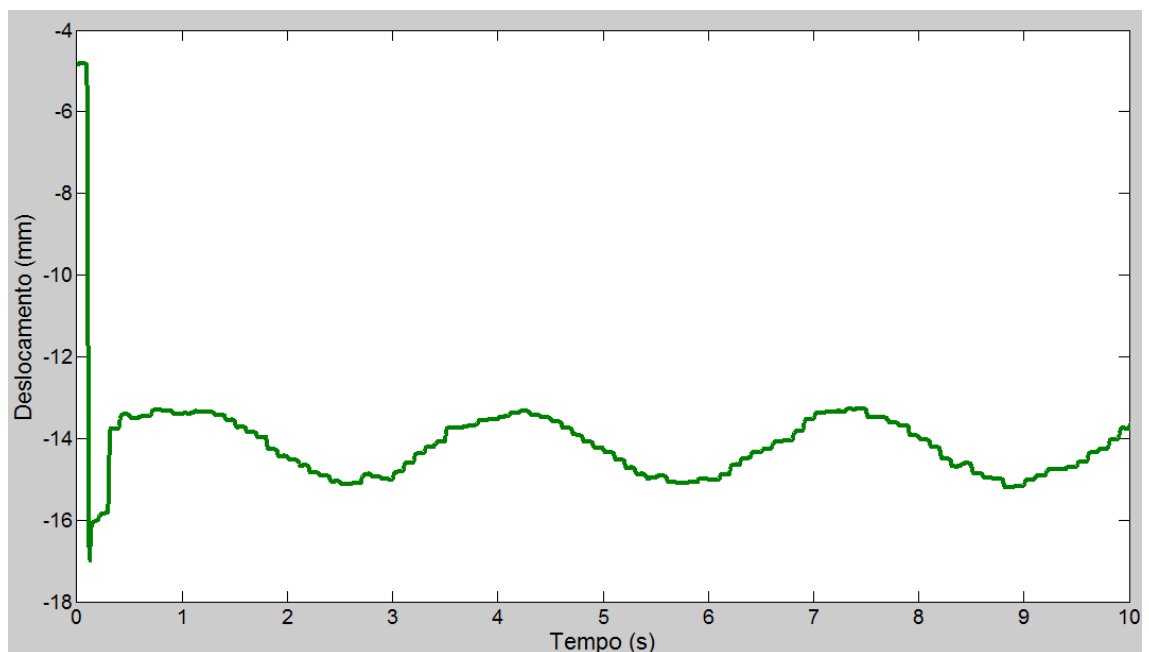


Figura 6.5 Resposta do Experimento2-Modelo1 HIL-Sem Atraso-Proposta1

A figura 6.6 mostra simultaneamente a resposta simulada da figura 5.4 e a resposta experimental da figura 6.5 obtidas do deslocamento da massa suspensa. O resultado mostra a perfeita concordância das respostas transiente e permanente do sistema.

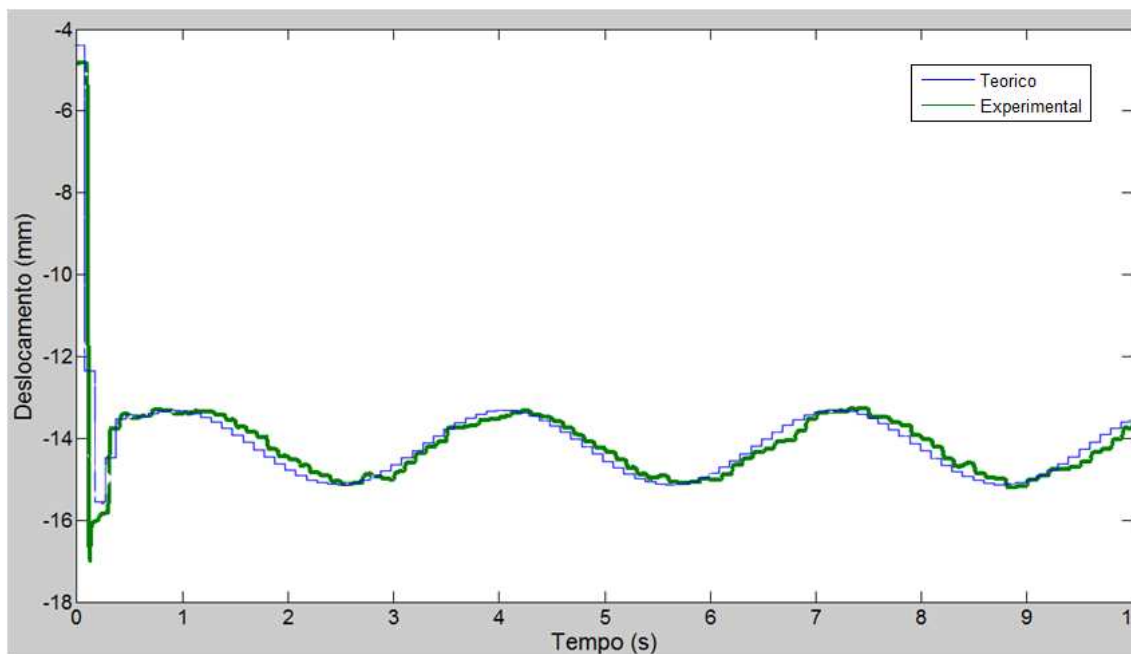


Figura 6.6 Comparação teórica e experimental das Respostas do Modelo1 HIL Sem atraso – Proposta1

6.2.3. Experimento3-Modelo1 HIL-Sem Atraso-Proposta2

O diagrama de blocos desenvolvido para realizar o teste experimental 3 da técnica de HIL sem atraso é apresentado na figura 6.7. A sequência de simulação é a mesma apresentada na seção 4.8. Nesta modelagem foi utilizado um tempo total de simulação de 10 s, uma taxa de amostragem tipo “single task” de 0.01 s e como método de integração o ODE 4 (Runge-Kutta).

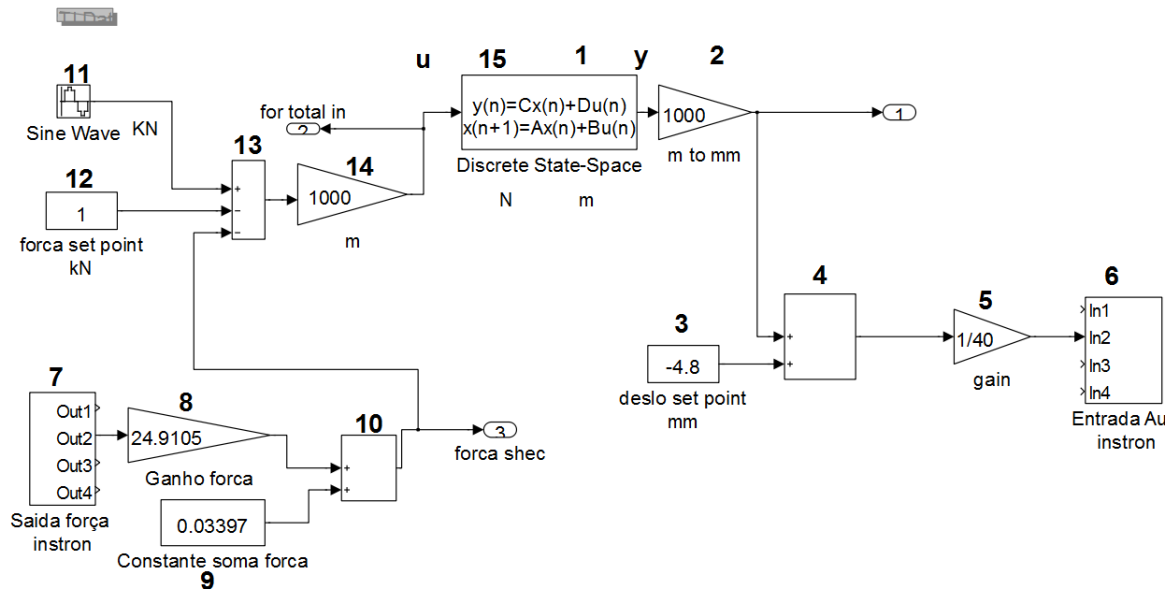


Figura 6.7 Diagrama de blocos Experimento3 Modelo1 HIL-Sem atraso – Proposta 2

Através do diagrama de blocos da figura 6.7, foi gerado primeiro o código C pelo MATLAB/SIMULINK apresentado no Anexo C e em seguida foi alterado o código segundo a proposta 2 apresentada na seção 4.8. Finalmente o código obtido foi carregado na placa de aquisição DSpace 1102. Este código final modificado é apresentado no Anexo D e apresenta as seguintes mudanças:

- Na função “void MdlOutputs(int_T tid)” foi acertado o instante de tempo de cálculo de cada operação, ou seja, muda-se quando necessário o tempo de amostragem de 0.1s correspondente ao código 1 no comando “ if (rtMlsSampleHit(rtM_hilestado, 1, tid)) { /* Sample time: [0.1, 0.0] ” para o código 0 correspondente ao comando “ if (rtMlsSampleHit(rtM_hilestado, 0, tid)) { /* Sample time: [0.01, 0.0] “. Portanto mudou-se o instante de tempo das operações de código 1 para 0 nas operações correspondentes desde a leitura “7” do conversor A/D até a entrada do sistema “15”.
- Na função “void MdlUpdate(int_T tid)”, foram acrescentadas as linhas sublinhadas, com o objetivo de armazenar, o que foi calculado no instante de tempo anterior (k-1), as variáveis de estado $x(k)$ necessárias para se calcular a cada amostragem de 0.01s, ainda no instante de tempo (k) e o próximo vetor de estados $x(k+1)$. Nesta função

também se mudou o instante de tempo das operações que calculam o próximo vetor de estados $x(k+1)$ de código 1 para 0.

A figura 6.8 apresenta a resposta medida do deslocamento da massa suspensa através do sistema de aquisição Instron.

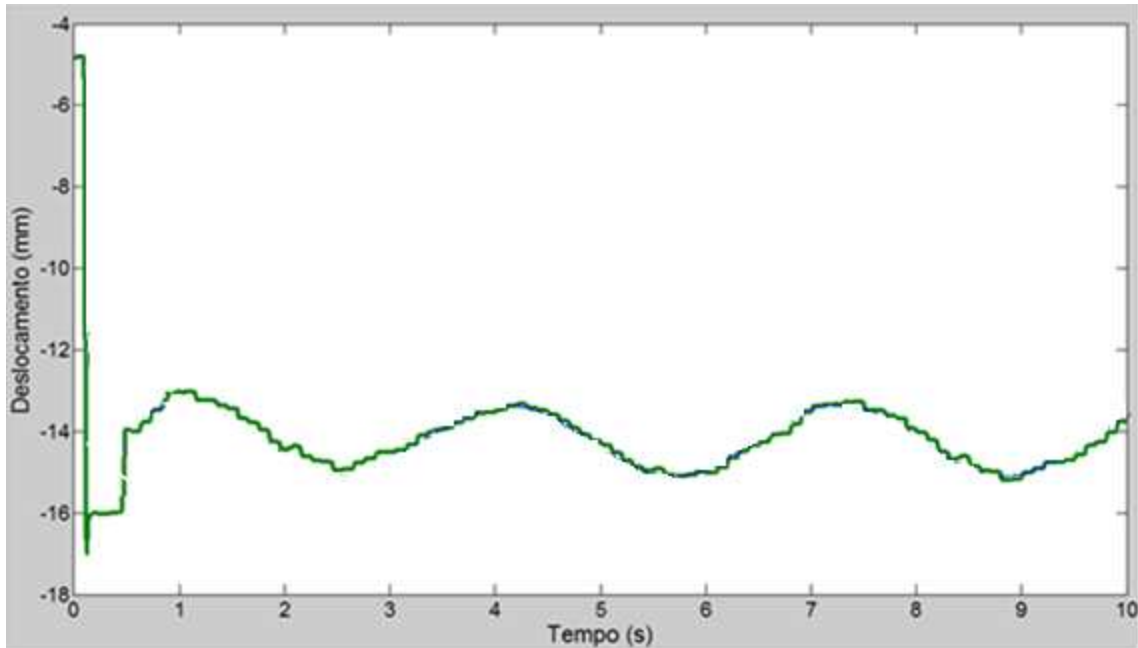


Figura 6.8 Resposta do Experimento3-Modelo1 HIL-Sem Atraso-Proposta2

A figura 6.9 mostra simultaneamente a resposta simulada da figura 5.4 e a resposta experimental da figura 6.7 obtidas do deslocamento da massa suspensa. O resultado mostra a perfeita concordância das respostas transiente e permanente do sistema.

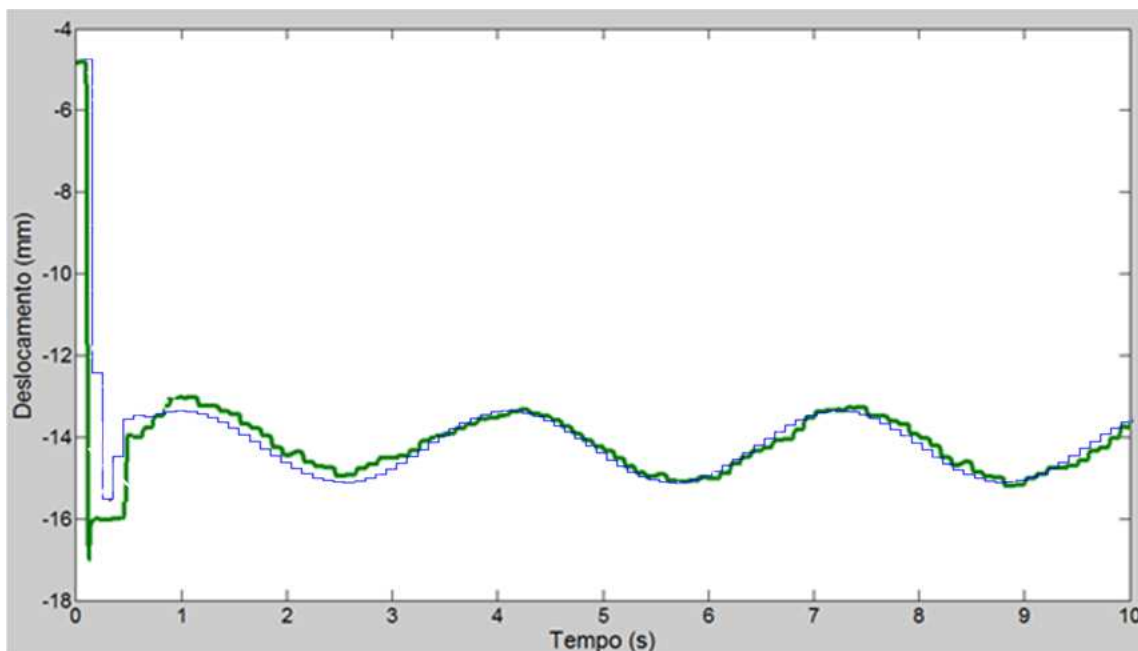


Figura 6.9 Comparação teórica e experimental das Respostas do Modelo1 HIL sem atraso – Proposta2

6.3. Experimental – Modelo 2 - HIL

Adotou-se neste modelo o mesmo referencial absoluto do atuador e a respectiva posição inicial onde se inicia a reação da mola e o curso máximo e mínimo do atuador, ou seja, o modelo apresenta um “*setpoint*” igual ao do atuador de -4.8 mm e uma amplitude de deslocamento máxima do cilindro de 40 mm, além de apresentar como parâmetros físicos as constantes da massa de 200 Kg, do amortecimento de 14 KN*s/m e da sua força peso constante de 1.9 KN. O referencial da resposta foi adotado em relação à posição inicial do “*setpoint*” correspondente a posição da nola sem deformação e não em relação ao equilíbrio estático.

No software Simulink foram utilizados como tempo total de simulação o valor de 10 s, com uma taxa de amostragem tipo “*single task*” de 0.1 s e como método de integração o ODE 4 (Runge-Kutta).

O sinal de excitação foi considerado como um sinal tipo senoidal com amplitude 1 KN de zero a pico e com uma frequência de 2 rad/s.

Na simulação obtém-se a resposta do deslocamento da massa suspensa onde novamente se espera que devido ao sistema ser referenciado a partir da posição inicial, com condições iniciais nulas, com uma entrada harmônica e uma força constante, a resposta permanente da

massa deverá oscilar na frequência da força harmônica de excitação em torno da posição de equilíbrio estático.

6.3.1. Experimento4 – Modelo2 HIL- Com Atraso

O diagrama de blocos desenvolvido para realizar o teste experimental 4 da técnica de HIL com atraso é apresentado na figura 6.10. A sequência de simulação é a mesma apresentada na seção 4.6. Nesta modelagem foi utilizado um tempo total de simulação de 10 s, uma taxa de amostragem tipo “single task” de 0.1 s e como método de integração o ODE 4 (Runge-Kutta).

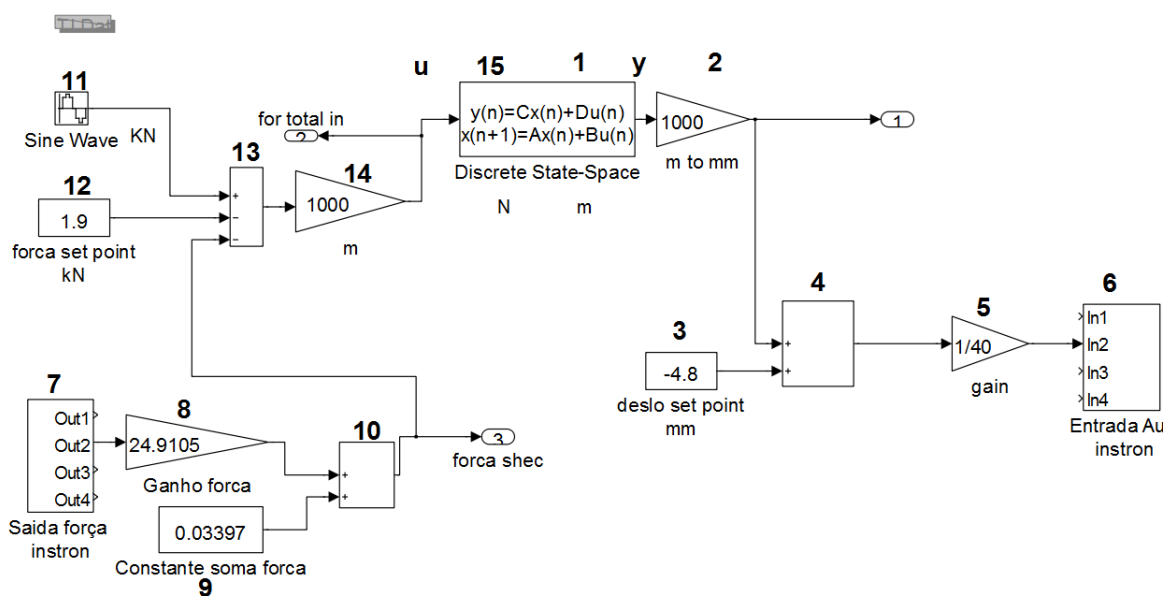


Figura 6.10 Diagrama de blocos do Experimento4 Modelo2 HIL com atraso

O código gerado pelo MATLAB/SIMULINK correspondente ao modelo da figura 6.10 que é carregado na placa de aquisição DSpace 1102 é apresentado no Anexo E.

Na figura 6.11, mostra-se a resposta obtida do deslocamento da massa suspensa, onde pode se observar a resposta transiente e a resposta permanente do sistema.

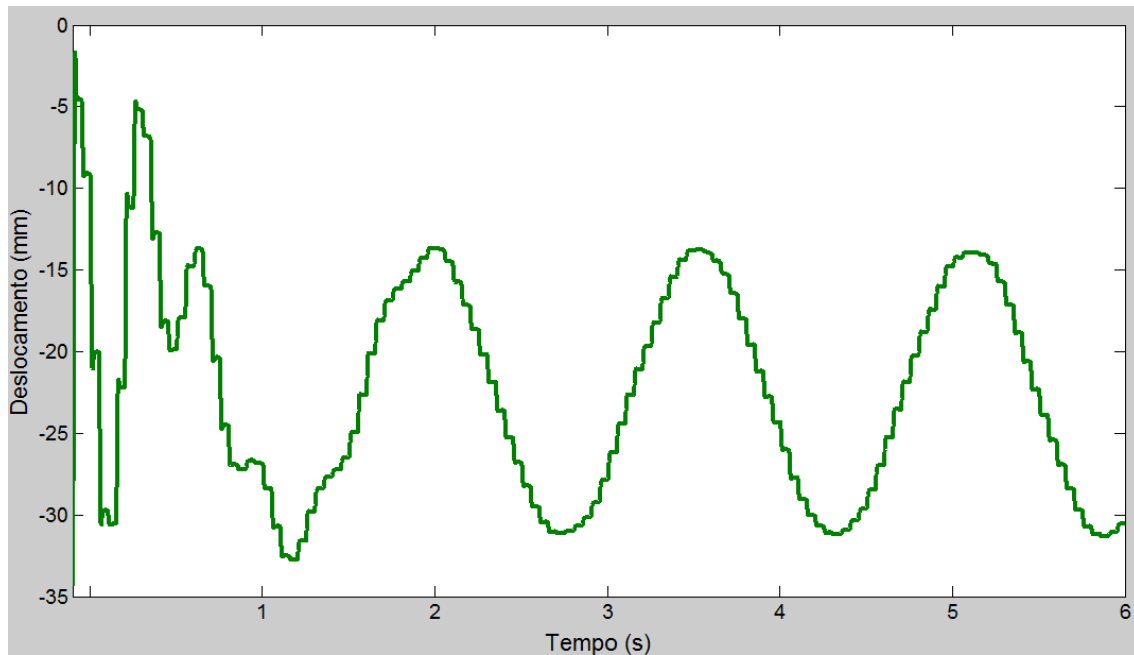


Figura 6.11 Resposta do Experimento4 Modelo2 HIL com atraso

A figura 6.12 mostra simultaneamente a resposta simulada da figura 5.6 e a resposta experimental da figura 6.11 obtidas do deslocamento da massa suspensa. O resultado mostra a perfeita concordância das respostas transiente e permanente.

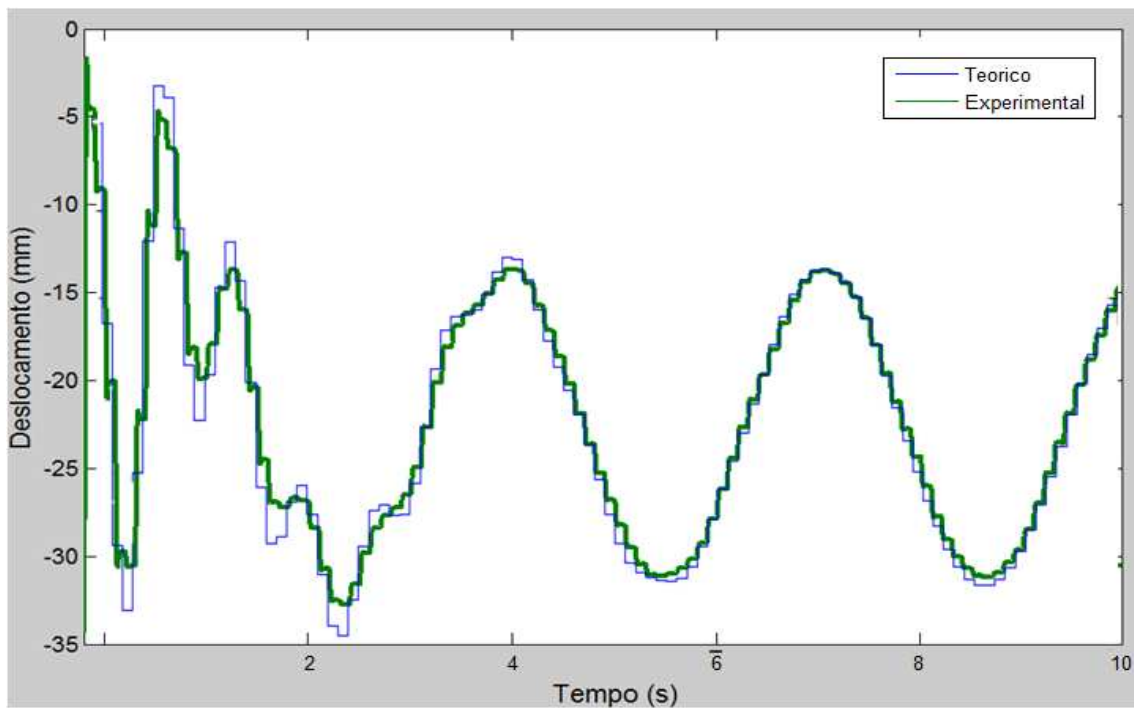


Figura 6.12 Comparação teórica e experimental das Respostas do Modelo2 HIL com atraso

6.3.2. Experimento5-Modelo2 HIL-Sem Atraso-Proposta1

O diagrama de blocos desenvolvido para realizar o teste experimental 5 da técnica de HIL sem atraso é apresentado na figura 6.13. A sequência de simulação é a mesma apresentada na seção 4.7. Nesta modelagem foi utilizado um tempo total de simulação de 10 s, uma taxa de amostragem tipo “single task” de 0.1 s e como método de integração o ODE 4 (Runge-Kutta).

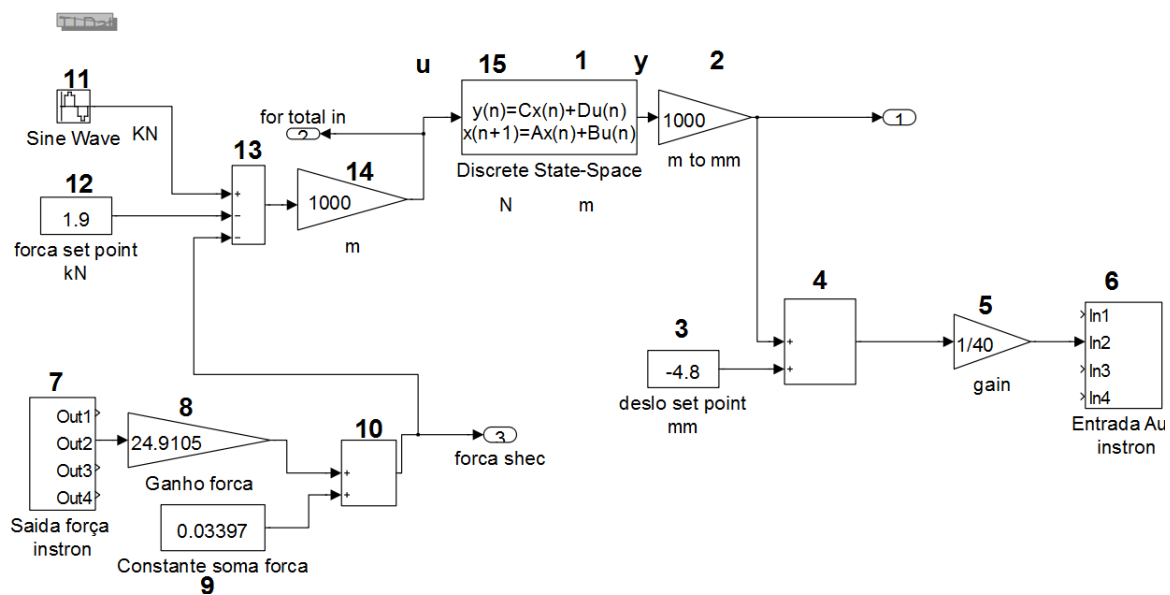


Figura 6.13 Diagrama de blocos Experimento5 Modelo2 Hil-Sem atraso – Proposta1

Através do diagrama de blocos da figura 6.13, foi gerado primeiro o código C pelo MATLAB/SIMULINK apresentado no Anexo E e em seguida foi alterado o código segundo a proposta 1 apresentada na seção 4.7. Finalmente o código obtido foi carregado na placa de aquisição DSpace 1102. Este código final modificado é apresentado no Anexo F e apresenta as seguintes mudanças:

- Na função “void MdlOutputs(int_T tid)” foram acrescentadas as linhas sublinhadas com o objetivo de se calcular no início do instante de tempo (k) o vetor de estados $x(k)$ correspondente ao instante de tempo (k).

- Na função “void MdlUpdate(int_T tid)”, foram acrescentadas as linhas sublinhadas, com o objetivo de se armazenar do instante (k-1) as variáveis de estado $x(k-1)$ necessárias para se calcular no início do instante de tempo (k) o vetor de estados $x(k)$ correspondente ao instante de tempo (k).

A figura 6.14 apresenta a resposta medida do deslocamento da massa suspensa através do sistema de aquisição Instron.

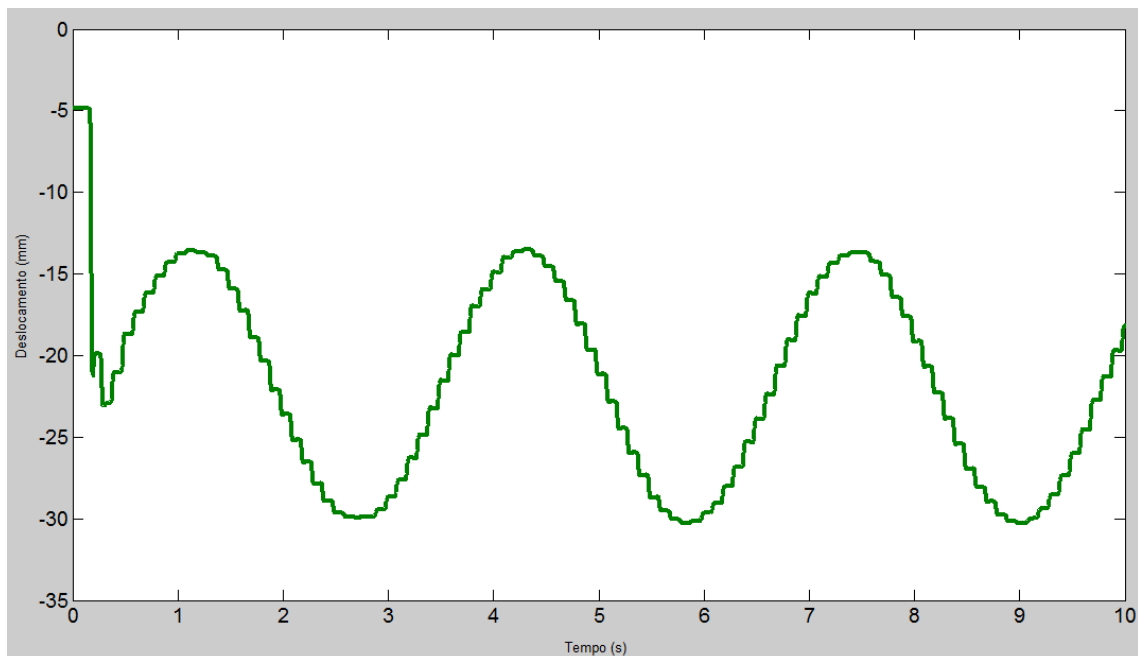


Figura 6.14 Resposta do Experimento5-Modelo2 HIL-Sem Atraso-Proposta1

A figura 6.15 mostra simultaneamente a resposta simulada da figura 5.8 e a resposta experimental da figura 6.53 obtidas do deslocamento da massa suspensa. O resultado mostra a perfeita concordância das respostas transiente e permanente do sistema.

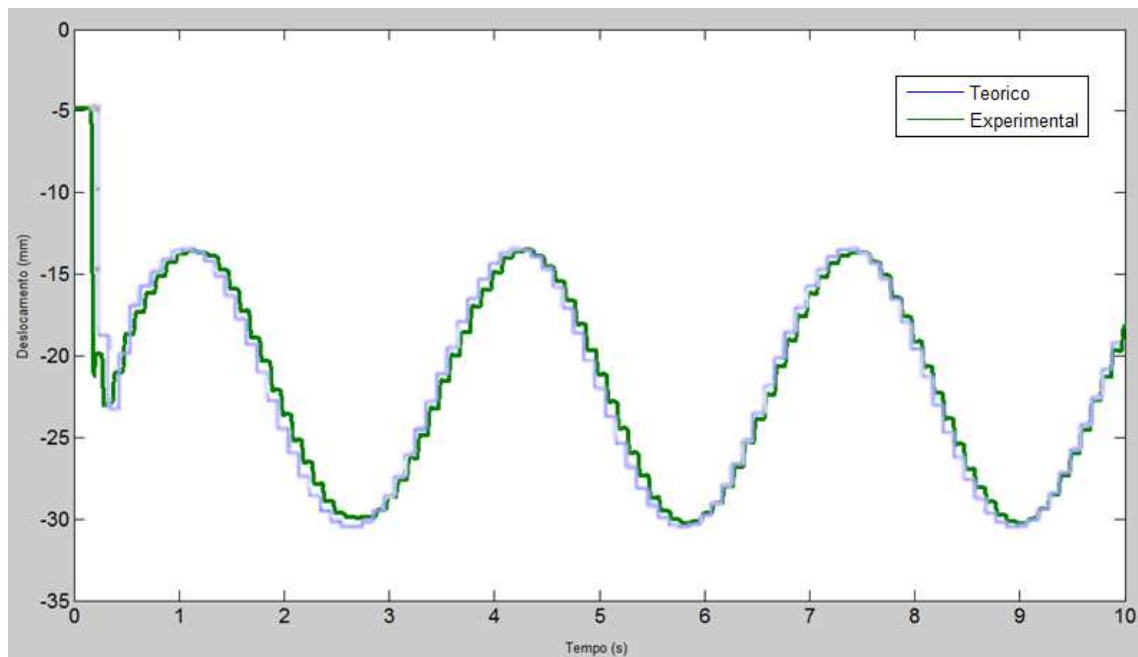


Figura 6.15 Comparação teórica e experimental das Respostas do Modelo2 HIL Sem atraso – Proposta1

6.3.3. Experimento6-Modelo2 HIL-Sem Atraso-Proposta2

O diagrama de blocos desenvolvido para realizar o teste experimental 6 da técnica de HIL sem atraso é apresentado na figura 6.16. A sequência de simulação é a mesma apresentada na seção 4.8. Nesta modelagem foi utilizado um tempo total de simulação de 10 s, uma taxa de amostragem tipo “single task” de 0.01 s e como método de integração o ODE 4 (Runge-Kutta).

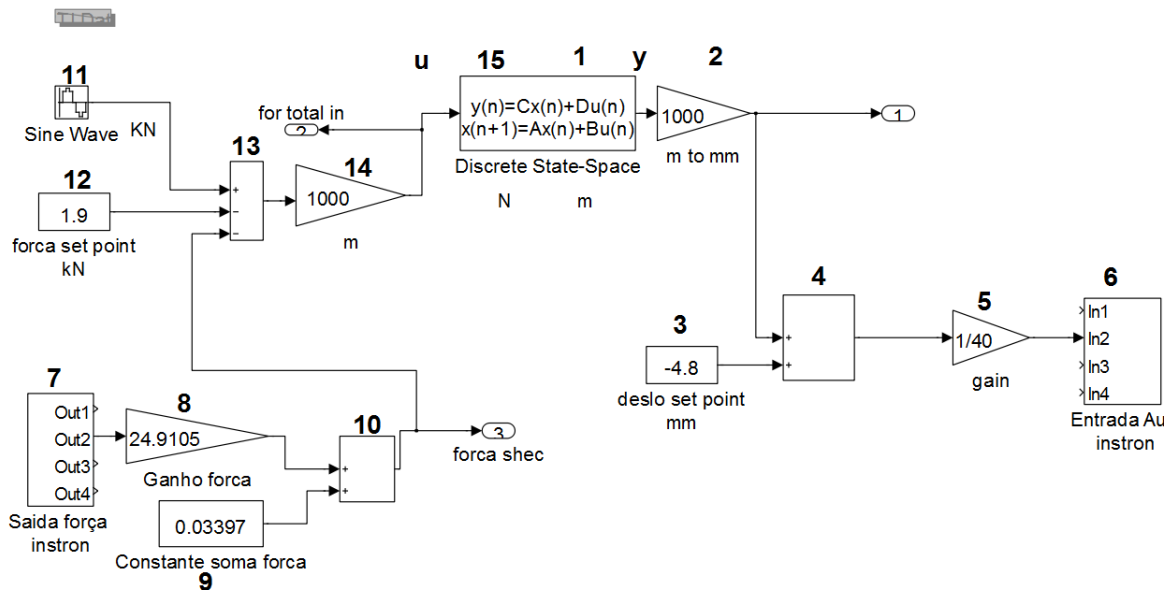


Figura 6.16 Diagrama de blocos Experimento6 Modelo2 HIL-Sem atraso – Proposta 2

Através do diagrama de blocos da figura 6.16, foi gerado primeiro o código C pelo MATLAB/SIMULINK apresentado no Anexo G e em seguida foi alterado o código segundo a proposta 2 apresentada na seção 4.8. Finalmente o código obtido foi carregado na placa de aquisição DSpace 1102. Este código final modificado é apresentado no Anexo H e apresenta as seguintes mudanças:

- Na função “void MdlOutputs(int_T tid)” foi acertado o instante de tempo de cálculo de cada operação, ou seja, muda-se quando necessário o tempo de amostragem de 0.1s correspondente ao código 1 no comando “ if (rtMlsSampleHit(rtM_hilestado, 1, tid)) { /* Sample time: [0.1, 0.0] ” para o código 0 correspondente ao comando “ if (rtMlsSampleHit(rtM_hilestado, 0, tid)) { /* Sample time: [0.01, 0.0] “. Portanto mudou-se o instante de tempo das operações de código 1 para 0 nas operações correspondentes desde a leitura “7” do conversor A/D até a entrada do sistema “15”.
- Na função “void MdlUpdate(int_T tid)”, foram acrescentadas as linhas sublinhadas, com o objetivo de armazenar, o que foi calculado no instante de tempo anterior (k-1), as variáveis de estado $x(k)$ necessárias para se calcular a cada amostragem de 0.01s,

ainda no instante de tempo (k), o próximo vetor de estados $x(k+1)$. Nesta função também se mudou o instante de tempo das operações que calculam o próximo vetor de estados $x(k+1)$ de código 1 para 0.

A figura 6.17 apresenta a resposta medida do deslocamento da massa suspensa através do sistema de aquisição Instron.

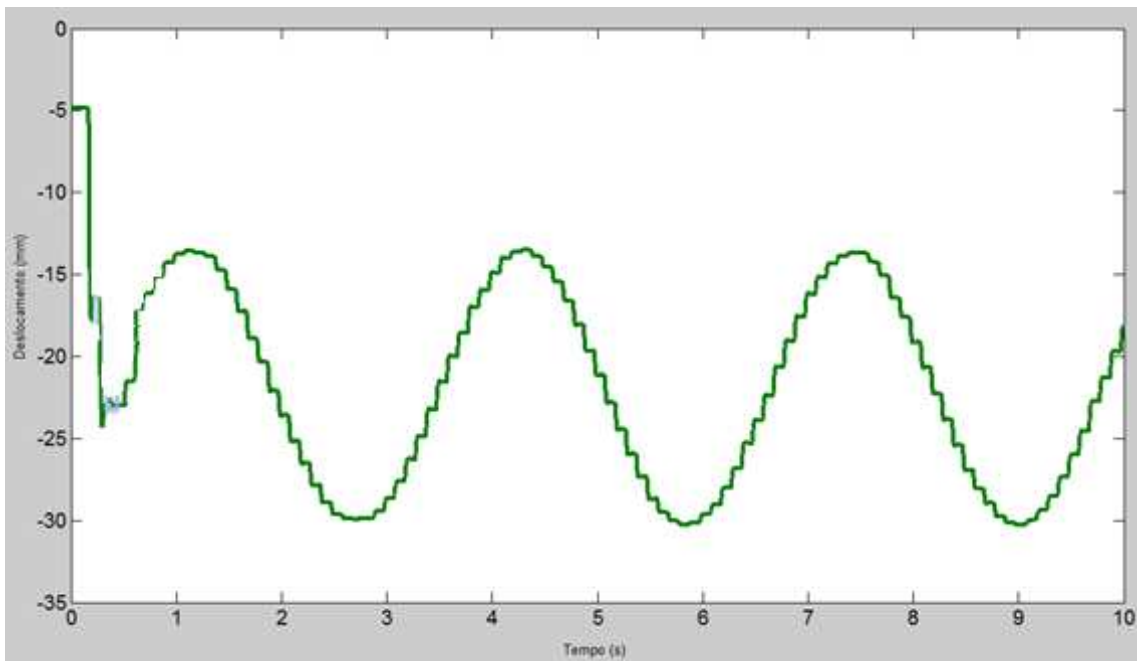


Figura 6.17 Resposta do Experimento6-Modelo2 HIL-Sem Atraso-Proposta2

A figura 6.18 mostra simultaneamente a resposta simulada da figura 5.8 e a resposta experimental da figura 6.17 obtidas do deslocamento da massa suspensa. O resultado mostra a perfeita concordância das respostas transiente e permanente do sistema.

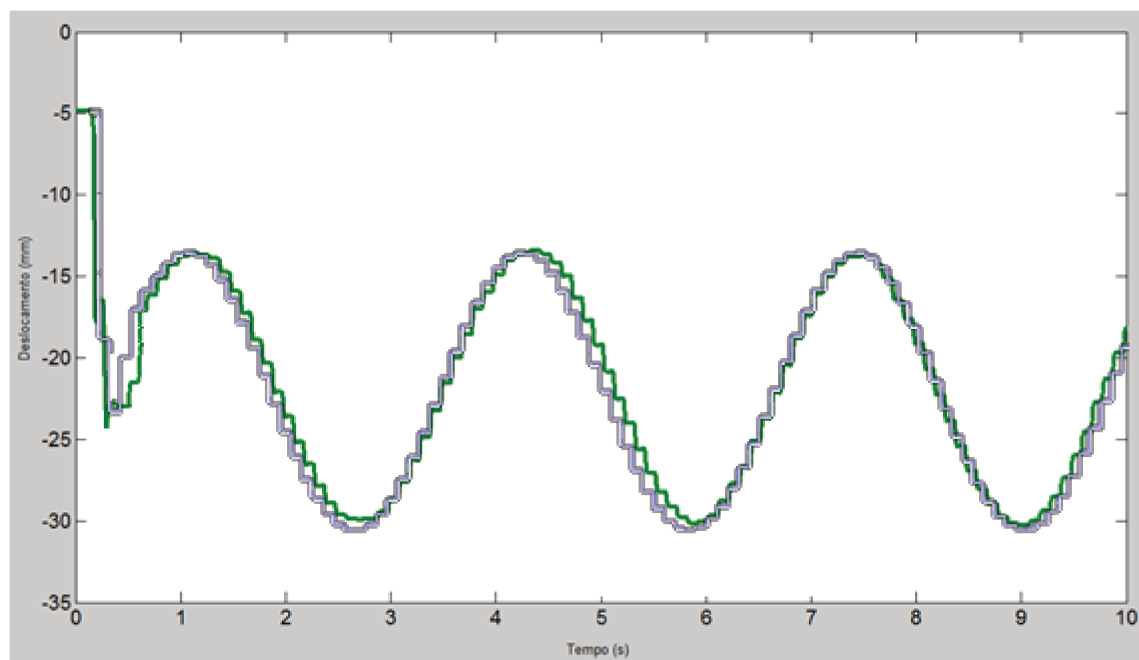


Figura 6.18 Comparação teórica e experimental das Respostas do Modelo2 HIL sem atraso – Proposta2

CAPITULO 7

CONCLUSÃO E SUGESTÕES PARA TRABALHOS FUTUROS

7.1. Conclusão

Este trabalho apresentou o desenvolvimento da técnica HIL que tem sido usada em diferentes projetos de pesquisa e de produção nos campos acadêmicos e industriais, conseguindo obter o comportamento de um sistema composto de partes modeladas matematicamente e partes compostas de componentes físicos geralmente complexos segundo as condições impostas sem a necessidade de se construir um protótipo em grande escala. Ratificando através de testes experimentais que todos os componentes físicos apresentam um atraso nos sinais já que o sistema composto pelo atuador e o componente físico não apresentam um comportamento ideal devido a diferentes motivos já discutidos na seção 2.2 como o atraso gerado pela etapa de filtragem dos sinais, pelas características dinâmicas do sistema atuador componentes físicos, conversões A/D e D/A nas medições, etapa de controle, atuação do sistema e apropriada sequência de cálculos. Avaliaram-se estratégias para eliminar este atraso levando em conta o sequenciamento da execução das tarefas no processo com a condição de continuar a ser em tempo real, obtendo ótimos resultados.

No desenvolvimento da técnica HIL foram levantados vários fatores importantes que segundo o modelo esquemático criado neste projeto, foram considerados fundamentais para obtenção do resultado alcançado. Por exemplo, vários cuidados foram tomados na correta configuração e funcionamento das ferramentas usadas na aquisição e atuação dos componentes físicos ou virtuais; no procedimento de uma boa qualidade de calibração das variáveis que interagem entre os sistemas, sendo elas neste trabalho a força e o deslocamento; nas limitações de *hardware* como na impossibilidade de se realizar ativação dos conversores no instante de tempo desejado; no tempo de amostragem mínimo o qual está limitado ao mínimo tempo de resposta do atuador, nas limitações de software para a modelagem do sistema bem como também nas suas modificações implementadas unicamente no código C.

De acordo com estas considerações, as duas propostas demonstram segundo os resultados obtidos que estes métodos têm um grande aporte na otimização da simulação e avaliação dos processos através da técnica do HIL, diminuindo o tempo de implementação, reduzindo os custos e sendo ainda muito mais importante a obtenção correta do comportamento transiente e permanente do sistema híbrido.

Portanto eliminando o atraso conseguimos obter respostas corretas sem a necessidade de se acrescentar no modelo um atraso que pode ser observado em trabalhos anteriores, mudando com isto suas características dinâmicas.

Entretanto cabe ressaltar que cada caso apresenta a sua particularidade de utilização. No caso da proposta 1 existe a limitação de que na equação 4.2 a matriz D do modelo de estado discreto seja igual à zero e que o tempo mínimo de amostragem seja maior ou igual ao atraso mínimo geral pelo sistema de atuação. No caso da proposta 2 existe a limitação da escolha do menor valor possível para o menor tempo de amostragem. A escolha deste menor tempo de amostragem deveria ser maior ou igual à soma dos tempos de todas as tarefas a serem executadas neste tempo de amostragem. Outra limitação é em relação ao maior tempo de amostragem que está limitado a um número inteiro múltiplo da menor taxa de amostragem.

7.2. Trabalhos Futuros

Como trabalho futuro o modelo esquemático proposto da técnica HIL sem atraso deve ser aplicado no desenvolvimento de sistemas mais complexos para avaliar as possíveis respostas em diferentes condições; no campo da indústria automotiva, implementar na modelagem e avaliação de sistemas de $\frac{1}{4}$ de veículo, $\frac{1}{2}$ veículo e veículo completo, nas aplicações com elementos hidropneumáticas, e em outros campos como em estruturas vibratórias, com a avaliação e estimação de parâmetros dos componentes físicos.

Segundo os trabalhos já feitos anteriormente e vistos na revisão bibliográfica onde foi apresentado o fenômeno do atraso, aplicar novos esquemas e propostas criadas para se obter resultados mais confiáveis e corretos em tempo real é um item desejado, levando-se sempre em consideração os aspectos de software e *hardware* disponíveis para a aplicação.

Para continuar com o desenvolvimento deste trabalho, sugere-se interagir com mais detalhe em diversos itens como nas ferramentas dos softwares; nas atualizações; nas novas versões de *toolbox* criados, os quais possibilitem mudar e proporcionar maior liberdade nas configurações e características que foram limitações neste trabalho, para poder conseguir o mesmo resultado aqui obtido sendo nas mesmas plataformas deste trabalho como também em outras onde seja possível realizar a técnica de HIL.

REFERÊNCIAS

BATTERBEE, D. C. and N. D. SIMS (2007). "Hardware-in-the-loop simulation of magnetorheological dampers for vehicle suspension systems." Proceedings of the Institution of Mechanical Engineers Part I-Journal of Systems and Control Engineering 221(I2): 265-278.

BORGUES, A. Instrumentação Virtual Aplicada a um Laboratório com Acesso pela Internet. 2002. Teses (Mestrado) Faculdade de Engenharia Eletrica, Escola Politécnica da Universidade de São Paulo, São Paulo.

BURNS, A., WELLINGS A. Real-time Systems and Programming Languages. ADDISON WESLEY, 2001.661p.

BUTAZZO, G. Hardware Real-Time Computing Systems. KLUWER ACADEMIC PUBLISHERS, 1997.379p.

CHENG, C. Development and Numerical Simulation of Hybrid Effective Force Testing Method. 2007. Teses (Doutorado) Faculdade de Engenharia Civil e Ambiental, Universidade Lehigh, Pennsylvania.

CANALE, M.; FAGIANO, L.; RAZZA, V. Vehicle lateral stability control via approximated NMPC: real-time implementation and software-in-the-loop test. In: DECISION AND CONTROL, 2009 HELD JOINTLY WITH THE 2009 28TH CHINESE CONTROL CONFERENCE ON, Shangai, 2009. p.4596-4601.

CHANG T., CONG D., YE Z., HAN H. Time problems in hil simulation for on-orbit docking and compensation. Proceedings of the IEEE Second IEEE Conference on Industrial Electronics and Applications. pp. 841-846, 2007.

CRAVOTTA, R. (26 de maio de 2005). Electronic design, strategy, news. Edn, mixing the real with the virtual. Acesso em 9 de novembro de 2011, disponível em electronic design, strategy, news. Edn , mixing the Real with the virtual: <http://www.edn.com/contents/images/601832.pdf>

DARBY, A. P., WILLIAMS, M. S. and BLAKEBOROUGH, A., 2002. Stability and delay compensation for real-time substructure testing. *Journal of Engineering Mechanics*, 128, 1276--1284.

DA SILVA, H. Simulação com hardware in the loop aplicada a vehiculos submarinos semi-autonomos. 2008. Teses (Mestrado) Escola Politécnica, Universidade de São Paulo, São Paulo.

DEL SIGNORE, M. J.; KROVI, V. Virtual prototyping and hardware-in-the-loop testing for musculoskeletal system analysis, Proceedings of the IEEE international conference Mechatronics and automation, pp. 394-399, 2005, Ontario, canada.

GAWTHROP P.J., VIRDEN D.W. , NEIL D. S.A. , WAGG D.J. Emulator-based control for actuator-based hardware-in-the-loop testing. emulator-based control for actuator based hardware in-the-loop testing. Control Engineering Practice 16. pp.897-908. 2008.

GMBH. Real-time interface (rti and rti-mp) implementation guide Dspace. 2008.

GMBH. Manual guide ControlDesk Dspace. 2003.

HAGIWARA, K. Development of automatic transmission control system using hardware-in-the-loop simulation system. Proceedings of the Society of Automotive Engineers of Japan Inc., v. 23, n.2, pp. 55-59, 2002.

HORIUCHI, T., INOUE, M., KONNO, T. NAMITA, Y. (1999), Real-time hybrid experimental system with actuator delay compensation and its application to a piping system with energy absorber. Earthquake Engng. Struct. Dyn., 28: 1121–1141.

KARPENKO, M. Hardware-in-the-loop simulator for research on fault tolerant control of electrohydraulic actuators in a flight control application. Proceedings of the American Control Conference, pp.7, Junho 2006.

KITCHING, K. J., D. J. COLE, et al. (2000). "Performance of a Semi-Active Damper for Heavy Vehicles." Journal of Dynamic Systems, Measurement, and Control 122(3): 498-506.

LINJAMA, M., T. Virvalo, et al. (2000). "Hardware-in-the-loop environment for servo system controller design, tuning and testing." Microprocessors and Microsystems 24(1): 13-21.

LINS, A.R. A técnica de hardware-in-the-loop no auxílio de projetos mecatrônicos. Revista Mecatrônica Atual. Ano 6, ed. 35, 2007.

LINS, A.R. Aplicações de hardware in the loop no desenvolvimento de uma mão robótica. 2007. Teses (Doutorado) Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos.

LIU, J. W. S. (2000). Real-Time systems. Upper Saddle River, NJ, Prentice Hall.

MISSELHORN, W.E. Verification of hardware-in-the loop as a valid testing method for suspension development. 2004. Teses (Mestrado) Faculdade de Engenharia Mecânica, Universidade de Pretoria, Pretoria.

NOOMWONGS, N., H. YOSHIDA, et al. (2003). "Study on handling and stability using tire hardware-in-the-loop simulator." JSAE Review 24(4): 457-464.

PASSOS, W.D. Utilização de ferramentas de prototipagem rápida direcionada à concepção de sistemas embarcados baseados em computação reconfigurável. 2008. Teses (Mestrado) Faculdade de Engenharia Mecânica, Universidade Estadual de Campinas, Campinas.

PETERSHEIM, M. D. and S. N. BRENNAN (2009). "Scaling of hybrid-electric vehicle powertrain components for Hardware-in-the-loop simulation." *Mechatronics* 19(7): 1078-1090.

PETIT SUÁREZ, J. F., M. A. MANTILLA VILLALOBOS, et al. (2011). "Estimación de las componentes simétricas instantáneas de señales eléctricas usando filtros sintonizados." *Ingeniare. Revista chilena de ingeniería* 19: 110-121.

REINHORN, A.M.,SIVASELVAN, M., et al. Real-time dynamic hybrid testing of structural systems. Proceedings of the 13th World Conference on Earthquake Engineering Canada, paper 1664, 2004.

RILL, G. (2012). *Road vehicle dynamics: fundamentals and modeling*. Boca Raton, FL, CRC Press.

RÖCK, S. (2011). "Hardware in the loop simulation of production systems dynamics." *Production Engineering* 5(3): 329-337.

SHIAKOLAS, P. S. and D. PIYABONGKARN (2003). "Development of a real-time digital control system with a hardware-in-the-loop magnetic levitation device for reinforcement of controls education." *Ieee Transactions on Education* 46(1): 79-87.

VATH, A., Z. LEMĚS, et al. (2006). "Dynamic modelling and hardware-in-the-loop testing of PEMFC." *Journal of Power Sources* 157(2): 816-827.

VIVERO, I. M., Estudio del comportamiento dinámico de un vehículo utilizando la herramienta simmechanics de matlab. Engenharia Técnica Industrial Mecânica, Universidade Carlos III, Madrid.

WALLACE, M., J. SIEBER, et al. (2005). "Delay Differential Equation Models for Real-Time Dynamic Substructuring." *ASME Conference Proceedings* 2005(47438): 875-882.

WILLIAMS, M. S., Real-time hybrid testing in structural dynamics. Proceedings of the 5th Australasian Congress on Applied Mechanics, Brisbane, Australia, 2007

WU, X., VLADIMIROVA, T., 2008 NASAESA Conference on Adaptive Hardware and Systems, 424-431 (2008).

YOUNG, S. J. (1982). *Real time languages, design and development*, E. Horwood.

ANEXO A – CODIGO GERADO MODELO 1 PROPOSTA 1

```

/*
 * Real-Time Workshop code generation for Simulink
 * model "hilestadocomatrasso56exc01.mdl".
 *
 * Model Version          : 1.743
 * Real-Time Workshop file version : 5.0 $Date:
 * 2002/05/30 19:21:33 $
 * Real-Time Workshop file generated on : Sat Feb 11
 * 20:13:59 2012
 * TLC version           : 5.0 (Jun 18 2002)
 * C source code generated on : Sat Feb 11
 * 20:13:59 2012
 */

#include <math.h>
#include <string.h>
#include "hilestadocomatrasso56exc01.h"
#include "hilestadocomatrasso56exc01_private.h"
#include "ext_work.h"

#include "hilestadocomatrasso56exc01_dt.h"

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

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

/* External output (root outputs fed by signals with
auto storage) */
ExternalOutputs rtY;

/* Parent Simstruct */
static rtModel_hilestadocomatrasso56exc01 model_S;
rtModel_hilestadocomatrasso56exc01 *const
rtM_hilestadocomatrasso56exc01 =
    &model_S;

/* Declaration needed for RTW malloc() usage */
const char
*RT_MEMORY_ALLOCATION_ERROR = "RTW
memory allocation error.";

/* Initial conditions for root system: '<Root>' */
void MdlInitialize(void)
{
    /* DiscreteStateSpace Block: <Root>/Discrete State-
    Space */
    rtDWork.Discrete_State_Space_DSTATE[0] =
    rtP.Discrete_State_Space_X0[0];

    rtDWork.Discrete_State_Space_DSTATE[1] =
    rtP.Discrete_State_Space_X0[1];
}

/* Enable for root system: '<Root>' */
void MdlEnable(void)
{
    /* Sin Block: <Root>/Sine Wave */
    rtDWork.Sine_Wave_IWORK.SystemEnable =
    (int_T) TRUE;
}

/* Start for root system: '<Root>' */
void MdlStart(void)
{
    /* dSPACE I/O Board DS1102 Unit:DAC */
    ds1102_da(2, 0);

    MdlInitialize();
    MdlEnable();
}

/* Outputs for root system: '<Root>' */
void MdlOutputs(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);

    /* DiscreteStateSpace: '<Root>/Discrete State-
    Space' */
    {
        rtB.Discrete_State_Space =
        rtP.Discrete_State_Space_C*rtDWork.Discrete_State
        _Space_DSTATE[0];
    }

    /* Gain: '<Root>/m to mm'
    *
    * Regarding '<Root>/m to mm':
    * Gain value: rtP.m_to_mm_Gain
    */
    rtB.m_to_mm = rtB.Discrete_State_Space *
    rtP.m_to_mm_Gain;

    /* Output: '<Root>/volts' */
    rtY.volts = rtB.m_to_mm;

```

```

/* Sum: '<Root>/Sum1' incorporates:
 * Constant: '<Root>/deslo set point mm'
 */
rtB.Sum1 = rtB.m_to_mm +
rtP.deslo_set_point_mm_Value;

/* Gain: '<Root>/gain'
 *
 * Regarding '<Root>/gain':
 * Gain value: rtP.gain_Gain
 */
rtB.gain = rtB.Sum1 * rtP.gain_Gain;

/* dSPACE I/O Board DS1102 Unit:DAC */
ds1102_da(2, rtB.gain);

/* dSPACE I/O Board DS1102 Unit:ADC */
rtB.S_Function2_b = (real_T) ds1102_ad(2);

/* Gain: '<Root>/Ganho forca'
 *
 * Regarding '<Root>/Ganho forca':
 * Gain value: rtP.Ganho_forca_Gain
 */
rtB.Ganho_forca = rtB.S_Function2_b *
rtP.Ganho_forca_Gain;

/* Sum: '<Root>/Sum6' incorporates:
 * Constant: '<Root>/Constante soma forca'
 */
rtB.Sum6 = rtB.Ganho_forca +
rtP.Constante_soma_forca_Value;

/* Sin: '<Root>/Sine Wave' */
/* check enable state */
if (
(int_T)rtDWork.Sine_Wave_IWORK.SystemEnable
) {
    rtDWork.Sine_Wave_RWORK.LastSin =
sin(rtP.Sine_Wave_Freq *
    rtmGetT(rtm_hilestadocomatrasso56exc01));
    rtDWork.Sine_Wave_RWORK.LastCos =
cos(rtP.Sine_Wave_Freq *
    rtmGetT(rtm_hilestadocomatrasso56exc01));
    rtDWork.Sine_Wave_IWORK.SystemEnable =
(int_T)FALSE;
}

/* output sine */

rtB.Sine_Wave =
rtP.Sine_Wave_Amp *
(
    (rtDWork.Sine_Wave_RWORK.LastSin *
rtP.Sine_Wave_PCos

```

```

+ rtDWork.Sine_Wave_RWORK.LastCos *
(rtP.Sine_Wave_PSin)) *
    rtP.Sine_Wave_HCos
+ (rtDWork.Sine_Wave_RWORK.LastCos *
rtP.Sine_Wave_PCos
- rtDWork.Sine_Wave_RWORK.LastSin *
(rtP.Sine_Wave_PSin)) *
    rtP.Sine_Wave_Hsin
) + rtP.Sine_Wave_Bias;

/* Sum: '<Root>/Sum2' incorporates:
 * Constant: '<Root>/forca set point kN'
 */
rtB.Sum2 = rtB.Sine_Wave -
rtP.forca_set_point_kN_Value - rtB.Sum6;

/* Gain: '<Root>/m'
 *
 * Regarding '<Root>/m':
 * Gain value: rtP.m_Gain
 */
rtB.m = rtB.Sum2 * rtP.m_Gain;

/* Output: '<Root>/for total in' */
rtY.for_total_in = rtB.m;

/* Output: '<Root>/forca shec' */
rtY.forca_shec = rtB.Sum6;

/* user code (Output function Trailer) */
/* dSPACE Data Capture block: (none) */
/* ... Service number: 1 */
/* ... Service name: (default) */
{
    host_service(1, 0)
}

/* 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);

    /* DiscreteStateSpace Block: <Root>/Discrete State-
    Space */
    {
        static real_T xnew[2];
        xnew[0] = (rtP.Discrete_State_Space_B[0])*rtB.m;
        xnew[0] +=

```

```

(rtP.Discrete_State_Space_A[0])*rtDWork.Discrete_
State_Space_DSTATE[0]
+
(rtP.Discrete_State_Space_A[1])*rtDWork.Discrete_
State_Space_DSTATE[1];

xnew[1] = (rtP.Discrete_State_Space_B[1])*rtB.m;
xnew[1] +=

(rtP.Discrete_State_Space_A[2])*rtDWork.Discrete_
State_Space_DSTATE[1];

(void)memcpy(&rtDWork.Discrete_State_Space_DS
TATE[0], xnew,
sizeof(real_T)*2);
}

/* Sin Block: <Root>/Sine Wave */
{
    real_T hold_sin;
    real_T hold_cos;

    hold_sin =
rtDWork.Sine_Wave_RWORK.LastSin;
    hold_cos =
rtDWork.Sine_Wave_RWORK.LastCos;

    rtDWork.Sine_Wave_RWORK.LastSin = hold_sin
* rtP.Sine_Wave_HCos + hold_cos *
    rtP.Sine_Wave_Hsin;
    rtDWork.Sine_Wave_RWORK.LastCos =
hold_cos * rtP.Sine_Wave_HCos - hold_sin *
    rtP.Sine_Wave_Hsin;
}
}

/* Terminate for root system: '<Root>' */
void MdlTerminate(void)
{
    if(rtM_hilestadocomatraso56exc01 != NULL) {
    }
}

/* Function to initialize sizes */
void MdlInitializeSizes(void)
{
    rtM_hilestadocomatraso56exc01->Sizes.numContStates = (0); /* Number of
continuous states */
    rtM_hilestadocomatraso56exc01->Sizes.numY =
(3); /* Number of model outputs */
    rtM_hilestadocomatraso56exc01->Sizes.numU =
(0); /* Number of model inputs */

    rtM_hilestadocomatraso56exc01->
>Sizes.sysDirFeedThru = (0); /* The model is not
direct feedthrough */
    rtM_hilestadocomatraso56exc01->
>Sizes.numSampTimes = (1); /* Number of sample
times */
    rtM_hilestadocomatraso56exc01->Sizes.numBlocks
= (23); /* Number of blocks */
    rtM_hilestadocomatraso56exc01->
>Sizes.numBlockIO = (13); /* Number of block
outputs */
    rtM_hilestadocomatraso56exc01->
>Sizes.numBlockPrms = (22); /* Sum of parameter
"widths" */
}

/* Function to initialize sample times */
void MdlInitializeSampleTimes(void)
{
    /* task periods */
    rtM_hilestadocomatraso56exc01->
>Timing.sampleTimes[0] = (0.1);

    /* task offsets */
    rtM_hilestadocomatraso56exc01->
>Timing.offsetTimes[0] = (0.0);
}

/* Function to register the model */
rtModel_hilestadocomatraso56exc01
*hilestadocomatraso56exc01(void)
{
    (void)memset((char
*)rtM_hilestadocomatraso56exc01, 0,
sizeof(rtModel_hilestadocomatraso56exc01));

    {
        /* Setup solver object */
        static RTWSolverInfo rt_SolverInfo;
        rtM_hilestadocomatraso56exc01->solverInfo =
(&rt_SolverInfo);

        rtsiSetSimTimeStepPtr(rtM_hilestadocomatraso56exc
01->solverInfo,
            &rtM_hilestadocomatraso56exc01->
>Timing.simTimeStep);
        rtsiSetTPtr(rtM_hilestadocomatraso56exc01->
>solverInfo,
            &rtmGetTPtr(rtM_hilestadocomatraso56exc01));
        rtsiSetStepSizePtr(rtM_hilestadocomatraso56exc01->
>solverInfo,
            &rtM_hilestadocomatraso56exc01->
>Timing.stepSize);
    }
}

```

```

    rtsiSetdXPtr(rtM_hilestadocomatraso56exc01-
>solverInfo,
    &rtM_hilestadocomatraso56exc01-
>ModelData.derivs);

rtsiSetContStatesPtr(rtM_hilestadocomatraso56exc01
->solverInfo,
    &rtM_hilestadocomatraso56exc01-
>ModelData.contStates);

rtsiSetNumContStatesPtr(rtM_hilestadocomatraso56e
xc01->solverInfo,
    &rtM_hilestadocomatraso56exc01-
>Sizes.numContStates);

rtsiSetErrorStatusPtr(rtM_hilestadocomatraso56exc01
->solverInfo,

&rtmGetErrorStatus(rtM_hilestadocomatraso56exc01
));

rtsiSetRTModelPtr(rtM_hilestadocomatraso56exc01-
>solverInfo,
    rtM_hilestadocomatraso56exc01);
}

/* 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, 1 *
sizeof(int_T));
        (void)memset((char_T *)&mdlSampleHits[0], 0, 1
* sizeof(int_T));

        rtM_hilestadocomatraso56exc01-
>Timing.sampleTimes = (&mdlPeriod[0]);
        rtM_hilestadocomatraso56exc01-
>Timing.offsetTimes = (&mdlOffset[0]);
        rtM_hilestadocomatraso56exc01-
>Timing.sampleTimeTaskIDPtr = (&mdlTsMap[0]);

```

```

    rtmSetTPtr(rtM_hilestadocomatraso56exc01,
&mdlTaskTimes[0]);
    rtM_hilestadocomatraso56exc01-
>Timing.sampleHits = (&mdlSampleHits[0]);
}

rtsiSetSolverMode(rtM_hilestadocomatraso56exc01-
>solverInfo,
    SOLVER_MODE_SINGLETASKING);

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

/* block I/O */
{
    void *b = (void *) &rtB;
    rtM_hilestadocomatraso56exc01-
>ModelData.blockIO = (b);

    {
        int_T i;

        b =&rtB.Discrete_State_Space;
        for (i = 0; i < 13; i++) {
            ((real_T*)b)[i] = 0.0;
        }
    }

    /* external outputs */
    {
        rtM_hilestadocomatraso56exc01-
>ModelData.outputs = (&rtY);

        rtY.volts = 0.0;
        rtY.for_total_in = 0.0;
        rtY.forca_shec = 0.0;
    }

    /* parameters */
    rtM_hilestadocomatraso56exc01-
>ModelData.defaultParam = ((real_T *) &rtP);

    /* data type work */
    {
        void *dwork = (void *) &rtDWork;
        rtM_hilestadocomatraso56exc01->Work.dwork =
(dwork);
        (void)memset((char_T *) dwork, 0,
sizeof(D_Work));
        {
            int_T i;

```

```

    real_T *dwork_ptr = (real_T *)
    &rtDWork.Discrete_State_Space_DSTATE[0];

    for (i = 0; i < 4; 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_hilestadocomatraso56exc01-
>SpecialInfo.mappingInfo = (&dtInfo);

    dtInfo.numDataTypes = 14;
    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_hilestadocomatraso56exc01->modelName =
("hilestadocomatraso56exc01");
rtM_hilestadocomatraso56exc01->path =
("hilestadocomatraso56exc01");

rtmSetTStart(rtM_hilestadocomatraso56exc01, 0.0);
rtM_hilestadocomatraso56exc01->Timing.tFinal =
(-1);
rtM_hilestadocomatraso56exc01->Timing.stepSize
= (0.1);

```

```

rtM_hilestadocomatraso56exc01->solverInfo, 0.1);

rtM_hilestadocomatraso56exc01-
>Sizes.checksums[0] = (685574356U);
rtM_hilestadocomatraso56exc01-
>Sizes.checksums[1] = (1605301032U);
rtM_hilestadocomatraso56exc01-
>Sizes.checksums[2] = (458317466U);
rtM_hilestadocomatraso56exc01-
>Sizes.checksums[3] = (2531217308U);

{
    static const EnableStates rtAlwaysEnabled =
SUBSYS_ENABLED;

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

    rtM_hilestadocomatraso56exc01->extModeInfo =
(&rt_ExtModeInfo);

    rteiSetSubSystemModeVectorAddresses(&rt_ExtMo
deInfo, sysModes);

    sysModes[0] = &rtAlwaysEnabled;

    rteiSetModelMappingInfoPtr(&rt_ExtModeInfo,
    &rtM_hilestadocomatraso56exc01-
>SpecialInfo.mappingInfo);

    rteiSetChecksumsPtr(&rt_ExtModeInfo,
    rtM_hilestadocomatraso56exc01-
>Sizes.checksums);

    rteiSetTPtr(&rt_ExtModeInfo,
    rtmGetTPtr(rtM_hilestadocomatraso56exc01));
}

return rtM_hilestadocomatraso56exc01;
}

```

ANEXO B – CODIGO CORRIGIDO MODELO 1 PROPOSTA 1

```

/*
 * Real-Time Workshop code generation for Simulink
model "hilestado.mdl".
 *
 * Model Version          : 1.726
 * Real-Time Workshop file version : 5.0 $Date:
2002/05/30 19:21:33 $
 * Real-Time Workshop file generated on : Mon Jan
23 22:48:15 2012
 * TLC version           : 5.0 (Jun 18 2002)
 * C source code generated on : Mon Jan 23
22:48:16 2012
 */

#include <math.h>
#include <string.h>
#include "hilestado.h"
#include "hilestado_private.h"
#include "ext_work.h"

#include "hilestado_dt.h"

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

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

/* External output (root outputs fed by signals with
auto storage) */
ExternalOutputs rtY;

/* Parent Simstruct */
static rtModel_hilestado model_S;
rtModel_hilestado *const rtM_hilestado = &model_S;

/* Declaration needed for RTW malloc() usage */
const char
*RT_MEMORY_ALLOCATION_ERROR = "RTW
memory allocation error.";

/* Initial conditions for root system: '<Root>' */
void MdlInitialize(void)
{

/* DiscreteStateSpace Block: <Root>/Discrete State-
Space */
rtDWork.Discrete_State_Space_DSTATE[0] =
rtP.Discrete_State_Space_X0[0];
rtDWork.Discrete_State_Space_DSTATE[1] =
rtP.Discrete_State_Space_X0[1];
rtB.Sine_Wave=0;

```

```

}

/* Enable for root system: '<Root>' */
void MdlEnable(void)
{

/* Sin Block: <Root>/Sine Wave */
rtDWork.Sine_Wave_IWORK.SystemEnable =
(int_T) TRUE;
}

/* Start for root system: '<Root>' */
void MdlStart(void)
{

/* dSPACE I/O Board DS1102 Unit:DAC */
ds1102_da(2, 0);

MdlInitialize();
MdlEnable();
}

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

    rtB.S.Function2 b = (real_T) ds1102_ad(2);
    rtB.Ganho_forca = rtB.S.Function2 b *
rtP.Ganho_forca Gain;
    rtB.Sum6 = rtB.Ganho_forca -
rtP.Constante_soma_forca Value;
    rtB.Sum2 = rtB.Sine_Wave -
rtP.forca_set_point_kN Value - rtB.Sum6;
    rtB.m = rtB.Sum2 * rtP.m Gain;

    {
        static real_T xnew[2];
        xnew[0] =
(rtP.Discrete_State_Space_B[0])*rtB.m;
        xnew[0] +=

(rtP.Discrete_State_Space_A[0])*rtP.Discrete_State
Space_X0[0]
        +
(rtP.Discrete_State_Space_A[1])*rtP.Discrete_State
Space_X0[1];

        xnew[1] =
(rtP.Discrete_State_Space_B[1])*rtB.m;
        xnew[1] +=

```

```
(rtP.Discrete_State_Space_A[2])*rtP.Discrete_State_Space_X0[1];
```

```
(void)memcpy(&rtDWork.Discrete_State_Space_DST_ATE[0], xnew, sizeof(real_T)*2);  
_l
```

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

```
/* DiscreteStateSpace: '<Root>/Discrete State-Space' */  
{  
    rtB.Discrete_State_Space =
```

```
    rtP.Discrete_State_Space_C*rtDWork.Discrete_State_Space_DSTATE[0];  
}
```

```
/* Gain: '<Root>/m to mm'  
*  
* Regarding '<Root>/m to mm':  
* Gain value: rtP.m_to_mm_Gain  
*/  
rtB.m_to_mm = rtB.Discrete_State_Space *  
rtP.m_to_mm_Gain;
```

```
/* Sum: '<Root>/Sum1' incorporates:  
* Constant: '<Root>/deslo set point mm'  
*/  
rtB.Sum1 = rtB.m_to_mm +  
rtP.deslo_set_point_mm_Value;
```

```
/* Gain: '<Root>/gain'  
*  
* Regarding '<Root>/gain':  
* Gain value: rtP.gain_Gain  
*/  
rtB.gain = rtB.Sum1 * rtP.gain_Gain;
```

```
/* Output: '<Root>/volts' */  
rtY.volts = rtB.gain;
```

```
/* dSPACE I/O Board DS1102 Unit:DAC */  
ds1102_da(2, rtB.gain);
```

```
/* dSPACE I/O Board DS1102 Unit:ADC */  
rtB.S_Function2_b = (real_T) ds1102_ad(2);
```

```
/* Gain: '<Root>/Ganho forza'  
*  
* Regarding '<Root>/Ganho forza':  
* Gain value: rtP.Ganho_forca_Gain  
*/  
rtB.Ganho_forca = rtB.S_Function2_b *  
rtP.Ganho_forca_Gain;
```

```
/* Sum: '<Root>/Sum6' incorporates:  
* Constant: '<Root>/Constante soma forza'  
*/  
rtB.Sum6 = rtB.Ganho_forca -  
rtP.Constante_soma_forca_Value;
```

```
/* Sin: '<Root>/Sine Wave' */  
/* check enable state */  
if (  
(int_T)rtDWork.Sine_Wave_IWORK.SystemEnable  
) {  
    rtDWork.Sine_Wave_RWORK.LastSin =  
    sin(rtP.Sine_Wave_Freq *  
        rtmGetT(rtm_hilestado));  
    rtDWork.Sine_Wave_RWORK.LastCos =  
    cos(rtP.Sine_Wave_Freq *  
        rtmGetT(rtm_hilestado));  
    rtDWork.Sine_Wave_IWORK.SystemEnable =  
(int_T)FALSE;  
}
```

```
/* output sine */  
  
rtB.Sine_Wave =  
    rtP.Sine_Wave_Amp *  
    (  
        (rtDWork.Sine_Wave_RWORK.LastSin *  
        rtP.Sine_Wave_PCos  
        + rtDWork.Sine_Wave_RWORK.LastCos *  
        (rtP.Sine_Wave_PSin)) *  
        rtP.Sine_Wave_HCos  
        + (rtDWork.Sine_Wave_RWORK.LastCos *  
        rtP.Sine_Wave_PCos  
        - rtDWork.Sine_Wave_RWORK.LastSin *  
        (rtP.Sine_Wave_PSin)) *  
        rtP.Sine_Wave_Hsin  
        ) + rtP.Sine_Wave_Bias;
```

```
/* Sum: '<Root>/Sum2' incorporates:  
* Constant: '<Root>/forca set point kN'  
*/  
rtB.Sum2 = rtB.Sine_Wave -  
rtP.forca_set_point_kN_Value - rtB.Sum6;
```

```
/* Gain: '<Root>/m'  
*  
* Regarding '<Root>/m':
```

```

* Gain value: rtP.m_Gain
*/
rtB.m = rtB.Sum2 * rtP.m_Gain;

/* Outport: '<Root>/for total in' */
rtY.for_total_in = rtB.m;

/* Outport: '<Root>/forca shec' */
rtY.forca_shec = rtB.Sum6;

/* user code (Output function Trailer) */
/* dSPACE Data Capture block: (none) */
/* ... Service number: 1 */
/* ... Service name: (default) */
{
    host_service(1, 0)
}

/* 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);

    rtP.Discrete_State_Space_X0[0]=rtDWork.Discrete
State_Space_DSTATE[0];

    rtP.Discrete_State_Space_X0[1]=rtDWork.Discrete
State_Space_DSTATE[1];

    /* DiscreteStateSpace Block: <Root>/Discrete State-
    Space */
    {
        static real_T xnew[2];
        xnew[0] = (rtP.Discrete_State_Space_B[0])*rtB.m;
        xnew[0] +=

(rtP.Discrete_State_Space_A[0])*rtDWork.Discrete_
State_Space_DSTATE[0]
        +
(rtP.Discrete_State_Space_A[1])*rtDWork.Discrete_
State_Space_DSTATE[1];

        xnew[1] = (rtP.Discrete_State_Space_B[1])*rtB.m;
        xnew[1] +=

(rtP.Discrete_State_Space_A[2])*rtDWork.Discrete_
State_Space_DSTATE[1];

```

```

(void)memcpy(&rtDWork.Discrete_State_Space_DS
TATE[0], xnew,
            sizeof(real_T)*2);
    }

/* Sin Block: <Root>/Sine Wave */
{
    real_T hold_sin;
    real_T hold_cos;

    hold_sin =
rtDWork.Sine_Wave_RWORK.LastSin;
    hold_cos =
rtDWork.Sine_Wave_RWORK.LastCos;

    rtDWork.Sine_Wave_RWORK.LastSin = hold_sin
* rtP.Sine_Wave_HCos + hold_cos *
    rtP.Sine_Wave_Hsin;
    rtDWork.Sine_Wave_RWORK.LastCos =
hold_cos * rtP.Sine_Wave_HCos - hold_sin *
    rtP.Sine_Wave_Hsin;
}

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

/* Function to initialize sizes */
void MdlInitializeSizes(void)
{
    rtM_hilestado->Sizes.numContStates = (0); /*
Number of continuous states */
    rtM_hilestado->Sizes.numY = (3); /* Number of
model outputs */
    rtM_hilestado->Sizes.numU = (0); /* Number of
model inputs */
    rtM_hilestado->Sizes.sysDirFeedThru = (0); /* The
model is not direct feedthrough */
    rtM_hilestado->Sizes.numSampTimes = (1); /*
Number of sample times */
    rtM_hilestado->Sizes.numBlocks = (23); /* Number
of blocks */
    rtM_hilestado->Sizes.numBlockIO = (13); /*
Number of block outputs */
    rtM_hilestado->Sizes.numBlockPrms = (22); /* Sum
of parameter "widths" */
}

/* Function to initialize sample times */
void MdlInitializeSampleTimes(void)

```



```

{
    /* task periods */
    rtM_hilestado->Timing.sampleTimes[0] = (0.1);

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

/* Function to register the model */
rtModel_hilestado *hilestado(void)
{
    (void)memset((char *)rtM_hilestado, 0,
    sizeof(rtModel_hilestado));

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

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

        rtsiSetRTModelPtr(rtM_hilestado->solverInfo,
        rtM_hilestado);
    }

    /* 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, 1 *
    sizeof(int_T));
    (void)memset((char_T *)&mdlSampleHits[0], 0, 1
    * sizeof(int_T));

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

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

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

        {
            int_T i;

            b = &rtB.Discrete_State_Space;
            for (i = 0; i < 13; i++) {
                ((real_T*)b)[i] = 0.0;
            }
        }
    }

    /* external outputs */
    {
        rtM_hilestado->ModelData.outputs = (&rtY);

        rtY.volts = 0.0;
        rtY.for_total_in = 0.0;
        rtY.forca_shec = 0.0;
    }

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

    /* data type work */
    {
        void *dwork = (void *) &rtDWork;
    }
}

```

```

rtM_hilestado->Work.dwork = (dwork);
(void)memset((char_T *) dwork, 0,
sizeof(D_Work));
{
    int_T i;
    real_T *dwork_ptr = (real_T *)
&rtDWork.Discrete_State_Space_DSTATE[0];

    for (i = 0; i < 4; 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_hilestado->SpecialInfo.mappingInfo =
(&dtInfo);

    dtInfo.numDataTypes = 14;
    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_hilestado->modelName = ("hilestado");
rtM_hilestado->path = ("hilestado");

rtmSetTStart(rtM_hilestado, 0.0);

```

```

rtM_hilestado->Timing.tFinal = (-1);
rtM_hilestado->Timing.stepSize = (0.1);
rtsiSetFixedStepSize(rtM_hilestado->solverInfo,
0.1);

rtM_hilestado->Sizes.checksums[0] =
(2955614308U);
rtM_hilestado->Sizes.checksums[1] =
(4257650311U);
rtM_hilestado->Sizes.checksums[2] =
(1383797446U);
rtM_hilestado->Sizes.checksums[3] =
(3793418720U);

{
    static const EnableStates rtAlwaysEnabled =
SUBSYS_ENABLED;

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

    rtM_hilestado->extModeInfo =
(&rt_ExtModeInfo);

    rteiSetSubSystemModeVectorAddresses(&rt_ExtMo
deInfo, sysModes);

    sysModes[0] = &rtAlwaysEnabled;

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

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

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

return rtM_hilestado;

```

ANEXO C – CODIGO GERADO MODELO 1 PROPOSTA 2

```

/*
 * Real-Time Workshop code generation for Simulink
 * model "hilestado.mdl".
 *
 * Model Version          : 1.724
 * Real-Time Workshop file version : 5.0 $Date:
 * 2002/05/30 19:21:33 $
 * Real-Time Workshop file generated on : Mon Jan
 * 23 21:26:06 2012
 * TLC version           : 5.0 (Jun 18 2002)
 * C source code generated on : Mon Jan 23
 * 21:26:07 2012
 */

#include <math.h>
#include <string.h>
#include "hilestado.h"
#include "hilestado_private.h"
#include "ext_work.h"

#include "hilestado_dt.h"

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

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

/* External output (root outputs fed by signals with
auto storage) */
ExternalOutputs rtY;

/* Parent Simstruct */
static rtModel_hilestado model_S;
rtModel_hilestado *const rtM_hilestado = &model_S;

/* Declaration needed for RTW malloc() usage */
const char
*RT_MEMORY_ALLOCATION_ERROR = "RTW
memory allocation error.";

/* Initial conditions for root system: '<Root>' */
void MdlInitialize(void)
{
    /* DiscreteStateSpace Block: <Root>/Discrete State-
    Space */
    rtDWork.Discrete_State_Space_DSTATE[0] =
    rtP.Discrete_State_Space_X0[0];
    rtDWork.Discrete_State_Space_DSTATE[1] =
    rtP.Discrete_State_Space_X0[1];
}

/* Enable for root system: '<Root>' */

void MdlEnable(void)
{
    /* Sin Block: <Root>/Sine Wave */
    rtDWork.Sine_Wave_IWORK.SystemEnable =
    (int_T) TRUE;
}

/* Start for root system: '<Root>' */
void MdlStart(void)
{
    /* dSPACE I/O Board DS1102 Unit:DAC */
    ds1102_da(2, 0);

    MdlInitialize();
    MdlEnable();
}

/* Outputs for root system: '<Root>' */
void MdlOutputs(int_T tid)
{
    if (rtMIsSampleHit(rtM_hilestado, 1, tid)) { /*
    Sample time: [0.1, 0.0] */

        /* DiscreteStateSpace: '<Root>/Discrete State-
        Space' */
        {
            rtB.Discrete_State_Space =

            rtP.Discrete_State_Space_C*rtDWork.Discrete_State
            _Space_DSTATE[0];
        }

        /* Gain: '<Root>/m to mm'
        *
        * Regarding '<Root>/m to mm':
        * Gain value: rtP.m_to_mm_Gain
        */
        rtB.m_to_mm = rtB.Discrete_State_Space *
        rtP.m_to_mm_Gain;

        /* Sum: '<Root>/Sum1' incorporates:
        * Constant: '<Root>/deslo set point mm'
        */
        rtB.Sum1 = rtB.m_to_mm +
        rtP.deslo_set_point_mm_Value;

        /* Gain: '<Root>/gain'
        *
        * Regarding '<Root>/gain':
        * Gain value: rtP.gain_Gain
        */
    }
}

```

```

rtB.gain = rtB.Sum1 * rtP.gain_Gain;

/* Output: '<Root>/volts' */
rtY.volts = rtB.gain;

/* dSPACE I/O Board DS1102 Unit:DAC */
ds1102_da(2, rtB.gain);
}

if (rtmIsSampleHit(rtM_hilestado, 0, tid)) { /*
Sample time: [0.01, 0.0] */

/* dSPACE I/O Board DS1102 Unit:ADC */
rtB.S_Function2_b = (real_T) ds1102_ad(2);

/* Gain: '<Root>/Ganho forza'
*
* Regarding '<Root>/Ganho forza':
* Gain value: rtP.Ganho_forca_Gain
*/
rtB.Ganho_forca = rtB.S_Function2_b *
rtP.Ganho_forca_Gain;

/* Sum: '<Root>/Sum6' incorporates:
* Constant: '<Root>/Constante soma forza'
*/
rtB.Sum6 = rtB.Ganho_forca -
rtP.Constante_soma_forca_Value;
}

if (rtmIsSampleHit(rtM_hilestado, 1, tid)) { /*
Sample time: [0.1, 0.0] */

/* Sin: '<Root>/Sine Wave' */
/* check enable state */
if (
(int_T)rtDWork.Sine_Wave_IWORK.SystemEnable
) {
    rtDWork.Sine_Wave_RWORK.LastSin =
sin(rtP.Sine_Wave_Freq *
    rtmGetTaskTime(rtM_hilestado, tid));
    rtDWork.Sine_Wave_RWORK.LastCos =
cos(rtP.Sine_Wave_Freq *
    rtmGetTaskTime(rtM_hilestado, tid));
    rtDWork.Sine_Wave_IWORK.SystemEnable =
(int_T)FALSE;
}

/* output sine */

rtB.Sine_Wave =
rtP.Sine_Wave_Amp *
(
    (rtDWork.Sine_Wave_RWORK.LastSin *
rtP.Sine_Wave_PCos

```

```

    + rtDWork.Sine_Wave_RWORK.LastCos *
(rtP.Sine_Wave_PSin)) *
    rtP.Sine_Wave_HCos
    + (rtDWork.Sine_Wave_RWORK.LastCos *
rtP.Sine_Wave_PCos
    - rtDWork.Sine_Wave_RWORK.LastSin *
(rtP.Sine_Wave_PSin)) *
    rtP.Sine_Wave_Hsin
    ) + rtP.Sine_Wave_Bias;
}

if (rtmIsSampleHit(rtM_hilestado, 0, tid)) { /*
Sample time: [0.01, 0.0] */

/* Sum: '<Root>/Sum2' incorporates:
* Constant: '<Root>/forca set point kN'
*/
rtB.Sum2 = rtB.Sine_Wave -
rtP.forca_set_point_kN_Value - rtB.Sum6;

/* Gain: '<Root>/m'
*
* Regarding '<Root>/m':
* Gain value: rtP.m_Gain
*/
rtB.m = rtB.Sum2 * rtP.m_Gain;

/* Output: '<Root>/for total in' */
rtY.for_total_in = rtB.m;

/* Output: '<Root>/forca shec' */
rtY.forca_shec = rtB.Sum6;
}

/* user code (Output function Trailer) */
/* dSPACE Data Capture block: (none) */
/* ... Service number: 1 */
/* ... Service name: (default) */
if (rtmIsSampleHit(rtM_hilestado, 0, tid)) {
    host_service(1, 0)
}
}

/* Update for root system: '<Root>' */
void MdlUpdate(int_T tid)
{
    if (rtmIsSampleHit(rtM_hilestado, 0, tid)) { /*
Sample time: [0.01, 0.0] */

/* DiscreteStateSpace Block: <Root>/Discrete
State-Space */
{
    static real_T xnew[2];
    xnew[0] =
(rtP.Discrete_State_Space_B[0])*rtB.m;

```

```

    xnew[0] +=

(rtP.Discrete_State_Space_A[0])*rtP.Discrete_State_
Space_X0[0]
    +
(rtP.Discrete_State_Space_A[1])*rtP.Discrete_State_
Space_X0[1];

    xnew[1] =
(rtP.Discrete_State_Space_B[1])*rtB.m;
    xnew[1] +=

(rtP.Discrete_State_Space_A[2])*rtP.Discrete_State_
Space_X0[1];

(void)memcpy(&rtDWork.Discrete_State_Space_DS
TATE[0], xnew,
    sizeof(real_T)*2);
}

}

if (rtmIsSampleHit(rtM_hilestado, 1, tid)) { /*
Sample time: [0.1, 0.0] */

    /* DiscreteStateSpace Block: <Root>/Discrete
State-Space */
    {
        static real_T xnew[2];

rtP.Discrete_State_Space_X0[0]=rtDWork.Discrete_
State_Space_DSTATE[0];

rtP.Discrete_State_Space_X0[1]=rtDWork.Discrete_
State_Space_DSTATE[1];
        xnew[0] =
(rtP.Discrete_State_Space_B[0])*rtB.m;
        xnew[0] +=

(rtP.Discrete_State_Space_A[0])*rtDWork.Discrete_
State_Space_DSTATE[0]
        +
(rtP.Discrete_State_Space_A[1])*rtDWork.Discrete_
State_Space_DSTATE[1];

        xnew[1] =
(rtP.Discrete_State_Space_B[1])*rtB.m;
        xnew[1] +=

(rtP.Discrete_State_Space_A[2])*rtDWork.Discrete_
State_Space_DSTATE[1];

        (void)memcpy(&rtDWork.Discrete_State_Space_DS
TATE[0], xnew,
            sizeof(real_T)*2);
    }
}

```

```

/* Sin Block: <Root>/Sine Wave */
{
    real_T hold_sin;
    real_T hold_cos;

    hold_sin =
rtDWork.Sine_Wave_RWORK.LastSin;
    hold_cos =
rtDWork.Sine_Wave_RWORK.LastCos;

    rtDWork.Sine_Wave_RWORK.LastSin =
hold_sin * rtP.Sine_Wave_HCos + hold_cos
    * rtP.Sine_Wave_Hsin;
    rtDWork.Sine_Wave_RWORK.LastCos =
hold_cos * rtP.Sine_Wave_HCos - hold_sin
    * rtP.Sine_Wave_Hsin;
}
}

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

/* Function to initialize sizes */
void MdlInitializeSizes(void)
{
    rtM_hilestado->Sizes.numContStates = (0); /*
Number of continuous states */
    rtM_hilestado->Sizes.numY = (3); /* Number of
model outputs */
    rtM_hilestado->Sizes.numU = (0); /* Number of
model inputs */
    rtM_hilestado->Sizes.sysDirFeedThru = (0); /* The
model is not direct feedthrough */
    rtM_hilestado->Sizes.numSampTimes = (2); /*
Number of sample times */
    rtM_hilestado->Sizes.numBlocks = (24); /* Number
of blocks */
    rtM_hilestado->Sizes.numBlockIO = (13); /*
Number of block outputs */
    rtM_hilestado->Sizes.numBlockPrms = (22); /* Sum
of parameter "widths" */
}

/* Function to initialize sample times */
void MdlInitializeSampleTimes(void)
{
    /* task periods */
    rtM_hilestado->Timing.sampleTimes[0] = (0.01);
    rtM_hilestado->Timing.sampleTimes[1] = (0.1);
}

```

```

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

/* Function to register the model */
rtModel_hilestado *hilestado(void)
{
    (void)memset((char *)rtM_hilestado, 0,
sizeof(rtModel_hilestado));

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

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

        rtsiSetRTModelPtr(rtM_hilestado->solverInfo,
rtM_hilestado);
    }

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

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

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

        {
            int_T i;

            b = &rtB.Discrete_State_Space;
            for (i = 0; i < 13; i++) {
                ((real_T*)b)[i] = 0.0;
            }
        }
    }

    /* external outputs */
    {
        rtM_hilestado->ModelData.outputs = (&rtY);

        rtY.volts = 0.0;
        rtY.for_total_in = 0.0;
        rtY.forca_shec = 0.0;
    }

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

    /* data type work */
    {
        void *dwork = (void *) &rtDWork;
        rtM_hilestado->Work.dwork = (dwork);
        (void)memset((char_T *) dwork, 0,
sizeof(D_Work));
    }
}

```

```

    int_T i;
    real_T *dwork_ptr = (real_T *)
&rtDWork.Discrete_State_Space_DSTATE[0];

    for (i = 0; i < 4; 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_hilestado->SpecialInfo.mappingInfo =
(&dtInfo);

    dtInfo.numDataTypes = 14;
    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_hilestado->modelName = ("hilestado");
rtM_hilestado->path = ("hilestado");

rtmSetTStart(rtM_hilestado, 0.0);
rtM_hilestado->Timing.tFinal = (-1);
rtM_hilestado->Timing.stepSize = (0.01);

```

```

    rtsiSetFixedStepSize(rtM_hilestado->solverInfo,
0.01);

    rtM_hilestado->Sizes.checksums[0] =
(1386391271U);
    rtM_hilestado->Sizes.checksums[1] =
(2858818304U);
    rtM_hilestado->Sizes.checksums[2] =
(1537771595U);
    rtM_hilestado->Sizes.checksums[3] =
(3302005126U);

    {
        static const EnableStates rtAlwaysEnabled =
SUBSYS_ENABLED;

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

        rtM_hilestado->extModeInfo =
(&rt_ExtModeInfo);

        rteiSetSubSystemModeVectorAddresses(&rt_ExtMo
deInfo, sysModes);

        sysModes[0] = &rtAlwaysEnabled;

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

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

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

    return rtM_hilestado;
}

```

ANEXO D – CODIGO CORRIGIDO MODELO 1 PROPOSTA 2

```

/*
 * Real-Time Workshop code generation for Simulink
 model "hilestado.mdl".
 *c
 * Model Version          : 1.724
 * Real-Time Workshop file version : 5.0 $Date:
 2002/05/30 19:21:33 $
 * Real-Time Workshop file generated on : Mon Feb
 06 14:35:23 2012
 * TLC version            : 5.0 (Jun 18 2002)
 * C source code generated on : Mon Feb 06
 14:35:23 2012
 */

#include <math.h>
#include <string.h>
#include "hilestado.h"
#include "hilestado_private.h"
#include "ext_work.h"

#include "hilestado_dt.h"

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

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

/* External output (root outputs fed by signals with
auto storage) */
ExternalOutputs rtY;

/* Parent Simstruct */
static rtModel_hilestado model_S;
rtModel_hilestado *const rtM_hilestado = &model_S;

/* Declaration needed for RTW malloc() usage */
const char
*RT_MEMORY_ALLOCATION_ERROR = "RTW
memory allocation error.";

/* Initial conditions for root system: '<Root>' */
void MdlInitialize(void)
{

    /* DiscreteStateSpace Block: <Root>/Discrete State-
Space */
    rtDWork.Discrete_State_Space_DSTATE[0] =
rtP.Discrete_State_Space_X0[0];
    rtDWork.Discrete_State_Space_DSTATE[1] =
rtP.Discrete_State_Space_X0[1];
}

/* Enable for root system: '<Root>' */
void MdlEnable(void)
{

    /* Sin Block: <Root>/Sine Wave */
    rtDWork.Sine_Wave_IWORK.SystemEnable =
(int_T) TRUE;
}

/* Start for root system: '<Root>' */
void MdlStart(void)
{

    /* dSPACE I/O Board DS1102 Unit:DAC */
    ds1102_da(2, 0);

    MdlInitialize();
    MdlEnable();
}

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

    if (rtmIsSampleHit(rtM_hilestado, 1, tid)) { /*
Sample time: [0.1, 0.0] */

        /* DiscreteStateSpace: '<Root>/Discrete State-
Space' */
        {
            rtB.Discrete_State_Space =

rtP.Discrete_State_Space_C*rtDWork.Discrete_State
_Space_DSTATE[0];
        }

        /* Gain: '<Root>/m to mm'
*
* Regarding '<Root>/m to mm':
* Gain value: rtP.m_to_mm_Gain
*/
        rtB.m_to_mm = rtB.Discrete_State_Space *
rtP.m_to_mm_Gain;

        /* Sum: '<Root>/Sum1' incorporates:
* Constant: '<Root>/deslo set point mm'
*/
        rtB.Sum1 = rtB.m_to_mm +
rtP.deslo_set_point_mm_Value;

        /* Gain: '<Root>/gain'
*
* Regarding '<Root>/gain':

```



```

    * Gain value: rtP.gain_Gain
    */
    rtB.gain = rtB.Sum1 * rtP.gain_Gain;

    /* Output: '<Root>/volts' */
    rtY.volts = rtB.gain;

    /* dSPACE I/O Board DS1102 Unit:DAC */
    ds1102_da(2, rtB.gain);
}

if (rtmIsSampleHit(rtM_hilestado, 0, tid)) { /*
Sample time: [0.01, 0.0] */

    /* dSPACE I/O Board DS1102 Unit:ADC */
    rtB.S_Function2_b = (real_T) ds1102_ad(2);

    /* Gain: '<Root>/Ganho forca'
    *
    * Regarding '<Root>/Ganho forca':
    * Gain value: rtP.Ganho_forca_Gain
    */
    rtB.Ganho_forca = rtB.S_Function2_b *
rtP.Ganho_forca_Gain;

    /* Sum: '<Root>/Sum6' incorporates:
    * Constant: '<Root>/Constante soma forca'
    */
    rtB.Sum6 = rtB.Ganho_forca -
rtP.Constante_soma_forca_Value;
}

if (rtmIsSampleHit(rtM_hilestado, 1, tid)) { /*
Sample time: [0.1, 0.0] */

    /* Sin: '<Root>/Sine Wave' */
    /* check enable state */
    if (
(int_T)rtDWork.Sine_Wave_IWORK.SystemEnable
) {
        rtDWork.Sine_Wave_RWORK.LastSin =
sin(rtP.Sine_Wave_Freq *
    rtmGetTaskTime(rtM_hilestado, tid));
        rtDWork.Sine_Wave_RWORK.LastCos =
cos(rtP.Sine_Wave_Freq *
    rtmGetTaskTime(rtM_hilestado, tid));
        rtDWork.Sine_Wave_IWORK.SystemEnable =
(int_T)FALSE;
    }

    /* output sine */

    rtB.Sine_Wave =
    rtP.Sine_Wave_Amp *
    (

```

```

        (rtDWork.Sine_Wave_RWORK.LastSin *
rtP.Sine_Wave_PCos
        + rtDWork.Sine_Wave_RWORK.LastCos *
(rtP.Sine_Wave_PSin)) *
        rtP.Sine_Wave_HCos
        + (rtDWork.Sine_Wave_RWORK.LastCos *
rtP.Sine_Wave_PCos
        - rtDWork.Sine_Wave_RWORK.LastSin *
(rtP.Sine_Wave_PSin)) *
        rtP.Sine_Wave_Hsin
        ) + rtP.Sine_Wave_Bias;
    }

    if (rtmIsSampleHit(rtM_hilestado, 0, tid)) { /*
Sample time: [0.01, 0.0] */

        /* Sum: '<Root>/Sum2' incorporates:
        * Constant: '<Root>/forca set point kN'
        */
        rtB.Sum2 = rtB.Sine_Wave -
rtP.forca_set_point_kN_Value - rtB.Sum6;

        /* Gain: '<Root>/m'
        *
        * Regarding '<Root>/m':
        * Gain value: rtP.m_Gain
        */
        rtB.m = rtB.Sum2 * rtP.m_Gain;

        /* Output: '<Root>/for total in' */
        rtY.for_total_in = rtB.m;

        /* Output: '<Root>/forca shec' */
        rtY.forca_shec = rtB.Sum6;
    }

    /* user code (Output function Trailer) */
    /* dSPACE Data Capture block: (none) */
    /* ... Service number: 1 */
    /* ... Service name: (default) */
    if (rtmIsSampleHit(rtM_hilestado, 0, tid)) {
        host_service(1, 0)
    }
}

/* Update for root system: '<Root>' */
void MdlUpdate(int_T tid)
{
    if (rtmIsSampleHit(rtM_hilestado, 1, tid)) { /*
Sample time: [0.1, 0.0] */

```

/* guardei informações do Xk de 0.1 */

rtP.Discrete_State_Space_X0[0]=rtDWork.Discrete_State_Space_DSTATE[0];

rtP.Discrete_State_Space_X0[1]=rtDWork.Discrete_State_Space_DSTATE[1];

/* DiscreteStateSpace Block: <Root>/Discrete State-Space */

```
{
    static real_T xnew[2];
    xnew[0] =
    (rtP.Discrete_State_Space_B[0])*rtB.m;
    xnew[0] +=
```

```
(rtP.Discrete_State_Space_A[0])*rtDWork.Discrete_State_Space_DSTATE[0]
    +
    (rtP.Discrete_State_Space_A[1])*rtDWork.Discrete_State_Space_DSTATE[1];
```

```
    xnew[1] =
    (rtP.Discrete_State_Space_B[1])*rtB.m;
    xnew[1] +=
```

```
(rtP.Discrete_State_Space_A[2])*rtDWork.Discrete_State_Space_DSTATE[1];
```

```
(void)memcpy(&rtDWork.Discrete_State_Space_DS
TATE[0], xnew,
    sizeof(real_T)*2);
}
```

/* Sin Block: <Root>/Sine Wave */

```
{
    real_T hold_sin;
    real_T hold_cos;
```

```
    hold_sin =
    rtDWork.Sine_Wave_RWORK.LastSin;
    hold_cos =
    rtDWork.Sine_Wave_RWORK.LastCos;
```

```
    rtDWork.Sine_Wave_RWORK.LastSin =
    hold_sin * rtP.Sine_Wave_HCos + hold_cos
    * rtP.Sine_Wave_Hsin;
    rtDWork.Sine_Wave_RWORK.LastCos =
    hold_cos * rtP.Sine_Wave_HCos - hold_sin
    * rtP.Sine_Wave_Hsin;
}
}
```

/* calculo a cada 0.01 do proximo xk de 0.1 */

if (rtMIsSampleHit(rtM_hilestado, 0, tid)) { /*
Sample time: [0.01, 0.0] */

static real T xnew[2];
xnew[0] =
(rtP.Discrete_State_Space_B[0])*rtB.m;
xnew[0] +=

(rtP.Discrete_State_Space_A[0])*rtP.Discrete_State_Space_X0[0]
+
(rtP.Discrete_State_Space_A[1])*rtP.Discrete_State_Space_X0[1];

xnew[1] =
(rtP.Discrete_State_Space_B[1])*rtB.m;
xnew[1] +=

(rtP.Discrete_State_Space_A[2])*rtP.Discrete_State_Space_X0[1];

(void)memcpy(&rtDWork.Discrete_State_Space_DST
ATE[0], xnew,
sizeof(real_T)*2);

}

```
}

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

/* Function to initialize sizes */

```
void MdlInitializeSizes(void)
{
    rtM_hilestado->Sizes.numContStates = (0); /*
Number of continuous states */
    rtM_hilestado->Sizes.numY = (3); /* Number of
model outputs */
    rtM_hilestado->Sizes.numU = (0); /* Number of
model inputs */
    rtM_hilestado->Sizes.sysDirFeedThru = (0); /* The
model is not direct feedthrough */
```

```

    rtM_hilestado->Sizes.numSampTimes = (2); /*
Number of sample times */
    rtM_hilestado->Sizes.numBlocks = (24); /* Number
of blocks */
    rtM_hilestado->Sizes.numBlockIO = (13); /*
Number of block outputs */
    rtM_hilestado->Sizes.numBlockPrms = (22); /* Sum
of parameter "widths" */
}

/* Function to initialize sample times */
void MdlInitializeSampleTimes(void)
{
    /* task periods */
    rtM_hilestado->Timing.sampleTimes[0] = (0.01);
    rtM_hilestado->Timing.sampleTimes[1] = (0.1);

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

/* Function to register the model */
rtModel_hilestado *hilestado(void)
{
    (void)memset((char *)rtM_hilestado, 0,
sizeof(rtModel_hilestado));

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

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

        rtsiSetRTModelPtr(rtM_hilestado->solverInfo,
rtM_hilestado);
    }

    /* 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_hilestado->Timing.sampleTimes =
(&mdlPeriod[0]);
    rtM_hilestado->Timing.offsetTimes =
(&mdlOffset[0]);
    rtM_hilestado->Timing.sampleTimeTaskIDPtr =
(&mdlTsMap[0]);
    rtmSetTPtr(rtM_hilestado, &mdlTaskTimes[0]);
    rtM_hilestado->Timing.sampleHits =
(&mdlSampleHits[0]);
}

    rtsiSetSolverMode(rtM_hilestado->solverInfo,
SOLVER_MODE_SINGLETASKING);

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

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

        {
            int_T i;

            b = &rtB.Discrete_State_Space;
            for (i = 0; i < 13; i++) {
                ((real_T*)b)[i] = 0.0;
            }
        }
    }

    /* external outputs */
    {

```

```

rtM_hilestado->ModelData.outputs = (&rtY);

rtY.volts = 0.0;
rtY.for_total_in = 0.0;
rtY.forca_shec = 0.0;
}

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

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

        for (i = 0; i < 4; 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_hilestado->SpecialInfo.mappingInfo =
(&dtInfo);

    dtInfo.numDataTypes = 14;
    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_hilestado->modelName = ("hilestado");
rtM_hilestado->path = ("hilestado");

rtmSetTStart(rtM_hilestado, 0.0);
rtM_hilestado->Timing.tFinal = (-1);
rtM_hilestado->Timing.stepSize = (0.01);
rtsiSetFixedStepSize(rtM_hilestado->solverInfo,
0.01);

rtM_hilestado->Sizes.checksums[0] =
(1386391271U);
rtM_hilestado->Sizes.checksums[1] =
(2858818304U);
rtM_hilestado->Sizes.checksums[2] =
(1537771595U);
rtM_hilestado->Sizes.checksums[3] =
(3302005126U);

{
    static const EnableStates rtAlwaysEnabled =
SUBSYS_ENABLED;

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

    rtM_hilestado->extModeInfo =
(&rt_ExtModeInfo);

    rteiSetSubSystemModeVectorAddresses(&rt_ExtMo
deInfo, sysModes);

    sysModes[0] = &rtAlwaysEnabled;

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

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

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

return rtM_hilestado;
}

```

ANEXO E – CODIGO GERADO MODELO 2 PROPOSTA 1

```

/*
 * Real-Time Workshop code generation for Simulink
model "hilestadocomatrasso70.mdl".
 *
 * Model Version          : 1.743
 * Real-Time Workshop file version : 5.0 $Date:
2002/05/30 19:21:33 $
 * Real-Time Workshop file generated on : Sat Feb 11
20:34:30 2012
 * TLC version           : 5.0 (Jun 18 2002)
 * C source code generated on : Sat Feb 11
20:34:31 2012
 */

#include <math.h>
#include <string.h>
#include "hilestadocomatrasso70.h"
#include "hilestadocomatrasso70_private.h"
#include "ext_work.h"

#include "hilestadocomatrasso70_dt.h"

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

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

/* External output (root outputs fed by signals with
auto storage) */
ExternalOutputs rtY;

/* Parent Simstruct */
static rtModel_hilestadocomatrasso70 model_S;
rtModel_hilestadocomatrasso70 *const
rtM_hilestadocomatrasso70 = &model_S;

/* Declaration needed for RTW malloc() usage */
const char
*RT_MEMORY_ALLOCATION_ERROR = "RTW
memory allocation error.";

/* Initial conditions for root system: '<Root>' */
void MdlInitialize(void)
{
    /* DiscreteStateSpace Block: <Root>/Discrete State-
Space */
    rtDWork.Discrete_State_Space_DSTATE[0] =
rtP.Discrete_State_Space_X0[0];
    rtDWork.Discrete_State_Space_DSTATE[1] =
rtP.Discrete_State_Space_X0[1];
}

/* Enable for root system: '<Root>' */
void MdlEnable(void)
{
    /* Sin Block: <Root>/Sine Wave */
    rtDWork.Sine_Wave_IWORK.SystemEnable =
(int_T) TRUE;
}

/* Start for root system: '<Root>' */
void MdlStart(void)
{
    /* dSPACE I/O Board DS1102 Unit:DAC */
    ds1102_da(2, 0);

    MdlInitialize();
    MdlEnable();
}

/* Outputs for root system: '<Root>' */
void MdlOutputs(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);

    /* DiscreteStateSpace: '<Root>/Discrete State-
Space' */
    {
        rtB.Discrete_State_Space =
rtP.Discrete_State_Space_C*rtDWork.Discrete_State
_Space_DSTATE[0];
    }

    /* Gain: '<Root>/m to mm'
 *
 * Regarding '<Root>/m to mm':
 * Gain value: rtP.m_to_mm_Gain
 */
    rtB.m_to_mm = rtB.Discrete_State_Space *
rtP.m_to_mm_Gain;

    /* Outport: '<Root>/volts' */
    rtY.volts = rtB.m_to_mm;

    /* Sum: '<Root>/Sum1' incorporates:
 * Constant: '<Root>/deslo set point mm'
 */
}

```

```

rtB.Sum1 = rtB.m_to_mm +
rtP.deslo_set_point_mm_Value;

/* Gain: '<Root>/gain'
*
* Regarding '<Root>/gain':
* Gain value: rtP.gain_Gain
*/
rtB.gain = rtB.Sum1 * rtP.gain_Gain;

/* dSPACE I/O Board DS1102 Unit:DAC */
ds1102_da(2, rtB.gain);

/* dSPACE I/O Board DS1102 Unit:ADC */
rtB.S_Function2_b = (real_T) ds1102_ad(2);

/* Gain: '<Root>/Ganho forca'
*
* Regarding '<Root>/Ganho forca':
* Gain value: rtP.Ganho_forca_Gain
*/
rtB.Ganho_forca = rtB.S_Function2_b *
rtP.Ganho_forca_Gain;

/* Sum: '<Root>/Sum6' incorporates:
* Constant: '<Root>/Constante soma forca'
*/
rtB.Sum6 = rtB.Ganho_forca +
rtP.Constante_soma_forca_Value;

/* Sin: '<Root>/Sine Wave' */
/* check enable state */
if (
(int_T)rtDWork.Sine_Wave_IWORK.SystemEnable
) {
    rtDWork.Sine_Wave_RWORK.LastSin =
sin(rtP.Sine_Wave_Freq *
    rtmGetT(rtm_hilestadocomatraso70));
    rtDWork.Sine_Wave_RWORK.LastCos =
cos(rtP.Sine_Wave_Freq *
    rtmGetT(rtm_hilestadocomatraso70));
    rtDWork.Sine_Wave_IWORK.SystemEnable =
(int_T)FALSE;
}

/* output sine */

rtB.Sine_Wave =
rtP.Sine_Wave_Amp *
(
    (rtDWork.Sine_Wave_RWORK.LastSin *
rtP.Sine_Wave_PCos
    + rtDWork.Sine_Wave_RWORK.LastCos *
(rtP.Sine_Wave_PSin)) *
    rtP.Sine_Wave_HCos

```

```

    + (rtDWork.Sine_Wave_RWORK.LastCos *
rtP.Sine_Wave_PCos
    - rtDWork.Sine_Wave_RWORK.LastSin *
(rtP.Sine_Wave_PSin)) *
    rtP.Sine_Wave_Hsin
    ) + rtP.Sine_Wave_Bias;

/* Sum: '<Root>/Sum2' incorporates:
* Constant: '<Root>/forca set point kN'
*/
rtB.Sum2 = rtB.Sine_Wave -
rtP.forca_set_point_kN_Value - rtB.Sum6;

/* Gain: '<Root>/m'
*
* Regarding '<Root>/m':
* Gain value: rtP.m_Gain
*/
rtB.m = rtB.Sum2 * rtP.m_Gain;

/* Output: '<Root>/for total in' */
rtY.for_total_in = rtB.m;

/* Output: '<Root>/forca shec' */
rtY.forca_shec = rtB.Sum6;

/* user code (Output function Trailer) */
/* dSPACE Data Capture block: (none) */
/* ... Service number: 1 */
/* ... Service name: (default) */
{
    host_service(1, 0)
}

/* 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);

/* DiscreteStateSpace Block: <Root>/Discrete State-
Space */
{
    static real_T xnew[2];
    xnew[0] = (rtP.Discrete_State_Space_B[0])*rtB.m;
    xnew[0] +=
(rtP.Discrete_State_Space_A[0])*rtDWork.Discrete_
State_Space_DSTATE[0]

```

```

+
(rtP.Discrete_State_Space_A[1])*rtDWork.Discrete_
State_Space_DSTATE[1];

xnew[1] = (rtP.Discrete_State_Space_B[1])*rtB.m;
xnew[1] +=

(rtP.Discrete_State_Space_A[2])*rtDWork.Discrete_
State_Space_DSTATE[1];

(void)memcpy(&rtDWork.Discrete_State_Space_DS
TATE[0], xnew,
sizeof(real_T)*2);
}

/* Sin Block: <Root>/Sine Wave */
{
    real_T hold_sin;
    real_T hold_cos;

    hold_sin =
rtDWork.Sine_Wave_RWORK.LastSin;
    hold_cos =
rtDWork.Sine_Wave_RWORK.LastCos;

    rtDWork.Sine_Wave_RWORK.LastSin = hold_sin
* rtP.Sine_Wave_HCos + hold_cos *
    rtP.Sine_Wave_Hsin;
    rtDWork.Sine_Wave_RWORK.LastCos =
hold_cos * rtP.Sine_Wave_HCos - hold_sin *
    rtP.Sine_Wave_Hsin;
}
}

/* Terminate for root system: '<Root>' */
void MdlTerminate(void)
{
    if(rtM_hilestadocomatraso70 != NULL) {
    }
}

/* Function to initialize sizes */
void MdlInitializeSizes(void)
{
    rtM_hilestadocomatraso70->Sizes.numContStates =
(0); /* Number of continuous states */
    rtM_hilestadocomatraso70->Sizes.numY = (3); /*
Number of model outputs */
    rtM_hilestadocomatraso70->Sizes.numU = (0); /*
Number of model inputs */
    rtM_hilestadocomatraso70->Sizes.sysDirFeedThru
= (0); /* The model is not direct feedthrough */
    rtM_hilestadocomatraso70->Sizes.numSampTimes
= (1); /* Number of sample times */

```

```

    rtM_hilestadocomatraso70->Sizes.numBlocks =
(23); /* Number of blocks */
    rtM_hilestadocomatraso70->Sizes.numBlockIO =
(13); /* Number of block outputs */
    rtM_hilestadocomatraso70->Sizes.numBlockPrms =
(22); /* Sum of parameter "widths" */
}

/* Function to initialize sample times */
void MdlInitializeSampleTimes(void)
{
    /* task periods */
    rtM_hilestadocomatraso70-
>Timing.sampleTimes[0] = (0.1);

    /* task offsets */
    rtM_hilestadocomatraso70->Timing.offsetTimes[0]
= (0.0);
}

/* Function to register the model */
rtModel_hilestadocomatraso70
*hilestadocomatraso70(void)
{
    (void)memset((char *)rtM_hilestadocomatraso70, 0,
sizeof(rtModel_hilestadocomatraso70));

    {
        /* Setup solver object */
        static RTWSolverInfo rt_SolverInfo;
        rtM_hilestadocomatraso70->solverInfo =
(&rt_SolverInfo);

        rtsiSetSimTimeStepPtr(rtM_hilestadocomatraso70-
>solverInfo,
        &rtM_hilestadocomatraso70-
>Timing.simTimeStep);
        rtsiSetTPtr(rtM_hilestadocomatraso70->solverInfo,
        &rtmGetTPtr(rtM_hilestadocomatraso70));
        rtsiSetStepSizePtr(rtM_hilestadocomatraso70-
>solverInfo,
        &rtM_hilestadocomatraso70->Timing.stepSize);
        rtsiSetdXPtr(rtM_hilestadocomatraso70-
>solverInfo,
        &rtM_hilestadocomatraso70->ModelData.derivs);
        rtsiSetContStatesPtr(rtM_hilestadocomatraso70-
>solverInfo,
        &rtM_hilestadocomatraso70-
>ModelData.contStates);

        rtsiSetNumContStatesPtr(rtM_hilestadocomatraso70-
>solverInfo,
        &rtM_hilestadocomatraso70-
>Sizes.numContStates);
    }
}

```

```

    rtsiSetErrorStatusPtr(rtM_hilestadocomatraso70-
>solverInfo,
    &rtmGetErrorStatus(rtM_hilestadocomatraso70));

    rtsiSetRTModelPtr(rtM_hilestadocomatraso70-
>solverInfo,
    rtM_hilestadocomatraso70);
}

/* 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, 1 *
sizeof(int_T));
    (void)memset((char_T *)&mdlSampleHits[0], 0, 1
* sizeof(int_T));

    rtM_hilestadocomatraso70->Timing.sampleTimes
= (&mdlPeriod[0]);
    rtM_hilestadocomatraso70->Timing.offsetTimes =
(&mdlOffset[0]);
    rtM_hilestadocomatraso70-
>Timing.sampleTimeTaskIDPtr = (&mdlTsMap[0]);
    rtmSetTPtr(rtM_hilestadocomatraso70,
&mdlTaskTimes[0]);
    rtM_hilestadocomatraso70->Timing.sampleHits =
(&mdlSampleHits[0]);
}
    rtsiSetSolverMode(rtM_hilestadocomatraso70-
>solverInfo,
    SOLVER_MODE_SINGLETASKING);

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

/* block I/O */
{
    void *b = (void *) &rtB;

```

```

    rtM_hilestadocomatraso70->ModelData.blockIO =
(b);

    {
        int_T i;

        b = &rtB.Discrete_State_Space;
        for (i = 0; i < 13; i++) {
            ((real_T*)b)[i] = 0.0;
        }
    }

/* external outputs */
{
    rtM_hilestadocomatraso70->ModelData.outputs =
(&rtY);

    rtY.volts = 0.0;
    rtY.for_total_in = 0.0;
    rtY.forca_shec = 0.0;
}

/* parameters */
rtM_hilestadocomatraso70-
>ModelData.defaultParam = ((real_T *) &rtP);

/* data type work */
{
    void *dwork = (void *) &rtDWork;
    rtM_hilestadocomatraso70->Work.dwork =
(dwork);
    (void)memset((char_T *) dwork, 0,
sizeof(D_Work));
    {
        int_T i;
        real_T *dwork_ptr = (real_T *)
&rtDWork.Discrete_State_Space_DSTATE[0];

        for (i = 0; i < 4; 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_hilestadocomatraso70-
>SpecialInfo.mappingInfo = (&dtInfo);

```



```

dtInfo.numDataTypes = 14;
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_hilestadocomatrasso70->modelName =
("hilestadocomatrasso70");
rtM_hilestadocomatrasso70->path =
("hilestadocomatrasso70");

rtmSetTStart(rtM_hilestadocomatrasso70, 0.0);
rtM_hilestadocomatrasso70->Timing.tFinal = (-1);
rtM_hilestadocomatrasso70->Timing.stepSize =
(0.1);
rtsiSetFixedStepSize(rtM_hilestadocomatrasso70-
>solverInfo, 0.1);

rtM_hilestadocomatrasso70->Sizes.checksums[0] =
(3257707441U);
rtM_hilestadocomatrasso70->Sizes.checksums[1] =
(724251010U);
rtM_hilestadocomatrasso70->Sizes.checksums[2] =
(1189394762U);

```

```

rtM_hilestadocomatrasso70->Sizes.checksums[3] =
(865115017U);

{
    static const EnableStates rtAlwaysEnabled =
SUBSYS_ENABLED;

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

    rtM_hilestadocomatrasso70->extModeInfo =
(&rt_ExtModeInfo);

rteiSetSubSystemModeVectorAddresses(&rt_ExtMo
deInfo, sysModes);

    sysModes[0] = &rtAlwaysEnabled;

    rteiSetModelMappingInfoPtr(&rt_ExtModeInfo,
    &rtM_hilestadocomatrasso70-
>SpecialInfo.mappingInfo);

    rteiSetChecksumsPtr(&rt_ExtModeInfo,
    rtM_hilestadocomatrasso70->Sizes.checksums);

    rteiSetTPtr(&rt_ExtModeInfo,
    rtmGetTPtr(rtM_hilestadocomatrasso70));
}

return rtM_hilestadocomatrasso70;
}

```

ANEXO F – CODIGO CORRIGIDO MODELO 2 PROPOSTA 1

```

/*
 * Real-Time Workshop code generation for Simulink
model "hilestadosematrasso70.mdl".
 *
 * Model Version          : 1.744
 * Real-Time Workshop file version : 5.0 $Date:
2002/05/30 19:21:33 $
 * Real-Time Workshop file generated on : Sat Feb 11
21:10:33 2012
 * TLC version            : 5.0 (Jun 18 2002)
 * C source code generated on : Sat Feb 11
21:10:34 2012
 */

#include <math.h>
#include <string.h>
#include "hilestadosematrasso70.h"
#include "hilestadosematrasso70_private.h"
#include "ext_work.h"

#include "hilestadosematrasso70_dt.h"

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

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

/* External output (root outputs fed by signals with
auto storage) */
ExternalOutputs rtY;

/* Parent Simstruct */
static rtModel_hilestadosematrasso70 model_S;
rtModel_hilestadosematrasso70 *const
rtM_hilestadosematrasso70 = &model_S;

/* Declaration needed for RTW malloc() usage */
const char
*RT_MEMORY_ALLOCATION_ERROR = "RTW
memory allocation error.";

/* Initial conditions for root system: '<Root>' */
void MdlInitialize(void)
{

    /* DiscreteStateSpace Block: <Root>/Discrete State-
Space */
    rtDWork.Discrete_State_Space_DSTATE[0] =
rtP.Discrete_State_Space_X0[0];
    rtDWork.Discrete_State_Space_DSTATE[1] =
rtP.Discrete_State_Space_X0[1];
}

/* Enable for root system: '<Root>' */
void MdlEnable(void)
{

    /* Sin Block: <Root>/Sine Wave */
    rtDWork.Sine_Wave_IWORK.SystemEnable =
(int_T) TRUE;
}

/* Start for root system: '<Root>' */
void MdlStart(void)
{

    /* dSPACE I/O Board DS1102 Unit:DAC */
    ds1102_da(2, 0);

    MdlInitialize();
    MdlEnable();
}

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

    rtB.S_Function2_b = (real_T) ds1102_ad(2);
    rtB.Ganho_forca = rtB.S_Function2_b *
rtP.Ganho_forca_Gain;
    rtB.Sum6 = rtB.Ganho_forca -
rtP.Constante_soma_forca_Value;
    rtB.Sum2 = rtB.Sine_Wave -
rtP.forca_set_point_kN_Value - rtB.Sum6;
    rtB.m = rtB.Sum2 * rtP.m_Gain;

    {
        static real_T xnew[2];
        xnew[0] = (rtP.Discrete_State_Space_B[0])*rtB.m;
        xnew[0] +=
        (rtP.Discrete_State_Space_A[0])*rtP.Discrete_State
        Space_X0[0]
        +
        (rtP.Discrete_State_Space_A[1])*rtP.Discrete_State
        Space_X0[1];

        xnew[1] = (rtP.Discrete_State_Space_B[1])*rtB.m;
        xnew[1] +=
        (rtP.Discrete_State_Space_A[2])*rtP.Discrete_State
        Space_X0[1];

        (void)memcpy(&rtDWork.Discrete_State_Space_DS
        TATE[0], xnew,

```

```

    sizeof(real_T)*2);
}

```

```

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

```

```

UNUSED_PARAMETER(tid);

```

```

/* DiscreteStateSpace: '<Root>/Discrete State-
Space' */
{

```

```

    rtB.Discrete_State_Space =

```

```

rtP.Discrete_State_Space_C*rtDWork.Discrete_State
_State_DSTATE[0];
}

```

```

/* Gain: '<Root>/m to mm'

```

```

 *
 * Regarding '<Root>/m to mm':
 * Gain value: rtP.m_to_mm_Gain
 */

```

```

rtB.m_to_mm = rtB.Discrete_State_Space *
rtP.m_to_mm_Gain;

```

```

/* Output: '<Root>/volts' */
rtY.volts = rtB.m_to_mm;

```

```

/* Sum: '<Root>/Sum1' incorporates:
 * Constant: '<Root>/deslo set point mm'
 */

```

```

rtB.Sum1 = rtB.m_to_mm +
rtP.deslo_set_point_mm_Value;

```

```

/* Gain: '<Root>/gain'

```

```

 *
 * Regarding '<Root>/gain':
 * Gain value: rtP.gain_Gain
 */

```

```

rtB.gain = rtB.Sum1 * rtP.gain_Gain;

```

```

/* dSPACE I/O Board DS1102 Unit:DAC */
ds1102_da(2, rtB.gain);

```

```

/* dSPACE I/O Board DS1102 Unit:ADC */
rtB.S_Function2_b = (real_T) ds1102_ad(2);

```

```

/* Gain: '<Root>/Ganho forca'
 *

```

```

 * Regarding '<Root>/Ganho forca':
 * Gain value: rtP.Ganho_forca_Gain
 */

```

```

rtB.Ganho_forca = rtB.S_Function2_b *
rtP.Ganho_forca_Gain;

```

```

/* Sum: '<Root>/Sum6' incorporates:
 * Constant: '<Root>/Constante soma forca'
 */

```

```

rtB.Sum6 = rtB.Ganho_forca +
rtP.Constante_soma_forca_Value;

```

```

/* Sin: '<Root>/Sine Wave' */
/* check enable state */

```

```

if (
(int_T)rtDWork.Sine_Wave_IWORK.SystemEnable
) {
    rtDWork.Sine_Wave_RWORK.LastSin =
sin(rtP.Sine_Wave_Freq *
    rtmGetT(rtm_hilestadosematraso70));
    rtDWork.Sine_Wave_RWORK.LastCos =
cos(rtP.Sine_Wave_Freq *
    rtmGetT(rtm_hilestadosematraso70));
    rtDWork.Sine_Wave_IWORK.SystemEnable =
(int_T)FALSE;
}

```

```

/* output sine */

```

```

rtB.Sine_Wave =
rtP.Sine_Wave_Amp *
(
    (rtDWork.Sine_Wave_RWORK.LastSin *
rtP.Sine_Wave_PCos
    + rtDWork.Sine_Wave_RWORK.LastCos *
(rtP.Sine_Wave_PSin)) *
    rtP.Sine_Wave_HCos
    + (rtDWork.Sine_Wave_RWORK.LastCos *
rtP.Sine_Wave_PCos
    - rtDWork.Sine_Wave_RWORK.LastSin *
(rtP.Sine_Wave_PSin)) *
    rtP.Sine_Wave_Hsin
    ) + rtP.Sine_Wave_Bias;

```

```

/* Sum: '<Root>/Sum2' incorporates:
 * Constant: '<Root>/forca set point kN'
 */

```

```

rtB.Sum2 = rtB.Sine_Wave -
rtP.forca_set_point_kN_Value - rtB.Sum6;

```

```

/* Gain: '<Root>/m'

```

```

 *
 * Regarding '<Root>/m':
 * Gain value: rtP.m_Gain
 */

```

```

rtB.m = rtB.Sum2 * rtP.m_Gain;

/* Outport: '<Root>/for total in' */
rtY.for_total_in = rtB.m;

/* Outport: '<Root>/forca shec' */
rtY.forca_shec = rtB.Sum6;

/* user code (Output function Trailer) */
/* dSPACE Data Capture block: (none) */
/* ... Service number: 1 */
/* ... Service name: (default) */
{
    host_service(1, 0)
}

/* 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);

    rtP.Discrete_State_Space_X0[0]=rtDWork.Discrete
State_Space_DSTATE[0];

    rtP.Discrete_State_Space_X0[1]=rtDWork.Discrete
State_Space_DSTATE[1];

    /* DiscreteStateSpace Block: <Root>/Discrete State-
    Space */
    {
        static real_T xnew[2];
        xnew[0] = (rtP.Discrete_State_Space_B[0])*rtB.m;
        xnew[0] +=

        (rtP.Discrete_State_Space_A[0])*rtDWork.Discrete_
        State_Space_DSTATE[0]
        +
        (rtP.Discrete_State_Space_A[1])*rtDWork.Discrete_
        State_Space_DSTATE[1];

        xnew[1] = (rtP.Discrete_State_Space_B[1])*rtB.m;
        xnew[1] +=

```

```

        (rtP.Discrete_State_Space_A[2])*rtDWork.Discrete_
        State_Space_DSTATE[1];

        (void)memcpy(&rtDWork.Discrete_State_Space_DS
        TATE[0], xnew,
        sizeof(real_T)*2);
    }

    /* Sin Block: <Root>/Sine Wave */
    {
        real_T hold_sin;
        real_T hold_cos;

        hold_sin =
        rtDWork.Sine_Wave_RWORK.LastSin;
        hold_cos =
        rtDWork.Sine_Wave_RWORK.LastCos;

        rtDWork.Sine_Wave_RWORK.LastSin = hold_sin
        * rtP.Sine_Wave_HCos + hold_cos *
        rtP.Sine_Wave_Hsin;
        rtDWork.Sine_Wave_RWORK.LastCos =
        hold_cos * rtP.Sine_Wave_HCos - hold_sin *
        rtP.Sine_Wave_Hsin;
    }
}

/* Terminate for root system: '<Root>' */
void MdlTerminate(void)
{
    if(rtM_hilestadosematraso70 != NULL) {
    }
}

/* Function to initialize sizes */
void MdlInitializeSizes(void)
{
    rtM_hilestadosematraso70->Sizes.numContStates =
    (0); /* Number of continuous states */
    rtM_hilestadosematraso70->Sizes.numY = (3); /*
    Number of model outputs */
    rtM_hilestadosematraso70->Sizes.numU = (0); /*
    Number of model inputs */
    rtM_hilestadosematraso70->Sizes.sysDirFeedThru =
    (0); /* The model is not direct feedthrough */
    rtM_hilestadosematraso70->Sizes.numSampTimes =
    (1); /* Number of sample times */
    rtM_hilestadosematraso70->Sizes.numBlocks =
    (23); /* Number of blocks */
    rtM_hilestadosematraso70->Sizes.numBlockIO =
    (13); /* Number of block outputs */
    rtM_hilestadosematraso70->Sizes.numBlockPrms =
    (22); /* Sum of parameter "widths" */
}

```

```

/* Function to initialize sample times */
void MdlInitializeSampleTimes(void)
{
    /* task periods */
    rtM_hilestadosematraso70->Timing.sampleTimes[0]
= (0.1);

    /* task offsets */
    rtM_hilestadosematraso70->Timing.offsetTimes[0]
= (0.0);
}

/* Function to register the model */
rtModel_hilestadosematraso70
*hilestadosematraso70(void)
{
    (void)memset((char *)rtM_hilestadosematraso70, 0,
sizeof(rtModel_hilestadosematraso70));

    {
        /* Setup solver object */
        static RTWSolverInfo rt_SolverInfo;
        rtM_hilestadosematraso70->solverInfo =
(&rt_SolverInfo);

        rtsiSetSimTimeStepPtr(rtM_hilestadosematraso70-
>solverInfo,
        &rtM_hilestadosematraso70-
>Timing.simTimeStep);
        rtsiSetTPtr(rtM_hilestadosematraso70->solverInfo,
        &rtmGetTPtr(rtM_hilestadosematraso70));
        rtsiSetStepSizePtr(rtM_hilestadosematraso70-
>solverInfo,
        &rtM_hilestadosematraso70->Timing.stepSize);
        rtsiSetdXPtr(rtM_hilestadosematraso70-
>solverInfo,
        &rtM_hilestadosematraso70->ModelData.derivs);
        rtsiSetContStatesPtr(rtM_hilestadosematraso70-
>solverInfo,
        &rtM_hilestadosematraso70-
>ModelData.contStates);

        rtsiSetNumContStatesPtr(rtM_hilestadosematraso70-
>solverInfo,
        &rtM_hilestadosematraso70-
>Sizes.numContStates);
        rtsiSetErrorStatusPtr(rtM_hilestadosematraso70-
>solverInfo,
        &rtmGetErrorStatus(rtM_hilestadosematraso70));

        rtsiSetRTModelPtr(rtM_hilestadosematraso70-
>solverInfo,
        rtM_hilestadosematraso70);
    }
}

```

```

/* 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, 1 *
sizeof(int_T));
        (void)memset((char_T *)&mdlSampleHits[0], 0, 1
* sizeof(int_T));

        rtM_hilestadosematraso70->Timing.sampleTimes
= (&mdlPeriod[0]);
        rtM_hilestadosematraso70->Timing.offsetTimes =
(&mdlOffset[0]);
        rtM_hilestadosematraso70-
>Timing.sampleTimeTaskIDPtr = (&mdlTsMap[0]);
        rtmSetTPtr(rtM_hilestadosematraso70,
&mdlTaskTimes[0]);
        rtM_hilestadosematraso70->Timing.sampleHits =
(&mdlSampleHits[0]);
    }
    rtsiSetSolverMode(rtM_hilestadosematraso70-
>solverInfo,
    SOLVER_MODE_SINGLETASKING);

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

    /* block I/O */
    {
        void *b = (void *) &rtB;
        rtM_hilestadosematraso70->ModelData.blockIO =
(b);

        {
            int_T i;

            b =&rtB.Discrete_State_Space;
            for (i = 0; i < 13; i++) {
                ((real_T*)b)[i] = 0.0;
            }
        }
    }
}

```

```

    }
    }
}

/* external outputs */
{
    rtM_hilestadosematraso70->ModelData.outputs =
(&rtY);

    rtY.volts = 0.0;
    rtY.for_total_in = 0.0;
    rtY.forca_shec = 0.0;
}

/* parameters */
rtM_hilestadosematraso70-
>ModelData.defaultParam = ((real_T *) &rtP);

/* data type work */
{
    void *dwork = (void *) &rtDWork;
    rtM_hilestadosematraso70->Work.dwork =
(dwork);
    (void)memset((char_T *) dwork, 0,
sizeof(D_Work));
    {
        int_T i;
        real_T *dwork_ptr = (real_T *)
&rtDWork.Discrete_State_Space_DSTATE[0];

        for (i = 0; i < 4; 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_hilestadosematraso70-
>SpecialInfo.mappingInfo = (&dtInfo);

    dtInfo.numDataTypes = 14;
    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_hilestadosematraso70->modelName =
("hilestadosematraso70");
rtM_hilestadosematraso70->path =
("hilestadosematraso70");

rtmSetTStart(rtM_hilestadosematraso70, 0.0);
rtM_hilestadosematraso70->Timing.tFinal = (-1);
rtM_hilestadosematraso70->Timing.stepSize =
(0.1);
rtsiSetFixedStepSize(rtM_hilestadosematraso70-
>solverInfo, 0.1);

rtM_hilestadosematraso70->Sizes.checksums[0] =
(2908208231U);
rtM_hilestadosematraso70->Sizes.checksums[1] =
(915290472U);
rtM_hilestadosematraso70->Sizes.checksums[2] =
(1767097667U);
rtM_hilestadosematraso70->Sizes.checksums[3] =
(3068413926U);

{
    static const EnableStates rtAlwaysEnabled =
SUBSYS_ENABLED;

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

    rtM_hilestadosematraso70->extModeInfo =
(&rt_ExtModeInfo);

    rteiSetSubSystemModeVectorAddresses(&rt_ExtMo
deInfo, sysModes);

    sysModes[0] = &rtAlwaysEnabled;

    rteiSetModelMappingInfoPtr(&rt_ExtModeInfo,
    &rtM_hilestadosematraso70-
>SpecialInfo.mappingInfo);

    rteiSetChecksumsPtr(&rt_ExtModeInfo,
    rtM_hilestadosematraso70->Sizes.checksums);

    rteiSetTPtr(&rt_ExtModeInfo,
    rtmGetTPtr(rtM_hilestadosematraso70));
}

return rtM_hilestadosematraso70;
}

```

ANEXO G – CODIGO GERADO MODELO 2 PROPOSTA 2

```

/*
 * Real-Time Workshop code generation for Simulink
 * model "hilestado.mdl".
 *
 * Model Version          : 1.724
 * Real-Time Workshop file version : 5.0 $Date:
 * 2002/05/30 19:21:33 $
 * Real-Time Workshop file generated on : Mon Jan
 * 23 21:26:06 2012
 * TLC version           : 5.0 (Jun 18 2002)
 * C source code generated on : Mon Jan 23
 * 21:26:07 2012
 */

#include <math.h>
#include <string.h>
#include "hilestado.h"
#include "hilestado_private.h"
#include "ext_work.h"

#include "hilestado_dt.h"

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

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

/* External output (root outputs fed by signals with
auto storage) */
ExternalOutputs rtY;

/* Parent Simstruct */
static rtModel_hilestado model_S;
rtModel_hilestado *const rtM_hilestado = &model_S;

/* Declaration needed for RTW malloc() usage */
const char
*RT_MEMORY_ALLOCATION_ERROR = "RTW
memory allocation error.";

/* Initial conditions for root system: '<Root>' */
void MdlInitialize(void)
{

    /* DiscreteStateSpace Block: <Root>/Discrete State-
Space */
    rtDWork.Discrete_State_Space_DSTATE[0] =
rtP.Discrete_State_Space_X0[0];
    rtDWork.Discrete_State_Space_DSTATE[1] =
rtP.Discrete_State_Space_X0[1];
}

/* Enable for root system: '<Root>' */
void MdlEnable(void)
{

    /* Sin Block: <Root>/Sine Wave */
    rtDWork.Sine_Wave_IWORK.SystemEnable =
(int_T) TRUE;
}

/* Start for root system: '<Root>' */
void MdlStart(void)
{

    /* dSPACE I/O Board DS1102 Unit:DAC */
    ds1102_da(2, 0);

    MdlInitialize();
    MdlEnable();
}

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

    if (rtmIsSampleHit(rtM_hilestado, 1, tid)) { /*
Sample time: [0.1, 0.0] */

        /* DiscreteStateSpace: '<Root>/Discrete State-
Space' */
        {
            rtB.Discrete_State_Space =

rtP.Discrete_State_Space_C*rtDWork.Discrete_State
_Space_DSTATE[0];
        }

        /* Gain: '<Root>/m to mm'
*
* Regarding '<Root>/m to mm':
* Gain value: rtP.m_to_mm_Gain
*/
        rtB.m_to_mm = rtB.Discrete_State_Space *
rtP.m_to_mm_Gain;

        /* Sum: '<Root>/Sum1' incorporates:
* Constant: '<Root>/deslo set point mm'
*/
        rtB.Sum1 = rtB.m_to_mm +
rtP.deslo_set_point_mm_Value;

        /* Gain: '<Root>/gain'
*
* Regarding '<Root>/gain':

```

```

    * Gain value: rtP.gain_Gain
    */
    rtB.gain = rtB.Sum1 * rtP.gain_Gain;

    /* Output: '<Root>/volts' */
    rtY.volts = rtB.gain;

    /* dSPACE I/O Board DS1102 Unit:DAC */
    ds1102_da(2, rtB.gain);
}

if (rtmIsSampleHit(rtM_hilestado, 0, tid)) { /*
Sample time: [0.01, 0.0] */

    /* dSPACE I/O Board DS1102 Unit:ADC */
    rtB.S_Function2_b = (real_T) ds1102_ad(2);

    /* Gain: '<Root>/Ganho forza'
    *
    * Regarding '<Root>/Ganho forza':
    * Gain value: rtP.Ganho_forca_Gain
    */
    rtB.Ganho_forca = rtB.S_Function2_b *
rtP.Ganho_forca_Gain;

    /* Sum: '<Root>/Sum6' incorporates:
    * Constant: '<Root>/Constante soma forza'
    */
    rtB.Sum6 = rtB.Ganho_forca -
rtP.Constante_soma_forca_Value;
}

if (rtmIsSampleHit(rtM_hilestado, 1, tid)) { /*
Sample time: [0.1, 0.0] */

    /* Sin: '<Root>/Sine Wave' */
    /* check enable state */
    if (
(int_T)rtDWork.Sine_Wave_IWORK.SystemEnable
) {
        rtDWork.Sine_Wave_RWORK.LastSin =
sin(rtP.Sine_Wave_Freq *
rtmGetTaskTime(rtM_hilestado, tid));
        rtDWork.Sine_Wave_RWORK.LastCos =
cos(rtP.Sine_Wave_Freq *
rtmGetTaskTime(rtM_hilestado, tid));
        rtDWork.Sine_Wave_IWORK.SystemEnable =
(int_T)FALSE;
    }

    /* output sine */

    rtB.Sine_Wave =
rtP.Sine_Wave_Amp *
(

```

```

        (rtDWork.Sine_Wave_RWORK.LastSin *
rtP.Sine_Wave_PCos
+ rtDWork.Sine_Wave_RWORK.LastCos *
(rtP.Sine_Wave_PSin)) *
rtP.Sine_Wave_HCos
+ (rtDWork.Sine_Wave_RWORK.LastCos *
rtP.Sine_Wave_PCos
- rtDWork.Sine_Wave_RWORK.LastSin *
(rtP.Sine_Wave_PSin)) *
rtP.Sine_Wave_Hsin
) + rtP.Sine_Wave_Bias;
    }

    if (rtmIsSampleHit(rtM_hilestado, 0, tid)) { /*
Sample time: [0.01, 0.0] */

        /* Sum: '<Root>/Sum2' incorporates:
        * Constant: '<Root>/forca set point kN'
        */
        rtB.Sum2 = rtB.Sine_Wave -
rtP.forca_set_point_kN_Value - rtB.Sum6;

        /* Gain: '<Root>/m'
        *
        * Regarding '<Root>/m':
        * Gain value: rtP.m_Gain
        */
        rtB.m = rtB.Sum2 * rtP.m_Gain;

        /* Output: '<Root>/for total in' */
        rtY.for_total_in = rtB.m;

        /* Output: '<Root>/forca shec' */
        rtY.forca_shec = rtB.Sum6;
    }

    /* user code (Output function Trailer) */
    /* dSPACE Data Capture block: (none) */
    /* ... Service number: 1 */
    /* ... Service name: (default) */
    if (rtmIsSampleHit(rtM_hilestado, 0, tid)) {
        host_service(1, 0)
    }
}

/* Update for root system: '<Root>' */
void MdlUpdate(int_T tid)
{
    if (rtmIsSampleHit(rtM_hilestado, 0, tid)) { /*
Sample time: [0.01, 0.0] */

        /* DiscreteStateSpace Block: <Root>/Discrete
State-Space */
        {

```



```

    static real_T xnew[2];
    xnew[0] =
    (rtP.Discrete_State_Space_B[0])*rtB.m;
    xnew[0] +=

    (rtP.Discrete_State_Space_A[0])*rtP.Discrete_State_
    Space_X0[0]
    +
    (rtP.Discrete_State_Space_A[1])*rtP.Discrete_State_
    Space_X0[1];

    xnew[1] =
    (rtP.Discrete_State_Space_B[1])*rtB.m;
    xnew[1] +=

    (rtP.Discrete_State_Space_A[2])*rtP.Discrete_State_
    Space_X0[1];

    (void)memcpy(&rtDWork.Discrete_State_Space_DS
    TATE[0], xnew,
    sizeof(real_T)*2);
    }

    }

    if (rtmIsSampleHit(rtM_hilestado, 1, tid)) { /*
    Sample time: [0.1, 0.0] */

    /* DiscreteStateSpace Block: <Root>/Discrete
    State-Space */
    {
        static real_T xnew[2];

        rtP.Discrete_State_Space_X0[0]=rtDWork.Discrete_
        State_Space_DSTATE[0];

        rtP.Discrete_State_Space_X0[1]=rtDWork.Discrete_
        State_Space_DSTATE[1];
        xnew[0] =
        (rtP.Discrete_State_Space_B[0])*rtB.m;
        xnew[0] +=

        (rtP.Discrete_State_Space_A[0])*rtDWork.Discrete_
        State_Space_DSTATE[0]
        +
        (rtP.Discrete_State_Space_A[1])*rtDWork.Discrete_
        State_Space_DSTATE[1];

        xnew[1] =
        (rtP.Discrete_State_Space_B[1])*rtB.m;
        xnew[1] +=

        (rtP.Discrete_State_Space_A[2])*rtDWork.Discrete_
        State_Space_DSTATE[1];

```

```

    (void)memcpy(&rtDWork.Discrete_State_Space_DS
    TATE[0], xnew,
    sizeof(real_T)*2);
    }

    /* Sin Block: <Root>/Sine Wave */
    {
        real_T hold_sin;
        real_T hold_cos;

        hold_sin =
        rtDWork.Sine_Wave_RWORK.LastSin;
        hold_cos =
        rtDWork.Sine_Wave_RWORK.LastCos;

        rtDWork.Sine_Wave_RWORK.LastSin =
        hold_sin * rtP.Sine_Wave_HCos + hold_cos
        * rtP.Sine_Wave_Hsin;
        rtDWork.Sine_Wave_RWORK.LastCos =
        hold_cos * rtP.Sine_Wave_HCos - hold_sin
        * rtP.Sine_Wave_Hsin;
    }
    }

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

    /* Function to initialize sizes */
    void MdlInitializeSizes(void)
    {
        rtM_hilestado->Sizes.numContStates = (0); /*
        Number of continuous states */
        rtM_hilestado->Sizes.numY = (3); /* Number of
        model outputs */
        rtM_hilestado->Sizes.numU = (0); /* Number of
        model inputs */
        rtM_hilestado->Sizes.sysDirFeedThru = (0); /* The
        model is not direct feedthrough */
        rtM_hilestado->Sizes.numSampTimes = (2); /*
        Number of sample times */
        rtM_hilestado->Sizes.numBlocks = (24); /* Number
        of blocks */
        rtM_hilestado->Sizes.numBlockIO = (13); /*
        Number of block outputs */
        rtM_hilestado->Sizes.numBlockPrms = (22); /* Sum
        of parameter "widths" */
    }

    /* Function to initialize sample times */

```

```

void MdlInitializeSampleTimes(void)
{
    /* task periods */
    rtM_hilestado->Timing.sampleTimes[0] = (0.01);
    rtM_hilestado->Timing.sampleTimes[1] = (0.1);

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

/* Function to register the model */
rtModel_hilestado *hilestado(void)
{
    (void)memset((char *)rtM_hilestado, 0,
    sizeof(rtModel_hilestado));

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

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

        rtsiSetRTModelPtr(rtM_hilestado->solverInfo,
        rtM_hilestado);
    }

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

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

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

        {
            int_T i;

            b = &rtB.Discrete_State_Space;
            for (i = 0; i < 13; i++) {
                ((real_T*)b)[i] = 0.0;
            }
        }
    }

    /* external outputs */
    {
        rtM_hilestado->ModelData.outputs = (&rtY);

        rtY.volts = 0.0;
        rtY.for_total_in = 0.0;
        rtY.forca_shec = 0.0;
    }

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

```

```

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

        for (i = 0; i < 4; 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_hilestado->SpecialInfo.mappingInfo =
(&dtInfo);

    dtInfo.numDataTypes = 14;
    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_hilestado->modelName = ("hilestado");
rtM_hilestado->path = ("hilestado");

```

```

rtmSetTStart(rtM_hilestado, 0.0);
rtM_hilestado->Timing.tFinal = (-1);
rtM_hilestado->Timing.stepSize = (0.01);
rtsiSetFixedStepSize(rtM_hilestado->solverInfo,
0.01);

rtM_hilestado->Sizes.checksums[0] =
(1386391271U);
rtM_hilestado->Sizes.checksums[1] =
(2858818304U);
rtM_hilestado->Sizes.checksums[2] =
(1537771595U);
rtM_hilestado->Sizes.checksums[3] =
(3302005126U);

{
    static const EnableStates rtAlwaysEnabled =
SUBSYS_ENABLED;

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

    rtM_hilestado->extModeInfo =
(&rt_ExtModeInfo);

    rteiSetSubSystemModeVectorAddresses(&rt_ExtMo
deInfo, sysModes);

    sysModes[0] = &rtAlwaysEnabled;

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

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

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

return rtM_hilestado;
}

```

ANEXO H – CODIGO CORRIGIDO MODELO 2 PROPOSTA 2

```

/*
 * Real-Time Workshop code generation for Simulink
 * model "hilestado.mdl".
 *c
 * Model Version          : 1.724
 * Real-Time Workshop file version : 5.0 $Date:
 2002/05/30 19:21:33 $
 * Real-Time Workshop file generated on : Mon Feb
 06 14:35:23 2012
 * TLC version           : 5.0 (Jun 18 2002)
 * C source code generated on : Mon Feb 06
 14:35:23 2012
 */

#include <math.h>
#include <string.h>
#include "hilestado.h"
#include "hilestado_private.h"
#include "ext_work.h"

#include "hilestado_dt.h"

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

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

/* External output (root outputs fed by signals with
auto storage) */
ExternalOutputs rtY;

/* Parent Simstruct */
static rtModel_hilestado model_S;
rtModel_hilestado *const rtM_hilestado = &model_S;

/* Declaration needed for RTW malloc() usage */
const char
*RT_MEMORY_ALLOCATION_ERROR = "RTW
memory allocation error.";

/* Initial conditions for root system: '<Root>' */
void MdlInitialize(void)
{
    /* DiscreteStateSpace Block: '<Root>/Discrete State-
Space */
    rtDWork.Discrete_State_Space_DSTATE[0] =
rtP.Discrete_State_Space_X0[0];
    rtDWork.Discrete_State_Space_DSTATE[1] =
rtP.Discrete_State_Space_X0[1];
}

}

/* Enable for root system: '<Root>' */
void MdlEnable(void)
{
    /* Sin Block: '<Root>/Sine Wave */
    rtDWork.Sine_Wave_IWORK.SystemEnable =
(int_T) TRUE;
}

/* Start for root system: '<Root>' */
void MdlStart(void)
{
    /* dSPACE I/O Board DS1102 Unit:DAC */
    ds1102_da(2, 0);

    MdlInitialize();
    MdlEnable();
}

/* Outputs for root system: '<Root>' */
void MdlOutputs(int_T tid)
{
    if (rtmIsSampleHit(rtM_hilestado, 1, tid)) { /*
Sample time: [0.1, 0.0] */

        /* DiscreteStateSpace: '<Root>/Discrete State-
Space' */
        {
            rtB.Discrete_State_Space =

rtP.Discrete_State_Space_C*rtDWork.Discrete_State
_Space_DSTATE[0];
        }

        /* Gain: '<Root>/m to mm'
        *
        * Regarding '<Root>/m to mm':
        * Gain value: rtP.m_to_mm_Gain
        */
        rtB.m_to_mm = rtB.Discrete_State_Space *
rtP.m_to_mm_Gain;

        /* Sum: '<Root>/Sum1' incorporates:
        * Constant: '<Root>/deslo set point mm'
        */
        rtB.Sum1 = rtB.m_to_mm +
rtP.deslo_set_point_mm_Value;
    }
}

```

```

/* Gain: '<Root>/gain'
*
* Regarding '<Root>/gain':
* Gain value: rtP.gain_Gain
*/
rtB.gain = rtB.Sum1 * rtP.gain_Gain;

/* Output: '<Root>/volts' */
rtY.volts = rtB.gain;

/* dSPACE I/O Board DS1102 Unit:DAC */
ds1102_da(2, rtB.gain);
}

if (rtmIsSampleHit(rtM_hilestado, 0, tid)) { /*
Sample time: [0.01, 0.0] */

/* dSPACE I/O Board DS1102 Unit:ADC */
rtB.S_Function2_b = (real_T) ds1102_ad(2);

/* Gain: '<Root>/Ganho forza'
*
* Regarding '<Root>/Ganho forza':
* Gain value: rtP.Ganho_forca_Gain
*/
rtB.Ganho_forca = rtB.S_Function2_b *
rtP.Ganho_forca_Gain;

/* Sum: '<Root>/Sum6' incorporates:
* Constant: '<Root>/Constante soma forza'
*/
rtB.Sum6 = rtB.Ganho_forca -
rtP.Constante_soma_forca_Value;
}

if (rtmIsSampleHit(rtM_hilestado, 1, tid)) { /*
Sample time: [0.1, 0.0] */

/* Sin: '<Root>/Sine Wave' */
/* check enable state */
if (
(int_T)rtDWork.Sine_Wave_IWORK.SystemEnable
) {
    rtDWork.Sine_Wave_RWORK.LastSin =
sin(rtP.Sine_Wave_Freq *
    rtmGetTaskTime(rtM_hilestado, tid));
    rtDWork.Sine_Wave_RWORK.LastCos =
cos(rtP.Sine_Wave_Freq *
    rtmGetTaskTime(rtM_hilestado, tid));
    rtDWork.Sine_Wave_IWORK.SystemEnable =
(int_T)FALSE;
}

/* output sine */

```

```

rtB.Sine_Wave =
    rtP.Sine_Wave_Amp *
    (
        (rtDWork.Sine_Wave_RWORK.LastSin *
rtP.Sine_Wave_PCos
        + rtDWork.Sine_Wave_RWORK.LastCos *
rtP.Sine_Wave_PSin)) *
        rtP.Sine_Wave_HCos
        + (rtDWork.Sine_Wave_RWORK.LastCos *
rtP.Sine_Wave_PCos
        - rtDWork.Sine_Wave_RWORK.LastSin *
rtP.Sine_Wave_PSin)) *
        rtP.Sine_Wave_Hsin
    ) + rtP.Sine_Wave_Bias;
}

if (rtmIsSampleHit(rtM_hilestado, 0, tid)) { /*
Sample time: [0.01, 0.0] */

/* Sum: '<Root>/Sum2' incorporates:
* Constant: '<Root>/forca set point kN'
*/
rtB.Sum2 = rtB.Sine_Wave -
rtP.forca_set_point_kN_Value - rtB.Sum6;

/* Gain: '<Root>/m'
*
* Regarding '<Root>/m':
* Gain value: rtP.m_Gain
*/
rtB.m = rtB.Sum2 * rtP.m_Gain;

/* Output: '<Root>/for total in' */
rtY.for_total_in = rtB.m;

/* Output: '<Root>/forca shec' */
rtY.forca_shec = rtB.Sum6;
}

/* user code (Output function Trailer) */
/* dSPACE Data Capture block: (none) */
/* ... Service number: 1 */
/* ... Service name: (default) */
if (rtmIsSampleHit(rtM_hilestado, 0, tid)) {
    host_service(1, 0)
}
}

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

if (rtmIsSampleHit(rtM_hilestado, 1, tid)) { /*
Sample time: [0.1, 0.0] */

```

/* guardei informações do Xk de 0.1 */

rtP.Discrete_State_Space_X0[0]=rtDWork.Discrete_State_Space_DSTATE[0];

rtP.Discrete_State_Space_X0[1]=rtDWork.Discrete_State_Space_DSTATE[1];

/* DiscreteStateSpace Block: <Root>/Discrete State-Space */

```
{
    static real_T xnew[2];
    xnew[0] =
(rtP.Discrete_State_Space_B[0])*rtB.m;
    xnew[0] +=
(rtP.Discrete_State_Space_A[0])*rtDWork.Discrete_State_Space_DSTATE[0]
+
(rtP.Discrete_State_Space_A[1])*rtDWork.Discrete_State_Space_DSTATE[1];
```

```
    xnew[1] =
(rtP.Discrete_State_Space_B[1])*rtB.m;
    xnew[1] +=
```

```
(rtP.Discrete_State_Space_A[2])*rtDWork.Discrete_State_Space_DSTATE[1];
```

```
(void)memcpy(&rtDWork.Discrete_State_Space_DS
TATE[0], xnew,
    sizeof(real_T)*2);
}
```

/* Sin Block: <Root>/Sine Wave */

```
{
    real_T hold_sin;
    real_T hold_cos;

    hold_sin =
rtDWork.Sine_Wave_RWORK.LastSin;
    hold_cos =
rtDWork.Sine_Wave_RWORK.LastCos;
```

```
    rtDWork.Sine_Wave_RWORK.LastSin =
hold_sin * rtP.Sine_Wave_HCos + hold_cos
    * rtP.Sine_Wave_Hsin;
```

```
    rtDWork.Sine_Wave_RWORK.LastCos =
hold_cos * rtP.Sine_Wave_HCos - hold_sin
    * rtP.Sine_Wave_Hsin;
}
}
```

/* calculo a cada 0.01 do proximo xk de 0.1 */

if (rtMIsSampleHit(rtM_hilestado, 0, tid)) { /*
Sample time: [0.01, 0.0] */

```
    static real_T xnew[2];
    xnew[0] =
(rtP.Discrete_State_Space_B[0])*rtB.m;
    xnew[0] +=
```

```
(rtP.Discrete_State_Space_A[0])*rtP.Discrete_State_Space_X0[0]
+
(rtP.Discrete_State_Space_A[1])*rtP.Discrete_State_Space_X0[1];
```

```
    xnew[1] =
(rtP.Discrete_State_Space_B[1])*rtB.m;
    xnew[1] +=
```

```
(rtP.Discrete_State_Space_A[2])*rtP.Discrete_State_Space_X0[1];
```

```
(void)memcpy(&rtDWork.Discrete_State_Space_DS
TATE[0], xnew,
    sizeof(real_T)*2);
```

_I

}

/* Terminate for root system: '<Root>' */

```
void MdlTerminate(void)
{
    if(rtM_hilestado != NULL) {
    }
}
```

/* Function to initialize sizes */

```
void MdlInitializeSizes(void)
{
    rtM_hilestado->Sizes.numContStates = (0); /*
Number of continuous states */
    rtM_hilestado->Sizes.numY = (3); /* Number of
model outputs */
```

```

    rtM_hilestado->Sizes.numU = (0);    /* Number of
model inputs */
    rtM_hilestado->Sizes.sysDirFeedThru = (0); /* The
model is not direct feedthrough */
    rtM_hilestado->Sizes.numSampTimes = (2); /*
Number of sample times */
    rtM_hilestado->Sizes.numBlocks = (24); /* Number
of blocks */
    rtM_hilestado->Sizes.numBlockIO = (13); /*
Number of block outputs */
    rtM_hilestado->Sizes.numBlockPrms = (22); /* Sum
of parameter "widths" */
}

/* Function to initialize sample times */
void MdlInitializeSampleTimes(void)
{
    /* task periods */
    rtM_hilestado->Timing.sampleTimes[0] = (0.01);
    rtM_hilestado->Timing.sampleTimes[1] = (0.1);

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

/* Function to register the model */
rtModel_hilestado *hilestado(void)
{
    (void)memset((char *)rtM_hilestado, 0,
sizeof(rtModel_hilestado));

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

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

        rtsiSetRTModelPtr(rtM_hilestado->solverInfo,
rtM_hilestado);

```

```

    }

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

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

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

            {
                int_T i;

                b =&rtB.Discrete_State_Space;
                for (i = 0; i < 13; i++) {
                    ((real_T*)b)[i] = 0.0;
                }
            }
        }
    }

```

```

}

/* external outputs */
{
    rtM_hilestado->ModelData.outputs = (&rtY);

    rtY.volts = 0.0;
    rtY.for_total_in = 0.0;
    rtY.forca_shec = 0.0;
}

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

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

        for (i = 0; i < 4; 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_hilestado->SpecialInfo.mappingInfo =
(&dtInfo);

    dtInfo.numDataTypes = 14;
    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_hilestado->modelName = ("hilestado");
rtM_hilestado->path = ("hilestado");

rtmSetTStart(rtM_hilestado, 0.0);
rtM_hilestado->Timing.tFinal = (-1);
rtM_hilestado->Timing.stepSize = (0.01);
rtsiSetFixedStepSize(rtM_hilestado->solverInfo,
0.01);

rtM_hilestado->Sizes.checksums[0] =
(1386391271U);
rtM_hilestado->Sizes.checksums[1] =
(2858818304U);
rtM_hilestado->Sizes.checksums[2] =
(1537771595U);
rtM_hilestado->Sizes.checksums[3] =
(3302005126U);

{
    static const EnableStates rtAlwaysEnabled =
SUBSYS_ENABLED;

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

    rtM_hilestado->extModeInfo =
(&rt_ExtModeInfo);

    rteiSetSubSystemModeVectorAddresses(&rt_ExtMo
deInfo, sysModes);

    sysModes[0] = &rtAlwaysEnabled;

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

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

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

return rtM_hilestado;
}

```