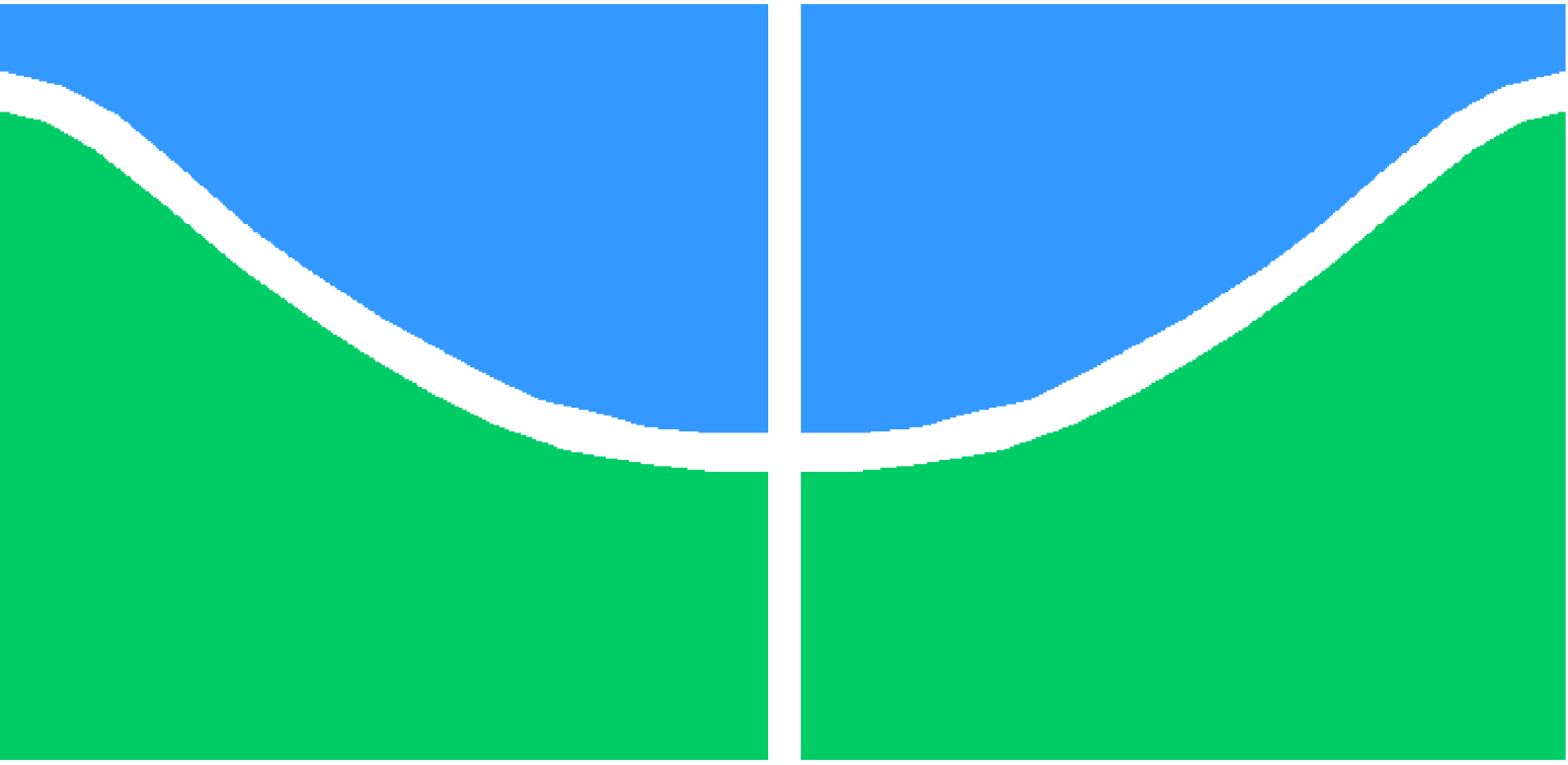


Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia Eletrônica

Desenvolvimento de uma plataforma de simulação Hardware in the Loop de baixo custo

Autor: Alceu Bernardes Castanheira de Farias
Orientador: Renato Vilela Lopes

Brasília, DF
2016

Alceu Bernardes Castanheira de Farias

Desenvolvimento de uma plataforma de simulação Hardware in the Loop de baixo custo

Monografia submetida ao curso de graduação em (Engenharia Eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletrônica).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Renato Vilela Lopes

Coorientador: André Murilo de Almeida Pinto

Brasília, DF

2016

Alceu Bernardes Castanheira de Farias
Desenvolvimento de uma plataforma de simulação Hardware in the Loop de baixo custo/ Alceu Bernardes Castanheira de Farias. – Brasília, DF, 2016-
112 p. : il. (algumas color.) ; 30 cm.

Orientador: Renato Vilela Lopes

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2016.

1. Simulação Hardware in the Loop. 2. Sistemas de controle. I. Renato Vilela Lopes. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Desenvolvimento de uma plataforma de simulação Hardware in the Loop de baixo custo

CDU 02:141:005.6

Alceu Bernardes Castanheira de Farias

Desenvolvimento de uma plataforma de simulação Hardware in the Loop de baixo custo

Monografia submetida ao curso de graduação em (Engenharia Eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletrônica).

Trabalho aprovado. Brasília, DF, 30 de junho de 2016:

Renato Vilela Lopes
Orientador

Evandro Leonardo S. Teixeira
Convidado 1

Suzana Moreira Avila
Convidado 2

Brasília, DF
2016

Este trabalho é dedicado à minha mãe, Máuria Maria Castanheira, que sempre fez o possível e o impossível para que eu pudesse alcançar meus sonhos. Tudo que eu conquistei e ainda hei de conquistar é devido ao seu apoio e amor incondicionais.

Agradecimentos

Primeiramente, a Deus, por possibilitar a oportunidade de chegar a esse momento.

Aos familiares que sempre me apoiaram e estiveram ao meu lado, em especial minha mãe, Máuria Maria Castanheira, por todo seu amor, carinho e dedicação para me proporcionar o melhor e permitir que eu pudesse alcançar meus objetivos.

Ao meu irmão, Wesley Ribeiro Castanheira, um exemplo não só como engenheiro, mas também como pessoa.

Aos meus orientadores, Renato Vilela Lopes e André Murilo de Almeida Pinto, por toda a orientação, paciência e ajuda prestada, não somente nesse trabalho, mas também em vários outros aspectos relacionados à minha trajetória acadêmica. Pelas referências que são para mim.

À minha namorada, Mônica Damasceno, pelo apoio e paciência durante todo o período desse trabalho. Pela ajuda e pelos incentivos nos momentos complicados.

Aos amigos e colegas da Universidade de Brasília, campus Gama - FGA. Pelo aprendizado, companheirismo e projetos na companhia de tantos engenheiros/ futuros engenheiros talentosos.

A todos os professores que ajudaram tanto neste trabalho quanto em toda minha carreira acadêmica, seja em diferentes disciplinas ou projetos, por proporcionarem um enorme aprendizado e por serem modelos de profissionais para mim.

A todos os que ajudaram no desenvolvimento deste trabalho, diretamente ou indiretamente.

"Todo mundo tem problemas.
Todo mundo lida com mil coisas.
Mas é a forma como você administra isso que importa.
(Gabriel Goffi)

Resumo

A crescente necessidade de testar e prototipar projetos sob situações cada vez mais realistas resultou no desenvolvimento de novos tipos de simulação. Nesse cenário, um tipo de simulação que tem ganhado grande notoriedade e aplicabilidade é a simulação *Hardware In the Loop* (HIL). Essa técnica permite que componentes reais e virtuais de um sistema sejam testados simultaneamente, de forma que o sistema possa ser submetido a condições mais realísticas/extremas sem comprometer o sistema real ou um protótipo construído especificamente para testes. O objetivo deste trabalho foi desenvolver uma plataforma HIL de baixo custo, para ser utilizada em diferentes tipos de aplicações, diferentemente da maioria dos modelos comerciais, que são voltados para uma aplicação específica, como aeronáutica, automotiva, eletrônica de potência, etc. Todas as etapas necessárias para o correto funcionamento da plataforma HIL serão descritas neste trabalho, bem como os exemplos de modelos e controladores utilizados para a validação da mesma.

Palavras-chaves: Simulação *Hardware in the Loop*. Sistemas de controle.

Abstract

The increasing need for testing and prototyping projects under realistic situations is responsible for the advancement of new types of simulation. In this scenario, one type of simulation which has gained high notoriety and applicability is the Hardware in the Loop (HIL) simulation. This technique allows real and virtual components of a system to be tested together, making it possible to test the system under more realistic (and even extreme) conditions without harming the real system or a prototype built only for the purpose of testing. The main objective of this work was to develop a low-cost HIL simulation platform, to be used for many different applications, unlike most commercial ones, which are developed for one exclusive field of application, such as aerospace, automotive, power electronics and many others. All the steps needed for the correct operation of the HIL platform will be described during this work, as well as the models and controller examples used for validating it.

Key-words: Hardware in the Loop simulation. Control Systems.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

HIL	*Hardware in the Loop*
ECU	*Electronic Central Unit*
ADC	*Analog-to-Digital Converter*
DAC	*Digital-to-Analog Converter*
PWM	*Pulse Width Modulation*
GPIO	*General Purpouse Input/Output*
RAM	*Random Access Memory*
DSP	*Digital Signal Processor*
I^2C	*Inter-Integrated Circuit*
SPI	*Serial Peripheral Interface*
RS232	*Recommended Standard 232*
PCI	*Peripheral Component Interconnect*
SFTP	*Secure File Transfer Protcol*
FTP	*File Transfer Protocol*
SCP	*Secure Copy Protocol*
SSH	*Secure Shell*
DTO	*Device Trees Overlays*
PID	Proporcional integrador derivativo
PI	Proporcional integrador
LQR	*Linear Quadratic Regulator*

Lista de símbolos

m_1	Massa suspensa
m_2	Massa não-suspensa
k_1	Rigidez do pneu
k_2	Rigidez da mola
b	Coeficiente de amortecimento
K	Matriz de ganhos do LQR
u_{ss}	Valor de regime estacionário do sistema
x_{ss}	Valor de referência do sistema

Sumário

1 Introdução

O surgimento de computadores cada vez mais modernos, o avanço das tecnologias de informação e o uso de novas técnicas de programação possibilitaram o surgimento de ferramentas computacionais cada vez mais sofisticadas e poderosas. Com isso, novas metodologias têm surgido para o desenvolvimentos de sistemas.

Em geral, o desenvolvimento de um sistema se dá em duas frentes: teste de *software* e teste de *hardware* (ALTERA, 2013). Primeiramente, um sistema é testado e otimizado utilizando-se *softwares* de simulação. Sem esse tipo de ferramenta, projetos complexos necessitariamente deveriam ser implementados na prática para serem validados e terem sua funcionalidade plena confirmada, acarretando em aumento de custo e tempo de desenvolvimento do projeto. A simulação permite que o comportamento do sistema seja testado antes da sua implementação, ajudando a entender melhor como o mesmo funciona (HOSSEINPOUR; HAJIHOSSEINI, 2009). Uma vez obtidos os resultados desejados em simulação, o produto pode ser finalmente implementado e testado na prática.

Apesar do ciclo de projeto citado acima ser bem tradional, o mesmo apresenta dificuldades. A principal delas refere-se às ferramentas de simulação, que, apesar de terem se tornado práticas recorrentes e amplamente utilizadas nas mais diversas aplicações, não garantem, em geral, o sucesso total de um projeto. De fato, em alguns casos, simulações não são capazes de replicar as condições de operação na forma como estão presentes no mundo real (LU et al., 2007).

Simulações, por exemplo, podem não ser suficientes para validar diversos tipos de controladores. Normalmente os sistemas reais são muito caros, e, se o controlador não for devidamente validado em *hardware*, não há garantias de que o sistema se comportará da maneira esperada (NI, 2012). Consequentemente, os primeiros testes no sistema real podem apresentar falhas, aumentando os riscos desta etapa, uma vez que qualquer erro pode causar danos, tanto ao sistema quanto às pessoas que estejam presentes na realização dos testes.

Um exemplo clássico da situação descrita acima é o desenvolvimento de um veículo aéreo do tipo quadrirotor. Esses sistemas são inerentemente instáveis, e assim, mesmo que o controlador responsável pela estabilização da aeronave apresente um bom desempenho em ambiente de simulação, a chance da ocorrência de falhas nos primeiros testes experimentais existe, podendo acarretar problemas como os descritos no parágrafo anterior.

Existem algumas maneiras de tentar mitigar essa deficiência do ambiente de simulação. Uma primeira abordagem, seria a construção de uma bancada/protótipo mais simples para efetuar os testes necessários para validar o sistema. Entretanto, trata-se de

uma estratégia de desenvolvimento lento, em que os custos podem ser bastante elevados.

Uma segunda abordagem é conhecida como *Hardware In the Loop* (HIL). Esse tipo de simulação tem ganhado destaque em diversas áreas da engenharia, como a automotiva, eletrônica de potência e sistemas de controle (LU et al., 2007), como uma opção de testar um sistema sob condições mais realistas sem ser necessário produzir ou confeccionar o sistema final na prática. Uma simulação HIL pode ser utilizada durante o desenvolvimento de um sistema como na figura abaixo (Fig. 1), onde a mesma é implementada em um ciclo de projeto conhecido como *V-Cycle*, voltado para sistemas digitais.

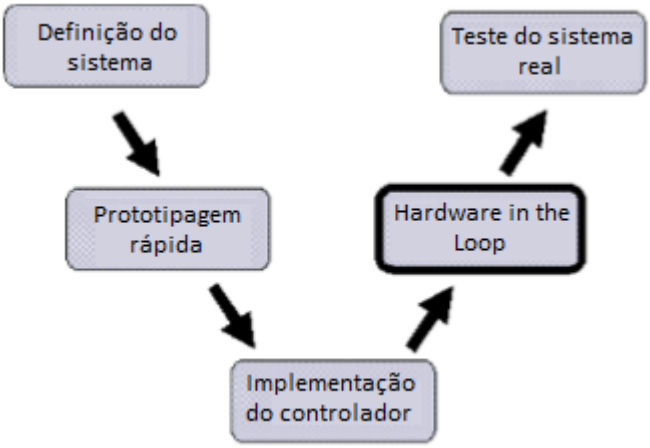


Figura 1 – Típico ciclo de projeto *V-Cycle* de um sistema digital com HIL. Adaptado de: (NI, 2012).

De maneira geral, uma simulação HIL é caracterizada pela conexão entre componentes reais e componentes simulados de um sistema. Geralmente, o *hardware* e o *software* da aplicação final são implementados com componentes reais, ao passo que sensores, atuadores e processos físicos são simulados, seja parcialmente ou totalmente (ISERMANN; SCHAFFNIT; SINSEL, 1999). A figura 2 mostra um exemplo de HIL, onde uma unidade de controle eletrônico (*Electronic Control Unit* em inglês, ou ECU) real é testada juntamente com o modelo do motor de um automóvel, que é simulado.

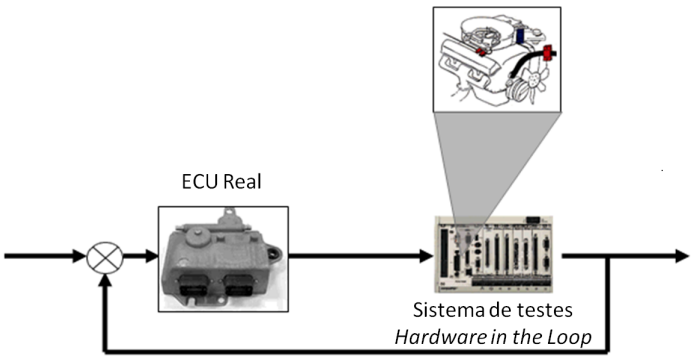


Figura 2 – Exemplo de um *Hardware in the Loop*. Adaptado de: (NI, 2012).

As principais vantagens de uma simulação HIL incluem (ISERMANN; SCHAFFNIT; SINSEL, 1999):

- Teste de *hardware* e *software* sem a necessidade de operar um processo real.
- Testar o efeito que falhas em sensores, atuadores ou outros componentes podem causar no sistema, sem a necessidade de construir e colocar o sistema real sob essas situações de risco.
- Teste de operação do sistema sob condições de risco, que poderiam causar acidentes ou até mesmo morte para pessoas envolvidas, sem a necessidade de testar o sistema final construído.

Além disso, plataformas HIL são uma alternativa bastante utilizada para testar sistemas de controle. Como dito anteriormente, se o controlador não for devidamente validado em *hardware*, não há garantias de que o sistema se comportará da maneira esperada. Com esse tipo de simulação, é possível embarcar o controlador desejado em *hardware*, e observar se o mesmo garante o funcionamento do sistema de forma adequada, sem ser necessário construir o sistema final para testar e otimizar o sistema de controle da aplicação.

Apesar de todas essas vantagens e de sua crescente popularização, nem todos os projetos utilizam simulações HIL em seu ciclo de desenvolvimento. Uma das principais barreiras para a utilização desse tipo de estratégia é o custo. Existem plataformas comerciais para a realização de simulações HIL, dentre as quais se destacam as plataformas da empresa dSPACE, que é responsável por construir plataformas para diferentes aplicações como engenharia biomédica, indústrias aeroespacial e automotiva, até mesmo para propósitos acadêmicos (DSPACE, 2015). Entretanto, a grande maioria dessas plataformas possui custo elevado.

Um exemplo é a plataforma SCALEXIO, usada para simular ECUs, com aplicações voltadas à eletrônica veicular. O curso de engenharia Automotiva da Faculdade UnB Gama está em processo de compra desta plataforma, por um valor de aproximadamente 108 mil euros.

Outra dificuldade encontrada é o fato de que a grande maioria dos modelos comerciais disponíveis são voltados para uma aplicação específica. Uma plataforma HIL voltada para a indústria química não pode ser usada para testar um sistema automotivo, por exemplo.

Assim, o objetivo desse trabalho é o desenvolvimento de uma plataforma de simulações e testes HIL, de baixo custo, que possa ser utilizada para a validação de diferentes tipos de projetos/sistemas, em especial no âmbito acadêmico. Assim, a ideia proposta é

a de projetar um sistema mais simples e menos realista do ponto de vista de protocolos de comunicação, interface e componentes, mas que por outro lado possibilite a simulação de diversos tipos de sistemas. Isso tornaria possível, por exemplo, testar o modelo de um veículo aéreo não tripulado sob a ação de um determinado controlador e, depois, testar o modelo de uma suspensão automotiva com um tipo de controle diferente. Para isto, bastar alterar o modelo matemático do sistema de interesse em *software* e o controlador embarcado no *hardware* real.

Essa abordagem constitui uma inovação com relação aos principais trabalhos encontrados na literatura e às plataformas HILs comerciais que geralmente possuem a característica de utilização de interface e componentes voltados para uma aplicação específica. Como efeito, essas plataformas permitem simulações mais realistas, mas impossibilitam a avaliação de outros tipos de sistemas.

Desta forma, apesar de mais simples, a plataforma proposta deve ser capaz de validar o modelo testado, exibindo o mesmo número de entradas e saída do sistema real e do controlador durante a realização dos testes, bem como o comportamento dessas entradas e saídas mediante as condições de operação utilizadas para a simulação HIL.

Neste trabalho, serão apresentados todos os passos utilizados para o projeto e construção de uma plataforma HIL, como instalação, calibração e modos de utilização de cada componente e como os mesmos interagem entre si para que a simulação HIL ocorra. Além disso, serão abordados os modelos e controladores utilizados para testar a plataforma, ajudando não somente a validar a mesma, mas também possibilitando um melhor entendimento do trabalho realizado.

1.1 Objetivo geral do projeto

Desenvolver uma plataforma HIL de baixo custo, voltada para o meio acadêmico, que seja capaz de simular modelos de diferentes áreas de aplicação.

1.2 Objetivos específicos do projeto

- Estudar diferentes tipos de plataforma de simulação HIL voltadas para diversas aplicações.

- Desenvolver a arquitetura de uma plataforma HIL de baixo custo e capaz de simular modelos de diferentes áreas de aplicação.

- Apresentar como a plataforma HIL desenvolvida funciona, através da explicação de cada componente que compõe a mesma.

- Demonstrar os passos de calibração e instalação realizados para o correto funcionamento da plataforma HIL.
- Utilizar a plataforma HIL para validar um sistema seguidor de referência com o uso de um controlador PI.
- Utilizar a plataforma HIL para validar o modelo de uma suspensão automotiva com o uso de um controlador LQR.

1.3 Contribuição do trabalho

A principal contribuição do trabalho é desenvolver uma plataforma HIL que não seja voltada para uma única área de aplicação, possibilitando a simulação de diferentes tipos de sistemas mecânicos. Mesmo que os modelos sejam voltadas para o meio acadêmico, e, em geral, mais simples, trata-se de uma grande contribuição, pois a grande maioria das plataformas HIL existentes foram projetadas para o estudo de aplicações bem específicas, sejam elas modelos comerciais ou acadêmicos.

1.4 Estrutura do trabalho

Esse trabalho está dividido em cinco capítulos, organizados da seguinte maneira:

- **Capítulo 1:** Introdução do tema deste trabalho, assim como a apresentação dos objetivos e a principal contribuição do mesmo.
- **Capítulo 2:** Apresentação da descrição formal de um sistema *Hardware in the Loop*, bem como suas configurações gerais e principais características e requisitos, de maneira mais aprofundada do que apresentado neste capítulo.
- **Capítulo 3:** Demonstração da metodologia do trabalho, desde o projeto da plataforma até sua efetiva construção. Descrição dos componentes que formam a plataforma HIL, e como os mesmos interagem entre si para o funcionamento da mesma.
- **Capítulo 4:** Resultados experimentais obtidos através da simulação HIL de modelos para validação da plataforma desenvolvida.
- **Capítulo 5:** Conclusões finais do trabalho e descrição dos possíveis trabalhos futuros que podem ser realizados para a continuação do projeto.

2 Fundamentos teóricos

2.1 Definição e desenvolvimento histórico

Conforme mencionado anteriormente, o desenvolvimento de produtos e processos modernos exige mais do que etapas de simulação computacional, já que a mesma não é capaz de reproduzir com fidelidade algumas condições de operação reais do sistema em questão.

A técnica de simulação HIL se propõe nesse sentido, a trazer uma integração entre componentes implementados em *hardware* e funções/componentes/processos simulados em *software* (ISERMANN; SCHAFFNIT; SINSEL, 1999). Devido à essa versatilidade, é possível poupar tempo e custos no projeto, simulando componentes/modelos que seriam muito caros ou demandariam muito tempo para serem construídos, ao mesmo tempo que uma resposta em *hardware* real é obtida, permitindo uma validação mais efetiva do sistema do que simplesmente uma simulação computacional do mesmo.

É importante ressaltar que, por mais que as simulações muitas vezes não consigam retratar todas as condições reais de operação de um sistema de forma adequada, ainda são uma etapa importante para um projeto. De fato, simulações ainda são uma etapa necessária de um projeto, mas, em certas aplicações, não são suficientes para validar o mesmo.

As primeiras simulações HIL foram realizadas na área aeronaútica, em especial para testes de vôo em tempo real, de modo a testar instrumentos de aviação com um *cockpit* fixo (ISERMANN; SCHAFFNIT; SINSEL, 1999), que posteriormente foi substituído por um *cockpit* que podia se mover de acordo com o movimento do avião. Essa era uma tentativa de se obter um novo método para o treinamento de pilotos, por exemplo. Nesse caso, o *cockpit* era real, mas os movimentos do avião eram gerados por atuadores elétricos e hidráulicos, que foram sendo gradualmente substituídos por componentes análogos mais sofisticados até que fossem finalmente implementados por processos computacionais (ANDERSON, 1962). A simulação HIL propriamente dita foi utilizada na etapa de teste dos componentes dos componentes do avião, como o corpo e diferentes tipos de suspensão sendo substituidos por atuadores elétricos ou hidráulicos, funcionando como "máquinas de teste"(DROSDOL; KADING; PANIK, 1985). Além disso, diferentes tipos de estímulos eram simulados, como por exemplo, excitações sofridas pela roda de um avião devido ao contato com a superfície da pista de pouso.

Outro tipo interessante de simulação é o chamado teste dinâmico de motores, no qual os motores das aeronaves são componentes reais, mas o veículo propriamente

dito e engrenagens são simuladas por outro *hardware* (motores AC ou DC, por exemplo) em junção com um computador simulando diversos processos digitais (ISERMANN; SCHAFFNIT; SINSEL, 1999).

Com o desenvolvimento de sistemas eletrônicos digitais de controle, as simulações HIL foram sendo aprimoradas cada vez mais (HUBER; JONNER; DEMEL, 1988). Com o avanço de processos computacionais de alta performance e, eventualmente, a utilização de computação paralela em associação com processadores *Reduced Instruction Set Computing* (RISC) equipados com chips internos de memória RAM (memória de acesso randômico, do inglês *Random Access Memory*) e links de comunicação de alta velocidade, além de processadores de sinais digitais (do inglês, *Digital Signal Processors* ou DSP), permitiram que simulações em tempo real de sistemas hidráulicos completos, sensores, sistemas de suspensão, entre outros componentes pudessem ser efetuados de forma mais eficiente, viabilizando ainda mais as simulações do tipo HIL (HANSELMANN, 1993).

Apesar do grande avanço que esse tipo de simulação permitiu à indústria aeroespacial, essa não é a única área em que simulações HIL são amplamente utilizadas. A indústria automotiva, por exemplo, é uma área que está frequentemente recorrendo a esse tipo de estratégia, uma vez que esse tipo de simulação é reconhecida como uma maneira efetiva de testar estratégias de controle automotivo, assim como diagnosticar funcionalidades (PALLADINO; FIENGO; LANZO, 2012). A simulação HIL permite que os testes sejam realizados de maneira mais rápida e eficiente, poupando tempo e custos.

Além dessas, existem diversos campos e áreas que se utilizam dessa técnica de simulação devido à sua eficiência, como a área de eletrônica de potência (LU et al., 2007), e até mesmo na integração e teste dos computadores a bordo da missão Cassini (cujo destino é Saturno): AACS (do inglês, *Attitude and Articulation Control Subsystem*) e CDS (do inglês, *Command and Data Subsystem*). Além disso, a técnica também é utilizada na validação do software de vôo para esses computadores, o FSW (do inglês, *Flight Software*), e dos programas que governam toda a atividade da espaçonave (BADARUDDIN; HERNANDEZ; BROWN, 2007).

O que faz com que essa técnica seja amplamente utilizada em tantas áreas são suas inúmeras vantagens, que já foram discutidas anteriormente, como a possibilidade de testar componentes de *hardware* e *software* sem a necessidade de operar um processo real, em condições e restrições extremas que muitas vezes não podem ser testadas em ambientes de simulação, além de testar o efeito que uma falha em sensores, atuadores ou outros componentes podem causar no sistema, entre várias outras (ISERMANN; SCHAFFNIT; SINSEL, 1999). Porém, existem algumas desvantagens. Além do elevado custo das plataformas comerciais existentes, esse tipo de simulação em geral necessita de pessoas qualificadas que saibam como configurar e definir bem os modelos a serem simulados, o que muitas vezes pode envolver o uso de modelos matemáticos suficiente-

mente precisos e acuratos para determinada aplicação (BADARUDDIN; HERNANDEZ; BROWN, 2007).

Mesmo assim, as vantagens do método, em muitos casos, superam as suas desvantagens, tornando a simulação HIL cada vez mais popular e utilizado na indústria.

2.2 Configuração de um Hardware in the Loop

O *Hardware in the Loop* é uma técnica bastante flexível. A decisão de quais componentes serão implementados em *hardware* e quais serão simulados cabe aos indivíduos responsáveis pelos testes. Mesmo assim, é possível estabelecer uma configuração genérica para o sistema que será simulado, como no mostrado na figura 3.

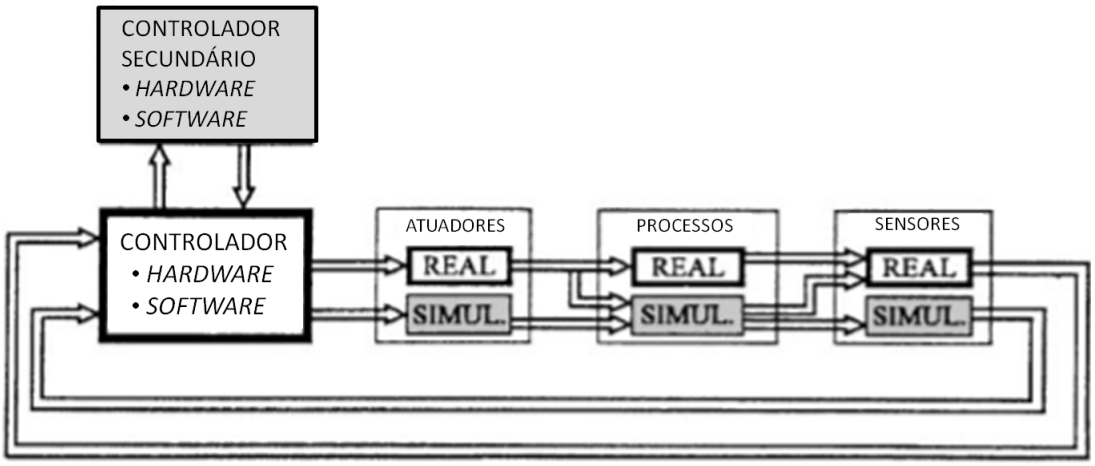


Figura 3 – Configuração genérica de um *Hardware in the Loop*. Adaptado de: (ISERMANN; SCHAFFNIT; SINSEL, 1999)

Como é possível ver na figura acima, o sistema é composto por partes bem distintas: atuadores (componentes que produzem movimento em resposta a um comando elétrico, mecânico, manual, etc.), processos (conjunto de procedimentos realizados pelo sistema enquanto o mesmo encontra-se em funcionamento) e sensores (componentes que realizam medições de diferentes tipos de grandezas físicas, necessárias para o correto funcionamento do sistema) e o sistema de controle, que pode conter uma ou mais unidades de controle (controlador e controlador secundário), responsáveis por implementar a estratégia adequada para controlar o sistema.

O interessante da simulação HIL é que, pela figura 3, é possível ver que qualquer um desses componentes pode ser implementado em *hardware* ou *software* (ou até mesmo partes de um componente implementadas em *hardware* e outras partes do mesmo componente implementadas em *software*), sejam eles pertencentes ao modelo a ser simulado ou ao sistema de controle. Porém, como dito anteriormente, o *hardware* e o *software* da aplicação

final costumam ser implementados com componentes reais, enquanto sensores, atuadores e processos físicos costumam ser simulados(ISERMANN; SCHAFFNIT; SINSEL, 1999).

De maneira geral, têm-se que o *hardware* de uma plataforma de simulação HIL utilizada na prática é composto por (SILVA, 2008):

- **Computador *host*:** inclui *softwares* com simuladores, editores de texto, compiladores, depuradores de código e o que mais for necessário para a devida simulação do sistema e desenvolvimento dos algoritmos de controle a serem testados;
- **Computador *target*:** componente onde será executado o *software* gerado pelo computador *host*. Pode variar desde um complexo sistema com múltiplos processadores dedicados até um microprocessador;
- **Interface:** atua sobre os sinais do sistema, especialmente entre os computadores *host* e *target*, fazendo com que o sinal seja adequado a cada módulo da plataforma, garantindo a operação correta do sistema. Geralmente é composto por componentes discretos (resistores, capacitores, amplificadores operacionais), filtros, atenuadores, conversores analógico-digitais/digital-analógicos, etc.

Essa estrutura está representada na figura abaixo (Fig. 4).

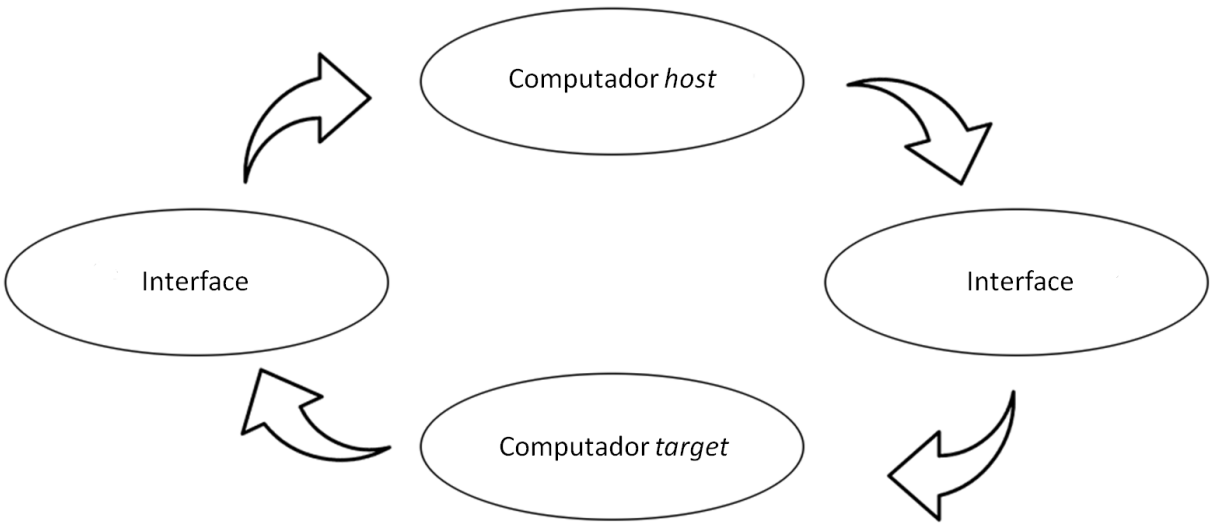


Figura 4 – Diagrama de blocos da plataforma de simulação HIL proposta.

O computador *host* corresponde ao componente onde se encontra o modelo matemático do sistema físico a ser testado. Para a plataforma proposta, esse computador conterá o *software* MATLAB, juntamente com o Simulink, pelo qual muitos modelos de sistemas são desenvolvidos. Esse computador também pode possuir suporte para a programação do computador *target*.

O computador *host* deve se comunicar com o computador *target* e, para isso, é necessário realizar a interface entre ambos. Dependendo da escolha do *host* é necessário respeitar diferentes tipos de parâmetro, como tensões e corrente de operação.

Por exemplo, pode-se querer simular o modelo de um sistema físico com saídas que variam entre -5 e 5 V, mas o computador *target* só trabalha com entradas em uma faixa de 0 a 5 V, podendo ser danificado com valores negativos de tensão. Outro exemplo seria o caso em que o modelo do sistema possui saídas analógicas, mas o *target* só possui entradas digitais. Em ambos os casos, os sinais provenientes do *host* não podem ser simplesmente conectados ao *target*, sendo necessário utilizar interfaces que permitam a conexão entre ambos os computadores, sem nenhum tipo de dano aos componentes e garantindo as condições de operação corretas de ambos.

Uma vez que o sinal foi adequado às condições de operação, o mesmo pode ser conectado ao *target*, onde encontra-se o sistema de controle embarcado em *hardware*. Uma estratégia que têm sido comumente utilizada é o fato de o *target* ser um um microprocessador, o que traz algumas vantagens interessantes. Atualmente, existem muitos modelos com conversores analógico-digitais (em inglês, *Analog-to-Digital Converter* ou ADC), conversores digital-analógicos (em inglês, *Digital-to-Analog Converter* ou DAC), saídas analógicas do tipo *Pulse Width Modulation* (PWM) e diversas funcionalidades embutidas, que podem ser usadas até mesmo para o condicionamento do sinal proveniente do computador *host*.

Uma vez que dados são processados pelo controlador e a simulação realizada, os resultados são interfaciados entre *target* e *host*, para poderem retornar ao modelo, tornando possível a visualização dos resultados da simulação do sistema.

2.3 Exemplos de plataformas HIL

Alguns exemplos de plataformas HIL podem ser obtidos observando modelos comerciais existentes no mercado, como as da dSPACE (Fig. 5).

Figura 5 – Exemplo de típico *Hardware in the Loop* comercial da dSPACE. Fonte: (PALLADINO; FIENGO; LANZO, 2012)

Essa plataforma comercial da dSPACE é desenvolvida principalmente para a indústria automotiva, providenciando todos os sinais elétricos necessários para a implementação de uma ECU. Os modelos a serem testados podem ser desenvolvidos usando o *software* MATLAB e seu ambiente de desenvolvimento Simulink (BADARUDDIN; HERNANDEZ; BROWN, 2007).

O simulador é equipado com um processador DS1006 e possui unidades de I/O (entradas e saídas para a realização de testes). Uma placa DS2211 HIL I/O é usada para as entradas e saídas da simulação, enquanto os modelos de componentes são simulados computacionalmente. Essa plataforma é padrão para aplicações automotivas, permitindo a simulação de um típico motor de oito cilindros, por exemplo (BADARUDDIN; HERNANDEZ; BROWN, 2007).

Outro exemplo de *Hardware in the Loop* comercial da própria dSPACE é o SCALEXIO, já citado anteriormente (figura 6).

Figura 6 – Exemplo de uma plataforma *Hardware in the Loop* SCALEXIO da dSPACE. Fonte: (WALTER, 2014)

Essa plataforma é ainda mais completa do que o modelo descrito anteriormente, apresentando várias características como (DSPACE, 2015):

- Geração de diversos tipos de sinais comuns a uma ECU;
- Medição de sinais;
- Condicionador integrado de sinais;
- Unidade de roteamento de falhas (do inglês, *Failure Routing Unit* ou FRU);
- Processador *quad-core* Intel® CoreTM i7-860. Um dos quatros núcleos de processamento é reservado para otimização de performance do sistema sendo testado, enquanto os outros três estão disponíveis para a computação dos modelos simulados;

- Suporte para diferentes protocolos de comunicação:
  - CAN;
  - Lin/K-Line;
  - FlexRay;
  - UART (RS232, RS422, RS485 interface serial).

Os exemplos citados até agora são de plataformas comerciais, bastante robustas e potentes. Tendo em vista o objetivo do trabalho, torna-se necessário pesquisar exemplos de plataformas HIL que foram desenvolvidas em ambiente acadêmico, com uma arquitetura mais próxima da que será desenvolvida neste trabalho.

Um primeiro exemplo pode ser visto na figura 7.

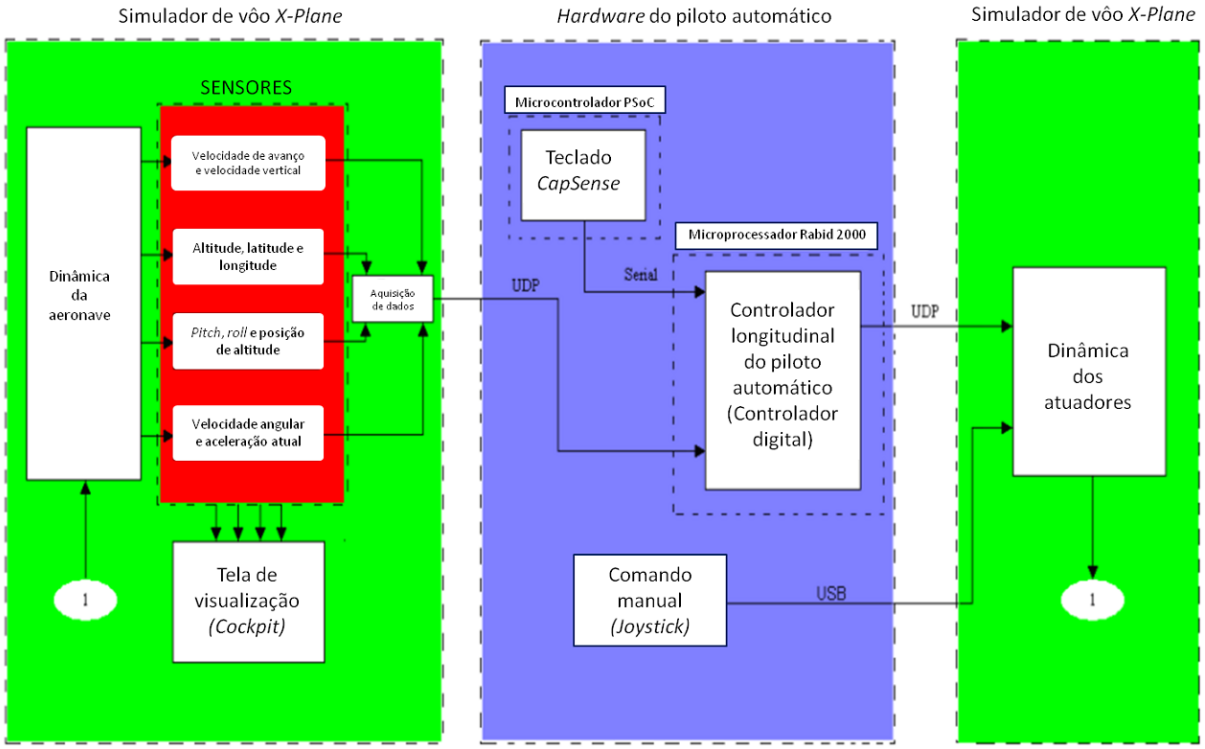


Figura 7 – Plataforma *Hardware in the Loop* desenvolvida para simulação da dinâmica de vôos de Boieng 747-400. Adaptado de: (SANTOS; OLIVEIRA, 2011)

Esse modelo de plataforma *Hardware in the Loop* foi desenvolvido para testar características de vôo de um avião Boieng 747-400, além de modelar sensores e atuadores em tempo real para se comunicarem com a cabine de *hardware* de piloto automático longitudinal. Os sensores estimulam os dados de saída referentes à posição e altitude no simulador (*X-Plane*). Os atuadores são responsáveis por simular como o *hardware* de piloto automático pode mudar a superfície de controle da fuselagem, o que é feito no mundo real controlando servo-motores na superfície de controle correspondente. O

processo simulado indica como o avião reage às entradas enviadas pelo *hardware* de piloto aumotático longitudinal. O sistema embarcado executa o simulador de vôo, juntamente com o controlador da aeronave (SANTOS; OLIVEIRA, 2011).

Um outro exemplo interessante de plataforma HIL desenvolvida em pesquisas acadêmicas é o trabalho de H.M.Silva (SILVA, 2008), em sua tese de Mestrado, que utiliza simulações HIL em um veículo submarino semi-autônomo, conforme mostrado na figura 8.

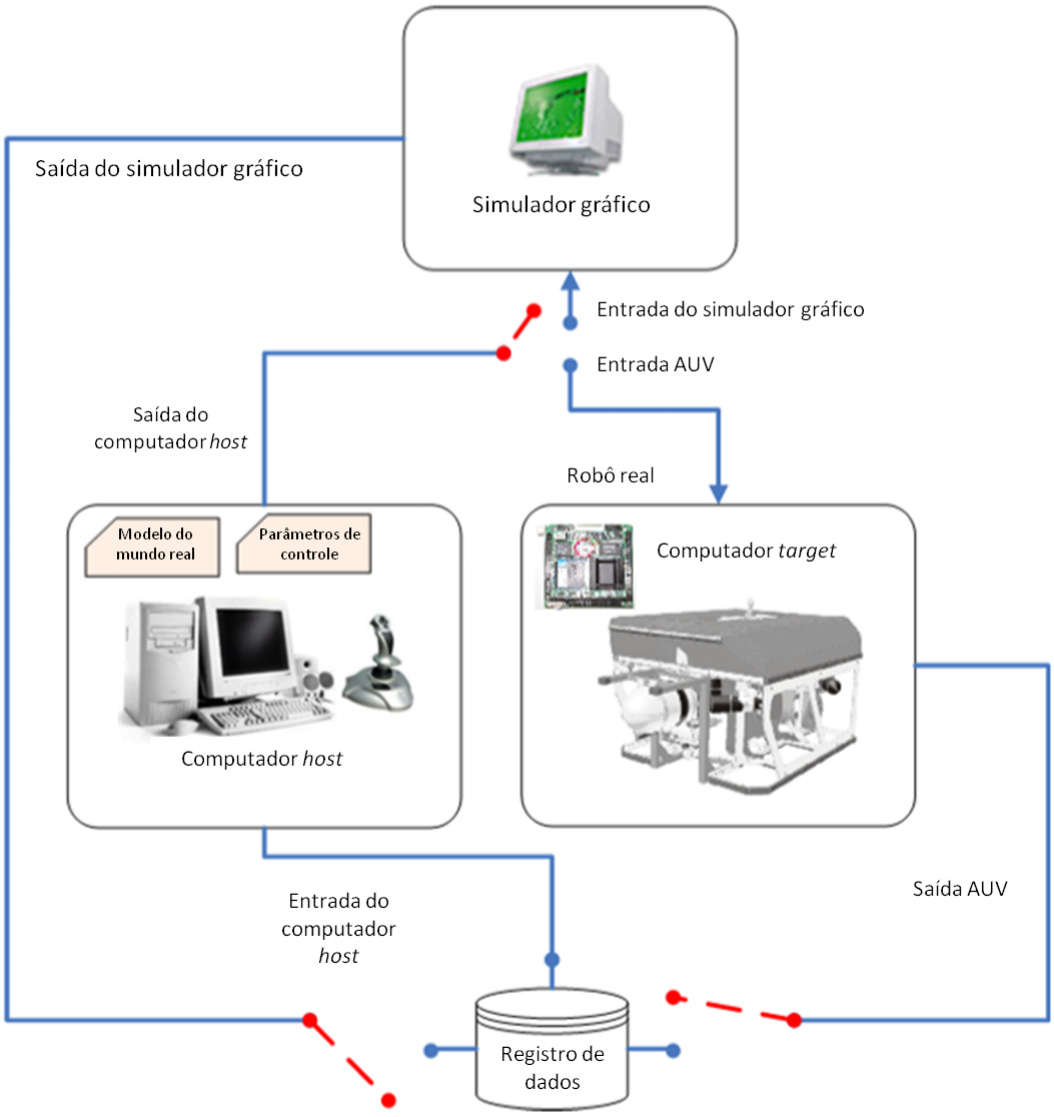


Figura 8 – Plataforma HIL desenvolvida para simulação de veículos submarinos semi-autônomos. Adaptado de: (SILVA, 2008)

Nesse tipo de plataforma o sistema a ser testado (Robô Real), é composto por *hardware* e *software*. As entradas do sistema correspondem a forças vetoriais geradas pelo meio (empuxo e corrente, por exemplo) e o vetor de torque aplicado tanto pelos planos de controle quanto pelos propulsores do veículo.

O computador *host* é responsável tanto por simular as condições subaquáticas referentes ao meio físico em que o veículo se encontra (topografia da profundeza dos oceanos e a presença de agentes marinhos - peixes ou até mesmo outros robôs), quanto por gerar parâmetros de controle (velocidade, aceleração, gravidade, empuxo, sentido e direção de correntezas, etc.), utilizados para adequar os algoritmos de controle empregados. Essas etapas são indicadas na figura 8 como modelo do mundo real e parâmetros de controle.

Por fim, os dados são enviados para um registro de dados, para que possam ser analisados posteriormente.

Um último exemplo é uma plataforma HIL desenvolvida para VANTs (Veículos Autônomos Não-Tripulados) de asas rotativas (Fig. 9).

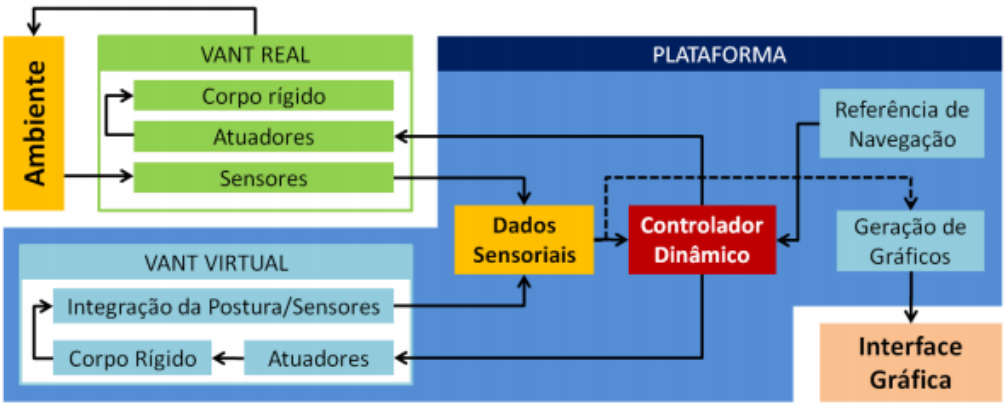


Figura 9 – Diagrama de blocos da plataforma *Hardware in the Loop* desenvolvida para simulação de VANTs de asas rotativas. Fonte: (PIZETTA et al., 2012)

Esse projeto de plataforma HIL é voltado para simulações com quadrirotores, e é capaz de realizar simulações com modelos reais, modelos simulados ou modelos contendo tanto partes reais quanto simuladas. Ao se estabelecer um protocolo de comunicação bidirecional entre a aeronave escolhida e a plataforma, juntamente com uma *flag* de permissão ativada, a plataforma inicia o processo de coleta dos dados sensoriais do veículo.

Para o caso de um quadrirotor real, os sinais de controle são transmitidos ao sistema embarcado do veículo, que os aplica diretamente aos atuadores. As novas condições de vôo são então transmitidas à plataforma HIL e o ciclo reinicia. No caso de um quadrirotor simulado, os sinais de controle são enviados ao modelo dinâmico da aeronave. Posteriormente, através de métodos de integração numérica, determina-se a posição futura desejada para a aeronave, atualizando os sinais de comando e os dados sensoriais do sistema (PIZETTA et al., 2012).

Diferentemente dos projetos anteriores, a plataforma a ser desenvolvida neste trabalho não será destinada a uma aplicação específica. O objetivo é que outros modelos e sistemas possam ser conectados à plataforma e simulados, sendo essa uma das principais contribuições desse trabalho.

2.4 Restrições de tempo e simulação em tempo real

Em uma simulação HIL é muito importante levar em conta a taxa de amostragem do sistema a ser testado. Esse parãmetro leva em consideração o instante de tempo que dados são lidos ou enviados de uma parte do sistema a outro. Assim, de certa forma, a taxa de amostragem é o que diferencia os sinais contínuos existentes em simulação computacional dos sinais discretos obtidos durante a simulação HIL e a implementação do sistema na prática.

A taxa de amostragem do modelo simulado indica a taxa com a qual a simulação lê resultados na entrada do sistema, bem como a taxa na qual novas saídas estão disponíveis. Para o controlador embarcado em *hardware*, a taxa de amostragem é responsável por mostrar o tempo necessário para o controlador processar as entradas e disponibilizar variáveis de controle em suas saídas (LOCKHART, 2016).

Muitas vezes as simulações com controladores implementados em *hardware* são utilizadas para casos em que se lida com processos relativamente rápidos, como controladores para aviões, automóveis, motores de combustão e outros tipos de componentes. Em geral, o período de amostragem costuma ser um valor entre 1 e 10 ms, o que configura uma frequência de amostragem entre 100 Hz e 1kHz para o controlador, ao passo que os processos mecânicos tendem a ser simulados com uma taxa de amostragem de 0.5 a 10 ms (ISERMANN; SCHAFFNIT; SINSEL, 1999).

Os fatores que limitam a taxa de amostragem de uma simulação HIL podem ser vários, mas os principais são:

- a utilização/possibilidade de utilizar processamento em paralelo;
- o método de integração utilizado;
- em função de quais parâmetros as equações diferenciais que constituem o modelo são implementadas.

A técnica de simulação HIL, entretanto, é bem flexível: é possível utilizá-la para obter-se as mesmas vantagens já discutidas em processos mais lentos, que não requerem uma taxa de amostragem tão grande. De fato, em casos desse tipo, é necessário simular o sistema com uma taxa de amostragem maior do que em tempo real, uma vez que torna-se interessante simular o comportamento do sistema ao longo de semanas, meses ou até anos em algumas horas ou dias. Isso pode ser evidenciado, por exemplo, no controle de processos químicos que demoram muito tempo para serem efetuados, ou até mesmo em sistema de controle de potência e outras áreas básicas da indústria (ISERMANN; SCHAFFNIT; SINSEL, 1999).

Nesses casos, os fatores que limitam a velocidade de simulação não são dados pelo processo em si a ser simulado, mas pela capacidade da plataforma HIL de acelerar o processo de simulação, que geralmente é limitado pelo uso/ausência de (ISERMANN; SCHAFFNIT; SINSEL, 1999):

- *Timers* e unidades de captura/comparação;
- Conversores analógico-digitais (ADC) e conversores digital-analógicos (DAC);
- Atuadores reais;
- Constantes de tempo de componentes analógicos, como resistores e capacitores.

É importante notar que a plataforma HIL a ser desenvolvida deve ser construída de forma que outros projetos possam ser simulados na mesma. Isso inclui processos que exigem taxas de amostragem mais lenta, portanto os fatores acima devem ser levados em consideração no desenvolvimento da mesma.

Além disso, é importante ressaltar que, apesar de estar intimamente ligada a simulações em tempo real, a simulação HIL tecnicamente não é obrigatoriamente em tempo real. Em muitos casos, é desejável e até mesmo vantajoso testar-se a plataforma em tempo real para obter resultados mais precisos e mais próximos do que se obtêm na prática. Entretanto, como visto anteriormente, existem processos mais lentos que devem ser simulados acima da frequência de amostragem em tempo real.

Outro fator importante é que simulações em tempo real em geral são mais complexas de serem realizadas, especialmente para processos a serem simulados com taxa de amostragem elevadas. Entretanto, nesses casos, a simulação em tempo real dá uma taxa de confiabilidade maior aos resultados, tornando-se algo desejável.

3 Projeto e Construção da Plataforma HIL

Este capítulo tem por finalidade apresentar todas as etapas de projeto e construção da plataforma HIL, desde a sua concepção, passando pela escolha de componentes e descrevendo as soluções propostas para os problemas enfrentados.

A metodologia deste trabalho se deu através de pesquisa e escolha dos componentes que compõe a arquitetura de plataforma de simulação HIL, de forma que ela que possa ser utilizada para diversas áreas de aplicação e que possua baixo custo, quando comparada com modelos de plataforma HIL comerciais.

Uma vez definido o projeto da arquitetura, a plataforma foi construida e alguns sistemas e tipos de controladores foram utilizados para testar a performance da plataforma, responsável por validar os modelos testados. Os procedimentos e os resultados referentes a esses testes serão apresentados no próximo capítulo.

3.1 Concepção da plataforma HIL

Com base nas definições de plataforma HIL aparesentadas na seção 2.2, a arquitetura da plataforma HIL desenvolvida neste projeto corresponde ao diagrama abaixo (Fig. 10).

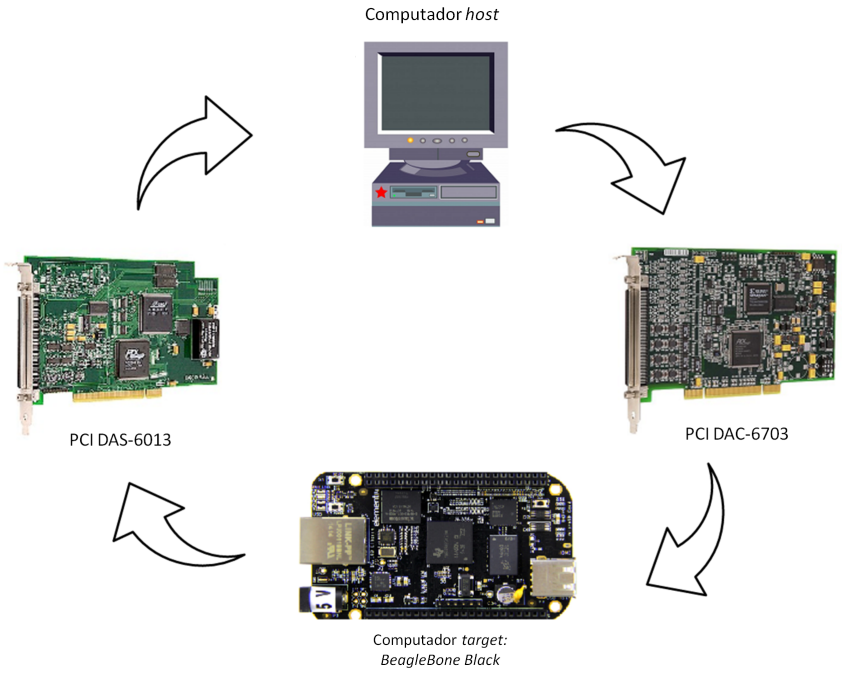


Figura 10 – Modelo de arquitetura HIL proposta.

A arquitetura da figura anterior possui certas semelhanças com a apresentada na figura 4, na seção 2.2. O computador *host* corresponde ao computador da figura 10 e as interfaces correspondem às placas que compõe o sistema de aquisição de dados (do inglês, *Data Acquistion System* ou DAQ): PCI DAC-6703 e PCI DAS-6013. Por fim, o microprocessador *BeagleBone Black* corresponde ao computador *target*.

Uma figura da plataforma construida pode ser encontrada abaixo (Fig. 11).

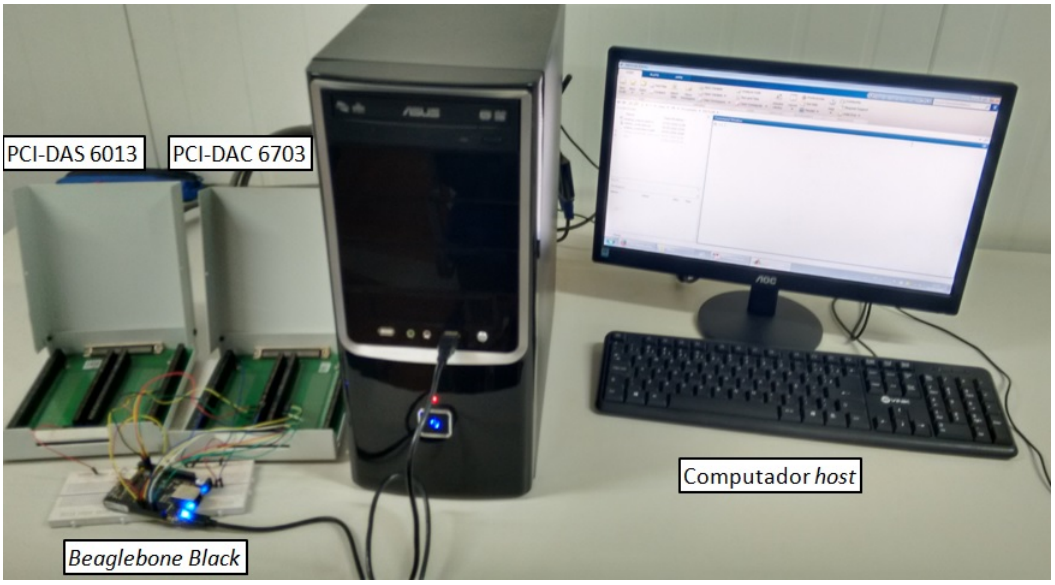


Figura 11 – Plataforma de simulação HIL construida.

O computador *host* possui as ferramentas necessárias para o desenvolvimento de modelos a serem testados, bem como as ferramentas necessárias para desenvolver os algoritmos de controle para o microprocessador. Isso inclui o *software* MATLAB, com o Simulink e as *toolboxes* necessárias para o desenvolvimento dos modelos a serem testados, como a *Data Acquisition Toolbox*, que será explorada mais adiante. O computador também possui *drivers* necessários para estabelecer conexão com a *BeagleBone Black*. Alguns *softwares* auxiliam na comunicação entre o computador e a *Beaglebone Black*, como WinSCP e PuTTY.

O WinSCP (cuja interface encontra-se representada na figura 12) corresponde a um cliente FTP (*File Transfer Protocol*), SFTP (*Secure File TRansfer Protocol*), e SCP (*Secure Copy*) *open-source* para Windows, possibilitando a troca de arquivos entre dois *hosts* remotos (*host* e *BeagleBone Black*). Assim, é possível desenvolver os algoritmos de controle no *host* e transferi-los de maneira simples para a *BeagleBone Black*.

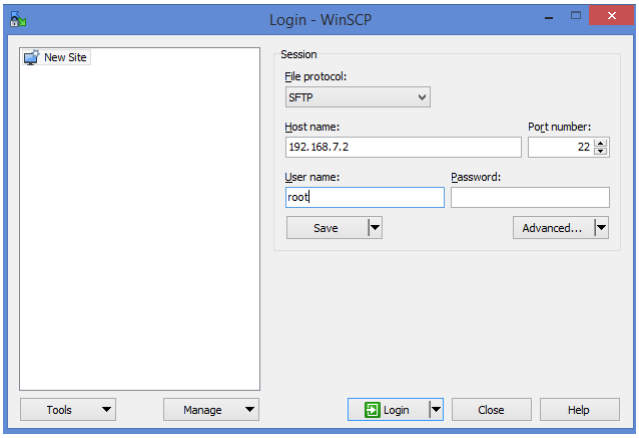


Figura 12 – Interface do *software* WinSCP.

Caso deseje-se utilizar a própria *BeagleBone Black* para o desenvolvimento dos algoritmos de controle, pode-se optar por conectá-la a um monitor extra com entrada HDMI ou utilizar o PuTTY (Fig. 13), um cliente *open-source* SSH(*Secure Shell*) e Telnet para Windows. Com os *drivers* da *BeagleBone Black* instalados no *host*, é possível estabelecer um protocolo de comunicação SSH (já que a *BeagleBone Black* possui um sistema operacional baseado em Linux, o Debian), executando e modificando os arquivos dentro da mesma usando o próprio computador *host*.

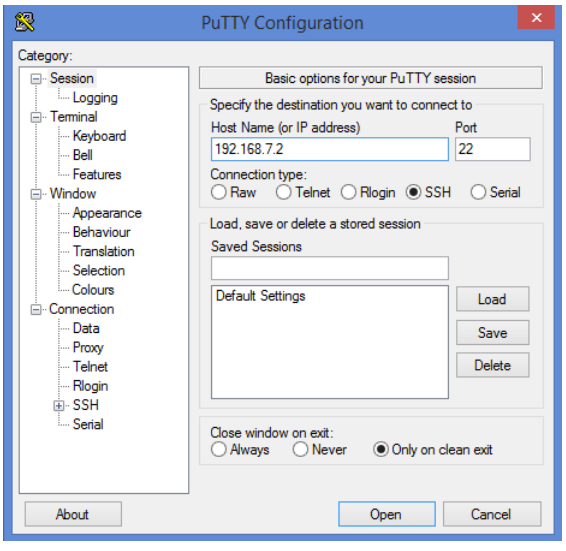
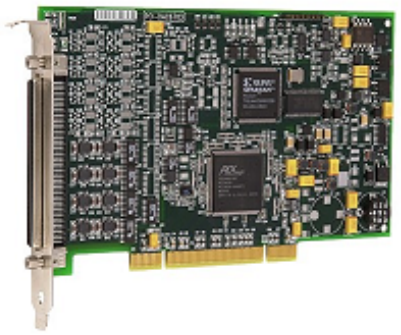
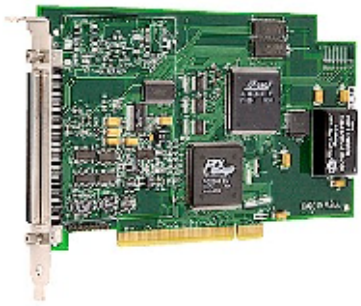


Figura 13 – Interface do *software* PuTTY.

Outros componentes fundamentais para a plataforma HIL são os sistemas de aquisição de dados. Eles são responsáveis por comunicar os dados provenientes do modelo matemático do sistema físico no computador *host* com os dados do controlador embarcado no computador *target*. Para isso, foram escolhidas duas placas de aquisição de dados: PCI-DAC 6703 e PCI-DAS 6013 (Fig. 14).



(a) Sistema de aquisição de dados PCI DAC 6073.



(b) Sistema de aquisição de dados PCI DAS-6013.

Figura 14 – Sistemas de aquisição de dados utilizados na plataforma HIL.

Esses modelos foram escolhidos tendo em mente a questão do baixo custo desejado para a plataforma HIL proposta. Primeiramente, esses modelos da *Measurement Computing* conectam-se ao computador através de barramentos PCI. Isso torna muito fácil o processo de instalação das mesmas, já que são sistemas de aquisição *Plug and Play*, ou seja, ao conectar ambos os componentes no barramento PCI do computador *host*, o próprio computador os reconhece e instala automaticamente os *drivers* necessários para sua utilização.

Além disso, o barramento PCI, possui bom custo em relação a sua velocidade de operação. Sistemas de aquisição de dados USB costumam ser mais baratos, mas possuem taxas de amostragem baixas. Barramentos *PCI Express* possuem taxas de amostragem maiores de funcionamento, mas, em geral, possuem custo mais elevado.

A PCI-DAC 6703 é um sistema de aquisição de dados que conta com:

- 16 saídas analógicas de 16-bits;
- 8 entradas/saídas digitais;
- Tensão máxima de operação das saídas analógicas: -10V a 10V.

Por possuir saídas analógicas, a PCI-DAC 6703 funciona como um DAC, convertendo as informações digitais do modelo em simulação para valores analógicos, possibilitando que as saídas do modelo no MATLAB sejam conectadas às entradas analógicas da *Beaglebone Black*.

Por sua vez, a PCI-DAS 6013 possui:

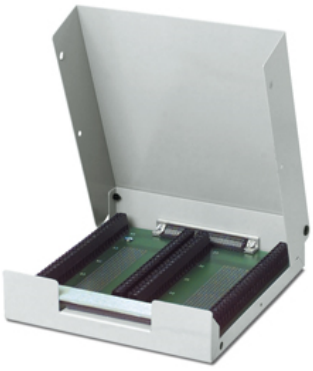
- 16 entradas analógicas de 16 bits;
- 8 entradas/saídas digitais;

- 2 contadores de 16-bits;
- Tensão máxima de operação das entradas analógicas: -10V a 10V.

Através de suas entradas analógicas, a PCI-DAS 6013 funciona como um ADC, convertendo as informações analógicas que cheguem ao modelo em informações digitais, possibilitando que as saídas do algoritmo de controle na *Beaglebone Black* sejam ligadas às entradas do modelo sendo simulado no Simulink.

Ambos os sistemas de aquisição de dados possuem interfaces de comunicação com o MATLAB através da *Data Acquisition Toolbox* do Simulink, característica importante já que os modelos a serem testados na plataforma são desenvolvidos nesse meio.

Além disso, os dois sistemas de aquisição de dados se conectam a uma caixa conectora de sinais SBC-100 (Fig. 15a) através de um cabo C100MS-1M (Fig. 15b). Assim é possível fazer a interface de todos os pinos e canais da placa de aquisição de dados com componentes exteriores, como microprocessadores.



(a) Caixa conectora de sinais SBC-100. Fonte: (COMPUTING, 2009)

(b) Cabo C100MS-1M. Fonte: (COMPUTING, 2009).

Figura 15 – Caixa conectora de sinais SBC-100 e cabo C100MS-1M.

O custo total dessas placas foi de R$ 12334,00, sendo a PCI DAC-6703 mais cara por se tratar de um sistema que provê saídas analógicas. Essa funcionalidade é mais difícil de ser encontrada e, portanto, torna esse componente mais caro, custando R$ 7644,75. A PCI DAS-6013 é mais barata, custando R$ 4689,25.

Por fim, houve a escolha do *target*, realizada após uma análise da performance de vários microprocessadores de baixo custo disponíveis no mercado: *Arduino Due*, *Intel Galileo* e *Raspberry Pi 2B* e *BeagleBone Black*.

Cada um dos modelos citados foi avaliado de acordo com os seguintes parâmetros:

- **Velocidade de processamento:** A frequência de *clock* de cada componente é importante para garantir que a simulação ocorrerá com a velocidade necessária

para testar-se situações realistas;

- **Quantidade de memória disponível:** A quantidade de memória para armazenar dados e códigos desenvolvidos para o microprocessador;
- **Preço:** O preço de cada componente;
- **Número de entradas/saídas disponíveis:** Quanto mais complexo o modelo, maior o número de entradas e saídas necessárias para avaliar o comportamento do sistema de forma correta;
- **Comunicação com o Simulink:** O Simulink é uma das principais ferramentas para modelagem de sistemas e, portanto, comunicação com o mesmo é importante;
- **Facilidade de uso/programação:** Um sistema que seja mais fácil de lidar facilita o desenvolvimento/teste de diferentes modelos;
- **Quantidade de referências biliográficas existentes para cada um:** Material de apoio existente com diferentes projetos desenvolvidos com cada microprocessador analisado;
- **Número de funcionalidades:** PWMs, DACs, ADCs, diferentes protocolos de comunicação disponíveis, etc.

A tabela 1 mostra a performance de cada um dos microprocessadores analisados, mediante os critérios determinados acima.

Tabela 1 – Performance de vários microprocessadores de baixo custo disponíveis no mercado.

	Raspberry Pi 2B	**Intel Galileo**	**Arduino Due**	**BeagleBone Black**
Processamento	900 MHz	400 MHz	84 MHz	1 GHz
Memória	1 GB RAM	256 MB RAM	512 KB RAM	512 MB RAM
Média de preço	R$ 300,00	R$ 500,00	R$ 200,00	R$ 400,00
GPIOs	40	55	56	65
Pinos analógicos	Não	Sim	Sim	Sim
Referências existentes	Muitas	Poucas	Muitas	Poucas
Facilidade de uso	Fácil	Fácil	Fácil	Difícil
Comunicação com o Simulink	Sim	Não	Sim	Sim

A tabela 1 mostra que a *BeagleBone Black* (Fig. 16), modelo escolhido para ser o *target*, possui funcionalidades bem interessantes que podem ser exploradas para a plataforma a um custo acessível. Além de ser uma plataforma *open-source*, compatível com sistemas operacionais baseados em Linux, a *BeagleBone Black* possui os seguintes recursos:

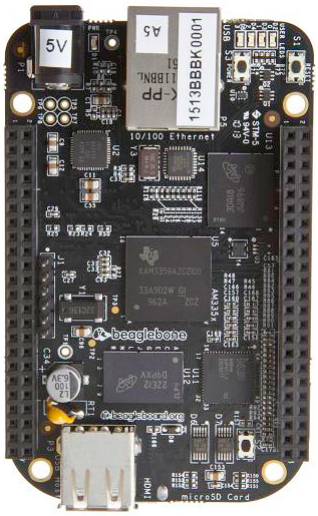


Figura 16 – *BeagleBone Black* e suas funcionalidades. Fonte: (BEAGLEBOARD.ORG, 2016)

- Processador AM335x 1 GHz ARM Cortex-A8
- 512MB de memória RAM DDR3
- 2 Unidades Programáveis de Tempo-Real (em inglês, *Programmable Real-Time Units ou PRUs*) de 32-bits a 200 MHz.
- 8 Entradas analógicas de 0 a 1.8V
- 8 Saídas PWM com tensão de 0 a 3.3V
- 65 GPIOS (do inglês, *General Purpouse Inputs and Outputs*)
- Pinos com protocolo de comunicação I^2C e SPI

A existência de entradas analógicas, como mostrado na tabela 1, é uma vantagem em relação a microprocessadores bem populares, como a Raspberry Pi, que só possui entradas e saídas digitais. O Arduino, por exemplo, microcontrolador muito popular no meio, possui entradas analógicas, mas não possui a mesma capacidade de processamento da *Beaglebone Black*.

O que geralmente leva a *Beaglebone Black* a ser uma escolha menos comum para projetos do que a Raspberry Pi e o Arduino, e consequentemente diminui o número de

referências bibliográficas disponíveis da mesma (como visto na tabela 1), é a dificuldade em utilizar a mesma. Isso se deve, em parte, à maneira como as suas funcionalidades são organizadas. Como a *Beaglebone Black* não possui um BIOS (do inglês,*Basic Input/Output System*), o sistema utiliza arquivos para descrever o *hardware* do microprocessador (MOLLOY, 2014). O processo é bem mais complicado do que simplesmente usar bibliotecas prontas para ler ou escrever em pinos digitas e analógicos, como no Arduino e na Raspberry Pi, já supracitados.

O principal problema enfrentado com esse tipo de operação é a falta de controle sobre as taxas de amostragem dos pinos na *BeagleBone Black*, que acabam sendo bem mais lentas, já que operações com arquivos acabam demandando maior esforço computacional.

Entretanto, graças à popularidade de microprocessadores baseados em arquiteturas ARM (*Advanced RISC Machine*), como a *BeagleBone Black*, foram desenvolvidos *kernels* customizados para configurar as funcionalidades de microprocessadores de maneira mais eficiente. Dentre elas, a que se destaca e recebe maior apoio por parte da comunidade atualmente é o uso de *Device Tree Overlays* (DTO).

A DTO é basicamente uma estrutura de dados no formato de árvore, onde cada nó representa um pino ou funcionalidade de pino na *Beaglebone Black*. Esse tipo de estrutura possibilita descrever um nível de hierarquia, mostrando quais funcionalidades são dependentes de cada pino e quais pinos possuem mais de uma funcionalidade.

A principal vantagem da utilização desse mecanismo, é que com uma DTO, é possível modificar a configuração dos pinos da *Beaglebone Black* enquanto os algoritmos estão sendo executados (MOLLOY, 2014).

Com base nisso, foi desenvolvida uma biblioteca chamada *libpruio*. Essa biblioteca possibilita que os pinos digitais e analógicos da *BeagleBone Black* sejam utilizados de maneira mais simples e com taxas de amostragem maiores, de até 200 kHz (FREEBASIC, 2016). As DTOs dessa biblioteca criam subsistemas configuráveis, que possibilitam a utilização de funcionalidades como ler/escrever em pinos digitais e analógicos através de instruções simples, como no Arduino e na Raspberry Pi. Além disso, ela carrega DTOs que possibilitam configurar as PRUs da *Beaglebone Black* de maneira mais simples, auxiliando a desenvolver algoritmos para a mesma em tempo real. Para maiores detalhes do funcionamento desta biblioteca o leitor deve se referir ao site do projeto (FREEBASIC, 2016).

Assim, utilizando-se a *libpruio* e com as funcionalidades interessantes que a *Beaglebone Black* possui, aliadas a uma boa capacidade de processamento, esse microprocessador se mostrou uma escolha viável para a plataforma.

3.2 Custo da plataforma

Tendo em vista que um dos objetivos do trabalho é desenvolver uma plataforma HIL de baixo custo, é importante mostrar o valor da plataforma, uma vez que todos seus componentes já foram introduzidos. A tabela 2 reune essas informações, apresentando o preço de cada componente individualmente e o custo total da plataforma HIL desenvolvida neste trabalho.

Tabela 2 – Tabela de custos da plataforma HIL construida. Cotação do dólar utilizada: R$ 3,71, referente à época de realização do orçamento da plataforma, em agosto de 2015.

Componente	**Preço (R$)**	**Preço ($)**
Computador *host*	R$ 1800,00	$ 485,18
PCI DAC-6703	R$ 7644,75	$ 2060,58
PCI DAS-6013	R$ 4689,25	$ 1263,95
BeagleBone Black	R$ 400,00	$ 107,82
Custo total	**R$ 14534,00**	$ 3917,52

O custo total descrito na tabela 2 é inferior ao custo da grande maioria das plataformas comerciais disponíveis no mercado. Entretanto, é importante ressaltar mais uma vez que a plataforma HIL projetada também é muito mais simples, de forma que a performance da mesma é bem inferior à dos modelos comerciais já supracitados.

Mesmo assim, a plataforma desenvolvida é capaz de realizar a simulação de modelos de diversas áreas de aplicação, importante contribuição do trabalho. Mesmo que se trate de uma simulação bem mais simples, mostrar o comportamento das entradas e saídas do modelo mediante determinada estratégia de controle embarcada em *hardware* já é uma importante etapa na validação dos sistemas desenvolvidos.

3.3 Funcionamento da plataforma

Uma vez que os componentes tenham sido descritos, é necessário entender como os mesmos interagem durante a simulação, para fazer com que a plataforma HIL funcione de maneira adequada.

A figura abaixo (Fig. 17) apresenta um diagrama com todas as etapas que ocorrem durante a simulação HIL de um sistema na plataforma desenvolvida.

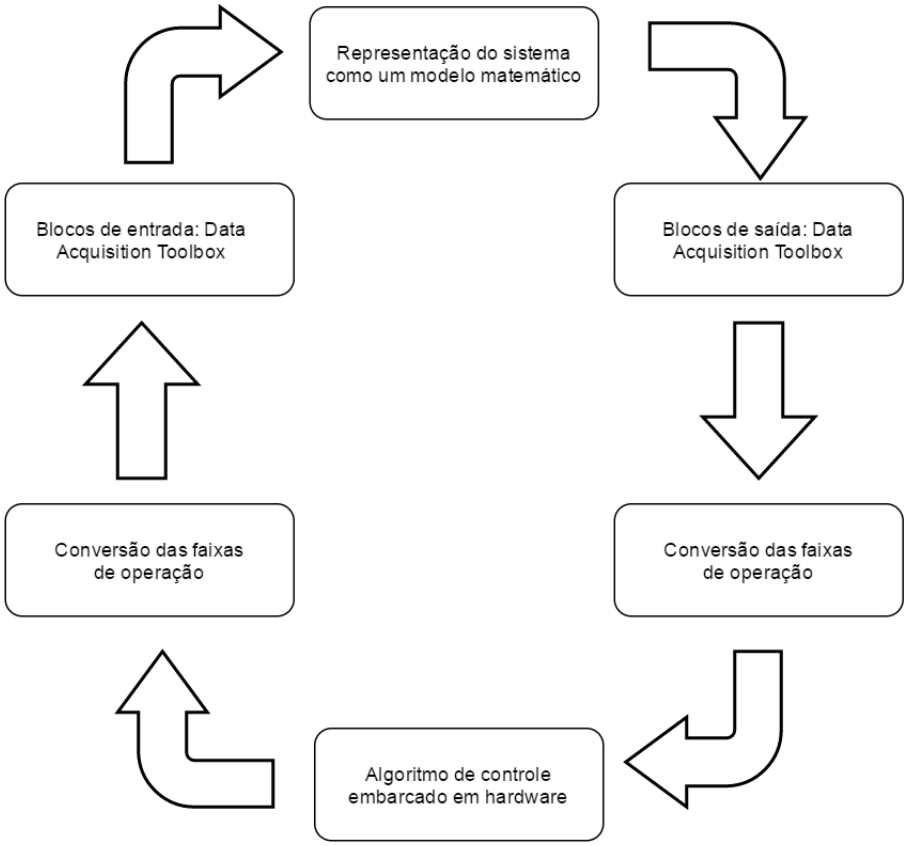


Figura 17 – Diagrama de funcionamento da plataforma.

Os modelos a serem simulados são feitos no computador *host*, em ambiente Simulink, no MATLAB. De maneira geral, os modelos são representados por blocos, que possuem uma descrição matemática do mesmo, através de funções de transferência ou representações em espaço de estados, por exemplo.

Para que o modelo se comunique com o controlador embarcado no *target*, são utilizados os sistemas de aquisição de dados. Estes possuem interface com o Simulink, através da *Data Acquisition Toolbox*, possibilitando que o modelo se comunique com ambas através de blocos no próprio arquivo do modelo no Simulink.

Cada funcionalidade é representada por um bloco: *Analog Input*, *Analog Output*, *Digital Input* e *Digital Output* (Fig. 18), que correspondem a entradas analógicas (disponíveis na PCI DAS-6013), saídas analógicas (disponíveis na PCI DAC-6703), entradas digitais (disponíveis em ambos os modelos de sistema de aquisição de dados) e saídas digitais (também presentes nos dois sistemas de aquisição de dados). Cada bloco reconhece automaticamente o sistema de aquisição de dados instalado no computador e permite que configurações sejam feitas, como quantidade de canais a serem utilizados, faixa de tensão de operação, taxa de amostragem, etc (Fig. 19).

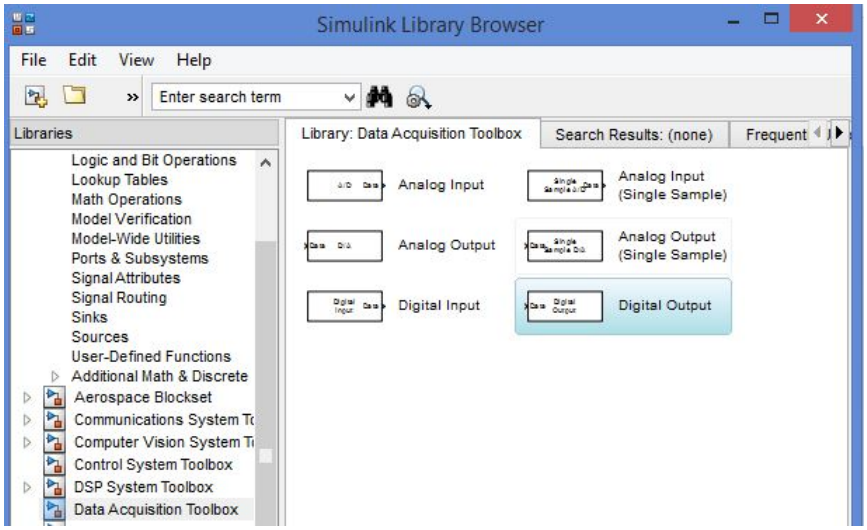


Figura 18 – Menu da *Data Acquisition Toolbox* no Simulink.

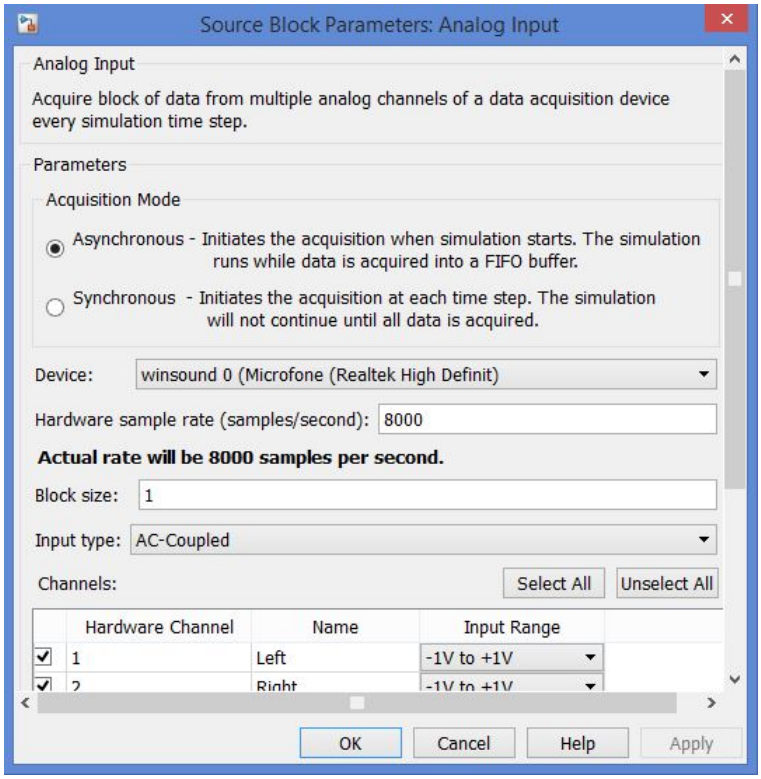


Figura 19 – Parâmetros de configuração dos blocos da *Data Acquisition Toolbox* no Simulink.

A placa PCI DAC-6703 disponibiliza 16 saídas analógicas, que recebem as saídas do modelo a ser simulado. As saídas são conectadas nos pinos de entrada analógica da *Beaglebone Black*, tomando as devidas precauções. Os pinos analógicos da *BeagleBone Black* funcionam com valores de tensão entre 0 e 1.8V e possuem uma resolução mínima de aproximadamente 0.439 mV, já que suas entradas analógicas são de 12 bits. O sistema de aquisição de dados possui faixa de operação de -10 a 10V, com uma resolução mínima de

aproximadamente 0,305 mV. Assim, na maioria das vezes é necessário converter os valores de saída do modelo da sua faixa de operação para uma faixa de 0 a 1.8V, impedindo que a *BeagleBone Black* seja danificada.

Esse tipo de operação pode ser realizada tanto através de circuitos divisores de tensão em *hardware*, ou utilizando equações de conversão de faixas de operação em *software*. A última estratégia foi a escolhida para o projeto por ser mais prática e simples de implementar, uma vez que a primeira metodologia exigiria a construção de circuitos para cada uma das saídas analógicas disponíveis. Além disso, é bem simples realizar essa conversão no MATLAB, utilizando blocos no Simulink para implementar a equação de conversão desejada. Assim, os dados podem ser transmitidos da PCI DAC-6703 para a *BeagleBone Black* sem riscos.

A equação utilizada para conversão de uma determinada faixa de valores de operação para uma nova faixa pode ser vista abaixo, na equação 3.1

$$V_C = (V_E - In_{min}) \left(\frac{In_{max} - In_{min}}{Out_{max} - Out_{min}} \right) + Out_{min} \tag{3.1}$$

em que V_C corresponde ao valor convertido, V_E é o valor de entrada, In_{max} e In_{min} correspondem aos valores máximo e mínimo da faixa de valores da entrada, enquanto Out_{max} e Out_{min} representam os valores máximo e mínimo da nova faixa de valores com a qual deseja-se trabalhar. Por exemplo, na conversão da faixa de operação dos sistemas de aquisição de dados para as entradas analógicas da *BeagleBone Black*, tem-se que In_{max} e In_{min} valem 10 e -10, respectivamente, enquanto Out_{max} e Out_{min} são 1.8 e 0, resultando em:

$$V_C = (V_E + 10) \left(\frac{20}{1.8} \right)$$

Uma vez que os dados estejam na *BeagleBone Black*, são convertidos novamente para a faixa de operação original do sistema sendo simulado, para que os cálculos e algoritmos de controle sejam efetuados com os dados corretos.

Com os dados devidamente calculados e a saída do controlador preparada, é necessário realizar mais conversões. A *BeagleBone Black*, assim como muitos microprocessadores, só possui saídas analógicas do tipo PWM. Apesar de essas saídas serem muito úteis em algumas situações, como no controle de motores, os valores em PWM não podem ser lidos de maneira efetiva por alguns sistemas, de forma que se torna necessário obter o valor da variável de comando na saída do controlador. A metodologia empregada para solucionar essa problemática será discutida em detalhes na próxima seção deste capítulo.

Uma vez obtido o valor de saída, o mesmo precisa se comunicar com o modelo novamente, de forma que as saídas do controlador se tornam entradas do modelo, fechando

o *loop* característico de uma simulação HIL. Para isso, são utilizadas as entradas analógicas da PCI-DAS 6013 (quando utilizadas variáveis de controle em PWM) ou entradas digitais disponíveis tanto na PCI-DAC 6703 quanto na PCI-DAS 6013 (para variáveis de controle que não podem ser transmitidas por PWM), característica a ser abordada na próxima seção.

Nesse ciclo de funcionamento, é possível perceber que os sistemas de aquisição de dados possuem papel fundamental no condicionamento dos sinais do sistema, tanto para o computador *host* quanto para o microprocessador. Sem a utilização correta desses componentes, não só a simulação pode ser prejudicada, como os demais componentes que compõe a plataforma HIL também podem ser danificados.

3.4 Transmissão das variáveis de comando

Como dito em seções anteriores, a *BeagleBone Black* conta com 8 saídas PWM. Esse tipo de técnica é muito comum em microcontroladores e microprocessadores, já que a maioria deles possui periféricos específicos para isso. É uma funcionalidade simples de implementar, e assim, uma das formas mais simples de produzir saídas analógicas (THOREN; STEWARD, 2015).

O problema com as saídas PWM é que elas não são interessantes em todas as situações. Esse tipo de saída costuma ser bastante útil em aplicações que envolvem o controle de motores, por exemplo, pois possibilitam um controle maior sobre o torque e a velocidade de rotação dos mesmos de forma mais eficiente do ponto de vista energético. Entretanto, outras aplicações não funcionam bem com sinais de comando em PWM, pois não são capazes de filtrar esse sinal para obter de maneira efetiva o valor da variável de comando, como no caso de motores. Assim, como um dos objetivos da plataforma HIL desenvolvida é a simulação de modelos de diferentes áreas de aplicação, é de suma importância que existam outras maneiras de transmitir os valores analógicos de uma variável de comando proveniente do algoritmo de controle, sem ser através de uma saída analógica PWM.

Existem algumas abordagens que visam resolver esse problema, como a filtragem do sinal PWM, obtendo-se uma saída analógica propriamente dita, como na figura 20.

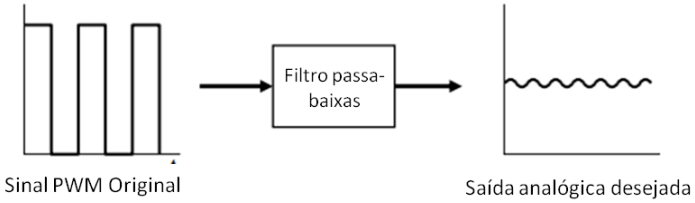


Figura 20 – Filtragem de um sinal PWM utilizando um filtro passa-baixas. Adaptado de: (ALTER, 2006).

A técnica de filtragem geralmente consiste na utilização de filtros passa-baixa. Esse tipo de filtro remove as componentes em alta frequência do sinal, fazendo com que somente o componente DC esteja presente ao final do processo (ALTER, 2006).

O problema desse processo é que ele gera um atraso relativamente grande. Na simulação do tipo HIL sempre existirá um pequeno atraso entre a variável de comando teórica e a variável de comando proveniente do algoritmo de controle embarcado em *hardware*, de forma que para obter uma conversão com boa resolução e atrasos pequenos nesse tipo de processo, seria necessária a implementação de circuitos de filtros de ordem elevada em *hardware* para cada saída PWM da *BeagleBone Black* Porém, quanto maior a ordem do filtro, maior a quantidade de componentes necessários para sua construção 20.

Uma segunda abordagem seria a obtenção do valor médio do sinal PWM, com o uso de blocos do Simulink. Essa técnica se baseia no fato de que nas saídas PWM, o valor médio do sinal está relacionado com o seu ciclo de trabalho, ou *duty cycle*. Por exemplo, para os sinais da *BeagleBone Black*, que possuem tensão de operação de 3.3V, um valor analógico de 1.65 (metade da escala de 0 a 3.3V) é convertido em um ciclo de trabalho de 50%. Como o valor mínimo do sinal PWM da *BeagleBone Black* é de 0V, a saída resultante corresponde a um sinal periódico retangular, cujo valor médio corresponde ao valor modulado.

Novamente, o problema dessa metodologia consiste na dificuldade em diminuir o atraso do sinal final. Muitos sistemas são sensíveis a atrasos em sua entrada, de forma que atrasos a princípio pequenos, acabam acarretando em instabilidade na saída do modelo.

Assim, a melhor solução não é adaptar o sinal analógico PWM de alguma saída da *BeagleBone Black*. A solução adotada foi a de transformar valores analógicos em digitais, transmitindo-os de forma paralela, através dos canais digitais das placas de aquisição de dados (PCI-DAC 6703 e PCI-DAS 6013) e dos GPIOs presentes na *BeagleBone Black*. Assim, é possível utilizar saídas PWM nas situações em que a mesma for interessante, mas também possuir uma alternativa para sistemas que não funcionam bem com esse tipo de sinal.

O interessante dessa técnica é que, além de diminuir consideravelmente o atraso existente nas abordagens anteriores, ela pode ser mais facilmente adaptada a outros microprocessadores, já que nem todos possuem saídas analógicas (sejam elas PWM ou não), mas a grande maioria possui vários GPIOS a disposição.

As etapas dessa metodologia de conversão são:

- **Escolha da resolução da conversão:** Tanto PCI-DAC6703 quanto PCI-DAS6013 possuem 8 canais digitais, que podem ser configurados como entrada e saída, disponibilizando 16 canais digitais ao todo. Isso permite que possam ser utilizados até 16 *bits* para a representação de um valor analógico, ou seja, $2^{16} - 1 = 65535$

possibilidades. É possível escolher então a quantidade de *bits* que possua a melhor resolução para o sistema em questão. Também é possível utilizar mais ou menos bits para representar mais de uma saída: se o sistema possui duas saídas, por exemplo, pode-se utilizar 8 *bits* para cada uma, ou até mesmo conferir maior resolução a uma das saídas, atribuindo 10 *bits* para sua representação e 6 *bits* para a representação da segunda saída.

- **Conversão dos valores analógicos da saída para valores digitais:** Uma vez que a resolução tenha sido escolhida, é possível converter os valores da saída para valores digitais. Para isso, basta saber a faixa de valores na saída do modelo e realizar uma conversão para a faixa de resolução adotada, por meio da equação 3.1. Por exemplo, a saída de um sistema cujos valores variam entre -1 e 1 (independente da unidade de medida do problema) ao ser representada por 8 *bits*, deve ser convertida da faixa de -1 a 1 para a faixa de 0 a $2^8 - 1 = 255$, resultando na relação:

$$S_D = (S_A + 1) * \frac{2}{255}$$

em que S_D corresponde a saída em valor digital e S_A ao valor analógico a ser convertido em S_D. Nesse exemplo, -1 corresponde a 0, e 1 corresponde a 255. Assim, têm-se a representação da variável de controle como um valor digital, em base binária: o valor 255 seria convertido em $(11111111)_2$, enquanto 0 seria convertido em $(00000000)_2$.

- **Associação de *bits* e GPIOs:** Cada *bit* do valor binário obtido deve ser associado a um GPIO da *BeagleBone Black*, para que o número binário seja enviado de forma paralela aos sistemas de aquisição de dados, chegando até o modelo do sistema sendo simulado no MATLAB.

- **Conexão entre GPIOs e canais digitais dos sistemas de aquisição de dados:** Como tanto a PCI-DAC 6703 quanto a PCI-DAS 6013 possuem canais digitais, qualquer uma pode ser usada aqui para conexão dos GPIOs (é possível até mesmo utilizar ambas ao mesmo tempo). Cada GPIO é conectado a um canal digital em um dos sistemas de aquisição de dados e cada canal utilizado é representado em um bloco de *Digital Input* no Simulink, através da *Data Acquisition Toolbox*.

- **Habilitar *enable* de leitura:** Cada vez que o valor analógico da variável de comando muda, sua representação em binário também muda. Como cada *bit* é atribuido a um GPIO, e não há uma maneira de atualizar todos os GPIOS ao mesmo tempo na *BeagleBone Black*, é necessário adicionar uma variável de *enable*, ou seja, que habilita a escrita do valor binário no MATLAB, para que o valor enviado pelo microprocessador seja lido somente quando todos os *bits* que compõem o número binário forem atualizados. O *enable* é desabilitado após a escrita do valor binário no MATLAB.

- **Reconstrução do valor analógico no Simulink:** O valor analógico pode ser reconstruido no Simulink através dos valores de cada *bit* obtido de forma paralela. Em uma palavra de 8 *bits*, como no exemplo adotado, o bit mais significativo corresponde a $2^{8-1} = 128$ vezes o valor do *bit* em questão. A palavra pode ser reconstruida, conhecendo-se o valor de cada *bit* e seu peso para reconstrução do valor analógico:

$$V_A = \sum_{i=0}^{i=n-1} b_i 2^i$$

em que n corresponde ao número de *bits* do valor binário a ser convertido em analógico e b_i ao valor do i-ésimo *bit*.

A questão do pino de *enable* é muito importante nessa metodologia. Sem isso, toda vez que o valor da variável de comando muda, são lidos vários valores intermediários que são incorretos, devido à transição individual de cada *bit*. Isso gera um sinal bastante ruidoso, tornando necessário a utilização do *enable* no MATLAB, através do bloco de *Enabled Subsystem*, repesentado na figura 21. Esse bloco garante que a saída do bloco só será atualizada quando a variável de *enable* for igual a 1. Enquanto a mesma for 0, a saída do sistema mantêm seu valor anterior.

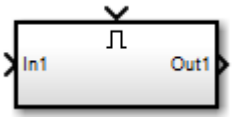


Figura 21 – Bloco de *Enabled Subsystem* disponível no Simulink.

Com esses passos, é possível obter o valor analógico calculado pelo algoritmo de controle embarcado na *BeagleBone Black*, com o mínimo de atraso possível, requisito que as abordagens anteriores não conseguiram atender.

3.5 Ciclo de desenvolvimento na plataforma

Agora que o funcionamento da plataforma HIL já foi apresentado, assim como parte das funcionalidades da plataforma, é importante explicar como funciona o ciclo de desenvolvimento para utilização da mesma:

- **Simulação computacional do modelo do sistema:** O primeiro passo é realizar a simulação do modelo que está sendo desenvolvido, juntamente com o sistema de controle, no próprio Simulink. Como dito na seção 2, simulações ainda são partes essenciais de um projeto, mesmo que não reproduzam exatamente o comportamento do sistema em situações reais. Essa etapa é importante, pois é através dela que é

possível obter conhecimento sobre as condições de operação do sistema a ser controlado, como as faixas de variação das saídas e entradas do modelo, por exemplo. Isso é crucial para o interfaciamento com o controlador e correto uso da plataforma.

- **Configuração das entradas e saídas usando a *Data Acquisition Toolbox*:** Uma vez que o comportamento do sistema já é conhecido, pode-se escolher as faixas de operação que serão utilizadas nos sistemas de aquisição de dados, bem como as taxas de amostragem necessárias para a correta simulação do modelo proposto.

- **Elaboração das equações de conversão de faixas de operação:** Implementar as conversões que serão utilizadas para condicionar os sinais que entram e saem tanto do modelo quanto do controlador. Essas conversões foram discutidas nas seções anteriores, e podem ser implementadas utilizando a equação 3.1.

- **Elaboração do algoritmo de controle em C:** É possível desenvolver algoritmos em diversas linguagens, mas para esse projeto foram priorizados os algoritmos desenvolvidos em linguagem C. Isso se deve ao fato de ser uma linguagem de programação amplamente conhecida, de fácil portabilidade de uma plataforma para outra e que possui boa performance computacional.

- **Embarcar algoritmos de controle em hardware:** Uma vez que todas as conversões foram implementadas, através de blocos no Simulink, é necessário embarcar o algoritmo de controle na *BeagleBone Black*. Isso envolve por exemplo, ler canais analógicos e enviar os dados através de saídas analógicas PWM da placa. Para isso, a biblioteca *libpruio* é utilizada, permitindo que essas funcionalidades sejam acessadas com um única instrução, como Adc->Value para obter o valor de uma entrada ou pwm_ setValue para configurar uma saída analógica PWM com frequência e *duty cycle* desejados.

- **Realizar conexões entre os componentes da plataforma HIL:** Conexões devem ser feitas entre o microprocessador e os sistemas de aquisição de dados. Não há conexão física propriamente dita entre o modelo e os sistemas de aquisição de dados, uma vez que essa interface já é feita automaticamente com os blocos da *Data Acquisition Toolbox*. Os canais de saída da PCI DAC-6703 são conectados nos pinos analógicos de entrada da *BeagleBone Black* e os pinos de saída da *BeagleBone Black* são conectados aos canais de entrada da PCI DAS-6013. Os GPIOs, caso utilizados, podem ser conectados nos canais digitais de ambas as placas.

- **Realizar a simulação HIL:** Por fim, com todos os itens anteriores implementados, pode-se realizar a simulação HIL e averiguar o comportamento do sistema.

Para facilitar a simulação HIL propriamente dita, foram feitos alguns *templates*, ou seja, arquivos padrões e pré-configurados para realizar essas etapas de maneira mais

simples. Foram desenvolvidos três arquivos: hil_ setup.m, model_ template.slx e control_ template.c. Tanto hil_ setup.m quanto control_ template.c encontram-se em anexo no final deste trabalho.

O primeiro arquivo (hil_ setup.m) é um *script* que deve ser executado antes de cada simulação HIL. Ele carrega todos os parâmetros do sistema, como limites em cada entrada e saída, número de *bits* a ser utilizado para representação do sinal de comando, variáveis utilizadas na representação matemática do sistema, tempo de amostragem, etc. Todos esses dados são preenchidos pelo usuário, de forma que seja possível testar diferentes condições de operação para o modelo, sem ser necessário fazer modificações no modelo propriamente dito.

O arquivo model_ template.slx já contem um modelo pré-configurado para realizar a simulação HIL. Ele é separado em três subsistemas: entradas, modelo e saídas, como mostra a figura 22. O subsistema de entradas já possui uma entrada configurada para variável de comando através dos canais digitais dos sistemas de aquisição de dados e uma entrada analógica proveniente da PCI-DAS 6013. O subsistema modelo é onde encontra-se a representação matemática do sistema físico a ser testado. Por fim, o subsistema de saídas possui quatro saídas analógicas da PCI-DAC 6703 pré-configuradas para funcionar com os limites estabelecidos no arquivo hil_setup.m.

Figura 22 – *Template* de modelo para simulação HIL.

O *template* control_ template.c é um exemplo de como o algoritmo de controle deve ser embarcado na *BeagleBone Black*. Ele já possui em seu cabeçalho as bibliotecas necessárias para seu funcionamento, desde bibliotecas padrões da linguagem C (como stdio.h e stdlib.h), como uma biblioteca desenvolvida para implementar funções que realizam as etapas descritas na seção 3.4, como converter um valor analógico em binário ou ler um canal analógico do microprocessador usando a biblioteca *libpruio*. Antes do programa principal são definidos os valores mínimos e máximos das entradas e saídas do controlador, de forma que qualquer mudança de condições de operação do sistema podem ser efetuadas diretamente nessa seção.

Há uma seção específica para o algoritmo de controle, que o usuário da plataforma deve editar para embarcar a estratégia de controle desejada. Após a seção do algoritmo,

há funções responsáveis por transmitir o valor da variável de comando para o sistema de aquisição de dados, e, posteriormente, ao modelo do sistema físico no MATLAB.

É importante salientar que para a simulação funcionar corretamente, os valores máximos e mínimos das entradas e saídas do modelo e do controlador devem ser iguais. Todas essas informações podem ser editadas pelo usuário, de forma que o mesmo deve realizar modificações com muita atenção. Entradas e saídas podem ser adicionadas ou removidas, desde que o usuário siga os mesmos padrões descritos nesses três arquivos.

4 Validação experimental da plataforma HIL

Este capítulo descreve os testes realizados para validar experimentalmente a pltaforma desenvolvida neste trabalho. Em um primeiro momento, realizou-se a calibração da plataforma, para que fosse possível realizar simulações HIL de forma adequada.

Posteriormente, foram realizadas as simulações HIL de dois modelos distintos, com duas estratégias de controle diferentes. Primeiramente, foi realizada a simulação HIL de um sistema com referência variável, controlado por um algoritmo de controle PI embarcado na *BeagleBone Black*. Por fim, foi realizada a simulação HIL do modelo de uma suspensão automotiva, desta vez com um controlador LQR embarcado na *BeagleBone Black*.

Cada uma das seções a seguir trata de um dos testes descritos acima, apresentando os resultados experimentais obtidos com a plataforma para cada um.

4.1 Calibração da plataforma HIL

Pelos assuntos abordados em seções anteriores, é possível perceber que os sinais de entrada e saída do modelo passam por diversos componentes durante a simulação HIL. Por isso, é necessário garantir que o erro entre os sinais recebidos em cada etapa da simulação seja o menor possível, para que os resultados da plataforma HIL sejam confiáveis.

Nesse contexto, torna-se necessário realizar um processo de calibração nas etapas mais cruciais desse processo.

O ponto mais importante no processo como um todo é o momento em que um valor é recebido pela *BeagleBone Black*. É preciso garantir que a variação entre a saída do modelo e os dados de entrada da *BeagleBone Black* seja mínima, para que os dados corretos possam utilizados no algoritmo de controle embarcado.

Como dito na seção 3.2, esses componentes possuem uma resoluções mínimas diferentes: 0.305 mV nos sistemas de aquisição de dados e 0.439 mV da *BeagleBone Black*, aproximadamente. Por isso, ocorre uma diferença entre os valores enviados do *host* e os valores lidos no *target*, uma vez que o primeiro possui uma resolução maior e consegue representar valores menores do que a resolução mínima da *BeagleBone Black*. Qualquer valor menor do que 0.439 mV e maior que 0.305 mV consegue ser enviado pelo *host* de forma adequada, mas é lido na *BeagleBone Black* como 0.439 mV. Na prática, isso influencia nas casas decimais dos números enviados do Simulink para a *BeagleBone Black*.

O experimento realizado para obter a equação de calibração dos dados enviados pelo MATLAB e recebidos pela *BeagleBone Black* consistiu em enviar sinais constantes

do Simulink para o microprocessador, através dos blocos de saída analógica da PCI-DAC 6703 na *Data Acquisition Toolbox* e dos pinos de entrada analógicos da *BeagleBone Black*, como mostrado na figura 23. Nesse teste, os dados foram recebidos, lidos e visualizados na própria *BeagleBone Black*, utilizando a função Adc->Value, da biblioteca *libpruio* para ler o valor obtido nos canais analógicos do microprocessador. Essa função retorna um valor inteiro, que varia entre 0 e 65520, que corresponde a um valor na faixa entre 0 e 1.8V, faixa de operação dos canais analógicos da *BeagleBone Black*.

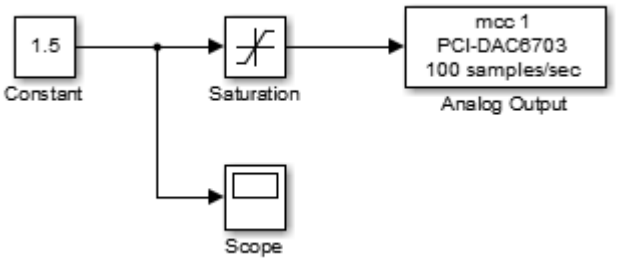


Figura 23 – Sistema de testes para calibração dos valores obtidos pelas entradas analógicas da *BeagleBone Black*.

É muito importante que os dados enviados do Simulink possuam os mesmos valores que os dados lidos na *BeagleBone Black*, pois esse erro pode se propagar durante a simulação causando incertezas e resultados incorretos. Assim, a equação 3.1 pode ser usada para obter o valor correto em Volts da medição realizada em um canal analógico da *BeagleBone Black* (Eq. 4.1):

$$V_C = V_e \left(\frac{1.8}{65520} \right) \tag{4.1}$$

onde V_C é o valor convertido, lido na *BeagleBone Black*, e V_E corresponde ao valor proveniente do sistema de aquisição de dados PCI-DAC6703.

Os dados obtidos ao realizar-se esse teste encontram-se na tabela abaixo (Tab. 3).

Tabela 3 – Dados enviados do Simulink recebidos na textitBeagleBone Black.

Dados enviados do Simulink	**Dados obtidos na *BeagleBone Black***
0,00000	0,00004
0,10000	0,10003
0,20000	0,20226
0,30000	0,30491
0,40000	0,40696
0,50000	0,50907
0,60000	0,61115
0,70000	0,71423
0,80000	0,81581
0,90000	0,91852
1,00000	1,02075
1,10000	1,12263
1,20000	1,22509
1,30000	1,32760
1,40000	1,42987
1,50000	1,53194
1,60000	1,63466
1,70000	1,73680

Com esses dados, é possível realizar um ajuste linear entre os dados enviados pelo Simulink e os dados lidos na *BeagleBone Black*, através do método de mínimos quadrados. A equação referente a essa operação é mostrada abaixo (Eq. 4.2), bem como uma representação gráfica da mesma (Fig. 24).

$$y = 0.9777x + 0.002 \tag{4.2}$$

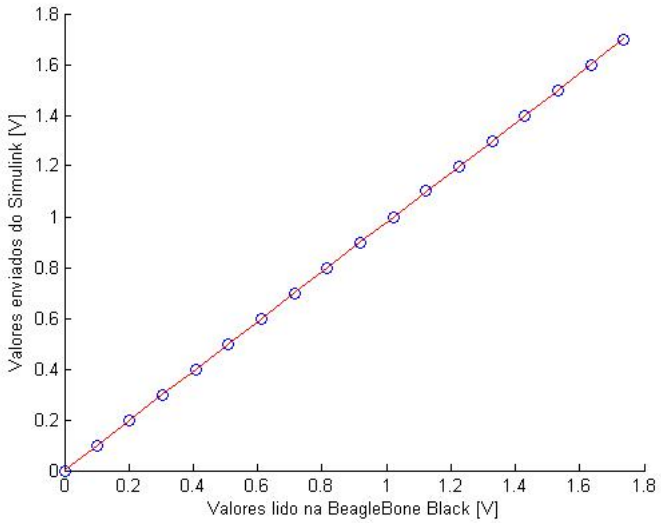


Figura 24 – Ajuste linear entre valores enviados do Simulink e dados lidos na *BeagleBone Black*.

Na equação 4.2, y corresponde aos dados calibrados e x corresponde aos dados lidos na *BeagleBone Black*. Essa equação 4.2 foi obtida utilizando a função *polyfit* do MATLAB, que realiza o cálculo dos coeficientes da equação de ajuste linear referentes aos valores experimentais obtidos na tabela 3. Como se trata de um ajuste linear, a função *polyfit* foi utilizada com o parâmetro n = 2, ou seja, cálculo de um polinômio de ordem 2.

Não é necessário realizar a calibração dos dados enviados da *BeagleBone Black* para o Simulink, pois eles são obtidos através dos *bits* enviados pelo microprocessador. Como esses valores correspondem a 0 ou 1, o MATLAB consegue reconstruir exatamente o mesmo valor que foi enviado pela *BeagleBone Black*.

4.2 Simulação de um sistema com controlador PI

O primeiro modelo simulado na plataforma HIL foi uma função de transferência de primeira ordem, como mostrado na figura abaixo (Fig. 25). Esse modelo recebe como entradas a variável de controle proveniente do algoritmo embarcado na *BeagleBone Black* e uma medição de referência, proveniente de um potenciômetro que varia entre os valores 0 e 1.8 V.

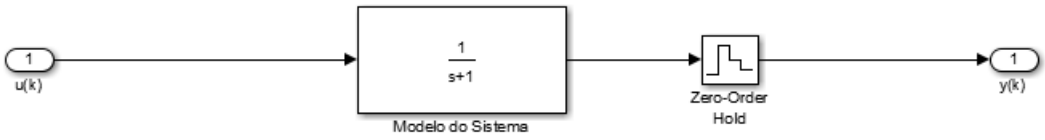


Figura 25 – Modelo do sistema de teste com referência variável.

Esse tipo de sistema é um bom primeiro teste para a plataforma, pois trata-se de um modelo relativamente fácil de controlar, cuja dinâmica não é prejudicada pelo uso de saídas analógicas de PWM do microprocessador. Isso ocorre porque a função de transferência utilizada corresponde a um filtro de primeira ordem. Isso permite que o sistema filtre o sinal PWM proveniente do algoritmo de controle, obtendo o valor da variável de comando.

Esse modelo não representa um sistema físico específico. Apesar disso, é um sistema que permite testar se a comunicação entre todos os componentes da plataforma funciona adequadamente durante uma simulação HIL, além de validar um primeiro algoritmo de controle embarcado na *BeagleBone Black*.

O objetivo do controlador desse sistema é manter a saída do modelo no valor de referência estabelecido pela medição do valor do potenciômetro. A estratégia de controle adotada para esse fim foi o uso de um controlador PI (controlador proporcional inegrativo). Esse controlador é uma variação da estrutura clássica do controlador PID ou controlador

proporcional, integrativo e derivativo, apresentada na figura 26. O controlador PI não é nada mais do que um controlador PID cuja constante derivativa é igual a 0.

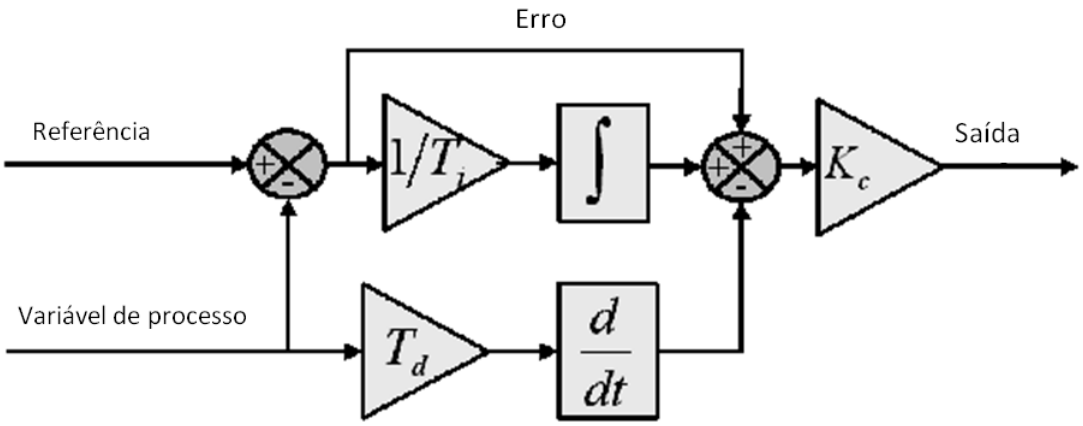


Figura 26 – Diagrama de blocos de um controlador PID padrão. Adaptado de: (INSTRUMENTS, 2013).

O controlador PI possui comportamento semelhante ao PID, de forma que cada componente é responsável por implementar uma característica do sistema de controle, atuando sobre o erro de medição presente, que é caracterizado pela diferença entre o valor medido e a referência do sistema. O fator de proporcionalidade (K_P) influencia diretamente o ganho do sistema, bem como sua velocidade de resposta. Entretanto, um valor muito alto de K_P acarreta em um circuito menos estável. Já a constante integral (K_I), atua diretamente sobre o erro de estado estacionário, diferença final entre variável de controle e referência, fazendo com que o mesmo seja conduzido para zero. No controlador PI, a constante derivativa K_D é igual a zero, de forma que a mesma não é utilizada. No controlador PID, essa componente faz com que o sistema reaja mais fortemente a variações no parãmetro de erro, oferecendo uma correção ao erro presente nas medições. Entretanto, quanto maior o valor de K_D, maior o tempo de resposta do sistema.

No controlador PI, como a componente derivativa corresponde a 0, a saída do sistema possui resposta mais rápida. Entretanto, o mesmo também é menos susceptível a variações de medição na referência do sistema. Apesar disso, ele é uma boa alternativa quando o sistema possui sinais de medição muito ruidosos (INSTRUMENTS, 2013), como no caso do modelo utilizado nesse teste, cujo sinal de referência provém de um potenciômetro.

Outro fator importante para a escolha dessa estratégia de controle é que tanto PID quanto PI são alternativas de controle simples de implementar e eficientes. De fato, graças a técnicas consolidadas de sintonização e o avanço do controle digital, o PID continua sendo amplamente utilizado, de forma que cerca de 90% dos controladores na indústria são implementados utilizando algoritmos baseados em PID, como o controlador PI (ANG; CHONG; LI, 2005).

Para a simulação HIL desse modelo, foram utilizadas duas entradas analógicas e uma saída. A primeira entrada corresponde à variável de controle u(k) proveniente do algoritmo de controle PI embarcado na *BeagleBone Black*. Essa entrada vem da saída PWM 2A da *BeagleBone Black* (correspondente ao pino P_ 21 na mesma) através do canal A0 da PCI-DAS 6013. O valor dessa entrada é convertido da escala de 0 a 3.3V da saída PWM da *BeagleBone Black* para a escala de 0 a 1.8V da referência, através do bloco 'Conversão 0 a 3.3V para 0 a 1.8V', que implementa a equação 3.1 com os valores descritos acima. A segunda entrada do modelo é o valor medido no potenciômetro, que servirá como referência (*Set-Point*) para o sistema. Esse valor pode ser alterado dinamicamente durante o teste, mudando-se o valor do potenciômetro. As entradas do sistema podem ser vistas na figura 27.

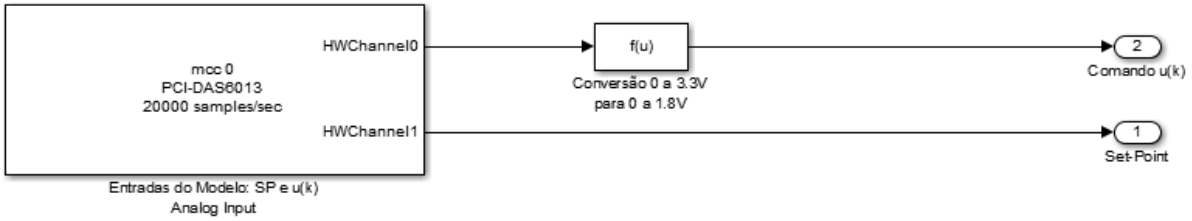


Figura 27 – Entradas do modelo com referência variável no Simulink.

Para se obter valores corretos na simulação, é necessário tomar muito cuidado com as taxas de amostragem escolhidas. Valores incorretos podem causar o fenômeno conhecido como *aliasing*, onde o sinal amostrado é reconstituido de forma diferente do sinal original, gerando resultados incorretos na simulação. A variável de controle será obtida através de um sinal em PWM, cuja frequência é de 2kHz (valor padrão para saídas PWM da *BeagleBone Black*). O teorema de Nyquist diz que, para amostrar de maneira correta um sinal é necessário ter uma frequência de amostragem no mínimo duas vezes a frequência máxima do sinal a ser amostrado. Na prática, utilizam-se sistemas com frequências de amostragem de 10 vezes a frequência máxima do sinal (LOCKHART, 2016), o que resulta na utilização de uma frequência de amostragem de 20kHz para a PCI-DAS 6013.

Por fim, a saída do modelo corresponde ao sinal que é controlado pelo algoritmo de controle PI na *BeagleBone Black*. Por isso, a saída do modelo é conectada ao canal de entrada analógico AIN0 (pino P9_ 39) da*BeagleBone Black*, através do canal A0 da PCI-DAC 6703. Antes, entretanto, a mesma passa pelo bloco que implementa a calibração dos valores enviados do Simulink para a *BeagleBone Black* (conforme visto na seção 4.1) e um bloco de saturação, impedindo que valores superiores a 1.8V e inferiores a 0V cheguem ao canal analógico da *BeagleBone Black*, protegendo o microprocessador. As saidas do modelo podem ser vistas na figura 28.

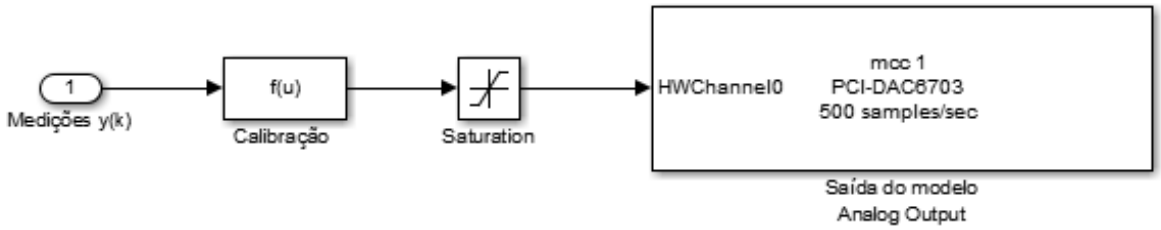


Figura 28 – Saídas do modelo com referência variável no Simulink.

Com o sistema preparado, é possível realizar a simulação HIL do mesmo utilizando a plataforma desenvolvida. As figuras 29 e 30, mostram os resultados da simulação do modelo descrito nessa seção, para diferentes valores de referência. A saída do sistema corresponde à curva em azul, enquanto o valor da referência (valor do potenciômetro) corresponde à curva em vermelho.

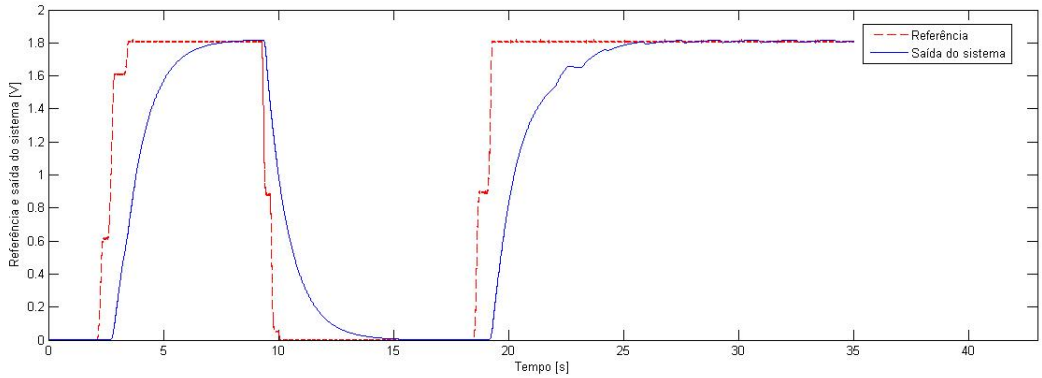


Figura 29 – Simulação HIL do sistema com referência variável e controlador PID para referências iguais a 0 e 1.8.

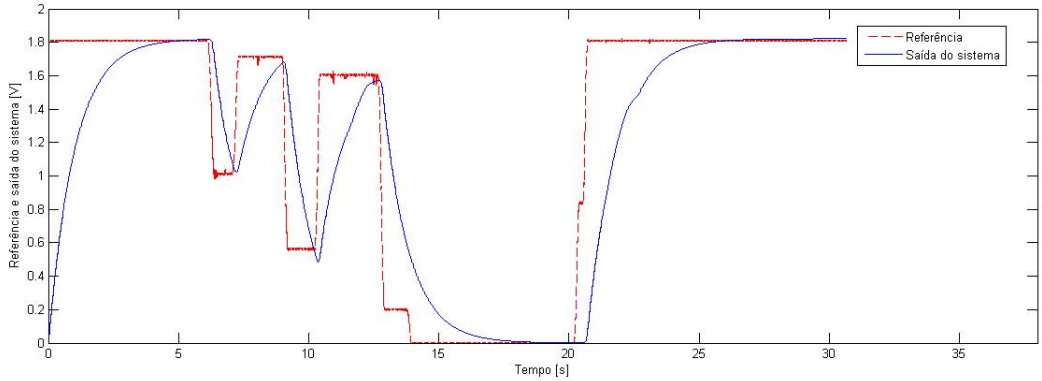


Figura 30 – Simulação HIL do sistema com referência variável e controlador PID para diferentes valores de referência.

É possível perceber que o controlador atua no sistema, fazendo com que a saída siga a referência, estabilizando no valor lido pelo potenciômetro toda vez que ocorre

uma mudança. Entretanto, é interessante notar que a resposta do sistema possui um certo atraso. Isso ocorre porque o algoritmo de controle deve ler uma variável de entrada (proveniente do MATLAB e do sistema de aquisição de dados), processá-la e retornar um valor para a variável de comando, que percorrerá o sistema de aquisição de dados e retornará ao MATLAB. Além disso, o próprio sistema atua como um filtro, atrasando ainda mais o sinal de saida.

O atraso sempre ocorrerá na simulação HIL, de forma que o importante é minimiza-lo ao máximo, para garantir que o mesmo, ao se propagar pelo sistema, não interfira na dinâmica e na simulação do modelo proposto.

O funcionamento correto do sistema permite ver que a comunicação entre modelo, sistemas de aquisição de dados e controlador funcionam bem e que as saídas PWM do microprocessador também funcionaram de maneira adequada durante a simulação, sem a ocorrência de problemas de *aliasing*. Também foi possível validar um primeiro algoritmo de controle na *BeagleBone Black*, que apresentou performance satisfatória.

Entretanto, já foi dito em seções anteriores que nem todos os sistemas utilizam saídas PWM. A simulação realizada, inclusive, mostra que a filtragem de um sinal PWM, para obtenção do valor da variável de comando, gera atrasos significativos na saída, conforme discutido na seção 3.4. O sistema de teste simples apresentado nessa seção consegue utilizar esse recurso sem ter sua dinâmica afetada, mas é necessário realizar testes com modelos que não conseguem lidar com esse tipo de variável de controle, para validar ainda mais a plataforma e permitir o teste de outros tipos de sistemas, como proposto nos objetivos deste trabalho.

4.3 Modelo de suspensão automotiva controlado por LQR

O segundo exemplo utilizado para testar a plataforma foi o modelo de uma suspensão automotiva, com o *Linear Quadratic Regulator*, ou LQR, sendo a estratégia de controle adotada.

Um sistema de suspensão veicular é um mecanismo composto por molas, amortecedores e ligações que conectam a carroceria do veículo às rodas, permitindo que o chassi seja isolado das vibrações provocadas pelo terreno irregular onde o automóvel se encontra (AGHARKAKLI; SABET; BAROUZ, 2012). O sistema a ser testado nessa seção corresponde a um sistema de suspensão passiva, que pode ser aproximado por um sistema massa mola-amortecedor, como na figura 31. Esse tipo de modelagem é conhecida como modelagem de $\frac{1}{4}$ de veículo, onde m_2 corresponde à massa suspensa e m_1 à massa não-suspensa, composta por roda, pneu, disco de freio e outros. Outros parãmetros presentes no modelo são k_2 (que corresponde à rigidez da mola), c (coeficiente de amortecimento), k_1 (que representa a rigidez do pneu), u (que corresponde à excitação de entrada) e os des-

locamentos das massas suspensa e não-suspensa, representados por x e y, respectivamente (OGATA; MAYA; LEONARDI, 2003).

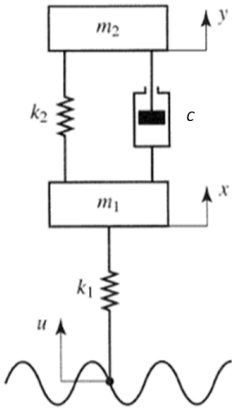


Figura 31 – Sistema de suspensão passiva em modelagem de $\frac{1}{4}$ de veículo. Adaptado de: (OGATA; MAYA; LEONARDI, 2003).

Essa modelagem da suspensão automotiva passiva será representada através de um modelo matemático em espaço de estados no Simulink. Nesse tipo de modelo, as entradas, saídas e variáveis de estados são relacionadas entre si através de equações diferenciais, de forma que o estado atual do sistema pode ser representado através de vetores nesse espaço (OGATA; MAYA; LEONARDI, 2003). Esse tipo de representação do modelo é bem mais sensível a atrasos nos sinais de controle, constituindo um bom teste após a implementação do sistema controlado por PI na seção anterior, onde atrasos no sinal de controle não possuiam grande impacto na dinâmica do sistema. A figura abaixo (Fig. 32) mostra a resposta do modelo da suspensão automotiva, em espaço de estados e com o LQR, quando há um atraso de 0.03s no sinal de controle.

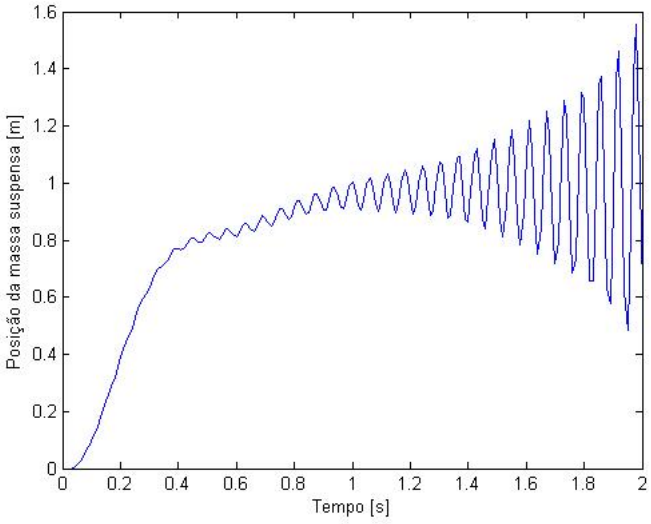


Figura 32 – Instabilidade na posição da massa suspensa devido a atraso de 0.03s no sinal de controle.

Com esse resultado, é possível perceber que é necessário utilizar a abordagem descrita na seção 3.4, enviando o valor da variável de comando analógica por meio digital, de forma paralela.

As saídas do modelo em espaço de estados que estão presentes no modelo correspondem à posição da massa suspensa (x_1), posição da massa não-suspensa (x_2), velocidade da massa suspensa (x_3) e velocidade da massa não-suspensa (x_4).

A estratégia adotada para controlar o modelo da suspensão automotiva foi a técnica de controle conhecida como LQR. Trata-se de um controlador ótimo, ou seja, que realiza operações de otimização para controlar um sistema mediante as restrições de uma função custo, sendo um método amplamente utilizado em problemas de controle. As principais vantagens desse método se baseam no fato do LQR oferecer uma maneira bem sistemática de cálculo da matriz de ganhos de *feedback* de cada estado, bem como produzir um sistema estável ao final do processo (OHRI et al., 2014).

Ao utilizar-se o LQR, escolhe-se um valor de ganho K, de forma que a performance do sistema seja otimizada para uma dada função de custo. Essa matriz de ganhos K pode ser calculada utilizando funções já existentes no MATLAB, como base nos parâmetros do sistema a ser controlado, ou no próprio algoritmo de controle. Nestes testes, utilizou-se da primeira estratégia, com o uso da função *dlqr* no MATLAB, que retorna a matriz K que estabiliza e otimiza um sistema em espaço de estados discretizado, como o modelo da suspensão automotiva adotado.

A equação geral de controle do LQR para esse sistema é representada por (Eq.4.3).

$$u(k) = u_{ss} + K(x_{ss} - x) \tag{4.3}$$

em que $u(k)$ corresponde à variável de comando do processo, u_{ss} ao valor do sistema em regime estacionário, K à matriz de ganhos otimizada do processo, x_{ss} ao valor da referência ou *set-point* do sistema e x corresponde ao valor da saída.

A equação 4.3 pode ser representada por diagramas de blocos, conforme representado na figura 33. Essa figura representa um sistema de três saídas sendo controlado com uso do LQR. Nela é possível ver que cada saída do sistema, representadas em um vetor de saídas x, é multiplicada por uma constante da matriz de ganhos K: x_1 é multiplicado por k_1, x_2 por k_2 e x_3 por k_3. Essas multiplicações são somadas e realimentadas negativamente no sistema, que ainda recebe a soma do termo de regime estacionário u_{ss}, estabelecendo a estratégia de controle do LQR.

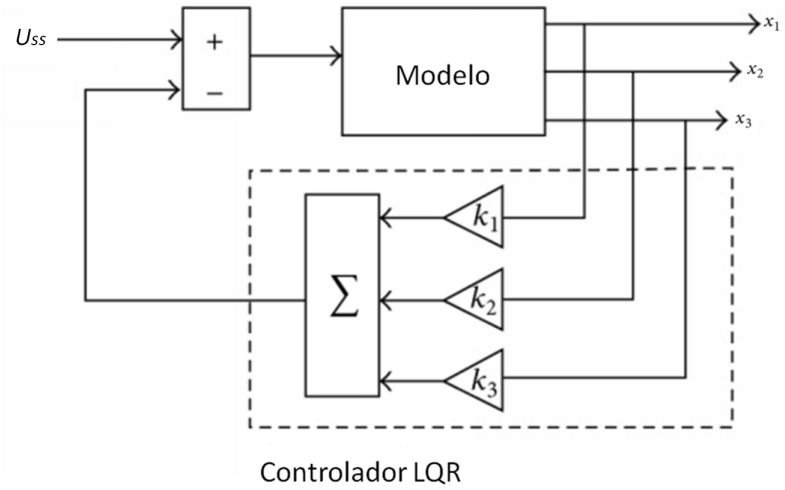


Figura 33 – Diagrama de blocos de um controlador LQR para um sistema de três saídas. Adaptado de: (FAHAMI; ZAMZURI; MAZLAN, 2015)

Com a equação 4.3, o LQR pode estabilizar o sistema com base na matrix K para qualquer valor de *set-point* adotado.

Para realizar a simulação HIL desse sistema, alguns parâmetros do sistema precisam ser definidos para a representação matemática do modelo da suspensão automotiva. Os dados utilizados foram retirados de (SOUSA; ÁVILA, 2015) e encontram-se listados na tabela 4. Além disso, têm-se as matrizes A, B, C e D correspondentes, que representam o modelo em espaço de estados e possuem coeficientes que dependem desses parâmetros. Também são apresentadas as matrizes de ponderação de estados e de comando (Q e R, respectivamente) utilizadas na simulação.

Tabela 4 – Parâmetros para simulação HIL do modelo de suspensão automotiva passiva. Fonte: (SOUSA; ÁVILA, 2015).

Parâmetro	**Símbolo**	**Valor**
Massa suspensa	m_1	203.0 kg
Massa não-suspensa	m_2	26.0 kg
Rigidez do pneu	k_1	66574.0 N/m
Rigidez da mola	k_2	7832.2 N/m
Coeficiente de amortecimento	c	1500 Ns.m

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{k_2}{m_2} & \frac{k_2}{m_2} & -\frac{c}{m_2} & \frac{c}{m_2} \\ \frac{k_2}{m_1} & \frac{(-k_2-k_1)}{m1} & \frac{c}{m_1} & -\frac{c}{m_1} \end{bmatrix} \tag{4.4}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{k_1}{m_1} \end{bmatrix} \tag{4.5}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

$$D = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.7}$$

$$Q = \begin{bmatrix} 1000 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

$$R = 1 \tag{4.9}$$

Com o valor dessas matrizes, é possível discretizar o sistema e calcular um valor de K utilizando a função dlqr do MATLAB. O período de amostragem utilizado foi de 10 ms, resultando na matriz de ganhos otimizados K mostrada a seguir.

$$K = \begin{bmatrix} 0.1426 & -0.9578 & 0.0299 & 0.0072 \end{bmatrix} \tag{4.10}$$

A última definição que precisa ser feita é a quantidade de *bits* necessários para a representação da variável de comando, já que a mesma será transmitida através de canais digitais dos sistemas de aquisição de dados. Para as simulações realizadas com esse modelo, foram utilizados 9 *bits* para representar a saída de comando do sistema, o que garante uma resolução mínima de $\frac{1}{2^9} = 1.95\text{x}10^{-3}$.

Todos esses parâmetros encontram-se no *script* setup_ hil.m, que é executado antes de cada simulação. Isso, além de garantir agilidade nas simulações, também possibilita mudanças nos parâmetros das matrizes que descrevem o sistema de maneira mais fácil.

O primeiro teste realizado com o sistema foi de estabilização da massa suspensa na posição 0 m. A posição inicial adotada para a massa suspensa foi de 1 m, enquanto as demais saídas do sistema possuiam estado inicial 0 m. Esse teste corresponde ao exemplo

clássico de estabilização do LQR em 0, em que x_{ss} e u_{ss} são iguais a 0. Isso gera a seguinte equação de controle a ser utilizada (Eq. 4.11):

$$u(k) = Kx \tag{4.11}$$

Nesta equação, K é a matriz de ganhos e x representa o vetor com as saídas do modelo.

Seguindo as etapas de desenvolvimento na plataforma, descritas na seção 3.6, convém realizar uma simulação do comportamento teórico do sistema, representado pelo modelo da figura 34. Assim, é possível obter os valores máximos e mínimos de cada entrada e saída do modelo, para saber as condições de operação em que a simulação HIL deve ocorrer. As figuras 35, 36 e 37 apresentam as valores da variável de comando e de cada saída após a simulação, enquanto a tabela 5 lista os limites de operação de cada variável do sistema.

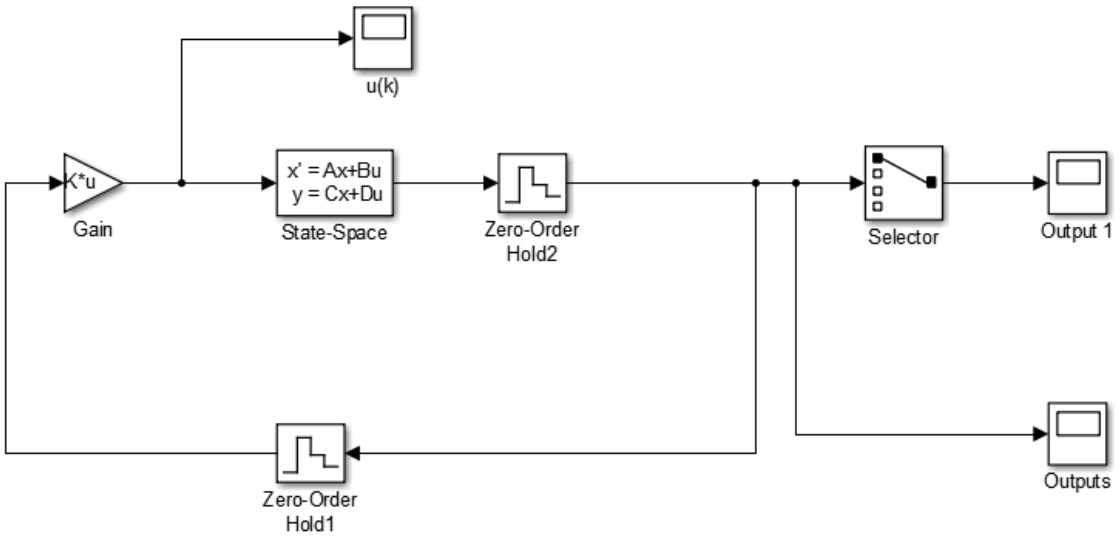


Figura 34 – Sistema de referência teórico para estabilização da posição da massa suspensa em 0 m.

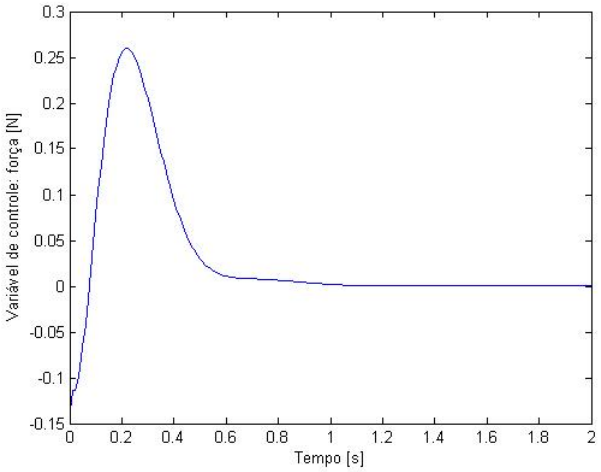


Figura 35 – Variável de controle teórica para estabilização da posição da massa suspensa em 0 m.

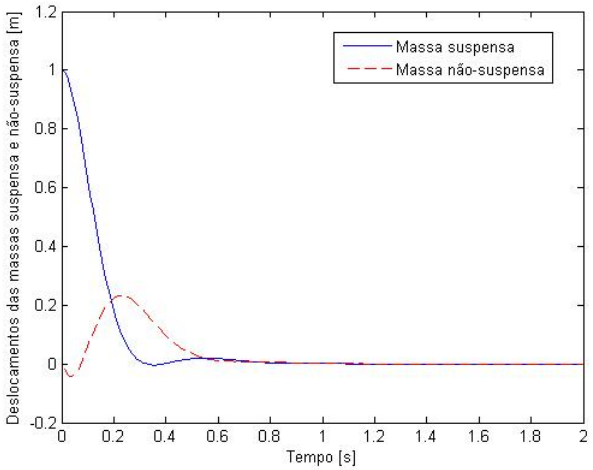


Figura 36 – Variação dos deslocamentos das massas suspensa(azul) e não-suspensa (vermelho) para estabilização da posição da massa suspensa em 0 m.

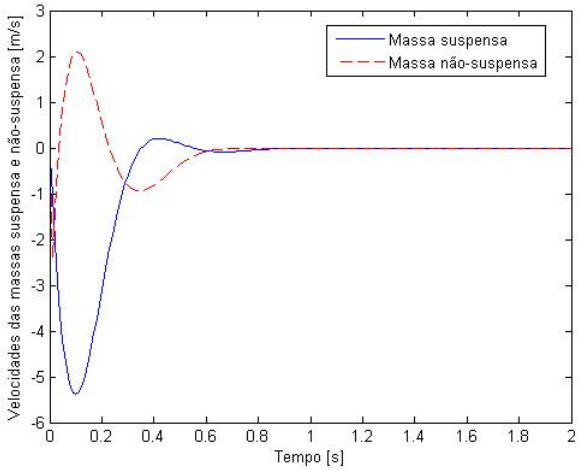


Figura 37 – Variação das velocidades das massas suspensa(azul) e não-suspensa (vermelho) para estabilização da posição da massa suspensa em 0 m.

Tabela 5 – Valores máximos e mínimos para cada entrada e saída do modelo de suspensão automotiva com estabilização da posição da massa suspensa na posição 0.

Parâmetro	**Valor mínimo**	**Valor máximo**
Variável de controle	-0.1426 N	0.2598 N
Posição da massa suspensa	0.000 m	1.000 m
Posição da massa não-suspensa	-0.050 m	0.2598 m
Velocidade da massa suspensa	-5.385 m/s	0.2000 m/s
Velocidade da massa não-suspensa	-2.396 m/s	2.106 m/s

Em posse desses valores, é possível finalmente realizar a simulação HIL do modelo proposto. Graças aos *templates* citados na seção 3.6, basta preencher esses dados tanto no *template* do algoritmo de controle, quanto no *script* setup_ hil.m, para preparar as condições do sistema.

O sistema possui como entrada a variável de controle proveniente do algoritmo LQR embarcado na *BeagleBone Black*. Essa variável é transmitida utilizando 9 GPIOs da *BeagleBone Black*, que são conectados aos canais digitais D0 a D7 da PCI DAC-6703 e ao canal digital D0 da PCI DAS-6013. O GPIO que representa o *enable* para leitura da variável de controle no MATLAB é conectado no canal digital D1 da PCI DAS-6013. Os demais blocos desse subsistema são responsáveis por reconstruir um valor analógico a partir da variável de controle em formato digital obtida, realizando as etapas descritas na seção 3.6. Todo o subsistema de entradas descrito acima encontra-se na figura 38.

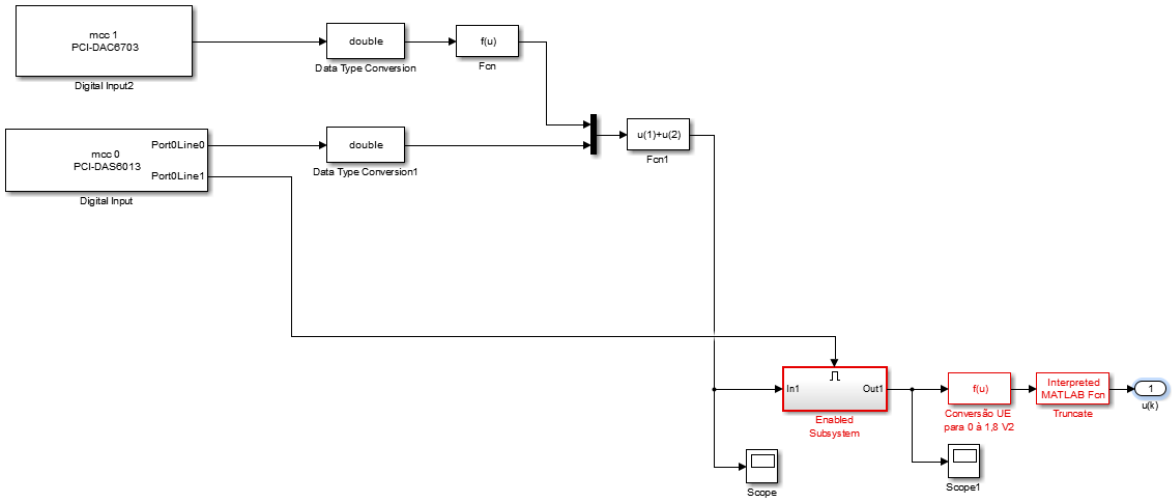


Figura 38 – Entradas do modelo de suspensão automotiva controlado por LQR configuradas para a simulação HIL.

As saídas do sistema são conectadas a quatro saídas analógicas da PCI DAC-6703: a posição da massa suspensa é conectado ao canal A0, posição da massa não-suspensa é conectada ao canal A1, velocidade da massa suspensa conecta-se ao canal A2 e, por fim, a velocidade da massa suspensa é conectada ao canal A3. Todas são

convertidas de suas faixas de operação listadas na tabela 5 para a faixa de 0 a 1.8V (faixa de operação dos canais analógicos da *BeagleBone Black*), passam pelo bloco de calibração e de saturação para garantir que os valores corretos serão lidos no microprocessador, sem que ocorram danos ao mesmo. A figura 39 representa essa configuração das saídas do modelo no Simulink.

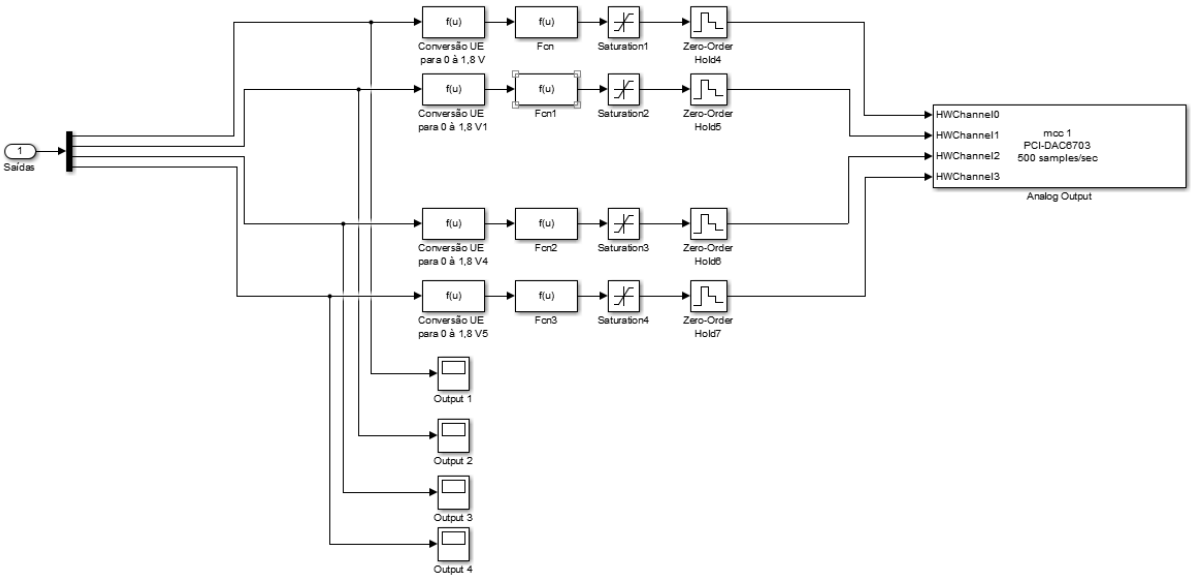


Figura 39 – Saidas do modelo de suspensão automotiva controlado por LQR configuradas para a simulação HIL.

A posição da massa suspensa é conectada à entrada analógico AIN0 da *BeagleBone Black* (pino P9_ 39) e a posição da massa não-suspensa, à entrada AIN1 (pino P9_40). A velocidade da massa suspensa é conectada à entrada analógica AIN2 (pino P9_37), enquanto a velocidade da massa não-suspensa é conectada à entrada analógica AIN3 (pino P9_38). Todas essas informações são lidas no algoritmo de controle, através da função Adc -> Value. Além disso, cada valor lido é convertido de volta para a sua faixa de valores original, para que os cálculos referentes à lei de controle do LQR sejam feitos com os valores reais de cada saída.

Essa etapa é um passo muito importante, já que é necessária para garantir que os dados submetidos à estratégia de controle embarcada na *BeagleBone* sejam corretos. A função que realiza essa conversão já está implementada no *template* de algoritmo de controle e integrada com os valores máximos e mínimos correspondentes a cada entrada do controlador. Esses valores são definidos antes do programa principal, na seção de definição de constantes, como mostrado na seção anterior.

Com tudo isso preparado, as simulação HIL foi realizada, e seus resultados podem ser vistos na figuras 40 e 41.

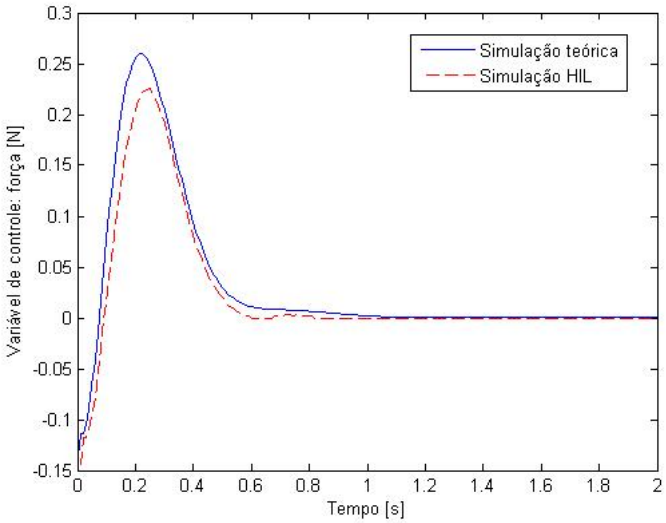


Figura 40 – Variável de controle obtida durante a simulação HIL do modelo de suspensão automotiva para estabilização da massa suspensa na posição 0 m (vermelho), comparado com o valor obtido em simulação teórica (azul).

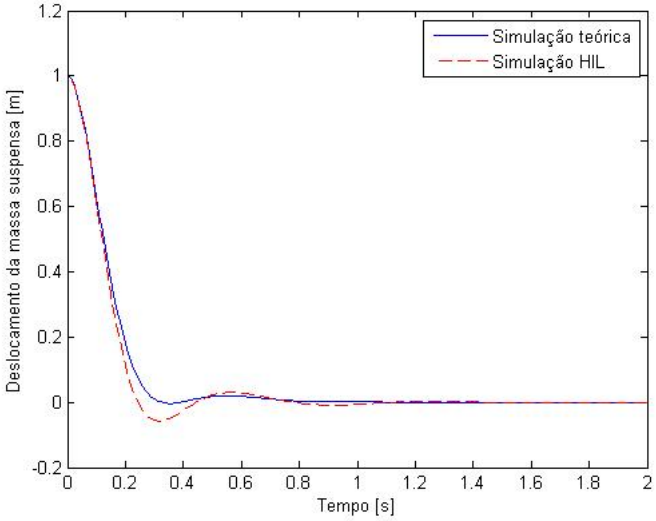


Figura 41 – Saída correspondente ao deslocamento da massa suspensa durante a simulação teórica (azul) e a simulação HIL (vermelho) para estabilização de sua posição em 0 m.

Com os gráficos, é possível perceber que o sistema de controle funciona, estabilizando a posição da massa suspensa em 0. Entretanto, na simulação HIL, a posição da mola assume valores negativos de maior magnitude do que na simulação teórica do sistema. Isso ocorre porque a simulação teórica não leva em consideração atrasos na variável de comando. A simulação HIL produz um atraso de 0.01s, ou um período de amostragem, responsável por essa ligeira diferença de comportamento.

Para demonstrar isso, pode-se realizar a simulação teórica do modelo atrasando o sistema em um período de amostragem, resultando na figura 42.

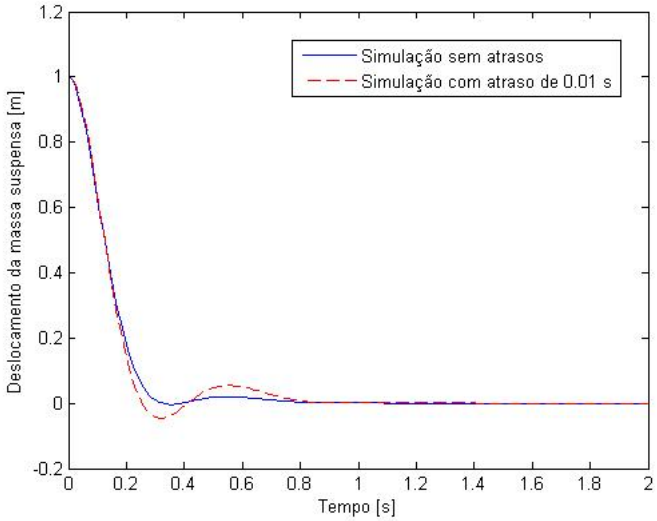


Figura 42 – Simulação teórica da massa suspensa estabilizando na posição 0 m com variável de controle atrasada em 0.01s (vermelho) e sem atrasos (azul).

É possível notar que realmente a suspensão assume valores negativos de maior magnitude quando atrasos são considerados, alcançando um valor mínimo diferente da simulação que não considera atrasos.

Além de validar o controlador para este caso, o teste também possibilita ver que a metodologia de transmissão da variável de comando de forma digital e paralela também se mostrou efetiva. É importante ressaltar que essa conversão, na prática, precisa ocorrer a uma taxa maior que a taxa de amostragem do sistema, para gerar o menor atraso possível. Os experimentos mostraram que, em geral, cerca de 1000 vezes mais rápida para garantir um atraso mínimo no sinal de comando. Isso é facilmente configurável no Simulink já que cada bloco utilizado pode ter sua taxa de amostragem configurada, e o bloco de *Digital Input* herda a taxa de amostragem dos blocos aos quais ele está conectado.

A limitação dessa metodologia é o fato de que alguns valores assumidos pela variável de controle não correspondem a um número binário inteiro. Por exemplo, em uma faixa com valores que variam entre -5 e 5, com uma resolução de 8 bits, o número 0 quando convertido para a faixa de 0 a $2^8 - 1 = 255$, corresponde a 127.5. O método descrito na seção 3.4 converte somente a parte inteira desse valor para binário, de forma que 0 corresponderia a 127. Mas, 127 na realidade corresponde ao número -0.019 quando o valor analógico é reconstruido no MATLAB. Assim, quando o valor da variável de controle fosse igual a 0, na verdade, seria exibido no Simulink o valor -0.019. Isso se torna importante quando o valor que estabiliza a saída do sistema não pode ser representado como um inteiro na faixa de valores adotada. Assim, é necessário realizar o truncamento

do número. Para o exemplo definido acima, pode-se definir que os valores 127 e 128 devem ser interpretados como 0, possibilitando que o sistema estabilize nesse valor.

Durante a simulação HIL deste teste, a variável de comando oscila entre os valores -0.002 e 0.00578, já que o valor 0 não corresponde a um valor inteiro quando convertido para a faixa de operação das variáveis de comando. A função de truncamento faz com que a mesma estabilize de fato em 0, como mostrado anteriormente.

Após atestar o funcionamento do controlador para estabilização da posição da massa suspensa em 0, pode-se realizar um novo teste, procurando estabilizar a posição da massa suspensa em valores maiores que 0. Para isso, é necessário adotar uma nova equação de controle, já que u_{ss} e x_{ss} não serão mais iguais a zero.

O segundo teste corresponde a estabilizar a posição da massa suspensa em 1. A posição inicial da mesma será 0, assim como a posição inicial de todas as outras saídas. Em casos onde a referência de estabilidade para o sistema é diferente de 0, u_{ss} recebe o valor em que a massa suspensa deve estabilizar, enquanto x_{ss} recebe esse valor para as duas primeiras saídas do sistema, x_1 e x_2 (referentes às posições das massas suspensa e não-suspensa), enquanto x_3 e x_4 permanecem com $x_{ss} = 0$ (referente às velocidades das massas suspensa e nao-suspensa). Assim, a equação 4.3 é modificada para:

$$u(k) = 1 - K_1(1 - x_1) - K_2(1 - x_2) - K_3(0 - x_3) - K_4(0 - x_4) \tag{4.12}$$

Antes de realizar a simulação HIL, é interessante realizar mais um vez a simulação teórica do sistema de controle nessas condições (Fig. 43 a Fig. 46), para saber a nova faixa de valores necessária para simular as novas entradas e saídas do sistema. Os resultados estão descritos na tabela 6.

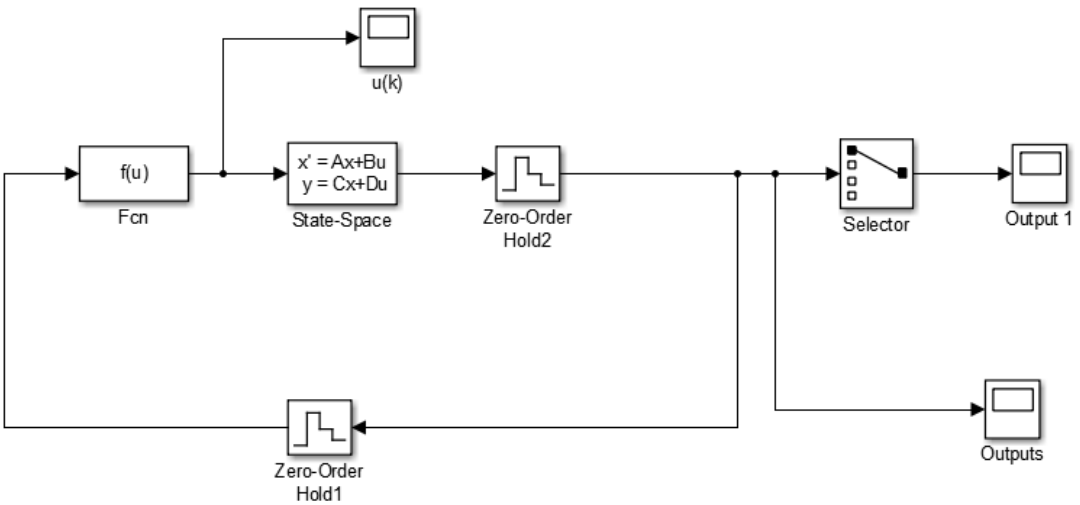


Figura 43 – Sistema de referência teórico para estabilização da posição da massa suspensa em 1 m.

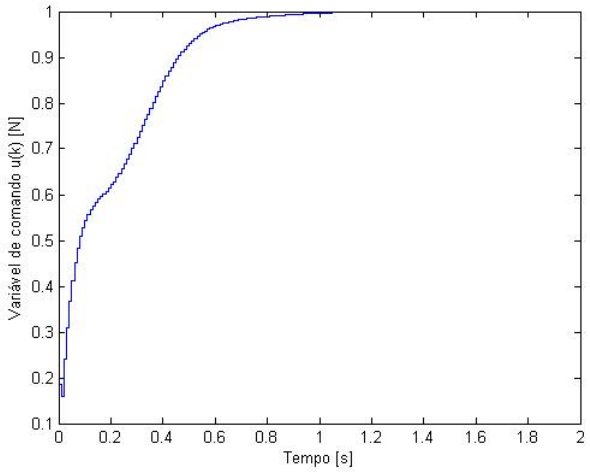


Figura 44 – Variável de controle teórica para estabilização da posição da massa suspensa em 1 m.

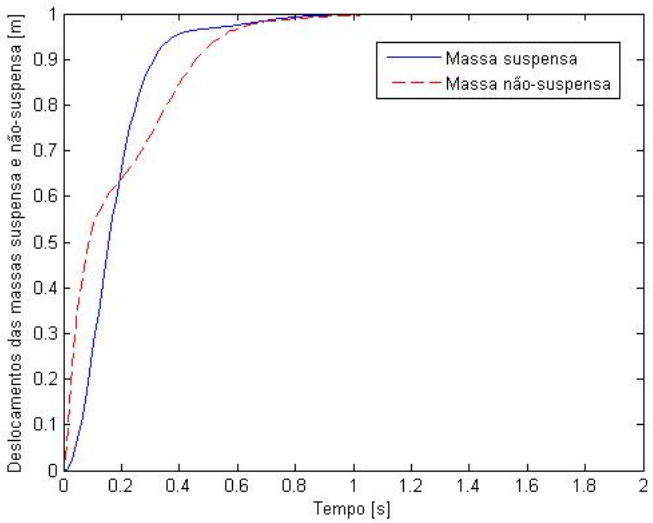


Figura 45 – Variação dos deslocamentos das massas suspensa(azul) e não-suspensa (vermelho) para estabilização da massa suspensa em 1 m.

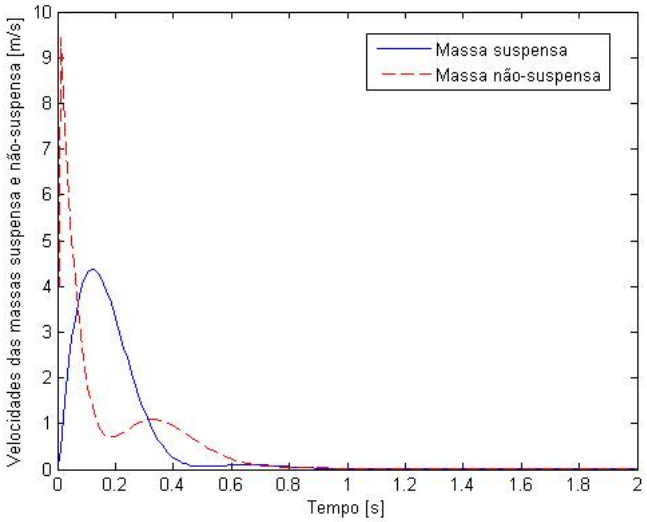


Figura 46 – Variação das velocidades das massas suspensa(azul) e não-suspensa (vermelho) para estabilização da massa suspensa em 1 m.

Tabela 6 – Valores máximos e mínimos para cada entrada e saída do modelo de suspensão automotiva com estabilização da posição da massa na posição 1 m.

Parâmetro	**Valor mínimo**	**Valor máximo**
Variável de controle	0 N	1 N
Posição da massa suspensa	0 m	1 m
Posição da massa não-suspensa	0 m	1 m
Velocidade da massa suspensa	0 m	4.37 m/s
Velocidade da massa não-suspensa	0 m/s	9.42 m/s

Aplicando a nova equação de controle no algoritmo de controle, e utilizando valores máximos e mínimos compatíveis com as entradas e saidas do modelo teórico apresentados na tabela 6, é possível realizar a simulação HIL do modelo. Como as únicas alterações deste teste em relação ao anterior correspondem aos valores máximos e mínimos na entrada e nas saídas do modelo, só é necessários atualizar esses valores, tanto no *script* hil_ setup.m quanto no algoritmo de controle.

Os resultados experimentais obtidos na simulação HIL desse modelo, para estabilização da massa suspensa na posição 1 m, são apresentados nas figuras 47 e 48.

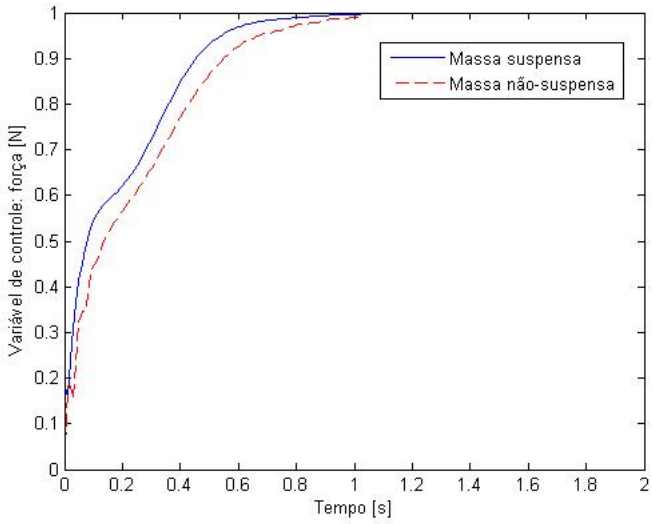


Figura 47 – Variável de controle obtida durante a simulação HIL do modelo de suspensão automotiva para estabilização da massa suspensa na posição 1 m (vermelho) comparado com a variável de controle obtida em simulação teórica (azul).

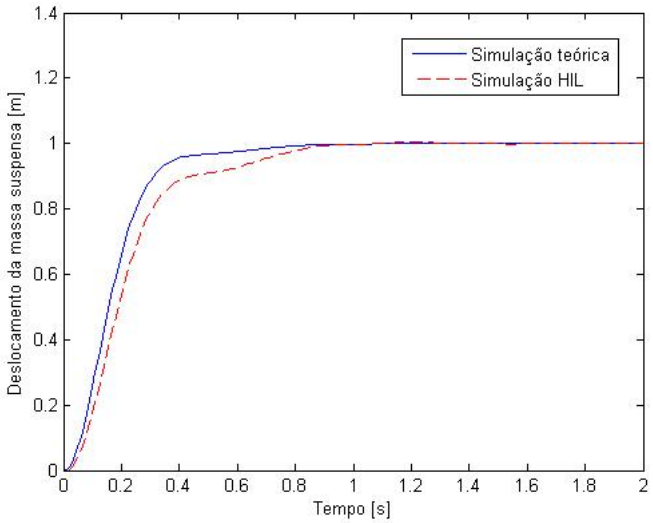


Figura 48 – Variação do deslocamento da massa suspensa durante simulação HIL (vermelho) e durante simulação teórica (azul) para estabilização da sua posição em 1 m.

É possível ver que, assim como no teste de estabilização em 0 m, o controlador funciona bem e estabiliza a saída na posição desejada, ou seja, em 1 m. Mais uma vez é produzido o atraso de um ciclo de amostragem (0.01s), o que produz um ligeiro *overshoot* de 0.01 antes que o sistema alcance a posição 1 m. Entretanto, isso não causa nenhum problema no comportamento do mesmo.

Assim como no exemplo anterior, o valor da variável de comando oscila entre os valores 0.9963 e 1.004, já que o valor 1 não corresponde a um valor inteiro na faixa que representa a variável de comando convertida em binário. Assim, devido à resolução do método de conversão, torna-se necessário truncar o valor da saída em 1 no intervalo proposto.

O último teste realizado com esse sistema, é estabelecer uma lei de controle que permita estabilizar a posição da massa suspensa em qualquer posição. Para isso, com base no teste anterior, basta modificar a equação 4.3 para que u_{ss} seja igual ao valor de referência adotado e que x_{ss} seja igual a esse valor quando aplicada às saídas x_1 e x_2, que correspondem às posições das massas suspensa e não-suspensa, respectivamente. O valor de x_{ss} continua sendo 0 quando aplicado às saídas x_3 e x_4 (velocidade da massa suspensa e velocidade da massa não-suspensa).

Essas considerações resultam na equação definida abaixo (Eq. 4.13).

$$u(k) = x_{ref} - K_1(x_{ref} - x_1) - K_2(x_{ref} - x_2) - K_3(0 - x_1) - K_4(0 - x_4) \tag{4.13}$$

em que x_{ref} corresponde ao valor de referência em que a massa suspensa deve

estabilizar. Para os testes, utilizou-se uma faixa de valores de referência que variam de 0 a 1.8 m, faixa de valores de operação das entradas analógicas da *BeagleBone Black*.

Antes de realizar a simulação HIL para esse caso, é importante saber os limites das entradas e saídas do sistema quando a referência do mesmo encontra-se em 1.8 m. Para isso, basta substituir $x_{ref} = 1.8$ na equação 4.13, e realizar a simulação teórica, utilizando os mesmos procedimentos dos testes anteriores. O resultado é exibido nas figuras abaixo (Fig. 49 e Fig. 50) e seus valores listados na tabela 7. É possível notar nessas figuras que o comportamento das saídas é muito parecido com o comportamento teórico obtido no teste anterior, mas os valores assumidos pelas velocidades de cada massa possui valores máximos maiores, como esperado.

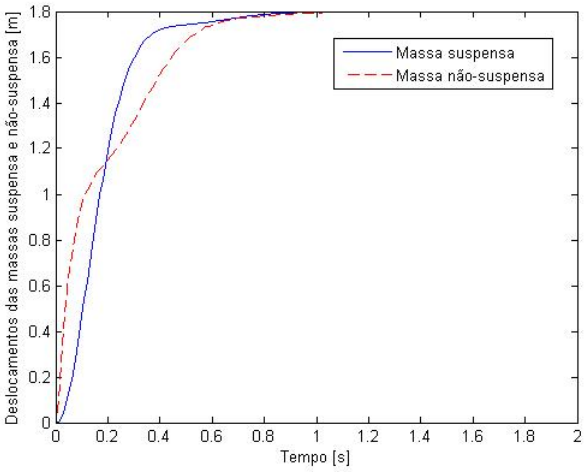


Figura 49 – Variação da posição das massas suspensa(azul) e não-suspensa (vermelho) durante simulação teórica para estabilização da massa suspensa na posição 1.8 m.

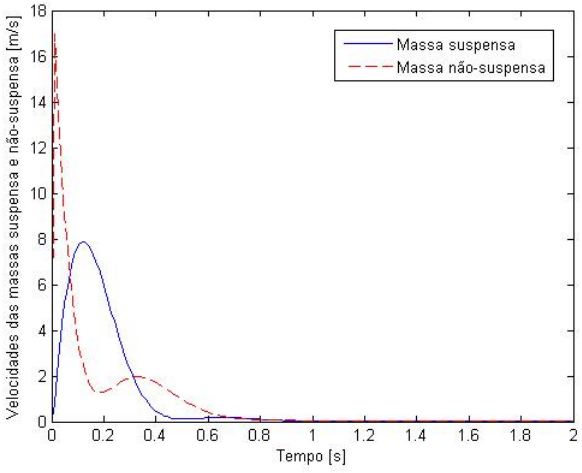


Figura 50 – Variação da velocidade das massas suspensa(azul) e não-suspensa (vermelho) durante simulação teórica para estabilização da massa suspensa na posição 1.8 m.

Tabela 7 – Valores máximos e mínimos para cada entrada e saída do modelo de suspensão automotiva com estabilização da posição da massa suspensa na posição 1.8 m.

Parâmetro	**Valor mínimo**	**Valor máximo**
Variável de controle	0.000 N	1.800 N
Posição da massa suspensa	0.000 m	1.800 m
Posição da massa não-suspensa	0.000 m	1.800 m
Velocidade da massa suspensa	0.000 m/s	7.874 m/s
Velocidade da massa não-suspensa	0.000 m/s	16.959 m/s

Mais uma vez, realizando os mesmos procedimentos experimentais que nos testes anteriores, utilizando faixas de valores máximos e mínimos para as entradas e saídas compatíveis com os dados apresentados na tabela 7 e embarcando a nova equação de controle na *BeagleBone Black*, é possível realizar a simulação HIL desse novo sistema. A referência foi implementada como sendo uma constante no Simulink, que pode ser modificada pelo usuário durante a simulação (Fig. 51). Essa constante é transmitida para a *BeagleBone Black* como mais uma entrada analógica, através do canal A4 da PCI DAC-6703 e conectada ao pino de entrada analógica AIN5 (pino P9_ 36).

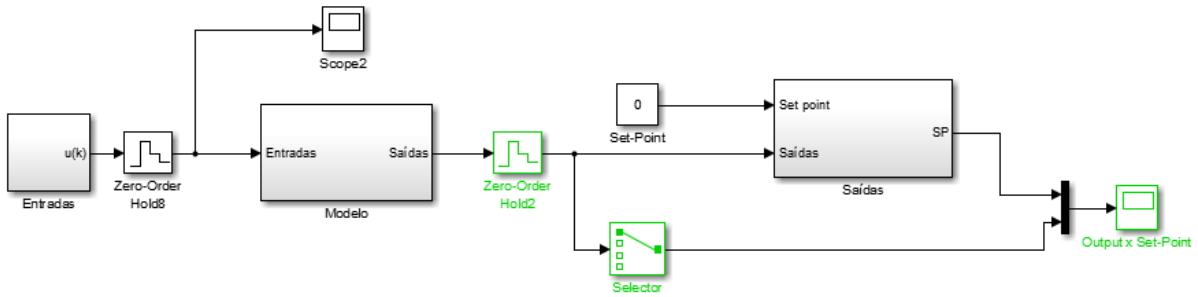


Figura 51 – Modelo de suspensão automotiva com referência variável para estabilização da posição da massa suspensa.

O resultado da simulação HIL desse sistema é apresentado na figura 52.

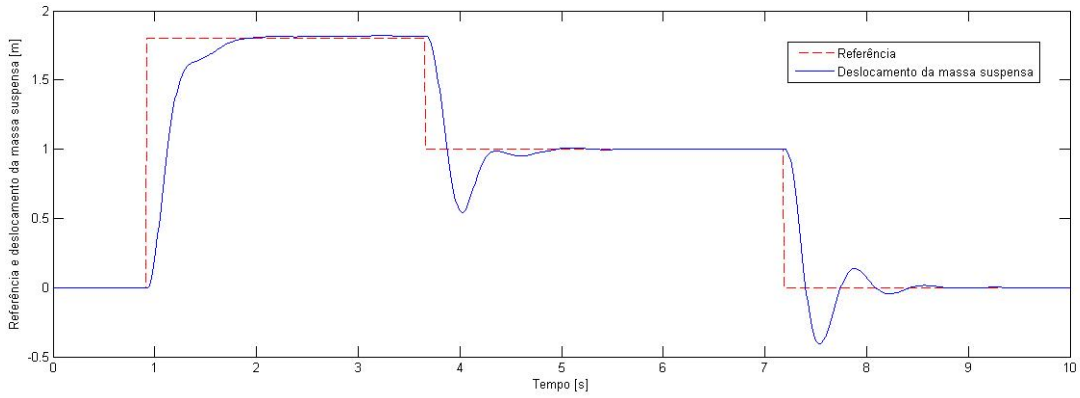


Figura 52 – Simulação HIL do modelo de suspensão automotiva com referência variável (em vermelho) para estabilizaçao da massa suspensa (em azul) por meio de um controlador LQR.

É possível perceber que o sistema estabiliza muito próximo do valor da referência, de forma que pode-se considerar que o resultado obtido para esse modelo é satisfatório, uma vez que o erro em regime estacionário do mesmo é bem pequeno, embora ainda exista.

Com a realização de todos os testes descritos nessa seção, foi possível averiguar que a plataforma HIL construida conseguiu simular um sistema com dinâmica mais rápida, produzindo resultados satisfatórios. Como dito no início deste trabalho, as simulações realizadas são bem mais simples, mostrando somente como as saídas do modelo comportam-se mediante a aplicação de uma determinada estratégia de controle. Apesar disso, as simulações mostraram que, mesmo sendo mais simples, demonstram ser um recurso interessante, tornando evidente que comportamento de sistemas quando implementados com componentes reais pode ser diferente do comportamento do mesmo em ambiente de simulação computacional, mesmo que essa diferença seja pequena.

Foram testados dois sistemas diferentes e duas estratégias de controle diferentes, mostrando que é possível simular modelos de outras áreas de aplicação, bastando ter sua representação matemática em Simulink, além de ajustar as faixas de operação do sistema para produzir resultados corretos.

5 Conclusão

Após todo o levantamento bibliográfico realizado, construção da plataforma HIL e de todos os testes efetuados, é possível perceber que o objetivo proposto foi alcançado. Foram testados tanto modelos simples, de dinãmica mais lenta, como no exemplo do sistema de referência variável com um controlador PI, quanto modelos mais complexos, com dinâmicas e respostas mais rápidas, como a suspensão automotiva controlada com LQR. Em ambos os casos foram obtidos resultados satisfatórios.

A plataforma HIL construida possui baixo custo quando comparada com as plataformas comerciais. Seu custo total é de R$ 14534,00 e modelos comerciais de plataforma HIL custam bem mais do que isso, chegando a valores na casa de 100000 euros. Obviamente, as plataformas comerciais são muito mais robustas e completas do que a plataforma desenvolvida neste trabalho, além de serem capazes de produzir resultados com maior acurácia e precisão. Entretanto, a plataforma desenvolvida possui uma contribuição muito importante além do seu baixo custo: ela é capaz de simular sistemas de diferentes áreas de aplicação, ao contrário da maioria das plataformas comerciais.

O resultado deste trabalho é uma plataforma HIL que pode simular diferentes tipos de modelo, mostrando como suas saídas se comportam para determinadas variáveis de controle atribuidas em suas entradas. Com essas informações, é possível saber como o modelo se comporta em condições mais realistas de operação, lidando com atrasos e erros de medição que geralmente não são levados em consideração em simulações computacionais. Esta é a principal vantagem da simulação HIL, porque, apesar de não ser uma simulação definitiva, a mesma é capaz de atuar como um estágio intermediário entre simulação computacional e implementação do sistema na prática.

Se o comportamento do sistema não for adequado, ou o sistema não é robusto o suficiente a variações de medição, por exemplo, sabe-se que quando implementado na prática, o sistema terá comportamento parecido. Pode-se então, melhorar o modelo ou o algoritmo de controle para otimizar a performance do sistema proposto.

Para a plataforma desenvolvida, desde que sejam respeitadas as condições de operação nas entradas e saídas do modelo, podem ser simulados sistemas de diferentes áreas de aplicação. Pode-se também utilizar outros modelos de microprocessador ou microcontrolador para embarcar o algoritmo de controle, desde que as mesmas possuam funcionalidades compatíveis com os modos de operação adotados nesse trabalho. Essas características aumentam a flexibilidade da plataforma desenvolvida, o que também é uma contribuição importante deste trabalho.

Isso não significa que a plataforma desenvolvida aqui seja um modelo definitivo

e que pode ser utilizada para a simulação de qualquer tipo de sistema. Existem várias melhorias que podem ser implementadas e serão discutidas na próxima seção com mais detalhes. A principal limitação da plataforma é com relação a taxas de amostragem, já que os modelos não são executados em tempo real. A biblioteca *libpruio* garante à *BeagleBone Black* um comportamento bem próximo ao tempo real, mas, como ainda ocorre na prática uma certa diferença entre os intervalos de execução do algoritmo de controle, a rigor, a mesma também não funciona em tempo real. Isso limita a taxa de amostragem na qual os modelos podem ser simulados, já que haverá uma diferença entre simulação do modelo e geração das variáveis de comando por parte do microprocessador. Enquanto o microprocessador executar o algoritmo de controle a uma taxa maior que o que a taxa de amostragem do modelo, é possível simular o sistema utilizando blocos no Simulink como o *Zero-Order Hold* ou *Rate Transition* para amostrar o valor da variável de comando na frequência desejada. Entretanto, se a simulação deve ocorrer numa taxa de amostragem maior que o microprocessador consegue executar o algoritmo de controle, a simulação não é possível.

Como dito no capítulo 2, a simulação em tempo real não é obrigatória em uma simulação HIL, mas é algo desejado, pois permite um maior controle sobre a simulação e garante resultados mais precisos. Apesar dessa limitação, a plataforma desenvolvida se mostrou capaz de simular modelos diferentes, obtendo resultados significativos. Assim, a plataforma HIL desenvolvida constitui um bom ponto de partida para um projeto que ainda pode ser desenvolvido bastante. À medida que a plataforma HIL for sendo incrementada e novos modelos forem sendo testados, a plataforma se mostrará um recurso muito importante para a validação de sistemas de controle em âmbito acadêmico. Até lá, o trabalho desenvolvido até aqui cumpre os objetivos listados no primeiro capítulo, e pode-se considerar que sucesso foi obtido com o mesmo.

5.1 Trabalhos futuros

Como dito na seção anterior, apesar dos resultados satisfatórios obtidos, há espaço para melhorias na plataforma HIL desenvolvida. A primeira melhoria, já comentada também na seção anterior, é implementar a simulação do modelo em tempo real. Esse tipo de simulação garante que tanto o modelo quanto o algoritmo de controle são executados no mesmo instante de tempo, tornando a simulação ainda mais precisa. Isso também diminui atrasos e questões de temporização que não permitem a simulação de sistemas com dinâmicas extremamente rápidas, cuja simulação HIL, em seu estado atual, não produziria resultados satisfatórios.

A simulação em tempo real para o modelo pode ser implementada utilizando-se a *toolbox Simulink Desktop Real-Time* ou a *toolbox Simulink Real-Time*. A primeira utiliza

somente um computador para realizar a simulação em tempo real, enquanto a segunda *toolbox* implementa o modelo em tempo real utilizando dois computadores: um computador *host* que cria um executável a partir do modelo no Simulink e um computador *target*, que recebe esse executável e executa o modelo em tempo real. A intenção era implementar essas metodologias ainda neste trabalho, mas devido a problemas de configuração e compatibilidade dos sistemas de aquisição de dados utilizados, não foi possível implementar essa funcionalidade a tempo.

A implementação do algoritmo de controle em tempo real também não foi implementada neste trabalho, apesar de ser a intenção. A *BeagleBone Black* possui PRUs, que são processadores dedicados à execução em tempo real. Entretanto, eles são bastante complexos de serem utilizados, de forma que não foi possível implementar essa funcionalidade antes do prazo final de entrega deste trabalho. A biblioteca *libpruio* auxilia nesse processo, fazendo com o tempo de execução do algoritmo de controle na *BeagleBone Black* não varie muito, mas como esse tempo não é fixo e preciso, não é possível considerar que a mesma esteja funcionando em tempo real.

A utilização de um módulo separado de RTC (sigla para *Real Time Clock*) pode ser uma boa alternativa para ser implementada no futuro, já que existem exemplos disponíveis de sua utilização não só para a *BeagleBone Black*. Além disso, é uma estratégia facilmente utilizável com outros microprocessadores/microcontroladores como Raspberry Pi e Arduino.

Outra funcionalidade que pode ser implementada no futuro é a utilização de conversores A/D para converter saídas analógicas dos modelos a serem simulados em saídas digitais. Isso é interessante porque nem todos os microprocessadores possuem entradas analógicas como a *BeagleBone Black*, mas a grande maioria possui muitos GPIOs que podem ser usados como entradas ou saídas digitais. Isso traria ainda mais flexibilidade para a plataforma, pois poderiam ser utilizados ainda mais modelos de microprocessadores como opções de *hardware* para embarcar algoritmos de controle, como a Raspberry Pi, por exemplo, que não possui pinos próprios para funcionarem como entradas analógicas.

Melhorias na interface da plataforma HIL com o usuário também podem ser implementadas. Neste trabalho, a interface se deu através de *templates*, que são o primeiro passo para a implementação de opções mais amigáveis para o usuário, como menus interativos, botões e outros recursos que tornem mais fácil a realização das simulações na plataforma.

Todas essas funcionalidades, se implementadas, possibilitarão não somente a simulação de mais modelos, de áreas de aplicação diversas, mas uma plataforma HIL ainda mais completa e capaz de produzir resultados ainda melhores. Isso não significa que a plataforma atual não apresente uma boa performance, mas com essas melhorias pode evoluir e tornar-se um projeto ainda mais completo.

Referências

AGHARKAKLI, A.; SABET, G. S.; BAROUZ, A. Simulation and analysis of passive and active suspension system using quarter car model for different road profile. *International Journal of Engineering Trends and Technology*, v. 3, n. 5, p. 636–644, 2012. Citado na página 72.

ALTER, D. M. Using PWM output as a digital-to-analog converter on a TMS320F280X digital signal controller. *Texas Instruments, Dallas, TX, Application Report SPAA88*, 2006. Citado 3 vezes nas páginas 15, 58 e 59.

ALTERA. *Hardware in the Loop from the MATLAB/Simulink Environment*. 2013. <https://www.altera.com/en_US/pdfs/literature/wp/wp-01208-hardware-in-the-loop.pdf>. Acesso em 24 de maio de 2015. Citado na página 27.

ANDERSON, O. *A Design Tool (Research and Development) Simulator Survey Report*. [S.l.], 1962. Citado na página 33.

ANG, K. H.; CHONG, G.; LI, Y. PID control system analysis, design, and technology. *IEEE transactions on control systems technology*, IEEE, v. 13, n. 4, p. 559–576, 2005. Citado na página 69.

BADARUDDIN, K. S.; HERNANDEZ, J. C.; BROWN, J. M. The importance of Hardware-in-the-Loop testing to the cassini mission to saturn. In: IEEE. *Aerospace Conference*. [S.l.], 2007. p. 1–9. Citado 3 vezes nas páginas 34, 35 e 38.

BEAGLEBOARD.ORG. *BeagleBone Black*. 2016. <https://beagleboard.org/black>. Acesso em 12 de maio de 2016. Citado 2 vezes nas páginas 15 e 52.

COMPUTING, M. *PCI-DAS6013 and PCI-DAS6014 User's Guide*. 2009. Citado na página 49.

DROSDOL, J.; KADING, W.; PANIK, F. The Daimler-Benz driving simulator. *Vehicle System Dynamics*, Taylor & Francis, v. 14, n. 1-3, p. 86–90, 1985. Citado na página 33.

DSPACE. *Application Fields*. 2015. <http://www.dspace.com/en/inc/home/applicationfields/stories.cfm#filterterms=term-133>. Acesso em 24 de março de 2015. Citado 2 vezes nas páginas 29 e 38.

FAHAMI, S. M. H.; ZAMZURI, H.; MAZLAN, S. A. Development of Estimation Force Feedback Torque Control Algorithm for Driver Steering Feel in Vehicle Steer by Wire System: Hardware in the Loop. *International Journal of Vehicular Technology*, Hindawi Publishing Corporation, v. 2015, 2015. Citado 2 vezes nas páginas 16 e 75.

FREEBASIC. *Libpruio: Features*. 2016. <http://users.freebasic-portal.de/tjf/Projekte/libpruio/doc/html/_cha_features.html>. Acesso em 02 de maio de 2016. Citado na página 53.

HANSELMANN, H. *Hardware-in-the Loop simulation as a standard approach for development, customization, and production test of ECU's*. [S.l.], 1993. Citado na página 34.

HOSSEINPOUR, F.; HAJIHOSSEINI, H. Importance of simulation in manufacturing. *World Academy of Science, Engineering and Technology*, v. 51, p. 285–288, 2009. Citado na página 27.

HUBER, W.; JONNER, W.-D.; DEMEL, H. *Simulation, performance and quality evaluation of ABS and ASR*. [S.l.], 1988. Citado na página 34.

INSTRUMENTS, N. *Explicando a teoria PID*. 2013. <http://www.ni.com/white-paper/3782/pt/>. Acesso em 22 de maio de 2016. Citado 2 vezes nas páginas 16 e 69.

ISERMANN, R.; SCHAFFNIT, J.; SINSEL, S. Hardware-in-the-Loop simulation for the design and testing of engine-control systems. *Control Engineering Practice*, Elsevier, v. 7, n. 5, p. 643–653, 1999. Citado 9 vezes nas páginas 15, 28, 29, 33, 34, 35, 36, 42 e 43.

LOCKHART, R. W. *What You Really Need to Know About Sample Rate*. 2016. <http://www.dataq.com/data-acquisition/general-education-tutorials/what-you-really-need-to-know-about-sample-rate.html/>. Acesso em 12 de maio de 2016. Citado 2 vezes nas páginas 42 e 70.

LU, B. et al. A low-cost real-time Hardware-in-the-Loop testing approach of power electronics controls. *IEEE Transactions on Industrial Electronics*, IEEE, v. 54, n. 2, p. 919–931, 2007. Citado 3 vezes nas páginas 27, 28 e 34.

MOLLOY, D. *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*. [S.l.]: John Wiley & Sons, 2014. Citado na página 53.

NI. *Projeto e teste de ECUs com os produtos da National Instruments*. 2012. <http://www.ni.com/white-paper/3312/pt/>. Acesso em 07 de junho de 2015. Citado 3 vezes nas páginas 15, 27 e 28.

OGATA, K.; MAYA, P. Á.; LEONARDI, F. *Engenharia de controle moderno*. [S.l.]: 4ª edição, Pearson Prentice Hall, 2003. Citado 2 vezes nas páginas 16 e 73.

OHRI, J. et al. GA tuned LQR and PID controller for aircraft pitch control. In: IEEE. *2014 IEEE 6th India International Conference on Power Electronics (IICPE)*. [S.l.], 2014. p. 1–6. Citado na página 74.

PALLADINO, A.; FIENGO, G.; LANZO, D. A portable Hardware-in-the-Loop (HIL) device for automotive diagnostic control systems. *ISA transactions*, Elsevier, v. 51, n. 1, p. 229–236, 2012. Citado 3 vezes nas páginas 15, 34 e 37.

PIZETTA, I. H. et al. Uma plataforma hardware-in-the-loop para vants de asas rotativas. In: *Anais do XIX Congresso Brasileiro de Automática, SBA, Campina Grande, PB*. [S.l.: s.n.], 2012. p. 3565–3570. Citado 2 vezes nas páginas 15 e 41.

SANTOS, S. R. B. d.; OLIVEIRA, N. M. F. D. Longitudinal autopilot controllers test platform Hardware in the Loop. In: IEEE. *Systems Conference (SysCon), 2011 IEEE International*. [S.l.], 2011. p. 379–386. Citado 3 vezes nas páginas 15, 39 e 40.

SILVA, H. M. d. *Simulação com Hardware in the Loop aplicada a veículos submarinos semi-autônomos*. Tese (Doutorado) — Universidade de São Paulo, 2008. Citado 3 vezes nas páginas 15, 36 e 40.

SOUSA, D. F.; ÁVILA, S. Suspensão ativa utilizando controle PID em um modelo de 1/4 de veículo. In: ECT. *VII Encontro de Ciência e Tecnologia do Campus UnB Gama*. [S.l.], 2015. Citado 2 vezes nas páginas 19 e 75.

THOREN, M.; STEWARD, C. *Accurate, Fast Settling Analog Voltages from Digital PWM Signals*. [S.l.], 2015. Citado na página 58.

WALTER, S. *Technical Proposal v1.0 for Universidade de Brasilia (UnB) - Generic SCALEXIO HIL System*. [S.l.], 2014. Citado 2 vezes nas páginas 15 e 38.

Anexos

ANEXO A – Hil setup.m

```
1  %% hil_setup.m
2
3  % Author: Alceu Bernardes Castanheira de Farias
4
5  % This file has the main objective of adapting the
       model_template.slx file
6  % to the characteristics of the model to be simulated.
       Just write the needed
7  % information and execute this script before simulating.
8
9  clc; clear all
10 %% System Operation Ranges
11
12 % Mininum value for all system inputs. If you wish to
       use a different range
13 % for each input, change in_min for in_x_min on
       model_template.slx, where x
14 % corresponds to the desired input.
15 in_min = 0.00;
16
17 % Max value for all system inputs. If you wish to use a
       different range
18 % for each input, change in_max for in_x_max on
       model_template.slx, where x
19 % corresponds to the desired input.
20 in_max = 0.00;
21
22 % Input 1 minimum and max values
23 in_1_min = 0.00;
24 in_1_max = 2.00;
25
26 % Mininum value for all system outputs. If you wish to
       use a different range
27 % for each input, change out_min for out_x_min on
       model_template.slx, where x
```

```
28          % corresponds to the desired output.
29          out_min = 0.00;
30
31          % Max value for all system inputs. If you wish to use a
                different range
32          % for each input, change out_max for out_x_max on
                model_template.slx, where x
33          % corresponds to the desired output.
34          out_max = 0.00;
35
36          % Output 1 minimum and max values
37          out_1_min = 0.00;
38          out_1_max = 2.00;
39
40          % Output 2 minimum and max values
41          out_2_min = 0.00;
42          out_2_max = 2.00;
43
44          % Output 3 minimum and max values
45          out_3_min = 0.00;
46          out_3_max = 10.00;
47
48          % Output 4 minimum and max values
49          out_4_min = 0.00;
50          out_4_max = 20.00;
51
52          %% System Parameters
53
54          % Variables needed for simulating the desired model
55
56          % System Data
57          m1 = 26;
58          m2 = 203;
59          k1 = 200000;
60          k2 = 20000;
61          c = 1500;
62
63          % Sampling time
64          Ts=0.01;
```

```
65
66          % Number of bits for representing the control variable
67          res = 9;
68
69          % System model: continuous
70          A=[0 0 1 0; 0 0 0 1; -k2/m2 k2/m2 -c/m2 c/m2; k2/m1 (-k2
                -k1)/m1 c/m1 -c/m1];
71          B=[0;0;0;k1/m1];
72          C=eye(4);
73          D=zeros(4,1);
74
75          % System discretization
76          sist_c=ss(A,B,C,D);
77          sist_d=c2d(sist_c,Ts);
78          Ad=sist_d.a;Bd=sist_d.b;Cd=sist_d.c;Dd=sist_d.d;
79
80          % Discrete LQR gain calculation
81          Q=diag([100 1 1 1]);      % States Weightning
82          R=1;                      % Command Weightning
83          klqr_d=dlqr(Ad,Bd,Q,R);   % Gain calculation
```

ANEXO B – Control template.c

```
1  /* Template for developing control algorithms
2  in C for BeagleBone Black /*
3
4  Author: Alceu Bernardes Castanheira de Farias
5
6  This template uses libpruio library, from FREEBASIC:
7
8  Licence: GPLv3
9  Copyright 2014 by Thomas{ dOt ]Freiherr[ At ]gmx[ DoT }net
10
11 For more information on libpruio, acess the project website:
12 http://users.freebasic-portal.de/tjf/Projekte/libpruio/doc/html/
13 index.html
14
15 Compile the problem using: gcc -Wall -o executable_name
16 control_template.c -lpruio.
17 */
18
19 // Include libraries. DO NOT DELETE ANY LIBRARY FROM THIS
      SECTION. You may include additional ones.
20 #include "stdio.h"
21 #include "stdlib.h"
22 #include "math.h"
23 #include "string.h"
24 #include "time.h"
25 #include "sys/time.h"
26 #include "../c_wrapper/pruio.h" // include header
27 #include <termios.h>
28 #include <unistd.h>
29 #include <errno.h>
30 #include "../c_wrapper/pruio_pins.h"
31 #include "fp2bin.h"
32
33 /* System operation ranges definition. */
34
```

```
35  // Modify them to be compatible with the same operation
       ranges defined on Simulink.
36  #define IN_1_MIN 0
37  #define IN_1_MAX 2000
38  #define IN_2_MIN 0
39  #define IN_2_MAX 2000
40  #define IN_3_MIN 0
41  #define IN_3_MAX 10000
42  #define IN_4_MIN 0
43  #define IN_4_MAX 20000
44  #define OUT_1_MIN 0
45  #define OUT_1_MAX 2.0
46
47  /* VARIABLE DECLARATION */
48
49  // DO NOT ALTER THE VARIABLES BELOW
50
51  // Pointer for writing on BeagleBone Black files that
       correspond to its GPIOs.
52  FILE *fp;
53
54  // String that stores the control variable in binary.
55  char binString[9];
56
57  // Variables that store each bit from the control variable in
       binary.
58  char bit_8;
59  char bit_7;
60  char bit_6;
61  char bit_5;
62  char bit_4;
63  char bit_3;
64  char bit_2;
65  char bit_1;
66  char bit_0;
67
68  // Variables that store the value to be written on the
       correspondent GPIOs.
69  int gpio_11;
```

```
70  int gpio_12;
71  int gpio_7;
72  int gpio_8;
73  int gpio_9;
74  int gpio_10;
75  int gpio_15;
76  int gpio_16;
77  int gpio_17;
78
79  // Variables for reading BeagleBone Black. analog channels
80  int a0, a1, a2, a3;
81
82  // Variables for storing values read from BeagleBone Black
       analog channels in floar format.
83  float a0f, a1f, a2f, a3f;
84
85  // Enable for reading control variable on Simulink.
86  int enable;
87
88  // Control variable .
89  double u = 0;
90
91  // Variable that store the control variable value in digital
       form.
92  int u_out = 0;
93
94  /* Add here the variables needed for your control algorithm.
95  DO NOT ALTER THE VARIABLES ABOVE, unless you want to change
       the number or the GPIOs used.*/
96
97  /* FUNCTIONS*/
98
99  // DO NOT ALTER THE FUNCTIONS BELOW.
100
101 // Function to convert one range of operation into another.
102 int mapRange(int x, int in_min, int  in_max, int out_min, int
       out_max){
103   return (x - in_min)*(out_max - out_min)/(in_max - in_min) +
        out_min;
```

```
104 }
105
106 // Function to transform an analog value into a digital one.
107 int adc(float x, float in_min, float in_max, int out_min, int
      out_max){
108     return (x - in_min)*(out_max - out_min)/(in_max - in_min)
          + out_min;
109 }
110
111 // Function to convert an int variable into a binary string.
112 void int2bin(int a, char *binString){
113
114     int i = 1;
115     int binaryInt = 0;
116
117   while(a>0)
118   {
119     binaryInt = binaryInt + a % 2 * i;
120     i = i * 10;
121     a = a / 2;
122   }
123
124   sprintf(binString, "%09d", binaryInt);
125 }
126
127 /* Add here your control algorithm functions. DO ALTER THE
      ONES ABOVE. */
128
129 /* MAIN FUNCTION */
130 int main(int argc, char **argv)
131 {
132
133   // Driver structure construction for using libpruio. DO NOT
        ALTER.
134   pruIo *io = pruio_new(PRUIO_DEF_ACTIVE, 0x98, 0, 1);
135     if (pruio_config(io, 1, 0x1FE, 0, 4)){
136       printf("config failed (%s)\n", io->Errr);}
137     else {
138
```

```
139         while(1){
140
141         // Reset control variable reading enable. DO NOT ALTER.
142         enable = 0;
143
144         // Writing enable into the corresponding GPIO. DO NOT
               ALTER.
145         fp = fopen("/sys/class/gpio/gpio65/value", "w");
146         fprintf(fp, "%d", enable);
147         fclose(fp);
148
149         // Read analog inputs from BeagleBone Black. Already
               configured to convert the vaue read into the
               operation ranges defined on the system operation
               ranges section. Inputs may be included or removed,
               as long as the structure below is maintained.
150
151         // Read AIN0 and convert it to the input operation
               range.
152         a0 = io->Adc->Value[1];
153         a0 = mapRange(a0, 0, 65520, 0, 1800);
154         a0 = mapRange(a0, 0, 1800, IN_1_MIN, IN_1_MAX);
155         a0f = (float)a0/1000.00;
156         printf("%f\n", a0f);
157
158         // Read AIN1 and convert it to the input operation
               range.
159         a1 = io->Adc->Value[2];
160         a1 = mapRange(a1, 0, 65520, 0, 1800);
161         a1 = mapRange(a1, 0, 1800, IN_2_MIN, IN_2_MAX);
162         a1f = (float)a1/1000.00;
163
164         // Read AIN2 and convert it to the input operation
               range.
165         a2 = io->Adc->Value[3];
166         a2 = mapRange(a2, 0, 65520, 0, 1800);
167         a2 = mapRange(a2, 0, 1800, IN_3_MIN, IN_3_MAX);
168         a2f = (float)a2/1000.00;
169
```

```
170         // Read AIN3 and convert it to the input operation
               range.
171         a3 = io->Adc->Value[4];
172         a3 = mapRange(a3, 0, 65520, 0, 1800);
173         a3 = mapRange(a3, 0, 1800, IN_4_MIN, IN_4_MAX);
174         a3f = (float)a3/1000.00;
175
176         /* Insert your control equation here. Name your control
               variable u. */
177
178         // Convert control variable into a digital value. Alter
               only if you wish to change the number of bits to be
               used.
179         u_out = adc(u, OUT_1_MIN, OUT_1_MAX, 0, 511);
180
181         // Trasform control variable into binary. DO NOT ALTER.
182         int2bin(u_out,binString);
183
184         /* Associate each bit from the binary control variable
               to a GPIO.
185            Only alter if a different number of bits will be
                  used or if you
186            wish to assign different GPIOS. Check BeagleBone
                  Black header
187            in order to get the correct number for the desired
                  GPIOs. */
188
189         // Associate each bit with a char variable*/
190         bit_8 = binString[0];
191         bit_7 = binString[1];
192         bit_6 = binString[2];
193         bit_5 = binString[3];
194         bit_4 = binString[4];
195         bit_3 = binString[5];
196         bit_2 = binString[6];
197         bit_1 = binString[7];
198         bit_0 = binString[8];
199
200         // Turn each char variable into int */
```

```
201         gpio_11 = bit_8 - '0';
202         gpio_12 = bit_7 - '0';
203         gpio_7 = bit_6 - '0';
204         gpio_8 = bit_5 - '0';
205         gpio_9 = bit_4 - '0';
206         gpio_10 = bit_3 -'0';
207         gpio_15 = bit_2 - '0';
208         gpio_16 = bit_1 - '0';
209         gpio_17 = bit_0 - '0';
210
211         // Write each int variable to its correspondent GPIO.
212         fp = fopen("/sys/class/gpio/gpio27/value", "w");
213         fprintf(fp, "%d", gpio_17);
214         fclose(fp);
215
216         fp = fopen("/sys/class/gpio/gpio46/value", "w");
217         fprintf(fp, "%d", gpio_16);
218         fclose(fp);
219
220         fp = fopen("/sys/class/gpio/gpio47/value", "w");
221         fprintf(fp, "%d", gpio_15);
222         fclose(fp);
223
224         fp = fopen("/sys/class/gpio/gpio68/value", "w");
225         fprintf(fp, "%d", gpio_10);
226         fclose(fp);
227
228         fp = fopen("/sys/class/gpio/gpio69/value", "w");
229         fprintf(fp, "%d", gpio_9);
230         fclose(fp);
231
232         fp = fopen("/sys/class/gpio/gpio67/value", "w");
233         fprintf(fp, "%d", gpio_8);
234         fclose(fp);
235
236         fp = fopen("/sys/class/gpio/gpio66/value", "w");
237         fprintf(fp, "%d", gpio_7);
238         fclose(fp);
239
```

```
240         fp = fopen("/sys/class/gpio/gpio44/value", "w");
241         fprintf(fp, "%d", gpio_12);
242         fclose(fp);
243 
244         fp = fopen("/sys/class/gpio/gpio45/value", "w");
245         fprintf(fp, "%d", gpio_11);
246         fclose(fp);
247 
248         enable = 1;
249         fp = fopen("/sys/class/gpio/gpio65/value", "w");
250         fprintf(fp, "%d", enable);
251         fclose(fp);
252   }
253   }
254 
255     /* Destroy driver structure created on the beggining of
          main function. */
256     pruio_destroy(io);
257 
258     return 0;
259   }
```