LUÍS FELIPE DE BARROS E SILVA

Relatório Final do Trabalho de Conclusão de Curso: Comunicação em Tempo Real entre Plataforma Móvel e Simulador *Offshore* em Ambiente Imersivo

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do Título de Bacharel em Engenharia Mecatrônica

São Paulo
2012

LUÍS FELIPE DE BARROS E SILVA

Relatório Final do Trabalho de Conclusão de Curso: Comunicação em Tempo Real entre Plataforma Móvel e Simulador *Offshore* em Ambiente Imersivo

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do Título de Bacharel em Engenharia Mecatrônica

Área de concentração:
Engenharia Mecatrônica

Orientador:
Prof. Dr. Eduardo Aoun Tannuri

São Paulo
2012

Silva, Luís Felipe de Barros e
Comunicação m tempo real entre plataforma móvel e simulador offshore em ambiente imersivo / L.F.B. e Silva. -- São Paulo, 2012.
63 p.

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

1. Mecatrônica 2. Tempo-real 3. Simulação I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II. t.

Agradecimentos

Gostaria de agradecer em primeiro lugar à minha família pelo apoio e suporte que me deram durante todo o meu percurso universitário, em especial ao meu pai, Sérgio, minha madrasta, Cláudia e minhas queridas irmãs Vick e Carol.

Também gostaria agradecer aos meus amigos. Foram sempre importantíssimos nos momentos em que precisei de apoio e pensar em algo que não tivesse relação com os estudos. Em especial, meus amigos de Curitiba, do CAM, da natação, meus colegas de classe da Mecatrônica e aqueles que estiveram comigo em momentos importantes: Bruno, Gabriel, Iago e Nicola.

Um agradecimento especial à minha namorada, Francesca, que com muita paciência aguentou a minha ausência enquanto eu terminava meu percurso universitário.

Gostaria de agradecer à Escola Politécnica da USP e ao Politecnico di Torino que possibilitaram meu estudo de muita qualidade e gratuitamente.

Finalmente, gostaria de agradecer ao meu orientador, Eduardo Tannuri, que sempre com muita sabedoria me apoiou, incentivou e esteve disponível nos momentos em que precisei.

Claro, não poderia esquecer da minha mãe. Meu eterno exemplo, sempre presente no meu coração e mente e que me guia e fortalece em cada decisão e cada obstáculo na minha vida. Gostaria de compartilhar especialmente com ela este momento.

Luís Felipe de Barros e Silva

Resumo

Este trabalho de conclusão de curso foi desenvolvido no laboratório Tanque de Provas Numérico (TPN) da Escola Politécnica da USP. O objetivo foi criar um ambiente imersivo de simulação capaz de integrar duas funcionalidades já desenvolvidas no laboratório: a simulação de processos complexos de movimentação de corpos flutuantes sob diversas condições climáticas e a geração de uma visualização da movimentação destes corpos flutuantes simulados.

Para isto, foi adquirida uma plataforma do tipo *Hexapod* capaz de mover-se em seis graus de liberdade. O trabalho, então, era o de criar duas interfaces: uma com a plataforma e outro com o *software* desenvolvido e desenvolver um *software* intermediário para o tratamento dos dados recebidos do TPN para envio à plataforma.

A interface com o *software* TPN requeria uma comunicação com protocolo NMEA a uma frequência de 2Hz. Já a comunicação com a plataforma requeria comunicação *socket* usando protocolo UDP e uma frequência de 60Hz. A diferença nas frequências de envio e recebimento criou a necessidade de se desenvolver um algoritmo de extrapolação da série temporal recebido e nova discretização para envio à plataforma, com o menor prejuízo possível para a qualidade da série temporal.

Para este fim, foram testados diversos algoritmos de extrapolação, iniciando por algoritmos mais simples como o *Zero Order Hold* e o *First Order Hold* até algoritmos mais complexos, que calculavam o resíduo entre a posição esperada e a posição atual e reduziam este resíduo nos próximos passos de simulação além de integrar a aceleração e velocidades obtidas para extrapolação fiel ao movimento esperado na janela de tempo entre o recebimento de novos dados. A escolha do algoritmo foi baseada no tempo de cálculo de cada um e na capacidade de reproduzir o movimento real do corpo flutuante obtido através da simulação nos *clusters* do TPN.

Para o cálculo da janela temporal entre o envio dos dados, foi desenvolvida uma rotina de temporização especial. Esta rotina é capaz de manter a CPU ocupada e aguardar, com boa precisão, o momento para envio dos dados. Uma simples análise estatística foi realizada para mostrar que a rotina era eficiente e atingia os requisitos pré-estabelecidos, mesmo utilizando um sistema não desenvolvido especialmente para aplicações *real time*.

Finalmente, o envio de dados foi realizado utilizando o protocolo UDP, simples e leve, ideal para situações onde a temporização é muito importante e onde a perda de um pacote (pois o UDP não é 100% confiável) não afeta drasticamente a performance do sistema.

Palavras-chave: Mecatrônica, Tempo-Real, Simulação

Abstract

This work was developed in the headquarters of the Numerical Offshore Tank (TPN) laboratory located in the Universidade de Sao Paulo. The aim was creating an imersive simulation environment capable of integrating two main functionalities previously developed in the TPN: the simulation of complex offshore operations under several environmental conditions e the generation of a visualization of those simulations.

To achieve this objective, a Hexapod type platform capable of moving in six degrees of freedom was purchased. The work was then reduced to creating two interfaces: one with the TPN simulation software and the other with the platform. Besides, it was necessary to develop a software to treat data received from the TPN to send them correctly to the platform.

The TPN software interface was developed with a NMEA protocol communication at a 2Hz frequency. On the other hand, the communication with the Hexapod platform required a socket communication with UDP protocol at a frequency of 60Hz. This difference between the frequency in input and output made it necessary to develop an interpolation algorithm to treat the data received and discretize it at 60Hz to send to the platform, trying to maintain the quality of the time series.

In order to do that, several interpolation algorithms were tested, beggining with simple ones such as Zero Order Hold and First Order Hold until more complex ones, that would calculate the residuals between the real position of the platform and the desired position sent by the TPN software every time that a new package arrived and reduce this residuals until the next package would arrive. The choice of the bedt algorithm was based in the calculation time for each one of them and on the capability of reproducing the real movement of the simulated body even with low frequency of input communication.

To calculate the time window to send packages to the platform, it was developed a special purpose timer routine. This routine could maintain the CPU occupied e wait, with very good precision, the right moment to send the data. A simple statistical analysis was made to show that the developed code was efficient and satisfied the necessary pre-requisites, even working in a non-real-time operating system (Windows 7).

Finally, the code to send data using UDP protocol was developed. UDP is a simple and light protocol, well-suited for applications with strong time constraints and where eventual package losses will not affect drastically the performance of the system.

Keywords: Mechatronics, Real-Time, Simulation

Sumário

1 Introdução **1**

1.1 Motivação: O Tanque de Provas Numérico 1

1.2 A plataforma *Hexapod* 4

1.3 Comunicação com os equipamentos industriais e software 6

2 O projeto **9**

2.1 Especificações e requisitos de projeto 9

3 Metodologia **13**

3.1 Ambiente de simulação 13

3.2 Comunicação com o *software* TPN 14

3.3 Tratamento dos dados e minimização dos erros 15

3.4 Comunicação com a plataforma *Hexapod* 15

4 Bases teóricas **17**

4.1 Cinemática do mecanismo *Hexapod* 17

4.2 Dinâmica das embarcações 19

5 Comunicação com a plataforma *Hexapod* **21**

5.1 O protocolo UDP 21

5.2 O protocolo TCP/IP 22

5.3 Escolha do protocolo UDP 23

5.4 A biblioteca Winsock para *sockets* em Windows 24

5.5 Especificações do fabricante para envio de dados 25

5.6 A rotina de temporização para Windows 27

5.7 Validação da rotina de temporização 28

6 Comunicação com o *software* TPN **31**

6.1 O simulador do TPN 31

6.2 O protocolo NMEA 34

6.3 Programa de recebimento de dados do TPN usando NMEA 35

7 Tratamento dos dados recebidos para envio à plataforma 37

7.1 O problema das diferenças de frequências de recebimento e envio . . . 37
7.2 O Zero Order Hold 37
7.3 extrapolação dos dados recebidos em ingresso usando *First Order Hold* 40
7.4 O *First Order Hold* modificado 41
7.5 Algoritmo de integração das velocidades e amortização dos resíduos de posição 43
7.6 Algortimo de integração das acelerações recebidas 47
7.7 Utilizando o Fisrt Order Hold para as acelerações 48

8 Resultados com o algoritmo escolhido 55

8.1 Escolha do algoritmo apropriado 55
8.2 Análise dos algoritmos no espectro da frequência 56
8.3 Simulação do algoritmo escolhido na plataforma 57

9 Conclusão 61

10 Trabalhos futuros 63

Referências 64

Lista de Figuras

1.1 Sala de visualização do TPN 2
1.2 Simulação de plataformas TPN 3
1.3 Interface gráfica do software TPN 3
1.4 Plataforma tipo *Hexapod* (www.quanser.blogspot.com.br) 5
1.5 Esquema da solução proposta 6

3.1 Ambiente de simulação do Microsoft Visual Studio 14

4.1 Fotografia detalhando o mecanismo da plataforma utilizada 17
4.2 Desenho esquemático de uma plataforma *Hexapod* (retirado de [1]) . . 18
4.3 Representação esquemática das rotações de *roll, pitch* e *yaw* (Retirado de http://www.voodoo-world.cz/falcon/agf.html) 20

5.1 Visualização do *log* do Wireshark 29

6.1 Arquitetura do simulador TPN 32
6.2 Exemplo de visualização gerada através do *software* TPN 32
6.3 Outro exemplo de visualização 32
6.4 Joystick desenvolvido no TPN 33
6.5 Painel de controle desenvolvido no TPN 33
6.6 Simulação da ponte de comando desenvolvida 34

7.1 Pitch obtido com Zero Order Hold 38
7.2 Roll obtido com Zero Order Hold 39
7.3 Pitch obtido com First Order Hold 42
7.4 Roll obtido com First Order Hold 42
7.5 Pitch obtido com First Order Hold modificado 44
7.6 Roll obtido com First Order Hold modificado 44
7.7 Pitch obtido com algoritmo de integração de velocidades 46
7.8 Roll obtido com First Order Hold modificado 46
7.9 Pitch obtido com algoritmo de integração da aceleração 49
7.10 Roll obtido com algoritmo de integração da aceleração 49

7.11 Velocidade angular de Pitch obtido com algoritmo de integração da aceleração 50
7.12 Velocidade angular de Roll obtido com algoritmo de integração da aceleração 50
7.13 Pitch obtido com algoritmo de First Order Hold aplicado à aceleração . 52
7.14 Roll obtido com algoritmo de First Order Hold aplicado à aceleração . . 52
7.15 Velocidade angular de Pitch obtido com algoritmo de First Order Hold aplicado à aceleração 53
7.16 Velocidade angular de Roll obtido com algoritmo de First Order Hold aplicado à aceleração 53
7.17 Aceleração angular de Pitch obtido com algoritmo de First Order Hold aplicado à aceleração 54
7.18 Aceleração angular de Pitch obtido com algoritmo de First Order Hold aplicado à aceleração 54

8.1 Comparação entre série senoidal e extrapolação com Zero Order Hold . 57
8.2 Comparação entre extrapolação com Zero Order Hold e com First Order Hold 58
8.3 Comparação entre First Order Hold Modificado e Integração de Velocidades 58
8.4 Comparação entre integração de Acelerações e algoritmo final 59
8.5 Fotografia de instante de máxima rolagem da plataforma em simulação 59
8.6 Teste da plataforma com passageiro embarcado 60

Lista de Tabelas

5.1 Estrutura básica de pacote UDP 22
5.2 Estrutura básica de pacote UDP usando IPv4 23
5.3 Estrutura básica de pacote UDP usando IPv6 23
5.4 Especificações para envio de comprimento de atuadores à plataforma . 25
5.5 Especificações para envio de graus de liberdade à plataforma 26

8.1 Análise estatística da temporização dos algoritmos de extrapolação . . 56
8.2 Análise comparativa entre os algoritmos descritos 57

Capítulo 1

Introdução

1.1 Motivação: O Tanque de Provas Numérico

A construção do prédio do Tanque de Provas Numérico (TPN-USP), em parceria com a Petrobras, abriu portas para o desenvolvimento de diversos projetos tecnológicos na área de Engenharia Naval e Oceânica na Escola Politécnica. Diversos professores, alunos de graduação e pós-graduação e outros pesquisadores trabalham no laboratório em projetos de excelência na área de Engenharia Naval, rompendo barreiras de conhecimento e interagindo também com outras áreas da engenharia, como a Mecânica e a Mecatrônica.

O TPN é um laboratório pioneiro em hidrodinâmica aplicada, fruto de uma colaboração entre a Petrobras e as principais instituições de ensino do país, dentre elas a Escola Politécnica da USP. Seu principal objetivo é atuar como parceiro da indústria *offshore* e de petróleo e colaborar para a obtenção da auto-suficiência da produção de petróleo nacional como uma poderosa ferramenta para projeto e análise de sistemas flutuantes de produção de óleo e gás.

Um sistema *offshore* tipicamente consiste de um ou mais corpos flutuantes ancorados por diversas linhas de amarração e *risers* (que são dutos para extração de petróleo) sob as mais diversas condições ambientais.

Atuando de maneira complementar a um tanque de provas convencional, o TPN pode executar ensaios gerando basicamente as mesmas variáveis, mas em um modo mais veloz e econômico. Cada experimento produz dados não somente da série temporal de cada variável analisada, mas também uma estimativa de valores extremos, distribuição de probabilidades, análise do domínio espectral, entre outros.

O núcleo do TPN é um cluster de computadores (120 processadores atualmente, expansível até 400 processadores), que é hoje um dos maiores agrupamentos do Brasil para fins de pesquisa. Além do núcleo numérico o TPN tem conduzido pes-

quisa e desenvolvimento em computação gráfica sobre diversas plataformas, sendo esta agora a segunda grande linha de trabalho do laboratório.

Dentre os projetos desenvolvidos no laboratório está a criação de uma sala de realidade virtual para simulação dos movimentos semelhante aqueles experimentados a bordo de um navio, conforme mostra a figura 1.1. Esta sala é equipada com 44 poltronas confortáveis, um sistema de projeção de imagens em três dimensões e uma plataforma do tipo *Hexapod* com 4 poltronas montadas em sua superfície superior.

O sistema é chamado "4D", pois além da sensação de tridimensionalidade proporcionada pela projeção, o usuário ainda tem a sensação do movimento dos navios gerada através da plataforma *Hexapod*.

Figura 1.1: Sala de visualização do TPN

O sistema de projeção de imagens é baseado no principal *software* do laboratório, também chamado de TPN. A estrutura de simulação do TPN é baseada na discretização temporal do comportamento dinâmico do sistema flutuante, em particular os navios e plataformas, sob a ação de esforços externos decorrentes de condições ambientais, como corrente, vento e ondas, da dinâmica das linhas - amarração e *risers* - ou da interação entre corpos múltiplos, também conhecido como efeito sombra. O simulador é capaz de realizar uma análise acoplada de todos estes parâmetros, gerando elevada performance, agilidade e economia no processo de análise e desen-

volvimento de sistemas flutuantes. Um exemplo de visualização gráfica de plataforma e amarrações em simulações pode ser visto na figura 1.2.

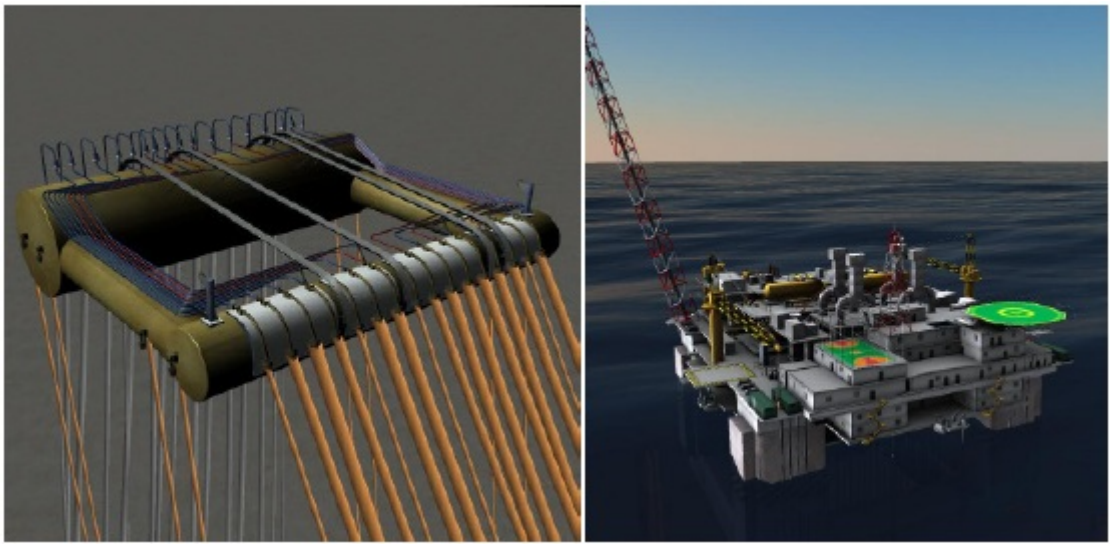

Figura 1.2: Simulação de plataformas TPN

Como resultado das simulações, o *software* produz as variáveis de posição, velocidade e aceleração dos seis graus de liberdade do corpo flutuante simulado, que são usadas como entrada para o sistema de geração e projeção de imagens, conforme pode ser observado na figura 1.3.

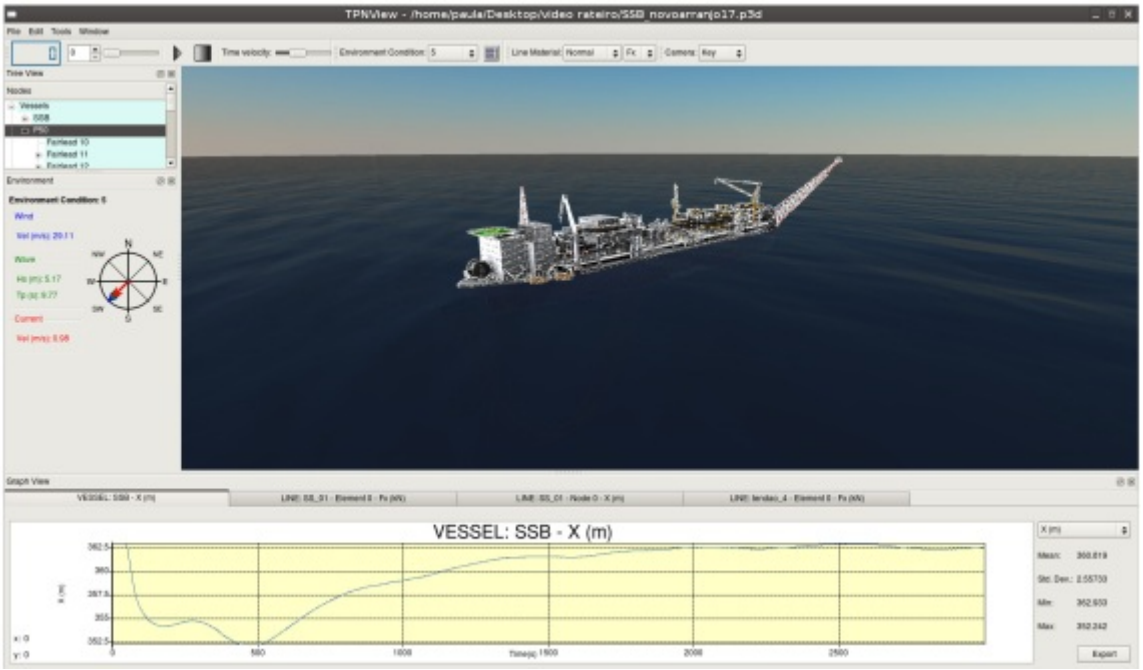

Figura 1.3: Interface gráfica do software TPN

Desta maneira, diversos experimentos podem ser conduzidos. Em especial a simulação de processos extremamente complexos da indústria petroleira, como o

procedimento de alívio das plataformas, onde o navio aliviador pode se aproximar muito da plataforma, com riscos de colisão ou rompimento dos cabos que retiram o petróleo, dependendo das condições climáticas, como velocidade dos ventos, valores de corrente e tamanho das ondas. Com esta visualização, se permite a observação em uma maneira muito mais próxima à real de uma operação do gênero.

Além disso, o conjunto plataforma/sistemas de projeção pode ser simulado para testes de conforto no interior das embarcações. É muito comum que pessoas se sintam mal ao fazer viagens utilizando navios, em especial devido aos movimentos de rotação periódica da embarcação. Para testar estes efeitos e tentar reduzir a sensação de mal-estar, simulações podem ser conduzidas utilizando-se da estrutura de simulação de movimento de embarcações.

Uma outra aplicação do sistema é o treinamento ou reciclagem dos responsáveis por conduzir os navios aliviadores da Petrobras, que trazem para o continente o petróleo produzido pelas plataformas marítimas. É possível, utilizando os simuladores já desenvolvidos no TPN, fazer ensaios com diversos navios plataformas da empresa, como a P-35 e a P-51, instaladas na bacia de Campos.

Através das variáveis de saída do programa, foi concebida a ideia de reproduzir os movimentos em escala em uma plataforma de dimensões médias (capaz de ser montada dentro de uma sala de aula), de maneira a simular o movimento do navio, com ênfase nos movimentos de rotação de *pitch*, isto é, rotação sobre o eixo transversal do corpo e *roll*, rotação sobre o eixo longitudinal do corpo, aumentando a percepção de realidade virtual em um ambiente imersivo.

Este trabalho de conclusão de curso é baseado no desenvolvimento de um *software* que atue no recebimento e tratamento de dados do software TPN, isto é, as informações sobre posição, velocidade e aceleração das variáveis de *roll* e *pitch*, utilizando o protocolo NMEA; no tratamento destes dados utilizando algoritmos para extrapolação e garantia de continuidade nos movimentos, através da linguagem C++; na programação e configuração da plataforma descrita previamente e no envio dos dados tratados, utilizando comunicação *socket* e protocolo UDP.

1.2 A plataforma *Hexapod*

A plataforma do tipo *Hexapod* é um equipamento de seis graus de liberdade. A plataforma possui seis pernas que podem alterar seus comprimentos de acordo com os comandos enviados a cada um dos seus atuadores (um para cada perna), um exemplo deste tipo de plataforma pode ser visto na figura 1.4.

Esta plataforma é extremamente versátil e permite a reprodução de movimen-

Figura 1.4: Plataforma tipo *Hexapod* (www.quanser.blogspot.com.br)

tos em seis graus de liberdade, isto é, permite o deslocamento nos eixos x, y e z, além de permitir as rotações de *Roll* (rotação ao redor do eixo longitudinal do corpo), *Pitch* (rotação ao redor do eixo transversal do corpo) e *Yaw* (rotação em torno do eixo vertical passando pelo corpo).

Através de uma matriz de transformação de coordenadas, a movimentação nos graus de liberdade descritos acima pode ser obtida através de um comando adequado enviado aos atuadores da plataforma. Dois exemplos simples são:

- Translação no eixo z:

  Para efetuar uma translação no eixo z, basta enviar um comando aos atuadores para aumentar o comprimento l_i, com $i = 1, 2, ..., 6$ para cada uma das pernas do mecanismo de um valor d. É fácil notar que tal comando criará um movimento no eixo z do mecanismo, também conhecido como movimento de *Heave*.

- Rotação de *Yaw*:

  Para obter uma rotação de *Yaw*, isto é, uma rotação ao redor do eixo vertical que passa pela plataforma, basta enviar um comando de extensão de um valor d aos atuadores que comandam o comprimento das pernas alternadas, alterando o valor dos comprimentos l_i, com $i = 1, 3, 5$ e, ao mesmo tempo, enviar um comando de retração às demais pernas do mesmo valor d. Nota-se também, que estes comandos simultâneos geram uma rotação ao redor do eixo vertical da plataforma, ou rotação de *Yaw*.

Com a devida notação matricial, que será exposta mais adiante, pode-se calcular o efeito de qualquer alteração de comprimento d_i na perna l_i no sistema de referência fixo. Da mesma forma, para qualquer movimento de translação e rotação desejados, pode-se calcular qual o comprimento de cada uma das pernas do mecanismo necessário para possibilitar tal movimentação.

1.3 Comunicação com os equipamentos industriais e software

O presente trabalho de conclusão de curso prevê a integração entre dois atores: o software TPN e uma plataforma de simulação de movimentos do tipo *Hexapod*. Para este fim, é necessário conhecer a fundo como ocorre a comunicação com cada um desses dois elementos. Um esquema da solução proposta para este escopo é mostrada na figura 1.5.

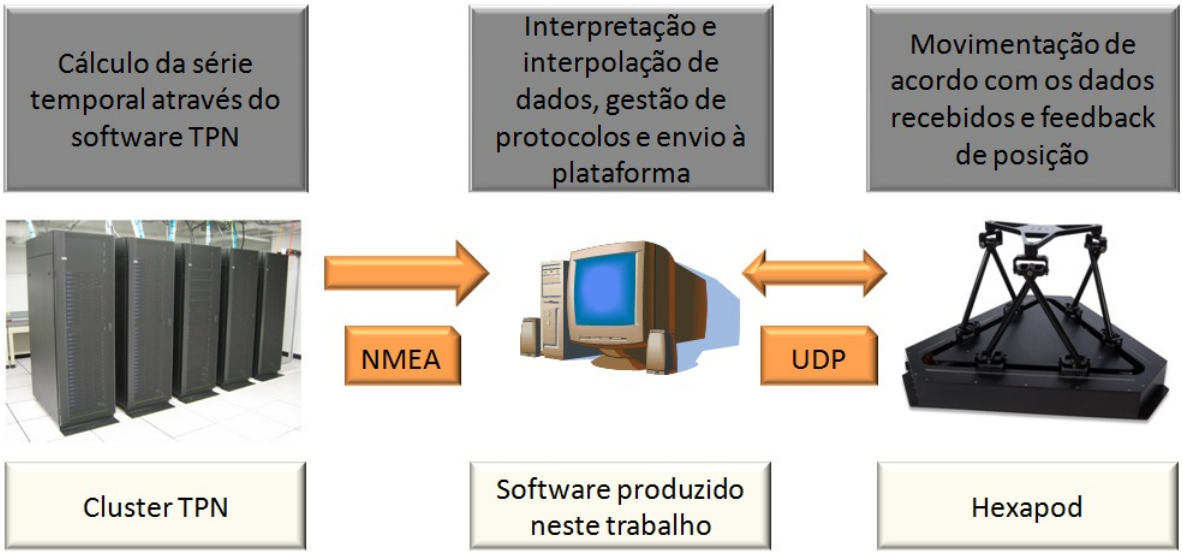

Figura 1.5: Esquema da solução proposta

No caso do *software* TPN, o programa exporta dados em uma determinada frequência, usando protocolo NMEA, típico de aplicações navais, como ecos, sonares e o GPS. Atualmente, o padrão utilizado é o NMEA 0183 que tem a tendência de ser substituído pelo NMEA 2000 nos próximos anos.

Já para a comunicação com a plataforma *Hexapod* o protocolo utilizado é do tipo UDP, usando as especificações do fabricante da plataforma, a *Moog, Inc.*

O *software* desenvolvido neste trabalho é capaz de obter os dados do TPN, tratá-los e interpolá-los afim de enviar dados à plataforma na frequência especificada pela fabricante com o intuito de manter uma suavidade no movimento e maximizar a

sensação de realidade virtual em um ambiente imersivo. Além disso, os erros entre posição desejada e posição medida devem ser minimizados a cada instante.

Por fim, o algoritmo desenvolvido neste trabalho visa minimizar as descontinuidades obtidas quando um novo dado de posição do TPN é recebido, distribuindo o erro entre posição atual e nova posição recebida nos próximos envios de dados até o recebimento de uma nova posição do TPN.

Capítulo 2

O projeto

2.1 Especificações e requisitos de projeto

Como um projeto de engenharia, o presente trabalho possui diversas restrições, limitações e especificações. Neste caso, as especificações dizem respeito a:

- **Comunicação com o *software* TPN sem alteração das especificações pré-existentes para este projeto**

  Esta especificação implica que o projeto desenvolvido não pode alterar as especificações do TPN. Isso inclui, especialmente, a plataforma de desenvolvimento (uso de Windows), temporização de recebimento de informações (pacotes recebidos com frequência de 2Hz) e código com capacidade de integração ao *software* TPN.

- **Temporização para envio de pacotes à plataforma *Hexapod***

  O PLC presente na plataforma de 6 graus de liberdade requer envio de pacotes a uma frequência de 60Hz. Uma das grandes restrições deste projeto, então, é o tratamento dos sinais e envio das informações dentro da janela de tempo pré-estabelecida, isto é, um período máximo de aproximadamente 16,6 ms. Todos os cálculos presentes na rotina de comunicação, assim como as próprias funções de envio e recebimento de dados da plataforma devem ser executados dentro desta janela temporal.

- **Protocolo de comunicação com a plataforma**

  Ainda como imposição devido ao PLC presente na plataforma *Hexapod*, a comunicação entre esta e o *software* desenvolvido neste projeto deve ser do tipo UDP (*User Datagram Protocol*). Apesar de o TCP (*Transmission Control Protocol)* ser uma alternativa viável, mais segura e confiável (não obstante seja mais lento).

Devido a este requisito, eventuais mecanismo de verificação da integridade de dados, confirmação de recebimento ordenado e correto dos pacotes e outros mecanismos devem ser desenvolvidos utilizando o protocolo UDP.

- **Máxima sensação de imersão na reprodução dos movimentos**

  Este requisito de projeto implica que os movimentos que geram a sensação de imersão devem ser reproduzidos integralmente. Neste caso, os movimentos de deriva, isto é, aqueles movimentos mais lentos que não geram acelerações significativas, como os movimentos de translação no plano em um navio (em especial nos eixos x e y), não precisam ser reproduzidos. Por outro lado, os movimentos de rotação ao longo do eixo longitudinal do corpo (*Roll*) e de rotação ao longo do eixo transversal do corpo (*Pitch*), devem ser reproduzidos integralmente, por serem essenciais para a simulação da sensação de estar a bordo de um navio.

- **Continuidade dos movimentos**

  A diferença entre a frequência de recebimento dos dados e envio dos mesmos através do *software* desenvolvido neste trabalho traz, intrinsecamente, uma necessidade de extrapolação dos dados recebidos para envio à plataforma. Esta extrapolação, contudo, deve estar de acordo com a realidade da movimentação de um navio. Como as forças que atuam no navio, através da lei da terceira lei de Newton ($F = ma$), produzem uma aceleração no corpo, e a primeira derivada dos graus de liberdade é a integração destas acelerações, isto implica que as velocidades são contínuas. Da mesma forma, as posições dos graus de liberdade também devem ser contínuas. Por essas razões, na reprodução dos movimentos, isto é, na escolha dos algoritmos de extrapolação que geram as posições enviadas à plataforma, requer-se que ao menos a série temporal de posição enviada à plataforma deva ser contínua.

- **Movimentação em tempo real**

  O movimento produzido pela plataforma deverá reproduzir a sequência de dados recebidos através do *software* TPN em tempo real. Isto significa que o tratamento dos dados deve ocorrer *online* e os algoritmos de controle devem usar somente os dados já recebidos através da comunicação com o TPN. Por determinação das especificações de projeto, o atraso entre a reprodução do movimento na plataforma e o recebimento dos dados deve ser mínimo, não permitindo a implementação de algoritmos de controle que necessitem do conhecimento de toda a série temporal para efetuar os cálculos (implementação típica de algoritmos de

controle *offline*). Essa especificação restringe significativamente a quantidade de algoritmos a serem analisados.

O objetivo final deste último requisito é o de a projeção tridimensional ser sempre sincronizada ao movimento gerado na plataforma de seis graus de liberdade. Eventuais atrasos gerariam um efeito de dessincronização que é altamente indesejável para a qualidade do trabalho realizado.

Capítulo 3

Metodologia

3.1 Ambiente de simulação

O ambiente de desenvolvimento utilizado neste trabalho é o Microsoft Visual Studio [2] e o sistema operacional utilizado é Windows. A escolha de tais plataformas é baseada no desenvolvimento prévio dos programas do TPN, que se utilizam de Windows, evitando o conflito de sistemas operacionais e permitindo a portabilidade como um todo do *software* desenvolvido neste trabalho junto com o TPN. Além disso, a Microsoft fornece ótimas bibliotecas para a gestão de comunicação UDP, que são amplamente utilizadas neste trabalho.

O Visual Studio é um ótimo ambiente para o *debug* do programa desenvolvido. Sua interface é amigável ao usuário e, ao mesmo tempo, permite o desenvolvimento de programas complexos. Além disso, oferece completo suporte à linguagem C++, utilizada neste trabalho. Estas são as principais razões para a escolha desta IDE (*Integrated Development Environment*) para este projeto. O ambiente de trabalho do Visual Studio pode ser visto na figura 3.1.

O Windows é sabidamente um sistema operacional que não foi desenvolvidos para aplicações *Hard Real Time* (no qual o desrespeito às imposições temporais pode prejudicar seriamente o funcionamento do sistema). Para este escopo, existem sistemas operacionais específicos, como o VxWorks. No entanto, para esta aplicação onde a perda eventual de uma *deadline* é aceitável e o requisito temporal não é extremamente justo (*Soft Real Time*), a utilização de Windows com o software sendo executado em alta prioridade - inclusive existe a opção de rodar um aplicativo em tempo real - e uma rotina de medição de tempo de alta precisão são suficientes para atingir os objetivos esperados.

O principal fator, no entanto, para a tomada da decisão a respeito do sistema operacional, como explicado anteriormente, é a integração com o desenvolvimento

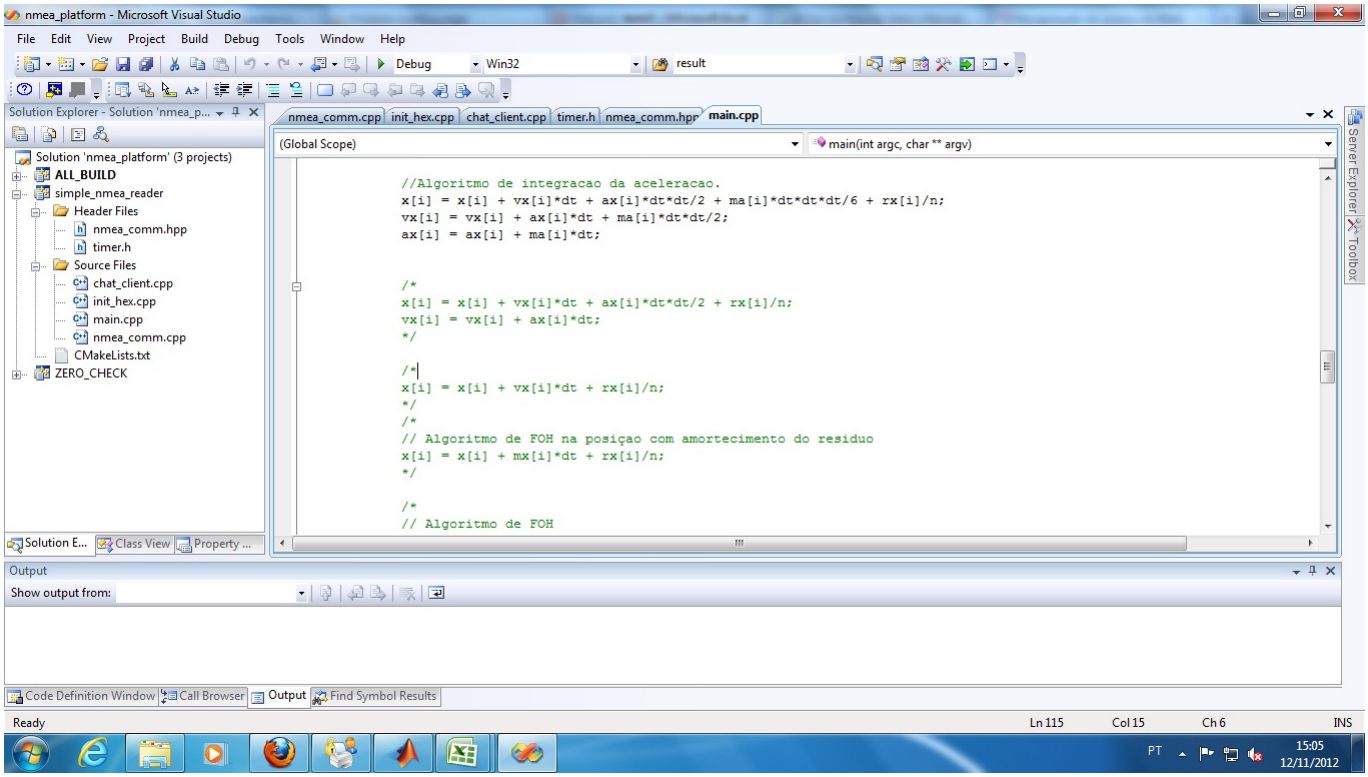

Figura 3.1: Ambiente de simulação do Microsoft Visual Studio

anterior do software *TPN*.

3.2 Comunicação com o *software* TPN

Para a comunicação com o *software* de simulação de embarcações TPN, a metodologia é baseada na análise do programa, em especial das variáveis de saída do programa. As variáveis de interesse para este trabalho são os graus de liberdade de rotação, em especial os valores de *Roll* e *Pitch*, assim como as respectivas derivadas primeira e segunda.

As saídas do programa podem ser acessadas lendo-se as mensagem produzidas pelo TPN, baseadas no protocolo NMEA [3].

O ambiente de simulação para o programa desenvolvido será o Microsoft Visual Studio [2]. Através do Visual Studio, poderá ser feito o *debug* das informações recebidas para conferência com os dados esperados. Para o *debug*, será criado um *struct* com os valores obtidos e que vem atualizado conforme os dados são recebidos do TPN.

Após esta fase, os dados estarão prontos para serem tratados dentro do *software* desenvolvido neste trabalho.

3.3 Tratamento dos dados e minimização dos erros

Para o tratamento e minimização de dados, o livro usado foi o [4]. Técnicas de controle digital, amostragem, algoritmos de *Zero Order Hold* e *First Order Hold* foram consultados nesse livro para desenvolvimento da técnica de extrapolação de dados recebidos e um algoritmo modificado, descrito mais adiante neste trabalho, foi utilizado para o envio de dados à plataforma *Hexapod*.

3.4 Comunicação com a plataforma *Hexapod*

Para a comunicação com o a plataforma *Hexapod* o protocolo utilizado é o UDP (User Datagram Protocol), que será descrito em detalhes posteriormente. É um protoclo mais rápido e leve que o TCP e pode ser usado em aplicações onde a temporização é essencial e a perda de um pacote não diminui drasticamente a *performance* do sistema.

Como referência para a comunicação utilizando UDP, foi usado o livro [5].

A plataforma de desenvolvimento é o Microsoft Visual Studio [2] do *software* deste trabalho, em especial por ser a plataforma de desenvolvimento padrão no laboratório TPN e, por esse motivo, facilitar eventuais *upgrades*, atualizações e modificações posteriores no código e no programa por parte de estudantes que no futuro venham a continuar este trabalho.

Os dados devem ser enviados de acordo com as especificações do fabricante da plataforma *Hexapod*, a empresa *Moog, Inc.* Estas especificações incluem formato dos dados, configurações iniciais, precisão e frequência de envio. Estas especificações também serão discutidas nos próximos capítulos deste trabalho. Caso estas especificações não sejam respeitadas, a plataforma não se movimenta de acordo com o previsto e a sensação de imersão no ambiente do navio pode ser reduzida ou até mesmo anulada.

Sintetizando, existem três requisitos principais para a comunicação com a plataforma. A primeira diz respeito ao formato dos dados, que devem respeitar um padrão pré-estabelecido para serem interpretados corretamente pelo PLC instalado na plataforma. O segundo diz respeito à forma de comunicação, utilizando protocolo UDP. E, finalmente, o terceiro requisito que especifica a temporização da comunicação, estabelecendo uma frequência específica de 60Hz que caso não venha respeitada, prejudica a continuidade do movimento.

Capítulo 4

Bases teóricas

4.1 Cinemática do mecanismo *Hexapod*

Este trabalho de conclusão de curso baseia-se na comunicação com um equipamento industrial do tipo plataforma *Hexapod*. Uma visão mais detalhada do mecanismo pode ser observada na figura 4.1 e um desenho esquemático para gerar as equações de cinemática do mecanismo em 4.2, conforme [1].

Figura 4.1: Fotografia detalhando o mecanismo da plataforma utilizada

A partir do desenho esquemático, é possível fazer a transformação de coordenadas saindo da base de comprimento de cada uma das seis pernas da plataforma *Hexapod* para os graus de liberdade de rotação (*Roll, Pitch e Yaw*), que atendem às

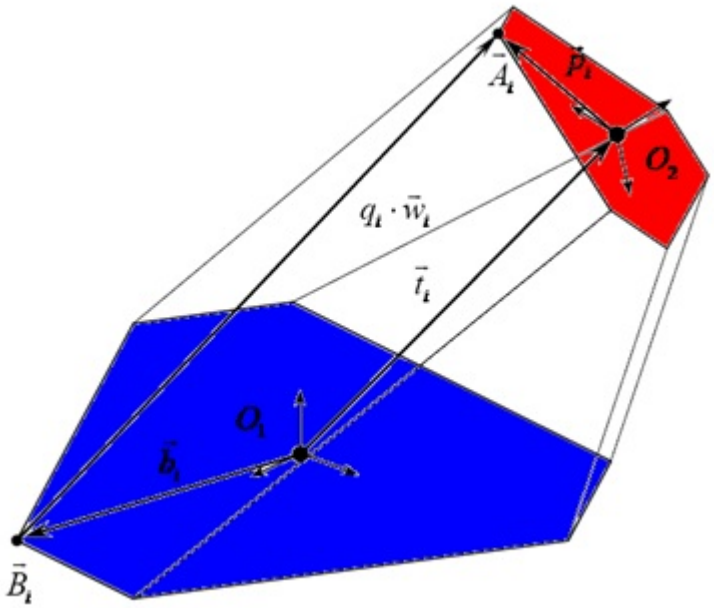

Figura 4.2: Desenho esquemático de uma plataforma *Hexapod* (retirado de [1])

necessidades deste trabalho. Os vetores $\vec{A_i}$ que unem a base superior do mecanismo à superior - representando as pernas do mecanismo - podem ser descritos da seguinte maneira, conforme [6]:

Para $i = 1, 2, ..., 6$:

$$\vec{A_i} = \vec{t_i} + [T] * \vec{p_i} \tag{4.1}$$

onde:

$\vec{t_i} = O_2 - O_1$;

O_2 é a origem do sistema móvel de coordenadas;

O_1 é a origem do sistema fixo de coordenadas.

O vetor $\vec{t_i}$ é a translação entre os sistemas de coordenada, e representa as variáveis de deslocamento nos eixos x, y e z e que possuem analogia direta com os movimentos de *Heave, Surge* e *Sway* que se deseja reproduzir.

Já a segunda parcela do lado direito da equação 4.1 representa a rotação do mecanismo através da matriz de rotação [T]. Esta matriz pode ser descrita através dos

ângulos de rotação de *roll, pitch* e *yaw*, conforme a equação:

$$[T] = \begin{bmatrix} cos\Theta cos\psi & sin\varphi sin\Theta cos\psi - cos\varphi sin\Theta & cos\psi + sin\varphi sin\psi \\ cos\Theta sin\varphi & sin\Theta sin\varphi sin\psi + cos\Theta cos\psi & cos\varphi sin\Theta sin\psi - sin\varphi cos\psi \\ -sin\Theta & sin\varphi cos\Theta & cos\varphi cos\Theta \end{bmatrix} \quad (4.2)$$

Essa matriz é obtida simplesmente projetando os eixos rotacionados sobre o eixo original fixo.

Já os vetores $\vec{p_i}$ representam as coordenadas das juntas das pernas com a base superior do mecanismo em relação à base de coordenadas móvel, que é aquela cuja origem se encontra no centro da plataforma superior.

As rotações de *roll, pitch* e *yaw* são definidas como:

1. Rotação de um ângulo ψ ao redor do eixo z fixo é definido como ângulo de *yaw*.
2. Rotação de um ângulo Θ ai redor do eixo y móvel, obtido após a rotação do passo anterior, é definido como ângulo de *pitch*.
3. Finalmente, uma rotação de um ângulo φ ao redor do novo eixo x, obtido após o passo anterior, é definido como ângulo de *roll*.

Ou seja, *roll* é a rotação em torno do eixo longitudinal do corpo, *pitch* é o ângulo que mede a rotação em torno do eixo transversal do corpo e o *yaw* é o ângulo de rotação em torno a eixo fixo na terra que aponta para cima (eixo Z).

A figura 4.3 mostra em maneira esquemática as rotações do tipo *roll, pitch* e *yaw*.

4.2 Dinâmica das embarcações

Em um primeiro momento, as séries temporais utilizadas em testes preliminares foram senoidais do tipo:

$$x_i(t) = Asen\frac{2 * \pi}{T}t \quad (4.3)$$

onde:

A é a amplitude da onda senoidal;

T é o período de oscilação da onda senoidal;

t é o tempo;

e x_i são os graus de liberdade do navio.

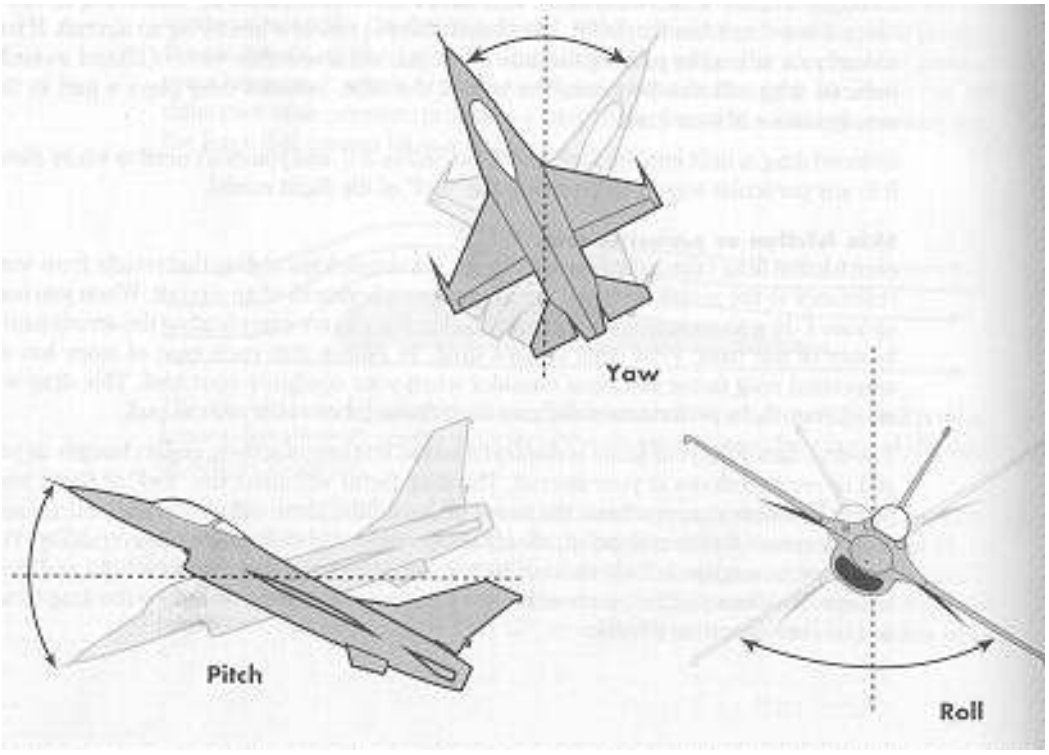

Figura 4.3: Representação esquemática das rotações de *roll, pitch* e *yaw* (Retirado de http://www.voodoo-world.cz/falcon/agf.html)

Uma descrição mais completa da movimentação de embarcações é discutida em [7].

O programa TPN envia as saídas para o *software* desenvolvido neste trabalho com uma frequência de amostragem de 2Hz, o que implica um período de amostragem $T_a = 0,5s$. Por outro lado, devido às especificações do fabricante da plataforma *Hexapod*, os dados devem ser enviados ao equipamento com frequência de 60Hz, o que implica período de amostragem de $T_a \cong 0,0166s$.

Este fato cria a necessidade um tratamento dos sinais recebidos e uma nova amostragem com a frequência requerida pela plataforma. Além disso, para adequação do movimento à realidade do movimento de uma embarcação, é necessário um algoritmo para controlar as descontinuidades na série temporal enviada ao *Hexapod*. Este algoritmo será descrido posteriormente neste relatório.

Capítulo 5

Comunicação com a plataforma *Hexapod*

5.1 O protocolo UDP

O UDP (User Datagram Protocol), junto com o TCP (Transmission Control Protocol), são os protocolos de comunicação usados tipicamente em conjunto com o IP (Internet Protocol), formando a série de protocolos utilizados em conexões através da Internet [5].

Ao contrário do TCP, o UDP usa um mecanismo de transmissão extremamente simplificado, com o intuito de tornar os pacotes UDP - também chamados datagramas - leves e simples. O UDP não garante que a ordem dos pacotes enviados esteja correta, que eles não sejam duplicados ou mesmo que eles sejam simplesmente entregues. Propõe apenas soma de verificação para algum controle da integridade dos pacotes e uma porta de endereçamento.

As principais características do UDP são:

- Não necessita de conexão;
- Orientado a mensagens;
- Requer menos recursos do sistema operativo;
- Simples e leve;
- Sem mecanismos de garantia de entrega, de ordem ou de duplicação dos pacotes ou mecanismos de controle de congestão na rede.

A estrutura dos pacotes UDP é a seguinte:

offset(bits)	0 - 15	16-31
0	Source Port Number	Destination Port Number
32	Length	Checksum
64+	Data	

Tabela 5.1: Estrutura básica de pacote UDP

5.2 O protocolo TCP/IP

O TCP é um protocolo de comunicação mais confiável que o UDP. Isso porque possui diversos mecanismos que conferem o pacote de dados enviados.

As principais características do TCP são:

- Orientado à conexão
- Ponto a ponto
- Confiável
- Full duplex
- Garantia de entrega em ordem dos pacotes
- Mecanismo de controle de fluxo de transmissão

Atualmente, o protocolo TCP pode ser estruturado de duas maneiras diversas: o IPv4 e o IPv6. A origem do IPv6 é devida ao exaurimento iminente dos endereços de IP da versão 4. A estrutura dos pacotes é mostrada a seguir:

- **IPv4**

offset(bits)	0 - 7	8-15	16-23	24-31
0	Source Address			
32	Destination Address			
64	Zeros	Protocol	UDP length	
96	Source Port Number		Destination Port Number	
128	Length		Checksum	
160+	Data			

Tabela 5.2: Estrutura básica de pacote UDP usando IPv4

- **IPv6**

offset(bits)	0 - 7	8-15	16-23	24-31
0	Source Address			
128	Destination Address			
256	UDP length			
288	Zeros			Next Header
320	Source Port Number		Destination Port Number	
352	Length		Checksum	
384+	Data			

Tabela 5.3: Estrutura básica de pacote UDP usando IPv6

5.3 Escolha do protocolo UDP

O UDP é usado em casos nos quais a velocidade de transmissão é um fator primordial e a falha na entrega de um pacote não acarreta uma falha grave no sistema. Muitas vezes, em aplicações em tempo real satisfazem esses requisitos e portanto utilizam o UDP ao invés do TCP.

O *software* desenvolvido neste trabalho é um exemplo de aplicação onde a temporização do envio dos dados na frequência de projeto é mais importante do que a falha na entrega de alguns pacotes esparsos. O PLC da plataforma que recebe os

pacotes foi programado de maneira que o movimento é tanto suave quanto a temporização seja respeitada.

Tendo isso em visto, a escolha óbvia diante dos requisitos e restrições de projeto foi o protocolo UDP.

5.4 A biblioteca Winsock para *sockets* em Windows

A bliblioteca utilizada para comunicação com a plataforma *Hexapod* utilizando UDP será Winsock, cuja documentação está disponível em [8]. Esta biblioteca foi criada para a utilização de sockets TCP/IP e UDP/IP em Windows. O Winsock ou Windows Sockets API é uma especificação técnica que define e interage entre o sistema operativo Windows e os serviços de rede.

Utilizando o Winsock, pode-se utilizar funções simples para a comunicação UDP, como send(), sendto(), recv() e recvfrom(). A comunicação criada neste trabalho é baseado nestas funções. Parte do código para definir a estrutura de dados e enviá-los à plataforma é mostrado abaixo:

```
1
// Inicializar Winsock para comunicacao UDP
3
WSADATA info;
5 WSAStartup ( MAKEWORD(2,2), &info );

7 sock=socket(AF_INET,SOCK_DGRAM,0);

9 // Iniciar a comunicacao NMEA e criar a estrutura de dados para
     armazenar as informacoes
  nmea_comm comm("127.0.0.1", 54325, 8);
11
  // Rotina de tratamento e interpolacao dos dados
13 while (true){
  ...
15 // Saida da rotina: valores de xroll e xpitch

17
  val0 = htonl(0x00000082);
19 val1 = htonl ( *((unsigned int *) &xroll) );
  val2 = htonl ( *((unsigned int *) &xpitch) );
21
  memcpy ( &data[0], &val0, sizeof(val0) );
23 memcpy ( &data[1], &val1, sizeof(val1) );
  memcpy ( &data[2], &val2, sizeof(val2) );
25
  // Garante loop com passo de 0.0166 segundos (60Hz)
27 waitbusy(dt, basetsclast, calfreq);
  basetsclast=gettsc();
29 status = send ( sock, (char *)data, 8*sizeof(float), 0 );
  }
31
  closesocket(socket);
33 WSACleanup();
  return 0;
```

5.5 Especificações do fabricante para envio de dados

Os dados que devem ser enviados à plataforma *Hexapod* devem seguir as especificações contidas no manual do fabricante, *Moog, Inc.* A empresa especifica, por exemplo, o formato dos dados de posição enviados a cada instante à plataforma, conforme a tabela 5.4 caso se opte por enviar o comprimento de cada uma das pernas do mecanismo. Além disso, existe uma frequência específica para envio de dados, que neste caso é de 60Hz, ou seja, um conjunto de dados a cada 0,0166s aproximadamente.

Palavra no.	Dados	Descrição	Unidade	Formato
0	MCW	Palavra de comando	-	32 bit unsigned int
1	length_ a	Comprimento perna A	m	32 bit float
2	length_ b	Comprimento perna B	m	32 bit float
3	length_ c	Comprimento perna C	m	32 bit float
4	length_ d	Comprimento perna D	m	32 bit float
5	length_ e	Comprimento perna E	m	32 bit float
6	length_ f	Comprimento perna F	m	32 bit float
7	-	Em branco	-	-

Tabela 5.4: Especificações para envio de comprimento de atuadores à plataforma

Neste trabalho, optou-se por enviar diretamente os graus de liberdade (DOF) à plataforma. Desta maneira, o padrão utilizado para os dados a serem enviados seguem o padrão da tabela 5.5.

A fim de iniciar a comunicação com a plataforma *Hexapod*, o fabricante requer uma série de comandos iniciais de configuração e definição de parâmetros. A rotina init_ hex(int sock) foi desenvolvida para este fim. O primeiro parâmetro é relacionado à configuração UDP com a plataforma, que é realizada através do IPv4 10.10.10.10 na porta 991. O código para esta configuração é mostrado abaixo:

Palavra no.	Dados	Descrição	Unidade	Formato
0	MCW	Palavra de comando	-	32 bit unsigned int
1	value_ roll	Roll	rad	32 bit float
2	value_ pitch	Pitch	rad	32 bit float
3	value_ heave	Heave	m	32 bit float
4	value_ yaw	Yaw	rad	32 bit float
5	value_ surge	Surge	m	32 bit float
6	value_ sway	Sway	m	32 bit float
7	-	Em branco	-	-

Tabela 5.5: Especificações para envio de graus de liberdade à plataforma

```
int init_hex (int sock){

unsigned int data[8];
int status;
struct sockaddr_in target;

memset((char *)&target,0,sizeof(target));
target.sin_family=AF_INET;
target.sin_addr.s_addr=inet_addr("10.10.10.10");// endereco IP de
    destino
target.sin_port=htons(991); // porta de destino
```

Após isso, a plataforma requere uma série de comandos para sinalizar o início de uma simulação. São eles:

- **Reset:** Usado para cancelar as configurações antigas e utilizar as configurações atuais.
- **Mode:** Configurar o modo de envio de dados. No caso deste trabalho, o modo escolhido é o DOF (graus de liberdade).
- **Engage:** Prepara a plataforma para o início de envio dos dados. A plataforma ativa um *timer* próprio e espera o envio de dados por parte do *software*. Caso os dados não sejam enviados, um erro de *timeout* ocorre e a plataforma é desligada.

A seguir, segue o código com a continuação da rotina init_ hex() com estas configurações:

```
printf("Resetando o Hexapod \n");
data[0] = htonl(0x000000A0);     //RESET
status = send ( sock, (char *)data, 8*sizeof(float), 0 );
fprintf( stderr, "Status = %d (%s)\n", status, strerror (status) );
Sleep(5000);

printf("Configuracoes iniciais \n");
data[0] = htonl(0x000000AA);     //Degrees of Freedom Mode (DOF)
status = send ( sock, (char *)data, 8*sizeof(float), 0 );
fprintf( stderr, "Status = %d (%s)\n", status, strerror (status) );
Sleep(1000);


printf("Preparar para a simulacao...\n");
data[0] = htonl(0x000000B4);     //ENGAGE
status = send ( sock, (char *)data, 8*sizeof(float), 0 );
fprintf( stderr, "Status = %d (%s)\n", status, strerror (status) );
Sleep(7800);

return status;
}
```

5.6 A rotina de temporização para Windows

Para garantir a temporização em Windows, foi desenvolvida uma rotina especial que garante, com a precisão requerida pela comunicação, a frequência de envio de dados necessária para a comunicação (60Hz).

A rotina tem como principal método *gettsc()*, escrito em linguagem *assembly*. Esta parte lê o registrador "RDTSC"que retorna o "*Time Stamp Counter*"presente nos processadores Intel (a partir da geração *Pentium*). Este método é utilizado para contar com precisão os ciclos de *clock* do processador.

Além deste método, foram desenvolvidos os métodos *timelapse()* e *waitbusy()*. O primeiro retorna o tempo que se passou desde a última volta que o método *gettsc()* foi chamado e o segundo mantém a CPU ocupada durante um período de tempo pré-definido.

Por fim, o método *HRPCFreqCal()* calcula a frequência do processador para transformar os ciclos de *clock* em períodos de tempo.

O código com todos os métodos explicados acima é transcrito a seguir:

```
#include <windows.h>
#include <stdio.h>

__forceinline ULONGLONG gettsc(void) {
 ULONG hi,lo;
 LARGE_INTEGER r;
 __asm {
     rdtsc
     mov hi,edx
     mov lo,eax
 }
 r.HighPart = hi;
 r.LowPart = lo;
 return r.QuadPart;
}

__forceinline double timelapse(ULONGLONG basetsc, double calfreq) {
   ULONGLONG currenttsc;
   currenttsc = gettsc();
   return ( currenttsc-basetsc )/calfreq;
}

__forceinline double waitbusy(double finaltime, ULONGLONG basetsc,
   double calfreq) {
   double remainingtime;
   remainingtime = finaltime - timelapse(basetsc,calfreq);
   do {
       remainingtime = finaltime - timelapse(basetsc,calfreq);
   } while (remainingtime > 0);
   return remainingtime;
}

double HRPCFreqCal(double CalInterval)
{
  LARGE_INTEGER Freq_int64;
  ULONGLONG basetsc,curtsc;
  FILETIME baseft,curft;
  ULONGLONG basest, curst;
  ULONGLONG deltsc, delst;
  Sleep(5);
  basetsc = gettsc();
  GetSystemTimeAsFileTime(&baseft);
  Sleep(( (DWORD) CalInterval )*1e3);
  curtsc = gettsc();
  GetSystemTimeAsFileTime(&curft);
  basest = ((ULONGLONG) baseft.dwHighDateTime)<<32 | ((ULONGLONG) baseft
     .dwLowDateTime);
  curst = ((ULONGLONG) curft.dwHighDateTime)<<32 | ((ULONGLONG) curft.
     dwLowDateTime);
  deltsc = curtsc - basetsc;
  delst = curst - basest;
  return ((double) deltsc) / ((( (double) delst)) / 1e7);
}
```

5.7 Validação da rotina de temporização

Para verificar se realmente a rotina de temporização estava garantindo a temporização adequada, foi utilizado o software *Wireshark* [9]. O *software Wireshark* é

um programa gratuito e de código aberto para análise do pacotes enviados através da rede.

Utilizando este programa, é possível analisar todos os pacotes distribuídos na rede à qual o computador que roda o *Wireshark* está conectado. Desta forma, utilizando filtros apropriados, pode-se separar apenas os pacotes destinados à plataforma *Hexapod* e, desta maneira, verificar se o requisito temporal de envio de pacotes está sendo respeitado.

Para isso, foram feitas diversas capturas de pacotes na rede. O *log* de uma dessas capturas é mostrado na figura 5.1. Os filtros utilizados para separar essa captura dizem respeito ao endereço de IPv4 utilizado no programa desenvolvido neste trabaho para envio dos pacotes UDP ("10.10.10.11") e o IPv4 do destino ("10.10.10.10"). Desta forma, foram isolados os pacotes a serem analisados.

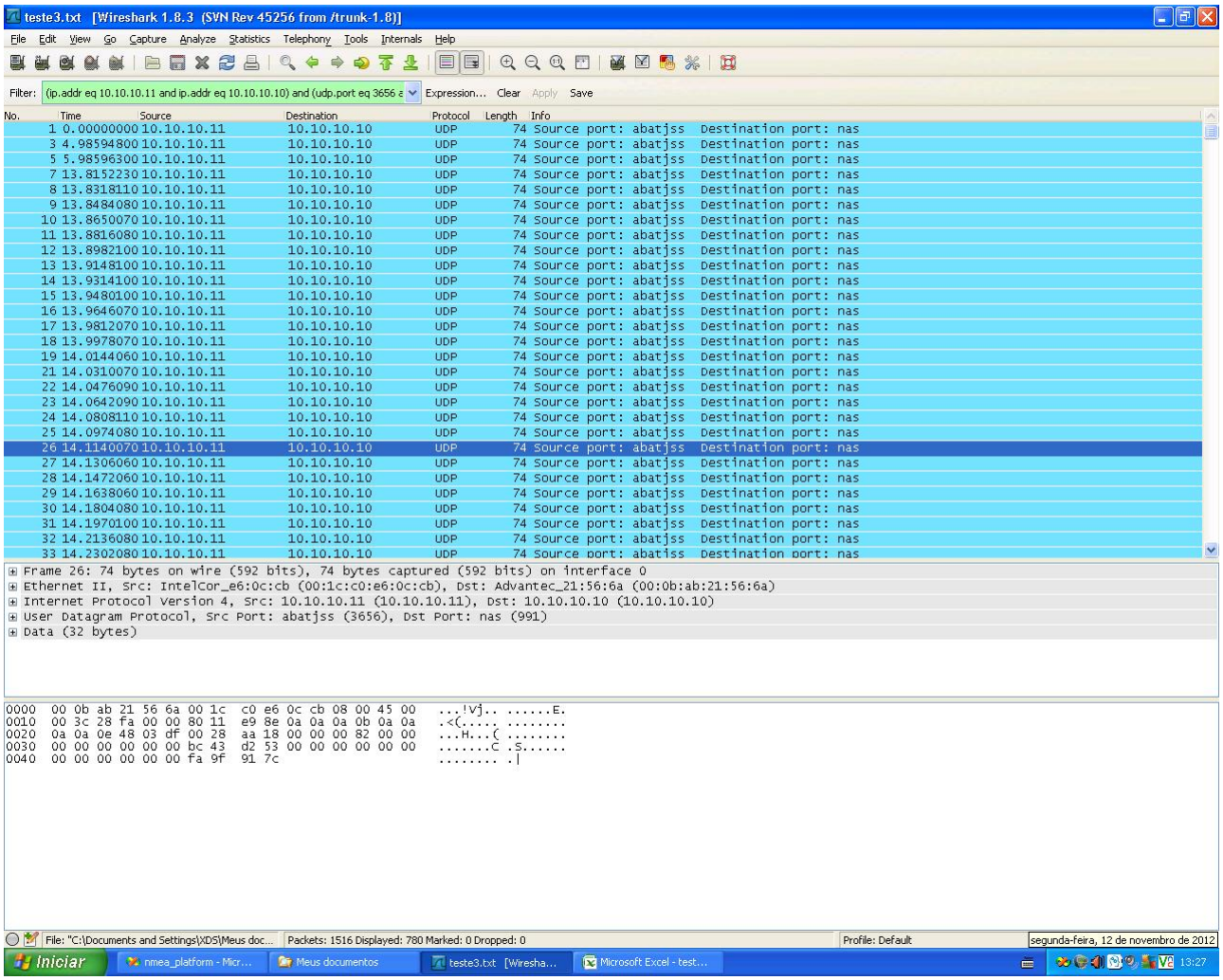

Figura 5.1: Visualização do *log* do Wireshark

Como a precisão do *Wireshark* nos *timestamps* fornecidos é da ordem de microsegundo e a precisão necessária é da ordem de milissegundo, a ferramenta pode

ser utilizada para fazer uma análise estatística da janela temporal entre o envio de um pacote a outro para confirmar e validar a rotina de temporização como adequada à esta aplicação.

A média ($E(x)$) dos intervalos de tempo entre o envio de um pacote e o envio do próximo (x_i) é dada por [10]:

$$E(x) = \frac{\sum_{i=1}^{N} x_i}{N} \tag{5.1}$$

E o desvio padrão é calculado por:

$$\sigma(x) = \sqrt{\sum_{i=1}^{N} (x_i - E(x))^2} \tag{5.2}$$

Utilizando as fórmulas acima, a rotina de temporização forneceu os seguintes parâmetro (com $N = 1253$):

- Média aritmética: $E(x) = 16,601ms$
- Desvio padrão: $\sigma(x) = 0,031ms$

De acordo com as especificações do fabricante da plataforma, a precisão necessária é de décimos de milissegundo e a precisão obtida com a rotina de temporização deste trabalho é de centésimos de milissegundo. Portanto, a rotina é adequada e satisfaz os requisitos necessários.

Capítulo 6

Comunicação com o *software* TPN

6.1 O simulador do TPN

O laboratório Tanque de Provas Numérico (TPN) da Escola Politécnica desenvolveu nos últimos anos um simulador, em uma parceria com a Petrobras, capaz de reproduzir operações *offshore* como o *offloading* de plataformas da própria Petrobras.

Para a representação dos corpos, um modelo dinâmico com 6 graus de liberdade é utilizado. Este modelo inclui forças hidrodinâmicas, aerodinâmicas e de ondas. O simulador realiza os cálculos utilizando os algoritmos de DP ("*Dynamic Positioning*"), que considera a eficiência e a dinâmica de cada um dos propulsores presentes no corpo flutuante simulado [11].

O simulador é dividido em duas partes principais que se intercomunicam via *socket* e protocolo "http". A primeira é o módulo de cálculo, integrando as equações diferenciais dos modelos matemáticos presentes no código numérico do TPN que vem sendo desenvolvido e aprimorado há 15 anos.

A arquitetura do simulador representando as interfaces de comunicação é mostrada na figura 6.1

Já o módulo de visualização visa a garantia de imersão do operador dos instrumentos desenvolvidos. Esse processo é baseado na ferramenta *Unity3D*, que permite a modelagem de diversos cenários com qualidade gráfica muito boa e facilidade de implementação. As figuras 6.2 e 6.3 mostram exemplos de visualização gráfica gerada pelo *software* TPN.

Para aumentar a sensação de imersão no ambiente de uma ponte de comando de um corpo flutuante, foram desenvolvidos painéis de controle capazes de mostrar todos os instrumentos relacionados, como informação a respeito do DP, sistemas de posicionamento, bússola e estado dos propulsores, entre outros. Este painel foi desenvolvido em um projeto separado e comunica com o *software* também via *socket*

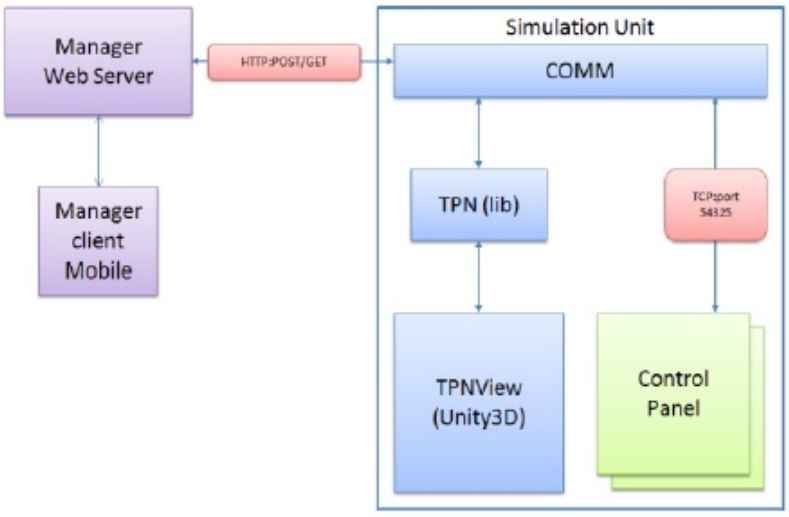

Figura 6.1: Arquitetura do simulador TPN

Figura 6.2: Exemplo de visualização gerada através do *software* TPN

Figura 6.3: Outro exemplo de visualização

de rede. A figura 6.4 mostra o joystick desenvolvido no TPN para comando dos propulsores dos corpos flutuantes simulados, já a figura 6.5 mostra o painel de controle desenvolvido no TPN.

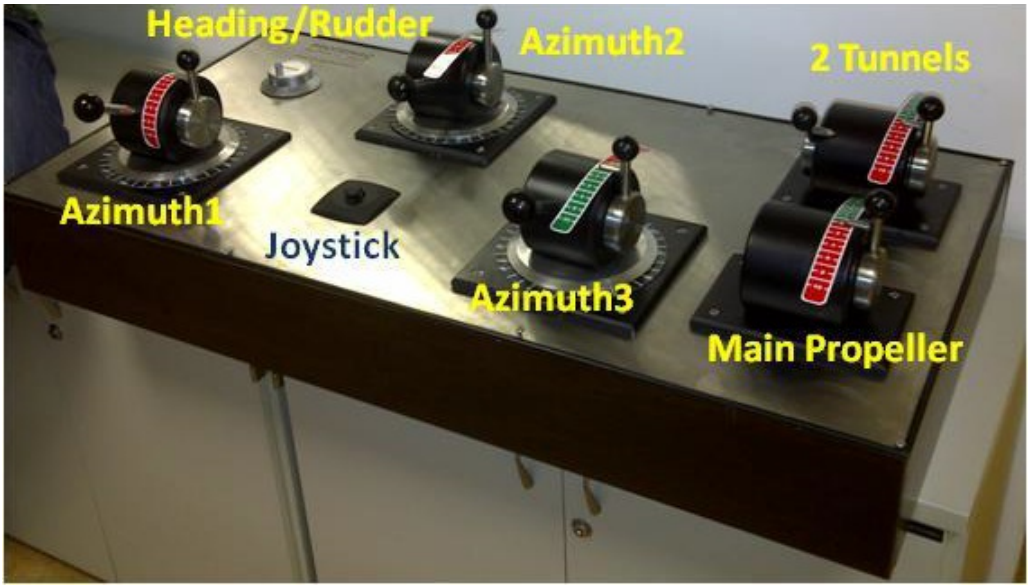

Figura 6.4: Joystick desenvolvido no TPN

Figura 6.5: Painel de controle desenvolvido no TPN

Por fim, uma simulação da ponte de comando desenvolvida no TPN, com diversas telas de visualização e o painel de controle mostrado nas figuras anteriores é mostrado em 6.6.

Figura 6.6: Simulação da ponte de comando desenvolvida

6.2 O protocolo NMEA

O protocolo NMEA é um conjunto de especificações criada pela *"National Marine Electronics Association"* muito utilizado na indústria naval. Atualmente, a versão atual é o padrão NMEA 0183, que vem sendo substituído pelo padrão NMEA 2000.

O padrão elétrico utilizado pelo NMEA é o RS-422. O NMEA 0183 utiliza comunicação serial com codificação ASCII e normalmente a mensagem pode ser enviada a partir de um dispositivo para diversos outros ao mesmo tempo (*"multicast"*).

A camada de Data Link do NMEA tem como padrão 8 bits de dado, 1 bit de paridade e não apresenta mecanismos de paridade ou de *handshake*.

Na camada de aplicação, vale o seguinte conjunto de regras:

1. A mensagem NMEA padrão deve iniciar com um sinal "$".
2. Após o "$", os dois próximos caracteres ASCII identificam o emissor.
3. 3 caracteres indicando o tipo de mensagem enviada.
4. Após a identificação da mensagem e do emissor, vem os campos de dados, separados por vírgulas.
5. Opcionalmente, pode-se utilizar um mecanismo de *checksum* ao final da mensagem
6. A mensagem termina com uma nova linha (comandos de*Carriage Return* e *Line Feed*).

Um exemplo de mensagem enviada pelo TPN utilizando protocolo NMEA é:

```
1
$S5WSD,10,11.496,260.086*73
3 $S5PRC,10,1,10.013,90.000,0.963,225.000,0.005,ON*1d
$S5PRC,10,2,83.466,360.000,-6.399,243.000,0.621,ON*93
5 $S5PRC,10,3,10.578,358.997,-0.811,243.000,0.644,ON*57
$S5PRC,10,4,0.312,90.000,-0.034,225.000,-0.000,ON*a6
7 $S5PRC,10,5,332.748,0.000,-24.267,54.600,2.008,ON*3a
$S5PRD,10,1,10.013,90.000,0.963,225.000,0.005,ON*ab
9 $S5PRD,10,2,83.466,360.000,-6.399,243.000,0.621,ON*ce
$S5PRD,10,3,10.578,358.997,-0.811,243.000,0.644,ON*0a
11 $S5PRD,10,4,0.312,90.000,-0.034,225.000,-0.000,ON*44
$S5PRD,10,5,332.748,0.000,-24.267,54.600,2.008,ON*d8
```

6.3 Programa de recebimento de dados do TPN usando NMEA

Para a comunicação com o *software* TPN foi desenvolvida uma classe somente com este propósito. A classe possui métodos para analisar as mensagens NMEA enviadas pelo *software* e armazená-las em uma estrutura de dados chama *vessel*. O código da classe é reportado abaixo:

```
1
nmea_comm::nmea_comm(std::string p_ip, int p_port, int p_vessel_id) :
3 comm(p_ip, p_port)
{
5   vessel.id          = p_vessel_id;

7   commands["PTDPP"] = new nmea_ptdpp(p_vessel_id,
        vessel.x    , vessel.y      , vessel.z    ,
9       vessel.roll , vessel.pitch  , vessel.yaw) ;
    commands["PTDPV"] = new nmea_ptdpv(p_vessel_id,
11      vessel.vx   , vessel.vy     , vessel.vz   ,
        vessel.vroll, vessel.vpitch , vessel.vyaw);
13  commands["PTDPA"] = new nmea_ptdpa(p_vessel_id,
        vessel.ax   , vessel.ay     , vessel.az   ,
15      vessel.aroll, vessel.apitch , vessel.ayaw);
}
17
nmea_comm::position nmea_comm::get()
19 {
    vessel.new_data = false;
21  std::vector<std::string> result;
    while(!(result = comm.get()).empty()) {
23    if(commands.find(result[0]) != commands.end()) {
        try {
25        commands[result[0]]->exec(result);
          vessel.new_data = true;
27      } catch(int) {
        }
29    }
    }
31  return vessel;
}
33
nmea_comm::~nmea_comm()
35 {
```

```
}
```

Capítulo 7

Tratamento dos dados recebidos para envio à plataforma

7.1 O problema das diferenças de frequências de recebimento e envio

Conforme descrito na introdução deste trabalho, são necessárias duas frequências de trabalho distintas para comunicação. A primeira, dada pelo período de amostragem T_{a1}, depende do *software* TPN. A configuração do TPN especifica o período com que as posições do navio ensaiado serão disponibilizadas para envio, usando protocolo NMEA, para o exterior.

Já para a plataforma *Hexapod* é necessário um envio de dados à uma frequência de 60Hz, ou seja, um período de amostragem de $T_{a2} = 0.0166s$. Como se pode notar, o fato de as frequências de envio e recebimento de dados serem distintas implica na necessidade de tratar os dados a serem enviados. Como $T_{a2} < T_{a1}$, o programa desenvolvido neste trabalho deve criar um algoritmo para calcular os dados a serem enviados à plataforma enquanto a nova posição não é recebida do TPN.

Para este fim, foram testados diversos algoritmos que serão descritos a seguir. Cada um com suas vantagens e desvantagens que serão discutidas nas próximas seções. Após uma investigação de cada um desses algoritmos, foi feita a escolha daquele que melhor se adapta ao escopo deste trabalho.

7.2 O Zero Order Hold

O Zero Order Hold é o algoritmo mais simples discutido aqui. Basicamente, se trata de receber uma posição do TPN e, enquanto uma nova posição não é enviada,

manter a saída constante. Assim que um novo dado chega, a saída muda imediatamente para este novo valor.

Este algoritmo tem como grande vantagem a sua simplicidade. Ele pode ser implementado utilizando poucas linhas de código e portanto pode ser ideal para sistemas onde há muito pouco tempo de processamento livre para se executar os cálculos ou ainda quando não há memória para se armazenar os dados gerados anteriormente.

Como grandes desvantagens, este algoritmo não utiliza os dados passados para efetuar qualquer tipo de previsão a respeito dos novos dados a serem produzidos. Isto gera grandes descontinuidades na série temporal enviada. Além de o erro entre a posição real e a posição enviada para a plataforma serem normalmente muito grandes, fora no momento em que a nova posição é enviada - neste caso, a posição enviada e a real são iguais.

Neste trabalho, os dados são recebidos com uma frequência baixa: 2Hz, ou seja, um período de $0,5s$ e devem ser enviadas a 60Hz ou $0,0166s$. É inaceitável para o movimento da plataforma que existam descontinuidades tão grandes como as presentes utilizando este algoritmo. Portanto, é necessário algum tipo de previsão a respeito dos novos dados a serem enviados. Estes algoritmos serão discutidos a seguir.

Os gráficos obtidos com este algoritmo são mostrados a seguir. A figura 7.1 mostra o movimento de *Pitch* e a figura 7.2 mostra o movimento de *Roll* geradas a partir do algoritmo de Zero Order Hold.

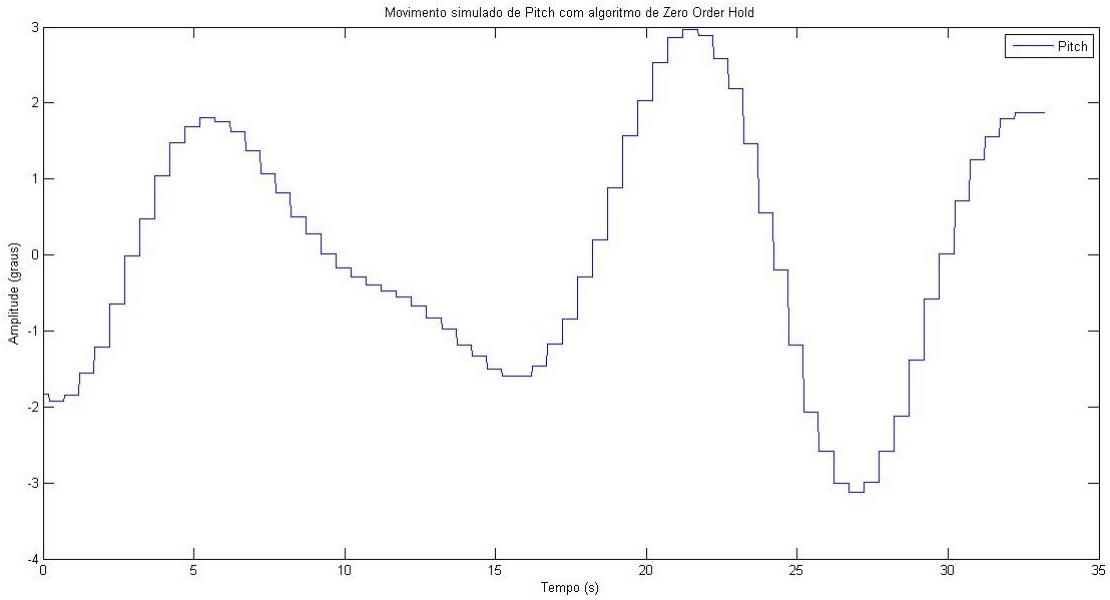

Figura 7.1: Pitch obtido com Zero Order Hold

O código utilizado para se fazer os testes utilizando o Zero Order Hold é

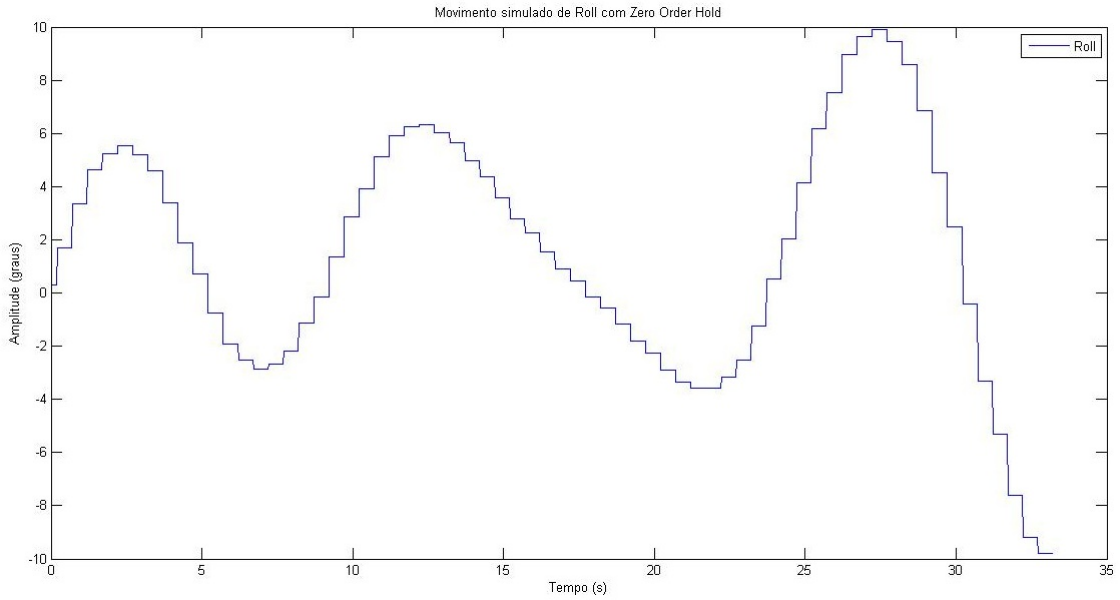

Figura 7.2: Roll obtido com Zero Order Hold

mostrado abaixo. Notar que a função $comm.get()$ acessa o *struct* de dados que se comunica ao TPN. Caso os dados tenham sido atualizados, a variável booleana $pos.new\_data$ recebe o valor *true*. E assim que as posições são atualizadas no *software*, este velor é mudado para *false*.

```
1
while (true){
3   pos = comm.get();

5   // Rampa inicial para atingir altura de Heave
    if (t<T){
7   z = (t/T)*(-0.2);
    }
9
    else if (pos.new_data){
11    // atualiza valor do roll
      xreal[0] = pos.roll;
13
      // atualiza valor do pitch
15    xreal[1] = pos.pitch;

17    pos.new_data = false;
    }
19
    // interpolacao dos dados:
21  for (i=0; i<2; i++){
      // algoritmo de zero order hold
23    x[i] = xreal[i];

25    //Controle de aceleracoes, velocidades e posicoes da plataforma

27    if (fabs(ax[1] > AMAX || fabs(vx[1] > VMAX || fabs(x[i]) > PMAX){
        printf ("Excedidos os limites de seguranca da plataforma");
29      return -1;
      }
31  }
```

```
33

35 }
```

7.3 extrapolação dos dados recebidos em ingresso usando *First Order Hold*

Supondo, inicialmente, que as série temporais de todos graus de liberdade do navio que serão reproduzidas possam ser representadas através de uma onda senoidal do tipo:

$$x(t) = Asen\left(\frac{2\pi}{T}t\right) \tag{7.1}$$

Onde:

A é a amplitude da onda senoidal;

T é o período da onda senoidal;

t é o tempo;

x(t) é a série temporal de um grau de liberdade genérico.

Amostrando a série temporal descrita por 7.1 em intervalos de tempo T_{a1}, obtemos a seguinte série discretizada:

$$y_k(kT_{a1}) = Asen\left(\frac{2\pi}{T}kT_{a1}\right) \tag{7.2}$$

O método de *First Order Hold* consiste em calcular o primeiro termo da expansão em séries de Taylor da série temporal amostrada, ou simplesmente calcular a reta que une os dois últimos pontos amostrados, conforme a equação 7.3, obtendo o coeficiente angular m.

$$m = \frac{y_k - y_{k-1}}{T_{a1}} \tag{7.3}$$

Os dados a serem enviados a uma frequência mais elevada, $y'_(kn + j)$ caracterizada pelo período de amostragem T_{a2}, são então calculados utilizando o valor m, através da equação 7.4:

$$y'_{kn+j+1} = y'_{kn+j} + mT_{a2} \tag{7.4}$$

Sendo que o primeiro termo de cada extrapolação, y'_{kn}, é calculado através do último dado recebido de forma que $y'_{kn} = y_k$, onde n é a relação entre os perídos de amostragem, tal que $n = \frac{T_a1}{T_a2}$.

Para simulação de uma amostragem utilizando o *First Order Hold*, o código abaixo foi utilizado. Os gráficos obtidos demonstram o comportamento deste tipo de simulação. Em especial, deve-se notar as descontinuidades que ocorrem quando um novo dado é recebido.

```
1
while (true){
3   pos = comm.get();

5   // Rampa inicial para atingir altura de Heave
    if (t<T){
7   z = (t/T)*(-0.2);
    }
9
    else if (pos.new_data){
11    // calcula o coeficiente angular m
      mx[0] = (pos.roll - x[0])/Ta1;
13    mx[1] = (pos.pitch - x[1])/Ta1;

15    // atualiza valores de roll e pitch
      x[0] = pos.roll;
17    x[1] = pos.pitch;

19    pos.new_data = false;
    }
21
    // interpolacao dos dados:
23  for (i=0; i<2; i++){
      // algoritmo de first order hold
25    x[i] = x[i] + mx[i]*dt;

27    //Controle de aceleracoes, velocidades e posicoes da plataforma

29    if (fabs(ax[1] > AMAX || fabs(vx[1] > VMAX || fabs(x[i]) > PMAX){
        printf ("Excedidos os limites de seguranca da plataforma");
31      return -1;
      }
33  }
```

Os resultados da simulação pode ser vistos nas figuras 7.3 e 7.4. As descontinuidades apresentadas são fatores que depreciam a qualidade do movimento da plataforma *Hexapod*, pois representam desconforto, na forma de "trancos"durante o movimento. Para resolver este problema, o método foi alterado de maneira a retirar as descontinuidades obtidas, conforme descrito na próxima seção.

7.4 O *First Order Hold* modificado

Para retirar as descontinuidades obtidas utilizando o método FOH, o algoritmo desenvolvido consiste no cálculo de um resíduo r, que mede o erro entre o último ponto enviado com a frequência de amostragem mais elevada e o novo dado recebido, calculado através da equação 7.5.

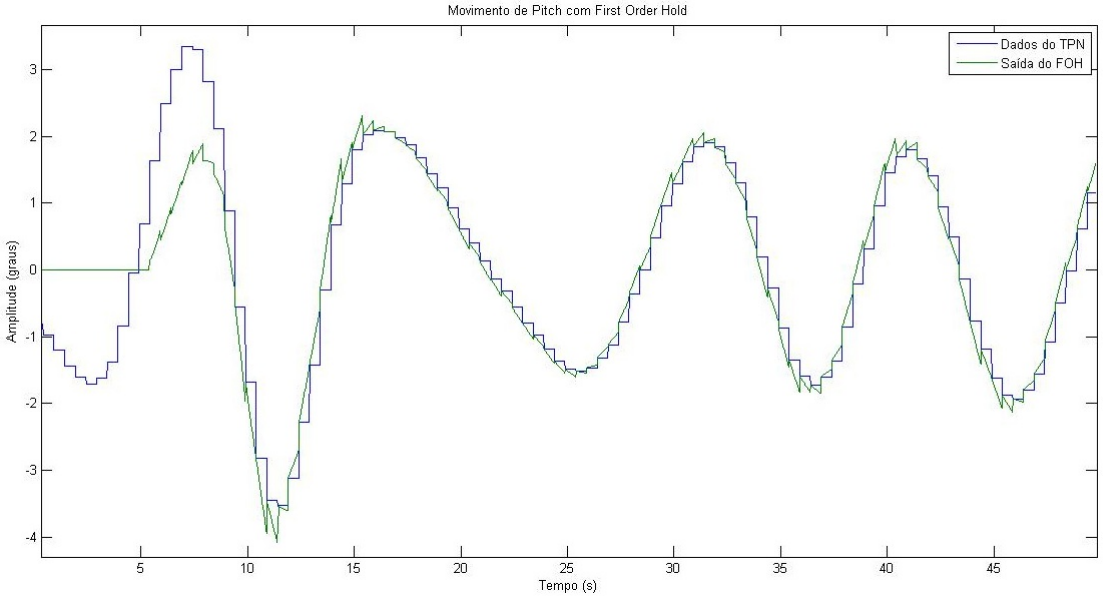

Figura 7.3: Pitch obtido com First Order Hold

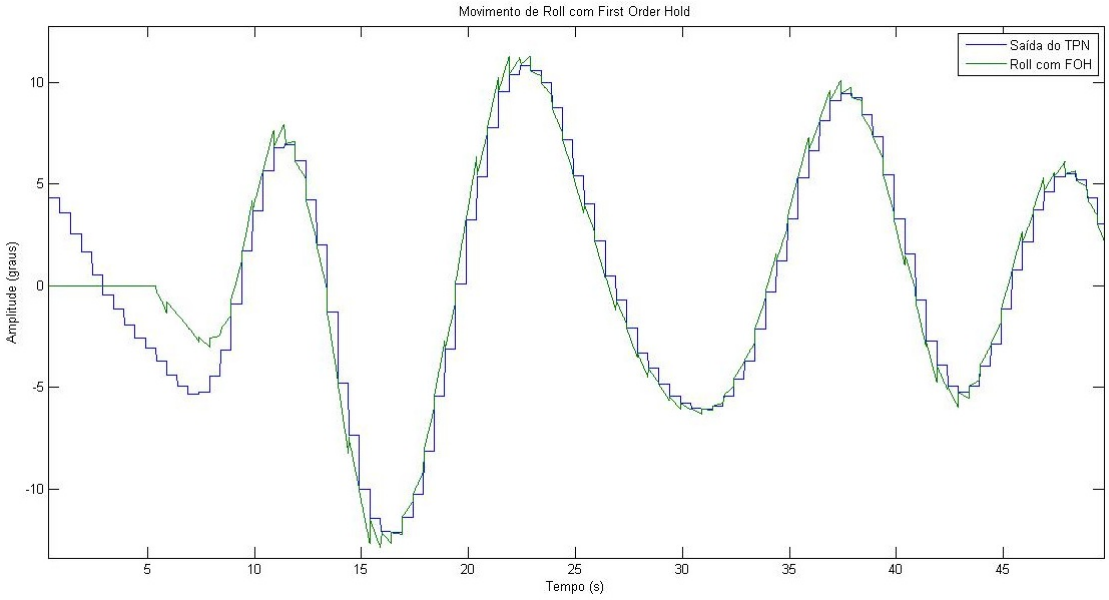

Figura 7.4: Roll obtido com First Order Hold

$$r = y_k - y'_{kn} \tag{7.5}$$

Ao invés de corrigir este resíduo imediatamente, como no caso do método FOH, pode-se amortizar a redução do resíduo r durante o próximo período de extrapolação de maneira que, ao calcular o valor $y'_{(k+1)n}$, o resíduo r tenha desaparecido. Ou seja:

$$y'_{(k+1)n} = y'_{kn} + mnT_{a2} + r \tag{7.6}$$

Portanto, os passos intermediários são calculados através da seguinte equação:

$$y'_{kn+j+1} = y'_{kn+j} + mT_{a2} + \frac{r}{n} \tag{7.7}$$

Para este cálculo, o seguinte código foi utilizado:

```
while (true){
  pos = comm.get();

  // Rampa inicial para atingir altura de Heave
  if (t<T){
  z = (t/T)*(-0.2);
  }

  else if (pos.new_data){
    // calcula o coeficiente angular m
    mx[0] = (pos.roll - areal[0])/Ta1;
    mx[1] = (pos.pitch - areal[1])/Ta1;

    // atualiza valores de roll e pitch
    xreal[0]= pos.roll;
    xreal[1] = pos.pitch;

    // calcula os residuos
    rx[0] = xreal[0] - x[0];
    rx[1] = xreal[1] - x[1];

    pos.new_data = false;
  }

  // interpolacao dos dados:
  for (i=0; i<2; i++){
    // algoritmo de first order hold com residuos
    x[i] = x[i] + mx[i]*dt + rx[i]/n;

    //Controle de aceleracoes, velocidades e posicoes da plataforma

    if (fabs(ax[1] > AMAX || fabs(vx[1] > VMAX || fabs(x[i]) > PMAX){
      printf ("Excedidos os limites de seguranca da plataforma");
      return -1;
    }
  }
```

Os resultados da simulação podem ser vistos nas figuras 7.5 e 7.6. As indesejadas continuidades desaparecem, mas o movimento ainda apresenta descontinuidades na sua primeira derivada, como pode ser observado pelas "pontas"que aparecem nos gráficos de movimento.

7.5 Algoritmo de integração das velocidades e amortização dos resíduos de posição

O próximo algoritmo considerado baseia-se no fato de a informação gerada a partir do *software* TPN não dizer respeito apenas à posição do navio simulado, mas

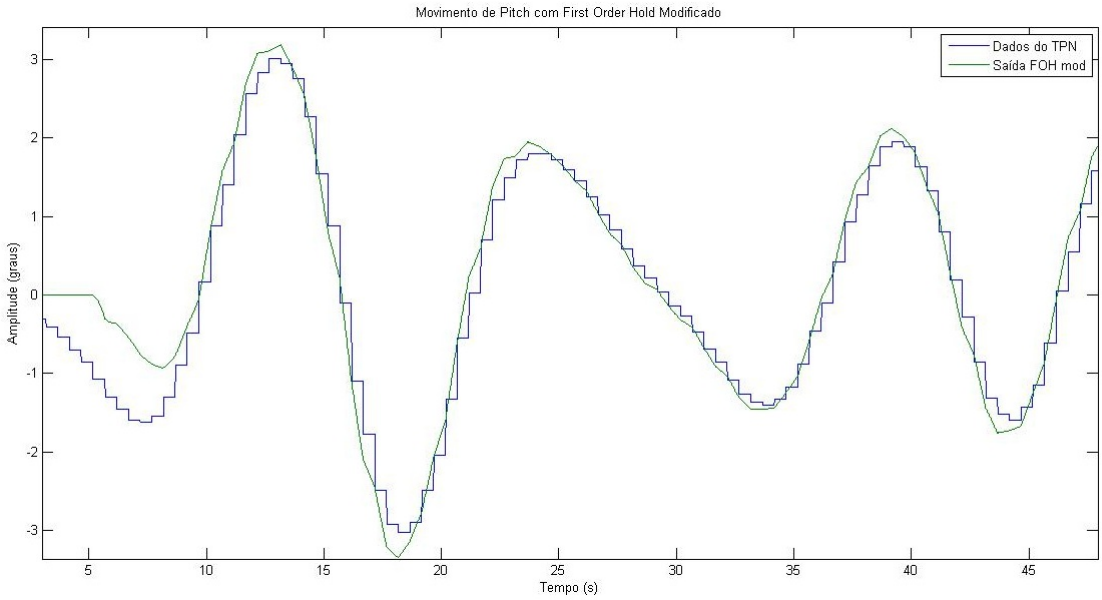

Figura 7.5: Pitch obtido com First Order Hold modificado

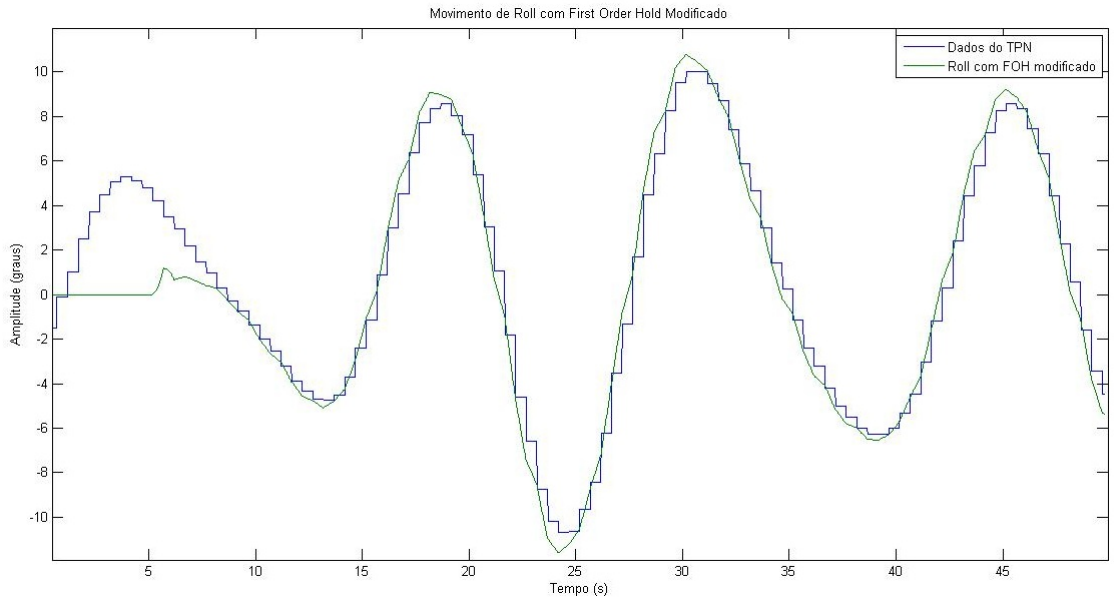

Figura 7.6: Roll obtido com First Order Hold modificado

também às velocidades e acelerações do mesmo. Essas informações não estavam sendo utilizadas anteriormente, mas podem ser de grande valia para a extrapolação dos dados.

Este algoritmo consiste em prever a posição do navio utilizando o algoritmo de Zero Order Hold para a velocidade, isto é, para cada novo conjunto de dados recebidos do TPN, extrai-se a velocidade, que será constante até que os próximos dados cheguem. Esta velocidade será integrada em tempos discretos para a obtenção das novas posições. Além disso, para evitar as descontinuidades indesejadas, é utilizado

o artifício de amortização dos resíduos como no algoritmo anterior.

A grande vantagem deste algoritmo é que, ao invés de se calcular o coeficiente angular utilizando-se os dois últimos valores de posição - que nada mais é que a velocidade média durante o período, utiliza-se uma medida mais precisa para a extrapolação: a velocidade instantânea obtida diretamente através da saída do *software* TPN.

Como a posição é a integral da velocidade, conforme 7.8:

$$x(t) = \int v(t)dt \tag{7.8}$$

Temos que:

$$x(i+1) = x(i) + v(i)\Delta t + r_x(i)/n \tag{7.9}$$

Onde $v(i)$ é o último dado de velocidade recebido e $r_x(i)/n$ é o termo referente à amortização do erro de posição calculado quando o último dado de posição foi recebido.

O código para implementação deste algoritmo é mostrado abaixo:

```
while (true){
  pos = comm.get();

  // Rampa inicial para atingir altura de Heave
  if (t<T){
  z = (t/T)*(-0.2);
  }

  else if (pos.new_data){


    // atualiza valores de roll e pitch
    xreal[0] = pos.roll;
    xreal[1] = pos.pitch;

    // atualiza as velocidades
    vx[0] = pos.vroll;
    vx[1] = pos.vpitch;

    // calcula os residuos
    rx[0] = xreal[0] - x[0];
    rx[1] = xreal[1] - x[1];

    pos.new_data = false;
  }

  // interpolacao dos dados:
  for (i=0; i<2; i++){
    // algoritmo de integracao das velocidades
    x[i] = x[i] + vx[i]*dt + rx[n];

    //Controle de aceleracoes, velocidades e posicoes da plataforma

    if (fabs(ax[1] > AMAX || fabs(vx[1] > VMAX || fabs(x[i]) > PMAX){
      printf ("Excedidos os limites de seguranca da plataforma");
      return -1;
```

```
        }
38    }
```

O resultado das simulações com estes algoritmos pode ser visto nas figuras 7.7 (Pitch) e 7.8 (Roll). Os resultados já são evidentemente melhores, mas os próximos algoritmos mostram que ainda existe margem para melhora.

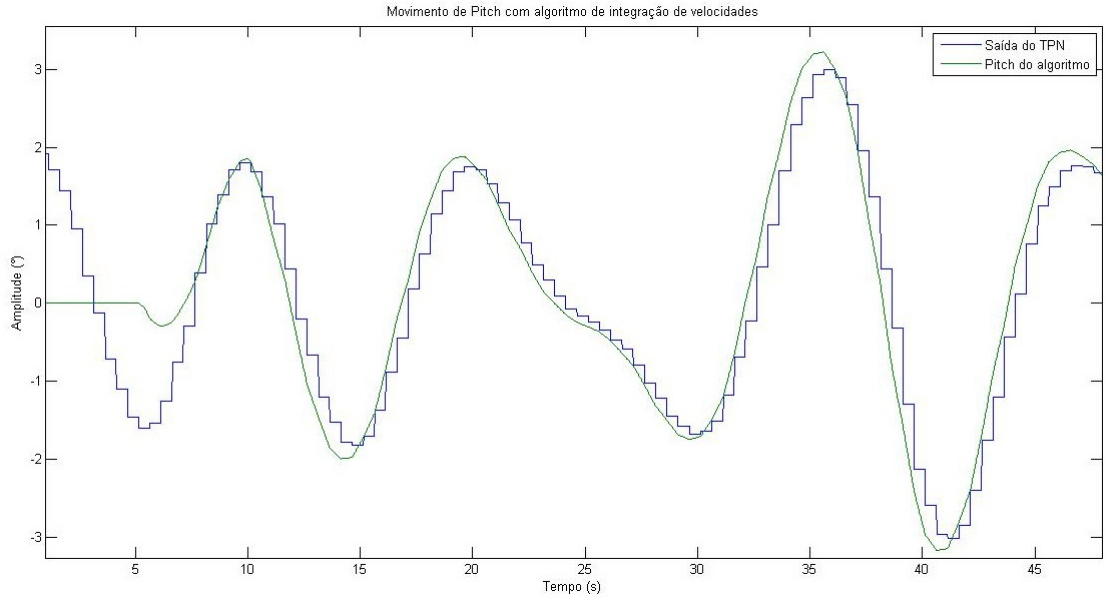

Figura 7.7: Pitch obtido com algoritmo de integração de velocidades

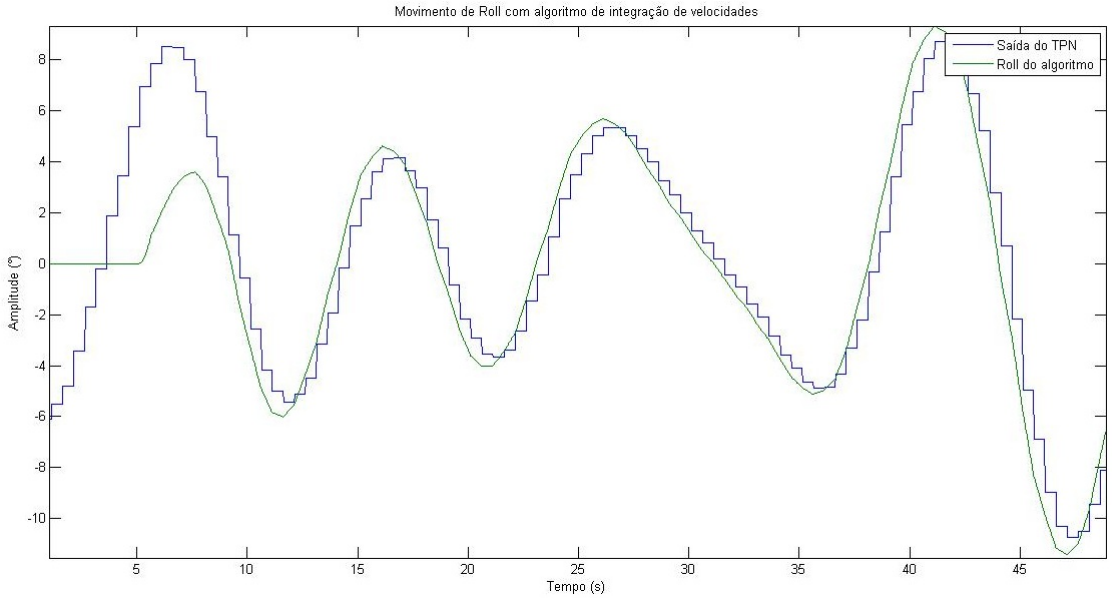

Figura 7.8: Roll obtido com First Order Hold modificado

7.6 Algortimo de integração das acelerações recebidas

Uma continuidade do algoritmo anterior é, além de utilizar as velocidades recebidas do *software* TPN, utilizar também as acelerações. Desta forma, no período de extrapolação, as velocidades são obtidas pela integração da aceleração e a posição pela integral dupla da aceleração. A aceleração é mantida constante através do algoritmo de Zero Order Hold. A cada novo recebimento de dados, a aceleração e a velocidade são atualizadas. A posição ainda se utiliza do artifício de amortização dos resíduos para evitar as descontinuidades.

Como a velocidade é calculada como a integral da aceleração, conforme **??**:

$$v(t) = \int a(t)dt \tag{7.10}$$

A posição pode ser calculada por:

$$x(t) = \int \int a(t)dt \tag{7.11}$$

Que na forma discretizada torna-se:

$$x(i+1) = x(i) + v(i)\Delta t + \frac{a(i)(\Delta t)^2}{2} + r_x/n \tag{7.12}$$

E,

$$v(i+1) = v(i) + a(i)\Delta t \tag{7.13}$$

O código utlizado para este método é mostrado abaixo:

```
while (true){
  pos = comm.get();

  // Rampa inicial para atingir altura de Heave
  if (t<T){
  z = (t/T)*(-0.2);
  }

  else if (pos.new_data){


    // atualiza valores de roll e pitch
    xreal[0] = pos.roll;
    xreal[1] = pos.pitch;

    // atualiza as velocidades
    vx[0] = pos.vroll;
    vx[1] = pos.vpitch;

    // atualiza as aceleracoes
    ax[0] = pos.aroll;
    ax[1] = pos.apitch;

    // calcula os residuos
    rx[0] = xreal[0] - x[0];
    rx[1] = xreal[1] - x[1];

    pos.new_data = false;
  }

  // interpolacao dos dados:
  for (i=0; i<2; i++){
    // algoritmo de integracao das aceleracoes
    x[i] = x[i] + vx[i]*dt + ax[i]*dt*dt/2 + rx[n];
    v[i] = v[i] + ax[i]*dt;

    //Controle de aceleracoes, velocidades e posicoes da plataforma

    if (fabs(ax[1] > AMAX || fabs(vx[1] > VMAX || fabs(x[i]) > PMAX){
      printf ("Excedidos os limites de seguranca da plataforma");
      return -1;
    }
  }
```

Os resultados obtidos com este algoritmo são reportados nas figuras 7.9 e 7.10. Também são mostradas as velocidades angulares de Pitch e Roll nas figuras 7.11 e 7.12. Nota-se que agora a velocidade utilizada não é mais constante, mas obtida pela integração das velocidades recebidas. Desta maneira, a posição, obtida pela integração da velocidade reproduz ainda mais fielmente o movimento simulado através do TPN, mesmo com a baixa frequência em que os dados são recebidos.

7.7 Utilizando o Fisrt Order Hold para as acelerações

Finalmente, uma última modificação que pode ser feita no algoritmo é utilizar o algoritmo de First Order Hold aplicado às acelerações recebidas e, com essas

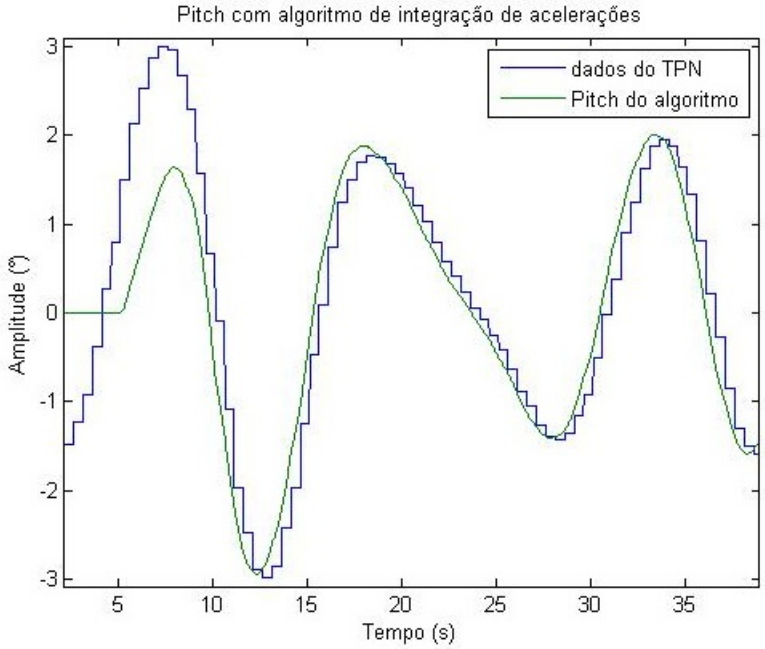

Figura 7.9: Pitch obtido com algoritmo de integração da aceleração

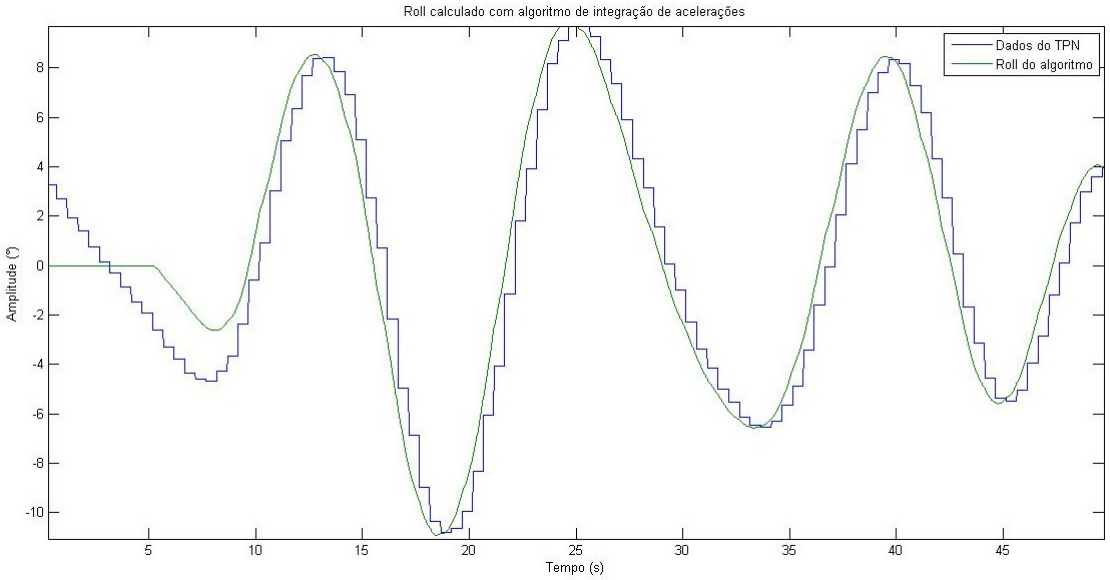

Figura 7.10: Roll obtido com algoritmo de integração da aceleração

acelerações, calcular as velocidades e posições através de integração.

Para isto, deve-se calcular o coeficiente angular da reta que une os dois últimos valores recebidos de aceleração, conforme 7.14 e, a cada passo de extrapolação calcular a nova aceleração utilizando este coeficiente angular, conforme 7.15.

$$m_a(t) = \frac{a_k - a_{k-1}}{\Delta t_1} \tag{7.14}$$

$$a(t) = m_a(t) * t + a_0 \tag{7.15}$$

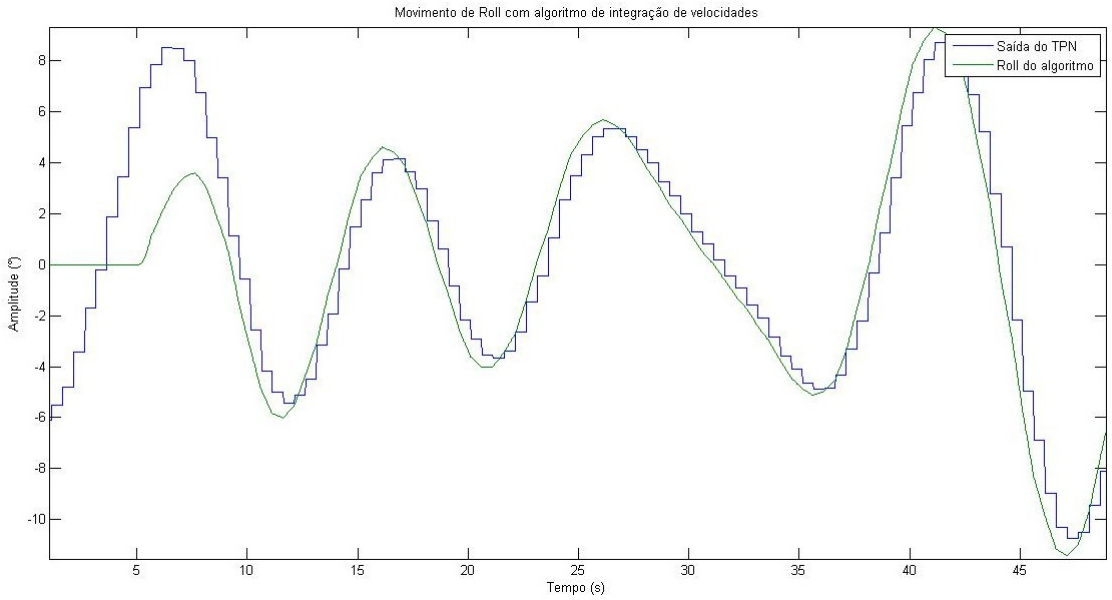

Figura 7.11: Velocidade angular de Pitch obtido com algoritmo de integração da aceleração

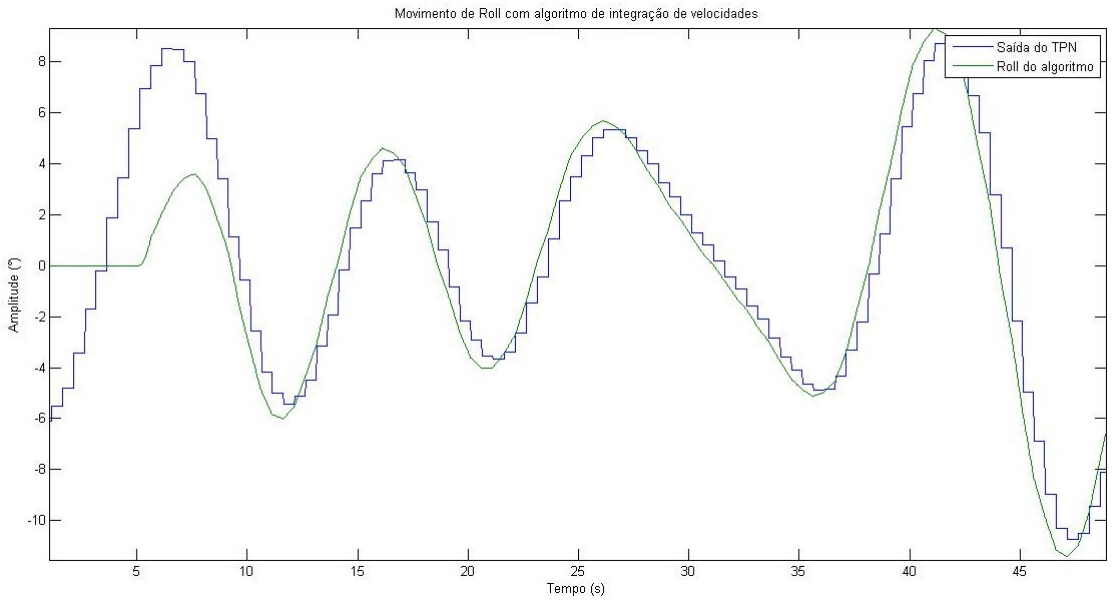

Figura 7.12: Velocidade angular de Roll obtido com algoritmo de integração da aceleração

Então a velocidade pode ser calculada por:

$$v(t) = \int a(t)dt \tag{7.16}$$

Ou, na forma discretizada:

$$v(i+1) = v(i) + a(i)\Delta t + m_a(i)\frac{(\Delta t)^2}{2} \tag{7.17}$$

E a posição pode ser calculada por:

$$x(i+1) = x(i) + v(i)\Delta t + a(i)\frac{(\Delta t)^2}{2} + m_a(i)\frac{(\Delta t)^3}{6} + r_x(i)/n \tag{7.18}$$

O código que implementa este método é mostrado abaixo:

```
while (true){
  pos = comm.get();

  // Rampa inicial para atingir altura de Heave
  if (t<T){
  z = (t/T)*(-0.2);
  }

  else if (pos.new_data){

    // calcula o coeficiente angular da aceleracao
    ma[0] = (pos.aroll - areal[0])/Ta1;
    ma[1] = (pos.apitch - areal[1])/Ta1;

    // atualiza valores de roll e pitch
    xreal[0] = pos.roll;
    xreal[1] = pos.pitch;

    // atualiza as velocidades
    vx[0] = pos.vroll;
    vx[1] = pos.vpitch;

    // atualiza as aceleracoes
    ax[0] = pos.aroll; areal[0]= pos.aroll;
    ax[1] = pos.apitch; areal[1] = pos.apitch;

    // calcula os residuos
    rx[0] = xreal[0] - x[0];
    rx[1] = xreal[1] - x[1];

    pos.new_data = false;
  }

  // interpolacao dos dados:
  for (i=0; i<2; i++){
    // algoritmo de integracao das aceleracoes
    x[i] = x[i] + vx[i]*dt + ax[i]*dt*dt/2 + ma[i]*dt*dt*dt/6 + rx[n];
    v[i] = v[i] + ax[i]*dt + ma[i]*dt*dt/2;
    ax[i] = ax[i] + ma[i]*dt;

    //Controle de aceleracoes, velocidades e posicoes da plataforma

    if (fabs(ax[1] > AMAX || fabs(vx[1] > VMAX || fabs(x[i]) > PMAX){
      printf ("Excedidos os limites de seguranca da plataforma");
      return -1;
    }
  }
```

Os resultados das simulações com este algoritmo são mostrados nas figuras 7.13, 7.14, 7.15, 7.16, 7.17 e 7.18.

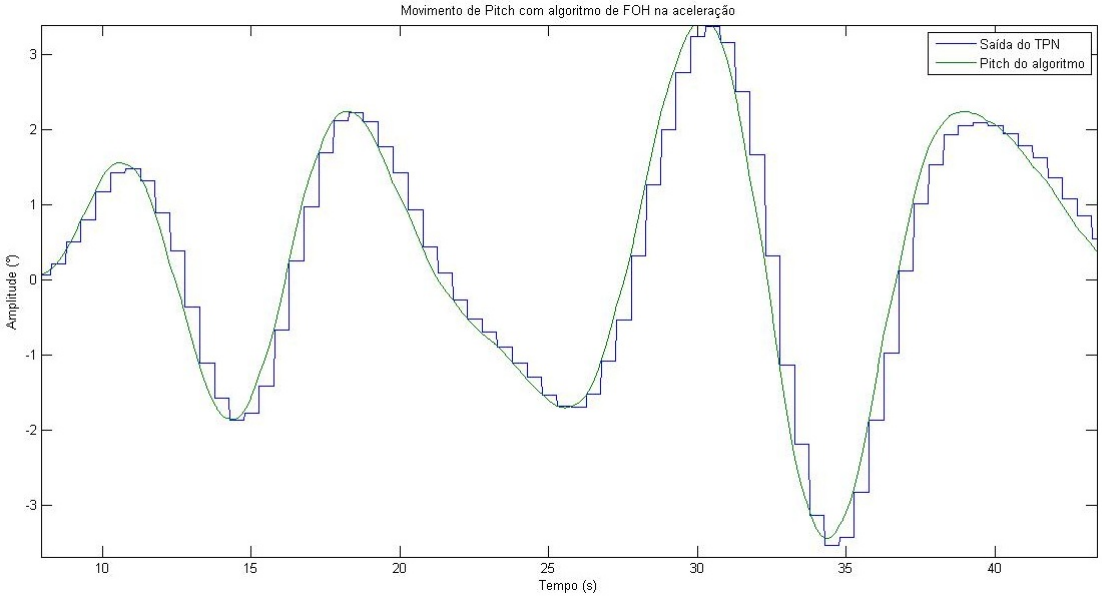

Figura 7.13: Pitch obtido com algoritmo de First Order Hold aplicado à aceleração

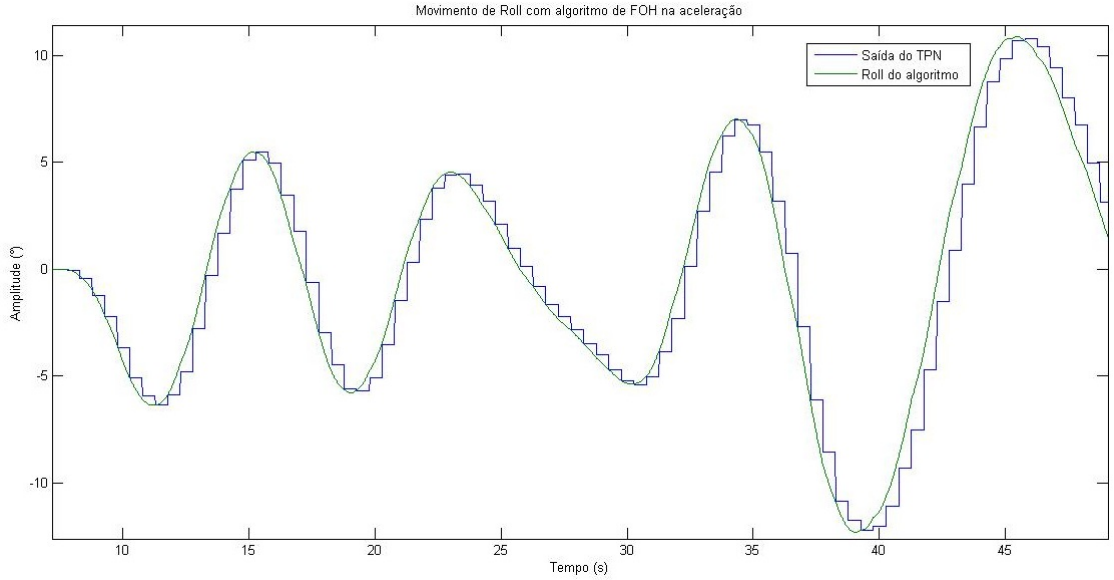

Figura 7.14: Roll obtido com algoritmo de First Order Hold aplicado à aceleração

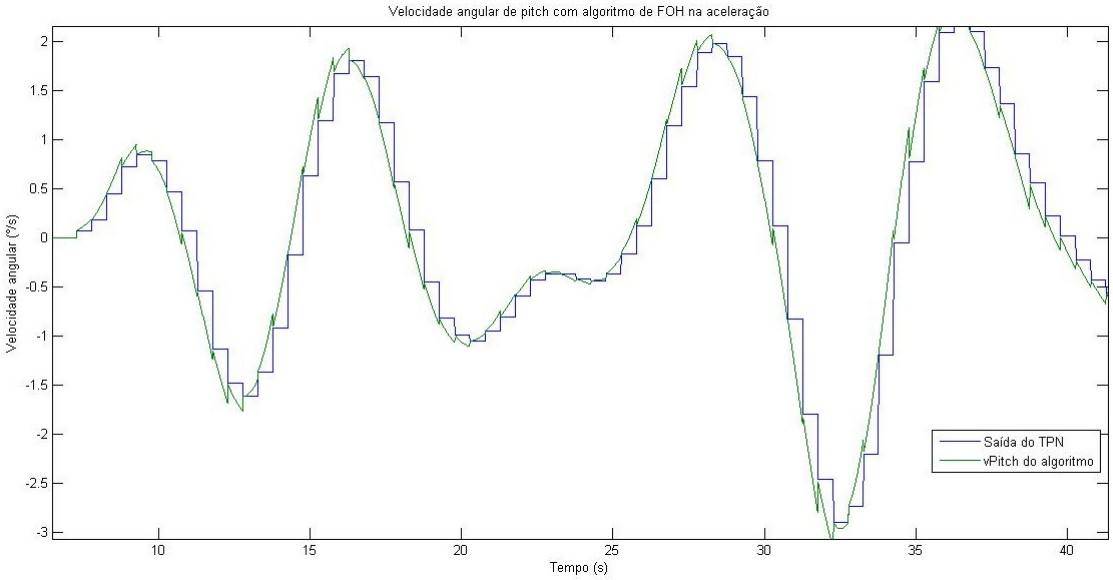

Figura 7.15: Velocidade angular de Pitch obtido com algoritmo de First Order Hold aplicado à aceleração

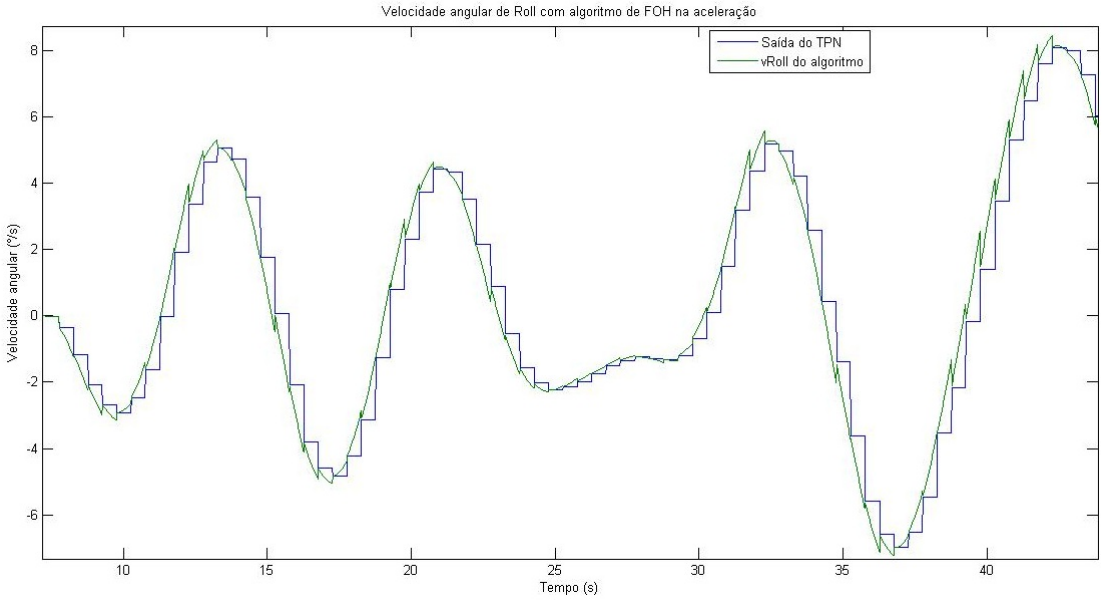

Figura 7.16: Velocidade angular de Roll obtido com algoritmo de First Order Hold aplicado à aceleração

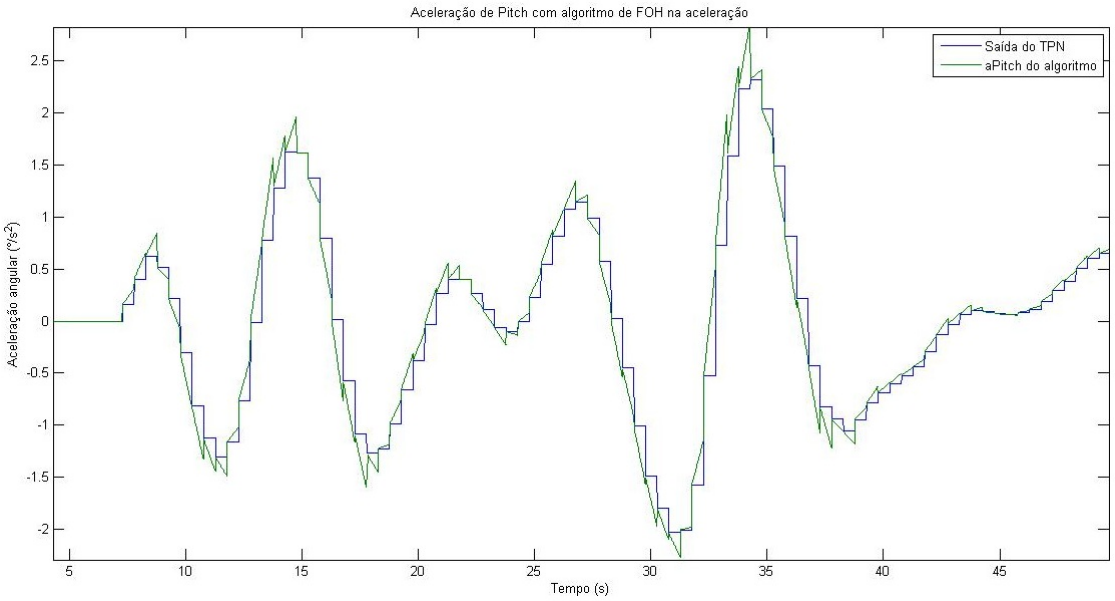

Figura 7.17: Aceleração angular de Pitch obtido com algoritmo de First Order Hold aplicado à aceleração

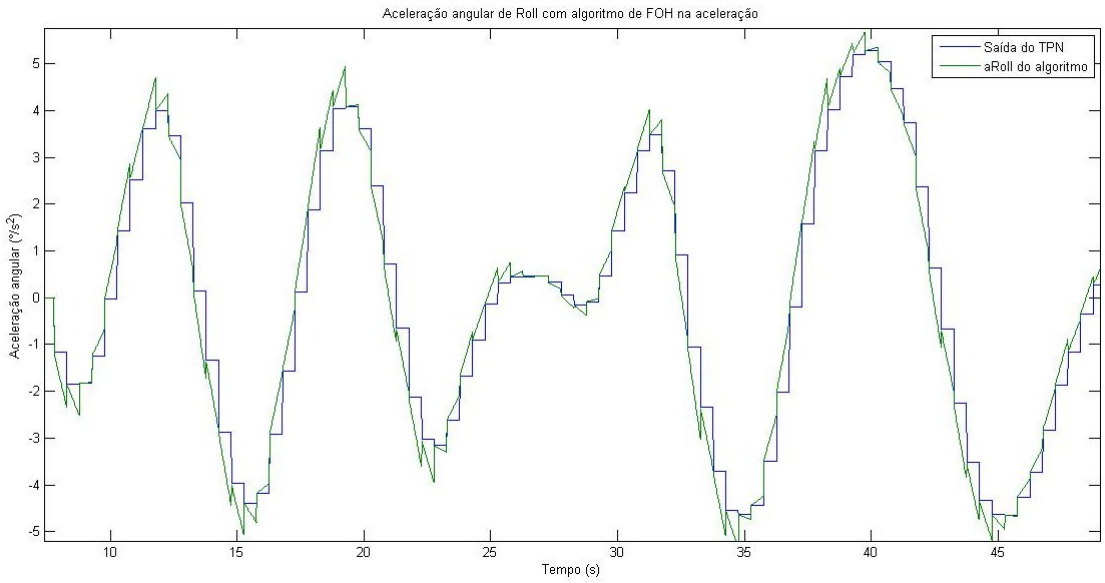

Figura 7.18: Aceleração angular de Pitch obtido com algoritmo de First Order Hold aplicado à aceleração

Capítulo 8

Resultados com o algoritmo escolhido

8.1 Escolha do algoritmo apropriado

A escolha do algoritmo apropriado para movimentação da plataforma é baseada em dois requisitos principais: o tempo de cálculo para não exceder a janela prevista para o envio de dados à plataforma e, cumprindo o primeiro requisito, aquele que reproduza o mais fielmente possível o movimento gerado pelos *clusters* do TPN.

Além dos algoritmos expostos nas seções precedentes, outros algoritmos de extrapolação também foram testados. Em especial, é importante ressaltar que o método de amortização dos resíduos, utilizado a partir do algoritmo de *First Order Hold* modificado do capítulo anterior foi testado também para a velocidade. No entanto, a integração desse termo residual tornava as séries temporais de velocidade simuladas extremamente instáveis e as novas posições divergiam.

Desta forma, foram descartados algoritmos com a correção proposta na velocidade ou na aceleração e a solução selecionada deveria conter apenas a correção residual na posição para evitar efeitos de deriva.

Após isso, foi feita uma análise de quanto tempo a CPU levava para executar os cálculos dos diversos algoritmos e foi constatado que as rotinas para construção e envio de mensagens UDP era muito mais custosa computacionalmente que o algoritmo em si. Contudo, mesmo adicionando os tempos construção e envio de mensagem via UDP, recepção de mensagens via NMEA e cálculo dos algoritmos, o tempo livre da CPU ainda era muito grande, provando que este requisito de projeto poderia ser tranquilamente satisfeito, ainda que os cálculos se tornassem mais complexos.

A temporização a ser respeitada era de 16,6ms para efetuar todos os cálculos descritos acima. No entanto, o algoritmo mais custoso computacionalmente - o

de integração das velocidades com cálculo do resíduo de posição - revelou uma média de tempo de cálculo (feita utilizando a rotina de temporização desenvolvida) de: $E(x_{int\_vel}) = 0,090ms$ com um desvio padrão de $\sigma_{int\_vel} = 0.33ms$ e com o maior valor encontrado na simulação de $max\{x_{int\_vel}\} = 3,471ms$.

Um resumo da análise estatística feita com os algoritmos é mostrada na tabela abaixo (espaço amostral de aproximadamente 10.000 amostras).

Método	Média (μs)	Desvio Padrão (μs)	Maior Valor (μs)
ZOH	86,16	332,9	2679
FOH	86,91	330,7	2475
FOH modificado	87,0	332,2	2874
Integração velocidades	89,8	339,8	2949
Integração acelerações	90,4	340,5	3392
FOH na aceleração	89,5	338,0	3471

Tabela 8.1: Análise estatística da temporização dos algoritmos de extrapolação

Desta forma, com todos os algoritmos passando pelo teste de tempo de execução, a escolha fica reduzida aquele com melhor reprodução do movimento simulado nos *clusters* do TPN. De acordo com os gráficos expostos anteriormente neste capítulo, a escolha é o algoritmo de *First Order Hold* aplicado às acelerações recebidas do TPN.

Uma comparação entre os algoritmos escolhidos pode ser vista na tabela **??**, em especial, pode-se notar o grau das funções que interpolam as séries de posição, velocidade e aceleração utlizadas.

8.2 Análise dos algoritmos no espectro da frequência

Nesta seção, será mostrada a análise dos algoritmos no domínio da frequência. A figura 8.1 mostra o espectro de frequência da série contínua original de uma senoide (em azul) e do algoritmo de *Zero Order Hold*. Pode-se notar que o algoritmo de extrapolação adiciona um ruído na frequência de recebimento dos dados (2Hz) e seus múltiplos inteiros. O objetivo dos algoritmos desenvolvidos aqui, então, pode ser interpretado como a tentativa de reduzir ao máximo este ruído.

As figuras seguintes 8.2, 8.3 e 8.4 mostram a evolução dos algoritmos, reduzindo cada vez mais o ruído indesejado e se aproximando muito do espectro obtido

Método	Continuidade x	Grau x	Grau $\dot{x}$	Grau $\ddot{x}$	Satisfaz tempo?
ZOH	Não	0	Não usa	Não usa	Sim
FOH	Não	1	Não usa	Não usa	Sim
FOH modificado	Sim	1	Não usa	Não usa	Sim
Integração velocidades	Sim	1	0	Não usa	Sim
Integração acelerações	Sim	2	1	0	Sim
FOH na aceleração	Sim	3	2	1	Sim

Tabela 8.2: Análise comparativa entre os algoritmos descritos

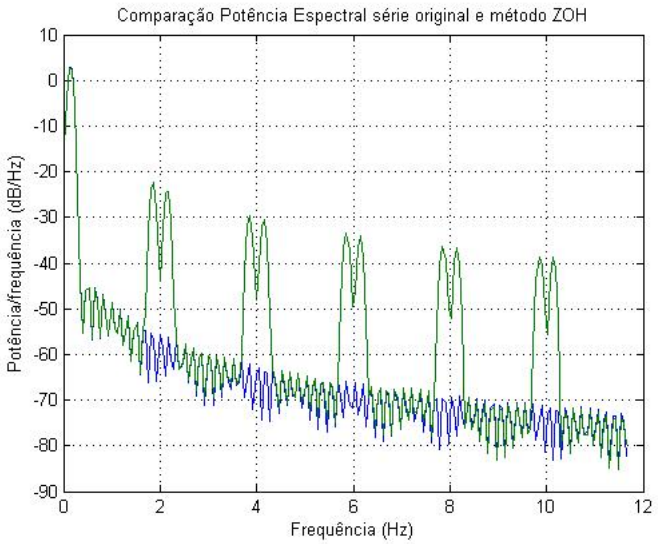

Figura 8.1: Comparação entre série senoidal e extrapolação com Zero Order Hold

para a série temporal contínua inicial.

8.3 Simulação do algoritmo escolhido na plataforma

Nesta seção, são reproduzidas algumas imagens da simulação feita no laboratório Tanque de Provas Numérico (TPN) da Escola Politécnica da USP. O autor do trabalho testou as séries temporais enviadas e os requisitos de temporização, segurança e qualidade do movimento.

As figuras 8.5 e 8.6 abaixo mostram os testes feitos:

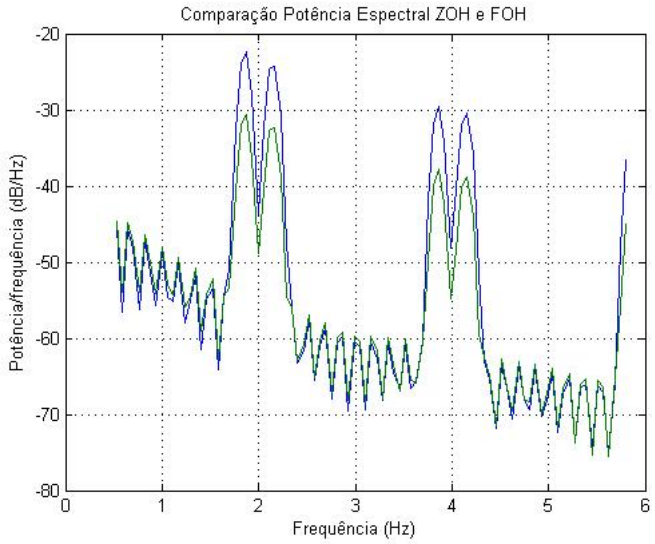

Figura 8.2: Comparação entre extrapolação com Zero Order Hold e com First Order Hold

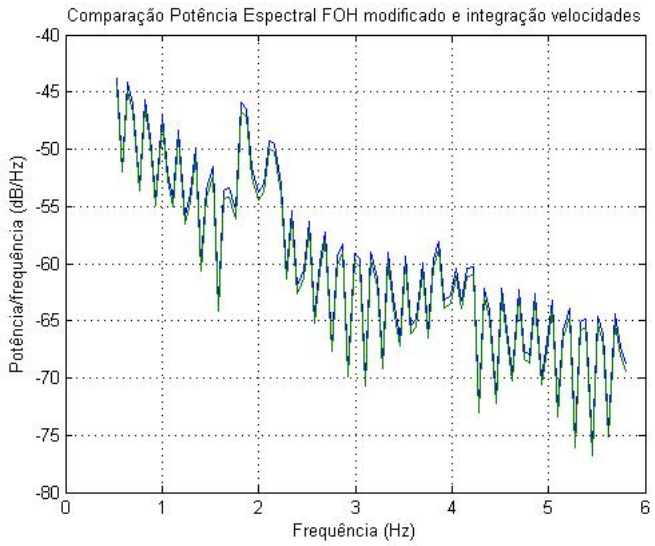

Figura 8.3: Comparação entre First Order Hold Modificado e Integração de Velocidades

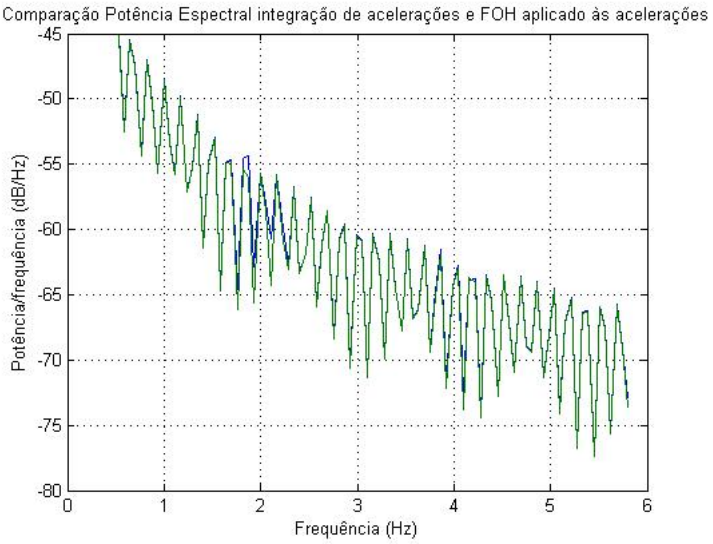

Figura 8.4: Comparação entre integração de Acelerações e algoritmo final

Figura 8.5: Fotografia de instante de máxima rolagem da plataforma em simulação

Figura 8.6: Teste da plataforma com passageiro embarcado

Capítulo 9

Conclusão

Este trabalho de conclusão de curso foi desenvolvido no laboratório Tanque de Provas Numérico (TPN) da Escola Politécnica da USP. O objetivo foi criar um ambiente imersivo de simulação capaz de integrar duas funcionalidades já desenvolvidas no laboratório: a simulação de processos complexos de movimentação de corpos flutuantes sob diversas condições climáticas e a geração de uma visualização da movimentação destes corpos flutuantes simulados.

Para isto, foi adquirida uma plataforma do tipo *Hexapod* capaz de mover-se em seis graus de liberdade. O trabalho, então, era o de criar duas interfaces: uma com a plataforma e outro com o *software* desenvolvido e desenvolver um *software* intermediário para o tratamento dos dados recebidos do TPN para envio à plataforma.

A interface com o *software* TPN requeria uma comunicação com protocolo NMEA a uma frequência de 2Hz. Já a comunicação com a plataforma requeria comunicação *socket* usando protocolo UDP e uma frequência de 60Hz. A diferença nas frequências de envio e recebimento criou a necessidade de se desenvolver um algoritmo de extrapolação da série temporal recebido e nova discretização para envio à plataforma, com o menor prejuízo possível para a qualidade da série temporal.

Para este fim, foram testados diversos algoritmos de extrapolação, iniciando por algoritmos mais simples como o *Zero Order Hold* e o *First Order Hold* até algoritmos mais complexos, que calculavam o resíduo entre a posição esperada e a posição atual e reduziam este resíduo nos próximos passos de simulação além de integrar a aceleração e velocidades obtidas para extrapolação fiel ao movimento esperado na janela de tempo entre o recebimento de novos dados.

A escolha do algoritmo foi baseada no tempo de cálculo de cada um e na capacidade de reproduzir o movimento real do corpo flutuante obtido através da simulação nos *clusters* do TPN. Foi feito um quadro comparativo entre os algoritmos e uma análise estatística do tempo de execução de cada um deles para auxiliar a decisão

sobre qual algoritmo era mais adaptado ao problema descrito neste trabalho.

Para o cálculo da janela temporal entre o envio dos dados, foi desenvolvida uma rotina de temporização especial. Esta rotina é capaz de manter a CPU ocupada e aguardar, com boa precisão, o momento para envio dos dados. Uma simples análise estatística foi realizada para mostrar que a rotina era eficiente e atingia os requisitos pré-estabelecidos, mesmo utilizando um sistema não desenvolvido especialmente para aplicações *real time*, o Windows 7. Para obter os dados de tempo de envio, foi utilizado o analisador de protocolos de rede *Wireshark*.

Finalmente, o envio de dados foi realizado utilizando o protocolo UDP, simples e leve, ideal para situações nas quais o respeito a requisitos temporais é muito importante e onde a perda de um pacote não afeta drasticamente a performance do sistema, já que o protocolo não é 100% confiável.

Os principais trechos de códigos utilizados foram transcritos neste relatório. Os resultados obtidos foram muito bons. A série temporal gerada atingia os pré-requisitos estabelecidos nos capítulos iniciais deste trabalho, o movimento final gerado na plataforma foi muito semelhante a aquele produzido pelas simulações nos *clusters* do TPN apesar da baixa frequência de envio de dados, provando que os algoritmos de extrapolação foram eficientes.

Capítulo 10

Trabalhos futuros

Alguns tópicos podem ser melhorados em trabalhos futuros em continuação a este trabalho de conclusão de curso, eles serão discutidos abaixo:

- **Troca de sistema operativo:** Neste trabalho, foi usado um sistema que não foi desenvolvido especialmente para sistemas *real-time*, o Windows. Na tentativa de sanar as deficiências do Windows em relação à temporização, foi desenvolvida uma rotina especial que utiliza código em *assembly* para comunicar com o *clock* do processador. Caso um sistema operacional mais adaptado à aplicações *real-time* fosse utilizada, a temporização poderia ser garantida com maior segurança e maior precisão.
- **Estudo de algoritmos alternativos:** Outros algoritmos de extrapolação poderiam ser desenvolvidos. O foco principal poderia ser a garantia de continuidade tanto na posição quanto na velocidade e na aceleração da série temporal a ser enviada à plataforma.
- **Aplicação das técnicas para outros graus de liberdade:** Seguindo a mesma técnica, outros graus de liberdade também podem ser reproduzidos pela plataforma. Um cuidado especial deve ser tomado para a plataforma *Hexapod*, pois com altos valores de *Heave*, o curso para *Roll* e *Pitch* fica sensivelmente reduzido.

Referências

[1] FUCATU, C. H. **Proposta para Participação no Programa de Inovação Tecnológica em Pequenas Empresas**. São Paulo, 2007.

[2] HORTON, I. **Beggining Visual C++ 6**. p. 1224, 1998.

[3] **National Marine Electronics Association: NMEA data information**. http://www.nmea.org. 9 10 2012.

[4] FRANKLIN, G. F.; POWELL, J. D. **Digital Control of Dynamic Systems (3rd Edition)**. p. 850, 1997.

[5] STEVENS, W. R. **Unix Network Programming, volume 1**. p. 1009, 1998.

[6] FRANçA, L. N. F.; MATSUMURA, A. Z. **Mecânica Geral**. São Paulo, p. 256, 2004.

[7] TANNURI, E. A. **Desenvolvimento de metodologia de projeto de sistema de posicionamento dinâmico aplicado a operações em alto mar**. São Paulo, 2001.

[8] QUINN, B.; SHUTE, D. **Windows Sockets Network Programming**. p. 656, 1995.

[9] CHAPPELL, L. **Wireshark Network Analysis - Second Edition: The Official Wireshark Certified Network Analyst Study Guide**. p. 226, 2012.

[10] MILONE, G. **Estatística Geral e Aplicada, 1a. Edição**. p. 498, 2003.

[11] TANNURI, E. A.; FUCATU, C. H.; MAKIYAMA, H. S.; TANIGUCHI, D.; RATEIRO, F.; MASETTI, I. Q. **Development of an Innovative Real-Time Simulator for DP-Shuttle Tanker/FPSO Offshore Connection Operation**. *OMAE2012*, July 2012.