LUCIANO ONDIR FREIRE

SIMULADOR EM TEMPO REAL DE UM VEÍCULO SUBMARINO

São Paulo
2010

LUCIANO ONDIR FREIRE

SIMULADOR EM TEMPO REAL DE UM VEÍCULO SUBMARINO

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro.

São Paulo
2010

LUCIANO ONDIR FREIRE

SIMULADOR EM TEMPO REAL DE UM VEÍCULO SUBMARINO

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro.

Área de Concentração: Engenharia Mecatrônica e Sistemas Mecânicos.

Orientador: Prof. Dr. Ettore Apolônio de Barros

São Paulo

2010

FICHA CATALOGRÁFICA

Freire, Luciano Ondir
Simulador em tempo real de um veículo submarino/L.O. Freire. – São Paulo, 2009. 69 p.

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

1.Simulação em Tempo Real 2.Veículos Submarinos. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t.

RESUMO

Este trabalho busca desenvolver um simulador em tempo real para um veículo submarino a partir de um simulador Hardware-in-the-Loop para a navegação e controle de um veículo autônomo submarino desenvolvido em [1], agregando mais um modelo matemático [2] e criando uma estrutura de software flexível o suficiente para permitir a aplicação em diversas necessidades com um mínimo de esforço por parte de um programador. Para diminuir o esforço de desenvolvimento e de posterior manutenção foi usada principalmente uma linguagem gráfica, a saber, MatLab Simulink. Para conferir flexibilidade foi adotada uma estrutura de software em camadas, separando a solução das equações de movimento, a interface com sensores e atuadores reais e a interface com o usuário. As aplicações previstas são o apoio à decisão durante o projeto de um submarino, apoio à identificação de parâmetros hidrodinâmicos em ensaios com modelos em escala, apoio ao projeto de controladores (piloto automático) e aplicação em simuladores para treinamento de pessoal.

ABSTRACT

This work seeks to develop a real-time simulator for a underwater vehicle from a Hardware-in-the-loop simulator for an autonomous underwater vehicle developed in [1], adding a matemathical model [2] and creating a software structure flexible enough to allow the application of the simulator in many cases with minimum efforts of a programmer. In order to reduce the development and a posteriori upkeep efforts, it was used mainly a graphical language, the Matlab Simulink. To achive the necessary flexibility it was adopted a layered software structure, separating the movement equations resolution, the sensors and actuators interface and the human machine interface. The possible applications are decision support during project of a underwater vehicle, support to parameter identification experiments with scale models, controlator design support and personnel training facilities.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADC	Conversor analógico-digital
ARM7	*Advanced RISC Machines* versão 7
A/D	Analógico-digital
CAN	Rede de área para controle
CI	Circuito integrado
DAC	Conversor digital-analógico
D/A	Digital-analógico
FAT	Tabela de alocação de arquivos. *File Allocation Table*
HILS	*Hardware-in-the-loop simulation*
IHM	Interface homem-máquina
LIPO	Lítio-polímero (bateria)
PCB	Placa de circuito impresso
PWM	Modulação por largura de pulso
RF	Rádio freqüência
RPM	rotações por minuto
RTW	*Real-Time Workshop*
TTL	Lógica transistor-transistor
RC	Rádio controle
RF	Rádio Frequência
SMD	Dispositivo montado sobre superfície
SPI	Interface serial com periféricos

1 INTRODUÇÂO

Dado o objetivo assumido pelo País de construir submarinos, incluindo aqueles de propulsão nuclear, uma série de conhecimentos e tecnologias torna-se necessária. Uma delas, imprescindível ainda na fase de projeto conceitual de um submarino (não necessariamente nuclear) é prever o comportamento dinâmico das várias alternativas propostas de projeto e verificar o atendimento aos requisitos. Em uma fase intermediária de projeto está o desenvolvimento de algoritmos de controle para o submarino projetado. Outra necessidade, em uma fase posterior, é a existência de instalações para treinamento de pessoal, tais como simuladores.

Todas as necessidades acima podem ser satisfeitas com um simulador em tempo real de um veículo submarino. Para prever o comportamento e realizar projeto de controladores não é indispensável o uso de simuladores em tempo real, apesar de ser um recurso valioso, pois permite a visualização tridimensional do fenômeno físico, dando uma noção intuitiva, porém bastante útil do comportamento do veículo ou da validade do modelo.

A utilização de simuladores é uma técnica bastante diversificada na área de engenharia, tanto academicamente quanto na indústria, pois a utilização de simuladores permite realizar muitos testes em um sistema a um custo muito menor do que o uso de modelos físicos. Durante as iterações de projeto de um submarino um simulador encontra um uso muito importante que é a verificação das especificações de projeto em relação à manobrabilidade do submarino inicialmente projetado.

Entretanto, uma vez que submarinos são extremamente dispendiosos tanto financeiramente como em termos de pessoal e tempo, o uso de modelos físicos após certa fase do projeto para refinar as estimativas das derivadas hidrodinâmicas não pode ser prescindido. Foi observado que o tempo de ensaio de um modelo físico também não é muito barato, e requer bastante trabalho de pós-processamento. Considerando que tais modelos são relativamente complexos e usam muitos sensores, existe sempre a possibilidade de haja falha (dano ou descalibração) de pelo menos um deles durante o manuseio.

Se não houver uma realimentação da coerência dos dados obtidos durante o ensaio, existe sempre a possibilidade de ser constatada no pós-processamento dos

dados uma falha nos sensores ou no modelo que comprometa todo o ensaio. Dessa forma, viu-se que, mesmo o ensaio de um modelo físico fica enormemente facilitado e enriquecido se comparado com simulações em tempo real.

Uma vez que a forma final do submarino a ser construído está estabelecida, faz-se necessário fazer o projeto do controlador, de maneira a possibilitar a manutenção de cota e de rumo, não só a baixa profundidade, onde existe a ação de ondas, como a alta profundidade. Nesse sentido, simuladores são requeridos para fazer o ajuste das constantes do controlador e verificar o atendimento às especificações.

Por fim, durante a fase de construção do submarino é necessário preparar o pessoal para fazer parte da tripulação. Para isso são necessários simuladores de grande porte que emulem o comportamento dinâmico do submarino em questão. Tais simuladores requerem um simulador em tempo real capaz de se comunicar com atuadores de grande porte e receber comandos da tripulação para dar aos alunos uma experiência realista do comportamento do submarino, preparando-os da melhor maneira possível para a operação do meio.

Este trabalho busca estabelecer uma arquitetura genérica e extensível para satisfazer todas as necessidades ora vistas da Marinha do Brasil. Primeiramente, será desenvolvido um simulador de manobras com a finalidade de prever a manobrabilidade de um projeto de submarino. Nessa parte, o esforço de desenvolvimento será basicamente a resolução das equações diferenciais de movimento. Em seguida será feito um protótipo de um sistema para investigação de derivadas hidrodinâmicas que seguirá uma arquitetura capaz de ser incorporada em sistemas de *simulação hardware-in-the-loop*, simuladores de grande porte e veículos autônomos.

2 OBJETIVOS

O presente trabalho busca propor uma estrutura genérica e expansível para atender as necessidades das várias aplicações de um simulador e fazer duas implementações da mesma com aplicações distintas. Uma para simulação de manobras e outra para auxílio a ensaios.

2.1 Proposição de arquitetura

O primeiro objetivo desse trabalho é propor uma arquitetura de software de simulação em tempo real capaz de atender as múltiplas necessidades da Marinha do Brasil. Tal arquitetura precisa ser capaz de possibilitar ou não a simulação em tempo real, precisa ser capaz de interagir com sensores e atuadores, e ter muita flexibilidade para apresentar os resultados da simulação.

Para que a simulação em tempo real seja garantida, uma série de cuidados deve ser tomada, partindo do sistema operacional, que necessariamente deve ser de tempo real. Vale lembrar que um software de tempo real não deve atender apenas a prazos temporais, mas deve também atender requisitos de simultaneidade, previsibilidade e robustez [30]. Para que isso seja garantido, o uso de ferramentas específicas é quase imprescindível.

A interação com sensores e atuadores também é uma tarefa bastante complexa, visto que existem diversos protocolos de comunicação e a necessidade de um meio para que seja feita comunicação (ethernet, RS232, por exemplo). Logo, para que um software seja capaz de se comunicar com um número indeterminado de sensores e atuadores, o uso de redes de dados se torna quase obrigatório em muitos casos.

Por fim, a apresentação dos dados pode variar muito em forma e tipos. Para fazer simulações para determinação do raio de giro, por exemplo, uma simulação normal pode ser usada, e a apresentação deve ser feita na forma de gráficos. No caso de simuladores de grande porte, faz-se necessário a simulação em tempo real e a apresentação dos dados deve ser feita de maneira similar à dos indicadores de sensores usados a bordo, de maneira a adestrar o pessoal em condições realistas.

Para a solução de todos os problemas listados acima, a proposição inicial é uma arquitetura em camadas, de maneira que, para cada aplicação, sejam necessárias apenas pequenas mudanças no software e hardware.

2.2 Implementação de um simulador de manobras

O segundo objetivo é implementar um simulador de veículos submarinos para apoio a decisão durante a fase de projeto de um submarino seguindo a estrutura proposta acima. Para manter a compatibilidade com modelos já amplamente utilizados, foi implementado o modelo de [2]. Para reduzir o esforço de desenvolvimento e adicionar confiabilidade ao simulador, partiu-se de um simulador *hardware-in-the-loop* [1] já testado. Esse simulador busca a solução de equações de movimento de um veículo submarino de massa constante em um meio infinito e estático, ou seja, na ausência de efeitos de superfície ou de fundo, de efeitos de onda ou efeitos de alagamento ou esvaziamento de tanques.

Entretanto, a inclusão de tais variáveis é desejável para apoiar a decisões de projeto, e, uma vez que esse simulador visa ter um grau de expansibilidade, foi seguida uma estrutura modular na qual existe a possibilidade de adicionar tais características com pouco esforço de desenvolvimento.

Nessa implementação o esforço de desenvolvimento principal fica em torno da camada de solução de equações diferenciais.

2.3 Implementação de um sistema de identificação de derivadas

O terceiro objetivo é aplicar a mesma estrutura a um sistema de identificação de derivadas hidrodinâmicas composto de um veículo submarino e uma estação base. Trata-se de um aperfeiçoamento de um sistema já pré-existente com o mesmo objetivo. Serão introduzidas modificações com o objetivo de tornar o sistema mais confiável e flexível, de modo a suportar melhor o ambiente marinho, ser capaz de integrar um número indefinido de sensores e atuadores.

Esse sistema deve permitir a interação entre usuário e a plataforma que simula um veículo submarino autônomo, integrar atuadores e sensores e fazer um armazenamento de dados redundante. Uma vez que o veículo deve se deslocar em um

meio líquido, e ondas de rádio sofrem elevada atenuação na água, será usado um cabo para controle remoto ligado a uma bóia. Na implementação do sistema de identificação dinâmica a maior parte do esforço será nas camadas de fluxo de dados e interface com o usuário.

Esse sistema não é exatamente um simulador no sentido estrito de que haja uma solução de equações diferenciais por meio de software, mas faz uso de uma estrutura pensada para ser usada em um ambiente de simulação interativo. Dessa forma, mostra-se a adaptabilidade da estrutura proposta e sua ampla aplicabilidade. Fica, por outro lado, implementado todos os elementos de um simulador genérico, a saber, a solução de equações diferenciais, interface com o mundo físico através de sensores e atuadores e a interface com o usuário através de uma interface gráfica.

3 REVISÃO BIBLIOGRÁFICA

3.1 Simulação Numérica

Primeiramente foram buscados os simuladores de manobra de submarinos existentes. Uma vez que não existem muitos simuladores do gênero publicados, foi incluído na busca qualquer simulador com algum grau de similaridade funcional a um simulador de manobra de veículos submarinos. Foi verificado que os simuladores mais antigos foram todos escritos em Fortran e geralmente faziam a simulação numérica, ou seja, toda a trajetória do veículo era exibida na forma de um gráfico. Não podemos, no entanto, dizer que o uso do Fortran tenha sido encerrado. Ele ainda aparece em publicações recentes, ainda que ao lado de ferramentas mais modernas, como Maple [3].

Tabela 3.1 Ferramentas usadas para simulação de manobras de veículos submarinos

Ano	Plataforma simulada	Ferramentas	Ref.
1990	Drone (veículo sob controle remoto)	Fortran	[4]
1992	AUV	ANSI C, Prolog, Lisp e CLIPS	[5]
2000	Theseus AUV	Não menciona	[6]
2000	Submarinos	Fortran	[7]
2000	microAUV	ANSI C	[8]
2007	Submarinos	Maple, Fortran 77	[3]

3.2 Simulação em tempo real

De maneira geral, simulação em tempo real é usada para verificar o funcionamento de dispositivos físicos, simulando por meio de software o comportamento de um sistema, recebendo entradas de um controlador (hardware e software) em fase de desenvolvimento e fornecendo as saídas para o mesmo, mesclando nessa transmissão de informações elementos de hardware e software. O objetivo final é ter um software e hardware totalmente validados e prontos para a aplicação no sistema final sem por em risco os desenvolvedores ou uma plataforma de desenvolvimento, além de economizar tempo. A esse tipo de simulação dá-se o nome de *hardware-in-the-loop* (HILS ou HIL). Vale lembrar que um simulador para treinamento de pessoal tem uma grande semelhança com esse conceito, mudando

apenas o fato de que o controlador é o ser humano. Devido a isso, foram pesquisadas obras que tratam de HILS com ênfase em veículos submarinos.

Tabela 3.2 Ferramentas usadas para simulação hardware in the loop

Ano	Plataforma simulada	Ferramentas	Ref.
2001	AUV	HLA, CORESIM	[9]
2003	AUV	Linux, QNX	[10]
2003	UAV	RMUS, CommLibX	[11]
2006	UAV	MatLab Simulink e dSPACE	[12]
2008	AUV	MatLab Simulink	[1]
2008	UUV	MatLab Simulink e Constellation	[13]
2009	AUV	MatLab Simulink	[14]

Nota-se a uma tendência nos trabalhos mais recentes ao uso do MatLab Simulink. Segundo [15], havia na época (2002), cinco fabricantes diferentes de ferramentas para desenvolvimento e simulação de sistemas embarcados, listados abaixo.

Tabela 3.3 Ferramentas disponíveis no mercado para fazer simulação hardware in the loop

Vendor	Ferramenta	Suporte
Opal-RT	Ferramentas para desenvolvimento de algoritmo e implementação rápida	MatLab Simulink e Matrixx Autocode
XANALOG	Computador para prototipar controladores	MatLab Simulink Real-Time Workshop
Applied Dynamics International	Computador para simulação em tempo real	MatLab Simulink, Matrixx Autocode, C, Fortran, EASY 5 e ADSIM
Quanser Consulting	Software e hardware para criar um ambiente de desenvolvimento integrado de algoritmos de controle	MatLab Simulink Real-Time Workshop
dSPACE Inc.	Software e hardware para criar um ambiente de desenvolvimento integrado de algoritmos de controle	MatLab Simulink e dSPACE

Observa-se que o uso ou a compatibilidade com o Simulink está presente em todas as ferramentas, de modo que, para desenvolvimento de sistemas embarcados e sua simulação, o uso do Simulink e Real Time Workshop desponta quase que como a ferramenta padrão.

Em acordo com o ponto de vista acima, o autor de [16] defende o uso de HILS para redução dos custos de desenvolvimento de sistemas de tempo real. As ferramentas que este autor propõe são o MatLab Simulink e o dSPACE. Vale ressaltar que ele defende também o uso dessas ferramentas para automatizar a geração de código e eliminar restrições impostas por código feito à mão em sistemas críticos. Outra vantagem é a possibilidade de fazer testes do sistema em condições extremas, nas quais seria muito caro usar protótipos.

Devido a essas possibilidades, HILS permite a redução do tempo de desenvolvimento de sistemas complexos, diminuição dos custos e o aumento da confiabilidade.

3.3 Hardware embarcado

Uma vez que um dos objetivos desse trabalho é a implementação de um sistema de identificação de derivadas hidrodinâmicas, foi também pesquisado o atual estado da arte em relação à arquitetura de hardware e software usada em veículos submarinos, notadamente, em veículos autônomos, pois essa é a linha de pesquisa do laboratório e o objetivo final dos trabalhos desenvolvidos até então. Logo, é bastante desejável desenvolver sistemas apropriados à aplicação em veículos autônomos.

Tabela 3.4 Arquitetura usadas em AUVs

Ref.	data	arquitetura	Sistema Operacional	Rede
[17]	1994	centralizada	VxWorks	LONTalk
[18]	1999	Não informado	Não informado	CAN
[19]	2000	descentralizada	Não informado	CAN
[20]	2001	descentralizada	QNX	Ethernet
[21]	2005	descentralizada	Não informado	I^2C, Ethernet
[22]	2006	centralizada	RT-Linux	I^2C, ethernet
[23]	2006	descentralizada	Não informado	CAN
[24]	2006	centralizada	Windows XP	Ethernet
[25]	2007	centralizada	LINUX	I^2C
[26]	2007	centralizada	Não informado	Ethernet
[27]	2008	centralizada	µC/OS II	CAN

Nessa pesquisa foram encontrados 11 artigos que descrevem sistemas similares ao presente no laboratório e verificou-se que a maioria (pelo menos seis dentre os onze) usa uma arquitetura centralizada, ou seja, um único computador faz todas as tarefas, incluindo interface com sensores e atuadores. Por outro lado, existem também várias implementações de arquiteturas descentralizadas, ou seja, em que há vários módulos de baixo poder de processamento fazendo tarefas distintas e conversando entre si por meio de uma rede de dados.

Quanto aos sistemas operacionais, houve uma grande variedade, não havendo uma tendência mais forte por parte dos autores. Chamou a atenção o sistema operacional chamado µC/OSII, que pode ser embarcado em microprocessadores de pequeno porte, atende requisitos de tempo real e possui certificações de segurança aeronáutica. Além disso, de acordo com [27], é bastante fácil de ser configurado e usado, e pode ser usado gratuitamente por universidades para fins de pesquisa e ensino.

O uso de rede de dados ficou bastante concentrado em três padrões: ethernet (cinco autores), CAN (quatro autores) e I^2C (três autores). Deve ser feita a ressalva de que dois autores que empregam barramento I^2C o fazem para sistemas de teste em bancada. Apenas em [25] temos um veículo destinado à operação no mar a 100 metros de profundidade que usa esse barramento.

Foi notado que em vários trabalhos [18][19][22][24][25], houve a aplicação de microprocessadores para inicializar e configurar dispositivos, converter dados e filtrar entradas, independentemente da arquitetura empregada.

Os autores de [18] e [19] afirmam que o uso de redes CAN aumenta a confiabilidade de um sistema se comparado com o uso de outras redes de dados e leva a uma redução no preço do sistema, bem como no espaço ocupado pelos cabos de rede.

O autor de [21] afirma que o uso de ethernet apresenta como desvantagem o grande número de cabos e o espaço requerido para o uso do hub. Esse problema ganha proporções maiores quando se pensa em sistemas redundantes.

4 METODOLOGIA

Dadas as múltiplas necessidades de simulação, é necessário desenvolver uma estrutura de software que permita que o simulador venha a ter, em versões futuras, alto grau de confiabilidade, flexibilidade e facilidade de manutenção. Para isso, deverá ser estruturado de maneira modular, o que sugere a separação em camadas.

Por um lado existe a necessidade de desenvolvimento de um software para simulação de submarinos. Por outro lado, existe também a necessidade de desenvolvimento de software embarcado no veículo submarino. Para facilitar o reuso e a confiabilidade dos sistemas desenvolvidos, a estrutura desenvolvida deve permitir o uso em ambas as aplicações.

Tais aplicações possuem diferenças, porém, possuem também grandes similaridades. Ambas devem usar ferramentas específicas de tempo real por estar envolvendo sistemas críticos cuja falha pode resultar em perdas materiais e humanas. Ambas tratam do mesmo sistema, tendo várias variáveis e usuários comuns. Além disso, de maneira geral, estarão em ambientes comuns, que é a presença de atmosfera úmida e salina, pois, usualmente simuladores de treinamento de pessoal ficam próximas ao mar e veículos submarinos, em sua maioria, são usados no mar.

Isso requer, por exemplo, especial cuidado com os circuitos eletrônicos que compõem o sistema, pois a umidade e a contaminação iônica degradam muito a confiabilidade de sistemas críticos. Por isso, o projeto de *hardware* terá uma série de cuidados com o objetivo de melhorar a robustez do sistema desenvolvido.

Além do tempo real, ambas aplicações requerem uma interface com o mundo físico por meio de sensores e atuadores e uma interface com o usuário, por meio de controles e representações gráficas. Necessitam ainda de alta extensibilidade, pois o número de sensores e atuadores é *a priori*, indefinido.

Pelas razões acima exposta, a linha de desenvolvimento a ser seguida será a de uma arquitetura capaz de atender a ambas aplicações, de modo que tenhamos um único padrão de desenvolvimento para a simulação numérica, para a simulação HIL e para a arquitetura de controle embarcada.

Assim, busca-se a redução do tempo de familiarização dos alunos com a infra-estrutura usada, a redução do número de componentes utilizados, o reuso e a

conseqüente redução no esforço de desenvolvimento juntamente com ganhos na confiabilidade, pois aumenta-se o número de componentes de software e hardware já extensivamente testados em várias aplicações.

Ainda com o intuito de diminuir o tempo com o treinamento de pessoal, será tentado, sempre que possível, fazer uso de ferramentas amplamente difundidas, tais como o MatLab, o Simulink e a linguagem C, pois esse projeto será conduzido com a ajuda de alunos de iniciação científica, que usualmente já tem contato com essas ferramentas. Existem ferramentas ainda mais poderosas, mas sua difusão geralmente é mais limitada, e o tempo de treinamento para que um engenheiro atinja o seu domínio costuma ser mais longo.

Para o desenvolvimento da solução das equações de movimento, será usado o Simulink, pois sua natureza gráfica e modular facilita a implementação, a manutenção e a verificação do código. Possui ainda a possibilidade de transformar modelos criados em software capaz de rodar em tempo real por meio do *Real-Time Workshop* (RTW), que é um conjunto de ferramentas integradas ao Simulink capaz de gerar código para ser executado em um sistema operacional de tempo real e possui ainda vários drivers de diversos tipos de rede, como ethernet e CAN. Isso significa que é possível desenvolver aplicações no ambiente do Simulink que podem funcionar como um software de tempo real que se comunica com diversos sensores ou atuadores por meio de uma rede. Tudo isso com pequeno esforço de implementação de código e com fácil manutenção.

No software embarcado, nos processadores em que não é possível usar o RTW, será usada a linguagem C, seguindo, onde aplicável, a orientação a objetos com os seus pressupostos, a saber, modularidade, encapsulamento de atributos e métodos e visibilidade.

Também, em face da necessidade de clareza e facilidade de manutenção do software criado, buscou-se seguir a filosofia Unix onde aplicável, cujas recomendações julgadas mais importantes são: fazer módulos pequenos que façam apenas uma coisa, que trabalhem em conjunto, com interface bem definida e projetados para uso futuro.

Para facilitar o desenvolvimento robusto do código escrito em linguagem C, foi adotado o padrão MISRA C, que é adotado pela indústria automobilística com o

objetivo de tornar o código mais portável e seguro no contexto de sistemas embarcados críticos. Esse padrão não pode ser sempre seguido à risca, mas permite que sejam feitos pequenos desvios, desde que devidamente documentados. Busca permitir uma fácil análise estática do código através de regras que impõem simplicidade e clareza, além de evitar recursos e práticas que usualmente trazem problemas ao longo prazo, como problemas de diferença entre compiladores, corrupção de memória e fluxo imprevisível da execução do programa.

Seguindo esses padrões e essa arquitetura, espera-se pavimentar uma via sólida que permita o fácil desenvolvimento de simuladores, de modelos livres, de HILS e de veículos autônomos em um ambiente típico de universidade e com possibilidade de alcançar bons níveis de confiabilidade.

5 ARQUITETURA DE SOFTWARE

A estrutura do software adotada será em três camadas, sendo a primeira para a solução das equações diferenciais de movimento em tempo real, a segunda para conversão de protocolos e fluxo de dados e a terceira para a interface com o usuário. A seguir serão apresentadas todas as camadas do sistema com suas respectivas características.

5.1 Camada de simulação

Essa camada funciona de maneira similar a um HILS, ou seja, por um lado recebe as entradas ou comandos para o sistema e por outro disponibiliza as saídas ou reação que o sistema físico a ser simulado mostraria.

Um HILS, como mostrado na figura abaixo, simula por meio de software uma planta real para testar e validar um controlador em termos de hardware e software. Apresenta as vantagens de facilitar o processo de desenvolvimento e diminuir o custo, já que o seu uso dispensa boa parte dos testes com plataformas físicas, além de diminuir o número de acidentes, aumentando a segurança. Não podemos deixar de observar que um simulador de treinamento de pessoal tem uma função semelhante, onde o controlador é o ser humano.

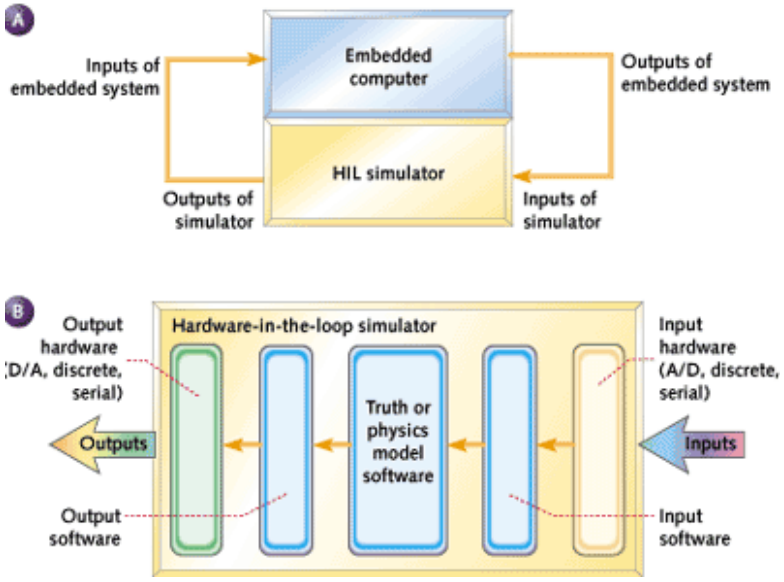


Figura 5.1 Diagrama de blocos de um simulador hardware-in-loop

A camada de simulação é o módulo central do sistema, pois terá como funções:

1 Simulação de sinais de sensores, inserindo ruídos e atrasos;

2 Simulação da dinâmica dos atuadores, inserindo erros e tempos de resposta;

3 Realização de todos os cálculos de dinâmica do veículo;

Operando em tempo real, possui um alto custo computacional, pois a solução das equações de movimento do veículo deve ser obtida em uma freqüência maior do que a do controlador. Isto ocorre devido ao critério de Nyquist, onde as freqüências dos simuladores devem ser pelo menos duas vezes maiores do que a freqüência do controlador, no entanto em casos práticos é recomendável que sejam de cinco a dez vezes maiores [1]. Devido a esse fato, este módulo será usualmente implementado em um computador de alto desempenho. Como neste módulo todas as informações da simulação (sinais e dados) convergem e também de onde todas as informações provêem, este módulo deve possuir um subsistema de comunicação bastante robusto. Este subsistema deve, além de efetuar a transmissão de todas as informações em tempo real (para isso devem ser utilizados protocolos de alto desempenho, como o ethernet), manter todos os módulos sincronizados .

Esse módulo pode precisar atender requisitos de tempo real, ou seja, atender a prazos de execução de tarefas, possuir estabilidade e previsibilidade. A razão é a interatividade com sistemas reais e pessoas, pois um atraso ou um comportamento anormal pode ser danoso ao material e ao pessoal envolvido. Dessa forma, o uso de ferramentas de tempo real, tais como o xPC target do MatLab, para o seu desenvolvimento é muito importante, pois possibilita o rápido desenvolvimento e uma manutenção facilitada se comparada com ferramentas normais, como a linguagem C.

5.2 Camada de fluxo de dados

Esse módulo não necessariamente será executado em um único computador. Dada a grande diversidade de protocolos de rede usados pelos sensores e atuadores nos dias de hoje (por exemplo, RS232, RS485, CAN, ethernet) e interfaces (+/-10V, 4-20mA, PWM-RC, tensão analógica), é necessário uma tradução de todas essas linguagens para um protocolo comum e depois uma conversão para o ambiente de simulação, que será intrinsecamente de tempo real.

Uma vez que o número de atuadores e sensores é, a priori, indefinido, e pode somar um grande número, é recomendável o uso de unidades de baixo custo e poder computacional apenas o suficiente para fazer a conversão dos dados que trafegam por elas. O uso de sensores ou atuadores já integrados com uma rede de dados é bastante recomendável, como usado pela indústria automotiva, que usa equipamentos com capacidade de se comunicar em um barramento CAN. O protocolo comum de comunicação deve ser robusto ao ambiente (resistência a corrosão, umidade, impactos, esforços e vibração), robusto ao ruído, deve possuir uma largura de banda que atenda ao mínimo necessário e de preferência, ser de baixo custo e fácil acesso.

A importância dessa camada está na possibilidade de inserir, no futuro, mais sensores, atuadores ou interfaces com o usuário sem alterar a estrutura já existente.

Podemos fazer uma divisão didática dessa camada em outras subcamadas, a saber, uma subcamada de interface com o mundo físico, uma subcamada de conversão sinais, uma subcamada de processamento de dados, uma subcamada de controle de rede, uma subcamada de barramento de dados e por fim uma subcamada de conversão para o ambiente de simulação.

Na primeira subcamada, temos os sensores, que traduzem uma grandeza física do meio para sinais elétricos, e atuadores, que atuam sobre uma grandeza física. De maneira geral, esses elementos não trocam dados diretamente com um computador devido às relativamente altas tensões ou correntes envolvidas. Tipicamente uma porta lógica pode fornecer até 20 miliamperes a 5 volts, enquanto que o motor de propulsão de um AUV pode consumir 10 amperes a 30 volts. No caso dos sensores, a relação costuma ser inversa. As tensões de saída, geralmente da ordem de milivolts, costumam ser bem menores do que a escala que o conversor analógico-digital possui, que normalmente vai de 0 a 5 volts ou de 0 a 3.3 volts.

Dessa forma, para que haja o tráfego de dados, é necessária a presença de instrumentação e condicionamento de sinal, no caso dos sensores, e de conversores de potência, que transformam sinais de baixa tensão e corrente em sinais capazes de causar uma atuação sobre o meio físico. Esses elementos formam a segunda subcamada.

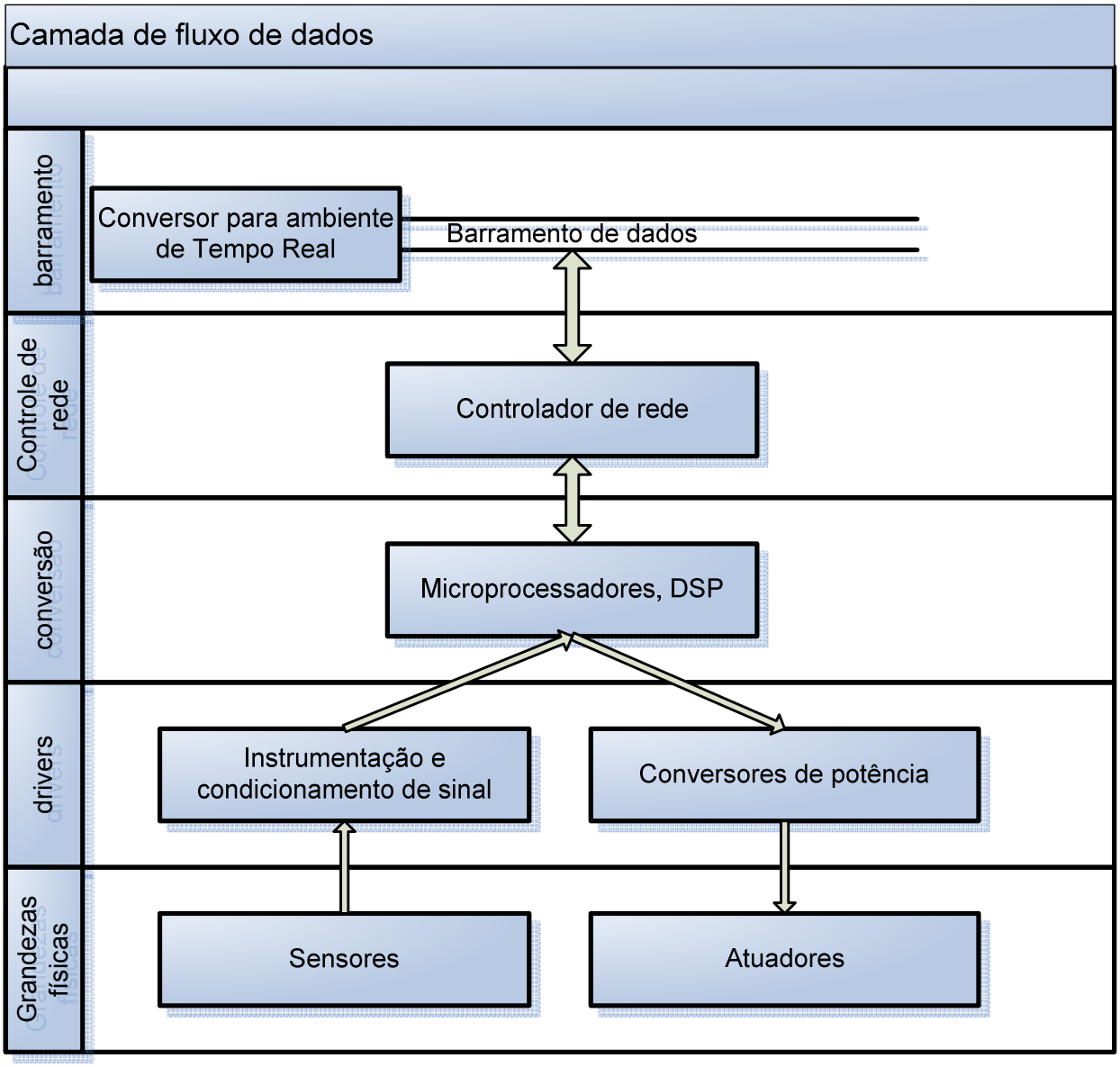


Figura 5.2 Subcamadas da camada de fluxo de dados

Os dados que chegam pela rede vem em um formato estranho aos circuitos de conversão de potência, sendo ainda preciso uma conversão dependente do tipo do atuador. Da mesma maneira, a saída dos condicionadores de sinal ainda está no formato analógico e carece de uma conversão analógico-digital e de um tratamento para ficar em um formato adequado ao tráfego na rede. Todas essas tarefas são feitas por um microprocessador que constitui a subcamada de processamento.

A tarefa de inserir e recuperar dados do barramento da rede não é trivial. Uma vez que existem vários nós na rede com igualdade de direitos, existem potenciais problemas de colisão, que são resolvidos ou não de modo diferente por cada tipo de protocolo. Além disso, podem haver problemas de ruídos causados por fenômenos externos. Tal problema costuma ser atenuado por meio de checagens da transmissão. Esses e ainda uma série de problemas não mencionados, precisam ser resolvidos em um curto espaço de tempo, são geralmente resolvidos por recursos de hardware que constituem a subcamada de controle de rede.

Existem atualmente no mercado muitos sensores e atuadores com mais camadas integradas, ou seja já com a subcamada de conversão de sinais ou até a subcamada controle de rede integrada, o que reduz o esforço de desenvolvimento e pode adicionar maior confiabilidade com um custo financeiro maior e uma incapacidade de substituir apenas o elemento danificado. Isso significa que um sistema atuador possa eventualmente ter que ser trocado devido a uma pequena falha no controlador de rede. Apesar desses inconvenientes, o emprego de sistemas com todas as subcamadas integradas tem sido uma tendência na indústria automotiva, que usa principalmente barramento CAN. Uma vez que tais componentes são produzidos em massa para a indústria automobilística, eles têm boa confiabilidade e baixo custo.

A subcamada de barramento de dados é bastante simples, constituindo-se basicamente de cabos, conectores e demais elementos que permitem a transmissão do sinal com boa qualidade. Entretanto, são as coisas mais simples que podem impedir o funcionamento de um equipamento. Assim é sempre recomendável seguir rigorosamente as especificações do padrão do protocolo escolhido.

Por fim, temos a camada de conversão dos dados do barramento para o ambiente de simulação em tempo real. No caso do xPC Target do simulink, existem já bibliotecas prontas para vários protocolos e drivers para vários equipamentos comerciais, tais como placas controladoras de rede ethernet ou CAN. Obviamente o hardware compatível com as bibliotecas deve ser adquirido.

Nada impede, por exemplo, que essa conversão ser feita por meio de outro protocolo, mas nesse caso, será necessário um maior esforço do programador.

5.3 Camada de interface com o usuário

Essa camada estará estruturada de maneira a possibilitar o controle e visualização de dados provenientes de um veículo a ser simulado na primeira camada.

Tem a função de traduzir, para o usuário humano, os estados do veículo de maneira intuitiva e direta, bem como permitir a interação do usuário e do veículo virtual.

Como o módulo de interface não é um módulo crítico para o sistema de simulação, ele não precisa necessariamente ser implementado em tempo real, pois pequenos atrasos na interface não mudam o entendimento do comportamento do veículo pelo usuário. Pode, por exemplo ser implementado de modo a ser aberto um um navegador padrão de internet. Não necessariamente haverá apenas uma visualização, já que, estando os dados disponíveis na rede, vários usuários poderiam visualizar algum estado da planta simulada. Isso tem relevância para um simulador de grande porte que adestra uma equipe na qual cada indivíduo gerencia um aspecto diferente do veículo e tem acesso a diferentes atuadores e sensores.

Como no módulo de interface não existe requisitos pesados de processamento, ele pode ser implementado em um computador pessoal comum (baixo a médio desempenho).

Será composto basicamente por um display gráfico e por um sistema de entrada de comandos do usuário.

6 DESENVOLVIMENTO DA ARQUITETURA DE HARDWARE

De acordo com o exposto na seção de objetivos, o terceiro objetivo era a implementação de um sistema para levantamento de derivadas hidrodinâmicas, no qual será implementado uma versão mais completa da estrutura proposta, com as camadas de fluxo de dados e interface com o usuário. A camada de simulação, para esta aplicação específica está sendo substituída por um modelo físico, a saber um veículo submarino que emula um veículo autônomo, construído para a finalidade de validar experimentalmente teorias hidrodinâmicas.

6.1 Antecedentes

Como exibido em [14], o AUV Pirajuba possuía uma arquitetura de controle híbrida, seguindo a arquitetura GESAM, proposta em [30]. Fazia uso de um PC104 com sistema operacional de tempo real (QNX), um barramento ethernet, software desenvolvido usando ferramentas gráficas (Rhapsody) e microprocessadores Rabbit 3000 como interface com os sensores e atuadores. O PC104 lia diretamente as saídas de uma bússola eletrônica e de uma unidade de medição inercial por meio de duas portas RS232.

Essa arquitetura, que possuía uma enorme confiabilidade de software, uma vez que usava linguagens gráficas foi pensada para a aplicação em veículos autônomos, que precisam resolver problemas de navegação, guiagem e controle. Na aplicação de identificação dinâmica foram observadas algumas desvantagens, tais como um grande número de cabos ethernet causando um certo congestionamento no limitado espaço interno do veículo, freqüentes problemas de mal contato nos conectores RJ-45 do hub-switch e dificuldade de acesso a componentes para reposição. Somando-se a isso, observou-se que o aprendizado necessário para fazer uma manutenção no sistema demandava um tempo muito longo, incompatível com o tempo disponível que um aluno no curso de graduação fazendo iniciação científica possui.

Abaixo, a lista de equipamentos usados no AUV.

Tabela 6.1 Equipamentos do AUV Pirajuba

Sistema	E/S	Interface	Protocolo
IMU VG600AA 202	Entrada e saída	RS232	binário
	Saída	Analógica	0 – 5V
Bússola TCM2	Entrada e saída	RS232	ASCII
Servos Rádio Controle	Entrada	Digital	PWM – RC
	Saída	Analógica	0,7-1,3V
Sensor Pressão MLH	Saída	Analógica	4-20mA
Servomotor Maxon	Entrada e saída	Analógica	±10V
Giroscópio KVH	Saída	Analógica	0-5V
Sensor de presença de líquido	Saída	Digital	Lógica TTL
Acelerômetros ADXL	Entrada e saída	RS232	binário
Odômetro Doppler - DVL	Entrada e saída	RS232	ASCII
Modem de Rádio Frequência	Entrada e saída	RS232	binário

6.2 Arquitetura escolhida

Dadas as circunstâncias e os equipamentos disponíveis, percebeu-se que era necessário o uso de microprocessadores para fazer a inicialização de alguns sensores, fazer a conversão de protocolos, e estabelecer a interface com o usuário. O uso de microprocessadores de 8 bits resolve quase todas as necessidades, porém não permite uma expansão do firmware embarcado. O preço de um microprocessador de 32 bits é similar ao de outro de 8 bits com recursos semelhantes de memória volátil e não-volátil, porém tem mais periféricos e um poder computacional bem mais elevado por ser de uma arquitetura mais moderna e efetuar contas de 32 bits em um único ciclo de relógio.

Foi cogitada a opção de empregar placas com processadores capazes de embarcar sistemas operacionais tais como Linux ou QNX. Porém, não foi achado algo que possuísse todos os periféricos requeridos para fazer a interface com os sensores e atuadores. De maneira geral, foi observado que tais placas podem possuir conversores analógico-digitais, geradores de PWM e conversores digital-analógicos, porém sempre em pequena quantidade e nunca todos esses periféricos em uma única placa. Por outro lado, essas placas possuem muitas entradas e saídas para transmissão de altas taxas de dados, tais como ethernet e USB.

Foi escolhido o LPC2148 da NXP semiconductors, que possui arquitetura ARM7 e que apresentava maior relação custo-benefício em termos de memória permanente, memória volátil e número de periféricos. Tem ainda disponibilidade imediata em módulos produzidos no País a um baixo custo. Uma vez que esse microprocessador possuía recursos de hardware que possibilitava a interface com todos os equipamentos disponíveis, seria necessário apenas um modelo de hardware para todas as tarefas, reduzindo o tempo de aprendizado dos alunos. Além disso, o poder computacional desse hardware já permite a aplicação de filtros mais elementares e tarefas mais complexas.

O uso de uma arquitetura centralizada foi descartado pela necessidade de, a posteriori, introduzir novos sensores. Levando em conta as experiências anteriores com ethernet e a análise dos autores vista na revisão bibliográfica, foi decidido, com o fim de aumentar a confiabilidade, a robustez mecânica, diminuir o custo, diminuir o tempo para obtenção de sobressalentes, reduzir os cabos e o volume de bordo, abandonar por hora o uso de ethernet e adotar uma rede CAN.

Seguindo a linha proposta em [27], decidiu-se fazer uso do sistema operacional μC-OSII, que é escrito em ANSI C, é portável para o ARM7, tem uma estrutura bastante simples e é bastante didático para alunos de iniciação científica em termos conceituais de sistemas operacionais. Vale lembrar que é um sistema operacional preemptivo de tempo real certificado para equipamentos médicos e aeronáuticos, que de acordo com o julgamento ora feito, aumentaria a confiabilidade e a facilidade de manutenção dos softwares gerados. Podemos ainda lembrar que softwares desenvolvidos em sistemas de plano de frente e de fundo (um laço infinito e diversas funções de interrupção), do inglês *foreground/background*, não possuem modularidade, ou seja, uma alteração em uma parte do código pode afetar o funcionamento de outra, prejudicando muito a flexibilidade e a manutenabilidade. Com um sistema operacional de tempo real confiável, é possível dividir cada tarefa em módulos totalmente independentes entre si, facilitando o projeto do software, simplificando o código, adicionando confiabilidade e facilitando a manutenção.

Dadas as limitações do número de entradas e saídas de cada unidade do microprocessador, é necessário contar com várias unidades, distribuindo as tarefas. A

comunicação entre cada módulo se dará por meio de uma rede CAN. O veículo conta com três vasos estanques onde serão colocados os equipamentos. O vaso mais a ré será o de propulsão, que possuirá apenas o motor e um sensor de presença de líquido. Haverá um vaso que protegerá os servos RC e o driver do servomotor de propulsão. Neste vaso estarão montadas as superfícies de controle.

O maior vaso será chamado de principal, pois tem maior quantidade de sensores, leva as baterias que alimentam o veículo e faz a comunicação com a estação base.

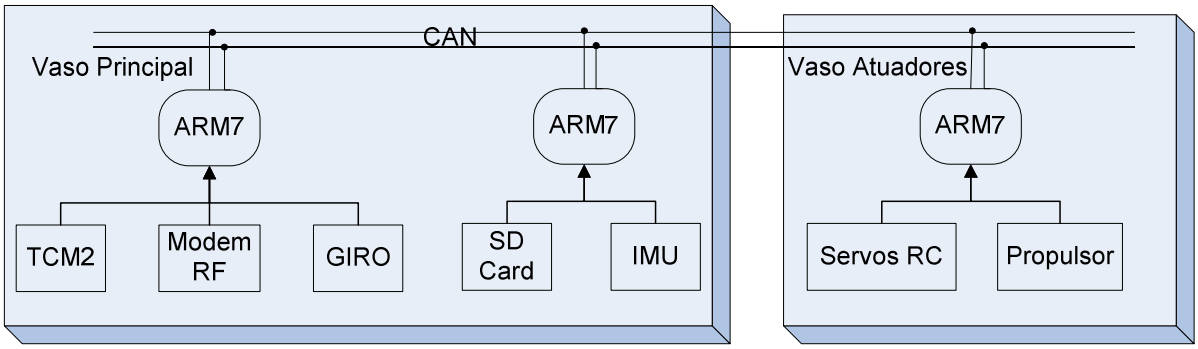


Figura 6.1 Diagrama de blocos simplificado do AUV Pirajuba

6.3 Circuitos de condicionamento e conversão de sinais

O LPC2148 trabalha com tensões na faixa de 0 a 3.3V. Entretanto temos equipamentos com sinais de diferentes tensões, tais como ±10V, 0-5V, 0-33V, RS232 (±15V). Para contornar esse problema foram usados amplificadores operacionais, amplificadores de instrumentação e divisores de tensão resistivos. Os sinais de ±10V que são fornecidos pelo driver do servomotor de propulsão foram convertidos para a escala de 0 a 3.3V com um circuito do tipo amplificador inversor seguido de um filtro passivo para que pudesse ser feita a A/D.

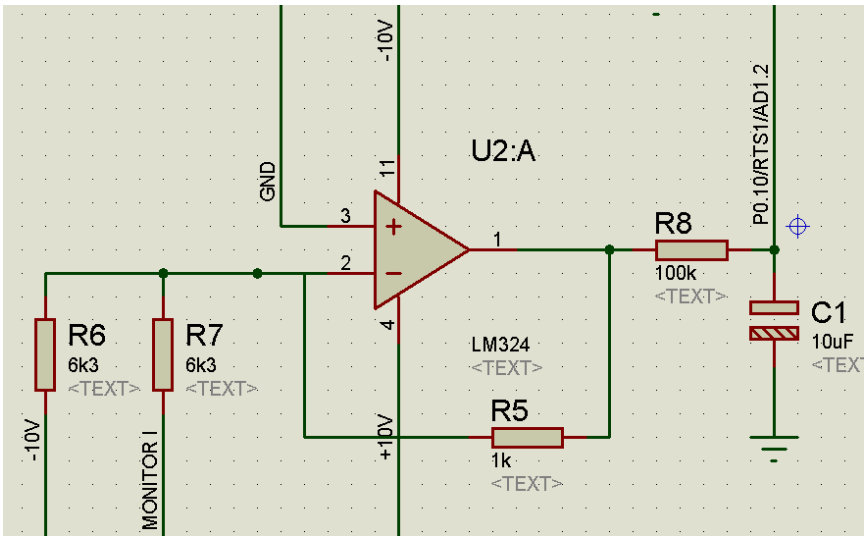


Figura 6.2 Circuito de conversão +/-10V para 3.3V. Monitor I é um sinal de saída do driver e P0.10/RTS1/AD1.2 é o pino do LPC2148 que fará a conversão A/D

Para gerar a ordem de rotação para o driver de ±10V a partir do DAC do LPC2148, foi usado um circuito de amplificação e deslocamento do sinal composto de um INA126. O servomotor possui ainda um sinal para habilitá-lo, que impede o seu funcionamento caso esteja a uma tensão abaixo de 4V. Uma vez que o nível alto de tensão do microprocessador é 3.3V, foi necessário usar um transistor e um pino do tipo dreno-aberto (quando o nível lógico é 1, o pino se comporta como se estivesse isolado; quando o nível lógico é 0, o pino está ligado na terra) para fazer a habilitação ou desabilitação desse atuador por software.

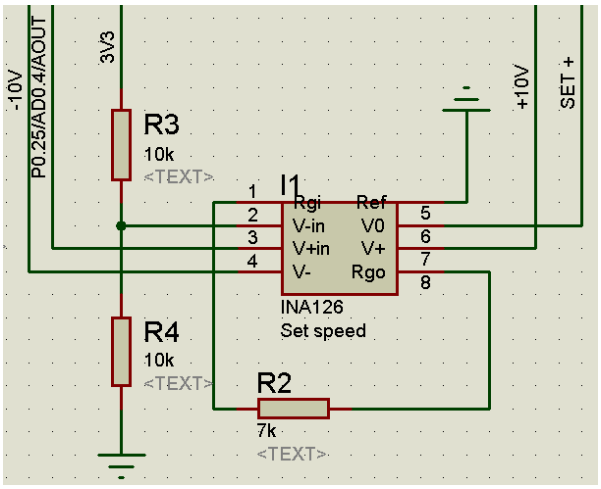


Figura 6.3 Circuito de conversão de 3.3V para +/-10V. SET+ é um sinal de entrada do driver e P0.25/AD0.4/AOUT é o pino do LPC2148 que faz a conversão D/A

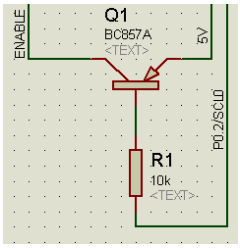


Figura 6.4 Circuito de habilitação do servomotor

Quantos aos outros sinais, foram usados divisores de tensão resistivos. Para a conversão dos sinais TTL para RS232 foram usados circuitos integrados MAX232 em encapsulamentos SMD.

6.4 Controlador de rede

Para fazer o controle da rede CAN, foi usado o circuito integrado MCP2515 da Microchip, em conjunto com o *transceiver* MCP2551 do mesmo fabricante. Esse CI tem dois *buffers* para recepção e três para transmissão. Usa um protocolo SPI com freqüência de até 10MHz para comunicação com o microprocessador. Possui ainda um pino que gera uma interrupção no microprocessador em caso de um evento qualquer. Permite que os dados trafeguem no barramento CAN com freqüência de até 1MHz, o que é bem maior do que a necessidade atual. Ao fazermos a soma de todos os dados

a ser transmitidos pela rede no espaço de tempo de 1 segundo, vemos que a largura de banda utilizada é inferior a 5% da banda disponível com o barramento a 1MHz. Devido a isso, atualmente o barramento vem sendo usado a 500kHz com o objetivo de reduzir o número de erros.

Esse CI, após ser inicializado e propriamente configurado pelo microprocessador, faz todo o trabalho de baixo nível preconizado pelo padrão CAN, tal como verificação de erros das mensagens recebidas, filtragem das mensagens indesejadas e retransmissão das mensagens perdidas. Tudo isso liberta o programador de ter que resolver esses problemas por software, permitindo que, após a codificação de um software de baixo nível para controlar esse CI, basta fazer uso de funções para enviar e receber mensagens. A título de informação, a possibilidade de que passe despercebido um único bit errado em uma mensagem CAN é da ordem de 10^{-14}[28].

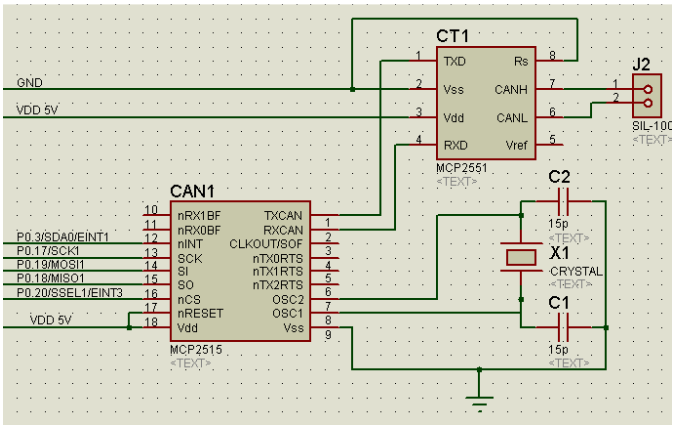


Figura 6.5 Circuito controlador de rede

6.5 Considerações de confiabilidade

Foi observada a presença de umidade dentro dos vasos de pressão da primeira versão do veículo apesar do mesmo ter passado nos testes de estanqueidade. Uma vez que existe a presença de muitos equipamentos eletrônicos, a exposição à água em forma de vapor é bastante preocupante em termos de confiabilidade. Se considerarmos que essa arquitetura se destina ao uso no mar, ainda temos a complicação da contaminação iônica, que diminui ainda mais o tempo médio entre falhas. Para dar uma idéia quantitativa de quanto é essa diminuição na vida do equipamento, vamos recorrer

ao modelo de Peck [29], que fornece o fator de aceleração da reação de oxidação de um componente eletrônico em função da umidade e temperatura.

$$A_k = \left(\frac{H_{de}}{H_{du}}\right)^{2.66} \times e^{\frac{E_a}{k}\left(\frac{1}{T_{ju}} - \frac{1}{T_{je}}\right)} \quad (1)$$

Onde:

A_k = Fator de aceleração da reação.

H_{de} = Umidade relativa no dispositivo em condições de ensaio.

H_{du} = Umidade relativa no dispositivo em condições de uso.

E_a= Energia de ativação. Para circuitos integrados fica em torno de 0.7 elétrons-volt.

T_{ju} = Temperatura no interior do dispositivo em condições de uso.

T_{je} = Temperatura no interior do dispositivo em condições de ensaio.

k = Constante de Boltzman (k = 8.62x10^{-5} eV/K).

De acordo com esse modelo, para uma elevação de temperatura de 20ºC para 70ºC, teremos uma redução no tempo de vida de cerca de 10 vezes, mantendo a umidade constante. O efeito do aumento da umidade relativa ainda é mais dramático. Se usarmos uma atmosfera com 10% de umidade relativa como referência, teremos uma redução de cerca de 450 vezes no tempo de vida do equipamento exposto a uma

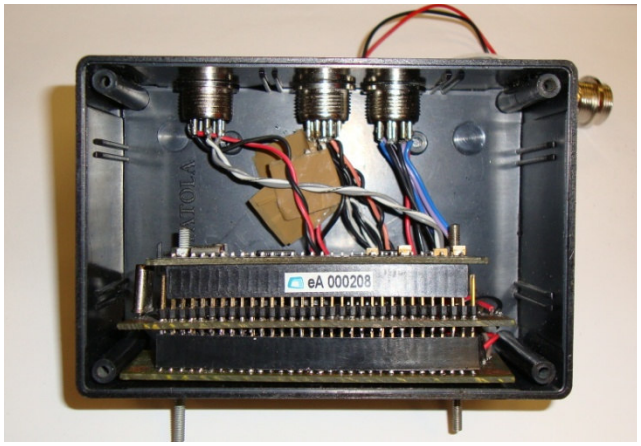


Figura 6.6 Caixa protetora dos circuitos eletrônicos

atmosfera saturada com vapor. Ainda não foi levada em conta a contaminação iônica devido à salinidade da água do mar.

Podemos então afirmar que a proteção dos circuitos eletrônicos é muito importante se quisermos obter uma boa confiabilidade do sistema como um todo. A primeira medida tomada foi a adoção de caixas plásticas lacráveis para isolar a atmosfera dos componentes eletrônicos do resto do vaso de pressão. Assim, fica efetiva a colocação de sacos de sílica para absorver a umidade local interna da caixa. Depois, na confecção das placas de circuito impresso foram usadas trilhas de maior largura do que o normalmente usado. Para ilustrar, a largura padrão das trilhas normalmente usadas é 10 milésimos de polegada, enquanto a menor trilha usada foi de 30 milésimos. Assim, aumenta-se a área a ser corroída até que haja a falha.

Outra medida foi a retirada de todas as fontes de calor significativas de dentro das caixas de proteção, tais como reguladores de tensão. O circuito que mais dissipa calor é o próprio microprocessador, que, conforme observado em laboratório, sofre um aquecimento muito leve devido à própria dissipação. Mais uma medida protetora será a aplicação de verniz sobre as placas prontas, de modo a proteger as trilhas e os pinos dos circuitos integrados da atmosfera circundante.

6.6 Considerações de robustez mecânica

Um dos problemas mais difíceis de resolver em sistemas embarcados são os contatos elétricos de má qualidade caso sejam usados conectores inapropriados. Em sistemas em fase de desenvolvimento isso é ainda mais crítico, pois o manuseio das

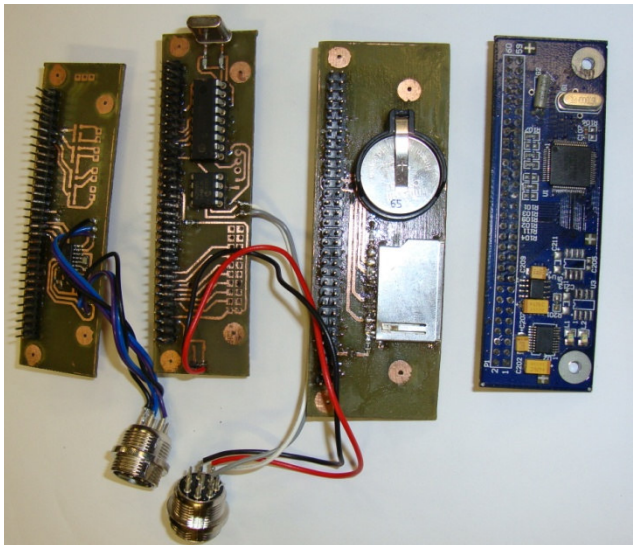


Figura 6.7 Módulo e-LPC64 e placas corroídas

peças é constante devido às freqüentes mudanças de configuração e testes.

Para conferir flexibilidade e facilidade de confecção das placas de circuito impresso, foi concebido um padrão de placas auxiliares com dimensões similares à dos módulos que contém os microprocessadores e que possuem um conector similar aos do padrão PC-104, com um conector DIN fêmea com 64 pinos embaixo e uma barra de pinos dupla em cima. Dessa forma, cada nó fica composto por uma pilha de placas e todos os pinos do módulo têm acesso a todas as placas da pilha. Em outras palavras, temos um barramento de 60 pinos (o módulo com o microprocessador possui 60 pinos) e mais 4 pinos extras para comunicação entre as placas.

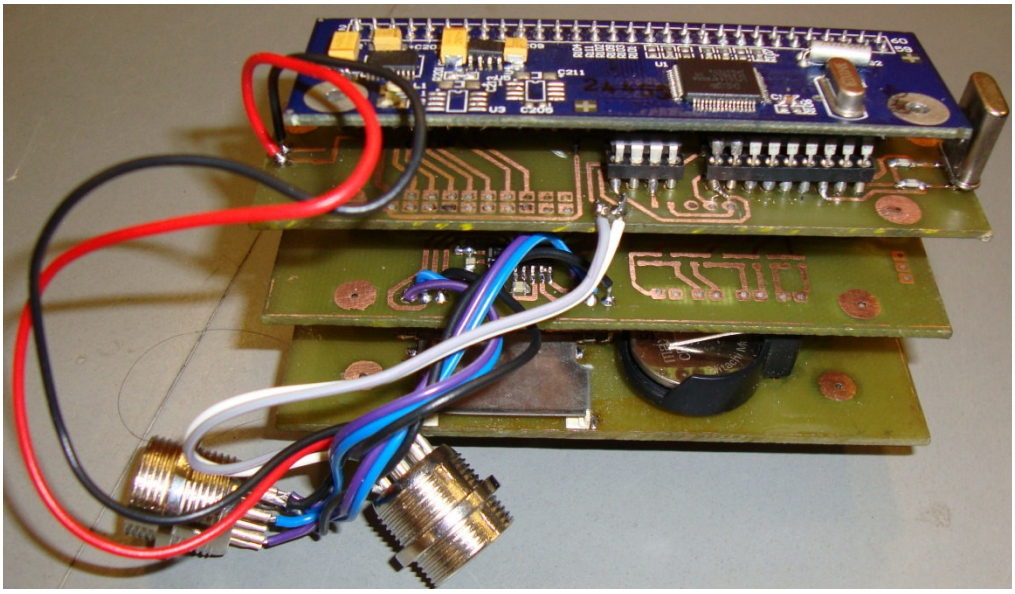


Figura 6.8 Pilha de placas com o módulo eLPC-64, placa com driver CAN, placa serial RS232 e placa com bateria e cartão de memória

Uma vez que temos um conector com um grande número de pinos, a conexão entre as placas fica bastante robusta, expondo cada pino, individualmente, a menores esforços de flexão. Para deixar o sistema ainda mais robusto, e aproveitando a própria concepção do módulo microprocessado, são passados dois parafusos sem fim com espaçadores na extremidade oposta ao conector, deixando o sistema bastante robusto.

Figura 6.9 Conectores Mike

Na parede da caixa protetora foram colocados conectores padrão automotivo do tipo MIKE com oito pinos, que possuem formato cilíndrico, o que facilita a abertura do furo para sua colocação. Esses conectores contam com uma blindagem metálica e molas bastante robustas. Contam ainda com rosca para garantir sua fixação e são polarizados, ou seja, é impossível a sua colocação em uma orientação errada. Na parte interna, os fios que saem dos conectores machos presos na parede da caixa são soldados diretamente nas placas, eliminando a necessidade de outros conectores e a possibilidade de falhas. Com isso espera-se reduzir o problema de contatos elétricos.

6.7 Considerações sobre ruído

Ao desenhar o PCB, alguns cuidados foram tomados para minimizar os ruídos. Foi utilizado um plano de terra na face inferior do circuito, de forma que toda trilha de sinal tenha uma "trilha" de terra próxima. Tomou-se cuidado para ter o menor número possível de trilhas no plano inferior e, quando necessárias, traçadas pelos cantos da placa para evitar divisões no plano.

Na escolha dos componentes foi dada preferência aos de encapsulamento SMD, devido à sua menor indutância e por ficarem posicionados mais próximos à placa [31]. Os componentes foram distribuídos em zonas de mesma função, para diminuir a interferência entre as diversas partes do circuito. O cristal do controlador CAN foi posicionado de forma a ficar afastado dos outros componentes por ser uma importante fonte de emissões eletromagnéticas. Os filtros dos sinais dos sensores foram

posicionados próximos ao conector do microcontrolador, de forma a filtrar os ruídos captados pela trilha entre o componente e a entrada. As trilhas foram feitas com o menor comprimento possível e as conexões ao terra foram preferencialmente feitas diretamente ao plano da face inferior através de vias, aumentando o isolamento entre os componentes.

Dessa forma foi possível reduzir consideravelmente os ruídos das medições, em comparação com as placas usadas para teste, feitas sem essas precauções.

6.8 Confecção das placas

Para a fabricação das placas foi utilizado o método de transferência térmica, que consiste em imprimir o circuito em transparência numa impressora laser, e depois transferir a para a placa de cobre utilizando um ferro de passar roupas. Experimentalmente foi determinado o tempo e a temperatura para realizar a transferência, aproximadamente 20s por face com 80% da potência máxima do ferro de 1350W. Temperaturas muito altas e tempos prolongados de aquecimento fazem com que a transparência se deforme, borrando o desenho. Após a transferência ter sido realizada a placa foi furada utilizando brocas de 0.9mm e as falhas foram retocadas com canetas especiais para circuito impresso. Em seguida, as placas foram corroídas com percloreto de ferro e envernizadas para evitar a oxidação do cobre.

A precisão obtida através desse método é consideravelmente boa para métodos artesanais, permitindo inclusive trilhas passando entre os pinos do conector, que tem um espaçamento de um décimo de polegada. Neste caso, as trilhas foram desenhadas maiores do que o espaço disponível, e após terem sido transferidas ao cobre foram raspadas à mão com um objeto de ponta bem fina e cortante para não entrar em contato com o os pinos.

6.9 Software embarcado

Microprocessadores possuem hoje um extenso leque de opções para sua programação com ferramentas gratuitas. A linguagem mais comum é o ANSI C. Além disso, o sistema operacional a ser utilizado foi codificado em C. Entretanto, por se tratar de um projeto de software que pode adquirir um tamanho considerável com o passar

do tempo e introdução de novas funcionalidades e equipamentos, visto que o hardware permite alguma expansão, uma dose de disciplina na arquitetura de software se faz necessária, no sentido de permitir a modularidade, compartimentalização e verificação do código gerado.

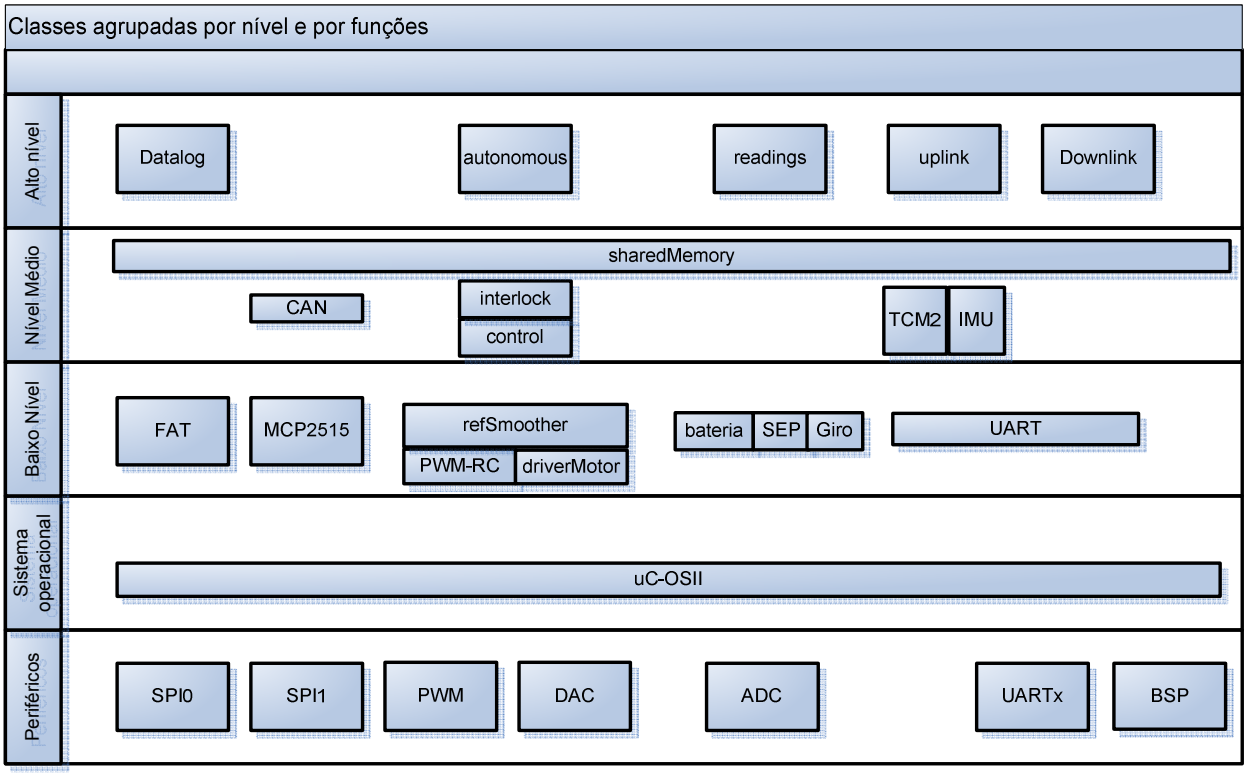


Figura 6.10 Visão geral do software embarcado

Embora a linguagem C não tenha sido feita de modo a prever o uso de modo orientado a objetos, é possível usar alguns conceitos básicos como encapsulamento, agrupamento simultâneo de atributos e métodos e visibilidade. Dessa forma é possível deixar o código bastante modular e fácil de ser verificado.

Para facilitar o entendimento, os módulos de código foram divididos por função e por nível. No gráfico abaixo, a posição vertical de um objeto indica o seu nível e a posição horizontal indica aproximadamente os objetos com os quais ele se relaciona.

No nível de periféricos existem objetos que fazem a inicialização dos periféricos internos do microprocessador, bem como controlam seu comportamento através de

seus métodos. São muito usados aqui os registradores de configuração ou de entrada e saída de dados e o uso de linguagem binária é predominante. No modelo adotado, cada periférico é um objeto e possui atributos, que são os estados dos periféricos em si, e métodos, que são as funções que obtém dados ou modificam esses atributos.

Acima desse nível, o sistema operacional disponibiliza uma série de recursos para os objetos acima, tais como semáforos, possibilidade criação de seções críticas de código, tempos de espera, caixas de correio e temporizadores, entre outros.

Acima do sistema operacional temos as funções de baixo nível, que usam os métodos do sistema operacional e dos objetos contendo os periféricos. Alguns, mais complexos, como a porta serial assíncrona, usam métodos privados de interrupção para tratar a chegada de dados. Outros, além disso, possuem uma *thread* própria, como é o caso do MCP2515.

Nível acima, temos os objetos que prestam serviço aos objetos de mais alto nível coordenando as ações de objetos inferiores e alguns fazem o controle de periféricos externos ao microprocessador, como os sensores e o controlador CAN. Neste nível vale ressaltar a classe *sharedMemory*, que providencia, para todos os objetos acima e abaixo, todos os dados necessários, vindos pela rede ou não.

Essa classe, que possui uma instância em cada nó da rede, permite que um objeto de nível superior possa ser facilmente transferido de um nó para outro sem causar complicações adicionais, pois ela faz com que todas as informações estejam disponíveis igualmente em todos os nós, exceto quando esse objeto lida com recursos que são intrínsecos a um determinado nó.

Todos os dados obtidos por qualquer objeto é transferido para a instância local da *sharedMemory*, que envia pela rede os dados apropriados e mantém armazenados a última atualização dos dados vindos da rede. Além disso, fornece métodos que retornam os dados necessários a cada objeto.

No nível mais alto estão as funções finais do sistema, que é o armazenamento de dados, a realização de manobras autônomas, leituras periódicas de sensores, envio de dados para a estação base e recepção e decodificação dos comandos enviados pela estação base.

6.10 Detalhamento dos objetos

A seguir, será feita uma descrição breve dos módulos constituintes do sistema para melhorar o entendimento da estrutura seguida. Os módulos do nível periféricos não serão abordados por ser apenas de configuração e uso de recursos de *hardware*, dispensando maiores explicações. A única exceção é o objeto BSP.

6.10.1. Objeto BSP

Do inglês, *Board Support Package,* oferece ao sistema operacional as funções necessárias ao seu funcionamento, como, por exemplo uma interrupção periódica para chamar a rotina de *tick* do sistema operacional. Oferece ainda uma estrutura que permite que qualquer outro objeto tenha um método chamado a cada *tick*. Esse método, que no caso não deve retornar e nem receber nenhum valor, é chamado *callback*. Uma das utilidades é, por exemplo, a possibilidade de ser feita uma leitura periódica de um sensor com alta freqüência sem requerer a existência de uma tarefa para isso. Outra utilidade implementada por esse módulo é uma pilha de códigos de erro. As ações de cada objeto do sistema podem falhar por uma infinidade de razões. Para facilitar o diagnóstico da falha, foi criado um código de erros com um *byte* de extensão. A cada erro ocorrido, qualquer objeto pode inserir o código correspondente na pilha. Depois, esse código é recuperado e armazenado no objeto *datalog* e enviado para a estação base.

6.10.2. Objeto FAT

Esse objeto implementa rotinas de criação, alteração, apagamento e movimentação de arquivos em uma memória permanente formatada de acordo com o padrão FAT. O interesse, no caso, é usar um *SD-card* para fazer o armazenamento de dados obtidos pelos sensores durante o ensaio.

6.10.3. Objeto MCP2515

Faz o controle, usando os métodos do objeto SPI1, do CI que controla a rede CAN. Implementa métodos para efetuar os comandos definidos pela interface SPI do

CI. Possui uma *thread* interna, que só é acessível por outros objetos através de funções de *callback* fornecidas na inicialização do objeto. Além disso, possui uma rotina de interrupção para reagir aos eventos de término de envio de uma mensagem e recepção de nova mensagem. A rotina de interrupção, para evitar que a latência ou tempo de resposta do sistema fique lenta, apenas sinaliza um semáforo para que a *thread* interna do objeto faça a verificação do dispositivo e tome as medidas cabíveis.

6.10.4. Objetos refSmoother, supControl, driverMotor

O objeto supControl traduz os comandos em forma de décimos de grau para o número binário que gerará a largura de pulso desejada, além de fazer a inicialização dos periféricos de PWM. O objeto driverMotor usa o periférico DAC para gerar a tensão de referência para o servomotor de propulsão, fazendo também a conversão da ordem em RPM para o número binário apropriado. Já o objeto refSmoother, abreviado de suavizador da referência, gera uma rampa a partir dos comandos enviados. Dessa forma evita-se variações bruscas de torque e corrente nos atuadores.

6.10.5. Objeto bateria

O AUV Pirajuba usa baterias de lítio-polímero (LIPO), que permitem uma grande quantidade de carga por espaço e por peso, além de uma grande corrente de descarga. O problema é que tais baterias são bastante delicadas, requerendo carregadores especiais e sistema de monitoração do nível, pois uma descarga completa inutiliza a bateria.

Outro problema bastante complexo é que a tensão nos terminais da bateria varia muito em função da corrente. Assim, para obtermos o nível real da bateria temos que levar em conta a corrente consumida e a tensão nos terminais da bateria. Devido a uma certa discordância entre fabricantes de baterias de LIPO, foi levantada experimentalmente a curva do modelo presente no laboratório e que será embarcado.

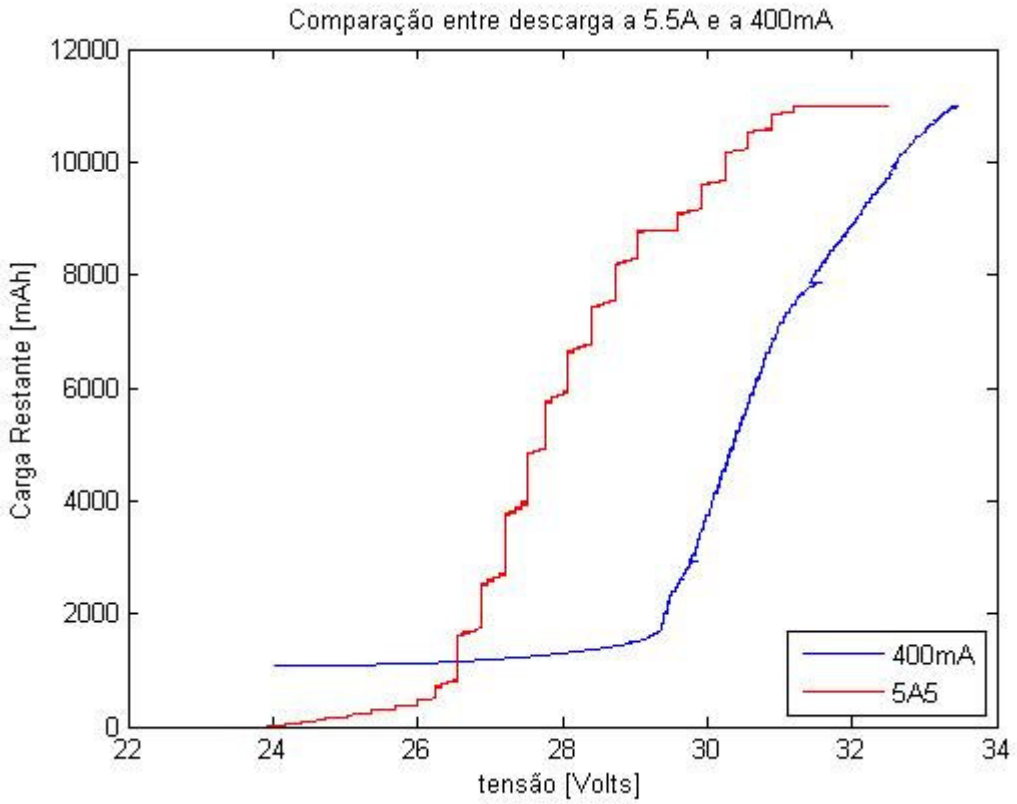


Figura 6.11 Curva de tensão x descarga das baterias LIPO do AUV Pirajuba

O objeto bateria usa o ADC para obter a tensão nos terminais da bateria e obtém o percentual de carga existente para as duas condições ensaiadas acima por meio de um polinômio de segundo grau. Depois, usando a corrente do motor, adicionada do consumo fixo do sistema, que é por volta de 400mA, faz uma interpolação linear entre as dois percentuais obtidos.

6.10.6. Objeto SEP ou servoPosition

Na concepção original do AUV, tinha sido admitido que o servo RC teria um erro de posicionamento desprezível. No entanto, foi verificado que o erro de posicionamento pode chegar a até 2º sob carga, o que aumenta quando temos um eixo com engaxetamento para permitir a atuação dos lemes por um servo dentro do vaso de pressão. Isso é considerado excessivo para ser desprezado nos testes de identificação dinâmica.

Foi descoberto que a realimentação de posição dos servos é feita por meio de um resistor variável solidário ao eixo de saída do servo e que esse sinal poder ser facilmente lido por um ADC do microprocessador, requerendo previamente uma amplificação do sinal.

Esse objeto faz a leitura dos servos e as disponibiliza para o armazenamento de dados. A leitura desses ângulos permitir verificar o desvio em função da carga e a existência de folgas no acoplamento entre os servos e os eixos dos lemes. Em um ensaio bastante aproximado, verificou-se que o desvio em função do momento máximo esperado nos lemes será de cerca de 0.5º. Entretanto ainda verificou-se que o leme sofre deflexões um pouco maiores devido à elasticidade do acoplamento que é de plástico (cerca de 0.7º).

6.10.7. Objeto Giro

Para obter uma medida mais precisa da velocidade angular em torno do eixo vertical, foi adicionado um giroscópio de fibra ótica, que tem uma saída de 0-5V e um sinal de referência para compensar o erro devido ao aumento da temperatura. Para diminuir a dispersão da leitura, são feitas múltiplas leituras a cada amostra. Além disso, é usado o recurso de *callback* do BSP para efetuar uma amostra a cada *tick* do sistema operacional, que no caso, roda a 100Hz.

6.10.8. Objetos TCM e IMU

Esses dois sensores se comunicam através de uma porta serial RS232 e tem um funcionamento bastante similar. Quando o sistema é ligado, os respectivos objetos ativam suas *threads* e fazem a configuração do dispositivo. Periodicamente, os sensores enviam uma mensagem com a última leitura, sendo a freqüência de atualização da TCM2 de 14Hz e da IMU de 50Hz. Cada objeto, ao receber uma mensagem, checa o *checksum* para verificar se não há um erro na mensagem. Caso a mensagem esteja correta, é feita a decodificação para o padrão binário. Depois disso, os dados recebidos são enviados pelo barramento CAN via *sharedMemory*.

6.10.9. Objetos *interlock* e *control*

O *interlock* funciona como uma trava de segurança. Uma vez que podem haver danos à bateria se houver uma descarga muito grande, é imperativo que os atuadores sejam desligados caso isto aconteça. Outro problema que pode ocorrer é a perda de comunicação com a estação base. Caso isso ocorra, será acionado uma sequência de emergência. Para permitir que o veículo volte a receber comandos, é preciso enviar pelo menos uma mensagem retornando ao modo ocioso.

O objeto *control* faz a configuração do dispositivo, verificando qual o modo de operação atual e tomando as medidas adequadas. Funciona como uma máquina de estados, sendo que o estado inicial é *idle* (ocioso), em que todos os atuadores estão desligados.

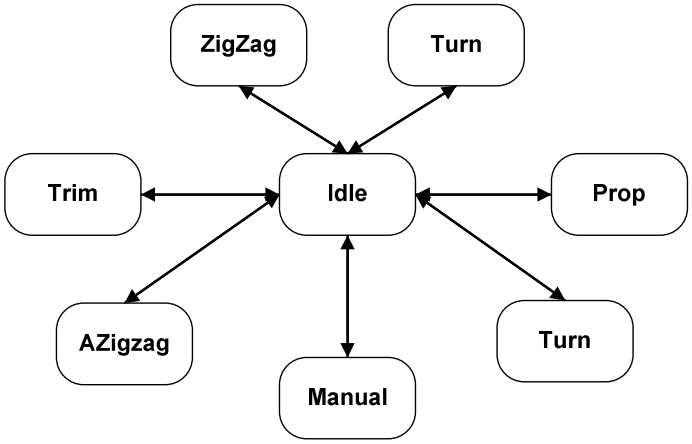


Figura 6.12 Diagrama de estados do objeto *control*

Todas as transições ocorrem do estado ocioso ou para o estado ocioso, sem exceção. Foi padronizado dessa forma para que as manobras tivessem melhor repetibilidade. Isso implica que, ao colocarmos o sistema em qualquer um dos modos de manobra, o veículo iniciará uma rotina pré-definida de aceleração, estabelecimento de regime permanente, execução da manobra e posterior desligamento.

Leme padrão é o ângulo de leme em décimos de grau que o sistema usará para efetuar as manobras. A guinada padrão é o desvio a partir do rumo inicial que o veículo fará antes de inverter o leme na manobra de zigzag.

A montagem dos lemes não é feita em um sistema mecânico que permita repetibilidade. Para compensar isso, é necessário um sistema de trimagem para levar

os lemes à posição ideal. Para facilitar, há dois (vertical e horizontal) ângulos de trim que afetam os dois lemes colineares ao mesmo tempo, denominado “trim conjunto” e dois ângulos de trim (inferior e esquerdo) que afetam apenas um dos lemes (trim individual). Assim, primeiro acerta-se o alinhamento entre os lemes usando o trim individual e depois acerta-se empiricamente qual o ângulo de trim conjunto que deixa o veículo percorrendo uma linha reta. Obviamente, o valor exato do ângulo de trim conjunto será determinado no modo manual, quando o usuário possui total controle sobre o veículo.

O fluxo de informações segue a tabela abaixo:

Tabela 6.2 Fluxo de informações do objeto *control*

Comando Joystick	**idle**	**trim**	**prop**	**Manual**
Câmera horizontal	Leme padrão	Trim leme inferior	Velocidade lemes	Trim lemes verticais
Câmera vertical	Guinada padrão	Trim leme esquerdo	sensor	Trim lemes horizontais
Manche eixo horizontal			Tempo de aquisição	Leme vertical
Manche eixo vertical			Profundidade padrão	Leme horizontal
Manche eixo rotação			Tempo de aceleração	Aileron
Acelerador		Velocidade padrão	Aceleração motor	Velocidade

Ao entrar no modo de trim, os valores dos últimos ângulos de trim conjunto são automaticamente armazenados pelo objeto *control.* Nesse modo é possível alterar os trim individuais e a velocidade padrão a ser usada pelo objeto *autonomous* para fazer as manobras.

No modo “prop”, são alterados: o tempo de ensaio (Tempo de aquisição, depois do qual o veículo é automaticamente desativado), o tempo de aceleração e estabelecimento do regime permanente em linha reta (Tempo de aceleração), a profundidade padrão a ser buscada, caso esteja em um modo que use manutenção de profundidade por meio de um controlador (AZigZag, ATurn), e as inclinações das rampas dos atuadores (velocidade dos lemes e aceleração do motor). O campo sensor

define o sensor a ser usado para a manobra de zigzag, que pode ser a bússola, a integral do giroscópio ou a saída de um filtro.

6.10.10. Objeto *autonomous*

Esse objeto possui uma *thread* interna que fica desativada esperando que o sistema entre em algum estado de manobra. Possui ainda atributos como o leme padrão, a guinada padrão, velocidade padrão, tempo de aquisição e tempo de aceleração que são controlados pelo objeto *control* com base nos comandos recebidos da estação base. Quando efetuamos a transição para um estado de manobra, que será necessariamente a partir do estado ocioso, com todos os atuadores desligados, a *thread* aciona os lemes com os respectivos ângulos de trim, o que deverá ser o zero do ponto de vista mecânico, aciona o motor, que deverá a ser acelerado de acordo com a aceleração pré-estabelecida, e esperará a que o veículo atinja o regime permanente. Após esse tempo, será iniciada a manobra definida.

Caso a manobra seja AZigZag ou ATurn, será usado um controlador para manter a profundidade a partir das leituras do sensor de pressão.

6.10.11. Objeto Datalog

Esse objeto possui uma *thread* interna própria e tem como objetivo criar uma cópia de segurança dos dados gerados no ensaio. Recupera todos os dados do objeto *sharedMemory*, gera uma *string* no padrão ASCII, e, ao acumular mais do que 512 bytes, armazena na memória permanente que é gerenciada pelo objeto FAT, que por sua vez usa o objeto SPI0. Também funciona como uma máquina de estados. Seus estados são 3: Ocioso, quando armazena uma leitura do sistema por segundo; Desligado, quando fecha o arquivo no cartão de memória e para toda atividade para evitar danos no desligamento; e manobrando, quando armazena a 10Hz todos os dados.

Figura 6.13 Diagrama de estados do objeto *Datalog*

6.10.12. Objeto *readings*

Esse objeto possui uma *thread* interna própria e faz a leitura das posições dos servos (via *servoPosition*), lê os sensores de presença de líquido dos vasos de propulsão e de atuadores e aquisita a corrente e a velocidade do motor via *driverMotor*. Faz todas essas tarefas a 10Hz e envia, por meio do objeto *sharedMemory*, essas informações pelo barramento CAN.

6.10.13. Objetos *uplink* e *downlink*

Ambos possuem uma *thread* interna própria. O *downlink* faz a leitura das ordens vindas pela estação base e verifica se não houve perda de contato, ou seja, não recebe mensagens válidas por um período de tempo. A cada mensagem recebida, os dados são interpretados e enviados pela rede CAN. Seja houver uma perda de contato, é passado um aviso para o objeto *interlock,* que iniciará a manobra de emergência. O objeto *uplink* faz a leitura do sensor de nível de líquido, do nível das baterias e do giroscópio e envia os dados pertinentes pela rede CAN via *sharedMemory*. Depois recupera todos os dados da *sharedMemory* e envia via serial RS232 para a estação base. Esse processo ocorre a 10Hz.

7 IMPLEMENTAÇÃO EM MATLAB

7.1 Camada de simulação

Tendo sido já proposta a arquitetura para um simulador genérico, foi feita uma implementação de um simulador de manobras de veículos submarinos para apoiar a decisão, partindo de um trabalho inicial [1], que já implementava HILS com um modelo analítico e semi-empírico. Foi acrescentado um modelo comumente usado para simulação de manobras de submarinos [2] para manter a compatibilidade com métodos de estimação de derivadas hidrodinâmicas em uso. O modelo hidrodinâmico implementado está detalhado no apêndice A.

Nesse software, que não requer uma interface interativa com o usuário, nem precisa fazer a simulação em tempo real, o esforço de desenvolvimento ficou restrito à solução das equações de movimento dentro da estrutura proposta, ou seja, usando o MatLab Simulink de forma que se possa facilmente verificar a correção das equações e seja fácil embarcar o código gerado em um computador através do RTW, que é uma ferramenta de tempo real. Dessa forma foi feita uma implementação da camada de simulação. Para o cálculo em tempo real das integrais de arrasto cruzado do modelo proposto em [2], foi usada uma técnica chamada quadratura gaussiana.

Uma ressalva importante é o uso do modelo de propulsão descrito em [3], que diferentemente do modelo em [2], faz correções no torque e no empuxo gerado pelo hélice devido ao ângulo de ataque local. Esse modelo de propulsão basea-se no método clássico para modelagem de propulsão de navios.

7.2 Camada de interface com usuário

Usando uma biblioteca para o Simulink de RS232, foi desenvolvido um software com as seguintes funções: leitura dos dados enviados pelo software embarcado; aquisição de comandos de um joystick típico de simuladores de vôo; arquivamento automático de dados relevantes; e apresentação, em um ambiente de realidade virtual, da atitude do veículo, dos ângulos de leme e de outros dados julgados relevantes para a operação do veículo.

Vale ressaltar que este ambiente de realidade virtual pode ser visualizado em navegadores padrão de internet, o que possibilita desde já a visualização de dados em vários computadores ligados em uma rede ethernet.

8 DISCUSSÃO

Para o teste do simulador de manobras no meio líquido foi usado um conjunto de derivadas hidrodinâmicas típicas de um submarino, retiradas de [3]. Para a obtenção dessas derivadas foi criado um submarino virtual com os lemes verticais e horizontais iguais, casco cilíndrico e vela, e por meio de CFD, foram obtidos os coeficientes. Foi simulada uma manobra de giro sem controle de profundidade e foram obtidas as respostas típicas de um submarino.

Quanto à implementação do sistema de auxílio a ensaios, foram feitos testes em bancada simulando o teste a ser feito em uma piscina, por exemplo. A manobra ensaiada foi o zigzag, sendo que foram feitas três variantes: uma com deflexões de 5º de leme e guinadas de 5º à direita e à esquerda do rumo inicial, outra com deflexões de 10º de leme e 10º de guinada e a última com 20º de deflexões do leme e 20º de guinada. Todos os sinais foram aquisitados a 10Hz.

A manobra constitui-se várias fases. Primeiro há um tempo de aceleração do motor, que começa com a ativação do propulsor e termina quando o mesmo atinge a rotação final. Depois os lemes são mantidos neutros e espera-se o veículo atingem uma condição de regime permanente. Depois disso, é iniciada a manobra com uma deflexão de leme levando a uma guinada para a esquerda. Quando a proa do veículo ultrapassa o ângulo de guinada, o leme é invertido e o veículo começa a sofrer uma aceleração angular para o outro lado, até ultrapassar o ângulo de guinada, quando o leme é invertido novamente. Para a movimentação do veículo foi usada a força humana.

A posição angular do leme exibida é a medição feita pelo sistema de aquisição da posição angular dos servos que usa o potenciômetro que o próprio atuador usa para a realimentação. Trata-se então da posição efetiva dos lemes, e não do comando enviado. Observa-se que os atuadores tem sua velocidade angular limitada pelo objeto *refSmoother*, o que diminui os trancos e picos de corrente, bem como ocasiona menor desgaste. Verifica-se ainda que existem erros locais de posicionamento, causados tanto pelo erro do servo, que de acordo com medidas feitas no laboratório, ficam em torno de 0.5º, como também problemas de não linearidade do medidor, que apresenta erros maiores a 20º.

Para referência foi usada a integral do giroscópio, pois o laboratório possui muitos objetos de ferro magnetizado, tais como armários e mesas, além de aparelhos que geram forte ruído eletromagnético, como fontes. A desvantagem é a deriva gradual que pode ser vista durante a fase inicial do ensaio. São mostradas também as comparações entre a integral do giroscópio e a saída da bússola eletrônica.

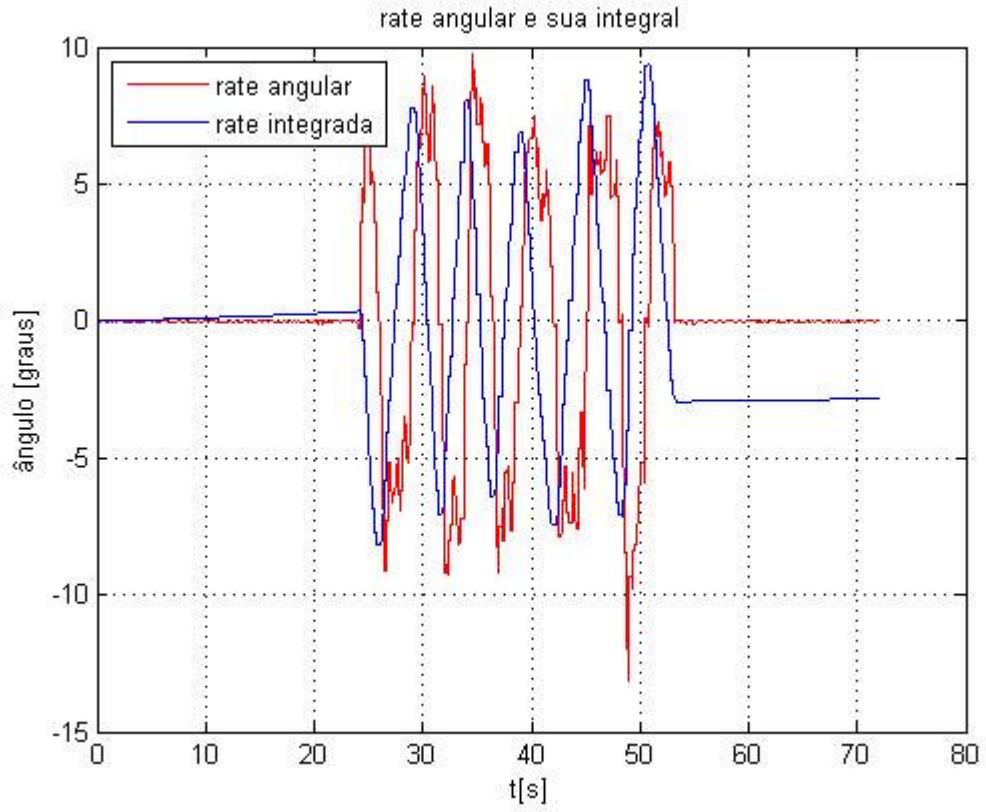


Figura 8.1 Leitura do giroscópio e sua integral

Como pode ser visto acima, durante o estado de repouso a integral da velocidade angular deriva lentamente, apesar de visualmente parecer que o ruído é bem pequeno na medida da velocidade angular. Vemos que, na guinada a velocidade angular sofre oscilações, e isso se deve à baixa rigidez e amortecimento do atuador do movimento do veículo, que no caso é o ser humano.

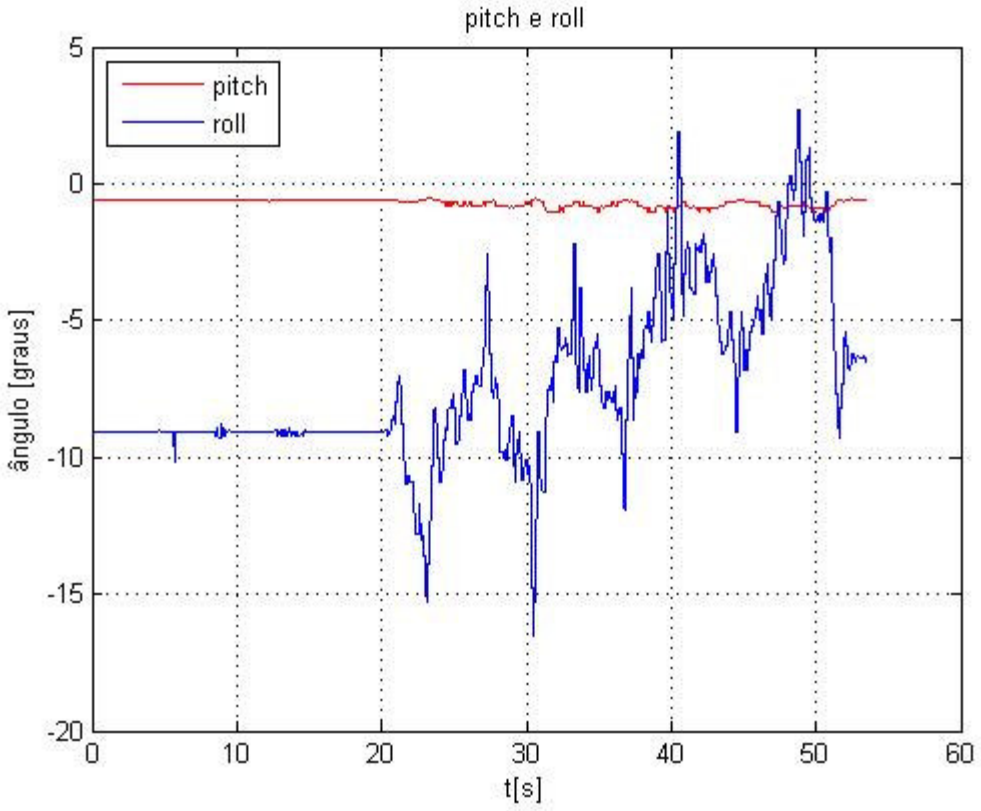


Figura 8.2 Ângulos de *roll* e *pitch* medidos pelo inclinômetro da bússola

Nota-se que o ângulo de *pitch* varia bem pouco neste caso, pois o movimento se dá por meio de deslizamento sobre um plano. Por outro lado, o houve bastante interferência no ângulo de *roll.*

A próxima figura faz uma comparação do ângulo de leme e do *yaw*, e exibe algum ruído na medição da deflexão dos lemes. Note que a inversão dos lemes se dá no ponto quase exato em que o *yaw* passa por +/- 5º.

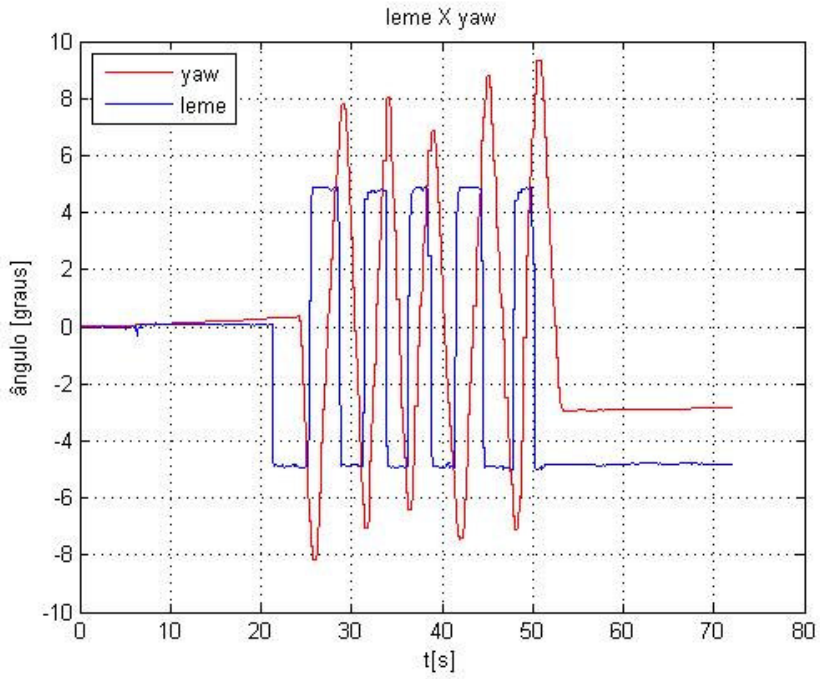


Figura 8.3 Comparação do ângulo de leme e do *yaw*

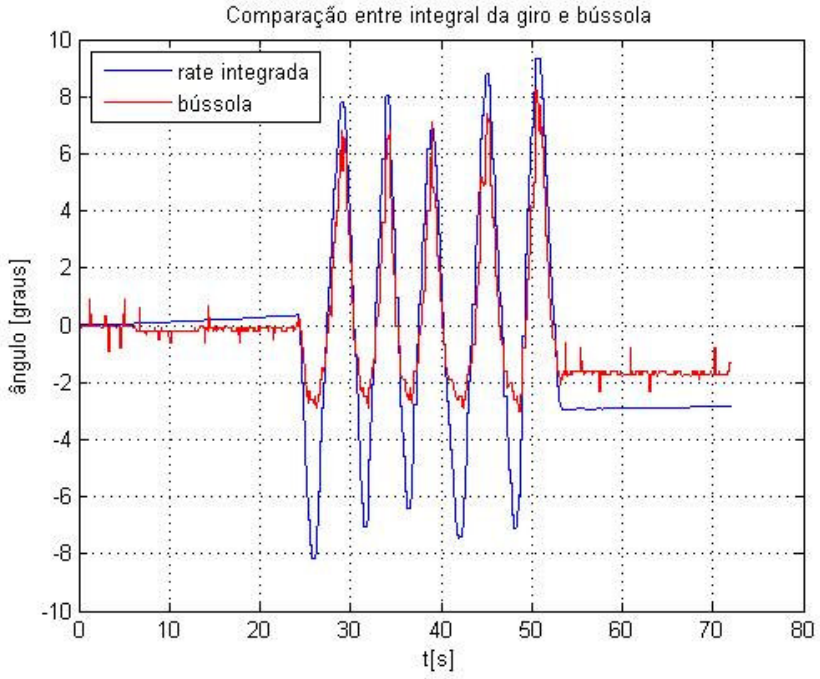


Figura 8.4 Comparação da saída da bússola e da integral da saída do giroscópio

Observe que para um dos lados a saída da bússola acompanha melhor a integral do giroscópio do que para outro. Isso se deve a uma fonte local de distúrbio magnético, a saber, um armário de ferro ao lado do veículo. Observa-se também picos de ruído de até 1º, provavelmente causados pelos equipamentos do laboratório. Em laboratório, pode ver que as medições de campo magnético ficam bastante comprometidas.

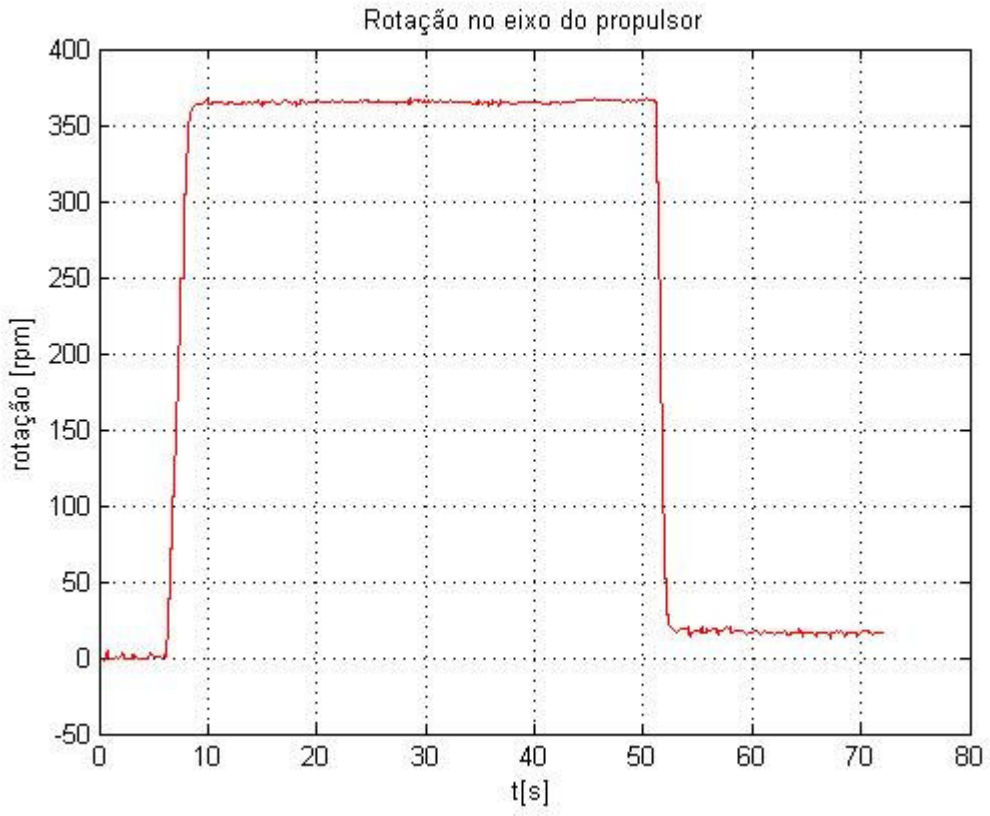


Figura 8.5 Leitura da rotação do eixo do motor

Na leitura da rotação do eixo do motor, observa-se um pequeno ruído e alguma histerese.

9 CONCLUSÃO

O estabelecimento de uma estrutura de software permite que o desenvolvimento e manutenção do código sejam feitos de maneira mais segura e uniforme, independendo do número de pessoas envolvidas no projeto.

Uma estrutura genérica pode ser demasiadamente complexa para uma aplicação específica, mas dentro de um contexto institucional, ou seja, quando o código passa pelas mãos de mais de uma pessoa durante o ciclo de vida, a adoção de uma estrutura é bastante recomendável.

A separação em camadas fornece, no caso, um grande potencial de modularidade e flexibilidade. A estrutura ora usada permite grande variação de uso em veículos submarinos com pequenas alterações no código. As aplicações previstas são: simulação de manobras para verificação de requisitos; simulação para verificar o desempenho de controladores; controle de veículos, caso em que seriam usadas apenas as camadas de fluxo de dados e interface com o usuário; simulação hardware-in-the-loop para validar ao mesmo tempo o software e o hardware de um controlador; simulação em centros de treinamento de pessoal; e aplicação em veículos autônomos, caso em que seria usada parte da camada de fluxo de dados.

A exeqüibilidade de um projeto em termos institucionais também é muito importante. Um determinado recurso pode ser possuidor de vantagens técnicas enormes, porém, para que seja virtualmente inútil, basta que ele tenha um dos seguintes atributos: excessivamente caro; disponibilidade reduzida, tempo de familiarização muito longo.

Existem também ao longo do trabalho preocupações no sentido de aumentar a confiabilidade do sistema como um todo, levando em conta o ambiente ao qual um veículo submarino é exposto. Na parte de hardware, essa preocupação se traduziu na preocupação de implementar um sistema mecanicamente robusto e na preocupação de isolar os equipamentos eletrônicos da atmosfera do meio ambiente. Na parte de software, foi seguido um certo rigor para o desenvolvimento de modo a manter as características de tempo real, a saber, resposta temporal, simultaneidade, previsibilidade e robustez. No sentido de manter o código portável foi seguido até certo ponto o padrão MISRA C.

Uma possibilidade ainda não explorada é a expansão do sistema com um hardware compatível com o RTW com capacidade para se comunicar em uma rede CAN. Dessa forma seria possível, com pequeno esforço de desenvolvimento, permitir a interação do sistema com filtros para obtenção precisa de dados, como posição, orientação e velocidade. Outra possibilidade ainda não mencionada é a interação de atuadores e sensores de forma que se tenha uma plataforma que mude sua atitude de acordo com a saída de um simulador, e dentro dessa plataforma um conjunto de sensores, filtros e controlador enviam comandos para o veículo simulado, dessa forma não validando apenas o controlador, mas o conjunto completo.

APÊNDICE A – CONVENÇÃO DE NOMES E DEFINIÇÕES

Tick – unidade de tempo que o sistema operacional espera para verificar se existe uma tarefa de maior prioridade pronta para ser ativada. Também se aplica à rotina chamada para fazer essa verificação.

Checksum – um ou dois bytes que contém a soma ou o “ou exclusivo” de todos os bytes que compõem a mensagem

APÊNDICE B – MODELO DINÂMICO

$$\left[M_{ij} + A_{ij}\right]\{\dot{V}\} + F_I(V) = F_H(V) + F_C(V, \delta) + F_G + F_E + F_T$$

Onde:

$V = \{u \quad v \quad w \quad p \quad q \quad r\}^T$, ou seja, as velocidades lineares e angulares do centro do referencial fixo ao corpo do veículo.

$$M_{ij} = \begin{bmatrix} m & 0 & 0 & 0 & mz_G & -my_G \\ 0 & m & 0 & -mz_G & 0 & mx_G \\ 0 & 0 & m & my_G & -mx_G & 0 \\ 0 & -mz_G & my_G & I_{xx} & -I_{xy} & -I_{xz} \\ mz_G & 0 & -mx_G & -I_{xy} & I_{yy} & -I_{yz} \\ -my_G & mx_G & 0 & -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}$$

O termo A_{ij} refere-se à massa de água que o veículo acelera junto ao seu corpo devido à componente viscosa do arrasto e é enunciada da seguinte forma [14]:

$$A_{ij} = \begin{bmatrix} X_{\dot{u}} & 0 & 0 & 0 & 0 & 0 \\ 0 & Y_{\dot{v}} & 0 & Y_{\dot{p}} & 0 & Y_{\dot{r}} \\ 0 & 0 & Z_{\dot{w}} & 0 & Z_{\dot{q}} & 0 \\ 0 & K_{\dot{v}} & 0 & K_{\dot{p}} & 0 & K_{\dot{r}} \\ 0 & 0 & M_{\dot{w}} & 0 & M_{\dot{q}} & 0 \\ 0 & N_{\dot{v}} & 0 & N_{\dot{p}} & 0 & N_{\dot{r}} \end{bmatrix}$$

O termo $F_I(V)$ representa as forças inerciais que dependem das velocidades lineares e angulares e são enunciadas da seguinte maneira:

$$F_I(V) = m\begin{bmatrix} -v.r + w.q - x_G(q^2 + r^2) + y_G.p.q + z_G.p.r \\ -w.p + u.r - y_G(r^2 + p^2) + z_G.q.r + x_G.p.q \\ -u.q + v.p - z_G(p^2 + q^2) + x_G.p.r + y_G.q.r \\ y_G(-u.q + v.p) - z_G(-w.p + u.r) \\ z_G(-v.r + w.q) - x_G(-u.q + v.p) \\ x_G(-w.p + u.r) - y_G(-v.r + w.q) \end{bmatrix}$$

$$+\begin{bmatrix} 0 \\ 0 \\ 0 \\ (I_{zz} - I_{yy}).q.r - I_{xz}.p.q + I_{yz}(r^2 - q^2) + I_{xy}.p.r \\ (I_{xx} - I_{zz}).p.r - I_{xy}.q.r + I_{xz}(p^2 - r^2) + I_{yz}.p.q \\ (I_{yy} - I_{xx}).p.q - I_{yz}.p.r + I_{xy}(q^2 - p^2) + I_{xz}.q.r \end{bmatrix}$$

O termo F_G e F_E representam a força da gravidade e do empuxo, respectivamente, depois de feitas as transformações para o sistema de referência solidário ao corpo e é enunciado abaixo:

$$F_G = m.g.\begin{bmatrix} -\sin(\theta) \\ \sin(\varphi).\cos(\theta) \\ \cos(\varphi).\cos(\theta) \\ y_G.\cos(\varphi).\cos(\theta) - z_G.\sin(\varphi).\cos(\theta) \\ -x_G.\cos(\varphi).\cos(\theta) - z_G.\sin(\theta) \\ x_G.\sin(\varphi).\cos(\theta) + y_G.\sin(\theta) \end{bmatrix} \text{ e}$$

$$F_E = \rho.g.\nabla\begin{bmatrix} \sin(\theta) \\ -\sin(\varphi).\cos(\theta) \\ -\cos(\varphi).\cos(\theta) \\ -y_E.\cos(\varphi).\cos(\theta) + z_E.\sin(\varphi).\cos(\theta) \\ x_E.\cos(\varphi).\cos(\theta) + z_E.\sin(\theta) \\ -x_E.\sin(\varphi).\cos(\theta) - y_E.\sin(\theta) \end{bmatrix},$$

onde ∇ é o volume do veículo.

O termo $F_H(V)$ representa as forças hidrodinâmicas aplicadas ao casco em função das velocidades lineares e angulares, enunciadas abaixo, sendo ρ a densidade do fluido:

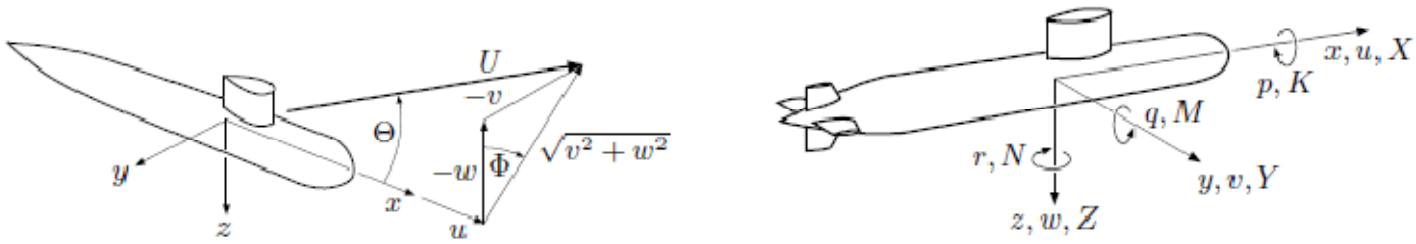
$$F_H(V) =$$

$$= \frac{\rho}{2}\begin{bmatrix} l^4\left(X'_{qq}q^2 + X'_{rr}r^2 + X'_{rp}rp\right)+l^3\left(X'_{vr}vr + X'_{wq}wq\right)+l^2(X'_{vv}v^2 + X'_{ww}w^2) \\ l^4\left(Y'_{p|p|}p|p| + Y'_{pq}pq\right)+l^3\left(Y'_{ur}ur + Y'_{up}up + Y'_{wp}wp\right) \\ \quad +l^2\left(Y'_{*}u^2 + Y'_{v}uv + Y'_{v|v|R}v\left|(v^2+w^2)^{\frac{1}{2}}\right|\right)+C_d\int_l h(x)v(x)(v^2(x)+w^2(x))^{\frac{1}{2}}\,dx \\ l^3\left(Z'_{q}uq + Z'_{vp}vp\right)+l^2\left(Z'_{*}u^2 + Z'_{w}uw + Z'_{|w|}u|w| + Z'_{ww}\left|w(v^2+w^2)^{\frac{1}{2}}\right|\right) \\ \quad -C_d\int_l b(x)w(x)(v^2(x)+w^2(x))^{\frac{1}{2}}\,dx \\ l^5\left(K'_{qr}qr + K'_{p|p|}p|p|\right)+l^4\left(K'_{p}up + K'_{r}ur + K'_{wp}wp\right)+l^3(K'_{*}u^2 + K'_{vR}uv) \\ \quad +l^3(u^2+v_S^2+w_S^2)\beta_S^2[K'_{4S}\sin(4\varphi_S) + K'_{8S}\sin(8\varphi_S)] - K_Q \\ l^5M'_{rp}rp + l^4M'_{q}uq + C_d\int_l xb(x)w(x)\{v^2(x)+w^2(x)\}^{\frac{1}{2}}\,dx \\ \quad l^3\left(M'_{*}u^2 + M'_{w}uw + M'_{w|w|R}w\left|(v^2+w^2)^{\frac{1}{2}}\right| + M'_{|w|}u|w| + M'_{ww}\left|w(v^2+w^2)^{\frac{1}{2}}\right|\right) \\ l^5N'_{pq}pq + l^4\left(N'_{p}up + N'_{r}ur\right) + l^3\left(N'_{*}u^2 + N'_{v}uv + N'_{v|v|R}v\left|(v^2+w^2)^{\frac{1}{2}}\right|\right) \\ \quad +C_d\int_l xh(x)v(x)(v^2(x)+w^2(x))^{\frac{1}{2}}\,dx \end{bmatrix}$$

O termo $F_C(V)$ representa as forças hidrodinâmicas aplicadas ao veículo em função das velocidades lineares e angulares e deflexões de superfícies de controle, enunciadas abaixo, sendo ρ a densidade do fluido:

$$F_C(V,\delta) = \frac{\rho}{2}\begin{bmatrix} l^2(X'_{\delta r\delta r}u^2\delta r^2 + X'_{\delta s\delta s}u^2\delta s^2 + X'_{\delta b\delta b}u^2\delta b^2) \\ l^2\left(Y'_{\delta r}u^2\delta r + Y'_{\delta r\eta}u^2\delta r\left(\eta - \frac{1}{C}\right)C\right) \\ l^2\left(Z'_{\delta s}u^2\delta s + Z'_{\delta b}u^2\delta b + Z'_{\delta s\eta}u^2\delta s\left(\eta - \frac{1}{C}\right)C\right) \\ l^3\left(K'_{\delta r}u^2\delta r + K'_{\delta r\eta}u^2\delta r\left(\eta - \frac{1}{C}\right)C\right) \\ l^3\left(M'_{\delta s}u^2\delta s + M'_{\delta b}u^2\delta b + M'_{\delta s\eta}u^2\delta s\left(\eta - \frac{1}{C}\right)C\right) \\ l^3\left(N'_{\delta r}u^2\delta r + N'_{\delta r\eta}u^2\delta r\left(\eta - \frac{1}{C}\right)C\right) \end{bmatrix}$$

APÊNDICE C – GLOSSÁRIO DO MODELO DINÂMICO



Como mostrado na figura acima, todas as grandezas são refenciadas solidariamente ao veículo.

$[x_G, y_G, z_G]$	Coordenadas do centro de gravidade no referencial solidário ao veículo
$[x_E, y_E, z_E]$	Coordenadas do centro de empuxo no referencial solidário ao veículo
I_{xx}, I_{yy}, I_{zz}	Momentos de inércia no eixo x, y e z
I_{xy}, I_{xz}, I_{yz}	Produtos de inércia nos planos xy, xz e yz
u	Componente da velocidade linear do veículo no eixo x
v	Componente da velocidade linear do veículo no eixo y
v_S	Componente da velocidade linear do leme horizontal de ré do veículo no eixo y
w	Componente da velocidade linear do veículo no eixo z
w_S	Componente da velocidade linear do leme horizontal de ré do veículo no eixo z
Θ_S	Ângulo de ataque Θ no leme horizontal de ré do veículo
ϕ_S	Ângulo de incidência ϕ no leme horizontal de ré do veículo
p	Componente da velocidade angular do veículo no eixo x
q	Componente da velocidade angular do veículo no eixo y
r	Componente da velocidade angular do veículo no eixo z
X	Componente da força hidrodinâmica aplicada no veículo no eixo x
Y	Componente da força hidrodinâmica aplicada no veículo no eixo y
Z	Componente da força hidrodinâmica aplicada no veículo no eixo z
K	Componente do momento hidrodinâmico aplicado no veículo no eixo x
M	Componente do momento hidrodinâmico aplicado no veículo no eixo y
N	Componente do momento hidrodinâmico aplicado no veículo no eixo z
$\theta\varphi\psi$	Angulos de Euler
$X_{\dot{u}}$	Coeficiente de massa adicional em $\dot{u}$ devido à variação de u
$Y_{\dot{v}}$	Coeficiente de massa adicional em $\dot{v}$ devido à variação de v
$Y_{\dot{p}}$	Coeficiente de massa adicional em $\dot{v}$ devido à variação de p
$Y_{\dot{r}}$	Coeficiente de massa adicional em $\dot{v}$ devido à variação de r
$Z_{\dot{w}}$	Coeficiente de massa adicional em $\dot{w}$ devido à variação de w
$Z_{\dot{q}}$	Coeficiente de massa adicional em $\dot{w}$ devido à variação de q

$K_{\dot{p}}$	Coeficiente de massa adicional em $\dot{p}$ devido à variação de p
$K_{\dot{v}}$	Coeficiente de massa adicional em $\dot{p}$ devido à variação de v
$K_{\dot{r}}$	Coeficiente de massa adicional em $\dot{p}$ devido à variação de r
$M_{\dot{w}}$	Coeficiente de massa adicional em $\dot{q}$ devido à variação de w
$M_{\dot{q}}$	Coeficiente de massa adicional em $\dot{q}$ devido à variação de q
$N_{\dot{v}}$	Coeficiente de massa adicional em $\dot{r}$ devido à variação de v
$N_{\dot{p}}$	Coeficiente de massa adicional em $\dot{r}$ devido à variação de p
$N_{\dot{r}}$	Coeficiente de massa adicional em $\dot{r}$ devido à variação de r
$X'_{qq} = \frac{X_{qq}}{\frac{\rho l^4}{2}}$	Derivada adimensional da força hidrodinâmica no eixo x devido a q^2
$X'_{rr} = \frac{X_{rr}}{\frac{\rho l^4}{2}}$	Derivada adimensional da força hidrodinâmica no eixo x devido a r^2
$X'_{rp} = \frac{X_{rp}}{\frac{\rho l^4}{2}}$	Derivada adimensional da força hidrodinâmica no eixo x devido a $r * p$
$X'_{vr} = \frac{X_{vr}}{\frac{\rho l^3}{2}}$	Derivada adimensional da força hidrodinâmica no eixo x devido a $v * r$
$X'_{wq} = \frac{X_{wq}}{\frac{\rho l^3}{2}}$	Derivada adimensional da força hidrodinâmica no eixo x devido a $w * q$
$X'_{vv} = \frac{X_{vv}}{\frac{\rho l^2}{2}}$	Derivada adimensional da força hidrodinâmica no eixo x devido a v^2
$X'_{ww} = \frac{X_{ww}}{\frac{\rho l^2}{2}}$	Derivada adimensional da força hidrodinâmica no eixo x devido a w^2
K_T	Empuxo gerado pelo hélice
$Y'_{p\|p\|} = \frac{Y_{p\|p\|}}{\frac{\rho l^4}{2}}$	Derivada adimensional da força hidrodinâmica no eixo y devido a $p\|p\|$
$Y'_{pq} = \frac{Y_{pq}}{\frac{\rho l^4}{2}}$	Derivada adimensional da força hidrodinâmica no eixo y devido a pq
$Y'_{ur} = \frac{Y_{ur}}{\frac{\rho l^3}{2}}$	Derivada adimensional da força hidrodinâmica no eixo y devido a ur
$Y'_{up} = \frac{Y_{up}}{\frac{\rho l^3}{2}}$	Derivada adimensional da força hidrodinâmica no eixo y devido a up
$Y'_{wp} = \frac{Y_{wp}}{\frac{\rho l^3}{2}}$	Derivada adimensional da força hidrodinâmica no eixo y devido a wp
$Y'_{*} = \frac{Y_{*}}{\frac{\rho l^2}{2}}$	Derivada adimensional da força hidrodinâmica no eixo y devido a u^2

$Y'_v = \frac{Y_v}{\frac{\rho l^2}{2}}$	Derivada adimensional da força hidrodinâmica no eixo y devido a uv
$Y'_{v\|v\|R} = \frac{Y_{v\|v\|R}}{\frac{\rho l^2}{2}}$	Derivada adimensional da força hidrodinâmica no eixo y devido a $v\left\|(v^2+w^2)^{\frac{1}{2}}\right\|$
$Z'_q = \frac{Z_q}{\frac{\rho l^3}{2}}$	Derivada adimensional da força hidrodinâmica no eixo z devido a uq
$Z'_{vp} = \frac{Z_{vp}}{\frac{\rho l^3}{2}}$	Derivada adimensional da força hidrodinâmica no eixo z devido a vp
$Z'_* = \frac{Z_*}{\frac{\rho l^2}{2}}$	Derivada adimensional da força hidrodinâmica no eixo z devido a u^2
$Z'_w = \frac{Z_w}{\frac{\rho l^2}{2}}$	Derivada adimensional da força hidrodinâmica no eixo z devido a uw
$Z'_{\|w\|} = \frac{Z_{\|w\|}}{\frac{\rho l^2}{2}}$	Derivada adimensional da força hidrodinâmica no eixo z devido a $u\|w\|$
$Z'_{ww} = \frac{Z_{ww}}{\frac{\rho l^2}{2}}$	Derivada adimensional da força hidrodinâmica no eixo z devido a $\left\|w(v^2+w^2)^{\frac{1}{2}}\right\|$
$K'_{qr} = \frac{K_{qr}}{\frac{\rho l^5}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo x devido a qr
$K'_{p\|p\|} = \frac{K_{p\|p\|}}{\frac{\rho l^5}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo x devido a $p\|p\|$
$K'_p = \frac{K_p}{\frac{\rho l^4}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo x devido a up
$K'_r = \frac{K_r}{\frac{\rho l^4}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo x devido a ur
$K'_{wp} = \frac{K_{wp}}{\frac{\rho l^4}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo x devido a wp
$K'_* = \frac{K_*}{\frac{\rho l^3}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo x devido a u^2
$K'_{vR} = \frac{K_{vR}}{\frac{\rho l^3}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo x devido a uv
K_Q	Torque gerado pelo hélice

$M'_{pr} = \frac{M_{pr}}{\frac{\rho l^5}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo y devido a pr
$M'_{q} = \frac{M_{q}}{\frac{\rho l^4}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo y devido a uq
$M'_{*} = \frac{M_{*}}{\frac{\rho l^3}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo y devido a u^2
$M'_{w} = \frac{M_{w}}{\frac{\rho l^3}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo y devido a uw
$M'_{w\|w\|R} = \frac{M_{w\|w\|R}}{\frac{\rho l^3}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo y devido a $w\left\|(v^2+w^2)^{\frac{1}{2}}\right\|$
$M'_{\|w\|} = \frac{M_{\|w\|}}{\frac{\rho l^3}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo y devido a $u\|w\|$
$M'_{ww} = \frac{M_{ww}}{\frac{\rho l^3}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo y devido a $\left\|w(v^2+w^2)^{\frac{1}{2}}\right\|$
$N'_{pq} = \frac{N_{pq}}{\frac{\rho l^5}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo z devido a pq
$N'_{p} = \frac{N_{p}}{\frac{\rho l^4}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo z devido a up
$N'_{r} = \frac{N_{r}}{\frac{\rho l^4}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo z devido a ur
$N'_{*} = \frac{N_{*}}{\frac{\rho l^3}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo z devido a u^2
$N'_{v} = \frac{N_{v}}{\frac{\rho l^3}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo z devido a uv
$N'_{v\|v\|R} = \frac{N_{v\|v\|R}}{\frac{\rho l^3}{2}}$	Derivada adimensional do momento hidrodinâmico no eixo z devido a $v\left\|(v^2+w^2)^{\frac{1}{2}}\right\|$
δr	Deflexão do leme vertical em radianos
δb	Deflexão do leme horizontal de vante em radianos
δs	Deflexão do leme horizontal de ré em radianos
η	Rendimento da superfície de controle
C	Fator de escala
$w(x)$	Velocidade local w em x do casco
$v(x)$	Velocidade local v em x do casco
$h(x)$	Altura (calado) do casco do veículo em x
$b(x)$	Largura (boca) do casco do veículo em x

C_d	Coeficiente de arrasto

BIBLIOGRAFIA

[1]. DANTAS J. L. D., "**Hardware in the Loop para a Simulação do Sistema de Navegação e Controle de Veículos Autônomos Submarinos"**, Trabalho de Conclusão de Curso, Escola Politécnica da Universidade de São Paulo, São Paulo, Brasil, 2008.

[2]. FELDMAN J., "**DTNSRDC Revised Standard Submarine Equations of Motion"**, Report of DTNSRDC/SPD-0393-09, 1979.

[3]. WATT, G. D., **Modeling and simulating unsteady six degrees of freedom submarine rising maneuvers**, Technical Report, Defense Research and Development Canada, 2007.

[4]. MARKOV, A. B., **A nonlinear six Degree-of-freedom flight simulation model v. 2: software documentation,** Technical Report, Defense Research and Development Canada,1990.

[5]. BRUTZMAN D. P.,KANAYAMA Y. E ZIDA M. J "**Integrated Simulation for Rapid Development of Autonomous Underwater Vehicles**", *Proceedings of the IEEE Symposium on Autonomous Underwater Vehicle Technology*, p 3-10, 1992.

[6]. SETO, M. L.; WATT, G. D., **Dynamics and control Simulator for Theseus AUV,** Defense Research and Development Canada, 2000.

[7]. ISE RESEARCH LIMITED, **SUBMO3: A Frequency domain simulation tool for submarine motion in waves, and its use with maneuvering models**, Defense Research and Development Canada, 2000.

[8]. LEPAGE Y. G. E HOLAPPA, K.W., **"Simulation and control of an autonomous underwater vehicle equipped with a vectored thruster"**, OCEANS 2000 MTS/IEEE Conference and Exhibition, p. 2129-2134 v. 3, 2000.

[9]. LANE D. M., FALCONER G. J., Randall G**., "Interoperability and Synchronisation of Distributed Hardware-in-the-Loop Simulation for Underwater Robot Development: Issues and Experiments"**, International Conference on Robotics & Automation, IEEE, Seoul, Korea, 2001.

[10]. SONG, F., AN, P. E., FOLLECO, A., "**Modeling and Simulation of Autonomous Underwater Vehicles: Design and Implementation"**, Journal of Oceanic Engineering, IEEE, vol. 28, no. 2, 2003.

[11]. GÖKTOGAN A. H., NETTLETON E., RIDLEY M., SUKKARIEH S., "**Real Time Multi-UAV Simulator"**, International Conference on Robotics & Automation, IEEE, Taipel, Taiwan,2003.

[12]. SHIXIANJUN, JIAKUN S., HONGXING L., "**Hardware-in-the-Loop Simulation Framework Design For a UAV Embedded Control System"**, Chinese Control Conference, IEEE, Harbin, Heilongjiang, 2006.

[13]. DA SILVA, H. M., "**Simulação com Hardware in the loop Aplicada a Veículos Submarinos Semi-Autônomos"**, Dissertação (Mestrado), 2008, Tese (Doutorado), Escola Politécnica, Universidade de São Paulo, São Paulo, 2008.

[14]. DANTAS, J. L. D., de BARROS, E. A., "**A real-time simulator for AUV development**", Proceedings of COBEM 2009, 20th International Congress of Mechanical Engineering, Gramado-RS, Brasil, 2009.

[15]. ANAKWA W.K. [ET AL.] "**Environments For Rapid Implementation Of Control Algorithms And Hardware In The Loop Simulation**", *IECON Proceedings (Industrial Electronics Conference)*, v 3, p 2288-2293, 2002

[16]. MACLAY D. "**Simulation gets into the loop**", *IEE Review*, v. 43, n. 3, p. 109, May 15, 1997.

[17]. SMITH, S. M., "**An Approach to Intelligent Distributed Control for Autonomous Underwater Vehicles**", *Proceedings of the IEEE Symposium on Autonomous Underwater Vehicle Technology*, p 105-111, 1994.

[18]. ORTIZ, A., "**Improving the safety of AUVs**", OCEANS '99 MTS/IEEE, v. 2, p. 979-984, 1999.

[19]. OLIVER, G. et al., "**RAO: a low cost AUV for testing**", OCEANS 2000 MTS/IEEE Conference and Exhibition, v. 1, p 397-401, 2000.

[20]. MARCO, D. B., HEALEY, A. J., "**Comand, Control and Navigation Experimental Results with the NPS ARIES AUV**", IEEE Journal of Oceanic Engineering, v. 26, N. 4, p 466-476, 2001.

[21]. WANG, I. et al., "**Modular Hardware Infrastructure fo Autonomous Underwater Vehicles**", OCEANS 2005. Proceedings of MTS/IEEE, v. 3, p 2652-2655, 2005.

[22]. WATANABE, K., "**An AUV Based Experimental System for the Underwater Technology Education**", OCEANS 2006-Asia Pacific, p 1-7, 2007.

[23]. ZHANG, H. W. et al., "**CAN bus based control system for autonomous underwater vehicle**", Jiqiren/Robot, v. 28, n. 4, p 448-452, 2006. Em língua chinesa.

[24]. CUFF, T. R., WALL, R.W., "**Support Platform and Communications to Manage Cooperative AUV Operations",** OCEANS 2006-Asia Pacific, 2007.

[25]. ACOSTA, G. G., "**Low-cost Autonomous Underwater Vehicle for pipeline and cable inspections**", International Symposium on Underwater Technology, UT 2007 - International Workshop on Scientific Use of Submarine Cables and Related Technologies 2007, p 331-336, 2007.

[26]. HASSAAN KHALID, M. et al., "**FATCAR-AUV: Fault tolerant control architecture of AUV**", Proceedings of International Bhurban Conference on Applied Sciences and Technology, IBCAST, p 161-167, 2007.

[27]. ZHANG, L. et al., "**Design and experiment of automatic pilot for long range AUVs**", 2008 3rd IEEE Conference on Industrial Electronics and Applications, p 1824-1827, 2008.

[28]. VOSS, W., "**A Comprehensible Guide to Controller Area Network**", 2nd ed., Greenfield, Massachusetts, United States of America, Copperhill Media Corporation, 2005, 176 p.

[29]. OKURA, J. H., "**Metodologia de qualificação de componentes eletrônicos**", 1995, 176 p. Dissertação (Mestrado) – Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, Campinas, 1995.

[30]. AMIANTI, G., "**Arquitetura de software aviônico de um VANT com requisitos de homologação**", 2008, 278 p. Dissertação (Mestrado) – Escola Politécnica da Universidade de São Paulo, Universidade de São Paulo, São Paulo, 2008.

[31]. TEXAS INSTRUMENTS INC., **"PCB Design Guidelines for Reduced EMI",** Application Report**,**1999, 23 p.