Universidade Federal
do Rio de Janeiro
Escola Politécnica

DESENVOLVIMENTO DO SOFTWARE PARA O POSICIONAMENTO DINÂMICO DO ROV LUMA

Gabriel da Silva Martins Loureiro

Projeto de Graduação apresentado ao Corpo Docente do Curso de Engenharia de Controle e Automação da Escola Politécnica da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro de Controle e Automação.

Orientador: Fernando Cesar Lizarralde

Rio de Janeiro
Fevereiro de 2017

DESENVOLVIMENTO DO SOFTWARE PARA O POSICIONAMENTO DINÂMICO DO ROV LUMA

Gabriel da Silva Martins Loureiro

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE CONTROLE E AUTOMAÇÃO.

Examinado por:

Prof. Fernando Cesar Lizarralde, D.Sc.

Prof. Liu Hsu, Docteur d'État

Eng. Alex Fernandes Neves, M.Sc.

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2017

da Silva Martins Loureiro, Gabriel

Desenvolvimento do Software para o Posicionamento Dinâmico do ROV LUMA / Gabriel da Silva Martins Loureiro. – Rio de Janeiro: UFRJ/Escola Politécnica, 2017.

XV, 107 p.: il.; 29, 7cm.

Orientador: Fernando Cesar Lizarralde

Projeto de Graduação – UFRJ/Escola Politécnica/ Curso de Engenharia de Controle e Automação, 2017.

Referências Bibliográficas: p. 84 – 87.

1. Posicionamento Dinâmico. 2. LUMA. 3. Controle Automático. 4. ROV. I. Cesar Lizarralde, Fernando. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Controle e Automação. III. Título.

Aos meus pais e irmão, por sempre me apoiarem.

Agradecimentos

Primeiramente, agradeço a Deus por tudo que me deu na vida. Tenho consciência que, nos momentos mais difíceis dessa etapa, Ele estava me guiando pelo melhor caminho.

Agradeço aos meus pais, Fernando e Nazaré, que sempre me apoiaram, ensinaram todos os valores para formação do meu caráter e deram o suporte que eu precisava para obter sucesso em todas as etapas da minha vida. Agradeço, também, ao meu irmão Bruno pelas risadas, partidas de Fifa, pelos sábados e domingos sagrados de jogos da Premier League e por ser meu melhor amigo. Eles são a base da minha vida e o motivo por todo o meu esforço. Durante a minha vida, minhas ações foram para orgulhá-los e recompensar todos os sacrifícios que eles fizeram por mim.

Ao meu orientador, Fernando Lizarralde, pelas dicas, críticas, ensinamentos e sugestões durante o desenvolvimento desse projeto e a minha formação acadêmica. Ao pessoal do Lead, em especial a Alex Neves, por toda a paciência em tirar as minha dúvidas, por todas as dicas e por ter sido um grande chefe, me fazendo despertar um gosto por programar que não imaginava ter. Também não posso deixar de agradecer ao meu amigo João Monteiro por todas as dicas e ensinamentos durante o projeto. Esse trabalho não seria possível sem essas pessoas.

Agradeço, também, as pessoas que, se no começo eram colegas de curso, hoje posso chamar de amigos, em especial a: Isabella, Luis Gustavo, Adriana, Guilherme, Rodrigo, Pedro, Fernanda, Bruno e Rob. Tive a sorte de conhecer grandes pessoas e posso dizer que eles tiveram uma enorme contribuição durante esses 5 anos. Foi uma grande experiência conhecê-los e espero levar essas amizades durante toda a minha vida.

Por fim, gostaria de agradecer aos meus amigos fora do curso. Mesmo que indiretamente, eles tiveram uma grande contribuição para a minha formação como engenheiro. Já são quase 10 anos contando com eles: João Paulo, José Felipe, João Renato, Maria Beatriz, Isabella, Pedro Henrique, Augusto, Duncan, Lucas e André.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Controle e Automação

DESENVOLVIMENTO DO SOFTWARE PARA O POSICIONAMENTO DINÂMICO DO ROV LUMA

Gabriel da Silva Martins Loureiro

Fevereiro/2017

Orientador: Fernando Cesar Lizarralde

Departamento: Engenharia de Controle e Automação

Este projeto tem como foco o desenvolvimento e implementação do sistema de *software* para o posicionamento dinâmico e uma interface gráfica para a instrumentação a bordo para o ROV LUMA. Para isso, foi necessário o estudo da modelagem cinemática e dinâmica do veículo e, também, as características do sensores utilizados. Dessa forma, foi crucial implementar um método de localização do ROV no ambiente. A localização é realizada medindo a distância e a orientação do robô em relação a uma parede à sua frente a partir da medição de um sonar.

O sistema foi desenvolvido em linguagem C++, utilizando ROS como *framework* de comunicação e *Qt* como ferramenta gráfica. Além disso, o sistema é validado utilizando um simulador das condições de operação do veículo, permitindo a análise do sistema implementado com realismo adequado.

Palavras-chave: ROV, LUMA, Posicionamento Dinâmico, ROS, Simulador, Software.

Abstract of Graduation Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Control and Automation Engineer

SOFTWARE DEVELOPMENT OF THE ROV LUMA'S DYNAMIC POSIOTINING

Gabriel da Silva Martins Loureiro

February/2017

Advisor: Fernando Cesar Lizarralde

Department: Control and Automation Engineering

This project focuses on the development and implementation of a software system for the dynamic positioning and an interface for onboard instrumentation for the LUMA ROV. For this, it was necessary to study the kinematic modeling and dynamics of the vehicle and also the characteristics of the sensors used. Thus, it was crucial to implement a method of locating the ROV in the environment. The location of the vehicle is accomplished by measuring the distance and orientation of the robot relative to a wall in front of it by the measurement performed by a sonar.

The system was developed in C ++ language, using ROS as a communication framework and Qt as a graphical tool. In addition, the system is validated using a simulator of the operating conditions of the vehicle, allowing analysis of the implemented system with adequate realism.

Keywords: ROV, LUMA, Dynamic Positioning, ROS, Simulator, Software.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

ARW	*Antireset Windup*, p. 62
DOF	*Degrees of Freedom*, p. 8
GPCB	*General Package Component Base*, p. 39
GPCR	*General Package Component Robot*, p. 39
GPS	*Global Positioning System*, p. 21
GPT	*General Package Tool*, p. 39
GSCAR	*Grupo de Simulação e Controle em Automação e Robótica*, p. 1
HROV	*Hybrid Remotely Operated Vehicle*, p. 3
IMU	*Inertial Measurement Unit*, p. 21
LPCB	*Luma Package Component Base*, p. 39
LPCR	*Luma Package Component Robot*, p. 39
LPT	*Luma Package Tool*, p. 39
LUMA	*Light Underwater Mobile Asset*, p. 3
MISO	*Multiple Input Single Output*, p. 55
P-PI	*Proporcional-Proporcional Integrador*, p. 8
RANSAC	Random Sample Consensus, p. 33
ROS	*Robot Operating System*, p. 6
ROV	*Remotely Operated Vehicle*, p. 1
SISO	*Single Input Single Output*, p. 55
SNAME	*Society of Naval Architects & Marine Engineers* , p. 13
Sonar	*Sound Navigation and Ranging*, p. 24

Capítulo 1

Introdução

1.1 Motivação

Veículos operados remotamente (ROVs) possuem um papel importante na área de exploração de ambientes subaquáticos. Esses veículos estão substituindo o homem na realização de tarefas consideradas de alto risco para a vida por serem capazes de realizar tarefas em profundidades impraticáveis para o ser humano. ROVs vem sendo utilizados minuciosamente na pesquisa de fenômenos subaquáticos, estudos da fauna e flora oceânica. Também apresentam funcionalidades na área petrolífera para exploração de petróleo como inspeção visual, corte de cabos, aperto de parafuso, reparo de estruturas e perfuração. Na área militar são usados para a recuperação de torpedos e contramedidas de minas (McFarlane (2000)).

No meio acadêmico, o interesse pelo estudo da vida marinha aumentou consideravelmente no últimos anos devido a possibilidade de alcançar áreas antes não alcançáveis. Nesse aspecto, há o ROV LUMA (*Light Underwater Mobile Asset*), desenvolvido pelo Grupo de Simulação e Controle em Automação e Robótica (GSCAR) do Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia em parceria com a AMPLA.

Apesar da utilização de ROVs ser datada há bastante tempo, a operação ainda é árdua para o operador. A principal causa dessa dificuldade é a complexidade do comportamento dinâmico do ROV (Hsu et al. (2000*a*)). De modo geral, operar os graus de liberdade do veículo simultaneamente é complicado para o operador. Nesse sentido, busca-se simplificar a operação dos veículos. Conforme Hsu et al. (2000*a*), uma alternativa é o controle automático de posição, de modo que o controlador atua no nível mais baixo de um sistema hierárquico, enquanto o piloto executaria tarefas mais complexas de alto nível.

1.2 Objetivos

O principal objetivo deste projeto é apresentar o desenvolvimento e implementação de um software, com interface gráfica para a instrumentação a bordo, de posicionamento dinâmico do ROV LUMA. Também trata do desenvolvimento de um algoritmo para a localização do veículo no ambiente a partir das medições do estado do veículo, ou seja, a posição e orientação do robô em relação a uma referência. É realizado o controle de três graus de liberdade: profundidade, rumo e a distância no eixo x em relação a uma referência fixa. Dessa forma, a operação torna-se parcialmente autônoma. Para a validação do sistema, foi feita a integração com um simulador nas condições de operação do robô.

1.3 Veículo Submarino Operado Remotamente

ROV é um robô submarino, operado remotamente, que permite observação à distância do fundo do mar e estruturas submarinas. São equipados de câmeras de vídeos para captura de imagens e outros sensores que fornecem dados que auxiliam a navegação. São ligados a um navio ou outra estrutura na superfície por um cabo umbilical que tem como função comunicação bidirecional e, também, o transporte de carga para o veículo.

O primeiro ROV é datado na década de 1950 e foi chamado "o POODLE ". Contudo, a marinha americana deu o primeiro passo para um sistema operacional a fim de localizar torpedos que foram perdidos no fundo do mar (Marzbanrad et al. (2011)). Desde então, diferentes projetos foram realizados com objetivos abrangentes, principalmente com o avanço das indústrias *offshore* de óleo e gás. Contudo, a operação desses veículos são complexas. A infraestrutura necessária para a operação requer um conjunto de equipamentos ocasionando no aumento dos custos (Ribas et al. (2010)).

Segundo Brun (2012), o mercado de ROV apresenta três aspectos dinâmicos notáveis: o aumento do interesse por mini ou pequenos ROVs devido ao custo reduzido e aumento de funcionalidades; o aumento do número de sensores que podem ser colocados no veículos melhorando a navegação; redução dos custos da plataforma de operação em relação ao custo dos instrumentos. A redução do custo relativo indicaria que a tecnologia estaria atingindo a maturidade.

A integração de múltiplos sensores e sistemas demanda o desenvolvimento de uma interface gráfica. Existe uma tendência em simplificar as informações colocando-as em uma única tela. Dito isso, há o aumento do interesse por uma maior automação nos veículos. Os ROVs atuais apresentam controle de posição automático para profundidade, rumo ou movimento ao longo de um plano. O controle efetivo exige

um conjunto de sinais para manter as posições desejadas.

Na literatura, há diversos métodos de controle. Em Aras et al. (2012) é utilizado um sensor inercial e um sensor de profundidade. Yen and Guo (2016) e Maalouf et al. (2012) apresentam veículos subaquáticos desenvolvidos com a finalidade de seguir uma parede. Na literatura, alguns trabalhos apresentam os problemas relacionados ao controle do posicionamento dinâmico, como De Souza and Maruyama (2007); Dukan et al. (2011); Hsu et al. (2000*b*); Qingjun et al. (2016). Uma tendência é o desenvolvimento de veículos híbridos, denominado HROV, do inglês *Hybrid Remotely Operated Vehicle*. Estes são veículos que apresentam duas configurações de operação: autônoma e remota.

1.4 ROV LUMA

O ROV Luma, Figura 1.1, trata-se um ROV, desenvolvido pelo Grupo de Simulação e Controle em Automação e Robótica (GSCAR) do Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia em parceria com a AMPLA. É um veículo não tripulado do tipo *open-frame* (estrutura aberta) e, inicialmente, tinha como finalidade a inspeção de túneis subaquáticos em usinas hidrelétricas (Carneiro et al. (2006)).

Com o avanço do interesse em estudos da vida marinha, no projeto PROANTAR-CNPq o robô passou a ser testado para a exploração da fauna e flora marinha na Antártica, captura de imagens da geografia oceânica na Baía do Almirantado (Antártica). Na primeira versão da Luma, a profundidade máxima era de 150m.

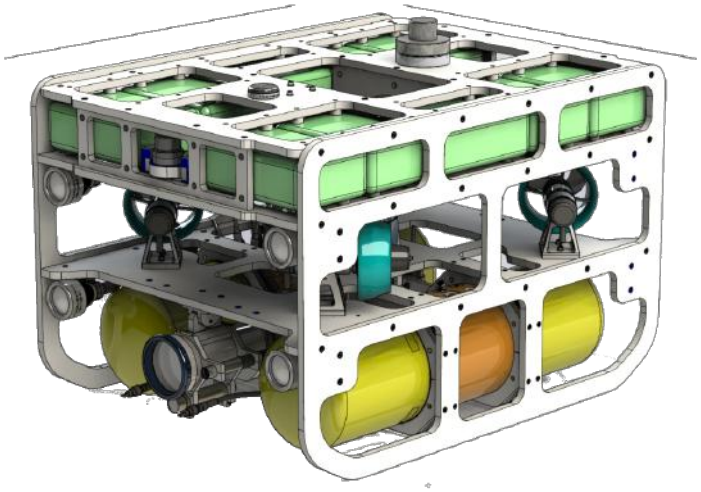


Figura 1.1: Imagem renderizada do ROV LUMA.

Recentemente, o projeto foi reformulado, apresentando, agora, as seguintes ca-

racterísticas:

- Profundidade até 1000m
- Sensores e tecnologia de última geração
- Controle de posicionamento de câmeras
- Vídeo de alta qualidade
- Comunicação serial padrão RS485
- Eletrônica embarcada
- Computador com barramento PC104 embarcado

O ROV possui 10 flutuadores com a finalidade de aumentar o volume do veículo nesta parte sem o aumento do peso. O posicionamento dos flutuadores foi feito de modo que o deslocamento do fluxo de água seja facilitado. Os flutuadores são ajustados de acordo com o centro de massa do veículo para que fique alinhado ao sistema inercial. Os componentes mais pesados da Luma são posicionados na parte inferior. Dentro desses componentes, estão:

- 3 câmeras fixas de vídeo de definição padrão (SD);
- 1 câmera de vídeo de definição padrão com controle *Pan & Tilt*;
- 1 câmera de vídeo de alta definição (HD);
- *laser* para ser usado como referência na câmera HD;
- 1 sonar;
- 4 luminárias;
- 1 altímetro;
- 1 unidade de navegação inercial;
- 1 sonda para aquisição de características do ambiente subaquático;
- 5 propulsores com *encoder*.

A Figura 1.2 apresenta uma diagrama geral da comunicação do ROV Luma.

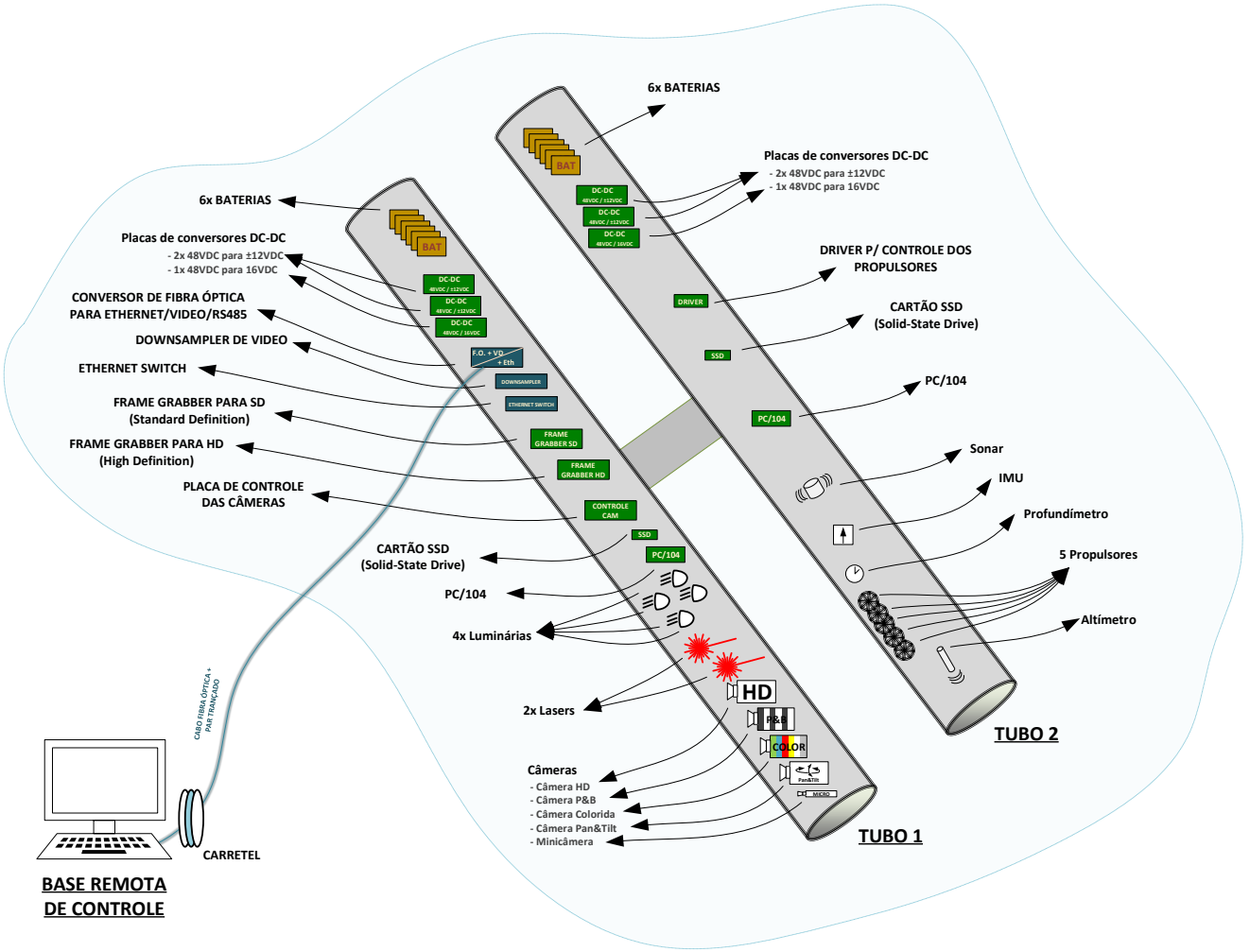


Figura 1.2: Diagrama geral da Luma.

Para o controle, os equipamentos a bordo são utilizados para medição do estado do ROV. O profundímetro para o sistema de controle de profundidade, a unidade de medição inercial para o sistema de controle de rumo e o auxílio a localização e navegação e o sonar para mapeamento do ambiente, desvio de obstáculo e controle da distância em relação a um marco físico, como uma parede.

A comunicação no sistema de controle é feita da seguinte forma: o sistema desenvolvido para o controle transmite o valor de força dos 4 graus de liberdade (x,y,z e rumo) para o microcontrolador embarcado através de uma mensagem de padrão RS485. Ao receber a mensagem, o microcontrolador atua nos propulsores. Além disso, também envia mensagem que contém novos valores de profundidade e orientação para a base. A figura 1.3 exibe essa comunicação.

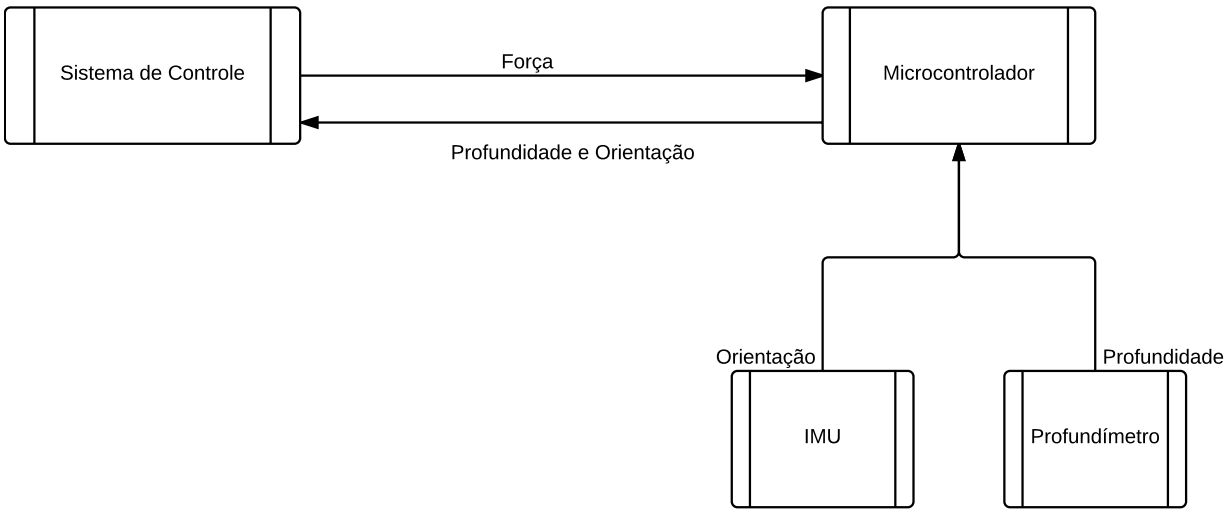


Figura 1.3: Comunicação entre base e microcontrolador.

Na versão mais recente do ROV, foi desenvolvida uma nova *interface* gráfica. A principal característica dessa interface é o caráter modular de modo que esta pode ser utilizada por outros ROVs. Devido a essa característica, foi necessário adotar:

- Linux/Ubuntu como sistema operacional,
- Qt como *framework* de interface gráfica,
- Robot Operating System como *middleware* de comunicação.

Além disso, foi estudada a modelagem dinâmica da Luma (Goulart (2007)) e as características dos sensores necessários para o controle.

1.5 Robot Operating System - ROS

ROS é um *framework* flexível para o desenvolvimento de software com aplicações robóticas. É um conjunto de ferramentas, bibliotecas e convenções com o objetivo de simplificar a tarefa de desenvolver o comportamento de robôs para multiplataformas. Foi desenvolvido de modo distribuído e modular, de modo que cabe ao usuário definir o que usará do sistema ROS. Uma funcionalidade importante é a capacidade de trocar mensagens entre computadores na mesma rede. As principais características do ROS podem ser resumidas em:

- Comunicação ponto a -a-ponto *peer-to-peer*;
- Gratuito e aberto;
- Multilinguagens de programação;
- Baseado em ferramentas;

Como apresentado em (Quigley et al., 2009), não é um sistema operacional, mas apresenta funcionalidades semelhantes como realizar mensagens entre processos. É uma estrutura de comunicação que age como uma camada do sistema operacional nativo nos computadores conectados em rede. Essa comunicação é feita por meio de mensagens transmitidas através dos chamados nó de ROS. Os conceitos dessa comunicação são explicados a seguir:

- Nós: nós são processos que executam algum tipo de computação. Um sistema de controle de um robô geralmente tem múltiplos nós. Um nó de ROS é escrito utilizando bibliotecas como *roscpp* (C++) ou *rospy* (Python).
- Master: fornce registro e busca de nomes. Sem o Master, os nós não seria capazes de se encontrar, trocar mensagens ou chamar serviços.
- Mensagens: consiste apenas de uma estrutura de dados. Suportam tipos primitivos ou outras estruturas aninhadas.
- Tópicos: as mensagens são transmitidas por meio de um mecanismo de *publish/subscribe*. Um nó envia uma mensagem de ROS publicando-a em um tópico. O tópico é o nome que identifica o conteúdo da mensagem. Outro nó que queira utilizar os dados dessa mensagem publicada deve se subscrever ao tópico apropriado. Nesse tipo de comunicação, podem haver múltiplos *publishers* e *subscribers* para um único tópico ou um único nó pode publicar ou subscrever diferentes tópicos. Em geral, *publisher* e *subscriber* não são conscientes um do outro. A figura 1.4 demonstra esse modo de comunicação.

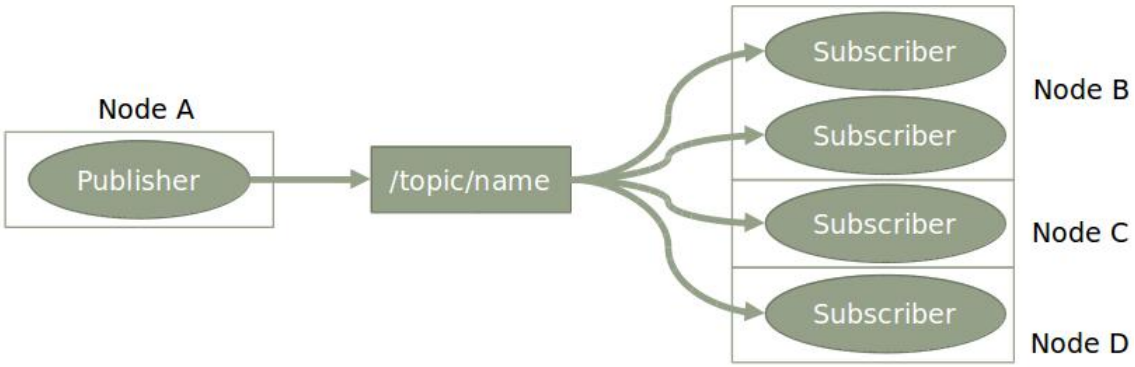


Figura 1.4: Comunicação por Tópicos

- Serviços: apesar do sistema *publish/subscribe* ser bastante flexível, essa comunicação em uma direção não é adequada para casos de pedido/resposta, que normalmente são necessários em uma sistema distribuído. Nesse aspecto, essa forma de comunicação é feita por meio de serviços. Estes são definidos por um par de mensagens, sendo uma para pedido, denominada *request* e outra para a resposta, chamada de *response*. Um nó oferece um serviço através de um nome e um cliente pode utilizar esse serviço enviando esses *request* e esperando a resposta.

Figura 1.5: Comunicação por Serviços

Ainda, é ressaltada outra importante ferramenta do ROS: os *bags*. *Bags* são formas de salvar e repetir os dados de uma mensagem. Dessa forma, são importantes para armazenar dados de sensores, que podem ser difíceis de coletar, mas são necessários para o desenvolvimento, testes de algoritmos e realizar análises de falhas em operações.

Na Literatura, já há pesquisas de veículos submarinos desenvolvidos em ROS (Lawrance et al. (2016), Tipsuwan and Hoonsuwan (2015), DeMarco et al. (2011)).

1.6 Estrutura do Texto

No Capítulo 1 foram apresentados a motivação do projeto e os objetivos que este pretende alcançar. Consequentemente, é apresentada uma explicação sobre veículos operados remotamente e o ROV Luma. Finalizando, é apresentado o ROS,

a *framework* de comunicação utilizada no algoritmo e interface desenvolvidos no trabalho.

No Capítulo 2 são apresentados o modelo cinemático do veículo e parametrização para descrever o estado do veículo. Ainda, apresenta as equações dinâmicas que regem o modelo. Também é apresentado a modelagem dos propulsores e a estratégia de desacoplamento utilizada.

O Capítulo 3 descreve os sensores utilizados para a medição dos estados do ROV Luma. É detalhada a transformada de coordenadas para cada dispositivo. Apresenta de forma clara e direta o levantamento das características do sonar com o intuito do desenvolvimento de um algoritmo que simulasse esse comportamento. Além disso, também é definido o método de localização do veículo no ambiente. Esse método é fundamental para manter a distância até uma referência fixa a frente do robô.

O Capítulo 4 aborda a interface gráfica para operação remota e o posicionamento dinâmico do ROV Luma. É feita uma descrição completa da arquitetura do *software* do robô e um detalhamento das ferramentas utilizadas no controle. Além disso, será explicado o funcionamento da comunicação via ROS entre os diversos componentes que fazem parte do sistema de controle.

O Capítulo 5 apresenta o sistema de controle. Foi abordado todas condições necessárias para o controle real do ROV Luma. Utilizando uma estratégia de controle linear P-PI, foi descrito o controlador para os 3 DOF.

O Capítulo 6 descreve os simulador utilizado para validar o sistema desenvolvido. Além disso, também é exibido o resultado das simulações realizadas, com base em diferentes parâmetros.

O Capítulo 7 apresenta as conclusões do projeto como também propostas para trabalhos futuros.

Capítulo 2

Modelagem do ROV

Para o projeto do controle de cada grau de liberdade do robô, é fundamental a análise da modelagem do veículo. Nesse caso, o Capítulo aborda os modelos cinemáticos e dinâmicos do ROV. Esses modelos são essenciais para o projeto de um sistema de posicionamento dinâmico.

Desse modo, inicialmente é apresentado o modelo cinemático de um ROV, explicando o sistema de coordenadas inercial e móvel. É ressaltado, também, a parametrização do ROV Luma.

Apresenta-se o modelo dinâmico de um ROV, equações que regem o movimento do veículo. Além disso, é feita a modelagem dos propulsores utilizados na Luma. Nesse caso, é apresentado um modelo não-linear, sem o efeito da compensação. Ainda é definido o conceito de matriz de acoplamento e desacoplamento, sendo esta última necessária para o projeto de um controle linear para cada DOF.

2.1 Modelo Cinemático

Para a realização do controle do ROV, é necessário, primeiramente, analisar seu modelo cinemático. Este modelo relaciona a velocidade do veículo nas coordenadas do corpo para a velocidade nas coordenadas inerciais. Tendo isso em mente, para a modelagem do veículo, este é considerado um corpo rígido.

Um corpo rígido é descrito completamente no espaço com a sua posição e sua orientação em relação a um sistema de coordenadas fixo. Dessa forma, considerando um sistema de coordenadas ortonormal inercial como $\bar{E}_i = [\vec{x}_i \ \vec{y}_i \ \vec{z}_i]$, a posição de um ponto O' no corpo rígido em relação ao sistema coordenadas inercial é expressa por:

$$\vec{o'} = o'_x\vec{x}_i + o'_y\vec{y}_i + o'_z\vec{z}_i \tag{2.1}$$

onde a tripla $[o'_x \ o'_y \ o'_z]^T$ corresponde ao vetor $\vec{o'} \in \mathbb{R}^3$ no sistema de coordenadas do

corpo. A posição pode ser escrita, então, como:

$$\vec{o'} = \begin{bmatrix} o'_x \\ o'_y \\ o'_z \end{bmatrix}. \tag{2.2}$$

Consequentemente, considerando, agora, o sistema de coordenadas ortonormal do corpo $\bar{E} = [\vec{x}_b \ \vec{y}_b \ \vec{z}_b]$, tem-se:

$$\vec{x}_b = \bar{E}_i x_{ib} \tag{2.3}$$

$$\vec{y}_b = \bar{E}_i y_{ib} \tag{2.4}$$

$$\vec{z}_b = \bar{E}_i z_{ib} \tag{2.5}$$

sendo x_{ab}, y_{ab}, z_{ab} as coordenadas de $\vec{x}_b, \vec{y}_b, \vec{z}_b$ no sistema de coordenadas $\bar{E}_i$. Com isso, define-se a matriz de rotação:

$$R_{ib} = \begin{bmatrix} \vec{x}_i \cdot \\ \vec{y}_i \cdot \\ \vec{z}_i \cdot \end{bmatrix} \begin{bmatrix} \vec{x}_b & \vec{y}_b & \vec{z}_b \end{bmatrix} = \begin{bmatrix} (\vec{x}_i \cdot \vec{x}_b) & (\vec{x}_i \cdot \vec{y}_b) & (\vec{x}_i \cdot \vec{z}_b) \\ (\vec{y}_i \cdot \vec{x}_b) & (\vec{y}_i \cdot \vec{y}_b) & (\vec{y}_i \cdot \vec{z}_b) \\ (\vec{z}_i \cdot \vec{x}_b) & (\vec{z}_i \cdot \vec{y}_b) & (\vec{z}_i \cdot \vec{z}_b) \end{bmatrix} \tag{2.6}$$

Por fim, a configuração do corpo em relação ao inercial é definida pelo par (p_{ib}, R_{ib}). Tem-se que $p_{ib} \in \mathbb{R}^3$ define a posição do corpo em relação a referência inercial e $R_{ib} \in SO(3)$. $SO(3)$ é o conjunto de todas as matrizes $\mathbb{R}^{3\times3}$ chamado de Grupo Especial Ortonormal de dimensão 3, definido por:

$$SO(3) = \left\{ R \in \mathbb{R}^{3\times3} : R^T R = I \text{ e } \det(R) = 1 \right\} \tag{2.7}$$

Agora, dado o par (p_{ib}, R_{ib}) e um ponto q_b no sistema $\bar{E}_b$, a representação deste ponto em $\bar{E}_i$ é dado pela operação:

$$q_i = p_{ib} + R_{ib} q_b \tag{2.8}$$

Pode-se, também, utilizar a transformação homogênea na qual a equação de q_i é escrita na forma compacta:

$$\begin{bmatrix} q_i \\ 1 \end{bmatrix} = \begin{bmatrix} R_{ib} & p_{ib} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} q_b \\ 1 \end{bmatrix} \tag{2.9}$$

e define-se:

$$\bar{q} = \begin{bmatrix} q_i \\ 1 \end{bmatrix}, \quad T_{ib} = \begin{bmatrix} R_{ib} & p_{ib} \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times4}$$

Uma parametrização para orientação bastante utilizada é por meio dos ângulos de Euler, em que qualquer rotação é descrita pela combinação de três rotações elementares ao redor dos eixos x,y,z, que são definidos:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Há diferentes definições para os ângulos de Euler, porém, em veículos submarinos, é comum o uso dos ângulos de Euler ZYX usualmente denominados de ângulo de jogo (ϕ, *roll*), ângulo de arfagem (θ,*pitch*) e ângulo de rumo (ψ,*yaw*). Dessa forma, a matriz resultante

$$R_{ib} = R_z(\psi)R_y(\theta)R_x(\phi)$$

rotaciona o sistema de referência do corpo rígido para o sistema de referência inercial. De forma completa, tem-se:

$$\begin{aligned} R_{ib} &= \begin{bmatrix} c(\psi)c(\theta) & -s(\psi)c(\phi) + c(\psi)s(\theta)s(\phi) & s(\psi)s(\phi) + c(\psi)s(\theta)c(\phi) \\ s(\psi)c(\theta) & c(\psi)c(\phi) + s(\psi)s(\theta)s(\phi) & -c(\psi)s(\phi) + s(\psi)s(\theta)c(\phi) \\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi) \end{bmatrix} \\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \end{aligned} \tag{2.10}$$

onde $c(\cdot) := \cos(\cdot)$ e $s(\cdot) := \sin(\cdot)$. Dada essa matriz de rotação, é possível obter os ângulos de *roll, pitch e yaw* por:

$$\phi = atan2(r_{32}, r_{33}), \tag{2.11}$$

$$\theta = atan2(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}), \tag{2.12}$$

$$\phi = atan2(r_{21}, r_{11}) \tag{2.13}$$

Desse modo, no ROV Luma, o sistema de coordenadas inercial é considerado na superfície. A frente do veículo é considerado x positivo, z positivo da superfície para

o fundo do oceano e y positivo para o lado segundo a regra da mão direita, como na Figura 2.1:

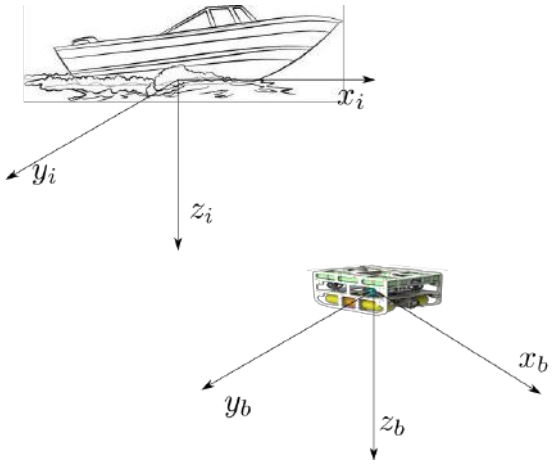


Figura 2.1: Sistema de coordenadas inicial

No modelo cinemático do ROV, a posição e orientação são dados no sistema de coordenadas inercial, enquanto as velocidades lineares e angulares são representadas no sistema de coordenadas do veículo. Desta maneira, o estado do veículo é definido por 12 variáveis, tal que:

$$x_v = \left[p_{ib}^T \ \Phi_{ib}^T\right]^T \tag{2.14}$$

$$\dot{x}_v = \left[\ \dot{p}_{ib}^T \ \dot{\Phi}_{ib}^T\right]^T \tag{2.15}$$

sendo p_{ib} e Φ_{ib} a posição e orientação na referência inercial, definidos, respectivamente por:

$$p_{ib} := \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \Phi_{ib} := \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$

e as velocidades na referência do corpo:

$$v_b := \begin{bmatrix} u \\ v \\ w \end{bmatrix}, \quad \omega_b := \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

A nomenclatura utilizada adotada é pelo padrão SNAME (*Society of Naval Architects & Marine Engineers*), conforme a tabela 2.1

Tabela 2.1: Tabela de nomenclatura padrão SNAME

descrição	velocidade lineares e angulares no sistema do corpo	posição e ângulos no sistema inercial
translação no eixo x	u	x
translação no eixo y	v	y
translação no eixo z	w	z
rotação no eixo x	p	ϕ
rotação no eixo y	q	θ
rotação no eixo z	r	ψ

As velocidades angulares em relação ao sistema de coordenadas fixo podem ser representadas como as derivadas dos ângulos de orientação. Do ponto de vista físico, ω_b é mais intuitivo que $\dot{\Phi}$, porém, a integral ω_b não tem interpretação física. Deste modo, a relação entre ω_b e $\dot{\Phi}$ é estabelecida pelo Jacobiano da Representação da forma:

$$\dot{\Phi} = J_R(\Phi)\omega_b \tag{2.16}$$

Para determinar o jacobiano da representação, é necessário definir dois sistemas de coordenadas auxiliares $E_1 = [\vec{x}_1, \vec{y}_1, \vec{z}_1]$ e $E_2 = [\vec{x}_2, \vec{y}_2, \vec{z}_2]$. Como no ROV Luma a representação é feita pelos ângulos *roll-pitch-yaw*, então o *roll* ocorre ao redor de $\vec{x}_i = \vec{x}_1$ e leva do sistema inercial E_i até E_1. O *pitch* ocorre ao redor de $\vec{y}_1$, levando de E_1 até E_2. Por fim, o yaw ocorre em $\vec{z}_2$ e leva de E_2 até E_b. Logo:

$$\begin{aligned}
\vec{\omega} &= \dot{\phi}\vec{x}_i + \dot{\theta}\vec{y}_1 + \dot{\psi}\vec{z}_2 \\
\omega &= \begin{bmatrix}\dot{\phi}\\0\\0\end{bmatrix} + (R_x)^T \begin{bmatrix}0\\\dot{\theta}\\0\end{bmatrix} + (R_y R_x)^T \begin{bmatrix}0\\0\\\dot{\psi}\end{bmatrix} \\
\omega &= \begin{bmatrix}1\\0\\0\end{bmatrix}\dot{\phi} + \begin{bmatrix}0\\\cos(\phi)\\-\sin(\phi)\end{bmatrix}\dot{\theta} + \begin{bmatrix}-\sin(\theta)\\\sin(\phi)\cos(\theta)\\\cos(\phi)\cos(\theta)\end{bmatrix}\dot{\psi} \\
w_b = J_R^{-1}\dot{\Phi} &= \begin{bmatrix}1 & 0 & -\sin(\theta)\\0 & \cos(\phi) & \sin(\phi)\cos(\theta)\\0 & -\sin(\phi) & \cos(\phi)\cos(\theta)\end{bmatrix}\begin{bmatrix}\dot{\phi}\\\dot{\theta}\\\dot{\psi}\end{bmatrix}
\end{aligned} \tag{2.17}$$

Logo,

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \frac{1}{\cos(\theta)} \begin{bmatrix} 1 & \sin(\phi)\sin(\theta) & \cos(\phi)\sin(\theta) \\ 0 & \cos(\phi)\cos(\theta) & -\sin(\phi)\cos(\theta) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \omega_b \tag{2.18}$$

Nota-se que o Jacobiano para essa representação é singular quando

$$\theta = \pm\pi/2.$$

No entanto, considerando o ROV Luma, o projeto foi realizado tal que o ângulo θ permaneça suficientemente pequeno de modo que essa singularidade e os problemas relacionados a ela são evitados. Por sua vez, também é necessário transformar as velocidades lineares para a referência fixa. Essa transformação é feita com o uso da equação 2.10, tal que:

$$\dot{p}_{ib} = R_{ib} v_b \tag{2.19}$$

Portanto, reescrevendo a equação 2.15, o modelo cinemático do veículo é dado por:

$$\dot{x}_v = \begin{bmatrix} R_{ib} v_b & J_R w_b \end{bmatrix}^T \tag{2.20}$$

2.2 Modelo Dinâmico

Para a realização do controle de um ROV, é fundamental conhecer corretamente a modelagem dinâmica do veículo. Neste projeto, será apresentado brevemente a modelagem do ROV Luma, que foi estudado detalhadamente em Goulart (2007) e Cunha (1992). Ambos consideraram o veículo submerso em fluído ideal. Com isso, o modelo do veículo é definido por:

$$(M_{cr} + M_A)\dot{\vec{v}} + (C_{CR}(v) + C_A(v) + D_L)\vec{v} + D_Q\vec{v}|\vec{v}| - W - \epsilon = \tau_p \tag{2.21}$$

onde os parâmetros são de acordo com a seguinte tabela:

Tabela 2.2: Parâmetros da dinâmica do ROV

parâmetros	descrição
$\vec{v}$	vetor velocidade linear e angular do veículo no sistema do corpo
M_{CR}	matriz de inércia do corpo rígido
C_{CR}	matriz de Coriolis e centrípeta do corpo rígido
M_A	matriz de inércia adicional
C_A	matriz de Coriolis e centrípeta adicional
D_L	matriz de arrasto linear
D_Q	matriz de arrasto quadrática
W	vetor peso-flutuação
τ_p	vetor força-momento aplicado pelos propulsores
ϵ	vetor das pertubações devidas à dinâmica do cabo e à correnteza marinha

2.2.1 Modelagem dos Propulsores

Um modelo exato dos propulsores é necessário para que se possa determinar corretamente a força total e o momento gerado pela combinação vetorial de todos os propulsores.

No modelo estático adotado em Cunha (1992). o empuxo e o momento axial gerados por cada propulsor são dados por:

$$F_{pi} = C_T^*(\sigma)\frac{\rho}{8}\left[v_{wi}^2 + (0,7\pi n_{pi}D^2)\right]\pi D^2 M_{pi} = C_Q^*(\sigma)\frac{\rho}{8}\left[v_{wi}^2 + (0,7\pi n_{pi}D^2)\right]\pi D^3 \tag{2.22}$$

onde os parâmetros são descritos pela tabela a seguir

Tabela 2.3: Parâmetros da dinâmica dos propulsores

parâmetros	descrição
v_{wi}	velocidade da água que entra no disco do i-ésimo propulsor $[m/s]$
D	diâmetro da hélice dos propulsores $[m]$
ρ	massa específica da água $[Kg/m^3]$
n_{pi}	velocidade de rotação do i-ésimo propulsor $[rps]$
C_T^* e C_Q^*	coeficientes que relacionam n_{pi} com F_{pi} e M_{pi}

Os parâmetros C_T^* e C_Q^* são:

$$C_T^* = \frac{8F_{pmax}}{0,7\rho\pi^3 n_{pmax}^2 D^4}, \quad C_Q^* = \frac{8M_{pmax}}{0,7\rho\pi^3 n_{pmax}^2 D^5}$$

Assim, o componente das forças e momentos no sistema de coordenadas do ROV é dado por:

$$F_p = \begin{bmatrix} F_{px} \\ F_{py} \\ F_{pz} \end{bmatrix} = \sum_{i=1}^{n} F_{pi} P_{pi}, \tag{2.23}$$

$$M_p = \begin{bmatrix} M_{px} \\ M_{py} \\ M_{pz} \end{bmatrix} = \sum_{i=1}^{n} \left[M_{pi} P_{pi} + R_{pi} \times (F_{pi} P_{pi}) \right] \tag{2.24}$$

onde R_{pi} e P_{pi} são, respectivamente, o vetor que presenta o centro de empuxo do i-ésimo propulsor e vetor unitário que define a direção do empuxo do i-ésimo propulsor. Para isso, são necessárias algumas hipóteses conforme Carneiro et al. (2006) e Goulart (2007):

- O acoplamento cruzado devido à interferência do fluxo de água de um propulsor em outro é desprezado;
- v_{wi} é assumida como sendo a componente da velocidade relativa do veículo, paralela ao eixo de rotação do hélice definido por P_{pi} ;
- A inércia do propulsor é desprezível

Sendo assim, a posição e orientação dos propulsores do veículo determinam a contribuição ao movimento do veículo. Com base na figura 2.2

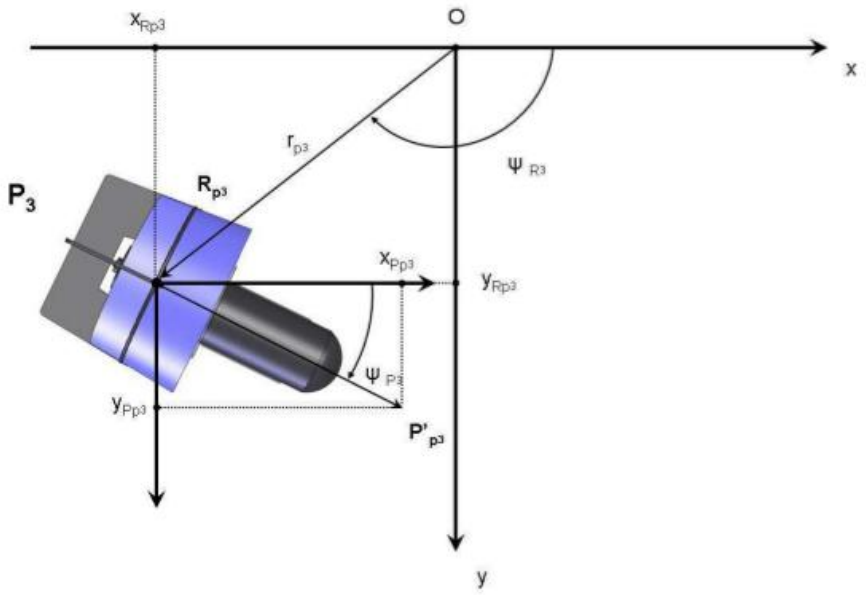


Figura 2.2: Diagrama de posicionamento e orientação (Extraído de Goulart (2007)).

Desse modo, define-se os vetores $\vec{R}_{pi}$ e $\vec{P}_{pi}$ como a posição e orientação do i-ésimo propulsor no sistema de coordenadas do veículo, respectivamente. Os propulsores são posicionados no plano xy. Logo, é obtido:

$$R_{pi} = \begin{bmatrix} x_{R_{pi}} \\ y_{R_{pi}} \\ 0 \end{bmatrix} = r_{pi} \begin{bmatrix} \cos(\Psi_{ri}) \\ \sin(\Psi_{ri}) \\ 0 \end{bmatrix}, \tag{2.25}$$

$$P_{pi} = \frac{1}{\left|P'_{pi}\right|} \begin{bmatrix} x_{R_{pi}} \\ y_{R_{pi}} \\ 0 \end{bmatrix} = \begin{bmatrix} \cos(\Psi_{ri}) \\ \sin(\Psi_{ri}) \\ 0 \end{bmatrix} \tag{2.26}$$

2.2.2 Acoplamento e Desacoplamento

As forças e momentos resultantes no sistema de coordenadas do ROV são obtidas a partir da contribuição dos empuxos e torques gerados por cada propulsor. Essa relação é dada a partir da matriz de acoplamento.Entretanto, dependendo das características deste veículo, simplificações podem ser feitas. Para viabilizar o projeto do controlador, essas simplificações são necessárias.

Com isso, considerando o ROV Luma, que possui 5 propulsores, os momentos gerados pelos propulsores em torno dos seus próprios eixos podem ser desprezados. Dessa forma, a matriz de acoplamento é dada por:

$$\begin{bmatrix} F_{px} \\ F_{py} \\ F_{pz} \\ M_{px} \\ M_{py} \\ M_{pz} \end{bmatrix} = \begin{bmatrix} P_{p1} & P_{p2} & P_{p3} & P_{p4} & P_{p5} \\ R_{p1} \times P_{p1} & R_{p2} \times P_{p2} & R_{p3} \times P_{p3} & R_{p4} \times P_{p4} & R_{p5} \times P_{p5} \end{bmatrix} \begin{bmatrix} F_{p1} \\ F_{p2} \\ F_{p3} \\ F_{p4} \\ F_{p5} \end{bmatrix} \tag{2.27}$$

Também podem ser desprezados os momentos em torno dos eixos x e y, pois estes não são controlados diretamente. Então, o modelo simplificado é dado por:

$$\begin{bmatrix} F_{px} \\ F_{py} \\ F_{pz} \\ M_{pz} \end{bmatrix} = A_{4\times5} \begin{bmatrix} F_{p1} \\ F_{p2} \\ F_{p3} \\ F_{p4} \\ F_{p5} \end{bmatrix} \tag{2.28}$$

onde $A_{4x5} \in \mathbb{R}^{4\times5}$ é a matriz de acoplamento, derivada de 2.27, simplificada.

Dessa forma, consegue-se simplificar a relação entre a dinâmica dos propulsores e

os movimentos translacionais ao longo do plano xy, ao longo do eixo z e movimentos rotacionais ao longo do eixo z.

Nota-se que a dinâmica de um veículo é bastante acoplada, um conjunto de propulsores deve ser acionado de modo que ocorra movimento translacional ou rotacional. Dessa forma, conforme Goulart (2007), o mecanismo de desacoplamento tem como finalidade possibilitar o controle de grau de liberdade separadamente.

Portanto, para o controle, a matriz de desacoplamento é a pseudo-inversa da matriz de acoplamento dada em 2.28.

$$\begin{bmatrix} \bar{F}_{p1} \\ \bar{F}_{p2} \\ \bar{F}_{p3} \\ \bar{F}_{p4} \\ \bar{F}_{p5} \end{bmatrix} = D_{5\times4} \begin{bmatrix} \bar{F}_{px} \\ \bar{F}_{py} \\ \bar{F}_{pz} \\ \bar{M}_{pz} \end{bmatrix} \tag{2.29}$$

onde $D \in \mathbb{R}^{5\times4}$.

2.3 Conclusões

Neste capítulos foram abordados os modelos cinemáticos e dinâmicos de um ROV geral e da Luma. O modelo cinemático permite relacionar as velocidades lineares e angulares do veículo no sistema de coordenadas inercial. Por sua vez, a modelagem dinâmica permite analisar os efeitos das forças e momentos que regem o movimento do ROV. Além disso, também foi estudado a modelagem dos propulsores. A verificação desses modelos é fundamental para viabilização do posicionamento dinâmico.

Capítulo 3

Modelo dos Sensores

Nesse Capítulo será tratado os sensores utilizados no sistema de controle desenvolvido. O ROV Luma possui um conjunto de sensores para medição aproximada da orientação, da profundidade e da distância em relação a um ponto em uma referência. Cada sensor possui um sistema de coordenadas independentes $E_s = [x_s \ y_s \ z_s]^T$dos outros sensores. Deste modo, a medição deve ser mapeada de E_s para o sistema de coordenadas utilizado. Ainda, considerando a definição de corpo rígido, a velocidade angular de cada sensor será igual a velocidade angular do corpo multiplicada pela matriz de rotação que leve para as coordenadas do sensor, logo, $\omega_{sb} = R_{sb}\omega_b$. Para o controle, é fundamental ter essas medições para a realimentação da malha.

A comunicação dos sensores com a base pode ser feita de duas formas. Com um computador embarcado, que se comunica com o computador base por fibra óptica ou cabo de rede Ethernet, ou diretamente utilizando um cabo umbilical. O diagrama abaixo apresenta uma configuração.

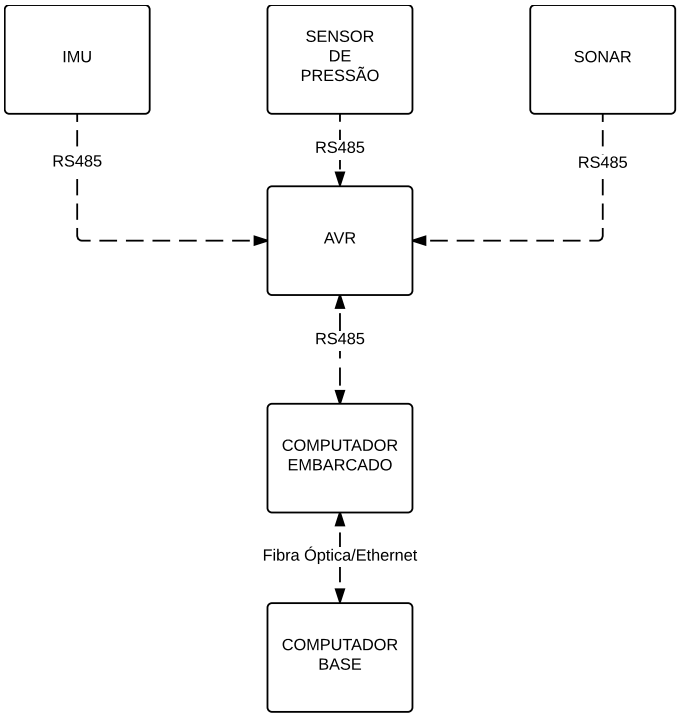


Figura 3.1: Configuração de comunicação do ROV LUMA utilizando computador embarcado.

Em ambos os casos, as trocas das mensagens de ROS na interface serão detalhados no Capítulo 4. Dessa forma, este capítulo abordará os sensores utilizados para medição dos estados da LUMA.

3.1 Unidade de Medição Inercial

A IMU (*Inertial Measurement Unit*) utilizada no veículo é produzido pela *Advanced Navigation* e é da série *Advanced Navigation Spatial*, ilustrado na Figura 3.2. Ela possui um conjunto de acelerômetros, giroscópios, magnetômetros e um receptor GPS. No sistema de controle do ROV, a IMU fornecerá as medidas de orientação.

Figura 3.2: IMU *Advanced Navigation Spatial*.

O aparelho disponibilza medidas da orientação na parametrização dos ângulos *roll-pitch-yaw*. Também fornece a acurácia de suas medidas. Estas são ordem 0.2^o para os ângulos de jogo e arfagem e 0.8^o para o rumo.

Conforme Frangipani (2015), o rumo depende principalmente da medição do campo magnético terrestre através de magnetômetros. Dessa forma, estruturas ferromagnéticas que esteja próximas da IMU podem distorcer o campo, resultando em uma medida menos precisa. Para atenuar essas interferências, a unidade inercial pode ser calibrada. Contudo, interferências dinâmicas causadas, que podem ser geradas por motores elétricos por exemplo, não são facilmente atenuados e representam um maior problema, conforme *Advanced Navigation Spatial*.

Dessa forma, o sistema de coordenadas da IMU é ilustrado pela Figura 3.3. O dispositivo é colocado no ROV de modo que o sistema de coordenadas E_{imu} esteja alinhado com o sistema de coordenadas inercial E_i e do veículo E_b. Portanto, a relação entre os sistemas de coordenadas do veículo é

$$E_{imu} = E_b,$$

de modo que

$$\Phi_{ib} = R_{ib}\Phi_b = \Phi_{imu}$$

Figura 3.3: Sistema de coordenadas da unidade inercial (Extraído de *Advanced Navigation Spatial*).

A mensagem de ROS correspondente aos valores fornecidos pela unidade inercial é dada por:

```
float64 roll
float64 pitch
float64 yaw
```

3.2 Profundímetro

O sensor de profundidade da Luma é o *Velki HPX* (Figura 3.4). A medição da pressão exercida pela água sobre o veículo permite obter a profundidade do ROV. A pressão P sobre um objeto submerso em um fluído é dado pela equação

$$P = \rho g h$$

onde g é a aceleração da gravidade e h é a altura do fluído sobre o objeto.

Para isso, em um modelo ideal, a densidade da água e gravidade são constantes, fazendo com que a pressão e a profunidade apresentam relação linear. O sensor utilizado apresenta acurácia de 0,05% usando comunicação serial $RS485$.

Figura 3.4: Profundímetro utilizado no ROV Luma.

Desse modo, o profundímetro é posicionado tal que o plano xy no sistema de coordenadas E_{prof} esteja coplanar com o sistema de coordenadas do robô. Assim, considerando o sistema de coordenadas inercial na superfície, a profundidade é dada diretamente pelo valor do sensor, como demonstrado na Figura 3.5.

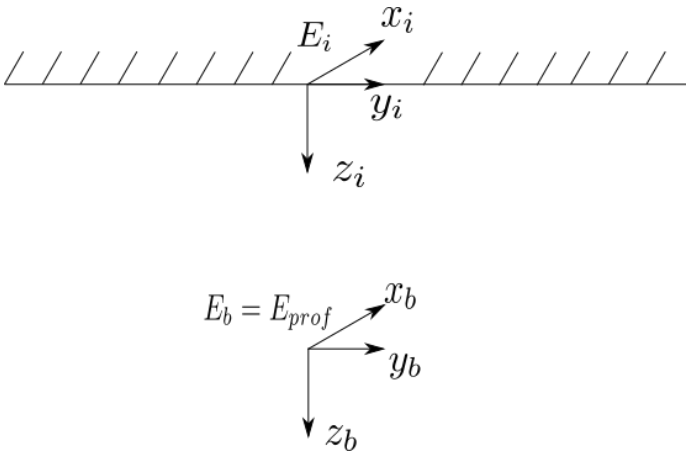


Figura 3.5: Sistema de coordenadas do profundímetro.

A mensagem de ROS utilizadas correspondentes ao valor de medição do sensor de pressão é listada abaixo.

```
# Valor de profundidade e referência
float32 depth
uint32 depthref
```

3.3 Sonar

Sonar, do inglês *Sound Navigation and Ranging*, é um aparelho que utiliza propagação de onda sonoras para navegação, comunicação ou detecção de objetos. Estes podem ser divididos em dois tipos: sonar passivo, caracterizado por escutar o som de outras fontes (como outros veículos), e sonar ativo, que emite pulsos sonoros por um transmissor e escuta os ecos.

No caso do ROV Luma, é utilizado um sonar ativo da série *Tritech Micron*, vide Figura 3.6.

Figura 3.6: Sonar utilizada no ROV Luma.

A grande característica do sonar ativo é que usam um transmissor e receptor de som. O transmissor gera um pulso de som e depois escuta as reflexões deste pulso. Normalmente é utilizado um formador de feixe de modo que concentre a potência acústica em um feixe, que pode varrer ângulos específicos.

Ao medir o tempo que o pulso é gerado pelo transmissor, receptor recebe sua reflexão e conhecendo a velocidade do som no ambiente, é possível obter a distância do obstáculo ao sonar. A onda pode ter uma frequência fixa ou usar tecnologia chirp para mudança de frequência.

Conforme explicado em Tritech International Ltd (n.d.), um sonar com frequência constante, a resolução do alcance é determinado pelo comprimento do

pulso transmitido. Quanto menor o pulso, maior a resolução alcançável e vice-versa. Com base na Figura 3.7, se dois objetos estão à uma pequena distância entre si, eles não podem ser distinguidos. O problema será que o sonar só mostrará um grande objeto ao invés de objetos múltiplos.

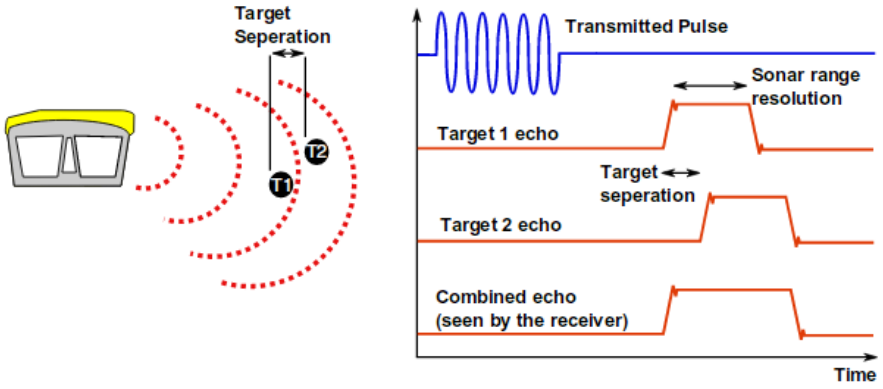


Figura 3.7: Pulso de um sonar (Extraído de Tritech International Ltd (n.d.)).

O sonar utilizado na Luma usa processamento de sinal CHIRP que supera essa limitação modificando a frequência dentro de uma faixa de valores durante a transmissão do pulso. Isso cria uma característica acústica única para cada alvo, o sonar sabe o que foi transmitido e quando. Dessa forma, quando dois ecos se sobrepõe, os reflexos não se combinam em um. A Figura 3.8 ilustra o resultado.

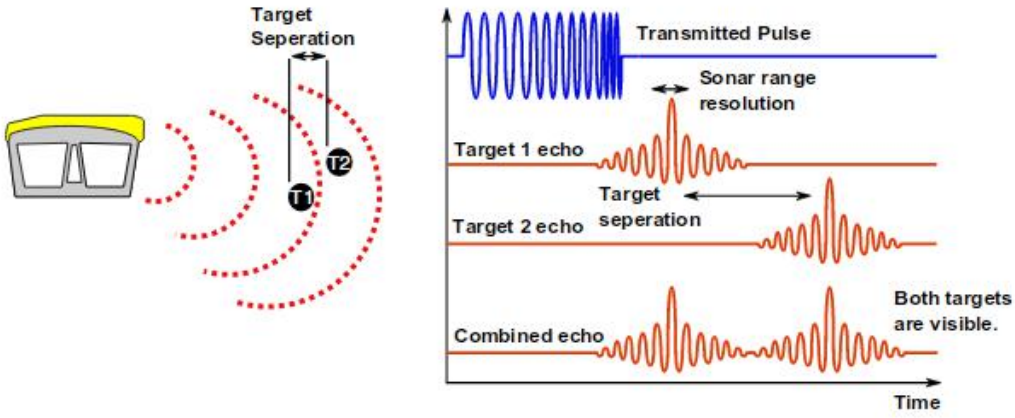


Figura 3.8: Pulso de um sonar com tecnologia CHIRP(Extraído de Tritech International Ltd (n.d.)).

Além disso, essa tecnologia oferece avanços na rejeição de ruído e remove ruídos aleatórios ou fora da banda de frequências.

O sonar será utilizado para medir a distância do ROV até um marco, nesse caso uma parede. Ele é posicionado na frente do veículo, de modo que o espaço do sistema de coordenadas do sonar E_{sonar} esteja transladado $p0_{bs}$ sistema de coordenadas do veículo. Nesse caso, p_{bs} consiste em uma translação ao longo do eixo x e outra ao

longo do eixo z do veículo. Além disso, E_{sonar} está rotacionado π rad em torno do eixo y.

Baseado na Figura 3.9, os valores de um obstáculo medido pelo sensor, nas coordenadas do robô, são dados por

$$p_{bs} = R_y(\pi)p_{sw} + p0_{bs}$$

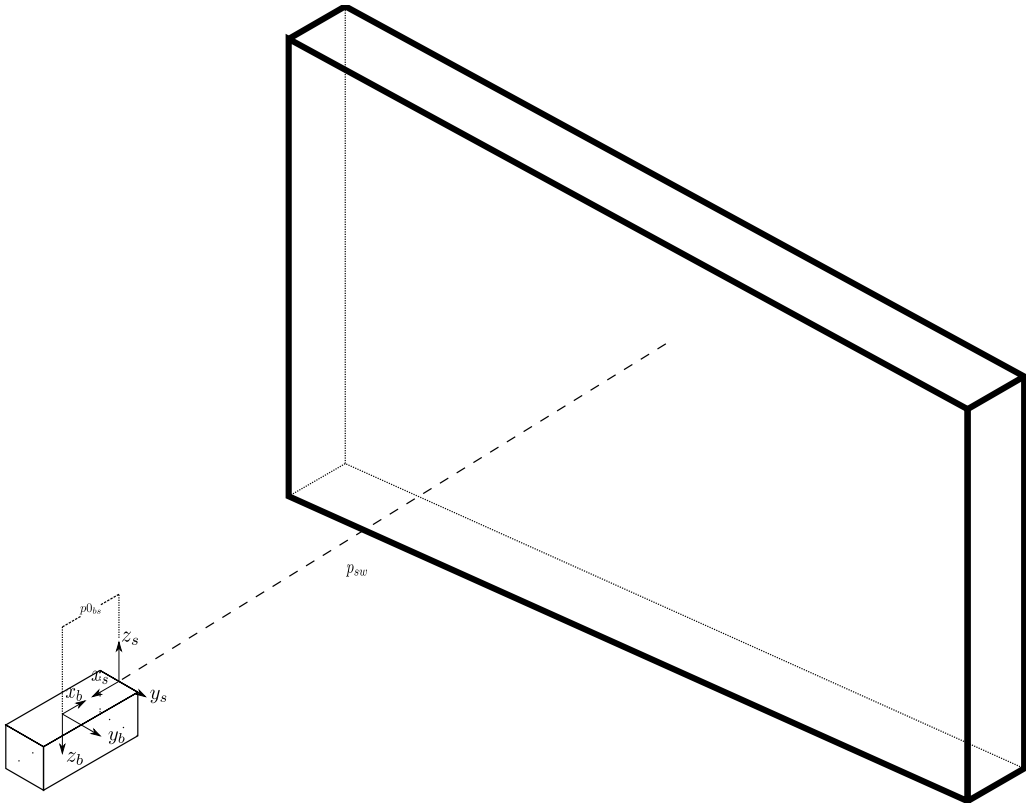


Figura 3.9: Sistema de coordenadas do sonar.

onde $p_{sw} \in \mathbb{R}^3$ é a medida da distância da parede até o sonar no sistema de coordenadas do sonar, $p0_{sb} \in \mathbb{R}^3$ é a posição do sonar no sistema de coordenadas do veículo e $p_{sb} \in \mathbb{R}^3$ é a medição da parede no sistema de coordenadas do veículo.

Dessa forma, os dados do sonar são transmitidos através de mensagens padrão do ROS: *PointCloud* e *PoseStamped*. A mensagem de configuração do sonar é personalizada do driver utilizado. Estas são apresentadas a seguir, respectivamente.

```
#Mensagem PointCloud
#Tempo de aquisição dos dados, id do frame da coordenada.
Header header
#Vetor de pontos em 3d. Cada Point32 deve ser interpretado como ponto
# em 3d na coordenada dado em header
geometry_msgs/Point32[] points
#Mesmo tamanho de points, vetor contendo os valores de intensidade
# para cada ponto
```

```
ChannelFloat32[] channels
```

```
#Mensagem PoseStamped
Header header
# Posição e orientação
Pose pose
```

```
# Tritech Micron configuration parameters.
Header header
# Se o sonar está voltado para cima.
bool inverted
# Se a varredura é contínua ou por setor.
bool continuous
# Se a varredura é no sentid horário.
bool scanright
# Se a intensidade da varreduta é um número de 8 ou 4 bits.
bool adc8on
# Ganho inicial do pulso sonar. Varia entre 0 e 1.
float64 gain
# Mapeamento da intensidade em dB. Varia de 0 a 80 dB.
float64 ad_low
float64 ad_high
# Limite de varredura direito e esquerdo no modo por setor.
# Os valores estão em radianos e variam de 0 a 2 pi.
float64 left_limit
float64 right_limit
# Alcance da varredura em metros
float64 range
# Número de bins por fatia.
int16 nbins
# Tamanho do passo do motor em radianos.
float64 step
```

Consequentemente, para compreensão dos dados emitidos pelos sensores, é necessário localizar o ROV no ambiente.

3.4 Método de Localização

A localização de um robô consiste em estimar a posição e orientação em relação a um referencial inercial a partir das informações dos sensores. Desse modo, quando o problema trata-se da localização de um robô móvel, o problema resume-se a obter os ângulos (*roll-pitch-yaw*) e as coordenadas cartesianas (x,y,z).

Considerando o caso do sonar utilizado na Luma, a localização se restringe ao plano xy e a orientação em torno do eixo z. O sonar é usado para encontrar o *landmark*. Os *landmarks* são características, como paredes, portas, curvas do ambiente

que possam ser utilizadas para a localização do robô (Siegwart et al. (2011)). Nesse projeto, o ponto de referência é uma parede de modo que o problema é restringindo em obter a distância no eixo x e a orientação em torno do eixo z (yaw). A primeira etapa para o desenvolvimento de um método de localização foi o estudo das características do sonar.

3.4.1 Estudo do Sonar

Antes de realizar testes experimentais do sonar a bordo do veículo, é necessário realizar uma análise dos dados do sensor com o objetivo desenvolver um script que simulasse o mesmo comportamento. Para isso, realizou-se experimentos com o sonar com o auxílio de um *driver* feito em *Python* desenvolvido pela *McGill Robotics*. O pacote faz a comunicação com o sonar, interpreta o vetor de intensidades fornecido por ele e publica 3 mensagens de ROS. As mensagens são descritas pela tabela a seguir:

Tabela 3.1: Mensagens de ROS do sonar.

nome do tópico	tipo de mensagem	descrição
Scan	*PointCloud*	conjuntos de pontos em $[x\ y\ z]$ e respectivas intensidades
Heading	*PoseStamped*	orientação do raio de medição do sonar
Config	Mensagem Personalizada	conjunto dos parâmetros do sonar

Primeiramente, para a realização do teste, foi necessário fazer uma hipótese:

Hipótese 3.1. *A água está suficientemente limpa de tal forma que não há partículas em suspensão que possam ser detectadas pelo sonar*

Além disso, o teste foi em uma caixa d'água e realizado nas seguintes condições:

- Não há parede (obstáculo)
- Sonar posicionado no centro da caixa d'água
- Raio real da caixa d'água $r_{real} \approx 0.65m$
- Resolução do sonar de 1.8^o
- Alcance igual a $range = 1.5\ m$
- Número de valores por medição $nbins = 200$

- Pontos equidistantes, ou seja, cada par de ponto consecutivo no vetor retornado pelo sonar estão a $\frac{range}{nbins}$ de distância entre si

A figura abaixo foi retirada da ferramenta de visualização de mensagens do *ROS* chamada *rviz*

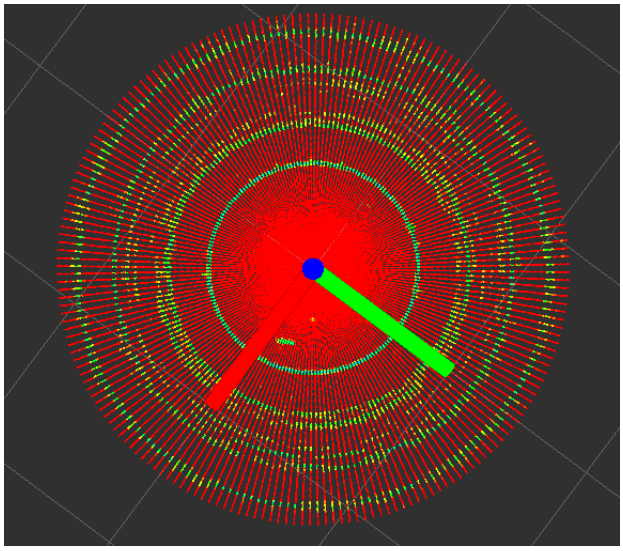


Figura 3.10: Imagem no *rviz*

Como pode ser observado, os dados não filtrados do sonar apresentam muitos ecos e não é possível realizar uma análise adequada. Por isso, é necessário filtrar os dados originais. Segundo Ribas et al. (2010), um tratamento é escolher os valores de maior intensidade. Consequentemente, é possível propor uma nova abordagem com base em uma segunda hipótese:

Hipótese 3.2. *Pontos de maior intensidade dados pelo sonar, no caso ideal, correspondem à parede a ser detectada e não há outro objeto reflexivo*

Com isso, foi proposto a utilização da mensagem *Laserscan* de *ROS* devido à facilidade do tratamento de dados. Esta mensagem utilizada valores de *range*, por isso, torna-se necessário transformar o par *(x,y)*, dado pela mensagem *Pointcloud*, em valores de distância. Deste modo, também é fundamental conhecer a orientação da medição do sonar.

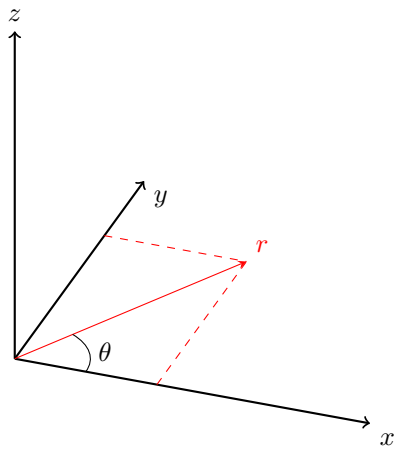


Figura 3.11: Sistema de coordenadas genérico

Como mostrado na figura acima, considerando um sistema de coordenadas genérico, a distância de um ponto no plano xy até a origem é dado por:

$$r = \sqrt{x^2 + y^2} \tag{3.1}$$

e o ângulo desse range é de

$$\theta = atan2(y, x) \tag{3.2}$$

Dessa forma, os dados referentes à Figura 3.10 foram filtrados de modo que só os pontos de maior intensidade fossem escolhidos e o resultado é mostrado abaixo:

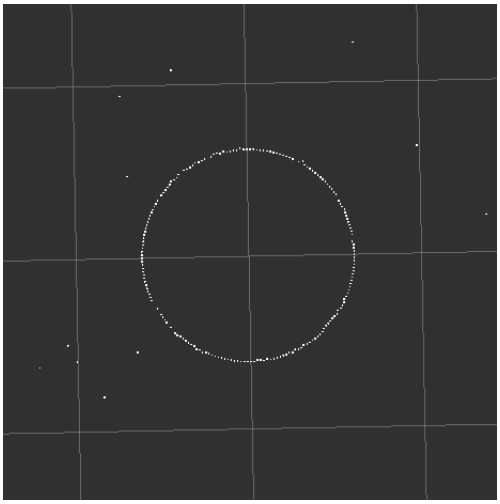


Figura 3.12: Mensagem *Laserscan* publicada

Feito isso, o próximo passo foi analisar os novos dados para verificar a validade do novo tratamento. Com o auxílio do *Matlab (Mathwork Inc.)* foi levantado o histograma do valores de *range* demonstrados abaixo:

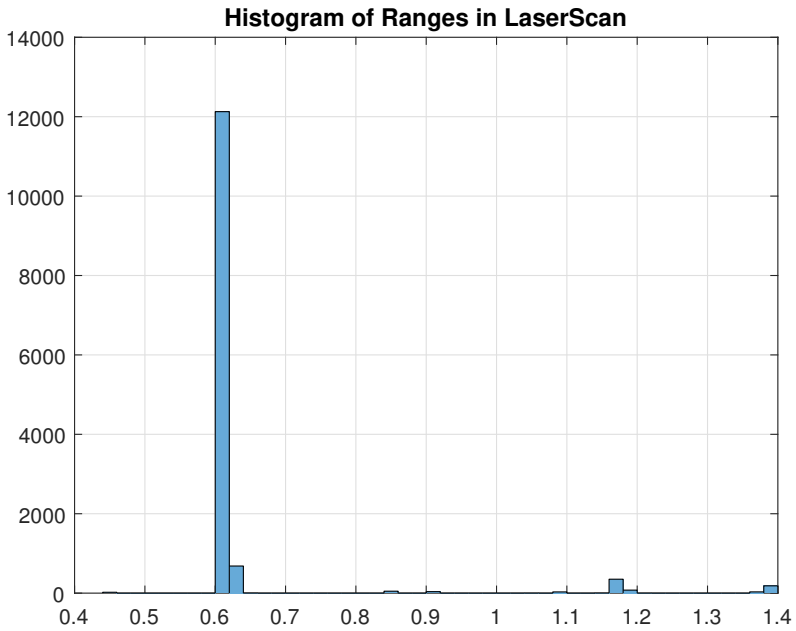


Figura 3.13: Histograma dos *ranges*.

Os dados dessas medições podem ser modelados como uma distribuição gaussiana cuja função de densidade probabilística é dada por:

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\sigma^2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Com isso, a média e a variância são dados por:

$$\mu = 0.6164\ m$$
$$\sigma^2 = 0.0194\ m$$

Dessa forma, foi realizado um segundo teste. Entretanto, foi colocada uma parede na frente do sonar, de modo que a distância do centro do sonar até a parede seja $d_{real} = 0.80$. As configurações do sonar foram mantidas do primeiro teste. A figura a seguir corresponde a uma volta completa:

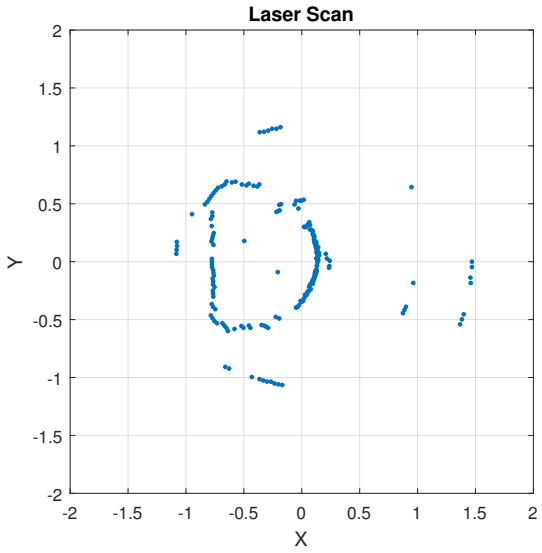


Figura 3.14: Gráfico corresponde a uma volta completa

Como observa-se na figura 3.14, os valores correspondentes à parede em relação ao sistema de coordenadas do sonar são constantes no eixo x. Desse modo, foi realizado uma análise probabilística dos pontos nesse eixo. Para isso, os dados foram segmentados considerando apenas o obstáculo desejado.

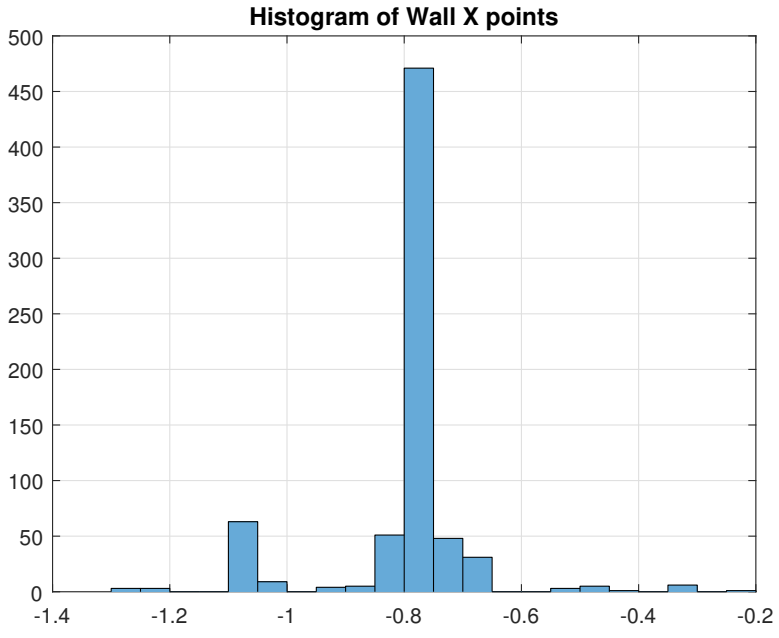


Figura 3.15: Histograma dos valores no eixo x

Analogamente ao teste anterior, os dados podem ser modelados por uma distribuição gaussiana de médias e a variância dados por:

$$\mu = -0.7963\ m$$
$$\sigma^2 = 0.0147\ m$$

3.4.2 Identificação de *landmarks*

Como mencionado, o objetivo é identificar uma parede e localizar-se em um eixo. Para isso, é utilizado o sonar que publica uma mensagem de ROS do tipo *Laserscan*. Contudo, esse processo de identificação apresenta alguns problemas como a presença de ruído, uma enorme nuvem de pontos e uma grande presença de *outliers*, ou seja, pontos fora da reta que corresponderia à parede. Na literatura, há estudos sobre diferentes métodos para identificação, como em Arun et al. (1987),Jang et al. (2005),Fischler and Bolles (1981) e Nguyen et al. (2005). Dois métodos utilizados nesse projeto foram o método dos Mínimos Quadrados e o RANSAC (*Random Sample Consensus*). No caso do Mínimos Quadrados, este não é considerado ideal para os dados iniciais, de modo que apresenta problemas com linhas "verticais" e é muito sensível a *outliers*.

Com isso, como explicado em Fischler and Bolles (1981), o método do RANSAC é adequado na presença de *outliers*. É um método iterativo que permite uma estimação robusta dos parâmetros.

O método proposto para a localização consiste em duas etapas. A primeira etapa do método de identificação é utilizado o RANSAC para obter um conjunto de linhas dentro de uma tolerância, denominados de *inliers*, retirando os *outliers*. Depois, as linhas obtidas são filtradas de modo que a melhor seja escolhida. Finalmente, aplica-se o método dos mínimos quadrados para obter os parâmetros da reta que parametrizam a parede.

O pseudocódigo abaixo resume o método para a identificação do *landmark*.

Algorithm 1 Pseudocódigo RANSAC

Require: $N \rightarrow$ quantidade de iterações.

$t \rightarrow$ limite da distância dos inliers até a reta.

$tol\_inl \rightarrow$ quantidade mínima de inliers para ser um bom modelo da reta.

for $i = 1$ to N **do**
  selecionar dois pontos aleatórios.
  encontrar os parâmetros da reta de acordo com esses dois pontos.
  encontrar *inliers* entre os pontos restantes de acordo com um limite ***t***.
  if quantidade de inliers > tol_inl **then**
    armazenar esses pontos
  end if
end for
fazer filtragem das linhas obtidas
utilizar método dos mínimos quadrados para obter parâmetros da reta
return parâmetros da reta.

3.4.3 Localização do ROV

No caso geral, uma reta no plano pode ser definida pela equação:

$$ax + by + c = 0, \tag{3.3}$$

onde a, b e c são valores reais constantes com a e b não iguais a zero simultaneamente. Desse modo, a distância da reta a um ponto (x_0, y_0) é:

$$d(ax + by + c = 0, (x_0, y_0)) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

Para provar esse resultado, considere a Figura 3.16:

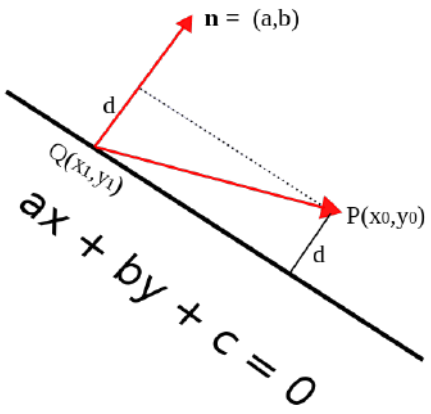


Figura 3.16: Projeção de vetores (Imagem extraída de *Wikipedia*).

Sendo P o ponto dado por (x_0, y_0) e uma reta dada pela equação 3.3. Definindo um ponto $Q = (x_0, y_0)$ sobre a reta e $\vec{n}$ o vetor (a, b) com origem em Q. O vetor n é perpendicular a reta e a distância do ponto P até a reta é igual ao comprimento ortogonal da projeção de $\overrightarrow{QP}$ em $\vec{n}$. O comprimento dessa projeção é dado por:

$$d = \frac{\left|\overrightarrow{QP} \cdot \vec{n}\right|}{\|\vec{n}\|} \tag{3.4}$$

Com isso, tem-se:

$$\begin{aligned} \overrightarrow{QP} &= (x_0 - x_1, y_0 - y_1) \\ \overrightarrow{QP} \cdot \vec{n} &= a(x_0 - x_1) + b(y_0 - y_1) \\ \|\vec{n}\| &= \sqrt{a^2 + b^2} \end{aligned}$$

o que resulta em:

$$d = \frac{|a(x_0 - x_1) + b(y_0 - y_1)|}{\sqrt{a^2 + b^2}}$$

e como $c = -ax_1 - by_1$, d é igual a equação 3.3. Além disso, os pontos (x_1, y_1) são:

$$x_1 = \frac{b(bx_0 - ay_0) - ac}{a^2 + b^2} \quad y_1 = \frac{a(-ax_0 + by_0) - bc}{a^2 + b^2}$$

Neste projeto, a parede foi parametrizada por uma reta da forma

$$ax - y + b = 0$$

Dessa forma, considerando as equações da demonstração anterior e a figura 3.17, a posição do sonar será a origem do sistema de coordenadas e:

$$\begin{aligned} \|\vec{n}\| &= \sqrt{a^2 + 1} \\ x_p &= \frac{-ab}{a^2 + 1} \\ y_p &= \frac{b}{a^2 + 1} \\ d &= \sqrt{x_p^2 + y_p^2} \\ \theta &= atan2(b, -ab) \end{aligned}$$

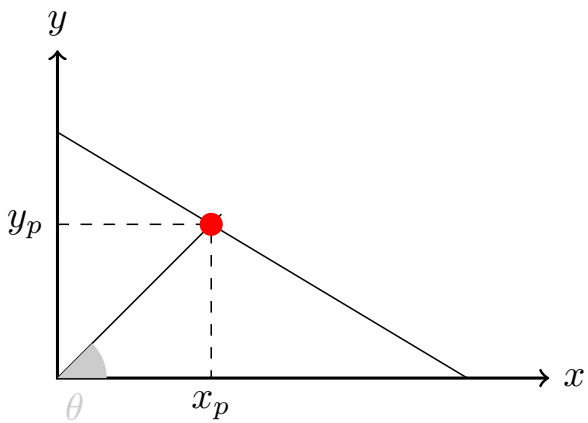


Figura 3.17: Projeção de vetores no caso do ROV.

Portanto, a menor distância do ROV até a parede será dado por $d = \sqrt{x_p^2 + y_p^2}$. O método utilizado para identificação dos marcos retorna o coeficiente linear e angular de uma reta dada pela equação $y = ax + b$. Sabe-se que a orientação do veículo é dado por:

$$\psi = \arctan(a)$$

Com a orientação e a distância até o marco, é possível localizar o veículo no ambiente em relação ao eixo x inercial. Deve-se ressaltar que esse método apresenta singularidades quando o coeficiente angular é demasiado elevado, ou seja, quando o ângulo $\theta \approx 0$.

3.5 Conclusões

Neste capítulos foram abordados os sensores utilizados e a transformação de coordenadas do sistema de coordenadas do sensor para o veículo foram detalhados. Os dados desses sensores serão utilizados para o projeto do controlador em malha fechada. Vale ressaltar que as transformações foram realizadas considerando o caso ideal, ou seja, o valor do sensor corresponde exatamente ao real.

Foi abordado o problema da localização do veículo no ambiente. Foi apresentado um estudo das características do sonar utilizado no ROV Luma. Verificou-se que os dados iniciais do sensor podem ser filtrados escolhendo os pontos de maior intensidade. Essa filtragem facilitou o tratamento dos valores para a localização.

For, também, implementado o método de localização do veículo. Com isso, para a primeira etapa de identificação da parede, a utilização de dois métodos em conjunto (Ransac e Mínimos Quadrados) tornou-se adequada para obter os parâmetros do *landmark* desejado. Com a reta correspondente a parede, uma abordagem geométrica permitiu a localização do ROV no ambiente de forma apropriada.

Capítulo 4

Interface para Operação Remota e Posicionamento Dinâmico do ROV Luma

Nesse capítulo será abordado a interface desenvolvida para a operação do ROV Luma. Serão detalhados as janelas gráficas que auxiliam no problema do posicionamento dinâmico e operação remota do veículo. Além disto, as mensagens de ROS utilizadas e a lógica por trás dessa comunicação entre interface e robô serão detalhadas. No Apêndice B, os arquivos de configurações e outras janelas gráficas são apresentadas.

4.1 Interface Gráfica

Para operação de robô, é necessário enviar comandos e receber dados dos diversos sensores que o compõem. Nesse aspecto uma interface gráfica possui uma grande inportância para o controle e monitoramento de um ROV.

A interface do ROV Luma, denominada *RobotGUI* foi idealizado a partir das seguintes ferramentas:

- Sistema Operacional: Linux - Ubuntu 12.04
- Linguagem de programação: C++
- *Middleware* de comunicação: ROS Fuerte
- *Framework* para interface gráfica: Qt

O conceito principal do software é separar interface gráfica, objetos de comunicação, objetos de controle e modelo. Com a abstração do ROS para realizar a comunicação entre nós via rede Ethernet, esse conceito torna-se possível.

Desse modo, *RobotGUI* consiste em um nó de ROS. O sistema foi criado com o intuito de que uma interface gráfica pudesse ser desenvolvida e reutilizada em vários robôs. É difícil mesclar uma interface com vários nós de ROS porque cada nó é um processo, e a interface tem que ser um único processo. Portanto foi proposto uma arquitetura de *software* que pudesse juntar o vários nós de ROS em um nó que também fosse uma interface feita em Qt. Vale ressaltar que o ROS possui uma interface gráfica para muitas coisas, mas é para o desenvolvedores, não o usuário final. *RobotGUI* tenta ser mais apresentável ao usuário final.

Assim, a implementação foi feta com base em duas classes principais: *Components* e *Tools*.

De forma geral, os *Components* são responsáveis para lógica de comunicação e controle. Eles realizam a comunicação entre si através de mensagens e serviços de ROS apresentados no Capítulo 1. Também realizam a comunicação serial RS485 com a eletrônica embarcada no robô. Components são codificados de forma modular, ou seja, podem ser alterados e executados de forma independente. Além disso, representam dispositivos ou sensores e podem se conectar a múltiplas *Tools*. Components poderiam ser executados individualmente porque . No entanto, quando um Component deve se conectar a uma Tool, o Component deve ser iniciado dentro do *RobotGUI*.

Por sua vez, as *Tools* correspondem a janela gráfica na interface. São responsáveis por enviar os comandos do operador para os *Components* e pela visualização de dados de diversos sensores e reconfiguração dos parâmetros do robô. No caso específico de controle, através das *Tools* é possível configurar o *joystick*, definir os parâmetros do controlador e enviar valores de referência.

Dessa forma, a figura 4.1 apresenta o diagrama de exemplo da comunicação entre *Components* e *Tools*. Os outros *Components* e *Tools* são descritos no Apêndica B.

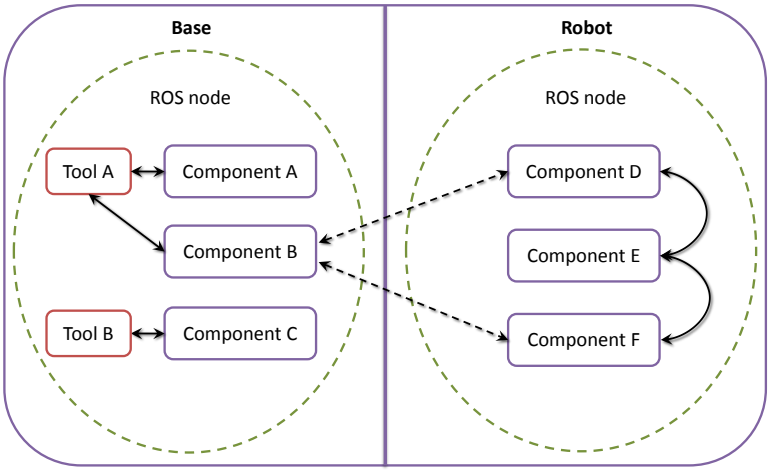


Figura 4.1: Comunicação entre *Components* e *Tools*.

RobotGUI é um nó de ROS definido em um pacote de ROS. Ele é capaz de iniciar outros pacotes, carregando e iniciando todos os componentes declarados. As Tools são carregadas, mas o usuário é quem decide quais Tools serão executadas. Esta arquitetura permite que Components e Tools se conectem mesmo quando são definidos em pacotes diferentes. Os principais pacotes utilizados na interface do ROV Luma são apresentados abaixo.

- *Robot Package*: é uma biblioteca que contém Components e Tools definidos. Podem ser dependentes de outros, implicando que Components ou Tools que são definidos em um pacote podem ser derivados (como classe de C++) de Components ou Tools de outros pacotes. Isto é ótimo no desenvolvimento do código em orientação a objetos. Suas classes são identificadas pela sigla GPCB(*General Package Component Base*), GPCR(*General Package Component Robot*) e GPT(*General Package Tool*)

- *Luma Package*: pacote, que depende de General Package, contendo códigos específicos para o ROV Luma como a estrutura de controle, ligar luminárias e outros dispositivos. A identificação é feita de maneira análoga ao *General Package*: LPCB(*Luma Package Component Base*), LPCR(*Luma Package Component Robot*) e LPT(*Luma Package Tool*). A Figura 4.2 apresenta a inteface gráfica *RobotGUI*.

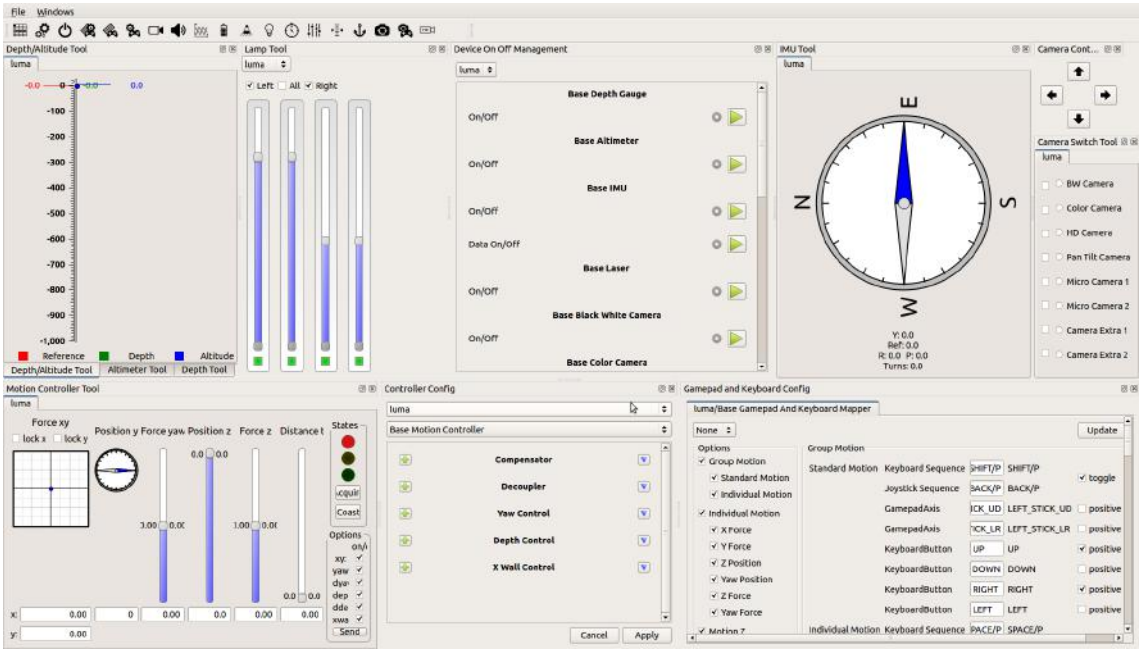


Figura 4.2: *RobotGUI* com Tools do *Luma Package*.

Para iniciar a interface, é necessário executar o comando:

```
roscd LUMA/bin
./RobotGUI
```

No caso do ROV LUMA estiver com o PC104 embarcado, os *Base Components* são iniciados no computador base, enquanto os *Robot Components* são iniciados no computador embarcado. A comunicação entre a base e robô é feita utilizado mensagens de ROS que podem ser transmitidas por fibra óptica ou cabo de rede Ethernet (no projeto Luma, o cabo de rede apresenta o limite de 100 m). Dessa forma, os *Components* no robô transmitem as mensagens serias padrão RS485 para o microcontrolador embarcado. A Figura 4.3 apresenta um diagrama geral dessa forma de comunicação.

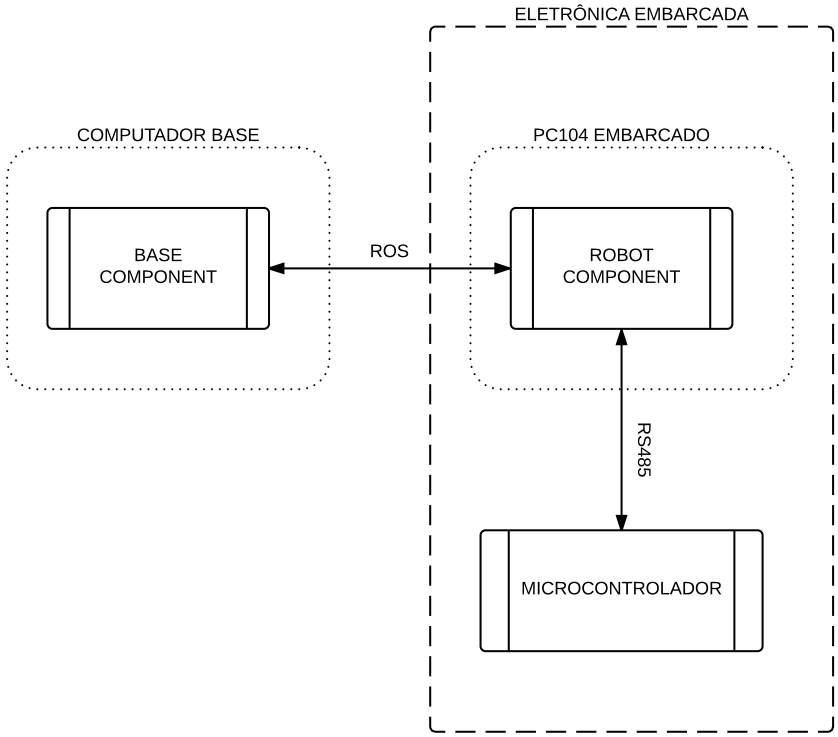


Figura 4.3: Diagrama de comunicação Base e Component com PC104 embarcado.

No caso da conexão do ROV Luma ser apenas com o cabo umbilical, as trocas de mensagens entre *Base* e *Robot* são realizadas no computador base. As mensagens seriais são enviadas diretamente para o ROV através do cabo umbilical. O diagrama a seguir exibe essa forma de comunicação.

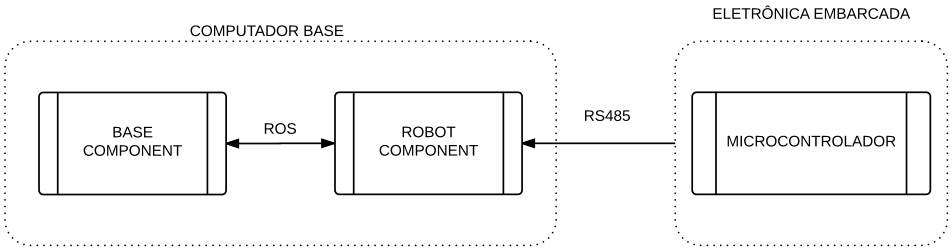


Figura 4.4: Diagrama de comunicação Base e Component sem PC104 embarcado.

Dessa forma, para a tarefa de controle, alguns *Components* e *Tools* são essenciais para o operador. As seções a seguir os apresentam.

4.1.1 *Components e Tools*

Primeiramente, a comunicação pelo RS485 foi implementada de modo que uma mensagem é enviada por vez. Envia-se a mensagem do computador para o microcontrolador, espera-se a resposta do AVR para depois mandar a próxima mensagem.

Com isso, cada *Robot Component* possui um cliente que utiliza um serviço de ROS presente no LPCR485Server. Durante esse serviço, a mensagem serial é escrita e enviada ao microcontrolador do robô e, também, a resposta é lida. Dessa forma, o valor correspondente na resposta pode ser publicado para o *Robot Components* ou ser utilizada de outra forma. Além disso, alguns serviços podem ser chamados periodicamente devido ao componente LPCRSupervisor. Este publica uma mensagem, em um período fixo, no tópico *luma/Cycle* enquanto alguns *Robot Components* estão subscritos a esse tópico.

No escopo do controle e na operação do ROV Luma, há uma rede de comunicação entre os Components do *General Package* e *Luma Package*. Como mencionado, o LPCRSupervisor publica periodicamente uma mensagem de ROS no tópico *luma/Cycle*. A cada mensagem publicada nesse tópico, os *Robot Components* correspondentes aos dispositivos chamam um serviço de ROS. O serviços são realizados no LPCR485Server que escreve a mensagem serial para a eletrônica do robô. Ao terminar a leitura da resposta do microcontrolador, os dados são transmitidos para os *Base Components* dos sensores.

O componente responsável pelo controle do ROV Luma é denominado LPCRMotionController. Ele possui *subscribers* que obtêm os valores medidos pelos sensores. Este então realiza o algoritmo de controle, com período de 100 ms, e envia a resposta para o veículo por meio de um serviço no LPCR485Server. Além disso, também é responsável por definir os parâmetros do controle de cada grau de liberdade, dos compensadores dos propulsores, dos elementos da matriz de desacoplamento.

Quanto a operação remota, o componente LPCRGamepadAndKeyboardMapper mapeia os valores dos botões e dos eixos do console e publica uma mensagem recebida em GPCRController.

A figura 4.5 apresenta um diagrama geral da comunicação entre os *Components*.

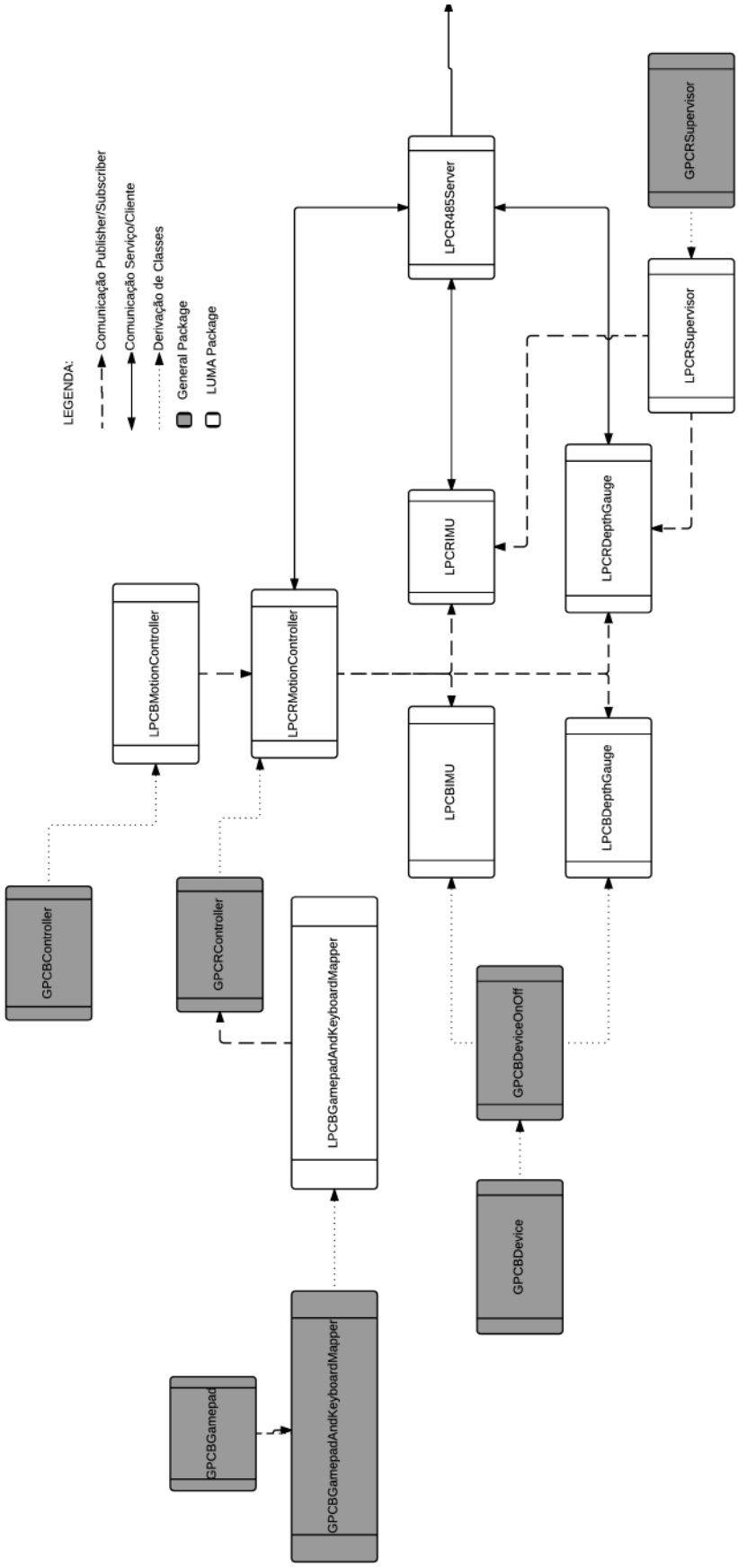


Figura 4.5: Diagrama geral da comunicação entre os Components.

Dessa forma, são apresentas as *Tools* que auxiliam o usuário para a operação e o posicionamento dinâmico do veículo. As outras *Tools* do sistema são apresentadas no Apêndice B.

GPTControllerConfig

A figura 4.6 apresenta a *tool* relaciona a configuração do controle. Ela foi desenvolvida de maneira genérica, de modo que é necessário escolher o robô a ser controlado. Ao escolher o robô, é possível escolher todos os controles que podem ser feitos. No caso do ROV LUMA. há apenas o controle de movimento, definido pelo *component* LPCBMotionController.

Com isso, aparece uma lista contendo todos os parâmetros de controle, como a compensação dos propulsores, os valores da matriz de desacoplamento e os parâmetros do controlador P-PI para a tarefa de posicionamento dinâmico dos 3 DOF do veículo. Esses parâmetros serão apresentados no Capítulo 5.

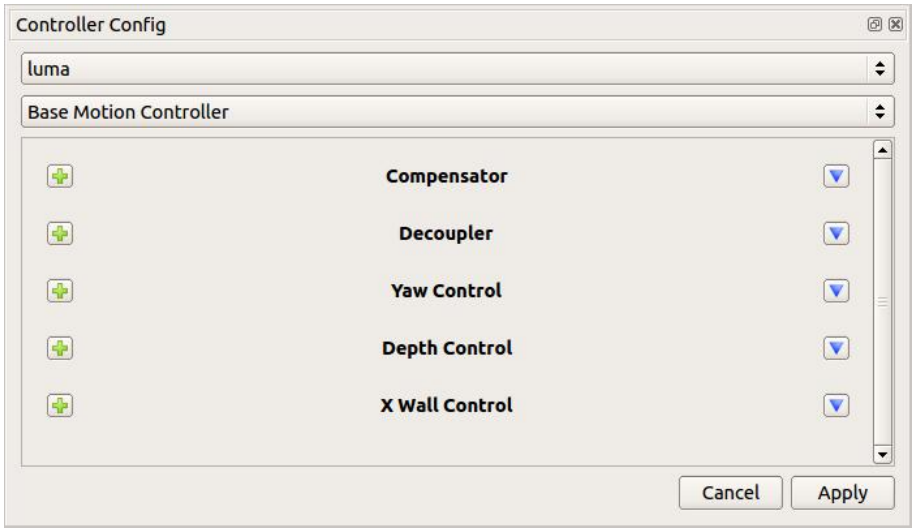


Figura 4.6: *GPTControllerConfig Tool*.

GPTGamepadAndKeyboardConfig

Essa janela configura o mapeamento do *joystick* para operação manual do veículo. A configuração dos botões e do eixos do console é feita de maneira personalizada pelo operador como demonstrado na Figura 4.7. No caso do ROV Luma, a operação manual pode ser feita enviando sinais de controle para todos os graus de liberdade simultaneamente ou enviando individualmente.

Ao fechar a interface, a configuração do joystick é salva em um arquivo *xml*. Desse modo, ao abrir a interface novamente, esta carregará a última configuração salva. Também é possível alterar esse arquivo externamente.

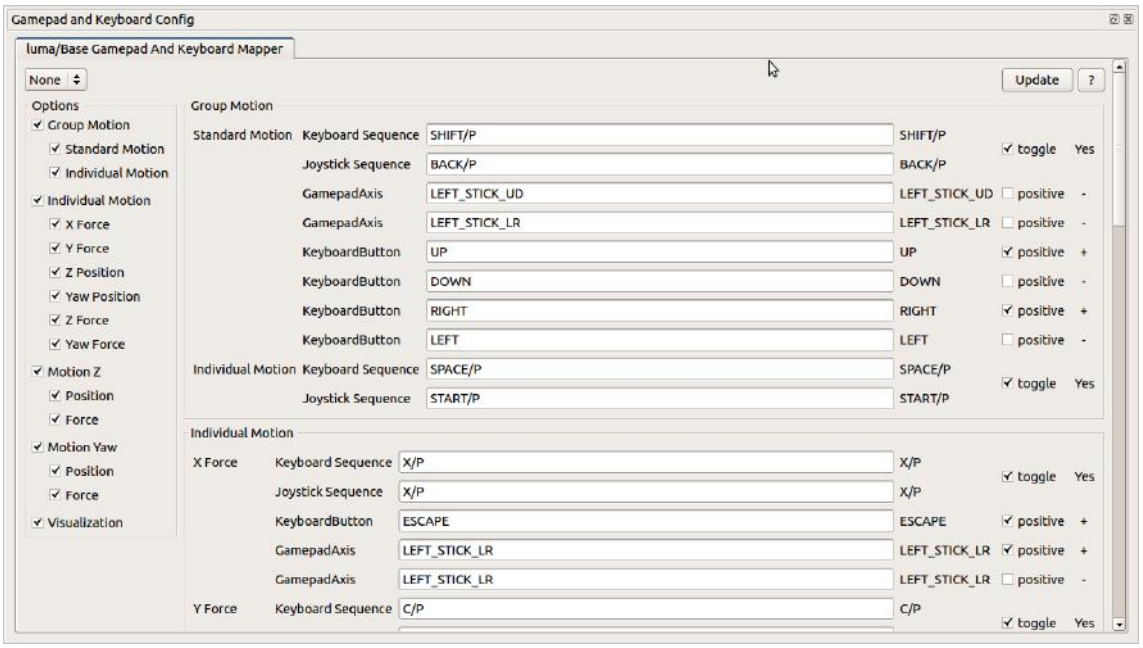


Figura 4.7: *GamepadAndKeyboardConfig Tool.*

GPTPlotter

Tool responsável pro traçar graficamente as variáveis do sistema. Permite adicionar diferentes gráficos à medida que for necessário. Apresenta diferentes funcionalidades como parar o gráfico, redimensionamento, alterar parâmetros como estilo e tamanho de linhas e salvar o gráfico em uma imagem.

O gráfico apresenta dois eixos. O eixo vertical mostra valores de amplitude enquanto o eixo horizontal demonstra o tempo no formato HH:MM:SS. Essa janela é ilustrada na Figura 4.8.

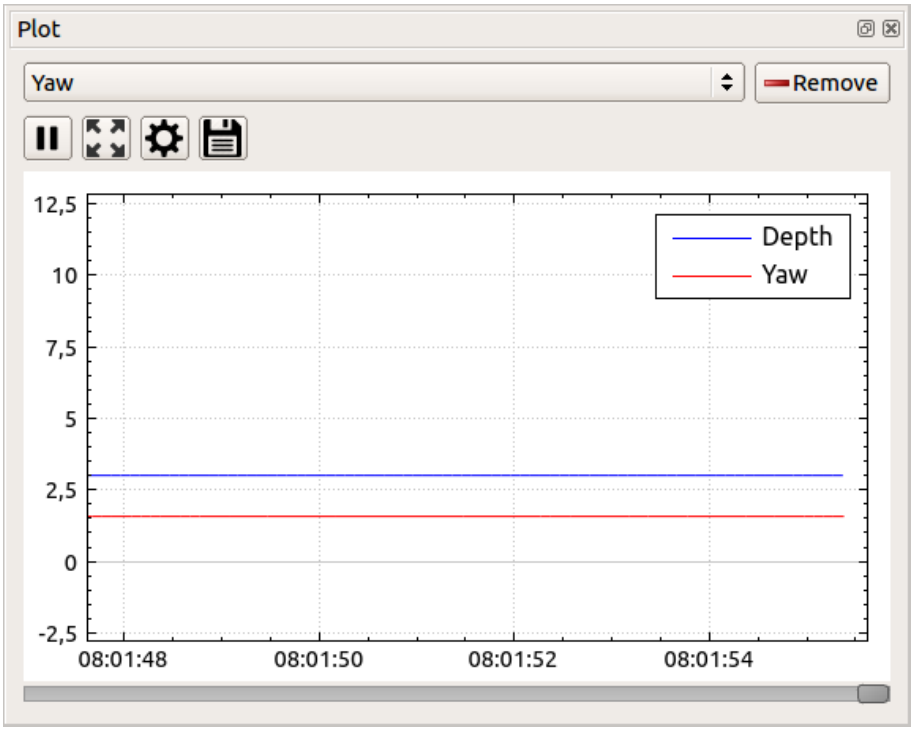


Figura 4.8: *Plotter Tool*.

LPTDepthGauge

Desenvolvida propriamente para o Luma, é composto por dois *sliders*, sendo um responsável pro mostrar o valor de profundidade medido pelo profundímetro e o outro que exibe o valor de referência enviado pelo operador. Está conectada aos componentes do profundímetro (LPCBDepthGauge).

A mensagem de ROS enviado é denominada *LPDepth.msg*, com formato:

```
float32 depth
uint32 depthref
```

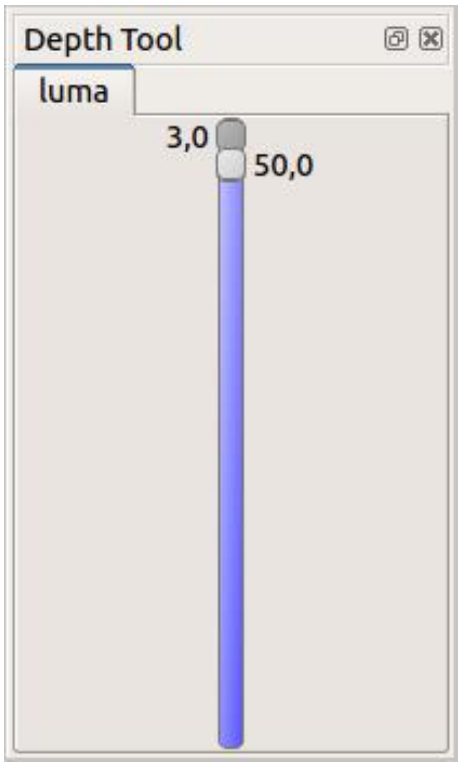


Figura 4.9: *Tool* do profundímetro.

LPTIMU

A figura 4.10 apresenta a Tool da IMU. Composto por uma bússola que exibe os valores do ângulos de *rumo*, *pitch* e *yaw* (em graus) medidos pelo sensor de unidade inercial e também apresenta o valor de referência para o rumo. O indicador azul representar o valor de *yaw* atual do robô e o indicador vermelho o valor de referência para o *yaw*. Também demonstra o número de voltas em torno do eixo z. A mensagem de ROS utilizada apresenta o seguinte formato:

```
float64 roll
float64 pitch
float64 yaw
```

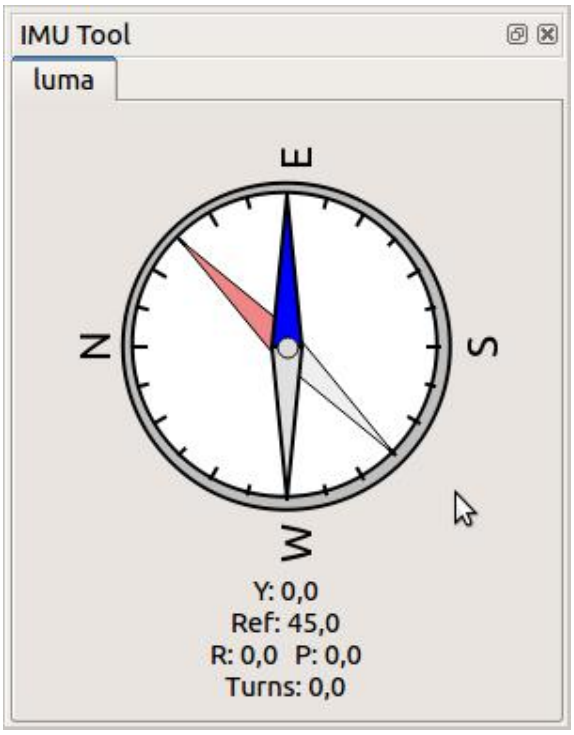


Figura 4.10: *IMU Tool*.

LPTMotionController

Tool responsável pelo controle do veículo. Conforme exibido pela figura 4.11, é possível enviar valores de referência, para o controle de posição em malha fechada ou o controle por força em malha aberta, para cada grau de liberdade do ROV. No caso do controle por força, há 4 valores para os quatro graus de liberdade. Para o controle de posição, são enviados 3 valores de posição e 1 de força para o eixo y. A mensagem de ROS utilizada no sistema é dada abaixo. O vetor *modes* contém os modos de controle para DOF, *originid* indica quem está enviando a mensagem, como *joystick* ou a Tool.

```
uint8 originid
uint8[] modes
float64[] data
```

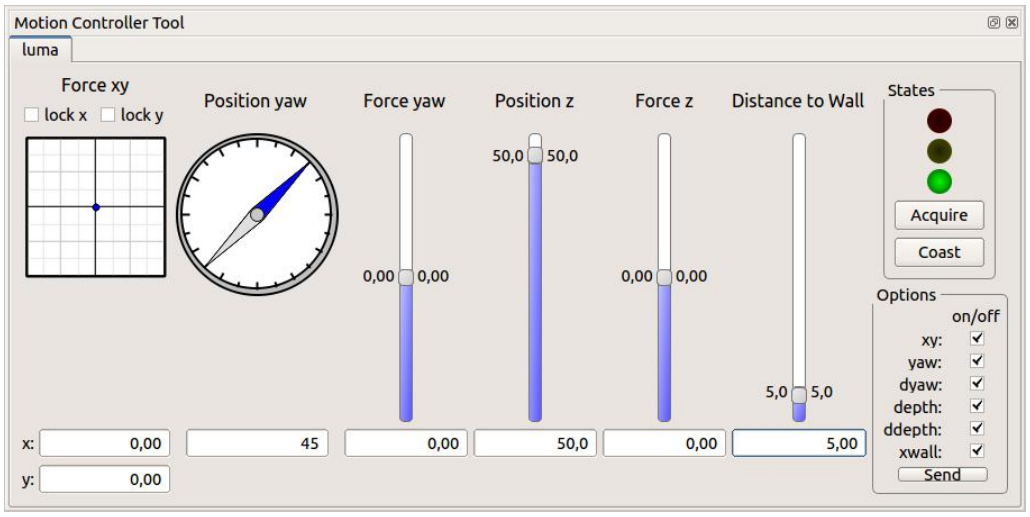


Figura 4.11: *Tool* do *MotionController*.

4.2 Operação Remota

Para a operação remota do ROV, são necessários dispositivos que permitem o usuário controlar o veículo manualmente. No caso do ROV LUMA, o dispositivo de entrada é o joystick da *MicroSoft XBox 360*, que possui 6 eixos e 11 botões e é ilustrado na Figura 4.12. Nesse sentido, os botões podem ter como finalidade acionar modos de controle enquanto os eixos possuem como função enviar comandos para a atuação no grau de liberdade desejado.

Figura 4.12: *Joystick* utilizado na Luma.

No sistema de controle desenvolvido para o ROV Luma, o sinal do eixo do dispositivo correspondente ao DOF que está a ser controlado é retransmitido diretamente como valor de força ou momento para o eixo. Para isso, é utilizado o pacote *joy* de ROS como base para o Component GPCBGamepad. Esse pacote consiste em um driver genérico contendo um nó de ROS que realiza a leitura dos eixos e dos botões do console.

O GPCBGamepad publica uma mensagem com o estado atual de todos os eixos e botões do *joystick*. Outro Component denominado LPCBGamepadAndKeyboard-

Mapper traduz os valores dos joysticks como um sinal de controle a ser executado no robô. Cada comando que corresponde a translação e rotação nos eixos do sistema de coordenadas do veículo é normalizado na faixa [-1,1]. Quando não há aplicação de comando no dispositivo, é publicado o valor 0. A tabela 4.1 resume os botões e eixos do controle.

Tabela 4.1: Botões e eixos do *joystick*.

Entrada	índice	
A	0	botão
B	1	botão
X	2	botão
Y	3	botão
LB	4	botão
RB	5	botão
Back	6	botão
Start	7	botão
Power	8	botão
Button left stick	9	botão
Button right stick	10	botão
Left/Right axis stick left	0	eixo
Up/Down axis stick left	1	eixo
LT	2	eixo
Left/Right axis stick right	3	eixo
Up/Down axis stick right	4	eixo
RT	5	eixo

Dessa forma, o LPCBGamepadAndKeyboardMapper envia o valor de força desejado por uma mensagem de ROS igual à utilizada na Tool do *MotionController*, dada por:

```
uint8 originid
uint8[] modes
float64[] data
```

Portanto, o operador consegue controlar manualmente os 4 graus de liberdade do veículo. Esse controle foi dividido em dois modos: *Standard Motion* e *Individual Motion*. Os modos são acionados pelo *joystick* e serão detalhados no Capítulo 5. É importante ressaltar que o sistema foi desenvolvido de forma que outros *joysticks* tenham implementação fácil e rápida.

Como exibido na Figura 4.7, o usuário deve clicar no botão Update ao conectar

o *joystick* no computador base. A tabela 4.2 apresenta os comandos do *joystick* necessários para operar nos dois modos do ROV Luma.

Tabela 4.2: Tabela com comandos de operação do ROV.

Modo de Controle	Entrada	Descrição
Standard Motion	Back	iniciar Standard Motion
	Left/Right axis stick left	controle no eixo x
	Up/down axis stick left	controle no eixo y
	LB	acionar controle no eixo z no Standard Motion
	RB	acionar controle do yaw no Standard Motion
	Left/Right axis stick right	controle do yaw
	Up/down axis stick right	controle do eixo z
Individual Motion	Start	iniciar Individual Motion
	X	acionar controle no eixo x no Individual Motion
	Y	acionar controle no eixo y no Individual Motion
	B	acionar controle no eixo z no Individual Motion
	A	acionar controle do rumo no Individual Motion
	Left/Right axis stick left	controle do eixo x ou rumo no Individual Motion
	Up/down axis stick left	controle do eixo y ou z no Individual Motion

4.3 Conclusões

Neste Capítulo, as ferramentas necessárias para a operação do ROV Luma foram detalhadas. Foi detalhado a arquitetura de *software* da Luma e a interface gráfica responsável por fazer a conexão homem-máquina.

Conforme explicado, a propriedade modular da arquitetura de software permite o desenvolvimento da interface para outros ROVs de maneira fácil e prática. Além disso, conforme o escopo do posicionamento dinâmico e operação remota, foram apresentados as Tools que auxiliam o operador nesse aspecto. Também é apresentado o *joystick* de operação do veículo.

Capítulo 5

Estratégias de Controle

Neste capítulo será apresentada a estratégia de controle para o ROV Luma. Para isto, são novamente detalhadas todas as etapas anteriores ao desenvolvimento do controlador. O veículo terá três malhas de controle que poderão estar ativas simultaneamente ou individualmente. Estas malhas correspondem ao controle automático do rumo, da profundidade e da distância do robô até uma parede a sua frente. Desse modo, o operador poderá controlar por força apenas o deslocamento lateral paralelo à parede do ambiente.

Primeiramente, são expostos os compensadores utilizados no ROV para a linearização da dinâmica dos propulsores. Para a tarefa de controle, é necessário que os propulsores estejam compensados. Posteriormente é descrito uma nova abordagem da dinâmica do veículo abordado no Capítulo 2. Esta nova abordagem permitirá o controle de cada grau de liberdade. Por fim, após as simplificações feitas, é proposto o controlador P-PI.

5.1 Compensadores

Conforme detalhado em Goulart (2007), os propulsores utilizados no ROV LUMA possuem as seguintes não-linearidades:

- Saturação,
- Zona Morta,
- Característica quadrática velocidade x empuxo.

Ainda, apresentam uma dinâmica de primeira ordem com uma constante de tempo associada.

Os limites de operação do propulsor são definidos pela saturação. A característica quadrática e da zona morta dos propulsores foram levantadas experimentalmente em

Goulart (2007). No experimento, a velocidade de rotação da hélica é medida de pela frequência de pulsos no *encoder* do propulsor. A Figura 5.1 ilustra o resultado:

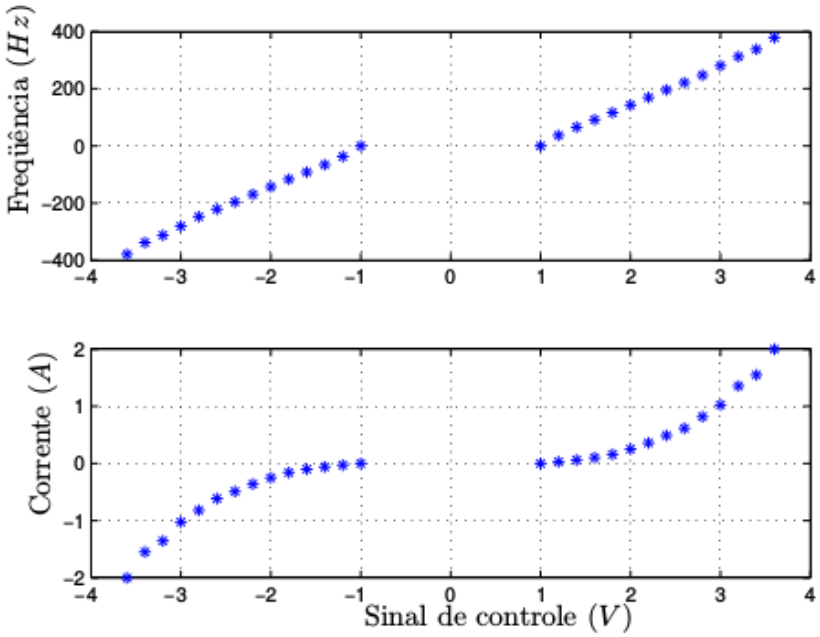


Figura 5.1: Curvas características dos propulsores (Extraído de Goulart (2007)).

Para o projeto efetivo do controlador, é fundamental compensar essas características. Com isso, são adicionados dois compensadores: estático e de zona morta. Eles tem como finalidade garantir maior linearidade na resposta dos propulsores. A compensação da saturação dos propulsores será explicada na seção 5.4.1.

5.1.1 Compensador Estático

A característica quadrática dos propulsores apresenta comportamento não-linear acentuado. A compensação é feita, segundo Cunha (1992), utilizando um operador com característica inversa de modo que o comportamento dos controladores torna-se linear independente das condições de operação.

Dessa forma, o compensador não-linear utilizado no ROV LUMA é do tipo raiz quadrada e dado por:

$$\bar{n}_{pi} = sgn(\bar{F}_{pi}\alpha_i^+)\sqrt{\left|\frac{\bar{F}_{pi}}{\alpha_i^*}\right|}, \tag{5.1}$$

onde $\bar{F}_{pi}$ é o empuxo do i-ésimo propulsor e:

$$a_i^* = \begin{cases} \alpha_i^+ = C_T^*(0^o)\frac{\rho}{8}0,7^2\pi^3D^4, & \text{se } sgn(\bar{F}_{pi})\alpha_i^+ \geq 0; \\ \alpha_i^- = C_T^*(180^o)\frac{\rho}{8}0,7^2\pi^3D^4, & \text{se } sgn(\bar{F}_{pi})\alpha_i^+ < 0. \end{cases} \tag{5.2}$$

O esquema do compensador é resumido pela figura 5.2.

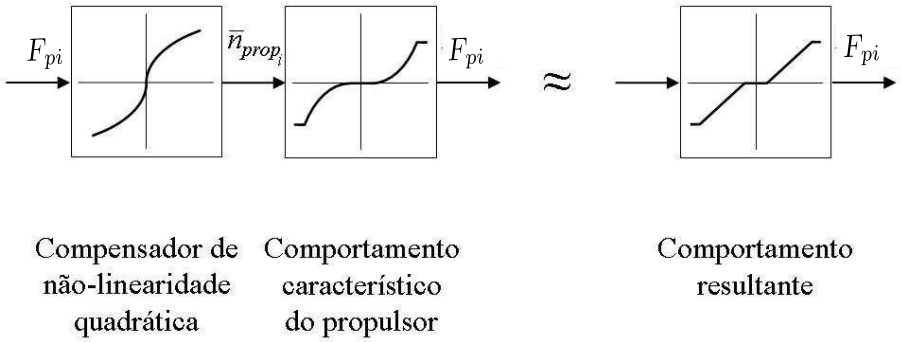


Figura 5.2: Compensação da característica quadrática do propulsor (Adaptado de Goulart (2007)).

5.1.2 Compensador de Zona Morta

Conforme descrito em Goulart (2007), os motores dos propulsores apresentam uma zona morta quando alimentados por uma tensão na faixa -1 e $+1$ V, ou seja, os motores não funcionam quando alimentados por uma tensão abaixa de um valor mínimo (u_{min}).

Assim, essa não-linearidade é compensada adicionando essa tensão mínima u_{min} ao módulo do sinal de controle (Carneiro et al. (2006)):

$$\bar{u}_{pi} \begin{cases} \bar{n}_{pi} + u_{min}, & \text{se } \bar{n}_{pi} > 0; \\ \bar{n}_{pi} - u_{min}, & \text{se } \bar{n}_{pi} < 0; \end{cases} \tag{5.3}$$

A figura 5.3 ilustra a compensação do propulsor:

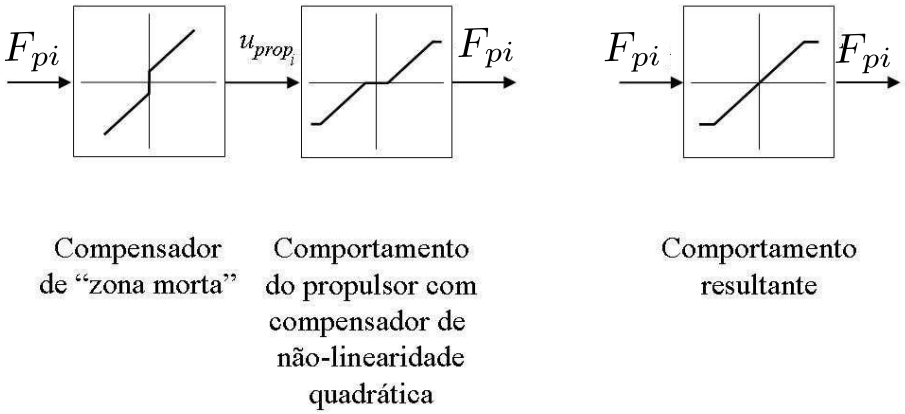


Figura 5.3: Compensação da zona morta do propulsor (Adaptado de Goulart (2007)).

5.2 Modelo Simplificado da Dinâmica

A partir das explicações do Capítulo 2, percebe-se que a dinâmica do veículo é bastante acoplada, isto é, para o movimento translacional ou rotacional ao longo de um eixo, é necessário o acionamento de um conjuntos de propulsores simultaneamente. Dessa forma, a dinâmica de cada grau de liberdade é representado por um sistema multivariável (MISO). Entretanto, ao utilizar uma matriz de desacoplamento, torna-se possível simplificar a dinâmica de cada DOF como um sistema SISO composto por um duplo integrador. A figura abaixo exemplifica a dinâmica para o rumo, que deveria ser originalmente modelada por um sistema com três entradas e uma saída. O propulsor central (P_5) e a força no eixo z (F_z) foram desconsideradas no modelo, pois afetam pouco a dinâmica do rumo.

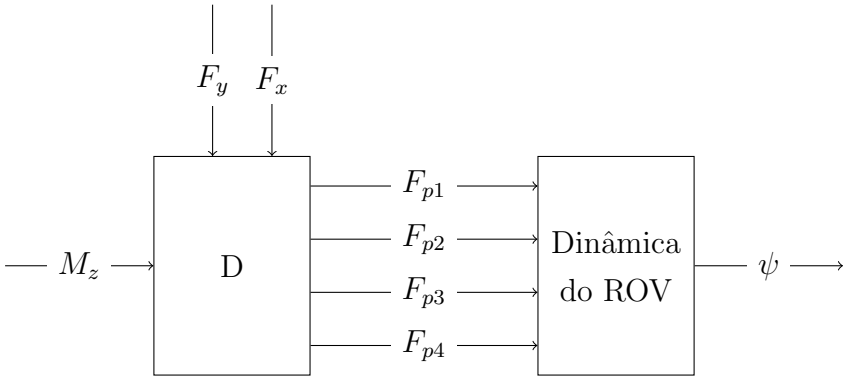


Figura 5.4: Modelo de sistema SISO para o rumo.

De maneira geral, as forças em cada propulsor são dadas por:

$$\begin{bmatrix} F_{p1} \\ F_{p2} \\ F_{p3} \\ F_{p4} \\ F_{p5} \end{bmatrix} = D \begin{bmatrix} F_{px} \\ F_{py} \\ F_{pz} \\ M_{pz} \end{bmatrix} \tag{5.4}$$

sendo F_{pi} o empuxo no i-ésimo propulsor, F_{px},F_{py},F_{pz}, as forças resultantes nos eixos x,y,z, respectivamente, e M_{pz} o momento resultante no eixo z. Desse modo, segundo Cunha (1992), desconsiderando as forças hidrodinâmicas, a dinâmica para um grau de liberdade genérico é dada por:

$$\ddot{x}(t) = ku(t) \tag{5.5}$$

onde u corresponde a força ou momento resultante, k ao ganho de alta frequência do sistema.

Dessa forma, conforme da Cunha et al. (1991), utilizando uma matriz de desaco-

plamento completa, é possível projetar um controlador linear de uma variável para cada DOF. Também é necessário que os propulsores estejam devidamente compensados e não operem saturados. Além disso, o ROV Luma é passivamente estável, de modo que os ângulos *roll* e *pitch* são próximos de 0 quando o veículo estiver em operação.

5.2.1 Identificação do Ganho de Alta Frequência

Como demonstrado na seção anterior, a dinâmica considerada no controle apresenta um ganho de alto frequência k. Para isso, foi implementando o Método do Relé, desenvolvido em Åström and Wittenmark (2013). Este método consiste em uma malha de realimentação negativa com a adição de um relé, como demonstrada na figura a seguir.

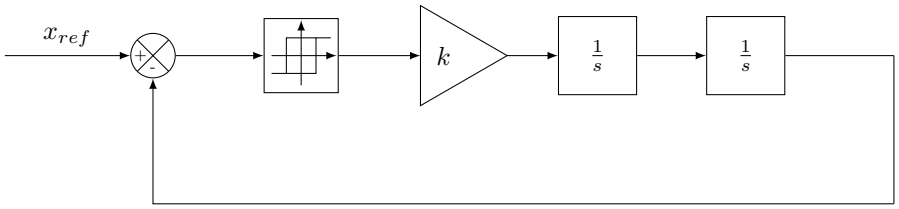


Figura 5.5: Diagrama de blocos para o método do Relé

Método do Relé

Considerando uma realização no espaço de estados do sistema de 2ª ordem dado pela equação 5.5, como estados $x_1 = x$ e $x_2 = \dot{x}$:

$$\frac{dx_1}{dt} = x_2, \tag{5.6}$$

$$\frac{dx_2}{dt} = ku. \tag{5.7}$$

O sinal de controle fornecido pelo relé dado por:

$$u = -\delta sgn(x),$$

$$sgn(x) = \begin{cases} 1, \text{ se } x > 0; \\ -1, \text{ se } x \leq 0; \end{cases}$$

resulta em uma trajetória no plano de fases formada por dois ramos de parábola. Para $u = -\delta$, tem-se:

$$\frac{dx_2}{x_1} = -\frac{k\delta}{x_2}, \tag{5.8}$$

$$x_2^2 = -2k\delta(x - x_{1_0}) + x_{2_0}^2 \tag{5.9}$$

onde $x_{1_0} := x(0)$ e $x_{2_0} := \dot{x}(0)$. Analogamente, para $u = \delta$, é obtido o que corresponde a figura 5.6:

$$x_2^2 = 2k\delta(x - x_{1_0}) + x_{2_0}^2 \tag{5.10}$$

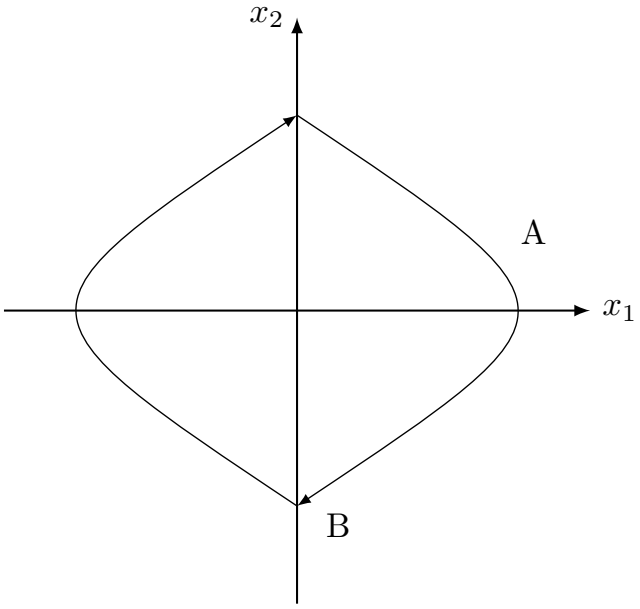


Figura 5.6: Trajetória de fases para o método do Relé.

O ganho de alta frequência pode ser encontrado em função do período de oscilação da trajetória demonstrada na figura 5.6. Para determinar o período de oscilação, considere o ponto $(A > 0, 0)$ como condição inicial e o ponto $(0, B)$ como condição final. Neste trecho, a dinâmica é dada por $\ddot{x} = -k\delta$, implicando em:

$$x_2 = \dot{x} = -k\delta t$$
$$x(t) = A - k\delta\frac{t^2}{2},\ \forall t \geq 0.$$

Desse modo, o tempo necessário para o estado ir de $(A > 0, 0)$ até $(0, B)$ é de:

$$t = \sqrt{\frac{2A}{k\delta}}$$

O percurso de $(A > 0, 0)$ até $(0, B)$ corresponde a um quarto do trajeto descrito pela Figura 5.6. Então o período de oscilação é $T = 4t$, logo, o ganho k é definido por:

$$k = \frac{32A}{\delta T^2} \tag{5.11}$$

Com isso, ao utilizar este método, é esperado um comportamento para cada DOF da seguinte forma:

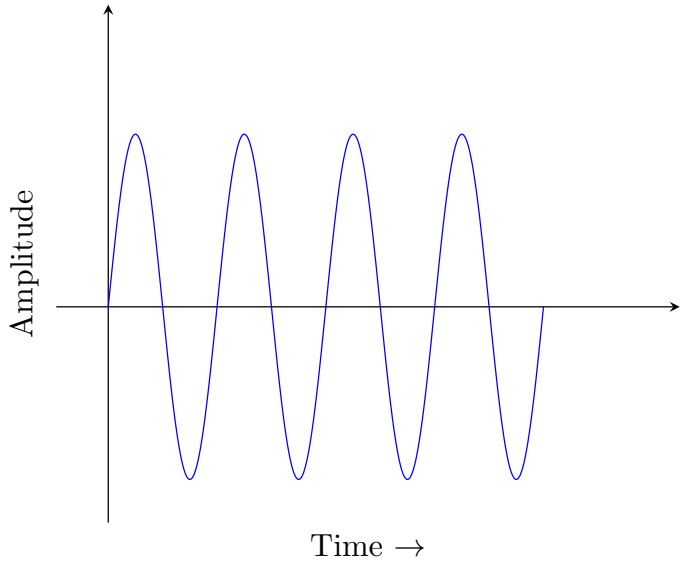


Figura 5.7: Oscilação do DOF com o método do Relé.

5.3 Modos de Controle

No ROV Luma há diferentes modos de operação, descritos a seguir:

1. Modo Manual: este modo é dividido em dois modos. O primeiro, chamado de *Standard Motion*, permite o operador controlar os 4 DOF do veículo, enquanto no segundo, denominado *Individual Motion*, o operador controla apenas 1 grau de liberdade. Os valores correspondentes ao *joystick* representam o vetor $\begin{bmatrix} F_x & F_y & F_z & M_z \end{bmatrix}$, onde cada elemento representa o valor de força (ou momento) desejado para cada eixo. Neste modo, o veículo Luma é operado em malha aberta.

2. Modo Rumo Automático: esse modo é ativado assim que é escolhido o modo de controle *P-PI* pela interface gráfica. Dessa forma, uma malha de controle é fechada, sendo que a medição da orientação do ROV é realizado pelo sensor de medição inercial descrito no Capítulo 2.

3. Modo Profundidade Automática: a profundidade passa a ser controlada automaticamente no momento que o operador escolhe o controle *P-PI* (detalhado

na próxima seção) pela interface. Com isso, fecha-se uma malha de controle para a manutenção do valor da profundidade. A medição é feita pelo sensor de pressão detalhado no Capítulo 2.

4. Modo Distância em x Automático: para ativação desse método é necessário que o controle *P-PI* seja escolhido e que o sonar esteja detectando uma parede para que possa fechar a malha de controle.A realimentação é realizada com a obtenção da distância do veículo à parede explicado no Capítulo 4.

O controle *P-PI* foi escolhido devido à experiência do laboratório e trabalhos anteriores, como Cunha (1992); Cunha et al. (1995); Hsu et al. (2000*a*). Ele é adequado para obter os objetivos propostos, pois apresenta fácil implementação, o objetivo de controle permite utilizar um controle linear e há possibilidade de sintonia *in loco*.

5.3.1 Estratégia de Conversão de Coordenadas

Antes da implementação do controle, é necessário estabelecer um critério para a conversão de coordenadas. O critério utilizado é a conversão do erro de posição. Conforme Cunha (1992), esta estratégia consiste em converter o erro de posição e a velocidade de translação (calculados no sistema inercial) para o sistema de coordenadas o veículo, enquanto os sinais de comando (forças e momentos) permanecem representados no sistema inercial. Com isso, o erro é calculado pela transformação

$$e_{bi} = R_z(\psi)^{-1}(p_{ref} - p)_i$$

onde e_{bi}, p_{ref} *e* $p_i \in \mathbb{R}^{3\times1}$

O diagrama de blocos da Figura 5.8 ilustra a conversão do sistema.

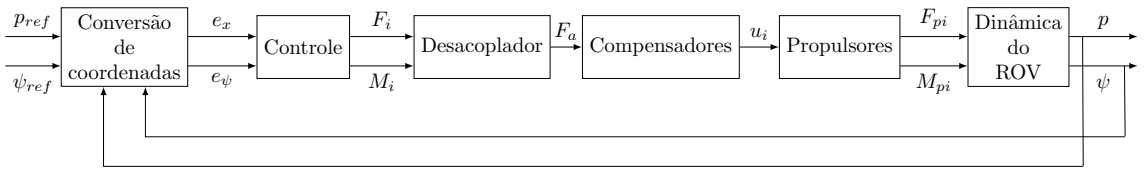


Figura 5.8: Diagrama de blocos para a estratégia de conversão do erro

Esta estratégia permite controle linear em razão do ganho de alta frequência k ser constante.

5.4 Controlador P-PI

A utilização de um controlador P-PI promove a eliminação do erro de regime causado por pertubações externas. Esse sistema é formado por uma malha interna PI

para controle de velocidade e uma malha externa com um controlador proporcional para o controle de posição.

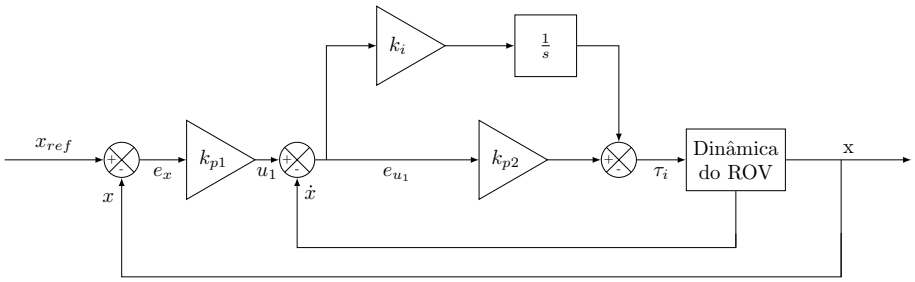


Figura 5.9: Diagrama de blocos para o P-PI

A figura 5.9 mostra o esquema de controle aplicado para um grau de liberdade genérico x do ROV. O ajuste do ganho é baseado na alocação de pólos. Contudo, os pólos não podem ser escolhidos arbitrariamente. Para isso, é feito o ajuste nas seguintes etapas:

- Ajusta-se a malha de controle de velocidade (malha interna).
- Ajusta-se a malha de controle de posição (malha externa).

No ajuste da malha PI (malha interna), abre-se a malha externa. Com isso, a equação característica da função de transferência para um DOF genérico é dado por

$$s^2 + kk_{p2}s + kk_i = 0. \tag{5.12}$$

onde k é o ganho de alta frequência para o grau de liberdade controlado.

Assim, os ganhos k_{p2} e k_i são calculados de forma que os pólos da malha interna sejam duplos e localizados em $a = bc$, onde b é pólo dominante do controlador P-PI e $c > 1$. Então, considerando a equação 5.12, os ganhos da malha são:

$$k_{p2} = -\frac{2cb}{k}, \tag{5.13}$$

$$k_i = \frac{(cb)^2}{k}, \tag{5.14}$$

Na próxima etapa é realizado o ajuste da malha externa. A malha é fechada e é obtido a seguinte equação característica:

$$s^3 - 2cbs^2 + (c^2b^2 - 2cbk_{p1}) + (cb)^2k_{p1} = 0 \tag{5.15}$$

Com isso, a equação acima deve ser tal que tenha b como pólo dominante. Para obtenção desse resultado, é necessário que ganho da malha P seja

$$k_{p1} = \frac{b(2c - c^2 - 1)}{c(c-2)} \tag{5.16}$$

Dessa forma, a alocação de pólos no sistema completo ficaria conforme a figura 5.10.

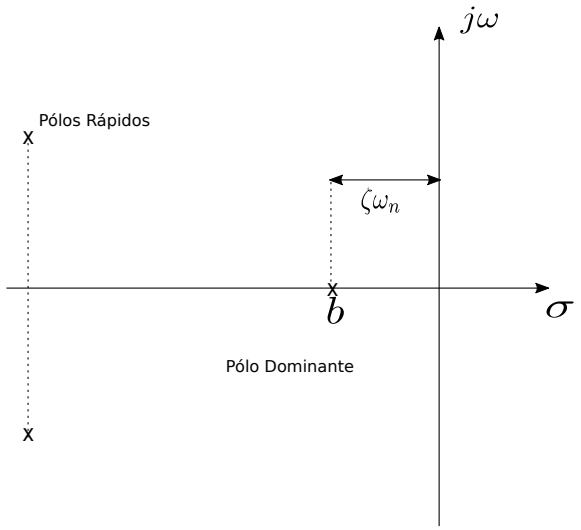


Figura 5.10: Alocação de pólos para o Controlador P-PI

Depois de realizado a estimação dos parâmetros, considera-se a seguinte observação:

Observação 1. O controle P-PI requer que a velocidade ($\dot{x}$) do ROV seja conhecida, porém esta não é medida diretamente. Sendo assim, uma alternativa adotada é a estimação da velocidade a partir da posição (x). Para isso, cada velocidade na coordenada desejada é estimada por um filtro *lead* de primeira ordem com função de transferência

$$L(s) = \frac{s}{(\tau s + 1)}, \tag{5.17}$$

onde τ é a constante de tempo do filtro.

O pólo do filtro *lead* está, portanto, em $-1/\tau$. A constante de tempo τ deve ser alto suficiente a fim de evitar amplificação do ruído de alta frequência. No entanto, instabilidade em malha fechada pode acontecer se o valor de τ for muito alto.

5.4.1 *Antireset Windup*

De acordo com Russel (1984), considera-se ainda:

Observação 2. Deve-se saturar a velocidade de referência da malha interna (v_1) a fim de que a velocidade máxima de deslocamento do ROV (u_{max}) seja limitada quando

o operador solicita movimentos amplos. Esta saturação permite que melhorar o desempenho do controlador de acordo com a limitação da velocidade do ROV.

Dessa forma, como proposto em Åström and Wittenmark (2013), é utilizado uma estratégia ARW *Antireset Windup* com o objetivo de evitar a saturação dos propulsores.

Portanto, um diagrama detalhado do controle implementado é apresentado na figura

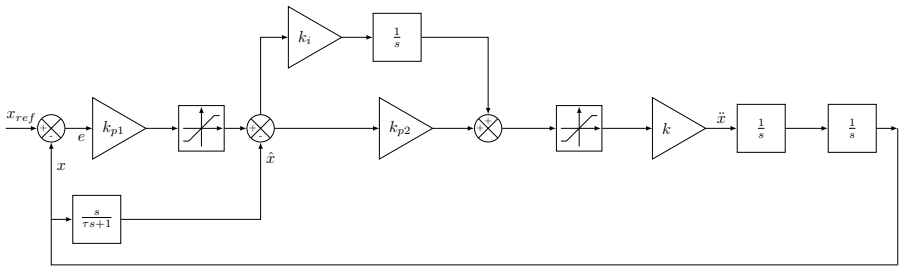


Figura 5.11: Diagrama de blocos detalhado para o P-PI

5.4.2 Algoritmo Implementado

O algoritmo implementado para o Método do Relé e do Controle P-PI são ilustrado abaixo. Esse é chamado periodicamente pela camada de controle do software da Luma e foi implementado em linguagem C++. O filtro *lead* dado pela equação 5.17 foi discretizado utilizando a aproximação *backward* Euler.

```
ControlSystemTemplate<2,1,5>::CalculateReturn RelayControl::calculate(const Eigen::
    Matrix<double, 2, 1>& u)
{
//Cálculo do erro
double error = u(1,0) - u(0,0);

double tmp = fabs(error);
double tmpctrl;

if (tmp > param.delta)
{
tmpctrl =  param.u_max * error/tmp;
param.u_old = tmpctrl;
vectors.output(0,0) = tmpctrl;
}

else if (!tmp)
{
vectors.output(0,0) = 0.0;
}
else
{
vectors.output(0,0) = param.u_old;
}

param.u_old = vectors.output(0,0);
```

```
return vectors;
}
```

```
ControlSystemTemplate<2,1,5>::CalculateReturn PPIControl::calculate(const Eigen::
    Matrix<double, 2, 1>& input)
{
double e, aux;
PPI_Y = input(0,0);
e = PPI_REF - PPI_Y; //Cálculo do erro de medição

//Caso seja controle do rumo, erro ficará entre [-180,180]
if (ang)
{
e = fmod(e, 2*M_PI);
if(e  > M_PI)
e  = e  - 2*M_PI;
else if(e  < -M_PI)
e  = e  + 2*M_PI;
}

//Cálcula da derivada
PPI_DY = param.kbackward * (PPI_DY + (PPI_Y - PPI_Y_OLD)/param.tau);
aux = param.kp * e;
e_line = saturation(aux, param.vmax, -param.vmax) - PPI_DY;
double u = saturation(param.kd * e_line + PPI_I, param.umax, -param.umax);

ControlSystemTemplate<2,1,5>::CalculateReturn ret; //vetor que armazena resultado

ret.output(0,0) = u;

PPI_I = saturation(PPI_I+ param.ki * e_line * param.h, 1.0, -1.0);
PPI_Y_OLD = PPI_Y;

return ret;
}

double PPIControl::saturation(double x, double x_max, double x_min)
{
if (x >= x_max) return x_max;
if (x <= x_min) return x_min;
return x;
}
```

5.5 Conclusões

Neste Capítulo foi apresentado a estratégia de controle para a Luma. Primeiramente foi analisado as não-linearidades dos propulsores, como a zona morta e a característica quadrática velocidade × empuxo. Dessa forma, são apresentados as compensações necessárias para a linearização da dinâmica dos propulsores.

Em seguida, para a realização do controle, foi realizado a simplificação do modelo dinâmico do veículo para um grau de liberdade. Essa simplificação é dada pela

equação 5.5 e é possível devido a utilização de uma matriz de desacoplamento. O ganho de alta frequência k é obtido a partir do Método do Relé.

Por fim, são apresentados os modos de controle do ROV Luma e o desenvolvimento completo do controlador P-PI implementado.

Capítulo 6

Resultados das Simulações

Neste Capítulo, será ilustrado o simulador utilizado e as simulações do algoritmo de controle implementado.Conforme exibido na Figura 1.3, a comunicação entre o sistema de controle e o microcontrolador embarcado é realizado por meio de troca de mensagens RS485. Dessa forma, a comunicação com o simulador foi feita utilizando mensagens de ROS. O nó de ROS do simulador receberá o valor do controle desejado para cada grau de liberdade. Com isso, resolverá as equações dinâmicas do movimento do ROV e transmitirá uma mensagem contendo posição e orientação do veículo. O diagrama abaixo resume a comunicação entre os sistemas.

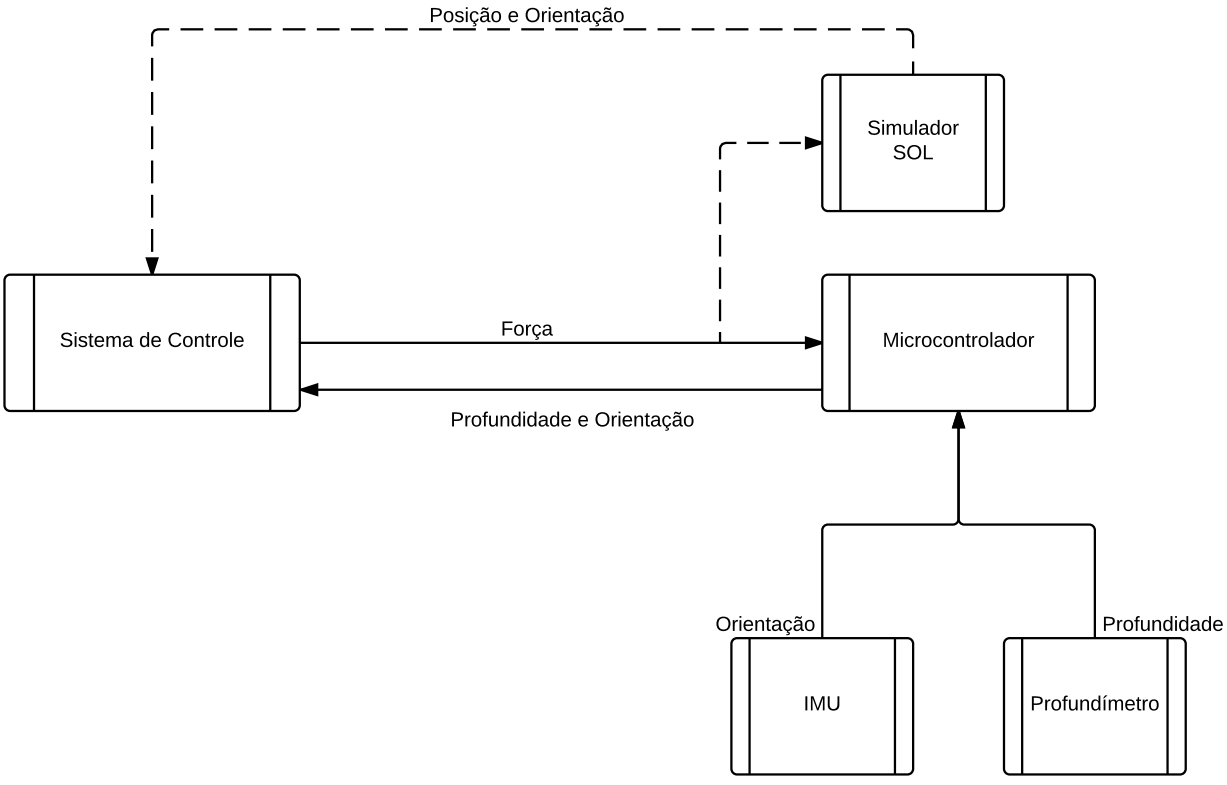


Figura 6.1: Diagrama de comunicação entre o sistema de controle e o simulador.

Para isso, as mensagens de ROS foram salvas em um *bag*. Posteriormente, foi utilizado o *MatLab* (*Mathwork Inc*) para melhor visualização dos gráficos.

As simulações foram divididas em etapas. Primeiramente, cada grau de liberdade foi excitado individualmente tal que sua resposta ficasse oscilatória. Com isso, foram obtidos o ganho de alta frequência para cada DOF. Posteriormente, o controlador P-PI foi projetado e simulado individualmente. Por fim, todos os controladores foram acionados simultaneamente e o comportamento do sistema foi analisado.

6.1 Simulador do ROV

A simulação do movimento do ROV é baseada no programa SOL desenvolvido em Hsu et al. (2000*a*). Este foi encapsulado em uma biblioteca estática que é utilizada para gerar um nó de ROS que calcula e transmite a Pose (posição e orientação) e a tf (transformadas dos *frames*) do ROV.

O simulador SOL resolve as equações de movimento do veículo abordadas em 2.21 utilizando algoritmos de integração como Runge-Kutta de quarta ordem ou Euler. Dessa forma, foi feita a integração desse simulador com a interface desenvolvida para o ROV Luma.

Como foi feito de forma modular, esse simulador permite a simulação de diferentes ROV's. Nos arquivos de configuração, é possível definir a quantidade de propulsores, a alocação destes e a matriz de desacoplamento correspondente. A medição pode ser feita de maneira exata ou apresentando ruídos. Também é implementado um observador de estados que pode ser ideal ou com um derivador causal. Uma das principais características do simulador é implementação dos efeitos da navegação de veículo na água. Estes efeitos estão descritos em Hsu et al. (2000*a*) são:

- efeito inercial: efeitos causados pelas massas e inércias do sistema;
- efeito gravitacional: forças e momentos causados pela gravidade e pela flutuação;
- efeito hidrodinâmico: efeitos de arraste que dissipam energia;
- efeito do cabo umbilical: efeito da força exercida pelo cabo umbilical sobre o ROV;
- efeito dos propulsores: efeitos das forças dos propulsores e das suas não-linearidades;
- efeito de quantização: quantização dos sinais de comando.

Desse modo, para uma integração total, no simulador do ROV são configurados os parâmetros do veículo abordados no Capítulo 2 e este publica uma mensagem de ROS em um tópico, no sistema de coordenadas inercial, da posição e orientação do ROV. Com taxa de amostragem $f = 100\ Hz$ ($10\ ms$) o nó de ROS atualiza o estado do veículo, ou seja, executa a simulação através da resolução da equação diferencial pelo tempo especificado pelo período de amostragem.

O *MotionController* do *software* recebe esses valores de medição e de referência enviados pela interface para o cálculo do erro e executa o algoritmo do controlador P-PI. O sinal de controle é, então, publicado para o simulador de ROV. O nó do *Component* de controle do *software* apresenta uma taxa de amostragem de 10 Hz (100 ms). As mensagens utilizadas para o estado do veículo no sistema de coordenadas inercial e o sinal de controle no sistema de coordenadas do veículo são, respectivamente *Pose* e *Twist*. Estas mensagens são definidas a seguir:

```
geometry_msgs/Point position
float64 x
float64 y
float64 z
geometry_msgs/Quaternion orientation
float64 x
float64 y
float64 z
float64 w
```

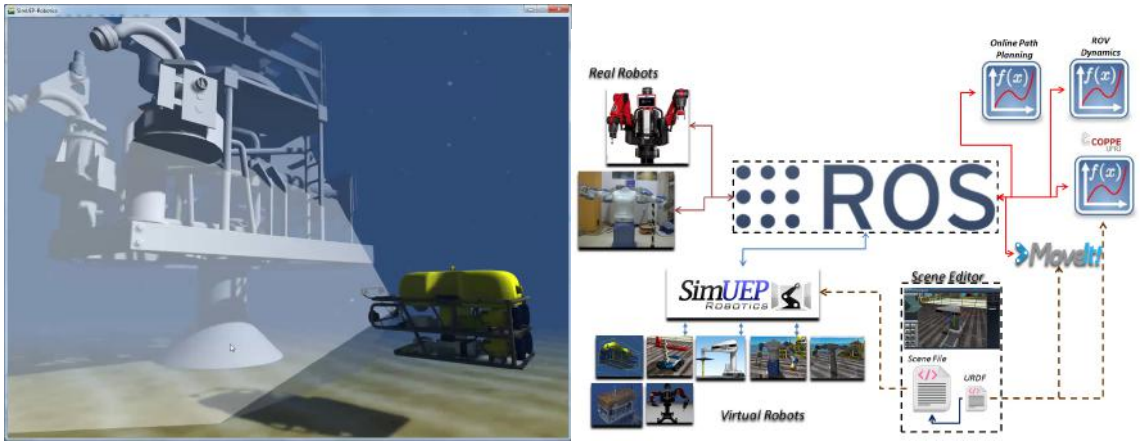
```
geometry_msgs/Vector3 linear
float64 x
float64 y
float64 z
geometry_msgs/Vector3 angular
float64 x
float64 y
float64 z
```

O algoritmo implementado é exibido a seguir. Foi necessário realizar mudanças de sinais para simular o sistema de coordenadas do ROV Luma, isto é, que o movimento do ROV correspondesse ao movimento esperado.

```
ros::Rate loop_rate(1/intervalo_de_amostragem);
while(_myOk){
// Make a spin to get new commands
ros::spinOnce();

// Execute the ROV model and refresh the states
executar_rovs(&vor);

// Escrever a mensagem de transformadas de coordenadas
```

```
// Posição
T.setOrigin(tf::Vector3(vor.r[1].estado.atual.posicao.x,
vor.r[1].estado.atual.posicao.y,
vor.r[1].estado.atual.posicao.z));
// Orientação - roll/pitch/yaw para Quaternion
quat.setEuler(vor.r[1].estado.atual.atitude.x,
vor.r[1].estado.atual.atitude.y,
vor.r[1].estado.atual.atitude.z);
T.setRotation(quat);

//Publish TF
tf_broad.sendTransform(tf::StampedTransform(T, ros::Time::now(), "/
    world", "/rov"));

//Publish pose
poseTFToMsg(T,pose);
pose_pub.publish(pose);

//Criação das mensagens da imu e do profundímetro
RobotPS::RPY rpy;
rpy.roll = vor.r[1].estado.atual.atitude.x;
rpy.pitch = vor.r[1].estado.atual.atitude.y;
//Sinal modificado para seguir o comportamento da LUMA
rpy.yaw = -vor.r[1].estado.atual.atitude.z;

LUMA::LPDepth depth;
depth.depth = -vor.r[1].estado.atual.posicao.z;

depth_pub.publish(depth);
rpy_pub.publish(rpy);

loop_rate.sleep();
}
```

É possível, também, integrar o simulador com o ambiente de realidade virtual *SimUEP-Robotics*. Esse ambiente de realidade virtual é detalhado em Carvalho et al. (2014). Com isso, o nó de ROS implementa as equações dinâmicas do veículo, enquanto o *SimUEP-Robotics* pode ser utilizado para a animação 3D do movimento do veículo. As figuras seguintes apresentam uma imagem do ambiente virtual e a forma de integração com ROS.



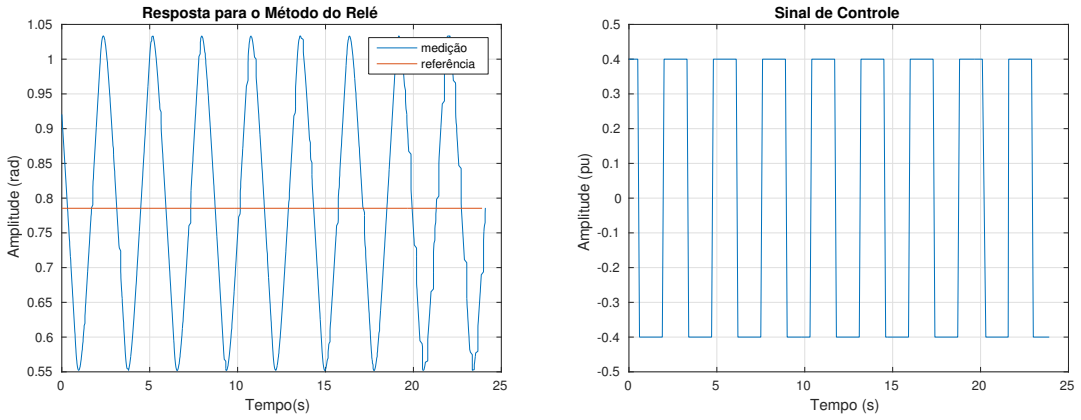
(a) (b)

Figura 6.2: (a) ROV implementado no ambiente de realidade virtual e (b) integração ROS e *SimUEP-Robotics*(Extraído de Carvalho et al. (2014))

6.2 Controle de Rumo

6.2.1 Identificação do Ganho de Alta Frequência

A primeira etapa para o controle do rumo é a identificação do ganho de alta frequência. Baseado no desenvolvimento da seção 5.2.1, foi feita uma realimentação com relé de amplitude $\delta = \pm 0.4$, histerese de 0.174533 (10°) e condição inicial de 45° . As figuras seguintes ilustram o resultado.



(a) (b)

Figura 6.3: Resposta (a) e sinal de controle (b) para o Método do Relé do rumo

Dessa forma, a amplitude e o período foram

$$A = 0.2476\ rad$$

$$T = 2.7720\ s$$

Conforme a equação 5.11, o ganho de alto frequência será:

$$k_{rumo} = \frac{32A}{\delta T^2} = 2.5778 \tag{6.1}$$

Encontrado o ganho de alta frequência, é possível realizar o projeto do controlador P-PI.

6.2.2 Controle P-PI

Primeiramente, é necessário estimar os parâmetros do controlador. Conforme explicado, aloca-se o pólo dominante do P-PI em $b = -0.5$ e é definido a constante $c = 3$. Dessa forma, conforme as equações 5.13, 5.14 e 5.16, tem-se:

$$\begin{aligned} k_{p2} &= -\frac{2cb}{k} = 1.1638, \\ k_i &= \frac{(cb)^2}{k} = 0.8728, \\ k_{p1} &= \frac{b(2c - c^2 - 1)}{c(c-2)} = 0.6667 \end{aligned}$$

O pólo do filtro *lead* foi alocado em $1/\tau = -5$. A taxa de amostragem foi de 100 Hz. Dessa forma, foi feita simulação na condição inicial $\psi_0 = 0°$. Posteriormente muda-se o valor de referência para $\psi_2 = 45°$ e em seguida para $\psi_2 = -20°$. Diferentemente do método da Relé, a saturação do sinal de controle e da velocidade do ROV é $u_{max} = 1$ e $v_{max} = 1$, respectivamente. As figuras 6.4a, 6.4b e 6.4c ilustram o resultado obtido.

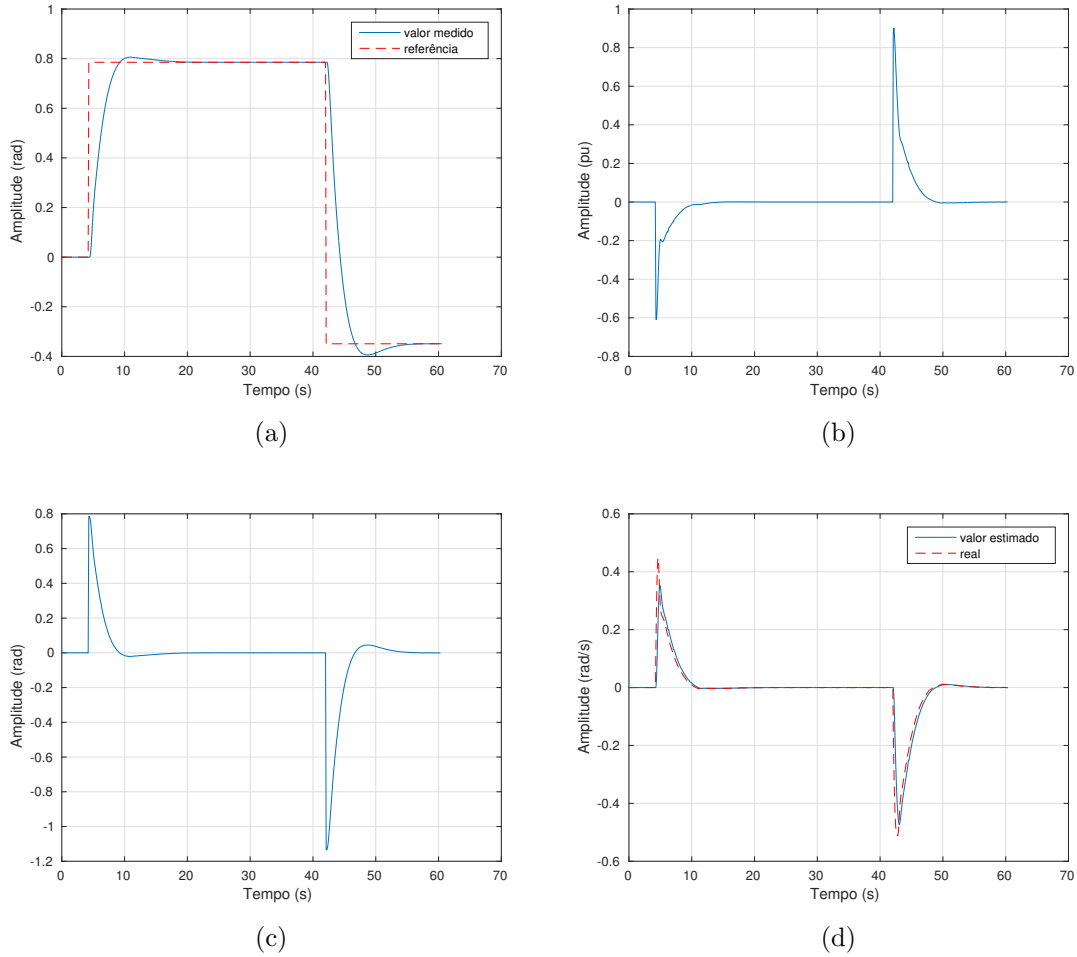


Figura 6.4: Respostas para o controle do rumo usando o controlador P-PI.(a) valor medido e referência do rumo em radianos, (b) sinal de controle, (c) erro em *rad*, (d) comparação da velocidade real e velocidade estimada em rad/s.

Durante o controle para $\psi = 45°$, a resposta obteve como valor máximo $\psi = 46.1861°$. Com isso, o *overshoot* foi de 2.5681%. Para o movimento até $-20°$ o *overshoot* foi de 12.8050% com o ROV chegando ao valor $\psi = -22.561°$ até estabilizar. Além disso, durante o movimento, o ROV LUMA seguiu o caminho mais rápido, conforme esperado. O filtro *lead* conseguiu estimar a velocidade do veículo de maneira adequada.

Em resumo, a tabela 6.1 apresenta todos os parâmetros relacionados ao controle do rumo:

Tabela 6.1: Parâmetros do controle do rumo

Parâmetros	Valor
k_{p2}	1.1638
k_i	0.8728
k_{p1}	0.6667
τ	0.2
k_{back}	0.6667
h	0.1
u_{max}	1
v_{max}	1

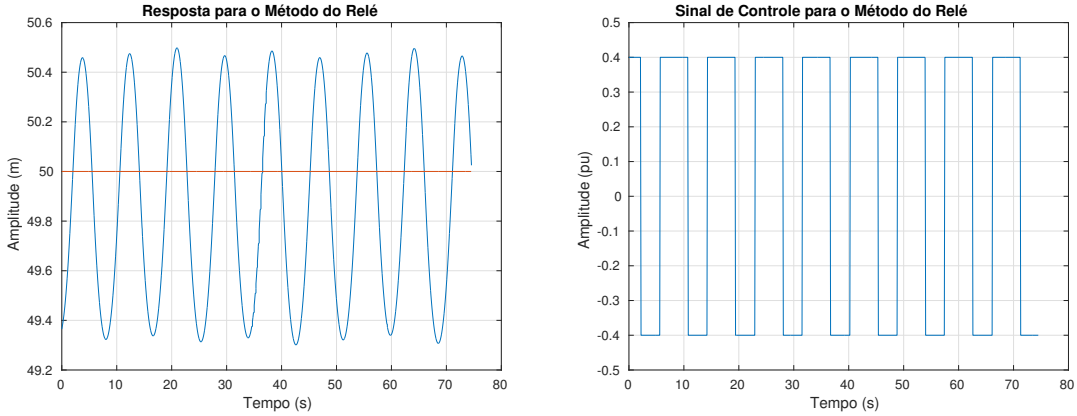
6.3 Controle de Profundidade

6.3.1 Identificação do Ganho de Alta Frequência

Conforme explicado no Capítulo 5, para identificar o ganho k, o sistema foi realimentado com um relé com histerese de 0.1 m e amplitude na faixa [-1,1]. A condição inicial do veículo era

$$z_0 = 50\ m$$

As figuras abaixo ilustram o resultado obtido.



(a) (b)

Figura 6.5: Respostas para a profundidade pelo Método do Relé. (a) Resposta (b) Sinal de controle

Observando o resultado, é obtido uma amplitude e período de oscilação:

$$A = 0.5m \quad T = 8.7s$$

Assim, com base na equação 5.11, o ganho de alto frequêcia é dado por:

$$k_{prof} = \frac{32A}{\delta T^2} = 0.5285 \tag{6.2}$$

6.3.2 Controle P-PI

Diferentemente do controle do rumo, o pólo do controle foi alocado em $b = -0.2$ enquanto a constante $c = 3$ foi mantida. Com isso, os ganhos do controlador são dados por:

$$\begin{aligned} k_{p2} &= -\frac{2cb}{k} = 2.2707, \\ k_i &= \frac{(cb)^2}{k} = 0.6812, \\ k_{p1} &= \frac{b(2c - c^2 - 1)}{c(c-2)} = 0.2667 \end{aligned}$$

Dessa vez, a condição inicial da posição em relação ao eixo z é mantida em $z_0 = 50\ m$. Posteriormente, o veículo é levado para a posição final $z_f = 65m$. O sinal de controle e velocidade foram saturados com valor máximo 1 igual ao controle de rumo. O pólo do filtro *lead* foi alocado em -2. As figuras a seguir apresentam a resposta obtida.

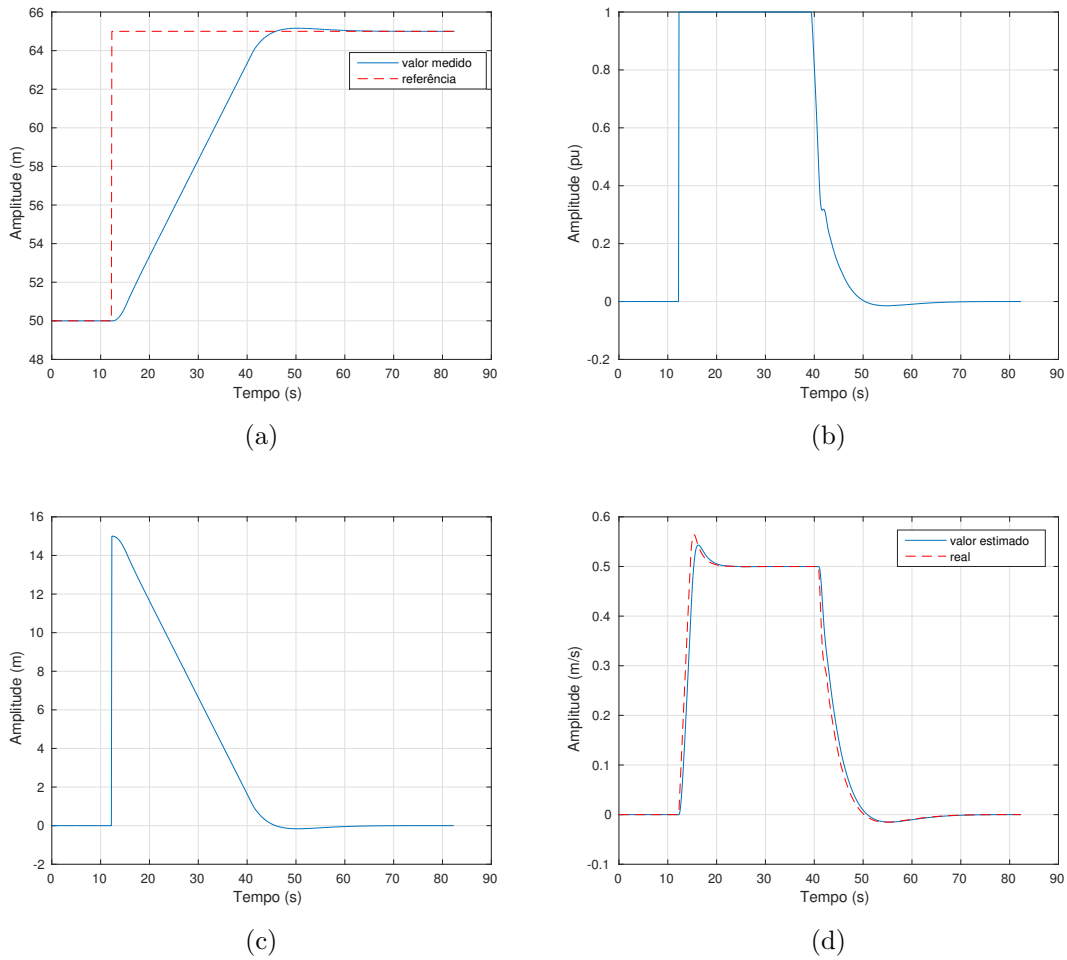


Figura 6.6: Respostas para o controle de profundidade usando o controlador P-PI.(a) valor medido e referência de profundidade em metros, (b) sinal de controle, (c) erro em metros, (d) comparação da velocidade real e estimada em m/s.

O valor máximo atingindo pelo veículo foi $z_{max} = 65.1607\ m$, o que significa *overshoot* de $0.2472\%\ m$. Ainda, durante o movimento até o novo valor de referência, controlador atuou saturado. A velocidade estimada pelo filtro correspondeu a velocidade real do ROV. O tempo de assentamento foi de aproximadamente $30\ s$. Como esperado, a estabilização para a profundidade é mais lenta que a do rumo.

Dessa forma, os parâmetros do projeto do controle de profundidade são resumidos pela tabela 6.2.

Tabela 6.2: Parâmetros do controle de profundidade

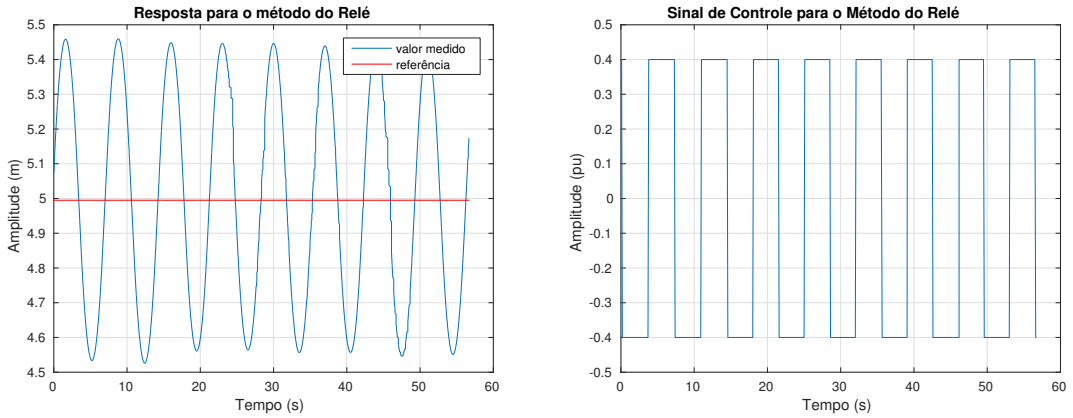
Parâmetros	Valor
k_{p2}	2.2707
k_i	0.6812
k_{p1}	0.2667
τ	0.5
k_{back}	0.8333
h	0.1
u_{max}	1
v_{max}	1

6.4 Controle no eixo X

6.4.1 Identificação do Ganho de Alta Frequência

Na etapa de identificação do ganho de alta frequência para o controle no eixo x, a condição inicial não poderia ser $x_0 = 0$, pois implicaria o ROV colidir com o obstáculo. Dessa forma, a condição inicial foi $x_0 = 5\ m$ e o relé apresentava amplitude na faixa de ± 0.4 e histerese de 0.1 m. Também foi considerada a orientação do robô como $\psi_0 = 0$ de modo que o veículo estivesse com a sua parte frontal na direção da parede.

Assim, o resultado obtido é apresentado nas figuras 6.7a e 6.7b.



(a) (b)

Figura 6.7: Respostas para o eixo x pelo Método do Relé. (a) Resposta (b) Sinal de controle

Os valores de amplitude e período de oscilação foram medidos e são dados por:

$$A = 0.46m \quad T = 7.213s$$

Consequentemente, o ganho de alta frequência é dado por:

$$k_x = \frac{32A}{\delta T^2} = 0.7073 \tag{6.3}$$

6.4.2 Controle P-PI

A alocação do pólo do controlador foi feita de modo análogo ao controle de profundidade, logo, $b = -0.2$ e $c = 3$. Dessa forma, os ganhos do controlador são:

$$\begin{aligned} k_{p2} &= -\frac{2cb}{k} = 1.6965, \\ k_i &= \frac{(cb)^2}{k} = 0.509, \\ k_{p1} &= \frac{b(2c - c^2 - 1)}{c(c-2)} = 0.2667 \end{aligned}$$

A condição inicial do veículo no eixo x foi $x_0 = 5m$ e em seguida o veículo foi levado para $x_f = 8m$. Conforme os outros testes, o sinal de controle e a velocidade são saturadas com valor máximo de 1 e a constante de tempo do filtro foi $\tau = 2$. O resultado é apresentado nas figuras e na tabela a seguir, respectivamente.

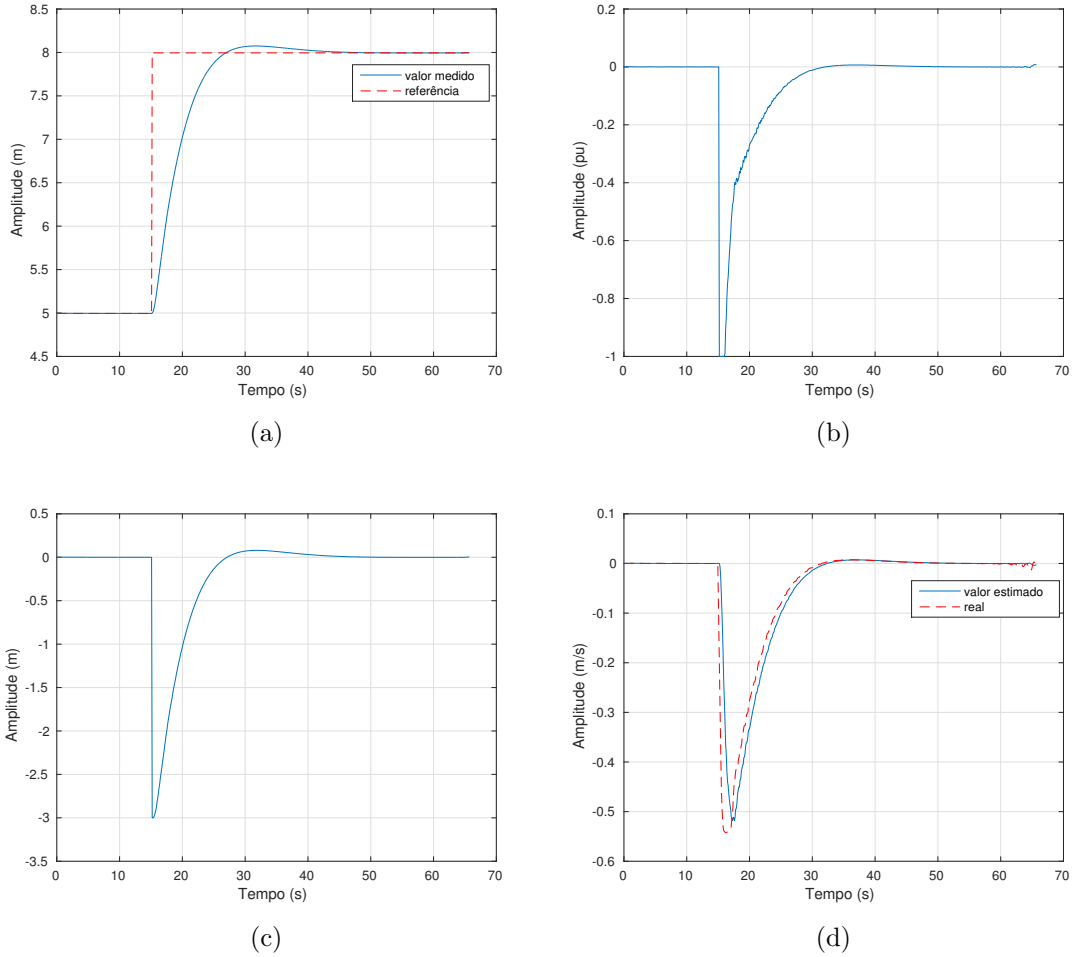


Figura 6.8: Respostas para o controle no eixo x usando o controlador P-PI.(a) distância a parede e o valor desejado em metros, (b) sinal de controle, (c) erro em metros,(d) comparação da velocidade estimada e real em m/s.

Tabela 6.3: Parâmetros do controle ao longo do eixo x

Parâmetros	Valor
k_{p2}	1.6965
k_i	0.509
k_{p1}	0.2667
τ	0.5
k_{back}	0.8333
h	0.1
u_{max}	1
v_{max}	1

O valor máximo que o veículo chegou foi de

$$x_{max} = 8.0748\ m$$

o que significa *overshoot* de 0.9263%. Além disso, a resposta estabilizou aproximadamente em 25 s.

Teste com Orientação Diferente de Zero

A simulação exibida pela Figura 6.8 foi realizada considerando o ROV com orientação $\psi = 0°$. Dessa forma, foi necessário simular, também, o caso da orientação do veículo estiver diferente de zero. A orientação do robô foi controlada para estabilizar em $\psi = 45°$. Em seguida foi feito movimento semelhante ao primeiro teste: translação de 5 m até 8 m no sistema de coordenadas inercial da parede. Utilizando os mesmo parâmetros dados pela tabela 6.3, as figuras abaixo apresentam o resultado.

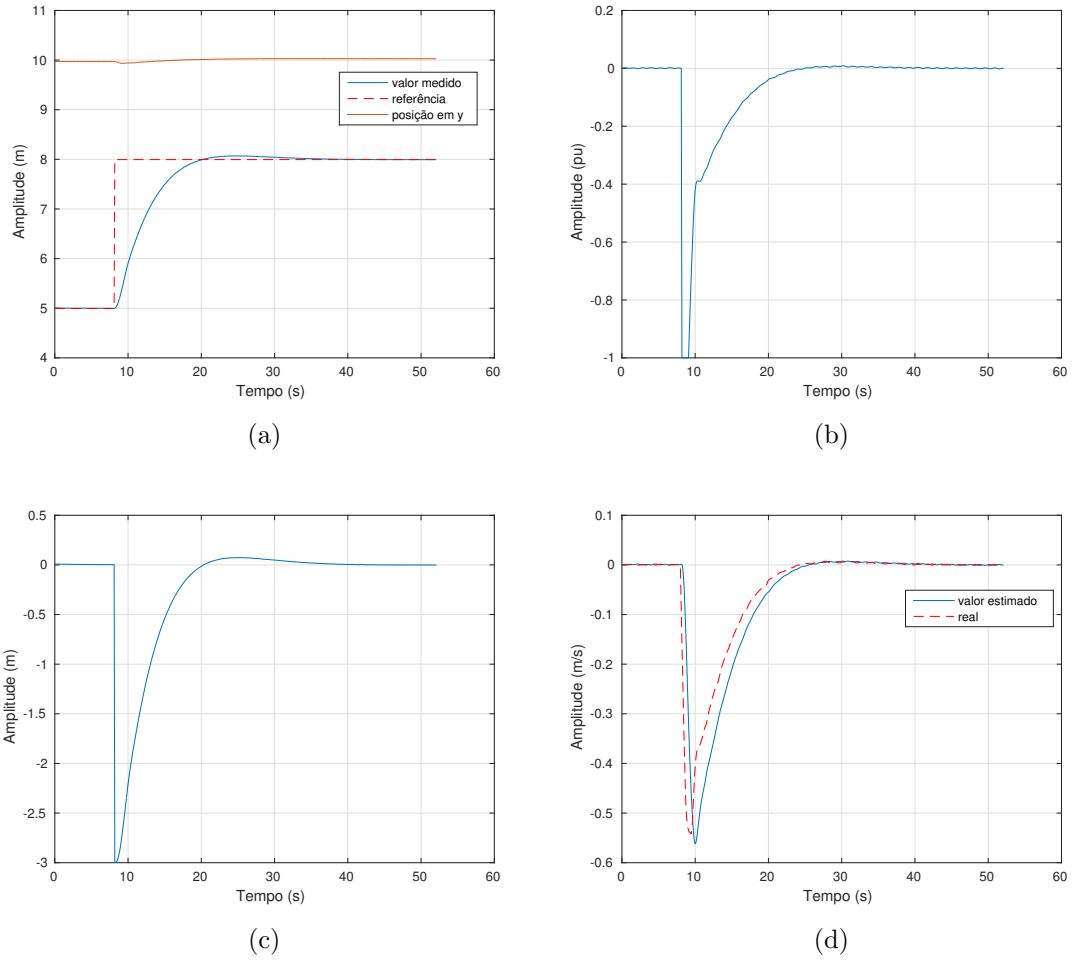


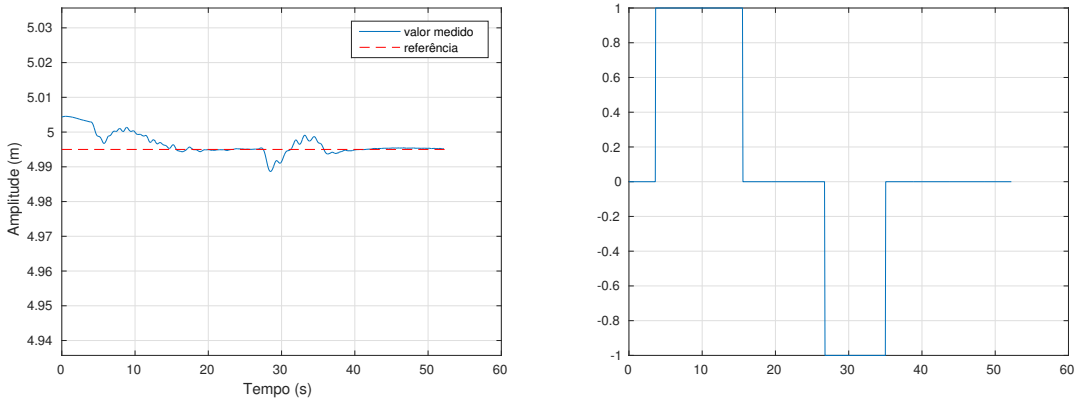
Figura 6.9: Respostas para o controle no eixo x usando o controlador P-PI e orientação a $\psi = 45°$.(a) distância a parede e o valor desejado em metros, (b) sinal de controle, (c) erro em metros,(d) comparação da velocidade estimada e real em m/s.

Analisando o resultado, nota-se o comportamento adequado do sistema. A posição do sistema no eixo y foi mantida. A resposta para o grau de liberdade

controlado teve como valor máximo $x_{max} = 8.0683$, o que representa *overshoot* de 0.8465%. No entanto, o desempenho do filtro *lead* foi pior que no caso primeiro caso.

Teste com Translação no eixo y

A próxima etapa da simulação é aplicar uma força no eixo y para verificação se o controlador manterá o valor de referência no eixo x. As condições de operação foram orientação $\psi = 0°$ e $x_{ref} \approx 5\ m$. Novamente, os parâmetros de controle foram os mesmos da tabela 6.3.



(a) (b)

Figura 6.10: Respostas para o controle no eixo x usando o controlador P-PI e aplicando uma força no eixo y.(a) distância a parede e o valor desejado em metros, (b) forças aplicadas no eixo y normalizado em ± 1.

Portanto, esse teste permitiu analisar o desacoplamento do sistema no plano xy. Conforme a escala da Figura 6.10, durante o deslocamento no eixo y, a posição do ROV no eixo x oscilou na ordem de $10^{-2}\ m$. Quando a translação em y parava, o controlador corrigia a posição do veículo. Dessa forma, considerando os efeitos gravitacionais e de flutuação que o ROV sofre, o comportamento do sistema foi adequado.

6.5 Conclusões

Neste Capítulo foi apresentado o simulador da dinâmica do ROV. No caso de simular o sistema de posicionamento dinâmico do veículo, é necessário que esse simulador resolva as equações que modelam a operação do robô. É importante ressaltar que foi utilizado um observador ideal no simulador, ou seja, o valor observado é exato ao valor medido, com exceção das velocidades que foram estimadas utilizando um filtro *lead*.

Em seguida foi exibido a ação do controlador P-PI para cada grau de liberdade. O controlador obedece as especificações para o controle do rumo, da profundidade e da conservação da distância do veículo até a parede.

No caso específico do controle no eixo x, foram feitos testes considerando orientação diferente de zero, isto é, a parte frontal do robô não está na mesma direção da parede. Além disso, foi feito o controle no caso de deslocamentos laterais. Nesse caso, o acoplamento no plano xy demonstrou ser bastante pequeno.

Capítulo 7

Conclusões e Trabalhos Futuros

Este capítulo apresenta, de forma geral, as conclusões obtidas durante o desenvolvimento do projeto. Além disso, também são propostas ideias para aperfeiçoamento e continuidade deste trabalho.

7.1 Conclusões Gerais

O estudo do modelo cinemático e dinâmico do veículo se mostrou essencial para a concepção do problema do posicionamento dinâmico. A partir deste modelo, conclui-se que desacoplando o sistema, ou seja, considerando cada grau de liberdade de maneira independente, um controlador linear pode ser projeto. Dessa forma, sensores são utilizados para medição da posição e da orientação do veículo.

Durante o estudo dos sensores utilizados no ROV Luma, estes se mostraram apropriados como instrumento de medição do estado do veículo para o projeto de um controlador. Contudo, no aspecto do controle de posição no eixo x, foi observado que inicialmente é necessário localizar o robô no ambiente. Desta forma, foi implementado um algoritmo de identificação e localização do ROV Luma a partir dos dados medidos pelo sonar.

Feita a análise do *hardware* do sistema, foi possível planejar e desenvolver o sistema de *software* para o posicionamento dinâmico do ROV Luma. Nesse aspecto, a utilização de uma interface gráfica favorece a operação, permitindo que o operador observe o estado do veículo e o controle de maneira rápida e fácil. Além disso, a utilização de ROS e inclusão dos algoritmos de controle no computador base ou embarcado permite maior versatilidade e poder computacional.

Ao projetar o controlador, foi necessário primeiro analisar os propulsores do ROV Luma. Como foi abordado, estes apresentam não-linearidades que precisam ser compensadas para a operação do veículo. Ainda, o uso do controlador P-PI proporciona correção das perturbações que o ROV sofre durante o movimento. Além disso, apresenta fácil sintonização e simples implementação.

Por fim, foi necessário utilizar um modelo realista que incluísse a dinâmica dos propulsores, não-linearidades, efeitos hidrodinâmicos, do cabo umbilical, de gravidade e flutuação para a validação do sistema desenvolvido. Consequentemente, ao simular a operação do ROV Luma, foi observado que o sistema de posicionamento dinâmico desenvolvido demostrou-se adequado, atendendo especificações exigidas para melhorar a manobrabilidade do ROV.

Portanto, a contribuição principal deste trabalho foi o desenvolvimento deste sistema de posicionamento dinâmico para o ROV Luma, permitindo que o robô apresente operação parcialmente autônoma. O sistema foi implementado em C++ de modo que já é possível realizar testes experimentais.

7.2 Trabalhos Futuros

A primeira etapa para a continuidade do projeto é realizar testes experimentais no ROV Luma para validação do comportamento real do sistema desenvolvido. Além disso, pode-se propor aperfeiçoamentos e novas linhas de estudos. Nesse sentido, pode-se citar:

- Desenvolver o sistema de software para versão mais recente do ROS.
- Realizar a integração do sistema desenvolvido com o ambiente de realidade virtual. Implementação do modelo do ROV Luma no ambiente SimUEP.
- Desenvolvimento de um observador de estados para melhorar a medição dos sensores. Um observador de estados é proposto em Khadhraoui et al. (2015) e em Xia et al. (2013), neste último é implementado uma rede neural no observador.
- Implementar outros métodos de localização do veículo no ambiente. Esses métodos poderiam ser probabilísticos como o Filtro de Kalman Estendido como explicado em Siegwart et al. (2011). Em Frangipani (2015) é proposto um método de localização submarina utilizando uma única referência acústica.
- Implementação de controles mais elegantes como VS-MRAC (Variable Structure Model-Reference Adaptive Control) desenvolvido em Cunha et al. (1995), controle usando lógica fuzzy desenvolvido em Ju et al. (2002).
- Desenvolver algoritmo de SLAM (*Simultanerous Localization and Mapping*).
- Desenvolvimento do controle por servo-visão utilizando a câmera de alta definição do ROV Luma.

- Desenvolver o controle para o grau de liberdade do eixo y. O ROV Luma apresentaria a configuração de um AUV (*Autonomous Underwater Vehicle*). Com isso, será possível desenvolver um sistema de planejamento de trajetórias e tarefas.

Referências Bibliográficas

Aras, M., Shahrieel, M., Ab Azis, F. and Othman, M. N. (2012), A low cost 4 dof remotely operated underwater vehicle integrated with imu and pressure sensor, *in* '4th International Conference on Underwater System Technology: Theory and Applications 2012 (USYS'12)', pp. 18–23.

Arun, K. S., Huang, T. S. and Blostein, S. D. (1987), 'Least-squares fitting of two 3-d point sets', *IEEE Transactions on pattern analysis and machine intelligence* (5), 698–700.

Åström, K. J. and Wittenmark, B. (2013), *Computer-controlled systems: theory and design*, Courier Corporation.

Brun, L. (2012), 'Rov/auv trends: Market and technology', *Marine Technology Reporter* **5**(7), 48–51.

Carneiro, R. F., Leite, A. C., Peixoto, A. J., Goulart, C., Costa, R. R., Lizarralde, F. and Hsu, L. (2006), 'Underwater robot for tunnel inspection: design and control', *Proc 12th Latin-American Congr on Automatic Control, Salvador, Brazil* .

Carvalho, F., Raposo, A., Santos, I. and Galassi, M. (2014), Virtual reality techniques for planning the offshore robotizing, *in* 'Industrial Informatics (INDIN), 2014 12th IEEE International Conference on', IEEE, pp. 353–358.

Cunha, J. (1992), Projeto e estudo de simulação de um sistema de controle a estrutura variável de um veículo submarino de operação remota, Master's thesis, Programa de Engenharia Elétrica, COPPE/UFRJ, Rio de Janeiro.

Cunha, J., Costa, R. R. and Hsu, L. (1995), 'Design of a high performance variable structure position control of rovs', *IEEE Journal of Oceanic Engineering* **20**(1), 42–55.

da Cunha, J. P., Costa, R. R. and Hsu, L. (1991), Input/output variable structure position control of a remotely operated underwater vehicle, *in* 'Advanced

Robotics, 1991.'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on', IEEE, pp. 1305–1310.

De Souza, E. C. and Maruyama, N. (2007), 'Intelligent uuvs: Some issues on rov dynamic positioning', *IEEE Transactions on Aerospace and Electronic Systems* **43**(1).

DeMarco, K., West, M. E. and Collins, T. R. (2011), An implementation of ros on the yellowfin autonomous underwater vehicle (auv), *in* 'OCEANS 2011', IEEE, pp. 1–7.

Dukan, F., Ludvigsen, M. et al. (2011), Dynamic positioning system for a small size rov with experimental results, *in* 'OCEANS, 2011 IEEE-Spain', IEEE, pp. 1–10.

Fischler, M. A. and Bolles, R. C. (1981), 'Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography', *Communications of the ACM* **24**(6), 381–395.

Frangipani, V. (2015), Localização submarina utilizando uma única referência acústica via filtro ukf, Master's thesis, Programa de Engenharia Elétrica, COPPE/UFRJ, Rio de Janeiro.

Goulart, C. (2007), Modelagem, simulação e controle de um veículo submarino de operação remota, Master's thesis, Universidade Federal do Rio de Janeiro.

Hsu, L., Costa, R. R., Lizarralde, F. and da Cunha, J. P. V. S. (2000*a*), 'Avaliação experimental da modelagem e simulação da dinâmica de um veículo submarino de operação remota', *Revista Controle e Automação* **11**(2), 82–93.

Hsu, L., Costa, R. R., Lizarralde, F. and Da Cunha, J. P. V. S. (2000*b*), 'Dynamic positioning of remotely operated underwater vehicles', *IEEE Robotics & Automation Magazine* **7**(3), 21–31.

Jang, G., Kim, S., Kim, J. and Kweon, I. (2005), Metric localization using a single artificial landmark for indoor mobile robots, *in* 'Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on', IEEE, pp. 2857–2862.

Ju, D., Xiaoguang, Z. and Min, T. (2002), Fuzzy logic control in autonomous rov navigation, *in* 'TENCON'02. Proceedings. 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering', Vol. 3, IEEE, pp. 1566–1569.

Khadhraoui, A., Beji, L., Otmane, S. and Abichou, A. (2015), Stabilizing control based observer for a remotely operated vehicle (rov-observer), *in* 'Control, Engineering & Information Technology (CEIT), 2015 3rd International Conference on', IEEE, pp. 1–7.

Lawrance, N. R., Somers, T., Jones, D., McCammon, S. and Hollinger, G. A. (2016), Ocean deployment and testing of a semi-autonomous underwater vehicle, *in* 'OCEANS 2016 MTS/IEEE Monterey', IEEE, pp. 1–6.

Maalouf, D., Creuze, V. and Chemori, A. (2012), State feedback control of an underwater vehicle for wall following, *in* 'Control & Automation (MED), 2012 20th Mediterranean Conference on', IEEE, pp. 542–547.

Marzbanrad, A., Sharafi, J., Eghtesad, M. and Kamali, R. (2011), Design, construction and control of a remotely operated vehicle (rov), *in* 'ASME 2011 International Mechanical Engineering Congress and Exposition', American Society of Mechanical Engineers, pp. 1295–1304.

McFarlane, J. R. (2000), Underwater technology 2000 rovs and auvs: tools for exploring, exploiting and defending the ocean frontier, *in* 'Underwater Technology, 2000. UT 00. Proceedings of the 2000 International Symposium on', IEEE, pp. 465–471.

Murray, R. M., Li, Z., Sastry, S. S. and Sastry, S. S. (1994), *A mathematical introduction to robotic manipulation*, CRC press.

Nguyen, V., Martinelli, A., Tomatis, N. and Siegwart, R. (2005), A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics, *in* 'Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on', IEEE, pp. 1929–1934.

Qingjun, Z., Song, L., Huiting, L., Ming, Z. and Xiaoqiang, D. (2016), Research on dynamic positioning of model-converted rov anti-waves based on micro inertial navigation sensors, *in* 'Sensing Technology (ICST), 2016 10th International Conference on', IEEE, pp. 1–6.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. and Ng, A. Y. (2009), Ros: an open-source robot operating system, *in* 'ICRA workshop on open source software', Vol. 3, Kobe, p. 5.

Ribas, D., Ridao, P. and Neira, J. (2010), *Underwater SLAM for structured environments using an imaging sonar*, Vol. 65, Springer.

Russel, G. (1984), *A design methodology for nonlinear systems*, IEEE, chapter 8, pp. 129–144.

Siciliano, B., Sciavicco, L., Villani, L. and Oriolo, G. (2010), *Robotics: modelling, planning and control*, Springer Science & Business Media.

Siegwart, R., Nourbakhsh, I. R. and Scaramuzza, D. (2011), *Introduction to autonomous mobile robots*, MIT press.

Tipsuwan, Y. and Hoonsuwan, P. (2015), Design and implementation of an auv for petroleum pipeline inspection, *in* 'Information Technology and Electrical Engineering (ICITEE), 2015 7th International Conference on', IEEE, pp. 382–387.

Tritech International Ltd (n.d.), *Tritech Micron Product Manual*. (acessado Fevereiro 6, 2017).
URL: `http://www.tritech.co.uk/media/products/small-rov-mechanical-sector-scanning-sonar-tritech-micron_hardware_manual.pdf`

Xia, G., Pang, C. and Liu, J. (2013), Neural-network-based adaptive observer design for autonomous underwater vehicle in shallow water, *in* 'Natural Computation (ICNC), 2013 Ninth International Conference on', IEEE, pp. 216–221.

Yen, W.-K. and Guo, J. (2016), Wall following control of a robotic fish using dynamic pressure, *in* 'OCEANS 2016-Shanghai', IEEE, pp. 1–7.

Apêndice A

Dispositivos

Neste apêndice são apresentados as características dos dispostivos utilizados no ROV Luma.

A.1 Unidade Inercial

Tabela A.1: Tabela das características da IMU.

Navegação	
Acurácia da posição horizontal	2.0 m
Acurácia da posição vertical	3.0 m
Acurácia da posição horizontal (com DGNSS)	0.6 m
Acurácia da posição vertical (com DGNSS)	1.0 m
Acurácida da velocidade	0.05 m/s
Acurácia *roll* e *pitch* (estática)	0.1 °
Acurácia *yaw* (estático)	0.5 °
Acurácia *roll* e *pitch* (dinâmico)	0.2 °
Acurácia *yaw* (dinâmica com GNSS)	0.2 °
Acurácia *yaw* (apenas magnético)	0.8 °
Alcance da orientação	ilimitado
Taxa de saída	até 1000 Hz
Latência	0.4 ms
Comunicação	
Interface	RS232
Velocidade	4800 até 2M de *baudrate*
Protocolo	AN Packet Protocol or NMEA

Tabela A.2: Tabela das características dos sensores da IMU.

Sensores	Acelerômetros	Giroscópios	Magnetômetros	Pressão
Alcance	2g, 4g, 16g	250 °/s, 500°/s, 2000°/s	2G, 4G, 8G	10 até 120KPa
Instalabilidade do *bias*	20ug	3°/hr	-	10 PA
Bias inicial	< 5mg	< 0.2°/s	-	< 100 PA
Erro de escala inicial	< 0.06 %	< 0.04 %	< 0.07 %	-
Estabilidade do fator de escala	< 0.06 %	< 0.05 %	< 0.09 %	-
Não linearidades	< 0.05 %	< 0.05 %	< 0.08 %	-
Erro de alinhamento entre eixos	< 0.05 °	< 0.05 °	< 0.05 °	-
Densidade do ruído	100 ug/?Hz	0.004 °/s/?Hz	210 uG/?Hz	0.56 Pa/?Hz
Largura de banda	400 Hz	400 Hz	110 Hz	50 Hz

A.2 Sonar

Tabela A.3: Tabela das características do Sonar.

Acústica	
Frequência de operação	CHIRP centrado em 700kHz
Largura de banda	35^o vertical e 3^o horizontal
Alcance máximo	75 m
Alcance mínimo	0.3 m
Resolução do alcance	aproximadamente 7.5 mm
Resolução mecânica	0.45^o, 0.9^o,1.8^o
Setor de exame	Até 360^o
Exame contínuo	Sim
Eletrônica, Comunicação e *Software*	
Potência	12 - 48 V em 4VA (média)
Comprimento máximo do cabo	10000m usando RS485
Protocolo de comunicação	RS485 (par trançado), RS232
Controle de superfície	computador com porta serial padrão ou USB-RS232/RS485 conversor
Software de controle	Tritech Seanet Pro, Micron software ou protocolo de comando baixo-nível
Ferramentas do software	zoom acústico, medição de imagem, operação invertida
Físico	
Peso no ar	324g
Peso na água	180g
Avaliação de profunidade	padrão 750m, 3000m opcional
Alcance da temperatura	-10 até 35^oC (-20 até 50^oC em armazém)

A.2.1 Pacote de ROS

O pacote de ROS utilizado configura e se comunica com o sonar utilizado no ROV Luma.

Instalação

Primeiramente, é necessário clonar o repositório com o seguinte comando:

```
git clone https://github.com/mcgill-robotics/ros-tritech-micron.git
    tritech_micron
```

Dependências

Antes de compilar o pacote, é necessário instalar algumas dependências com os comando

```
rosdep update
rosdep install tritech_micron
```

Compilação

Antes de executar o pacote, é necessário compilá-lo com o comando:

```
catkin_make
```

Execução

Para execução, é necessário conectar o sonar utilizando comunicação serial RS485 e iniciar o pacote com o comando:

```
roslaunch tritech_micron tritech_micron.launch port:=</path/to/sonar>
    frame:=<frame_id>
```

onde *path* e *frame* são argumentos do launch de ROS:

- *port*: porta serial para a leitura, padrão: /dev/sonar.
- *frame*: *frame* para publicar as mensagens, padrão: sonar.

O pacote irá tentar realizar a conexão com o sonar até obter sucesso.

O *launch* do sonar é apresentado abaixo.

```
<launch>
<!-- Read arguments -->
<arg name="port" default="/dev/sonar"/>
<arg name="frame" default="sonar"/>
```

```
<node name="tritech_micron"
pkg="tritech_micron"
type="scan.py"
output="screen"
respawn="true">
<!-- Set ROS parameters -->
<param name="port" value="$(arg port)"/>
<param name="frame" value="$(arg frame)"/>
</node>
</launch>
```

Ao executar o arquivo, o *launch* inicia um nó chamado `tritech_micron` que executa o script denominado `scan.py`.

A.3 Profundímetro

Tabela A.4: Tabela das características do Profundímetro.

Especificação	Valores
Faixa de medição	0.8...1.2BAR — 0...1BAR até 0...1000BAR
Alimentação	8...28 Vcc
Acurácia	$0,1\%FS$
Taxa de saída	400 Hz
Resolução Digital	$0.002\%FS$
Faixa de Temperatura	$-40...120°C$
Sinal de saída	RS485 — 4...20mA — 0...10VCC
Resistência a Pressão	10^7 ciclos de pressão 0...100%FS a $25°C$
Resistência a Vibração	$20g$ (5...2000 Hz, com amplitude máxima de ± 3 mm)
Isolamento	$100M\Omega/50V$

Apêndice B

Configurações da Interface

Neste apêndice são apresentados os arquivos de configuração da interface do ROV Luma. Também são ilustradas as outros Tools que compõem o *software*

B.1 Instalação

O projeto usa GIT como instrumento para controle de versão. Assim, os repositórios de GIT foram criados no servidor do laboratório LEAD da COPPE/UFRJ. O servidor pode ser acessado dentro ou fora do laboratório. É necessário ter um *username/password* registrado no servidor para poder acessá-lo. O procedimento a seguir demonstra como instalar os repositórios da interface *RobotGUI*.

- Primeiramente é necessário clonar os repositórios do RobotPS e do LUMA dentro da pasta de trabalho do ROS.

```
cd ~/fuerte_workspace/sandbox
git clone -b fuerte-devel ssh://username@146.164.26.53:4000/opt/ros/robot_ps.
    git
git clone ssh://username@146.164.26.53:4000/opt/luma/software/luma.git
```

- Feito isso, é necessário realizar a compilação dos pacotes.

```
roscd RobotPS
rosmake
roscd LUMA
rosmake
```

B.2 Execução

Para executar a interface, basta rodar o comando:

```
roscd LUMA/bin
./RobotGUI
```

B.3 Configuração Inicial

Ao iniciar o *software RobotGUI*, este lê o arquivo *initialconfiguration.xml* para configurar os parâmetros iniciais. O arquivo define valores iniciais para variáveis de alguns *Components*. Também contém os *timeouts* de cada mensagem do LPCR485Server, ou seja, o tempo de espera necessário para esperar a resposta da mensagem. Para o controle, neste arquivo são salvos os parâmetros dos controlador.

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
<LPCR485Server>
<Timeouts offset="0" offset_unit="m">
<DepthGauge value="15" unit="m" />
<DepthRead value="300" unit="m" />
<LampIntensity value="25" unit="m" />
<Laser value="15" unit="m" />
<Propulsor value="20" unit="m" />
<PropulsorDriver value="20" unit="m" />
<Compensator value="90" unit="m" />
<Decoupler value="70" unit="m" />
<YawControlParameters value="80" unit="m" />
<DepthControlParameters value="80" unit="m" />
<VelocityRead value="30" unit="m" />
<CameraSwitch value="20" unit="m" />
<CameraController value="20" unit="m" />
<CameraOnOff value="20" unit="m" />
<IMUread value="70" unit="m" />
<HDCameraPhoto value="5" unit="m" />
<HDCameraUSBMode value="5" unit="m" />
<AltimeterRead value="100" unit="m" />
<TemperatureRead value="20" unit="m" />
<HDCameraZoom value="1" unit="u" />
<HDCameraZoomSpeed value="1" unit="u" />
<HDCameraFocus value="1" unit="u" />
<HDCamera value="1" unit="u" />
<CompensatorRead value="90" unit="m" />
<DecouplerRead value="90" unit="m" />
<YawControlParametersRead value="60" unit="m" />
<DepthControlParametersRead value="60" unit="m" />
</Timeouts>
</LPCR485Server>
<LPCRDepthGauge PowerOn="true" />
<LPCRLamp>
<lamp_1 intensity="0" />
<lamp_2 intensity="0" />
<lamp_3 intensity="0" />
<lamp_4 intensity="0" />
```

```
</LPCRLamp>
<LPCRLaser PowerOn="false" />
<LPCRPropulsor PowerOn="false" DriverOn="true" />
<LPCRMotionController>
<ControlOptions zvelocityposition="false" yawvelocityposition="false"
    zforceposition="true" yawforceposition="true" />
<Compensator>
<Propulsor1 config_static_compensator="0" config_dead_zone_compensator
    ="0" static_gain="1.700000" dead_zone="1.000000" delta_dead_zone
    ="0.500000" saturation="100" />
<Propulsor2 config_static_compensator="0" config_dead_zone_compensator
    ="0" static_gain="1.700000" dead_zone="1.000000" delta_dead_zone
    ="0.500000" saturation="100" />
<Propulsor3 config_static_compensator="0" config_dead_zone_compensator
    ="0" static_gain="1.700000" dead_zone="1.000000" delta_dead_zone
    ="0.500000" saturation="100" />
<Propulsor4 config_static_compensator="0" config_dead_zone_compensator
    ="0" static_gain="1.700000" dead_zone="1.000000" delta_dead_zone
    ="0.500000" saturation="100" />
<Propulsor5 config_static_compensator="0" config_dead_zone_compensator
    ="0" static_gain="1.700000" dead_zone="1.000000" delta_dead_zone
    ="0.500000" saturation="100" />
</Compensator>
<Decoupler>
<Row1 first_element="0.315094" second_element="0.455000" third_element
    ="0.647200" />
<Row2 first_element="0.315100" second_element="-0.455000" third_element
    ="-0.647200" />
<Row3 first_element="0.315100" second_element="-0.366400" third_element
    ="0.647200" />
<Row4 first_element="0.315100" second_element="0.366400" third_element
    ="-0.647200" />
</Decoupler>
<YawControl yaw_controltype="0" yaw_kd="1.163800" yaw_kp="0.6667000"
    yaw_ki="0.87280" yaw_delta="0.174533" yaw_kbackward="0.333300"
    yaw_tau="0.05000" yaw_h="0.100000" yaw_vmax="1.000000" yaw_umax
    ="1.000000" yaw_i="0.000000" />
<DepthControl depth_controltype="0" depth_kd="2.27070" depth_kp
    ="0.266700" depth_ki="0.68120" depth_delta="0.100000"
    depth_kbackward="0.333300" depth_tau="0.05000" depth_h="0.100000"
    depth_vmax="1.000000" depth_umax="1.000000" depth_i="0.000000" />
<XWallControl xwall_controltype="1" xwall_kd="1.6965" xwall_kp
    ="0.266700" xwall_ki="0.50900" xwall_delta="0.100000"
    xwall_kbackward="0.33330" xwall_tau="0.050" xwall_h="0.100"
    xwall_vmax="1.000000" xwall_umax="1.000000" xwall_i="0.000000" />
</LPCRMotionController>
<GPCRIMU baudrate="115200" period="100" PID="0403" VID="6001"
```

```
    manufacturer="FTDI" dproduct="FT232R USB UART" serial="AH01QISV" />
<LPCRCameraSwitch SelectedCamera="1" />
<LPCRHDCamera zoom_auto="true" focus_auto="true" resolution="false" />
</root>
```

B.4 Configuração do *Joystick*

O arquivo *lumadefaultsequences.xml* apresenta a configuração padrão do *joystick* para os modos de operação remota do ROV Luma. É possível editar de modo *offline*. Ao modificar os comandos na interface, os novos comandos são salvos nesse arquivo.

```
<?xml version="1.0" encoding="UTF-8" ?>
<RobotPS>
<Group Name="Group Motion">
<Mode Name="Standard Motion" GamepadSequence="BACK/P" KeyboardSequence="SHIFT/P"
    ToogleMode="Yes" ToggleMode="yes">
<Value ID="0" Input="GamepadAxis" Value="LEFT_STICK_UD" Sign="Negative" />
<Value ID="1" Input="GamepadAxis" Sign="Negative" Value="LEFT_STICK_LR" />
<Value ID="2" Input="KeyboardButton" Value="UP" />
<Value ID="3" Input="KeyboardButton" Sign="Negative" Value="DOWN" />
<Value ID="4" Input="KeyboardButton" Value="RIGHT" />
<Value ID="5" Input="KeyboardButton" Sign="Negative" Value="LEFT" />
</Mode>
<Mode Name="Individual Motion" GamepadSequence="START/P" KeyboardSequence="SPACE/P"
     ToogleMode="Yes" ToggleMode="yes" />
</Group>
<Group Name="Individual Motion">
<Mode Name="X Force" GamepadSequence="X/P" KeyboardSequence="X/P" ToogleMode="Yes"
    ToggleMode="yes">
<Value ID="0" Input="KeyboardButton" Value="ESCAPE" />
<Value ID="1" Input="GamepadAxis" Value="LEFT_STICK_LR" Sign="Positive" />
<!--Value ID="2" Input="KeyboardButton" Sign="Negative" Value="BACKSPACE" /-->
<Value ID="2" Input="GamepadAxis" Sign="Negative" Value="LEFT_STICK_LR" />
</Mode>
<Mode Name="Y Force" GamepadSequence="Y/P" KeyboardSequence="C/P" ToogleMode="Yes"
    ToggleMode="yes">
<Value ID="0" Input="GamepadAxis" Value="LEFT_STICK_UD" />
<Value ID="1" Input="KeyboardButton" Value="ESCAPE" />
<!--Value ID="2" Input="GamepadAxis" Sign="Negative" Value="RT_AXIS" /-->
<!--Value ID="3" Input="GamepadAxis" Value="LT_AXIS" />
<Value ID="4" Input="GamepadAxis" Value="RT_AXIS" /-->
<Value ID="2" Input="None" Value="" />
</Mode>
<Mode Name="Z Position" GamepadSequence="LB/P" KeyboardSequence="J/P" ToogleMode="
    Yes" ToggleMode="yes">
<Value ID="0" Input="GamepadAxis" Value="LT_AXIS" />
<Value ID="1" Input="None" Value="" />
<Value ID="2" Input="None" Sign="Negative" Value="" />
</Mode>
<Mode Name="Yaw Position" GamepadSequence="RB/P" KeyboardSequence="H/P" ToogleMode
    ="Yes" ToggleMode="yes">
<Value ID="0" Input="GamepadAxis" Value="RT_AXIS" Sign="Negative" />
<Value ID="1" Input="None" Value="" />
<Value ID="2" Input="None" Sign="Negative" Value="" />
</Mode>
```

```
<Mode Name="Z Force" GamepadSequence="B/P" KeyboardSequence="O/P" ToogleMode="Yes"
    ToggleMode="yes">
<Value ID="0" Input="GamepadAxis" Value="LEFT_STICK_UD" />
<Value ID="1" Input="None" Value="" />
<Value ID="2" Input="None" Sign="Negative" Value="" />
</Mode>
<Mode Name="Yaw Force" GamepadSequence="A/P" KeyboardSequence="I/P" ToogleMode="Yes
    " ToggleMode="yes">
<Value ID="0" Input="GamepadAxis" Value="LEFT_STICK_LR" />
<Value ID="1" Input="None" Value="" />
<Value ID="2" Input="None" Sign="Negative" Value="" />
</Mode>
</Group>
<Group Name="Motion Z">
<Mode Name="Position" GamepadSequence="" KeyboardSequence="" ToogleMode="Yes"
    ToggleMode="yes">
<Value ID="0" Input="None" Value="" />
<Value ID="1" Input="None" Sign="Negative" Value="" />
<Value ID="2" Input="None" Value="" />
<Value ID="3" Input="None" Sign="Negative" Value="" />
</Mode>
<Mode Name="Force" GamepadSequence="LB/P" KeyboardSequence="W/P" ToogleMode="Yes"
    ToggleMode="yes">
<Value ID="0" Input="GamepadAxis" Value="RIGHT_STICK_UD" Sign="Negative" />
<Value ID="1" Input="None" Sign="Negative" Value="" />
<Value ID="2" Input="None" Value="" />
<Value ID="3" Input="None" Sign="Negative" Value="" />
</Mode>
</Group>
<Group Name="Motion Yaw">
<Mode Name="Position" GamepadSequence="" KeyboardSequence="" ToogleMode="Yes"
    ToggleMode="yes">
<Value ID="0" Input="None" Value="" />
<Value ID="1" Input="None" Value="" />
<Value ID="2" Input="None" Sign="Negative" Value="" />
</Mode>
<Mode Name="Force" GamepadSequence="RB/P" KeyboardSequence="Q/P" ToogleMode="Yes"
    ToggleMode="yes">
<Value ID="0" Input="GamepadAxis" Value="RIGHT_STICK_LR" Sign="Negative" />
<Value ID="1" Input="None" Value="" />
<Value ID="2" Input="None" Sign="Negative" Value="" />
</Mode>
</Group>
<Group Name="Visualization">
<Modes />
</Group>
</RobotPS>
```

B.5 Lista de *Components*

Cada *Component*, carregado no *RobotGUI* quando uma instância do Luma Package é iniciada, é definido por um arquivo *xml* denominado *lumapackagecomponents.xml*. Quando o veículo está em operação utilizando um computador embarcado, no computador base só são listados os *Base Components*, enquanto no com-

putador embarcado são definidos os *Robots Components*.

```
<?xml version="1.0" encoding="UTF-8"?>
<RobotPS>
<Component name="Base Depth Gauge"></Component>
<Component name="Base Altimeter"></Component>
<Component name="Base IMU"></Component>
<Component name="Base Lamp 1"></Component>
<Component name="Base Lamp 2"></Component>
<Component name="Base Lamp 3"></Component>
<Component name="Base Lamp 4"></Component>
<Component name="Base Laser"></Component>
<Component name="Base Battery"></Component>
<Component name="Base Propulsor 1"></Component>
<Component name="Base Propulsor 2"></Component>
<Component name="Base Propulsor 3"></Component>
<Component name="Base Propulsor 4"></Component>
<Component name="Base Propulsor 5"></Component>
<Component name="Base Propulsors"></Component>
<Component name="Base Camera Switch"></Component>
<Component name="Base Camera Controller"></Component>
<Component name="Base Supervisor"></Component>
<Component name="Controller Signals"></Component>
<Component name="Base Motion Controller"></Component>
<Component name="Base Gamepad And Keyboard Mapper"></Component>
<Component name="Base Black White Camera"></Component>
<Component name="Base Color Camera"></Component>
<Component name="Base HDCamera"></Component>
<Component name="Base Pan Tilt Camera"></Component>
<Component name="Base Micro Camera 1"></Component>
<Component name="Base Micro Camera 2"></Component>
<Component name="Base Camera Extra 1"></Component>
<Component name="Base Temperature Sensor"></Component>
<!--Component name="Base Micro Camera 2"></Component-->
<Component name="Robot 485 Server"></Component>
<Component name="Robot Depth Gauge"></Component>
<Component name="Robot IMU"></Component>
<Component name="Robot Lamp 1"></Component>
<Component name="Robot Lamp 2"></Component>
<Component name="Robot Lamp 3"></Component>
<Component name="Robot Lamp 4"></Component>
<Component name="Robot Laser"></Component>
<Component name="Robot Battery"></Component>
<Component name="Robot Propulsor 1"></Component>
<Component name="Robot Propulsor 2"></Component>
<Component name="Robot Propulsor 3"></Component>
<Component name="Robot Propulsor 4"></Component>
<Component name="Robot Propulsor 5"></Component>
```

```
<Component name="Robot Propulsors"></Component>
<Component name="Robot Motion Controller"></Component>
<Component name="Robot CameraOnOff"></Component>
<!--Component name="Robot HDCamera"></Component-->
<Component name="Robot Camera Switch"></Component>
<Component name="Robot Camera Controller"></Component>
<!--Component name="Robot Altimeter"></Component-->
<Component name="Robot Temperature Sensor"></Component>
<Component name="Robot Supervisor"></Component>
</RobotPS>
```

B.6 Tools

B.6.1 GPTDataTable

A Figura B.1 exibe a GPTDataTable Tool. As colunas representam todos os *Components* relativos aos dispositivos do robô, enquanto as linhas correspondem as variáveis de cada *Component*. Dessa forma, tem o intuito de apresentar os valores das variáveis dos dispositivos, como, por exemplo, intensidade das luminárias, orientação da IMU.

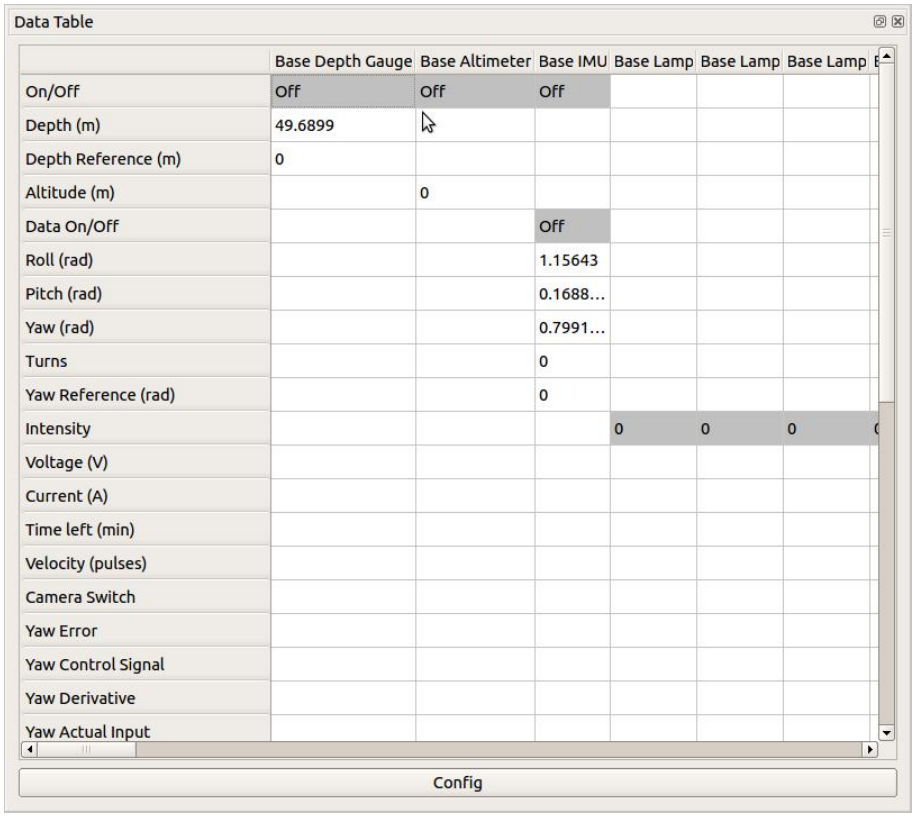


Figura B.1: Tool do DataTable

B.6.2 GPTDeviceManagement

Essa Tool é responsável por gerenciar cada dispositivo. Primeiramente, é escolhido uma instância do robô. Em seguida é escolhido um dispositivo (profundímetro, altímetro e outros) ou um grupo de dispositivos (lâmpadas, propulsores). Ao escolher o dispositivo desejado, aparece as variáveis correspondentes e, com isso, o operador pode ligar ou desligar, mudar valor da variável. Na Figura B.2 tem-se o exemplo das luminárias onde é possível definir um valor para intensidade.

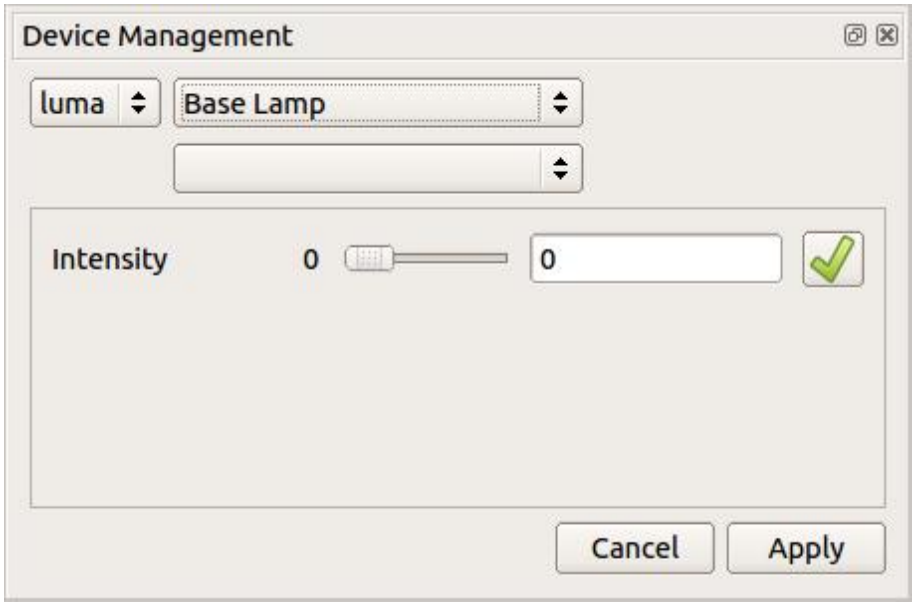


Figura B.2: Tool do DeviceManagement

B.6.3 GPTDeviceOnOffManagement

A Figura B.3 exibe a GPTDeviceOnOffManagement Tool. Nela são listados todos os dispositivos do robô, como profundímetro, altímetro, laser e outros. Cada botão liga ou desliga o dispositivo correspondente. Foi definido que o botão *play* liga o dispositivo e o *stop* o desliga.

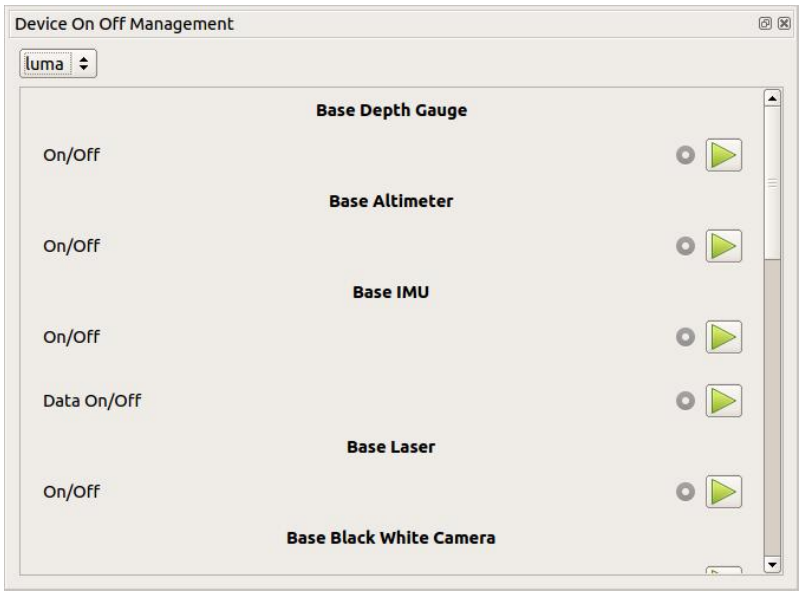


Figura B.3: Tool do DeviceOnOffManagement

B.6.4 GPTGamepadAndKeyboardViewer

Apresenta os modos de controle utilizado o *joystick* ou o teclado. Quando um modo de controle é selecionado, é também marcado em verde na Tool.

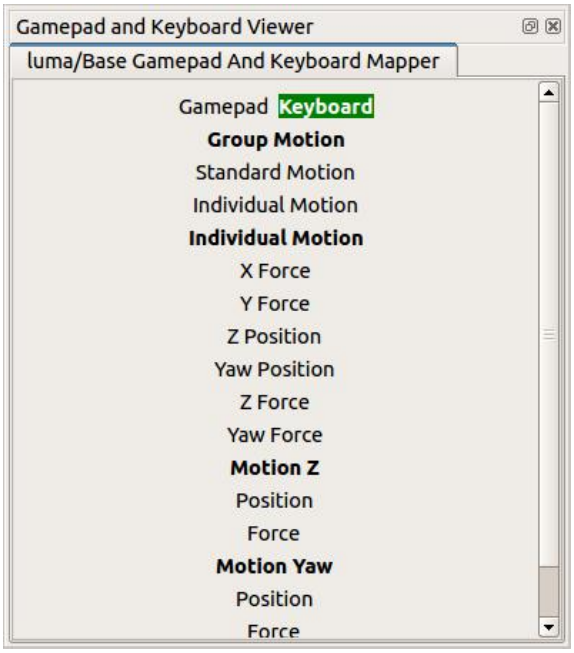


Figura B.4: Tool do GamepadAndKeyboardViewer

B.6.5 GPTVideoViewer

As Figuras B.5 e B.6 mostram a Tool de vídeo e a janela de configuração respectivamente. Ao clicar duas vezes na janela principal, abre-se a de configuração.

Esta última é divida em duas seções. A primeira apresenta um número de linhas e colunas que o operador deseja para o *grid* das câmeras. O botão Apply aplica todas as mudanças para a janela principal. A outra seção lista todas as câmeras que pertencem a uma instância do robô. Ao escolher um elemento do *grid* e depois em um elemento da lista, o usuário seleciona um vídeo para ser alocado no *grid*.

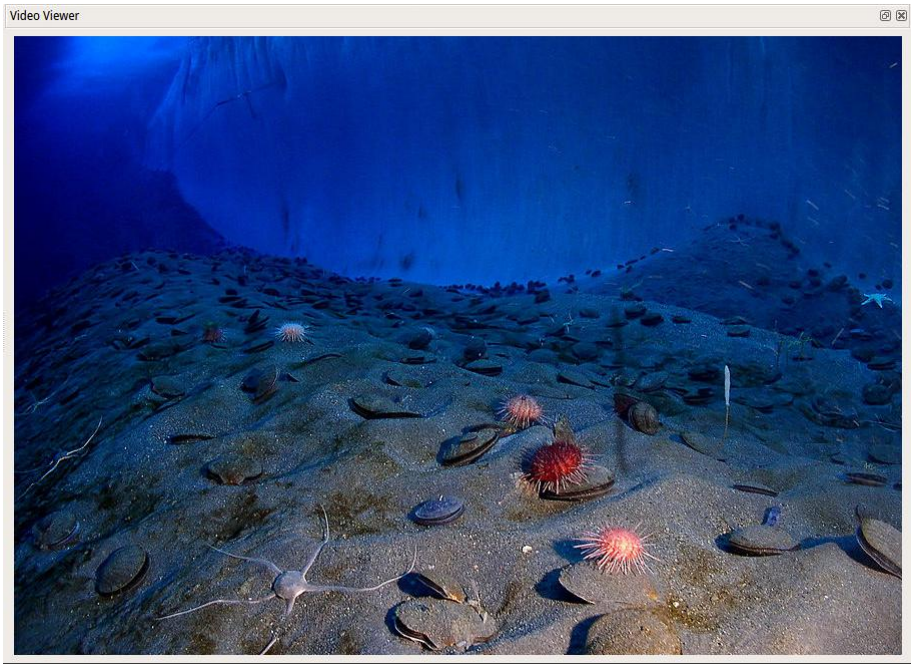


Figura B.5: Tool do VideoViewer

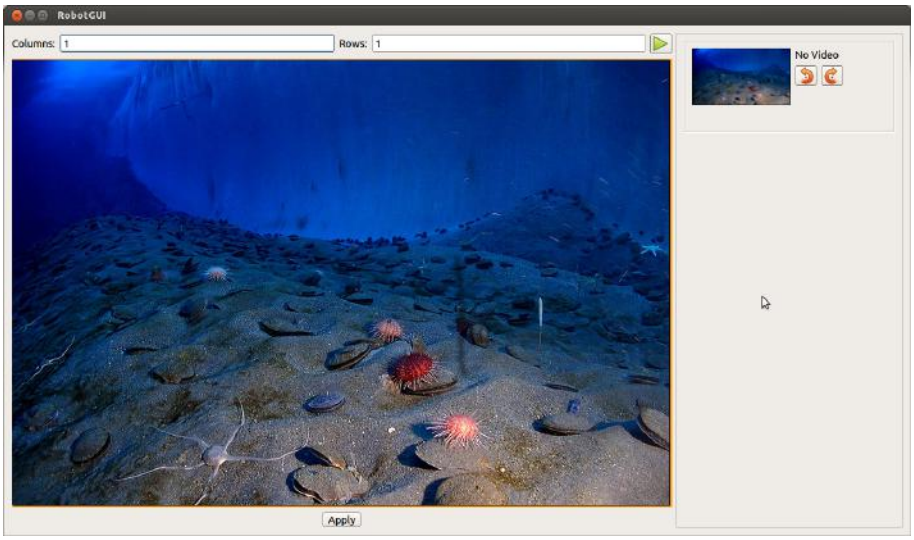


Figura B.6: Tool do VideoViewer

B.6.6 LPTLamp

Esta tool representa as luminárias do ROV Luma. É possível escolher três opções ao clicar nas caixas de seleção: comandar as lâmpadas da esquerda, todas as

lâmpadas ou somente as da direita. Se não for selecionado nenhuma dessa opção, cada luminária é comandada individualmente. Os quatro *sliders* correspondem a intensidade de iluminação desejada. A cor vermelha em cada *slider* significa que a lâmpada correspondente está desligada, enquanto a cor verde representa que está ligada.

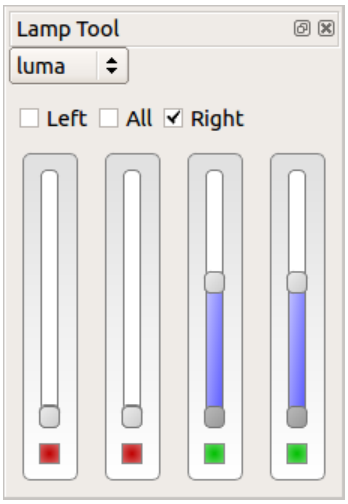


Figura B.7: Tool da lâmpadas

B.6.7 LPTAltimeter

Tool correspondente a leitura dos dados do altímetro. O computador base recebe o valor de leitura e o apresenta na Tool.

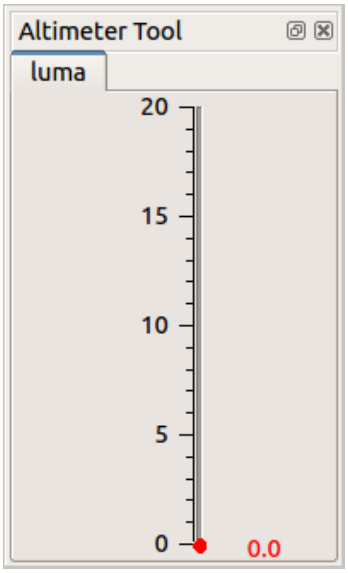


Figura B.8: Tool do Altímetro

B.6.8 LPTDepthAlt

A Figura B.9 ilustra a Tool correspondente aos dados do sensor de pressão e do altímetro. Conforme a legenda, os dados do altímetro são representados pela cor azul, o verde representa os dados do profundímetro e a referência é apresentada em vermelho

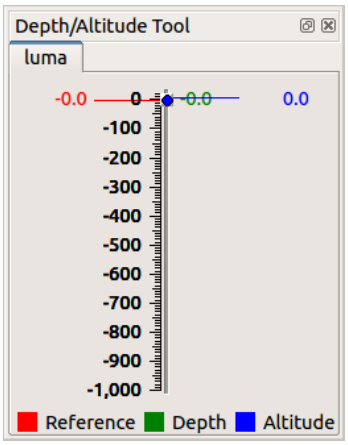


Figura B.9: Tool do profundímetro e altímetro

B.6.9 LPTCameraSwitch

A Figura B.10 exibe a Tool responsável pela seleção das câmeras. Lista todas as câmeras disponíveis na Luma. Os botões selecionam a câmera desejada, enquanto as caixas de seleção são marcadas quando a seleção for feita corretamente.

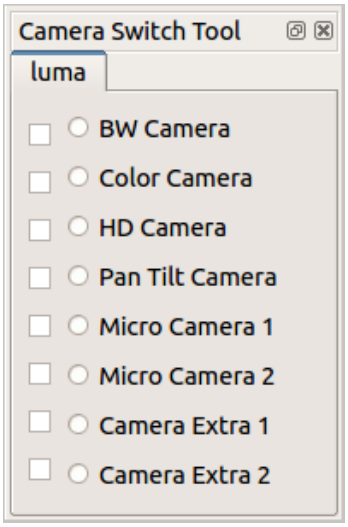


Figura B.10: Tool do *switch* das câmeras

B.6.10 LPTCameraController

Tool responsável pelo controle da câmera *Pan&Tilt*. Ao pressionar o botão, câmera moverá para o lado desejado até o operador soltar o botão.

Figura B.11: Tool do controle da câmera *Pan&Tilt*

B.6.11 LPTHDCamera

A implementação das mensagens seriais da câmera HD ainda não foram feitas. Contudo, a Tool das suas configurações jó foi idealizada. Nela será possível definir a qualidade da imagem, aumentar ou diminuir o zoom e outras configurações. Também será possível ativar o modo usb para transferência de arquivos.

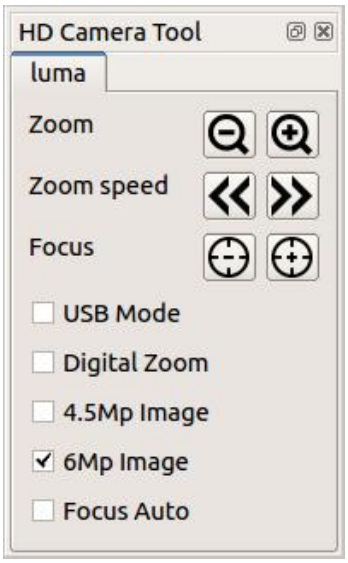


Figura B.12: Tool das configurações da câmera HD

B.7 Mensagens Seriais

O protocolo de comunicação da Luma é por mensagens seriais padrão RS485.O computador base envia essas mensagens para o PC104 embarcado e este devolve

uma mensagem de resposta com o comando original somando 0x80 e um byte de status. Por exemplo, a mensagem para ligar o laser é 80 01 55 81 e a resposta é dada por 80 02 D5 00 81, como demonstrado na figura B.13.

Tabela B.1: Tabela de mensagens seriais da Luma

nome	comando	descrição	resposta
SERIAL_MESSAGE_SOT	0x80	começo de transmissão	
SERIAL_MESSAGE_EOT	0x81	fim de transmissão	
SERIAL_MESSAGE_STATUS_OK	0x00	mensagem de status	
S_M_C_DEPTH_GAUGE_ON	0x47	ligar profundímetro	0xC7
S_M_C_DEPTH_GAUGE_OFF	0x48	desligar profundímetro	0xC8
$S\_M\_C\_DEPTH_READ$	0x36	leitura do profundímetro	0xB6
S_M_C_LAMP_INTENSITY_WRITE	0x53	definir intensidade das luminárias	0xD3
S_M_C_LASER_ON	0x55	ligar laser	0xD5
S_M_C_LASER_OFF	0x56	desligar laser	0xD6
S_M_C_PROPULSOR_VELOCITY_READ	0x1B	leitura velocidade dos propulsores	0x9B
S_M_C_PROPULSORS_ON	0x04	ligar propulsores	0x84
S_M_C_PROPULSORS_OFF	0x0E	desligar propulsores	0x8E
S_M_C_PROPULSORS_DRIVER_ON	0x03	ligar driver dos propulsores	0x83
S_M_C_PROPULSORS_DRIVER_OFF	0x1D	desligar driver dos propulsores	0x9D
S_M_C_MOTION_CONTROLLER_CONTROL	0x19	ligar controle	0x99
S_M_C_MOTION_CONTROLLER_YAW_ON	0x32	ligar controle de rumo	0xB2
S_M_C_MOTION_CONTROLLER_YAW_OFF	0x33	desligar controle de rumo	0xB3
S_M_C_MOTION_CONTROLLER_YAW_REF_WRITE	0x3C	escrever referência do controle de rumo	0xBC
S_M_C_MOTION_CONTROLLER_DEPTH_ON	0x37	ligar controle de profunidade	0xB7
S_M_C_MOTION_CONTROLLER_DEPTH_OFF	0x38	desligar controle de profundidade	0xB8
S_M_C_MOTION_CONTROLLER_DEPTH_REF_WRITE	0x3D	escrever referência	0xBD
S_M_C_MOTION_CONTROLLER_PARAM_COMPENSATOR_WRITE	0x26	escrever parâmetros dos compensadores	0xA6
S_M_C_MOTION_CONTROLLER_PARAM_DECOUPLER_WRITE	0x28	escrever matriz de desacoplamento	0xA8
S_M_C_MOTION_CONTROLLER_PARAM_YAW_CONTROL_WRITE	0x30	escrever parâmetros do controle de rumo	0xB0
S_M_C_MOTION_CONTROLLER_PARAM_DEPTH_CONTROL_WRITE	0x3A	escrever parâmetros	0xBA
S_M_C_MOTION_CONTROLLER_YAW_REF_READ	0x3E	leitura da referência do rumo	0xBE
S_M_C_MOTION_CONTROLLER_DEPTH_REF_READ	0x3F	leitura da referência da profundidade	0xBF
S_M_C_MOTION_CONTROLLER_PARAM_COMPENSATOR_READ	0x27	leitura dos compensadores	0xA7
S_M_C_MOTION_CONTROLLER_PARAM_DECOUPLER_READ	0x29	leitura da matriz de desacoplamento	0xA9
S_M_C_MOTION_CONTROLLER_PARAM_YAW_CONTROL_READ	0x31	leitura de parâmetros	0xB1
S_M_C_MOTION_CONTROLLER_PARAM_DEPTH_CONTROL_READ	0x3B	leitura de parâmetros	0xBB
S_M_C_SWITCH_CAMERA	0x0A	switch das câmeras	0x8A
S_M_C_TILT_UP_CAMERA	0x23	movimento de tilt pra cima	0xA3
S_M_C_TILT_DOWN_CAMERA	0x24	movimento de tilt pra baixo	0xA4
S_M_C_PAN_RIGHT_CAMERA	0x21	movimento pra direita	0xA1
S_M_C_PAN_LEFT_CAMERA	0x22	movimento pra esquerda	0xA2
S_M_C_STOP_CAMERA	0x25	parar câmera	0xA5
S_M_C_POWER_CAM	0x07	ligar câmeras	0x87
S_M_C_IMU_READ	0x2A	leitura da imu	0xAA
S_M_C_IMU_ON	0x05	ligar imu	0x85
S_M_C_IMU_OFF	0x0F	desligar imu	0x8F
S_M_C_IMU_DATA_ON	0x06	ligar dados da imu	0x86
S_M_C_IMU_DATA_OFF	0x10	desligar dados da imu	0x90
S_M_C_ALTIMETER_READ	0x2B	leitura do altímetro	0xAB
S_M_C_TEMP_TUBE_1	0x5B	sensor de temperatura do tubo 1	0xDB
S_M_C_TEMP_TUBE_2	0x5C	sensor de temperatura do tubo 2	0xDC

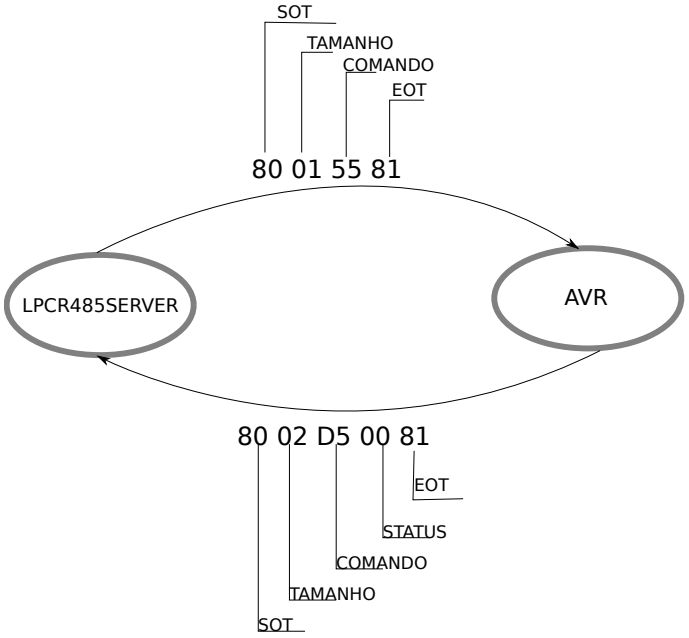


Figura B.13: Exemplo de troca de mensagem serial.