Universidade Federal de Ouro Preto
Escola de Minas
CECAU - Colegiado do Curso de
Engenharia de Controle e Automação

Caio Camilo Alves Valadares

Posicionamento Dinâmico em Embarcações Marítimas: Fundamentação Teórica e Estudo de um Simulador

Monografia de Graduação

Ouro Preto, 2025

Caio Camilo Alves Valadares

Posicionamento Dinâmico em Embarcações Marítimas: Fundamentação Teórica e Estudo de um Simulador

Trabalho apresentado ao Colegiado do Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheira(o) de Controle e Automação;

Orientadora: Dra. Adrielle de Carvalho Santana.

Ouro Preto
2025

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA CONTROLE E AUTOMACAO

FOLHA DE APROVAÇÃO

Caio Camilo Alves Valadares

Posicionamento Dinâmico em Embarcações Marítimas: Fundamentação Teórica e Estudo de um Simulador

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Engenharia de Controle e Automação

Aprovada em 04 de setembro de 2025

Membros da banca

Dra. Adrielle de Carvalho Santana - Orientadora - DECAT - Universidade Federal de Ouro Preto
Dr. Agnaldo José da Rocha Reis - Convidado - DECAT - Universidade Federal de Ouro Preto
MSc. João Carlos Vilela de Castro - Convidado - DECAT - Universidade Federal de Ouro Preto

Adrielle de Carvalho Santana, orientadora do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 08/09/2025

Documento assinado eletronicamente por **Adrielle de Carvalho Santana**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 08/09/2025, às 08:38, conforme horário oficial de Brasília, com fundamento no art. 6°, § 1°, do Decreto n° 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0972778** e o código CRC **F606F0E6**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591533 - www.ufop.br

Dedico este trabalho à minha noiva, cujo incentivo firme — e, por vezes, quase ameaçador (te amo) — foi essencial; à minha família, que me deu a base da minha formação pessoal e acadêmica, tornando tudo possível ao me moldar e ensinar a sempre buscar conhecimento e oportunidades; e aos amigos que a UFOP me presenteou.

Agradecimentos

Meus profundos agradecimentos aos professores Fernando Cortez Sica, Paulo Monteiro, Alan Kardek, Agnaldo Reis, Bruno Baroni, João Carlos Viela, Cocota, e especialmente à minha orientadora Adrielle de Carvalho, pelo valioso conhecimento compartilhado ao longo da minha graduação na grandiosa e histórica Universidade Federal de Ouro Preto.

"The Universe is under no obligation to make sense to you." (Neil deGrasse Tyson)

Resumo

Este trabalho aborda os sistemas de posicionamento dinâmico aplicados a embarcações marítimas, uma tecnologia que, embora existente há várias décadas, mantém elevada relevância e apresenta contínuo crescimento em virtude da necessidade humana de explorar cada vez mais o ambiente marinho. Esses sistemas permitem que grandes navios sejam orientados e posicionados geograficamente de forma precisa, atendendo a inúmeras finalidades. O presente trabalho de conclusão de curso desenvolve a fundamentação teórica por meio de uma revisão bibliográfica baseada em autores consagrados e centros de pesquisa especializados em sistemas de orientação e posicionamento. Adicionalmente, é realizado um detalhamento do simulador *PythonVehicleSimulator*, acompanhado de uma análise comparativa em relação ao referencial teórico. Por fim, são apresentadas propostas de trabalhos futuros que visam aprimorar o funcionamento do simulador, além de possibilitar sua utilização como plataforma para o desenvolvimento e teste de sensores reais em sistemas de posicionamento dinâmico.

Palavras-chaves: python. posicionamento dinâmico. simulação. MIMO. controle de embarcações. alocação de empuxo. modelagem matemática. observador de estados. filtro de ondas.

Abstract

This work addresses dynamic positioning systems applied to maritime vessels, a technology that, although existing for several decades, remains highly relevant and continues to grow due to the human need to increasingly explore the marine environment. These systems allow large ships to be guided and geographically positioned with high precision, serving a wide range of purposes. This undergraduate thesis develops the theoretical foundation through a literature review based on established authors and research centers specialized in navigation and positioning systems. Additionally, a detailed description of the *PythonVehicleSimulator* is provided, accompanied by a comparative analysis with the theoretical framework. Finally, proposals for future work are presented, aiming to enhance the simulator's performance and to enable its use as a platform for the development and testing of real sensors in dynamic positioning systems.

Key-words: python. dynamic positioning. simulation. MIMO. vessel control. thrust allocation. mathematical modeling. state observer. wave filter.

Lista de ilustrações

Figura 1 – Diagrama representando os graus de liberdade sob uma embarcação . . 17
Figura 2 – Diagrama representando os referenciais usados para descrever os movimentos da embarcação 19
Figura 3 – Diagrama em blocos exemplificando o sistema *feedforward* 22
Figura 4 – Frequências associadas ao movimento de uma embarcação 23
Figura 5 – Funcionamento do filtro de Kalman 27
Figura 6 – Tipos de propulsores 28
Figura 7 – Esquema forças e distribuição dos atuadores no casco 30
Figura 8 – Influência dos propulsores nos movimentos 31
Figura 9 – Ilustração das múltiplas soluções de alocação de empuxo com a mesma força resultante 32
Figura 10 – Comportamento da embarcação ao longo da simulação 47
Figura 11 – Trajetória da embarcação nos três eixos do sistema de referência NED (*North-East-Down*). 47
Figura 12 – Atuação dos propulsores ao longo da simulação 48
Figura 13 – Comportamento da embarcação, após ajustes 48
Figura 14 – Atuação dos propulsores ao longo da simulação, após ajustes 49

Lista de tabelas

Tabela 1 – Tabela de Graus de Liberdade (DOF) e suas Correspondências 18

Lista de abreviaturas e siglas

DECAT	Departamento de Engenharia de Controle e Automação
UFOP	Universidade Federal de Ouro Preto
NTNU	Norwegian University of Science and Technology (Universidade Norueguesa de Ciência e Tecnologia)
GPS	Global Positioning System (Sistema de Posicionamento Global)
DP	Dynamic Positioning (Posicionamento Dinâmico)
DOF	Degrees of Freedom (Graus de Liberdade)
USV	Unmanned Surface Vehicle (Veículo de Superfície Não Tripulado)
WF	Wave Filtering (Filtro de Ondas)
LF	Low Frequency (Baixa Frequência)
HF	High Frequency (Alta Frequência)
RPM	Rotações por Minuto
PID	Proporcional, Integral e Derivativo
NED	North-East-Down (Norte, Leste e eixo vertical para baixo)
MIMO	Multiples Inputs And Multiples Outputs (Multiplas Entradas e Multiplas Saídas)

Lista de símbolos

ν	Vetor de velocidades no referencial fixo ao corpo: $\nu = [u, v, w, p, q, r]^T$
$\dot{\nu}$	Derivada temporal de ν (acelerações)
η	Vetor de posição e atitude: $\eta = [x, y, z, \phi, \theta, \psi]^T$
$\mathbf{M}_{RB}$	Matriz de inércia do corpo rígido
$\mathbf{M}_A$	Matriz de massa adicionada
$\mathbf{C}_{RB}(\nu)$	Matriz de Coriolis e centrífuga do corpo rígido, função de ν
$\mathbf{C}_A(\nu)$	Matriz de Coriolis associada à massa adicionada, função de ν
$\mathbf{D}(\nu)$	Matriz de amortecimento (viscoso e potencial), função de ν
$\mathbf{g}(\eta)$	Vetor de forças de restauração (gravidade e empuxo), função de η
τ_{ext}	Vetor de forças e momentos externos (propulsão, vento e ondas)
$\widehat{\ }$	Estimativa de uma variável (valor calculado pelo observador/filtro)
$-$	Valor predito de uma variável (antes da medição, também chamado de estimativa a priori)
$+$	Valor atualizado de uma variável (após a medição, também chamado de estimativa a posteriori)
$\sim$	Indica diferença ou resíduo de uma variável (por exemplo, diferença entre medição real e estimada, ou até mesmo o erro)

Sumário

1 INTRODUÇÃO 13
1.1 Objetivos 14
1.1.1 Objetivo Geral 14
1.1.2 Objetivos Específicos 14
1.1.2.1 Revisão Teórica 14
1.1.2.2 Paralelo Entre Teoria e Simulador 14
1.1.2.3 Proposta de Melhorias e Novos Trabalhos 14
1.1.3 Justificativa 14
1.1.4 Organização do Texto 15

2 REVISÃO BIBLIOGRÁFICA E FUNDAMENTAÇÃO TEÓRICA . . 16
2.1 Forças Atuantes e Graus de Liberdade em Embarcações 16
2.2 Modelagem Matemática da Dinâmica da Embarcação 18
2.3 Princípios do Controle e Operação 20
2.3.1 Ação Integral 20
2.3.2 Controlador *Feedfoward* 21
2.3.3 Filtro de Ondas 22
2.3.4 Observador de Estados 24
2.3.4.1 Observador de Luenberger 24
2.3.4.2 Filtro de Kalman 25
2.3.5 Alocação de Empuxo 28
2.3.5.1 Matriz de Configuração dos Propulsores 30
2.3.5.2 Solução de Otimização 31
2.3.6 Controlador 33

3 PARALELO ENTRE TEORIA E SIMULADOR 37
3.1 Inicialização e Ajuste de Referências 37
3.2 Modelo Matemático 40
3.3 Controlador e Alocação de Polos 41
3.4 Dinâmica da Embarcação 45
3.5 Resultados e Análise da Simulação 46

4 RESULTADOS E DISCUSSÕES 50

5 CONSIDERAÇÕES FINAIS E PERSPECTIVAS DE TRABALHOS FUTUROS 52

Referências 54

ANEXOS 57

ANEXO A – CÓDIGO MAIN.PY 58

ANEXO B – CÓDIGO MAINLOOP.PY 61

ANEXO C – CÓDIGO SUPPLY.PY 64

ANEXO D – CÓDIGO GNC.PY 71

ANEXO E – CÓDIGO CONTROL.PY 81

1 Introdução

O posicionamento dinâmico é uma tecnologia fundamental que permite a manutenção da posição de embarcações e plataformas em espaços oceânicos, sem o uso de âncoras ou sistemas de fixação. Utilizando propulsores e sistemas automatizados, essa tecnologia tem sido aplicada com sucesso em diversas operações, desde a perfuração de poços até a instalação de equipamentos submarinos.

A história do posicionamento dinâmico, conforme Kvaal, Østby e Breivik (2022), começa nos anos 1960, quando a colaboração entre empresas de petróleo como Continental, Union, Superior e Shell resultou no CUSS 1, a primeira embarcação equipada com um sistema de posicionamento por *thruster* (propulsor). Apesar das limitações iniciais, como controle manual e sistemas de referência rudimentares, essa embarcação marcou um marco na evolução da tecnologia.

A verdadeira revolução ocorreu com o desenvolvimento do primeiro sistema de posicionamento dinâmico automático em 1961, criado por Howard Shatto para a empresa Shell, conforme visto em Breivik, Kvaal e Østby (2015). O sistema utilizava *taut wire position* (uso de cabos tensionados ligados ao fundo do mar para determinar a posição da embarcação com alta precisão), que, combinado com um controlador PID, permitiu que a embarcação se controlasse autonomamente, ajustando os propulsores de acordo com a necessidade.

A evolução do sistema de posicionamento dinâmico passou por diversas fases significativas. Inicialmente, utilizavam-se cabos conectando a embarcação ao leito marinho para detectar as movimentações. Contudo, com o avanço tecnológico, sistemas mais sofisticados, como hidrofones, GPS e giroscópios, foram gradativamente incorporados, promovendo um aumento expressivo na precisão e na confiabilidade do sistema.

Atualmente, segundo Kumar (2020), o controle de posição e direção é fundamental em operações em alto-mar (comumente chamado de *offshore*), como as realizadas pela Petrobras no Brasil. Esse tipo de controle é imprescindível não apenas na perfuração de poços e na instalação de equipamentos submarinos e veículos operados remotamente (ROVs - *Remotely Operated Vehicle*), mas também em aplicações militares e civis, tais como navios de guerra e de cruzeiro.

Atualmente, observa-se um esforço global no desenvolvimento de novas tecnologias capazes de simular as condições marítimas, com o objetivo de aprimorar estratégias de controle. Entre esses recursos, destaca-se o *Marine Systems Simulator* (MSS) *toolbox*, desenvolvido para MATLAB, que conta ainda com um complemento em Python denominado *Python Vehicle Simulator*. Embora este último ainda esteja em fase de aperfeiçoamento, já

se mostra bastante funcional, oferecendo diversas embarcações previamente modeladas e diferentes modos de controle, como posicionamento dinâmico (DP - *Dynamic Positioning*), controle de rota, profundidade e orientação. Ambos os simuladores foram desenvolvidos pela NTNU (*Norwegian University of Science and Technology*) e serão explorados ao longo deste trabalho.

1.1 Objetivos

1.1.1 Objetivo Geral

Este trabalho tem como objetivo analisar o *Python Vehicle Simulator*, desenvolvido pela NTNU (*Norwegian University of Science and Technology*).

1.1.2 Objetivos Específicos

1.1.2.1 Revisão Teórica

Realizar uma revisão da literatura sobre sistemas de posicionamento dinâmico (DP), analisando diferentes autores e identificando as tecnologias necessárias para alcançar elevados níveis de controle.

1.1.2.2 Paralelo Entre Teoria e Simulador

Relacionar o referencial teórico ao simulador, demonstrando como a teoria é aplicada na prática e de que forma as diferentes soluções e tecnologias se integram.

1.1.2.3 Proposta de Melhorias e Novos Trabalhos

Identificar possíveis melhorias no simulador e propor a implementação de novas estratégias de controle que possam ampliar seu potencial de aplicação.

1.1.3 Justificativa

O avanço da exploração de petróleo e gás em águas ultraprofundas, aliado ao aumento da distância das embarcações em relação ao continente, tem ampliado a necessidade por tecnologias capazes de garantir maior precisão e eficiência nas operações. Nesse contexto, os sistemas de posicionamento dinâmico destacam-se pela economia de combustível e pela alta confiabilidade, qualidades essenciais diante da crescente complexidade e dos desafios impostos pelo ambiente *offshore*.

Desde o desenvolvimento do primeiro sistema de posicionamento dinâmico, no início da década de 1960, até as soluções mais modernas, como o uso de GPS e radares avançados, a evolução dessa tecnologia tem sido determinante para assegurar a segurança

e a eficácia das operações marítimas. Centros de pesquisa de referência mundial, como a NTNU, investem continuamente no aprimoramento desses sistemas, por meio de modelos e metodologias que viabilizam testes e inovações. Tais soluções abrangem desde protótipos em escala reduzida de embarcações até modelos matemáticos sofisticados, capazes de simular e otimizar o desempenho do posicionamento dinâmico em diferentes condições ambientais.

1.1.4 Organização do Texto

A continuação deste trabalho está organizada a seguinte forma:

- O capítulo 2 apresenta a revisão teórica, organizada em subseções que progridem dos fundamentos matemáticos e da modelagem dinâmica até as técnicas de controle e de alocação de empuxo, fornecendo a base conceitual necessária para a implementação no simulador;
- A correlação entre o referencial teórico do capítulo anterior e o *Python Vehicle Simulator* está presente no capítulo 3, descrevendo o mapeamento entre teoria e arquitetura do simulador e comparando, em cada etapa de implementação, modelos, algoritmos e resultados simulados, com vista a evidenciar aderências, limitações e oportunidades de melhoria;
- Já o capítulo 4 apresenta os resultados da comparação e as observações levantadas durante o desenvolvimento do trabalho;
- Por fim, o capítulo 5 encerra o trabalho com as considerações e sugestões para trabalhos futuros.

2 Revisão Bibliográfica e Fundamentação Teórica

Neste capítulo é apresentado o referencial teórico sobre as forças atuantes, a modelagem matemática da embarcação e os princípios de controle aplicáveis a sistemas de DP. São discutidos os fenômenos físicos relevantes, as formulações dinâmicas utilizadas na modelagem e as estratégias de controle e de alocação de empuxo comumente empregadas na literatura.

2.1 Forças Atuantes e Graus de Liberdade em Embarcações

Em ambientes marítimos, as embarcações estão constantemente sujeitas à ação de forças naturais, como correntes oceânicas, ondas e ventos. A incidência dessas forças em diferentes pontos do casco pode provocar reações que torcem a estrutura, causam rotações e deslocam o navio para posições indesejadas. Ventos intensos, por sua vez, podem gerar forças de deriva que comprometem a estabilidade da embarcação, dificultando a manutenção da posição ou da rota estabelecida.

A interação dessas forças com a estrutura resulta em movimentos complexos, representados por seis graus de liberdade (DOF, do inglês *Degrees of Freedom*), que descrevem todos os possíveis deslocamentos lineares e rotações de um corpo no espaço. Utilizando um sistema de coordenadas cartesianas para orientação espacial em relação à embarcação, o movimento ao longo do eixo longitudinal (x) é denominado avanço (do inglês *surge*), representando o deslocamento para frente e para trás. O deslocamento lateral, ao longo do eixo transversal (y), é chamado de deriva (do inglês *sway*), enquanto o movimento no eixo z é conhecido como movimento vertical (do inglês *heave*). Além desses deslocamentos lineares, ocorrem movimentos de rotação em torno de cada eixo: a rotação em torno do eixo x é chamada de rolagem (do inglês *roll*), em torno do eixo y de arfagem (do inglês *pitch*) e, em relação ao eixo z, guinada (do inglês *yaw*). Estes movimentos são ilustrados pela Figura 1.

Na literatura acadêmica, é comum encontrar a classificação de modelos de embarcações com base nos graus de liberdade considerados para fins de controle, simulação ou análise de estabilidade. Esses modelos variam de acordo com o nível de detalhamento exigido pelo estudo ou aplicação, podendo ser agrupados conforme a quantidade de movimentos considerados na modelagem dinâmica (FOSSEN, 2011, cap. 1):

- Modelos que consideram apenas um grau de liberdade, geralmente utilizados para o

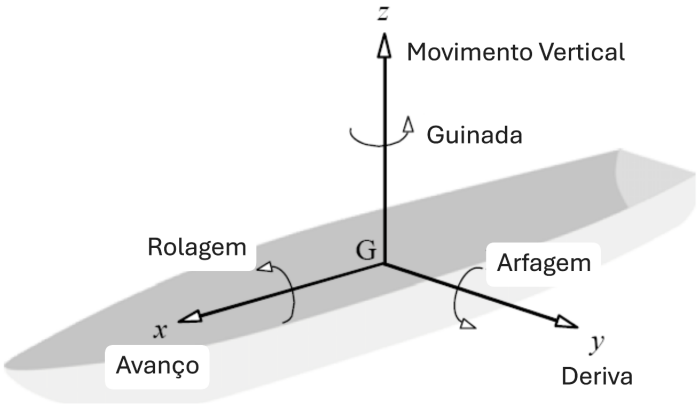

Figura 1 – Diagrama representando os graus de liberdade sob uma embarcação

Fonte: Adaptado de Ibrahim e Grace (2018).

controle de velocidade ao longo de uma linha reta (*surge*) ou para o direcionamento angular da embarcação (*yaw*). São empregados em aplicações simples ou em fases iniciais de modelagem.

- Modelos que consideram os movimentos nos eixos x, y e z (*surge*, *sway* e *yaw*), portanto, com uma ação em três graus de liberdade. Esse tipo de modelagem é amplamente utilizado em sistemas de posicionamento dinâmico (DP do inglês, *Dynamic Positioning*) e no controle de rotas, onde é necessário controlar o deslocamento planar e a orientação da embarcação.

- Com quatro graus de liberdade, estendem-se os modelos de 3 DOF com a adição da rotação em torno do eixo longitudinal (*roll*), sendo úteis quando se deseja controlar o balanço lateral da embarcação, especialmente em mares mais agitados ou em projetos onde o conforto e a estabilidade são aspectos críticos.

- Por fim, modelos completos e complexos, que incorporam todos os deslocamentos lineares (surge, sway e heave) e rotações (*roll*, *pitch* e *yaw*). Esses modelos são amplamente empregados em simulações com alto grau de realismo, no desenvolvimento de projetos de engenharia naval avançada, bem como na análise do comportamento dinâmico de embarcações em mar aberto e no desenvolvimento de veículos submersíveis, que operam sob todas as condições de movimento possíveis no ambiente marítimo.

A Tabela 1 relaciona os graus de liberdade às suas respectivas nomenclaturas e representações matemáticas, de modo a fornecer uma visão organizada dos movimentos que compõem a dinâmica de embarcações e servindo como referência para a modelagem adotada neste trabalho.

DOF	Motion (descrição)	Força/Momento	Velocidades	Posição e Ângulos
1	*Surge* (movimento no eixo x)	X	u	x
2	*Sway* (movimento no eixo y)	Y	v	y
3	*Heave* (movimento no eixo z)	Z	w	z
4	*Roll* (rolamento, rotação eixo x)	K	p	Φ (phi)
5	*Pitch* (arfagem, rotação eixo y)	M	q	Θ (theta)
6	*Yaw* (guinada, rotação eixo z)	N	r	Ψ (psi)

Tabela 1 – Tabela de Graus de Liberdade (DOF) e suas Correspondências

Fonte: Extraído de Fossen e Fjellstad (1995).

2.2 Modelagem Matemática da Dinâmica da Embarcação

Em Fossen e Fjellstad (1995), é desenvolvida uma modelagem não linear para veículos subaquáticos que, dada a devida restrição em graus de liberdade, uma vez que veículos subaquáticos geralmente exigem controle em 6 DOF, é usada em modelos de 3 DOF para atender ao posicionamento dinâmico.

Para a modelagem é comumente usado um referencial no corpo da embarcação, em inglês *body-fixed*, representado pela Figura 2, que se trata de um sistema de coordenadas que se move rigidamente com o veículo, ou seja, suas origens e eixos estão fixos no corpo da embarcação, acompanhando sua posição e orientação em cada instante. Neste referencial, a velocidade linear e angular do veículo é representada pelo vetor $\nu = [u, v, w, p, q, r]^{\mathrm{T}}$, onde u, v, w são as componentes de velocidade linear nas direções do corpo (*surge*, *sway*, *heave*), e p, q, r são as componentes de velocidade angular (*roll*, *pitch*, *yaw*).

O referencial fixo na Terra é frequentemente referido, na literatura em inglês, como *earth-fixed*, representado pela Figura 2, é um sistema de coordenadas inercial, fixo em relação à Terra, utilizado para descrever a posição e a orientação de um veículo de forma absoluta. Nesse referencial, o vetor $\eta = [x, y, z, \phi, \theta, \psi]^{\mathrm{T}}$ representa a posição linear (x, y, z) e a orientação angular (ϕ, θ, ψ) do veículo em relação à Terra, correspondendo aos deslocamentos e rotações no sistema inercial. O uso do referencial *earth-fixed* é essencial para relacionar o movimento do veículo com o ambiente externo e para projetar sistemas de controle que mantêm ou alteram sua posição e atitude no espaço absoluto.

Baseando-se na segunda lei de Newton, que rege o momento angular e linear, obtém-se a equação reduzida:

$$(\mathbf{M}_{RB} + \mathbf{M}_A)\dot{\nu} + [\mathbf{C}_{RB}(\nu) + \mathbf{C}_A(\nu)]\,\nu + \mathbf{D}(\nu)\nu + \mathbf{g}(\eta) = \tau_{ext} \tag{2.1}$$

Considera-se como base que M_{RB} é a matriz de inércia rígida de ordem n×n, na qual m é a massa total do corpo, x_G,y_G e z_G são as coordenadas para o centro de massa, adicionalmente I_x,I_y e l_z são os momentos de inércia, que expressam a resistência do corpo a variações do seu movimento rotacional em torno dos eixos principais:

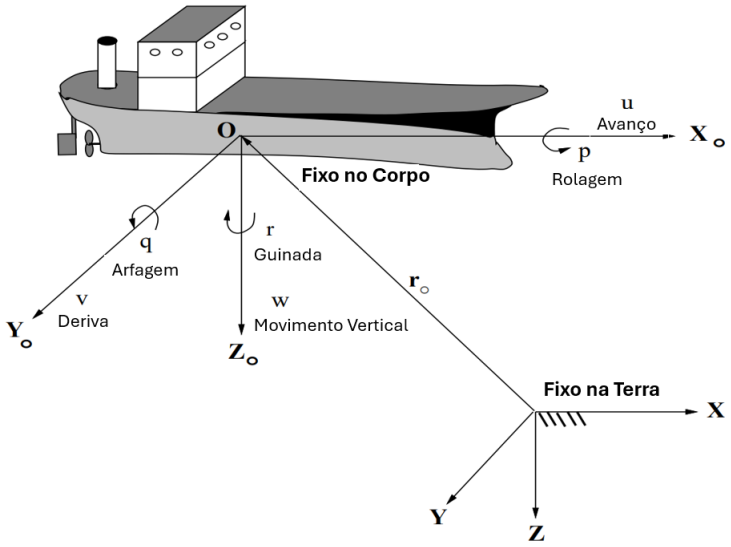

Figura 2 – Diagrama representando os referenciais usados para descrever os movimentos da embarcação

Fonte: Adaptado de Fossen e Fjellstad (1995).

$$\mathbf{M}_{RB} = \begin{bmatrix} m & 0 & 0 & 0 & mz_G & -my_G \\ 0 & m & 0 & -mz_G & 0 & mx_G \\ 0 & 0 & m & my_G & -mx_G & 0 \\ 0 & -mz_G & my_G & I_x & -I_{xy} & -I_{xz} \\ mz_G & 0 & -mx_G & -I_{xy} & I_y & -I_{yz} \\ -my_G & mx_G & 0 & -I_{xz} & -I_{yz} & I_z \end{bmatrix} \tag{2.2}$$

A matriz $C_{RB}(\nu)$ contém os termos de Coriolis, que surgem devido à rotação do referencial ligado ao corpo em relação ao referencial inercial (*body-fixed* e *earth-fixed*, respectivamente). Representada por:

$$\mathbf{C}_{RB}(\nu) = \begin{bmatrix} 0 & 0 & 0 & m(y_G q + z_G r) & m(y_G p + w) & m(z_G p - v) \\ 0 & 0 & 0 & m(x_G q - w) & -m(z_G r + x_G p) & m(z_G q + u) \\ 0 & 0 & 0 & m(x_G r + v) & m(y_G r - u) & -m(x_G p + y_G q) \\ -m(y_G q + z_G r) & -m(x_G q - w) & -m(x_G r + v) & 0 & -(I_{xz} p + I_{yz} q + I_z r) & I_{xy} p + I_y q + I_{yz} r \\ -m(y_G p + w) & m(z_G r + x_G p) & -m(y_G r - u) & I_{xz} p + I_{yz} q + I_z r & 0 & -(I_x p + I_{xy} q + I_{xz} r) \\ -m(z_G p - v) & -m(z_G q + u) & m(x_G p + y_G q) & -(I_{xy} p + I_y q + I_{yz} r) & I_x p + I_{xy} q + I_{xz} r & 0 \end{bmatrix}. \tag{2.3}$$

As matrizes elaboradas em (2.2) e (2.3) são de ordem 6×6, portanto abrangendo 6 graus de liberdade. Quando necessário, dependendo do sistema em fase de projeto, pode ser ajustado para $n \times n$ para atender a n graus de liberdade, limitado a 6 DOF.

A matriz $D(\nu)$ representa os termos dissipativos como atrito e amortecimento potencial e viscoso (em decorrência do meio por onde a embarcação se desloca), em função

de ν. Por sua vez, τ_{ext} é o vetor de forças e momentos externos aplicados ao veículo (como propulsão, vento e ondas). Já o termo $\mathbf{g}$ representa as forças restauradoras do modelo, que, neste caso, correspondem à gravidade e ao empuxo gerado pela água.

O sistema, então, é modelado de acordo com sua aplicação e as características da embarcação a ser controlada. Em determinadas situações, torna-se necessário considerar os efeitos da massa adicionada ao sistema durante o movimento. Esse fenômeno decorre do deslocamento do fluido ao redor do corpo, no caso do sistema de posicionamento dinâmico, a água, que, quando sofre acelerações ou desacelerações, gera uma inércia adicional que se opõe ao movimento. Embora o impacto da massa adicionada seja relativamente pequeno em sistemas de DP, seus efeitos são incorporados à modelagem por meio das matrizes M_A e C_A, as quais, somadas às componentes de corpo rígido, descrevem com maior precisão o comportamento dinâmico da embarcação.

2.3 Princípios do Controle e Operação

Um sistema de posicionamento dinâmico, apesar de bem estruturado e modelado para representar as características do objeto real a ser controlado, pode ser submetido a diversas situações imprevisíveis que o ambiente marítimo proporciona. Portanto, sistemas de DP costumam ser implementados com recursos essenciais que minimizam falhas e tornam o sistema mais robusto. Segundo Fossen (2011, cap. 12), é fundamental que um sistema comercial conte com uma ação integral, um controlador *feedforward*, filtro de ondas, observador de estados e um controle de alocação de empuxo.

2.3.1 Ação Integral

A ação integral, por sua construção, acumula o erro ao longo do tempo para corrigir desvios em relação à referência desejada, permitindo ao controlador compensar forças lentamente variáveis. Portanto, quanto maior for o tempo de persistência do erro, maior será a atuação dessa componente. Segundo Bosgra, Kwakernaak e Meinsma (2001, p. 70–75), sistemas de controle que utilizam integração, além de eliminarem o erro em regime permanente, são extremamente indicados para aplicações que necessitam atenuar baixas frequências, devido à sua efetividade.

A ação integral pode ser definida por:

$$b(t) = K_i \int_0^t e(\tau)\, d\tau \tag{2.4}$$

Sendo o viés b (do inglês, *bias*) introduzido pelo elemento integral ao integrar o erro, essa operação acumula todo o histórico do erro até o instante presente, permitindo ao controlador compensar forças lentamente variáveis. Em um sistema de posicionamento

dinâmico (DP), essa ação é particularmente relevante por corrigir perturbações de baixa frequência, como correntes oceânicas e dinâmicas ambientais não modeladas.

2.3.2 Controlador *Feedfoward*

O controle *feedforward* (controle antecipativo) é uma técnica de controle utilizada para antecipar a ação de distúrbios externos antes mesmo que estes entrem no *feedback* (realimentação) do controlador principal. Diferente do controle por *feedback*, que reage ao erro após sua ocorrência, o *feedforward* atua de maneira proativa, utilizando informações conhecidas ou previsíveis sobre as perturbações.

Em um sistema de DP, essa ação se torna vital por corrigir perturbações de baixa frequência, como correntes oceânicas e dinâmicas ambientais não modeladas. De acordo com Elliott e Sutton (1996, p. 214–223), "O controle *feedforward* baseia-se na existência de algum conhecimento prévio da perturbação a ser controlada, que está contido em um sinal de referência que aciona a fonte secundária através do controlador" (tradução própria).

Essa abordagem é particularmente eficaz em sistemas onde é possível medir ou estimar a perturbação com antecedência. Em sistemas de DP é utilizado *Wind FeedForward*, um modelo matemático que representa como o sistema responde em casos de variações de vento que possuem frequências e forças medianas, ideal para complementar o controle integrador descrito na Seção 2.3.1, que atua em baixa frequência. Adicionalmente, Fossen (2011, cap. 12) destaca que rajadas de vento não podem ser compensadas, uma vez que os atuadores não possuem capacidade para mover uma embarcação de grande porte nas faixas de frequência associadas a essas rajadas, geralmente altas. Além disso, em Fossen (2011, cap. 8) é destacado que a inércia da embarcação é suficientemente grande para que rajadas sejam consideradas desprezíveis, não exigindo compensação no sistema de controle.

No sistema proposto por Sørensen, Sagatun e Fossen (1996), para a modelagem do controle *Wind Feedforward*, foram utilizados sensores instalados na embarcação capazes de realizar a leitura direta da força e direção do vento. Por meio da aplicação de um filtro de Kalman e de um observador de Luenberger (ainda a serem abordados neste trabalho), foi possível estimar as forças do vento atuando nos eixos *surge*, *sway* e *yaw*, uma vez que a força do vento pode ser decomposta e, portanto, sua influência pode ser modelada matematicamente em cada sentido de movimento. Essas forças estimadas, representadas por τ_{wind}, ao serem multiplicadas por uma matriz de ganho G_w, resultam no termo de controle de *feedforward* τ_{ff} aplicado ao sistema, conforme a seguinte expressão:

$$\tau_{\text{ff}} = -G_w \cdot \tau_{\text{wind}} \tag{2.5}$$

Onde:

$$\mathbf{G}_w = \text{diag}(g_{w1}, g_{w2}, g_{w3}) \tag{2.6}$$

Com $0 \leq g_{wi} \leq 1$, para $i \in \mathbb{Z}^+$, $i = 1, 2, 3, \ldots$, em que cada termo representa um ganho ajustado para o eixo correspondente.

A Figura 3 apresenta um modelo proposto por Qu et al. (2015), com o objetivo de implementar uma estratégia de controle *feedforward* de vento em um veículo de superfície não tripulado (USV - do inglês Unmanned Surface Vehicle). O diagrama ilustra a estrutura do sistema de controle utilizado, destacando a atuação antecipada com base em sinais de referência provenientes das forças do vento.

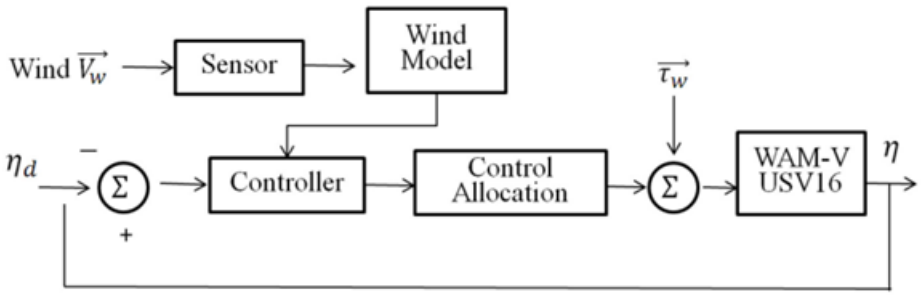

Figura 3 – Diagrama em blocos exemplificando o sistema *feedforward*

Fonte: Extraído de Qu et al. (2015).

2.3.3 Filtro de Ondas

Ao longo do desenvolvimento dos sistemas de DP, um dos principais desafios enfrentados foi a presença de ruídos indesejados no sistema de controle, especialmente aqueles causados por forças externas atuando, como as induzidas pelas ondas. Para mitigar os efeitos dessas oscilações, tornou-se necessário implementar técnicas que impedissem a entrada de componentes de alta frequência (como as forças induzidas por ondas) no laço de realimentação do controlador (*feedback*).

Distúrbios de alta-frequência que entram no controlador tendem a gerar respostas bruscas, resultando em maior desgaste do sistema de propulsão. Além disso, em picos muito elevados, Fossen (2011, cap. 11) destaca que os sistemas de propulsão não conseguem mover a embarcação a tempo (devido ao seu tamanho e inércia) para compensar esses distúrbios com eficácia. Esses distúrbios são ilustrados através da Figura 4, onde é possível ver o distúrbio após a separação da componente de baixa frequência, LF.

A filtragem de ondas (WF, do inglês *Wave Filtering*) é uma funcionalidade essencial em sistemas de posicionamento dinâmico (DP) modernos. Essa técnica tem como objetivo decompor as medições de posição e rumo em dois componentes distintos: uma parte

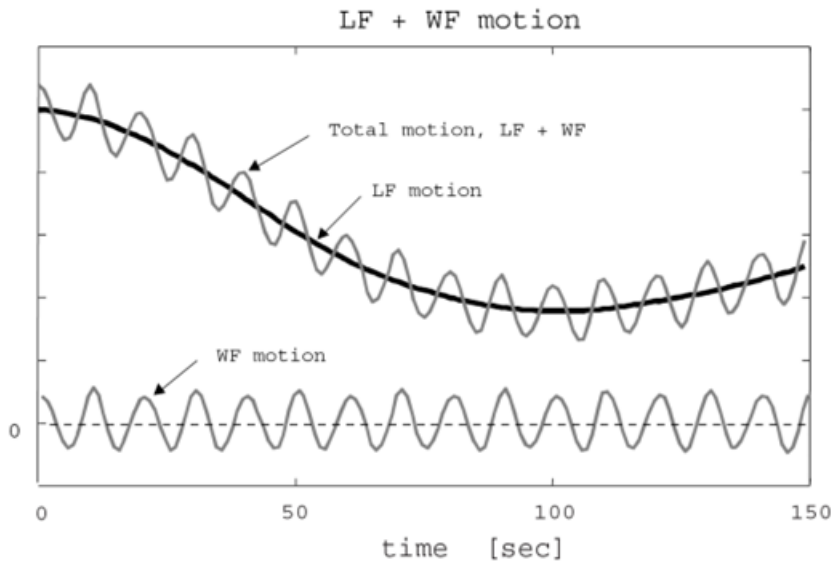

Figura 4 – Frequências associadas ao movimento de uma embarcação

Fonte: Extraído de Fossen e Strand (1999).

de baixa frequência (LF, do inglês *Low Frequency*), associada a movimentos lentos da embarcação, e outra de alta frequência (HF, do inglês *High Frequency*), relacionada às oscilações rápidas causadas por ondas.

Diversas abordagens têm sido desenvolvidas para a filtragem de componentes indesejados de alta frequência, como as oscilações induzidas por ondas, incluindo o uso de filtros passa-baixa e observadores de estado. Entre esses observadores, o filtro de Kalman se destaca como o mais amplamente utilizado na indústria marítima. Trabalhos mais recentes, como Popov e Milanov (2019, p. 205–216), propõem variações adaptativas do filtro de Kalman, capazes de ajustar automaticamente seus parâmetros de acordo com as condições do mar, com o objetivo de otimizar a filtragem de ondas em sistemas de DP.

De forma distinta, Loueipour et al. (2015) propõe uma técnica alternativa, que reconstrói os movimentos de LF e WF utilizando apenas dados de posicionamento. A abordagem combina um filtro *notch*, um filtro seletivo que atenua faixas estreitas de frequência específicas, com um observador não linear, permitindo que os distúrbios de LF sejam estimados. Um dos principais diferenciais dessa técnica é que a filtragem independe de modelos matemáticos da embarcação, proporcionando versatilidade, mesmo diante de incertezas e variações no ambiente operacional.

As estimativas de posição, velocidade e rumo isentas da influência de ondas (*wave-free estimates*), provenientes dos filtros ou observadores implementados, são então utilizadas na malha de realimentação, garantindo ao sistema de controle sinais mais limpos e adequados à atuação eficiente do sistema propulsivo.

2.3.4 Observador de Estados

Observadores de estado, em sistemas de posicionamento dinâmico, possuem um papel essencial na busca pela filtragem de ruído e estimativa de variáveis impossíveis de serem medidas diretamente ou que elevariam o custo do projeto de tal forma que o inviabilizaria. Fannemel (2008), em sua tese de mestrado, elabora que estimadores de estado possibilitam estimar a velocidade linear e angular da embarcação, bem como combinar diferentes sensores como giroscópios e GPS para uma estimativa mais assertiva. Apresenta também o papel de compensar forças externas como descrito brevemente na Seção 2.3.3 deste trabalho.

Em seu livro Fossen (2011), traz modelos matemáticos de observadores que atualmente são utilizados em diversas áreas da engenharia e pesquisa. A seguir, é realizada uma breve explicação de dois dos que mais se destacam.

2.3.4.1 Observador de Luenberger

Em seu trabalho, Luenberger (1966) propôs um observador de estados para sistemas lineares. Por ser linear e invariante no tempo, apresenta uma implementação computacional mais simples e leve. Contudo, por sua linearidade, não é indicado a ser utilizado diretamente para controlar sistemas estocásticos; entretanto, como apresentado por Fossen (2011), pode ser implementado em filtros de ondas, descritos na Seção 2.3.3, como alternativa a filtros passa-baixa e *notch*. A adição de um observador modelado com as características específicas do sistema pode resultar em desempenho mais satisfatório.

O observador de Luenberger atua em sistemas determinísticos e de maneira recursiva, usando seu erro para realimentar o filtro e aumentar a acurácia. É um sistema que não considera ruído explicitamente, e é matematicamente formulado por um sistema observável e invariante no tempo:

$$\dot{\hat{x}} = A\hat{x} + Bu + \gamma(y, \hat{y}) \tag{2.7}$$

$$\hat{y} = Hx \tag{2.8}$$

Portanto, modelando o sistema em espaço de estados, x representa o vetor de estados, u são as entradas de controle aplicadas e y as saídas medidas, onde H é a *matriz de saída* que relaciona os estados às medições, A descreve a dinâmica do processo e B como u afeta $\dot{x}$.

Sendo K uma matriz de ganhos do observador que é definido ao injetar y como:

$$\gamma(y, \hat{y}) = K\varepsilon \tag{2.9}$$

$$\varepsilon = y - \hat{y} \tag{2.10}$$

$$\varepsilon = Hx \tag{2.11}$$

Assim:

$$\dot{\hat{x}} = A\hat{x} + Bu + K\varepsilon \tag{2.12}$$

2.3.4.2 Filtro de Kalman

O filtro de Kalman se assemelha ao observador de Luenberger por ser um estimador de estados recursivo, ou seja, atualiza continuamente suas estimativas a cada nova medição disponível. Sua principal vantagem está na capacidade de atenuar os efeitos de ruídos tanto no modelo do processo quanto nas medições, fornecendo estimativas mais confiáveis do estado real do sistema.

Para o desenvolvimento de um filtro de Kalman, assume-se que o sistema é linear e observável. Essa condição é essencial para garantir que os estados internos possam ser inferidos corretamente a partir das saídas medidas, permitindo a convergência das estimativas.

O modelo do processo define a evolução do estado entre os instantes $k-1$ e k. Em sua forma geral, essa evolução é descrita por:

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \tag{2.13}$$

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \nu_k \tag{2.14}$$

Nessas expressões, F é a matriz de transição de estados, e B é a matriz de entrada de controle. A matriz H representa a relação entre os estados e as medições do sistema.

Os termos $\mathbf{w}_{k-1} \sim \mathcal{N}(0, \mathbf{Q})$ e $\nu_k \sim \mathcal{N}(0, \mathbf{R})$ representam os ruídos do processo e de medição, respectivamente. Ambos são modelados como variáveis aleatórias com distribuição normal de média zero e covariâncias $\mathbf{Q}$ e $\mathbf{R}$, que quantificam a incerteza associada a cada fonte de erro.

> "A covariância é uma medida estatística que expressa como duas variáveis aleatórias variam em conjunto. Quando duas ou mais variáveis são definidas

> em um espaço de probabilidade, torna-se útil descrever como elas se relacionam — ou seja, como se comportam conjuntamente ao longo de diferentes realizações.” Montgomery (2020, p. 123)

No trabalho intitulado *Introduction to Kalman Filter and Its Applications*, Kim e Bang (2018) apresentam uma abordagem algorítmica para a formulação do filtro, dividindo-o em dois passos principais:

1. Predição:
   - Estimativa do estado:
   $$\hat{x}_k^- = F\hat{x}_{k-1}^+ + Bu_{k-1} \tag{2.15}$$
   - Estimativa da covariância do erro:
   $$P_k^- = FP_{k-1}^+F^T + Q \tag{2.16}$$
2. Atualização das variáveis (recursividade):
   - Medição residual (diferença entre medição e previsão):
   $$\tilde{y}_k = z_k - H\hat{x}_k^- \tag{2.17}$$
   - Cálculo do ganho de Kalman:
   $$K_k = P_k^-H^T\left(R + HP_k^-H^T\right)^{-1} \tag{2.18}$$
   - Atualização do estado estimado:
   $$\hat{x}_k^+ = \hat{x}_k^- + K_k\tilde{y}_k \tag{2.19}$$
   - Atualização da covariância do erro:
   $$P_k^+ = (I - K_kH)P_k^- \tag{2.20}$$

Importante referenciar que $\hat{x}_k^-$ é a estimativa predita do estado no instante k, antes da medição, e $\hat{x}_k^+$ é a estimativa corrigida após a medição. P é a matriz de covariância do erro, que expressa a incerteza associada à estimativa e, por fim, K_k é o ganho de Kalman. A Figura 5 apresenta, de forma gráfica, o funcionamento do filtro de Kalman e a correlação entre suas etapas.

Posteriormente, foi desenvolvido o filtro de Kalman estendido (*Extended Kalman Filter* – EKF), que, como exemplificado por Kim e Bang (2018), pode ser utilizado de forma eficiente em sistemas não lineares, como o posicionamento dinâmico, nos quais não

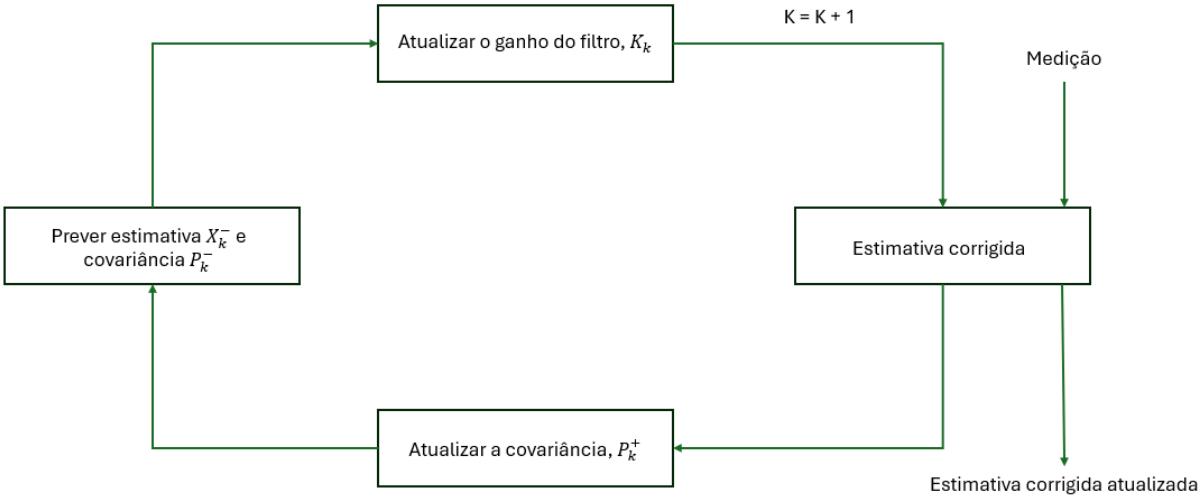

Figura 5 – Funcionamento do filtro de Kalman

Fonte: Adaptado de Fannemel (2008).

é possível representar diretamente os modelos do processo ou da variável medida apenas com matrizes constantes.

Portanto, o filtro estendido se difere do convencional pelo uso das funções não lineares $f(x_{k-1}, u_{k-1})$ e $h(\hat{x}_k)$ nas equações que modelam o sistema:

$$x_k = f(x_{k-1}, u_{k-1}) + w_{k-1} \tag{2.21}$$

E assim, seguindo o algoritmo proposto para o cálculo recursivo:

$$z_k = h(x_k) + \nu_k \tag{2.22}$$

1. Predição:

   - Estimativa do estado:
   $$\hat{x}_k^- = f(\hat{x}_{k-1}^+, u_{k-1}) \tag{2.23}$$
   - Estimativa da covariância do erro:
   $$P_k^- = F_{k-1} P_{k-1}^+ F_{k-1}^T + Q \tag{2.24}$$

2. Atualização das variáveis (recursividade):

   - Medição residual (diferença entre medição e previsão):
   $$\tilde{y}_k = z_k - h(\hat{x}_k^-) \tag{2.25}$$

- Cálculo do ganho de Kalman:

$$K_k = P_k^- H_k^T \left(H_k P_k^- H_k^T + R\right)^{-1} \tag{2.26}$$

- Atualização do estado estimado:

$$\hat{x}_k^+ = \hat{x}_k^- + K_k \tilde{y}_k \tag{2.27}$$

- Atualização da covariância do erro:

$$P_k^+ = (I - K_k H_k) P_k^- \tag{2.28}$$

2.3.5 Alocação de Empuxo

Para locomoção, embarcações de centenas de toneladas contam com uma enorme gama de propulsores poderosos que possuem características únicas auxiliando da maneira mais eficiente no movimento planejado para o navio durante a sua vida útil. Comumente observa-se três tipos de atuadores (Figura 6):

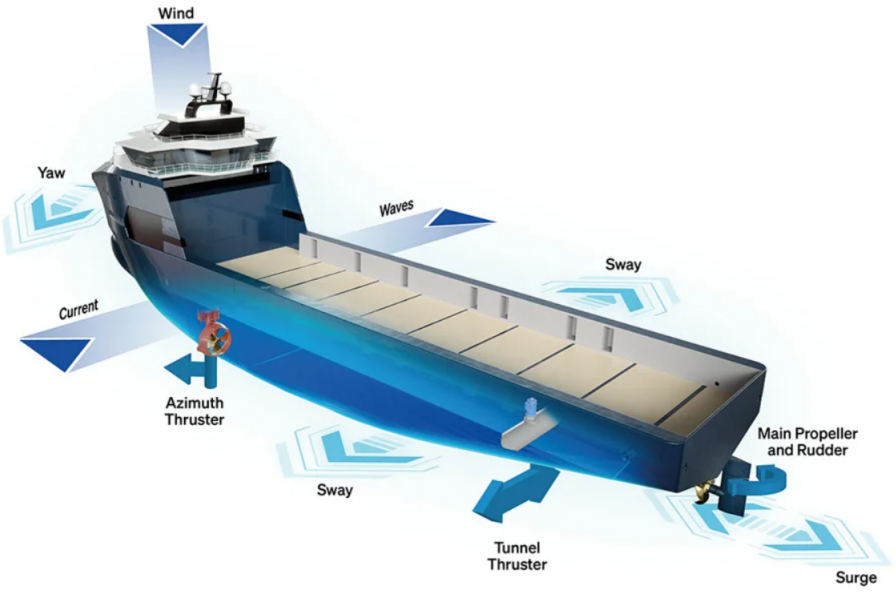

Figura 6 – Tipos de propulsores

Fonte: Extraído de KONGSBERG (2025).

- **Propulsor principal (*Main Propeller*)**: São dispositivos robustos que proporcionam maior empuxo para a embarcação durante movimentos longitudinais (avanço e ré). Geralmente são instalados na popa da embarcação e, na maioria dos casos, possuem hélices de direção fixa, embora possam apresentar superfícies móveis denominadas “leme” para auxiliar no controle da embarcação.

- **Propulsores de túnel (*Tunnel Thrusters* ou *Bow Thrusters*)**: Propulsores instalados transversalmente no casco, podendo ser posicionados em diferentes pontos ao longo do comprimento da embarcação para melhor adequação ao projeto. Possuem menor capacidade de geração de empuxo em comparação com o propulsor principal, sendo utilizados como auxiliares durante manobras e em sistemas de posicionamento dinâmico (DP).
- **Propulsores azimutais (*Azimuth Thrusters*)**: Possuem funcionalidade semelhante aos propulsores de túnel, com a diferença de que realizam rotação em torno do eixo Z, direcionando o empuxo para facilitar manobras de forma mais eficiente. Podem ser instalados tanto na proa quanto na popa da embarcação. Devido a sua construção apresentam limitações quanto à liberdade angular, por esse motivo é necessário adotar técnicas rebuscadas de otimização para indicar a angulação mais adequada para responder à necessidade de controle.

O controlador do sistema de DP deve então, a partir de todas as estimações e modelagens descritas nas seções anteriores deste trabalho, orientar e acionar os atuadores disponíveis na embarcação afim de permitir o movimento e intensidade desejada. Para isso é utilizada a seguinte relação(FOSSEN; SAGATUN; SØRENSEN, 1996):

$$f = Ku \tag{2.29}$$

$$\tau = T(\alpha)f \tag{2.30}$$

$$\tau = T(\alpha)Ku \tag{2.31}$$

Essa equação expressa a relação entre o vetor de forças ,f, e momentos ,τ, resultantes no corpo da embarcação, e as entradas de controle aplicadas aos propulsores ,$\mathbf{u}$, por meio da matriz de configuração de propulsão $T(\alpha)$, que depende dos ângulos de orientação dos propulsores azimutais (para projetos que não envolvem controle angular, a matriz T não recebe entradas), e da matriz K, uma matriz diagonal que contém os coeficientes de empuxo de cada atuador. Essa relação permite modelar como a orientação e o empuxo de cada propulsor contribuem para o movimento translacional e rotacional da embarcação.

$$K = \text{diag}[k_1, k_2, \dots, k_r] \tag{2.32}$$

As entradas de controle do sistema são as razões de passo (*pitch ratio*) de cada propulsor, definidas pela razão entre o passo geométrico, que corresponde à distância percorrida por rotação, e o diâmetro do propulsor.

$$\mathbf{u} = \left[p_1 - p_{10}(p_1 - p_{10}), \quad p_2 - p_{20}(p_2 - p_{20}), \quad \ldots, \quad p_r - p_{r0}(p_r - p_{r0})\right]^{\mathrm{T}} \tag{2.33}$$

No modelo considerado, a força de empuxo de um propulsor é aproximada por uma relação que envolve o coeficiente de força, dependente da rotação, e a diferença entre a razão de passo e um valor de referência denominado deslocamento de passo (*offset*). Esse deslocamento de passo é definido como o ponto no qual o empuxo produzido é nulo. A expressão implica que a magnitude do empuxo cresce quadraticamente com essa diferença. Para um conjunto de propulsores, as entradas de controle são organizadas em um vetor que considera, para cada atuador, a diferença entre a razão de passo e seu respectivo deslocamento de passo, com a devida preservação do sinal.

2.3.5.1 Matriz de Configuração dos Propulsores

Parte da modelagem do sistema de controle necessita que seja possível expressar matematicamente como a embarcação se comporta dado determinado impulso nos seus atuadores. Para isso, faz-se necessário o uso de um sistema de coordenadas com o ponto central escolhido ,O, usado para definir as equações de movimento, conforme a Figura 7.

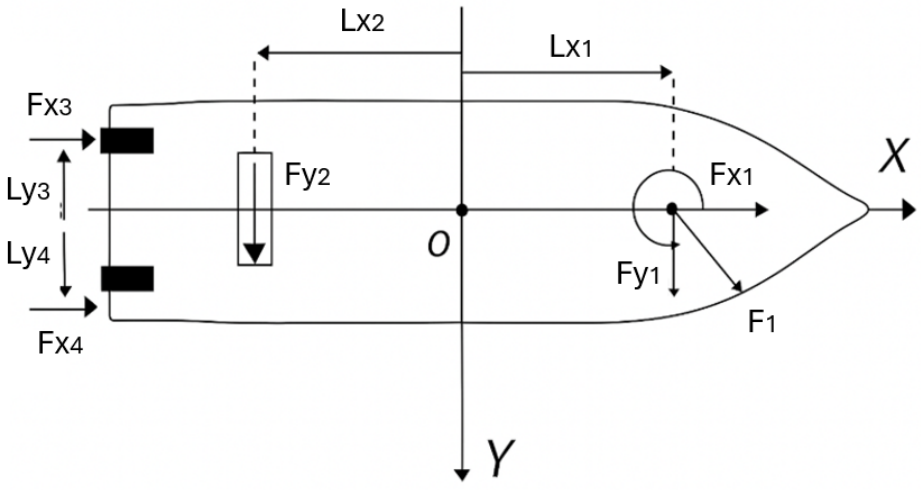

Figura 7 – Esquema forças e distribuição dos atuadores no casco

Fonte: Elaborado pelo autor (2025).

Dessa forma, a matriz de configuração espacial dos atuadores proposta por Fossen (2011, cap. 12) é representada conforme o exemplo na Figura 8. Observa-se que os propulsores principais, por apresentarem $\alpha = 0°$, atuam predominantemente no movimento de avanço e podem influenciar o movimento de guinada. Já os propulsores de túnel, com $\alpha = 90°$, atuam no movimento de deriva e exercem influência nos movimentos de rolagem e guinada. Por fim, os atuadores azimutais dependem diretamente do ângulo de incidência α e podem influenciar todos os movimentos, desde que não haja restrições no seu giro.

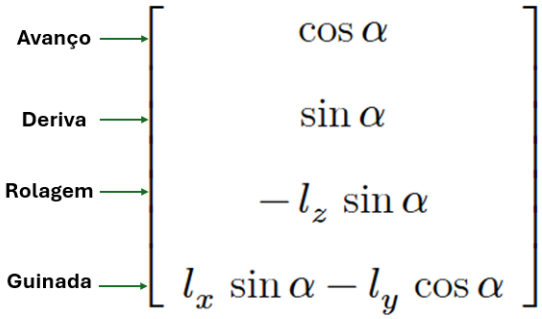

Figura 8 – Influência dos propulsores nos movimentos

Fonte: Elaborado pelo autor (2025).

Portanto, a matriz T pode ser representada conforme a Equação 2.34, na qual cada elemento t_i corresponde a um atuador responsável por contribuir para o movimento da embarcação. É importante ressaltar que as matrizes T e K devem estar alinhadas de modo que a sequência de suas colunas represente, de forma correspondente, o mesmo propulsor.

$$T(\alpha) = \begin{bmatrix} t_1 & t_2 & \cdots & t_i \end{bmatrix} \tag{2.34}$$

2.3.5.2 Solução de Otimização

Em função de exigências regulamentares de segurança, embarcações modernas frequentemente apresentam número de propulsores superior ao necessário para o funcionamento básico; assim, quando o sistema é sobre-atuado, isto é, possui entradas de controle em quantidade maior ou igual ao número de graus de liberdade ($r \geq n$), surge a necessidade de escolher uma solução ótima de alocação de controle, pois podem existir infinitas combinações de forças que gerem o mesmo empuxo resultante Fossen (2011, cap. 12.3.2). Como exemplificado em Rindarøy (2013) (Figura 9), torna-se necessário definir um critério de otimização.

Fossen e Sagatun (1991) propuseram que, ao alocar empuxo em uma direção, se minimizasse o uso de empuxo por propulsor (reduzindo o esforço nos atuadores), essa ideia

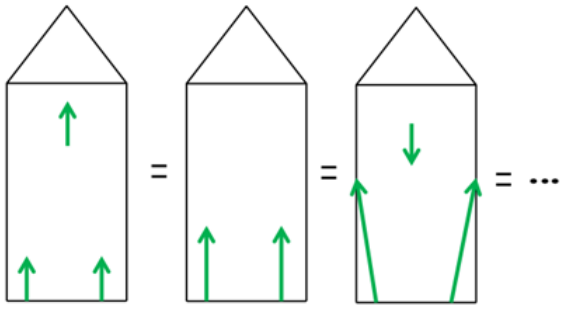

Figura 9 – Ilustração das múltiplas soluções de alocação de empuxo com a mesma força resultante

Fonte: Extraído de Rindarøy (2013)

levou à formulação por mínimos quadrados com multiplicadores de Lagrange (FOSSEN, 1994):

$$L(\mathbf{f}, \lambda) = \mathbf{f}^{\mathrm{T}} W \mathbf{f} + \lambda^{\mathrm{T}} (\tau - T\mathbf{f}) \tag{2.35}$$

Nesse contexto, $\lambda \in \mathbb{R}^r$ representa o vetor de multiplicadores de Lagrange, L denota a Lagrangiana associada ao problema, $\mathbf{f}$ corresponde ao vetor de forças de controle (variável de interesse em relação à qual será realizada a diferenciação), e W é uma matriz definida positiva que pondera a aplicação dessas forças, sendo usualmente ajustada de modo que o uso de superfícies de controle seja menos custoso do que o emprego de propulsores.

Após o desenvolvimento e a redução para a pseudo-inversa de Moore-Penrose, obtém-se:

$$T^{\dagger} = T^{\mathrm{T}} \left(T T^{\mathrm{T}} \right)^{-1} \tag{2.36}$$

Entretanto:

$$\mathbf{f} = T_w^{\dagger} \, \tau \tag{2.37}$$

A solução ótima é apresentada por:

$$\mathbf{u} = K^{-1} \, T_w^{\dagger} \, \tau \tag{2.38}$$

Na expressão $T_w^{\dagger}$, o subscrito w indica a ponderação pela matriz de pesos W e o símbolo † denota a *pseudoinversa* de Moore–Penrose, utilizada para obter a solução de

mínimos quadrados em sistemas sobreatuados.

> A *pseudoinversa* de Moore-Penrose é uma matriz que pode atuar como substituta parcial da inversa de uma matriz nos casos em que esta não existe. Essa matriz é frequentemente utilizada para resolver sistemas de equações lineares quando o sistema não possui solução única ou possui múltiplas soluções. MathWorks (2025) (tradução própria)

A solução sem restrições apresentada por Fossen e Sagatun (1991) e detalhada em Fossen (2011) aplica-se a casos em que os ângulos α dos atuadores permanecem fixos ao longo do tempo. Um ponto de atenção nesse método é a ausência de limitação explícita para o empuxo dos propulsores, o que exige a implementação de mecanismos de proteção para evitar saturação durante a operação do sistema de controle.

Outros autores propuseram abordagens distintas para a solução ótima. Rindarøy (2013), por exemplo, focou na otimização do consumo de combustível da embarcação, considerando não apenas o esforço mínimo para uma tarefa, mas também o porte dos atuadores e o custo energético associado. Já Ding et al. (2024) apresentaram um método que minimiza simultaneamente o erro de alocação, o consumo de potência e o desgaste mecânico, diferenciando-se por otimizar três fatores relevantes de forma integrada.

2.3.6 Controlador

Com o sistema dinâmico devidamente modelado conforme Seção 2.2, torna-se possível estabelecer os passos necessários para o projeto do controlador.

Segundo Fossen e Fjellstad (1995), o seguinte problema se apresenta:

> Para manipuladores robóticos, é comum transformar a trajetória de estado desejada para as coordenadas do espaço das juntas, aplicando a cinemática inversa. Dessa forma, a matriz de transformação cinemática pode ser evitada no esquema de controle. Isso não é possível para veículos marítimos, pois a integral
>
> $$\int_0^t \nu \, d\tau$$
>
> não possui uma interpretação física.

Dessa forma, como não é possível aplicar diretamente a cinemática inversa em sistemas marítimos, adota-se uma abordagem alternativa baseada na utilização da matriz Jacobiana de transformação. Essa matriz permite relacionar as velocidades medidas no referencial do corpo, representadas por $\nu = [u, v, w, p, q, r]^\top$, com as derivadas do vetor de

posição e orientação no referencial inercial, expresso por $\eta = [x, y, z, \phi, \theta, \psi]^\top$. As variáveis no referencial inercial são formuladas com base no sistema de coordenadas NED (*North-East-Down*). Dessa maneira, é possível expressar as equações cinemáticas de forma geral como:

$$\dot{\eta} = J(\eta)\,\nu \tag{2.39}$$

Neste contexto, η representa o vetor de posição e orientação da embarcação em relação ao referencial inercial (geralmente associado à Terra), enquanto ν é o vetor de velocidades lineares e angulares no referencial do corpo (ligado à embarcação). A matriz $J(\eta)$ é uma matriz de transformação que depende dos ângulos de Euler e é responsável por realizar a conversão entre os referenciais, conforme definido por Fossen (1994).

Dentre as diversas técnicas de controle, destaca-se o uso do controlador PID não linear, uma abordagem amplamente discutida na literatura especializada para sistemas de múltiplas entradas e múltiplas saídas (MIMO). Essa técnica é apresentada nas obras de Fossen (2011), Alme (2008) e DeRensis (2013), adotando a seguinte estrutura para a lei de controle:

$$\tau = \mathbf{g}(\eta) + \mathbf{J}^\top(\eta)\,\tau_{\text{PID}} \tag{2.40}$$

Nessa equação, $\mathbf{g}(\eta)$ representa os termos associados às forças restauradoras, como peso e empuxo, enquanto τ_{PID} corresponde ao esforço de controle fornecido pelo controlador PID, definido por:

$$\tau_{\text{PID}} = -\mathbf{K}_p\,\tilde{\eta} - \mathbf{K}_d\,\dot{\eta} - \mathbf{K}_i \int_0^t \tilde{\eta}(\tau)\,d\tau \tag{2.41}$$

O vetor de erro $\tilde{\eta}$ é definido como:

$$\tilde{\eta} = \eta - \eta_d \tag{2.42}$$

sendo η_d o vetor que representa a posição e a orientação desejadas, e η os valores atuais medidos.

Em Fossen (2021), o autor propõe um algoritmo para cálculo dos ganhos, baseado na definição prévia da matriz de banda $\Omega_b = \text{diag}(\omega_{b1}, \ldots, \omega_{b6}) > 0$ e dos coeficientes de amortecimento $\mathbf{Z} = \text{diag}(\zeta_1, \ldots, \zeta_6) > 0$, para cada grau de liberdade. A partir desses valores, os ganhos proporcional, derivativo e integral são calculados de modo a satisfazer critérios de desempenho (sobressinal, tempo de acomodação e tempo de subida) previamente estabelecidos.

A frequência natural é dada por:

$$\omega_n = \frac{1}{\sqrt{1 - 2\zeta^2 + \sqrt{4\zeta^4 - 4\zeta^2 + 2}}}\,\omega_b \tag{2.43}$$

E, portanto, ganhos do controlador são calculados conforme:

$$\mathbf{K}_p = \mathbf{M}(\eta)\,\omega_n^2 \tag{2.44}$$

$$\mathbf{K}_d = 2\,\mathbf{M}(\eta)\,\mathbf{Z}\,\omega_n \tag{2.45}$$

$$\mathbf{K}_i = \frac{1}{10}\mathbf{K}_p\,\omega_n \tag{2.46}$$

Em sistemas de posicionamento dinâmico, devido à natureza de baixas velocidades envolvidas, é admissível considerar um modelo com amortecimento linear. Conforme mencionado anteriormente, esse tipo de simplificação é válida pois, em operações típicas de DP, a embarcação não realiza manobras bruscas, o que reduz a influência de termos não lineares da hidrodinâmica.

Além disso, é comum adotar um modelo em apenas três graus de liberdade, considerando apenas os movimentos de *surge*, *sway* e *yaw* ($\eta = [x, y, \psi]^\top$), visto que o objetivo principal é controlar a posição horizontal da embarcação em determinada coordenada geográfica. Movimentos como *roll*, *pitch* e *heave* são geralmente negligenciados nesses modelos, pois têm pouca influência direta no posicionamento horizontal. No entanto, é importante destacar que, sob condições de mar severamente agitado, essas aproximações podem se tornar inválidas, comprometendo o desempenho ou até a viabilidade do sistema de posicionamento dinâmico.

Outra simplificação frequentemente adotada é a consideração de que o termo de Coriolis, representado pela matriz $\mathbf{C}(\nu)$, possui contribuição desprezível no regime de operação do DP. Assim, assume-se que seus efeitos podem ser compensados pela ação integrativa do controlador, dispensando sua inclusão explícita no modelo de controle.

Com base nessas premissas, o modelo dinâmico do sistema pode ser simplificado da seguinte forma:

$$\dot{\eta}_p = \nu \tag{2.47}$$

$$\mathbf{M}\dot{\nu} + \mathbf{D}\nu = \mathbf{R}(\psi)\mathbf{b} + \tau + \tau_{\text{wind}} + \tau_{\text{wave}} \tag{2.48}$$

$$\dot{\mathbf{b}} = \mathbf{0} \tag{2.49}$$

Onde M representa a matriz de inércia e D os termos dissipativos associados a atrito e amortecimento, ambas dependentes de v, que corresponde ao vetor de velocidades lineares e angulares da embarcação. Nesse contexto, $\mathbf{b}$ é tratado como um vetor de viés lentamente variante, expresso no referencial $\{n\}$, utilizado para representar a influência das correntes oceânicas sobre o movimento da embarcação.

Em sistemas restritos a 3 DOF, a matriz de transformação $\mathbf{J}(\eta)$, anteriormente utilizada para converter entre referenciais em 6 DOF, é substituída pela matriz de rotação $\mathbf{R}(\psi)$. Dessa forma, a relação entre os referenciais pode ser representada como:

$$\eta_p = \mathbf{R}^\top(\psi)\eta \tag{2.50}$$

Com isso, a formulação do controle pode ser expressa de maneira simplificada por:

$$\tau = \mathbf{B}\mathbf{u} \tag{2.51}$$

O vetor $\mathbf{u}$ representa as entradas de controle aplicadas a cada atuador, Equação 2.33. $\mathbf{B}$ é a matriz de configuração dos propulsores, definida como o produto das matrizes $\mathbf{T}$ e $\mathbf{K}$, portanto:

$$\mathbf{B} = \mathbf{T} \cdot \mathbf{K} \tag{2.52}$$

Nessa relação, $\mathbf{K}$ é uma matriz diagonal que contém os coeficientes de propulsão individuais de cada atuador, enquanto $\mathbf{T}$ descreve o arranjo físico dos atuadores ao longo do casco da embarcação, considerando suas posições e orientações.

3 Paralelo Entre Teoria e Simulador

Este capítulo tem como objetivo estabelecer um paralelo entre o referencial teórico apresentado no capítulo anterior e a implementação prática realizada por meio do *PythonVehicleSimulator*. O referido simulador, disponível publicamente no repositório online Fossen (2025), foi originalmente concebido como um suplemento ao *Marine Systems Simulator* (MSS), uma biblioteca desenvolvida para o ambiente *Matlab/Simulink* com foco na modelagem e controle de sistemas marítimos.

Embora o *Matlab* seja amplamente utilizado na indústria e no meio acadêmico, oferecendo uma vasta gama de ferramentas para projetos e ajustes de modelos e controladores, a linguagem de programação *Python* tem ganhado popularidade por ser mais acessível, de código aberto e amplamente difundida entre pesquisadores e desenvolvedores. Por esse motivo, torna-se relevante compreender como foi realizada a adaptação do simulador para *Python*, bem como explorar as possibilidades de complementação e expansão da biblioteca com o objetivo de torná-la uma ferramenta mais robusta e amplamente aplicável à simulação de sistemas de embarcações *offshore*.

O código do simulador apresenta uma estrutura bem organizada, com comentários explicativos que indicam as etapas sendo executadas e algumas referências à teoria apresentada no livro *Handbook of Marine Craft Hydrodynamics and Motion Control* (FOSSEN, 2011). No entanto, essas referências são feitas de forma ampla, remetendo à teoria completa, o que pode dificultar a compreensão por parte de leitores leigos ou entusiastas, uma vez que se trata de um conteúdo técnico relativamente complexo.

Com o objetivo de facilitar a consulta, as funções do código-fonte estão apresentadas nos Anexos, conforme apresentado a seguir:

- Anexo A - `main.py`
- Anexo B - `mainLoop.py`
- Anexo C - `supply.py`
- Anexo D - `gnc.py`
- Anexo E - `control.py`

3.1 Inicialização e Ajuste de Referências

O código do simulador requer o uso da biblioteca `NumPy`, amplamente utilizada para operações com vetores, matrizes e álgebra linear. Dessa forma, a instalação dessa

biblioteca é um pré-requisito para a execução do simulador.

A rotina principal é implementada no arquivo `main.py`, por meio do qual é possível selecionar diferentes cenários de simulação previamente configurados. Para o presente trabalho, será utilizada a opção denominada *supply vessel*, que representa uma embarcação de suporte de médio porte, comumente empregada no transporte de cargas para plataformas *offshore*.

Ao selecionar o modo de simulação, é possível inicializar algumas variáveis importantes por meio da função:

```
supply('DPcontrol', x_d, y_d, psi_d, V_c, beta_c)
```

Onde:

- `x_d`: posição final desejada em X (m);
- `y_d`: posição final desejada em Y (m);
- `psi_d`: ângulo de guinada (*yaw*) final desejado (em graus);
- `V_c`: velocidade da corrente (m/s);
- `beta_c`: direção da corrente (ângulo de guinada em graus).

Com o objetivo de garantir que a simulação seja executada de forma síncrona em todos os módulos do código, reproduzindo o comportamento de um sistema digitalizado real, são utilizadas duas variáveis principais: `sampleTime` e `N`. A variável `sampleTime` define o intervalo de tempo entre cada ciclo de cálculo (ou amostragem), enquanto `N` define o número total de amostras a serem simuladas. Por padrão, o autor estabelece um tempo de amostragem de 0,02 segundos e um total de 10.000 amostras. Esses parâmetros, no entanto, podem ser ajustados conforme a necessidade da aplicação.

Cada amostra corresponde a uma iteração do laço de simulação, em que cálculos são realizados de forma sequencial. No presente capítulo, será acompanhada uma dessas sequências de cálculo do início ao fim, com o objetivo de compreender seu funcionamento e estabelecer um paralelo com a teoria apresentada anteriormente.

Cada iteração do laço de simulação, ou seja, cada amostra computada, utiliza a função `Dpcontrol`, responsável por simular a dinâmica da embarcação considerando três graus de liberdade (*3 DOF*). Ao ser chamada, essa função inicia previamente algumas variáveis essenciais para o funcionamento do sistema, tais como:

```
self.ref = np.array([r_x, r_y, r_n * D2R], float)
```

```
self.V_c = V_current
self.beta_c = beta_current * D2R
self.controlMode = controlSystem
```

Essas linhas atribuem valores aos atributos do objeto (`self`), que serão usados ao longo da simulação. O vetor `self.ref` armazena a posição e orientação desejadas (converte-se o ângulo de graus para radianos com `D2R`, previamente definido como $\pi/180$), enquanto `self.V_c` e `self.beta_c` representam a velocidade e direção da corrente marítima. Por fim, `self.controlMode` define qual sistema de controle será utilizado, no caso `"Dpcontrol"`, que direcionará para a função específica.

Posteriormente, são definidas as características físicas do modelo da embarcação simulada:

```
m = 6000.0e3
self.L = 76.2
self.T_n = 1.0
self.n_max = np.array([250, 250, 250, 250, 160, 160], float)
self.nu = np.array([0, 0, 0, 0, 0, 0], float)
self.u_actual = np.array([0, 0, 0, 0, 0, 0], float)
```

A massa total da embarcação é definida como 6.000 toneladas (através da variável m), enquanto o comprimento é especificado por `self.L`, igual a $76,2$ metros. A constante de tempo dos propulsores (`self.T_n`) é definida como 1 segundo, representando a inércia na resposta dos atuadores.

A embarcação simulada conta com seis propulsores: quatro *tunnel thrusters* (dois localizados na proa e dois na popa) e dois *main propellers*. As velocidades iniciais desses atuadores são definidas pelo vetor `self.nu`, enquanto os valores atualizados a cada iteração são armazenados em `self.u_actual`. O vetor `self.n_max` determina os limites máximos operacionais de cada propulsor, considerando que os diferentes atuadores possuem capacidades e finalidades distintas a bordo da embarcação. Esses limites são definidos em rotações por minuto (RPM).

A seguir, é criada a matriz diagonal $\mathbf{K}$, que representa o coeficiente de empuxo de cada propulsor, conforme exemplificado na Equação 2.32 deste trabalho. Observa-se que os propulsores principais apresentam uma geração de força significativamente maior quando comparados aos atuadores de manobra:

```
K = np.diag([3.2, 3.2, 3.2, 3.2, 31.2, 31.2])
```

De forma análoga, é definida a matriz **T**, que contém a configuração espacial dos atuadores no casco da embarcação, indicando em quais movimentos cada um exerce influência, conforme detalhado na Equação 2.34:

```
T = np.array([ [0, 0, 0, 0, 1, 1], [1, 1, 1, 1, 0, 0],
[30, 22, -22, -30, -8, 8] ], float)
```

A multiplicação das matrizes **T** e **K** gera a matriz **B**, conforme apresentada na Equação 2.52, que representa a contribuição de força de cada propulsor nos diferentes graus de liberdade nos quais atuam:

```
self.B = T @ K
```

3.2 Modelo Matemático

Um modelo dependente da velocidade para uma embarcação do tipo *supply vessel*, com comprimento previamente conhecido, pode ser derivado a partir dos dados apresentados por Fossen, Sagatun e Sørensen (1996). Esses dados foram obtidos por meio de ensaios em mar calmo, realizados com a embarcação a uma velocidade de $0,2\,\mathrm{m/s}$, condições ideais para a operação com posicionamento dinâmico.

Com base nesses experimentos e na fundamentação teórica apresentada na Seção 2.2, foi possível definir o modelo hidrodinâmico da embarcação, representado pelas matrizes de inércia e de amortecimento hidrodinâmico, respectivamente:

```
Mbis = np.array([[1.1274, 0, 0], [0, 1.8902, -0.0744],
[0, -0.0744, 0.1278]], float)

Dbis = np.array([[0.0358, 0, 0], [0, 0.1183, -0.0124],
[0, -0.0041, 0.0308]], float)

Tbis_inv = np.diag([1.0, 1.0, self.L])
```

Vale ressaltar que, em situações de manobras em baixas velocidades ou quando o objetivo é apenas a manutenção da posição da embarcação, o modelo dinâmico pode ser simplificado, conforme Equação 2.48. Nesses casos, utilizam-se apenas as matrizes de inércia e amortecimento, desprezando os efeitos de acelerações relativas mais complexas. Cabe destacar também que, no modelo adotado, não se considera o conceito de massa adicionada, como discutido na Seção 2.2, focando-se apenas nas características dinâmicas da embarcação em si.

As matrizes de inércia e amortecimento (**M3** e **D3**) são calculadas com base no sistema adimensional *bis*, conforme descrito por Fossen, Sagatun e Sørensen (1996)), seguindo as seguintes relações:

$$\mathbf{M} = m \cdot T^{-2} \left(T \mathbf{M}'' T^{-1} \right) \tag{3.1}$$

$$\mathbf{D} = m \cdot \sqrt{\frac{g}{L}} \cdot T^{-2} \left(T \mathbf{D}'' T^{-1} \right) \tag{3.2}$$

Considera-se que $T = \text{diag}\{1, 1, L\}$, em que L representa o comprimento da embarcação, g é a aceleração gravitacional, m corresponde à massa do veículo, e $\mathbf{M}''$ e $\mathbf{D}''$ são as matrizes de inércia e amortecimento no sistema *bis* não dimensionado. Dessa forma, é possível realizar a transformação do sistema *bis* para o modelo que representa o sistema físico da embarcação. Assim, são realizados os cálculos a partir do modelo obtido nos ensaios para gerar uma matriz de inércia e amortecimento tridimensionais, com o objetivo de preparar as variáveis para a rotina de alocação de polos do controlador:

```
self.M3 = m * Tbis_inv @ Mbis @ Tbis_inv

self.M3inv = np.linalg.inv(self.M3)

self.D3 = m * math.sqrt(g / self.L) * Tbis_inv @ Dbis @ Tbis_inv
```

A matriz `self.M3` representa a matriz de inércia tridimensional ajustada, considerando o formato físico da embarcação e sua massa total. Já `self.M3inv` corresponde à sua inversa, que será utilizada nos cálculos de alocação de polos do sistema de controle, permitindo a relação direta entre os torques aplicados e as acelerações resultantes do sistema. Por fim, a matriz `self.D3` representa a matriz de amortecimento ajustada, refletindo a resistência hidrodinâmica realista da embarcação, na qual o fator gravitacional assegura a proporcionalidade com a escala física do sistema.

3.3 Controlador e Alocação de Polos

Para o ajuste do controlador, conforme indicado na Seção 2.3.6, é necessária a elaboração dos vetores `zeta` e `wn`, os quais representam o comportamento dinâmico desejado que o sistema de controle deve impor sobre as variáveis controladas:

```
self.wn = np.diag([0.1, 0.1, 0.2])
self.zeta = np.diag([1.0, 1.0, 1.0])
```

A matriz `self.wn` define as frequências naturais desejadas para cada modo de movimento (*surge*, *sway* e *yaw*), enquanto `self.zeta` define o amortecimento crítico para cada um desses modos, garantindo estabilidade e suavidade na resposta dinâmica.

O simulador utilizado foi desenvolvido para representar uma ampla gama de operações marítimas, não se limitando apenas ao posicionamento dinâmico. Por esse motivo, as variáveis do sistema consideram os seis graus de liberdade (6 DOF) do movimento da embarcação. No entanto, ao aplicar o sistema de controle específico para DP, é necessário restringir a análise aos três graus de liberdade principais (*surge*, *sway* e *yaw*) que são os mais relevantes para o controle horizontal da posição e orientação.

Dessa forma, são definidos os vetores `eta3` e `nu3`, que representam, respectivamente, a posição e atitude (η) e as velocidades (ν) restritas aos três DOF de interesse:

```
eta3 = np.array([eta[0], eta[1], eta[5]])
nu3 = np.array([nu[0], nu[1], nu[5]])
```

De posse do modelo reduzido com três graus de liberdade e da representação das frequências naturais desejadas para o sistema, torna-se possível prosseguir com o cálculo dos ganhos do controlador, conforme Equação 2.44, Equação 2.45 e Equação 2.46.

```
M3_diag = np.diag(np.diag(M3))
D3_diag = np.diag(np.diag(D3))

Kp = wn @ wn @ M3_diag
Kd = 2.0 * zeta @ wn @ M3_diag - D3_diag
Ki = (1.0 / 10.0) * wn @ Kp
```

Como parte da implementação de um sistema de controle PID, é necessário identificar o erro atual do sistema em relação ao ponto de referência, uma vez que esse erro influencia diretamente a equação de controle adotada. Para tanto, utiliza-se a seguinte relação:

```
e = eta3 - np.array([x_d, y_d, psi_d])
e[2] = ssa(e[2])
```

Nesse cálculo, o vetor de erro `e` representa a diferença entre a posição e orientação atuais da embarcação (armazenadas em `eta3`) e os valores desejados ou de referência (`x_d`, `y_d` e `psi_d`). A terceira componente, que corresponde ao erro angular em *yaw* (ψ), é ajustada pela função `ssa` (*Smallest Signed Angle*), a qual normaliza esse valor para o intervalo $[-\pi, \pi)$. Tal normalização é realizada para evitar o cálculo de ângulos fora desse

intervalo. O valor ajustado do ângulo é, então, utilizado para calcular a matriz de rotação $\mathbf{R}$, por meio da função:

```
R = Rzyx(0.0, 0.0, eta3[2])
```

Fundamentado na Equação 2.50, essa matriz é necessária para converter as variáveis do sistema entre os referenciais do corpo e do mundo. No caso de sistemas restritos a três graus de liberdade (movimentos em *surge*, *sway* e *yaw*), a matriz de rotação pode ser simplificada, considerando apenas o ângulo de yaw, pois os demais ângulos (*roll* e *pitch*) são irrelevantes no cenário aplicado.

Por fim, a lei de controle PID é aplicada por meio da seguinte equação:

```
tau = ( - np.matmul((R.T @ Kp), e) - np.matmul(Kd, nu3)
- np.matmul((R.T @ Ki), e_int) )
```

Observa-se que o termo integrativo é calculado ao final de cada iteração do laço de controle, utilizando o método de integração de Euler, que considera a taxa de amostragem do sistema e o erro atual acumulado. Dessa forma, garante-se que a ação integral represente de forma contínua a soma dos erros ao longo do tempo, contribuindo para a eliminação do erro em regime permanente, conforme esperado do controlador PID.

```
e_int += sampleTime * e
```

Com o vetor de controle τ definido, aplica-se a estratégia de alocação de empuxo, cujo objetivo é determinar a contribuição individual de cada propulsor necessária para produzir a força e o momento resultantes requeridos pelo sistema. A fundamentação teórica sobre este assunto foi abordada na Seção 2.3.5, que discute a alocação de empuxo, e mais especificamente na Seção 2.3.5.2, que trata da solução de otimização.

```
B_pseudoInv = self.B.T @ np.linalg.inv(self.B @ self.B.T)
u_alloc = np.matmul(B_pseudoInv, tau3)
```

Utiliza-se a matriz `B`, calculada anteriormente, para o cálculo de sua *pseudoinversa*, o que permite, por meio do método dos mínimos quadrados, determinar a alocação de empuxo.

No sistema proposto para representar propulsores do meio naval, o empuxo gerado varia de forma quadrática com a sua rotação, Equação 2.33. Para simplificar o cálculo de alocação de controle, essa relação é linearizada por meio de uma variável auxiliar que representa o valor quadrático da rotação, facilitando a utilização de métodos como

a *pseudoinversa* para distribuir as forças desejadas entre os atuadores. Após a alocação, essa variável precisa ser convertida de volta para o comando físico de rotação, processo que envolve extrair a raiz quadrada e preservar o sinal, garantindo que o sentido e a intensidade do empuxo sejam mantidos. Essa abordagem é descrita por Fossen (2011, Seção 12.3.5, *Case Study: DP Control Allocation System*) no contexto de modelagem e controle de embarcações com propulsores de comportamento não linear.

```
n = np.zeros(self.dimU)
for i in range(0, self.dimU):
    n[i] = np.sign(u_alloc[i]) * math.sqrt(abs(u_alloc[i]))
u_control = n
```

Como mencionado anteriormente, o sistema de DP é projetado principalmente para realizar pequenas manobras ou manter a posição atual da embarcação. Por esse motivo, é indesejável que os *setpoints* (referências) sejam aplicados de tal forma que o que poderia demandar forças abruptas dos atuadores, como longas distâncias.

Para evitar esse comportamento indesejado e acúmulo do erro integrativo, aplica-se uma suavização dos *setpoints* utilizando o método de Euler, que permite aproximar da sua derivada. Essa abordagem permite que os valores de referência comecem a partir do estado atual da embarcação que, na simulação, é a origem $(0, 0, 0)$ e se aproximem gradualmente dos valores desejados.

A suavização é implementada da seguinte forma:

```
T = 5.0 * np.array([1 / wn[0][0], 1 / wn[1][1], 1 / wn[2][2]])
x_d += sampleTime * (eta_ref[0] - x_d) / T[0]
y_d += sampleTime * (eta_ref[1] - y_d) / T[1]
psi_d += sampleTime * (eta_ref[2] - psi_d) / T[2]
```

Nota-se que a taxa com que os *setpoints* se aproximam dos valores desejados é determinada pelo vetor $\mathbf{T}$, o qual depende diretamente das frequências naturais ω_n. Esses parâmetros são definidos conforme os requisitos de desempenho estabelecidos no projeto de controle, assegurando uma transição gradual e segura rumo à referência final. No entanto, mesmo com essa técnica de suavização implementada, a definição de *setpoints* incompatíveis com a filosofia de operação dos sistemas de posicionamento dinâmico pode levar à geração de esforços excessivos pelos atuadores, ocasionando instabilidades, oscilações indesejadas ou perda de desempenho.

3.4 Dinâmica da Embarcação

Até o momento, o código forneceu ferramentas para modelar o comportamento da embarcação, realizar o controle e alocar o empuxo. No entanto, é necessário compreender como essas ações afetam a dinâmica do navio e como os dados são processados. Para isso, o resultado da alocação de empuxo é encaminhado para a função denominada `dynamics`, que inicia com a decomposição da velocidade da corrente que atua sobre a embarcação:

```
u_c = self.V_c * math.cos(self.beta_c - eta[5])
v_c = self.V_c * math.sin(self.beta_c - eta[5])
nu_c = np.array([u_c, v_c, 0, 0, 0, 0], float)
nu_r = nu - nu_c
```

O autor subtrai da velocidade da embarcação a influência da corrente marítima, que, nesta simulação, afeta apenas dois graus de liberdade: *surge* e *sway*. Dessa forma, obtém-se a velocidade relativa, ou seja, a velocidade real do casco da embarcação em relação à massa de água, e não apenas aquela esperada a partir do comando dos propulsores.

Em seguida, verifica-se e limita-se a atuação dos propulsores, assegurando que não excedam seus limites físicos de saturação. Por meio da variável `tau3`, obtém-se o vetor de forças e momentos resultantes a partir da multiplicação da matriz $\mathbf{B}$, que indica o coeficiente de empuxo de cada propulsor, com o vetor `n_squared`, que contém os valores de rotação dos propulsores após a aplicação da saturação.

```
n_squared = np.zeros(self.dimU)
for i in range(0, self.dimU):
    n[i] = sat(n[i], -self.n_max[i], self.n_max[i])
    n_squared[i] = abs(n[i]) * n[i]
tau3 = np.matmul(self.B, n_squared)
```

A simulação utiliza as velocidades da embarcação nos movimentos *surge*, *sway* e *yaw* para calcular a aceleração (`nu3_dot`), conforme as seguintes linhas de código:

```
nu3_r = np.array([nu_r[0], nu_r[1], nu_r[5]])
nu3_dot = np.matmul(self.M3inv, tau3 - np.matmul(self.D3, nu3_r))
nu_dot = np.array([nu3_dot[0], nu3_dot[1], 0, 0, 0, nu3_dot[2]])
n_dot = (u_control - u_actual) / self.T_n
```

Observa-se que é empregada a velocidade relativa previamente determinada, a qual representa o deslocamento real do casco em relação à água, desconsiderando a influência

da corrente. Em seguida, a aceleração obtida é expandida para um vetor de seis graus de liberdade (6 DOF), sendo os componentes não utilizados preenchidos com zeros:

```
nu_dot = np.array([nu3_dot[0], nu3_dot[1], 0, 0, 0, nu3_dot[2]])
n_dot = (u_control - u_actual) / self.T_n
```

Esse procedimento permite manter a estrutura do modelo compatível com sistemas que operam em 6 DOF, ainda que apenas três componentes (*surge*, *sway* e *yaw*) sejam utilizados no controle de posicionamento dinâmico. Essa abordagem visa tornar a simulação mais flexível e aplicável a diferentes tipos de embarcações ou cenários operacionais.

Como proposto por Blanke, Lindegaard e Fossen (2000), a modelagem dos propulsores pode ser representada como um sistema de primeira ordem, com tempo de resposta característico T_n, conforme a equação:

$$T_n \dot{n} + n = n_d \tag{3.3}$$

Essa formulação descreve a dinâmica dos atuadores, representando a resposta não instantânea dos propulsores aos comandos de controle.

A partir das derivadas temporais de posição e velocidade, as variáveis de estado são atualizadas a cada intervalo de amostragem utilizando o método explícito de Euler, permitindo uma integração numérica computacionalmente eficiente. Assim, a iteração é finalizada e a seguinte é iniciada, repetindo-se o procedimento até que o número total de amostras seja processado e o tempo de simulação seja atingido:

```
nu = nu + sampleTime * nu_dot
n = n + sampleTime * n_dot
```

3.5 Resultados e Análise da Simulação

O simulador analisado incorpora mecanismos de aquisição de dados ao longo de toda a execução, possibilitando, ao final do processo, a geração de gráficos que descrevem o comportamento dinâmico da embarcação. Esses resultados permitem uma análise detalhada da trajetória nos eixos x e y, bem como da variação do ângulo de guinada (*yaw*) em cada iteração do laço de controle, conforme ilustrado nas figuras Figura 10 e Figura 11. Para obter o primeiro resultado, foi utilizada a configuração padrão do simulador, com *setpoint* em $x_d = 4.0$, $y_d = 4.0$ e $\psi_d = 50°$, considerando ainda a correnteza com intensidade $V_c = 0.5$ e direção $\beta_c = 20°$. Foram coletadas 10000 amostras e usando o tempo de amostragem de 0,02 segundos, gerando 200 segundos de simulação.

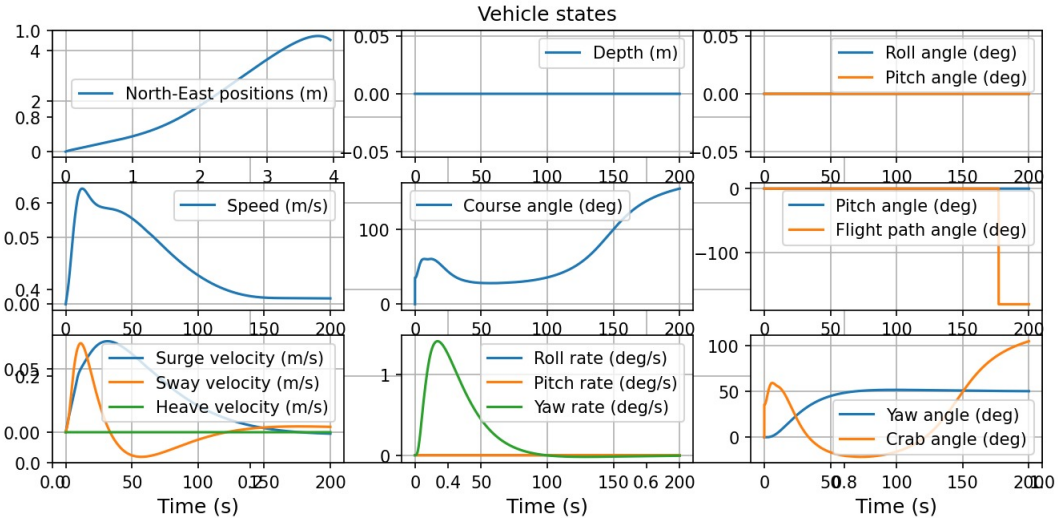

Figura 10 – Comportamento da embarcação ao longo da simulação

Fonte: Elaboração própria, a partir dos resultados da simulação.

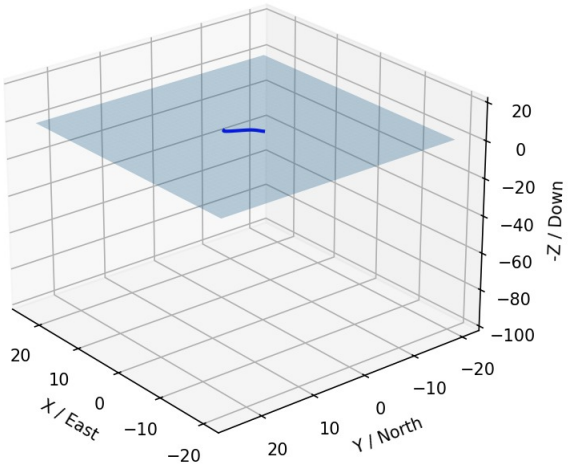

Figura 11 – Trajetória da embarcação nos três eixos do sistema de referência NED (*North-East-Down*).

Fonte: Elaboração própria, a partir dos resultados da simulação.

Além disso, é possível observar a distribuição de empuxo entre os diferentes atuadores, identificando o instante de acionamento e a intensidade de cada contribuição, conforme Figura 12. Essa visualização é fundamental para avaliar a eficiência da estratégia de alocação de empuxo e verificar se o sistema atende aos requisitos de estabilidade, precisão e consumo energético estabelecidos no projeto.

A análise conjunta das variáveis de posição, orientação e atuação dos propulsores fornece dados para validar o modelo de controle implementado, identificar possíveis pontos de otimização e assegurar que o comportamento obtido em simulação esteja em

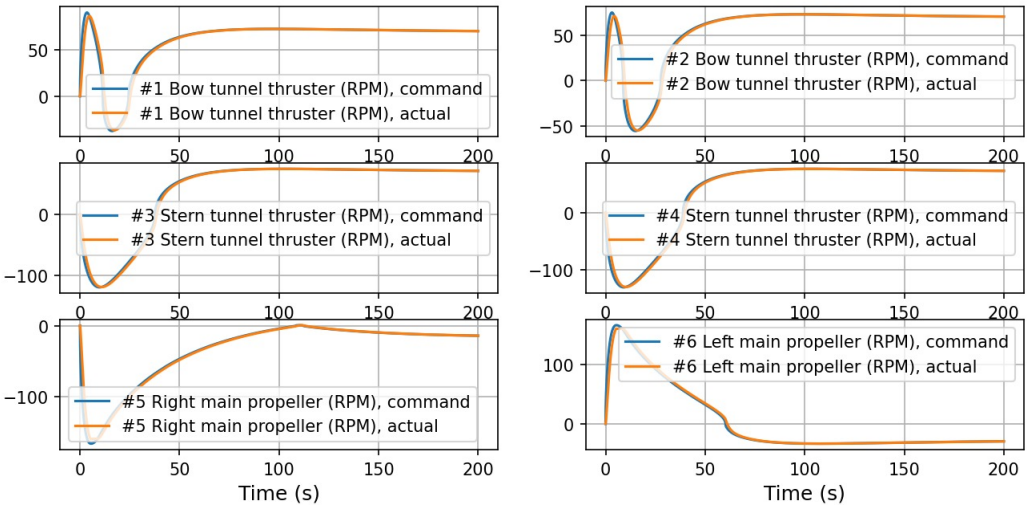

Figura 12 – Atuação dos propulsores ao longo da simulação

Fonte: Elaboração própria, a partir dos resultados da simulação.

conformidade com os objetivos do sistema de posicionamento dinâmico.

Nos resultados apresentados anteriormente, observa-se que a embarcação não atingiu o seu *setpoint* no tempo previsto para a simulação, como evidenciado pelo gráfico *Nort-East positions* contido na Figura 10. Dessa forma, duplicou-se o número de amostras de 10000 para 20000 porém o tempo de amostragem continuou 0,02 segundos, correspondendo a 400 s de simulação, e reduziu-se a velocidade da corrente de 0,5 m/s para 0,2 m/s. Com esses ajustes, verificou-se que, durante o intervalo de simulação, o objetivo foi alcançado e os propulsores permaneceram acionados apenas para contrabalançar a ação da corrente, Figura 13.

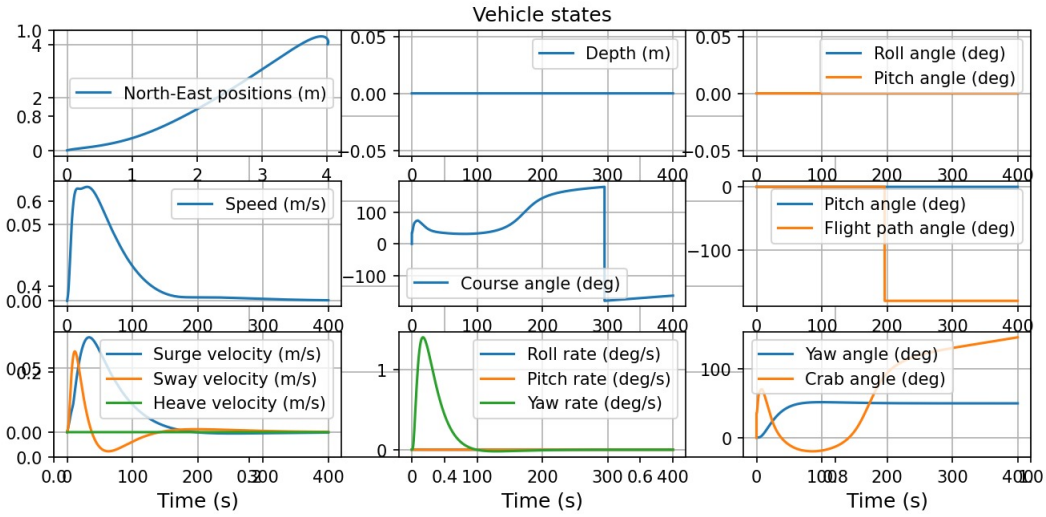

Figura 13 – Comportamento da embarcação, após ajustes

Fonte: Elaboração própria, a partir dos resultados da simulação.

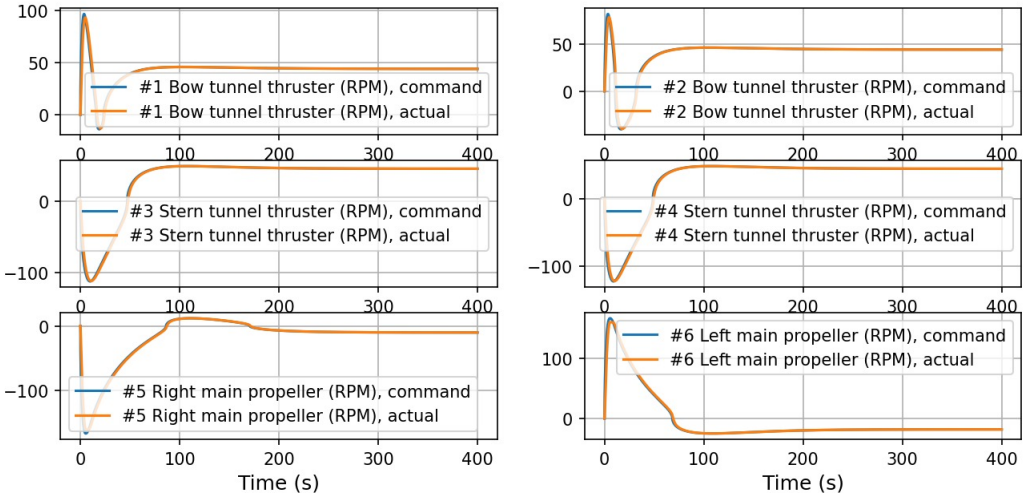

Figura 14 – Atuação dos propulsores ao longo da simulação, após ajustes

Fonte: Elaboração própria, a partir dos resultados da simulação.

4 Resultados e Discussões

O simulador implementado em *Python*, ao contrário do *MSS Toolbox*, apresenta uma implementação mais enxuta, com menos ferramentas integradas para uma simulação completa. Em termos teóricos, foram considerados controle com *Wind Feedforward*, ação integral e observadores para estimativa de estados; entretanto, na implementação, especialmente no contexto do sistema de DP, esses recursos não foram utilizados explicitamente. Parte dessa decisão deve-se às limitações inerentes à simulação computacional, sobretudo à dificuldade de inserir entradas que reproduzam fielmente as condições reais de um ambiente marítimo, cujo comportamento é caótico.

Mesmo com tais limitações, o simulador fornece uma base útil para a incorporação de novas ferramentas no futuro. A corrente oceânica considerada foi assumida constante ao longo do tempo de simulação; entretanto, considerando que o tempo de simulação é relativamente curto, é plausível que, na realidade, a corrente apresente variações. Assim, para avaliar o desempenho do controlador em um ambiente mais realista, seria interessante implementar variações aleatórias na direção e intensidade da corrente. Adicionalmente, efeitos de vento e rajadas não foram modelados, o que reduz a fidelidade do cenário em relação às condições reais de operação.

Adicionalmente, é notável que, por se tratar de um sistema de controle complexo, envolvendo diversos módulos de controle e modelagem, a apresentação prática dos dados é fundamental para auxiliar na elaboração de novas soluções e na análise de problemas. Embora os gráficos gerados sejam de grande utilidade, o simulador ainda carece de recursos visuais mais expressivos, como representações da força resultante dos propulsores após a otimização da alocação de empuxo, bem como gráficos em maior tamanho e resolução, que permitam compreender de forma mais clara o comportamento do sistema.

O controlador adotado para o problema de DP foi do tipo PID, o qual demonstrou robustez e desempenho satisfatório nas simulações realizadas. Embora na primeira simulação o sistema não tenha atingido o *setpoint*, tal resultado se deveu mais à limitação do tempo de simulação do que a falhas do controlador em si. Já na segunda tentativa, além da redução da intensidade da corrente marítima, fator que por si só já favorece a estabilização da embarcação, a ampliação do tempo de simulação garantiu uma resposta adequada. Para fins didáticos e experimentais, entretanto, o simulador se beneficiaria da inclusão de estratégias de controle alternativas. Métodos como o controle por modo deslizante (SMC, *Sliding Mode Control*), o controle H^{∞} e o controle preditivo baseado em modelo (MPC, *Model Predictive Control*) são amplamente explorados na literatura sobre embarcações marítimas e poderiam fornecer subsídios relevantes para a comparação dos

resultados obtidos com diferentes abordagens de controle.

Em resumo, apesar das limitações apontadas, o simulador em *Python* mostrou-se uma ferramenta útil na emulação do comportamento das embarcações estudadas e cumpre seu objetivo de suplementar o *MSS Toolbox* de forma simplificada, acessível e reprodutível para pesquisadores e estudantes. Além disso, promove uma aplicação da teoria de modelagem, controle e alocação de empuxo, evidenciando explicitamente, no código, as formulações apresentadas na literatura técnica.

5 Considerações Finais e Perspectivas de Trabalhos Futuros

Durante este trabalho, a fundamentação teórica foi desenvolvida de forma separada da análise do simulador, permitindo maior liberdade na busca por informações e modelos já bem estabelecidos. Dessa maneira, após a finalização da análise comparativa, foi possível observar que o simulador ainda apresenta grande potencial de evolução. Ainda assim, a elaboração desses conceitos mostrou-se fundamental para o objetivo deste trabalho, que consiste em identificar como tais recursos são aplicados na prática e de que forma podem ser utilizados para aprimorar o simulador em questão.

O presente trabalho destacou a importância da simulação em sistemas de controle, não apenas como ferramenta para validação e testes, mas também como recurso didático e educacional essencial. Muitas vezes, a teoria por trás dos sistemas de controle apresenta-se complexa e de difícil compreensão; contudo, a implementação prática por meio de um simulador possibilita visualizar e compreender o funcionamento dos conceitos isolados, bem como a forma como se inter-relacionam para compor um sistema completo.

Esse entendimento, possibilitado pela utilização do simulador, facilita o aprofundamento no tema, permitindo o desenvolvimento de novas teorias e a proposição de soluções mais eficazes para desafios futuros.

Dessa forma, este trabalho de conclusão de curso pode servir como um guia introdutório ao estudo da teoria e da implementação do simulador, constituindo uma base valiosa para futuras pesquisas e aplicações na área. Algumas sugestões incluem:

- Projetar um Processor-in-the-loop (PIL), executando o código de controle em um ambiente separado, representando o controlador real, enquanto sensores, atuadores e a dinâmica da embarcação permanecem simulados;
- Desenvolver um Hardware-in-the-loop (HIL), utilizando o modelo simulado da embarcação para testar sensores e atuadores reais, possibilitando, por exemplo, o desenvolvimento de sensores de vento, corrente oceânica e posição, bem como o controle da intensidade e direção de atuadores;
- Implementar outras estratégias de controle consolidadas na literatura para eventual comparação com o controlador PID;
- Simular a força do vento e implementar controle por meio de *Wind Feedforward*;
- Simular perturbações para a implementação de filtros de onda;

- Utilizar observadores para estimar variáveis e filtrar ruídos indesejáveis.

Assim, os pontos apresentados fornecem caminhos promissores para a evolução do simulador e para o aprofundamento dos estudos em sistemas de posicionamento dinâmico, consolidando este trabalho como um passo inicial para futuras pesquisas e aplicações práticas na área.

Referências

ALME, Jon. *Autotuned dynamic positioning for marine surface vessels*. 2008. Diss. (Mestrado) – Institutt for teknisk kybernetikk. Citado 1 vez na página 34.

BLANKE, Mogens; LINDEGAARD, Karl-Petter; FOSSEN, Thor I. Dynamic model for thrust generation of marine propellers. *IFAC Proceedings Volumes*, Elsevier, v. 33, n. 21, p. 353–358, 2000. Citado 1 vez na página 46.

BOSGRA, O. H.; KWAKERNAAK, H.; MEINSMA, G. *Design Methods for Control Systems*. 2001. P. 70–75. Citado 1 vez na página 20.

BREIVIK, Morten; KVAAL, Stig; ØSTBY, Per. From Eureka to K-Pos: Dynamic Positioning as a Highly Successful and Important Marine Control Technology. In: 16. IFAC-PAPERSONLINE. 2015. v. 48, p. 313–323. DOI: 10.1016/j.ifacol.2016.01.001. Citado 1 vez na página 13.

DERENSIS, Thomas P. *A robust linear dynamic positioning controller for a marine surface vehicle*. University of Rhode Island, 2013. Citado 1 vez na página 34.

DING, Qiang et al. Multi-Objective Optimization for Thrust Allocation of Dynamic Positioning Ship. *Journal of Marine Science and Engineering*, MDPI, v. 12, n. 7, p. 1118, 2024. Citado 1 vez na página 33.

ELLIOTT, S. J.; SUTTON, T. J. Performance of feedforward and feedback systems for active control. *IEEE Transactions on Speech and Audio Processing*, v. 4, n. 3, p. 214–223, mai. 1996. DOI: 10.1109/89.496217. Citado 1 vez na página 21.

FANNEMEL, Åsmund Våge. *Dynamic positioning by nonlinear model predictive control*. 2008. Diss. (Mestrado) – Institutt for teknisk kybernetikk. Citado 1 vez nas páginas 24, 27.

FOSSEN, Thor I; SAGATUN, Svein I. Adaptive control of nonlinear underwater robotic systems, 1991. Citado 2 vezes nas páginas 31, 33.

FOSSEN, Thor I; SAGATUN, Svein I; SØRENSEN, Asgeir J. Identification of dynamically positioned ships. *Control Engineering Practice*, Elsevier, v. 4, n. 3, p. 369–376, 1996. Citado 3 vezes nas páginas 29, 40, 41.

FOSSEN, Thor I; STRAND, Jann Peter. Passive nonlinear observer design for ships using Lyapunov methods: full-scale experiments with a supply vessel. *Automatica*, Elsevier, v. 35, n. 1, p. 3–16, 1999. Citado 0 vez na página 23.

FOSSEN, Thor I. *Chapter 15 – Controle PID para Sistemas Marítimos*. 2021. Lecture notes, https://github.com/cybergalactic/FossenHandbook/blob/main/CH15_PID_Control.pdf. Acesso em: 9 ago. 2025. Citado 1 vez na página 34.

FOSSEN, Thor I. *Guidance and Control of Ocean Vehicles*. John Wiley & Sons Ltd, 1994. ISBN 0-471-94113-1. Citado 2 vezes nas páginas 32, 34.

FOSSEN, Thor I. *Handbook of Marine Craft Hydrodynamics and Motion Control*. 1st. Chichester, UK: Wiley, 2011. Citado 13 vezes nas páginas 16, 20–22, 24, 31, 33, 34, 37, 44.

FOSSEN, Thor I. *PythonVehicleSimulator*. 2025. https://github.com/cybergalactic/PythonVehicleSimulator. Acesso em: 09 ago. 2025. Citado 1 vez na página 37.

FOSSEN, Thor I.; FJELLSTAD, Ola-Erik. Nonlinear Modelling of Marine Vehicles in 6 Degrees of Freedom. *International Journal of Mathematical Modelling of Systems*, v. 1, n. 1, 1995. Citado 2 vezes nas páginas 18, 19, 33.

IBRAHIM, Raouf; GRACE, I. M. *Ship schematic diagram showing the six degrees of freedom*. 2018. Figura em: Active vibration compensator on moving vessel by hydraulic parallel mechanism. Disponível em: https://www.researchgate.net/figure/Six-degrees-of-freedom-for-ship-motion_fig1_327901742 Acesso em: 10 jun. 2025. Citado 0 vez na página 17.

KIM, Youngjoo; BANG, Hyochoong. Introduction to Kalman filter and its applications. In: INTRODUCTION and implementations of the Kalman filter. IntechOpen, 2018. Citado 2 vezes na página 26.

KONGSBERG. *Dynamic Positioning Control Systems*. 2025. https://www.kongsberg.com/maritime/products/positioning-and-manoeuvring/dynamic-positioning/. Acesso em: 8 ago. 2025. Citado 0 vez na página 28.

KUMAR, Surender. *Dynamic positioning for engineers*. CRC Press, 2020. Citado 1 vez na página 13.

KVAAL, Stig; ØSTBY, Per; BREIVIK, Morten. DP and the Art of Perfect Positioning. *Modeling, Identification and Control*, Norwegian Society of Automatic Control, v. 43, n. 4, p. 141–159, 2022. DOI: 10.4173/mic.2022.4.3. Citado 1 vez na página 13.

LOUEIPOUR, Mehdi et al. Wave filtering and state estimation in dynamic positioning of marine vessels using position measurement. *IEEE Transactions on Instrumentation and Measurement*, IEEE, v. 64, n. 12, p. 3253–3261, 2015. Citado 1 vez na página 23.

LUENBERGER, D. G. Observers for multivariable systems. *IEEE Transactions on Automatic Control*, AC-11, n. 2, p. 190–197, 1966. Citado 1 vez na página 24.

MATHWORKS. *Moore-Penrose Pseudoinverse*. pt-BR. Acesso em: 9 ago. 2025. MathWorks, Inc. 2025. Disponível em: https://www.mathworks.com/help/matlab/ref/pinv.html. Citado 1 vez na página 33.

MONTGOMERY, Douglas C. *Introduction to statistical quality control*. John wiley & sons, 2020. Citado 1 vez na página 26.

POPOV, I; MILANOV, E. Wave filtering for marine DP system using adaptive iterated extended Kalman filter. In: SUSTAINABLE Development and Innovations in Marine Technologies. CRC Press, 2019. P. 205–216. Citado 1 vez na página 23.

QU, Huajin et al. *Wind feed-forward control of a USV*. IEEE, 2015. Citado 1 vez na página 22.

RINDARØY, Martin. *Fuel Optimal Thrust Allocation in Dynamic Positioning*. 2013. Master's thesis – Norwegian University of Science e Technology (NTNU), Department of Engineering Cybernetics, Trondheim. Disponível em: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/260863/644170_FULLTEXT01.pdf?sequence=1. Acesso em: 13 jul. 2025. Citado 2 vezes nas páginas 31–33.

SØRENSEN, Asgeir J; SAGATUN, Svein I; FOSSEN, Thor I. Design of a dynamic positioning system using model-based control. *Control Engineering Practice*, Elsevier, v. 4, n. 3, p. 359–368, 1996. Citado 1 vez na página 21.

Anexos

ANEXO A – Código main.py

```
# -*- coding: utf-8 -*-
"""
main.py: Main program for the Python Vehicle Simulator, which can be used
    to simulate and test guidance, navigation and control (GNC) systems.

Reference: T. I. Fossen (2021). Handbook of Marine Craft Hydrodynamics and
Motion Control. 2nd edition, John Wiley & Sons, Chichester, UK.
URL: https://www.fossen.biz/wiley

Author:     Thor I. Fossen
"""
import os
import sys
import webbrowser
import matplotlib.pyplot as plt
from python_vehicle_simulator.vehicles import (
    DSRV, frigate, otter, ROVzefakkel, semisub, shipClarke83, supply, tanker,
    remus100, torpedo
)
from python_vehicle_simulator.lib import (
    printSimInfo, printVehicleinfo, simulate, plotVehicleStates, plotControls,
    plot3D
)

### Simulation parameters ###
sampleTime = 0.02                   # sample time [seconds]
N = 10000                           # number of samples

# 3D plot and animation settings where browser = {firefox,chrome,safari,etc.}
numDataPoints = 50                  # number of 3D data points
FPS = 10                            # frames per second (animated GIF)
filename = '3D_animation.gif'       # data file for animated GIF
browser = 'safari'                  # browser for visualization of animated GIF

### Vehicle constructors ###
"""
Vehicle constructors:
  DSRV('depthAutopilot', z_d)
```

```
  frigate('headingAutopilot', U, psi_d)
  otter('headingAutopilot', psi_d, V_c, beta_c, tau_X)
  ROVzefakkel('headingAutopilot', U, psi_d)
  semisub('DPcontrol', x_d, y_d, psi_d, V_c, beta_c)
  shipClarke83('headingAutopilot', psi_d, L, B, T, Cb, V_c, beta_c, tau_X)
  supply('DPcontrol', x_d, y_d, psi_d, V_c, beta_c)
  tanker('headingAutopilot', psi_d, V_c, beta_c, depth)
  remus100('depthHeadingAutopilot', z_d, psi_d, V_c, beta_c)
  torpedo('depthHeadingAutopilot', z_d, psi_d, V_c, beta_c)

Call constructors without arguments to test step inputs, e.g. DSRV(), otter(),
etc.
"""

### Main program ###
def main():
    printSimInfo()

    no = input("Please enter a vehicle no.: ")

    vehicleOptions = {
        '1': lambda: DSRV('depthAutopilot', 60.0),
        '2': lambda: frigate('headingAutopilot', 10.0, 100.0),
        '3': lambda: otter('headingAutopilot', 100.0, 0.3, -30.0, 200.0),
        '4': lambda: ROVzefakkel('headingAutopilot', 3.0, 100.0),
        '5': lambda: semisub('DPcontrol', 10.0, 10.0, 40.0, 0.5, 190.0),
        '6': lambda: shipClarke83('headingAutopilot', -20.0, 70, 8, 6, 0.7,
        0.5, 10.0, 1e5),
        '7': lambda: supply('DPcontrol', 4.0, 4.0, 50.0, 0.5, 20.0),
        '8': lambda: tanker('headingAutopilot', -20, 0.5, 150, 20, 80),
        '9': lambda: remus100('depthHeadingAutopilot', 30, 50, 1525, 0.5, 170),
        '10': lambda: torpedo('depthHeadingAutopilot', 30, 50, 1525, 0.5, 170),
    }

    if no in vehicleOptions:
        vehicle = vehicleOptions[no]()
        printVehicleinfo(vehicle, sampleTime, N)
    else:
        print('Error: Not a valid simulator option')
        sys.exit()

    # Main simulation loop
```

```
    [simTime, simData] = simulate(N, sampleTime, vehicle)

    # 3D plots and animation
    plotVehicleStates(simTime, simData, 1)
    plotControls(simTime, simData, vehicle, 2)
    plot3D(simData, numDataPoints, FPS, filename, 3)

    """ Uncomment the line below for 3D animation in the web browswer.
    Alternatively, open the animated GIF file manually in your preferred
    browser. """
    # webbrowser.get(browser).open_new_tab('file://' +
    os.path.abspath(filename))

    plt.show()
    plt.close()

if __name__ == "__main__":
    main()
```

ANEXO B – Código mainLoop.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Main simulation loop called by main.py.

Author:     Thor I. Fossen
"""

import numpy as np
from .gnc import attitudeEuler

####################################################
# Function printSimInfo(vehicle)
####################################################
def printSimInfo():

    """
    Constructors used to define the vehicle objects as (see main.py for
    details):
        DSRV('depthAutopilot',z_d)
        frigate('headingAutopilot',U,psi_d)
        otter('headingAutopilot',psi_d,V_c,beta_c,tau_X)
        ROVzefakkel('headingAutopilot',U,psi_d)
        semisub('DPcontrol',x_d,y_d,psi_d,V_c,beta_c)
        shipClarke83('headingAutopilot',psi_d,L,B,T,Cb,V_c,beta_c,tau_X)
        supply('DPcontrol',x_d,y_d,psi_d,V_c,beta_c)
        tanker('headingAutopilot',psi_d,V_c,beta_c,depth)
        remus100('depthHeadingAutopilot',z_d,psi_d,V_c,beta_c)
    """

    print('-------------------------------')
    print('The Python Vehicle Simulator')
    print('-------------------------------')
    print(' 1 - Deep submergence rescue vehicle (DSRV): controlled by a stern
    plane, L = 5.0 m')
    print(' 2 - Frigate: rudder-controlled ship described by a nonlinear Nomoto
    model, L = 100.0 m')
```

```
    print(' 3 - Otter unmanned surface vehicle (USV): controlled by two
    propellers, L = 2.0 m')
    print(' 4 - ROV Zefakkel: rudder-controlled ship described by a nonlinear
    Nomoto model, L = 54.0 m')
    print(' 5 - Semisubmersible: controlled by tunnel thrusters and main
    propellers, L = 84.5 m')
    print(' 6 - Ship: linear maneuvering model specified by L, B and T using
    the Clarke (1983) formulas')
    print(' 7 - Offshore supply vessel: controlled by tunnel thrusters and main
    propellers, L = 76.2 m')
    print(' 8 - Tanker: rudder-controlled ship model including shallow water
    effects, L = 304.8 m')
    print(' 9 - REMUS 100: AUV controlled by stern planes, a tail rudder and a
    propeller, L = 1.6 m')
    print("10 - Torpedo: Inspired by the REMUS 100 AUV, configurable fins and
    propeller, L = 1.6 m")
    print('----------------------------')

##################################################
# Function printVehicleinfo(vehicle)
##################################################
def printVehicleinfo(vehicle, sampleTime, N):
    print('----------------------------')
    print('%s' % (vehicle.name))
    print('Length: %s m' % (vehicle.L))
    print('%s' % (vehicle.controlDescription))
    print('Sampling frequency: %s Hz' % round(1 / sampleTime))
    print('Simulation time: %s seconds' % round(N * sampleTime))
    print('----------------------------')


##################################################
# Function simulate(N, sampleTime, vehicle)
##################################################
def simulate(N, sampleTime, vehicle):

    DOF = 6                     # degrees of freedom
    t = 0                       # initial simulation time

    # Initial state vectors
    eta = np.array([0, 0, 0, 0, 0, 0], float)    # position/attitude, user
    editable
```

```
nu = vehicle.nu                                # velocity, defined by vehicle
class
u_actual = vehicle.u_actual                    # actual inputs, defined by
vehicle class

# Initialization of table used to store the simulation data
simData = np.empty( [0, 2*DOF + 2 * vehicle.dimU], float)

# Simulator for-loop
for i in range(0,N+1):

    t = i * sampleTime      # simulation time

    # Vehicle specific control systems
    if (vehicle.controlMode == 'depthAutopilot'):
        u_control = vehicle.depthAutopilot(eta,nu,sampleTime)
    elif (vehicle.controlMode == 'headingAutopilot'):
        u_control = vehicle.headingAutopilot(eta,nu,sampleTime)
    elif (vehicle.controlMode == 'depthHeadingAutopilot'):
        u_control = vehicle.depthHeadingAutopilot(eta,nu,sampleTime)
    elif (vehicle.controlMode == 'DPcontrol'):
        u_control = vehicle.DPcontrol(eta,nu,sampleTime)
    elif (vehicle.controlMode == 'stepInput'):
        u_control = vehicle.stepInput(t)

    # Store simulation data in simData
    signals = np.append( np.append( np.append(eta,nu),u_control), u_actual
    )
    simData = np.vstack( [simData, signals] )

    # Propagate vehicle and attitude dynamics
    [nu, u_actual]  =
    vehicle.dynamics(eta,nu,u_actual,u_control,sampleTime)
    eta = attitudeEuler(eta,nu,sampleTime)

# Store simulation time vector
simTime = np.arange(start=0, stop=t+sampleTime, step=sampleTime)[:, None]

return(simTime,simData)
```

ANEXO C – Código supply.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
supply.py:
    Class for an offshore supply vessel length L = 76.2.
    The constructors are:

    supply()
        Step inputs for the propeller speeds n1, n2, n3, n4, n5 and n6
    supply('DPcontrol',x_d,y_d,psi_d,V_c,beta_c)  DP control system
        x_d: desired x position (m)
        y_d: desired y position (m)
        psi_d: desired yaw angle (deg)
        V_c: current speed (m/s)
        beta_c: current direction (deg)

Methods:

    [nu,u_actual] = dynamics(eta,nu,u_actual,u_control,sampleTime)
        returns nu[k+1] and u_actual[k+1] using Euler's method.
        The control inputs are:

        u_control = n
        n = [  #1 Bow tunnel thruster (RPM),
               #2 Bow tunnel thruster (RPM),
               #3 Stern tunnel thruster (RPM),
               #4 Stern tunnel thruster (RPM),
               #5 Right main propeller (RPM),
               #6 Left main propeller (RPM)]

    u_alloc = controlAllocation(tau)
        Control allocation based on the pseudoinverse

    n = DPcontrol(eta,nu,sampleTime)
        Nonlinear PID controller for DP based on pole placement.

    n = stepInput(t) generates propellers step inputs.
```

```
References:

    T. I. Fossen, S. I. Sagatun and A. J. Sorensen (1996)
         Identification of Dynamically Positioned Ships
         Journal of Control Engineering Practice CEP-4(3):369-376
    T. I. Fossen (2021). Handbook of Marine Craft Hydrodynamics and Motion
         Control. 2nd. Edition, Wiley. URL: www.fossen.biz/wiley

Author:     Thor I. Fossen
"""
import numpy as np
import math
from python_vehicle_simulator.lib.control import DPpolePlacement
from python_vehicle_simulator.lib.gnc import sat

# Class Vehicle
class supply:
    """
    supply()                                        Propeller step inputs
    supply('DPcontrol',x_d,y_d,psi_d,V_c,beta_c)  DP control system

    Inputs:
        x_d: desired x position (m)
        y_d: desired y position (m)
        psi_d: desired yaw angle (deg)
        V_c: current speed (m/s)
        beta_c: current direction (deg)
    """

    def __init__(
        self,
        controlSystem="stepInput",
        r_x = 0,
        r_y = 0,
        r_n = 0,
        V_current = 0,
        beta_current = 0,
    ):

        # Constants
        D2R = math.pi / 180     # deg2rad
        g = 9.81                # acceleration of gravity (m/s^2)
```

```
if controlSystem == "DPcontrol":
    self.controlDescription = (
        "Nonlinear DP control (x_d, y_d, psi_d) = ("
        + str(r_x)
        + " m, "
        + str(r_y)
        + " m, "
        + str(r_n)
        + " deg)"
    )

else:
    self.controlDescription = "Step inputs n = [n1, n2, n3, n4, n5,
    n6]"
    controlSystem = "stepInput"

self.ref = np.array([r_x, r_y, r_n * D2R], float)
self.V_c = V_current
self.beta_c = beta_current * D2R
self.controlMode = controlSystem

# Initialize the supply vessel model
m = 6000.0e3        # mass (kg)
self.L = 76.2       # length (m)
self.T_n = 1.0      # prop. speed time constant (s)
self.n_max = np.array([250, 250, 250, 250,
                       160, 160], float) # RPM saturation limits
self.nu = np.array([0, 0, 0, 0, 0, 0], float) # initial velocity vector
self.u_actual = np.array([0, 0, 0, 0, 0, 0], float) # RPM inputs
self.name = "Offshore supply vessel (see 'supply.py' for more details)"

# Two tunnel thrusters in the bow, no. 1 and 2
# Two tunnel thrusters in the stern, no. 3 and 4
# Two main propellers aft, no. 3 and 4
self.controls = [
    "#1 Bow tunnel thruster (RPM)",
    "#2 Bow tunnel thruster (RPM)",
    "#3 Stern tunnel thruster (RPM)",
    "#4 Stern tunnel thruster (RPM)",
    "#5 Right main propeller (RPM)",
    "#6 Left main propeller (RPM)"
```

```
    ]
    self.dimU = len(self.controls)

    # Thrust coefficient and configuration matrices (Fossen 2021, Ch. 11.2)
    # Thrust_max(i) = K(i) * n_max(i)^2
    # Tunnel thruster: 3.2 * 250^2 = 200 kN
    # Main propeller: 31.2 * 160^2 = 799 kN
    K = np.diag([3.2, 3.2, 3.2, 3.2, 31.2, 31.2])
    T = np.array(
        [ [0, 0, 0, 0, 1, 1], [1, 1, 1, 1, 0, 0],
          [30, 22, -22, -30, -8, 8] ], float
    )
    self.B = T @ K

    # Tbis = np.diag( [1, 1, 1 / self.L],float)
    Tbis_inv = np.diag([1.0, 1.0, self.L])

    # 3-DOF model matrices - bis scaling (Fossen 2021, App. D)
    Mbis = np.array(
        [[1.1274, 0, 0], [0, 1.8902, -0.0744], [0, -0.0744, 0.1278]], float
    )

    Dbis = np.array(
        [[0.0358, 0, 0], [0, 0.1183, -0.0124], [0, -0.0041, 0.0308]], float
    )

    self.M3 = m * Tbis_inv @ Mbis @ Tbis_inv
    self.M3inv = np.linalg.inv(self.M3)
    self.D3 = m * math.sqrt(g / self.L) * Tbis_inv @ Dbis @ Tbis_inv

    # DP control system
    self.e_int = np.array([0, 0, 0], float)  # integral states
    self.x_d = 0.0  # setpoints
    self.y_d = 0.0
    self.psi_d = 0.0
    self.wn = np.diag([0.1, 0.1, 0.2])    # PID pole placement
    self.zeta = np.diag([1.0, 1.0, 1.0])


def dynamics(self, eta, nu, u_actual, u_control, sampleTime):
    """
```

```
        [nu,u_actual] = dynamics(eta,nu,u_actual,u_control,sampleTime)
        integrates the
        supply vessel equations of motion using Euler's method.
        """

        # Input vector
        n = u_actual  # propeller speed (RPM)

        # Current velocities
        u_c = self.V_c * math.cos(self.beta_c - eta[5]) # current surge
        velocity
        v_c = self.V_c * math.sin(self.beta_c - eta[5]) # current sway velocity

        nu_c = np.array([u_c, v_c, 0, 0, 0, 0], float) # current velocity
        vector
        nu_r = nu - nu_c  # relative velocity vector

        # Control forces and moments with propeller saturation
        n_squared = np.zeros(self.dimU)
        for i in range(0, self.dimU):
            n[i] = sat(n[i], -self.n_max[i], self.n_max[i])      # saturation
            n_squared[i] = abs(n[i]) * n[i]

        tau3 = np.matmul(self.B, n_squared)

        # 3-DOF dynamics
        nu3_r = np.array([nu_r[0], nu_r[1], nu_r[5]])
        nu3_dot = np.matmul(self.M3inv, tau3 - np.matmul(self.D3, nu3_r))

        # 6-DOF ship model and propeller speed dynamics
        nu_dot = np.array([nu3_dot[0], nu3_dot[1], 0, 0, 0, nu3_dot[2]])
        n_dot = (u_control - u_actual) / self.T_n

        # Forward Euler integration
        nu = nu + sampleTime * nu_dot
        n = n + sampleTime * n_dot

        u_actual = np.array(n, float)

        return nu, u_actual

    def controlAllocation(self, tau3):
```

```
    """
    u_alloc  = controlAllocation(tau3),  tau3 = [tau_X, tau_Y, tau_N]'
    u_alloc = B' * inv( B * B' ) * tau3
    """
    B_pseudoInv = self.B.T @ np.linalg.inv(self.B @ self.B.T)
    u_alloc = np.matmul(B_pseudoInv, tau3)  # squared propeller speed

    return u_alloc


def DPcontrol(self, eta, nu, sampleTime):
    """
    u = DPcontrol(eta,nu,sampleTime) is a nonlinear PID controller
    for DP based on pole placement:

    tau = -R' Kp (eta-r) - Kd nu - R' Ki int(eta-r)
    u = B_pseudoinverse * tau
    """
    eta3 = np.array([eta[0], eta[1], eta[5]])
    nu3 = np.array([nu[0], nu[1], nu[5]])

    [tau3, self.e_int, self.x_d, self.y_d, self.psi_d] = DPpolePlacement(
        self.e_int,
        self.M3,
        self.D3,
        eta3,
        nu3,
        self.x_d,
        self.y_d,
        self.psi_d,
        self.wn,
        self.zeta,
        self.ref,
        sampleTime,
    )

    u_alloc = self.controlAllocation(tau3)

    # u_alloc = abs(n) * n --> n = sign(u_alloc) * sqrt(u_alloc)
    n = np.zeros(self.dimU)
    for i in range(0, self.dimU):
        n[i] = np.sign(u_alloc[i]) * math.sqrt(abs(u_alloc[i]))
```

```
    u_control = n

    return u_control


def stepInput(self, t):
    """
    u = stepInput(t) generates propeller step inputs (RPM).
    """
    n = np.array([0, 0, 0, 0, 100, 100], float)

    if t > 30:
        n = np.array([50, 50, 50, 50, 50, 50], float)
    if t > 70:
        n = np.array([0, 0, 0, 0, 0, 0], float)

    u_control = n

    return u_control
```

ANEXO D – Código gnc.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
GNC functions.

Reference: T. I. Fossen (2021). Handbook of Marine Craft Hydrodynamics and
Motion Control. 2nd. Edition, Wiley.
URL: www.fossen.biz/wiley

Author:     Thor I. Fossen
"""

import numpy as np
import math

#-------------------------------------------------------------------------------

def ssa(angle):
    """
    angle = ssa(angle) returns the smallest-signed angle in [ -pi, pi )
    """
    angle = (angle + math.pi) % (2 * math.pi) - math.pi

    return angle

#-------------------------------------------------------------------------------

def sat(x, x_min, x_max):
    """
    x = sat(x,x_min,x_max) saturates a signal x such that x_min <= x <= x_max
    """
    if x > x_max:
        x = x_max
    elif x < x_min:
        x = x_min

    return x
```

```
#--------------------------------------------------------------------------

def Smtrx(a):
    """
    S = Smtrx(a) computes the 3x3 vector skew-symmetric matrix S(a) = -S(a)'.
    The cross product satisfies: a x b = S(a)b.
    """

    S = np.array([
        [ 0, -a[2], a[1] ],
        [ a[2],   0,     -a[0] ],
        [-a[1],   a[0],   0 ]  ])

    return S


#--------------------------------------------------------------------------

def Hmtrx(r):
    """
    H = Hmtrx(r) computes the 6x6 system transformation matrix
    H = [eye(3)     S'
         zeros(3,3) eye(3) ]       Property: inv(H(r)) = H(-r)

    If r = r_bg is the vector from the CO to the CG, the model matrices in CO
    and
    CG are related by: M_CO = H(r_bg)' * M_CG * H(r_bg). Generalized position
    and
    force satisfy: eta_CO = H(r_bg)' * eta_CG and tau_CO = H(r_bg)' * tau_CG
    """

    H = np.identity(6,float)
    H[0:3, 3:6] = Smtrx(r).T

    return H

#--------------------------------------------------------------------------

def Rzyx(phi,theta,psi):
    """
    R = Rzyx(phi,theta,psi) computes the Euler angle rotation matrix R in SO(3)
    using the zyx convention
```

```
    """

    cphi = math.cos(phi)
    sphi = math.sin(phi)
    cth  = math.cos(theta)
    sth  = math.sin(theta)
    cpsi = math.cos(psi)
    spsi = math.sin(psi)

    R = np.array([
        [ cpsi*cth, -spsi*cphi+cpsi*sth*sphi, spsi*sphi+cpsi*cphi*sth ],
        [ spsi*cth,  cpsi*cphi+sphi*sth*spsi, -cpsi*sphi+sth*spsi*cphi ],
        [ -sth,      cth*sphi,                 cth*cphi ] ])

    return R

#--------------------------------------------------------------------------

def Tzyx(phi,theta):
    """
    T = Tzyx(phi,theta) computes the Euler angle attitude
    transformation matrix T using the zyx convention
    """

    cphi = math.cos(phi)
    sphi = math.sin(phi)
    cth  = math.cos(theta)
    sth  = math.sin(theta)

    try:
        T = np.array([
            [ 1,  sphi*sth/cth,  cphi*sth/cth ],
            [ 0,  cphi,          -sphi],
            [ 0,  sphi/cth,      cphi/cth] ])

    except ZeroDivisionError:
        print ("Tzyx is singular for theta = +-90 degrees." )

    return T

#--------------------------------------------------------------------------
```

```
def attitudeEuler(eta,nu,sampleTime):
    """
    eta = attitudeEuler(eta,nu,sampleTime) computes the generalized
    position/Euler angles eta[k+1]
    """

    p_dot   = np.matmul( Rzyx(eta[3], eta[4], eta[5]), nu[0:3] )
    v_dot   = np.matmul( Tzyx(eta[3], eta[4]), nu[3:6] )

    # Forward Euler integration
    eta[0:3] = eta[0:3] + sampleTime * p_dot
    eta[3:6] = eta[3:6] + sampleTime * v_dot

    return eta


#------------------------------------------------------------------------------

def m2c(M, nu):
    """
    C = m2c(M,nu) computes the Coriolis and centripetal matrix C from the
    mass matrix M and generalized velocity vector nu (Fossen 2021, Ch. 3)
    """

    M = 0.5 * (M + M.T)     # systematization of the inertia matrix

    if (len(nu) == 6):      #  6-DOF model

        M11 = M[0:3,0:3]
        M12 = M[0:3,3:6]
        M21 = M12.T
        M22 = M[3:6,3:6]

        nu1 = nu[0:3]
        nu2 = nu[3:6]
        dt_dnu1 = np.matmul(M11,nu1) + np.matmul(M12,nu2)
        dt_dnu2 = np.matmul(M21,nu1) + np.matmul(M22,nu2)

        #C  = [  zeros(3,3)      -Smtrx(dt_dnu1)
        #      -Smtrx(dt_dnu1)  -Smtrx(dt_dnu2) ]
        C = np.zeros( (6,6) )
        C[0:3,3:6] = -Smtrx(dt_dnu1)
```

```
        C[3:6,0:3] = -Smtrx(dt_dnu1)
        C[3:6,3:6] = -Smtrx(dt_dnu2)

    else:   # 3-DOF model (surge, sway and yaw)
        #C = [ 0             0            -M(2,2)*nu(2)-M(2,3)*nu(3)
        #      0             0             M(1,1)*nu(1)
        #      M(2,2)*nu(2)+M(2,3)*nu(3)  -M(1,1)*nu(1)          0  ]
        C = np.zeros( (3,3) )
        C[0,2] = -M[1,1] * nu[1] - M[1,2] * nu[2]
        C[1,2] =  M[0,0] * nu[0]
        C[2,0] = -C[0,2]
        C[2,1] = -C[1,2]

    return C

#---------------------------------------------------------------------------

def Hoerner(B,T):
    """
    CY_2D = Hoerner(B,T)
    Hoerner computes the 2D Hoerner cross-flow form coeff. as a function of
    beam
    B and draft T.The data is digitized and interpolation is used to compute
    other data point than those in the table
    """

    # DATA = [B/2T  C_D]
    DATA1 = np.array([
        0.0109,0.1766,0.3530,0.4519,0.4728,0.4929,0.4933,0.5585,0.6464,0.8336,
        0.9880,1.3081,1.6392,1.8600,2.3129,2.6000,3.0088,3.4508, 3.7379,4.0031
        ])
    DATA2 = np.array([
        1.9661,1.9657,1.8976,1.7872,1.5837,1.2786,1.2108,1.0836,0.9986,0.8796,
        0.8284,0.7599,0.6914,0.6571,0.6307,0.5962,0.5868,0.5859,0.5599,0.5593
        ])

    CY_2D = np.interp( B / (2 * T), DATA1, DATA2 )

    return CY_2D

#---------------------------------------------------------------------------
```

```
def crossFlowDrag(L,B,T,nu_r):
    """
    tau_crossflow = crossFlowDrag(L,B,T,nu_r) computes the cross-flow drag
    integrals for a marine craft using strip theory.

    M d/dt nu_r + C(nu_r)*nu_r + D*nu_r + g(eta) = tau + tau_crossflow
    """

    rho = 1026               # density of water
    n = 20                   # number of strips

    dx = L/20
    Cd_2D = Hoerner(B,T)    # 2D drag coefficient based on Hoerner's curve

    Yh = 0
    Nh = 0
    xL = -L/2

    for i in range(0,n+1):
        v_r = nu_r[1]             # relative sway velocity
        r = nu_r[5]               # yaw rate
        Ucf = abs(v_r + xL * r) * (v_r + xL * r)
        Yh = Yh - 0.5 * rho * T * Cd_2D * Ucf * dx         # sway force
        Nh = Nh - 0.5 * rho * T * Cd_2D * xL * Ucf * dx    # yaw moment
        xL += dx

    tau_crossflow = np.array([0, Yh, 0, 0, 0, Nh],float)

    return tau_crossflow

#------------------------------------------------------------------------------

def forceLiftDrag(b,S,CD_0,alpha,U_r):
    """
    tau_liftdrag = forceLiftDrag(b,S,CD_0,alpha,Ur) computes the hydrodynamic
    lift and drag forces of a submerged "wing profile" for varying angle of
    attack (Beard and McLain 2012). Application:

      M d/dt nu_r + C(nu_r)*nu_r + D*nu_r + g(eta) = tau + tau_liftdrag

    Inputs:
        b:     wing span (m)
```

```
    S:     wing area (m^2)
    CD_0:  parasitic drag (alpha = 0), typically 0.1-0.2 for a streamlined
    body
    alpha: angle of attack, scalar or vector (rad)
    U_r:   relative speed (m/s)

Returns:
    tau_liftdrag:  6x1 generalized force vector
"""

# constants
rho = 1026

def coeffLiftDrag(b,S,CD_0,alpha,sigma):

    """
    [CL,CD] = coeffLiftDrag(b,S,CD_0,alpha,sigma) computes the hydrodynamic
    lift CL(alpha) and drag CD(alpha) coefficients as a function of alpha
    (angle of attack) of a submerged "wing profile" (Beard and McLain 2012)

    CD(alpha) = CD_p + (CL_0 + CL_alpha * alpha)^2 / (pi * e * AR)
    CL(alpha) = CL_0 + CL_alpha * alpha

    where CD_p is the parasitic drag (profile drag of wing, friction and
    pressure drag of control surfaces, hull, etc.), CL_0 is the zero angle
    of attack lift coefficient, AR = b^2/S is the aspect ratio and e is the
    Oswald efficiency number. For lift it is assumed that

    CL_0 = 0
    CL_alpha = pi * AR / ( 1 + sqrt(1 + (AR/2)^2) );

    implying that for alpha = 0, CD(0) = CD_0 = CD_p and CL(0) = 0. For
    high angles of attack the linear lift model can be blended with a
    nonlinear model to describe stall

    CL(alpha) = (1-sigma) * CL_alpha * alpha + ...
        sigma * 2 * sign(alpha) * sin(alpha)^2 * cos(alpha)

    where 0 <= sigma <= 1 is a blending parameter.

    Inputs:
        b:       wing span (m)
```

```
        S:       wing area (m^2)
        CD_0:    parasitic drag (alpha = 0), typically 0.1-0.2 for a
                 streamlined body
        alpha:   angle of attack, scalar or vector (rad)
        sigma:   blending parameter between 0 and 1, use sigma = 0 f
                 or linear lift
        display: use 1 to plot CD and CL (optionally)

    Returns:
        CL: lift coefficient as a function of alpha
        CD: drag coefficient as a function of alpha

    Example:
        Cylinder-shaped AUV with length L = 1.8, diameter D = 0.2 and
        CD_0 = 0.3

        alpha = 0.1 * pi/180
        [CL,CD] = coeffLiftDrag(0.2, 1.8*0.2, 0.3, alpha, 0.2)
    """

    e = 0.7             # Oswald efficiency number
    AR = b**2 / S       # wing aspect ratio

    # linear lift
    CL_alpha = math.pi * AR / ( 1 + math.sqrt(1 + (AR/2)**2) )
    CL = CL_alpha * alpha

    # parasitic and induced drag
    CD = CD_0 + CL**2 / (math.pi * e * AR)

    # nonlinear lift (blending function)
    CL = (1-sigma) * CL + sigma * 2 * np.sign(alpha) \
        * math.sin(alpha)**2 * math.cos(alpha)

    return CL, CD


[CL, CD] = coeffLiftDrag(b,S,CD_0,alpha,0)

F_drag = 1/2 * rho * U_r**2 * S * CD    # drag force
F_lift = 1/2 * rho * U_r**2 * S * CL    # lift force
```

```
    # transform from FLOW axes to BODY axes using angle of attack
    tau_liftdrag = np.array([
        math.cos(alpha) * (-F_drag) - math.sin(alpha) * (-F_lift),
        0,
        math.sin(alpha) * (-F_drag) + math.cos(alpha) * (-F_lift),
        0,
        0,
        0 ])

    return tau_liftdrag

#--------------------------------------------------------------------------

def gvect(W,B,theta,phi,r_bg,r_bb):
    """
    g = gvect(W,B,theta,phi,r_bg,r_bb) computes the 6x1 vector of restoring
    forces about an arbitrarily point CO for a submerged body.

    Inputs:
        W, B: weight and buoyancy (kg)
        phi,theta: roll and pitch angles (rad)
        r_bg = [x_g y_g z_g]: location of the CG with respect to the CO (m)
        r_bb = [x_b y_b z_b]: location of the CB with respect to th CO (m)

    Returns:
        g: 6x1 vector of restoring forces about CO
    """

    sth  = math.sin(theta)
    cth  = math.cos(theta)
    sphi = math.sin(phi)
    cphi = math.cos(phi)

    g = np.array([
        (W-B) * sth,
        -(W-B) * cth * sphi,
        -(W-B) * cth * cphi,
        -(r_bg[1]*W-r_bb[1]*B) * cth * cphi + (r_bg[2]*W-r_bb[2]*B) * cth *
        sphi,
        (r_bg[2]*W-r_bb[2]*B) * sth         + (r_bg[0]*W-r_bb[0]*B) * cth *
        cphi,
        -(r_bg[0]*W-r_bb[0]*B) * cth * sphi - (r_bg[1]*W-r_bb[1]*B) * sth
```

```
    ])

return g
```

ANEXO E – Código control.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Control methods.

Reference: T. I. Fossen (2021). Handbook of Marine Craft Hydrodynamics and
Motion Control. 2nd. Edition, Wiley.
URL: www.fossen.biz/wiley

Author:     Thor I. Fossen
"""

import numpy as np
from python_vehicle_simulator.lib.guidance import refModel3
from python_vehicle_simulator.lib.gnc import ssa, Rzyx

# SISO PID pole placement
def PIDpolePlacement(
    e_int,
    e_x,
    e_v,
    x_d,
    v_d,
    a_d,
    m,
    d,
    k,
    wn_d,
    zeta_d,
    wn,
    zeta,
    r,
    v_max,
    sampleTime,
):

    # PID gains based on pole placement
    Kp = m * wn ** 2.0 - k
```

```
    Kd = m * 2.0 * zeta * wn - d
    Ki = (wn / 10.0) * Kp

    # PID control law
    u = -Kp * e_x - Kd * e_v - Ki * e_int

    # Integral error, Euler's method
    e_int += sampleTime * e_x

    # 3rd-order reference model for smooth position, velocity and acceleration
    [x_d, v_d, a_d] = refModel3(x_d, v_d, a_d, r, wn_d, zeta_d, v_max,
    sampleTime)

    return u, e_int, x_d, v_d, a_d


# MIMO nonlinear PID pole placement
def DPpolePlacement(
    e_int, M3, D3, eta3, nu3, x_d, y_d, psi_d, wn, zeta, eta_ref, sampleTime
):

    # PID gains based on pole placement
    M3_diag = np.diag(np.diag(M3))
    D3_diag = np.diag(np.diag(D3))

    Kp = wn @ wn @ M3_diag
    Kd = 2.0 * zeta @ wn @ M3_diag - D3_diag
    Ki = (1.0 / 10.0) * wn @ Kp

    # DP control law - setpoint regulation
    e = eta3 - np.array([x_d, y_d, psi_d])
    e[2] = ssa(e[2])
    R = Rzyx(0.0, 0.0, eta3[2])
    tau = (
        - np.matmul((R.T @ Kp), e)
        - np.matmul(Kd, nu3)
        - np.matmul((R.T @ Ki), e_int)
    )

    # Low-pass filters, Euler's method
    T = 5.0 * np.array([1 / wn[0][0], 1 / wn[1][1], 1 / wn[2][2]])
    x_d += sampleTime * (eta_ref[0] - x_d) / T[0]
```

```
    y_d += sampleTime * (eta_ref[1] - y_d) / T[1]
    psi_d += sampleTime * (eta_ref[2] - psi_d) / T[2]

    # Integral error, Euler's method
    e_int += sampleTime * e

    return tau, e_int, x_d, y_d, psi_d

# Heading autopilot - Intergral SMC (Equation 16.479 in Fossen 2021)
def integralSMC(
    e_int,
    e_x,
    e_v,
    x_d,
    v_d,
    a_d,
    T_nomoto,
    K_nomoto,
    wn_d,
    zeta_d,
    K_d,
    K_sigma,
    lam,
    phi_b,
    r,
    v_max,
    sampleTime,
):

    # Sliding surface
    v_r_dot = a_d - 2 * lam * e_v - lam ** 2 * ssa(e_x)
    v_r     = v_d - 2 * lam * ssa(e_x) - lam ** 2 * e_int
    sigma   = e_v + 2 * lam * ssa(e_x) + lam ** 2 * e_int

    #  Control law
    if abs(sigma / phi_b) > 1.0:
        delta = ( T_nomoto * v_r_dot + v_r - K_d * sigma
                 - K_sigma * np.sign(sigma) ) / K_nomoto
    else:
        delta = ( T_nomoto * v_r_dot + v_r - K_d * sigma
                 - K_sigma * (sigma / phi_b) ) / K_nomoto
```

```
# Integral error, Euler's method
e_int += sampleTime * ssa(e_x)

# 3rd-order reference model for smooth position, velocity and acceleration
[x_d, v_d, a_d] = refModel3(x_d, v_d, a_d, r, wn_d, zeta_d, v_max,
sampleTime)

return delta, e_int, x_d, v_d, a_d
```