Universidade de São Paulo
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica e Computação

Desenvolvimento de Software de Controle para um Veículo Aquático Autônomo

Tiago Diadami Perez

São Carlos - SP
2014

Tiago Diadami Perez

Desenvolvimento de Software de Controle para um Veículo Aquático Autônomo

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em
eletrônica

Orientador: José Roberto Boffino de Almeida Monteiro

São Carlos - SP
2014

Diadami Perez, Tiago
D438d Desenvolvimento de Software de Controle para um Veículo Aquático Autônomo / Tiago Diadami Perez; orientador José Roberto Boffino de Almeida Monteiro. São Carlos, 2014.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2014.

1. Autônomo. 2. Não Tripulado. 3. Controlador. 4. Orientação. I. Título.

FOLHA DE APROVAÇÃO

Nome: Tiago Diadami Perez

Título: "Desenvolvimento de Software de controle para um veículo aquático autônomo"

Trabalho de Conclusão de Curso defendido e aprovado em 25/06/2014,

com NOTA 9,5 (nove, cinco), **pela Comissão Julgadora:**

Prof. Dr. José Roberto Boffino de Almeida Monteiro - (Orientador - SEL/EESC/USP)

Prof. Associado Ivan Nunes da Silva - (SEL/EESC/USP)

Mestre Thales Eugenio Portes de Almeida - (Doutorando - SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

"Our mission is to preserve, protect and promote the freedom to use, study, copy, modify, and redistribute computer software, and to defend the rights of Free Software users."

-Free Software Foundation

Resumo

T.(2014). Desenvolvimento de Software de Controle e Gerenciamento para um Veículo Aquático Autônomo (TCC) - Escola de Engenharia São Carlos, Universidade de São Paulo, São Carlos, 2014.

O presente trabalho trata do desenvolvimento de um sistema de controle para um veículo aquático de superfície autônomo não tripulado. É mostrado o desenvolvimento de um simulador numérico da dinâmica da embarcação, utilizado para validação do sistema de controle. Também é mostrado o desenvolvimento do sistema de orientação. Utiliza-se para o computador de bordo o computador de placa única *Raspberry PI* e como plataforma de desenvolvimento GNU/Linux. Todos códigos são feitos em linguagem de programação C e todos os *softwares* utilizados são de código aberto.

Palavras-chave: Autônomo, Não Tripulado, Controlador, Orientação.

Abstract

T. (2014). Development of an Control and Managing Software for an Autonomous Aquatic Vehicle (Final Paper) - Escola de Engenharia São Carlos, Universidade de São Paulo, São Carlos, 2014.

This work describes the development of a control system for a autonomous unmanned aquatic vehicle. It is shown the development of a numeric simulator for the dinamic of the boat, and it is used to validate the control system. It is also shown the development of the orientation system. For the onboard computer is used the single-board computer *Raspberry PI* and as development platform GNU/Linux. All codes was are written in programming language C and all softwares used are open source.

Keywords: Autonomous, Unmanned, Control, Orientation.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

A demanda por sistemas automatizados, robotizados e não tripulados é dirigida por aplicações que são inerentemente repetitivas, desagradáveis e perigosas. Atualmente, as tarefas que tipicamente são inclusas nessas categorias são agricultura, manuseio de cargas sobre convés de embarcações, transporte coletivo (repetitivo), exploração científica, mineração, manejo de resíduos (desagradável), resgates e combate a incêndios (perigoso) (FINN; SCHEDING, 2010).

Neste trabalho é desenvolvido um sistema de controle para um sistema não tripulado autônomo cuja atividade se enquadra como repetitiva. Este sistema é uma embarcação não tripulada autônoma voltada para o monitoramento ambiental. A embarcação é projetada para água doce e tem o intuito de realizar observações de peixes, da qualidade da água, bem como outros tipos de observações em águas fluviais - rios, lagos e represas.

A embarcação desenvolvida tem a forma de trimarã (Peng, Y.; Han, 2009), e conta com um sistema de propulsão por dois motores elétricos na parte de trás, um de cada lado do veiculo, de forma que as curvas são feitas por diferença de velocidade entre os motores (ALMEIDA, 2014).

O sistema de controle que é desenvolvido é capaz de interpretar um arquivo com coordenadas geográficas, traçar uma trajetória a partir destes pontos e fazer com que a embarcação siga por esta trajetória até o fim da missão.

1.1 O Sistema Embarcado

Um sistema embarcado é um computador de propósito especial que é desenvolvido para desempenhar pequenos conjuntos de tarefas específicas. No começo do desenvolvimento de sistemas embarcados não eram utilizados sistemas operacionais, os *softwares* eram desenvolvidos exclusivamente para os *hardwares* com muito pouco ou nenhum sistema multitarefa e sem interação com o usuário (RAGHAVAN; LAD; NEELAKANDAN, 2006, p.1). Porém as exigências para sistemas embarcados começaram a ficar mais complexas com o passar do tempo, e o

uso de sistemas operacionais se tornou mandatatório. Hoje há diversos sistemas operacionais direcionados exclusivamente para sistemas embarcados.

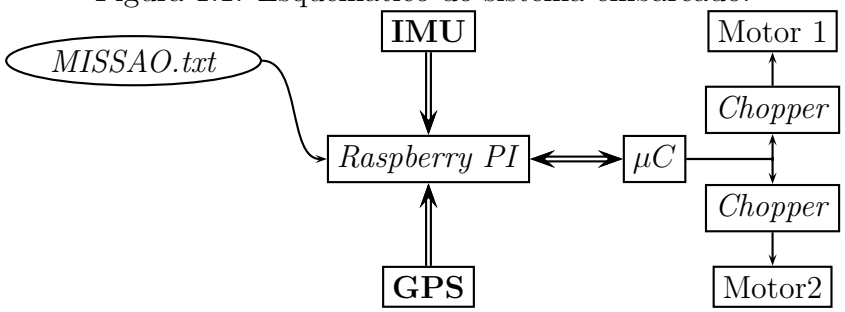
Deve-se levar muitos fatores em consideração na escolha do sistema operacional no projeto de sistemas embarcados, como custo, suporte de *hardwares*, suporte do desenvolvedor, desempenho, etc. Neste presente projeto escolheu-se como plataforma de desenvolvimento o sistema operacional GNU/Linux. Sabe-se que as forças armadas americanas utilizam como plataforma de desenvolvimento o GNU/Linux, eles possuem a maior base instalada da distribuição *Red Hat* [15].

Os fatores relevantes para a escolha do GNU/Linux como sistema operacional são listados a seguir:

- **Custo:** todas ferramentas necessárias, como compiladores, *linkers*, bibliotecas, etc., podem ser encontrados livremente na internet.
- **Suporte de *hardware*:** há suporte para uma porção de arquiteturas e dispositivos I/O *high-end*, dando independência para a escolha do *hardware* apropriado para o projeto.
- **Open Source:** há milhares de desenvolvedores ao redor do mundo que contribuem para o melhoramento do *Linux kernel* e outras aplicações, sendo possível acessar diversos *mainling lists* e seus arquivos para sanar dúvidas.

Após a escolha do sistema operacional deve-se listar as necessidades do projeto em termos de *hardware* para a escolha do computador de bordo adequada. O sistema tem uma unidade inercial (do inglês, Inertial Measurement Unit - IMU), um sistema de posicionamento global (do inglês, Global Positioning System - GPS), um micro-controlador (μC), dois conversores de tensão CC-CC (*Chopper*) e dois motores elétricos. O esquemático do sistema embarcado é mostrado na Figura 1.1.

Figura 1.1: Esquemático do sistema embarcado.



As principais necessidades para o presente projeto é que o computador de bordo possua UART para a comunicação com os micro-controladores dos motores, porta USB para comunicação com a unidade inercial, processamento com bom desempenho para cálculos, baixo consumo de energia e baixo custo. Para essas necessidades escolheu-se o computador de placa única *Raspberry Pi*, suas principais especificações são mostradas na tabela 1.1.

Tabela 1.1: Especificações do computador *Raspberry Pi*.

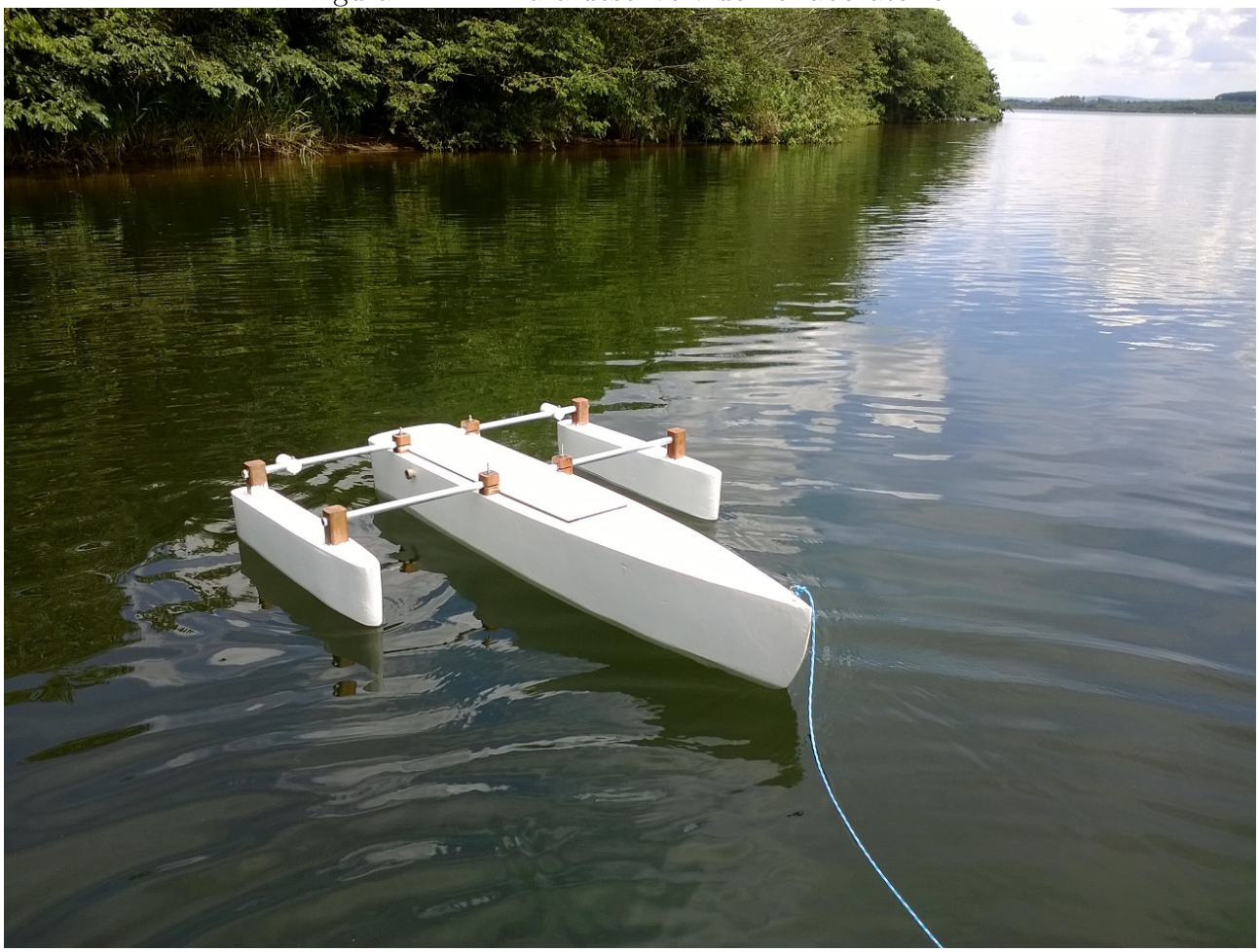
Modelo	**B**
CPU	700 MHz ARM11 *family*
Memória (SDRAM)	512 MB
Portas USB 2.0	2
Armazenamento	SD/MMC/slot cartão SDIO
Periféricos	8x GPIO, UART, I^2C, SPI
Consumo	700 mA (3.5 W)
Preço unitário	USD$ 35,00

1.2 A Embarcação

Tradicionalmente utiliza-se embarcações de um casco apenas, devido a sua simplicidade, simetria e diversos registros de projetos e estudos de sua operação. Por outro lado trimarãs oferecem algumas vantagens, especialmente a grande área útil para ser aproveitada. Devido a construção particular possui baixa resistência à ondas, as ondas são canceladas pelo efeito de interferência de onda oriundo de cada casco da embarcação, excelente *seakeeping* e baixo deslocamento de água (Peng, Y. P. Y.; Zhou; Han, 2007).

O conceito do trimarã é de uma embarcação com três cascos. Um casco central e dois outros menores conectados a este, denominados amas, que ajudam na estabilidade da embarcação. A embarcação desenvolvida no laboratório possui $1,80$ metro de comprimento por $1,60$ metro de largura. Construída em isopor de alta densidade e madeira, ela pesa 25 quilos. A Figura 1.2 mostra a embarcação pronta.

Figura 1.2: Trimarã desenvolvido no laboratório.



A dinâmica da embarcação é modelada como um corpo rígido com seis graus de liberdade. Com o sistema de coordenadas local fixo no centro de gravidade como mostra a figura 1.3, tem-se o movimento em X, Y, Z e as rotações nesses eixos. A dinâmica então pode ser descrita pelo sistema de equações 1.1, 1.2, 1.3, 1.4, 1.5 e 1.6 (Fossen, 1994). Onde I_x, I_y e I_z são os momentos de inercia em torno dos eixos dados pelas equações 1.7, 1.8 e 1.9, e m a massa do corpo dada pela equação 1.10, onde ρ_A a densidade de massa do corpo.

Tabela 1.2: Variáveis do sistema dinâmico (ALMEIDA, 2014).

	Posição e Ângulo	Velocidade	Força/Momento
Movimento na direção X	x	u	X
Movimento na direção Y	y	v	Y
Movimento na direção Z	z	w	Z
Rotação em relação a X	ϕ	p	K
Rotação em relação a Y	θ	q	M
Rotação em relação a Z	ψ	r	N

$$\begin{cases} m[\dot{u} - ur - wq - x_g(r^2 + q^2) + y_g(pq - \dot{r}) + z_g(pr - \dot{q})] = X & (1.1) \\ m[\dot{v} + wp + ur - y_g(r^2 + p^2) + z_g(qr - \dot{p}) + x_g(qp - \dot{r})] = Y & (1.2) \\ m[\dot{w} - uq + vp - z_g(p^2 + q^2) + x_g(rp - \dot{q}) + y_g(rq + \dot{p})] = Z & (1.3) \\ I_x\dot{p} + (I_z - I_y)qr + m[y_g(\dot{w} - uq + vp) - z_g(\dot{v} - uq + ur)] = K & (1.4) \\ I_y\dot{q} + (I_x - I_z)rp + m[z_g(\dot{u} - vr + wq) - z_g(\dot{v} - uq + vq)] = M & (1.5) \\ I_z + (I_y + I_x)pq + m[x_g(\dot{v} - wp + ur) - y_g(\dot{u} - vr + wq)] = N & (1.6) \end{cases}$$

$$\begin{cases} I_x = \int_v (y^2 + z^2)\rho_A dV & (1.7) \\ I_y = \int_v (x^2 + z^2)\rho_A dV & (1.8) \\ I_z = \int_v (x^2 + y^2)\rho_A dV & (1.9) \end{cases}$$

$$m = \int_v \rho_A dV \quad (1.10)$$

Desconsidera-se a ocorrência de ondulações no plano da água por onde governa o veículo aquático para simplificar as equações de sua dinâmica, dessa forma alguns graus de liberdade podem ser desconsiderados. Faz-se então $w = p = q = \dot{w} = \dot{p} = \dot{q} = 0$ e o sistema de equações fica reduzido as equações 1.11, 1.12 e 1.13.

$$\begin{cases} m(\dot{u} - vr - x_g r^2) = X & (1.11) \\ m(\dot{v} - ur - x_g r^2) = Y & (1.12) \\ I_z\dot{r}mx_g(v + ur) = N & (1.13) \end{cases}$$

Baseado no sistema de equações simplificado, foi desenvolvido o conjunto de equações para representar a dinâmica do veículo, considerando o formato de trimarã e o posicionamento dos propulsores fora da linha central. Na Figura 1.3 está representado um esboço da vista superior do veículo, para facilitar a análise. A dinâmica da embarcação é então descrita pelas equações 1.14, 1.15 e 1.16 (ALMEIDA, 2014).

Nota-se que o movimento do barco é governado somente pelos empuxos desenvolvidos pelos motores M_1 e M_2, que aplicam as forças T_1 e T_2, respectivamente. Os motores são simétricos à linha central da embarcação, caso os valores das forças T_1 e T_2 sejam iguais a embarcação desenvolve velocidade no eixo x, caso haja diferença entre as forças cria-se um torque forçando a embarcação a girar.

$$\begin{cases} m\dot{u} = T_1 + T_2 + F_x + mvr - D_h u - 2D_m u & (1.14) \\ m\dot{v} = L_{HT} v - 2L_{AT} v + F_y - 2L_M v - mur & (1.15) \\ I_z \dot{r} = -T_1 d_{YM} + T_2 d_{YM} - B_r - B_{Hv} v - B_{av} v - B_M v & (1.16) \end{cases}$$

Onde,

D_H e D_M são os arrastos no sentido do eixo $X_\circ$ devido ao corpo principal do veículo e os motores, respectivamente;

L_{HT} composição do arrasto no sentido do eixo $Y_\circ$ devido ao corpo principal do veículo;

L_{AT} composição do arrasto no sentido do eixo $Y_\circ$ devido às amas ;

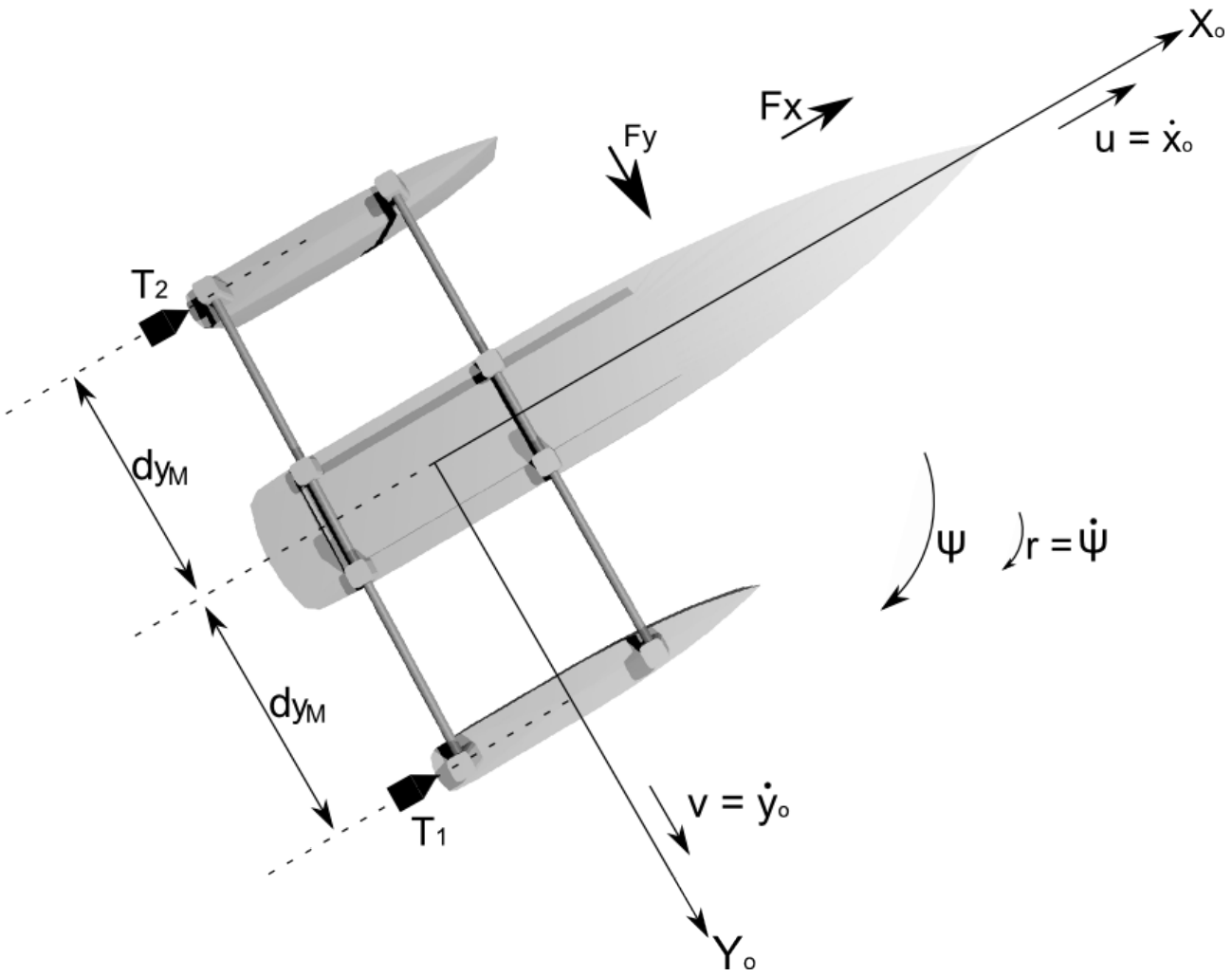
L_M composição do arrasto no sentido do eixo $Y_\circ$ devido aos motores ;

$B_{Hv} = L_{H1} d_{LH1} - L_{H2} d_{LH2}$, onde L_{H1} e L_{H2} são as porções do arrasto do corpo principal à e atrás do centro de massa, respectivamente, d_{LH1} e d_{LH2} a distância dessas porções ao centro de massa do veículo;

$B_{Av} = 2(L_{A1} d_{LA1} - L_{A2} d_{LA2})$, igualmente a L_{H1} e L_{H2}, L_{A1} e L_{A2} são as porções do arrasto das amas, com d_{LA1} e d_{LA2};

$B_r = L_{h1} d_{LH1}^2 + L_{H2} d_{LH2}^2 + 2(L_{A1} d_{LA1}^2 + L_{A2} d_{LA2}^2)$.

Figura 1.3: Visão superior da embarcação com eixos (ALMEIDA, 2014).



1.3 Simulação Computacional

A simulação computacional foi uma ferramenta científica pioneira na meteorologia e na física quântica logo após o término da segunda guerra mundial e desde então tornou-se indispensável no crescimento de diversas áreas de pesquisas. As áreas que usam intensivamente simulação computacional têm crescido, que incluem astrofísica, ciência dos materiais, engenharia, mecânica dos fluidos, economia, medicina, e muitas outras.

De modo grosseiro pode-se definir a simulação computacional como um programa que é executado em um computador, onde este programa utiliza métodos iterativos para explorar o comportamento aproximado de um modelo matemático (WINSBERG, 2013). Porém a simulação computacional vai além, para se realizar simulações que forneçam resultados confiáveis

é necessário seguir um processo meticuloso. Primeiro escolha-se o modelo matemático adequado para o sistema estudado, depois defini-se a melhor maneira de implementá-lo em um computador e por último executar o algoritmo e estudar os dados resultantes.

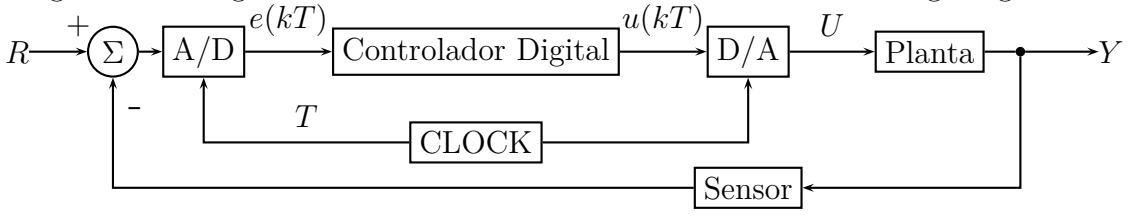
Neste trabalho a simulação computacional é fundamental, desenvolver um sistema de controle para uma embarcação sem simulação prévia pode ser uma tarefa árdua. Ambientes adequados para testes são custosos e disponíveis por poucas horas diárias, o que leva a um aumento de carga horária para a conclusão do projeto. Sem um prévio conhecimento da dinâmica da embarcação e da resposta do sistema de controle, os erros de testes podem ser mais graves levando a danos irreversíveis à embarcação.

1.4 Controladores Digitais

Decorrente da revolução do custo-eficiência dos computadores digitais a maioria dos sistemas de controle hoje em dia os utilizam para a implementação dos controladores. Controladores digitais dão aos desenvolvedores maior flexibilidade para mudanças nas leis de controle do sistema depois que o *hardware* foi implementado, pois as fórmulas que calculam os sinais de controles fazem parte do *software*. Muitas vezes, isso significa que desenvolvimento do *hardware* e do *software* são praticamente independentes (FRANKLIN; POWELL; EMAMI-NAENI, 2009).

A eletrônica analógica pode integrar e diferenciar sinais. Para um controlador digital efetuar tais tarefas, as equações diferencias devem ser aproximadas reduzindo-as em equações algébricas envolvendo soma, divisão e multiplicação. Além disso os controladores digitais se diferenciam dos analógicos em que os sinais devem ser amostrados e quantizados. Um sinal para ser usado em lógica digital primeiramente deve ser convertido por um conversor A/D (*analog-to-digital*) em um número digital quantizado. Uma vez que é calculado o próximo valor do sinal de controle, este deve ser extrapolado por um conversor D/A (*digital-to-analog*) para que seja aplicado ao atuador do processo. Como o sinal de controle não muda até a próxima amostra periódica a velocidade e largura de banda do controlador digital ficam restritamente limitadas, resultado direto da frequência de amostragem. A Figura 1.4 mostra o diagrama de blocos de um sistema com controlador digital genérico.

Figura 1.4: Diagrama de blocos de um sistema com controlador digital genérico.



Os principais tipos de controladores digitais são obtidos a partir de controladores analógicos. Para transformar uma função de transferência analógica em digital usa-se a transformada discreta de Fourier generalizada, nomeada de transformada z.

Dado um sinal $x(n)$ no tempo discreto, sabe-se que sua representação no tempo contínuo é $x_i(t) = \sum_{n=-\infty}^{\infty} x(n)\delta(t - nT)$, onde T é o período de amostragem. Sabe-se também que a

transformada de Fourier de $\delta(t - nT)$ é $e^{-j\Omega Tn}$, portanto a transformada de Fourier discreta direta e inversa são definidas pelas equações 1.17 e 1.18, respectivamente (DINIZ; SILVA; NETTO, 2004).

$$\begin{cases} X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} & (1.17) \\ x(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}d\omega & (1.18) \end{cases}$$

A partir da transformada discreta de Fourier define-se a transformada z de uma sequência $x(n)$ pela equação 1.19, onde z é uma variável complexa.

$$X(z) = \mathcal{Z}\{x(n)\} = \sum_{n=\infty}^{\infty} x(n)z^{-n} \tag{1.19}$$

Tendo definido a transformada z pode-se determinar o equivalente digital de um controlador analógico. Como os controladores analógicos geralmente são estudados como transformadas de Laplace, o primeiro problema a se resolver é descobrir o equivalente da variável z em s.

Uma das técnicas mais usadas é o método de Tustin's, ou comumente chamado de aproximação bilinear, que aproxima s em z.

Seja a função de transferência dada pela equação 1.20 e sua equivalente no tempo dada pela equação 1.21, onde fez-se $t = kT$ sendo T o período de amostragem.

$$\frac{U(s)}{E(s)} = \frac{1}{s} \tag{1.20}$$

$$u(kT) = \int_0^{kT-T} e(t)dt + \int_{kT-T}^{kT} e(t)dt \tag{1.21}$$

A equação 1.21 pode ser reescrita como $u(kT) = u(kT - T)$+área de $e(t)$ sob um período. Para o método de Tustin's utiliza-se a integração trapezoidal, isto é, a função $e(t)$ é aproximada por um segmento de reta a entre duas amostras. Aplicando então a integração trapezoidal na equação 1.21 obtém-se a equação 1.22.

$$u(kT) = u(kT - T) + \frac{T}{2}[e(kT - T) + e(kT)] \tag{1.22}$$

Basta aplicar a transformada z na equação 1.22 para chegar na equação 1.23. Por comparação entre as equações 1.21 e 1.23 concluí-se então que $s = \frac{2}{T}(\frac{1 - z^{-1}}{1 + z^{-1}})$.

$$\frac{U(z)}{E(z)} = \frac{1}{\frac{2}{T}(\frac{1 - z^{-1}}{1 + z^{-1}})} \tag{1.23}$$

Como exemplo, considere o controlador analógico mais utilizado, o PID com função de transferência dada pela equação 1.24. Aplicando-se o método de Tustin's finalmente obtém-se seu equivalente digital, cuja função de transferência é dada pela equação 1.25.

$$U(s) = (k_p + \frac{k_I}{s} + k_D s)E(s) \tag{1.24}$$

$$U(z) = (k_p + k_I \frac{T}{2}\frac{z+1}{z-1} + k_D \frac{2}{T}\frac{z-1}{z+1})E(z) \tag{1.25}$$

CAPÍTULO 2

DESENVOLVIMENTO E RESULTADOS

2.1 Simulador Numérico

A validação do controlador é via simulação. Utilizou-se um programa escrito em linguagem C para simular a dinâmica da embarcação junto com o controlador, tendo como base a *GNU Science Library* (http://www.gnu.org/software/gsl).

Como é mostrado nas equações 1.14, 1.15 e 1.16 , a simulação da dinâmica da embarcação trata-se da solução no tempo de um sistema de equações diferenciais de primeira ordem. Escolheu-se como método numérico para a resolução o de Runge-Kutta de quarta ordem. Ao invés de implementar o algorítimo utilizou-se funções já prontos da biblioteca *GSL _ Odeiv* pela facilidade e confiabilidade. Para se resolver o sistema de equações diferenciais utiliza-se a função
gls _ odeiv _ step_ apply. A cada iteração a função resolve o sistema para o tempo ($t_{inicial}$ + *passo*), até o valor de tempo desejado.

Antes de ser utilizado o simulador é testado para alguns casos intuitivos do comportamento da embarcação, para verificar se a dinâmica dela está sendo simulada de maneira satisfatória.

O primeiro teste é feito com os motores ligados a todo momento aplicando forças iguais e constantes, com a embarcação apontando para quatro pontos diferentes por 180 segundos. O resultado é mostrado nas figuras 2.1 e 2.2. Nota-se que como esperado para forças aplicadas pelos motores iguais e constantes, a embarcação atinge uma velocidade limite e a mantém.

Figura 2.1: Teste de aceleração da embarcação no simulador - Rota da embarcação

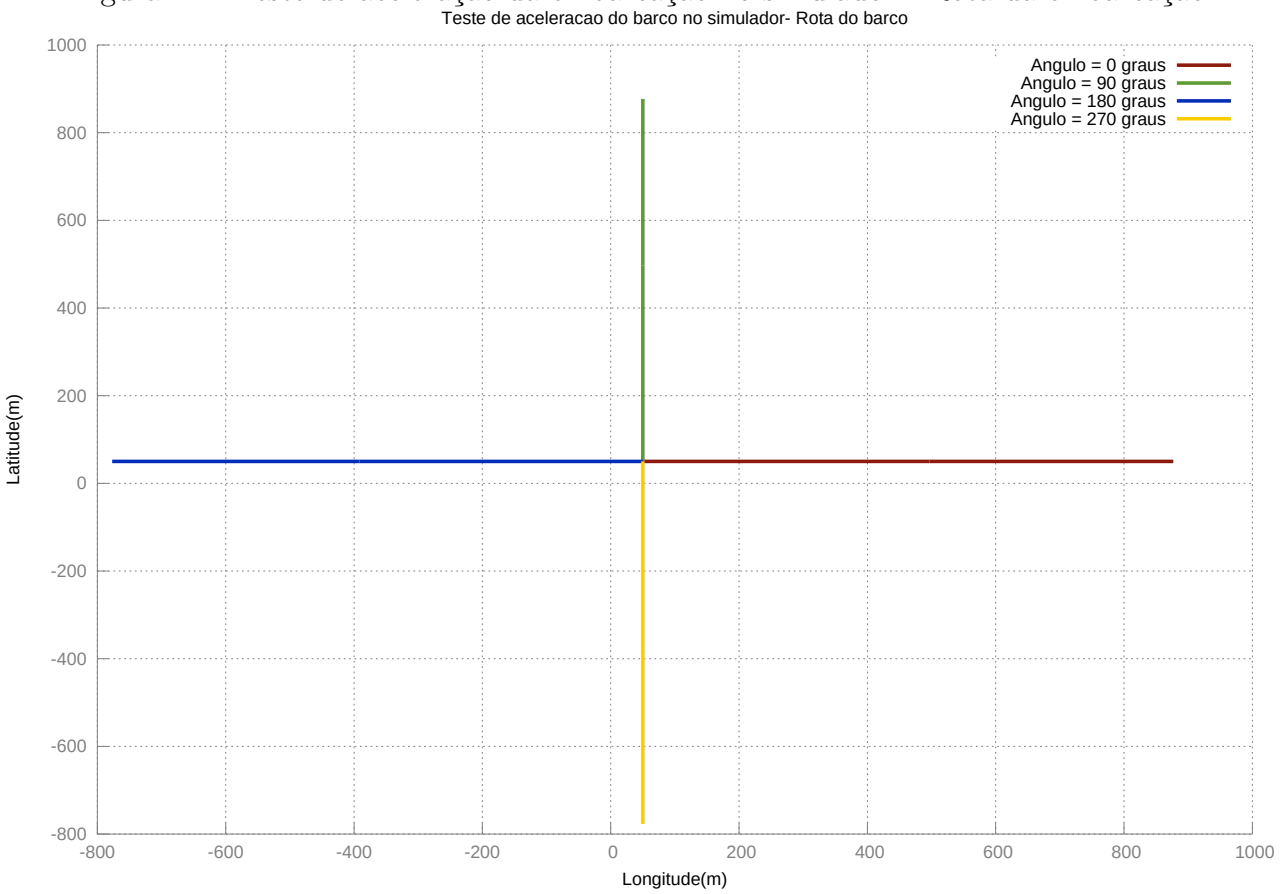
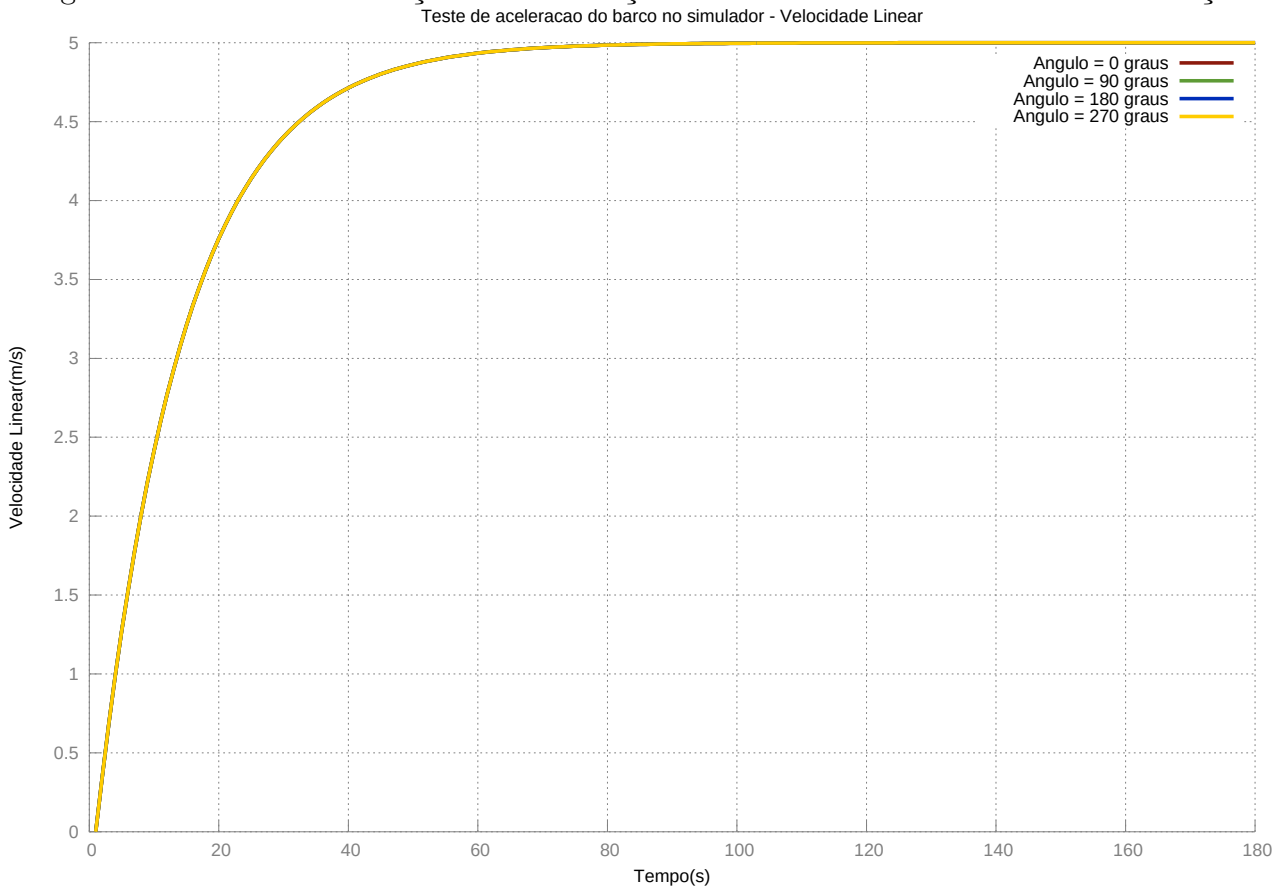


Figura 2.2: Teste de aceleração da embarcação no simulador - Velocidade da embarcação



O segundo teste é feito como o primeiro, porém os motores são desligados no instante $t = 100s$. O resultado é mostrado nas figuras 2.3 e 2.4. Nota-se que quando os motores são desligados o barco desacelera e para, como o esperado.

Figura 2.3: Teste de desaceleração da embarcação no simulador - Rota da embarcação

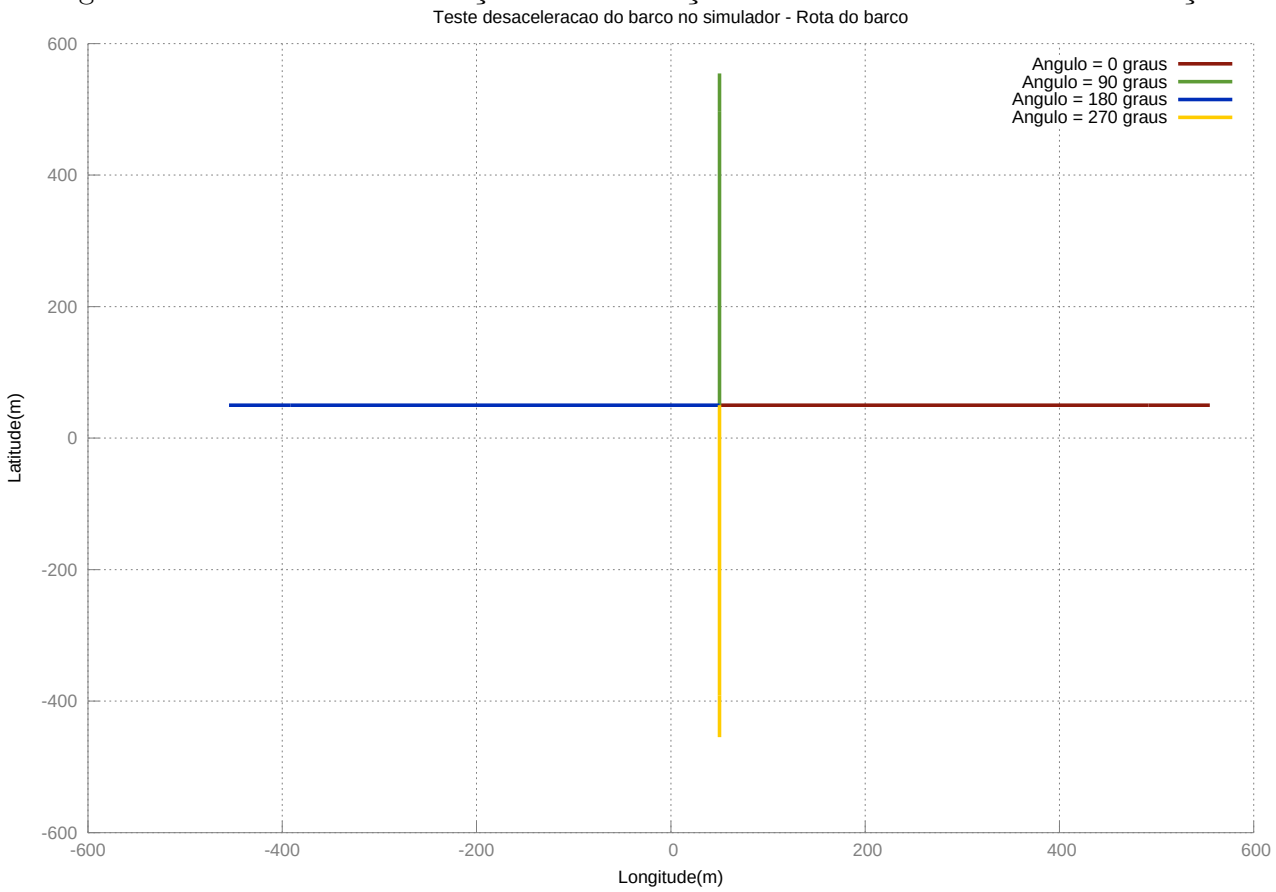
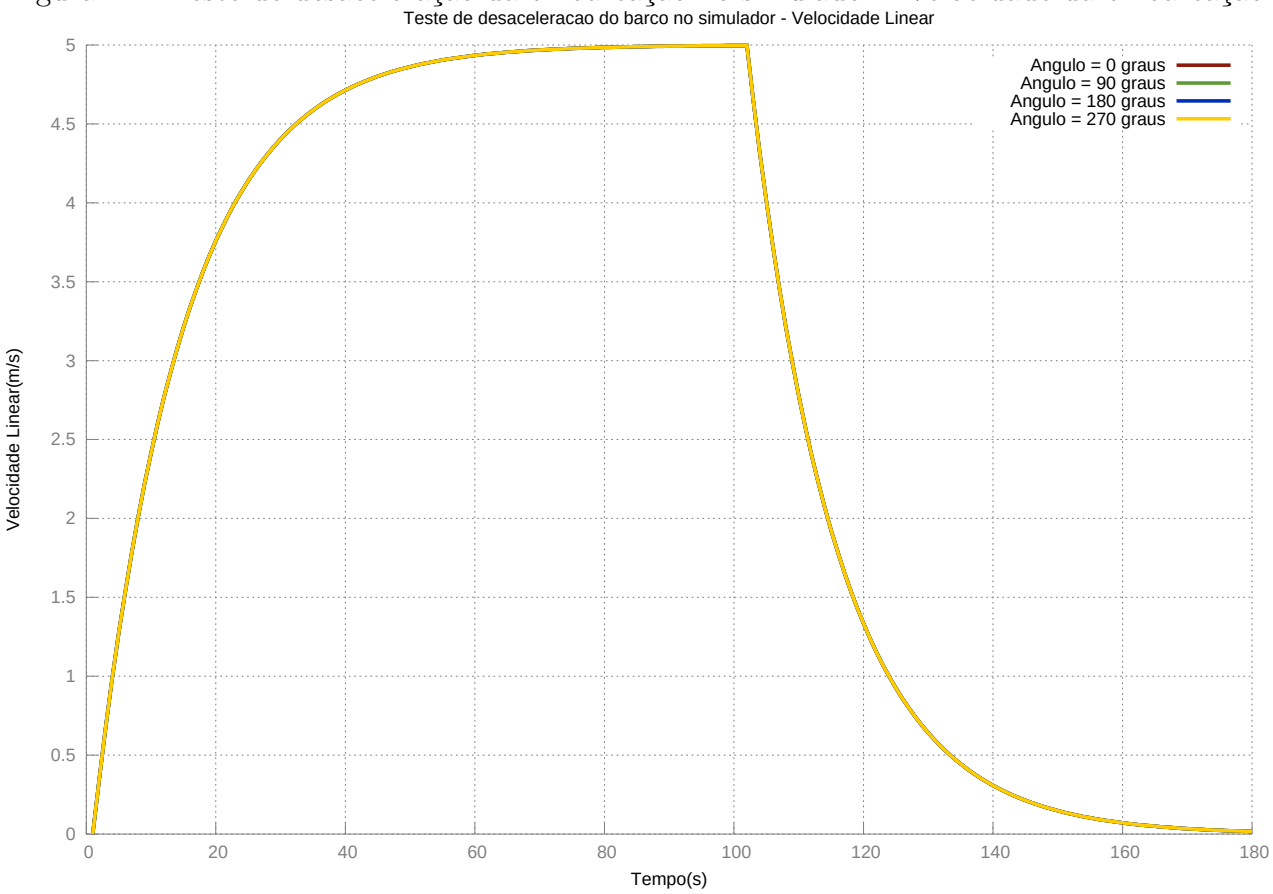


Figura 2.4: Teste de desaceleração da embarcação no simulador - Velocidade da embarcação



O terceiro teste é feito para avaliar o comportamento da embarcação em curvas. Somente um dos motores permanece ligado nos primeiros instantes, e no instante $t = 30s$ inverte-se. O resultado é mostrado nas figuras 2.5 e 2.6.

Figura 2.5: Teste de curva da embarcação no simulador - Rota da embarcação

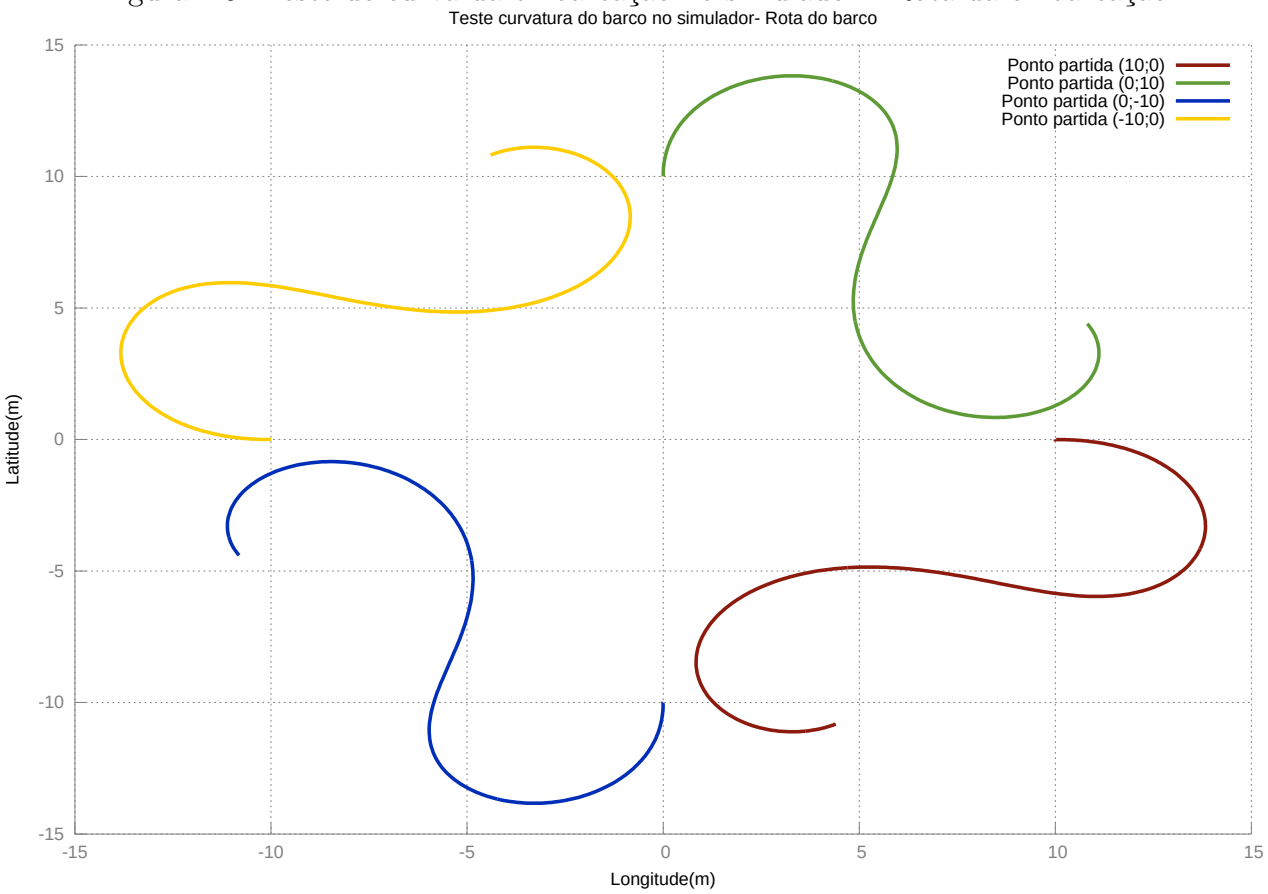
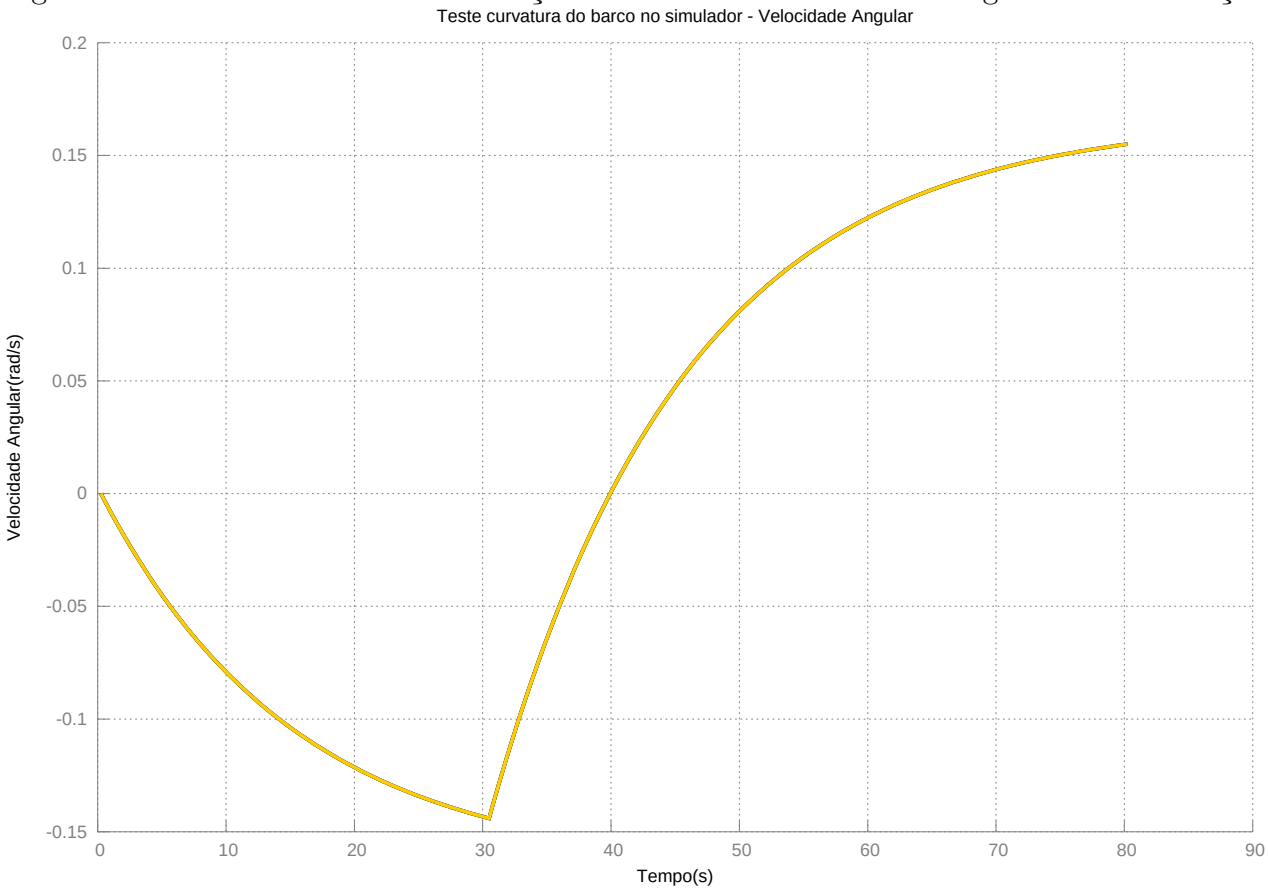


Figura 2.6: Teste de curva da embarcação no simulador - Velocidade angular da embarcação



O quarto teste é feito como o terceiro, porém deixa-se os motores desligados por um período curto de tempo antes de reverter qual motor que é deixado ligado. O resultado é mostrado nas figuras 2.7 e 2.8.

Figura 2.7: Teste de curva com pausa da embarcação no simulador - Rota da embarcação

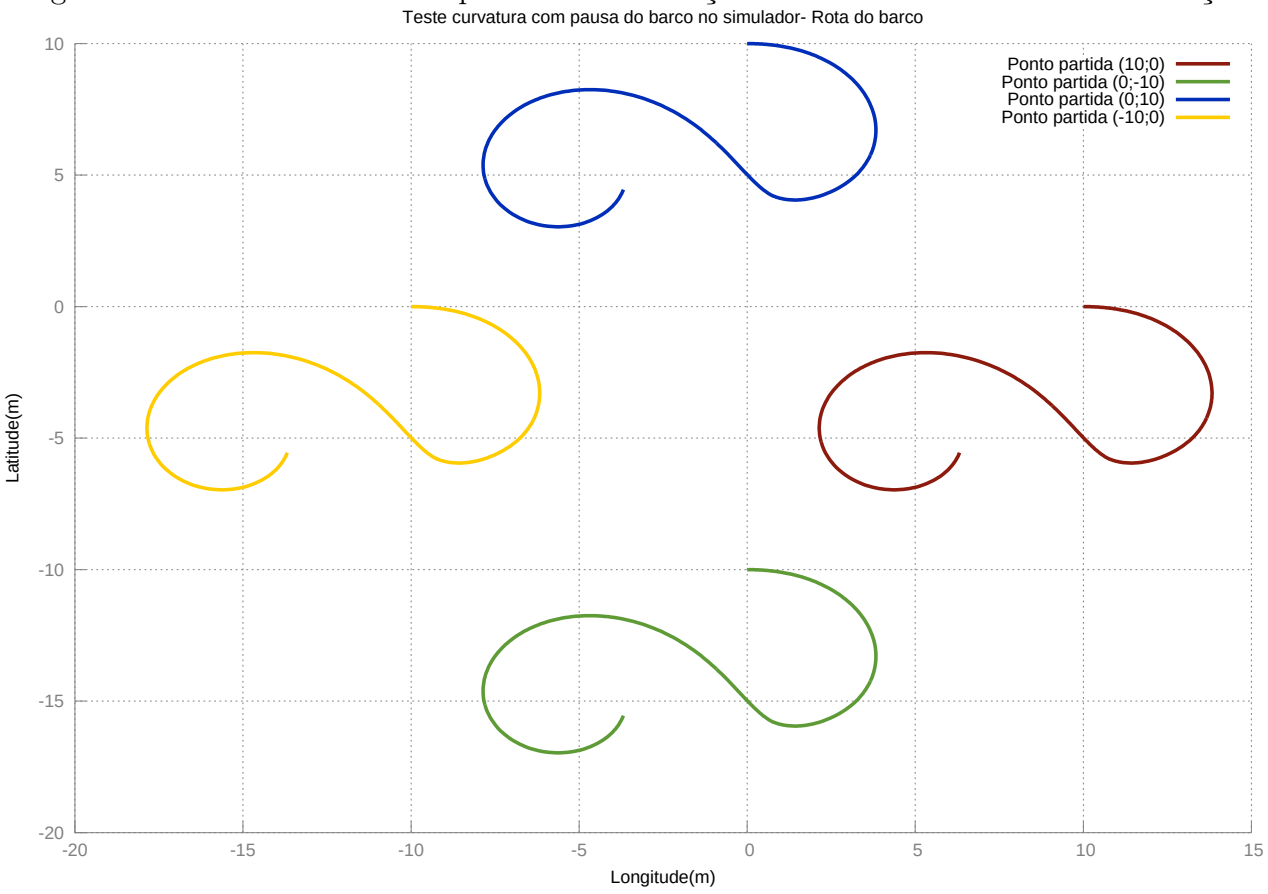
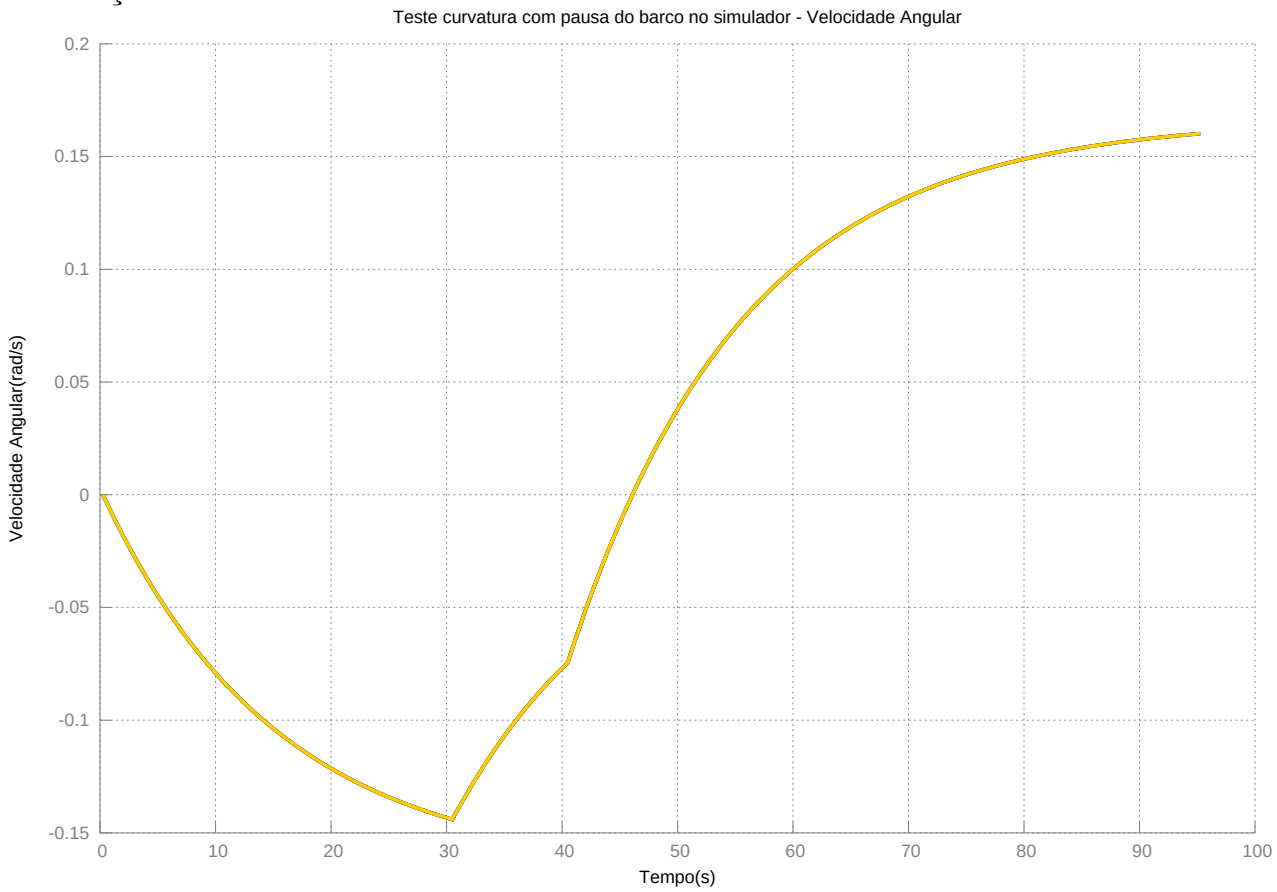


Figura 2.8: Teste de curva com pausa da embarcação no simulador - Velocidade angular da embarcação



A análise dos resultados obtidos concluí-se que o simulador é apropriado para a validação do sistema de controle e orientação da embarcação de forma segura, antes de implementá-lo fisicamente.

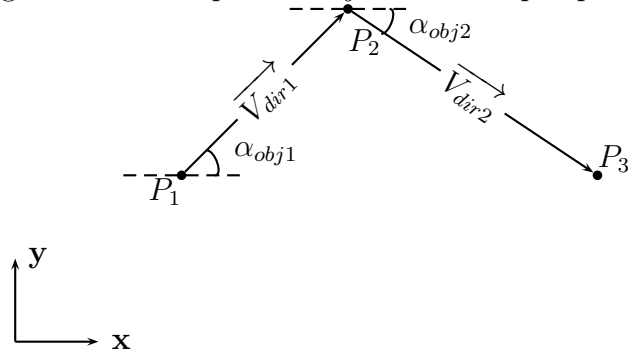
2.2 Sistema de Controle e Orientação

A embarcação funciona por meio missões, que são determinadas pelo usuário e carregadas no computador de bordo do veículo. As missões são descritas por pontos de coordenadas geográficas que formarão a trajetória que a embarcação deve seguir. Estas missões são escritas em um arquivo de texto nomeado como *MISSAO.txt* e deve conter os pontos de coordenadas geográficas em ordem direta do trajeto, ou seja, o primeiro ponto do arquivo será o início da missão e o último ponto do arquivo será o fim da missão. Esse arquivo deve ter a seguinte formatação, onde x é a longitude e y é a latitude:

x_0	y_0
.	.
.	.
.	.
x_n	y_n

Dado os pontos da missão são calculados os vetores direção da trajetória, $\overrightarrow{V_{dir}}$, pela equação 2.1. E tomando como referência o leste, sendo $\overrightarrow{V_{Leste}} = (1; 0)$, calcula-se os ângulos entre este com os vetores direção pela equação 2.2, são os ângulos objetivo α_{obj}. A Figura 2.9 mostra um exemplo de trajetória com três pontos.

Figura 2.9: Exemplo de trajetória dada por pontos.

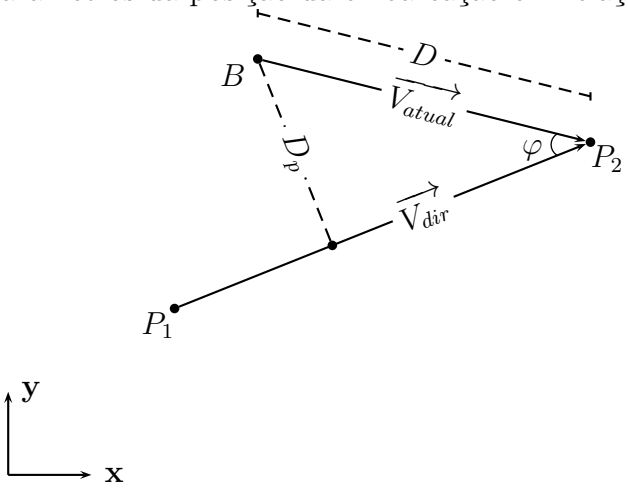


$$\begin{cases} \overrightarrow{V_{dir}} = \overrightarrow{P_1P_2} = P_2 - P1 & (2.1) \\ \alpha_{obj} = arccos(\dfrac{\overrightarrow{V_{dir}} \cdot \overrightarrow{V_{Leste}}}{\|\overrightarrow{V_{dir}}\| \cdot \|\overrightarrow{V_{Leste}}\|}) & (2.2) \end{cases}$$

Tendo os parâmetros da trajetória obtidos são calculados os parâmetros da posição da embarcação em relação a trajetória. Esses parâmetros são a distância entre a embarcação e a trajetória e a distância entre a embarcação e o ponto destino.

A distância entre a embarcação e a trajetória é dada por D_p e calculada com o auxilio do ângulo φ e do vetor $\overrightarrow{V_{atual}}$, dado pelas equações 2.3 e 2.4. Estes parâmetros são mostrados na Figura 2.10, onde o ponto B é a localização da embarcação.

Figura 2.10: Parâmetros da posição da embarcação em relação a trajetória.



$$\begin{cases} \varphi = arccos(\dfrac{\overrightarrow{V_{dir}} \cdot \overrightarrow{V_{atual}}}{||\overrightarrow{V_{dir}}|| \cdot ||\overrightarrow{V_{atual}}||}) & (2.3) \\ D_p = sin(\varphi) \cdot ||\overrightarrow{V_{atual}}|| & (2.4) \end{cases}$$

A distância entre a embarcação e o ponto destino é dada por D e é calculada pela equação 2.5.

$$D = ||\overrightarrow{V_{atual}}|| \tag{2.5}$$

Também é necessário saber em qual lado da trajetória a embarcação está. Para isto é feito um produto vetorial entre o $\overrightarrow{V_{teste}}$ e $\overrightarrow{V_{atual}}$, V_{Teste} é dado pela equação 2.6 e pode ser visto na figura 2.11. Nota-se que o resultado da coordenada z do produto vetorial tem sinal trocado em cada lado da trajetória. Então para referência troca-se o sinal de D_p de acordo com o sinal da coordenda z do produto vetorial, como nas equações 2.7 e 2.8. O sinal foi escolhido de acordo com a convenção da velocidade angular da embarcação.

$$\begin{cases} \overrightarrow{V_{Teste}} = \overrightarrow{P_1 B} = B - P_1 & (2.6) \\ \overrightarrow{V_{atual}} \times \overrightarrow{V_{Teste}} = (V_x, V_y, V_z) & (2.7) \\ D_p = D_p \dfrac{||V_z||}{V_z} & (2.8) \end{cases}$$

O ponto destino é atualizado caso a embarcação passe pela região de "chegada", um círculo de raio τ com centro no ponto destino atual. Caso não haja próximo ponto a missão é encerrada.

Todos parâmetros utilizados podem ser vistos na tabela 2.1 e são mostrados na Figura 2.11 para facilitar a análise[1], onde $\overrightarrow{B}$ é a direção que a embarcação está apontada.

[1]Nota-se que na Figura 2.11 o ângulo α_{atual} é contado a partir do oeste somente para fins estéticos, no programa desenvolvido começa-se contar a partir do leste.

Figura 2.11: Diagrama completo dos parâmetros utilizados no sistema de orientação da embarcação.

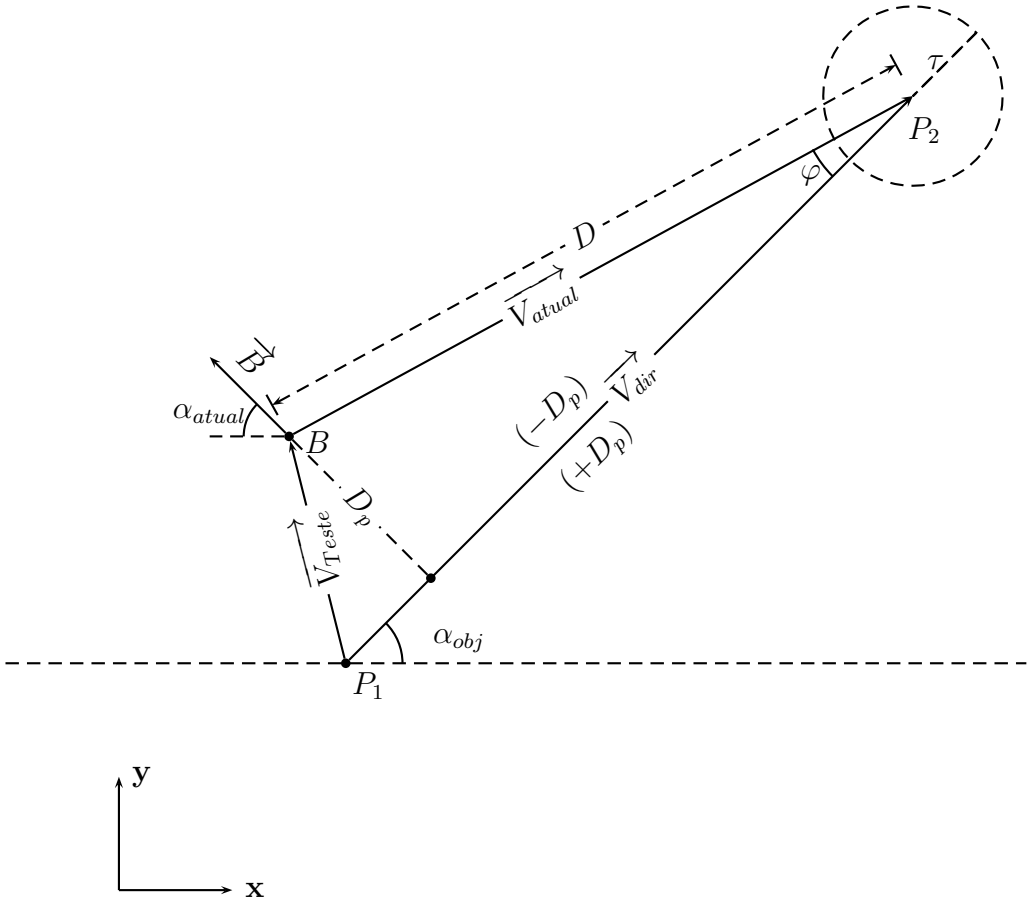


Tabela 2.1: Parâmetros utilizados no sistema de orientação.

Parâmetro	Descrição
α_{obj}	Ângulo de referência para paralelismo com a rota
α_{atual}	Ângulo absoluto da embarcação
φ	Ângulo entre $\overrightarrow{V_{atual}}$ e $\overrightarrow{V_{dir}}$
D_p	Distância entre embarcação e a rota
D	Distância entre embarcação e o ponto destino
τ	Raio da região de "chegada"
$\overrightarrow{V_{dir}}$	Vetor de direção do trajeto atual
$\overrightarrow{V_{atual}}$	Vetor de direção embarcação ao ponto destino
$\overrightarrow{V_{teste}}$	Vetor auxiliar

A partir destes parâmetros é feita o controle da embarcação. O objetivo do controlador é guiar a embarcação para a trajetória estabelecida pela missão até a sua conclusão, para qualquer ponto que a embarcação esteja.

Sabe-se que a velocidade angular e linear do barco são dadas pelas equações 2.9 e 2.10 respectivamente, onde k_{r1}, k_{r2}, k_{u1} e k_{u2} são constantes de proporcionalidade. Como a veloci-

dade angular é governada pela diferença das forças T_1 e T_2, faz-se o controlador para definir a tendência de giro da embarcação e em seguida calcula-se tais forças.

$$\begin{cases} r = T_1 k_{r1} - T_2 k_{r2} & (2.9) \\ u = T_1 k_{u1} + T_2 k_{u2} & (2.10) \end{cases}$$

A referência é α_{obj}, então o controlador fará com que a embarcação gire até que o ângulo $\alpha_{atual} = \alpha_{ob}$. Porém isso não é suficiente, caso a embarcação esteja longe da trajetória definida ele entrará numa outra trajetória paralela a esta. Para que a embarcação se alinhe a trajetória é imposto um *offset* para o ângulo absoluto da embarcação, α_{atual}.

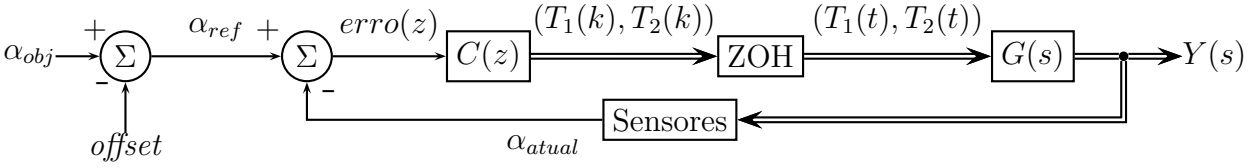
O *offset* tem que ser tal que quando a distância D_p seja grande o controlador guie a embarcação perpendicularmente à trajetória, e a medida que D_p diminui o *offset* também diminua para que a embarcação se alinhe a trajetória. Quer-se então uma função que resulte $\frac{\pi}{2}$ quando D_p muito grande, e próximo de 0 quando D_p for muito pequeno. A função escolhida para o *offset* é $\arctan(D_p)$.

Dado o ângulo objetivo α_{obj} e o *offset*, a referência para o controlador é dada pela equação 2.11 e o erro que alimenta o controlador pela equação 2.12. Nota-se que neste caso o sinal de D_p é bastante importante. Em um lado da trajetória o *offset* será $-\frac{\pi}{2}$ e do outro $\frac{\pi}{2}$, para D_p muito grande. Se não houvesse essa diferença a embarcação sempre seria guiada com um *offset* de $\frac{\pi}{2}$ e se estivesse do lado da errado da trajetória seria levada para longe dela perpendicularmente.

O sistema de controle pode ser resumido pelo diagrama de blocos da Figura 2.12, onde ZOH é um bloco *Zero-Order-Holder* para converter o sinal discreto do controlador em analógico e $G(s)$ é a planta do sistema.

$$\begin{cases} \alpha_{ref} = \alpha_{obj} - \arctan(K_{Dp} D_p) & (2.11) \\ E(z) = \alpha_{ref} - \alpha_{atual} & (2.12) \end{cases}$$

Figura 2.12: Diagrama de blocos do sistema de controle simplificado.



Utiliza-se um controlador do tipo proporcional derivativo (PD) para o sistema de controle. A escolha desse tipo de controlador é devido a dinâmica do barco e a estratégia de controle utilizada. Devido ao ambiente em que a embarcação é governada necessita-se atenção em acelerações exageradas, a desaceleração devido somente ao contato com a superfície da água acontece lentamente e por isso adiciona-se o efeito derivativo ao controlador. Não há necessidade de adicionar-se efeito integrativo ao controlador, basta observar que como o controle é feito pelo ângulo absoluto do barco e o sinal de controle gera velocidade angular, o efeito proporcional funciona como um efeito integrativo nesse caso.

A função de transferência analógica de um controlador do tipo PD é dada pela equação 2.13, e sua equivalente discretizada pelo método bilinear é dada pela equação 2.14.

$$\frac{U(s)}{E(s)} = (K_p + K_d s) \tag{2.13}$$

$$\frac{U(z)}{E(z)} = \frac{K_p(1+z^{-1}) + \frac{2}{T}K_d(1-z^{-1})}{(1+z^{-1})} \tag{2.14}$$

Definiu-se que o sinal de controle seria a tendência da embarcação girar. Pela equação 2.12 o erro pode admitir valores positivos ou negativos, dependendo se $\alpha_{obj} > \alpha_{atual}$ ou se $\alpha_{obj} < \alpha_{atual}$. Portanto o sinal do erro dita para qual lado a embarcação gira. Seguindo a convenção da velocidade angular da Figura 1.3 e pelas as equações 2.9 e 2.10, acham-se as forças T_1 e T_2 que os motores devem produzir. Essas forças são dadas pelas equações 2.15 e 2.16, onde K_{vref} determina a velocidade linear máxima constante que a embarcação deve permanecer.

$$\begin{cases} T_1(k) = K_{vref} - u(k) & (2.15) \\ T_2(k) = K_{vref} + u(k) & (2.16) \end{cases}$$

Também é necessário um saturador para limitar a velocidade angular da embarcação. Satura-se então a tendência de giro dada por $u(k)$. Os propulsores que são utilizados não funcionam de modo reverso, ou seja, não admitem valores T_1 e T_2 negativos. Desse modo $u(k)$ pode admitir valores no intervalo: $-K_{vel} \leq u(k) \leq K_{vel}$. A equação recursiva de $u(k)$ é dada por 2.17, e saturada pelas equações 2.18, 2.19 e 2.20, onde K_{ulim} é a tendência de giro máxima imposta.

$$u(k) = e(k)(K_p + \frac{2}{T}K_d) + e(k-1)(K_p - \frac{2}{T}K_d) - u(k-1) \tag{2.17}$$

$$\text{u(k)}= \begin{cases} u(k); \text{caso } -K_{ulim} \leq u(k) \leq K_{ulim} & (2.18) \\ K_{ulim}; \text{caso } u(k) \geq K_{ulim} & (2.19) \\ -K_{ulim}; \text{caso } u(k) \leq -K_{ulim} & (2.20) \end{cases}$$

O controlador PD é sintonizado manualmente com auxílio do simulador numérico, alterando-se as constantes K_p, K_d, K_{vel}, K_{ulim}, τ e K_{Dp} e também a frequência de amostragem $f_a = \frac{1}{T}$.

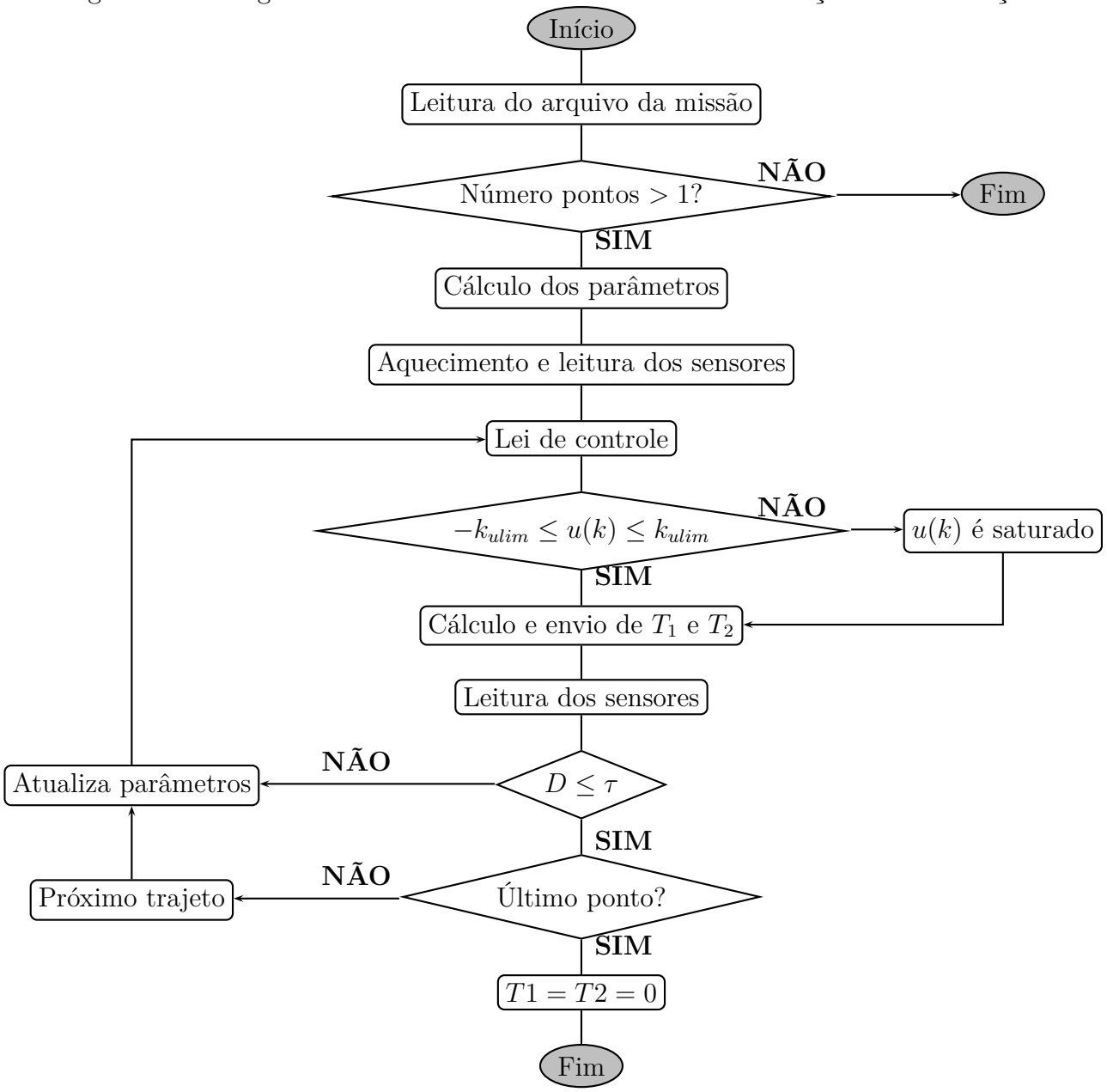
Chegou-se que a melhor performance é obtida quando utiliza-se os valores das constantes e frequência de amostragem mostrados na tabela 2.2.

Tabela 2.2: Valores das constantes do sistema de controle e orientação.

K_{vel}	3,6
K_p	8
K_d	0,008
K_{ulim}	2
K_{Dp}	0,12
f_a	4Hz
τ	10m

O programa executado para o sistema de controle e orientação segue a lógica do diagrama de fluxo mostrado na Figura 2.13.

Figura 2.13: Diagrama de fluxo do sistema de controle e orientação da embarcação.



2.2.0.1 Resultados

Os testes são feitos no simulador desenvolvido. Escolheu-se para os testes duas missões para a avaliação do controlador. A missão 1 formada por trajetos pequenos e ângulos entre trajetos grandes. E a missão 2 formada por um único trajeto bastante longo. Os pontos geográficos da missão 1 são mostrados na tabela 2.3 e os da missão 2 na tabela 2.4.

Tabela 2.3: Pontos geográficos da missão teste 1.

Longitude(m)	Latitude(m)
0	0
300	50
500	-50
450	-200
0	-200
0	0

Tabela 2.4: Pontos geográficos da missão teste 2.

Longitude(m)	Latitude(m)
0	0
1500	50

Para o controle da embarcação utiliza-se o controlador tipo PD desenvolvido dado pelas equações 2.15 e 2.16. Os resultados da missão 1 são mostrados pelas figuras 2.14, 2.15, 2.16 e 2.17. E os resultados da missão 2 são mostrados pelas figuras 2.18, 2.19, 2.20 e 2.21.

Os parâmetros de referências para o sistema de controle calculados da missão teste 1 são mostrados na tabela 2.5. A tabela 2.6 mostra os valores dos instantes de tempo em que a embarcação passa pelos pontos de destino para a missão teste 1.

Tabela 2.5: Parâmetros de referência do sistema de controle calculados para a missão teste 1.

Parâmetro \ Trajeto	1°	2°	3°	4°	5°
V_{dir}	$(300; 50)$	$(300; -150)$	$(-250; -100)$	$(-350; 0)$	$(0; 200)$
α_{obj} (rad)	0,165	-0.464	-2,716	3,142	1,570

Tabela 2.6: Instantes de tempo em que a embarcação passa pelos pontos da missão de teste 1.

Ponto	Instante de tempo (s)
1°	90
2°	157
3°	220
4°	370
5°	449

Figura 2.14: Trajeto da missão teste 1 com controlador PD.

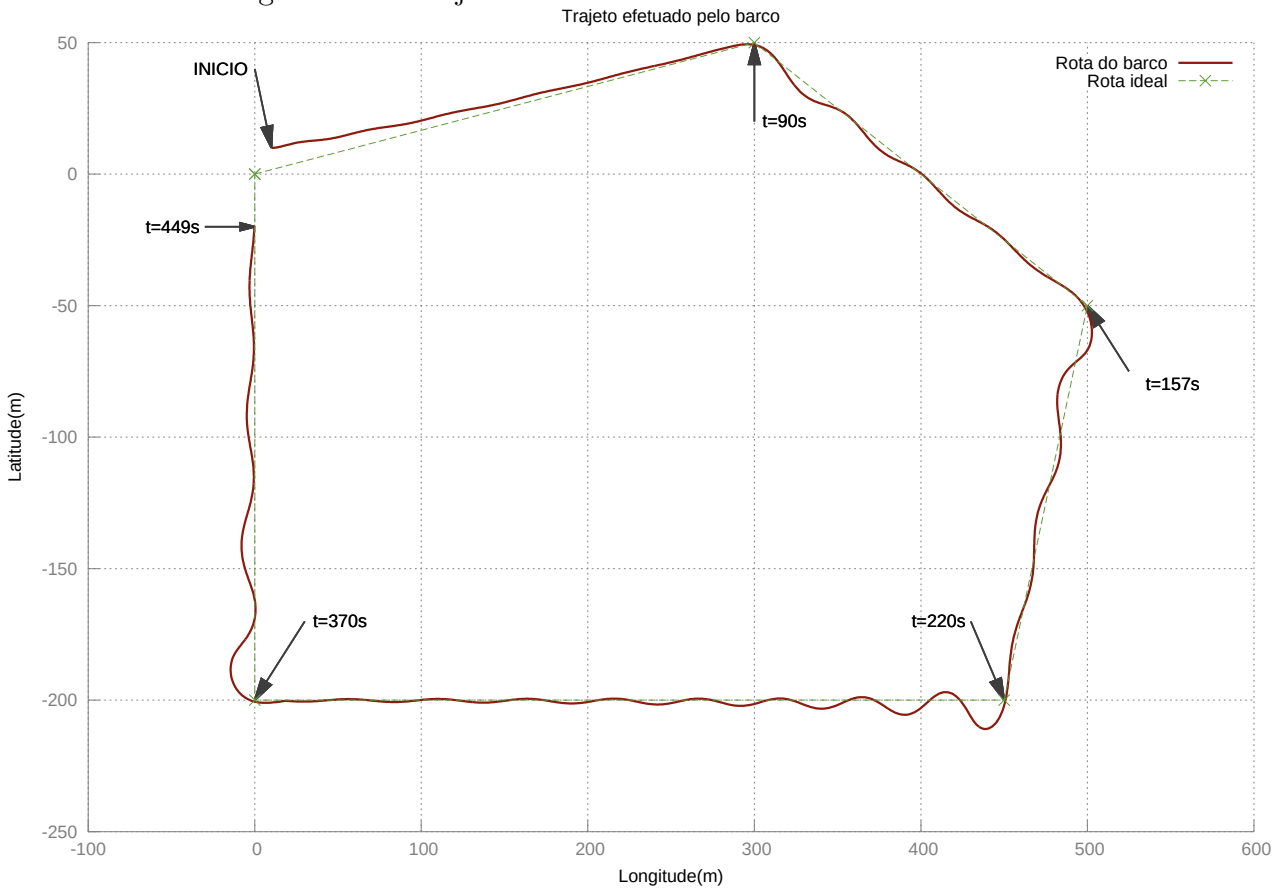


Figura 2.15: Sinal de erro para missão teste 1 e controlador PD.

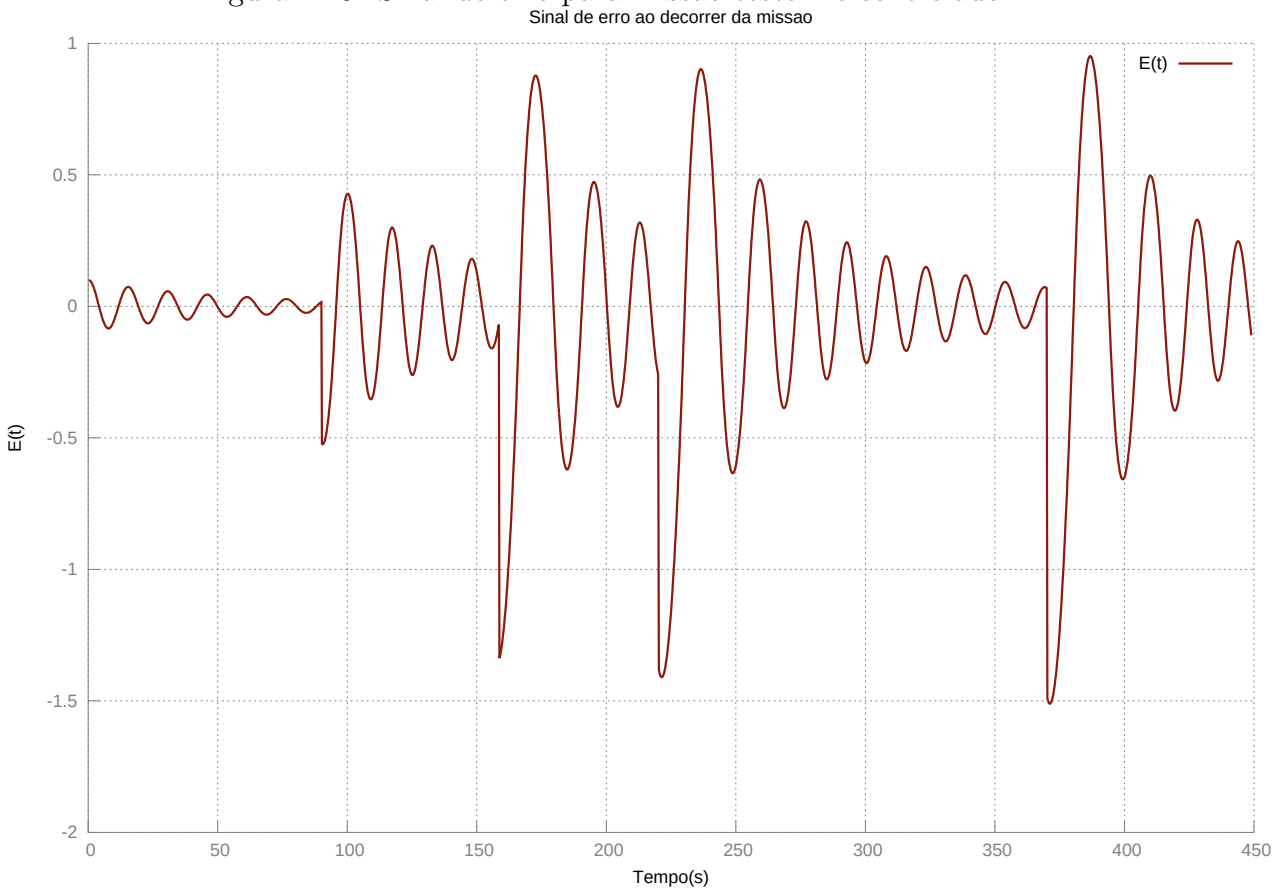


Figura 2.16: Sinais de controle para missão teste 1 e controlador PD.

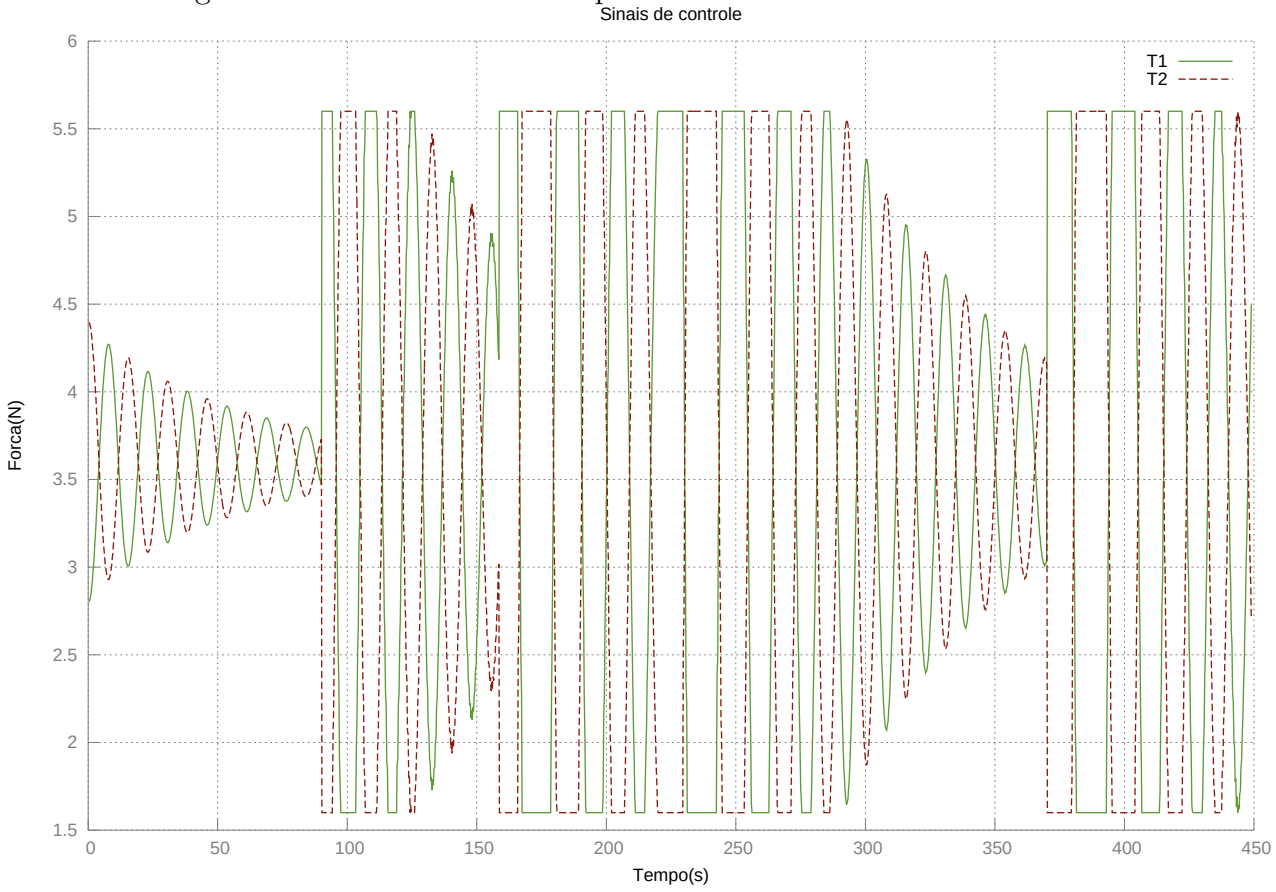


Figura 2.17: Velocidade desenvolvida para missão teste 1 e controlador PD.

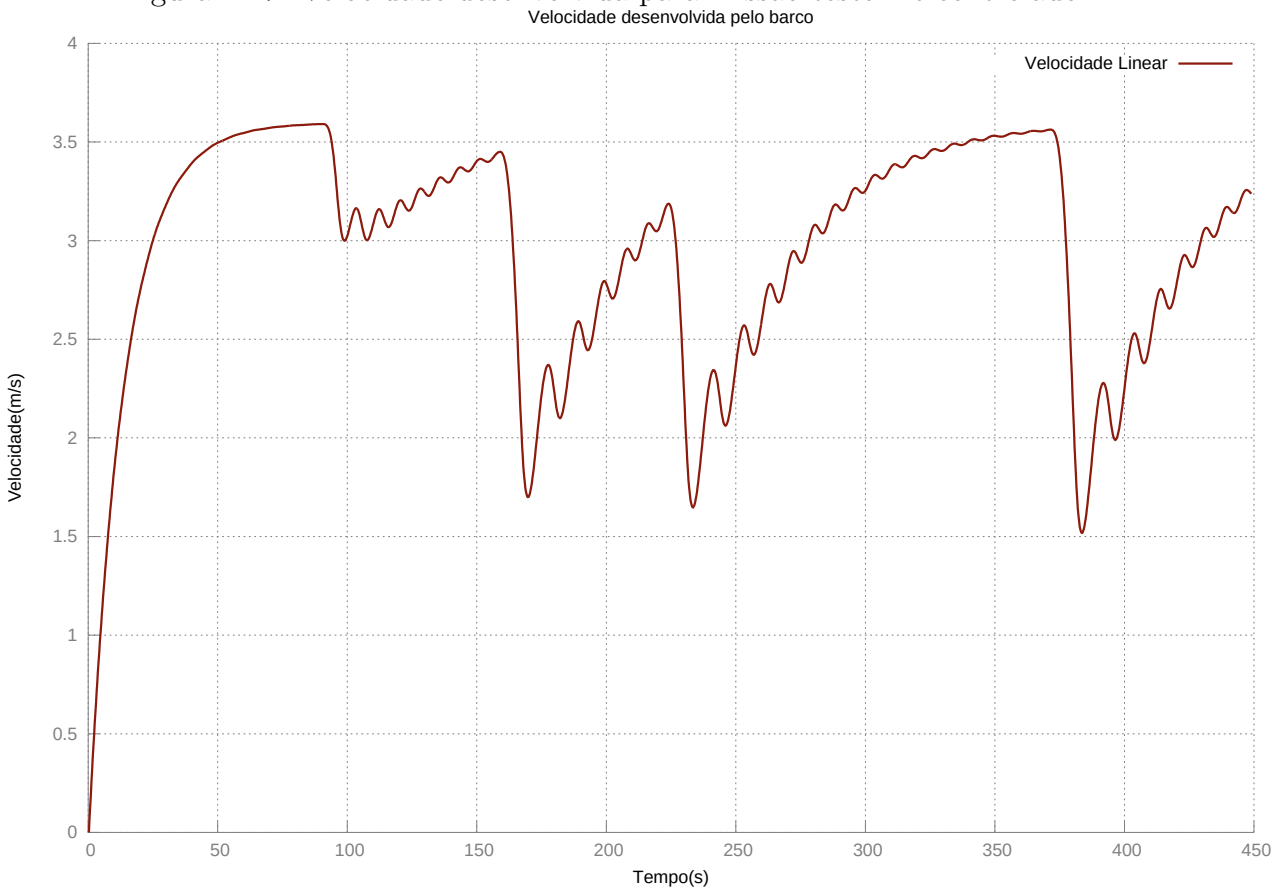


Figura 2.18: Trajeto da missão teste 2 com controlador PD.

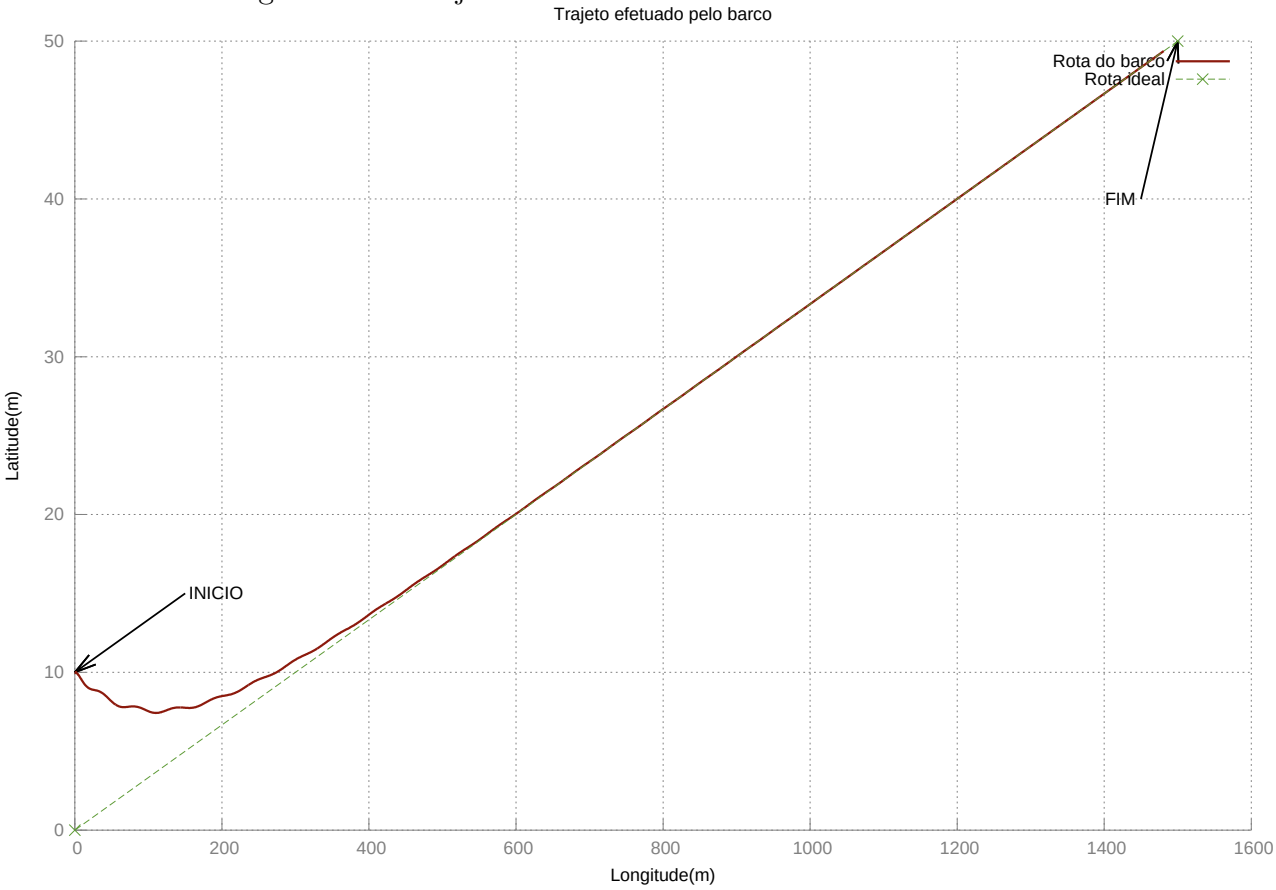


Figura 2.19: Sinal de erro para missão teste 2 e controlador PD.

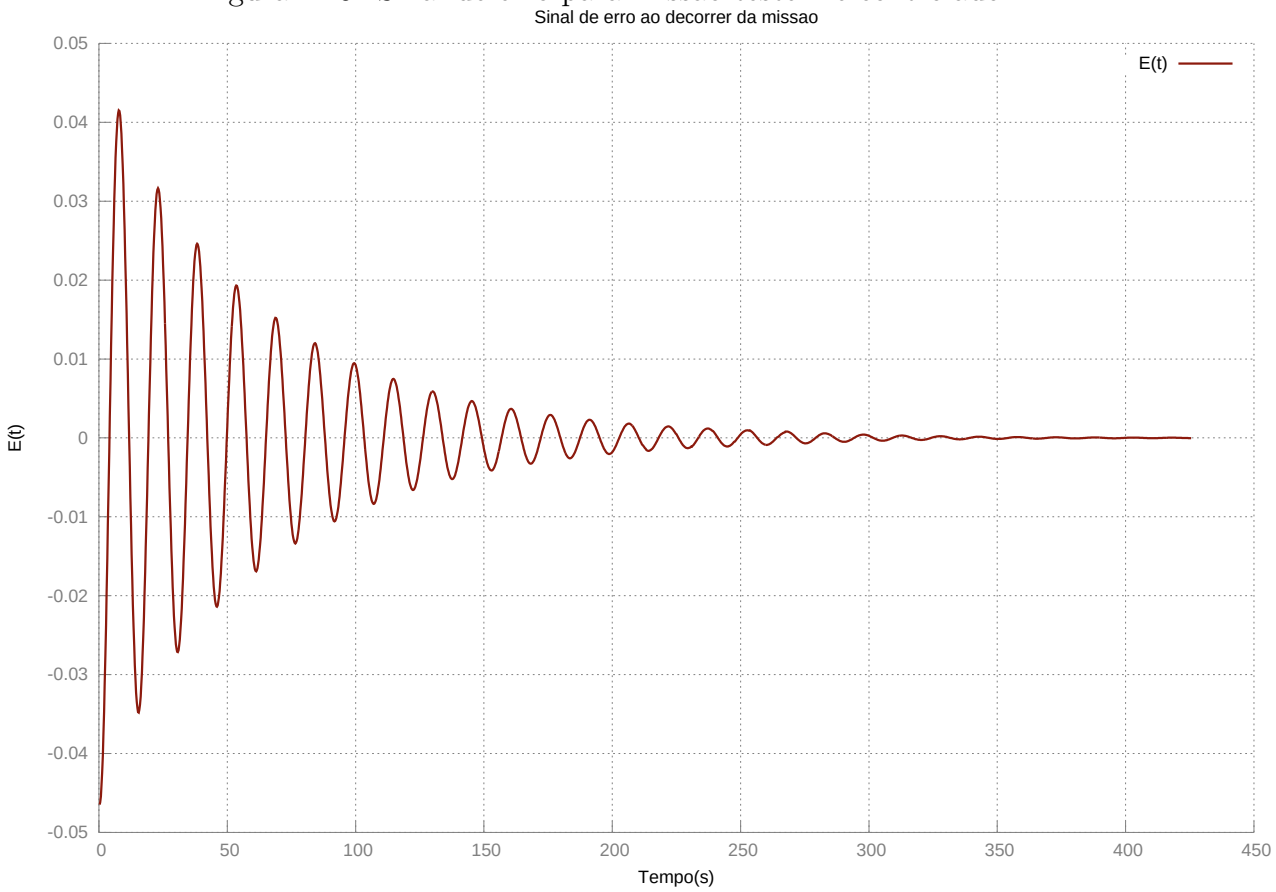


Figura 2.20: Sinais de controle para missão teste 2 e controlador PD.

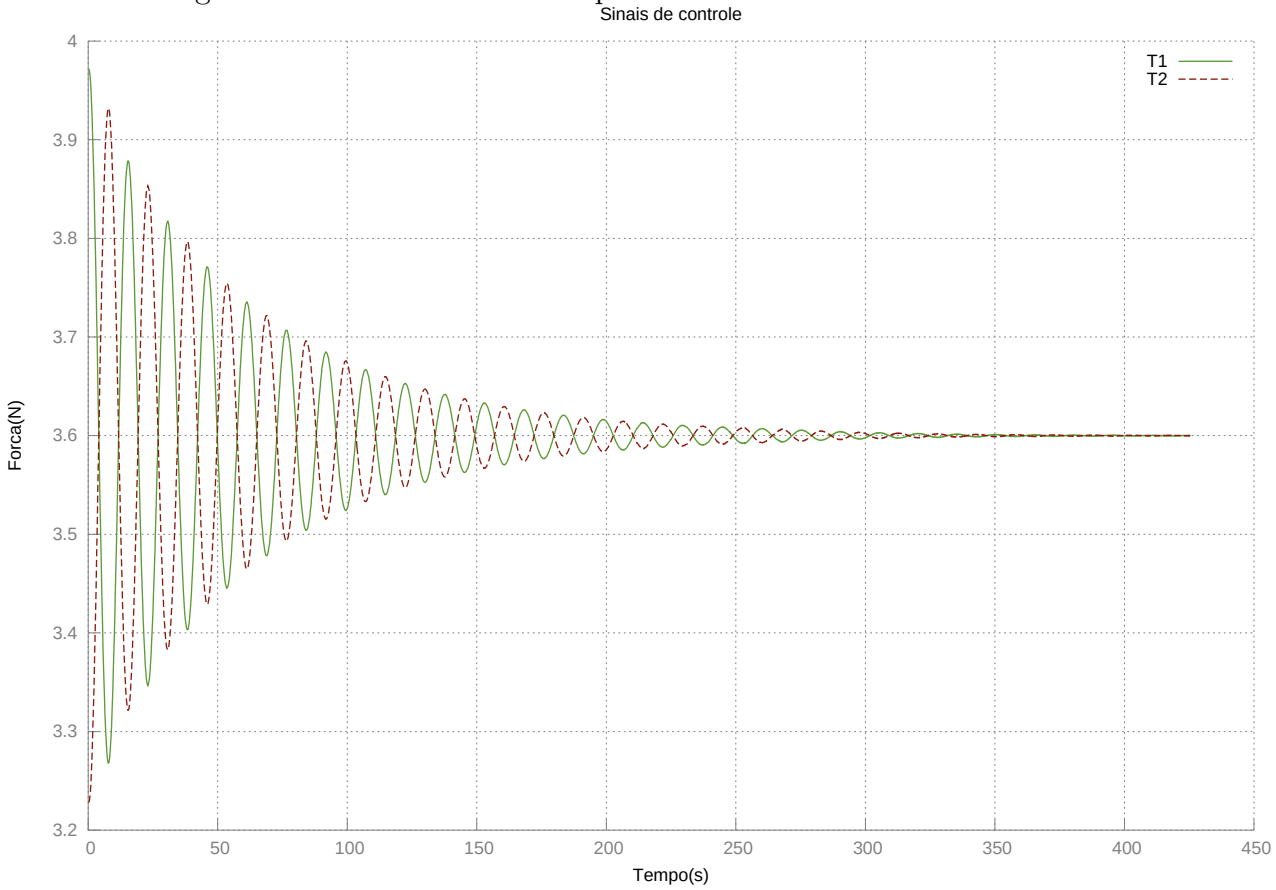
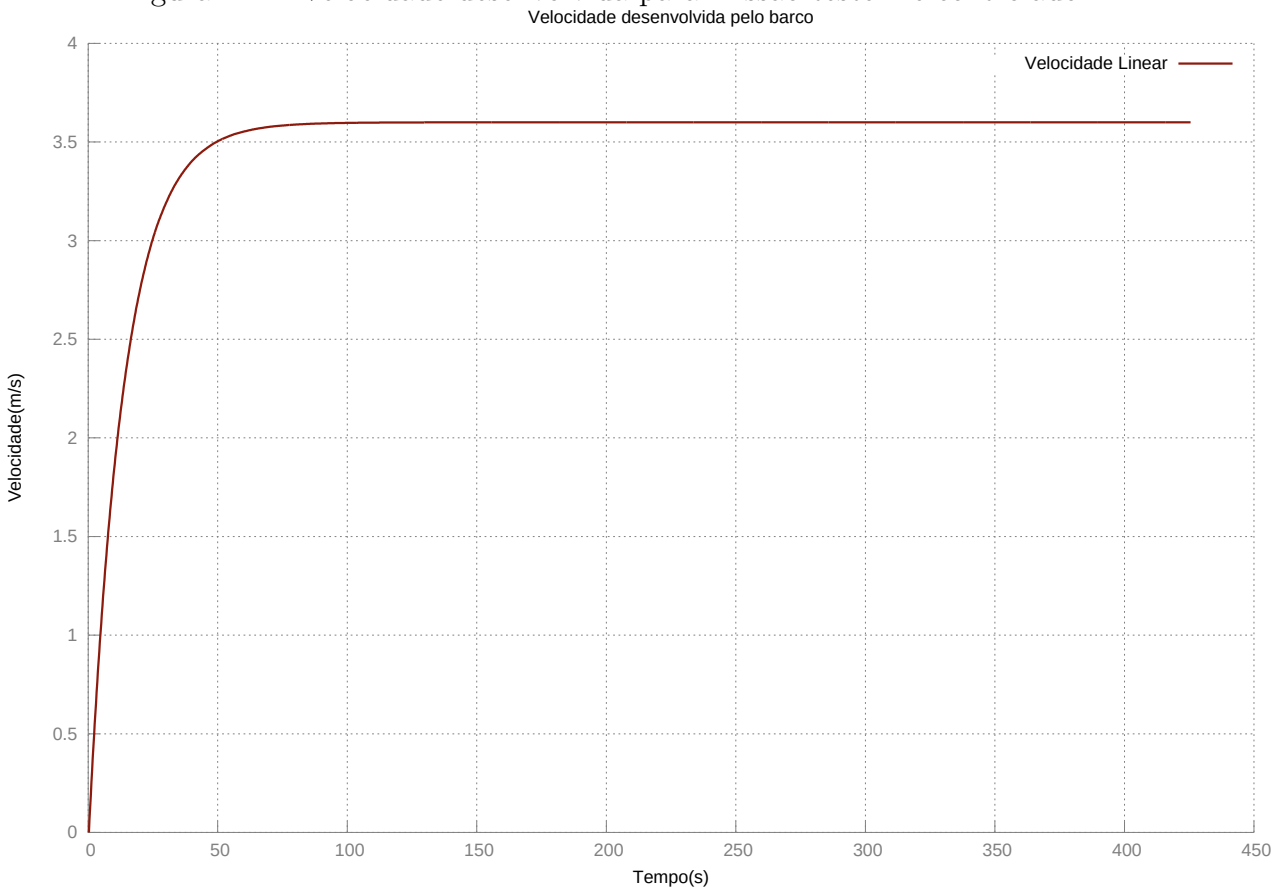


Figura 2.21: Velocidade desenvolvida para missão teste 2 e controlador PD.



2.2.0.2 Discussão

A análise da velocidade desenvolvida pela embarcação dada pela Figura 2.17, mostra que nas transições de trajeto a velocidade diminui o suficiente para que a curva seja feita da melhor forma, conseguinte a velocidade média da missão é mantida a maior possível. Nota-se que a velocidade diminui de acordo com o tamanho da correção de angulatura da embarcação para o novo trajeto, basta comparar a queda de velocidade nos instantes $t = 90s$ e $t = 370s$. Como o erro é proporcional a α_{ref} no instante $t = 90s$ tem-se um menor erro que em $t = 370s$, isto implica que o erro será zerado com mais rapidez no trajeto seguinte ao instante $t = 90s$ do que no instante $t = 370s$, observa-se essa ocorrência na figura 2.15.

O sinal de erro não é zerado completamente em nenhum trajeto da missão teste 1, pois os tamanhos dos trajetos são escolhidos para que o controlador trabalhe perto de seu limite. O controlador só consegue zerar o erro completamente para trajetos longos como na missão teste 2, observa-se o sinal de erro na figura 2.19. Nota-se que o sinal de erro é completamente zerado somente no instante de tempo $t = 350s$ aproximadamente. A velocidade máxima é alcançada rapidamente e se mantém constante por todo trajeto antes do erro ser zerado, que ocorre no instante de tempo $t = 100s$.

O sinal de erro não ser zerado completamente implica em um pouco de desvio de trajetória e o trajeto descrito pela embarcação tem perfil pouco ondulatório. Os desvios de trajetória aumentam a distância total percorrida pela embarcação, porém não afetam a execução das

missões.

2.3 *Software* de Controle

2.3.1 Desenvolvimento

A partir do sistema de controle e orientação cria-se uma biblioteca em linguagem C para o desenvolvimento de um *software* de controle. As principais subrotinas do sistema de controle e orientação e dos sensores são modularizadas para facilitar a escrita do programa principal e também facilitar a exportação destas para outros programas.

Esta biblioteca nomeada de *libarco.a* contém as subrotinas utilizadas para a unidade inercial, para os cálculos dos parâmetros utilizados no sistema de controle e orientação mostrados na tabela 2.1 e o controlador desenvolvido.

Para o desenvolvimento do *software* utiliza-se a ferramenta *Makefile* e o compilador do *gcc* (*GNU Compiler Collection*). A ferramenta *Makefile* é utilizada para que a compilação seja feita por um só comando e que não haja compilações desnecessárias. O programa principal é escrito no arquivo *main.c*, as funções da biblioteca são escritas no arquivo *libarco.c*, os protótipos dessas funções no arquivo *libarco.h* e o *Makefile* cria uma biblioteca estática *libarco.a*. Nota-se que uma vez que a biblioteca estática é criada o arquivo *libarco.c* é desnecessário para o desenvolvimento do programa principal, somente se alguma função necessite de alterações.

O arquivo *Makefile* é escrito da seguinte forma:

```
all: prog

#Compilador
CC = gcc

#Biblioteca
MYLIB = libarco.a

prog: main.o $(MYLIN)
 $(CC) -o prog main.o $(MYLIB)

$(MYLIB): $(MYLIB)(libarco.c)
main.o: main.c libarco.h
libarco.o: libarco.c
```

Para se compilar o programa basta executar o comando *make*, então cria-se o executável *prog* e a biblioteca *libarco.a*. Nota-se que se não houver mudanças nos arquivos *main.c* ou *libarco.c*, não são compilados.

2.3.2 Teste

O programa é testado aplicando uma senoide com período $T = 60s$ como α_{atual} e observa-se os sinais de controles gerados pelo programa para diferentes ângulos α_{obj}. Os resultados para $\alpha_{obj} = 0°$ e $alpha_{obj} = 45°$ são mostrados nas Figuras 2.22 e 2.23, respectivamente.

Figura 2.22: Teste do *software* de controle com $alpha_{obj} = 0°$.

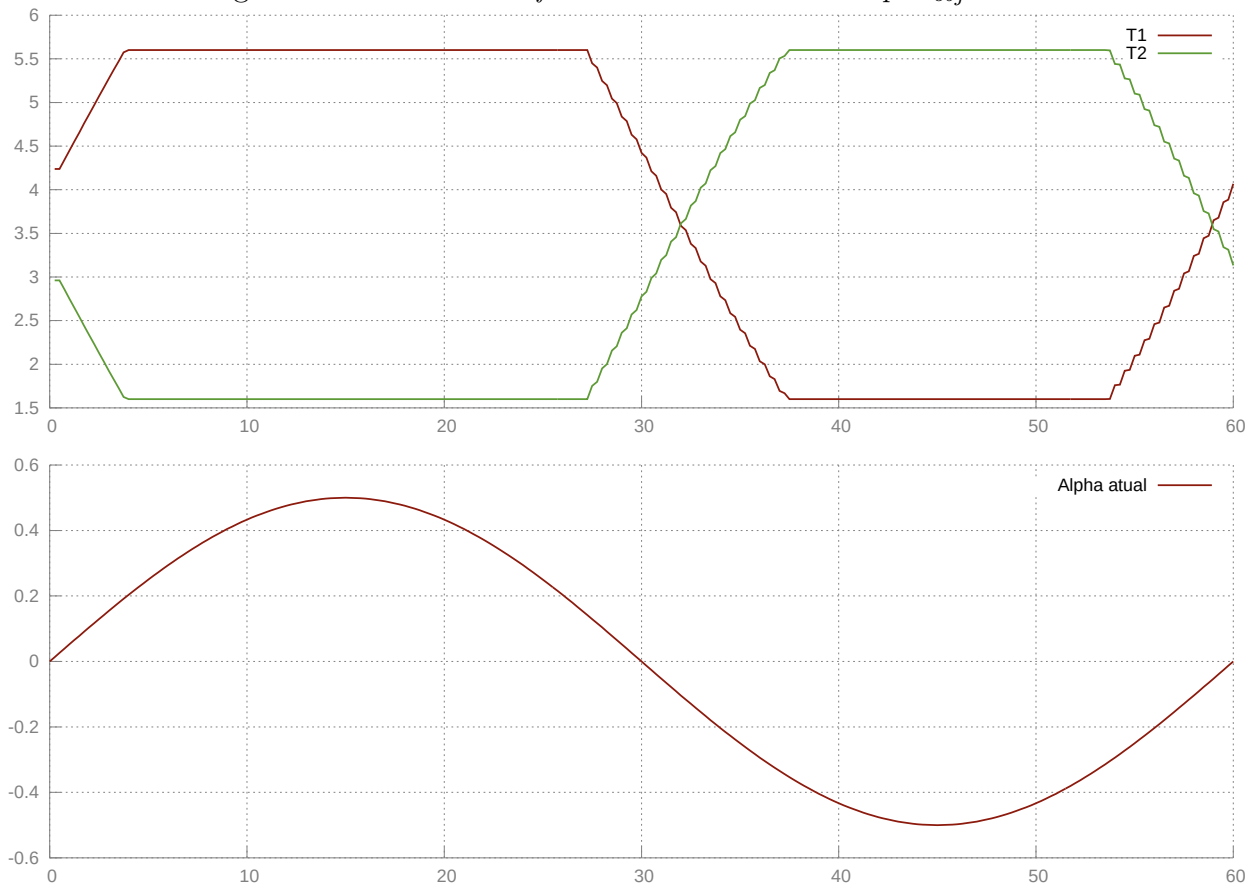
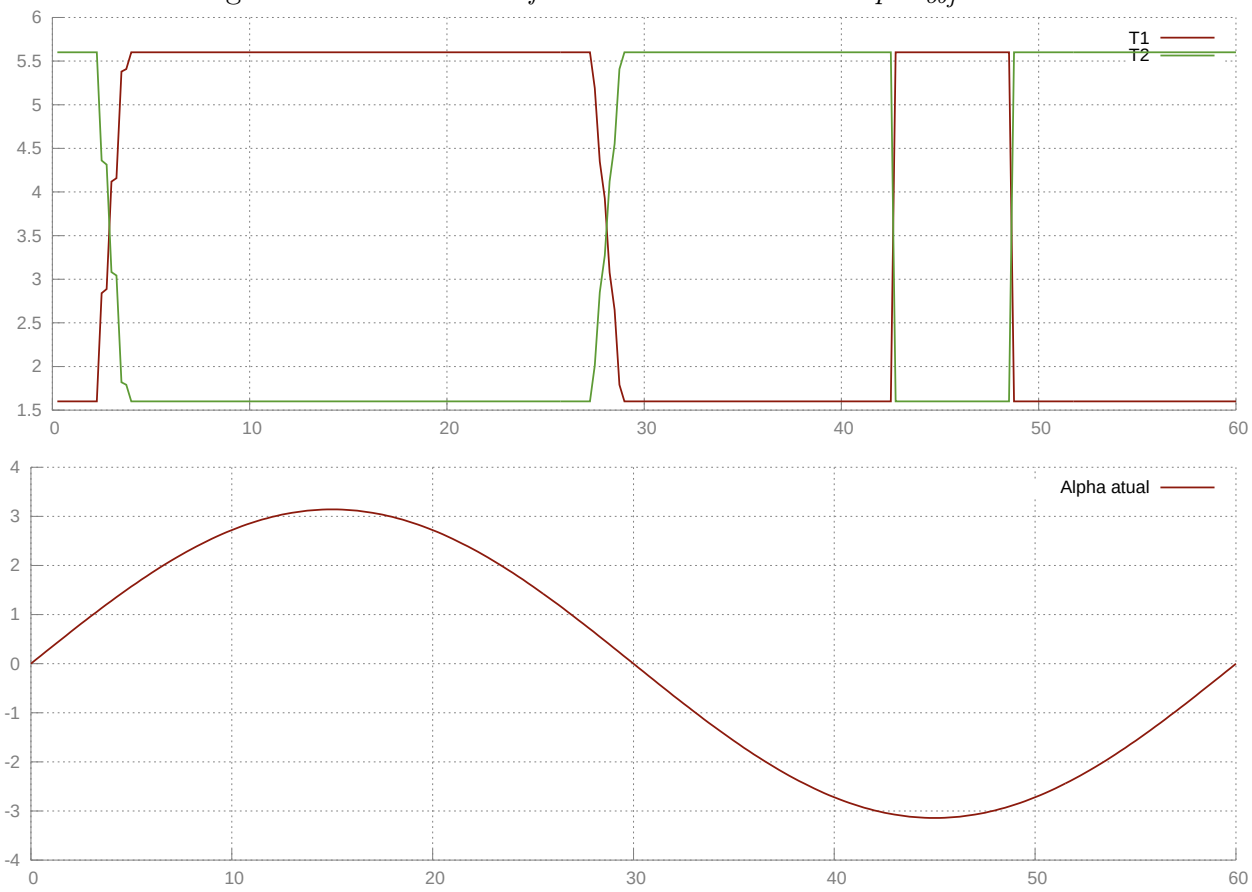


Figura 2.23: Teste do *software* de controle com $alpha_{obj} = 45°$.



Veja na Figura 2.22 que as forças T_1 e T_2 que o controlador calcula são as forças necessárias para que a embarcação atinja um ângulo absoluto igual a α_{obj}. Portanto o *software* de controle funciona da maneira esperada para o sistema de controle e orientação desenvolvido.

CAPÍTULO 3

CONCLUSÃO

Neste trabalho foi desenvolvido um simulador numérico para a dinâmica da embarcação do tipo Trimarã. A partir desse simulador foi desenvolvido o sistema de controle e orientação para a realização de missões.

O simulador numérico desenvolvido foi capaz de produzir um quadro da evolução do estado da embarcação no tempo. O sistema de controle e orientação pôde ser criado a partir desse simulador numérico.

O controlador do tipo proporcional derivativo desenvolvido para o sistema de controle e orientação foi capaz de guiar a embarcação por uma trajetória pré-determinada com bom desempenho. O sistema foi validado pelo simulador numérico e pode ser utilizado para o desenvolvimento do *software* de controle.

O *software* de controle desenvolvido foi capaz de executar o sistema de controle e orientação, permitindo que o sistema seja implementado fisicamente.

O presente trabalho pode servir como referência para trabalhos futuros no desenvolvimento de um sistema de controle e orientação para embarcações do tipo Trimarã e o *software* de controle pode ser reaproveitado para outros projetos facilmente, pela característica modular das principais funções. Pode-se aplicar outras técnicas mais avançadas de controle para melhorar ainda mais o desempenho da embarcação.

O desenvolvimento do projeto foi uma excelente oportunidade de aprendizado que reuniu diversas áreas de estudo da engenharia elétrica e computação. Como controle, processamento digital de sinais, linguagem de programação e cálculo numérico.

BIBLIOGRAFIA

[1] FINN, A.; SHEDING, S. Developments and Challenges for Autonomous Unmanned Vehicles, A Compendium. Springer-Verlag Berlin Heidelber, 2010.

[2] RAGHAVAN, P.; LAD, A.; NEELAKANDAN, S. Embedded Linux System Design and Development. Auerbach Publications, 2006.

[3] FOSSEN. Guidance and control of ocean vehicles. Baffins Lane, England: Wiley, 1994.

[4] PENG, Y. P. Y.; ZHOU, B. Z. B.; HAN, J. H. J. Hardware design and UKF-based tracking control design of Unmanned Trimaran Surface Vehicle. 2007 IEEE International Conference on Robotics and Biomimetics ROBIO, p. 1618-1623.

[5] ALMEIDA, T. E. P. Desenvolvimento de Um Sistema de Sensoriamento de Orientação e Filtragem para Um Veículo Aquático Autônomo de Superfície Aplicando Sensores de Baixo Custo. Dissertação (Mestrado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos. 2014.

[6] WINSBERG, E. Computer Simulation in Science, The Stanford Encyclopedia of Philosophy (Summer 2013 edition). Disponivel em: http://plato.stanford.edu/entries/simulations-science/. Acesso em: 13 abr. 2014.

[7] PENG, Y.; HAN, J. Design and modeling of Unmanned Trimaran Surface Vehicles. 2009 International Conference on Information and Automation, p. 751-756, jun. 2009.

[8] PRESS, W. H.; TEUKOLSKY, S. A.; VETTERLING, W. T.; FLAMMERY, B. P. Numerical Recipes in C. Press Syndicate of the University of Cambridge, 1992.

[9] GSL - GNU Scientific Library, numerical library for C and C++ programmers. Disponível em: http://www.gnu.org/software/gsl/. Acesso em: 5 mar. 2014.

[10] FRANKLIN, G. F.; POWELL, I. D.; EMAMI-NAEMI, A. Feedback Control of Dynamic Systems. Person Education, 6th ed., 2009.

[11] DINIZ, P. S. R.; SILVA, E. A. B.; NETTO, S. L. Processamento Digital de Sinais: Projeto e Análise de Sistemas. São Paulo: Bookman Editora, 2th ed., 2004.

[12] Linux From Scratch, step-by-step instructions for building your own custom Linux system, entirely from source code. Disponível em: http://www.linuxfromscratch.org. Acesso em: 27 abr. 2014.

[13] Buildroot, Making Embedded Linux Easy. Disponível em: http://buildroot.uclibc.org/. Acesso em: 3 mar. 2014.

[14] YAGHMOUR, K.; MASTERS, J.; BEN-YOSSEF, G.; GERUM, P. Building Embedded Linux Systems. O'Reilly Media, 2th ed., 2008.

[15] Open Technology within DoD, Intel Systems. Disponível em: http://web.archive.org/web/20110521185400/http://www.linux.com/archive/feed/61302. Acesso em: 9 jun. 2014.

APÊNDICE A

CONSTRUINDO UM SISTEMA *LINUX*

O tamanho do sistema operacional é algo extremamente crítico na maior parte dos projetos de sistemas embarcados. Os sistemas operacionais distribuídos as vezes não atendem aos requisitos de projetos por excesso de pacotes, o que acarreta o tamanho desnecessário, mesmo aqueles voltados a sistemas embarcados. A solução para este caso é a construção do zero de um sistema operacional.

A melhor solução é a escolha de construção de um sistema *Unix*, pois todos pacotes necessários para a construção são distribuídos gratuitamente sob a licença *GNU General Public License version 2*.

Há vários projetos na internet que ensinam como construir um sistema *Linux* do zero. Um deles é o projeto *Linux From Scratch*, que tem como intuito instruir a construção de um sistema *Linux* customizado passo-a-passo inteiramente do código fonte (http://www.linuxfromscratch.org/).

Geralmente em sistemas embarcados os computadores alvo têm uma arquitetura de CPU diferente do computador desenvolvedor, e um sub-projeto do LFS interessante é o *Cross Linux From Scratch*, que instrui a construção de um compilador cruzado para um sistema *Linux*.

A.1 O *Buildroot*

O processo de criação de um sistema *Linux* do zero é demorado se compilado pacote por pacote. Uma ferramenta muito útil para agilizar o processo é o *Buildroot*, cujo logotipo é: "Making Embedded Linux Easy". Esta é uma ferramenta eficiente e fácil de usar para a construção de um sistema *Linux* embarcado através de um compilador cruzado (http://buildroot.uclibc.org/).

As maiores características do *Buildroot* são:

- Pode-se controlar todo o desenvolvimento do projeto de um sistema embarcado: cross-compiling toolchain, geração do root filesystem, compilação do kernel image e do bootloader.

- Muito fácil de configurar, graças ao menuconfig, gconfig e xconfig. Construir um sistema *Linux* básico com *Buildroot* leva aproximadamente 15-30 minutos.
- Suporta diversos tipos filesystem para o root filesystem image: JFFS2, UBIFS, tarballs, romfs, cramfs, squashfs e mais.

A.2 Exemplo de construção de um sistema *Linux* mínimo

Neste exemplo será criado passo-a-passo um sistema *Linux* mínimo a fim de mostrar a facilidade do uso da ferramenta *Buildroot*. Todos os passos são efetudados por um terminal de comando em um sistema *Linux.*

Primeiramente deve-se adquirir os pacotes da ferramenta, para isto basta acessar o *git* do projeto com o seguinte comando:

```
git clone git://git.buildroot.net/buildroot
```

Neste exemplo utiliza-se a versão do *Buildroot* 2013.02. Após o *download* dos pacotes vá até a pasta do *Buildroot*:

```
cd /home/$nome do usuário$/buildroot
```

Dentro da pasta do *Buildroot* há os arquivos do *makefile* necessários. Para melhor visualização das configurações recomenda-se a utilização da configuração *menuconfig*. O pacote *ncurses-devel* é necessário para esta opção de configuração, para usuários *Linux* com *Aptitude* a instalação é feita da seguinte forma:

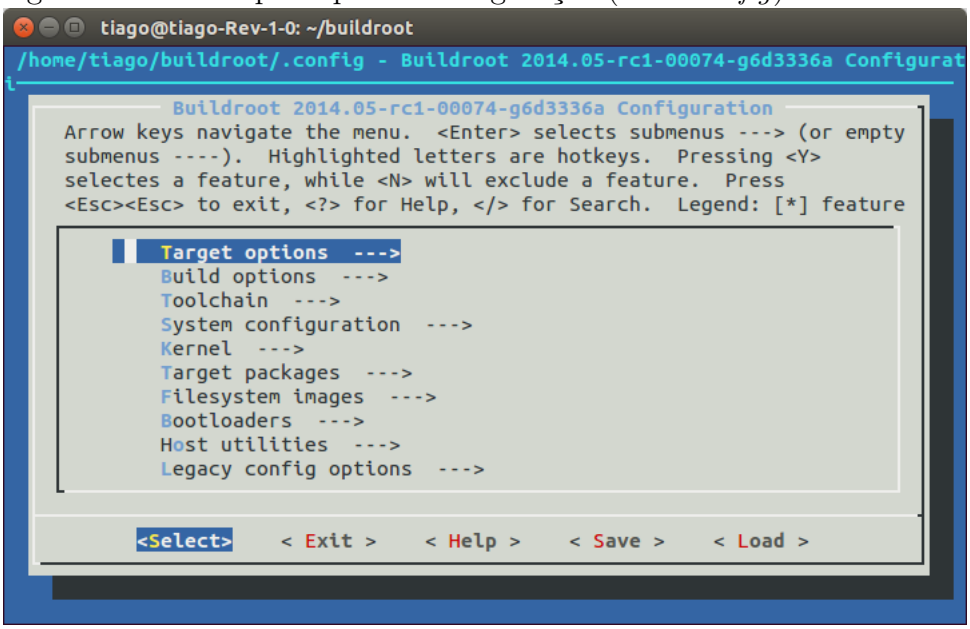
```
sudo apt-get install ncurses-dev
```

Agora todas ferramentas necessárias para a utilização do *Buildroot* estão instaladas. Para começar sua configuração entre com o comando (tenha certeza que esteja na pasta " /buildroot"):

```
 make menuconfig
```

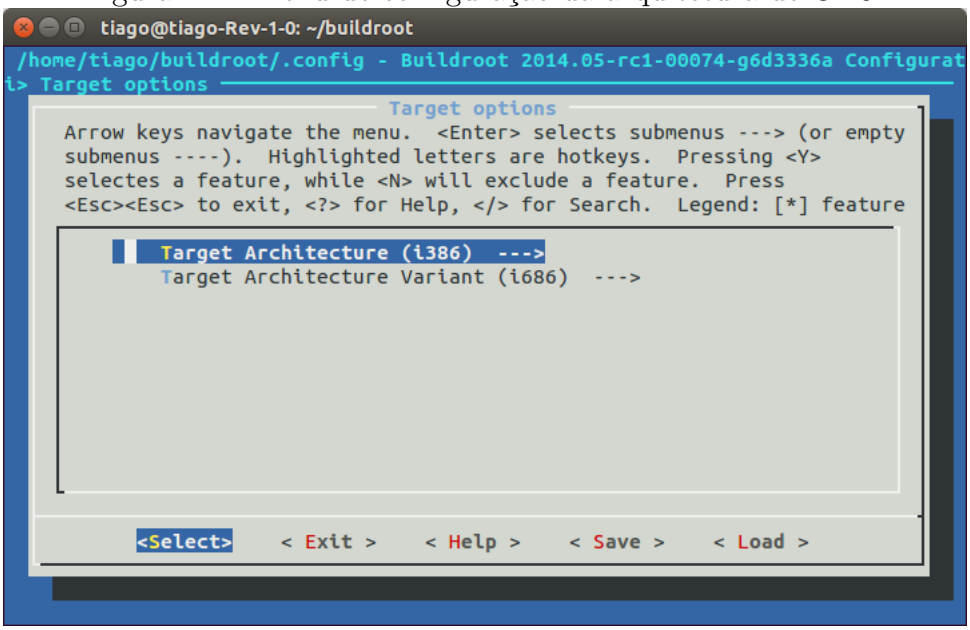
Note que um menu de configurações abriu, como visto na Figura A.1. A navegação por este menu é feita exclusivamente pelo teclado, as instruções estão no cabeçalho do menu.

Figura A.1: Menu principal de configuração (*menuconfig*) do *Buildroot*.



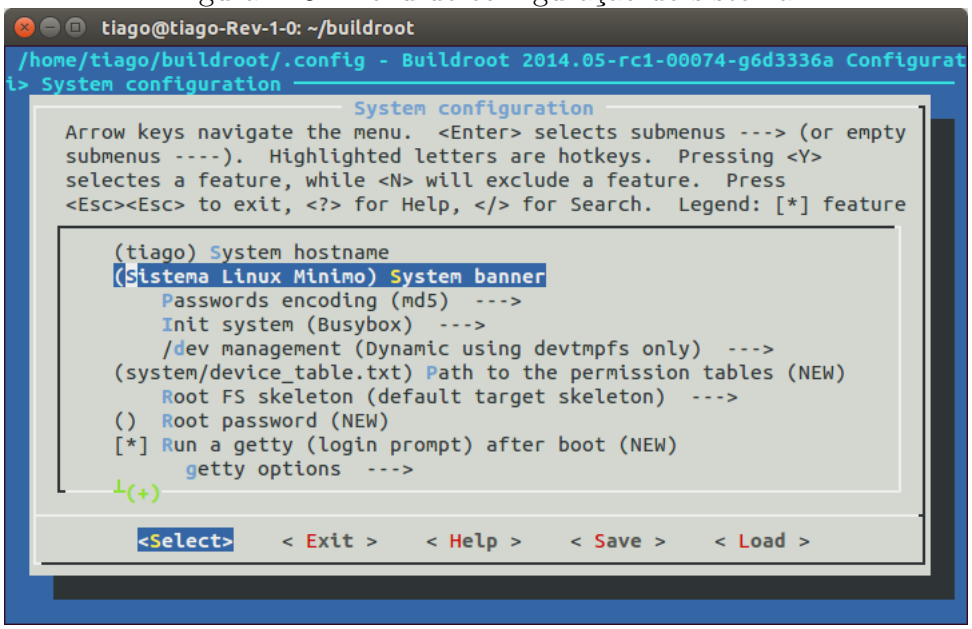
Entre na opção "Target option", como na Figura A.2. Nela escolhe-se para qual arquitetura de CPU será destinado o sistema. Neste exemplo escolheu-se a opções "i386"com *variant* "i686".

Figura A.2: Menu de configuração da arquitetura de CPU.



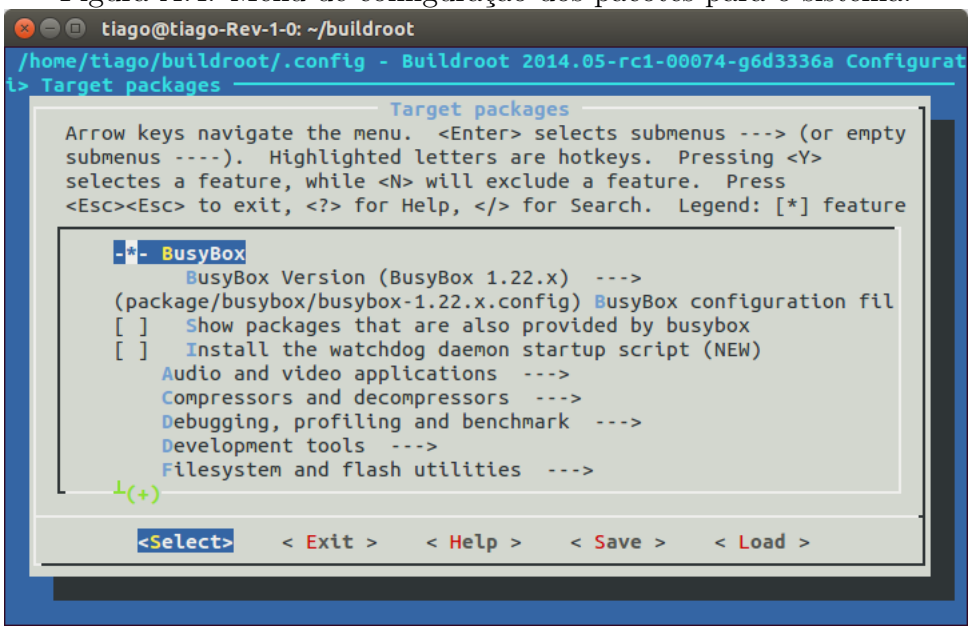
Volte para o menu principal e entre na opção "System configuration", como na Figura A.3. Neste menu pode-se alterar o sistema *Init*, nome do *host*, algumas opções para o *filesystem*, etc. Para um sistema mínimo é aconselhável a utilização do *Busybox* para o sistema *Init*, como esta opção é padrão do *Buildroot* altera-se somente o nome do *host* e do *banner* neste menu.

Figura A.3: Menu de configuração do sistema.



Volte para o menu principal e entre na opção "Target packages", como na Figura A.4. Neste menu escolhe-se os pacotes necessários para seu sistema, como editores de texto, servidores *web*, aplicativos de áudio e vídeo, .etc. Como padrão nenhum pacote é selecionado. Neste exemplo escolheu-se o pacote "uemacs", um micro editor de texto, e o "thttpd", um servidor *http ultra-small*.

Figura A.4: Menu de configuração dos pacotes para o sistema.

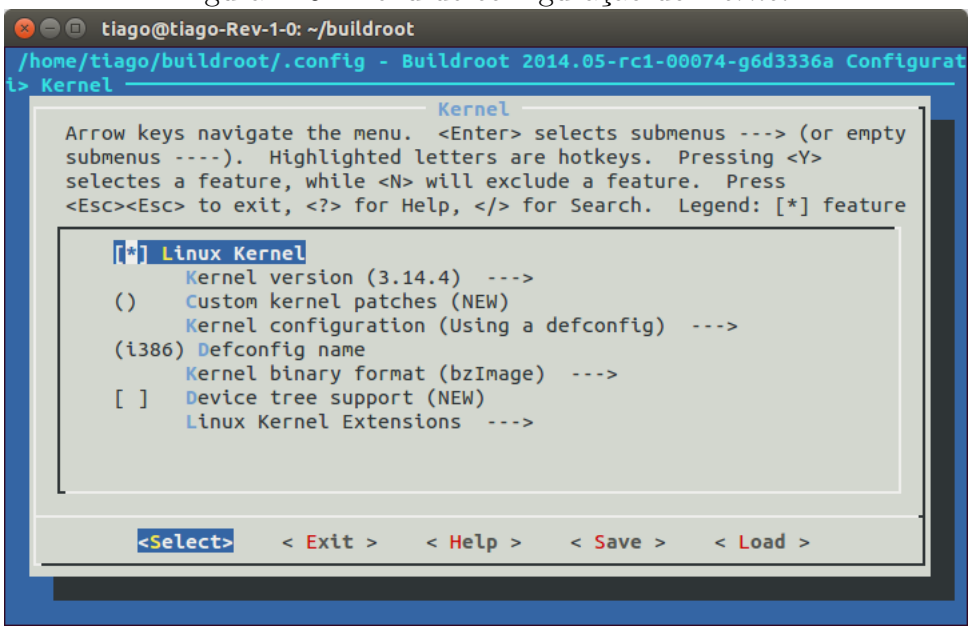


Volte para o menu principal e entre na opção "Kernel", como na Figura A.5. Neste menu escolhe-se as configurações do *Kernel*. A versão padrão do *Kernel*, nesta versão do *Buildroot*, é a 3.14.4. Porém pode-se acessar o site "https://www.kernel.org/", adquirir a *URL* da versão desejada e aponta-la na opção de versão "custom tarball".

Neste exemplo utilizou-se a versão de *Kernel* padrão. Para arquiteturas *x86* deve-se especificar a opção "Defconfig name", neste caso é "i386". Caso haja necessidade de configurações extras especificas do *kernel*, execute o seguinte comando após as configurações do *Buildroot*:

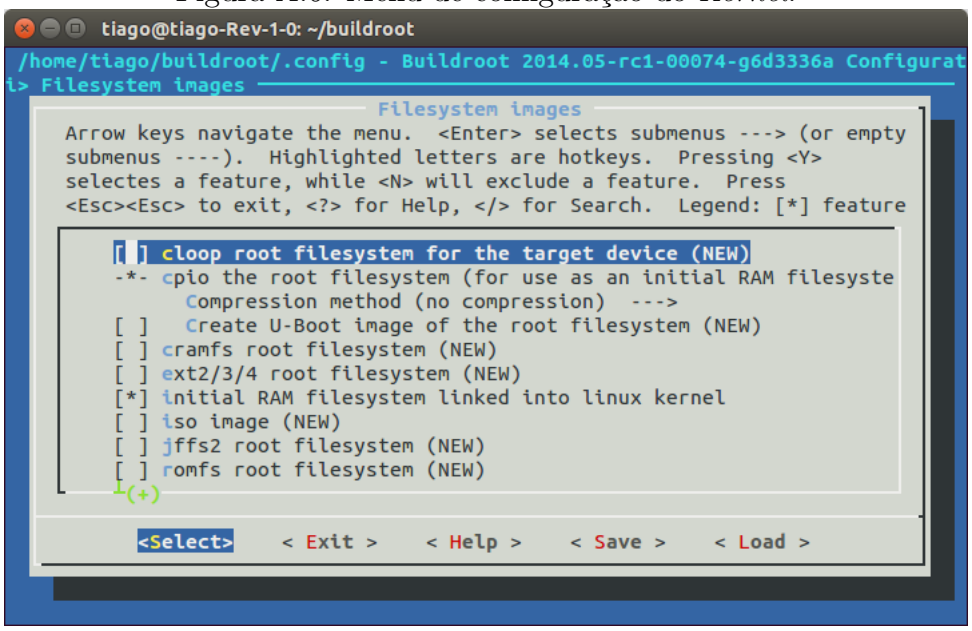
```
make linux-menuconfig
```

Figura A.5: Menu de configuração do *Kernel*.



Volte para o menu principal e entre na opção "Filesystem images", como na Figura A.6. Neste menu escolhe-se as configurações do *root filesystem* do sistema, comumente em sistemas *Linux* utiliza-se o *filesystem* ext2 ou ext4. Neste exemplo utiliza-se a opção "initial RAM filesystem linked into linux kernel", deste modo os arquivos de sistema estarão no arquivo do *kernel* e serão carregado na memória *RAM* na inicialização.

Figura A.6: Menu de configuração do *Kernel*.



Após as configurações selecione a opção *save* e depois saia do *menuconfig*. Para compilar o sistema basta digitar o seguinte comando:

```
make
```

A compilação pode levar alguns minutos. Neste exemplo a compilação durou mais de 20 minutos. Para testar o sistema utilizou-se o emulador *QEMU*, que pode ser obtido pelo comando:

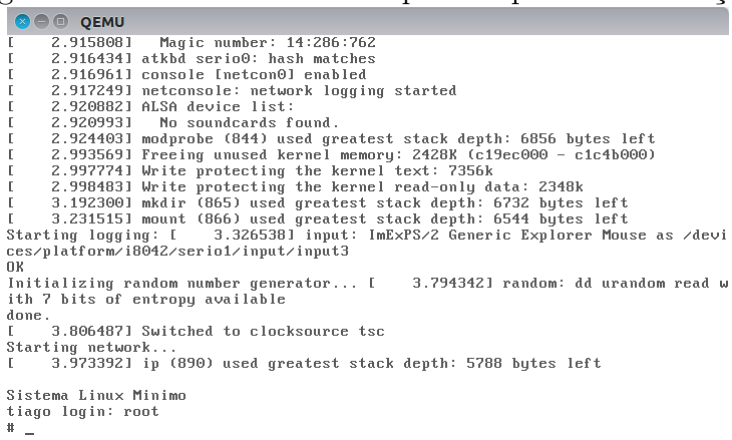
```
sudo apt-get install qemu
```

A imagem do *Kernel* compilada encontra-se no diretório /buildroot/output/images com o nome *bzImage*, o *bz* significa o tipo de compressão utilizada, nesse caso utilizou-se o algorítimo bzip2. Para emular o sistema compilado execute o seguinte comando:

```
qemu -kernel output/images/bzImage
```

Após o inicio do sistema faça o *login* com nome de usuário "root". O sistema está pronto para ser utilizado, como mostra a Figura A.7.

Figura A.7: Sistema *Linux* compilado após a inicialização.



Devido ao grande nível de customização permitido pelo *Buildroot* o sistema compilado tem apenas $15,2MB$.

APÊNDICE B

MÉTODO NUMÉRICO

Método de Runge-Kutta

O método clássico em simulação numérica é o método de Runge-Kutta. Este resolve problemas de valores iniciais propagando a solução sobre um intervalo combinando diversos passos do método de Euler, e com as informações adquiridas equipara à uma série de Taylor.

Seja um problema inicial dado pelas equações B.1 e B.2. Pelo método de Euler a solução seria dada somente por $y_n = hf(x_n, y_n)$, avançando a solução de x_n para $x_{n+1} \equiv x_n + h$.

$$\begin{cases} \dfrac{dy(x)}{dx} = f(x, y) & \text{(B.1)} \\ y(0) = \phi & \text{(B.2)} \end{cases}$$

O método de Runge-Kutta utiliza o principio do método de Euler para dar um passo probatório no ponto médio do intervalo. Depois utiliza os valores de x e y obtidos no ponto médio para dar um passo real através de todo o intervalor. Em suma o método de Runge-Kutta é dado pelas equações B.3, B.4 e B.5, onde $O(h^3)$ é o erro do passo. Este é nomeado de método de Runge-Kutta de segunda ordem (convencionalmente chama-se um método de n-ésima ordem se o erro é $O(h^{n+1})$) ou método do ponto médio (PRESS et al., 1992).

$$k_1 = hf(x_n, y_n) \tag{B.3}$$

$$k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \tag{B.4}$$

$$y_{n+1} = y_n + k_2 + O(h^3) \tag{B.5}$$

Em muitos casos o método de segunda ordem não é adequado, pois o erro por passo é significativo. Então utiliza-se o método de Runge-Kutta de ordem maior. Pelo aumento significativo da precisão e pouco aumento do esforço computacional, o método de quarta ordem é um dos mais utilizados, dado pelas equações B.6, B.7, B.8, B.9 e B.10.

$$k_1 = hf(x_n, y_n) \tag{B.6}$$

$$k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \tag{B.7}$$

$$k_3 = hf(x_n + \frac{h}{2}, y_n + + \frac{k_2}{2}) \tag{B.8}$$

$$k_4 = hf(x_n + h, y_n + k_3) \tag{B.9}$$

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5) \tag{B.10}$$