CÁSSIO KARASSAWA ZANONI

JOSÉ MASCARENHAS NEVES NETO

nota final
8.2 (oito e dois)

DESENVOLVIMENTO DE UM SIMULADOR PARA VEÍCULOS SUBMARINOS NÃO TRIPULADOS

Escola Politécnica da
Universidade de São Paulo
Projeto de Conclusão de Curso

São Paulo
2004

CÁSSIO KARASSAWA ZANONI
JOSÉ MASCARENHAS NEVES NETO

DESENVOLVIMENTO DE UM SIMULADOR PARA VEÍCULOS SUBMARINOS NÃO TRIPULADOS

Escola Politécnica da
Universidade de São Paulo
Projeto de Conclusão de Curso

Área de concentração:
Engenharia Mecatrônica

Orientador:
Prof. Dr. Newton Maruyama

São Paulo
2004

FICHA CATALOGRÁFICA

004.021-8

Neves Neto, José Mascarenhas
Desenvolvimento de um simulador para veículos submarinos não tripulados / J.M. Neves Neto, C.K. Zanoni. -- São Paulo, 2004.
94 p.

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

1.Submersíveis não tripulados 2.SIMULINK I.Zanoni, Cássio Karassawa II.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos III.t.

AGRADECIMENTOS

Ao amigo e orientador Prof Dr. Newton Maruyama, por todo o incentivo e suporte durante a realização deste árduo trabalho.

Ao colega engenheiro Eric Conrado de Souza, pela imensurável colaboração durante todas as etapas.

Aos nossos familiares e amigos pela enorme paciência e compreensão, sem as quais não conseguíamos completar esta importante etapa de nossas vidas.

A todos que direta ou indiretamente contribuíram para a realização deste trabalho.

A Deus, pela iluminação necessária para a conclusão deste trabalho.

RESUMO

O propósito deste trabalho foi o desenvolvimento de um sistema de realidade virtual para a simulação de um veículo submarino não tripulado UUV (*Unmanned Underwater Vehicle*) do tipo ROV (*Remotely Operated Vehicle*) composto por seis propulsores . O veículo recebe as informações de controle através de um cabo umbilical que influi significativamente no comportamento dinâmico do UUV e foi levado em consideração no modelo do veículo. Para o desenvolvimento do simulador, utilizou-se o modelo dinâmico detalhado do trabalho desenvolvido por Souza, (2002). Este foi implementado em SimuLink e associado a ferrameta de interface e simulação gráfica presente no MatLab, o *Virtual Reality Toolbox,* e a ferramenta de criação de ambientes gráficos de simulação, denominados *Virtual Worlds,* o V-Realm Builder . Um outro objetivo foi o de otimizar os blocos existentes no SimuLink através da implementação de S-Functions utilizando a linguagem C. No término da integração entre o modelo em SimuLink e a parte gráfica, concluiu-se que uma maior velocidade de simulação era necessária para que a visualização seja mais realista. Uma forma encontrada foi a de obter resultados numéricos rapidamente executando o modelo em Real Time, e o visualizando posteriormente no ambiente gráfico.

ABSTRACT

The purpose of this study is the development of a virtual reality system for the simulation of an UUV (Unmanned Underwater Vehicle) of the type ROV (Remotely Operated Vehicle) containing six propellers. The vehicle gets control information from surface using a cable, which affects significantly on the dynamic behavior of the UUV and was considered on the vehicle's model. For the simulator's development the dynamic model detailed by Souza (2003) was used. It was than implemented using the MatLab Tool Simulink and associated to the interface and graphical simulation tools also included in MaLab called Virtual Reality Toolbox. A software for the development of graphical environments called virtual worlds, the V-Realm Builder was also used. This study had also the purpose of optimize some "blocks" defined in Simulink, through the implementation of S-Functions using the C language. By the end of the integration between the SimuLink model and graphical interface, it was realized that a faster simulation speed was required in order to make the visualization more realistic. This problem could be solved by obtaining fast numerical results executing the model in Real Time, and than visualizing it in the graphical interface.

SUMÁRIO

LISTA DE FIGURAS

1 INTRODUÇÃO

O objetivo principal deste projeto é o desenvolvimento de um sistema de realidade virtual para a simulação do controle de um veículo submarino não tripulado, que a partir deste ponto será referenciado como UUV (do inglês, Unmanned Underwater Vehicle).

Como a própria denominação já indica o UUV é um veículo não tripulado para ser empregado em missões submarinas onde o ambiente é inóspito para os seres humanos devido a elevada pressão. Existem dois sub-grupos de UUVs: o ROV (Remotely Operated Vehicle), que se caracteriza por ser operado remotamente através de um cabo umbilical; e o AUV (Autonomous Underwater Vehicle), que é um veículo totalmente autônomo no que diz respeito a geração de trajetórias.

O veículo tratado neste estudo é um UUV ROV desenvolvido pelo departamento de Engenharia Mecatrônica e de Sistemas Mecânicos da Escola Politécnica da USP, denominado UUV ROV-CTPETRO.

Para possibilitar a execução deste sistema utilizaremos uma função do MatLab que permite a representação numérica tanto do UUV quanto do seu sistema de controle em linguagem C, diminuindo significativamente o tempo de simulação dos objetos em estudo e viabiliza a utilização do sistema de realidade virtual.

Na primeira etapa do projeto, o foco principal foi o estudo dos aspectos dinâmicos e de controle do UUV, que foram bastante detalhados em Souza (2003), e que serviram como base para o desenvolvimento da modelagem numérica do UUV durante todo o trabalho. Outro ponto estudado, foi uma maneira de otimizar o tempo de simulação do modelo através da utilização das ferramentas de desenvolvimento de funções em linguagem C (S-function) presentes no MatLab.

Para visualização do comportamento do modelo numérico, foi desenvolvido um ambiente gráfico utilizando a linguagem VRML, que integra-se de forma satisfatória ao modelo em MatLab.

2 MODELAGEM DO VEÍCULO

Para que seja possível a simulação do controle do ROV, que é o objetivo final deste trabalho, devemos obter um modelo matemático do veículo. Além disso, este modelo pode nos dar importantes informações sobre a controlabilidade, sobre a estabilidade e sobre o comportamento do mesmo nos domínios do tempo e da freqüência.

Como a obtenção deste modelo não é o foco principal deste trabalho, sim seu aprimoramento para efetuarmos as simulações que desejamos, este será exposto sucintamente tendo como base a tese de mestrado de Souza (2003).

Este veículo possui seis graus de liberdade dinamicamente acoplados, já que a dinâmica de veículos submarinos é não linear e multivariável. Por isso a modelagem é feita considerando os seis graus de liberdade.

Para facilitar o entendimento, a apresentação da modelagem será divida em Cinemática e Dinâmica e ao final será apresentada a união das duas representando o modelo completo do veículo.

2.1 Cinemática

Para caracterizar o movimento do veículo submarino, serão utilizados dois sistemas de coordenadas distintos: o sistema de referência móvel $0x_my_mz_m$ (fixo em relação ao baricentro do veículo) e o sistema de referência fixo 0xyz (fixo em relação à Terra). Os dois sistemas estão demonstrados na figura 2.1. Estabelecidos os sistemas, deve se indicar as notações que serão utilizadas para identificar as coordenadas de cada um dos seis graus de liberdade presentes nos sistemas.

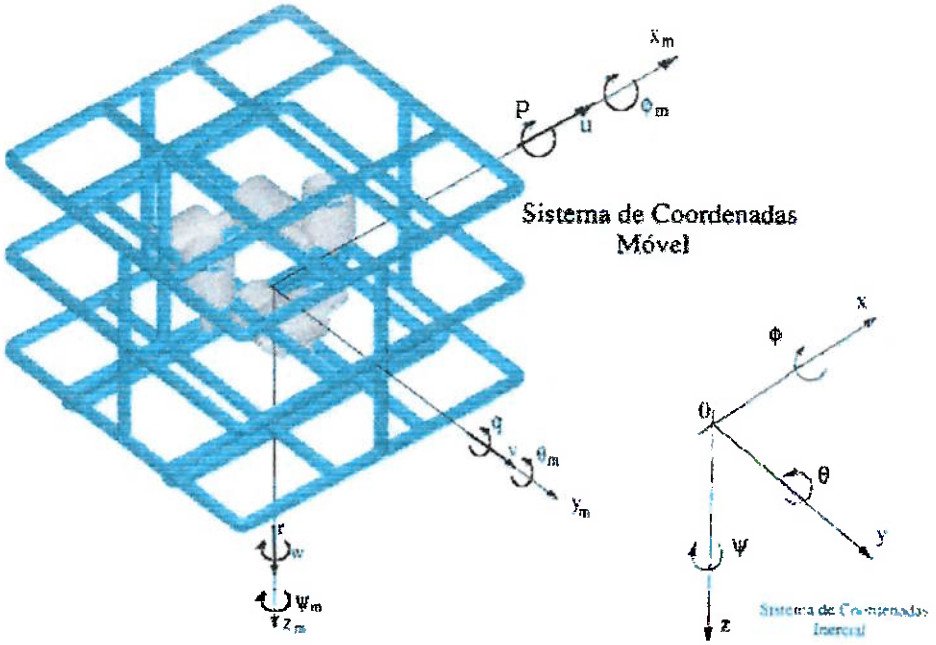

Figura 2.1 – Sistemas de Coordenadas

Sistema inercial:

- vetor posição:

$$\eta_1 = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- vetor orientação:

$$\eta_2 = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$

Obs: Serão utilizados os ângulos de Euler para definir a orientação do veículo durante todo o trabalho.

- vetor posição-orientação

$$\eta = \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} = [x, y, z, \phi, \theta, \psi]^T$$

- vetor de translação

$$\eta_1 = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- vetor de rotação

$$\eta_2 = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$

- vetor velocidade

$$\dot{\eta} = \begin{bmatrix} \dot{\eta}_1 \\ \dot{\eta}_2 \end{bmatrix} = [\dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}]^T$$

Sistema móvel:

- vetor posição

$$\nu_1 = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

- vetor orientação

$$\nu_2 = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

- vetor posição-orientação

$$\nu = \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix} = [u, v, w, p, q, r]^T$$

- vetor de translação

$$\nu_1 = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

- vetor de rotação

$$\nu_2 = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

- vetor velocidade

$$\nu = \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix} = [u, v, w, p, q, r]^T$$

A seguir, segue a tabela 2.1, onde está especificada a notação que será utilizada durante todo este trabalho.

Grau de liberdade	Nome	Ref. Inercial		Ref. Móvel	
		Pos./atit.	Veloc.	Pos./atit.	Veloc.
1	surge	X	$\dot{x}$	x_m	U
2	sway	Y	$\dot{y}$	y_m	V
3	heave	Z	$\dot{z}$	z_m	W
4	roll	Φ	$\dot{\varphi}$	φ_m	P
5	pitch	Θ	$\dot{\theta}$	θ_m	Q
6	yaw	Ψ	$\dot{\psi}$	ψ_m	R

Tabela 2.1 – Definições das variáveis

Neste momento é importante determinar a relação existente entre o os dois sistemas de coordenadas utilizados. Para o caso das velocidades lineares, esta relação ocorre através do operador de transformação não linear J_1, que é definido como:

$$\eta_1 = J_1(\eta_2)\nu_1$$

Onde

$$J_1(\eta_2) = \begin{bmatrix} \cos(\theta)\cos(\psi) & \text{sen}(\phi)\text{sen}(\theta)\cos(\psi) - \cos(\phi)\text{sen}(\psi) & \cos(\phi)\text{sen}(\theta)\cos(\psi) + \text{sen}(\phi)\text{sen}(\psi) \\ \cos(\theta)\text{sen}(\psi) & \text{sen}(\phi)\text{sen}(\theta)\text{sen}(\psi) + \cos(\phi)\cos(\psi) & \cos(\phi)\text{sen}(\theta)\text{sen}(\psi) - \text{sen}(\phi)\cos(\psi) \\ -\text{sen}(\theta) & \text{sen}(\phi)\cos(\theta) & \cos(\phi)\cos(\theta) \end{bmatrix}$$

Analogamente, para a relação entre as velocidades de rotação, será utilizado o operador não linear J_2, que é definido por:

$$\eta_2 = J_2(\eta_2)\nu_2$$

Onde

$$J_2(\eta_2) = \begin{bmatrix} 1 & \text{sen}(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\text{sen}(\phi) \\ 0 & \text{sen}(\phi)\sec(\theta) & \cos(\phi)\sec(\theta) \end{bmatrix}$$

Combinando os dois operadores, pode-se calcular a matriz J, que será utilizada para a transformação de coordenadas entre os sistemas. Esta matriz é construída da seguinte maneira:

$$J(\eta(= \begin{bmatrix} J_1(\eta_2) & 0_{3x3} \\ 0_{3x3} & J_2(\eta_2) \end{bmatrix} \in \Re^{6x6}$$

Definidas todas as coordenadas do veículo e suas relações, deve-se apresentar agora como serão representados os esforços externos no corpo do submarino. Eles serão representados pelo vetor τ, conforme definido abaixo:

$$\tau = [\tau_1^T, \tau_2^T]^T$$

Onde τ_1 corresponde as forças externas e τ_2 aos momentos externos que agem no corpo do veículo.

2.2 Dinâmica

Para obter o modelo dinâmico deste submarino utilizou-se a segunda lei de Newton para caracterizar a relação entre causa e efeito presente no sistema em estudo. Porém, devido as suas dimensões não é possível tratar o veículo como um ponto material, portanto utilizou-se as seguintes expressões de dinâmica de corpos rígidos em relação ao sistema de referência móvel descritas por Indiveri et al.(1998):

$$m[v_1 + v_2 \times v_1 + v_2 \times r_G + v_2 \times (v_2 \times r_G)] = \tau_1$$

Para translação e

$$\frac{d}{dt}(I_0 v_2) + mr_G \times (v_1 + v_2 \times v_1)] = \tau_2$$

Para rotação.

Deve-se observar que nas expressões acima, o operador x representa o produto vetorial, o vetor r_G corresponde à distância do centro de massa do veículo com relação ao sistema de coordenadas móvel e a matriz I_0 contém os momentos e

produtos de inércia do veículo com relação ao centro do sistema de coordenadas móvel.

Porém, para que seja possível fazer o modelamento numérico do veículo, deve-se escrever as equações acima na forma matricial. Fazendo os agrupamentos e manipulações necessárias, chegou-se as seguintes expressões matriciais:

$$M_{CR}\,\nu + C_{CR}(\nu)\nu = \tau$$

Onde a força de inércia é:

$$M_{CR}\,\nu = \begin{bmatrix} m\,\nu_1 + m\,\nu_2 \times r_G \\ I_0\,\nu_2 + m r_G\,\nu_2 \times \nu_1 \end{bmatrix}$$

E a força centrípeta e de Coriolis é dada por:

$$C_{CR}\nu = \begin{bmatrix} m\nu_2 \times \nu_1 + m\nu_2 \times (\nu_2 \times r_G) \\ \nu_2 \times (I_0\nu_2) + m r_G \times (\nu_2 \times \nu_1) \end{bmatrix}$$

Para a parametrização destas duas matrizes tem-se, que:

$$M_{CR} = M_{CR}^{T} > 0,$$

$$\dot{M}_{CR} = 0$$

e

$$C_{CR}(\nu) = C_{CR}^{T}(\nu) \forall \nu \in \Re^6$$

Através destas matrizes pode-se concluir que os esforços centrípeta e de Coriolis são dependentes de termos quadráticos dos componentes da velocidade do submarino. Estes termos apresentam multiplicação entre componentes de graus de liberdade distintos, caracterizando assim a não linearidade da dinâmica do sistema do submarino.

2.3 Esforços Hidrodinâmicos

Segundo Clayton e Bishop (1982) o conjunto de esforços hidrodinâmicos ao qual está submetido um veículo submarino completamente submerso (desprezando-se as ondas) é o seguinte:

- Esforços devido à massa adicionada;
- Arrasto ou dissipação hidrodinâmica (amortecimento hidráulico);
- Esforços de sustentação.

Pode-se deduzir que inicialmente, devido a ausência de velocidade, os esforços devido à massa adicional são mais significativos. Porém, assim que o submarino atinge uma velocidade mais elevada pode-se perceber que existe um aumento da importância dos esforços hidrodinâmicos.

2.3.1 Esforços Devido À Massa Adicionada

Estes esforços envolvem a movimentação forçada de partículas do fluido que envolvem o corpo do veículo durante o movimento do mesmo. Como se tratam de esforços de corpo rígido, estes podem ser decompostos como uma parte inercial e outra centrípeta. Yuh (1990) apresentou os esforços desta maneira:

$$\tau_A = -\frac{d}{dt} M_A \nu$$

Onde M_A é a matriz de inércia devido à massa adicionada. Escrevendo a equação dos esforços devido à massa adicionada τ_A, em função das matrizes de inércia M_A e de Coriolis, e parametrizando-a chegou-se a:

$$M_A \dot{\nu} + C_A(\nu)\nu = \tau_A$$

Os elementos da matriz M_A representam derivadas hidrodinâmicas com relação a aceleração do veículo e são escritos da seguinte maneira:

$$M_{A_{ij}} \stackrel{\Delta}{=} \frac{\delta\tau(i)}{\delta\nu(j)}(i, j = 1..n_{gl})$$

Onde $\tau(i)$ é a somatória dos esforços externos do i-ésimo grau de liberdade e $\upsilon(j)$ a aceleração do veículo no j-ésimo grau de liberdade.

A matriz C_A, assim como a matriz C_{CR} apresentada no item anterior, também possui elementos dependentes das componentes do vetor velocidade do veículo, causando assim termos quadráticos não lineares nas forças e momentos de Coriolis de massa adicionada.

2.3.2 Arrasto Hidrodinâmico

Pode-se relacionar o esforço hidrodinâmico devido ao arrasto a dois fenômenos: a fricção de superfície e o arrasto devido à pressão. O primeiro deve-se ao regime turbulento causado pelas tensões entre o fluido e a rugosidade da superfície do corpo e é dominante para baixas velocidades. O segundo deve-se à diferença de tensões normais à superfície do corpo e aparece mais intensamente com a maior velocidade do submarino.

A equação que quantifica o esforço de arrasto F_D é:

$$F_D = -0.5\rho \cdot C_D S|\nu|\nu$$

Porém, normalmente, o arrasto dinâmico possui dependência quadrática com a velocidade do veículo relativa ao fluido υ_R. Esta velocidade υ_R pode ser definida como :

$$\nu_R = \nu - \nu_C$$

Onde υ_C representa a velocidade da correnteza.

Considerando esta variação nos coeficientes de arrasto pode-se representar o esforço de arrasto da seguinte forma:

$$F_D(v, v_C) \propto C_D(v, v_C) f(v, v_C)$$

Onde F_D corresponde aos esforços hidrodinâmicos de arrasto, f é um vetor de funções não lineares das velocidades υ e υ_C e a matriz C_D contém os coeficientes de arrasto em função de υ e υ_C. Passando esta representação para a forma matricial chegou-se a:

$$F_D(v) = -0.5\rho|v_r|^2 \nabla_R^{2/3} \begin{bmatrix} C_{Fx}(\alpha,\cdot\beta) \\ C_{Fy}(\beta,\cdot\gamma) \\ C_{Fz}(\alpha,\cdot\gamma) \\ C_{Nx}(\gamma\cdot)\nabla_R^{1/3} \\ C_{Ny}(\alpha\cdot)\nabla_R^{1/3} \\ C_{Nz}(\beta\cdot)\nabla_R^{1/3} \end{bmatrix} - 0.5\rho\nabla_R^{5/3} \begin{bmatrix} 0 \\ 0 \\ 0 \\ C_{Np}p|p| \\ C_{Nq}q|q| \\ C_{Nr}r|r| \end{bmatrix}$$

onde

α = arctan (w_r/u_r),

β = arctan (v_r/u_r),

γ = arctan (w_r/v_r),

ρ = densidade do fluido,

υ_r = velocidade relativa do veículo

∇_R = volume total do veículo.

Que será a forma utilizada nos cálculos deste trabalho.

2.4 Esforços Ambientais

O principal efeito ambiental ao qual está sujeito o veículo submarino é o efeito das correntezas. Geralmente, este efeito é caracterizado por sua velocidade υ_r, que normalmente é definida para o referencial inercial. Pode-se considerar que a componente de rotação desta velocidade é nula e desta maneira podemos representar a correnteza no sistema móvel como:

$$\nu_{1_c} = J_1^{-1}(\eta_2)\eta_{1_c}$$

Já a velocidade relativa do fluido, como dito anteriormente, pode ser expressa como:

$$\nu_R = \nu - \nu_C$$

Devido à natureza das perturbações geradas pela correnteza serem de baixa freqüência, pode-se considerar sua variação temporal nula.

Para a simulação será utilizada a forma matricial da expressão do esforço da correnteza, que foi parametrizada por Lewis, Liscombr e Thomasson (1984) e por Kalske e Happonen (1991):

$$M_c \nu_c + C_c(\nu_c)\nu = \tau_c$$

Onde, para υ_{2c}=0 e υ_{2c}=0,

$$M_c \nu_c = \begin{bmatrix} \rho\nabla \nu_{1_c} \\ \rho\nabla r_B \times \nu_{1_c} \end{bmatrix} + M_A \nu_c$$

e

$$C_c(\nu_c)\nu = \begin{bmatrix} \rho\nabla \nu_2 \times \nu_{1_c} \\ \rho\nabla r_B \times (\nu_2 \times \nu_{1_c}) \end{bmatrix} + C_A(\nu_c)\nu$$

2.5 Esforços Restaurativos

Os dois principais esforços restaurativos presentes em um veículo submarino são o empuxo e o peso. Estas forças podem ser descritas no sistema referencial do veículo como:

$$\tau_G(\eta) = J_1^{-1}(\eta_2)\begin{bmatrix}0\\0\\W\end{bmatrix}, \tau_B(\eta) = -J_1^{-1}(\eta_2)\begin{bmatrix}0\\0\\B\end{bmatrix}$$

Onde

$W = mg$

$B = \rho g \nabla$

Com:

∇ = volume de fluido deslocado pelo veículo

g = aceleração da gravidade

E com estas forças pode-se chegar a expressão geral para forças e momento restaurativos que é:

$$G(\eta) = -\begin{bmatrix}\tau_G(\eta) + \tau_B(\eta)\\ r_G \times \tau_G(\eta) + r_B \times \tau_B(\eta)\end{bmatrix}$$

Sendo que o peso W é aplicado no centro de massa C_G de distância r_G em relação à origem do sistema de referência do veículo. Já o empuxo atua sobre C_B que possui uma distância r_B em relação à origem do sistema de coordenadas móvel, conforme a figura 2.2.

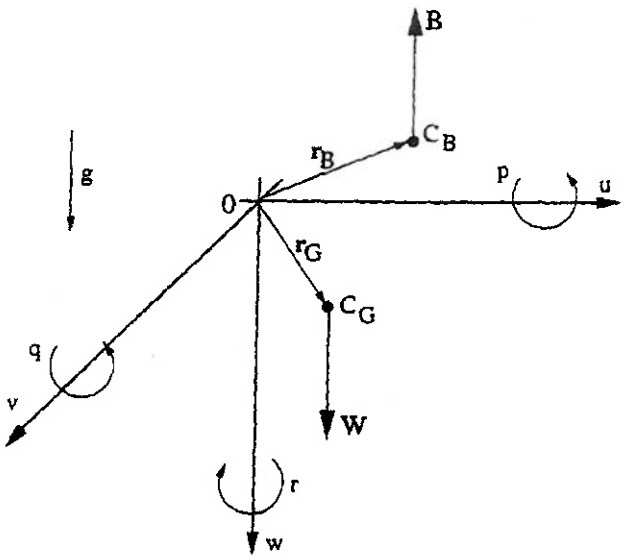

Figura 2.2 - Posição de C_B e C_G

O centro de massa e o local de aplicação do empuxo são variáveis que podem ser projetadas para determinar diferentes comportamentos do submarino quanto a estabilidade e a flutuabilidade, porém isto não será discutido neste trabalho.

2.6 Expressão Geral

Considerando todos os elementos apresentados chega-se a uma formulação geral para a dinâmica do veículo submarino, que é:

$$M\dot{\nu} + C(\nu)\nu + F_D(\nu) + G(\eta) = \tau$$

Onde a matriz de inércia M é o resultado da soma da matriz de inércia de corpo rígido M_{CR} e da matriz de inércia de massa adicionada M_A. O mesmo acontecendo para a matriz centrípeta C. Já a matriz G, representa os esforços restaurativos. Enquanto a parcela F_D, representa a dissipação hidrodinâmica e pode ser parametrizada como uma estrutura desacoplada:

$$F_D(\nu) \approx D(\nu)\nu$$

A equação apresentada no início do capítulo é relativa ao referencial do veículo, portanto para utilizá-la em relação ao sistema inercial deve-se escrevê-la como:

$$M_\eta(\eta)\dot{\eta} + C_\eta(\nu,\eta)\eta + D_\eta(\nu,\eta)\eta + G_\eta(\eta)\eta = \tau_\eta$$

Onde

$M_\eta(\eta) = J^{-T}(\eta)MJ^{-1}(\eta)$,

$C_\eta(\nu,\eta) = J^{-T}(\eta)[C(\nu) - MJ^{-1}(\eta)\dot{J}(\eta)]J^{-1}(\eta)$,

$D_\eta(\nu,\eta) = J^{-T}(\eta)D(\nu)J^{-1}(\eta)$,

$G_\eta(\eta) = J^{-T}(\eta)G(\eta)$,

$\tau_\eta = J^{-T}(\eta)\tau$

O vetor τ incorpora os esforços externos e tem a seguinte composição:

$\tau = \tau_c + \tau_{cabo} + \tau_{prop}$

Onde

τ_c = esforço devido à correnteza marítima

τ_{cabo} = esforço devido ao cabo umbilical

τ_{prop} = esforço devido ao sistema propulsor

Mas, através de uma manipulação, pode-se incorporar a componente τ_c (relativa à correnteza marítima) ao primeiro termo da equação inicial do capítulo. Assim, como equação geral do sistema submarino obteve-se:

$$M\dot{\nu} + C(\nu_r)\nu_r + F_D(\nu_r) + G(\eta) = \tau_{cabo} + \tau_{prop}$$

Como mencionado anteriormente, o comportamento do sistema estudado é não linear, portanto a equação acima representa apenas uma aproximação próxima para o estudo do controle do sistema. Para a verdadeira validação devem ser feitos testes com modelos físicos comprovando a validade deste modelo simplificado para determinadas situações. Já é possível antever que este modelo do submarino deve ser

válido apenas para a região de baixas freqüências de velocidade, devendo ser alterado se este não for o caso desejado.

No caso deste trabalho, que diz respeito à simulação de um sistema de controle para o veículo submarino, este modelo pode ser utilizado, pois propicia uma boa aproximação do modelo físico.

2.7 Cabo Umbilical

O cabo umbilical tem como principal função a comunicação do ROV com a embarcação que o controla, ou seja, é ele quem transmite energia, sinais de comando e sinais de monitoramento da plataforma de controle para o veículo submarino, como ilustrado na figura 2.3.

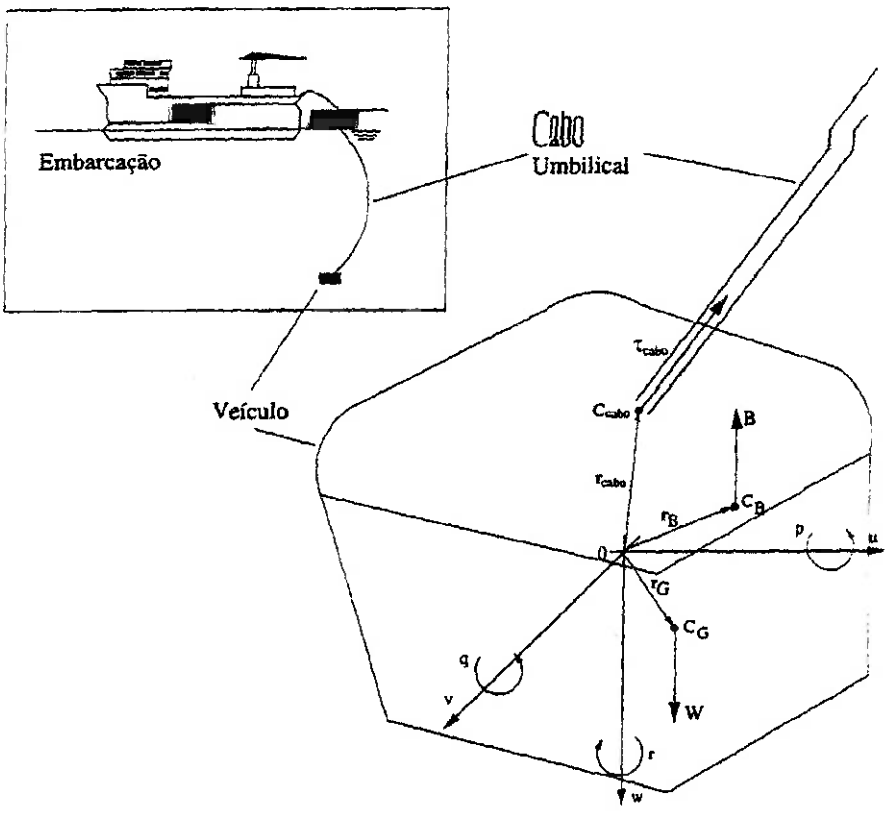

Figura 2.3 - Modelo de comunicação entra a embarcação e o ROV

Neste trabalho o cabo umbilical será modelado segundo o modelo de massas concentradas (Nomoto e Hattori, 1986); (Yokobiki, Koterayama, Yamaguchi e Nakamura, 2000); (Driscoll, Lueck e Nahon, 200), que aproxima o cabo por um modelo discreto composto de pequenos elementos cilíndricos extensíveis, que considera as massas concentradas nos nós entre os elementos (figura 2.4). Neste modelo os esforços são transmitidos dos elementos superiores para os inferiores e depois para o ROV através da ponta do cabo conectado ao veículo. Desta forma obtêm-se o esforço do cabo no submarino τ_{cabo}.

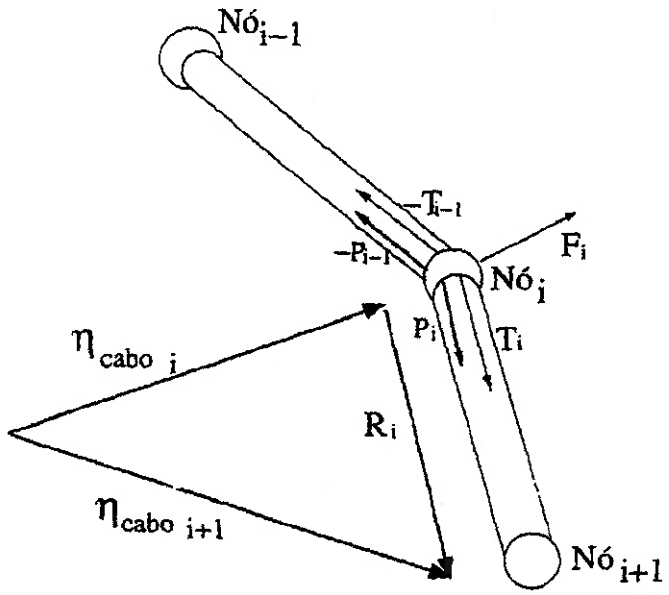

Figura 2.4 - Elementos do cabo umbilical

2.7.1 Modelagem do cabo umbilical

Pode-se dividir as forças que agem sobre o cabo em internas e externas. Nas internas se destacam a tensão T e a pressão P axiais (neste caso são desprezados propositalmente os efeitos torcionais e flexionais apesar de saber-se que para alguns casos indesejados estes efeitos podem vir a ser importantes). Quanto aos esforços externos, serão considerados o arrasto hidrodinâmico F_F e as forças restaurativas F_G para a modelagem do cabo umbilical.

Desta maneira a dinâmica de cada elemento em três dimensões fica determinado pela seguinte equação:

$$[M_i + M_{Ai}]_{cabo}\,\ddot{\eta}_{cabo} = (T_i + P_i)_{cabo} - (T_{i-1} + P_{i-1})_{cabo} + F_i$$

Onde

η_{cabo} = aceleração do i-ésimo nó

M_i = matriz de massa do i-ésimo elemento cilíndrico

M_{Ai} = matriz de massa adicionada do i-ésimo elemento cilíndrico

N_{cabo} = número de nós utilizados

$i = 1..N_{cabo}-1$

2.7.1.1 Forças Internas

Neste trabalho, cada um dos elementos do cabo umbilical será aproximado por um sistema massa-mola-amortecedor, como demonstrado na figura 2.5. Assim sendo T_{cabo}, que é a tensão normal sobre cada elemento pode ser representada da seguinte maneira:

$$T_{cabo_i} = E\frac{A_{cabo_i}}{l_{0_i}} R_i \left[1 - \frac{l_{0_i}}{|R_i|}\right]$$

Onde

E = módulo de Young do cabo.

A_{cabo} = seção transversal do cabo de diâmetro d_i.

l_0 = comprimento natural de cada elemento do cabo

$R_i = (\eta_{caboi+1} - \eta_{cabo})$

Onde η_{caboi} é a posição do respectivo elemento do cabo.

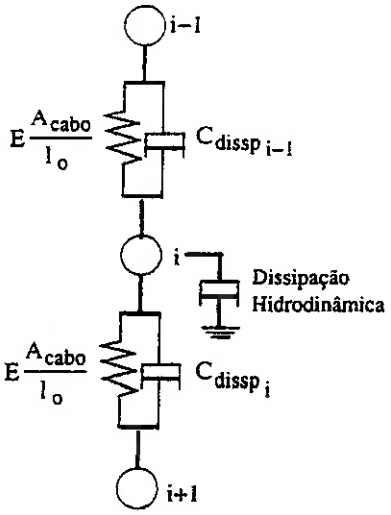

Figura 2.5 - Modelo de elemento do cabo umbilical

O atrito entre o isolamento dos fios condutores com as camadas dos materiais de proteção externos que revestem o cabo umbilical causam o efeito de amortecimento. Neste caso, o amortecimento é considerado natural e modelado proporcionalmente à diferença entre dois nós consecutivos da seguinte maneira:

$$P_i = C_{cabo}(\eta_{S_i} - \eta_{S_{i-1}})$$

Onde η_{Si} é a velocidade do elemento na direção s.

Como pode-se perceber acima, o amortecimento está descrito com relação ao sistema referenciado no veículo. Para transformá-lo em relação ao sistema inercial deve-se primeiramente projetar as velocidades dos nós na direção tangente do cabo como se segue:

$$\text{projeção} = \eta_{cabo_i} \frac{R_i}{|R_i|}$$

Onde R_i possui direção tangente ao cabo pois é obtido através da diferença de posição de dois nós consecutivos. O passo seguinte é a multiplicação escalar da projeção pelo versor tangente, obtendo a componente de velocidade na direção tangente ao cabo de cada nó:

$$P_i = C_{cabo}[(\eta_{cabo_i} - \eta_{cabo_{i-1}})\frac{R_i}{|R_i|}]\frac{R_i}{|R_i|}$$

$$P_i = C_{cabo}\frac{[(\eta_{cabo_i} - \eta_{cabo_{i-1}})R_i]R_i}{|R_i|^2}$$

$$P_i = C_{cabo}(\eta_{s_i} - \eta_{s_{i-1}})$$

2.7.1.2 Forças Externas

Conforme mostrado anteriormente, a resultante externa possui duas componentes, a força restaurativa F_G e a força de arrasto hidrodinâmico F_F, e pode ser representada da seguinte maneira:

$$F_i = \frac{1}{2}(F_{F_i} - F_{F_{i-1}}) + F_{G_i}$$

E a força hidrodinâmica possui componentes normal e tangencial conforme a expressão:

$$F_{F_i} = F_{n_i} + F_{t_i} = \frac{1}{2}\rho d_i(C_n U_{n_i}|U_{n_i}| + C_t U_{t_i}|U_{t_i}|)|R_i|$$

Onde

ρ = massa específica da água.

C_n = coeficiente de arrasto normal.

C_t = coeficiente de arrasto tangencial.

U_t = componente tangencial da velocidade de escoamento do fluido para o elemento do cabo:

$$U_{t_i} = \frac{[(\eta_c - \eta_{cabo_i})R_i]R_i}{|R_i|^2}$$

U_n = componente normal da velocidade de escoamento do fluido para o elemento do cabo:

$$U_{n_i} = (\eta_c - \eta_{cabo_i}) - U_{t_i}$$

η_c = velocidade da correnteza.

O método de discretização de massas concentradas possui algumas limitações quanto a convergência dos esforços internos aos nós, além de uma dificuldade em representar curvaturas muito acentuadas e de um tempo maior de convergência se o cabo estiver fora das condições iniciais. Porém, em Souza (2003) podemos comprovar a eficiência deste modelo para o caso aqui estudado através de comparações com o sistema físico.

2.8 Sistema propulsor

O UUV possui seis propulsores dispostos conforme mostrado na figura 2.6.

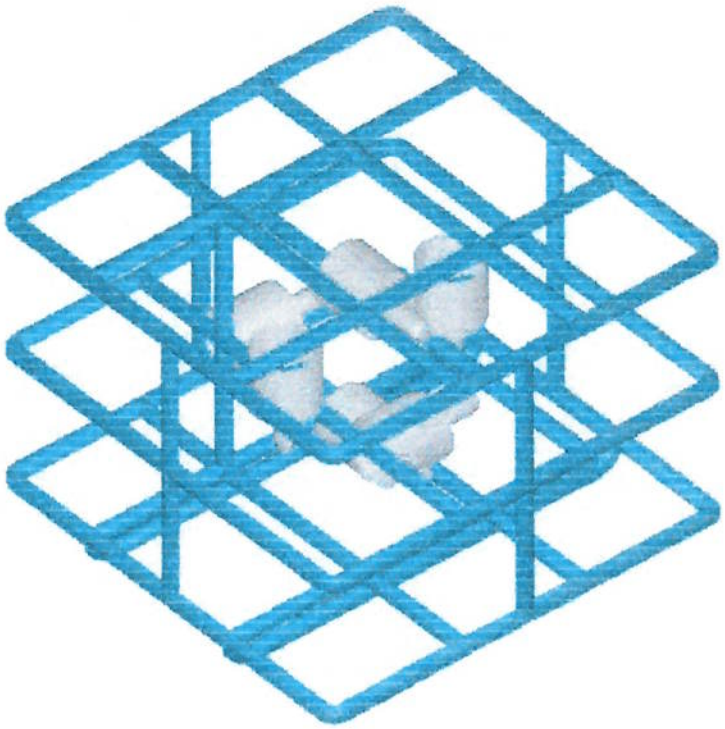

Figura 2.6 – Modelo do ROV

Para descrever a dinâmica do propulsor, o estudo foi dividido em duas partes: a primeira diz respeito a hidrodinâmica envolvida e a segunda trata mais especificamente da dinâmica do motor elétrico que aciona o propulsor.

2.8.1 Hidrodinâmica

Existem diversas abordagens para a modelagem deste efeito, porém, como indicado em Souza (2003), será utilizada aquela que proporciona melhor relação entre resultados e custos computacionais. Tal modelagem é a que segue a abordagem deste fenômeno em dois estados, com o acionamento do motor elétrico sendo feito através do controle de tensão.

Para esta análise foi considerada a existência de um envoltório ou duto ao redor do hélice (figura 2.7), o que resulta em melhores resultados para a resposta da propulsão (Allmendinger ed., 1990); (Healey et al., 1995).

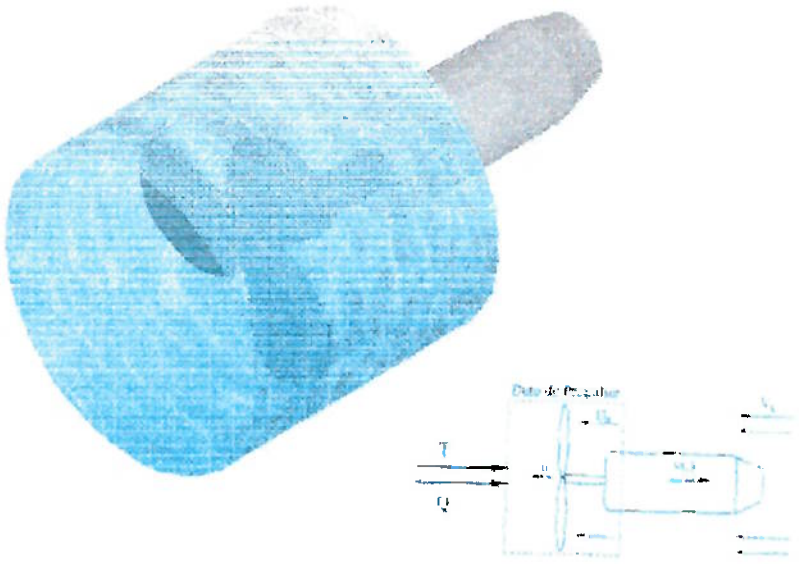

Figura 2.7 – Modelo de um propulsor

2.8.1.1 Mapeamento Da Hélice

Em um sistema de dois estados, pode-se identificar dois esforços que surgem da ação do propulsor, na direção longitudinal do mesmo: o empuxo (T) que é efetivamente o que faz a propulsão do veículo, e o torque (Q) referente ao carregamento hidrodinâmico sobre as pás da hélice do propulsor.

Define-se então:

U_a – velocidade média axial do fluido pelo invólucro do hélice

U_p – velocidade média tangencial do hélice

p_{prop} – passo do hélice

α_e – ângulo de ataque do hélice

Através do equacionamento apresentado em Lewis (1988) obteve-se o seguinte gráfico (figura 2.8), representando as relações geométricas dos ângulos de ataque α_e e do passo p_{prop}:

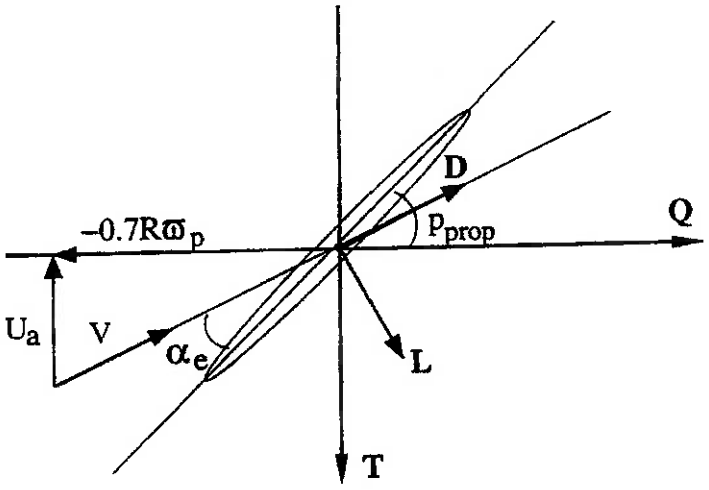

Figura 2.8 - Gráfico para cálculos da hélice

Definindo a velocidade tangencial relativa da hélice como:

$$U_p = 0,7R\omega_p$$

Onde R é o raio do hélice. Pode-se então definir o ângulo de ataque do hélice em relação ao fluido como:

$$\alpha_e = p_{prop} - \arctan(U_a/U_p)$$

E a velocidade resultante como:

$$V^2 = U_p^2 + U_a^2$$

Assim, utilizando Healey et al. (1995) pode-se determinar a força de arrasto (D) e de sustentação (L) do hélice no fluido, através dos coeficientes de arrasto (C_d) e sustentação (C_l) máximos:

$$D = 0,5\rho V^2 A_{prop} C_d$$
$$L = 0,5\rho V^2 A_{prop} C_l$$

Onde

$$C_d = C_{dmax}(1-\cos(2\alpha_e))$$
$$C_l = C_{lmax}\text{sen}(2\alpha_e)$$

Decompondo as forças de sustentação e arrasto de acordo com a figura, pode-se alcançar as expressões para os valores desejados de Q e T. Para isto basta utilizar:

$$\theta = p_{prop} - \alpha_e$$

$$T = L\cos(\theta)-D\text{sen}(\theta)$$
$$Q = 0{,}7R[L\text{sen}(\theta)+D\cos(\theta)]$$

2.8.1.2 Modelagem do Fluido

Para a determinação da velocidade U_a do fluido foi utilizada a expressão do momento linear do fluido para o volume de controle ao redor do propulsor. Tal maneira de obtenção de U_a, não é a mais completa do ponto de vista teórico, mas a literatura indica bons resultados práticos.

$$\dot{U}_a = -K_4 K_3^{-1}\overline{U_a}\left|\overline{U_a}\right|K_3^{-1}T$$

Onde:

$$K_3 = \rho A_{prop} L\gamma$$
$$K_4 = \rho A_{prop}\Delta\beta$$

Define-se:

$\Delta\beta$ = coeficiente do momento de fluxo em regime

L = comprimento do duto

γ = coeficiente de massa adicionada

A_{prop} = área da seção transversal do duto

Chega-se assim a:

$$\overline{U_a} = U_a - U_o$$

Onde U_o = velocidade do veículo em relação ao fluido.

Assim obtêm-se a seguinte malha de controle:

2.8.2 Motor CC

Do equacionamento do motor CC controlado por tensão de armadura:

$$L_a \frac{di_a}{dt} = -R_a i_a - 2\pi * K_{emf} n + V_m$$

$$2\pi * J_m \frac{dn}{dt} = K_t i_a - 2\pi * K_f n - Q(n, \overline{U_a})$$

Onde:

L_a = indutância da armadura

R_a = resistência da armadura

V_m = tensão da armadura

i_a = correente da armadura

K_t = constante de torque

J_m = inércia do motor

K_{emf} = constante da força contra-eletromotriz

K_f = coeficiente de atrito viscoso do motor

Considerando o hélice conectado diretamente ao eixo do motor, $n=\omega/2\pi$ e desprezando o efeito indutivo, obtêm-se:

$$\omega = K_1 \omega + K_2 V_m - K_Q Q$$

$$\omega_p = \omega/N$$

Com

$$K_1 = I_{prop}^{-1}[R_a^{-1}K_t K_{emf} + K_f]$$

$$K_2 = I_{prop}^{-1}[R_a^{-1}K_t]$$

$$K_Q = I_{prop}^{-1}$$

Onde:

I_{prop} = inércia resultante do rotor e do hélice

N = relação de transmissão

Aplicando a transformada de Laplace:

$$\omega(s) = \frac{1}{[L_a J_m s^2 + R_a J_m s + K_t]} V_m(s) - \frac{R_a[L_a s + 1]}{[L_a J_m s^2 + R_a J_m s + K_t]} Q(s)$$

Que pode ser escrito de maneira simplificada como:

$$\omega(s) = h_{vm}(s)V_m(s) - h_Q(s)Q(s)$$

2.8.3 Mapeamento do controlador

Para ser possível relacionar as saídas do controlador, ou as especificações de controle em sinais de acionamento, foi empregado um modelo quase-estático aproximando a relação entre propulsão e velocidade de rotação do hélice.

Neste modelo tem-se que o valor do empuxo em regime é:

$$T = \rho D_{helice}^4 K_t(J_o) n|n|$$

Onde:

ρ = densidade do fluido

$D_{hélice}$ = diâmetro da hélice

K_t = coeficiente de propulsão

O coeficiente de propulsão é função de uma grandeza adimensional denominada J_o, que não será discutida neste trabalho.

Adotando:

$$K_t = \alpha_1 + \alpha_2 \frac{V_a}{nD_{helice}}$$

e

$$V_a = (1-\omega_f)v(i)$$

Onde ω_f é um fator relacionado a "perda" de eficiência do fluxo de fluido pelo propulsor, que será tido como muito pequeno, devido a posição dos propulsores ser coincidente com os graus de liberdade. O índice i vai de 1 até o número total de graus de liberdade, e α_1 e α_2 foram determinados experimentalmente em de Souza (2003).

Tendo esta relação, pode-se mapear linearmente o esforço do controlador por;

$\tau_{contr} = Bu$

A matriz B presente na fórmula acima, consiste da composição entre a hidrodinâmica de cada atuador, especificada em B_{prop}, e sua posição na estrutura do veículo, especificada em B_{config}, permitindo assim, que o movimento acoplado em mais de um grau de liberdade seja atendido por um mesmo propulsor. Estas três matrizes estão mostradas abaixo para o caso do UUV estudado.

$$B_{prop} = \begin{bmatrix} b_1(J_o) & 0 & 0 & 0 & 0 & 0 \\ 0 & b_2(J_o) & 0 & 0 & 0 & 0 \\ 0 & 0 & b_3(J_o) & 0 & 0 & 0 \\ 0 & 0 & 0 & b_4(J_o) & 0 & 0 \\ 0 & 0 & 0 & 0 & b_5(J_o) & 0 \\ 0 & 0 & 0 & 0 & 0 & b_6(J_o) \end{bmatrix}$$

$$B_{config} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & -b & b & 0 & 0 \\ 0 & 0 & 0 & 0 & -c & c \\ a & -a & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} b_1(J_o) & b_2(J_o) & 0 & 0 & 0 & 0 \\ 0 & 0 & b_3(J_o) & b_4(J_o) & 0 & 0 \\ 0 & 1 & 0 & 0 & b_5(J_o) & b_6(J_o) \\ 0 & 0 & -b_3(J_o)b & b_4(J_o)b & 0 & 0 \\ 0 & 0 & 0 & 0 & -b_5(J_o)c & b_6(J_o)c \\ b_1(J_o)a & -b_2(J_o)a & 0 & 0 & 0 & 0 \end{bmatrix}$$

É possível notar pela matriz B, que os valores de a, b e c são os braços entre os propulsores e o centro de gravidade do UUV. Outro dado relevante é que no caso de rotação, os propulsores do eixo desejado são acionados para direções opostas, permitindo assim, a rotação em torno do eixo.

Para que a distribuição de esforços em cada propulsor seja ótima será utilizada a seguinte fórmula (Fossen, 1994):

$$\mathrm{B}_{\mathrm{W}}^{*} = \mathrm{W}^{-1}\mathrm{B}^{\mathrm{T}}(\mathrm{B}\mathrm{W}^{-1}\mathrm{B}^{\mathrm{T}})^{-1}$$

Porém, como não existem neste trabalho propulsores prioritários, pode-se assumir que W = I. E como o número de graus de liberdade é igual ao número de propulsores, pode-se ainda assumir que $B^*=B^{-1}$.

Finalmente determina-se que o sinal de referência deve ser o sinal de rotação da hélice através da seguinte relação:

$$\eta_{ref} = \text{sign}(B_W^* \tau_{ctrl}) \sqrt{B_W^* \tau_{ctrl}} = \frac{\omega_{ref}}{2\pi}$$

Este sinal será utilizado na malha de controle interna do propulsor.

3 SISTEMA DE CONTROLE

Como pode-se observar em Fossen,(1994) a estratégia mais empregada no controle de veículos submarinos é a utilização de controladores do tipo PID.
Porém, como os veículos submarinos possuem uma natureza de movimento não linear, a característica linear deste tipo de controlador deve ser levada em consideração para que o resultado obtido seja compatível com o desejado.
Para que este problema não seja relevante, deve-se definir o mais precisamente possível quais as movimentações deverão ser executadas pelo veículo submarino.

No caso deste trabalho, foi decidido pela implementação de um sistema de controle do tipo PID. Esta escolha foi feita em de Souza (2003), após testes entre alguns tipos de controladores lineares e de controladores não lineares para os seguintes casos:

1. Foi imposto um deslocamento e uma certa correnteza, porém foi desprezada a ação do cabo umbilical. Foram feitos dois testes, um considerando a ação dos propulsores, outro desconsiderando este fator.
2. Foram utilizados as mesmas condições e os mesmos parâmetros do caso anterior, mas desta vez levando em consideração a influência do cabo umbilical, que teve sua extremidade superior considerada fixa à origem do sistema de coordenadas inicial.

Após a apresentação dos resultados obtidos nesses dois casos típicos, fica compreensível a escolha em de Souza (2003) de um controlador linear do tipo PID, pois certamente representa a alternativa que melhor concilia a facilidade de implementação com a aproximação da situação real desejada.

3.1 Controlador PID

A primeira medida que deve ser tomada quando se fala de um sistema de controle do posicionamento de um veículo submarino é garantir que o erro de seja nulo no regime de posicionamento, isto é conseguido através da inserção de um termo integrador no sistema de controle.

O próximo passo é definir os coeficientes, proporcional k_P, integral k_I e derivativo k_D do controlador. Para isso, foi utilizado o um procedimento no qual os coeficientes do denominador da função de transferência de malha fechada

$$G_{mf}(s) = \frac{kk_D(k_P s + k_I)}{s^3 + (a + kk_D)s^2 + kk_P k_D s + kk_I k_D}$$

foram igualados aos termos do polinômio característico desejado, especificado através do coeficiente de amortecimento ζ, da freqüência natural desejada ω_n e da constante α. Estes parâmetros devem ser escolhidos de maneira que o sistema em malha fechada seja caracterizado por dois pólos complexos dominantes. Já as constantes $a(v^*(.))$ e k são obtidas através da função de transferência de malha aberta onde:

$$G_{planta}(s) = \frac{k}{s - a(u^*)}$$

Assim, resolvendo o sistema para os coeficientes e os parâmetros utilizados, obtêm-se os seguintes resultados, que os relacionam:

$$k_D = (\alpha_{polo}\zeta\omega_n + 2\zeta\zeta_n - a)/k,$$
$$k_P = (2\alpha_{polo}(\zeta\zeta_n)^2 + \omega_n{}^2)/(kk_D),$$
$$k_I = (\alpha_{polo}\zeta\omega_n{}^3)/(kk_D).$$

E considerando os seis graus de liberdade, a seguinte lei de controle para o controlador pode ser utilizada:

$$\tau_{ctrl}(s) = J^T(\eta\eta)_D\left(\frac{K_P s + K_I}{s}(\eta_{ref} - \eta) - \eta\right),$$

Onde as matrizes K_P, K_I e $K_D \in R^{6x6}$, são diagonais. Os elementos da diagonal de cada matriz são os coeficientes do controlador PID para cada um dos graus de liberdade. Nesta última equação deve-se notar a existência do operador J, que assim como na modelagem cinemática apresentada na primeira parte do trabalho, representa a passagem do sistema de coordenadas fixo na terra para o sistema de coordenadas móvel do veículo submarino.

4 IMPLEMENTAÇÃO DAS FUNÇÕES EM LINGUAGEM C

Para otimizar o processo de simulação do sistema de controle do UUV, será utilizada uma função do MatLab chamada S-Function. Esta função permite ao usuário criar seus próprios blocos contendo funções implementadas através de outras linguagens, tais como C, C++, Fortran e Ada. O comando S-Function possibilita que o usuário determine não só a função e seus parâmetros mas também a quantidade de entradas e saídas, evitando assim a utilização de multiplexadores.

Apesar da linguagem C não apresentar as ferramentas matemáticas presentes na linguagem do MatLab, sua funcionalidade e sua grande gama de aplicações, fez com que esta fosse escolhida para a implementação das S-Functions. Outros fatores que levaram a escolha desta linguagem foram o menor índice de erros durante a simulação numérica e a maior velocidade de processamento durante a simulação de sistemas de controle complexos. Para os métodos mais simples, que não consomem um elevado tempo de processamento, foram utilizados blocos prontos encontrados nas bibliotecas do programa Simulink.

Para a implementação destas funções em C foi utilizada uma função do MatLab chamada S-Function Builder, e o modo como foi feita esta programação será descrito resumidamente a seguir.

O S-Function Builder gera, através de dados inseridos pelo usuário, um algoritmo em linguagem C. Os dados que devem ser colocados pelo usuário são: entradas, saídas e parâmetros. O usuário deve informar ainda, a função que relaciona a saída com a entrada. Isto pode ser feito diretamente através de expressões numéricas inseridas no próprio campo das saídas (para casos que envolvem operações mais simples) do Builder ou através de métodos externos, em linguagem C, declarados nas bibliotecas externas (para casos que envolvem funções mais complexas).

Para tornar o programa mais flexível, de forma a poder ser utilizado para outros modelos de veículos submarinos, utilizou-se uma única biblioteca externa para armazenar todos os parâmetros do modelo estudado.

4.1 Implementação das funções do ROV

A seguir, será apresentado o modelo completo comentado do ROV com as funções em linguagem C.

A máscara a seguir (figura 4.1) representa o modelo matemático completo do ROV, possuindo duas entradas (sinal de controle e correnteza) e seis saídas (posição inercial do ROV, velocidade inercial do ROV, ação física dos propulsores, o cabo umbilical e velocidade relativa do ROV em relação à correnteza).

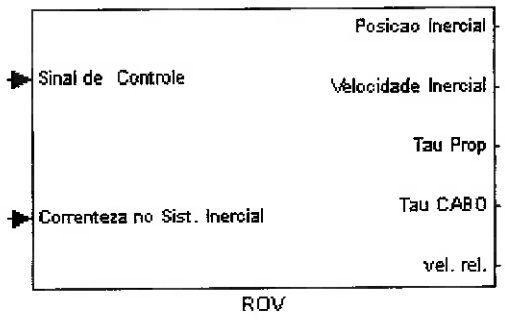

Figura 4.1 - Máscara inicial do ROV

A relação entre as entradas e saídas deste sistema é feita através das operações mostradas a seguir:

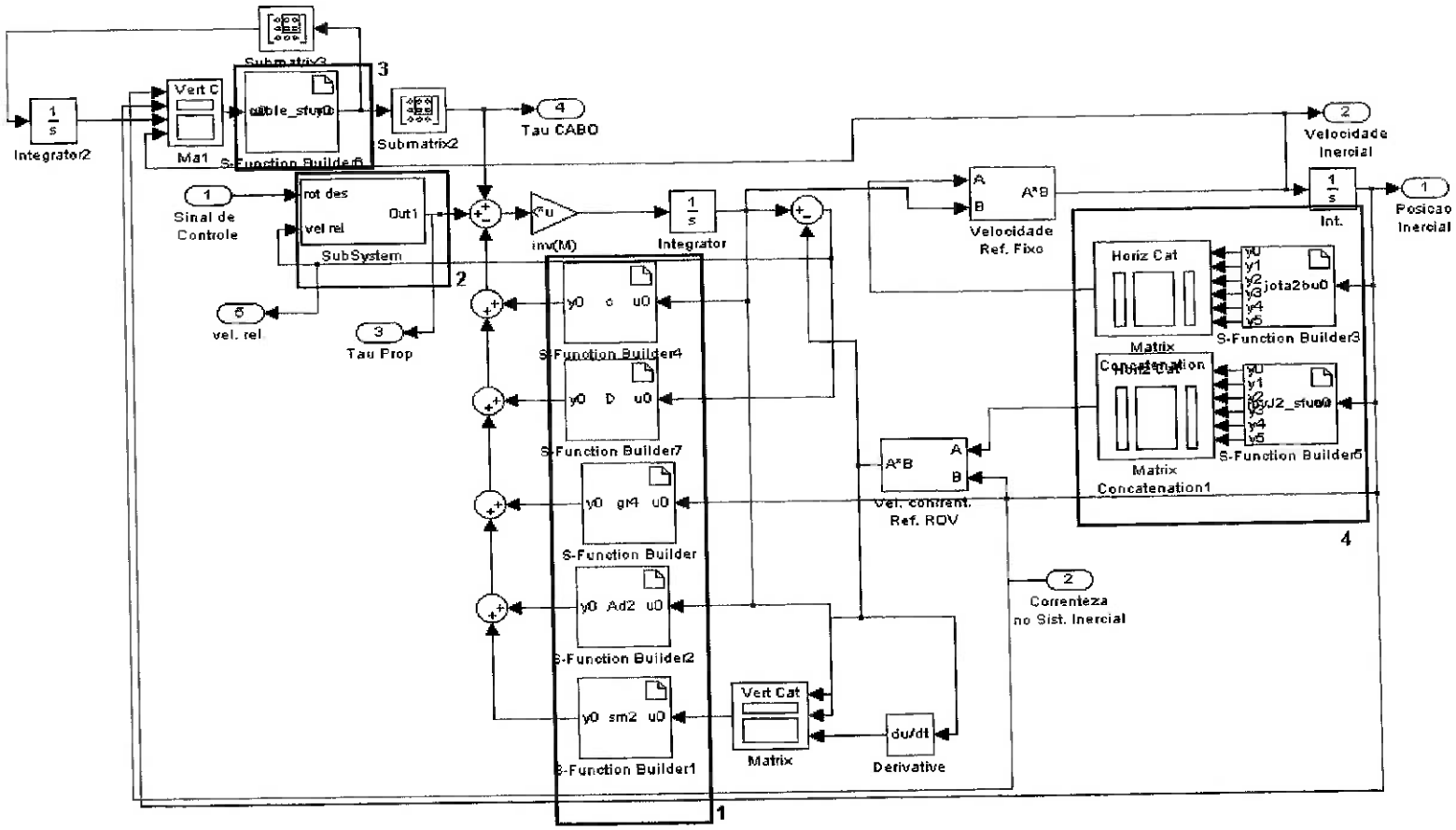

Figura 4.2 – Modelo do ROV em Simulink

Na figura 4.2, podemos identificar os diversos sub-sistemas presentes no modelo do ROV. Dentro do retângulo 1, pode-se visualizar os parâmetros da equação geral da dinâmica do veículo (apresentada no item 2.6). Os parâmetros C, D, G, Ma (massa adicionada) e SM (esforços devidos à correnteza) são obtidos através de blocos que representam S-Functions. Todos os códigos referentes a estas funções estão compilados no anexo A.

Dentro do retângulo 3, identifica-se a função que descreve o comportamento do cabo umbilical (descrito teoricamente no item 2.7). Pode-se notar que este bloco representa uma S-Function que simula o comportamento do cabo através de uma rotina implementada em linguagem C cujo código encontra-se no anexo A.

No retângulo 4 pode-se notar a existência de dois blocos que representam S-Functions. Eles são responsáveis pela passagem das coordenadas cinemáticas do referencial inercial para o referencial móvel. Isto é feito através da matriz J (conforme explicado no item 2.1).

Finalmente no retângulo 2, encontra-se outra máscara (que possui como entradas o sinal de controle e a velocidade relativa do modelo, e como saída a ação física dos propulsores). Ela descreve o sistema de propulsão do veículo que será detalhado a seguir:

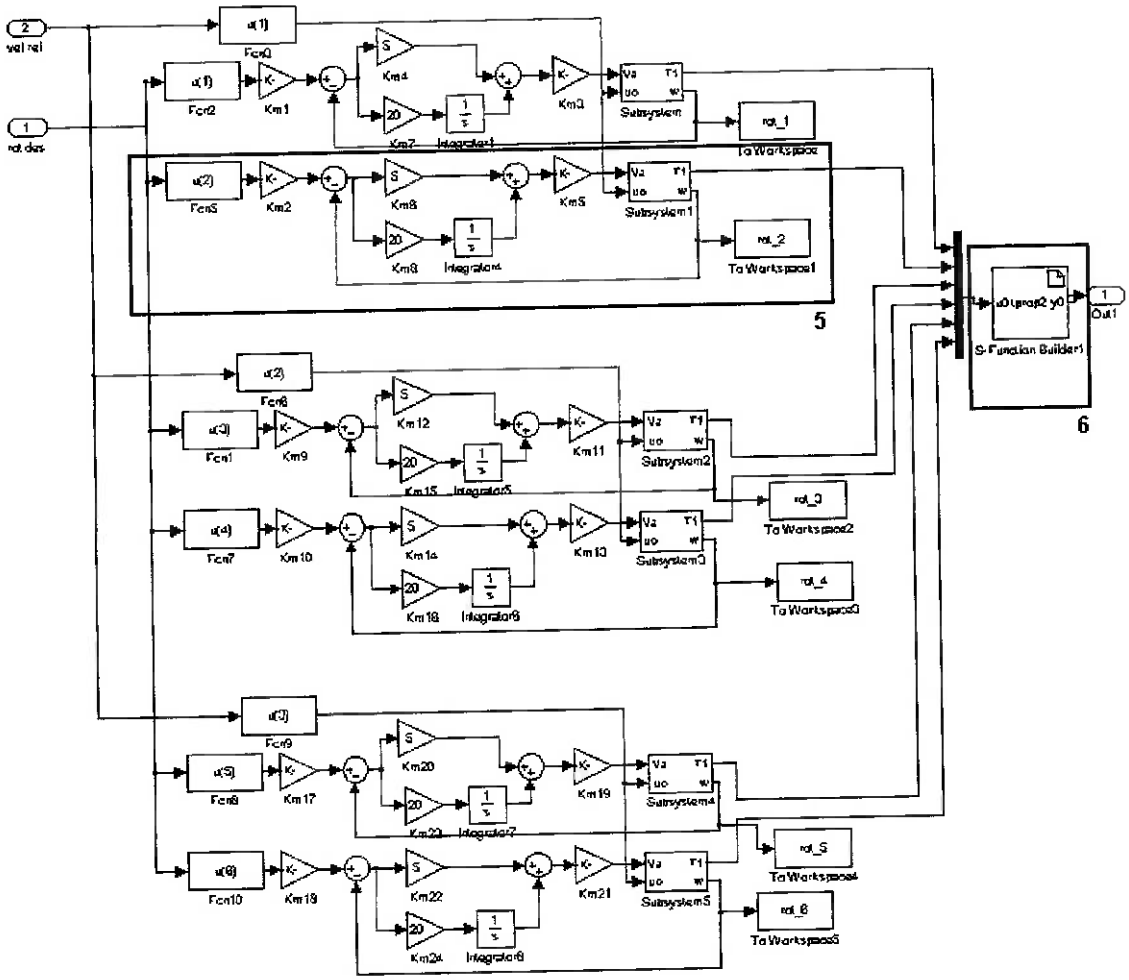

Figura 4.3 - Modelo do sistema de propulsão em Simulink

Abrindo esta máscara (figura 4.3) pode-se perceber a presença de um sistema de controle para cada um dos seis propulsores existentes. O sistema de cada propulsor (descrito no item 2.8) é representado pelo conjunto de blocos delimitados na figura acima pelo retângulo 5. Neste conjunto de blocos pode-se notar uma outra máscara que tem como entradas a velocidade do submarino em relação ao fluido e a tensão desejada para o controle do motor elétrico e como saídas o torque devido ao carregamento hidrodinâmico e o empuxo do propulsor. Esta máscara pode ser vista com mais detalhes na figura 5.4.

Identifica-se também dentro do retângulo 6 um bloco relativo a uma S-Function, que tem como função concatenar as saídas dos seis sistemas de propulsão existentes.

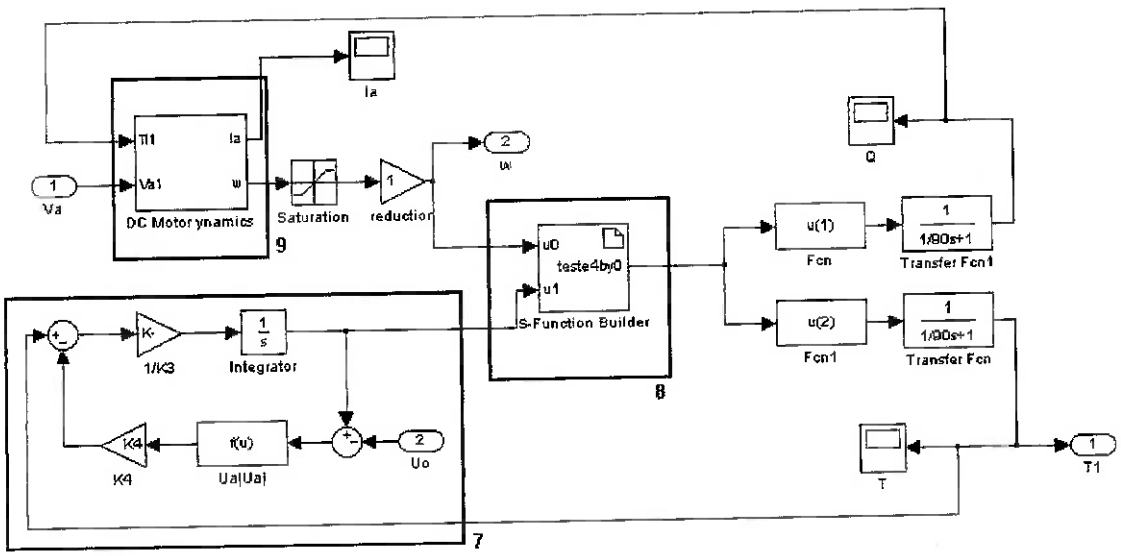

Figura 4.4 - Modelo do propulsor em Simulink

Na figura 4.4, verifica-se a presença de três retângulos:

O primeiro, de número 7, representa a modelagem do fluido, apresentada teoricamente no item 2.8.1.2.

O segundo, de número 8, mostra a S-Function que tem como função o mapeamento do hélice e foi apresentada no item 2.8.1.1.

Finalmente, tem-se o retângulo de número 9, que nos apresenta uma nova máscara responsável pelo modelo da dinâmica do motor DC, que impulsiona os propulsores. Esta máscara tem como entradas a tensão de controle e o torque devido ao carregamento hidrodinâmico e suas saídas são a velocidade angular e a corrente no motor. A teoria para este sistema está detalhada no item 2.8.2 e o conteúdo da máscara é mostrado a seguir (figura 4.5).

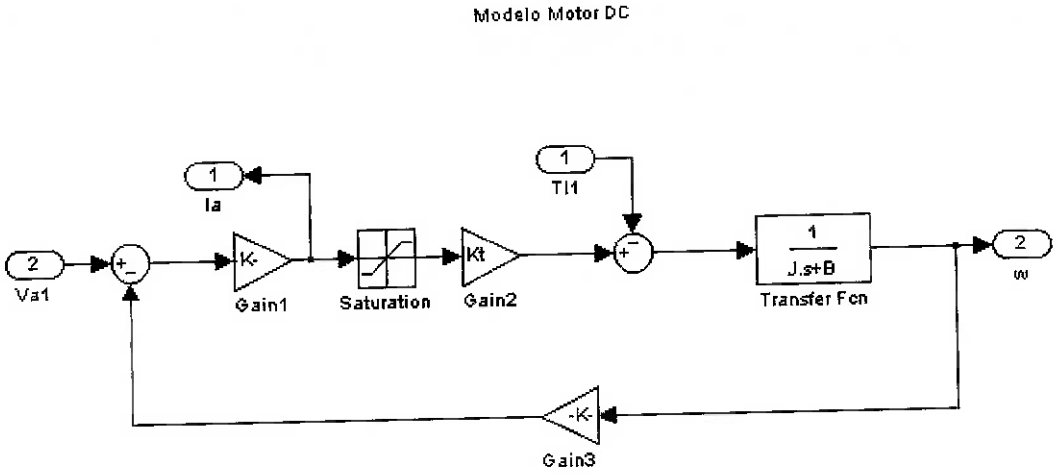

Figura 4.5 – Modelo do motor CC em Simulink

4.2 Implementação das funções de controle

Para a implementação do controlador PID foi utilizada a seguinte máscara (figura 4.6) no programa MatLab.

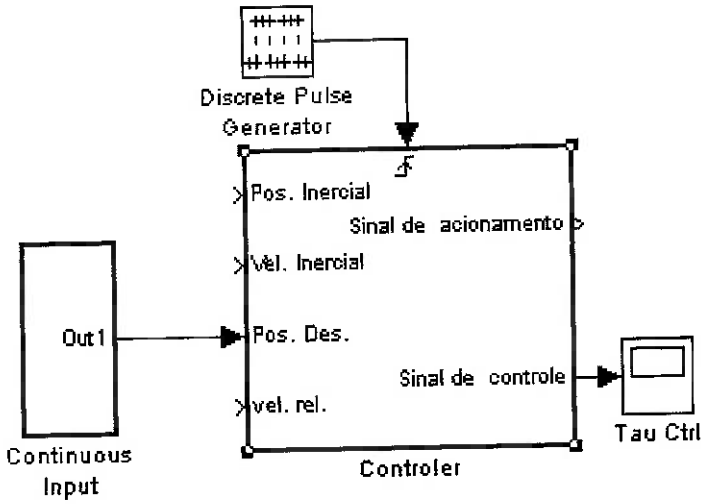

Figura 4.6 – Máscara de controle em Simulink

Pode-se perceber nesta máscara, as quatro entradas que serão utilizadas no controle do veículo submarino, três delas são referências à cinemática do veículo (velocidades relativa e inercial, e posição inercial) e a outra representa a posição desejada pelo

operador, com estes quatro dados serão obtidos os resultados que controlarão o submarino.

Esses resultados são mostrados nas duas saídas, uma representando o sinal de controle que vai agir sobre o sistema e propulsão do veículo, fazendo-o se movimentar para a posição desejada e a outra representa o sinal de controle que deve ser enviado para o veículo. Nesta simulação será utilizada a segunda saída apenas como um sinal gráfico para tornar possível a comparação dos resultados obtidos e a análise do sistema de controle do veículo. Este sinal é interpretado dentro do controlador para o controle do veículo submarino. Existe ainda uma entrada com um gerador de pulsos discretos que neste trabalho, funcionará apenas como um gatilho (trigger) do sistema, pois utilizaremos o controle discreto.

Ao abrir a máscara mostrada na figura 4.6 pode se observar o sistema de controle completo, que é como o mostrado na figura 4.7.

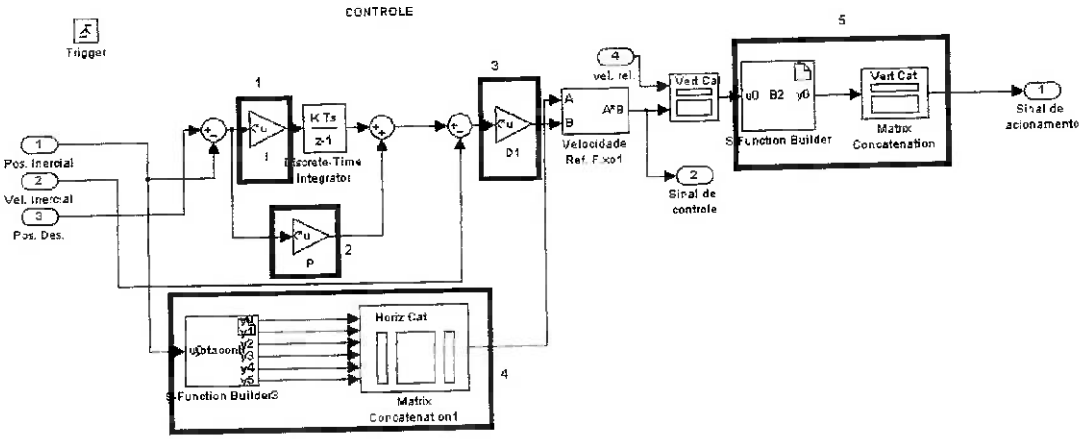

Figura 4.7 – Modelo do controle em Simulink

Nesta figura, pode-se perceber nos retângulos de números 1, 2 e 3 os ganhos integral, proporcional e derivativo respectivamente, caracterizando o controlador linear PID, que conforme explicado anteriormente foi o escolhido para ser utilizado neste caso.

Observando agora o retângulo de número 4, nota-se que ele representa a matriz do operador jota tranposta que foi apresentada na última equação do item anterior, e que tem como função transformar a posição a partir do referencial fixo em posição relativa ao referencial móvel. O código desta função também se encontra no anexo A. Mais uma vez, deve-se comentar o fato da existência da função de concatenação de matriz e justificá-la devido à impossibilidade de se utilizar uma saída de duas dimensões em uma S-Function.

Finalmente, o retângulo de número 5 é responsável pela transformação do sinal de controle em sinal de acionamento para o sistema de propulsão. O tamanho das matrizes desta função se justifica, pois como descrito no início do trabalho, o veículo possui seis graus de liberdade. O código desta função, como o das demais funções criadas em linguagem C, se encontra no anexo A.

5 AMBIENTE GRÁFICO

Para implementar o ambiente gráfico bem como para visualizar o comportamento do ROV utilizou-se neste trabalho a tecnologia de VRML (Virtual Reality Modeling Language). Com a utilização desta linguagem pôde-se criar a interatividade entre o modelo dinâmico do ROV construído em SimuLink , o usuário e o ambinte tridimensional de realidade virtual.

O termo VRML foi utilizado inicialmente por Tim Berners-Lee em 1994 ao discutir sobre a necessidade de se estabelecer normas para paginas da World Wide Web tridimensionais. Esta norma foi então implementada e em 1997 adotada como ISO/IEC 14772-1:1997 e referida como VRML97. Esta representa uma plataforma aberta e flexível para criação de cenas tridimensionais interativas (virtual worlds).

Para editar o VRML foi utilizado neste trabalho uma ferramenta do MatLab chamada Virtual Reality Toolbox. Esta inclui o software V-Realm Builder da Ligos Corp que possibilita a criação de *virtual worlds* com extrema eficiência.

Resumidamente, para o ambiente gráfico do trabalho foi criado um *virtual world,* utilizando o V-Realm Builder, e associado ao modelo do ROV implementado em SimuLink através do Virtual Reality toolbox. Este processo será explicado detalhadamente nos itens seguintes.

5.1 Sistema de coordenadas do VRML

Diferente do modelo do ROV, o VRML utiliza um sistema de coordenadas global com o eixo y apontando para cima de forma que o eixo z represente a aproximação e o afastamento do objeto de simulação da tela de visualização como mostrado na figura 5.1.

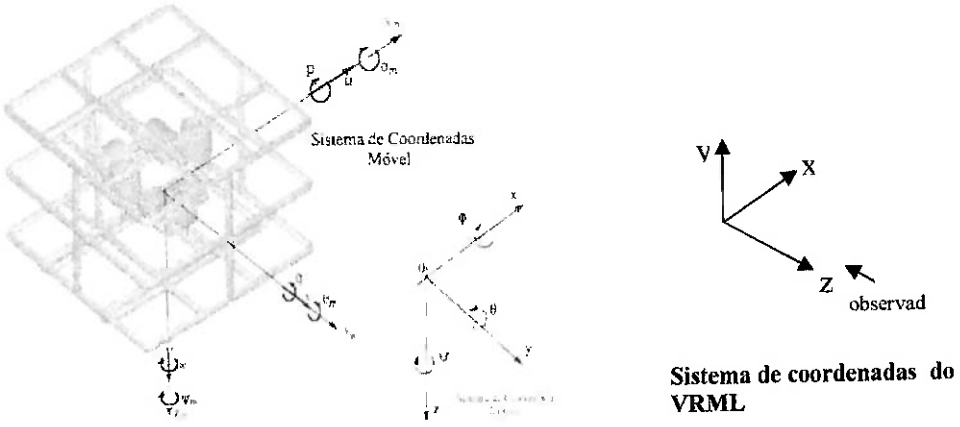

Figura 5.1 – Sistemas de coordenadas do ambiente gráfico

5.2 Virtual World

O modelo tridimensional do ROV foi criado utilizando-se a ferramenta de CAD Micro Station e exportada para o V-Realm Builder, como podemos verificar na figura 5.2:

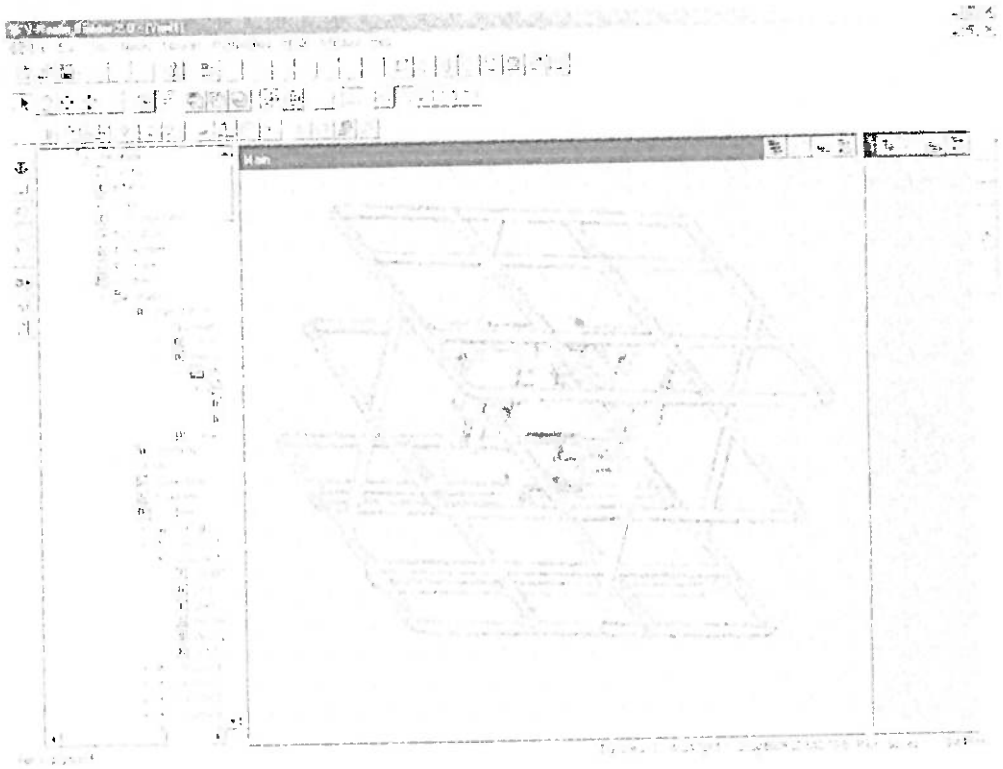

Figura 5.2 – Modelo do ROV para o ambiente gráfico

Para facilitar a visualização da movimentação do veículo, foi criado um Virtual World composto pelo ROV e alguns pontos de referência (figura 5.3).

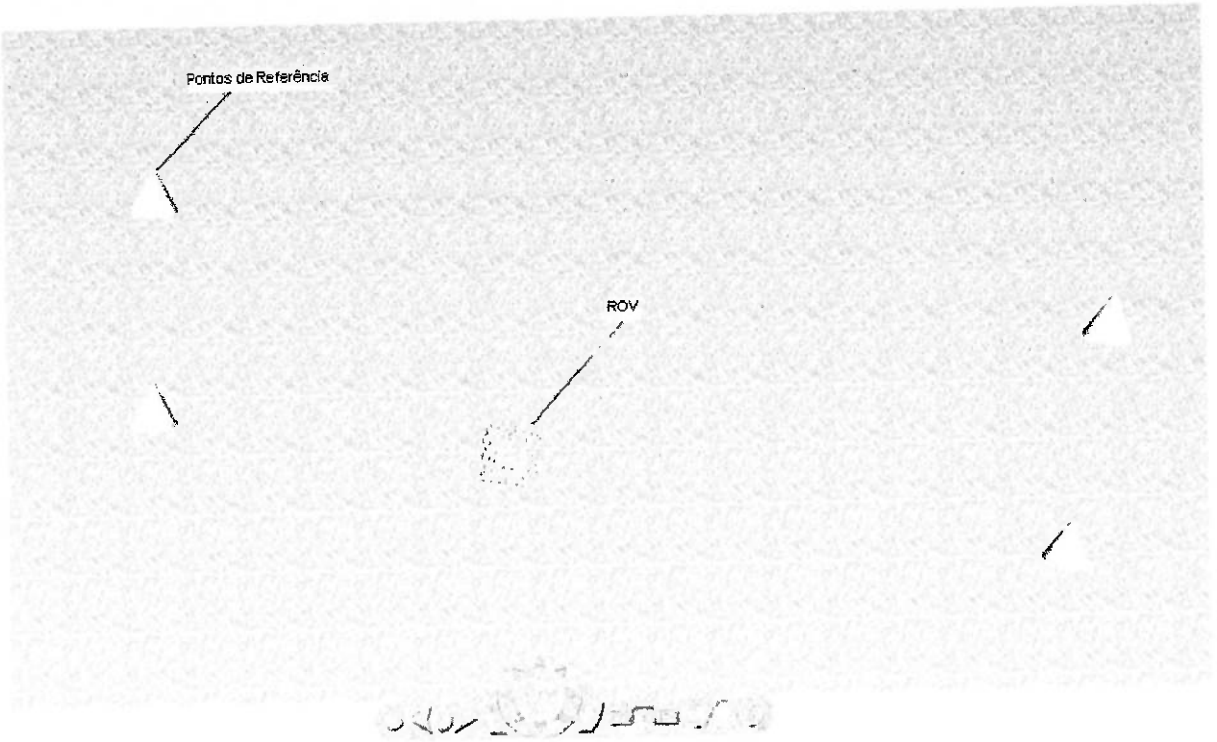

Figura 5.3 – Ambiente gráfico

5.3 Virtual Reality Toolbox

Para associar o Virtual World criado com o modelo em SimuLink, utilizou-se o Virtual Reality Toolbox (figura 5.4).

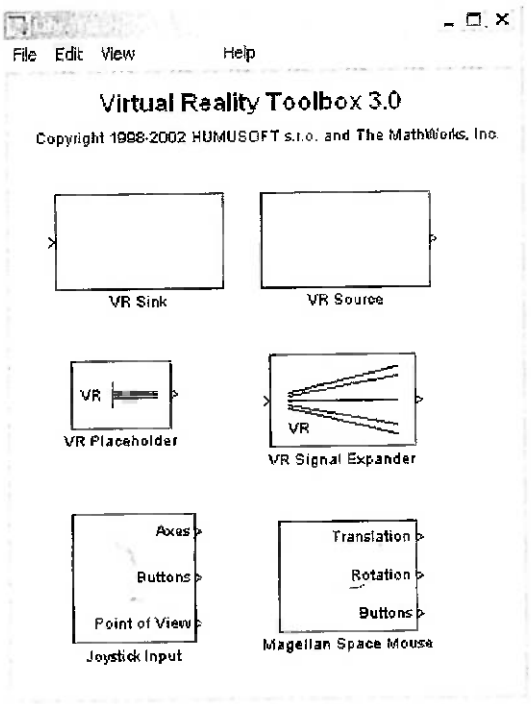

Figura 5.4 - Virtual Reality Toolbox

Através do bloco "VR Sink" as saídas da simulação (valores da posição e orientação do ROV) são passadas ao ROV no Virtual World através do vetor posição inercial. A interpretação dos valores contidos neste vetor gera o movimento realizado pelo ROV no ambiente Virtual.

6 SIMULAÇÕES E TESTES

Como mencionado no início deste trabalho, um dos objetivos era o de aprimorar o modelo desenvolvido em linguagem m através da utilização de funções na linguagem C. Desta maneira, o teste para validação destas novas funções desenvolvidas foi feito através da comparação dos resultados obtidos individualmente de cada uma delas para um mesmo valor de entrada. A seguir será demonstrado este procedimento para validação da função "invJ":

Primeiro cria-se um arquivo .mdl (Simulink) contendo apenas a MatLab function, uma entrada, e uma saída em forma de vetor conforme ilustrado na figura 6.1:

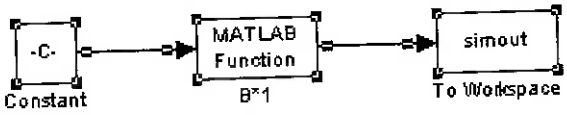

Figura 6.1 – Modelo da função m para teste

O resultado é armazenado na variável simout, cujos valores podem ser visualizados no workspace do MatLab.

Os resultados obtidos para esta simulação foram:

simout(:,:,1) =

2.7114
-1.4958
-1.7149
4.0823
-2.0501
6.1502

simout(:,:,2) =

2.7114
-1.4958
-0.9687
4.0823
-2.0501
2.1935

simout(:,:,3) =

2.7114
-1.4958
-0.9687
4.0823
-2.0501
2.1935

O mesmo procedimento foi tomado para a função desenvolvida em linguagem C (S-Function), como mostrado na figura 6.2.

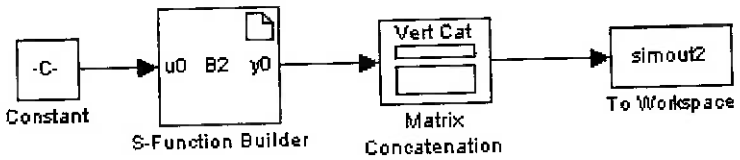

Figura 6.2 - Modelo da função c para teste

Deve-se mais uma vez ressaltar que o bloco Matrix Concatenation é utilizado pois o S-Function Builder não gera saídas com mais de uma linha. E como pode-se observar no resultado do exemplo da MatLab Function, a saída desejada contém 6 linhas e uma coluna.

simout2(:,:,1) =

2.7114
-1.4958
-1.7149
4.0823
-2.0501
6.1502

simout2(:,:,2) =

2.7114
-1.4958

-0.9687
4.0823
-2.0501
2.1935

simout2(:,:,3) =

2.7114
-1.4958
-0.9687
4.0823
-2.0501
2.1935

Este mesmo tipo de teste de validação foi feito para todas as funções geradas com a linguagem C, alterando-se os valores das constantes entre positivos, negativos e nulos, e os resultados obtidos foram sempre os mesmos das funções geradas no código m. Desta forma pôde-se garantir a coerência dos resultados das funções.

Após esta etapa de testes das funções individualmente, foram executados testes comparativos entre os modelos completos. Os resultados obtidos, novamente foram semelhantes para o modelo com as funções em linguagem C e m.

Para garantir que o comportamento do ROV seria corretamente representado no ambiente gráfico, foram realizados testes do Virtual World desacoplado do sistema em SimuLink. Foram colocadas constantes de posição de forma a direcionarr o ROV nos pontos de referências criados. O veículo desempenhou uma trajetória retilínea em direção a estes pontos mostrando-se capaz de representar o percurso percorrido pelo ROV com coerência.

O Virtual World foi acoplado aos modelos, e pôde-se observar o comportamento semelhante do ROV para os dois casos. Além disso, notou-se uma concordância

entre os valores numéricos dos vetores de saída dos modelos e a trajetória percorrida pelo ROV.

7 CONCLUSÃO

Com este trabalho, pôde-se mostrar a possibilidade da implementação de um simulador para ROV's utilizando-se a linguagem C. Isso só foi possível devido à utilização de S-functions, e de métodos e bibliotecas externas desenvolvidos paralelamente.

Apesar do MatLab possuir um ambiente mais apropriado para as definições e operações com matrizes pôde-se neste trabalho provar que com o desenvolvimento de algoritmos específicos é possível utilizar matrizes na linguagem C. Um exemplo deste fato foi o desenvolvimento de uma função que possibilita a inversão de uma matriz 6x6 utilizada para a transformação entre os dois sistemas de coordenadas usados no modelo do veículo submarino tratado neste trabalho.
Quanto ao tempo de execução, os resultados obtidos foram abaixo do esperado, ou seja, apesar dos resultados obtidos através dos dois modelos terem sido semelhantes, o tempo do modelo que contém as funções em C foi maior que o do modelo com as funções em MatLab. Isto teve como motivo principal o fato de o modelo ainda estar sendo executado dentro do ambiente MatLab. A eficácia do modelo com as funções em C só poderá ser garantido através da implementação de um arquivo executável que possa "rodar" fora do ambiente MatLab. Este arquivo deve ter como saída o vetor de posição e de orientação do ROV em função do tempo, para ser visualizado através do ambiente gráfico.

O ambiente gráfico foi gerado de maneira satisfatória e representa fidedignamente, o vetor de orientação e posição que este deve ter como entrada. Além disso, as referências são parte essencial do ambiente, sem as quais a movimentação do ROV não pode ser observada propriamente.

8 BIBLIOGRAFIA

Allmendinger ed., E. E. (1990). **Submersible Vehicle Systems Design,** *The Society of Naval Architects and Marine Engineers*

Clayton, B. R. e Bishop, R. E. (1982). **Mechanics of Marine Vehicles**, *Gulf Publishing Company, London*

Driscoll, F. R., Lueck, R. G. e Nahon, M. (2000). Development and Validation of Lumped-Mass Dynamics Model of a Deep-Sea ROV System, **Applied Ocean Research 22:** 169-182

Fossen, T. I. (1994). **Guidance and conontrol of ocean vehicles**, *John Wiley & Sons.*

Healey, A. J., Rock, S. M., Cody, S., Miles, D. e Brown, J. P. (1995). Toward an Improved Understanding of Thruster Dynamics for Underwater Vehicles, **IEEE Journal of Oceanic Engineering 20 (4):** 354-361.

Indiveri, G. (1998). **Modelling and Identification of Underwater Robotic Systems**, *Tese (doutorado), Università degli Studi di Genova Facoltà di Ingegneria, Itália.*

Kalske, S. e Happonen, K. (1991). Motion Simulation of Subsea Vehicles, **Proceedings of the First International Offshore and Polar Engineering Conference,** *Edinburgh, UK 2: 74-84*

Lewis, D. J., Lipscombr, J. M. e Thomasson, P. G. (1984). The Simulation of Remotely Operated Underwater Vehicles, **Proceedings of ROV'84 Conference,** *San Diego, USA pp.245-251.*

Lewis, ed., E. V. (1988). **Principles of Naval Architecture**, *vol. II*, The Society of Naval Architects and Marine Engineers

Nomoto, M. e Hattori, M. (1986). A Deep ROV "Dolphin 3K": Desing and Performance Analysis, **IEEE Journal of Oceanic Engineering 11 (3):** 373-391.

Ogata, Katsuhiko (2000). **Engenharia de Controle Moderno**, *Prentice Hall Inc.*

Overview of S-Functions (2003), *The MathWorks Inc.*

Souza, E. C. (2003). **Modelagem e Controle de submarinos não tripulados**, *Escola Politécnica da USP*

Yokobiki, T., Koterayama, W., Yamaguchi, S. e Nakamura, M. (2000). Dynamics and Control of a Towed Vehicle in Transient Mode, **International Journal of Offshore and Polar Engineering 10** (1): 19-25

Yuh, J. (1990). Modeling and Control of Underwater Robotic Vehicles, **IEEE Transactions on Systems, Man and Cybernetics 20** (6): 1475-1483

APÊNDICE A
S-FUNCTIONS CRIADAS

A.1 Introdução ao S-Function Builder

A janela que se abre ao se inserir um bloco do S-Function Builder (figura A.1.1) em um arquivo de Simulink tem o seguinte conteúdo (figura A.1.2):

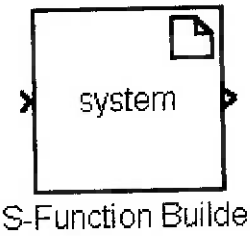

Figura A.1.1 – Bloco do S-Function Builder

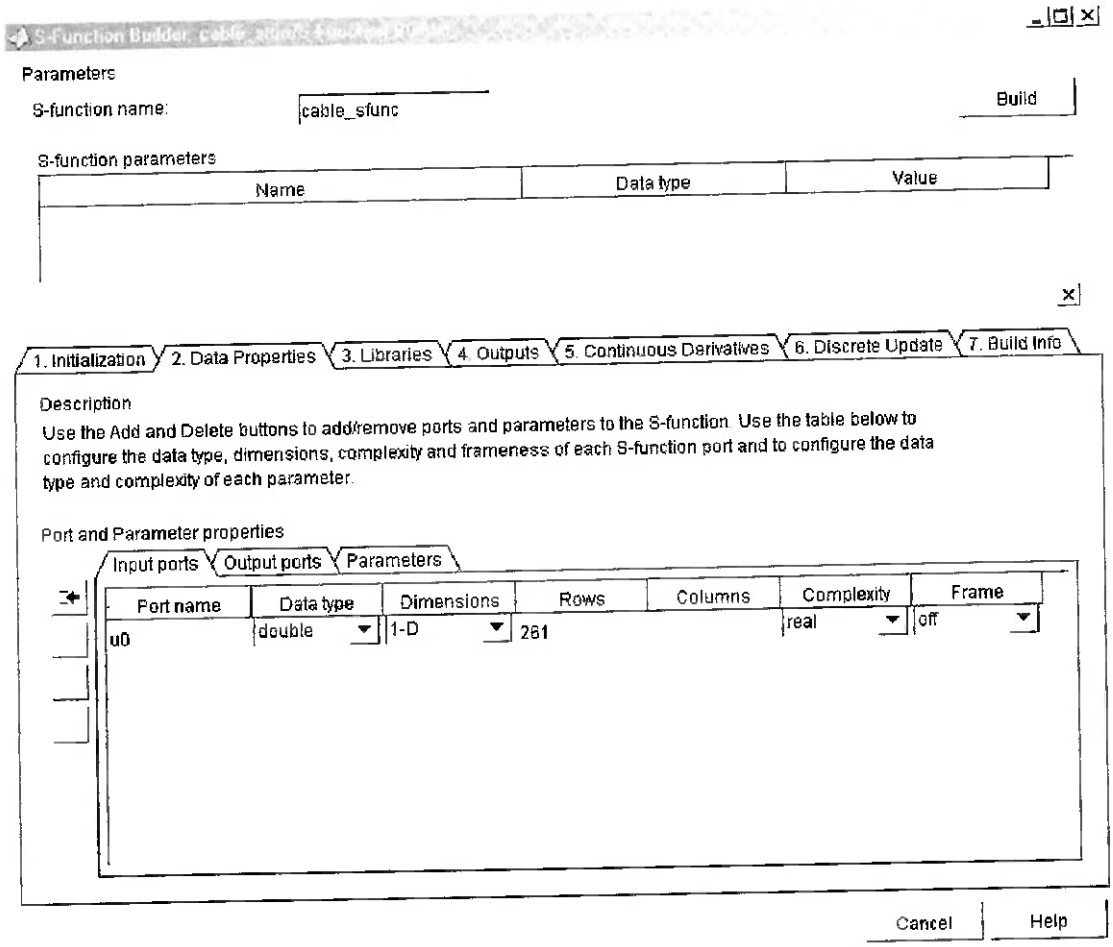

Figura A.1.2 – Tela do S-Function Builder

Para criar uma S-Functoin, devemos inserir as informações desejadas nas janelas deste programa. Segue uma descrição básica de cada uma das janelas:

Inicialization: Nesta janela devem ser colocados os dados relativos ao número de estados discretos e contínuos que se deseja uilizar, assim como suas condições iniciais. Outro dado importante é o tipo de tempo de amostra em que a S-Function deve ser executada.

Data properties: Nesta janela devem ser especificados as variáveis de entrada, de saída e os parâmetros, juntamente com as respectivas dimensões.

Libraries: Nesta janela devem ser colocados os nomes dos arquivos externos (por exemplo, um arquivo em linguagem C) que são utilizados dentro da S-Function. Deve ser colocado também o nome da função contida neste arquivo, que será referenciada pela S-Function.

Outputs: Nesta janela devem ser colocadas as expressões que relacionam as entradas e parâmetros com as saídas. No caso de S-Functions com referência externa, devem ser escritas as funções externas com os valores que serão passados a ela.

Continuos derivatives: Devem ser colocadas as derivadas contínuas utilizadas pela função.

Discrete updates: Devem ser colocadas as atualizações discretas realizadas pela função.

Build Info: Informa se a S-Function foi gerada corretamente e se ocorrerem problemas eles são listados nesta janela.

Preenchidas todas as informações deve ser pressionada a tecla Build, para gerar a S-Function.

Nos próximos capítulos, serão listadas as S-Functions desenvolvidas neste trabalho, segundo quatro parâmetros: entrada, parâmetros, referências externas e saída. Os demais parâmetros utilizados para a construção das S-Functions, representavam valores nulos ou tiveram suas configurações iniciais do MatLab mantidas.

Neste momento, uma explicação se faz necessária. Para as funções mais simples foi possível relacionar as variáveis de saída diretamente com as de entradas, dispensando assim o uso de referências (funções em linguagem C) externas, portanto, estas funções apresentarão no campo saída apenas um traço.

A.2 S-Functions do Controle

A.2.1 Função Jota Transposta

Entradas: u0[6]

Parâmetros: -

Referências externas: -

Saídas: y0[6], y0[1], y0[2], y0[3], y0[4], y0[5]

```
y0[0] = cos(u0[4])*cos(u0[5]);
y0[1] = cos(u0[4])*sin(u0[5]);
y0[2] = -sin(u0[4]);
y0[3] = 0;
y0[4] = 0;
y0[5] = 0;
y1[0] = sin(u0[4])*sin(u0[3])*cos(u0[5])-cos(u0[3])*sin(u0[5]);
y1[1] = sin(u0[3])*sin(u0[4])*sin(u0[5])+cos(u0[3])*cos(u0[5]);
y1[2] = sin(u0[3])*cos(u0[4]);
y1[3] = 0;
y1[4] = 0;
y1[5] = 0;
y2[0] = cos(u0[3])*sin(u0[4])*cos(u0[5])+sin(u0[3])*sin(u0[5]);
y2[1] = cos(u0[3])*sin(u0[4])*sin(u0[5])-sin(u0[3])*cos(u0[5]);
y2[2] = cos(u0[3])*cos(u0[4]);
y2[3] = 0;
y2[4] = 0;
y2[5] = 0;
y3[0] = 0;
y3[1] = 0;
y3[2] = 0;
y3[3] = 1;
y3[4] = 0;
y3[5] = 0;
y4[0] = 0;
y4[1] = 0;
y4[2] = 0;
y4[3] = sin(u0[3])*tan(u0[4]);
y4[4] = cos(u0[3]);
y4[5] = sin(u0[3])*(1/cos(u0[4]));
y5[0] = 0;
y5[1] = 0;
y5[2] = 0;
y5[3] = cos(u0[3])*tan(u0[4]);
y5[4] = -sin(u0[3]);
```

```
y5[5] = cos(u0[3])*(1/cos(u0[4]));
```

A.2.2 Função Inversa da matriz B

Entradas: u0[12]

Parâmetros: n_ant[6]

Referências externas: matrizb1w.c

```
        matrizB (       v1,       v2,       v3,       v4,
v5,       v6,       tau1,       tau2,       tau3,       tau4,
tau5,       tau6,       y0[],       n_ant[6]) {

        w_;
        D;
        ro;
        a;
        b;
        c;
        n_ref;
        alfa;
        beta;
        b_[6];
        T_config[6][6];
        B[6][6];
        base[6][6];
        ident[6][6];
        inv[6][6];
        U[6];
        c1[6];
        v[6];
        cl;
        pi;
        pil;
        pj;
    i;
    j;
    k;
    sign[6];
    itmp;
    indx[6];


w_ = 0.2;
D = 0.24;
ro = 1025;

a = 0.4;
b = 0.4;
c = 0.4;

n_ref = 89/(2*3.14159);
alfa = 0.31;
```

```
beta = -0.34*(1-w_)/(D*n_ref);

v[0]=v1;
v[1]=v1;
v[2]=v2;
v[3]=v2;
v[4]=v3;
v[5]=v3;

T_config[0][0] = 1;
T_config[0][1] = 1;
T_config[0][2] = 0;
T_config[0][3] = 0;
T_config[0][4] = 0;
T_config[0][5] = 0;
T_config[1][0] = 0;
T_config[1][1] = 0;
T_config[1][2] = 1;
T_config[1][3] = 1;
T_config[1][4] = 0;
T_config[1][5] = 0;
T_config[2][0] = 0;
T_config[2][1] = 0;
T_config[2][2] = 0;
T_config[2][3] = 0;
T_config[2][4] = 1;
T_config[2][5] = 1;
T_config[3][0] = 0;
T_config[3][1] = 0;
T_config[3][2] = -b;
T_config[3][3] = b;
T_config[3][4] = 0;
T_config[3][5] = 0;
T_config[4][0] = 0;
T_config[4][1] = 0;
T_config[4][2] = 0;
T_config[4][3] = 0;
T_config[4][4] = -c;
T_config[4][5] = c;
T_config[5][0] = a;
T_config[5][1] = -a;
T_config[5][2] = 0;
T_config[5][3] = 0;
T_config[5][4] = 0;
T_config[5][5] = 0;


    (i=0;i<6;i++) {
   (n_ant[i]>=0 && v[i]>0) {
alfa = alfa;
beta = beta;
}
   (n_ant[i]<0 && v[i]<=0) {
alfa = -alfa;
beta = -beta;
}
   (n_ant[i]>0&&v[i]<0) {
alfa = alfa;
```

```
beta = -beta;
}
   (n_ant[i]<0&&v[i]>0) {
alfa = -alfa;
beta = beta;
}
   (n_ant[i]!= 0) {
       (n_ant[i]>=0)
    b_[i]=(ro*D*D*D*D)*(alfa+(beta*v[i]));
       (n_ant[i]<0)
    b_[i]=(ro*D*D*D*D)*(alfa+(beta*v[i]*(-1)));}

b_[i]=(ro*D*D*D*D)*(alfa+(beta*v[i]));
}

B[0][0] = sqrt(b_[0]*b_[0]);
B[0][1] = 0;
B[0][2] = 0;
B[0][3] = 0;
B[0][4] = 0;
B[0][5] = 0;
B[1][0] = 0;
B[1][1] = sqrt(b_[1]*b_[1]);
B[1][2] = 0;
B[1][3] = 0;
B[1][4] = 0;
B[1][5] = 0;
B[2][0] = 0;
B[2][1] = 0;
B[2][2] = sqrt(b_[2]*b_[2]);
B[2][3] = 0;
B[2][4] = 0;
B[2][5] = 0;
B[3][0] = 0;
B[3][1] = 0;
B[3][2] = 0;
B[3][3] = sqrt(b_[3]*b_[3]);
B[3][4] = 0;
B[3][5] = 0;
B[4][0] = 0;
B[4][1] = 0;
B[4][2] = 0;
B[4][3] = 0;
B[4][4] = sqrt(b_[4]*b_[4]);
B[4][5] = 0;
B[5][0] = 0;
B[5][1] = 0;
B[5][2] = 0;
B[5][3] = 0;
B[5][4] = 0;
B[5][5] = sqrt(b_[5]*b_[5]);


base[0][0] = B[0][0];
base[0][1] = B[1][1];
base[0][2] = 0;
base[0][3] = 0;
base[0][4] = 0;
```

```
base[0][5] = 0;
base[1][0] = 0;
base[1][1] = 0;
base[1][2] = B[2][2];
base[1][3] = B[3][3];
base[1][4] = 0;
base[1][5] = 0;
base[2][0] = 0;
base[2][1] = 0;
base[2][2] = 0;
base[2][3] = 0;
base[2][4] = B[4][4];
base[2][5] = B[5][5];
base[3][0] = 0;
base[3][1] = 0;
base[3][2] = -b*B[2][2];
base[3][3] = b*B[3][3];
base[3][4] = 0;
base[3][5] = 0;
base[4][0] = 0;
base[4][1] = 0;
base[4][2] = 0;
base[4][3] = 0;
base[4][4] = -c*B[4][4];
base[4][5] = c*B[5][5];
base[5][0] = a*B[0][0];
base[5][1] = -a*B[1][1];
base[5][2] = 0;
base[5][3] = 0;
base[5][4] = 0;
base[5][5] = 0;

ident[0][0] = 1;
ident[0][1] = 0;
ident[0][2] = 0;
ident[0][3] = 0;
ident[0][4] = 0;
ident[0][5] = 0;
ident[1][0] = 0;
ident[1][1] = 1;
ident[1][2] = 0;
ident[1][3] = 0;
ident[1][4] = 0;
ident[1][5] = 0;
ident[2][0] = 0;
ident[2][1] = 0;
ident[2][2] = 1;
ident[2][3] = 0;
ident[2][4] = 0;
ident[2][5] = 0;
ident[3][0] = 0;
ident[3][1] = 0;
ident[3][2] = 0;
ident[3][3] = 1;
ident[3][4] = 0;
ident[3][5] = 0;
ident[4][0] = 0;
ident[4][1] = 0;
ident[4][2] = 0;
```

```
ident[4][3] = 0;
ident[4][4] = 1;
ident[4][5] = 0;
ident[5][0] = 0;
ident[5][1] = 0;
ident[5][2] = 0;
ident[5][3] = 0;
ident[5][4] = 0;
ident[5][5] = 1;


    (i=0;i<6;i++)
    indx[i]=i;

    (i=0;i<6;i++) {
    cl=0;
        (j=0;j<6;j++)
           (fabs(base[i][j])>cl)
            cl=fabs(base[i][j]);
    cl[i]=cl;
}

    (j=0;j<5;j++) {
    pil=0;
        (i=j;i<6;i++) {
        pi = fabs(base[indx[i]][j])/cl[indx[i]];
           (pi>pil) {
            pil=pi;
            k=i;
        }
    }
    itmp=indx[j];
    indx[j]=indx[k];
    indx[k]=itmp;
        (i=j+1;i<6;i++) {
        pj=base[indx[i]][j]/base[indx[j]][j];
        base[indx[i]][j]=pj;
            (k=j+1;k<6;k++) {
            base[indx[i]][k]=base[indx[i]][k]-pj*base[indx[j]][k];
        }
    }
}

    (i=0;i<5;i++) {
        (j=i+1;j<6;j++) {
            (k=0;k<6;k++) {
            ident[indx[j]][k]=ident[indx[j]][k]-
base[indx[j]][i]*ident[indx[i]][k];
        }
    }
}

    (i=0;i<6;i++) {
    inv[5][i]=ident[indx[5]][i]/base[indx[5]][5];
        (j=4;j>=0;j=j-1) {
        inv[j][i]=ident[indx[j]][i];
            (k=j+1;k<6;k++) {
            inv[j][i]=inv[j][i]-base[indx[j]][k]*inv[k][i];
```

```
        }
         inv[j][i]=inv[j][i]/base[indx[j]][j];
    }
}



U[0]=(inv[0][0]*tau1)+(inv[0][5]*tau6);
U[1]=(inv[1][0]*tau1)+(inv[1][5]*tau6);
U[2]=(inv[2][1]*tau2)+(inv[2][3]*tau4);
U[3]=(inv[3][1]*tau2)+(inv[3][3]*tau4);
U[4]=(inv[4][2]*tau3)+(inv[4][4]*tau5);
U[5]=(inv[5][2]*tau3)+(inv[5][4]*tau5);


    (j=0;j<6;j++) {
   (U[j]<0) {
    sign[j]=-1;
}

    sign[j]=1;
y0[j]=sqrt(fabs(U[j]))*sign[j];
n_ant[j]=y0[j];
}

       0;
}
```

Saídas: y0[6]

```
matrizB
(u0[0],u0[1],u0[2],u0[3],u0[4],u0[5],u0[6],u0[7],u0[8],u0[9],u0[10],
u0[11],y0,n_ant);
```

A.3 S-Functions do Veículo

A.3.1 Função Cabo

Entradas: u0[256]

Parâmetros: vide dados.h (Apêndice A capítulo A.4)

Referências externas: cable.c

```
        cable_function(       u0[],        y0[]){
    fi,teta,psi,Ucx,Ucy,Ucz;
    X1[M+1],Y1[M+1],Z1[M+1],X2[M+1],Y2[M+1],Z2[M+1];
    at,bt,ct,dt,et,ft,gt,ht,jt;
```

```
        MR[M+1], Tx[M+1], Ty[M+1], Tz[M+1];
        Utx, Unx, Uty, Uny, Utz, Unz, MUt, MUn;
        FFx[M+1], FFy[M+1], FFz[M+1], FGx[M+1], FGy[M+1], FGz[M+1];
        FDx, FDy, FDz, FDx_, FDy_, FDz_;
        FSx[M+1], FSy[M+1], FSz[M+1];
        At_c, An_c;
        DX1, DX2, DY1, DY2, DZ1, DZ2, DP1, DP2;
        alfa1_c, alfa2_c, alfa_c, beta1_c, beta2_c, beta_c;
        mc11, mc12, mc21, mc13, mc31, mc22, mc23, mc32, mc33;
        Mcabo[9], invMcabo[9], adj[9];
        acabo[3];
        FCx, FCy, FCz;
        dX2[M+1], dY2[M+1], dZ2[M+1];
        l[M+1];
        det;
    i,k;



   (i=0;i<M;i++){
    l[i]=221.95/(M-1);
}

fi = u0[3];
teta = u0[4];
psi = u0[5];
Ucx = u0[6];
Ucy = u0[7];
Ucz = u0[8];

   (i=0;i<M;i++){
    X1[i] = u0[12+i];
    Y1[i] = u0[12+M+i];
    Z1[i] = u0[12+2* M+i];
    X2[i] = u0[12+3* M+i]-Ucx;
    Y2[i] = u0[12+4* M+i]-Ucy;
    Z2[i] = u0[12+5* M+i]-Ucz;
}

X1[M-1] = u0[0];
Y1[M-1] = u0[1];
Z1[M-1] = u0[2];

X1[0] = 0;
Y1[0]= 0;
Z1[0] = 0;
X2[0] = 0-Ucx;
Y2[0] = 0-Ucy;
Z2[0] = 0-Ucz;

X2[M-1] = u0[12+6* M + 0]-Ucx;
Y2[M-1] = u0[12+6* M + 1]-Ucy;
Z2[M-1] = u0[12+6* M + 2]-Ucz;
```

```
at = cos(psi)*cos(teta);
bt = cos(psi)*sin(teta)*sin(fi)-sin(psi)*cos(fi);
ct = cos(psi)*sin(teta)*cos(fi)+sin(psi)*sin(fi);
dt = sin(psi)*cos(teta);
et = sin(psi)*sin(teta)*sin(fi)+cos(psi)*cos(fi);
ft = sin(psi)*sin(teta)*cos(fi)-cos(psi)*sin(fi);
gt = -sin(teta);
ht = cos(teta)*sin(fi);
jt = cos(teta)*cos(fi);

   (i=0;i<M-1;i++){
    MR[i] = pow((X1[i+1]-X1[i]),2) + pow((Y1[i+1]-Y1[i]),2) +
pow((Z1[i+1]-Z1[i]),2);
    Tx[i] = (A * E / l[i])  * (X1[i+1]-X1[i])*(1-
l[i]/(pow(MR[i],0.5))) ;
    Ty[i] = (A * E / l[i]) * (Y1[i+1]-Y1[i])*(1-
l[i]/(pow(MR[i],0.5))) ;
    Tz[i] = (A * E / l[i]) * (Z1[i+1]-Z1[i])*(1-
l[i]/(pow(MR[i],0.5))) ;

    Utx = ((-X2[i])*(X1[i+1]-X1[i]) + (-Y2[i])*(Y1[i+1]-Y1[i]) +(-
Z2[i])*(Z1[i+1]-Z1[i])) * (X1[i+1]-X1[i])/MR[i];
    Unx = -X2[i]-Utx;

    Uty = ((-X2[i])*(X1[i+1]-X1[i]) + (-Y2[i])*(Y1[i+1]-Y1[i]) +(-
Z2[i])*(Z1[i+1]-Z1[i])) * (Y1[i+1]-Y1[i])/MR[i];
    Uny = -Y2[i]-Uty;

    Utz = ((-X2[i])*(X1[i+1]-X1[i]) + (-Y2[i])*(Y1[i+1]-Y1[i]) +(-
Z2[i])*(Z1[i+1]-Z1[i])) * (Z1[i+1]-Z1[i])/MR[i];
    Unz = -Z2[i]-Utz;

    MUt = pow((pow(Utx,2) + pow(Uty,2) + pow(Utz,2)),0.5);
    MUn = pow((pow(Unx,2) + pow(Uny,2) + pow(Unz,2)),0.5);

    FFx[i] = 0.5* ro* dm*(Cn* Unx* MUn+Ct* Utx* MUt)*pow(MR[i],0.5);
    FFy[i] = 0.5* ro* dm*(Cn* Uny* MUn+Ct* Uty* MUt)*pow(MR[i],0.5);
    FFz[i] = 0.5* ro* dm*(Cn* Unz* MUn+Ct* Utz* MUt)*pow(MR[i],0.5);
    FGx[i] = 0.0;
    FGy[i] = 0.0;
    FGz[i] = Wc* l[i];
}

   (i=1;i<M-1;i++){
    FDx = (X1[i+1] - X1[i])*((X2[i+1] - X2[i])*(X1[i+1] - X1[i]) +
(Y2[i+1] - Y2[i])*(Y1[i+1] - Y1[i]) + (Z2[i+1] - Z2[i])*(Z1[i+1] -
Z1[i]))/MR[i];
    FDy = (Y1[i+1] - Y1[i])*((X2[i+1] - X2[i])*(X1[i+1] - X1[i]) +
(Y2[i+1] - Y2[i])*(Y1[i+1] - Y1[i]) + (Z2[i+1] - Z2[i])*(Z1[i+1] -
Z1[i]))/MR[i];
    FDz = (Z1[i+1] - Z1[i])*((X2[i+1] - X2[i])*(X1[i+1] - X1[i]) +
(Y2[i+1] - Y2[i])*(Y1[i+1] - Y1[i]) + (Z2[i+1] - Z2[i])*(Z1[i+1] -
Z1[i]))/MR[i];
```

```
    MR[i] = pow((X1[i]-X1[i-1]),2) + pow((Y1[i]-Y1[i-1]),2) +
pow((Z1[i]-Z1[i-1]),2);

    FDx_ = (X1[i] - X1[i-1])*((X2[i] - X2[i-1])*(X1[i] - X1[i-1]) +
(Y2[i] - Y2[i-1])*(Y1[i] - Y1[i-1]) + (Z2[i] - Z2[i-1])*(Z1[i] -
Z1[i-1]))/MR[i];
    FDy_ = (Y1[i] - Y1[i-1])*((X2[i] - X2[i-1])*(X1[i] - X1[i-1]) +
(Y2[i] - Y2[i-1])*(Y1[i] - Y1[i-1]) + (Z2[i] - Z2[i-1])*(Z1[i] -
Z1[i-1]))/MR[i];
    FDz_ = (Z1[i] - Z1[i-1])*((X2[i] - X2[i-1])*(X1[i] - X1[i-1]) +
(Y2[i] - Y2[i-1])*(Y1[i] - Y1[i-1]) + (Z2[i] - Z2[i-1])*(Z1[i] -
Z1[i-1]))/MR[i];

    FSx[i] = Tx[i]-Tx[i-1]+0.5*(FFx[i]+FFx[i-1])+FGx[i] + Cdiss*(FDx
- FDx_);
    FSy[i] = Ty[i]-Ty[i-1]+0.5*(FFy[i]+FFy[i-1])+FGy[i] + Cdiss*(FDy
- FDy_);
    FSz[i] = Tz[i]-Tz[i-1]+0.5*(FFz[i]+FFz[i-1])+FGz[i] + Cdiss*(FDz
- FDz_);

    At_c = 0;
    An_c = 1;

    DX1 = X1[i]-X1[i-1];
    DX2 = X1[i+1]-X1[i];
    DY1 = Y1[i]-Y1[i-1];
    DY2 = Y1[i+1]-Y1[i];
    DZ1 = Z1[i]-Z1[i-1];
    DZ2 = Z1[i+1]-Z1[i];
    DP1 = pow((pow(DX1,2) + pow(DY1,2)),0.5);
    DP2 = pow((pow(DX2,2) + pow(DY2,2)),0.5);

       (DX1 == 0){
        alfa1_c = 0.5* M_PI;
    }
        {
        alfa1_c = atan(DY1/DX1);
    }
       (DX2 == 0){
        alfa2_c = 0.5* M_PI;
    }
        {
        alfa2_c = atan(DY2/DX2);
    }
    alfa_c = (alfa1_c+alfa2_c)/2;

       (DP1 == 0){
        beta1_c = 0.5* M_PI;
    }
        {
        beta1_c = atan(DZ1/DP1);
    }
       (DP2 == 0){
        beta2_c = 0.5* M_PI;
    }
        {
        beta2_c = atan(DZ2/DP2);
    }
```

```
    beta_c = (beta1_c+beta2_c)/2;

    mc11= A*l[i-1]* Dc+pow((cos(alfa_c)),2)*pow((cos(beta_c)),2) *
At_c+(1-pow((cos(alfa_c)*cos(beta_c)),2))* An_c;
    mc12 = (At_c-
An_c)*sin(alfa_c)*cos(alfa_c)*(pow((cos(beta_c)),2));
    mc21 = mc12;
    mc13 = (At_c-An_c)*cos(alfa_c)*sin(beta_c)*cos(beta_c);
    mc31 = mc13;
    mc22 =A* l[i-1]* Dc+pow((sin(alfa_c)),2)*pow((cos(beta_c)),2)*
At_c+(1-pow((sin(alfa_c)),2)*pow((cos(beta_c)),2))* An_c;
    mc23 = (At_c-An_c)*sin(alfa_c)*sin(beta_c)*cos(beta_c);
    mc32 = mc23;
    mc33 = A* l[i-1]* Dc+pow((sin(beta_c)),2)*
At_c+pow((cos(beta_c)),2)* An_c;

    Mcabo[0]= mc11;
    Mcabo[1]= mc12;
    Mcabo[2]= mc13;
    Mcabo[3]= mc21;
    Mcabo[4]= mc22;
    Mcabo[5]= mc23;
    Mcabo[6]= mc31;
    Mcabo[7]= mc32;
    Mcabo[8]= mc33;

    det=(Mcabo[0]* Mcabo[4]* Mcabo[8] + Mcabo[1]* Mcabo[5]* Mcabo[6]
+Mcabo[2]* Mcabo[3]* Mcabo[7])-(Mcabo[2]* Mcabo[4]* Mcabo[6] +
Mcabo[0]* Mcabo[5]* Mcabo[7] + Mcabo[1]* Mcabo[3]* Mcabo[8]);

adj[0]=(Mcabo[4]* Mcabo[8])-(Mcabo[5]* Mcabo[7]);
adj[3]=-((Mcabo[3]* Mcabo[8])-(Mcabo[5]* Mcabo[6]));
adj[6]=(Mcabo[3]* Mcabo[7])-(Mcabo[4]* Mcabo[6]);
adj[1]=-((Mcabo[1]* Mcabo[8])-(Mcabo[2]* Mcabo[7]));
adj[4]=(Mcabo[0]* Mcabo[8])-(Mcabo[2]* Mcabo[6]);
adj[7]=-((Mcabo[0]* Mcabo[7])-(Mcabo[1]* Mcabo[6]));
adj[2]=(Mcabo[1]* Mcabo[5])-(Mcabo[2]* Mcabo[4]);
adj[5]=-((Mcabo[0]* Mcabo[5])-(Mcabo[2]* Mcabo[3]));
adj[8]=(Mcabo[0]* Mcabo[4])-(Mcabo[1]* Mcabo[3]);

    (k=0;k<9;k++){
     invMcabo[k] = (1/det)* adj[k];
}

    acabo[0] = invMcabo[0]* FSx[i] + invMcabo[1]* FSy[i] +
invMcabo[2]* FSz[i];
    acabo[1] = invMcabo[3]* FSx[i] + invMcabo[4]* FSy[i] +
invMcabo[5]* FSz[i];
    acabo[2] = invMcabo[6]* FSx[i] + invMcabo[7]* FSy[i] +
invMcabo[8]* FSz[i];

    dX2[i] = acabo[0];
    dY2[i] = acabo[1];
```

```
    dZ2[i] = acabo[2];
}

FSx[M-1] = 0.5*(FFx[M-2] + FGx[M-2]) - Tx[M-2] - Cdiss* FDx;

FSy[M-1] = 0.5*(FFy[M-2] + FGy[M-2]) - Ty[M-2] - Cdiss* FDy;

FSz[M-1] = 0.5*(FFz[M-2] + FGz[M-2]) - Tz[M-2] - Cdiss* FDz;

FCx = at*(FSx[M-1]) + dt*(FSy[M-1]) + gt*(FSz[M-1]);
FCy = bt*(FSx[M-1]) + et*(FSy[M-1]) + ht*(FSz[M-1]);
FCz = ct*(FSx[M-1]) + ft*(FSy[M-1]) + jt*(FSz[M-1]);

dX2[0] = 0;
dY2[0] = 0;
dZ2[0] = 0;

dX2[M-1] = 0;
dY2[M-1] = 0;
dZ2[M-1] = 0;

    (i=0;i<M;i++){
    X2[i] = u0[12+3* M+i];
    Y2[i] = u0[12+4* M+i];
    Z2[i] = u0[12+5* M+i];
}

X2[0] = 0;
Y2[0] = 0;
Z2[0] = 0;

X2[M-1] = u0[12+6* M ];
Y2[M-1] = u0[12+6* M + 1];
Z2[M-1] = u0[12+6* M + 2];

y0[0] = FCx;
y0[1] = FCy;
y0[2] = FCz;
y0[3] = (FCz* RCy-FCy* RCz);
y0[4] = (FCx* RCz-FCz* RCx);
y0[5] = (FCy* RCx-FCx* RCy);

    (i=6;i<M+6;i++){
     y0[i] = X2[i-6];
}

    (i=M+6;i<2*M+6;i++){
     y0[i] = Y2[i-M-6];
}

    (i=2* M+6;i<3* M+6;i++){
     y0[i] = Z2[i-2* M-6];
}

    (i=3* M+6;i<4* M+6;i++){
     y0[i] = dX2[i-3* M-6];
```

```
}

   (i=4*M+6;i<5*M+6;i++){
    y0[i] = dY2[i-4*M-6];
}

   (i=5*M+6;i<6*M+6;i++){
    y0[i] = dZ2[i-5*M-6];
}

       0;


}
```

Saídas: y0[252]

```
cable_function(u0,y0);
```

A.3.2 Função do Propulsor

Entradas: u0[1], u0[2]

Parâmetros: ro, Ap, R, p, CLmax, CDmax

Referências externas: helice.c

```
       helice (       wp,       Ua,       y0[]){

       Up;
       alfa_e;
       V2;
       Lift;
       Drag;
       theta;
       T;
       Q;

Up = 0.7*Rm*wp;
  (Up==0){
alfa_e=pm-(M_PI/2);
}

alfa_e = pm - atan(Ua/Up);

V2 = Up*Up + Ua*Ua;

Lift = 0.5*ro*V2*Ap*CLmax*sin(2*alfa_e);
Drag = 0.5*ro*V2*Ap*CDmax*(1-cos(2*alfa_e));
```

```
theta = pm-alfa_e;

T = Lift*cos(theta) - Drag*sin(theta);
Q = 0.7*Rm*(Lift*sin(theta) + Drag*cos(theta));

y0[0]=Q;
y0[1]=T;
       0;
}
```

Saídas: y0[2]

```
helice (u0[0],ul[0],y0,ro[0],Ap[0],R[0],p[0],CLmax[0],CDmax[0]);
```

A.3.3 Função Sistema de Propulsão

Entradas: u0[6]

Parâmetros: a,b,c.

Referências externas: -

Saídas: y0[6]

```
y0[0] = u0[0]+u0[1];
y0[1] = u0[2]+u0[3];
y0[2] = u0[4]+u0[5];
y0[3] = b[0]*(-u0[2]+u0[3]);
y0[4] = c[0]*(-u0[4]+u0[5]);
y0[5] = a[0]*(u0[0]-u0[1]);
```

A.3.4 Função C

Entradas: u0[6]

Parâmetros: m, ro, V, xG, yG, zG, Ix, Iy, Iz, Ixy, Ixz, Iyz

Referências externas: c_function.c

```
            c_function(      u0[],       y0[],       a[]){

      mS_v0[3][3];
      mS_wxrG[3][3];
      S_IOw[3][3];
      CRB[6][6];
   i,j;
```

```
    (i=0;i<6;i++)
y0[i]=0;


mS_v0[0][0]= 0;
mS_v0[0][1]= -a[0] * u0[2];
mS_v0[0][2]= a[0] * u0[1];
mS_v0[1][0]= a[0] * u0[2];
mS_v0[1][1]=  0;
mS_v0[1][2]=-a[0] * u0[0];
mS_v0[2][0]=-a[0] * u0[1];
mS_v0[2][1]=  a[0] * u0[0];
mS_v0[2][2]= 0;


mS_wxrG[0][0]=0; mS_wxrG[0][1]=-a[0]*(u0[4]*a[5]-u0[5]*a[4]);
mS_wxrG[0][2]= a[0]*(u0[5]*a[3]-u0[3]*a[5]);
mS_wxrG[1][0]=a[0]*(u0[4]*a[5]-u0[5]*a[4]);
mS_wxrG[1][1]=0;
mS_wxrG[1][2]=-a[0]*(u0[3]*a[4]-u0[4]*a[3]);
mS_wxrG[2][0]=-a[0]*(u0[5]*a[3]-u0[3]*a[5]);
mS_wxrG[2][1]=a[0]*(u0[3]*a[4]-u0[4]*a[3]);
mS_wxrG[2][2]=0;


S_IOw[0][0]= 0; S_IOw[0][1]=-a[8]*u0[5]+a[11]*u0[4]+a[10]*u0[3];
S_IOw[0][2]=a[7]*u0[4]-a[11]*u0[5]-a[9]*u0[3] ;
S_IOw[1][0]= a[8]*u0[5]-a[11]*u0[4]-a[10]*u0[3];
S_IOw[1][1]=0;
S_IOw[1][2]=-a[6]*u0[3]+a[10]*u0[5]+a[9]*u0[4] ;
S_IOw[2][0]= -a[7]*u0[4]+a[11]*u0[5]+a[9]*u0[3];
S_IOw[2][1]= a[6]*u0[3]-a[10]*u0[5]-a[9]*u0[4];
S_IOw[2][2]=0;


CRB[0][0]=0;
CRB[0][1]=0;
CRB[0][2]=0;
CRB[0][3]=-mS_v0[0][0]-mS_wxrG[0][0];
CRB[0][4]=-mS_v0[0][1]-mS_wxrG[0][1];
CRB[0][5]=-mS_v0[0][2]-mS_wxrG[0][2];

CRB[1][0]=0;
CRB[1][1]=0;
CRB[1][2]=0;
CRB[1][3]=-mS_v0[1][0]-mS_wxrG[1][0];
CRB[1][4]=-mS_v0[1][1]-mS_wxrG[1][1];
CRB[1][5]=-mS_v0[1][2]-mS_wxrG[1][2];

CRB[2][0]=0;
CRB[2][1]=0;
CRB[2][2]=0;
CRB[2][3]=-mS_v0[2][0]-mS_wxrG[2][0];
CRB[2][4]=-mS_v0[2][1]-mS_wxrG[2][1];
CRB[2][5]=-mS_v0[2][2]-mS_wxrG[2][2];

CRB[3][0]=-mS_wxrG[0][0];
```

```
CRB[3][1]=-mS_wxrG[0][1];
CRB[3][2]=-mS_wxrG[0][2];
CRB[3][3]=-S_IOw[0][0];
CRB[3][4]=-S_IOw[0][1];
CRB[3][5]=-S_IOw[0][2];
CRB[4][0]=-mS_wxrG[1][0];
CRB[4][1]=-mS_wxrG[1][1];
CRB[4][2]=-mS_wxrG[1][2];
CRB[4][3]=-S_IOw[1][0];
CRB[4][4]=-S_IOw[1][1];
CRB[4][5]=-S_IOw[1][2];
CRB[5][0]=-mS_wxrG[2][0];
CRB[5][1]=-mS_wxrG[2][1];
CRB[5][2]=-mS_wxrG[2][2];
CRB[5][3]=-S_IOw[2][0];
CRB[5][4]=-S_IOw[2][1];
CRB[5][5]=-S_IOw[2][2];


   (i=0;i<6;i++){
       (j=0;j<6;j++){
        y0[i]=y0[i] + CRB[i][j]*u0[j];
    }
}

       0;
}
```

Saídas: y0[6]

```
c_function(u0, y0, a);
```

A.3.5 Função Db2

Entradas: u0[6]

Parâmetros: ro, Vr, Cnwx, Cnwy, Cnwz

Referências externas: D_function3.c

```
                D_function2(        u0[],        y0[],        ro,
Vr,        Cnwx,        Cnwy,        Cnwz){

        Vr1;
        Vr2;
        Vr3;
        t;
        alfa, beta, gama;
        int1[6];
        int2[6];
        a,x;
    k;
```

```
        cfxa, cfxb, cfx;
        cfx5;
        cfyb, cfyg, cfy;
        cfy5;
        cfza, cfzg, cfz;
        cfz5;
        cnx5;
        cny5;
        cnz5;



Vr1 = u0[0];
Vr2 = u0[1];
Vr3 = u0[2];

t = pow(Vr1,2)+pow(Vr2,2)+pow(Vr3,2);

  (Vr1==0){
alfa=M_PI/2;
beta=M_PI/2;
}
    {
alfa = atan2(Vr3,Vr1);

beta = atan2(Vr2,Vr1);
}

  (Vr2==0)
gama=M_PI/2;

gama = atan2(Vr3,Vr2);


cfxa = -0.0020*pow(alfa,6) +0.0010*pow(alfa,5) +0.0157*pow(alfa,4) -
0.0132*pow(alfa,3) +0.1633*pow(alfa,2) +0.0419*alfa -0.5380;

cfxb = 0.0003*pow(beta,8) -0.0002*pow(beta,7) -0.0077*pow(beta,6)
+0.0033*pow(beta,5) +0.0345*pow(beta,4) -0.0177*pow(beta,3)
+0.2233*pow(beta,2) +0.0304*beta -0.6860;

   (cfxa>cfxb)
       cfx = cfxb;

       cfx = cfxa + 0.00000000000000000001;

cfx5 = sqrt(fabs(cfxa*cfxb))*cfx/fabs(cfx);

cfyb = -0.0002*pow(beta,8) +0.0014*pow(beta,7) +0.0048*pow(beta,6) -
0.0313*pow(beta,5) -0.0325*pow(beta,4) +0.3010*pow(beta,3)
+0.0676*pow(beta,2) -1.2508*beta +0.0124;

cfyg = 0.0003*pow(gama,8) +0.0009*pow(gama,7) -0.0054*pow(gama,6) -
0.0170*pow(gama,5) +0.0098*pow(gama,4) +0.0919*pow(gama,3)
+0.3170*pow(gama,2) -0.1164*gama -0.9083;
```

```
   (cfyb>cfyg)    /* escolha do sinal */
      cfy = cfyg;

      cfy = cfyb + 0.00000000000000000001;

cfy5 = sqrt(fabs(cfyb*cfyg))*cfy/fabs(cfy);

cfza = -0.0004*pow(alfa,8) +0.0007*pow(alfa,7)  +0.0060*pow(alfa,6)
-0.0203*pow(alfa,5) -0.0053*pow(alfa,4)  +0.2705*pow(alfa,3) -
0.0943*pow(alfa,2) -1.3668*alfa -0.1363;

cfzg = 0.0007*pow(gama,8) +0.0023*pow(gama,7) -0.0134*pow(gama,6) -
0.0430*pow(gama,5) +0.0733*pow(gama,4) +0.3067*pow(gama,3) -
0.1283*pow(gama,2) -1.0334*gama -0.0434;

   (cfza>cfzg)    /* escolha do sinal */
       cfz = cfzg;

       cfz = cfza + 0.00000000000000000001;

cfz5 = sqrt(fabs(cfza*cfzg))*cfz/fabs(cfz);

cnx5 = -0.0007*pow(gama,6) +0.0001*pow(gama,5) +0.0109*pow(gama,4) -
0.0018*pow(gama,3) -0.0293*pow(gama,2) +0.0077*gama -0.0334;

cny5 = -0.0001*pow(alfa,8) -0.0002*pow(alfa,7) +0.0020*pow(alfa,6)
+0.0056*pow(alfa,5) -0.0107*pow(alfa,4) -0.0407*pow(alfa,3) -
0.0011*pow(alfa,2) +0.0541*alfa +0.0243;

cnz5 = -0.0001*pow(beta,6) -0.0025*pow(beta,5) +0.0014*pow(beta,4)
+0.0280*pow(beta,3) -0.0065*pow(beta,2) -0.0424*beta +0.0010;

int1[0]= cfx5;
int1[1]= cfy5;
int1[2]= cfz5;
int1[3]= cnx5*pow(Vr,(0.33333333333333333333));
int1[4]= cny5*pow(Vr,(0.33333333333333333333));
int1[5]= cnz5*pow(Vr,(0.33333333333333333333));

int2[0] = 0;
int2[1] = 0;
int2[2] = 0;
int2[3] = Cnwx*u0[3]*abs(u0[3]);
int2[4] = Cnwy*u0[4]*abs(u0[4]);
int2[5] = u0[5]*abs(u0[5])*Cnwz;
a = 0.5*ro*t*pow(Vr,(0.66666666666666666666));
x = 0.5*ro*pow(Vr,(1.66666666666666666666));

   (k=0; k<6; k++){
    y0[k]=-((a*int1[k]) + (x*int2[k]));
}

       0;
}
```

Saídas: y0[6]

```
D_function2(u0,y0,ro[0],Vr[0],Cnwx[0],Cnwy[0],Cnwz[0]);
```

A.3.6 Função Gr4

Entradas: u0[6]

Parâmetros: W, B, xG, yG, zG, xB, yB, zB.

Referências externas: -

Saídas: y0[6]

```
y0[0] = (W[0]-B[0])*sin(u0[4]);
y0[1] = -(W[0]-B[0])*cos(u0[4])*sin(u0[3]);
y0[2] = -(W[0]-B[0])*cos(u0[4])*cos(u0[3]);
y0[3] =-(yG[0]*W[0]-yB[0]*B[0])*cos(u0[4])*cos(u0[3])+(zG[0]*W[0]-
zB[0]*B[0])*cos(u0[4])*sin(u0[3]);
y0[4] =(zG[0]*W[0]-zB[0]*B[0])*sin(u0[4])+(xG[0]*W[0]-
xB[0]*B[0])*cos(u0[4])*cos(u0[3]);
y0[5] =-(xG[0]*W[0]-xB[0]*B[0])*cos(u0[4])*sin(u0[3])-(yG[0]*W[0]-
yB[0]*B[0])*sin(u0[4]);
```

A.3.7 Função Ad2

Entradas: u0[6]

Parâmetros: Xud, Yvd, Zwd, Kpd, Mqd, Nrd

Referências externas: -

Saídas: y0[6]

```
y0[0]=-(Zwd[0]*u0[2]*u0[4] - Yvd[0]*u0[1]*u0[5]);
y0[1]=-(-Zwd[0]*u0[2]*u0[3] + Xud[0]*u0[0]*u0[5]);
y0[2]=-(Yvd[0]*u0[1]*u0[3] - Xud[0]*u0[0]*u0[4]);
y0[3]=-(-(Yvd[0] - Zwd[0])*u0[1]*u0[2] - (Mqd[0] -
Nrd[0])*u0[4]*u0[5]);
y0[4]=-(-(Zwd[0] - Xud[0])*u0[2]*u0[0] + (Kpd[0] -
Nrd[0])*u0[5]*u0[3]);
y0[5]=-(-(Xud[0] - Yvd[0])*u0[0]*u0[1] - (Kpd[0] -
Mqd[0])*u0[3]*u0[4]);
```

A.3.8 Função Sm2

Entradas: u0[18]

Parâmetros: ro, V, xB, yB, zB, Xud, Yvd, Zwd, Kpd, Mqd, Nrd

Referências externas: -

Saídas: y0[6]

```
y0[0]=-((ro[0]*V[0]-Xud[0])*u0[12]+(ro[0]*V[0]-Zwd[0])*u0[8]*u0[4]-
(ro[0]*V[0]-Yvd[0])*u0[7]*u0[5]);
y0[1]=-((ro[0]*V[0]-Yvd[0])*u0[13]+(ro[0]*V[0]-Xud[0])*u0[6]*u0[5]-
(ro[0]*V[0]-Zwd[0])*u0[8]*u0[3]);
y0[2]=-((ro[0]*V[0] - Zwd[0])*u0[14] + (ro[0]*V[0] -
Yvd[0])*u0[7]*u0[3] - (ro[0]*V[0] - Xud[0])*u0[6]*u0[4]);
y0[3]=-(-(zB[0]*ro[0]*V[0])*u0[13] + (yB[0]*ro[0]*V[0])*u0[14] +
ro[0]*V[0]*(yB[0]*(u0[7]*u0[3] - u0[6]*u0[4]) - zB[0]*(u0[6]*u0[5] -
u0[8]*u0[3])) - Zwd[0]*u0[8]*u0[1] + Yvd[0]*u0[7]*u0[2]);
y0[4]=-((zB[0]*ro[0]*V[0])*u0[12] - (xB[0]*ro[0]*V[0])*u0[14] +
ro[0]*V[0]*(zB[0]*(u0[8]*u0[4] - u0[7]*u0[5]) - xB[0]*(u0[7]*u0[3] -
u0[6]*u0[4])) + Zwd[0]*u0[8]*u0[0] - Xud[0]*u0[6]*u0[2]);
y0[5]=-(-(yB[0]*ro[0]*V[0])*u0[12] + (xB[0]*ro[0]*V[0])*u0[13] +
ro[0]*V[0]*(xB[0]*(u0[6]*u0[5] - u0[8]*u0[3]) - yB[0]*(u0[8]*u0[4] -
u0[7]*u0[5])) + Xud[0]*u0[6]*u0[1] - Yvd[0]*u0[7]*u0[0]);
```

A.3.9 Função Jota2b

Entradas: u0[6]

Parâmetros: -

Referências externas: -

Saídas: y0[6], y0[1], y0[2], y0[3], y0[4], y0[5]

```
y0[0] = cos(u0[4])*cos(u0[5]);
y0[1] = cos(u0[4])*sin(u0[5]);
y0[2] = -sin(u0[4]);
y0[3] = 0;
y0[4] = 0;
y0[5] = 0;
y1[0] = sin(u0[4])*sin(u0[3])*cos(u0[5])-cos(u0[3])*sin(u0[5]);
y1[1] = sin(u0[3])*sin(u0[4])*sin(u0[5])+cos(u0[3])*cos(u0[5]);
y1[2] = sin(u0[3])*cos(u0[4]);
y1[3] = 0;
y1[4] = 0;
y1[5] = 0;
y2[0] = cos(u0[3])*sin(u0[4])*cos(u0[5])+sin(u0[3])*sin(u0[5]);
y2[1] = cos(u0[3])*sin(u0[4])*sin(u0[5])-sin(u0[3])*cos(u0[5]);
y2[2] = cos(u0[3])*cos(u0[4]);
y2[3] = 0;
y2[4] = 0;
y2[5] = 0;
y3[0] = 0;
y3[1] = 0;
y3[2] = 0;
y3[3] = 1;
y3[4] = 0;
y3[5] = 0;
y4[0] = 0;
y4[1] = 0;
```

```
y4[2] = 0;
y4[3] = sin(u0[3])*tan(u0[4]);
y4[4] = cos(u0[3]);
y4[5] = sin(u0[3])*(1/cos(u0[4]));
y5[0] = 0;
y5[1] = 0;
y5[2] = 0;
y5[3] = cos(u0[3])*tan(u0[4]);
y5[4] = -sin(u0[3]);
y5[5] = cos(u0[3])*(1/cos(u0[4]));
```

A.3.10 Função InvJ2_sfunc

Entradas: u0[6]

Parâmetros: -

Referências externas: invJ2.c

```
            invJ2_function (       u0[],        y0[],        y1[],
      y2[],        y3[],        y4[],        y5[]){

      adj[9];
      det;
      aux[9];
      J[36];
   k;
      fi, theta, psi;

fi=u0[3];
theta=u0[4];
psi=u0[5];

J[0]=cos(theta)*cos(psi);
J[1]=sin(theta)*sin(fi)*cos(psi)-cos(fi)*sin(psi);
J[2]=cos(fi)*sin(theta)*cos(psi)+sin(fi)*sin(psi);
J[3]=0;
J[4]=0;
J[5]=0;
J[6]=cos(theta)*sin(psi);
J[7]=sin(fi)*sin(theta)*sin(psi)+cos(fi)*cos(psi);
J[8]=cos(fi)*sin(theta)*sin(psi)-sin(fi)*cos(psi);
J[9]=0;
J[10]=0;
J[11]=0;
J[12]=-sin(theta);
J[13]=sin(fi)*cos(theta);
J[14]=cos(fi)*cos(theta);
J[15]=0;
J[16]=0;
J[17]=0;
J[18]=0;
J[19]=0;
```

```
J[20]=0;
J[21]=1;
J[22]=sin(fi) * tan(theta);
J[23]=cos(fi) * tan(theta);
J[24]=0;
J[25]=0;
J[26]=0;
J[27]=0;
J[28]=cos(fi);
J[29]=-sin(fi);
J[30]=0;
J[31]=0;
J[32]=0;
J[33]=0;
J[34]=sin(fi) / cos(theta);
J[35]=cos(fi) / cos(theta);

aux[0]= J[0];
aux[3]= J[1];
aux[6]= J[2];
aux[1]= J[6];
aux[4]= J[7];
aux[7]= J[8];
aux[2]= J[12];
aux[5]= J[13];
aux[8]= J[14];

    det=(aux[0]* aux[4]* aux[8] + aux[1]* aux[5]* aux[6] +aux[2]*
aux[3]* aux[7])-(aux[2]* aux[4]* aux[6] + aux[0]* aux[5]* aux[7] +
aux[1]* aux[3]* aux[8]);

adj[0]=(aux[4]* aux[8])-(aux[5]* aux[7]);
adj[1]=-((aux[3]* aux[8])-(aux[5]* aux[6]));
adj[2]=(aux[3]* aux[7])-(aux[4]* aux[6]);
adj[3]=-((aux[1]* aux[8])-(aux[2]* aux[7]));
adj[4]=(aux[0]* aux[8])-(aux[2]* aux[6]);
adj[5]=-((aux[0]* aux[7])-(aux[1]* aux[6]));
adj[6]=(aux[1]* aux[5])-(aux[2]* aux[4]);
adj[7]=-((aux[0]* aux[5])-(aux[2]* aux[3]));
adj[8]=(aux[0]* aux[4])-(aux[1]* aux[3]);

y0[0] = (1/det)* adj[0];
y1[0] = (1/det)* adj[1];
y2[0] = (1/det)* adj[2];
y0[1] = (1/det)* adj[3];
y1[1] = (1/det)* adj[4];
y2[1] = (1/det)* adj[5];
y0[2] = (1/det)* adj[6];
y1[2] = (1/det)* adj[7];
y2[2] = (1/det)* adj[8];

aux[0]= J[21];
aux[3]= J[22];
aux[6]= J[23];
aux[1]= J[27];
aux[4]= J[28];
aux[7]= J[29];
```

```
aux[2]= J[33];
aux[5]= J[34];
aux[8]= J[35];

    det=(aux[0]* aux[4]* aux[8] + aux[1]* aux[5]* aux[6] +aux[2]*
aux[3]* aux[7])-(aux[2]* aux[4]* aux[6] + aux[0]* aux[5]* aux[7] +
aux[1]* aux[3]* aux[8]);

adj[0]=(aux[4]* aux[8])-(aux[5]* aux[7]);
adj[1]=-((aux[3]* aux[8])-(aux[5]* aux[6]));
adj[2]=(aux[3]* aux[7])-(aux[4]* aux[6]);
adj[3]=-((aux[1]* aux[8])-(aux[2]* aux[7]));
adj[4]=(aux[0]* aux[8])-(aux[2]* aux[6]);
adj[5]=-((aux[0]* aux[7])-(aux[1]* aux[6]));
adj[6]=(aux[1]* aux[5])-(aux[2]* aux[4]);
adj[7]=-((aux[0]* aux[5])-(aux[2]* aux[3]));
adj[8]=(aux[0]* aux[4])-(aux[1]* aux[3]);

y3[3] = (1/det)* adj[0];
y4[3] = (1/det)* adj[1];
y5[3] = (1/det)* adj[2];
y3[4] = (1/det)* adj[3];
y4[4] = (1/det)* adj[4];
y5[4] = (1/det)* adj[5];
y3[3] = (1/det)* adj[6];
y4[5] = (1/det)* adj[7];
y5[5] = (1/det)* adj[8];

y3[0] = 0;
y4[0] = 0;
y5[0] = 0;
y3[1] = 0;
y4[1] = 0;
y5[1] = 0;
y3[2] = 0;
y4[2] = 0;
y5[2] = 0;
y0[3] = 0;
y1[3] = 0;
y2[3] = 0;
y0[4] = 0;
y1[4] = 0;
y2[4] = 0;
y0[5] = 0;
y1[5] = 0;
y2[5] = 0;

        0;
}
```

Saídas: y0[6], y0[1], y0[2], y0[3], y0[4], y0[5]

```
invJ2_function(u0,y0,y1,y2,y3,y4,y5);
```

A.4 dados.h

Esta função contém os parâmetros utilizados na função cabo

A.5 Condições iniciais

Para a inicialização das variáveis com as condições inicias, foram criados quatro arquivos, que devem ser executados antes do modelo, para que todas as variáveis sejam "carregadas" no workspace.

A.5.1 Arquivo dados4.m

```
       ro Ap R p CLmax CDmax;

Ra = 1.7;
Imax = 12.6;
B = (60/(2*pi))*1.5e-5;

J = 1.0e-2;
Kemf = (60/(2*pi))*0.1086;
Kt = 1.27;

CLmax = 0.542;
CDmax = 1.25;
gama = 2;
dBeta = 1.86;

p = 22.5;

p = p*pi/180;
ro = 1025;
R = 0.24/2;
Ap = pi*(0.26/2)^2;
```

```
L = 0.1271;

K1 = inv(J)*(Kt*Kemf/Ra + B);
K2 = inv(J)*Kt/Ra;
Kh = inv(J);
K3 = ro*Ap*gama*L;
K4 = ro*Ap*dBeta;
```

A.5.2 Arquivo modelo.m

```
        g ro;
g = 9.81;
ro = 1025;

        xG yG zG xB yB zB;
xG = -0.013;
yG = 0;
zG = 0.006;
xB = -0.013;
yB = 0;
zB = -0.094;

        m V Vr Ix Iy Iz Ixy Ixz Iyz Xud Yvd Zwd Kpd Mqd Nrd;
m   = 200;
V = 0.2;
Vr = 0.378;
        W B;
W = m*g;
B = ro*g*V;

Ix  = 12.3;
Iy  = 17.7;
Iz  = 19.5;
Ixy = -0.2;
Ixz = -0.9;
Iyz =  0;
Xud = -30;
Yvd = -80;
```

```
Zwd = -80;
Kpd = -15;
Mqd = -30;
Nrd = -30;
      Cnwx Cnwy Cnwz;
Cnwx = -0.16;
Cnwy = -0.37;
Cnwz = -0.32;


      M Cn Ct RCx RCy RCz Cdiss dm A Wc E Dc;
M = 41;
Cn = 1.2;
Ct = 0.01;
RCx = 0.0;
RCy = 0.0;
RCz = 0.0;
Cdiss = 100;

dm = 0.03;
A = pi*(dm^2)/4;
Wc = 1.87;
E = 1.372E10;
Dc = 1.2947e+003;



      n_ant
n_ant = zeros(6,1);



I0    = [  Ix   -Ixy  -Ixz
          -Ixy   Iy   -Iyz
          -Ixz  -Iyz   Iz  ];

mS_rG = [  0    -m*zG   m*yG
          m*zG    0    -m*xG
         -m*yG   m*xG    0   ];

mI    = [ m 0 0
          0 m 0
          0 0 m ];

MRB   = [   mI   -mS_rG
          mS_rG    I0   ];
```

```
MA  = -[Xud  0    0    0    0    0
        0   Yvd   0    0    0    0
        0    0   Zwd   0    0    0
        0    0    0   Kpd   0    0
        0    0    0    0   Mqd   0
        0    0    0    0    0   Nrd ];
M_ = MA + MRB;
```

A.5.3 Arquivo init.m

```
       M l;

cl = 0;
phi1 = 30*pi/180;
    i = 1:M
    l(i) = 221.95/(M-1);


y0=zeros(1,6*M);

cl = l(1);

    i = 2:M
    phi1 = phi1 + 2.8e-2;
    y0(i) = y0(i-1)-cl*cos(phi1);
    y0(M+i) = 0.0;
    y0(2*(M)+i) = y0(2*M+i-1)+cl*sin(phi1);
    y0(3*(M)+i) = 0.0;
    y0(4*(M)+i) = 0.0;
    y0(5*(M)+i) = 0.0;
```

```
soma = 0;
    i=1:1:M-1
    soma = soma + ((y0(i+1)-y0(i))^2 + (y0(M+i+1)-y0(M+i))^2 +
(y0(2*M+i+1)-y0(2*M+i))^2)^0.5;
```

A.5.4 Arquivo pid.m

```
close
ta = 8;
zeta = 0.9;
wn = 4/(ta*zeta);
difer = 1.3;
alpha = 5;

Xu = 0;
Yv = 0;
Zw = 0;
Kp = 0;
Mq = 0;
Nr = 0;


Xuu = -139.2241;
Yvv = -221.6038;
Zww = -149.3293;


Kpp = -16.2;
Mqq = -37.48;
Nrr = -32.41;

m = 200;

Xud = -30*difer;
uref = 0.5;

ud = ss( (Xu+2*Xuu*uref)/(m-Xud), 1/(m-Xud) ,1,0);
[num_ud,den_ud] = ss2tf( (Xu+2*Xuu*uref)/(m-Xud), 1/(m-Xud) ,1,0);

A_hat = [ud.a 0;1 0];
B_hat = [ud.b;0];
P = roots([1 2*zeta*wn wn^2]);


Kd_u = ((alpha*zeta*wn + 2*zeta*wn) - ud.a)/ud.b
Kp_u = (wn^2 + 2*alpha*(zeta*wn)^2)/(ud.b*Kd_u)
Ki_u = (alpha*zeta*wn^3)/(ud.b*Kd_u)

roots([1 (alpha*zeta*wn + 2*zeta*wn) (2*alpha*(zeta*wn)^2 + wn^2)
alpha*zeta*wn^3])
roots([1 (ud.a + Kd_u*ud.b) (Kd_u*ud.b*Kp_u) (Kd_u*ud.b*Ki_u)])


Yvd = -80*difer;
vref = 0.5;

vd = ss( (Yv+2*Yvv*vref)/(m-Yvd), 1/(m-Yvd) ,1,0);
[num_vd,den_vd] = ss2tf( (Yv+2*Yvv*vref)/(m-Yvd), 1/(m-Yvd) ,1,0);
```

```
A_hat = [vd.a 0;1 0];
B_hat = [vd.b;0];
P = roots([1 2*zeta*wn wn^2]);


Kd_v = ((alpha*zeta*wn + 2*zeta*wn) - vd.a)/vd.b;
Kp_v = (wn^2 + 2*alpha*(zeta*wn)^2)/(vd.b*Kd_v);
Ki_v = (alpha*zeta*wn^3)/(vd.b*Kd_v);


Zwd = -80*difer;
wref = 0.3;

wd = ss( (Zw+2*Zww*wref)/(m-Zwd), 1/(m-Zwd) ,1,0);
[num_wd,den_wd] = ss2tf( (Zw+2*Zww*wref)/(m-Zwd), 1/(m-Zwd) ,1,0);
A_hat = [wd.a 0;1 0];
B_hat = [wd.b;0];
P = roots([1 2*zeta*wn wn^2]);


Kd_w = ((alpha*zeta*wn + 2*zeta*wn) - wd.a)/wd.b;
Kp_w = (wn^2 + 2*alpha*(zeta*wn)^2)/(wd.b*Kd_w);
Ki_w = (alpha*zeta*wn^3)/(wd.b*Kd_w);


Kpd = -15*difer;

Ix  = 12.3;
pref = 0.1;

pd = ss( (Kp+2*Kpp*pref)/(Ix-Kpd), 1/(Ix-Kpd) ,1,0);


A_hat = [pd.a 0;1 0];
B_hat = [pd.b;0];
P = roots([1 2*zeta*wn wn^2]);

Kd_p = ((alpha*zeta*wn + 2*zeta*wn) - pd.a)/pd.b;
Kp_p = (wn^2 + 2*alpha*(zeta*wn)^2)/(pd.b*Kd_p);
Ki_p = (alpha*zeta*wn^3)/(pd.b*Kd_p);


Mqd = -30*difer;

Iy  = 17.7;
qref = 0.1;

qd = ss( (Mq+2*Mqq*qref)/(Iy-Mqd), 1/(Iy-Mqd) ,1,0);


A_hat = [qd.a 0;1 0];
B_hat = [qd.b;0];
P = roots([1 2*zeta*wn wn^2]);

Kd_q = ((alpha*zeta*wn + 2*zeta*wn) - qd.a)/qd.b;
Kp_q = (wn^2 + 2*alpha*(zeta*wn)^2)/(qd.b*Kd_q);
Ki_q = (alpha*zeta*wn^3)/(qd.b*Kd_q);
```

```
Nrd = -30*difer;

Iz  = 19.5;
rref = 0.1;

rd = ss( (Nr+2*Nrr*rref)/(Iz-Nrd), 1/(Iz-Nrd) ,1,0);
[num_rd,den_rd] = ss2tf( (Nr+2*Nrr*rref)/(Iz-Nrd), 1/(Iz-Nrd) ,1,0);
A_hat = [rd.a 0;1 0];
B_hat = [rd.b;0];
P = roots([1 2*zeta*wn wn^2]);


Kd_r = ((alpha*zeta*wn + 2*zeta*wn) - rd.a)/rd.b;
Kp_r = (wn^2 + 2*alpha*(zeta*wn)^2)/(rd.b*Kd_r);
Ki_r = (alpha*zeta*wn^3)/(rd.b*Kd_r);
```