MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

ALGORITMOS PARA INTEGRAÇÃO EM SISTEMAS INERCIAIS SOLIDÁRIOS (STRAPDOWN)

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA (PIBIC/CNPq/INPE)

Bruno Mohallem Paiva (UNIFEI, Itajubá, MG, Bolsista PIBIC/CNPq)
E-mail: bmohallem2@hotmail.com

Prof. Dr. Antônio Félix Martins Neto (DMC/INPE, São José dos Campos, SP, Orientador)

COLABORADOR
Eng. M.C. Henrique Mohallem Paiva (doutorando, ITA, São José dos Campos, SP)

Maio de 2004

AGRADECIMENTOS

Gostaria de agradecer ao meu orientador, Prof. Martins Neto, pelo grande suporte dado durante o desenvolvimento deste trabalho e pela disposição em sempre me ajudar. Gostaria de agradecer também ao meu irmão Henrique, pelo auxílio no entendimento da teoria geral dos sistemas de navegação inercial e pelo fornecimento de material extra para pesquisa.

SUMÁRIO

I. Introdução 02
II. Objetivos 04
III. Nomenclatura 05
IV. Algoritmos utilizados 06
IV.I. Algoritmo I 06
IV.I.I. Introdução 06
IV.I.II. Equacionamento 06
IV.II. Algoritmo II 08
IV.II.I. Introdução 08
IV.II.II. Equacionamento 08
IV.III. Algoritmo III 09
IV.III.I. Introdução 09
IV.III.II. Equacionamento 09
V. Implementações dos algoritmos 10
V.I. Implementação em MatLab 10
V.II. Implementações em C++ 12
V.II.I. Implementação do Algoritmo I 17
V.II.II. Implementação do Algoritmo II 19
V.II.III. Implementação do Algoritmo III 22
VI. Análise dos erros dos algoritmos 24
VII. Conclusões e trabalhos futuros 34
VIII. Referências Bibliográficas 35

I. Introdução

A navegação inercial consiste essencialmente em utilizar, a bordo de um veículo, as sensações devidas ao movimento absoluto deste veículo, levando em conta a existência do campo de gravidade local [4]. Os sensores inerciais comumente utilizados são acelerômetros, que medem as acelerações específicas, e girômetros, que medem velocidade angular (girômetros de saída analógica) ou incrementos angulares (girômetros de saída digital) [7].

As medidas resultantes permitem estimar [4]:

a) a posição, a velocidade e a aceleração do veículo em um referencial conhecido;

b) a atitude, a velocidade angular e a aceleração angular do veículo em um referencial conhecido.

A determinação de atitude de um veículo, através de um procedimento inercial, pode ser efetuada de duas formas [4].:

a) estabilizando uma plataforma inercial na qual estão fixos giroscópios (ou girômetros) e medindo a atitude relativa entre o veículo e a plataforma;

b) fixando diretamente um bloco giroscópico (ou girométrico) sobre a estrutura do veículo, e processando os sinais fornecidos por essa aparelhagem. Os sistemas desse tipo são chamados de sistemas solidários ou "strapdown".

Sistemas "strapdown" apresentam a vantagem de uma maior simplicidade mecânica e oferecem certas facilidades para organizar a redundância do sistema de medida. Por outro lado, apresentam certas desvantagens, como a necessidade de cálculos mais complexos via software, realizados a uma freqüência mais elevada [4]. Contudo, o desenvolvimento computacional das últimas décadas viabilizou a utilização dessa alternativa, fazendo com que essa seja a opção mais adotada atualmente.

Em sistemas "strapdown", o conhecimento da atitude, que é determinada a partir das medidas giométricas, permite transformar medidas de força específica de acelerômetros fixos ao veículo para o sistema da horizontal local [7]. Nesta configuração, os algoritmos utilizados para determinação de atitude são responsáveis por grande parte da acurácia da navegação [8].

Para esse sistema de navegação, vários sistemas de coordenadas podem ser utilizados. Neste estudo serão utilizados os sistemas I, L e B, definidos a seguir:

1) O sistema I é um sistema inercial (sem rotação) usado como referência para medidas de rotação angular. Neste estudo, ele estará fixo no centro da Terra;
2) O sistema L é um sistema de nível local, fixo no centro de massa do veículo. Seu eixo Z é paralelo à direção vertical (em relação à Terra) e apontado para baixo. Os eixos X e Y

são posicionados convenientemente. Neste estudo utilizaremos o sistema NED (North, East, Down), isto é, o eixo X apontando para a direção Norte, o eixo Y apontando para a direção Leste e o eixo Z apontando para baixo.

3) O sistema B está fixo no centro de massa do veículo e acompanha suas rotações.

Os sistemas B e L variam no tempo em relação ao sistema I. Contudo, neste estudo, a variação de L no tempo será desprezível. $B_{I_{(m)}}$ será definido como a orientação do sistema B em relação ao sistema inercial I no instante t_m de atualização computacional.

Os cálculos envolvidos na navegação inercial são um pouco complexos, aparecendo freqüentemente equações diferenciais não-lineares. Hughes [1] detalha os conceitos algébricos da cinemática rotacional.

Para se calcular a matriz de rotação entre o sistema B e o sistema L, é necessário conhecer o vetor de rotação Φ, que está ligado diretamente com o vetor velocidade angular ω. Uma boa aproximação para o cálculo da velocidade angular é dada por:

$$\omega = \dot{\Phi} - \frac{1-\cos|\Phi|}{|\Phi|^2}\Phi\times\dot{\Phi} + \frac{1}{|\Phi|^2}\left(1-\frac{\operatorname{sen}|\Phi|}{|\Phi|}\right)\Phi\times\left(\Phi\times\dot{\Phi}\right)$$

Integrando os dois lados e isolando o primeiro termo do membro da direita, nota-se que o vetor de rotação dado é pela integral da velocidade angular, acrescido de outros termos, que, em geral, são muito pequenos. A saída do girômetro é correspondente à integral do vetor velocidade angular. A soma dos termos restantes é conhecida como incremento de coning e, em determinados casos, é pequena o suficiente para ser desprezada. O caso mais crítico, em que a amplitude da soma dos termos adicionais é máxima, é o movimento de coning puro.

Um movimento de coning puro é dado por um vetor de rotação do seguinte tipo:

$$\Phi(t) = \begin{bmatrix} \theta\,\operatorname{sen}(\omega t) \\ \theta\cos(\omega t) \\ 0 \end{bmatrix}$$

Assim, substituindo Φ na primeira equação por um vetor nesse formato, obtemos o seguinte vetor velocidade angular:

$$\omega(t) = \begin{bmatrix} \omega\operatorname{sen}\theta\cos(\omega t) \\ \omega\operatorname{sen}\theta\,\operatorname{sen}(\omega t) \\ \omega(1-\cos\theta) \end{bmatrix}$$

Nota-se que, apesar de a rotação ter sido realizada em torno de dois eixos, aparece uma componente de velocidade angular no terceiro eixo.

Vários algoritmos de aproximação para as equações cinemáticas são propostos na literatura, cada um com suas características e comportamentos próprios e erros inerentes. Em relação à obtenção da matriz de transformação entre os sistemas B e L, os algoritmos se diferem, em geral, na forma como calculam o incremento de coning. Como o movimento de coning puro é o caso mais crítico para a determinação do incremento de coning, ele é um caso muito bom para se verificar a acurácia de um algoritmo de navegação strapdown.

II. Objetivos

Os objetivos deste trabalho são a determinação da matriz C_B^L de conversão entre o sistema B (solidário ao corpo) e o sistema L (local e invariante no tempo), através da implementação em MatLab e C++ de diferentes algoritmos propostos na literatura, e a análise e a comparação dos erros de cada algoritmo.

No plano inicial de trabalho planejava-se utilizar os cálculos envolvendo as medidas giométricas e as medidas acelerométricas. Contudo, durante o desenvolvimento do trabalho, considerou-se mais adequado realizar um estudo mais aprofundado das medidas giométricas e deixar o estudo das integrações acelerométricas [6] para um trabalho posterior.

III. Nomenclatura

A, A_1, A_2, A_3	Sistemas de coordenadas arbitrários
$C_{A_2}^{A_1}$	Matriz de cossenos diretores que transforma um vetor expresso no sistema A_2 para o seu correspondente no sistema A_1.
I	Matriz identidade
V	Vetor sem nenhum sistema de coordenadas específico
V^A	Matriz coluna com as coordenas do vetor em relação ao sistema A
$V^A\times$ ou $\{V^A\}$	Produto vetorial de V^A, representado pela matriz quadrada a seguir, onde V_{XA}, V_{YA} e V_{ZA} são as componentes de V^A

$$\begin{bmatrix} 0 & -V_{ZA} & V_{YA} \\ V_{ZA} & 0 & -V_{XA} \\ -V_{YA} & V_{XA} & 0 \end{bmatrix}$$

O produto matricial de $(V^A\times)$ com outro vetor do sistema A é igual ao produto vetorial de V^A com esse outro vetor

IV. Algoritmos utilizados

Para a determinação da matriz C_B^L, são utilizados algoritmos compostos por duas ou três velocidades distintas. Em cada algoritmo são mesclados um ou dois algoritmos rápidos, utilizados para o cálculo do incremento de coning, e um algoritmo de velocidade normal, utilizado para a atualização da matriz de cossenos diretores.

Como os algoritmos realizam a atualização de C_B^L, é necessário conhecer seu valor inicial. Nos itens a seguir serão discutidos cada algoritmo e seus equacionamentos. Na seção V serão mostradas as implementações de cada um deles.

IV.I. Algoritmo I

IV.I.I. Introdução

Para a determinação da matriz C_B^L, Savage [5] propôs um algoritmo de atualização dessa matriz composto de duas velocidades distintas.

A taxa t_1 de atualização dos incrementos de coning é tipicamente quatro vezes mais rápida que a taxa t_2 de atualização da matriz de conversão.

Serão utilizados os índices "*l*" para o ciclo computacional para o cálculo dos incrementos de coning (etapa rápida), e "m" para o ciclo de atualização de C_B^L devido ao movimento angular de B (etapa de velocidade normal). Como neste trabalho a rotação do sistema L será considerada desprezível, não serão considerados os ciclos computacionais referentes a atualização desse sistema de coordenadas.

IV.I.II. Equacionamento

Utilizando as aproximações propostas por Savage [5], as seguintes equações foram utilizadas:

→ <u>Cálculos do efeito de coning (etapa rápida)</u>:

Inicialmente, contabiliza-se o incremento $\Delta\alpha_l$ da velocidade angular integrada, medido no instante t_l.

$$\begin{aligned} &\alpha_l = \alpha_{l-1} + \Delta\alpha_l \\ &\alpha_l = 0 \text{ em } t = t_{m-1} \\ &\alpha_m = \alpha_l(t_l = t_m) \end{aligned} \qquad (1)$$

$\Delta\alpha_l$ é a medida do girômetro no instante t_l.

Com o resultado obtido, calcula-se o incremento de coning:

$$\begin{aligned}
&\Delta\beta_l = \frac{1}{2}\left(\alpha_{l-1} + \frac{1}{6}\Delta\alpha_{l-1}\right)\times\Delta\alpha_l \\
&\beta_l = \beta_{l-1} + \Delta\beta_l \\
&\beta_l = 0 \text{ em } t = t_{m-1} \\
&\beta_m = \beta_l(t_l = t_m)
\end{aligned} \qquad (2)$$

Os cálculos expressos nas equações (1) e (2) são realizados com taxa de amostragem t_1 e repetidos $(t_2/t_1 - 1)$ vezes, obtendo, assim, α_m e β_m, que serão utilizados para o cálculo do vetor de rotação do sistema B.

→ Cálculos da atitude (velocidade normal):

Inicialmente calcula-se o vetor Φ_m de rotação do sistema B:

$$\Phi_m = \alpha_m + \beta_m \qquad (3)$$

Então calcula-se a matriz $C_{B_{l(m)}}^{B_{l(m-1)}}$ de rotação do sistema B em relação a ele próprio (para a atualização da matriz de cossenos diretores):

$$C_{B_{l(m)}}^{B_{l(m-1)}} = I + \frac{\text{sen}(|\Phi_m|)}{|\Phi_m|}(\Phi_m\times) + \frac{1-\cos(|\Phi_m|)}{|\Phi_m|^2}(\Phi_m\times)(\Phi_m\times) \qquad (4)$$

Caso $\Phi_m = 0$, então $C_{B_{l(m)}}^{B_{l(m-1)}} = I$

Como se está desconsiderando os efeitos da rotação do sistema de coordenadas local, finalmente, obtém-se a atualização da matriz C_B^L de rotação de B em relação a L:

$$C_{B_{l(m)}}^{L_l} = C_{B_{l(m-1)}}^{L_l} * C_{B_{l(m)}}^{B_{l(m-1)}} \qquad (5)$$

Os cálculos acima são realizados com taxa de amostragem t_2 e repetidos indefinidamente, enquanto se desejar obter a matriz de rotação do sistema B.

IV.II. Algoritmo II

IV.II.I. Introdução

Para a determinação da matriz C_B^L, Ignagni [2] propôs vários algoritmos compostos por duas ou três velocidades distintas. Dos algoritmos propostos, foram escolhidos um exemplar de duas velocidades distintas, abordado como Algoritmo II, e outro com três velocidades distintas, abordado com Algoritmo III.

Nos algoritmos de duas velocidades, o intervalo de tempo entre um e outro cálculo da matriz de cossenos diretores é chamado de *intervalo maior*. Esse intervalo maior é subdividido em *intervalos menores*, nos quais será contabilizada a correção de coning. Serão utilizados os índices "*n*" para o ciclo computacional do intervalo maior e o índice "*m*" para o ciclo do intervalo menor.

Nos algoritmos de três velocidades, o intervalo menor é subdividido em *subintervalos menores*, onde a correção de coning será refinada.

As seguintes nomenclaturas serão utilizadas nos algoritmos II e III:

$\Delta\theta_m$	Leitura do girômetro sobre o m-ésimo intervalo menor do n-ésimo intervalo maior
θ_m	Leitura do girômetro do começo do n-ésimo intervalo maior até o fim do m-ésimo intervalo menor.
$\Delta\theta_m(\mathrm{i})$	Leitura do girômetro sobre o i-ésimo subintervalo menor
$\Delta\theta_c(n)$	Correção de coning sobre o n-ésimo intervalo maior
M	Número de intervalos menores contidos no intervalo maior

IV.II.II. Equacionamento

Dos algoritmos de duas velocidades apresentados por Ignagni [2], optou-se por implementar o Algoritmo B proposto. Baseando-se nas aproximações propostas pelo referido autor, as seguintes equações foram utilizadas:

Inicialmente calcula-se o termo $\Delta\theta_c(n)$, conhecido como correção de coning, através da equação:

$$\Delta\theta_c(\mathrm{n}) = \frac{1}{2}\sum_{m=2}^{\mathrm{M}} \theta_{m-1} \times \Delta\theta_m \qquad (6)$$

Então se junta essa contribuição de coning à leitura do girômetro (representada pela integral da velocidade angular):

$$\Delta\theta_n = \int_{n-1}^{n} \omega dt + \Delta\theta_c(n) \qquad (7)$$

Com esse termo, faz-se o cálculo recursivo da atualização da matriz de transformação:

$$C_n = C_{n-1}\left(I + \{\Delta\theta_n\} + \tfrac{1}{2}\{\Delta\theta_n\}^2\right) \qquad (8)$$

IV.III. Algoritmo III

IV.III.I. Introdução

Conforme exposto no item IV.II.I, Ignagni também propôs algoritmos compostos por três velocidades.

Nesses algoritmos, o intervalo de tempo entre um e outro cálculo da matriz de cossenos diretores é chamado de intervalo maior. Esse intervalo maior é dividido em intervalos menores, os quais, por sua vez, são divididos em subintervalos menores. Serão utilizados os índices "*n*" para o ciclo computacional do intervalo maior, o índice "*n*" para o ciclo do intervalo menor e "*i*" para o ciclo do subintervalo menor.

Foi escolhido um dos algoritmos para três velocidades propostos (Algoritmo F) [2] para ser implementado.

IV.III.II. Equacionamento

Esse algoritmo é uma variação do Algoritmo II aqui descrito, com diferença apenas na obtenção do incremento de coning. Portanto, para o Algoritmo III ainda são válidas as equações (7) e (8) apresentadas do item IV.II.II.

Neste algoritmo, o incremento de coning é colocado de forma um pouco mais precisa, acrescentando uma parcela obtida através de cálculos sobre as medidas feitas dentro do subintervalo menor. A nova expressão para obtenção do incremento de coning é dada por:

$$\Delta\theta_c(n) = \frac{1}{2}\sum_{m=2}^{M} \theta_{m-1} \times \Delta\theta_m + \frac{9}{20}\sum_{m=1}^{M}[\theta_m(1) + 3\theta_m(2)] \times \theta_m(3) \qquad (9)$$

V. Implementações dos algoritmos

No primeiro plano de trabalho, foi proposta a implementação inicial dos algoritmos em MatLab e a posterior migração para C++. O objetivo da utilização inicial de MatLab era a familiarização com os algoritmos utilizando a simplicidade de implementação matricial oferecida por MatLab. O Algoritmo I foi primeiramente implementado em MatLab e, em seguida, transcrito para C++. Durante essa primeira migração, foi implementada uma classe em C++ para operações matriciais, que permitiu a implementação dos algoritmos nessa linguagem com a mesma facilidade do MatLab. Desta forma, optou-se por implementar os algoritmos II e III apenas em C++.

Utilizou-se para os três algoritmos as seguintes constantes: ângulo de coning de 2,0rad, período de coning de 12,0s e instante inicial de simulação em 0,0s.

A geração das medidas dos girômetros foi feita de maneira analítica, simulando um ambiente de coning puro. Essas medidas podem ser descritas como "perfeitas" e não levam em consideração erros inerentes à aquisição de dados.

Como a análise dos algoritmos foi feita através dos algoritmos em C++, nessas implementações foram geradas, de forma analítica, as matrizes teóricas que deveriam ser obtidas a cada instante. A avaliação dos erros dos algoritmos foi feita comparando o resultado obtido através do algoritmo com o resultado analítico. A análise dos erros será discutida posteriormente.

V.I. Implementação em MatLab

As constantes utilizadas na implementação do Algoritmo I em MatLab, inicialmente definidas no workspace, são:

```
delta_tl = 0.025
delta_tm = 0.1
tempo_init  = 0.0
maxtempo = 1800.0
t_coning = 12.0
teta = 2.0
```

Note que o intervalo de tempo para o ciclo *m* foi fixado em 0,1s e o para o ciclo *l* em 0,025s.

O Algoritmo I foi implementado da seguinte maneira em MatLab:

```
%%%
%%% Arquivo algoritmo1.m
%%%

Measurements=gyro_measurements(tempo_init,maxtempo,delta_tl,teta,t_coning); %geraçao
das medidas do girometro
mmax=fix(maxtempo/delta_tm); %indice maximo do ciclo m
phi=zeros(4,mmax); %vetor de rotaçao a cada instante m
Cvstempo=zeros(3,3,mmax); %matriz de cossenos diretores a cada instante m
C=coningmatrix(teta,t_coning,tempo_init); %matriz de cossenos diretores inicial
for m=0.0:delta_tm:(maxtempo-delta_tm)
  alfa = zeros(3,1);
  beta = zeros(3,1);
  alfa_old = zeros(4,1);
  delalfa_old = [0;0;0];
  delalfa = [0;0;0];
  for l=(m+delta_tl):delta_tl:(m+delta_tm)
    delalfa_old = delalfa;
    delalfa = Measurements(2:4, round((l+delta_tl)/delta_tl) ); %leitura do girometro
(soma-se delta_tl para ignorar 1.a leitura)
    alfa_old = alfa;
    alfa = alfa + delalfa;
    delta_beta = cross(0.5*(alfa_old+1/6*delalfa_old),delalfa);
    beta = beta + delta_beta;
    end
  phim = alfa+beta;
  nphim = norm(phim);
  skewphim = [0 -phim(3) phim(2); phim(3) 0 -phim(1); -phim(2) phim(1) 0];
  %geraçao da matriz Cbb de atualizaçao da matriz C devido a rotacao de B
  if nphim == 0.
    Cbb = eye(3,3);
  else
    Cbb = eye(3,3) + sin(nphim)/nphim*skewphim + (1-cos(nphim))/(nphim*nphim)*
skewphim*skewphim;
    end
  C=C*Cbb; %atualizacao da matriz C
  Cvstempo(1:3,1:3,round(m/delta_tm)+1)=C;   %armazenamento da saida
  end
```

Figura 1 – Arquivo algoritmo1.m

A saída desse programa é a variável `Cvstempo`, que armazena cada matriz de cossenos diretores ao final de cada ciclo *m*.

O programa chama as funções `coning_measure()` e `coningmatriz()`, definidas a seguir:

```
function measure=coning_measure(teta,t_coning,tempo,tempo_old)
%Este programa gera o vetor de medidas correspondente ao movimento conico
teta=teta*pi/180;
w=2*pi/t_coning;
measure = [ sin(teta)*(sin(w*tempo) - sin(w*tempo_old));...
        sin(teta)*(cos(w*tempo)-cos(w*tempo_old));...
        w*(1-cos(teta))*(tempo-tempo_old) ];
```

Figura 2 – Função coning_measure()

```
function C_coning=coningmatrix(teta,t_coning,tempo)
%Este programa gera a matriz de cossenos diretores correspondente ao movimento conico
teta=teta*pi/180;
w=2*pi/t_coning;
C_coning = [cos(teta)+(1-cos(teta))*sin(w*tempo)*sin(w*tempo) .5*(1-cos(teta))*
sin(2*w*tempo) sin(teta)*cos(w*tempo);...
        .5*(1-cos(teta))*sin(2*w*tempo)   cos(teta)+(1-cos(teta))*cos(w*tempo)*
cos(w*tempo) -sin(teta)*sin(w*tempo);...
        -sin(teta)*cos(w*tempo) sin(teta)*sin(w*tempo) cos(teta)];
```

Figura 3 – Função coningmatriz()

V.II. Implementações em C++

Como mencionado anteriormente, foi criada uma classe de objetos matriz, que facilitou enormemente a implementação dos algoritmos em C++. Essa classe foi implementada no arquivo matriz.h, definido como a seguir:

```
//Arquivo matriz.h
//Arquivo de definição da classe Matriz

#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>

#ifndef MATRIZ_H
#define MATRIZ_H

//Definição da classe template Matriz
template <class Tipo>
class Matriz
{
      public:
      Tipo D[4][4];
      int NL, NC;
      Tipo t;
      Matriz (int L, int C);
      void operator = (Matriz<Tipo> &);
      Tipo norma();
      Matriz<Tipo> transposta();
      Matriz<Tipo> skew();
      Matriz<Tipo> erro(Matriz<Tipo> &);
      Tipo maxModulo();
      void toIdentidade();
      void toZero();
      friend ostream& operator << (ostream& saida, Matriz<Tipo> m);
};

//Construtor
template <class Tipo>
Matriz<Tipo>::Matriz(int L, int C)
{
      NL = L;
      NC = C;
}
```

Figura 4 – Arquivo coning.h – Parte 1/5

```
//Método que retorna a transposta da matriz
template <class Tipo>
Matriz<Tipo> Matriz<Tipo>::transposta()
{
	Matriz<Tipo> transp = Matriz<Tipo>(NC, NL);
	for (int i=0; i<NL; i++)
	for (int j=0; j<NC; j++)
	transp.D[i][j] = D[j][i];
	return transp;
}


//Porcentagem de erro entre a matriz e a referencia
template <class Tipo>
Matriz<Tipo> Matriz<Tipo>::erro(Matriz<Tipo>& Referencia)
{
	Matriz<Tipo> matrizErro = Matriz<Tipo>(0, 0);
	if (NC == Referencia.NC && NL == Referencia.NL)
	{
		matrizErro.NC=NC ; matrizErro.NL=NL;
		matrizErro.toZero();
		for (int i=0; i<NL; i++)
		for (int j=0; j<NC; j++)
			if (Referencia.D[i][j]!=0.0 && D[i][j]!=0.0)
			{
				matrizErro.D[i][j] = (D[i][j]-
				Referencia.D[i][j])/Referencia.D[i][j];
			}
	}
	return matrizErro;
}


template <class Tipo>
Tipo Matriz<Tipo>::maxModulo()
{
	Tipo maximo;
	if (NL!=0 && NC!=0)
	{
		maximo=modulo(D[0][0]);
		for (int i=0; i<NL; i++)
		for (int j=0; j<NC; j++)
			if (modulo(D[i][j])>modulo(maximo))
				maximo = modulo(D[i][j]);
	}
	else maximo=0;
	return maximo;
}


//Método que retorna a matriz correspondente ao operador
//"produto vetorial" do vetor
template <class Tipo>
Tipo Matriz<Tipo>::norma()
{
	Tipo norma=0;
	if (NL==1)
	{
		for (int j=0; j<NC; j++)
		norma=norma + D[0][j]*D[0][j];
		norma=sqrt(norma);
	}
```

Figura 4 – Arquivo coning.h – Parte 2/5

```
	else if (NC==1)
	{
		for (int i=0; i<NL; i++)
		norma=norma + D[i][0]*D[i][0];
		norma=sqrt(norma);
	}
	return norma;
}

template <class Tipo>
Matriz<Tipo> Matriz<Tipo>::skew()
{
	Matriz skew(3,3);
	if (NL==3 && NC==1)
	{
		skew.D[0][0]=0;		skew.D[0][1]=-D[2][0];	skew.D[0][2]=D[1][0];
		skew.D[1][0]=D[2][0];	skew.D[1][1]=0;		skew.D[1][2]=-D[0][0];
		skew.D[2][0]=-D[1][0];	skew.D[2][1]=D[0][0];	skew.D[2][2]=0;
	}
	else
	{
		skew.NL=0; skew.NC=0;
	}
	return skew;
}

//Método que ajusta a matriz para matriz identidade
template <class Tipo>
void Matriz<Tipo>::toIdentidade()
{
	for (int i=0; i<NL; i++)
		for (int j=0; j<NC; j++)
		{
			if (i==j)
				D[i][j] = 1.0;
			else
				D[i][j] = 0.0;
		}
}

//Método que ajusta a matriz para matriz de zeros
template <class Tipo>
void Matriz<Tipo>::toZero()
{
	for (int i=0; i<NL; i++)
	for (int j=0; j<NC; j++)
	D[i][j] = 0.0;	//Mudar, caso Tipo = int
}


//Sobrecarga do operador =
template <class Tipo>
void Matriz<Tipo>::operator = (Matriz<Tipo> &P)
{
	for (int i=0; i<NL; i++)
	for (int j=0; j<NC; j++)
	D[i][j] = P.D[i][j];
}
```

Figura 4 – Arquivo coning.h – Parte 3/5

```
//Sobrecarga do operador +
template <class Tipo>
Matriz<Tipo> operator + (Matriz<Tipo> &P, Matriz<Tipo> &Q)
{
      Matriz<Tipo> R(P.NL, P.NC);
      for (int i=0; i<P.NL; i++)
      for (int j=0; j<P.NC; j++)
      R.D[i][j] = P.D[i][j] + Q.D[i][j];
      return R;
}


//Sobrecarga do operador -
template <class Tipo>
Matriz<Tipo> operator - (Matriz<Tipo> &P, Matriz<Tipo> &Q)
{
      Matriz<Tipo> R(P.NL, P.NC);
      for (int i=0; i<P.NL; i++)
      for (int j=0; j<P.NC; j++)
      R.D[i][j] = P.D[i][j] - Q.D[i][j];
      return R;
}

//Sobrecarga do operador * (produto por escalar) - Parte I
template <class Tipo>
Matriz<Tipo> operator * (const Tipo alpha, Matriz<Tipo> &Q)
{
      Matriz<Tipo> R(Q.NL, Q.NC);
      for (int i=0; i<Q.NL; i++)
      for (int j=0; j<Q.NC; j++)
      R.D[i][j] = alpha * Q.D[i][j];
      return R;
}

//Sobrecarga do operador * (produto por escalar) - Parte II
template <class Tipo>
Matriz<Tipo> operator * (Matriz<Tipo> &Q, const Tipo alpha)
{
      return (alpha*Q);
}

//Sobrecarga do operador * (produto matricial) - Parte III
template <class Tipo>
Matriz<Tipo> operator * (Matriz<Tipo> &P, Matriz<Tipo> &Q)
{
      Matriz<Tipo> R(P.NL, Q.NC);
      if (P.NC != Q.NL)
      {
            R.NL = R.NC = 0;
            return R;
      }
      for (int i=0; i<P.NL; i++)
      for (int j=0; j<Q.NC; j++)
      {
            R.D[i][j] = 0;
            for (int k=0; k<P.NC; k++)
            R.D[i][j] += P.D[i][k]*Q.D[k][j];
      }
      return R;
}
```

Figura 4 – Arquivo coning.h – Parte 4/5

```
//Produto vetorial (válido para vetores no R3
template <class Tipo>
Matriz<Tipo> operator ^ (Matriz<Tipo> &V1, Matriz<Tipo> &V2)
{
	Matriz<Tipo> produto(0,0);
	if (V1.NL==1 && V1.NC==3 && V2.NL==1 && V2.NC==3)
	{
		produto.NL=1; produto.NC=3;
		produto.D[0][0]=V1.D[0][1]*V2.D[0][2] - V1.D[0][2]*V2.D[0][1];
		produto.D[0][1]=V1.D[0][2]*V2.D[0][0] - V1.D[0][0]*V2.D[0][2];
		produto.D[0][2]=V1.D[0][0]*V2.D[0][1] - V1.D[0][1]*V2.D[0][0];
	}

	if (V1.NL==3 && V1.NC==1 && V2.NL==3 && V2.NC==1)
	{
		produto.NL=3; produto.NC=1;
		produto.D[0][0]=V1.D[1][0]*V2.D[2][0] - V1.D[2][0]*V2.D[1][0];
		produto.D[1][0]=V1.D[2][0]*V2.D[0][0] - V1.D[0][0]*V2.D[2][0];
		produto.D[2][0]=V1.D[0][0]*V2.D[1][0] - V1.D[1][0]*V2.D[0][0];
	}
	return produto;
}


template <class Tipo>
ostream& operator << (ostream& saida, Matriz<Tipo> m)
{
	saida << "t = " << m.t << "\n";
	for (int i=0; i<m.NL; i++)
	{
		for (int j=0; j<m.NC; j++)
		{
		saida << m.D[i][j];
		if (j<m.NC-1)
			saida << "\t";
		}
		saida << "\n";
	}
	saida << "\n\n";
	return saida;
}


#endif
```

Figura 4 – Arquivo coning.h – Parte 5/5

Além dessa classe auxiliar, contida no arquivo `matriz.h`, também foi criado um arquivo coning.h, com implementação de funções auxiliares relacionadas às medidas do girômetro e à manipulação de arquivos, além de uma função para retirar "lixos" que por ventura aparecessem nas matrizes. Foi considerado lixo qualquer termo da matriz que apresentasse queda de pelo menos duas ordens de grandeza em relação ao termo corresponde da matriz obtida na iteração anterior. Esse lixo é, então substituído por zero.

Todas entradas e saídas dos algoritmos são apresentadas em duas versões de arquivo: uma versão binária (.dat), de fácil leitura por parte do software, e uma versão texto (.txt),

amigável. Os arquivos `gyro.dat` e `gyro.txt` armazenam as leituras do girômetro, que, na verdade, seriam as entradas do software. Os arquivos `matriz.dat` e `matriz.txt` armazenam as matrizes de cossenos diretores a cada instante. Os arquivos `matrizTeorica.dat` e `matrizTeorica.dat` armazenam as matrizes teóricas em cada um desses instantes, obtidas por forma analítica. Os arquivos `erro.dat` e `erro.txt` armazenam o maior erro relativo de cada matriz.

V.II.I. Implementação do Algoritmo I

O Algoritmo I foi implementado da seguinte maneira:

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>
#include "matriz.h"
#include "coning.h"

//Parâmetros fixos
#define TEMPO_INIT 0.0     //Instante de início da simulação
#define MAX_TEMPO 20.0     //Instante de término da simulação
#define DELTA_TL 0.025     //Período de amostragem T(l)
#define DELTA_TM 0.1       //Período de amostragem T(m)
#define T_CONING 12.0      //Período de coning
#define TETA 2.0           //Ângulo de coning

//Função principal
void main()
{
      //Abertura dos arquivos
      //Cada arquivo possui uma versão binária e uma versão .txt amigável
      FILE *arquivoGyroBin, *arquivoMatriz, *arquivoMatrizBin, *arquivoMatrizTeorica,
      *arquivoMatrizTeoricaBin, *arquivoErro, *arquivoErroBin;
      arquivoMatriz=fopen("matriz.txt", "wt");
      arquivoMatrizBin=fopen("matriz.dat", "wb");
      arquivoMatrizTeorica=fopen("matrizTeorica.txt", "wt");
      arquivoMatrizTeoricaBin=fopen("matrizTeorica.dat", "wb");
      arquivoErro=fopen("erro.txt", "wt");
      arquivoErroBin=fopen("erro.dat", "wb");

      Matriz<double> C(3,3); //Matriz de cossenos diretores inicial
      Matriz<double> gyroMeasure(3,1); //Vetor com as medidas do girômetro
      Matriz<double> alfa(3,1);
      Matriz<double> beta(3,1);
      Matriz<double> alfa_old(3,1);
      Matriz<double> delalfa_old(3,1);
      Matriz<double> delalfa(3,1);
      Matriz<double> delta_beta(3,1);
      Matriz<double> phim(3,1);  //Vetor de rotação
      Matriz<double> skewphim(3,3);
      Matriz<double>  Cbb(3,3);   //Matriz de atualização de C devido ao movimento
angular de B
```

Figura 5 – Arquivo algoritmo1.cpp – Parte 1/3

```
	Matriz<double> C_Referencia(3,3);
	Matriz<double> C_Anterior(3,3);
	Matriz<double> C_ReferenciaAnterior(3,3);
	Matriz<double> matrizErro(3,3);

	double nphim;
	int m, l, mmax, lmax;
	double erroAtual;
	double maiorErro;

	//Geração das leituras do girômentro e inicialização das variáveis
	gyroMeasurements((double) TEMPO_INIT, (double) MAX_TEMPO, (double) DELTA_TL,
(double) TETA, (double) T_CONING);
	arquivoGyroBin=fopen("gyro.dat", "rb");
	fread(&gyroMeasure, sizeof(Matriz<double>), 1, arquivoGyroBin); //dispensando a
leitura inicial (t=0.0)
	C=coningMatrix(TETA, T_CONING, TEMPO_INIT);
	C_Anterior=C;
	C_ReferenciaAnterior=C;
	maiorErro = 0.0;
	lmax=(int) (DELTA_TM/DELTA_TL + 0.001); //soma-se 0.001 para evitar erros de
truncamento
	mmax=(int) (MAX_TEMPO/DELTA_TM + 0.001);

	for (m=1; m<=mmax; m++)
	{
		alfa.toZero();
		beta.toZero();
		alfa_old.toZero();
		delalfa_old.toZero();
		delalfa.toZero();

		for (l=1; l<=lmax; l++)
		{
			delalfa_old = delalfa;
			alfa_old = alfa;
			fread(&gyroMeasure, sizeof(Matriz<double>), 1, arquivoGyroBin);
			delalfa=gyroMeasure;
			alfa = alfa + delalfa;
			delta_beta = (0.5*(alfa_old + (1.0/6.0)*delalfa_old))^delalfa;
			beta = beta + delta_beta;
		}

		phim=alfa+beta;
		phim.t=(m*DELTA_TM);
		nphim=phim.norma();
		skewphim=phim.skew();

		Cbb.toIdentidade();
		if (nphim!=0)
			Cbb = Cbb + (sin(nphim)/nphim)*skewphim + ( (1-cos(nphim)) /
(nphim*nphim) )*skewphim *skewphim;
		C=C*Cbb;
		C.t=m*DELTA_TM;
		C_Referencia=coningMatrix(TETA, T_CONING, m*DELTA_TM);
		C_Referencia.t=C.t;

		//Retirada do lixo residual, caso seja necessário
		if (m>0.0)
		{
			retirarLixo(C_Anterior, C);
			retirarLixo(C_ReferenciaAnterior,C_Referencia);
		}
```

Figura 5 – Arquivo algoritmo1.cpp – Parte 2/3

```
            //Determinação do erro
            matrizErro=C.erro(C_Referencia);
            erroAtual=matrizErro.maxModulo();   //determinação   do   maior   erro   (em
módulo) da matriz
            if (erroAtual > maiorErro)
                  maiorErro = erroAtual;

            C_Anterior=C;
            C_ReferenciaAnterior=C_Referencia;

            //Gravação dos arquivos
            gravarArquivoTexto(arquivoMatriz, C, m*DELTA_TM);
            gravarArquivoBinario(arquivoMatrizBin, C);
            gravarArquivoTexto(arquivoMatrizTeorica, C_Referencia, m*DELTA_TM);
            gravarArquivoBinario(arquivoMatrizTeoricaBin, C_Referencia);
            fwrite(&erroAtual, sizeof(double), 1, arquivoErroBin);
            fprintf (arquivoErro, "%g\t%g\n", m*DELTA_TM, erroAtual);
      }

      //Última matriz de conversão calculada
      cout << "Arquivos binarios gerados:" << endl << "    gyro.dat    matriz.dat
matrizTeorica.dat   erro.dat\n\n";
      cout << "Arquivos textos correspondentes:" << endl << "   gyro.txt   matriz.txt
matrizTeorica.txt   erro.txt\n\n\n";
      cout << "Ultima matriz de transformacao de coordenadas:" << endl << C;

      fcloseall();   //Fechamento dos arquivos
      getch();       //Pausa
}
```

Figura 5 – Arquivo algoritmo1.cpp – Parte 3/3

V.II.II. Implementação do Algoritmo II

O Algoritmo II foi implementado da seguinte maneira:

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>
#include "matriz.h"
#include "coning.h"

//Parâmetros fixos
#define TEMPO_INIT 0.0     //Instante de início da simulação
#define MAX_TEMPO 20.0     //Instante de término da simulação
#define T_T 0.025          //Período de amostragem T
#define T_TAU 0.1          //Período de amostragem Tau
#define T_CONING 12.0      //Período de coning
#define TETA 2.0           //Ângulo de coning
```

Figura 6 – Arquivo algoritmo2.cpp – Parte 1/3

```
//Função principal
void main()
{
	//Abertura dos arquivos
	//Cada arquivo possui uma versão binária e uma versão .txt amigável
	FILE *arquivoGyroBin, *arquivoMatriz, *arquivoMatrizBin, *arquivoMatrizTeorica,
*arquivoMatrizTeoricaBin, *arquivoErro, *arquivoErroBin;
	arquivoMatriz=fopen("matriz.txt", "wt");
	arquivoMatrizBin=fopen("matriz.dat", "wb");
	arquivoMatrizTeorica=fopen("matrizTeorica.txt", "wt");
	arquivoMatrizTeoricaBin=fopen("matrizTeorica.dat", "wb");
	arquivoErro=fopen("erro.txt", "wt");
	arquivoErroBin=fopen("erro.dat", "wb");

	Matriz<double> C(3,3); //Matriz de cossenos diretores
	Matriz<double> gyroMeasure(3,1);
	Matriz<double> deltaTheta_n(3,1);
	Matriz<double> deltaTheta_c(3,1);
	Matriz<double> deltaTheta_m(3,1);
	Matriz<double> deltaTheta_m_old(3,1);
	Matriz<double> deltaTheta_m1(3,1);
	Matriz<double> deltaTheta_m2(3,1);
	Matriz<double> deltaTheta_m3(3,1);
	Matriz<double> parcial(3,1);
	Matriz<double> theta_m(3,1);
	Matriz<double> integral_w(3,1);
	Matriz<double> skewTheta_n(3,3);
	Matriz<double> C_atualizacao(3,3);
	Matriz<double> C_referencia(3,3);
	Matriz<double> C_anterior(3,3);
	Matriz<double> C_referenciaAnterior(3,3);
	Matriz<double> matrizErro(3,3);
	double erroAtual;
	double maiorErro;
	int n, m, l, nmax, incl;

	//Geração das leituras do girômentro e inicialização das variáveis
	gyroMeasurements(TEMPO_INIT, MAX_TEMPO, T_T, TETA, T_CONING);
	arquivoGyroBin=fopen("gyro.dat", "rb");
	fread(&gyroMeasure, sizeof(Matriz<double>), 1, arquivoGyroBin); //dispensando a
leitura inicial
	C=coningMatrix(TETA, T_CONING, TEMPO_INIT);
	C_anterior=C;
	C_referenciaAnterior=C;
	maiorErro=0.0;
	nmax=(int) (MAX_TEMPO/T_TAU + 0.001); //soma-se 0.001 para evitar erros de
truncamento
	incl=(int) (T_TAU/T_T + 0.001);

	for (n=0; n<nmax; n++)
	{
		deltaTheta_c.toZero();
		theta_m.toZero();
		deltaTheta_m.toZero();
		integral_w.toZero();

		for (m=1; m<=incl; m++)
		{
			theta_m = theta_m + deltaTheta_m;
			fread(&gyroMeasure, sizeof(Matriz<double>), 1, arquivoGyroBin);
			deltaTheta_m = gyroMeasure;
			deltaTheta_c = deltaTheta_c + theta_m^deltaTheta_m;
			integral_w = integral_w + deltaTheta_m;
		}
```

Figura 6 – Arquivo algoritmo2.cpp – Parte 2/3

```
            deltaTheta_c = (0.5)*deltaTheta_c;
            deltaTheta_n = integral_w + deltaTheta_c;
            skewTheta_n = deltaTheta_n.skew();

            C_atualizacao.toIdentidade();
            C_atualizacao = C_atualizacao+skewTheta_n + 0.5*skewTheta_n*skewTheta_n;
            C=C*C_atualizacao;
            C.t= (n+1)*T_TAU;
            C_referencia=coningMatrix(TETA, T_CONING, C.t);
            C_referencia.t=C.t;

            //Retirada do lixo residual, caso seja necessário
            if (m>0.0)
            {
                  retirarLixo(C_anterior, C);
                  retirarLixo(C_referenciaAnterior,C_referencia);
            }

            //Determinação do erro
            matrizErro=C.erro(C_referencia);
            erroAtual=matrizErro.maxModulo();
            if (erroAtual>maiorErro)
            maiorErro=erroAtual;

            C_anterior=C;
            C_referenciaAnterior=C_referencia;

            //Gravação dos arquivos
            gravarArquivoTexto(arquivoMatriz, C, C.t);
            gravarArquivoBinario(arquivoMatrizBin, C);
            gravarArquivoTexto(arquivoMatrizTeorica, C_referencia, (n+1)*T_TAU);
            gravarArquivoBinario(arquivoMatrizTeoricaBin, C_referencia);
            fwrite(&erroAtual, sizeof(double), 1, arquivoErroBin);
            fprintf (arquivoErro, "%g\t%g\n", (n+1)*T_TAU, erroAtual);
      }

      //Última matriz de conversão calculada
      cout << "Arquivos binarios gerados:" << endl << "     gyro.dat     matriz.dat
matrizTeorica.dat    erro.dat\n\n";
      cout << "Arquivos textos correspondentes:" << endl << "   gyro.txt    matriz.txt
matrizTeorica.txt    erro.txt\n\n\n";
      cout << "Ultima matriz de transformacao de coordenadas:" << endl << C;

      fcloseall();    //Fechamento dos arquivos
      getch();        //Pausa
}
```

Figura 6 – Arquivo algoritmo2.cpp – Parte 3/3

V.II.III. Implementação do Algoritmo III

O Algoritmo III foi implementado da seguinte maneira:

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>
#include "matriz.h"
#include "coning.h"

//Parâmetros fixos
#define TEMPO_INIT 0.0     //Instante de início da simulação
#define MAX_TEMPO 20.0     //Instante de término da simulação
#define T_T 0.025          //Período de amostragem T
#define T_TAU 0.1          //Período de amostragem Tau
#define T_CONING 12.0      //Período de coning
#define TETA 2.0           //Ângulo de coning

//Função principal
void main()
{
       //Abertura dos arquivos
       //Cada arquivo possui uma versão binária e uma versão .txt amigável
       FILE *arquivoGyroBin, *arquivoMatriz, *arquivoMatrizBin, *arquivoMatrizTeorica,
*arquivoMatrizTeoricaBin, *arquivoErro, *arquivoErroBin;
       arquivoMatriz=fopen("matriz.txt", "wt");
       arquivoMatrizBin=fopen("matriz.dat", "wb");
       arquivoMatrizTeorica=fopen("matrizTeorica.txt", "wt");
       arquivoMatrizTeoricaBin=fopen("matrizTeorica.dat", "wb");
       arquivoErro=fopen("erro.txt", "wt");
       arquivoErroBin=fopen("erro.dat", "wb");

       Matriz<double> C(3,3); //Matriz de cossenos diretores
       Matriz<double> gyroMeasure(3,1);
       Matriz<double> deltaTheta_n(3,1);
       Matriz<double> deltaTheta_c(3,1);
       Matriz<double> deltaTheta_m(3,1);
       Matriz<double> deltaTheta_m_old(3,1);
       Matriz<double> deltaTheta_m1(3,1);
       Matriz<double> deltaTheta_m2(3,1);
       Matriz<double> deltaTheta_m3(3,1);
       Matriz<double> parcial(3,1);
       Matriz<double> theta_m(3,1);
       Matriz<double> integral_w(3,1);
       Matriz<double> skewTheta_n(3,3);
       Matriz<double> C_atualizacao(3,3);
       Matriz<double> C_referencia(3,3);
       Matriz<double> C_anterior(3,3);
       Matriz<double> C_referenciaAnterior(3,3);
       Matriz<double> matrizErro(3,3);

       double erroAtual;
       double maiorErro;
       int n, m, nmax, inc1;

       //Geração das leituras do girômentro e inicialização das variáveis
       C_anterior=C;
       gyroMeasurements(TEMPO_INIT, MAX_TEMPO, (T_T/3.0), TETA, T_CONING);
       arquivoGyroBin=fopen("gyro.dat", "rb");
       fread(&gyroMeasure, sizeof(Matriz<double>), 1, arquivoGyroBin); //dispensando a
leitura inicial
```

Figura 7 – Arquivo algoritmo3.cpp – Parte 1/3

```
	C=coningMatrix(TETA, T_CONING, TEMPO_INIT);
	C_anterior=C;
	C_referenciaAnterior=C;
	maiorErro=0.0;
	nmax=(int) (MAX_TEMPO/T_TAU + 0.001); //soma-se 0.001 para evitar erros de
truncamento
	incl=(int) (T_TAU/T_T + 0.001);

	for (n=0; n<nmax; n++)
	{
		deltaTheta_c.toZero();
		theta_m.toZero();
		deltaTheta_m.toZero();
		integral_w.toZero();
		parcial.toZero();
		for (m=1; m<=incl; m++)
		{
			fread(&gyroMeasure, sizeof(Matriz<double>), 1, arquivoGyroBin);
			deltaTheta_m1=gyroMeasure;
			fread(&gyroMeasure, sizeof(Matriz<double>), 1, arquivoGyroBin);
			deltaTheta_m2=gyroMeasure;
			fread(&gyroMeasure, sizeof(Matriz<double>), 1, arquivoGyroBin);
			deltaTheta_m3=gyroMeasure;

			deltaTheta_m = deltaTheta_m1 + deltaTheta_m2 + deltaTheta_m3;
			parcial=parcial +(deltaTheta_m1+ 3.0*deltaTheta_m2)^deltaTheta_m3;

			if (m>1)
				deltaTheta_c = deltaTheta_c + theta_m^deltaTheta_m;

			theta_m = theta_m + deltaTheta_m;
			integral_w = integral_w + deltaTheta_m;
		}

		deltaTheta_c = (0.5)*deltaTheta_c + (9.0/20.0)*parcial;
		deltaTheta_n = integral_w + deltaTheta_c;
		skewTheta_n = deltaTheta_n.skew();

		C_atualizacao.toIdentidade();
		C_atualizacao = C_atualizacao +skewTheta_n + 0.5*skewTheta_n*skewTheta_n;
		C=C*C_atualizacao;
		C.t= (n+1)*T_TAU;
		C_referencia=coningMatrix(TETA, T_CONING, C.t);
		C_referencia.t=C.t;

		//Retirada do lixo residual, caso seja necessário
		if (m>0.0)
		{
			retirarLixo(C_anterior, C);
			retirarLixo(C_referenciaAnterior,C_referencia);
		}

		//Determinação do erro
		matrizErro=C.erro(C_referencia);
		erroAtual=matrizErro.maxModulo();
		if (erroAtual>maiorErro)
		maiorErro=erroAtual;

		C_anterior=C;
		C_referenciaAnterior=C_referencia;
```

Figura 7 – Arquivo algoritmo3.cpp – Parte 2/3

```
            //Gravação dos arquivos
            gravarArquivoTexto(arquivoMatriz, C, C.t);
            gravarArquivoBinario(arquivoMatrizBin, C);
            gravarArquivoTexto(arquivoMatrizTeorica, C_referencia, (n+1)*T_TAU);
            gravarArquivoBinario(arquivoMatrizTeoricaBin, C_referencia);
            fwrite(&erroAtual, sizeof(double), 1, arquivoErroBin);
            fprintf (arquivoErro, "%g\t%g\n", (n+1)*T_TAU, erroAtual);
      }

      //Última matriz de conversão calculada
      cout << "Arquivos binarios gerados:" << endl << "    gyro.dat    matriz.dat
matrizTeorica.dat   erro.dat\n\n";
      cout << "Arquivos textos correspondentes:" << endl << "   gyro.txt   matriz.txt
matrizTeorica.txt   erro.txt\n\n\n";
      cout << "Ultima matriz de transformacao de coordenadas:" << endl << C;

      fcloseall();   //Fechamento dos arquivos
      getch();       //Pausa
}
```

Figura 7 – Arquivo algoritmo3.cpp – Parte 3/3

VI. Análise dos erros dos algoritmos

Todas as análises dos três algoritmos foram feitas simulando um ambiente de coning puro e foram utilizados os seguintes parâmetros: ângulo de coning de 2,0rad, período de coning de 12,0s e instante inicial de 0,0s.

Para avaliação de erros, utilizou-se uma comparação, termo a termo, entre a matriz obtida em cada instante e a matriz obtida analiticamente nesse instante, obtendo, assim, uma matriz de erros correspondente. Para melhor análise, o erro foi normalizado ([matriz obtida – matriz teórica] / matriz teórica). O erro de cada instante é dado pelo módulo do termo de maior módulo da matriz de erros desse instante. Note que a maioria dos gráficos apresenta escala logarítmica (no eixo das abscissas, no eixo das ordenadas ou em ambos). Quando lê-se "Erro relativo (logaritmico)" nos gráficos, entenda-se que o erro relativo está sendo apresentado em escala logarítmica.

Nos algoritmos, foi empregado um filtro de "retirada de lixo". Para mostrar a atuação desse filtro, inicialmente foram feitas duas simulações de 60,0s com Algoritmo I, uma sem o filtro e outra com a utilização desse recurso. Os seguintes gráficos foram encontrados:

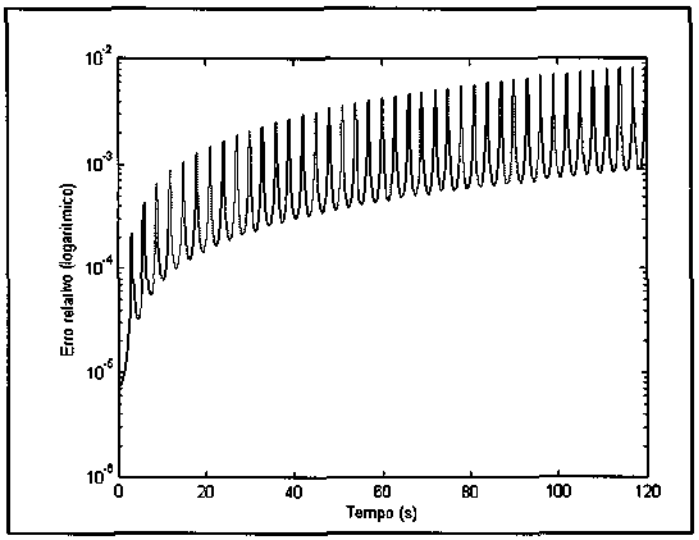

Figura 8.a – Comportamento do erro no Algoritmo I sem o filtro

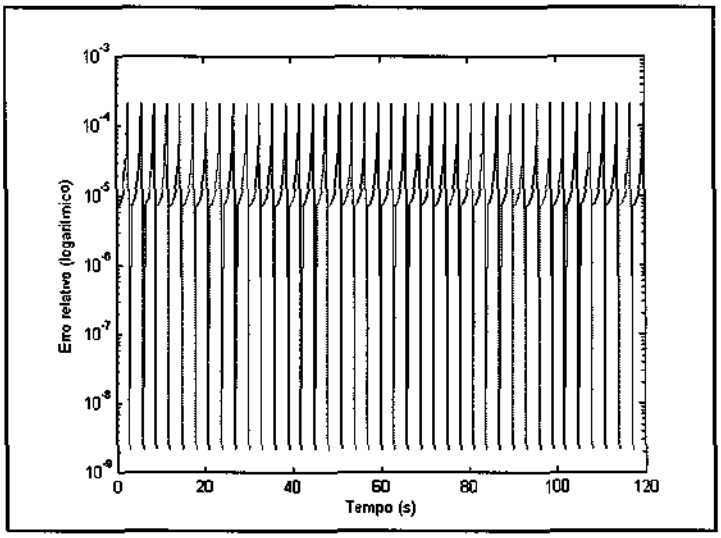

Figura 8.b – Comportamento do erro no Algoritmo I com o filtro

Nota-se um comportamento oscilatório associado ao erro. Contudo, sem a adição do filtro, o erro apresentou comportamento cumulativo considerável. Já com o filtro, o erro permanece basicamente puramente oscilatório. Assim, o filtro foi utilizado em todas as simulações seguintes. Para condições diferentes das simuladas neste trabalho, seria conveniente estudar a aplicabilidade do filtro.

Dando seqüência à analise, foi feita uma simulação com tempo máximo de 20,0s, para avaliar o erro em cada instante, observando assim o comportamento dos algoritmos. Para os Algoritmos I, II e III foram utilizados o período de 0,1s para os intervalo maior e 0,025s para o intervalo menor. No Algoritmo III o intervalo menor foi subdividido em três, gerando os subintervalos menores. Foram encontrados os gráficos apresentados na página seguinte.

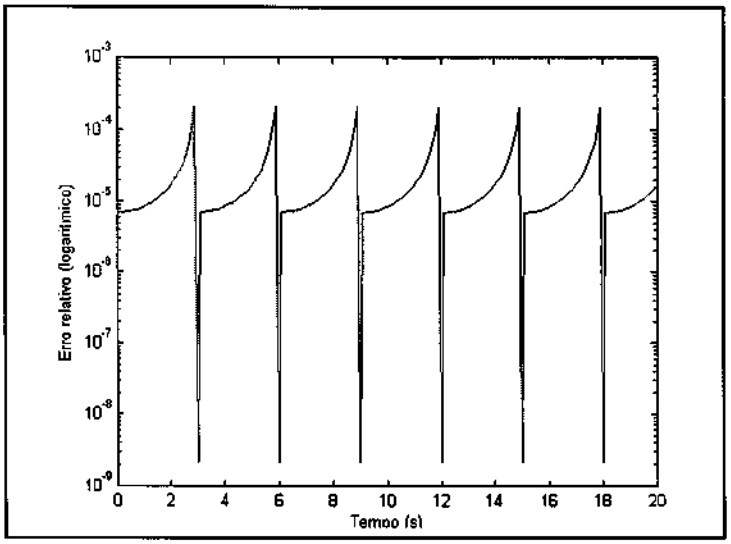

Figura 9.a – Comportamento do erro no Algoritmo I

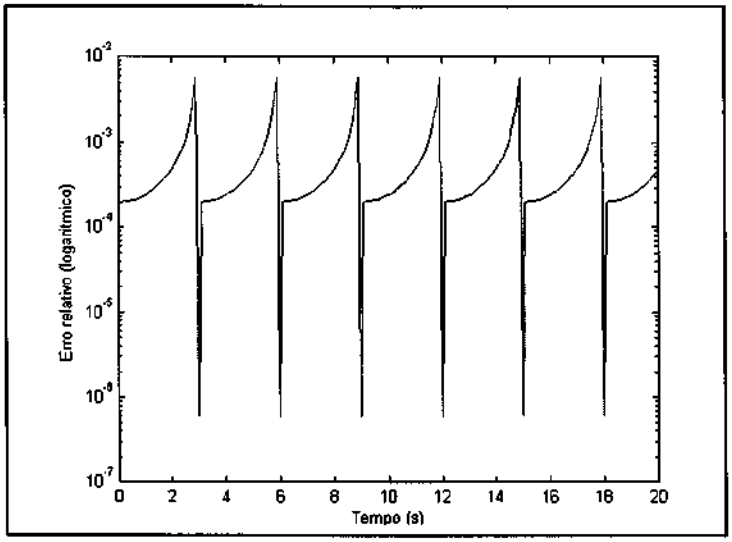

Figura 9.b – Comportamento do erro no Algoritmo II

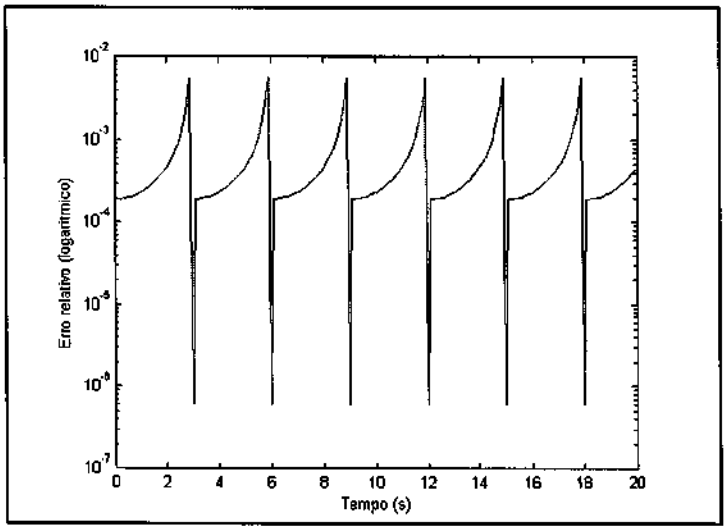

Figura 9.c – Comportamento do erro no Algoritmo III

Percebe-se nos três algoritmos que o erro é periódico, com um período de repetição de 3,0s. Notou-se, examinando as matrizes de erro (de dimensão 3x3), que os termos $a_{1,1}$, $a_{1,2}$, $a_{2,1}$ e $a_{3,1}$ foram os que apresentaram maiores erros dentro das matrizes. Contudo, pela utilização do algoritmo de "retirada de lixo" (os termos que tendem a zero são igualados a este valor), os erros no final de cada período de 3,0 são bem menores que os demais, já que os termos citados acima foram zerados (logo, os erros correspondentes a esses termos são zero) e os demais termos da matriz de erro apresentam valores de ordens de grandeza menores.

Nota-se que os erros do Algoritmo I são por volta de uma ordem de grandeza menores que os erros dos Algoritmos II e III. Como o Algoritmo III é um aperfeiçoamento do Algoritmo II, avaliou-se a melhoria nos erros. Abaixo é mostrado o gráfico com a diminuição percentual do erro relativo do Algoritmo III em relação ao do Algoritmo II.

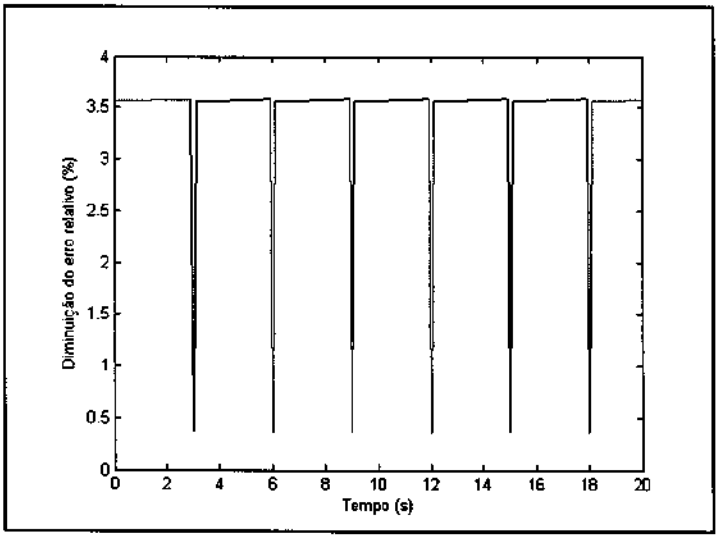

Figura 10 – Melhora do erro relativo do Algoritmo III em relação ao Algoritmo II

Nota-se que a diminuição relativa do erro é pouco mais de 3,5% com exceção dos instantes em que há a remoção dos "lixos", pontos em que o erro já é pequeno e a melhoria é inexpressiva.

Após essa análise, foi feita uma terceira simulação, de tempo máximo de 1800s, em que se avaliou a propagação do erro no tempo. Em cada instante foi avaliado o maior erro das matrizes até aquele instante. Foram obtidos os seguintes resultados:

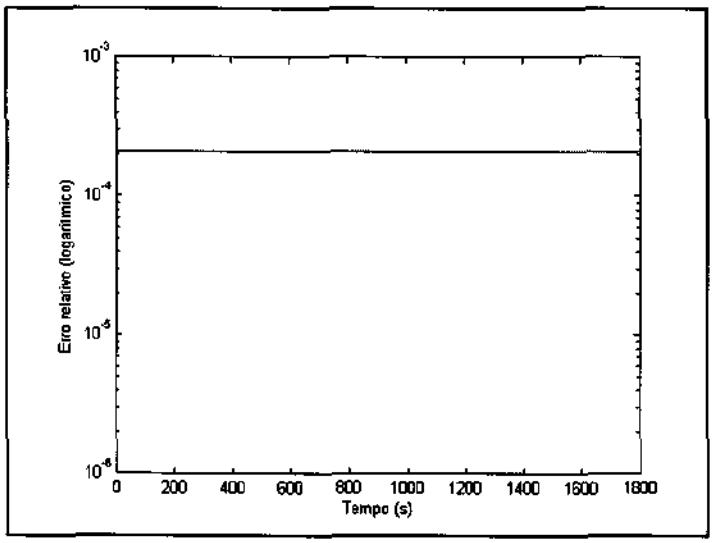

Figura 11.a – Propagação do erro no Algoritmo I

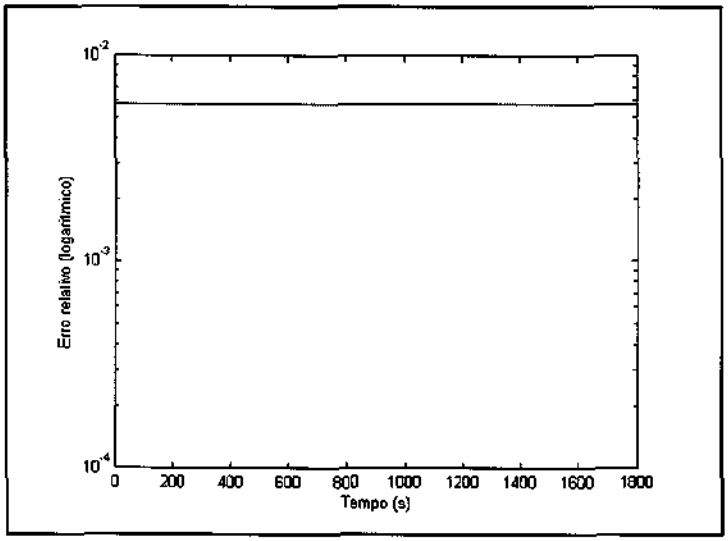

Figura 11.b – Propagação do erro no Algoritmo II

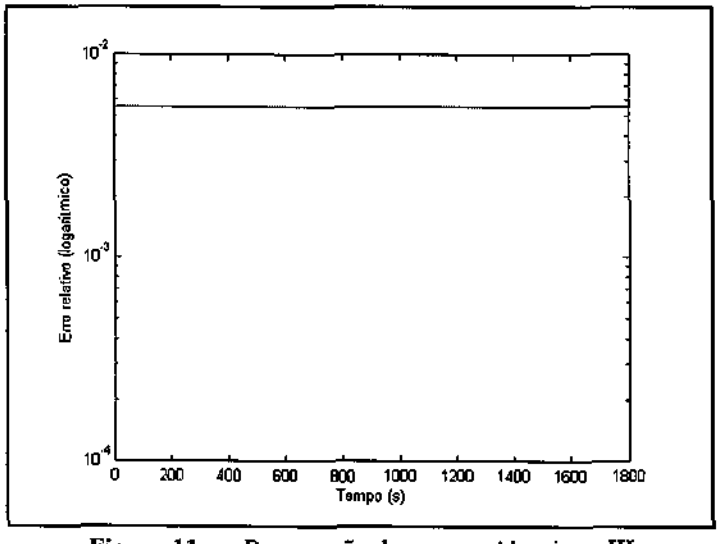

Figura 11.c – Propagação do erro no Algoritmo III

Observa-se nos gráficos que o maior erro tendeu a ficar constante no tempo, mesmo com uma simulação com tempo máximo de meia hora. Assim, observa-se que a amplitude os erros periódicos não variam de maneira significativa com o tempo, em um ambiente de coning puro.

Em seqüência, uma quarta simulação foi realizada, fazendo uma varredura em freqüência nos três algoritmos. A freqüência em questão é a de aquisição de dados do girômetro e cálculos efetuados. Para melhor comparação entre os algoritmos, a freqüência considera nos gráficos foi o inverso do período do intervalo menor. Contudo, ao efetuar a varredura, manteve-se sempre a relação entre os períodos: o período do intervalo maior sendo quatro vezes maior que o do intervalo menor, e o período deste sendo três vezes maior que o do subintervalo menor.

Na varredura em freqüência avaliou-se o erro no instante final da simulação para cada freqüência. Contudo, como se pôde notar na primeira simulação, em que se mostrou o comportamento dos erros em cada algoritmo, os erros são periódicos e crescentes dentro de cada período. Assim, para cada algoritmo, foram feitas três varreduras, uma com tempo final de 60,0s, outra com tempo final de 61,0s, e uma terceira, com tempo final de 61,8. Isso permite avaliar o erro em três pontos distintos dentro de um período de repetição do erro.

Nessa simulação foram obtidos os gráficos apresentados nas páginas seguintes:

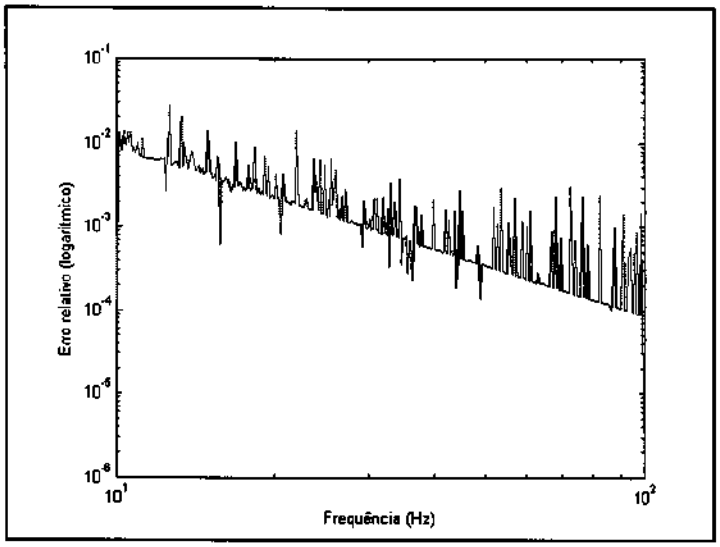

Figura 12.a – Varredura em freqüência – Algoritmo I ($t_{máx}$ = 61,0s)

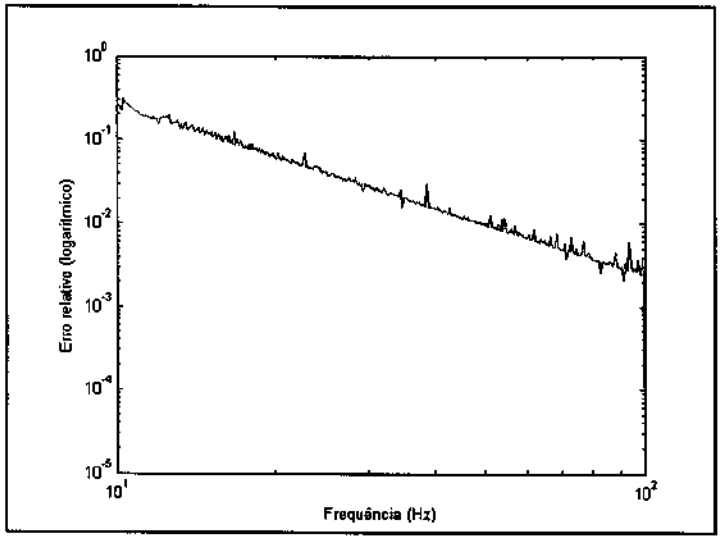

Figura 12.b – Varredura em freqüência – Algoritmo II ($t_{máx}$ = 61,0s)

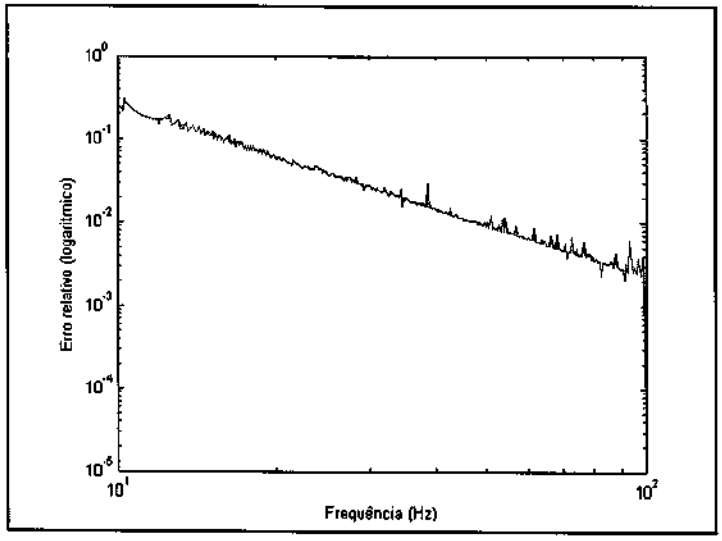

Figura 12.c – Varredura em freqüência – Algoritmo III ($t_{máx}$ = 61,0s)

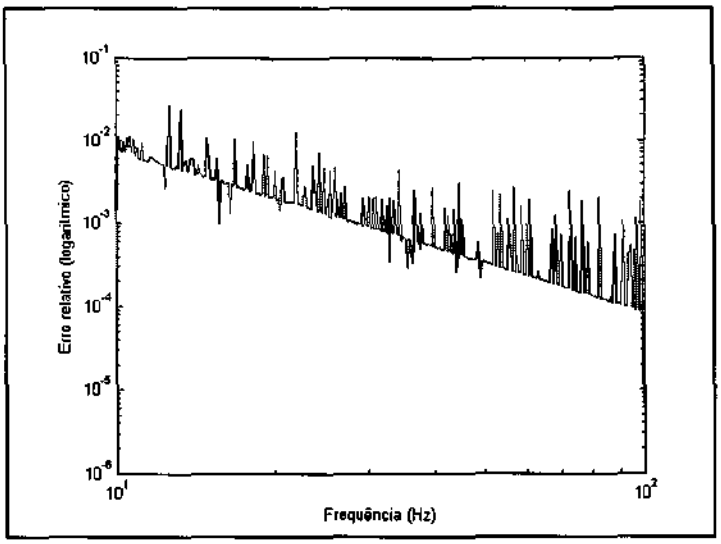

Figura 13.a – Varredura em freqüência – Algoritmo I ($t_{máx}$ = 62,0s)

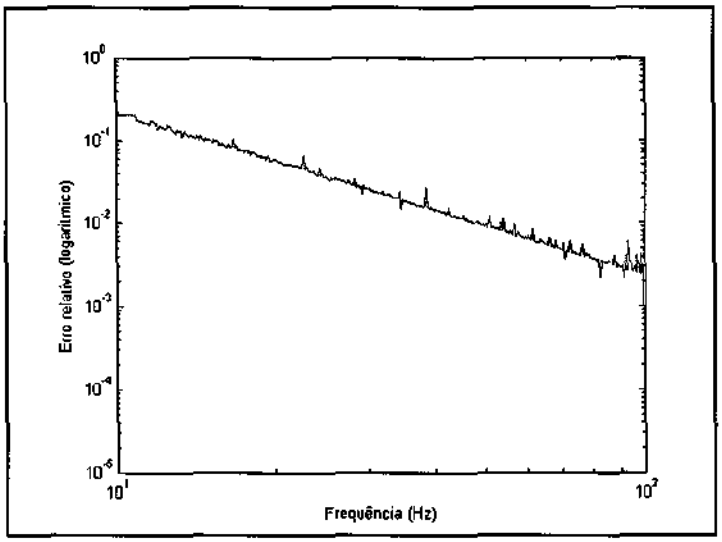

Figura 13.b – Varredura em freqüência – Algoritmo II ($t_{máx}$ = 62,0s)

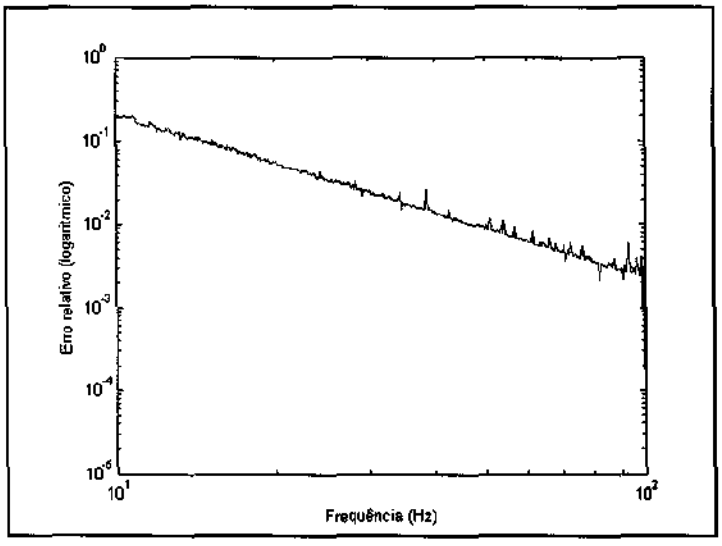

Figura 13.c – Varredura em freqüência – Algoritmo III ($t_{máx}$ = 62,0s)

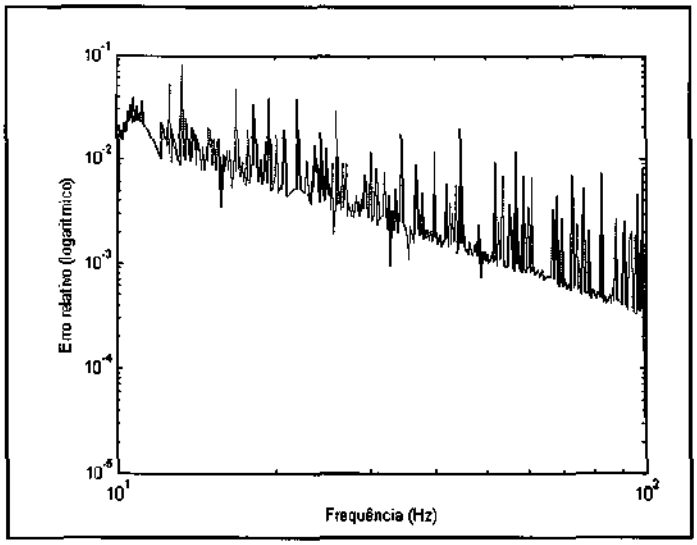

Figura 14.a – Varredura em freqüência – Algoritmo I ($t_{máx}$ = 62,8s)

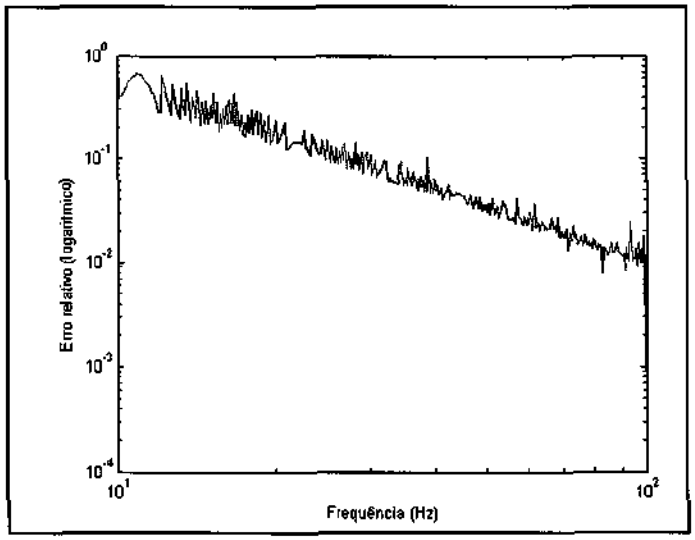

Figura 14.b – Varredura em freqüência – Algoritmo II ($t_{máx}$ = 62,8s)

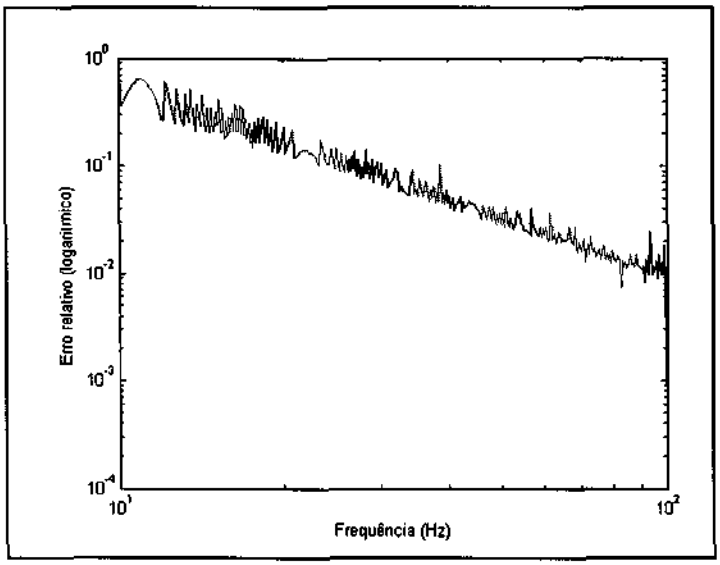

Figura 14.c – Varredura em freqüência – Algoritmo III ($t_{máx}$ = 62,8s)

Nota-se que, em todos os algoritmos, os erros tendem a diminuir com o aumento da freqüência de cálculo das iterações. Além disso, observa-se também uma considerável oscilação nos erros, dentro dessa tendência de queda, em especial nas varreduras no Algoritmo I e nas varreduras com tempo final de 62,8s (próximo ao ponto de pico de erro). Não se conseguiu determinar a causa exata dessa componente de alta freqüência, contudo imagina-se que uma possível causa seja erros de arredondamento. Uma proposta para novos trabalhos seria um melhor estudo dessas oscilações.

Nota-se, também, que no Algoritmo I os erros ficam em torno de uma ordem de grandeza abaixo dos erros nos outros dois algoritmos (desprezando-se as oscilações).

VII. Conclusões e trabalhos futuros

Os sistemas digitais de navegação inercial apresentam-se como uma boa opção para a determinação da localização espacial de veículos. Vários algoritmos para navegação inercial são propostos na literatura. Este trabalho foi focado nos algoritmos de determinação da matriz de transformação das coordenadas do sistema B, fixo ao corpo, para o sistema L, local.

Foram analisados três algoritmos, um proposto por Savage [5] (Algoritmo I), e dois propostos por Ignagni [2] (Algoritmos II e III).

Foi apresentada a idéia de algoritmos com diferentes velocidades para se efetuar os cálculos. Em cada algoritmo, utiliza-se uma parte rápida (dentro do chamado intervalo menor), para determinação do incremento de coning, e uma parte de velocidade normal (dentro do chamado intervalo maior), para a determinação da matriz de transformação. No Algoritmo III, o intervalo menor foi subdividido em três subintervalos menores, para melhorar a acurácia.

Foram apresentados os equacionamentos de cada algoritmo e a sua implementação em C++. Para o Algoritmo I, apresentou-se, também, sua implementação em MatLab.

Fez-se a análise dos algoritmos em um ambiente de coning puro, com ângulo de coning de 2,0rad e período de coning de 12,0s. Na primeira análise, utilizou-se 0,1s como período do maior intervalo e 0,025s para o intervalo menor. Notou-se que nesse tipo de ambiente o erro é cíclico, com período de 3,0s. Notou-se, também, que o Algoritmo I apresentou desempenho melhor que os demais, com erros em torno de uma ordem de grandeza menor. Comparando-se os Algoritmo III com o Algoritmo II, notou-se que o primeiro apresentou uma pequena melhora em relação ao último, com erros em torno de 3,5% menores.

Fazendo uma varredura em freqüência nos três algoritmos, variando o período dos intervalos de iteração, para três instantes distintos em que se avaliavam os erros, notou-se que o Algoritmo I novamente apresentou desempenho melhor que os demais, com erros em torno de uma ordem de grandeza menor. Contudo, esse foi o algoritmo que apresentou maior oscilação nos erros.

Pelas análises feitas, concluiu-se que o Algoritmo I foi o que apresentou o melhor comportamento em um ambiente de coning puro.

Como proposta para trabalhos futuros sugere-se a continuação deste trabalho, implementando a integração das medidas acelerométricas, através do algoritmo proposto por Savage [6]. Além disso, uma vez implementadas as integrações acelerométricas, seria possível estudar o impacto dos erros das matrizes de conversão na determinação da posição do móvel.

Outra proposta seria analisar o comportamento desses algoritmos com variações no período e no ângulo de coning, e também em ambientes diferentes de coning puro.

E por fim, também se pode estudar os conceitos e aplicações propostos Ignagni [3] para a otimização dos algoritmos de compensação de coning.

VIII. Referências Bibliográficas

[1] HUGHES, P.C., *Spacecraft Attitude Dynamics*, John Wiley, New York, 1986.

[2] IGNAGNI, M. B. Optimal Strapdown Attitude Integration. *Journal of Guidance, Control, and Dynamics*, v. 13, n 2, p. 363-369, 1990.

[3] IGNAGNI, M.B., Efficient Class of Optimized Coning Compensation Algorithms, *Journal of Guidance, Control and Dynamics*, Vol. 19, No 2, March-April 1996, p. 424-429.

[4] RADIX, J.C.. *Systèmes Inertiels à Composant Liés "Strap-Down"*. Cépadues Éditions, France, 1980.

[5] SAVAGE, P.G. Strapdown Inertial Navigation Integration Algorithm Design Part 1: Attitude Algorithms. *Journal of Guidance, Control, and Dynamics*, v. 21, n 1, p. 19-28, 1998.

[6] SAVAGE, P.G., Strapdown Inertial Navigation Integration Algorithm Design Part 2: Velocity and Position Algorithms, *Journal of Guidance, Control and Dynamics*, Vol. 21, No 2, March-April 1998, p. 208-221.

[7] WALDMANN, J. *Apostila do Curso de Sistemas de Navegação Inercial.* Instituto Tecnológico de Aeronáutica. São José dos Campos, Brasil, 1994.

[8] WALDMANN, J. Attitude determination algorithms, computational complexity, and the accuracy of terrestrial navigation with strapdown inertial sensors. In: *Anais do XIV Congresso Brasileiro de Automática*, Natal, Brasil, p. 2367-2372, 2002.