Universidade Estadual de Londrina
Centro de Tecnologia e Urbanismo
Departamento de Engenharia Elétrica

Software Simulador de um Determinador de Atitude

Luis Carlos de Albuquerque Silva

Londrina, 29 de Maio de 2008

Universidade Estadual de Londrina
Centro de Tecnologia e Urbanismo
Departamento de Engenharia Elétrica

Software Simulador de um Determinador de Atitude

Luis Carlos de Albuquerque Silva

Marcelo Carvalho Tosin

Orientador

Banca Examinadora

Marcelo Carvalho Tosin – Presidente

Fernando Cardoso Castaldo – UTFPR/Curitiba

Robinson Hoto – UEL/Londrina

Roberto V. L. Lopes – INPE/São José dos Campos

José Alexandre de França - UEL/Londrina

Dissertação submetida ao Departamento de Engenharia Elétrica da Universidade Estadual de Londrina, para preenchimento dos pré-requisitos parciais para obtenção do título de Mestre em Engenharia Elétrica

Londrina, 29 de Maio de 2008

à minha família: pais, esposa e filhos
os primeiros por contribuírem com a base da minha educação pessoal e à esposa e filhos
por estarem sempre me apoiando direta e indiretamente.

ALBUQUERQUE Silva, Luis Carlos de. Software Simulador de um Determinador de Atitude, 2007. Dissertação de Mestrado (Mestrado em Sistemas Eletrônicos) – Universidade Estadual de Londrina.

Resumo

Este trabalho propõe o desenvolvimento de um software para simulação de um determinador de atitude baseado em sensores microfabricados. O determinador, objeto de estudos, utiliza o campo geomagnético, o campo gravitacional e girômetros para o cálculo da atitude.

Este simulador foi desenvolvido utilizando linguagem de programação C++ e a biblioteca gráfica OpenGL. Ele simula o funcionamento de um hardware projetado com os sensores microfabricados, bem como suas funções de transferências e ruídos.

O simulador recebe como parâmetros de entrada os valores dos sensores de referência, além dos ganhos e offsets de cada um dos sensores e a temperatura inicial. Os ganhos, offsets e acelerações impostas ao sistema podem ser modificados durante as simulações, reproduzindo efeitos de variações dos parâmetros dos sensores com a temperatura e movimentos acelerados.

Os conceitos de orientação de corpos, referenciais inerciais, ângulos de Euler, quatérnions também são apresentados.

O problema da orientação de corpos é resolvido com a utilização do algoritmo TRIAD ou QUEST e a união das informações dos sensores de referências e giros é contornada através da introdução do Filtro de Kalman.

O funcionamento básico deste software consiste da simulação dos sinais elétricos dos sensores sob o efeito de rotações e vetores resultantes de aceleração e campo magnético também simulados. O software também simula a conversão analógico-digital destes sinais, realiza sobre os mesmos uma filtragem passa-baixa tipo média-móvel, para evitar *aliasing*. Posteriormente são realizados cálculos para a determinação da atitude utilizando-se do algoritmo TRIAD e de um Filtro de Kalman para "fundir" as informações da atitude calculadas a partir dos giros e pelo TRIAD. Os resultados da simulação são apresentados ao usuário através de projeções

tridimensionais de objetos que rotacionam segundo regras estabelecidas e segundo os algoritmos em teste.

Este simulador será utilizado como ferramenta para o desenvolvimento e testes de um determinador de atitude real. Futuramente este software poderá ser empregado em testes reais em mesa de rotação simulando sinais de sensores e testando modelos de calibração.

Por último, são apresentadas as conclusões sobre todo o trabalho, mostrando as dificuldades encontradas, bem como os meios utilizados para resolvê-las.

Abstract

This work proposes the development of an attitude determination simulation software based on MEMS sensors. The simulated attitude determination system uses the geomagnetic field, the gravitational field and girometers for the attitude calculations.

This simulator was developed using the C++ programming language and the OpenGL graphical library. It simulates the operation of a hardware designed and built with MEMS sensors. It also simulates the sensors transfer functions and noise.

The simulator receives as input the values of the reference sensors, the gains and offsets of each sensor and also the initial temperature. The gains, offsets and accelerations imposed to the system can be modified during simulation, reproducing the sensor's parameters temperature variation effects and also reproducing accelerated movements.

The concepts of a body orientation, inertial references, Euler angles and quaternions are also presented.

The body orientation problem is solved using the TRIAD or QUEST algorithms and the union of the reference sensors and girometers information is performed by the introduction of a Kalman filter.

This software basic operation consists on the sensors electrical signal output simulation under the rotation effects and also under the acceleration vector resultant and the magnetic field vector resultant. The software also simulates the analog to digital conversion of these signals and the low pass filtering operation to avoid aliasing. Further the software performs calculations to determine the attitude using the TRIAD algorithm and a Kalman filter to "fuse" the attitude information calculated form the girometers and by the TRIAD. The simulation results are shown to the user through tri-dimensional projections of objects that rotate following established rules and following the algorithms under test.

This simulator will be used as a tool to help develop and test a real attitude determination system. In a future this software will be employed during real tests with a rotation table simulating the sensors signals and testing calibration models.

Finally, conclusions of this work are presented showing its difficulties and also the methods found to solve them.

Agradecimentos

- A Deus por toda sua criação, e assim, permitir que os seres humanos possam trabalhar como co-criadores.

- Aos meus pais, que me deram as condições necessárias para desenvolver meus estudos.

- À minha esposa e filhos, que sempre me apoiaram nos desafios que encontrei pelo caminho.

- Ao meu orientador, Marcelo C. Tosin, que sempre esteve à disposição para me encaminhar na solução dos problemas.

- Ao colega-mestre Franciso Granziera Jr., que atuou como espécie de consultor e co-orientador, sem medir esforços.

- À AEB – Agência Espacial Brasileira que através do Projeto Uniespaço, permitiu a participação de simpósio no INPE.

- Ao Centro de Pesquisas e Desenvolvimento Leopoldo A. Miguez de Mello (Cenpes), pela oportunidade de apresentar este trabalho, em forma de palestra no, V Simpósio Brasileiro de Engenharia Inercial (SBEIN), na cidade do Rio de Janeiro.

Sumário

Lista de Acrônimos

AD	Analógico para Digital
ADC	Analog Digital Converter – Conversor Analógico-Digital (AD)
API	Application Programming Interface – Interface de Programação de Aplicativos
CAD	Computer-Aided Design – Desenho auxiliado por computador
DAC	Digital Analog Converter – Conversor Digital Analógico (DA)
FK	Filtro de Kalman
FPB	Filtro passa-baixa
FPB M.M.	Filtro passa-baixa tipo média-móvel
FT	Função de transferência
MEMS	Micro Eletro-Mechanic System – Sistema microeletromecânico
OpenGL	Open Graphic Library – Biblioteca Gráfica Aberta (código aberto)
POO	Programação Orientada a Objetos
QUEST	Quaternion Estimator – Estimador de Quatérnion
RMS	Root Mean Square – Raiz Quadrada da Média Quadrática
TRIAD	Trix-axis Attitude Determination – Determinação de attitude em três eixos

Lista de Figuras

Lista de Símbolos

$\boldsymbol{\psi}, \boldsymbol{\theta}$ *e* $\boldsymbol{\varphi}$	Ângulos de Euler, representando *Yaw, Pitch e Roll*, respectivamente.
$\mathbf{I}$	Matriz Identidade. Sua dimensão variará de acordo com o contexto, e será especificada.
$\mathbf{A}$	Matriz de Atitude.
$\mathbf{R}$	Matriz de rotação construída a partir dos ângulos de Euler
$\mathbf{R}_{\varphi}^{x}$	Matriz de rotação de $\boldsymbol{\varphi}$ graus em torno do eixo *x*
$\mathbf{R}_{\theta}^{y}$	Matriz de rotação de $\boldsymbol{\theta}$ graus em torno do eixo *y*
$\mathbf{R}_{\psi}^{z}$	Matriz de rotação de $\boldsymbol{\psi}$ graus em torno do eixo *z*
$\breve{q}, \breve{p}$ *e* $\breve{r}$	Quatérnions
q_0 *ou* q_4	Componente escalar do quatérnion
$q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$	Componentes vetoriais do quatérnion
$\mathbf{q}$	Parte vetorial do quatérnion
i, j e **k**	Números complexos que formam uma base canônica
$\otimes$	Operador do produto entre quatérnions
$\bullet$	Operador para produto interno
$\times$	Operador para produto vetorial
*	Operador para conjugado
$\lvert . \rvert$	Operador para norma
$\hat{\breve{q}}$	Quatérnion unitário
$\hat{\mathbf{u}}$	Vetor unitário
v, w	Vetores quaisquer
Q	Matriz de rotação constituída a partir de quatérnions
$\dot{\breve{q}}(t)$	Derivada temporal do quatérnion $\hat{\breve{q}}$
$\hat{\mathbf{v}}_i$	Vetores unitários (versores). Utilizados como vetores de referência para cálculo da matriz de atitude
$\hat{\mathbf{w}}_i$	Vetores unitários (versores). Utilizados como vetores de observação no cálculo da matrz de atitude.
$\hat{\mathbf{r}}_i$	Tríade de referência

$\hat{\mathbf{s}}_t$	Tríade de observação
$\mathbf{M}_{ref}$	Matriz de referência
$\mathbf{M}_{obs}$	Matriz de observação
$\langle \, . \, \rangle \, ou \, \mathbf{E}[\, . \,]$	Operadores de média
$\mathbf{P}_{\theta\theta}$	Matriz de covariância da atitude
$\mathbf{P}$	Matriz de covariância total
$\delta\mathbf{A}$	Matriz diferença de atitude (erro de atitude)
$\mathbf{tr}(\, . \,)$	Operador traço de uma matriz
$\mathbf{P}_{obs}$	Matriz de covariância da matriz de observação
$\mathbf{P}_{ref}$	Matriz de covariância da matriz de referência
$\boldsymbol{\sigma}_x^2$	Variância de um vetor qualquer
$\mathbf{P}_{qq}$	Matriz de covariância de um quatérnion
$\boldsymbol{\sigma}_{tot}^2$	Variância total
$\mathbf{x}(t)$	Vetor de estados
$\mathbf{w}(t)$	Vetor ruído do processo
$\mathbf{Q}(t)$	Matriz de covariância do ruído do sistema
$\mathbf{P}(t)$	Matriz de covariância do estado
$\boldsymbol{\Omega}_4(\, . \,)$	Operador linear de distribuição
$\boldsymbol{\omega}$	Velocidade angular
$\boldsymbol{\Delta\theta}$	Vetor ângulo incremental
$\mathbf{M}(\boldsymbol{\Delta\theta})$	Matriz de transição para o quatérnion
$\mathbf{u}$	Vetor de medidas dos giroscópios
$\mathbf{b}$	Vetor *bias* adicionado às medidas dos giroscópios
$\hat{\boldsymbol{\omega}}$	Vetor verlocidade angular estimada
$\tilde{\mathbf{K}}_k$	Matriz ganho de Kalman reduzida
$\tilde{\mathbf{P}}_k$	Matriz de covariância reduzida
$\tilde{\mathbf{Q}}_k$	Matriz covariância do ruído reduzida

Capítulo 1 – Introdução

1.1 – Tema

Atualmente torna-se importante o estudo, e implementação de dispositivos utilizados na orientação de corpos rígidos, isto significa, determinar a atitude destes corpos. Assim, estes dispositivos podem ser utilizados em uma infinidade de aplicações, sejam aeroespaciais, marítimas, terrestres, logística, jogos, etc. [AEB04]

Na prática, a determinação de atitude pode ser feita através da combinação de micro sensores eletromecânicos (MEMS), que transformarão grandezas como o campo gravitacional, o campo geomagnético, posição de estrelas, luz, em sinais elétricos. Além destes sensores, também podem ser utilizados sensores inerciais, que são baseados em princípios físicos ligados à inércia; eles são os acelerômetros, giroscópios mecânicos, piezelétricos.

Quando dois ou mais sensores em direções distintas são utilizados, é possível calcular uma matriz de rotação, chamada matriz de atitude.

Como todos os sensores apresentam erros (ruídos) inerentes à sua própria construção, a combinação dos sensores acarreta em acúmulo de erros, que se propagam ao longo do tempo de amostragem.

Neste sentido, faz-se necessária a utilização de filtros para estimar a orientação do corpo, a partir de dados estatísticos, otimizando a estimação. Nestes casos é muito utilizado o filtro de Kalman.

Em trabalho anterior [FGJ06], estas questões descritas acima foram resolvidas com o desenvolvimento de um hardware para a determinação de atitude.

Um dos grandes problemas para estabilização do hardware é a calibração dos sensores, pois os mesmos apresentam polarização (bias) e ganho, e estes valores variam em cada um dos sensores da mesma pastilha, além de variarem ao longo do tempo.

Assim, o desenvolvimento de uma ferramenta que simule os sinais de entrada e saída dos sensores e seus ruídos, além dos algoritmos para determinação de atitude e Filtro de Kalman, é uma forma para a solução destes problemas mencionados acima.

1.2 – Meta

O objetivo deste trabalho é desenvolver uma ferramenta para simulação de um determinador de atitude.

Ele é parte da pesquisa para apresentação do Projeto "Desenvolvimento de um Determinador de Atitude" [DET04], aprovado pela Agência Espacial Brasileira. Este estudo foi conduzido em 3 fases. Na primeira, foram desenvolvidos os modelos para resolver o problema da determinação de atitude. Também foram desenvolvidos modelos para sensores inerciais, e por último implementar um simulador para determinação de atitude.

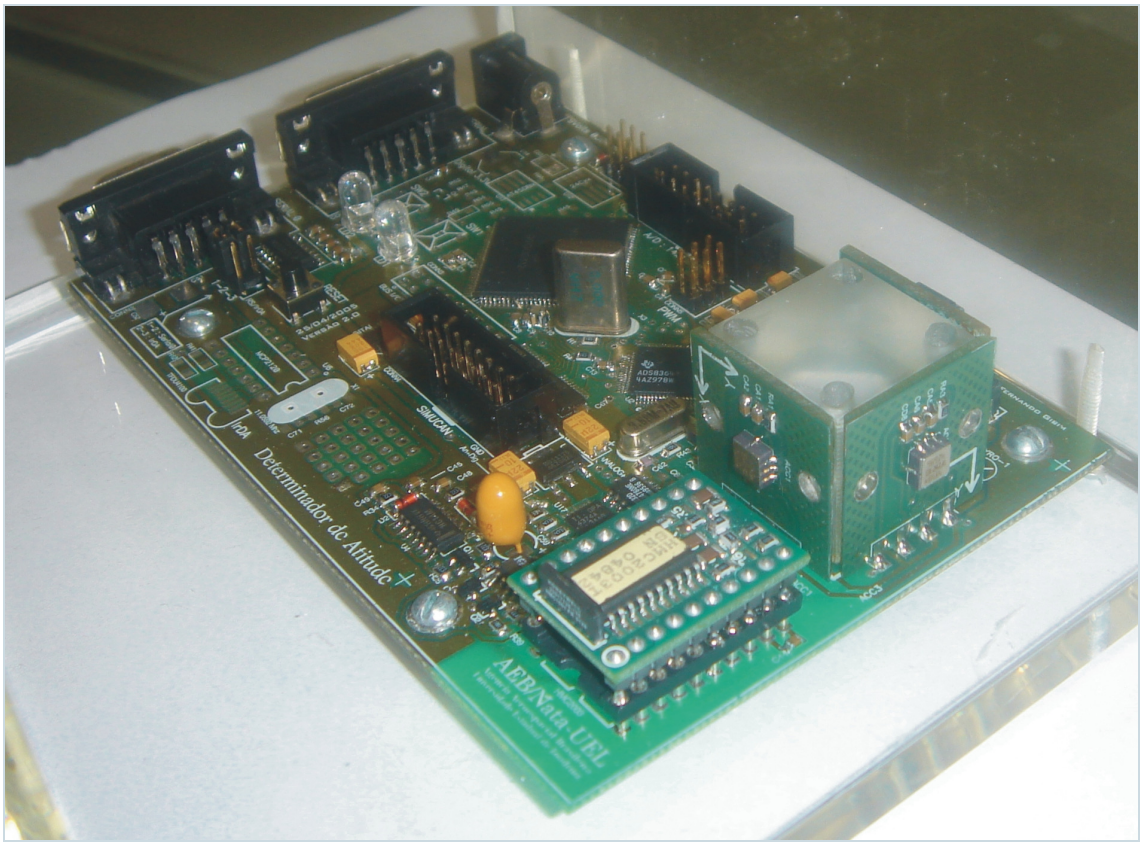


Figura 1.1 – Protótipo do determinador de atitude baseado em sensores MEMS

O simulador é dividido basicamente em dois subsistemas. O primeiro chamado de gerador de movimentos e o segundo chamado de sistema simulador. A figura abaixo simplifica o modelo do simulador.

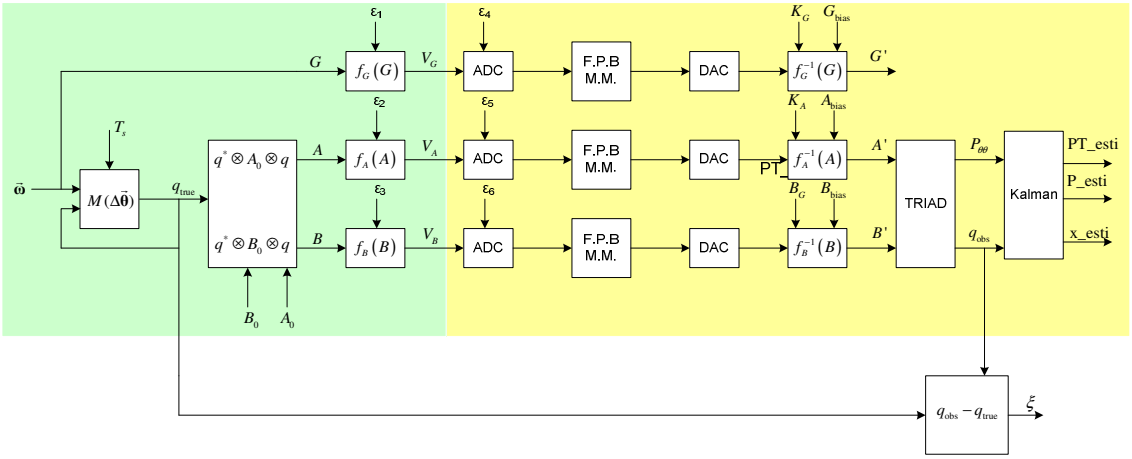


Figura 1.2 – Diagrama de bloco dos dois subsistemas

O software simula os sinais dos sensores, ruídos, conversão analógico-digital, filtro passa-baixa tipo média-móvel, conversão digital-analógico, algoritmo TRIAD e Filtro de Kalman.

O software foi desenvolvido em programação orientada a objetos, especificamente C++ e a apresentação gráfica utilizada foi a biblioteca gráfica OpenGL, e nenhuma norma de engenharia de software foi utilizada na elaboração do código-fonte do simulador.

Os valores resultantes da simulação podem ser armazenados em arquivos no formato texto, para serem comparados aos valores reais de um hardware.

Uma outra maneira de combinar o simulador ao protótipo do hardware, é conectá-los através de uma saída serial, tipo RS-232 e realizar ensaios em uma mesa de rotação. Ou seja, a mesa produz movimentos, os sinais dos sensores do hardware são transmitidos ao simulador, e este último apresenta na tela de um microcomputador os movimentos. E o processo inverso também pode ser ensaiado. O simulador gera os movimentos, que são transmitidos à mesa de rotação, e o hardware apresenta em um segundo microcomputador o resultado dos sinais lidos em seus sensores.

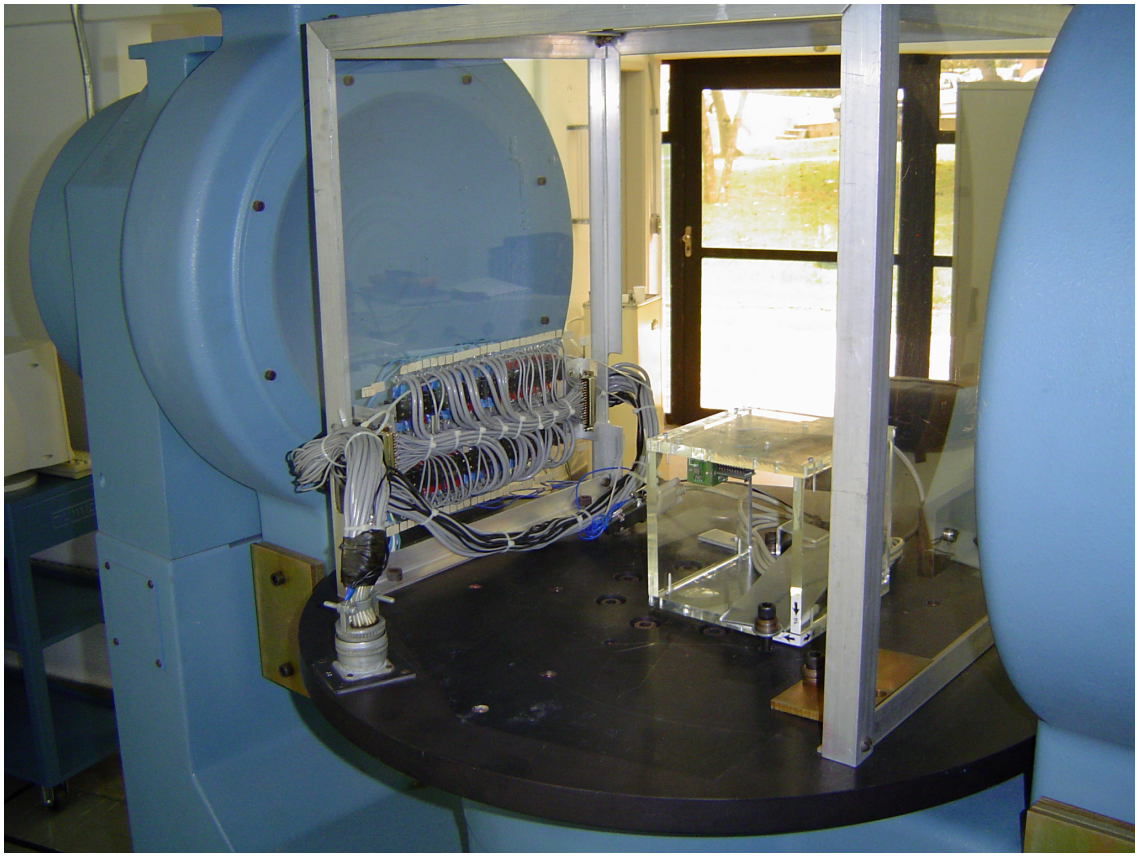


Figura 1.3 – Mesa de rotação. Protótipo do determinador de atitude no interior da mesa

Capítulo 2 – Sistemas de Coordenadas na Determinação de Atitude

2.1– Introdução

Um requisito básico na determinação de atitude de corpos rígidos é determinar a rotação necessária que deve ser aplicada ao sistema de coordenadas de referência, levando o mesmo ao sistema de coordenadas definido no corpo.

Assim, o primeiro passo para determinar a rotação é conhecer o que é sistema de coordenadas, e como defini-lo em uma referência e também no corpo a ser rastreado.

2.2- Definição do sistema de coordenadas

Considerando um ponto ou conjunto de pontos no plano. A localização relativa de cada ponto é definida com respeito ao sistema de coordenadas fixado no plano.

O que isto significa é que um ponto arbitrário é fixado no plano, sendo chamado de origem. Duas linhas têm interseção na origem, perpendicularmente, que podemos chamar, por exemplo, horizontalmente como eixo x e o outro verticalmente como eixo y. Os eixos são linhas de números reais, aumentando para a direita no eixo x e para cima no eixo y, que servirão como orientação.

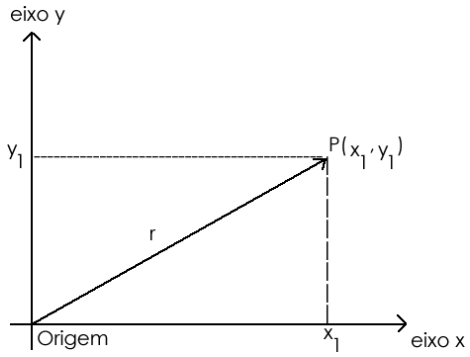


Figura 2.1 – Sistema de coordenadas bidimensional

A origem representa o ponto zero que é designada como o par de números reais (0, 0). Cada ponto distinto, P, no plano pode ser ordenado com um par de números reais (x1, y1) como mostrado na figura acima. Este conceito forma um plano de coordenadas, chamado R^2 .

A partir da criação do sistema de coordenadas no plano, é possível criar um sistema de coordenadas no espaço com 3 dimensões. O terceiro eixo necessário pode ser chamado de eixo z, e é perpendicular ao plano xy.

2.3- Regra da mão direita

Para rotações no plano é muito útil imaginar um eixo z que está diretamente fora do plano em direção ao observador e perpendicular aos eixos x e y na origem. Então, usando a regra da "mão direita", com o polegar na direção positiva do eixo z, uma rotação positiva, roda o eixo x para o eixo y.

A partir destes conceitos é possível compreender o uso de quatérnions, bem como as rotações em R^3 .

2.4- Rastreamento

Em aplicações sobre a Terra, como por exemplo, aviação, ou seja, um objeto remoto, como por exemplo, um avião, pode ser rastreado de algum ponto da superfície terrestre. O Plano Local Tangente é simplesmente um plano tangente à superfície da Terra neste ponto. Nós definimos um sistema de coordenadas inicial com os eixos X e Y, concordante com estes eixos as direções Norte e Leste respectivamente. O eixo Z está no geocentro, isto é, ele aponta para o centro da Terra. Aí teremos um sistema de coordenadas concordante com a regra da mão direita.

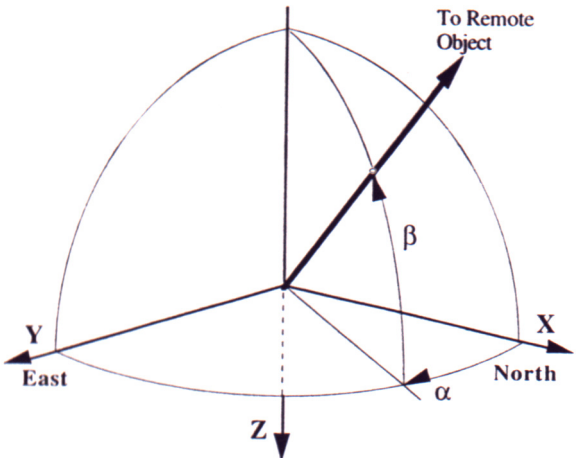


Figura 2.2 – Rastreamento de um objeto [KUI02]

Podemos definir um ângulo α, chamado Heading, que é o ângulo no plano tangente entre o Norte e a direção projetada para o objeto remoto. Também definimos um ângulo β, chamado Elevação, que é o ângulo entre o plano tangente e a direção ao objetivo remoto a ser rastreado, como mostra a figura acima. A Transformação de Rastreamento é primeiro uma rotação sobre o eixo Z através do ângulo α, seguida pela rotação sobre um novo eixo y através do ângulo β. Podemos notar que no sistema de coordenadas resultante o novo eixo x é está apontando diretamente para o objeto que está sendo rastreado.

2.5- Conclusões relativas ao projeto

Para o simulador desenvolvido neste trabalho, a gravidade e o campo geomagnético são boas referências para a determinação de atitude. Quando o corpo estiver em movimento não acelerado, o vetor gravidade apresentará módulo igual a um.

O campo geomagnético não possui a mesma inclinação e declinação por todo globo. Assim é necessário o uso de um modelo que diga qual a sua inclinação e declinação em determinado ponto sobre o globo terrestre.

Então para se minimizar estas e outras limitações é necessário o uso de outros sensores (girômetros) e executar algoritmos de filtragens.

Capítulo 3 – Representação da Atitude e Algoritmo TRIAD

3.1– Introdução

Muitas aplicações tais como micro veículos aéreos (ou micro air vehicles – MAVs), ou mesmo orientação e rastreamento de humanos e robôs utilizando sensores inerciais, necessitam de sistemas para determinação de atitude, pequenos e baratos. Com a proliferação de micro sensores inerciais de baixo custo baseados em sistemas eletro-mecânicos (MEMS), tornou-se viável construir sistemas de determinação de atitude pequenos e baratos.

A determinação de atitude baseada em sensores MEMS é feita pela utilização de três métodos ao mesmo tempo. O primeiro método é integrar a velocidade angular instantânea através da utilização de micro giroscópios. O segundo método é utilizar acelerômetros e magnetômetros de efeito Hall. Os ângulos pitch e roll [KUI02] são estimados pelo uso das componentes da gravidade deduzidas das medidas dos acelerômetros e o ângulo heading é determinado pela medida dos magnetômetros.

Assim, os valores da determinação de atitude normalmente são representados através de:

3.2– Ângulos de Euler [KUI02]

3.2.1- Introdução

Leonard Euler (1707-1783) foi um dos gigantes da matemática. Ele teve uma grande contribuição à Física em geral, e particularmente à matemática, tendo vários trabalhos na Mecânica e Dinâmica. Fazendo a ligação de um de seus teoremas a este trabalho, temos que:

A partir de dois sistemas de coordenadas independentes, podemos relacioná-los a um sistema de rotações (não mais que três) sobre os eixos de coordenadas, onde não podem existir duas rotações consecutivas sobre o mesmo eixo.

Quando dizemos “dois sistemas de coordenadas independentes são relacionados”, significa que uma seqüência de rotações sobre sucessivos eixos de coordenadas levará o primeiro quadrante ao segundo.

O ângulo de rotação sobre um sistema de coordenadas é chamado de Ângulo de Euler. Uma seqüência destas rotações é frequentemente chamada de Seqüência de Ângulos de Euler.

A restrição imposta pelo teorema que não podem existir duas rotações consecutivas sobre o mesmo eixo, leva-nos a não mais que doze combinações de rotações.

xyz	yzx	zxy
xzy	yxz	zyx
xyx	yzy	zxz
xzx	yxy	zyz

Estes ângulos são chamados de Ângulos de Euler, porque foi Leonard Euler quem primeiramente usou seqüência de ângulos para determinar os relacionamentos das órbitas na mecânica celestial. Entre as aplicações que utilizam estas seqüências de ângulos, estão: aeroespaciais e marítimas.

3.2.1– A Seqüência Aeroespacial

Entre as aplicações que utilizam as Seqüências de Ângulos de Euler, está o estudo Aeroespacial. Por exemplo, um instrumento colocado na cabina do piloto, chamado de indicador de Heading e Atitude, têm a função de relacionar a orientação do Sistema de Coordenadas do Avião e o Sistema de Referência (Plano Tangente Local e o Norte). Nesta aplicação o eixo x é positivo no sentido da cabine da aeronave, o eixo y à direita e o eixo z é normal ao plano formado pelos eixos x e y, apontando para baixo.

O sistema de coordenadas é definido em termos do plano tangente normal da Terra e o Norte. Isto é, o eixo Z é positivo em direção geocêntrica. O eixo X aponta ao norte no plano tangente e o eixo positivo Y a leste. Esta tríade de referências da Terra define um sistema de coordenadas ortonormal da mão direita.

Os termos Yaw, Pitch e Roll são utilizados para referenciar um desvio incremental ou perturbação sobre o estado nominal do heading e atitude da aeronave.

A seqüência de rotação Aeroespacial pode ser expressada como o produto de matriz:

$$R = R_{\phi}^{x} R_{\theta}^{y} R_{\psi}^{z} \tag{3.1}$$

$$= R_{\phi}^{x} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ -\sin\psi & \cos\psi & 0 \\ \sin\theta\cos\psi & \sin\theta\sin\psi & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} \cos\psi\cos\theta & \sin\psi\cos\theta & -\sin\theta \\ \begin{pmatrix} \cos\psi\sin\theta\sin\phi \\ -\sin\psi\cos\phi \end{pmatrix} & \begin{pmatrix} \sin\psi\sin\theta\sin\phi \\ +\cos\psi\cos\phi \end{pmatrix} & \cos\theta\sin\psi \\ \begin{pmatrix} \cos\psi\sin\theta\cos\phi \\ +\sin\psi\sin\phi \end{pmatrix} & \begin{pmatrix} \sin\psi\sin\theta\cos\phi \\ -\cos\psi\sin\phi \end{pmatrix} & \cos\theta\cos\phi \end{bmatrix}$$

Este produto de matrizes de rotação é por si mesmo uma matriz de rotação que representa a Seqüência Aeroespacial em termos de Heading e Atitude. Esta matriz pode ser vista como uma simples rotação sobre algum eixo que toma o sistema de coordenadas de referência no sistema de coordenadas do corpo. Isto é uma conseqüência importante do Teorema de Euler. Assim é possível encontrar o eixo e ângulo desta simples rotação, que poderia equivaler à seqüência de rotações de Euler para a aplicação Aeroespacial.

3.3– Quatérnions

3.3.1 - Introdução

Em 1843, Sir William Rowan Hamilton [KUI02] inventou o chamado número hiper-complexo de 4 elementos, que recebeu o nome de quatérnion. A regra principal é:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{3.2}$$

Um quatérnion, como o próprio nome sugere, pode ser considerado como um conjunto de quatro números reais, isto é, como um elemento de $\mathbb{R}^4$. Neste caso poderíamos escrever:

$$q = (q_0, q_1, q_2, q_3) \tag{3.3}$$

Onde q_0, q_1, q_2 e q_3 são simplesmente números reais ou escalares.

Existe uma forma alternativa de representar um quatérnion. Primeiro, definimos uma parte escalar como sendo um número real ou escalar, chamada de q_0. Em seguida, definimos uma parte vetorial, chamada $\boldsymbol{q}$, que é um vetor ordinário em R^3.

$$\boldsymbol{q} = \boldsymbol{i}q_1 + \boldsymbol{j}q_2 + \boldsymbol{k}q_3 \tag{3.4}$$

Onde $\boldsymbol{i}, \boldsymbol{j}\, e\, \boldsymbol{k}$ são ortonormais em R^3. Assim, um quatérnion pode ser definido como a soma:

$$q = q_0 + \boldsymbol{q} \;=\; q_0 + \boldsymbol{i}q_1 + \boldsymbol{j}q_2 + \boldsymbol{k}q_3 \tag{3.5}$$

Nesta soma, $\boldsymbol{q_0}$ é chamado de parte escalar do quatérnion enquanto $\boldsymbol{q}$ é chamada de parte vetorial. Os escalares q_0, q_1, q_2, q_3 são chamadas de componentes do quatérnion.

3.3.2 – Igualdade e Adição

Começamos dizendo que dois quatérnions são iguais se e somente se tiverem todas componentes exatamente iguais, isto é:

$$p = p_0 + \boldsymbol{i}p_1 + \boldsymbol{j}p_2 + \boldsymbol{k}p_3 \tag{3.6}$$

e

$$q = q_0 + \boldsymbol{i}q_1 + \boldsymbol{j}q_2 + \boldsymbol{k}q_3 \tag{3.7}$$

Então $p = q$ se e somente se

$$\begin{aligned} p_0 &= q_0 \\ p_1 &= q_1 \\ p_2 &= q_2 \end{aligned} \tag{3.8}$$

$$p_3 = q_3$$

A soma de dois quatérnions p e q acima é definida pela soma das componentes, isto é:

$$p + q = (p_0 + q_0) + \boldsymbol{i}(p_1 + q_1) + \boldsymbol{j}(p_2 + q_2) + \boldsymbol{k}(p_3 + q_3) \tag{3.9}$$

A partir da definição acima, a soma de quatérnions é o mesmo que somar quatro componentes reais, e assim satisfaz às propriedades de campo aplicadas à adição. Então, a soma de dois quatérnions resulta em outro quatérnion. Também existe um quatérnion zero, onde cada uma das componentes do quatérnion é zero. Além disso, cada quatérnion q tem um negativo ou adição inversa, denotado de $-q$, onde cada componente é negativa em relação à componente de q. É fácil de verificar que a adição de quatérnions é comutativa e associativa, porque a soma de números reais tem esta propriedade.

3.3.3 – Multiplicação

Como no caso de vetores em R^3, o produto de um escalar e um quatérnion é definido de maneira direta. Se c é um escalar e q é o quatérnion:

$$q = q_0 + \boldsymbol{i}q_1 + \boldsymbol{j}q_2 + \boldsymbol{k}q_3 \tag{3.10}$$

Então o produto do quatérnion q e o escalar c é dados por:

$$cq = cq_0 + \boldsymbol{i}cq_1 + \boldsymbol{j}cq_2 + \boldsymbol{k}cq_3 \tag{3.11}$$

Assim, a multiplicação de um quatérnion por um escalar é simplesmente multiplicar cada uma das componentes pelo escalar. Isto mostra que este produto resulta em um quatérnion.

O produto de dois quatérnions é mais complicado. Ele deve ser definido de modo que os seguintes produtos especiais fundamentais sejam satisfeitos:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{3.12}$$

$$ij = k = -ji \tag{3.13}$$

$$jk = i - kj \tag{3.14}$$

$$ki = j = -ik \tag{3.15}$$

Recordando que estes produtos são aqueles que W. R. Hamilton disse que são necessários para realizar seu objetivo. Estes produtos não são comutativos, de modo que o produto de dois quatérnions também não será.

3.4– Algorítmo TRIAD

3.4.1 - Introdução

Para se calcular a atitude de um corpo é necessário calcular a matriz de rotação que realize a transformação de coordenadas de vetores arbitrários entre dois sistemas de referência. Estes dois sistemas são: inercial e outro fixado no corpo, sendo que a base do sistema de coordenadas deve ser conhecida em ambos os sistemas.

Na prática para se obter os valores dos vetores em ambos os sistemas de coordenadas, são utilizados sensores, que apresentarão ruídos. Assim, o cálculo da atitude não apresentará uma solução exata e será necessária a realização de um algoritmo de aproximação. No caso do simulador como são utilizados dois modelos de sensores, o algoritmo TRIAD oferece uma opção simples para encontrar o valor aproximado da matriz de atitude, com baixo custo computacional.

3.4.2- Desenvolvimento

A partir de dois vetores unitários de referência $\hat{\mathbf{V}}_1$ e $\hat{\mathbf{V}}_2$ que são projeções dos vetores fonte no sistema de coordenadas inercial e os vetores $\hat{\mathbf{W}}_1$ e $\hat{\mathbf{W}}_2$ são as projeções dos vetores unitários de observação no sistema de coordenadas do corpo. Assim, é desejável encontrar uma matriz ortogonal $\mathbf{A}$ que satisfaça as seguintes equações:

$$A\mathbf{V}_1 = \mathbf{W}_1 \qquad A\mathbf{V}_2 = \mathbf{W}_2 \tag{3.16}$$

Para calular a matriz de atitude é necessário construir dois conjuntos de três vetores ortogonais unitários ou tríades, cada conjunto baseado em um sistema de coordenadas, ou seja, um na referência e outro no corpo.

$$\hat{\mathbf{r}}_1 = \hat{\mathbf{V}}_1 \qquad \hat{\mathbf{r}}_2 = \frac{\left(\hat{\mathbf{V}}_1 \times \hat{\mathbf{V}}_2\right)}{\left|\hat{\mathbf{V}}_1 \times \hat{\mathbf{V}}_2\right|} \qquad \hat{\mathbf{r}}_3 = \frac{\left(\hat{\mathbf{V}}_1 \times \left(\hat{\mathbf{V}}_1 \times \hat{\mathbf{V}}_2\right)\right)}{\left|\hat{\mathbf{V}}_1 \times \hat{\mathbf{V}}_2\right|} \tag{3.17}$$

$$\hat{\mathbf{s}}_1 = \hat{\mathbf{W}}_1 \qquad \hat{\mathbf{s}}_2 = \frac{\left(\hat{\mathbf{W}}_1 \times \hat{\mathbf{W}}_2\right)}{\left|\hat{\mathbf{W}}_1 \times \hat{\mathbf{W}}_2\right|} \qquad \hat{\mathbf{s}}_3 = \frac{\left(\hat{\mathbf{W}}_1 \times \left(\hat{\mathbf{W}}_1 \times \hat{\mathbf{W}}_2\right)\right)}{\left|\hat{\mathbf{W}}_1 \times \hat{\mathbf{W}}_2\right|} \tag{3.18}$$

Existe somente uma matriz $\mathbf{A}$ que roda $\hat{\mathbf{r}}_i$ para $\hat{\mathbf{s}}_i$, ou seja, que satisfaz a:

$$A\hat{\mathbf{r}}_i = \hat{\mathbf{s}}_i \qquad i = 1, 2, 3 \tag{3.19}$$

O desenvolvimento integral das equações do algoritmo, bem como o cálculo das matrizes de covariâncias do TRIAD podem ser encontrados em [FGJ06].

Capítulo 4 – O Filtro de Kalman no Simulador

4.1– Introdução

O estágio final do processo de simulação é alimentar o Filtro de Kalman, que tem uma classe específica como mencionado no capítulo anterior, e retorna os quatérnions estimados, além da matriz de covariância estimada.

Neste estágio os quatérnions gerados são armazenados em arquivo formato texto (txt) e seus valores são comparados com os valores dos quatérnions teóricos, através de gráficos gerados em MATLAB.

A seguir o princípio de funcionamento do Filtro de Kalman será descrito.

4.2– Princípios de funcionamento do filtro

4.2.1– Histórico

Em 1960, R. E. Kalman [KAL60] forneceu um modo alternativo de formular o problema de filtragem dos mínimos quadrados usando métodos de estado-espaço. Engenheiros, especialmente no campo da navegação, viram rapidamente a técnica de Kalman como uma solução prática para um número de problemas que eram difíceis de serem resolvidos usando o método de Wiener [KAY02]. As duas principais características de Kalman para formulação e solução de problemas são a modelagem vetorial de processos aleatórios sob a consideração e processamento recursivo da medida de ruído da entrada. Em seguida, procederemos aos detalhes da solução recursiva do problema de filtragem discreta de dados.

4.2.2– Formulação

O artigo de R. E. Kalman descreve a solução recursiva para o problema de filtragem discreta linear de dados como publicado em 1960. Neste mesmo tempo,

avanços na tecnologia de computação digital tornaram possível considerar a implementação de sua solução recursiva em aplicações numéricas em tempo real.

Assumindo que iremos estimar um processo estocástico, ele poderá ser modelado da seguinte forma:

$$\mathbf{x}_{k+1} = \phi_k \mathbf{x}_k + \mathbf{w}_k \tag{4.1}$$

A observação (medida) do processo é assumida para ocorrer em pontos discretos do tempo em concordância com o relacionamento linear:

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{4.2}$$

A elaboração da notação e dos vários termos das equações (4.1) e (4.2) são:

$\mathbf{x}_k$ = vetor de estado (n x 1) do processo no tempo t_k

ϕ_k = matriz (n x n) relacionando $\mathbf{x}_k$ a $\mathbf{x}_{k+1}$ na ausência de uma função forçando (se $\mathbf{x}_k$ é uma amostra de processo contínuo, ϕ_k é uma matriz de transição de estado.)

$\mathbf{w}_k$ = vetor (n x 1) – assumido para ser uma seqüência branca com estrutura de covariância conhecida

$\mathbf{z}_k$ = vetor de medida (m x 1) no tempo t_k

$\mathbf{H}_k$ = matriz (m x n) dando uma conexão ideal (sem ruído) entre o vetor de medida e o de estado no tempo t_k

$\mathbf{v}_k$ = medida de erro (m x 1) – assumido para ser uma seqüência com estrutura de covariância conhecida e tendo correlação cruzada igual a zero com $\mathbf{w}_k$ seqüências

As matrizes de covariância para os vetores $\mathbf{w}_k$ e $\mathbf{v}_k$ são dadas por:

$$E\left[\mathbf{w}_k \mathbf{w}_i^T\right] = \begin{Bmatrix} \mathbf{Q}_k, & i=k \\ 0, & i \neq k \end{Bmatrix} \tag{4.3}$$

$$E\left[v_k v_i^T \right] = \begin{Bmatrix} \mathbf{R}_k, & i=k \\ 0, & i \neq k \end{Bmatrix} \tag{4.4}$$

$$E\left[\mathbf{w}_k \mathbf{v}_i^T \right] = 0, \quad \text{para todo k e i} \tag{4.5}$$

Assume-se neste ponto que se tem uma estimativa inicial do processo em algum ponto no tempo t_k, e esta estimação é baseada em todo conhecimento sobre o processo anterior a t_k. Esta estimação a anterior será denotada como $\hat{\mathbf{x}}_k^-$ onde o "circunflexo" denota estimado, e o "negativo" significa a melhor estimativa anterior à assimilação da medida em t_k. (Note que o negativo como usado aqui não é relacionado em qualquer modo com a notação negativa usada em fatorização espectral.) Também assumimos que conhecemos a matriz de covariância dos erros associada com $\hat{\mathbf{x}}_k^-$. Isto é, definimos o erro estimado como sendo

$$e_k^- = \mathbf{x}_k - \hat{\mathbf{x}}_k^- \tag{4.6}$$

e a matriz de covariância dos erros é

$$\mathbf{P}_k^- = E\left[\mathbf{e}_k^- \mathbf{e}_k^{-T} \right] = E\left[\left(\mathbf{x}_k - \hat{\mathbf{x}}_k^- \right) \left(\mathbf{x}_k - \hat{\mathbf{x}}_k^- \right)^T \right] \tag{4.7}$$

Em muitos casos, começa-se o problema de estimação com nenhuma medida anterior. Assim, neste caso, se a média do processo é zero, a estimação inicial é zero, e a matriz de covariância dos erros associada é somente a matriz de covariância do próprio $\mathbf{x}$.

Assumindo a estimação anterior $\hat{\mathbf{x}}_k^-$, em seguida é necessário usar a medida $\mathbf{z}_k$ e melhorar a estimação anterior. Assim, utilizando uma mistura linear de medida de ruído e a estimação anterior em concordância com a equação

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-) \tag{4.8}$$

onde

$\hat{\mathbf{x}}_k$ = estimação atualizada

$\mathbf{K}_k$ = fator misturado (ganho do filtro)

A justificativa da forma especial da equação (4.8) será adiada até a próxima seção. O problema agora é encontrar o fator particular $\mathbf{K}_k$ que renderá uma atualização da estimação como ótima em algum sentido. Para este fim, a primeira expressão para a matriz de covariância dos erros associada com a estimação atualizada (posterior).

$$\mathbf{P}_k = E\left[\mathbf{e}_k\mathbf{e}_k^T\right] = E\left[\left(\mathbf{x}_k - \hat{\mathbf{x}}_k\right)\left(\mathbf{x}_k - \hat{\mathbf{x}}_k\right)^T\right] \tag{4.9}$$

Em seguida, substituindo a equação (4.2) na equação (4.8) e então substituindo a expressão resultante for $\hat{\mathbf{x}}_k$ na equação (4.9). O resultado é

$$\mathbf{P}_k = E\left\{\left[\left(\mathbf{x}_k - \hat{\mathbf{x}}_k^-\right) - \mathbf{K}_k\left(\mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-\right)\right]\left[\left(\mathbf{x}_k - \hat{\mathbf{x}}_k^-\right) - \mathbf{K}_k\left(\mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-\right)^T\right]\right\} \tag{4.10}$$

Então, realizando a esperança indicada e anotando $\left(\mathbf{x}_k - \hat{\mathbf{x}}_k^-\right)$ é estimação do erro anterior que é descorrelacionado com a medida de erro $\mathbf{v}_k$, temos

$$\mathbf{P}_k = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\mathbf{P}_k^-\left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T \tag{4.11}$$

Aqui a equação (4.11) é perfeitamente uma expressão geral para atualização da matriz de covariância do erro, e se aplica para qualquer ganho $\mathbf{K}_k$.

Retornando ao problema de otimização, é desejado encontrar o $\mathbf{K}_k$ particular que minimiza os termos individuais ao longo da diagonal principal de $\mathbf{P}_k$, porque estes

termos representam a estimação das variâncias dos erros para os elementos do vetor de estado que está sendo estimado. A otimização pode ser feita de várias maneiras. Isto será feito usando uma aproximação de cálculo diferencial direto, e para fazê-lo são necessárias duas fórmulas de matriz diferencial. Elas são

$$\frac{d\left[traço\left(\mathbf{AB}\right)\right]}{d\mathbf{A}} = \mathbf{B}^T, \quad (\mathbf{AB} \text{ devem ser quadradas}) \tag{4.12}$$

$$\frac{d\left[traço\left(\mathbf{ACA}^T\right)\right]}{d\mathbf{A}} = 2\mathbf{AC}, \quad (\mathbf{C} \text{ deve ser simétrico}) \tag{4.13}$$

onde a derivada de um escalar com respeito a matriz é definido por

$$\frac{ds}{d\mathbf{A}} = \begin{bmatrix} \frac{ds}{da_{11}} & \frac{ds}{da_{12}} & \cdots \\ \frac{ds}{da_{21}} & \frac{ds}{da_{22}} & \cdots \\ \vdots & & \end{bmatrix} \tag{4.14}$$

Agora utilizando temporariamente os subscritos na expressão $\mathbf{P}_k$, equação (4.11), e expandindo-a e reescrevendo na forma:

$$\mathbf{P}_k = \mathbf{P}^- - \mathbf{KHP}^- - \mathbf{P}^-\mathbf{H}^T\mathbf{K}^T + \mathbf{K}\left(\mathbf{HP}^-\mathbf{H}^T + \mathbf{R}\right)\mathbf{K}^T \tag{4.15}$$

Os segundo e terceiro termos são lineares em $\mathbf{K}$ e o quarto termo é quadrático em $\mathbf{K}$. As duas fórmulas de matrizes diferenciais podem ser aplicadas à equação (4.15). É desejável minimizar o traço de $\mathbf{P}$ porque ele é a soma dos erros quadrados médios nas estimativas de todos os elementos do vetor de estado. Aqui pode ser usado o argumento que erros quadrados médios individuais são minimizados também quando o total é minimizado, levando em consideração que se tem suficiente grau de liberdade na variação de $\mathbf{K}$, que é feito neste caso. Em seguida, será diferenciado o traço de $\mathbf{P}$ com relação à $\mathbf{K}$, e pode-se notar que o traço de $\mathbf{P}^-\mathbf{H}^T\mathbf{K}^T$ é igual ao traço de sua transposta $\mathbf{KHP}^-$. O resultado é

$$\frac{d\left[traço\left(\mathbf{P}\right)\right]}{d\mathbf{K}} = 2\left(\mathbf{HP}^{-}\right)^{T} + 2\mathbf{K}\left(\mathbf{HP}^{-}\mathbf{H}^{T} + \mathbf{R}\right) \tag{4.16}$$

Então, fazendo a derivada igual a zero e resolvendo para o ganho ótimo. O resultado é (com os subscritos re-inseridos)

$$\mathbf{K}_k = \mathbf{P}_k^{-}\mathbf{H}_k^{T}\left(\mathbf{H}_k\mathbf{P}_k^{-}\mathbf{H}_k^{T} + \mathbf{R}_k\right)^{-1} \tag{4.17}$$

Este particular $\mathbf{K}_k$, nomeado por mínima estimação dos erros quadrados médios, é chamado de ganho de Kalman.

A matriz de covariância associada à estimação ótima pode ser calculada agora. Referenciando à equação (4.11), tem-se

$$\mathbf{P}_k = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\mathbf{P}_k^{-}\left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k^{T}\right) + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^{T} \tag{4.18}$$

$$= \mathbf{P}_k^{-} - \mathbf{K}_k\mathbf{H}_k\mathbf{P}_k^{-} - \mathbf{P}_k^{-}\mathbf{H}_k^{T}\mathbf{K}_k^{T} + \mathbf{K}_k\left(\mathbf{H}_k\mathbf{P}_k^{-}\mathbf{H}_k^{T} + \mathbf{R}_k\right)\mathbf{K}_k^{T} \tag{4.19}$$

Substituindo a expressão do ganho ótimo, equação (4.17), na equação (4.19) tem-se

$$\mathbf{P}_k = \mathbf{P}_k^{-} - \mathbf{P}_k^{-}\mathbf{H}_k^{T}\left(\mathbf{H}_k\mathbf{P}_k^{-}\mathbf{H}_k^{T} + \mathbf{R}_k\right)^{-1}\mathbf{H}_k\mathbf{P}_k^{-} \tag{4.20}$$

ou

$$\mathbf{P}_k = \mathbf{P}_k^{-} - \mathbf{K}_k\left(\mathbf{H}_k\mathbf{P}_k^{-}\mathbf{H}_k^{T} + \mathbf{R}_k\right)\mathbf{K}_k^{T} \tag{4.21}$$

ou

$$\mathbf{P}_k = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\mathbf{P}_k^{-} \tag{4.22}$$

Destas três expressões para $\mathbf{P}_k$, a última dada por (4.22) é a mais simples, assim é mais frequentemente usada que as outras. Novamente, nota-se que a equação (4.18) é a válida para qualquer ganho, sub-ótimo ou não, visto que a equação (4.20), (4.21), e (4.22) são válidas somente para o ganho (ótimo) de Kalman.

É possível ter uma média de assimilação das medidas em t_k pelo uso da equação (4.8) com $\mathbf{K}_k$ igual ao ganho de Kalman dado em (4.17). Nota-se que é preciso $\hat{\mathbf{x}}_k^-$ e $\mathbf{P}_k^-$ para realizar isto, e pode ser antecipada uma necessidade similar no próximo passo em ordem para tornar ótimo o uso da medida z_{k+1}. A estimação atualizada $\hat{\mathbf{x}}_k^-$ é facilmente projetada adiante via matriz de transição. Assim tem-se

$$\hat{\mathbf{x}}_{k+1}^- = \phi_k \hat{\mathbf{x}}_k \tag{4.23}$$

A matriz de covariância do erro associada à $\hat{\mathbf{x}}_{k+1}^-$ é obtida formando primeiro a expressão para o erro anterior

$$\begin{aligned} \mathbf{e}_{k+1}^- &= \mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1}^- \\ &= \left(\phi_k \mathbf{x}_k + \mathbf{w}_k\right) - \phi_k \hat{\mathbf{x}}_k \\ &= \phi_k \mathbf{e}_k + \mathbf{w}_k \end{aligned} \tag{4.24}$$

Nota-se que $\mathbf{w}_k$ e $\mathbf{e}_k$ tem correlação cruzada igual a zero, porque $\mathbf{w}_k$ é processo com ruído para os passos seguintes de t_k. Assim a expressão de $\mathbf{P}_{k+1}^-$ pode ser escrita como

$$\begin{aligned} \mathbf{P}_{k+1}^- &= E\left[\mathbf{e}_{k+1}^- \mathbf{e}_{k+1}^{T-}\right] = E\left[\left(\phi_k \mathbf{e}_k + \mathbf{w}_k\right)\left(\phi_k \mathbf{e}_k + \mathbf{w}_k\right)^T\right] \\ &= \phi_k \mathbf{P}_k \phi_k^T + \mathbf{Q}_k \end{aligned} \tag{4.25}$$

Agora é necessário quantificar o tempo t_{k+1}, e a medida de $\mathbf{z}_{k+1}$ pode ser assimilada somente com o passo anterior.

As equações (4.8), (4.17), (4.22), (4.23), e (4.25) compreendem as equações do filtro de Kalman recursivo. A partir de um laço completo no algoritmo, ele pode

continuar ao infinito. As equações pertinentes e seqüência de passos são apresentadas na figura abaixo. Este é o resumo daquele que é conhecido como Filtro de Kalman.

Na verdade, o Filtro de Kalman é um algoritmo computacional para processar medidas discretas (a entrada) até a estimativa ótima (a saída).

Os projetos anteriores ao Filtro de Kalman eram frequentemente heurísticos. Wiener então surgiu em cena nos anos de 1940 e acrescentou um tipo mais sofisticado de problema de filtragem. O resultado final de sua solução foi um filtro com funções de peso ou uma função de transferência correspondente no domínio da frequência. Implementação em termos de elementos elétricos foi deixada para os projetistas. A versão de amostra de dados do problema de Wiener tornou-se sem solução (pelo menos no sentido prático) até o artigo de Kalman em 1960. Então, esta apresentação parecia ser abstrata num primeiro momento, engenheiros logo disseram que seu trabalho apresentava uma solução prática para um número de problemas de filtragem sem solução, especialmente no campo da navegação. Mais de 40 anos se passaram desde o artigo original de Kalman, e ainda existem numerosos artigos atualmente idealizando novas aplicações e variações do filtro de Kalman básico.

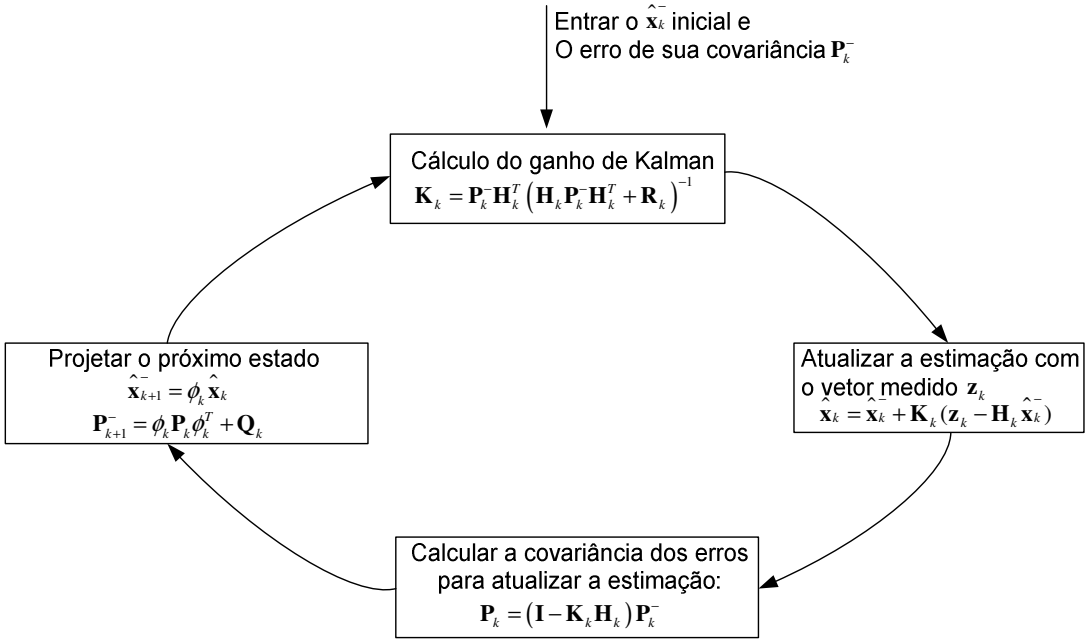


Figura 4.1 – Fluxograma do Filtro de Kalman

Capítulo 5 – Linguagem de Programação e Apresentação Gráfica

5.1– Introdução

Para o desenvolvimento do simulador, foi necessário escolher uma linguagem de programação amplamente encontrada no mercado, além de possuir recursos como programação orientada a objetos e suas derivações (polimorfismo, sobrecarga de operadores, herança, etc).

Assim, uma das opções atuais é a linguagem C++, que além de atender os requisitos anteriormente mencionados, também foi utilizada no desenvolvimento do hardware para Determinação de Atitude.

5.2– Uma breve descrição da linguagem C++

5.2.1- Linguagens de programação

Uma linguagem de programação é um conjunto de instruções e uma série de convenções especificamente projetadas para ordenar ao computador o que deve ser feito.

Quando escolhida uma linguagem de programação para criar um projeto, diferentes considerações devem ser realizadas. Primeiro deve-se decidir qual o nível da linguagem de programação é conhecido. O nível determina o quão próximo do hardware à linguagem de programação está. Nas linguagens de baixo nível, as instruções são escritas pensando diretamente na interface com o hardware, enquanto nas linguagens de "alto nível", o código escrito é mais abstrato ou conceitual.

Geralmente, código de alto nível é mais portátil, o que significa que pode trabalhar em muitas máquinas diferentes e com pouco número de alterações, entretanto linguagem de baixo nível é limitada a peculiaridades do hardware para o qual foi escrita.

A programação de baixo ou alto nível é escolhida para um projeto específico depende do tipo de programa que será desenvolvido. Por exemplo, quando um driver de hardware é desenvolvido para um sistema operacional, obviamente um nível mais baixo de programação é usado. Quando o projeto envolve uma grande aplicação, frequentemente uma linguagem de alto nível é escolhida, ou uma combinação de partes críticas são escritas em baixo nível e as demais em alto nível.

Embora existam linguagens que são escritas em baixo nível, como Assembly, onde conjuntos de instruções são adaptados para cada máquina, também existem linguagens de alto nível, como Java, que é projetada para ser totalmente independente da plataforma onde será executada. A linguagem C++ está em posição intermediária, desde que possa interagir diretamente com o hardware quase sem limitações, e pode abstrair camadas inferiores e pode trabalhar como uma das mais poderosas linguagens de alto nível.

No projeto do simulador foi utilizada a Linguagem C++, versão 6.0 (Build 10.157), da Borland Software Corporation. Ela foi escolhida por já existir um conhecimento anterior na sua estrutura, sintaxe e ambiente de desenvolvimento (API).

5.2.2- Características da Linguagem C++

5.2.2.1- Programação orientada a objetos

A possibilidade de orientar a programação a objetos permite que o programador projete aplicações com maior comunicação entre objetos que uma seqüência estruturada de códigos. Em suma, ela permite a reutilização de código de uma forma mais lógica e produtiva.

5.2.2.2- Portabilidade

O mesmo código em C++ pode ser compilado em diferentes tipos de computador e sistema operacional sem fazer qualquer mudança. C++ é a linguagem de programação mais usada e portátil do mundo.

5.2.2.3- Linhas de programação

Código escrito em C++ é muito menos extenso quando comparado a outras linguagens, desde o uso de caracteres especiais é preferido como palavras-chave, minimizando algum esforço para o programador.

5.2.2.4- Programação modular

O corpo de uma aplicação em C++ pode ser feito de vários arquivos de código-fonte que são compilados separadamente e então são linkados (ligados) juntamente. Em suma, esta característica permite linkar o código C++ com código produzido em outras linguagens, tais como Assembler ou C.

5.2.2.5- Compatibilidade

C++ é compatível com a linguagem C. Qualquer código escrito em C pode facilmente ser incluído em um programa C++ sem fazer qualquer mudança.

5.2.2.6- Velocidade

O código resultante de uma compilação em C++ é muito eficiente, operando tanto em linguagem de alto nível quanto baixo nível, reduzindo o tamanho do código.

5.3– Biblioteca para apresentação gráfica – OpenGL

5.3.1– Introdução

OpenGL é um ambiente para desenvolvimento portátil de aplicações gráficas em 2D e 3D. Desde sua introdução em 1992, OpenGL tornou-se a interface de programação de aplicação (API) para gráficos 2D e 3D, mais amplamente utilizada na indústria, trazendo milhares de aplicações para uma ampla variedade de plataformas de computadores. OpenGL traz inovação e velocidade no desenvolvimento de aplicações para incorporar um grande conjunto de renderização, mapeamento de texturas, efeitos especiais, e outras potentes funções de visualização. Desenvolvedores podem usufruir do poder do OpenGL para criar aplicações para praticamente todas as plataformas de estações de trabalho e computadores de mesa.

5.3.2– Alta Qualidade Visual e Performance

Qualquer aplicação computacional requer máxima performance de animação 3D, para simulação visual CAD, explorando alta qualidade e capacidades de alta performance. Estas capacidades permitem desenvolvedores em diversos mercados, tais como transmissão a cabo, CAD/CAM/CAE, entretenimento, imagens médicas, e realidade virtual para produzir e apresentar gráficos em 2D e 3D.

Entre as vantagens no desenvolvimento de aplicações utilizando OpenGL, encontram-se: criação de um consórcio independente padronizando a plataforma, estabilidade, confiança e portabilidade, escalabilidade, facilidade de uso e ampla documentação.

5.3.3– Fluxograma da visualização em OpenGL

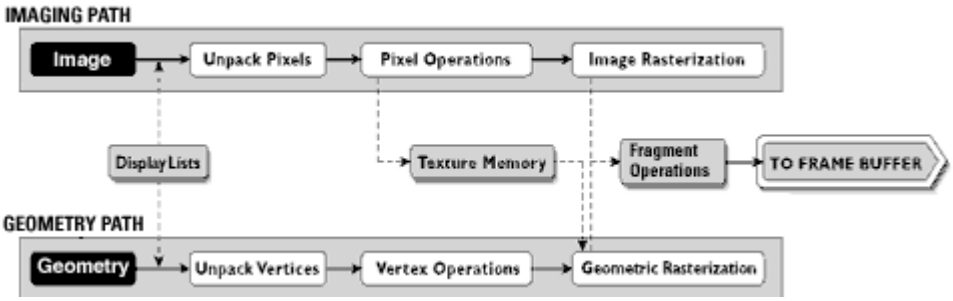


Figura 5.1 – Fluxograma do OpenGL

5.3.4– OpenGL no simulador de determinação de atitude

Como foi mencionada nas seções sobre sistemas de coordenadas na determinação de atitude, a álgebra de quatérnions é fundamental para acompanhamento das rotações do corpo que está sendo simulado.

Assim, utilizando uma linguagem de programação de alto nível em conjunto com uma biblioteca de apresentação gráfica que suporte esta aritmética, a simulação terá maior eficiência, e em trabalho posterior, poderá ser utilizada até mesmo na caracterização dos sensores que fazem parte do hardware de determinação de atitude.

A biblioteca OpenGL possui funções específicas para criar objetos em três dimensões, como cubos, polígonos, esferas, etc. Além de rotacinar ou transladar estes objetos pela tela, com velocidade e eficiência.

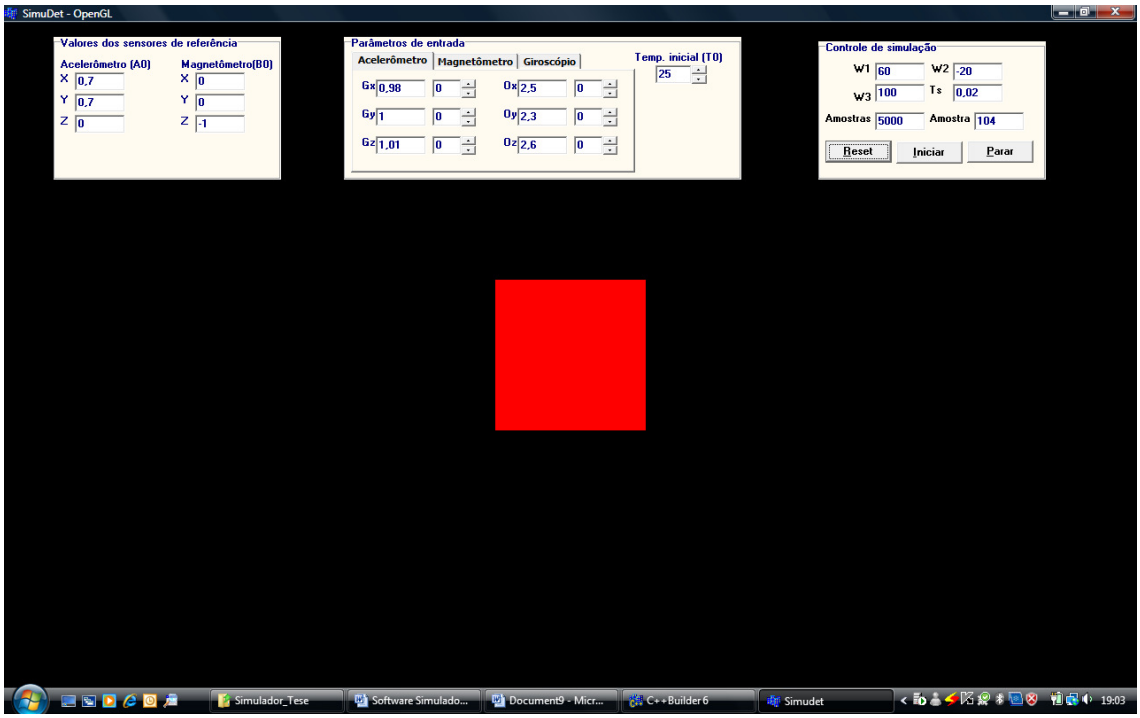


Figura 5.2 – Tela principal do simulador

5.4– Funcionalidades do Software Desenvolvido

Para aproximar o modelo do simulador ao máximo do hardware, uma idéia é permitir que o usuário possa alterar os parâmetros padrão (default) dos acelerômetros e magnetômetros.

Estes valores estão separados em duas caixas:

- Valores dos sensores de referência: nesta caixa, o simulador carrega os valores iniciais para os vetores do acelerômetro (A0) e magnetômetro (B0);
- Parâmetros de entrada: nesta caixa são inicializados os valores de tensões de referências (ganhos) e offsets de cada um dos sensores, além da temperatura inicial (T0).

Além destes parâmetros, o software permite que o usuário entre com os valores das componentes da velocidade angular W = (W1, W2, W3), a quantidade de amostras e a taxa de amostragem em segundos.

O simulador também possui alguns botões para sua execução: Reset posiciona o cubo na origem, além de zerar o número de amostras; Iniciar dá início à simulação; Parar que pára a simuação.

Quando o botão "Iniciar" é clicado, serão exibidos três cubos na tela, o primeiro mostrando o movimento do cubo ideal (sem ruídos), o segundo mostrando o movimento do cubo após o algoritmo TRIAD e o terceiro e último cubo, que mostra o movimento do cubo após o Filtro de Kalman.

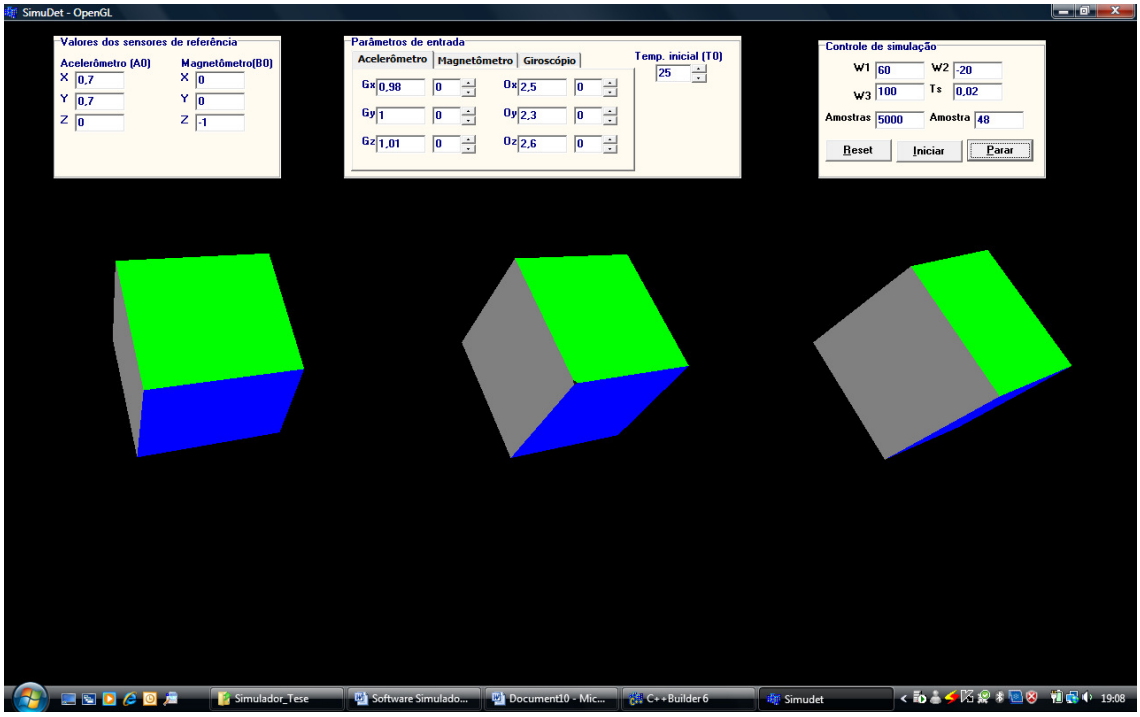


Figura 5.3 – Simulação contendo três cubos

Durante a simulação, todos os parâmetros de entrada podem ser alterados em tempo de execução, e isto é possível utilizando o conceito de Thread, da linguagem C++. A partir do momento que o botão "Iniciar" é clicado, o simulador dispara um componente Timer, e a cada intervalo deste timer, a thread "ThreadSimula" é carregada recebendo como parâmetros de entrada o quatérnion anterior, taxa de amostragem, ganhos e offsets dos sensores.

Desta maneira, alterando os parâmetros de entrada durante a simulação, é possível acelerar o cubo e ver as consequências no sistema.

5.5– Descrição dos principais itens do código fonte

Como mencionado anteriormente, a vantagem de se desenvolver o simulador em linguagem de programação orientada a objetos, possibilita a utilização vários de seus recursos, como por exemplo: herança, polimorfismo, sobrecarga de operadores, etc.

Assim, foram criadas as seguintes classes:

CFila():

O simulador realiza uma filtragem tipo passa-baixa média móvel, que precisa de uma fila circular, então, existem dois métodos, inicializaFila() e insereFila(), responsáveis para criar uma fila vazia e alimentá-la, respectivamente.

CKalman():

Uma das principais classes de todo o simulador, responsável pela Filtragem de Kalman.

Ela recebe como parâmetros de entrada, os vetores de referência, desvio padrão dos sensores, quatérnions estimados em estado anterior, matriz de covariância do estado anterior.

Entre os principais métodos, existem:

Inicializa(), executado uma única vez, para inicializar os parâmetros e calcular as matrizes necessárias nos outros estágios do filtro;

KalmanProp(), método necessário para calcular a propagação do filtro, recebe como parâmetros de entrada, o estado estimado, matriz de covariância estimada, matriz de ruído dos giros, vetor deslocamento angular, intervalo de propagação, e retorna o estado propagado e matriz de covariância reduzida propagada.

KalmanEst(), necessário para realizar a estimação do filtro, tem como parâmetros de entrada o estado propagado, matriz de covariância propagada, quatérnion de atitude (proveniente do QUEST), matriz de covariância e retorna, o estado estimado, covariância reduzida estimada.

COperacoes():

Classe que executa algumas operações necessárias para o filtro, e possui os seguintes métodos:

FiltroMM(), responsável pela geração dos valores do filtro passa-baixa média-móvel;

MultiplicaMV(), multiplica uma matriz por um vetor retornando outro vetor;

MultiplicaVV(), executa a multiplicação entre dois vetores, resultando uma matriz;

CQuaternion():

Classe responsável no cálculo de operações envolvendo quatérnions. Possui os métodos:

Conjq() retorna o conjugado de um quatérnion passado como parâmetro de entrada;

Mat2q() recebe como parâmetro de entrada, uma matriz e retorna um quatérnion;

q2Mat(), tem como parâmetro de entrada um quatérnion e retorna uma matriz;

norm(), normaliza o valor de entrada, que poderá ser um vetor ou um quatérnion;

RodaQuaternion() recebe como parâmetros de entrada, um quatérnion, um vetor e a taxa de amostragem, e retorna o quatérnion rodado sobre o vetor;

CTriad():

Classe que executa o algoritmo TRIAD. Ela recebe os parâmetros iniciais, como por exemplo, os valores dos vetores de referência e seus respectivos ruídos, além dos valores dos vetores no corpo. Ela retorna a matriz de covariância e matriz de atitude. Estes valores calculados serão utilizados na classe **CKalman**.

CSensor():

Esta classe é utilizada na caracterização dos sensores. Recebe como parâmetros de entrada, os valores de ganho, tensão de referência e bias, além do tipo do sensor (acelerômetro, magnetômetro ou giro). Dentre os métodos, tem um responsável para fazer a conversão analógico-digital, digital-analógico e função de transferência.

CThread():

Esta classe é carregada a cada intervalo programado no componente Timer. Ela recebe como parâmetros de entrada os valores dos vetores dos sensores de referência, ganhos e offsets dos sensores e temperatura. Os principais métodos são: Inicializa() – é carregado somente uma vez para inicializar os parâmetros do filtro de Kalman, Execute() – executa todo o processo de simulação e carrega as demais classes como TRIAD, Csensor, etc.

5.6– Detalhando a classe cKalman()

A classe responsável para simulação do Filtro de Kalman possui basicamente três métodos: Inicializa(), KalmanProp() e KalmanEst(). Estes métodos serão descritos em detalhes abaixo:

5.6.1– Método Inicializa()

Este método deve ser executado somente uma única vez para cada uma das instâncias da classe cKalman(). Aqui as variáveis serão inicializadas. Estas variáveis são:

SigMag, SigAcc, SigGyro, SigDGyro – Desvio padrão das medidas dos sensores

IT – número de iterações

x_est[] – array que contém os valores estimados (quatérnion e matriz de covariância)

x_prop[] – array que contém os valores propagados (quatérnion e matriz de covariância)

q_Tri[] – quatérnion que conterá os valores dos quatérnions após executação do algoritmo TRIAD

Q[] – matriz de ruído do Giro, possui dimensão 6 x 6

GT[] e QT[] – matrizes constantes, possuem dimensões 3 x 3 e 6 x 6 respectivamente

PT_est[] – matriz de covariância estimada, possui dimensão 6 x 6

H[] – matriz de leitura, com dimensão 4 x 4

std_desv[] – array contendo o valor do desvio padrão, com dimensão 7

Além de atribuir os valores iniciais a todos estas variáveis, este método também inicializa o primeiro valor dos vetores x_est[] e x_prop[], ambos conterão o valor de (0, 0, 0, 1) para os elementos que correspondem ao quatérnion e (0, 0, 0) para os elementos que correspondem covariância reduzida.

5.6.2– Método KalmanProp() [FGJ06]

Este método é responsável pela propagação dos estados, e receberá como parâmetros de entrada:

x_est[], PT_est, QT, Th, Ts – as variáveis x_est[], PT_est[] e QT já foram definidas no método Inicializa(), Th é o vetor deslocamento angular e Ts é o intervalo de propagação.

Neste método os giroscópios são medidos e sua saída é subtraída do bias, e é integrada no período de propagação.

Em seguida a matriz de atualização do quatérnion é atualizada. Neste momento, teremos os valores do vetor x_prop para o próximo estado, armazenando o quatérnion propagado e o bias propagado.

Por último, é realizada a propagação da matriz de covariância, ou seja, o valor da variável PT_prop é calculado. Esta matriz possui dimensão 6 x 6.

5.6.3– Método KalmanEst()

Este método é responsável pela estimação dos valores de quatérnion e matriz de covariância, e estes valores realimentarão o algoritmo, fechando o loop (laço).

Este método recebe como entrada os seguintes parâmetros:

x_prop[], PT_prop[], q_Tri[], P_Tri[] – as duas primeiras variáveis já foram mencionadas na seção anterior, e q_Tri[] e P_Tri[] são os valores de quatérnion e covariância, obtidos a partir da execução do algoritmo Tríade ou QUEST.

A partir destes parâmetros, o método separa as componentes do estado, ou seja, separa os valores de x_prop[] em q_prop[] e b_prop[] e monta a matriz E(Ksi).

Na seqüência monta a matriz S, H e HT, separa as covariâncias dos estados, que recebem o nome de Ptt_prop[] e Pbt_prop[].

Em seguida, calcula o novo ganho (reduzido), que é chamado de KT, atualiza a covariância reduzida, PT_esti. Calcula a covariância estendida, P_esti e o ganho estendido K.

Por último, atualiza o estado x_esti[] e renormaliza o quatérnion.

Assim, os valores de retorno do método, serão:

x_esti[] – estado estimado

PT_esti – covariância reduzida estimada

P_esti – covariância estendida estimada

5.6.4– Método Executa()

Este método é responsável pela execução dos outros métodos necessários para o cálculo dos valores estimados, ou seja, ele executa KalmanProp() e KalmanEst().

Antes que este método seja executado, o programa principal tem que passar os parâmetros, tais como: bias dos giros, valores dos vetores de referência e seus respectivos ruídos.

A partir destes parâmetros, o método realiza a simulação da leitura dos sensores e de seus ruídos, através do método GetMagAccOuts(). Em seguida, executa o método GetGyroOut(), para simular os valores dos Gyros.

Na seqüência, calcula o deslocamento angular (vetorial), gerando o vetor Th[].

Em seguida, executa o método KalmanProp(), calcula os valores de quatérnions a partir do algoritmo TRIAD e executa o método KalmanEst().

Por último, muda a atitude, fazendo a integração para o próximo quatérnion verdadeiro e retorna como saída o quatérnions estimado.

Neste ponto, o objeto do programa principal, objKalman, tem o valor do quatérnion estimado, e já pode ser realimentado para calcular o próximo quatérnion, e assim sucessivamente, até completar o número de amostras a serem simuladas.

O diagrama de blocos abaixo, mostra este loop para executar a classe cKalman().

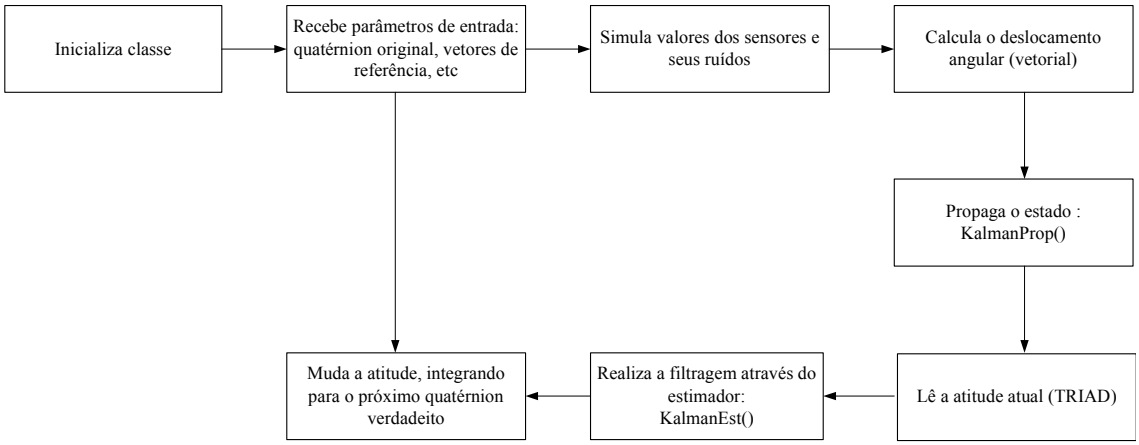


Figura 5.4 – Fluxograma principal do simulador

Capítulo 6 – O Procedimento de Auto-calibração

6.1– Introdução

Como foi demonstrado nos capítulos anteriores, um dos problemas para a determinação de atitude é a questão dos ruídos nos parâmetros de ganho e offset em cada um dos sensores. E ao longo do tempo, estes ruídos obrigam que o sistema seja calibrado em períodos da ordem de horas.

Assim, uma alternativa para resolver este problema é lembrar que o módulo do vetor gravidade sempre será 1g em condições de movimentos quasi-estáticos. Em contraste ao procedimento de calibração de acelerômetros uniaxiais, o procedimento de auto-calibração não requer nenhum conhecimento da orientação atual do acelerômetro triaixial e requer somente movimentos aleatórios quasi-estáticos.

6.2 Teoria

Em situação estática, a tensão de saída de um acelerômetro uniaxial é medida pelo ângulo θ [rad] entre o eixo do sensor do dispositivo e a direção da gravidade. Os parâmetros de sensibilidade e offset de um acelerômetro uniaxial pode ser obtido aplicando dois ângulos diferentes ao dispositivo, então, são obtidas duas equações com duas incógnitas. Assim, para calibrar um acelerômetro triaxial é necessário realizar seis movimentos, obtendo seis equações com seis incógnitas. Este é o menor conjunto de equações para conhecer as três sensibilidades $s_x, s_y \; e \; s_z \left[Vg^{-1} \right]$ e offsets $o_x, o_y \; e \; o_z \left[V \right]$ do dispositivo.

O procedimento de calibração consiste em duas partes: um detetor de momentos quasi-estáticos e um estimador de parâmetros. O detetor de momentos quasi-estáticos é usado para obter uma coleção destes momentos dentro de uma coleção de tensões de saída. Isto é feito através da medida do módulo do vetor gravidade, quando este vetor

tem pequenas variações em torno do valor 1g. Esta tensão pode ser determinada com a seguinte equação:

$$v_{in,\det ection} = \sqrt{v_x^2 + v_y^2 + v_z^2} \tag{6.1}$$

Onde $v_x, v_y \; e \; v_z$ são as tensões de saída dos acelerômetros. Então, o detetor de momentos quasi-estáticos é aplicado a $v_{in,\det ection}[V]$. A tensão de saída resultante $v_{out,\det ection}[V]$ será dada por:

$$v_{out,\det ection} = LPF\left(REC\left(HPF\left(v_{in,\det ection}\right)\right)\right) \tag{6.2}$$

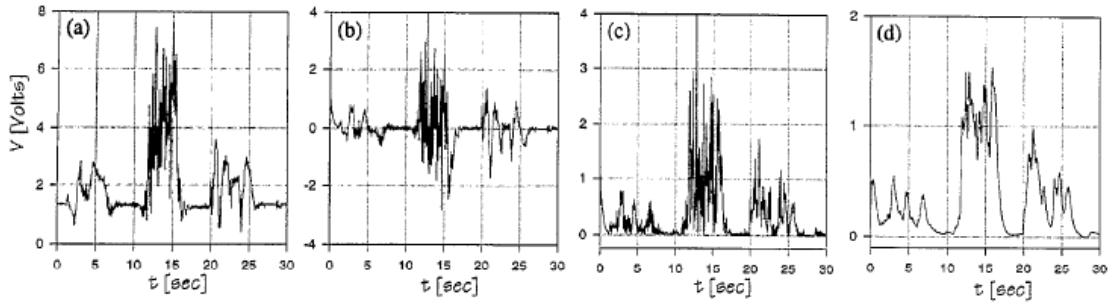


Figura 6.1 – Representação do funcionamento detetor de momentos estáticos: (a) o sinal composto do acelerômetro; (b) o sinal depois do filtro passa-alta; (c) o sinal depois do retificador; (d) o sinal depois do filtro passa-baixa.

O sinal de saída do filtro passa-baixa deve ser comparado á tensão de limiar. Quando este valor for menor que esta tensão de limiar, o movimento é estático, caso contrário será dinâmico.

A partir desta comparação de tensões, colecionam-se valores de tensões de movimentos quasi-estáticos, que serão utilizados em um procedimentos de estimação. Dentre estes procedimentos, pode-se utilizar o estimador de minímas variações, porque ele requer pouco ou nenhum conhecimento das funções densidades das variáveis aleatórias envolvidas no problema.

A tensão de saída em cada direção do acelerômetro triaxial pode ser descrita por:

$$v_{dir} = s_{dir} \cdot a_{dir} + o_{dir} \tag{6.3}$$

Assim:

$$\sqrt{\left(\frac{v_x - o_x}{s_x}\right)^2 + \left(\frac{v_y - o_y}{s_y}\right)^2 + \left(\frac{v_z - o_z}{s_z}\right)^2} = 1 \tag{6.4}$$

Isto é verdeiro no caso quasi-estático.

Assim, para realizar o procedimento de calibração, é necessário encontrar estes parâmetros através de um procedimento de minimização, e consequentemente estes valores de sensibilidade e offset serão utilizados para realimentar os valores dos sensores.

6.3 Conclusão

Uma das formas de encontrar valores aproximados para os parâmetros de sensibilidade e offset dos acelerômetros é encontrar os movimentos quasi-estáticos, colecionar estes valores, realizar a estimação e realimentar o sistema.

Capítulo 7 – O Simulador de Determinação de Atitude e Resultados Obtidos

7.1– Introdução

Neste capítulo, o simulador será detalhado em diagrama de blocos, bem com serão apresentados os gráficos contendo os valores em vários pontos do processo de simulação.

Estes gráficos foram gerados a partir do Matblab, e seus valores foram gerados em arquivos formato texto (txt), a partir do simulador.

7.2– Diagrama de blocos do simulador

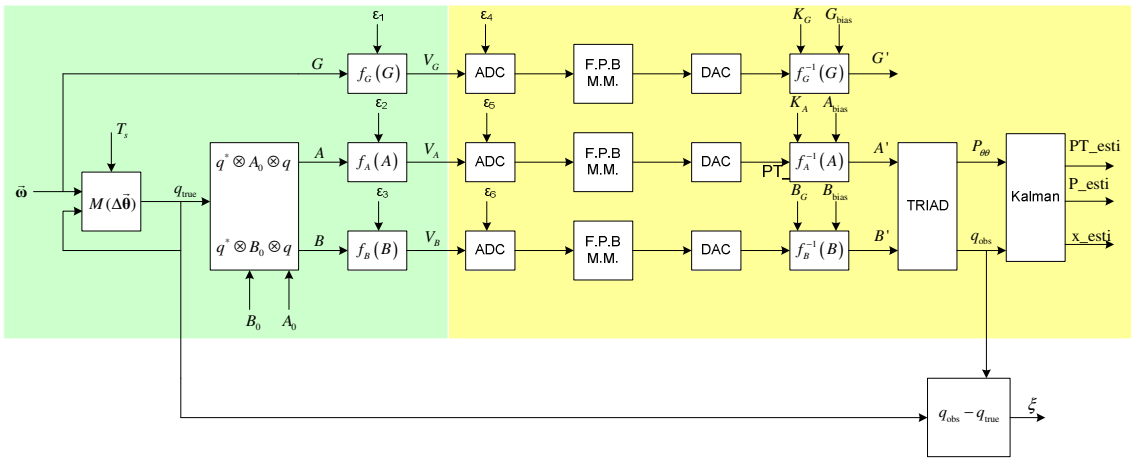


Figura 7.1 – Diagrama de blocos do simulador

Como é possível ver na figura, todo o sistema é dividido em dois subsistemas, ou seja, simulação dos sensores e processamento.

7.2.1– Subsistema simulador dos sensores

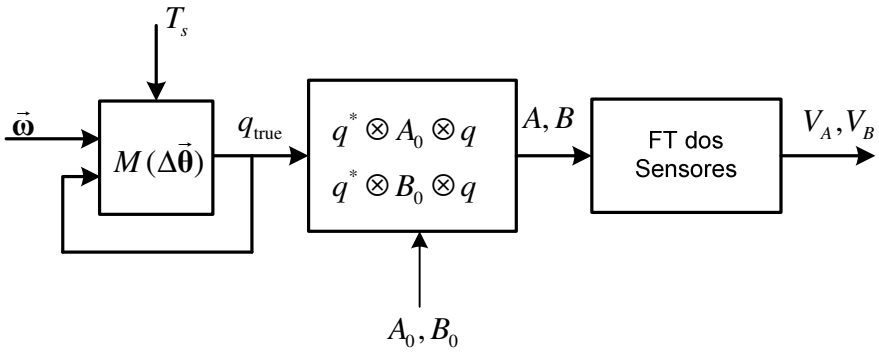


Figura 7.2 – Subsistema simulador dos sensores

Este subsistema é responsável pela simulação dos sensores, ou seja, a partir de uma seqüência de quatérnions gerados, é possível simular os sinais de tensões dos sensores. Três blocos compõem o subsistema: gerador de rotações, gerador de projeções, e funções de transferência dos sensores. Cada um destes blocos será detalhado a seguir.

7.2.1.1– Bloco gerador de rotações

A função do bloco é gerar a próxima posição angular representada por um quatérnion com base no quatérnion anterior e no incremento angular pela integral das velocidades angulares em um curto intervalo de tempo (o tempo de amostragem T_s), representado por:

$$\Delta\boldsymbol{\theta} = \int_t^{t+\Delta t} \boldsymbol{\omega}(t)\mathrm{d}t \tag{7.1}$$

A partir do incremento angular da equação (6.1) temos a matriz de transição para ajustar o quatérnion dada por:

$$\mathbf{M}(\Delta\boldsymbol{\theta}) = \cos(|\Delta\boldsymbol{\theta}|/2)\mathbf{I}_{4\times4} + \sin(|\Delta\boldsymbol{\theta}|/2)\frac{\boldsymbol{\Omega}_4(\Delta\boldsymbol{\theta})}{|\Delta\boldsymbol{\theta}|} \tag{7.2}$$

Onde $\mathbf{I}_{4\times4}$ é uma matriz identidade da dimensão indicada, e $\mathbf{\Omega}_4$ é o operador de distribuição que resulta em:

$$\mathbf{\Omega}_4(\Delta\boldsymbol{\theta}) = \begin{bmatrix} 0 & \theta_3 & -\theta_2 & \theta_1 \\ -\theta_3 & 0 & \theta_1 & \theta_2 \\ \theta_2 & -\theta_1 & 0 & \theta_3 \\ -\theta_1 & -\theta_2 & -\theta_3 & 0 \end{bmatrix} \tag{7.3}$$

Onde finalmente o próximo quatérnion, será dado por:

$$q(t+\Delta t) = \mathbf{M}(\Delta\boldsymbol{\theta})q(t) \tag{7.4}$$

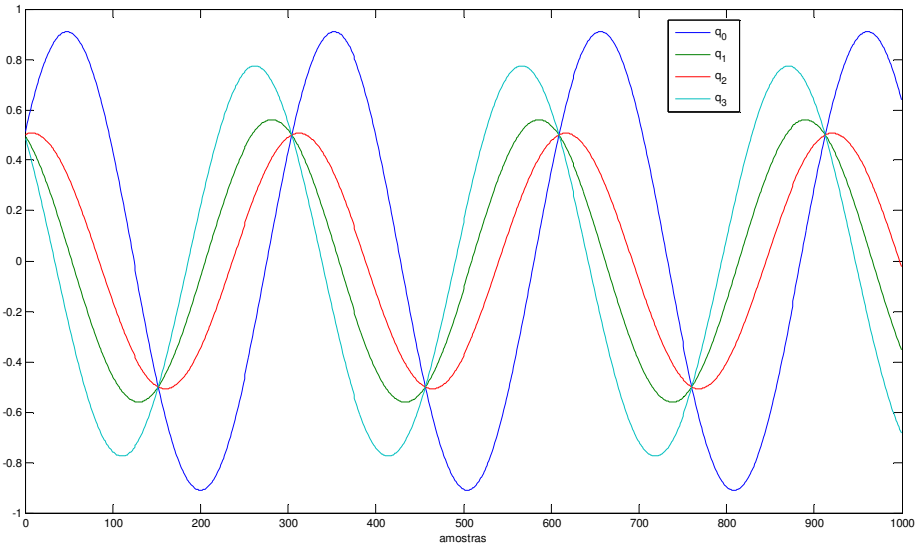


Figura 7.3 – Simulação

A figura mostra o resultado dos dados obtidos a partir de uma simulação com parâmetros de entrada de 1.000 amostras, velocidade angular $\boldsymbol{\omega} = (60, -20, 100)\ \frac{rad}{s}$, taxa de amostragem $T_s = 20\,ms$.

7.2.1.2– Bloco gerador de projeções

A partir dos quatérnions de entrada, o sistema executa o algoritmo de rotação, dado o valor das componentes dos sensores, medidas na origem, ou seja, $\mathbf{A_0}$ e $\mathbf{B_0}$, gerando assim vetores de leitura $\mathbf{A}$ e $\mathbf{B}$ na atitude instantânea do corpo. Esta transformação é feita aplicando a rotação por meio do produto quaterniônico dada por:

$$\begin{aligned} A &= q^* \otimes A_0 \otimes q \\ B &= q^* \otimes B_0 \otimes q \end{aligned} \tag{7.5}$$

7.2.1.3– Bloco função de transferência

Este bloco é responsável pela simulação da função de transferência dos sensores. Também neste ponto é adicionado o ruído característico para cada um dos tipos de sensores utilizados.

Este bloco é escrito de acordo com as informações fornecidas pelos fabricantes dos sensores. Os parâmetros mínimos que devem ser inseridos aqui são a sensibilidade do sensor, o off-set médio e o ruído rms. É possível representar minimamente a saída em tensão como:

$$V_A(b, G, \sigma) = GA + b + \sigma \cdot \text{randn} \tag{7.6}$$

Onde *G* é a matriz de ganho, *A* é o vetor contendo a grandeza física medida, *b* é o vetor off-set das medidas e σ é o desvio padrão determinado com base no ruído *rms*. A função randn gera um novo valor aleatório para ruído a cada amostragem.

A figura abaixo apresentará um gráfico da simulação dos acelerômetros.

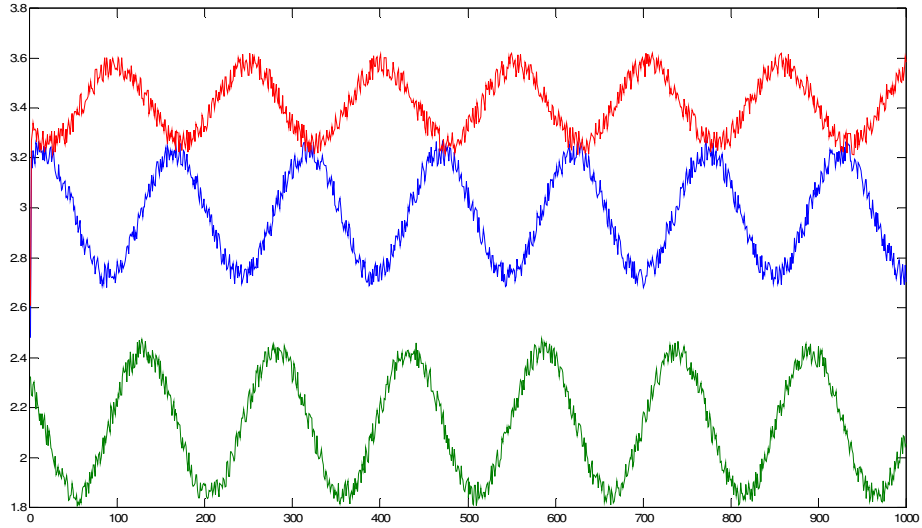


Figura 7.4 – Sinais elétricos dos sensores

Aqui fica bem nítida a presença dos ruídos nos sensores.

7.2.2– Subsistema de determinação de atitude

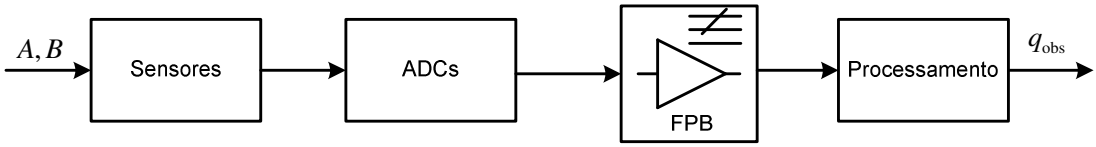


Figura 7.5 – Subsistema de determinação de atitude

Como a figura apresenta, este subsistema é divido em quatro blocos.

7.2.2.1– Bloco sensores

É responsável pela leitura das grandezas que compõem os sinais dos sensores. Neste bloco são aplicadas as funções de transferências dos sensores de forma inversa à da equação (7.6). Embora seja possível determinar os ganhos e off-sets, não é possível retirar o ruído.

7.2.2.2– Bloco filtro passa-baixa

Este bloco tem como função reduzir a banda, eliminando parte dos ruídos de alta-frequência provenientes dos sensores. Aqui o modelo utilizado foi o filtro passa-baixa tipo média-móvel.

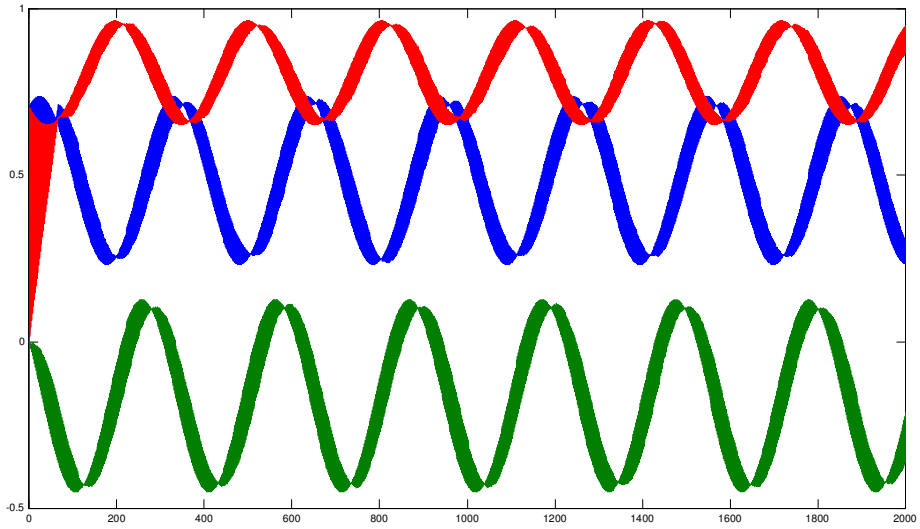


Figura 7.6 – Sinais após Filtro passa-baixa

A figura demonstra que o filtro passa-baixa foi eficiente, eliminando os ruídos presentes nas altas freqüências.

7.2.2.3– Bloco ADC

Este bloco é responsável pela conversão dos valores de tensão provenientes do bloco de filtragem, para valores digitais binários. Então, as componentes que antes eram valores de ponto flutuante, agora se tornar valores inteiros, que serão apresentados na saída do bloco.

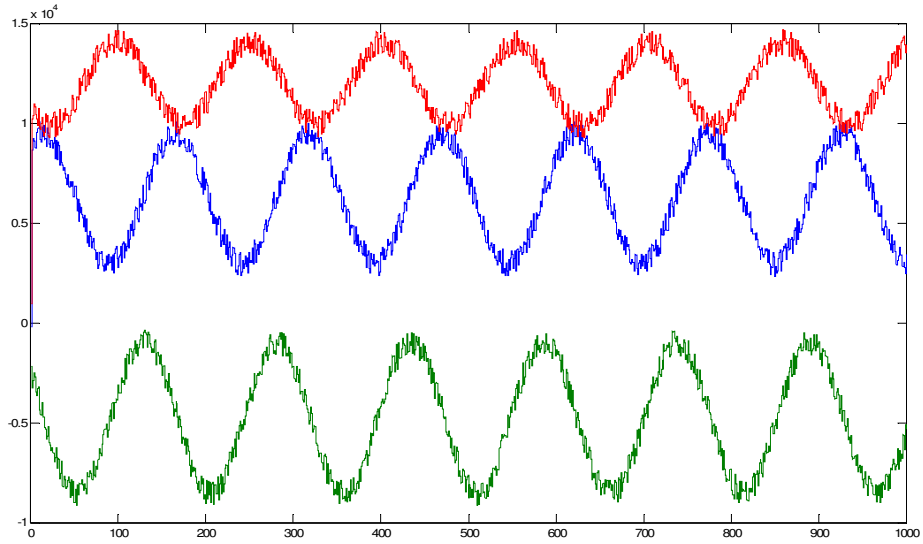


Figura 7.7 – Sinais dos sensores após conversão AD

A figura acima apresenta os valores de tensões na saída dos acelerômetros após a conversão analógico-digital.

Neste gráfico foi aplicada a função stairs do Matlab, e a figura a seguir apresentará o resultado desta função, e também a ampliação (zoom) feita no gráfico.

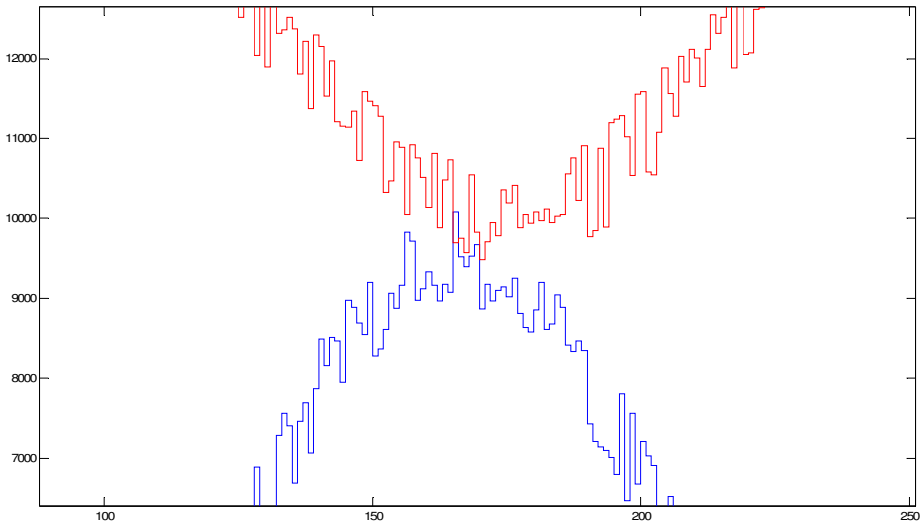


Figura 7.8 – Sinais dos sensores após conversão AD (utilizando função STAIRS do Matlab)

Aqui fica bem evidente a presença da digitalização dos sinais, existindo somente valores inteiros.

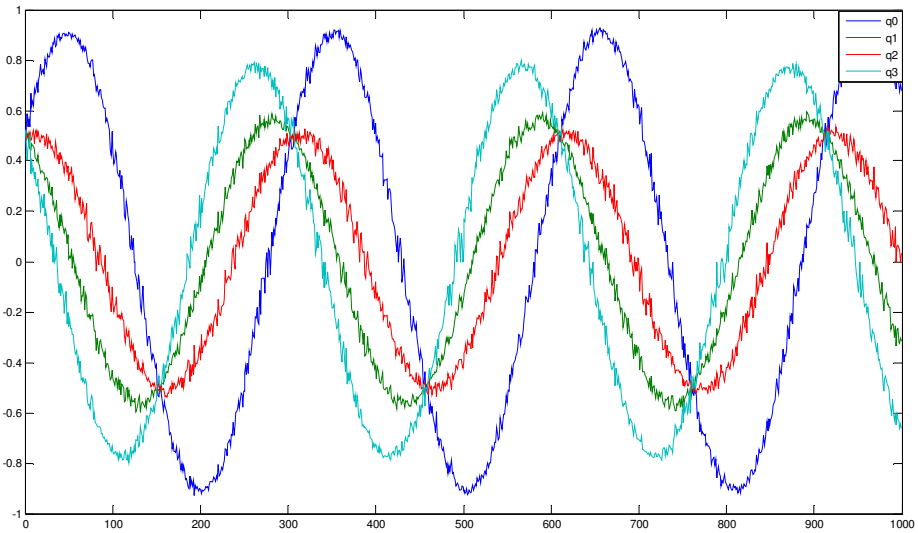
7.2.2.4– Bloco processamento

Neste bloco é executado o algoritmo de determinação de atitude como o TRIAD (TRI-axis Attitude Determination), ou o QUEST (Quaternion Estimator). O quatérnion de atitude aqui determinado é comparado com o quatérnion verdadeiro. Assim é possível observar a fonte de possíveis erros.

O simples fato de se poder testar a implementação em linguagem C de algoritmos, adiciona performance ao simulador. O parâmetro de desempenho computacional também pode ser avaliado neste item, como por exemplo, o número de multiplicações, soma e divisões realizadas.

Ainda neste bloco, o simulador implementa o Filtro de Kalman, e na saída teremos os valores dos quatérnions estimados e suas covariâncias.

Nas próximas figuras serão apresentados os valores da simulação, em vários pontos, ou seja, os valores dos quatérnions na saída do algoritmo QUEST, na saída de um cubo que sofre aceleração, e por último, na saída do cubo após a realização do Filtro de Kalman.



amostras

Figura 7.9 – Simulação do algoritmo TRIAD

Figura apresentando os quatérnions simulados, na saída do algoritmo TRIAD.

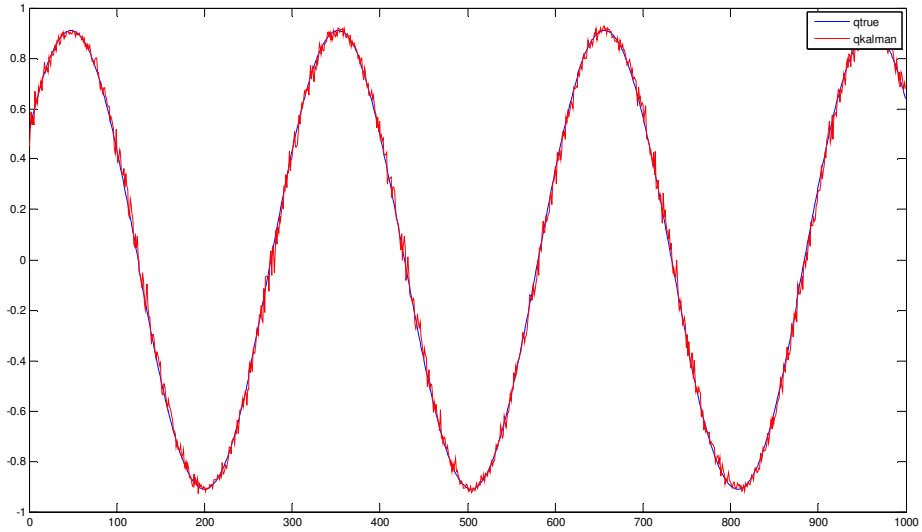


Figura 7.10 – Quatérnions simulados

Este gráfico apresenta uma comparação dos valores do quatérnion verdadeiro (qtrue) e o quatérnion na saída do algoritmo TRIAD (qtriade).

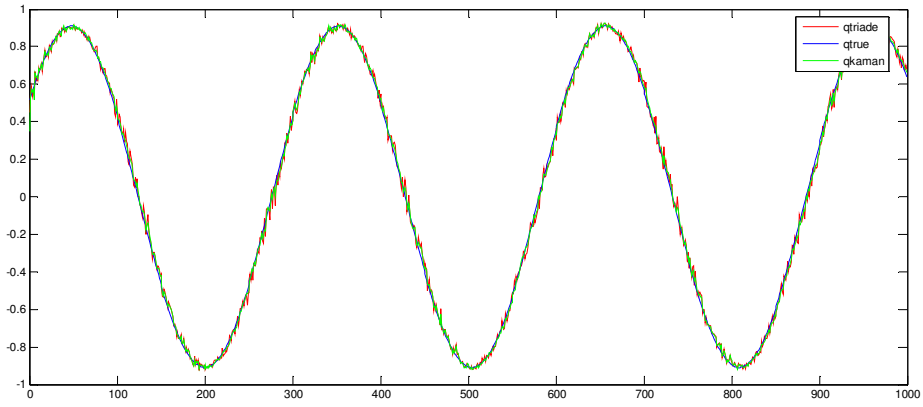


Figura 7.11 – Gráfico com os dados após o Filtro de Kalman

Este gráfico apresenta o resultado da simulação de Kalman.

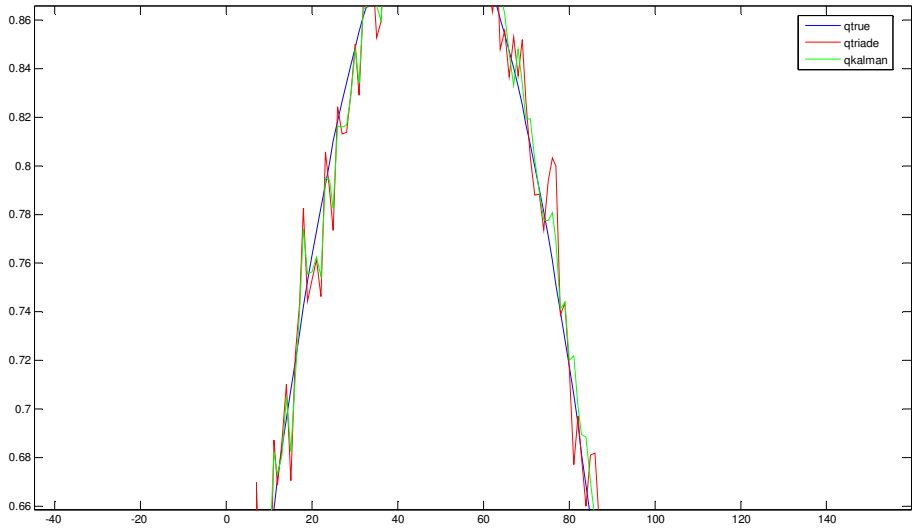


Figura 7.12 – Gráfico comparando uma componente em vários processos do simulador

Este gráfico apresenta a comparação entre os quatérnions, qtrue é o quatérnion real, qquest é o quatérnion da simulação do algoritmo QUEST e qkalman é o quatérnion resultante da simulação do Filtro de Kalman.

Neste gráfico é possível verificar a eficiência do Filtro de Kalman, e o seu resultado aproxima-se do quatérnion verdadeiro.

Capítulo 8 – Conclusões

O objetivo deste trabalho foi desenvolver um software para simular um determinador de atitude.

Este software vem simular os sensores presentes em um hardware determinador de atitude. Estes sensores são de baixo custo, tipo MEMS, como acelerômetros, magnetômetros e giros.

Para tornar o modelo destes sensores mais próximo da realidade, foram inseridos ruídos aleatórios em vários estágios da simulação, como na leitura dos valores em volts e nos conversores analógico-digital.

Um estudo sobre sistemas de coordenadas foi feito, além dos algoritmos para determinação de atitude como TRIAD e QUEST.

O princípio de funcionamento do algoritmo do Filtro de Kalman foi apresentado, como método de correção da leitura dos sensores giro.

O software utilizou de recursos da programação orientada a objetos, como herança, polimorfismo, sobrecarga de operadores. Isto foi feito através da linguagem C++.

A apresentação gráfica dos resultados do simulador foi desenvolvida com a biblioteca gráfica OpenGL.

Um conceito que poderá ser implementado, e de grande importância para todo processo, é desenvolver um algoritmo de auto-calibração dos acelerômetros, visto que, no mundo real, o determinador de atitude poderá sofrer acelerações.

Referências Bibliográficas

[APP04] *APPEL, P., Appel, P. Attitude Estimation from Magnetometer and Earth-Albedo- Corrected Coarse Sum Sensor Measurements*, Master Thesis, 2003.

[CHI00] CHIANG, Y. T., et al. *Data Fusion of Three Attitude Sensors*. SICE 2001, Nagoya, pg 234-239

[DUM99] DUMAN, I., *Design, implementation and testing of real-time software system for a quaternion-based attide estimation filter*, Master Thesis, NPS March 1999.

[FAL01] FALBEL, G., Paluszek, M. A. *An Ultra Low Weight/Low Cost Three Axis Attiude Sensor Readout System for Nano-Satellites.* IEEE 2001 Aerospace Conference Proceedings, pg. 2469-2481 vol. 5.

[FGJ06] GRANZIERA Jr., F., *Simulação e Implementação de Um Determinador de Atitude em Tempo Real Utilizando Sensores Microfabricados*, Dissertação de Mestrado, Universidade Estadual de Londrina.

[GEB04] GEBRE-EGZIABHER, D., et al. *Design of Multi-Sensor Attitude Determination Systems.* IEEE Transaction on Aerospace and Eletronics Systems, Vol. 40, Nº 2, Abril 2004, pg. 627-649.

[GEL74] GELB, A., Ed. *Applies Optimal Estimation*, MIT Press, Cambridge, Mass., 1974.

[HAY96] HAYKIN, S. *Adaptive Filter Theory,* 3º Ed. Prentice Hall Information and Systems Science Series, 1996.

[JAZ70] JAZWINSKI, A., *Stochastic Process and Filtering Theory*, Academic Press, NY, 1970.

[KAL60] KALMAN, R. E., *"A New Approach to Linear Filtering and Prediction Problems"* Paper 59-IRD-11 presented at ASME Intruments and Regulators Conference, March 29-April 2, 1959 (also Transaction of ASME, Series D, Journal of Basic Engineering, Vol. 82, March 1960, pp, 35-45).

[KUI02] KUIPERS, J. B., *Quaternion and Rotations Sequences*, Princeton University Press, 2002.

[KEF82] LEFFERTS, E. J., MARKLEY, F. L, SHUSTER, M. D, *Kalman filtering for spacecraft attitude estimation*. J. Guidance, Reston, v.5, n.5, p.417-429, 1982.

[JCL98] LOTTERS, J. C., *Procedure for in-use Calibration Triaxial Accelerometers In Medical Applications*, Sensors and Actuators A Physical, Elsevier, September 1998, pp 221-228.

[HAY02] HAYKIN, S, *Adaptive Filter Theory*, Prentice Hall, 200.

[MAR00] MARKLEY, F. L., MORTARI, D., *Quatérnion Attidue Estimation Using Vector Observations*, The Journal of the Astronautical Science, Vol. 48, Nº 2 e 3, April-September 2000, pp 359-380.

[MAR01] MARINS, J. L., *An Extended Kalman Filtering for Quaternion-Based Orientation Estimation Using MARG Sensors.* Proceedings of the IEEE 2001. pg 2003-2011.

[SAN00] SANTONI, F., Bolotti, F., *Attitude Determination Of Small Spinning Spacecraft Using Three Axis Magnetometer and Solar Panels,* Proceedings of the IEEE Aerospace Conference, Big Sky, USA, 2000, pg. 127-133.

[SOR66] SORENSON, H. W., *Kalman Filtering Techniques Advances in Control Systems*, Vol. 3 Ed. C. T. Leondes, Academic Press, MY, 1966.

[SHR04] SHREINER D., *OpenGL® Reference Manual: The Official Reference Document to OpenGL.* Addison Wesley Professional 2004.

[SHU80] SHUSTER, M. D., and S. D. Oh. *Three-Axis Attitude Determination from Vector Observations.* AIAA Journal of Guidance and Control 1980. Vol. 4, No. 1. pg. 70-77.

[SHU01] SHUSTER, M. D. *In QUEST of better attitudes.* (paper AAS 01-250) Advances in the Astronautical Sciences, Vol. 108, Part II, pp. 2089-2117, 2001.

[WAH66] WAHBA, G. *A least squares estimate of satellite attitude.* Siam Review, Philadelphia, v.8, i.3, p.384-386, 1966.

[ZAR02] ZARCHAN, P., Musoff, H., *Fundamentals of Kalman Filtering* – A Pratical Approach, Progress in Atronautics and Aeronautics Series, Vol. 1990 Ed. AIAA, 2002.

[AEB04] AEB, Agência Espacial Brasileira, *Projeto Uniespaço*, Anúncio de Oportunidades 01/2004, 2004.

[DET04] TOSIN, M. C., *Desenvolvimento de um Determinador de Atitude*, Projeto Uniespaço, Agência Espacial Brasileira, 2004.

Anexos

O código fonte do simulador foi anexado à dissertação em uma mídia digital CD.