LUCAS SÁBER ROCHA

VEÍCULO AÉREO NÃO-TRIPULADO: MODELAMENTO, SIMULAÇÃO E CONTROLE AUTOMÁTICO

São Paulo
2007

LUCAS SÁBER ROCHA

VEÍCULO AÉREO NÃO-TRIPULADO: MODELAMENTO, SIMULAÇÃO E CONTROLE AUTOMÁTICO

Monografia do Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Diploma de Engenheiro Mecatrônico

Área de Concentração: Engenharia Mecatrônica

Orientador: Prof. Dr. Marcelo Massarani

São Paulo
2007

FICHA CATALOGRÁFICA

Rocha, Lucas Sáber
Veículo aéreo não-tripulado: modelamento, simulação e controle automático / L.S. Rocha. -- São Paulo, 2007.
123p.

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

1.Aeronaves não tripuladas 2.Tempo real (Simulação computacional; Simulação numérica) 3.Controle de aeronaves 4.Engenharia de aeronaves I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t.

DEDICATÓRIA

Dedico este trabalho à sociedade brasileira.

AGRADECIMENTOS

Ao professor Dr. Marcelo Massarani, pela paciência de me aturar por tantos anos nas mais diversas aventuras e por sua pertinácia nos valores da vida.

Aos meus pais, pela intrepidez em assumir constantes novos desafios.

Não consentia que se lhe recusasse nada. Daí vinha a sua pertinácia na vida e a sua intrepidez no oceano.

(Victor Hugo)

RESUMO

O projeto consiste no modelamento de aeronave tendo em vista características como a atmosfera, a dinâmica de vôo e sua aerodinâmica, fatores gravitacionais, eólicos e características do motor. São calculadas então variáveis essenciais, como as velocidades lineares e angulares, a atitude da aeronave e seu posicionamento em relação a um sistema de coordenadas fixo na Terra; que são atualizadas a determinada taxa por métodos de integração. A simulação é feita em tempo real, podendo ser visualizada no programa Google Earth, tendo o usuário controle sobre as superfícies de controle da aeronave e a rotação e admissão do motor. É possível também efetuar o controle automático da aeronave, informando determinada rota para que ele execute sem a interferência humana. No trabalho ainda constam pesquisas de servo-motores e sensores disponíveis no mercado e sua calibração no software para a aplicação em aeronave real.

Palavras-chave: UAV (unmanned aerial vehicle). VANT (veículo aéreo não-tripulado). Controle automático. Piloto automático. Simulação em tempo real de aeronave. Modelamento de aeronave.

ABSTRACT

The current project consists in modelling an aircraft respecting characteristics such as atmospherics, flight dynamics and the respective aerodynamics, gravitational and wind factors and engine characteristics. Then essential variables are calculated, such as linear and angular speeds, the attitude of the airplane and its position according to a coordinate system fix at Earth; that are actualized at certain taxes by integration methods. The simulation is executed in real-time, being visualized with the program Google Earth; the user can control the surfaces of the aircraft, the engine speed and the manifold pressure. It is also possible to perform the airplane automatic control, informing a path to be fled without human interference. Still in this work, there are researches of servos and sensors available in the market and their calibration at the software to application in real airplane.

Keywords: UAV (unmanned aerial vehicle). Automatic control. Autopilot. Real-time aircraft simulation. Aircraft modelling.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SÍMBOLOS

a	velocidade do som
dpt	aumento de pressão atrás da hélice
g	aceleração da gravidade
g_0	aceleração da gravidade ao nível do mar
H	altitude geopotencial
M	número de Mach
p	velocidade angular de rolagem
P	potência do motor
p_0	pressão atmosférica ao nível do mar
p_s	pressão ambiente
q	velocidade angular de arfagem
q_c	pressão de impacto
q_{dyn}	pressão dinâmica
r	velocidade angular de guinada
R	constante universal dos gases para o ar
R_c	número de Reynolds com respeito à corda aerodinâmica média
R_e	número de Reynolds por unidade de comprimento
R_{terra}	raio da Terra
T	temperatura ambiente
T_0	temperatura do ar ao nível do mar
T_t	temperatura total
u	componente da velocidade em relação ao eixo X_B
v	componente da velocidade em relação ao eixo Y_B
V	velocidade do ar real
V_c	velocidade do ar calibrado
V_e	velocidade do ar equivalente
w	componente da velocidade em relação ao eixo Z_B
x_e	coordenada x em referencial fixo na Terra
y_e	coordenada x em referencial fixo na Terra
α	ângulo de ataque

β	ângulo de sideslip
γ	razão dos calores específicos a pressão e volume constantes para o ar
θ	ângulo de arfagem
λ	gradiente da temperatura na troposfera
μ	viscosidade dinâmica
ρ	densidade do ar
ρ_0	densidade do ar ao nível do mar
φ	ângulo de rolagem
ψ	ângulo de guinada

SUMÁRIO

INTRODUÇÃO

O progressivo aumento do uso de veículos aéreos não-tripulados exige uma constante otimização do sistema de controle. Para isso, algoritmos de controle são desenvolvidos em diversas linguagens, possibilitando o cálculo de sinais de saída relevantes ao vôo em tempo real. Esses sinais podem ser impressos em valores numéricos, gráficos ou através de animações que simulem o vôo.

No presente trabalho, através da análise de um modelo matemático que leva em consideração desde variáveis relativas ao ar até o modelo propulsor da aeronave, foi implementado um algoritmo que recebe algumas variáveis por condições pré-estabelecidas e calcula parâmetros para o controle do vôo ao longo do tempo. Entre os parâmetros estão a velocidade da aeronave, sua altitude e seu ângulo de ataque. Tendo os valores, um piloto automático pode entrar em ação para dar estabilidade ao veículo ou para fazê-lo seguir determinada trajetória.

O programa, que foi desenvolvido na linguagem Matlab, inicia pedindo os modelos aerodinâmicos, propulsores e geométricos da aeronave, sendo constantes ao longo da sua execução. Requisitando também condições iniciais de alguns parâmetros importantes, é feito o cálculo de parâmetros atmosféricos, aerodinâmicos, propulsores, gravitacionais, relativos ao vento e outros adicionais, totalizando 89 sinais de saída (podendo ser acrescentados outras variáveis que o usuário supor conveniente). O algoritmo então recalcula os parâmetros num intervalo de tempo controlável e prossegue o cálculo até um tempo fixo. Todos os dados são atualizados em tempo real e representados numericamente numa janela de controle, na qual o usuário tem acesso a comandos manuais das superfícies de controle e a comandos do motor. Pode também acionar o piloto automático na forma de vôo linear estável; requisitar curva para qualquer direção; direcionar o vôo para determinada coordenada em termos de latitude e longitude; ou ainda determinar pontos específicos de uma trajetória, na qual a aeronave voará de forma automática.

Almejando facilitar a visão da simulação pelo usuário, os dados são interpretados na forma gráfica de linhas representando a trajetória e então são mostrados, com tempo de atualização especificado, no programa Google Earth, que permite visualizar o mundo real através de imagens feitas por satélites.

Muitas hipóteses simplificadoras foram adotadas como requisitos para o uso do modelo matemático. Elas estão expostas ao longo do texto, ficando o programa aberto para que qualquer alteração possa ser feita em alguma função.

O texto é iniciado através de um breve histórico dos pilotos automáticos, dando origem aos conceitos por trás dos veículos aéreos não-tripulados. Segue-se um capítulo com a explicação da metodologia adotada, desde a seleção da linguagem apropriada para cada caso até as opções de controle. O terceiro capítulo aborda o modelamento matemático [RAUW 2005] e [NERIS 2001], sendo dividido em oito sub-capítulos: cálculo dos parâmetros atmosféricos, aerodinâmicos, propulsores, gravitacionais e eólicos, equações de movimento, parâmetros adicionais, o vetor de estado futuro e os modelos de entrada da aeronave. Os modos de controle são então abordados no quarto capítulo, enquanto a interface gráfica é discutida e apresentada no quinto. São então discutidos alguns conceitos por trás do desenvolvimento do algoritmo e demonstrado resultados. Por fim, a conclusão sintetiza as descobertas mais relevantes.

1 HISTÓRICO

A história dos pilotos automáticos remonta a antigos pescadores que prendiam a cana do leme e o timão dos barcos em posições fixas para produzir uma trajetória ótima e evitar que um número maior de pescadores se preocupasse com a navegação da embarcação. Dessa forma, eles poderiam contribuir melhor com o lançamento e recuperação das redes de pesca. Provavelmente o critério para a escolha da otimização da trajetória seria simplesmente manter o percurso e minimizar movimentos induzidos.

Somente após a revolução industrial, que métodos para direcionar automaticamente as embarcações foram contemplados e apenas no início do século XX os primeiros pilotos automáticos, propriamente ditos, foram utilizados. Para isso seria necessário o desenvolvimento de poderosos timões e de motores que permitissem o direcionamento. A grande motivação surgiu da necessidade de grandes barcos de guerra efetuarem manobras em alta velocidade.

Terminado o desenvolvimento dos mecanismos, a atenção se voltou a permitir que entradas fossem enviadas à máquina direcional a partir de um componente que buscasse a direção automaticamente. Assim, os girocompassos elétricos foram essenciais para a evolução dos pilotos automáticos. Eles permitiram acabar com os problemas de compassos magnéticos, que acumulavam erros devido ao campo magnético e sistemas elétricos nas embarcações, torpedos e submarinos.

O primeiro giroscópio elétrico foi demonstrado em 1890 e em 1908 foi patenteado por Anschutz o primeiro girocompasso direcionado para o Norte. Finalmente em 1911 Sperry patenteou seu compasso balístico, que incluía amortecimento vertical. Essa foi a invenção considerada a mais importante, e por integrar a engenharia mecânica com a elétrica, é considerada uma das primeiras invenções mecatrônicas, que na época tinham os pilotos automáticos baseados em PID, que até hoje dominam a industria de controle e ainda são usados em embarcações, apesar dos aprimoramentos com a digitalização permitida pelos computadores e pelo controle moderno.

Lawrence Sperry era piloto estadunidense e tinha interesse no desenvolvimento de um estabilizador giroscópico para aeronaves. Gostaria que seu

aparato permitisse ao avião a manutenção de sua trajetória e sua atitude sob qualquer circunstância. Apesar do efeito giroscópio já ter sido bem explorado na época, ainda não havia aplicação para aeronaves. Sperry imaginou que se os três eixos da aeronave, pudessem ser subordinados à estabilidade de um giroscópio, um sistema de controle automático poderia ser então desenvolvido. Uma aeronave sem o controle do piloto pode tender à instabilidade por fatores externos, mas Sperry imaginou que um giroscópio parafusado poderia manter a orientação original da aeronave. Então o inventor ligou as superfícies de controle (aileron, profundor e leme) a três giroscópios, permitindo que correções de vôo fossem introduzidas baseadas nos desvios das deflexões angulares entre a direção do vôo e as configurações iniciais do giroscópio.

O dispositivo de direcionamento atuaria mecanicamente no que o piloto agiria de forma instintiva. O controle desenvolvido por Sperry mantinha a configuração de deflexão nula para as superfícies, a menos que alguma ação de correção fosse necessária. Para isso era preciso uma ligação mecânica para as superfícies de controle. Os giroscópios precisavam de energia elétrica, obtida por um gerador acionado pelo vento e montado na asa superior, para manter a velocidade de rotação dos motores.

O estabilizador estava ligado mecanicamente ao mecanismo de controle da aeronave, portam a industria aeronáutica era fragmentada e diferentes fabricantes tinham métodos distintos de operar as superfícies de controle. O estabilizador de Sperry precisava de quatro giroscópios com rotação de sete mil rotações por minuto. Como um dos giroscópios se moviam em oposição ao movimento do avião, a ligação com válvulas iria atuar pistões operados por ar comprimido e conectado por alavancas ao controle das superfícies. E ainda um anemômetro que pudesse sentir uma velocidade do ar inadequada e instabilidade incipiente também estava ligado ao dispositivo e iria forçar uma ação corretiva. Todo o dispositivo, que pesava pouco mais de 18kg, estava comprimido num pequeno pacote (com dimensões 450x450x300 mm), pouco para algo tão sofisticado para a época.

Os pilotos automáticos modernos dividem um vôo em sete fases, incluindo a fase de taxiar, a decolagem, o vôo em ascensão, nivelado e em descida, a aproximação e o pouso. Existem controles para todas as fases com exceção para taxiar. Além disso existem funções automáticas para evitar colisões. Tudo isso é feito por softwares de computador, que lêem a posição atual da aeronave e

controlam um sistema de vôo para guiar a aeronave. Além do controle de vôo clássico, muitos pilotos automáticos permitem controlar o empuxo exercido para assim otimizar a velocidade, movendo o combustível para diferentes tanques numa atitude ótima no ar, permitem assim, que seja consumido menos combustível que um piloto humano. O piloto automático lê a posição e a atitude da aeronave através de um sistema de referencial inercial, que acumulam erros ao longo do tempo. Eles incorporam redutores de erro como o sistema carrossel, que rotaciona uma vez por minuto para dissipar os erros em diferentes direções e permitir um efeito nulo global. Os dados de posicionamento apresentam erros entre o sistema mecânico e o guiado a laser devido a propriedades físicas, e podem ser resolvidos por processamento digital de sinais, como o filtro de Kalman, que atua nos três eixos da atitude e nos três pontos de posicionamento em relação a um referencial no solo.

O filtro de Kalman é um filtro recursivo que estima o estado do sistema dinâmico a partir de uma série incompleta e de medições com ruídos. As aplicações deste filtro foram possíveis devido a problemas na estimação da trajetória do programa Apollo da NASA. Ele foi incorporado mais tarde no computador de navegação do Apollo. O desenvolvimento do filtro foi publicado em 1960 por Rudolf Kalman, porém um algoritmo similar havia sido publicado por Swerling em 1958. Este filtro também é conhecido como estimador linear quadrático (LQE). Ele é baseado em sistemas dinâmicos lineares discretizados no domínio do tempo, modelado por redes de Markov construídas por operadores lineares perturbados por ruído Gaussiano.

Diversos filtros foram desenvolvidos a partir deste, como o filtro estendido de Schmidt, o filtro de informação e uma variedade de filtros desenvolvidos por Bierman, Thornton e diversos outros.

2 METODOLOGIA

O presente trabalho possui três etapas distintas, o modelamento, a simulação e o controle da aeronave. A primeira envolve inicialmente a escolha das limitações do modelo matemático, para que possam ser selecionados os parâmetros mais relevantes. Nesta etapa buscou-se seguir o trabalho de [RAUW 2005] com o estudo da aeronave Beaver, porém o algoritmo permite a expansão para diversos tipos de aviões, respeitadas as restrições. A simulação busca demonstrar em tempo real o comportamento do veículo modelado num mundo virtual, construído a partir dos parâmetros selecionados na etapa do modelamento. Nela busca-se minimizar os erros gerados e propagados pelo modelo, tanto os de truncamento quanto os de aproximação, sendo necessária seleção do método de integração mais adequado, de acordo com o passo (tempo) desejado. Na simulação constam interfaces gráficas que fornecem dados na forma numérica e gráfica. A etapa do controle fornece ao usuário diversas possibilidades de escolha, discutidas adiante.

Essas etapas são o núcleo do algoritmo e para isso, foi necessário escolher as linguagens de programação mais adequadas, uma que fornecesse o alto nível exigido pela interface gráfica e outra de baixo nível, que pudesse ser incorporada ao sistema de controle real. Finalmente foi realizada uma etapa de testes para diferentes trajetórias de vôo para que o algoritmo pudesse ser adequado às características do mundo real. Assim, as etapas fundamentais do trabalho podem ser listadas da seguinte forma:

- Seleção das Linguagens
- Modelamento Matemático
    - Condições e Simplificações
    - Seleção de Parâmetros
    - Equacionamento
- Simulação
    - Dados Numéricos
    - Dados Gráficos
- Controle

- Testes de Vôo
- Adequação do algoritmo ao Mundo Real

Seleção das Linguagens

O projeto é dividido em dois algoritmos, com duas linguagens distintas: uma de alto nível e outro de nível inferior. O primeiro busca apresentar uma interface gráfica com o usuário, visando a possibilidade de diversos testes dentro do mundo virtual gerado. Este algoritmo é fragmentado de tal forma que a inserção de novos parâmetros sejam facilmente incorporados ao núcleo do programa. Assim, a linguagem selecionada deve permitir essa modularização, além de suportar uma interface gráfica simples e que forneça os dados mais importantes da simulação em tempo real. O segundo algoritmo será embarcado no sistema de vôo da aeronave e deve preferencialmente ter linguagem de baixo nível porque será executado por um microcomputados durante o vôo, não podendo haver desperdício de processamento.

Para a escolha da linguagem de alto nível mais apropriada, foram comparadas três linguagens conhecidas e que teriam a capacidade para as funções: C++, Java e MATLAB.

A linguagem C++ é a sucessora do C e acrescentou a esta, entre outras modificações, a possibilidade da orientação a objetos, permitindo o desenvolvimento de algoritmos com objetos que podem ser criados ou retirados de coleções de dados, que possuem grande disponibilidade e fácil acesso ao programador. É assim, considerada uma linguagem bem estruturada e facilmente utilizada. A linguagem também é considerada mais eficiente e mais portátil que sua antecessora, mas ainda sim apresenta deficiências. Uma característica marcante da linguagem são os tipos das variáveis, que são estáticas. A linguagem também permite maior controle do uso da memória, através da sua alocação dinâmica. A possibilidade de se trabalhar com linguagens de baixo nível é maior que as demais linguagens, porém essa característica tem como conseqüência uma complexidade maior do código. A sintaxe suportada é estruturada, genérica e orientada a objeto, tendo o programador permissão para diversas alternativas. Seu funcionamento não exige um ambiente sofisticado para o funcionamento.

A linguagem Java foi originalmente desenvolvida pela Sun visando uma melhor portabilidade, permitindo a execução de pequenas aplicações em páginas da web, porém o uso em programação de aplicativos passou a ser alto devido a

peculiaridades da linguagem, como a ausência de ponteiros, a coleta do lixo e a "máquina virtual", que permitiram o desenvolvimento de softwares que não travassem e não vazassem recursos do sistema. A linguagem Java possui um modelo de objeto bem simplificado e precisa ser compilado por um "byte-code", que é então rodado por uma máquina virtual Java. Em relação à linguagem C++ passa a ter menor facilidade no uso de ferramentas de baixo nível e sua metodologia de programação é exclusivamente orientada a objeto. Ela permite também que a execução dos códigos em fontes remotas possa ser realizada de forma segura. O mesmo programa pode ser executado em múltiplos sistemas operacionais e tem suporte para uso em redes de computadores. A função da coleta de lixo permite que linhas sem utilidade fiquem virtualmente invisíveis ao desenvolvedor, o que não acontece na linguagem C++, já que a possibilidade de diversas alternativas na programação permite a escolha de soluções que não sejam adequadas à situação. Em relação à portabilidade o único entrave é a necessidade das bibliotecas suportadas estarem em caminhos padrões para o Java. Comparada à linguagem C++, Java é menos flexível por haver perda do controle na alocação da memória. Outras caracaterísticas comuns às demais linguagens estão ausentes no Java, como as propriedades de classes, a herança múltipla e a sobre-carga do operador. A performance dos softwares pode ser afetada mais pela qualidade do compilador do que por propriedades intrínsecas da linguagem, perdendo assim em padronização e desempenho. Outras desvantagens constam da interpretação lenta e da compilação, que sofre penalização de performance inicial no tempo de carregamento e execução.

O MATLAB é um ambiente de computação numérica e uma linguagem de programação criada pela MathWorks que permite fácil manipulação de matrizes, plotagem de funções e dados, implementação de algoritmos e criação de interfaces com o usuário.

A linguagem é classificada como orientada a valor e é facilmente escrita e manipulada no ambiente de desenvolvimento, para então ser depurada pelo MATLAB. Assim, é bastante utilizado em desenvolvimento de protótipos para novos programas. Não há dependência de plataforma, havendo amplo suporte para a linguagem e facilidade na migração para novas plataformas. A quantidade de funções pré-definíveis é imensa, o que permite que problemas complexos e específicos sejam solucionados por contribuições de usuários na forma de "Tool

Boxes" (como o *Flight Dynamics Control*). Outra facilidade são os comandos para desenhos e imagens disponíveis, que podem ser apresentados em qualquer dispositivo de saída gráfica suportada pelo computador, sendo uma ótima opção para visualização de dados técnicos, e portanto uma ferramenta largamente utilizada no universo da engenharia.

As variáveis na programação da linguagem não são estáticas e não precisam apresentar tipos (apenas valores gravados possuem essa característica). A sintaxa é mais simples em relação às demais linguagens. Uma restrição é o fato do produto estar atrelado à sua empresa proprietária, a MathWorks. O processamento de sinais fica dificultado pela ausência de informações em taxas padronizadas e a indexação de vetores dificulta muitas idéias matemáticas e especialmente o processamento digital de sinais. A linguagem de programação, por ser imperativa, não atualiza as variáveis de forma automática em resposta a variações das entradas. Outro empecilho é a inexistência de suporte a referências, o que dificulta as estruturas de dados que possuam direções indiretas.

Logo, a comparação entre as três linguagens gera as seguintes características:

C++

- Linguagem eficiente e portátil
- Tipos estáticos
- Maior possibilidade de trabalho com linguagens de baixo nível
- Estrutura mais complexa
- Maior controle do uso da memória (alocação dinâmica da memória)
- Sintaxe estruturada, genérica e orientada a objeto
- Diversas alternativas permitidas ao programador
- Não é necessário ambiente sofisticado

Java

- Compilado por byte-code e rodado por máquina virtual Java
- Modelo de objeto simplificado
- Menores facilidades em baixo nível
- Metodologia exclusiva de programação orientada a objeto
- Mesmo programa é executado em múltiplos sistemas operacionais
- Suporte para uso em redes de computadores

- Execução de códigos por fontes remotas de forma segura
- Funções inúteis ficam invisíveis ao desenvolvedor
- Problemas de portabilidade
- Menor flexibilidade devido à perda do controle da alocação da memória
- Não possui as propriedades de classes, herança múltipla e sobre-carga do operador
- Não há padronização na performance por dependência no compilador
- Interpretação lenta
- Compilação sofre penalização de performance inicial no tempo de carregamento e execução

MATLAB

- Linguagem orientada a valor
- Manipulação simbólica
- Variáveis não são estáticas e não precisam ter tipos
- Sintaxe simplificada
- Usuário restrito à empresa proprietária do produto
- Maior uso em ferramentas de engenharia
- Processamento de sinais dificultado por ausência de informações em taxas padronizadas
- Indexagem de vetores dificulta muitas idéias matemáticas e processamento digital de sinais
- Linguagem de programação imperativa
- Não existe suporte a referências

Foi escolhida a linguagem MATLAB principalmente por disponibilizar de amplas ferramentas para a interface gráfica, além de ter uma melhor manipulação simbólica, um diferencial para facilitar o equacionamento. A existência de diversos “tool boxes” que trabalham com controle de vôo também facilitou a comparação de resultados na simulação. Além disso, a sintaxe simplificada e a existência de help com diversos exemplos de aplicação facilitaram o desenvolvimento.

Modelamento Matemático

Esta etapa pode ser dividida em três: a escolha das limitações, na forma de condições e simplificações; a escolha dos parâmetros relevantes e o equacionamento.

O modelo que será simulado nunca pode ser considerado como real, já que sempre existirá um conjunto de hipóteses simplificadoras. O modelo envolve não apenas a aeronave, mas também o ambiente, com suas características atmosféricas, formando assim um verdadeiro “mundo virtual”.

No equacionamento será calculada a pressão dinâmica originada da lei de Bernoulli, que somente é aplicável para velocidades do ar inferiores a 100m/s. Para o cálculo da temperatura é assumido que a temperatura ao nível do mar média é de 15ºC e que existe um decréscimo de 6,5ºC a cada 1000m de altitude. Essa é a característica da troposfera, o que limita o modelo a altitudes inferiores a 11.000km. O comportamento do vento está simplificado na forma de ventos unidirecionais constantes ou por rajadas de vento amortecidas.

As simplificações limitam a veracidade dos dados no “mundo real” por estarem presentes na aeronave, no ambiente e nos ventos. Todavia são adequadas para aeronaves de pequeno porte e que voem a baixas velocidades.

Para a maioria dos parâmetros foi seguida a metodologia abordada por RAUW (2005), em que foram selecionados parâmetros relacionados com a atmosfera, como a densidade do ar, a viscosidade dinâmica e a temperatura; parâmetros aerodinâmicos e do motor, na forma dos seus respectivos coeficientes para o cálculo das forças e momentos aerodinâmicos e do motor; parâmetros gravitacionais e eólicos; parâmetros referenciais. É possível assim, calcular as forças e momentos totais do modelo. É preciso, portanto, carregar as variáveis dos coeficientes aerodinâmicos e do motor, além da geometria, através da massa, dos momentos e produtos de inércia, dependendo de cada tipo de aeronave, função desempenhada pelo algoritmo.

O modelamento matemático carrega as constantes necessárias e calcula os parâmetros correspondentes para cada estado da aeronave. Esse estado envolve as velocidades lineares e angulares, a atitude da aeronave e seu posicionamento em relação a um sistema de coordenadas fixo na Terra. Essas variáveis são posteriormente derivadas para o cálculo da simulação.

Simulação

A simulação fornece o cálculo dos parâmetros para cada instante de tempo e através do MATLAB isso é fornecido em tempo real através de janelas.

Para que o algoritmo possa executar a simulação, ele precisa ser inicializado. Essa inicialização corresponde ao fornecimento pelo usuário das constantes do modelo e dos valores iniciais das variáveis de estado. Além disso, é preciso escolher um método de integração entre os disponíveis: método de Euler, Heun e Runge-Kutta (métodos usados em intervalos de tempo constantes). Cada método é utilizado para o cálculo do novo vetor de estado. E ainda é preciso selecionar o tipo do modelamento dos ventos.

A simulação usa um método numérico e, portanto, possui erros de duas naturezas: de truncamento e de aproximação. Quando são utilizadas equações matemáticas para aproximar a realidade por modelos, criam-se erros de truncamento. Os erros de aproximação se devem ao mundo digital dos computadores, que representam os valores numéricos por um número limitado de dígitos. O erro total é então calculado pela soma desses dois tipos de erro. Porém, eles possuem comportamentos diferentes conforme o passo da simulação (o tempo no caso do algoritmo) é alterado. Quando o passo aumenta, o erro de truncamento é aumentado porque aumentando a discretização a equação matemática a ser resolvida apresenta uma aproximação pior. Já o erro de aproximação é diminuído, porque o aumento no passo de discretização do método possibilita um menor número de passos e assim o número de operações é minimizado, evitando a propagação do erro. Esse comportamento possibilita a existência de um erro mínimo, que é a situação de otimização do erro total. E cada método apresenta o seu erro mínimo em uma quantidade de passos distinta. Cabe ao usuário selecionar o passo mais adequado de acordo também com a máquina em que o algoritmo será executado.

O próprio algoritmo então constrói um arquivo externo de tipo legível pelo software Google Earth e então o abre e o atualiza conforme taxa fornecida pelo usuário, o passo.

Controle

Um avião é controlado basicamente pelas superfícies de controle e pelo motor. As superfícies são os ailerons, os profundores, o leme e os flaps. No caso da

aeronave Beaver, modelo padrão do equacionamento usado, o motor é controlado pela rotação e pela pressão de admissão.

O programa desenvolvido permite que a aeronave seja controlada de diversas maneiras. Uma forma é a manual, em que é possível incrementar ou decrementar os ângulos das superfícies de controle. Para o controle automático, três modos estão disponíveis. O primeiro permite a escolha de uma direção. A aeronave então se ajusta e prossegue nessa direção indefinidamente, independente da intensidade e direção dos ventos. No segundo modo, o usuário fornece uma coordenada na forma de latitude e longitude e a aeronave voa até esse ponto permanecendo então em trajetória linear. No terceiro modo, o mais completo e que caracteriza um UAV autônomo, é fornecido um arquivo contendo diversos pontos (contendo latitude, longitude e altitude) e o avião voa de forma autônoma por esses pontos, constituindo uma missão através dos way-points.

De uma forma geral, o algoritmo de alto nível pode ser apresentado pelo seguinte esquema:

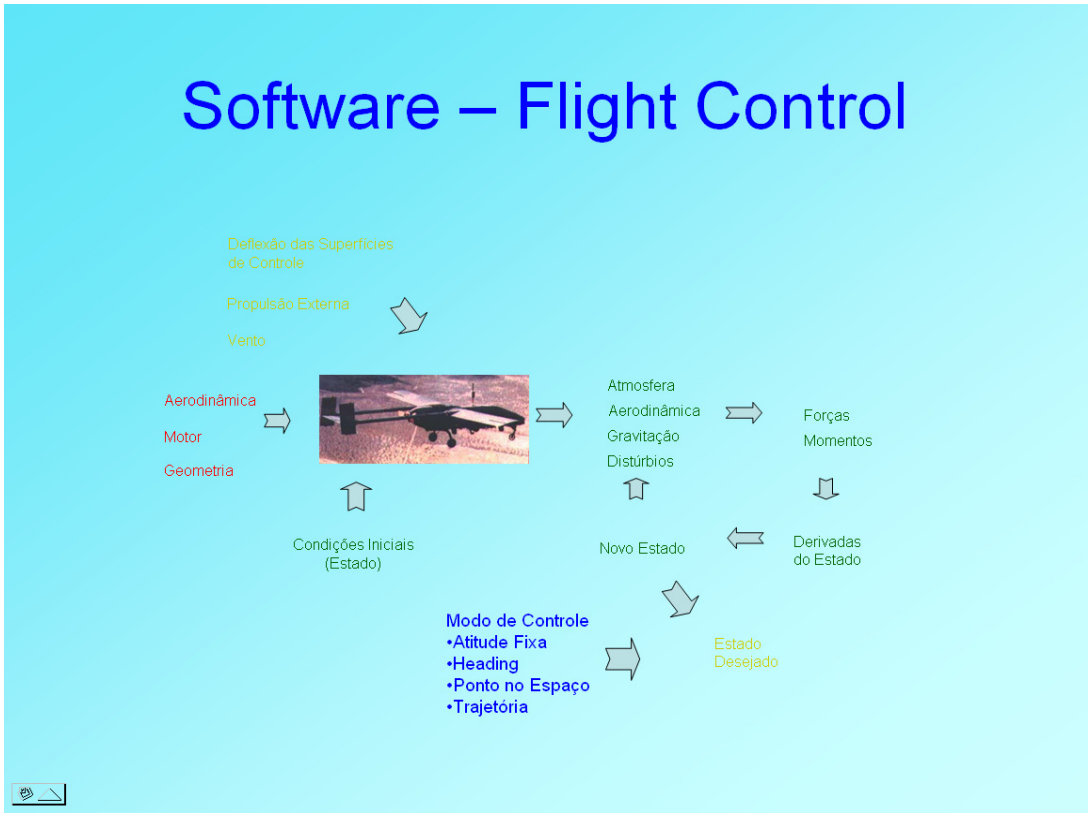

Figura 1 – Esquema do algoritmo de controle de vôo

3 MODELAMENTO MATEMÁTICO

3.1 Cálculo de Parâmetros Atmosféricos

No presente trabalho a representação essencial da aeronave é feita por doze variáveis que formarão um vetor de estado, que será continuamente atualizado por métodos de integração para que seja possível a previsão do movimento do veículo. As variáveis representam as velocidades lineares e angulares, a atitude da aeronave e seu posicionamento em relação a um sistema de coordenadas fixo na Terra. Ele é carregado na forma de variáveis de inicialização já que seus valores são necessários para o cálculo de demais parâmetros, como os atmosféricos. Este vetor será exposto no item referente aos modelos de entrada para a simulação.

Inicialmente calculam-se parâmetros que modelem a atmosfera já que para a aerodinâmica é relevante fazer considerações como a variação da aceleração gravitacional com a altitude da aeronave ou a proporcionalidade entre a densidade do ar com a temperatura, que por sua vez está relacionado com a altitude.

Para isso são necessárias algumas constantes do modelo terrestre. Os valores padrões para essas constantes se referem sempre à altitude do nível do mar e temperatura ambiente 15ºC.

A variação da aceleração da gravidade deve ser levada em consideração para os efeitos causados pelo peso da aeronave. Ela varia apenas com a altitude, sendo calculada por:

$$g = g_0 \cdot \left(\frac{R_{Terra}}{R_{Terra} + H} \right)^2 \tag{1.1}$$

sendo $g_0 = 9{,}80665 \left[m/s^2 \right]$ e $R_{Terra} = 6371020 \left[m \right]$

Na troposfera é possível admitir que a temperatura externa varia linearmente com a altitude. Isso é válido até altitudes próximas de 11.000 km, portanto pode ser

aplicado ao controle de vôo. O decréscimo é de 6,5 ºC a cada quilômetro de altitude numa ascensão. A figura a seguir ilustra esse comportamento [LUTGENS 2003]:

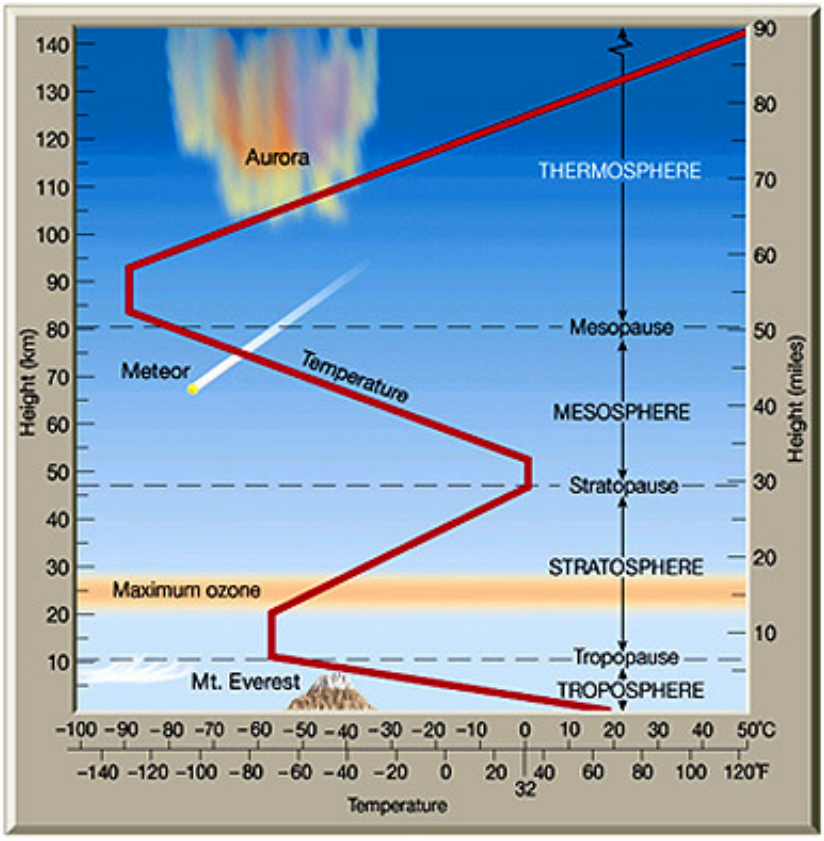

Figura 2 – Perfil de Temperatura da Atmosfera em função da Altitude

Assim adotaremos $\lambda = -0{,}0065\,[\mathrm{K/m}]$. Então, calculamos a temperatura para uma dada altitude:

$$T = T_0 + \lambda \cdot H \tag{1.2}$$

com $T_0 = 15\,[°C] = 288{,}15\,[K]$

Tendo a temperatura e a gravidade, podemos calcular a pressão estática do ar, que depende da altitude:

$$dp_s = -\rho \cdot g \cdot dH \Rightarrow p_s = p_0 \cdot \left(\frac{T_0}{T}\right)^{\frac{g}{\lambda \cdot R}} \tag{1.3}$$

com $R = 287{,}053\,[\mathrm{J/(K \cdot kg)}]$ e $p_0 = 101325\,[\mathrm{N/m^2}]$

A densidade do ar tem dependência da pressão estática e da temperatura, previamente calculadas:

$$\rho = \frac{p_S}{R \cdot T} \tag{1.4}$$

A viscosidade dinâmica pode ser calculada utilizando a equação de Sutherland, que utiliza duas constantes empíricas e uma variação com a temperatura. O valor da viscosidade não será utilizado no modelo, mas seu valor será um dos sinais de saída. Seu uso prático pode ser dado para um modelo de avião que leve em consideração a compressibilidade e efeitos de escala.

$$\mu = \frac{1{,}458 \cdot 10^{-6} \cdot T^{\frac{2}{3}}}{T + 110{,}4} \tag{1.5}$$

Com as cinco variáveis calculadas, formamos o vetor:

$$y_{atm} = [\rho \quad p_s \quad T \quad \mu \quad g]$$

Podemos calcular outros parâmetros que não dependam do modelo da aeronave, mas sim do fluido em que este se movimenta: o ar.

A velocidade do som pode ser dada por:

$$a = \sqrt{\gamma \cdot R \cdot T} \tag{1.6}$$

considerando $\gamma = 1{,}4$

O número de Mach, que utiliza o vetor de estado além da velocidade do som:

$$M = \frac{V}{a} \tag{1.7}$$

A pressão dinâmica, que é utilizada no cálculo das forças e momentos aerodinâmicos e propulsores, é alterada com a velocidade e a densidade do ar. A fórmula seguinte é uma conseqüência da aplicação da lei de Bernoulli que é

usualmente aplicada para velocidades do ar inferiores a 100 m/s, portanto o modelo terá essa limitação para aplicação.

$$q_{dyn} = \frac{\rho \cdot V^2}{2} \tag{1.8}$$

Assim, teremos o vetor y_{ad1} que será utilizado para solucionar as equações de movimento:

$$y_{ad1} = \begin{bmatrix} a & M & q_{dyn} \end{bmatrix}$$

Outras variáveis relacionadas com o ar podem ser calculadas sem o uso nas equações de movimento. Elas podem ter aplicações em medições em túneis de vento ou testes de vôo.

A pressão de impacto leva em consideração a compressibilidade do fluido (ar). Se a aeronave voar a velocidades supersônicas, as relações utilizadas no modelo, que são isoentrópicas, são substituídas por relações de choque supersônico. Assim, é pressuposto que o número de Mach seja inferior à unidade. Para o cálculo da pressão de impacto utiliza-se a seguinte expressão:

$$q_c = p_s \cdot \left[\left(1 + \frac{\gamma - 1}{2} \cdot M^2 \right)^{\frac{\gamma}{\gamma - 1}} - 1 \right] \tag{1.9}$$

Outras variáveis consideradas são as velocidades aerodinâmicas equivalente e calibrada:

$$V_e = \sqrt{\frac{2 \cdot q_{dyn}}{\rho_0}} \tag{1.10}$$

com $\rho_0 = 1{,}225 \left[kg / m^3 \right]$

$$V_c = \sqrt{\frac{2 \cdot \gamma}{\gamma - 1} \cdot \frac{p_0}{\rho_0} \cdot \left[\left(1 + \frac{q_c}{p_o} \right)^{\frac{\gamma - 1}{\gamma}} - 1 \right]} \tag{1.11}$$

Assim, temos um segundo vetor de parâmetros relacionados ao ar:

$$y_{ad2} = [q_c \quad V_e \quad V_c]$$

Ainda serão calculados sinais de saída que dependam do ar. Um deles é a temperatura total do ar, que é medida quando o ar é levado ao repouso isoentropicamente, que é diferente da temperatura do ar no repouso, por ser relativo à temperatura do sensor.

$$T_t = T \cdot \left(1 + \frac{\gamma - 1}{2} \cdot M^2\right) \tag{1.12}$$

Se efeitos de escala forem considerados no modelo (como no caso em que o modelo aerodinâmico é medido a partir de um túnel de vento usando um modelo de escala), será importante saber o número de Reynolds.

O número de Reynolds por unidade de comprimento é dado por:

$$R_e = \frac{\rho \cdot V}{\mu} \tag{1.13}$$

O número de Reynolds com respeito à corda média (portanto é necessário utilizar o modelo geométrico da aeronave) é o adimensional:

$$R_c = \frac{\rho \cdot V \cdot \bar{c}}{\mu} \tag{1.14}$$

Tendo os 3 parâmetros anteriores, formamos o vetor:

$$y_{ad3} = [T_t \quad R_e \quad R_c]$$

3.2 Cálculo de Parâmetros Aerodinâmicos

Para os sinais de saída correspondentes ao modelo aerodinâmico, serão calculados primeiramente 3 adimensionais: as taxas adimensionais de rolagem, arfagem e guinada. Utilizando as taxas de rolagem, arfagem e guinada contidas no vetor de estado e parâmetros geométricos, teremos o vetor adimensional:

$$y_{dl} = \left[\frac{p \cdot b}{2 \cdot V} \quad \frac{q \cdot \bar{c}}{V} \quad \frac{r \cdot b}{2 \cdot V} \right]$$

Utilizando os coeficientes do modelo aerodinâmico da matriz AM, demonstrada com mais detalhes no item “Modelos de Entrada da Aeronave”, é possível calcular os coeficientes de estabilidade e controle das forças e momentos aerodinâmicos.

$$C_{X_a} = C_{X_0} + C_{X_\alpha} \cdot \alpha + C_{X_{\alpha^2}} \cdot \alpha^2 + C_{X_{\alpha^3}} \cdot \alpha^3 + C_{X_q} \cdot \frac{q \cdot \bar{c}}{V} + C_{X_{\delta_r}} \cdot \delta_r + \\ + C_{X_{\delta_f}} \cdot \delta_f + C_{X_{\alpha\delta_f}} \cdot \alpha \cdot \delta_f \tag{2.1}$$

$$C_{Y_a} = C_{Y_0} + C_{Y_\beta} \cdot \beta + C_{Y_p} \cdot \frac{p \cdot b}{2 \cdot V} + C_{Y_r} \cdot \frac{r \cdot b}{2 \cdot V} + C_{Y_{\delta_a}} \cdot \delta_a + C_{Y_{\delta_r}} \cdot \delta_r + \\ + C_{Y_{\delta_r \alpha}} \cdot \delta_r \cdot \alpha + C_{Y_{\dot{\beta}}} \cdot \frac{\dot{\beta} \cdot b}{2 \cdot V} \tag{2.2}$$

$$C_{Z_a} = C_{Z_0} + C_{Z_\alpha} \cdot \alpha + C_{Z_{\alpha^3}} \cdot \alpha^3 + C_{Z_q} \cdot \frac{q \cdot \bar{c}}{V} + C_{Z_{\delta_e}} \cdot \delta_e + C_{Z_{\delta_e \beta^2}} \cdot \delta_e \cdot \beta^2 + \\ + C_{Z_{\delta_f}} \cdot \delta_f + C_{Z_{\alpha\delta_f}} \cdot \alpha \cdot \delta_f \tag{2.3}$$

$$C_{l_a} = C_{l_0} + C_{l_\beta} \cdot \beta + C_{l_p} \cdot \frac{p \cdot b}{2 \cdot V} + C_{l_r} \cdot \frac{r \cdot b}{2 \cdot V} + C_{l_{\delta_a}} \cdot \delta_a + C_{l_{\delta_r}} \cdot \delta_r + C_{l_{\delta_a \alpha}} \cdot \delta_a \cdot \alpha \tag{2.4}$$

$$C_{m_a} = C_{m_0} + C_{m_\alpha} \cdot \alpha + C_{m_{\alpha^2}} \cdot \alpha^2 + C_{m_q} \cdot \frac{q \cdot \bar{c}}{V} + C_{m_{\delta_e}} \cdot \delta_e + C_{m_{\beta^2}} \cdot \beta^2 + \\ + C_{m_r} \cdot \frac{r \cdot b}{2 \cdot V} + C_{m_{\delta_f}} \cdot \delta_f \tag{2.5}$$

$$C_{n_a} = C_{n_0} + C_{n_\beta} \cdot \beta + C_{n_p} \frac{p \cdot b}{2 \cdot V} + C_{n_r} \frac{r \cdot b}{2 \cdot V} + C_{n_{\delta_a}} \cdot \delta_a + C_{n_{\delta_r}} \cdot \delta_r + \\ + C_{n_q} \cdot \frac{q \cdot \bar{c}}{V} + C_{n_{\beta^3}} \cdot \beta^3 \tag{2.6}$$

Note que para o cálculo de C_{Y_a} é necessário a derivada temporal $\dot{\beta}$, o que deixaria a equação implícita. Para que ela fique explícita, o fator será suprimido durante o cálculo dos coeficientes e será levado em consideração após a derivação do vetor de estado.

Será possível então calcular as forças e os momentos aerodinâmicos:

$$X_a = C_{X_a} \cdot q_{dyn} \cdot S \tag{2.7}$$

$$Y_a = C_{Y_a} \cdot q_{dyn} \cdot S \tag{2.8}$$

$$Z_a = C_{Z_a} \cdot q_{dyn} \cdot S \tag{2.9}$$

$$L_a = C_{l_a} \cdot q_{dyn} \cdot S \cdot b \tag{2.10}$$

$$M_a = C_{m_a} \cdot q_{dyn} \cdot S \cdot \bar{c} \tag{2.11}$$

$$N_a = C_{n_a} \cdot q_{dyn} \cdot S \cdot b \tag{2.12}$$

Os valores estarão representados no vetor:

$$FM_{aero} = \begin{bmatrix} X_a & Y_a & Z_a & L_a & M_a & N_a \end{bmatrix}$$

3.3 Cálculo de Parâmetros Propulsores

Foram calculados as forças e momentos devido ao modelo aerodinâmico. Agora serão calculados as forças e momentos propulsores (devido ao modelo do motor). Primeiramente, encontraremos a potência do motor e o aumento da pressão atrás da hélice. O modelo do motor é dependente do tipo de avião. Para aviões com motores de pistão, a potência do motor é dada por:

$$P = 0{,}7355 \cdot \left\{ -326{,}5 + \left[0{,}00412(p_z + 7{,}4)(n + 2010) + (408{,}0 - 0{,}0965 \cdot n)\left(1{,}0 - \frac{\rho}{\rho_0}\right) \right] \right\} \quad (3.1)$$

E o aumento de pressão atrás da hélice pode ser calculado considerando o propulsor como um disco de tração ideal através da expressão:

$$dpt = 0{,}08696 + 191{,}18 \cdot \left(\frac{P}{\rho \cdot V^3 / 2} \right) \quad (3.2)$$

Da mesma forma que para o modelo aerodinâmico, calcularemos os coeficientes das forças e momentos propulsores. Desta vez utilizaremos a matriz EM conforme o item “Modelos de Entrada da Aeronave”:

$$C_{X_p} = C_{X_{dpt}} + C_{X_{\alpha dpt^2}} \cdot \alpha \cdot dpt^2 \quad (3.3)$$

$$C_{Y_p} = 0 \quad (3.4)$$

$$C_{Z_p} = C_{Z_{dpt}} \cdot dpt \quad (3.5)$$

$$C_{l_p} = C_{l_{\alpha^2 dpt}} \cdot \alpha^2 \cdot dpt \quad (3.6)$$

$$C_{m_p} = C_{m_{dpt}} \cdot dpt \quad (3.7)$$

$$C_{n_p} = C_{n_{dpt^3}} \cdot dpt^3 \tag{3.8}$$

Com os coeficientes, calculamos as forças e momentos propulsores:

$$X_p = C_{X_p} \cdot q_{dyn} \cdot S \tag{3.9}$$

$$Y_p = C_{Y_p} \cdot q_{dyn} \cdot S \tag{3.10}$$

$$Z_p = C_{Z_p} \cdot q_{dyn} \cdot S \tag{3.11}$$

$$L_p = C_{l_p} \cdot q_{dyn} \cdot S \cdot b \tag{3.12}$$

$$M_p = C_{m_p} \cdot q_{dyn} \cdot S \cdot \bar{c} \tag{3.13}$$

$$N_p = C_{n_p} \cdot q_{dyn} \cdot S \cdot b \tag{3.14}$$

Alocá-los-emos no vetor:

$$FM_{prop} = \lfloor X_p \quad Y_p \quad Z_p \quad L_p \quad M_p \quad N_p \rfloor$$

3.4 Cálculo dos Parâmetros Gravitacionais e devidos ao Vento

O peso do avião e a força causada pelo vento também influenciam na força total. Para a análise das forças, consideraremos a aceleração da gravidade calculada anteriormente e a velocidade e aceleração do vento, medidas por sensores. Assim, para as forças gravitacionais teremos:

$$X_{gr} = -m \cdot g \cdot \text{sen}(\theta) \tag{4.1}$$

$$Y_{gr} = m \cdot g \cdot \cos(\theta) \cdot \text{sen}(\varphi) \tag{4.2}$$

$$Z_{gr} = m \cdot g \cdot \cos(\theta) \cdot \cos(\varphi) \tag{4.3}$$

$$F_{grav} = \begin{bmatrix} X_{gr} & Y_{gr} & Z_{gr} \end{bmatrix}$$

E para as forças devidas ao vento, que devem ser consideradas em atmosferas turbulentas:

$$X_w = -m \cdot \left(\dot{u}_w + q \cdot w_w - r \cdot v_w \right) \tag{4.4}$$

$$Y_w = -m \cdot \left(\dot{v}_w - p \cdot w_w + r \cdot u_w \right) \tag{4.5}$$

$$Z_w = -m \cdot \left(\dot{w}_w + p \cdot v_w - q \cdot u_w \right) \tag{4.6}$$

$$F_{wind} = \begin{bmatrix} X_w & Y_w & Z_w \end{bmatrix}$$

Portanto, agora temos contribuições de forças externas em relação ao modelo aerodinâmico, ao modelo do motor, à aceleração gravitacional e ao vento. Temos também momentos externos devido ao modelo aerodinâmico e ao modelo propulsor. Assim, somaremos as forças e as colocaremos num único vetor:

$$F_x = X_a + X_p + X_{gr} + X_w \tag{4.7}$$

$$F_y = Y_a + Y_p + Y_{gr} + Y_w \tag{4.8}$$

$$F_z = Z_a + Z_p + Z_{gr} + Z_w \tag{4.9}$$

$$F_{tot} = \begin{bmatrix} F_x & F_y & F_z \end{bmatrix}$$

Igualmente somaremos os momentos externos e os alojaremos num vetor:

$$L = L_a + L_p \tag{4.10}$$

$$M = M_a + M_p \tag{4.11}$$

$$N = N_a + N_p \tag{4.12}$$

$$M_{tot} = \begin{bmatrix} L & M & N \end{bmatrix}$$

O vento pode ter comportamentos distintos numa simulação, afetando de forma significativa a aeronave. Para este trabalho serão apresentados três modelamentos eólicos distintos: vento constante unidirecional, vento seguindo perfil da camada limite da Terra e vento turbulento.

O primeiro tipo é o mais simples e não se aplica no mundo real, servindo a interesses simplesmente de simulação. Considerando ψ_w a direção do vento no plano horizontal, γ_w a direção vertical do vento e V_w o módulo da velocidade do vento fornecido pelo usuário, pode-se calcular as velocidades do vento na horizontal e na vertical:

$$V_{wh} = V_w \cdot \cos(\gamma_w) \tag{4.13}$$

$$w_w^E = V_w \cdot \text{sen}(\gamma_w) \tag{4.14}$$

Também é possível calcular as componentes da velocidade horizontal:

$$u_w^E = V_{wh} \cdot \cos(\psi_w - \pi) \tag{4.15}$$

$$v_w^E = V_{wh} \cdot \text{sen}(\psi_w - \pi) \tag{4.16}$$

Os valores obtidos para as componentes da velocidade do vento foram calculadas para um referencial fixo na Terra, porém no cálculo anterior para as forças aplicadas na aeronave devido ao vento, o referencial adotado era sensível à aeronave. Assim, faz-se necessário uma transformação das coordenadas através da aplicação sucessiva de matrizes de rotação para os ângulos de Euler:

$$\begin{bmatrix} u_w \\ v_w \\ w_w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & \text{sen}(\varphi) \\ 0 & -\text{sen}(\varphi) & \cos(\varphi) \end{bmatrix} \cdot \begin{bmatrix} \cos(\theta) & 0 & -\text{sen}(\theta) \\ 0 & 1 & 0 \\ \text{sen}(\theta) & 0 & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} \cos(\psi) & \text{sen}(\psi) & 0 \\ -\text{sen}(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} u_w^E \\ v_w^E \\ w_w^E \end{bmatrix}$$

Para o perfil da camada limite da Terra, imagina-se que a velocidade do vento aumenta de forma exponencial conforme aumenta-se a altitude, até que esta atinja 300m, quando a velocidade passa a ter comportamento constante.

O perfil numérico atmosférico padrão, que é baseado no padrão de temperatura adotado neste trabalho, considera que a velocidade do vento a 9,15m (altitude de referência para experimentos meteorológicos) é de 1m/s e a partir de 300m, a velocidade passa a ter módulo de 2,87m/s aproximadamente:

$$\begin{aligned} V_w &= V_{w_{9,15}} \cdot \frac{H^{0,2545} - 0,4097}{1,3470} \qquad (0 < H < 300\text{m}) \\ V_w &= 2,86585 \cdot V_{w_{9,15}} \qquad (H \geq 300\text{m}) \end{aligned} \tag{4.17}$$

3.5 Equações de Movimento para Aeronaves

Antes de utilizarmos as equações, calcularemos algumas variáveis auxiliares que nos serão úteis para a simulação do controle de vôo. Essas variáveis são unicamente dependentes do vetor de estado e foram alocadas num vetor:

$$y_{hlp} = [\cos\alpha \quad sen\alpha \quad \cos\beta \quad sen\beta \quad tg\beta \quad sen\psi \quad \cos\psi \quad sen\theta \quad \cos\theta \quad sen\varphi \quad \cos\varphi]$$

É possível calcular as componentes da velocidade da aeronave em relação a um eixo fixo nela mesma:

$$u = V \cdot \cos(\alpha) \cdot \cos(\beta) \tag{5.1}$$

$$v = V \cdot sen(\beta) \tag{5.2}$$

$$w = V \cdot sen(\alpha) \cdot \cos(\beta) \tag{5.3}$$

$$y_{bvel} = [u \quad v \quad w]$$

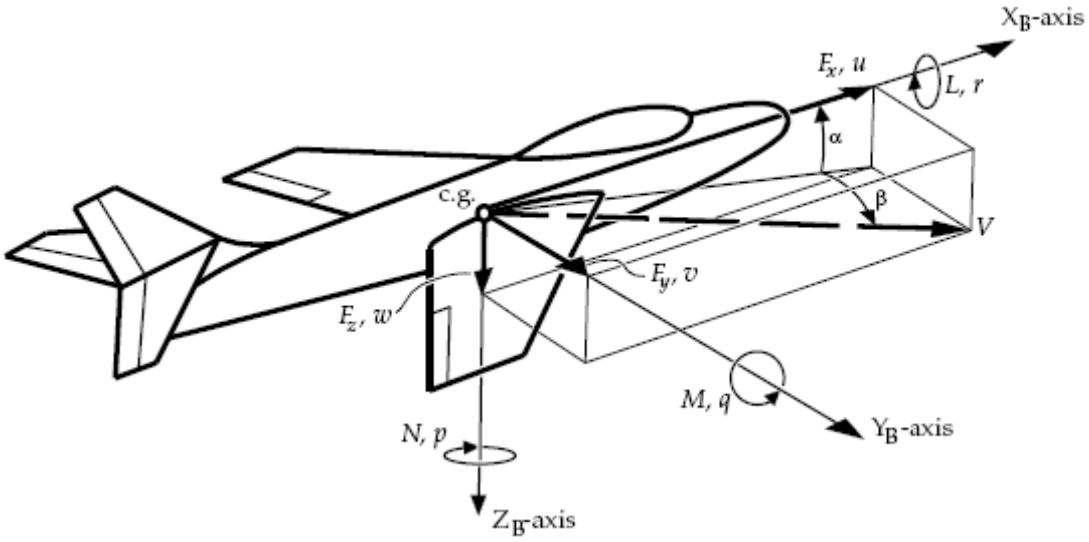

Figura 3 – Orientação das Velocidades Lineares e Angulares em relação a Referencial da Aeronave

Com todos os vetores calculados, finalmente temos condições de calcular a derivada do vetor de estado. Conseguimos calcular as derivadas das velocidades lineares do vetor utilizando o vetor F_{tot} calculado:

$$\dot{V} = \frac{1}{m} \cdot \left[F_x \cdot \cos(\alpha) \cdot \cos(\beta) + F_y \cdot \operatorname{sen}(\beta) + F_z \cdot \operatorname{sen}(\alpha) \cdot \cos(\beta) \right] \tag{5.4}$$

$$\dot{\alpha} = \frac{1}{V \cdot \cos(\beta)} \cdot \left\{ \frac{1}{m} \cdot \left[- F_x \cdot \operatorname{sen}(\alpha) + F_z \cdot \cos(\alpha) \right] \right\} - \left[p \cdot \cos(\alpha) + r \cdot \operatorname{sen}(\alpha) \right] \cdot \operatorname{tg}(\beta) + q \tag{5.5}$$

$$\dot{\beta} = \frac{1}{V} \cdot \left\{ \frac{1}{m} \cdot \left[- F_x \cdot \cos(\alpha) \cdot \operatorname{sen}(\beta) + F_y \cdot \cos(\beta) - F_z \cdot \operatorname{sen}(\alpha) \cdot \operatorname{sen}(\beta) \right] \right\} + p \cdot \operatorname{sen}(\alpha) - r \cdot \cos(\alpha) \tag{5.6}$$

Para o cálculo das derivadas das velocidades angulares utilizaremos os coeficientes inerciais da matriz GM2.

$$\begin{aligned} \dot{p} = {} & P_{pp} \cdot p^2 + P_{pq} \cdot p \cdot q + P_{pr} \cdot p \cdot r + P_{qq} \cdot q^2 + P_{qr} \cdot q \cdot r + P_{rr} \cdot r^2 + P_l \cdot L + \\ & + P_m \cdot M + P_n \cdot N + \left[\dot{p}' \right] \end{aligned} \tag{5.7}$$

$$\begin{aligned} \dot{q} = {} & Q_{pp} \cdot p^2 + Q_{pq} \cdot p \cdot q + Q_{pr} \cdot p \cdot r + Q_{qq} \cdot q^2 + Q_{qr} \cdot q \cdot r + Q_{rr} \cdot r^2 + Q_l \cdot L + \\ & + Q_m \cdot M + Q_n \cdot N + \left[\dot{q}' \right] \end{aligned} \tag{5.8}$$

$$\begin{aligned} \dot{r} = {} & R_{pp} \cdot p^2 + R_{pq} \cdot p \cdot q + R_{pr} \cdot p \cdot r + R_{qq} \cdot q^2 + R_{qr} \cdot q \cdot r + R_{rr} \cdot r^2 + R_l \cdot L + \\ & + R_m \cdot M + R_n \cdot N + \left[\dot{r}' \right] \end{aligned} \tag{5.9}$$

Os termos entre colchetes são referentes aos efeitos causados pelos pares giroscópicos e serão desprezados no modelo. Para estas derivadas, a aeronave deve ser considerada como um corpo rígido durante o movimento, a massa do avião é constante e a Terra é assumida ser fixa no espaço, sendo a rotação e a curvatura do planeta desprezadas.

Como as forças e os momentos externos dependem das variáveis de movimento do avião, suas derivadas são necessárias para as próximas iterações da simulação.

As derivadas dos ângulos de Euler, que dependem das velocidades angulares do próprio vetor de estado, são dadas por:

$$\dot{\psi} = \frac{q \cdot \mathrm{sen}(\varphi) + r \cdot \cos(\varphi)}{\cos(\theta)} \tag{5.10}$$

$$\dot{\theta} = q \cdot \cos(\varphi) - r \cdot \mathrm{sen}(\varphi) \tag{5.11}$$

$$\dot{\varphi} = p + \dot{\psi} \cdot \mathrm{sen}(\theta) \tag{5.12}$$

As derivadas das coordenadas horizontais em relação à superfície terrestre não são estritamente necessárias, porém serão calculadas para uso em navegação. Juntamente com elas calcularemos a derivada da altitude, que é muito importante para o controle.

Sendo:

$$u_e = u + u_w \tag{5.13}$$

$$v_e = v + v_w \tag{5.14}$$

$$w_e = w + w_w \tag{5.15}$$

Calculamos então as derivadas:

$$\begin{aligned}\dot{x}_e = &\{u_e \cdot \cos(\theta) + [v_e \cdot \mathrm{sen}(\varphi) + w_e \cdot \cos(\varphi)] \cdot \mathrm{sen}(\theta)\} \cdot \cos(\psi) - \\ &+ [v_e \cdot \cos(\varphi) - w_e \cdot \mathrm{sen}(\varphi)] \cdot \mathrm{sen}(\psi)\end{aligned} \tag{5.16}$$

$$\begin{aligned}\dot{y}_e = &\{u_e \cdot \cos(\theta) + [v_e \cdot \mathrm{sen}(\varphi) + w_e \cdot \cos(\varphi)] \cdot \mathrm{sen}(\theta)\} \cdot \mathrm{sen}(\psi) + \\ &+ [v_e \cdot \cos(\varphi) - w_e \cdot \mathrm{sen}(\varphi)] \cdot \cos(\psi)\end{aligned} \tag{5.17}$$

$$\dot{H} = u_e \cdot \mathrm{sen}(\theta) - [v_e \cdot \mathrm{sen}(\varphi) + w_e \cdot \cos(\varphi)] \cdot \cos(\theta) \tag{5.18}$$

Todas as derivadas foram calculadas mas a derivada do ângulo de sideslip precisa sofrer um ajuste para compensar o fato do coeficiente $C_{Y_{\dot{\beta}}}$ ter sido

desprezado (lembrando que ele foi desprezado para tornar a equação do coeficiente de força aerodinâmico explícita). Corrigindo a derivada:

$$\dot{\beta}_{corr} = \dot{\beta} \cdot \left[1 - \frac{\rho \cdot S \cdot b}{4 \cdot m} \cdot C_{Y_{\dot{\beta}}} \cdot \cos(\beta) \right] \tag{5.19}$$

Então $\dot{\beta}_{corr}$ será o novo $\dot{\beta}$. E o vetor contendo as derivadas do vetor de estado fica:

$$\dot{x} = \left[\dot{V} \quad \dot{\alpha} \quad \dot{\beta} \quad \dot{p} \quad \dot{q} \quad \dot{r} \quad \dot{\psi} \quad \dot{\theta} \quad \dot{\varphi} \quad \dot{x}_e \quad \dot{y}_e \quad \dot{H} \right]$$

3.6 Parâmetros Adicionais

Além dos sinais de saída anteriores, podemos calcular outros parâmetros, que não são necessários para as equações de estado e nem serão utilizados por outros modelos específicos, mas que podem ser úteis para futura análise.

Podemos encontrar variáveis relacionadas à rota de vôo como o ângulo do trajeto de vôo (muito útil durante a aproximação do avião à pista de pouso), a aceleração no trajeto de vôo (que é a aceleração na direção verdadeira do ar), o ângulo de azimute e o ângulo de inclinação da curva. Podem ser calculados por:

$$\gamma = \operatorname{arcsen}\left(\frac{\dot{H}}{V}\right) \tag{6.1}$$

$$\mathrm{fpa} = \frac{\dot{V}}{g_0} \tag{6.2}$$

$$\chi = \beta + \psi \tag{6.3}$$

$$\phi = \operatorname{arcsen}\left[\operatorname{sen}(\varphi)\cdot\cos(\theta)\right] \tag{6.4}$$

Alocando as variáveis num vetor:

$$y_{fp} = \begin{bmatrix}\gamma & \mathrm{fpa} & \chi & \phi\end{bmatrix}$$

Também como sinais de saída serão indicadas as componentes da aceleração do vento, calculadas a partir das componentes da velocidade.

$$\dot{u} = \dot{V}\cdot\cos(\alpha)\cdot\cos(\beta) - V\cdot\left[\dot{\alpha}\cdot\operatorname{sen}(\alpha)\cdot\cos(\beta) + \dot{\beta}\cdot\cos(\alpha)\cdot\operatorname{sen}(\beta)\right] \tag{6.5}$$

$$\dot{v} = \dot{V}\cdot\operatorname{sen}(\beta) + V\cdot\dot{\beta}\cdot\cos(\beta) \tag{6.6}$$

$$\dot{w} = \dot{V}\cdot\operatorname{sen}(\alpha)\cdot\cos(\beta) + V\cdot\left[\dot{\alpha}\cdot\cos(\alpha)\cdot\cos(\beta) + \dot{\beta}\cdot\operatorname{sen}(\alpha)\cdot\operatorname{sen}(\beta)\right] \tag{6.7}$$

Formamos então o vetor com as acelerações:

$$y_{uvw} = \begin{bmatrix} \dot{u} & \dot{v} & \dot{w} \end{bmatrix}$$

Finalmente calcularemos também acelerações e forças específicas, que são as saídas de acelerômetros, no centro de gravidade da aeronave. As acelerações cinemáticas ao longo de cada eixo (X, Y, Z) fixo no avião são dadas por:

$$a_{x,k} = \frac{F_x}{m \cdot g_0} \tag{6.8}$$

$$a_{y,k} = \frac{F_y}{m \cdot g_0} \tag{6.9}$$

$$a_{z,k} = \frac{F_z}{m \cdot g_0} \tag{6.10}$$

As saídas dos acelerômetros em cada eixo (X, Y, Z) são dadas por:

$$A_x = \frac{F_x - X_{gr}}{m \cdot g_0} \tag{6.11}$$

$$A_y = \frac{F_y - Y_{gr}}{m \cdot g_0} \tag{6.12}$$

$$A_z = \frac{F_z - Z_{gr}}{m \cdot g_0} \tag{6.13}$$

Os 6 parâmetros formarão o vetor:

$$y_{acc} = \begin{bmatrix} A_x & A_y & A_z & a_{x,k} & a_{y,k} & a_{z,k} \end{bmatrix}$$

Um outro parâmetro que merece destaque é a velocidade mínima suportada pelo avião para que não que haja o *stall*, condição que a aeronave não consegue

sustentação. Para o cálculo da velocidade mínima, iguala-se o peso com a sustentação, condição limite.

Sabe-se que a sustentação é obtida através de:

$L = C_L \cdot q_{dyn} \cdot S$, onde C_L é o coeficiente de sustentação, obtido graficamente.

Para o caso estudado, será utilizada uma aproximação linear obtida da aerodinâmica ideal através da Teoria do Aerofólio fino para o coeficiente de sustentação, sendo função do ângulo de ataque. Na aproximação [RAYMER 2003], é assumida asa com comprimento infinito e o ângulo de ataque tem limite máximo de aproximadamente 15º (consideração válida já que o algoritmo de controle reage a um aumento superior a 0.25 rad ~ 15º, aumentando automaticamente a rotação do motor, forçando assim sua queda):

$$C_L = 2 \cdot \pi \cdot \alpha \tag{6.14}$$

Portanto, através da eqüação seguinte, encontra-se a velocidade mínima suportada, que é a velocidade de *stall* V_{st}:

$$W = L \Rightarrow m \cdot g = C_L \cdot \frac{\rho \cdot V_{st}^{\,2}}{2} \cdot S \tag{6.15}$$

Para a interface gráfica no programa Google Earth é interessante os cálculos da latitude e longitude, também importantes para o controle de trajetória previamente estabelecidade. Especificando o usuário as latitudes e longitudes iniciais lat_0 e $long_0$ e sabendo que na superfície terrestre a variação da latitude é fixa em qualquer local, podemos calculá-la por:

$$lat = lat_0 + \frac{x_e}{111000} \tag{6.16}$$

Já que a latitude tem variação de aproximadamente 111km a cada grau. Todavia, a longitude não tem a mesma propriedade, sendo variável conforma a latitude. É facilmente vislumbrada a relação:

$$\text{long} = \text{long}_0 + \frac{y_e}{\frac{2\pi \cdot R_{\text{Terra}} \cdot \cos(\text{lat})}{360°}} \tag{6.17}$$

Assim, concluímos os parâmetros calculados para o modelo. Caso seja necessário adicionar um novo vetor contendo parâmetros, é necessário modificar o programa após o cálculo da derivada, juntamente com os 3 últimos vetores.

3.7. Novo Vetor de Estado

O próximo passo é a integração da derivada do vetor de estado para que um novo vetor de estado associado há um tempo transcorrido dê continuidade às iterações, calculando novamente os parâmetros a cima. Para isso antes é necessário escolher um método para a integração numérica: [SALLET 2004] e [GERSHENFELD 1999].

Inicialmente nós temos os dados referentes a um tempo t_n, que são o vetor de estado x_n, a derivada do vetor de estado $\dot{x}_n$ e queremos encontrar o vetor de estado do tempo t_{n+1}. Para isso consideraremos o intervalo de tempo fixo entre dois vetores de estado:

$$t_{n+1} - t_n = h_n \text{ , onde } h_n \text{ é um valor pequeno.}$$

Assim, utilizaremos métodos que sejam capazes de inferir x_{n+1} apenas utilizando os dados anteriores. No caso, foram utilizados 3 métodos distintos.

O primeiro método é o Método de Euler, uma aproximação mais grosseira que as demais, utilizando apenas valores do início de cada intervalo. Os demais métodos serão um refinamento dele. Considerando a seguinte a seguinte equação diferencial:

$$\frac{dy}{dx} = f(x, y),$$

$$y_{i+1} = y_i + f(x_i, y_i) \cdot \Delta x \Rightarrow x_{n+1} = x_n + \dot{x}_n \cdot h_n \qquad (7.1)$$

Este método “trunca” a expansão de Taylor no segundo termo, sendo portanto um método $O(\Delta t)$, com erro $O(\Delta t^2)$. Os demais métodos diminuem a ordem do erro.

Um método mais refinado é o Método de Heun (ou Método de Euler aperfeiçoado). Ele considera valores intermediários da minha função **f**, sendo necessário primeiramente calcular uma primeira aproximação utilizando o Método de Euler:

$$f_i = f(x_i, y_i) \tag{7.2}$$

$$y_{i+1} = y_i + f(x_i, y_i) \cdot \Delta x \tag{7.3}$$

$$f_{i+1} = f(x_{i+1}, y_{i+1}) \tag{7.4}$$

$$y_{i+1} = y_i + \frac{1}{2} \cdot (f_i + f_{i+1}) \cdot \Delta x \tag{7.5}$$

Ambos os métodos utilizados são de passo único, que diferentemente dos métodos de passo múltiplo, são mais difíceis de serem utilizados, porém são menos suscetíveis a problemas decorrentes de instabilidade da solução.

O terceiro método utilizado é o Método de Runge-Kutta de quarta ordem. O cálculo de x_{n+1} inclui mais pontos no intervalo de tempo, tendo o cálculo da derivada termos de $O(\Delta t^4)$, com erro $O(\Delta t^5)$. Dos 3 métodos é o mais utilizado, sendo calculado por:

$$f_0 = f(x_0, y_0) \tag{7.6}$$

$$f_1 = f[x_0 + (h/2), y_0 + (h/2) \cdot f_0] \tag{7.7}$$

$$f_2 = f[x_0 + (h/2), y_0 + (h/2) \cdot f_1] \tag{7.8}$$

$$f_3 = f[x_0 + h, y_0 + h \cdot f_2] \tag{7.9}$$

$$y_{i+1} = y_i + \frac{h}{6} \cdot (f_0 + 2 \cdot f_1 + 2 \cdot f_2 + f_3) \tag{7.10}$$

Note que todos os métodos utilizados utilizaram intervalos de tempo constantes, existem outros métodos que controlam o tamanho do intervalo automaticamente para obter uma acurácia específica. Um exemplo é o método dos pares Dormand-Prince.

3.8 Modelos de Entrada da Aeronave

Para a inicialização do cálculo dos sinais de saída, são necessárias informações sobre a aerodinâmica do modelo, sobre o motor da aeronave e sobre sua geometria. Essas informações não mudarão ao longo da execução do controle de vôo e portanto são constantes próprias de cada tipo de aeronave e motor.

O modelo aerodinâmico utiliza coeficientes para o cálculo das forças e momentos não-dimensionais tendo como referência um eixo fixo na aeronave (ortogonal, destrógiro e com origem no seu centro de gravidade). Os coeficientes estão alocados numa matriz AM (para o modelo *Beaver*, [TJEE 1988]) como mostra o esquema abaixo:

$$AM = \begin{bmatrix}
C_{X_0} & C_{Y_0} & C_{Z_0} & C_{l_0} & C_{m_0} & C_{n_0} \\
C_{X_\alpha} & 0 & C_{Z_\alpha} & 0 & C_{m_\alpha} & 0 \\
C_{X_{\alpha^2}} & 0 & 0 & 0 & C_{m_{\alpha^2}} & 0 \\
C_{X_{\alpha^3}} & 0 & C_{Z_{\alpha^3}} & 0 & 0 & 0 \\
0 & C_{Y_\beta} & 0 & C_{l_\beta} & 0 & C_{n_\beta} \\
0 & 0 & 0 & 0 & C_{m_{\beta^2}} & 0 \\
0 & 0 & 0 & 0 & 0 & C_{n_{\beta^3}} \\
0 & C_{Y_p} & 0 & C_{l_p} & 0 & C_{n_p} \\
C_{X_q} & 0 & C_{Z_q} & 0 & C_{m_q} & C_{n_q} \\
0 & C_{Y_r} & 0 & C_{l_r} & C_{m_r} & C_{n_r} \\
0 & 0 & C_{Z_{\delta_e}} & 0 & C_{m_{\delta_e}} & 0 \\
C_{X_{\delta_f}} & 0 & C_{Z_{\delta_f}} & 0 & C_{m_{\delta_f}} & 0 \\
0 & C_{Y_{\delta_a}} & 0 & C_{l_{\delta_a}} & 0 & C_{n_{\delta_a}} \\
C_{X_{\delta_r}} & C_{Y_{\delta_r}} & 0 & C_{l_{\delta_r}} & 0 & C_{n_{\delta_r}} \\
C_{X_{\alpha\delta_f}} & 0 & C_{Z_{\alpha\delta_f}} & 0 & 0 & 0 \\
0 & C_{Y_{\alpha\delta_r}} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & C_{l_{\alpha\delta_a}} & 0 & 0 \\
0 & 0 & C_{Z_{\beta^2\delta_e}} & 0 & 0 & 0 \\
0 & C_{Y_{\dot{\beta}}} & 0 & 0 & 0 & 0
\end{bmatrix}$$

Para o modelo do motor, os coeficientes que serão utilizados para o cálculo das forças e dos momentos propulsores estão contidos na matriz EM.

$$EM = \begin{bmatrix}
C_{X_{dpt}} & 0 & C_{Z_{dpt}} & 0 & C_{m_{dpt}} & 0 \\
0 & 0 & 0 & 0 & 0 & C_{n_{dpt^3}} \\
C_{X_{\alpha dpt^2}} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & C_{l_{\alpha^2 dpt}} & 0 & 0
\end{bmatrix}$$

Os dados relativos à geometria e à massa e sua distribuição estão contidos no vetor GM1. É assumido que a massa e a sua distribuição ao longo do avião permanecem constantes durante a simulação. Os dados contidos em GM1 são a corda aerodinâmica média, a largura da asa, a área da asa, os momentos e os produtos de inércia em relação ao eixo fixo no avião e a massa da aeronave.

$$GM1 = \begin{bmatrix} \bar{c} & b & S & I_{xx} & I_{yy} & I_{zz} & J_{xy} & J_{xz} & J_{yz} & m \end{bmatrix}$$

Com o vetor GM1 podemos encontrar os coeficientes inerciais e alocá-los numa matriz GM2 para utilizá-la em cálculo futuro (derivada temporal das velocidades angulares do vetor de estado).

Cálculo dos Coeficientes Inerciais:

O tensor de tensões pode ser representado por:

$$\vec{I} = \begin{bmatrix} I_x & -J_{xy} & -J_{xz} \\ -J_{yx} & I_y & -J_{yz} \\ -J_{zx} & -J_{zy} & I_z \end{bmatrix}$$

$$\therefore \det \vec{I} = I_x \cdot I_y \cdot I_z - 2 \cdot J_{xy} \cdot J_{xz} \cdot J_{yz} - I_x \cdot J_{yz}^{\,2} - I_y \cdot J_{xz}^{\,2} - I_z \cdot J_{xy}^{\,2}$$

Adotando:

$$I_1 = I_y \cdot I_z - J_{yz}^{\,2}$$

$$I_2 = J_{xy} \cdot I_z + J_{yz} \cdot J_{xz}$$

$$I_3 = J_{xy} \cdot J_{yz} + I_x \cdot J_{xz}$$

$$I_4 = I_x \cdot I_z - J_{xz}^{\,2}$$

$$I_5 = I_x \cdot J_{yz} + J_{xy} \cdot J_{xz}$$

$$I_6 = I_x \cdot I_y - J_{xy}^{\,2}$$

Sendo T_l , T_m, T_n, T_{pp}, T_{pq}, T_{pr}, T_{qq}, T_{qr} e T_{rr} uma representação para os coeficientes inerciais P, Q e R, teremos:

$$T_l = \frac{I_i}{\det \vec{I}}$$

$$T_m = \frac{I_j}{\det \vec{I}}$$

$$T_n = \frac{I_k}{\det \vec{I}}$$

$$T_{pp} = \frac{-\left(J_{xz} \cdot I_j - J_{xy} \cdot I_z\right)}{\det \vec{I}}$$

$$T_{pq} = \frac{\left[J_{xz} \cdot I_i - J_{yz} \cdot I_j - \left(I_y - I_x\right) \cdot I_z\right]}{\det \vec{I}}$$

$$T_{pr} = \frac{-\left[J_{xy} \cdot I_i + \left(I_x - I_z\right) \cdot I_j - J_{yz} \cdot I_z\right]}{\det \vec{I}}$$

$$T_{qq} = \frac{\left(J_{yz} \cdot I_i - J_{xy} \cdot I_k\right)}{\det \vec{I}}$$

$$T_{qr} = \frac{-\left[\left(I_z - I_y\right) \cdot I_i - J_{xy} \cdot I_j + J_{xz} \cdot I_k\right]}{\det \vec{I}}$$

$$T_{rr} = \frac{-\left(J_{yz} \cdot I_i - J_{xz} \cdot I_j\right)}{\det \vec{I}}$$

Se considerarmos: i = 1; j = 2; k = 3, encontraremos os coeficientes inerciais para P.

Da mesma forma: i = 2; j = 4; k = 5, define Q.

E i = 3; j = 5; k = 6 define R.

Assim, podemos montar a matriz GM2.

$$GM\ 2 = \begin{bmatrix} P_l & P_m & P_n & P_{pp} & P_{pq} & P_{pr} & P_{qq} & P_{qr} & P_{rr} \\ Q_l & Q_m & Q_n & Q_{pp} & Q_{pq} & Q_{pr} & Q_{qq} & Q_{qr} & Q_{rr} \\ R_l & R_m & R_n & R_{pp} & R_{pq} & R_{pr} & R_{qq} & R_{qr} & R_{rr} \end{bmatrix}$$

Essas 4 matrizes são os valores que definem características imutáveis da aeronave. Para que o sistema de controle atue são necessárias condições iniciais. Uma das condições é o vetor de estado no instante inicial da medição (t = 0), que representa variáveis importantes a serem controladas como a velocidade da aeronave, a altitude, ângulos e coordenadas em relação a um eixo fixo na Terra.

$$x = \begin{bmatrix} V & \alpha & \beta & p & q & r & \psi & \theta & \varphi & x_e & y_e & H \end{bmatrix}$$

As outras condições iniciais são as deflexões das superfícies de controle, a rotação do motor, a pressão de admissão do mesmo, a velocidade e a aceleração do vento. Estes parâmetros devem ser constantemente atualizados para a atuação do controle automático, sendo as únicas entradas do modelo.

$$u_{aero_0} = \begin{bmatrix} \delta_e & \delta_a & \delta_r & \delta_f \end{bmatrix}$$

$$u_{prop_0} = \begin{bmatrix} n & p_z \end{bmatrix}$$

$$u_{wind_0} = \begin{bmatrix} u_w & v_w & w_w & \dot{u}_w & \dot{v}_w & \dot{w}_w \end{bmatrix}$$

4 CONTROLE DA AERONAVE

O algortimo desenvolvido permite diversas formas de controle da aeronave. Como visto no capítulo 3.8, as variáveis controláveis estão relacionadas com as superfícies de controle, a rotação e pressão de admissão do motor, a velocidade e aceleração do cento.

As superfícies de controle são os *ailerons*, localizados na asa, permitindo a rolagem do avião. A direção do movimento das superfícies é contrária uma à outra, sendo sua deflexão δ_a a diferença entre os ângulos do aileron da asa direita em relação ao da asa esquerda. Os *elevators* (profundores) permitem a arfagem, ou seja, movimento de picar quando o nariz do avião está inclinado para baixo e movimento de cabrar estando o nariz para cima. A deflexão da superfície é dada por δ_e. O *rudder* (leme) permite o movimento em torno de um eixo imaginário na vertical, movimento conhecido como guinada e sendo a deflexão representada por δ_r. Os *flaps* são superfícies que favorecem o pouso e decolagem da aeronave por ajudar a sustentabilidade da aeronave a baixas velocidades (já que diminuem o arrasto aerodinâmico por aumentar a superfície projetada na direção perpendicular ao movimento). Estão localizadas nos bordos de fuga das asas e representadas por δ_f.

A potência do motor da aeronave está diretamente relacionada com a rotação do motor e a pressão de admissão, que são as entradas primárias para o controle do motor.

Assim, o controle manual é feito diretamente pelo usuário através da interface gráfica por botões que decrementam ou incrementam as deflexões ou a rotação e admissão do motor, com determinados limites estabelecidos. Outra forma possível é o controle através de comando por teclado em campo de controle. Todos os comandos são interpretados em tempo real.

O controle automático pode ser realizado de 4 formas distintas: vôo linear nivelado, vôo direcionado a ângulo específico, vôo direcionado a coordenada absoluta (latitude, longitude), vôo automático através de pontos intermediários (*way points*).

Vôo Linear Nivelado

O vôo linear nivelado exige a estabilidade da aeronave. Para isso primeiramente será forçado que as derivadas da atitude do veículo sejam nulas, ou seja, que as velocidades angulares nas três direções e representadas pelos três parâmetros do vetor de estado (p, q, r) tendam a zero. Não há uma forma direta pelas equações definidas anteriormente de atuar nas superfícies de controle e obter o resultado desejado. Assim, fazendo as manipulações para o método de Euler:

$$p_{NOVO} = 0 \Rightarrow \dot{p} = \frac{-p}{dt} \Rightarrow P_l \cdot L + P_n \cdot N + P_{pq} \cdot p \cdot q + P_{qr} \cdot q \cdot r = \frac{-p}{dt}$$

$$q_{NOVO} = 0 \Rightarrow \dot{q} = \frac{-q}{dt} \Rightarrow Q_m \cdot M + Q_{pp} \cdot p^2 + Q_{pr} \cdot p \cdot r + Q_{rr} \cdot r^2 = \frac{-q}{dt}$$

$$r_{NOVO} = 0 \Rightarrow \dot{r} = \frac{-r}{dt} \Rightarrow R_l \cdot L + R_n \cdot N + R_{pq} \cdot p \cdot q + R_{qr} \cdot q \cdot r = \frac{-r}{dt}$$

Porém,

$$L = C_{l_a} \cdot q_{dyn} \cdot S \cdot b + L_p = (C_{l_0} + C_{l_\beta} \cdot \beta + C_{l_p} \cdot \frac{p \cdot b}{2 \cdot V} + C_{l_r} \cdot \frac{r \cdot b}{2 \cdot V} + C_{l_{\delta_a}} \cdot \delta_a + C_{l_{\delta_r}} \cdot \delta_r + C_{l_{\delta_a \alpha}} \cdot \delta_a \cdot \alpha) \cdot q_{dyn} \cdot S \cdot b + L_p$$

$$M = C_{m_a} \cdot q_{dyn} \cdot S \cdot \bar{c} + M_p = (C_{m_0} + C_{m_\alpha} \cdot \alpha + C_{m_{\alpha^2}} \cdot \alpha^2 + C_{m_q} \cdot \frac{q \cdot \bar{c}}{V} + C_{m_{\delta_e}} \cdot \delta_e + C_{m_{\beta^2}} \cdot \beta^2 + C_{m_r} \cdot \frac{r \cdot b}{2 \cdot V} + C_{m_{\delta_f}} \cdot \delta_f) \cdot q_{dyn} \cdot S \cdot \bar{c} + M_p$$

$$N = C_{n_a} \cdot q_{dyn} \cdot S \cdot b + N_p = (C_{n_0} + C_{n_\beta} \cdot \beta + C_{n_p} \frac{p \cdot b}{2 \cdot V} + C_{n_r} \frac{r \cdot b}{2 \cdot V} + C_{n_{\delta_a}} \cdot \delta_a + C_{n_{\delta_r}} \cdot \delta_r + C_{n_q} \cdot \frac{q \cdot \bar{c}}{V} + C_{n_{\beta^3}} \cdot \beta^3) \cdot q_{dyn} \cdot S \cdot b + N_p$$

Assim, fixando uma determinada deflexão para os *flaps* δ_f, é possível escrever com as equações de q_{NOVO} e M uma expressão direta para a deflexão do *elevator* δ_e:

$$\delta_e = \frac{\frac{-\left(\frac{q}{dt} + Q_{pp} \cdot p^2 + Q_{pr} \cdot p \cdot r + Q_{rr} \cdot r^2\right)}{Q_m} - M_p}{q_{dyn} \cdot S \cdot \bar{c}} - C_{m_0} - C_{m_\alpha} \cdot \alpha - C_{m_{\alpha^2}} \cdot \alpha^2 - C_{m_q} \cdot \frac{q \cdot \bar{c}}{V} - C_{m_{\beta^2}} \cdot \beta^2 - C_{m_r} \cdot \frac{r \cdot b}{2 \cdot V} - C_{m_{\delta_f}} \cdot \delta_f}{C_{m_{\delta_e}}}$$

Para as demais deflexões, como é possível formar um sistema linear com duas equações envolvendo L e p_{NOVO} com N e r_{NOVO}, as entradas dos *ailerons* δ_a e do *rudder* δ_r podem ser estabelecidas por:

$$\delta_{a1} = -\frac{\left(\frac{p}{dt} + P_{pq} \cdot p \cdot q + P_{qr} \cdot q \cdot r - \frac{\left(\frac{r}{dt} - R_{pq} \cdot p \cdot q - R_{qr} \cdot q \cdot r \right) \cdot P_n}{R_n} \right) - L_p}{P_l - \frac{R_l}{R_n} \cdot P_n}$$

$$\delta_{a2} = -\frac{\frac{\left(\frac{r}{dt} + R_l \cdot \delta_{a1} + R_{pq} \cdot p \cdot q + R_{qr} \cdot q \cdot r \right)}{R_n} - N_p}{q_{dyn} \cdot S \cdot b}$$

$$\delta_a = \frac{\begin{array}{l} \delta_{a1} - \frac{\delta_{a2}}{C_{n_{\delta_r}}} \cdot C_{l_{\delta_r}} - C_{l_0} - C_{l_\beta} \cdot \beta - C_{l_p} \cdot \frac{p \cdot b}{2 \cdot V} + C_{l_r} \cdot \frac{r \cdot b}{2 \cdot V} + \\ (C_{n_0} + C_{n_\beta} \cdot \beta + C_{n_p} \frac{p \cdot b}{2 \cdot V} + \\ \frac{C_{n_r} \frac{r \cdot b}{2 \cdot V} + C_{n_q} \cdot \frac{q \cdot \bar{c}}{V} + C_{n_{\beta^3}} \cdot \beta^3)}{C_{n_{\delta_r}}} \cdot C_{l_{\delta_r}} \end{array}}{C_{l_{\delta_a}} + C_{l_{\alpha\delta_a}} \cdot \alpha - \frac{C_{n_{\delta_a}}}{C_{n_{\delta_r}}} \cdot C_{l_{\delta_r}}}$$

$$\delta_r = \frac{\delta_{a2} - \left(C_{n_0} + C_{n_\beta} \cdot \beta + C_{n_p} \frac{p \cdot b}{2 \cdot V} + C_{n_r} \frac{r \cdot b}{2 \cdot V} + C_{n_q} \cdot \frac{q \cdot \bar{c}}{V} + C_{n_{\beta^3}} \cdot \beta^3 + \delta_a \cdot C_{n_{\delta_a}} \right)}{C_{n_{\delta_r}}}$$

Assim, o controle automático estabelece a cada iteração os valores dessas três deflexões, para que as velocidades angulares sejam nulas. Porém somente essas condições não satisfazem um vôo linear nivelado já que também é necessário

que o ângulo de Euler da rolagem seja nulo ou a aeronave tenderia a virar para determinada direção continuamente e por conseqüência perderia a estabilidade.

Assim, através de um processo iterativo, caso o ângulo φ seja superior a 0 o *aileron* diminuirá levemente a deflexão para contrabalançar e da mesma forma, caso φ seja inferior a 0, o *aileron* aumentará. Esse valor é aumentado ou diminuído sobre o valor anteriormente calculado, sendo apenas um fator de correção. Quando as iterações convergirem para um valor próximo de zero, o fator é ignorado pelo algoritmo.

Da mesma forma pode haver a tendência do ângulo de ataque ter um valor muito maior que o ângulo de Euler da arfagem e isso acarretaria uma perda de altitude da aeronave. Para isso também há um fator de correção do *elevator* para que ambos valores estejam próximos. Porém pode haver o caso dos valores aumentarem concomitantemente. Assim, a rotação do motor é aumentada quando ambos sobem e diminuída caso desçam.

Com essas condições a aeronave terá um vôo linear nivelado. Porém, essa lógica funciona apenas para um caso de simulação pura e não para o mundo real. Esse fato é facilmente observável quando notamos a dependência dos parâmetros no sistema de controle. Um piloto só tem acesso às deflexões das superfícies de controle e às variáveis do motor. Assim, o algoritmo só pode ter dependência dos parâmetros que sensores e outros dispositivos eletro-mecânicos possam disponibilizar. De fato, é necessário que o sistema de controle seja independente das características aerodinâmicas e geométricas da aeronave.

Assim, no algoritmo desenvolvido foram utilizadas as seguintes malhas de controle para cada superfície controlada [RAUW 2001], caso estudado para o modelo “Beaver”. Os ganhos foram calculados utilizando as simulações desenvolvidas, por meio de iterações:

Para o controle dos profundores, Figura 4, a malha mantem o ângulo de *pitch* constante. Para ajudar a minimizar as variações do ângulo de *pitch* em resposta aos movimentos de rolagem da aeronave, foi adicionado um compensador.

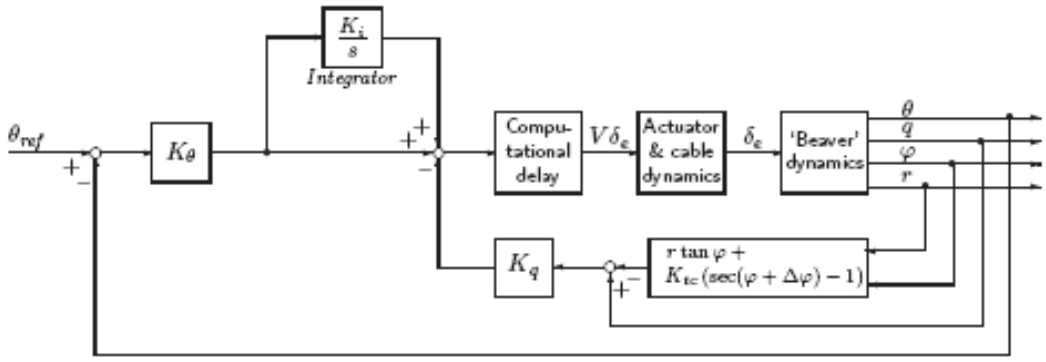

Figura 4 – Diagrama de Blocos para Controle dos Profundores

Para o controle dos *ailerons* e do leme, foi usada a seguinte malha:

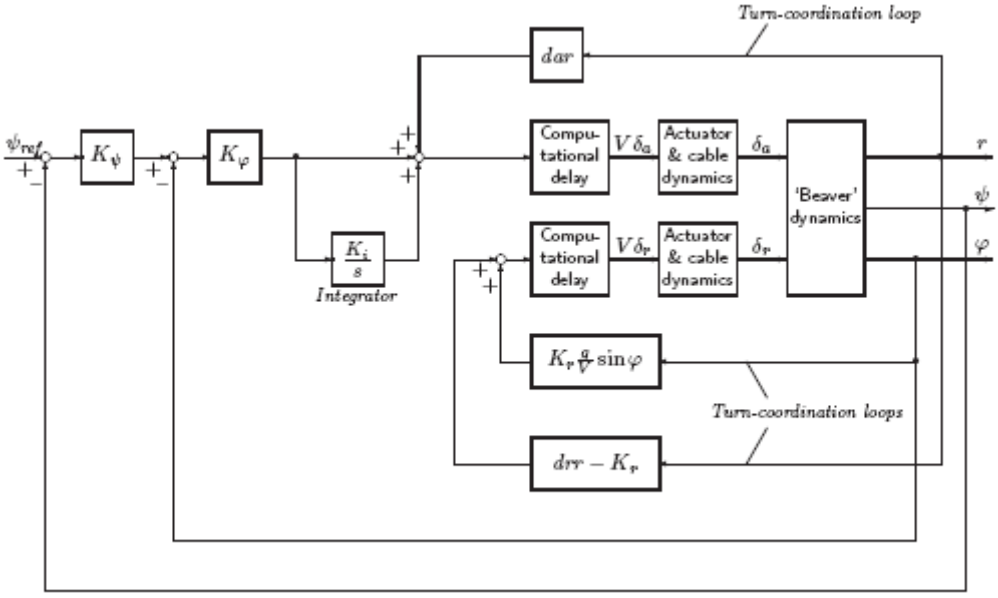

Figura 5 – Diagrama de Blocos para Controle dos *Ailerons* e do Leme

Esta malha objetiva atingir uma certa direção especificada. No caso do vôo linear nivelado, o algoritmo toma esta direção como o valor anterior ao acionamento do controle automático.

Em ambas as malhas, há a necessidade de levar em consideração a dinâmica dos atuadores e cabos. O algoritmo resolve então um sistema de equações de espaço de estados com constantes relativas aos atuadores. Existem também valores limitadores para cada superfície de controle, com objetivo de aproximar o modelo da realidade.

Vôo direcionado a Ângulo

O ângulo da direção em graus pode ser calculado para o 1º quadrante, ou seja, quando os novos valores de x_e e y_e são maiores que os anteriores, como:

$$head = \arctan\left(\frac{y_{e\,NOVO} - y_e}{x_{e\,NOVO} - x_e}\right) \cdot \frac{180}{\pi}$$

Quando o usuário requisita determinada direção, o controle automático verifica a direção atual e caso sejam distintas, o *rudder* e os *ailerons* atuam conforme a malha da figura 5 para consertar a atitude da aeronave. Estando a direção correta, o controle age como no caso do vôo linear nivelado.

Vôo direcionado a Coordenada Absoluta

Indicando uma coordenada na forma de latitude e longitude, o controle automático calcula a direção necessária para atingir o ponto dada a posição em cada instante. Assim, ele age como no caso do vôo direcionado a ângulo, porém calculando a direção como:

$$head = \arctan\left(\frac{long_{\,NOVO} - long}{lat_{\,NOVO} - lat}\right) \cdot \frac{180}{\pi}$$

Lembrando que este é o caso do primeiro quadrante.

Vôo Automático através de *Way Points*

O controle permite ainda que diversos pontos sejam especificados, formando assim uma trajetória qualquer. A aeronave fará um vôo direcionado a Coordenada Absoluta para cada ponto. E quando ela atinge a região, passa a ser guiada para o ponto seguinte. Terminando a trajetória, automaticamente o controle passa atuar como vôo linear nivelado.

Este é o modo UAV, o mais completo, em que é possível gerar uma missão para a aeronave e esta a executa de forma autônoma. Foram realizados diversos testes, como demonstrado a seguir, em que todo o sistema de controle agiu conjuntamente nos diversos vôos realizados.

5 INTERFACE GRÁFICA

O algoritmo da simulação e controle permite ao usuário uma interface gráfica, na qual ele tem acesso aos dados mais importantes para a simulação em tempo real, como as variáveis do vetor de estado, a direção da aeronave, o tempo decorrido, as deflexões da superfície de controle e a rotação e pressão de admissão do motor.

Antes de iniciar a simulação, o usuário precisa fornecer a opção desejada para o método de integração, o tempo entre cada iteração do simulador e a taxa de atualização da trajetória no programa Google Earth. Ainda é possível escolher um modelo para o vento no ambiente da simulação, bem como selecionar uma temperatura para altitude no nível do mar.

Tentando aproximar a simulação da realidade, é permitido escolher um tipo de servo para cada superfícies de controle, que ocasionarão atrasos na simulação devido ao tempo de resposta correspondente.

Iniciando a simulação, é possível controlar o avião manualmente ou escolher o tipo de controle automático desejado: vôo linear nivelado, fornecer uma direção (em graus) para vôo direcionado a ângulo, fornecer latitude e longitude para vôo direcionado a coordenada absoluta ou ainda pedir para que o algoritmo leia arquivo externo (com extensão .txt) contendo lista de pontos por coordenadas absolutas que formarão a trajetória do avião. Esse arquivo deve listar pontos contendo a latitude, longitude e altitude.

A simulação também permite a visualização de parâmetros adicionais, que são calculados a cada iteração e explicados nos capítulos anteriores, como os parâmetros atmosféricos, as derivadas do vetor de estado e os componentes da velocidade em relação a sistema na aeronave, com variáveis simbolizando ou não a interferência do vento.

Em qualquer momento da simulação o usuário pode alternar entre o tipo de controle automático e entre sua habilitação ou o modo manual. A simulação é terminada quando o usuário requisitar.

As figuras 4, 5 e 6 indicam as janelas da interface gráfica com o usuário. A figura 7 demonstra uma simulação realizada no Google Earth.

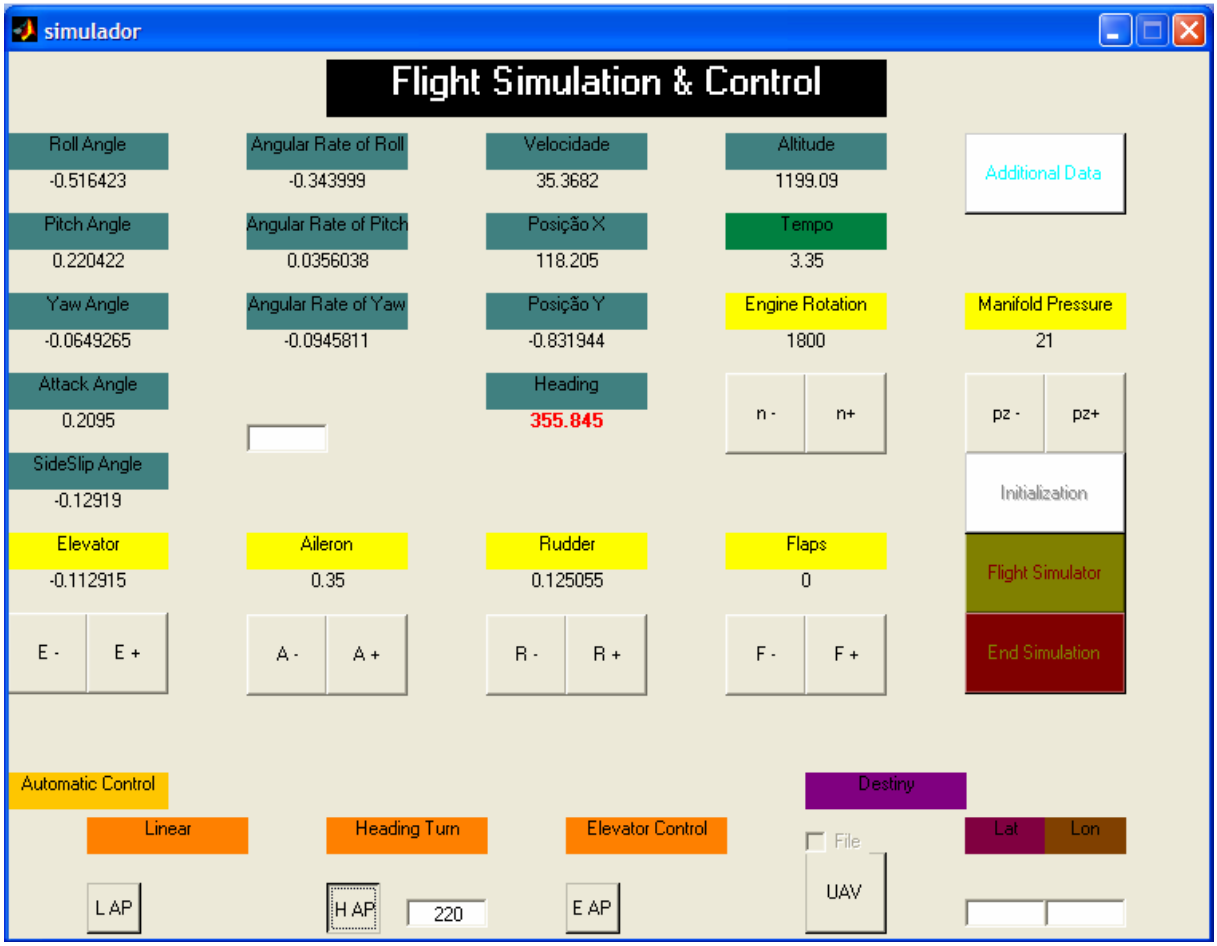

Figura 6 - Interface Gráfica – Janela Principal

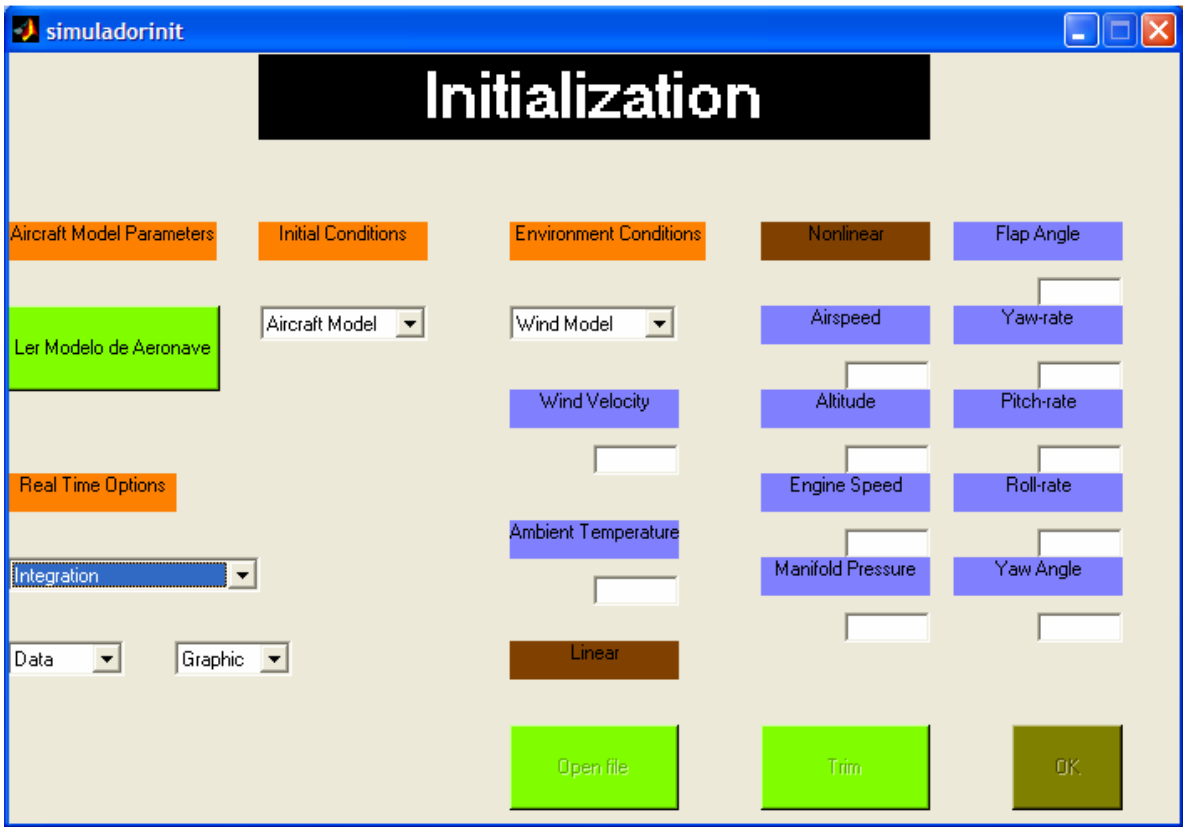

Figura 7 - Interface Gráfica - Inicialização

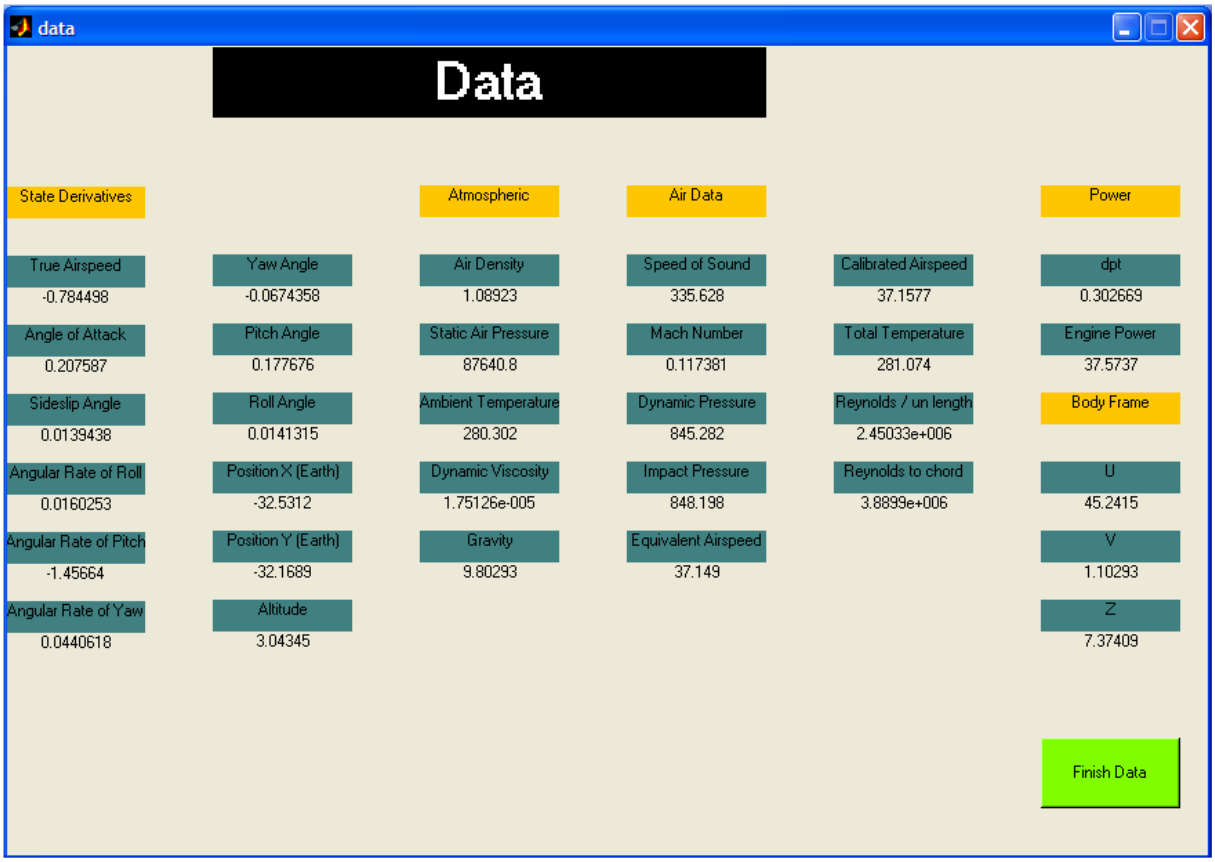

Figura 8 - Interface Gráfica – Dados Adicionais

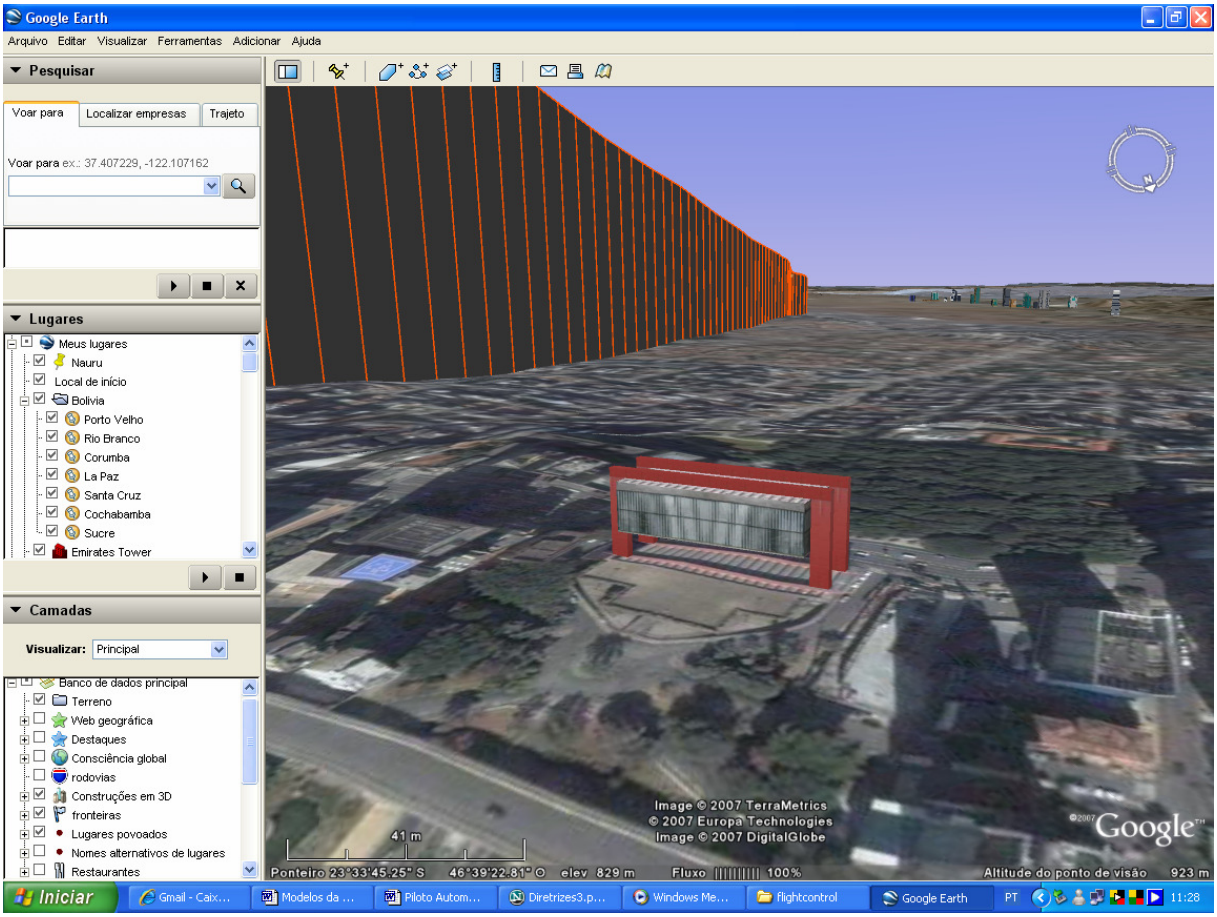

Figura 9 – Simulação em Tempo Real – Google Earth

6 RESULTADOS E DISCUSSÃO

O desenvolvimento de um software requer uma rotina de programação. Quanto mais sofisticado for o programa, maior é o número de erros contidos e para um certo número de operações tende a ser impossível que o software não possua *bugs*. Da mesma forma, neste programa de controle de vôo, que possui alto grau de complexidade matemática, existem bugs a serem descobertos, apesar das inúmeras tentativas para minimizá-los, portanto não há garantia total no seu funcionamento. A confiabilidade do programa pode ser aumentada com o seu uso e teste para as diferentes formas de simulação, caso não ocorram erros.

Foram realizados diversos vôos virtuais ao redor do mundo, principalmente entre aeroportos do Brasil. Esses vôos são determinados por uma trajetória desejada pelo usuário através dos *way points*, no modo de controle UAV. Como o programa ainda não dá suporte a pousos e decolagens, devido à necessidade de modelar o contato com o solo, é assumido que a aeronave percorre uma baixa altura em relação às pistas de cada aeroporto.

Diversos fatores podem limitar a simulação e um deles é a performance do computador. Como são realizados muitos cálculos num intervalo estabelecido pelo usuário, caso esse intervalo seja muito pequeno para determinado computador, existe a possibilidade de haver atraso na simulação em relação ao tempo real. De forma similar, escolhendo-se um método de integração mais complexo pode prejudicar a performance do computador.

Uma dificuldade considerável no modelamento é a compreensão dos sistemas de coordenadas a que cada variável se referem. No modelo foram usados referenciais fixos no solo, fixos em relação à posição inicial do avião, sensíveis à aeronave considerando desprezando e considerando a ação dos ventos. Assim, muitas vezes é necessário fazer uma transformação de coordenadas para observar o comportamento da aeronave.

A linguagem de programação utilizada possui uma grande vantagem (conforme discutido anteriormente) que é a disponibilidade de ajuda. Neste *help* podem ser assimilados diversos conceitos de algoritmos, com muitos exemplos práticos. Mesmo assim, por ser uma linguagem de alto nível, apresenta algumas

limitações. Um ponto interessante no desenvolvimento do algoritmo foi a determinação do menor pulso de clock identificado pela linguagem. Em determinado ponto da programação, não era possível utilizar intervalos de tempo menores que centésimos de segundo mesmo em máquinas de ponta. Foi descoberto que esse era aproximadamente o menor intervalo de tempo necessário para uma operação do MATLAB. Em versões mais atualizadas do software esse intervalo foi reduzido, sendo necessária a transferência do código para essa nova versão.

Um ponto relevante de serendipidade na simulação é que para pequenos intervalos de tempo o método de integração tem relevância insignificante, sendo destarte recomendável o uso do método de Euler.

Um elemento que pode causar distúrbio considerável em uma aeronave é o vento. Ele pode estar em qualquer direção e ter altas velocidades e o sistema de controle precisa corrigir a trajetória. No algoritmo existe uma grande simplificação que é a opção por vento constante ou na forma de rajadas modeladas por ondas harmônicas amortecidas.

O algoritmo desenvolvido na linguagem MATLAB permite que seja simulado um vôo no programa Google Earth. Porém, como isso é realizado? Na realidade, o algoritmo, em determinado ponto da rotina de simulação, gera um arquivo de extensão identificável pelo Google Earth. Esse arquivo contem um código que representa uma trajetória que será aberta pelo software Google Earth. Tanto a gravação quanto a chamada do programa é feita pelo MATLAB. O sistema operacional identifica qual programa permite abrir a extensão desejada, no caso o Google Earth. Todavia, isso geraria apenas uma foto e não uma simulação. Só que o algoritmo do MATLAB gera um arquivo e comanda a sua abertura a cada iteração, de forma dinâmica conforme a trajetória calculada pelo modelamento matemático. Assim, existe a impressão de haver uma animação da trajetória. Para futuros trabalhos, uma sugestão seria a apresentação da atitude da aeronave no Google Earth, por exemplo, através de pontos simbolizando o estado do avião.

É de se notar a diferença principal entre uma simulação de vôo e sua execução real: o tempo de resposta associado aos atrasos para captar cada dado. Tentando minimizar essa diferença, foram incluídos atrasos relativos a cada movimento das superfícies de controle associados servomotores. Para isso foram analisados os atrasos no tempo de resposta de alguns modelos e incluídos como

uma pausa de tempo específico para cada rotina. Diversos outros componentes do sistema embarcado podem gerar atrasos.

Durante a simulação é possível que ocorram alguns erros identificados. Caso a taxa de amostragem dos dados no Google Earth seja muito baixa e o computador não tenha performance adequada, o programa Google Earth pode interromper o funcionamento por não conseguir abrir muitos arquivos em pouco tempo. Nesse caso, o ideal é finalizar a simulação e escolher uma taxa maior. Ainda em relação à performance do computador, em algumas simulações foi detectada a ocorrência de atrasos no tempo devido à abertura do arquivo gráfico, propagando um certo erro na simulação.

Outro erro possível é a divergência dos valores no controle automático. Este fato ocorre quando o tempo para cada iteração é muito alto e o método de integração é menos sofisticado. Assim, é preciso conhecer uma relação adequada para cada tipo de computador, baseado em testes realizados pelo usuário. Ainda existe a possibilidade do arquivo em que a trajetória for gravada omitir certos pontos. Este caso está atrelado a um grande número de pontos, quando a trajetória é muito extensa. Mas na maioria dos casos não é relevante para a simulação, já que a perda é pequena quando comparado ao arquivo completo.

Um cuidado extra que o usuário deve ter é na determinação da trajetória. Para casos em que são exigidas manobras mais complexas, o sistema de controle provavelmente não funcionará e corre o risco do avião perder a estabilidade.

Para melhor ilustrar o comportamento do software, demonstro três testes de vôo: um vôo formando uma trajetória fechada com leves diferenças de altitude e dois vôos entre aeroportos.

Lembrando que para maiores intervalos de tempo, é necessário um método de integração mais refinado, buscou-se o maior intervalo no qual o método mais simples, o de Euler, funcionava adequadamente. Assim, para o computador testado, escolheu-se intervalo das iterações de 0,025s. A trajetória está atualizada no Google Earth na taxa de 2s, para melhor visualização gráfica. Nestes testes não foram consideradas as ações do vento, sendo sua velocidade nula.

Em todos os casos, a altitude fornecida está substituída pela altura do vôo devido a uma restrição do software: não existe o mapeamento das altitudes para cada coordenada geográfica. Ou seja, o software não identifica que existe uma

montanha em determinada região, o que impossibilitaria a detecção de colisão. Esse mapeamento das altitudes do globo fica como proposta para futuras versões.

O primeiro vôo é uma trajetória fechada nas proximidades de Pirassununga SP. Os pontos da trajetória em latitude, longitude e altitude são dados pela tabela 1. As três imagens a seguir demonstram a trajetória deste vôo no Google Earth.

Tabela 1 – Coordenadas do Vôo 1

Vôo 1	Latitude	Longitude	Altura (m)
Ponto 1	22º00'36.00'' S	47º19'48.00'' O	1220
Ponto 2	21º59'20.00'' S	47º18'54.00'' O	1200
Ponto 3	21º58'12.00'' S	47º17'60.00'' O	1400
Ponto 4	21º58'12.00'' S	47º16'12.00'' O	1200
Ponto 5	22º00'36.00'' S	47º16'12.00'' O	900
Ponto 6	22º00'36.00'' S	47º17'60.00'' O	1100
Ponto 7	21º59'20.00'' S	47º18'54.00'' O	1300

Figura 10 – Simulação do Vôo 1 – Imagem 1

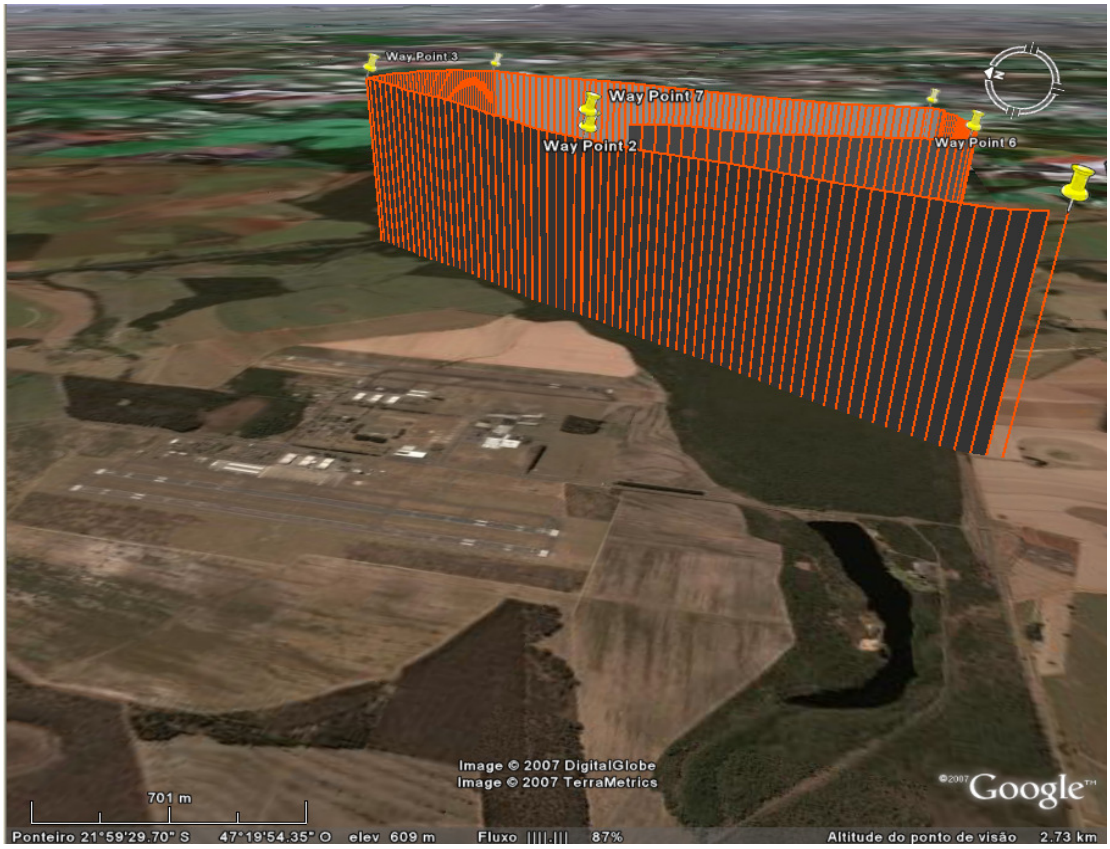

Figura 11 – Simulação do Vôo 1 – Imagem 2

Figura 12 – Simulação do Vôo 1 – Imagem 3

O segundo vôo percorre a cidade de São Paulo, partindo do aeroporto de Congonhas e pousando no aeroporto de Cumbica em Guarulhos. As coordenadas do vôo estão descritas na tabela 2 e as imagens seguintes referem-se a este vôo.

Tabela 2 – Coordenadas do Vôo 2

Vôo 2	Latitude	Longitude	Altura (m)
Ponto 1	23º38’17.20’’ S	46º39’04.54’’ O	10
Ponto 2	23º38’02.80’’ S	46º39’04.54’’ O	10
Ponto 3	23º37’17.94’’ S	46º39’35.71’’ O	10
Ponto 4	23º36’03.78’’ S	46º40’26.87’’ O	200
Ponto 5	23º32’04.67’’ S	46º37’45.77’’ O	1000
Ponto 6	23º27’02.77’’ S	46º32’48.52’’ O	500
Ponto 7	23º26’13.78’’ S	46º29’56.94’’ O	200
Ponto 8	23º26’02.54’’ S	46º28’56.03’’ O	10
Ponto 9	23º25’33.17’’ S	46º27’07.74’’ O	10

Figura 13 – Simulação do Vôo 2 – Imagem 1

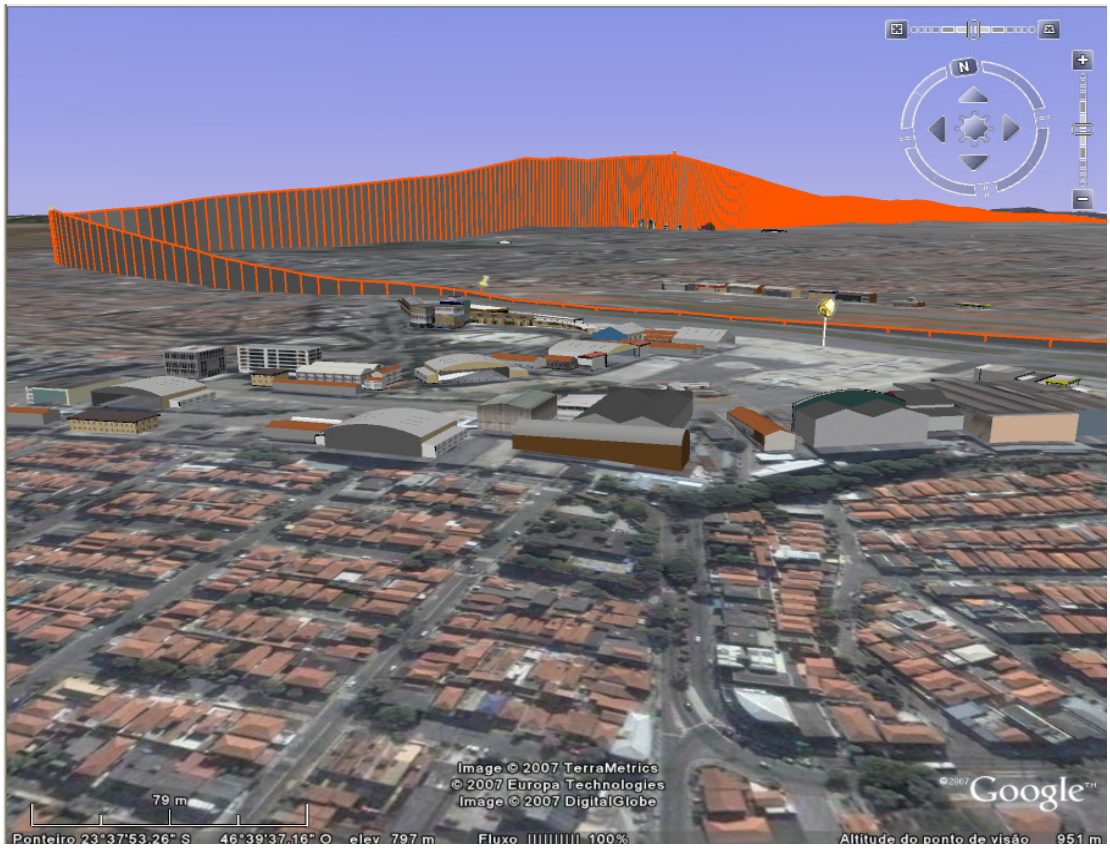

Figura 14 – Simulação do Vôo 2 – Imagem 2

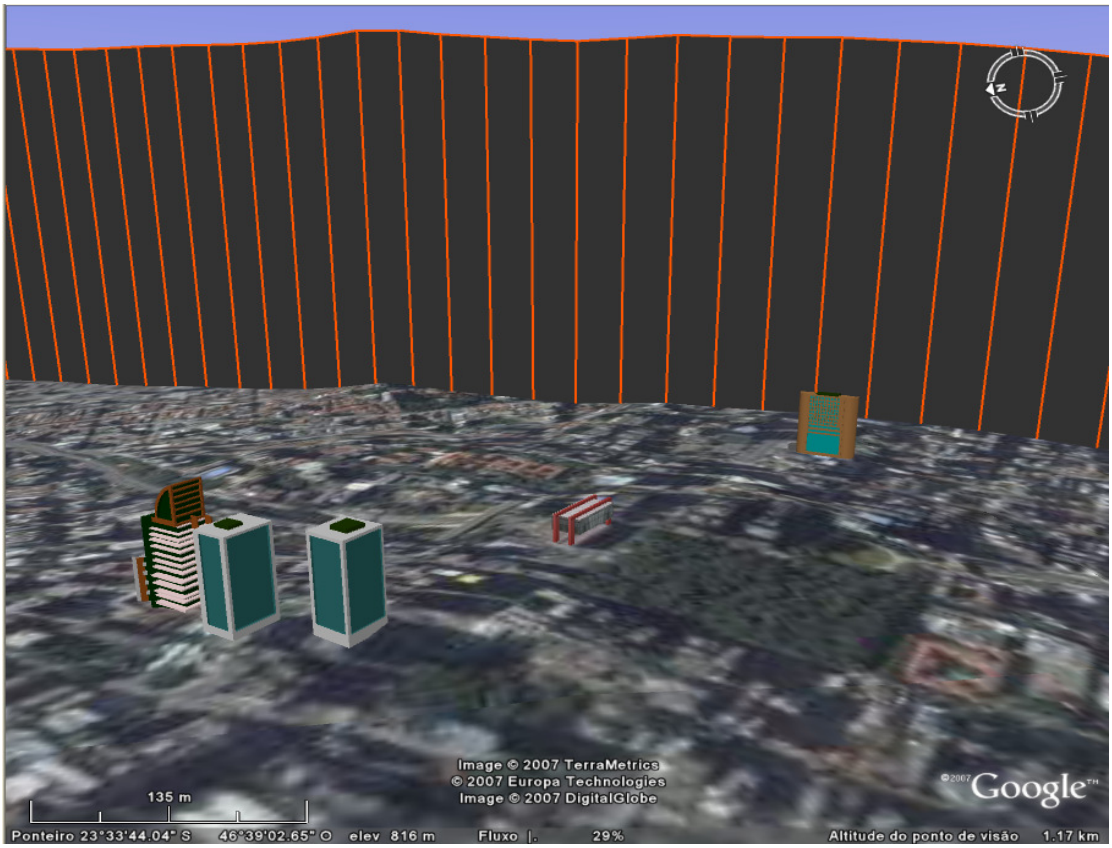

Figura 15 – Simulação do Vôo 2 – Imagem 3

O vôo de número três realiza a trajetória do aeroporto Santos Dumont no Rio de Janeiro até o aeroporto Tom Jobim, na mesma cidade, segundo os pontos da tabela 3:

Tabela 3 – Coordenadas do Vôo 3

Vôo 3	Latitude	Longitude	Altura (m)
Ponto 1	22º54'58.64'' S	43º09'44.96'' O	10
Ponto 2	22º54'21.42'' S	43º09'47.38'' O	10
Ponto 3	22º51'59.15'' S	43º09'57.96'' O	200
Ponto 4	22º51'37.76'' S	43º14'53.92'' O	500
Ponto 5	22º50'24.86'' S	43º18'42.48'' O	500
Ponto 6	22º48'46.58'' S	43º18'45.36'' O	200
Ponto 7	22º48'11.41'' S	43º15'39.96'' O	50
Ponto 8	22º48'01.87'' S	43º14'58.42'' O	10
Ponto 9	22º47'34.51'' S	43º13'12.36'' O	10

O tempo de simulação registrado foi de 982s, mas o tempo gasto de fato foi 985s, erro devido à abertura do arquivo gráfico, como explicado anteriormente. A seguir, seguem imagens ilustrativas deste percurso.

Figura 16 – Simulação do Vôo 3 – Imagem 1

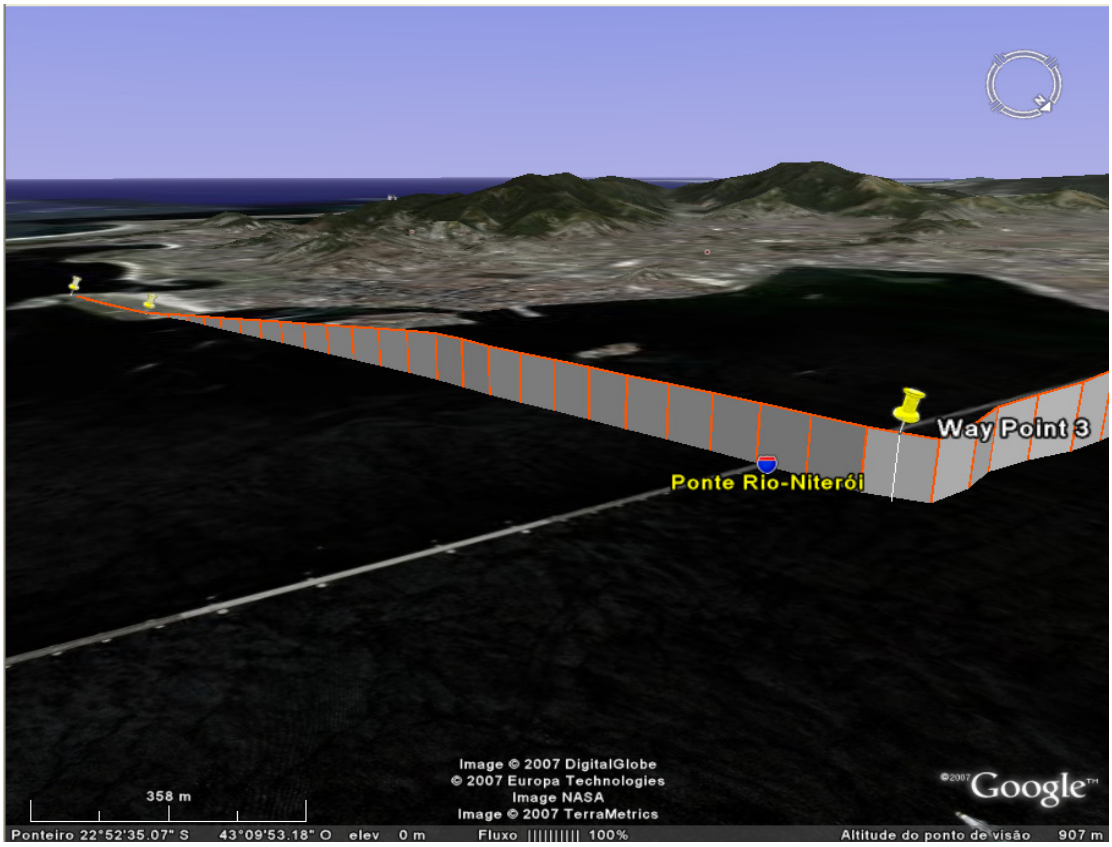

Figura 17 – Simulação do Vôo 3 – Imagem 2

Figura 18 – Simulação do Vôo 3 – Imagem 3

É preciso lembrar de todas as limitações que o algoritmo ainda possui. Nestes vôos foram utilizados o mesmo tipo de servo para cada superfícies de controle. Aqui existe ainda um ponto que não é real: é assumido que o servo, bem como todas as superfícies de controle, atingem o valor exato requisitado. Porém o comportamento dos motores possuem curvas de aceleração na forma de rampas, ou seja, aceleram, atingem um valor constante e posteriormente desaceleram. Assim, o modelo adotado força uma aceleração instantânea.

Assim, questões ainda pendentes para futuras versões do algoritmo, e que servem como sugestões para futuros trabalhos, são a melhoria do sistema de controle para ventos fortes, o modelamento do contato com o solo para pousos e decolagens, o mapeamento da topografia do globo para cada coordenada geográfica, a demonstração gráfica da atitude da aeronave. Ainda como sugestão, é possível desenvolver um algoritmo de reconhecimento de imagens para missões que exigem maior precisão. Esse algoritmo poderia ser utilizado em conjunto com a simulação no Google Earth, modificando a visão da aeronave para o referencial de uma câmera, facilmente ajustável no programa.

7 CONCLUSÕES

A simulação do controle de vôo objetiva construir um universo virtual com regras estabelecidas pelo modelamento matemático. É importante notar que por mais sofisticado que seja o modelo, ele nunca será igual ao mundo real, onde a presença de distúrbios são inevitáveis e ainda imprevisíveis. Na programação existe uma forte correlação entre o número de cálculos, proporcional ao número de iterações, e a sofisticação do modelo. Essas variáveis sempre geram erros e a função do programador é tentar minimizá-los, daí a busca pela otimização.

No algoritmo desenvolvido, procurou-se criar um ambiente com os parâmetros mais relevantes para a simulação, sendo considerado variáveis do ambiente, da aerodinâmica, do motor, da geometria, gravitacionais e eólicas. Com elas, foi possível determinar as forças e momentos totais para serem utilizadas nas equações de movimento, que permitiam o cálculo das derivadas das variáveis do vetor de estado. Utilizando então um método de integração, foi possível fazer uma previsão do estado seguinte da aeronave, dado um passo determinado. Esse conjunto de variáveis de estado no tempo determinou a trajetória do avião.

A simulação buscou demonstrar essa trajetória de forma gráfica para melhor visualização por parte do usuário, que podia observar em tempo real os valores de diversos parâmetros. Ainda era possível a manipulação de formas de controle, que basicamente atuavam nas superfícies de controle e no motor, variáveis que no mundo real são controladas pelo piloto. Essa manipulação era possível de ser feita de várias maneiras: de forma manual, em que o usuário controlava as deflexões das superfícies de controle; e de forma automáticas.

O controle automático possuía basicamente três modos distintos. O primeiro modo permitia que a aeronave mantivesse uma trajetória fixa. Esse tipo de controle serviu de base para as demais formas: o controle por coordenadas e o controle no modo UAV. O primeiro direcionava a aeronave para determinada posição e forçava a manutenção da trajetória naquela direção; o segundo era um conjunto de coordenadas que formava uma trajetória, sendo assim o modo mais complexo e que de fato simulava um UAV.

O controle da aeronave deve ser compreendido como um módulo independente, que possa ser usado num sistema embarcado sem a necessidade de efetuar todos os cálculos da simulação. A entrada do controle deve ser as variáveis do estado da aeronave, parâmetros que possam ser obtidos no mundo real através de sensores e outros tipos de equipamento e a única saída é o conjunto de parâmetros que determinam as deflexões da superfície de controle e valores para o motor: pressão de admissão e rotação do motor.

Com o objetivo de refinar a simulação, foram acrescentados parâmetros como a determinação dos ventos, que podem ser constantes, na forma de rajadas ou seguindo um perfil numérico atmosférico padrão. Também foi adicionado um cálculo para identificar quando o avião entra em *stall*, gerando instabilidade e tempos de atraso, na forma de resposta de servos, outrossim foram implementados objetivando reduzir a diferença entre o mundo virtual e o real.

Quando um sistema apresenta certo grau de complexidade, torna-se necessário facilitar a visualização do seu comportamento para uma melhor compreensão de sua dinâmica. A ferramenta desenvolvida pelo Google mostrou-se ser a mais apropriada, já que é de fácil manipulação por qualquer usuário e identifica com boa precisão as coordenadas geográficas no globo. Assim, foi tentado transportar os dados entre as duas plataformas: MATLAB e Google Earth.

Uma técnica utilizada e que pode ser ampliada para diversas aplicações é a criação de determinado código de uma linguagem por outro código em linguagem distinta. O MATLAB permite a abertura de um arquivo com determinada extensão pelo Windows, o que permitiu a manipulação entre os programas, ao salvar um documento de texto na extensão desejada. De forma semelhante, seria possível manipular qualquer programa que entenda um código. Por exemplo, poderia ser criada uma imagem com uma matriz de cores e editá-la de forma automática por outro programa ou ainda ser construído e editado um banco de dados usado em tempo real pela Internet, como as cotações das bolsas de valores.

A proposta que permanece é a utilização desse algoritmo de controle num sistema embarcado em UAV real. O algoritmo de controle só precisa ser ajustado para receber os dados dos sensores, GPS e demais equipamentos do sistema. Esses dados já serão processados em tempo real e fornecerão a resposta adequada para o controle automático. Recomenda-se que seja inicialmente testado em forma estática e posteriormente em vôo, para a calibragem de alguns parâmetros. O

desafio poderá popularizar o uso de UAVs já que o usuário não terá dificuldades em manipular a missão desejada.

REFERÊNCIAS

[1] GERSHENFELD, N. **The Nature of Mathematical Modeling.** Cambridge: Cambridge University Press, 1999.

[2] LUTGENS, F. K.; TARBUCK E. J. **The Atmosphere: An Introduction to Meteorology.** 9th ed. Upper Saddle River: Prentice Hall, 2003.

[3] NERIS, L. O. **Um Piloto Automático para as Aeronaves do Projeto ARARA.** 2001. 102 p. Dissertação (Mestrado) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2001.

[4] RAUW, M. **FDC 1.2 – A SIMULINK Toolbox for Flight Dynamics and Control Analysis.** 2nd ed. 2001. 275 p. Disponível em: <http://prdownloads.sourceforge.net/dutchroll/FDC12-report.pdf>. Acesso em: 08 set. 2007.

[5] _____. **FDC 1.4 – A SIMULINK Toolbox for Flight Dynamics and Control Analysis.** Draft Version 7. 2005. 253 p. Disponível em: <http://www.xs4all.nl/~rauw/fdcreport/FDC14_preview_007.pdf>. Acesso em: 08 set. 2007.

[6] RAYMER, D. P. **Simplified Aircraft Design for Homebuilders.** 1st ed. Los Angeles: Design Dimension Press, 2003.

[7] SALLET, G. **Ordinary Differential Equations with Scilab WATS Lectures Provisional notes.** Metz: Université De Metz, 2004. WATS Lectures, Provisional notes.

[8] TJEE, R. T. H. **Stability and control derivatives of De Havilland DHC-2 Beaver Aircraft.** Delft: Faculty of Aerospace Engineering, 1988. Relatório Técnico LR-556.

BIBLIOGRAFIA

[1] SCOTT, J. **Lift Coefficient & Thin Airfoil Theory.** 2003. AEROSPACEWEB, 2003. Disponível em: <http://www.aerospaceweb.org/question/aerodynamics/q0136.shtml>. Acesso em: 08 set. 2007.

[2] STEVENS, B. L.; LEWIS F. L. **Aircraft Control and Simulation.** 2nd ed. Hoboken: John Wiley & Sons, INC, 2003.

[3] THE MATHWORKS. **MATLAB®:** Documentation. Disponível em: <http://www.mathworks.com/access/helpdesk/help/helpdesk.html>. Acesso em: 08 set. 2007.

Apêndice – Código MATLAB

simulador.m

```
function varargout = simulador(varargin)
% SIMULADOR M-file for simulador.fig
%      SIMULADOR, by itself, creates a new SIMULADOR or raises the existing
%      singleton*.
%
%      H = SIMULADOR returns the handle to a new SIMULADOR or the handle to
%      the existing singleton*.
%
%      SIMULADOR('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in SIMULADOR.M with the given input
arguments.
%
%      SIMULADOR('Property','Value',...) creates a new SIMULADOR or raises
the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before simulador_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to simulador_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help simulador

% Last Modified by GUIDE v2.5 21-Apr-2007 17:56:49

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @simulador_OpeningFcn, ...
                   'gui_OutputFcn',  @simulador_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before simulador is made visible.
function simulador_OpeningFcn(hObject, eventdata, handles, varargin)
```

```
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to simulador (see VARARGIN)

% Choose default command line output for simulador
handles.output = hObject;
global AM;
global go;
go=0;
% Update handles structure
guidata(hObject, handles);
if length(AM)==0
    set(handles.pushbutton1,'Enable','off');
else
    set(handles.pushbutton1,'Enable','on');
end
set(handles.pushbutton2,'Enable','off');
set(handles.pushbutton3,'Enable','off');
set(handles.pushbutton4,'Enable','off');
set(handles.pushbutton5,'Enable','off');
set(handles.pushbutton6,'Enable','off');
set(handles.pushbutton7,'Enable','off');
set(handles.pushbutton8,'Enable','off');
set(handles.pushbutton9,'Enable','off');
set(handles.pushbutton10,'Enable','off');
set(handles.pushbutton12,'Enable','off');
set(handles.pushbutton13,'Enable','off');
set(handles.pushbutton14,'Enable','off');
set(handles.pushbutton15,'Enable','off');
set(handles.pushbutton18,'Enable','off');

% UIWAIT makes simulador wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = simulador_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global go;
global godata;
global uaero;
global uprop;
global int;
global dt;
global graphtemp;
```

```
global wind;
global T0;
global Vw;
global AM;
global EM;
global GM1;
global GM2;
global x;
global xdot;
global xinc0;
global uaero0;
global uprop0;
global head;
global yatm;
global yad1;
global yad2;
global yad3;
global ypow;
global ybvels;

atucab;
ganhos;

goeap=1;
godata=0;

set(handles.pushbutton17,'Enable','off');
set(handles.pushbutton2,'Enable','on');
set(handles.pushbutton3,'Enable','on');
set(handles.pushbutton4,'Enable','on');
set(handles.pushbutton5,'Enable','on');
set(handles.pushbutton6,'Enable','on');
set(handles.pushbutton7,'Enable','on');
set(handles.pushbutton8,'Enable','on');
set(handles.pushbutton9,'Enable','on');
set(handles.pushbutton10,'Enable','on');
set(handles.pushbutton12,'Enable','on');
set(handles.pushbutton13,'Enable','on');
set(handles.pushbutton14,'Enable','on');
set(handles.pushbutton15,'Enable','on');
set(handles.pushbutton18,'Enable','on');

set(handles.text21,'String','');
disp('Sistema de Controle de Vôo');
disp(' ');

%inicializacao;
p=1;
[latu,longu,altu] = textread('way3.txt','%f %f %f');
K=length(latu);
%disp('Variáveis Inicializadas');
%disp(' ');

xdot=0;
go=0;

cpu=cputime;
cpu0=cpu;
cont=1;
```

```
conto=cputime;
indge=1;
gap=1;

long0=-46.734;
lat0=-23.549;
fil=get(handles.checkbox1,'Value');
if fil==1
    long0=longu(1);
    lat0=latu(1);
else
    set(handles.checkbox1,'Enable','off');
end
long=long0;
lat=lat0;

while go==0
    entrada;
    atmosferico;
    aerodinamica;
    motor;
    gravitacao;
    vento;
    fmtotvel;
    xdot=derivacao(x(1,:),Ftot,Mtot,yhlp,GM1,GM2,cont,ybvels,yatm,AM);
    adicional;
    while go==0
        pause(0.0001)
        deltat(cont)=cputime-cpu0;
        if deltat(cont)>=dt*cont
            break
        end
    end
    integracao;
    set(handles.text1,'String',[x(1,1)]);
    set(handles.text7,'String',[x(1,10)]);
    set(handles.text9,'String',[x(1,11)]);
    set(handles.text6,'String',[x(1,12)]);
    set(handles.text22,'String',[x(1,7)]);
    set(handles.text24,'String',[x(1,8)]);
    set(handles.text26,'String',[x(1,9)]);
    set(handles.text32,'String',[x(1,4)]);
    set(handles.text28,'String',[x(1,5)]);
    set(handles.text30,'String',[x(1,6)]);
    set(handles.text36,'String',[x(1,2)]);
    set(handles.text34,'String',[x(1,3)]);
    set(handles.text12,'String',[dt*cont]);
    set(handles.text14,'String',[uaero(1)]);
    set(handles.text16,'String',[uaero(2)]);
    set(handles.text18,'String',[uaero(3)]);
    set(handles.text20,'String',[uaero(4)]);
    set(handles.text39,'String',[uprop(1)]);
    set(handles.text41,'String',[uprop(2)]);
    set(handles.text45,'String',[head]);
    if godata==1
        data
    end
    cont=cont+1;
end
cputime-cpu0
set(handles.pushbutton17,'Enable','on');
```

```
set(handles.pushbutton1,'Enable','off');
set(handles.pushbutton2,'Enable','off');
set(handles.pushbutton3,'Enable','off');
set(handles.pushbutton4,'Enable','off');
set(handles.pushbutton5,'Enable','off');
set(handles.pushbutton6,'Enable','off');
set(handles.pushbutton7,'Enable','off');
set(handles.pushbutton8,'Enable','off');
set(handles.pushbutton9,'Enable','off');
set(handles.pushbutton10,'Enable','off');
set(handles.pushbutton12,'Enable','off');
set(handles.pushbutton13,'Enable','off');
set(handles.pushbutton14,'Enable','off');
set(handles.pushbutton15,'Enable','off');
set(handles.pushbutton18,'Enable','off');
set(handles.checkbox1,'Enable','on');
set(handles.togglebutton1,'Value',0);
set(handles.togglebutton2,'Value',0);
set(handles.togglebutton3,'Value',0);
set(handles.togglebutton4,'Value',0);
clear all;

% --- Executes during object creation, after setting all properties.
function listbox2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --- Executes on selection change in listbox2.
function listbox2_Callback(hObject, eventdata, handles)
% hObject    handle to listbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns listbox2 contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from listbox2



% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
```

```
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns listbox1 contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from listbox1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global go;
go=1;


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
double


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global uaero;
uaero(1)=uaero(1)-0.01;
if uaero(1)<-0.5
    uaero(1)=-0.5
end

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global uaero;
uaero(1)=uaero(1)+0.01;
if uaero(1)>0.5
    uaero(1)=0.5
end

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global uaero;
uaero(2)=uaero(2)-0.01;
if uaero(2)<-0.5
    uaero(2)=-0.5
end

% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global uaero;
```

```
uaero(2)=uaero(2)+0.01;
if uaero(2)>0.5
    uaero(2)=0.5
end

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global uaero;
uaero(3)=uaero(3)-0.01;
if uaero(3)<-0.5
    uaero(3)=-0.5
end

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global uaero;
uaero(3)=uaero(3)+0.01;
if uaero(3)>0.5
    uaero(3)=0.5
end

% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global uaero;
if uaero(4)>0
    uaero(4)=uaero(4)-0.01;
end

% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global uaero;
uaero(4)=uaero(4)+0.01;

% --- Executes on button press in togglebutton1.
function togglebutton1_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton1
set(handles.togglebutton2,'Value',0);
set(handles.togglebutton3,'Value',0);
set(handles.togglebutton4,'Value',0);

% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu2 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu2


% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global uprop;
uprop(1)=uprop(1)-100;
if uprop(1)<0
    uprop(1)=0
end

% --- Executes on button press in pushbutton13.
function pushbutton13_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global uprop;
uprop(1)=uprop(1)+100;
if uprop(1)>5000
    uprop(1)=5000
end

% --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global uprop;
uprop(2)=uprop(2)-1;
if uprop(2)<0
    uprop(2)=0
end

% --- Executes on button press in pushbutton15.
function pushbutton15_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton15 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global uprop;
uprop(2)=uprop(2)+1;
if uprop(2)>100
    uprop(2)=100
end


% --- Executes on button press in togglebutton2.
function togglebutton2_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton2
set(handles.togglebutton1,'Value',0);
set(handles.togglebutton3,'Value',0);
set(handles.togglebutton4,'Value',0);

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a
double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
```

```
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a
double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a
double


% --- Executes on button press in pushbutton16.
function pushbutton16_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in togglebutton3.
function togglebutton3_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton3
set(handles.togglebutton1,'Value',0);
```

```
set(handles.togglebutton2,'Value',0);
set(handles.togglebutton4,'Value',0);

% --- Executes on button press in checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox1


% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu3 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu3


% --- Executes on button press in pushbutton17.
function pushbutton17_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

simuladorinit


% --- Executes on button press in togglebutton4.
function togglebutton4_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton4
set(handles.togglebutton1,'Value',0);
set(handles.togglebutton2,'Value',0);
set(handles.togglebutton3,'Value',0);
```

```
% --- Executes on button press in pushbutton18.
function pushbutton18_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

data
```

simuladorinit.m

```
function varargout = simuladorinit(varargin)
% SIMULADORINIT M-file for simuladorinit.fig
%      SIMULADORINIT, by itself, creates a new SIMULADORINIT or raises the
existing
%      singleton*.
%
%      H = SIMULADORINIT returns the handle to a new SIMULADORINIT or the
handle to
%      the existing singleton*.
%
%      SIMULADORINIT('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in SIMULADORINIT.M with the given input
arguments.
%
%      SIMULADORINIT('Property','Value',...) creates a new SIMULADORINIT or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before simuladorinit_OpeningFunction gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to simuladorinit_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help simuladorinit

% Last Modified by GUIDE v2.5 17-May-2007 16:49:06

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @simuladorinit_OpeningFcn, ...
                   'gui_OutputFcn',  @simuladorinit_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before simuladorinit is made visible.
function simuladorinit_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to simuladorinit (see VARARGIN)

% Choose default command line output for simuladorinit

handles.output = hObject;
global go1
go1=0;

set(handles.pushbutton1,'Enable','off');
set(handles.pushbutton11,'Enable','off');
set(handles.pushbutton12,'Enable','off');

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes simuladorinit wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = simuladorinit_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu2 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu2


% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu3.
```

```
function popupmenu3_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu3 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu3


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global go1;
global int;
global dt;
global graphtemp;
global AM;
global EM;
global GM1;
global GM2;
global x;
global xinc0;
global uaero;
global uprop;
global uaero0;
global uprop0;
global wind;
global T0;
global Vw;

int=1;
while int==1
    int = get(handles.popupmenu1,'Value');
    if int==1
        disp('Escolher Método de Integração!')
        pause(5)
    end
end
int2=1;
while int2==1
    int2 = get(handles.popupmenu2,'Value');
    if int2==1
        disp('Escolher Intervalo dos Dados!')
        pause(5)
    end
    if int2==2
        dt=0.001;
    elseif int2==3
        dt=0.01;
    elseif int2==4
        dt=0.025;
    elseif int2==5
        dt=0.05;
    elseif int2==6
        dt=0.1;
```

```
    elseif int2==7
        dt=1;
    end
end
int3=1;
while int3==1
    int3= get(handles.popupmenu3,'Value');
    if int3==1
        disp('Escolher Intervalo do Gráfico!')
        pause(5)
    end
    if int3==2
        graphtemp=2;
    elseif int3==3
        graphtemp=5;
    elseif int3==4
        graphtemp=10;
    elseif int3==5
        graphtemp=30;
    elseif int3==6
        graphtemp=60;
    end
end
int5=1;
while int5==1
    int5= get(handles.popupmenu19,'Value');
    if int5==1
        disp('Escolher Modelo do Vento!')
        pause(5)
    end
    if int5==2
        wind=1;
        Vw=str2num(get(handles.edit19,'String'));
        if isempty(Vw)
            Vw=5;
        end
    elseif int5==3
        wind=2;
    end
end

T0=str2num(get(handles.edit18,'String'));
if isempty(T0)==1
    T0=288.15;
end

if go1==1
    simulador
    close('simuladorinit')
else
    disp('Inicializar Modelo da Aeronave!');
end

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global go1;
global AM;
global EM;
```

```
global GM1;
global GM2;
inicializacao

disp('Variáveis Inicializadas');
disp(' ');
go1=1;

% --- Executes during object creation, after setting all properties.
function popupmenu15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu1.
function popupmenu15_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu16_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu2.
function popupmenu16_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: contents = get(hObject,'String') returns popupmenu2 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu2


% --- Executes during object creation, after setting all properties.
function popupmenu17_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu3.
function popupmenu17_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu3 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu3


% --- Executes on button press in pushbutton1.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton2.
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function popupmenu18_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
```

```
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu18.
function popupmenu18_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu18 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu18
int4= get(handles.popupmenu18,'Value');
if int4==2
    set(handles.pushbutton11,'Enable','on');
    set(handles.pushbutton12,'Enable','off');
elseif int4==3
    set(handles.pushbutton11,'Enable','off');
    set(handles.pushbutton12,'Enable','on');
end

% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global x;
global xinc0;
global uaero;
global uaero0;
global uprop;

condin;
set(handles.pushbutton1,'Enable','on');

% --- Executes during object creation, after setting all properties.
function edit9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit9_Callback(hObject, eventdata, handles)
% hObject    handle to edit9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit9 as text
%        str2double(get(hObject,'String')) returns contents of edit9 as a
double


% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global GM1;
global xinc0;
global uaero;
global uprop;
g0=9.80665;

xinc0 = [45 0 0 0 0 0 0 0 0 0 0 0]';
xfix=1;

V=str2num(get(handles.edit9,'String'));
H=str2num(get(handles.edit10,'String'));
n=str2num(get(handles.edit11,'String'));
pz=str2num(get(handles.edit12,'String'));
deltaf=str2num(get(handles.edit13,'String'));
psidot=str2num(get(handles.edit14,'String'));
thetadot=str2num(get(handles.edit15,'String'));
phidot=str2num(get(handles.edit16,'String'));
psi=str2num(get(handles.edit17,'String'));
if isempty(V)==1
    V=35;
end
if isempty(H)==1
    H=1219.2;
end
if isempty(n)==1
    n=1800;
end
if isempty(pz)==1
    pz=20;
end
if isempty(deltaf)==1
    deltaf=0;
end
if isempty(psidot)==1
    psidot=0;
end
if isempty(thetadot)==1
    thetadot=0;
end
if isempty(phidot)==1
    phidot=0;
end
if isempty(psi)==1
    psi=0;
end
G=psidot*V/g0;
phi=0;
ctrim = [V H psi pz G psidot thetadot phidot deltaf n phi]';
vtrim = [0 0 0 0 0 0]';
```

```
sysname='beaver';
rolltype='b';
turntype='c';
gammatype='m';
modfun   = ['xdot = feval(''',sysname,''',0,x,u,''outputs''); '];
modfun   = [modfun 'xdot = feval(''',sysname,''',0,x,u,''derivs'');']
warning off
feval(sysname,[],[],[],'compile');
clear ans % (the above feval statement somehow generates an 'ans' variable)
warning on
disp('Searching for stable solution. Wait a moment...');
disp(' ');
options = optimset('Display','off','TolX',1e-
10,'MaxFunEvals',5000,'MaxIter',3000);
[vtrimmed,fval,exitflag,output] = fminsearch('accost',vtrim,options,...
ctrim,rolltype,turntype,gammatype,modfun);
if exitflag == 0
      warning('Maximum number of iterations was exceeded!');
      disp(['- number of function evaluations: '
num2str(output.funcCount)]);
      disp(['- number of iterations: ' num2str(output.iterations)]);
   else
      disp('Converged to a trimmed solution...');
   end
[x,u] = acconstr(vtrimmed,ctrim,rolltype,turntype,gammatype);
eval(modfun);
feval(sysname,[],[],[],'term');
xinc0=x;
xdot0  = xdot;
uaero0 = u(1:4);
uprop0 = u(5:6);
for i=1:4
    uaero(i)=uaero0(i);
end
for i=1:2
    uprop(i)=uprop0(i);
end
for i=1:12
    x(1,i)=xinc0(i);
end
x
set(handles.pushbutton1,'Enable','on');

% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function edit10_Callback(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit10 as text
%        str2double(get(hObject,'String')) returns contents of edit10 as a
double


% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit11_Callback(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text
%        str2double(get(hObject,'String')) returns contents of edit11 as a
double


% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit12_Callback(hObject, eventdata, handles)
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit12 as text
%        str2double(get(hObject,'String')) returns contents of edit12 as a
double


% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit13_Callback(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit13 as text
%        str2double(get(hObject,'String')) returns contents of edit13 as a
double


% --- Executes during object creation, after setting all properties.
function edit14_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit14_Callback(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit14 as text
```

```
%        str2double(get(hObject,'String')) returns contents of edit14 as a
double


% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit15_Callback(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit15 as text
%        str2double(get(hObject,'String')) returns contents of edit15 as a
double


% --- Executes during object creation, after setting all properties.
function edit16_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit16_Callback(hObject, eventdata, handles)
% hObject    handle to edit16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit16 as text
%        str2double(get(hObject,'String')) returns contents of edit16 as a
double
```

```
% --- Executes during object creation, after setting all properties.
function edit17_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit17_Callback(hObject, eventdata, handles)
% hObject    handle to edit17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit17 as text
%        str2double(get(hObject,'String')) returns contents of edit17 as a
double


% --- Executes during object creation, after setting all properties.
function popupmenu19_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu19.
function popupmenu19_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu19 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu19


% --- Executes during object creation, after setting all properties.
function edit18_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit18_Callback(hObject, eventdata, handles)
% hObject    handle to edit18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit18 as text
%        str2double(get(hObject,'String')) returns contents of edit18 as a
double


% --- Executes during object creation, after setting all properties.
function edit19_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else

set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit19_Callback(hObject, eventdata, handles)
% hObject    handle to edit19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit19 as text
%        str2double(get(hObject,'String')) returns contents of edit19 as a
double
```

data.m

```
function varargout = data(varargin)
% DATA M-file for data.fig
%      DATA, by itself, creates a new DATA or raises the existing
%      singleton*.
%
%      H = DATA returns the handle to a new DATA or the handle to
```

```
%      the existing singleton*.
%
%      DATA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in DATA.M with the given input arguments.
%
%      DATA('Property','Value',...) creates a new DATA or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before data_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to data_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help data

% Last Modified by GUIDE v2.5 21-Apr-2007 18:25:28

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @data_OpeningFcn, ...
                   'gui_OutputFcn',  @data_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before data is made visible.
function data_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to data (see VARARGIN)

% Choose default command line output for data
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes data wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
global godata;
global yatm;
global yad1;
global yad2;
global yad3;
global xdot;
global ypow;
global ybvels;

godata=1;

set(handles.text39,'String',[yatm(1)]);
set(handles.text37,'String',[yatm(2)]);
set(handles.text35,'String',[yatm(3)]);
set(handles.text43,'String',[yatm(4)]);
set(handles.text41,'String',[yatm(5)]);
set(handles.text51,'String',[yad1(1)]);
set(handles.text49,'String',[yad1(2)]);
set(handles.text47,'String',[yad1(3)]);
set(handles.text55,'String',[yad2(1)]);
set(handles.text53,'String',[yad2(2)]);
set(handles.text61,'String',[yad2(3)]);
set(handles.text59,'String',[yad3(1)]);
set(handles.text57,'String',[yad3(2)]);
set(handles.text63,'String',[yad3(3)]);
set(handles.text70,'String',[xdot(1)]);
set(handles.text68,'String',[xdot(2)]);
set(handles.text66,'String',[xdot(3)]);
set(handles.text74,'String',[xdot(4)]);
set(handles.text72,'String',[xdot(5)]);
set(handles.text76,'String',[xdot(6)]);
set(handles.text82,'String',[xdot(7)]);
set(handles.text80,'String',[xdot(8)]);
set(handles.text78,'String',[xdot(9)]);
set(handles.text86,'String',[xdot(10)]);
set(handles.text84,'String',[xdot(11)]);
set(handles.text88,'String',[xdot(12)]);
set(handles.text92,'String',[ypow(1)]);
set(handles.text90,'String',[ypow(2)]);
set(handles.text97,'String',[ybvels(1)]);
set(handles.text95,'String',[ybvels(2)]);
set(handles.text100,'String',[ybvels(3)]);

% --- Outputs from this function are returned to the command line.
function varargout = data_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
global godata;
godata=0;
```

inicializacao.m

```
disp('Leitura do Modelo do Aviao');
disp(' ');
for i=1:19
    [AM(i,1) AM(i,2) AM(i,3) AM(i,4) AM(i,5) AM(i,6)]=textread('am.txt','%f
%f %f %f %f %f',1,'headerlines',i);
end
for i=1:4
    [EM(i,1) EM(i,2) EM(i,3) EM(i,4) EM(i,5) EM(i,6)]=textread('em.txt','%f
%f %f %f %f %f',1,'headerlines',i);
end
[GM1(1) GM1(2) GM1(3) GM1(4) GM1(5) GM1(6) GM1(7) GM1(8) GM1(9)
GM1(10)]=textread('gm1.txt','%f %f %f %f %f %f %f %f %f
%f',1,'headerlines',1);
I(1)=GM1(5)*GM1(6)-GM1(9)*GM1(9);
I(2)=GM1(7)*GM1(6)+GM1(9)*GM1(8);
I(3)=GM1(7)*GM1(9)+GM1(5)*GM1(8);
I(4)=GM1(4)*GM1(6)-GM1(8)*GM1(8);
I(5)=GM1(4)*GM1(9)+GM1(7)*GM1(8);
I(6)=GM1(4)*GM1(5)-GM1(7)*GM1(7);
detI=GM1(4)*GM1(5)*GM1(6)-2*GM1(7)*GM1(8)*GM1(9)-GM1(4)*GM1(9)^2-
GM1(5)*GM1(8)^2-GM1(6)*GM1(7)^2;
for i=1:3
    if i==1
        j=1; a=1; b=2; c=3;
    elseif i==2
        j=2; a=2; b=4; c=5;
    elseif i==3
        j=3; a=3; b=5; c=6;
    end
    GM2(i,1)=I(a)/detI;
    GM2(i,2)=I(b)/detI;
    GM2(i,3)=I(c)/detI;
    GM2(i,4)=-(GM1(8)*I(b)-GM1(7)*I(c))/detI;
    GM2(i,5)=(GM1(8)*I(a)-GM1(9)*I(b)-(GM1(5)-GM1(4))*I(c))/detI;
    GM2(i,6)=-(GM1(7)*I(a)+(GM1(4)-GM1(6))*I(b)-GM1(9)*I(c))/detI;
    GM2(i,7)=(GM1(9)*I(a)-GM1(7)*I(c))/detI;
    GM2(i,8)=-((GM1(6)-GM1(5))*I(a)-GM1(7)*I(b)+GM1(8)*I(c))/detI;
    GM2(i,9)=-(GM1(9)*I(a)-GM1(8)*I(b))/detI;
end
gm=fopen('GM2.txt','w');
for j=1:3
    for i=1:9
        fprintf(gm,'%f ',GM2(j,i));
    end
    fprintf(gm,'\n');
end
fclose(gm);
[uaero0(1) uaero0(2) uaero0(3) uaero0(4)]=textread('condin.txt','%f %f %f
%f',1,'headerlines',0);
[uprop0(1) uprop0(2)]=textread('condin.txt','%f %f',1,'headerlines',1);
[xdot0(1) xdot0(2) xdot0(3) xdot0(4) xdot0(5) xdot0(6) xdot0(7) xdot0(8)
xdot0(9) xdot0(10) xdot0(11) xdot0(12)]=textread('condin.txt','%f %f %f %f
%f %f %f %f %f %f %f %f',1,'headerlines',2);
[xinc0(1) xinc0(2) xinc0(3) xinc0(4) xinc0(5) xinc0(6) xinc0(7) xinc0(8)
xinc0(9) xinc0(10) xinc0(11) xinc0(12)]=textread('condin.txt','%f %f %f %f
%f %f %f %f %f %f %f %f',1,'headerlines',3);
for i=1:4
```

```
    uaero(i)=uaero0(i);
end
for i=1:2
    uprop(i)=uprop0(i);
end
for i=1:12
    x(1,i)=xinc0(i);
end
```

entrada.m

```
t=(cont-1)*dt;
com=get(handles.edit1,'String');

if strcmp(com,'s')==1
    uaero(1)=uaero(1)-0.01;
    set(handles.edit1,'String','');
end
if strcmp(com,'w')==1
    uaero(1)=uaero(1)+0.01;
    set(handles.edit1,'String','');
end
if strcmp(com,'a')==1
    uaero(2)=uaero(2)-0.01;
    set(handles.edit1,'String','');
end
if strcmp(com,'d')==1
    uaero(2)=uaero(2)+0.01;
    set(handles.edit1,'String','');
end

controle;
```

atmosferico.m

```
g0=9.80665;
Rterra=6371020;
lambda=-0.0065;
p0=101325;
R=287.053;
ro0=1.225;
gama=1.4;

yatm(5)=g0*(Rterra/(Rterra+x(1,12)))^2;
yatm(3)=T0+lambda*x(1,12);
yatm(2)=p0*(yatm(3)/T0)^(yatm(5)/1.86584);
yatm(1)=yatm(2)/(R*yatm(3));
yatm(4)=1.458*10^(-6)*yatm(3)^1.5/(yatm(3)+110.4);

yad1(1)=(401.8743*yatm(3))^0.5;
yad1(2)=x(1,1)/yad1(1);
yad1(3)=0.5*yatm(1)*x(1,1)^2;

yad2(1)=yatm(2)*((1+(gama-1)/2*yad1(2)^2)^(gama/(gama-1))-1);
yad2(2)=(yad1(3)*1.63265)^0.5;
yad2(3)=(579000*((1+yad2(1)/p0)^((gama-1)/gama)-1))^0.5;
```

```
yad3(1)=yatm(3)*(1+0.2*yad1(2)^2);
yad3(2)=(yatm(1)*x(1,1))/yatm(4);
yad3(3)=yad3(2)*GM1(1);
```

aerodinamica.m

```
ydl(1)=0.5*x(1,4)*GM1(2)/x(1,1);
ydl(2)=x(1,5)*GM1(1)/x(1,1);
ydl(3)=0.5*x(1,6)*GM1(2)/x(1,1);

ytmp(1)=1;
ytmp(2)=x(1,2);
ytmp(3)=x(1,2)^2;
ytmp(4)=x(1,2)^3;
ytmp(5)=x(1,3);
ytmp(6)=x(1,3)^2;
ytmp(7)=x(1,3)^3;
ytmp(8)=ydl(1);
ytmp(9)=ydl(2);
ytmp(10)=ydl(3);
ytmp(11)=uaero(1);
ytmp(12)=uaero(4);
ytmp(13)=uaero(2);
ytmp(14)=uaero(3);
ytmp(15)=x(1,2)*uaero(4);
ytmp(16)=x(1,2)*uaero(3);
ytmp(17)=x(1,2)*uaero(2);
ytmp(18)=x(1,3)^2*uaero(1);
ytmp(19)=0;
for i=1:6
    Caero(i)=0;
    for j=1:19
        sum=AM(j,i)*ytmp(j);
        Caero(i)=Caero(i)+sum;
    end
end

FMaero(1)=yad1(3)*Caero(1)*GM1(3);
FMaero(2)=yad1(3)*Caero(2)*GM1(3);
FMaero(3)=yad1(3)*Caero(3)*GM1(3);
FMaero(4)=yad1(3)*Caero(4)*GM1(3)*GM1(2);
FMaero(5)=yad1(3)*Caero(5)*GM1(3)*GM1(1);
FMaero(6)=yad1(3)*Caero(6)*GM1(3)*GM1(2);
```

motor.m

```
ypow(2)=0.7355*(-326.5+0.00412*(uprop(2)+7.4)*(uprop(1)+2010)+(408-
0.0965*uprop(1))*(1-yatm(1)/1.225));
ypow(1)=0.08696+191.18*(ypow(2)/(0.5*yatm(1)*x(1,1)^3));

clear ytmp;

ytmp(1)=ypow(1);
ytmp(2)=ypow(1)^3;
ytmp(3)=x(1,2)*ypow(1)^2;
```

```
ytmp(4)=ypow(1)*x(1,2)^2;
for i=1:6
    Cprop(i)=0;
    for j=1:4
        sum=EM(j,i)*ytmp(j);
        Cprop(i)=Cprop(i)+sum;
    end
end

FMprop(1)=yad1(3)*Cprop(1)*GM1(3);
FMprop(2)=yad1(3)*Cprop(2)*GM1(3);
FMprop(3)=yad1(3)*Cprop(3)*GM1(3);
FMprop(4)=yad1(3)*Cprop(4)*GM1(3)*GM1(2);
FMprop(5)=yad1(3)*Cprop(5)*GM1(3)*GM1(1);
FMprop(6)=yad1(3)*Cprop(6)*GM1(3)*GM1(2);
```

gravitacao.m

```
Fgrav(1)=-GM1(10)*yatm(5)*sin(x(1,8));
Fgrav(2)=GM1(10)*yatm(5)*cos(x(1,8))*sin(x(1,9));
Fgrav(3)=GM1(10)*yatm(5)*cos(x(1,8))*cos(x(1,9));
```

vento.m

```
gamaw=0;
psiw=90*3.141593/180;

if wind==1
    Vwh=Vw*cos(gamaw);
    Vwv=-Vw*sin(gamaw);
    uwinde(3)=Vwv;
    uwinde(1)=Vwh*cos(psiw-3.141593);
    uwinde(2)=Vwh*sin(psiw-3.141593);
elseif wind==2
    if x(1,12)<300
        Vw=1*(x(1,12)^0.2545-0.4097)/1.347;
    else
        Vw=2.86585*1;
    end
    Vwh=Vw*cos(gamaw);
    Vwv=-Vw*sin(gamaw);
    uwinde(3)=Vwv;
    uwinde(1)=Vwh*cos(psiw-3.141593);
    uwinde(2)=Vwh*sin(psiw-3.141593);
end

uwind(1)=cos(x(1,7))*cos(x(1,8))*uwinde(1)+sin(x(1,7))*cos(x(1,8))*uwinde(2
)-sin(x(1,8))*uwinde(3);
uwind(2)=cos(x(1,7))*sin(x(1,8))*sin(x(1,9))*uwinde(1)-
sin(x(1,7))*cos(x(1,9))*sin(x(1,7))*sin(x(1,8))*sin(x(1,9))*uwinde(2)+cos(x
(1,7))*cos(x(1,9))*cos(x(1,8))*sin(x(1,9))*uwinde(3);
uwind(3)=cos(x(1,7))*sin(x(1,8))*cos(x(1,9))*uwinde(1)+sin(x(1,7))*sin(x(1,
9))*sin(x(1,7))*sin(x(1,8))*cos(x(1,9))*uwinde(2)-
cos(x(1,7))*sin(x(1,9))*cos(x(1,8))*cos(x(1,9))*uwinde(3);

for i=4:6
```

```
    uwind(i)=0;
end

Fwind(1)=-GM1(10)*(uwind(4)+x(1,5)*uwind(3)-x(1,6)*uwind(2));
Fwind(2)=-GM1(10)*(uwind(5)-x(1,4)*uwind(3)+x(1,6)*uwind(1));
Fwind(3)=-GM1(10)*(uwind(6)+x(1,4)*uwind(2)-x(1,5)*uwind(1));
```

fmtotvel.m

```
for i=1:3
    Ftot(i)=FMaero(i)+FMprop(i)+Fgrav(i)+Fwind(i);
    Mtot(i)=FMaero(i+3)+FMprop(i+3);
end

yhlp(1)=cos(x(1,2));
yhlp(2)=sin(x(1,2));
yhlp(3)=cos(x(1,3));
yhlp(4)=sin(x(1,3));
yhlp(5)=tan(x(1,3));
yhlp(6)=sin(x(1,7));
yhlp(7)=cos(x(1,7));
yhlp(8)=sin(x(1,8));
yhlp(9)=cos(x(1,8));
yhlp(10)=sin(x(1,9));
yhlp(11)=cos(x(1,9));

ybvel(1)=x(1,1)*yhlp(1)*yhlp(3);
ybvel(2)=x(1,1)*yhlp(4);
ybvel(3)=x(1,1)*yhlp(2)*yhlp(3);

ybvels(1)=uwind(1)+ybvel(1);
ybvels(2)=uwind(2)+ybvel(2);
ybvels(3)=uwind(3)+ybvel(3);
```

derivacao.m

```
function xd = derivacao(xi,Ftot,Mtot,yhlp,GM1,GM2,cont,ybvels,yatm,AM)

xd(1)=(Ftot(1)*yhlp(1)*yhlp(3)+Ftot(2)*yhlp(4)+Ftot(3)*yhlp(2)*yhlp(3))/GM1
(10);
xd(2)=(-Ftot(1)*yhlp(2)+Ftot(3)*yhlp(1))/(GM1(10)*xi(1)*yhlp(3))-
yhlp(5)*(xi(4)*yhlp(1)+xi(6)*yhlp(2))+xi(5);
xd(3)=(-Ftot(1)*yhlp(1)*yhlp(4)+Ftot(2)*yhlp(3)-
Ftot(3)*yhlp(2)*yhlp(4))/(GM1(10)*xi(1))+xi(4)*yhlp(2)-xi(6)*yhlp(1);

clear ytmp;
ytmp(1)=Mtot(1);
ytmp(2)=Mtot(2);
ytmp(3)=Mtot(3);
ytmp(4)=xi(4)^2;
ytmp(5)=xi(4)*xi(5);
ytmp(6)=xi(4)*xi(6);
ytmp(7)=xi(5)^2;
ytmp(8)=xi(5)*xi(6);
ytmp(9)=xi(6)^2;
for i=1:3
```

```
    xd(i+3)=0;
    for j=1:9
        sum=GM2(i,j)*ytmp(j);
        xd(i+3)=xd(i+3)+sum;
    end
end

xd(7)=(xi(5)*yhlp(10)+xi(6)*yhlp(11))/yhlp(9);
xd(8)=xi(5)*yhlp(11)-xi(6)*yhlp(10);
xd(9)=xi(4)+yhlp(8)*xd(7);

v1=ybvels(1)*yhlp(9)+(ybvels(2)*yhlp(10)+ybvels(3)*yhlp(11))*yhlp(8);
v2=ybvels(2)*yhlp(11)-ybvels(3)*yhlp(10);
xd(10)=v1*yhlp(7)-v2*yhlp(6);
xd(11)=v1*yhlp(6)+v2*yhlp(7);
xd(12)=ybvels(1)*yhlp(8)-(ybvels(2)*yhlp(10)+ybvels(3)*yhlp(11))*yhlp(9);

xd(3)=xd(3)/(1-(yhlp(3)*yatm(1)*GM1(2)*GM1(3)*AM(19,2))/(4*GM1(10)));
```

adicional.m

```
yfp(1)=asin(xdot(12)/x(1,1));
yfp(2)=xdot(1)/g0;
yfp(3)=x(1,3)+x(1,7);
yfp(4)=x(1,9)*cos(x(1,8)*xinc0(2));

yuvw(1)=xdot(1)*yhlp(1)*yhlp(3)-
x(1,1)*(xdot(2)*yhlp(2)*yhlp(3)+xdot(3)*yhlp(1)*yhlp(4));
yuvw(2)=xdot(1)*yhlp(4)+x(1,1)*yhlp(3)*xdot(3);
yuvw(3)=xdot(1)*yhlp(2)*yhlp(3)+x(1,1)*(xdot(2)*yhlp(1)*yhlp(3)-
xdot(3)*yhlp(2)*yhlp(4));

yacc(1)=(Ftot(1)-Fgrav(1))/(GM1(10)*g0);
yacc(2)=(Ftot(2)-Fgrav(2))/(GM1(10)*g0);
yacc(3)=(Ftot(3)-Fgrav(3))/(GM1(10)*g0);
yacc(4)=Ftot(1)/(GM1(10)*g0);
yacc(5)=Ftot(2)/(GM1(10)*g0);
yacc(6)=Ftot(3)/(GM1(10)*g0);

CL=2*pi*x(2);
vst=(2*GM1(10)*yatm(5)/(GM1(3)*yatm(1)*CL))^0.5;
if x(1,1)<=vst
    set(handles.text21,'String','STALL');
end
if x(1,12)==0 & xdot(12)>=2
    set(handles.text21,'String','CRASH');
    go=1;
end

long=x(1,11)/((2*3.141593*Rterra*cos(lat*3.141593/180))/360)+long0;
lat=x(1,10)/111000+lat0;

goa=0;
if (cputime-conto)>=graphtemp
    goa=1;
end
if goa==1
```

```
    conto=cputime;
    lato(indge)=lat;
    longo(indge)=long;
    alto(indge)=x(1,12);
    ge=fopen('Google Earth SP.kml','w');

    fprintf(ge,'<?xml version="1.0" encoding="UTF-8"?>\n');
    fprintf(ge,'<kml xmlns="http://earth.google.com/kml/2.1">\n');
    fprintf(ge,'<Document>\n');
    fprintf(ge,'    <name>Google Earth SP.kml</name>\n');
    fprintf(ge,'    <Style id="sn_ylw-pushpin">\n');
    fprintf(ge,'        <IconStyle>\n');
    fprintf(ge,'            <scale>1.1</scale>\n');
    fprintf(ge,'            <Icon>\n');
    fprintf(ge,'
<href>http://maps.google.com/mapfiles/kml/pushpin/ylw-
pushpin.png</href>\n');
    fprintf(ge,'            </Icon>\n');
    fprintf(ge,'            <hotSpot x="20" y="2" xunits="pixels"
yunits="pixels"/>\n');
    fprintf(ge,'        </IconStyle>\n');
    fprintf(ge,'        <LineStyle>\n');
    fprintf(ge,'            <color>ff0055ff</color>\n');
    fprintf(ge,'            <width>2</width>\n');
    fprintf(ge,'        </LineStyle>\n');
    fprintf(ge,'    </Style>\n');
    fprintf(ge,'    <StyleMap id="msn_ylw-pushpin">\n');
    fprintf(ge,'        <Pair>\n');
    fprintf(ge,'            <key>normal</key>\n');
    fprintf(ge,'            <styleUrl>#sn_ylw-pushpin</styleUrl>\n');
    fprintf(ge,'        </Pair>\n');
    fprintf(ge,'        <Pair>\n');
    fprintf(ge,'            <key>highlight</key>\n');
    fprintf(ge,'            <styleUrl>#sh_ylw-pushpin</styleUrl>\n');
    fprintf(ge,'        </Pair>\n');
    fprintf(ge,'    </StyleMap>\n');
    fprintf(ge,'    <Style id="sh_ylw-pushpin">\n');
    fprintf(ge,'        <IconStyle>\n');
    fprintf(ge,'            <scale>1.3</scale>\n');
    fprintf(ge,'            <Icon>\n');
    fprintf(ge,'
<href>http://maps.google.com/mapfiles/kml/pushpin/ylw-
pushpin.png</href>\n');
    fprintf(ge,'            </Icon>\n');
    fprintf(ge,'            <hotSpot x="20" y="2" xunits="pixels"
yunits="pixels"/>\n');
    fprintf(ge,'        </IconStyle>\n');
    fprintf(ge,'    </Style>\n');

    if fil==1
    placemark=1;
    for placemark=1:p
    fprintf(ge,'    <Placemark>\n');
    fprintf(ge,'    <name>Way Point %d</name>\n',placemark);
    fprintf(ge,'        <Snippet maxLines="2">\n');
    fprintf(ge,' Way Point       </Snippet>\n');
    %fprintf(ge,'        <LookAt>\n');
    %fprintf(ge,'            <longitude>%f </longitude>\n',longo(i));
    %fprintf(ge,'            <latitude>%f </latitude>\n',lato(i));
    %fprintf(ge,'            <altitude>0</altitude>\n');
    %fprintf(ge,'            <range>493</range>\n');
```

```
    %fprintf(ge,'            <tilt>31</tilt>\n');
    %fprintf(ge,'            <heading>-159</heading>\n');
    %fprintf(ge,'        </LookAt>\n');
    fprintf(ge,'        <Point>\n');
    fprintf(ge,'        <extrude>1</extrude>\n');
    fprintf(ge,'        <altitudeMode>relativeToGround</altitudeMode>\n');
    fprintf(ge,'
<coordinates>%f,%f,%f</coordinates>\n',longu(placemark),latu(placemark),alt
u(placemark));
    fprintf(ge,'        </Point>\n');
    fprintf(ge,'    </Placemark>\n');
    placemark=placemark+1;
    end
    end

    fprintf(ge,'    <Placemark>\n');
    fprintf(ge,'        <name>Google Earth SP</name>\n');
    fprintf(ge,'        <styleUrl>#msn_ylw-pushpin</styleUrl>\n');
    fprintf(ge,'        <LineString>\n');
    fprintf(ge,'            <extrude>1</extrude>\n');
    fprintf(ge,'            <tessellate>1</tessellate>\n');
    fprintf(ge,'
<altitudeMode>relativeToGround</altitudeMode>\n');
    fprintf(ge,'            <coordinates>\n');
    for i=1:indge
        fprintf(ge,'%f,%f,%f ',longo(i),lato(i),alto(i));
    end
    fprintf(ge,'</coordinates>\n');
    fprintf(ge,'        </LineString>\n');
    fprintf(ge,'    </Placemark>\n');
    fprintf(ge,'</Document>\n');
    fprintf(ge,'</kml>\n');

    fclose(ge);
    indge=indge+1;
    winopen('Google Earth SP.kml');
end
```

integracao.m

```
global head;
if int==2
    if cont==1
        disp('Método de Euler')
    end
    for i=1:12
        %x(cont+1,i)=x(cont,i)+xdot(i)*dt;
        x(2,i)=x(1,i)+xdot(i)*dt;
    end
end
if int==3
    if cont==1
        disp('Metodo de Heun')
    end
    for i=1:12
        k1(i)=xdot(i);
        aj(i)=k1(i)*dt+x(1,i);
    end
    k2=derivacao(aj,Ftot,Mtot,yhlp,GM1,GM2,cont,ybvels,yatm,AM);
```

```
    for i=1:12
        x(2,i)=x(1,i)+0.5*dt*(k1(i)+k2(i));
    end
end
if int==4
    if cont==1
        disp('Metodo de Runge-Kutta')
    end
    for i=1:12
        k1(i)=xdot(i);
        aj(i)=x(1,i)+k1(i)*dt/2;
    end
    k2=derivacao(aj,Ftot,Mtot,yhlp,GM1,GM2,cont,ybvels,yatm,AM);
    for i=1:12
        aj(i)=x(1,i)+k2(i)*dt/2;
    end
    k3=derivacao(aj,Ftot,Mtot,yhlp,GM1,GM2,cont,ybvels,yatm,AM);
    for i=1:12
        aj(i)=x(1,i)+k3(i)*dt;
    end
    k4=derivacao(aj,Ftot,Mtot,yhlp,GM1,GM2,cont,ybvels,yatm,AM);
    for i=1:12
        x(2,i)=x(1,i)+dt*(k1(i)/6+k2(i)/3+k3(i)/3+k4(i)/6);
    end
end
if (x(2,11)-x(1,11))>0 & (x(2,10)-x(1,10))>0
    head=atan((x(2,11)-x(1,11))/(x(2,10)-x(1,10)))*180/3.141593;
elseif (x(2,11)-x(1,11))<0 & (x(2,10)-x(1,10))>0
    head=atan((x(2,11)-x(1,11))/(x(2,10)-x(1,10)))*180/3.141593+360;
elseif (x(2,11)-x(1,11))>0 & (x(2,10)-x(1,10))<0
    head=atan((x(2,11)-x(1,11))/(x(2,10)-x(1,10)))*180/3.141593+180;
elseif (x(2,11)-x(1,11))<0 & (x(2,10)-x(1,10))<0
    head=atan((x(2,11)-x(1,11))/(x(2,10)-x(1,10)))*180/3.141593+180;
end

if x(2,12)<=0
    x(2,12)=0;
end
%controle;
for i=1:12
    x(1,i)=x(2,i);
end
```

controle.m

```
global com0;
lap=get(handles.togglebutton1,'Value');
hap=get(handles.togglebutton2,'Value');
uap=get(handles.togglebutton3,'Value');
eap=get(handles.togglebutton4,'Value');
if lap==1
    level;
end
if hap==1
    com=str2num(get(handles.edit2,'String'));
    if goeap==1
        com0=str2num(get(handles.edit2,'String'));
    elseif com~=com0
        com0=str2num(get(handles.edit2,'String'));
```

```
        head0=head;
        %goeap=1;
    end
    if isempty(com)==1
        set(handles.edit2,'String','0');
        com=str2num(get(handles.edit2,'String'));
    end
    heading;
end
if uap==1
    if p==1
        p=p+1;
    end
    uav;
end
if eap==1
    elevator;
end

if uaero(1)<-0.5
    uaero(1)=-0.5;
end
if uaero(1)>0.5
    uaero(1)=0.5;
end
if uaero(2)<-0.35
    uaero(2)=-0.35;
end
if uaero(2)>0.35
    uaero(2)=0.35;
end
if uaero(3)<-0.3
    uaero(3)=-0.3;
end
if uaero(3)>0.3
    uaero(3)=0.3;
end
```

level.m

```
if int==2
    %inactuator=-Kq*(x(2,5)-xinc0(5))+Ktheta*(3.14/180-(x(2,8)-
xinc0(8)))+Ki_1*Ktheta*(3.14/180-(x(2,8)-xinc0(8)))*dt*cont;
    if goeap==1
        inactuatore2=0;
        inactuatora2=0;
        xce(1,1)=0;
        xce(2,1)=0;
        xce(3,1)=0;
        xca(1,1)=0;
        xca(2,1)=0;
        xca(3,1)=0;
        xcr(1,1)=0;
        xcr(2,1)=0;
        xcr(3,1)=0;
        for i=1:12
            xincontrol(i)=x(1,i);
        end
        for i=1:4
```

```
            uaerocontrol(i)=uaero(i);
        end
        goeap=0;
    end
    inactuatore1=-Kq*(x(2,5)-xincontrol(5))+Ktheta*(0-(x(2,8)-
xincontrol(8)));
    %inactuatore2=inactuatore2+Ki_1*Ktheta*(3.14/180-(x(2,8)-
xincontrol(8)))*dt;
    inactuatore2=inactuatore2+Ki_1*Ktheta*(0-(x(2,8)-xincontrol(8)))*dt;
    inactuatore=inactuatore1+inactuatore2;
    %inactuatora1=dar*(x(2,6)-xincontrol(6))+Kphi*(0-(x(2,9)-
xincontrol(9)));
    if head<=180
        inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*((head)*3.141593/180-(x(2,7)-xincontrol(7))))-
(x(2,9)-xincontrol(9)));
    elseif head<360
        inactuatora1=dar*(x(2,6)-xincontrol(6))+Kphi*((Kpsi*((head-
360)*3.141593/180-(x(2,7)-xincontrol(7))))-(x(2,9)-xincontrol(9)));
    end
    %inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*((170*3.141593/180-xincontrol(7))-(x(2,7)-0)))-
(x(2,9)-xincontrol(9)));
    %inactuatora2=inactuatora2+Ki_2*Kphi*(3.14*10/180-(x(2,9)-
xincontrol(9)))*dt;
    if head<=180
        inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*((head)*3.141593/180-
(x(2,7)-xincontrol(7))))-(x(2,9)-xincontrol(9)))*dt;
    elseif head<360
        inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*((head-
360)*3.141593/180-(x(2,7)-xincontrol(7))))-(x(2,9)-xincontrol(9)))*dt;
    end
    %inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*((170*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
    inactuatora=inactuatora1+inactuatora2;
    inactuatorr=(drr-Kr)*(x(2,6)-
xincontrol(6))+(Kr*g0/xincontrol(1))*sin(x(2,9)-xincontrol(9));

    uce(1,1)=inactuatore;
    uce(2,1)=0;
    xcdote=Aelv*xce+Belv*uce;
    xce=xce+dt*xcdote;
    outactuatore=Celv*xce+Delv*uce;
    uca(1,1)=inactuatora;
    uca(2,1)=0;
    xcdota=Aail*xca+Bail*uca;
    xca=xca+dt*xcdota;
    outactuatora=Cail*xca+Dail*uca;
    ucr(1,1)=inactuatorr;
    ucr(2,1)=0;
    xcdotr=Arud*xcr+Brud*ucr;
    xcr=xcr+dt*xcdotr;
    outactuatorr=Crud*xcr+Drud*ucr;

    uaero(1)=uaerocontrol(1)+outactuatore;
    uaero(2)=uaerocontrol(2)+outactuatora;
    %if x(2,7)>0.001
    %    uaero(3)=uaerocontrol(3)+outactuatorr+0.2;
    %elseif x(2,7)<-0.001
   %     uaero(3)=uaerocontrol(3)+outactuatorr-0.2;
   %else
```

```
        uaero(3)=uaerocontrol(3)+outactuatorr;
        %end

elseif int==3
    inactuator=-Kq*(x(2,5)-xinc0(5))+Ktheta*(3.14/180-(x(2,8)-
xinc0(8)))+Ki_1*Ktheta*(3.14/180-(x(2,8)-xinc0(8)))*dt*cont;
    if goeap==1
        xc(1,1)=0;
        xc(2,1)=0;
        xc(3,1)=0;
        goeap=0;
    end
    uc(1,1)=inactuator;
    uc(2,1)=0;
    xcdot=Aelv*xc+Belv*uc;
    xc=xc+dt*xcdot;
    outactuator=Celv*xc+Delv*uc
    uaero(1)=uaero0(1)+outactuator;
elseif int==4
end
```

heading.m

```
if int==2
    %inactuator=-Kq*(x(2,5)-xinc0(5))+Ktheta*(3.14/180-(x(2,8)-
xinc0(8)))+Ki_1*Ktheta*(3.14/180-(x(2,8)-xinc0(8)))*dt*cont;
    if goeap==1
        if hap==1
            altu(p)=x(1,12);
        end
        inactuatore2=0;
        inactuatore4=0;
        inactuatora2=0;
        xce(1,1)=0;
        xce(2,1)=0;
        xce(3,1)=0;
        xca(1,1)=0;
        xca(2,1)=0;
        xca(3,1)=0;
        xcr(1,1)=0;
        xcr(2,1)=0;
        xcr(3,1)=0;
        for i=1:12
            xincontrol(i)=x(1,i);
            xdotincontrol(i)=xdot(i);
        end
        for i=1:4
            uaerocontrol(i)=uaero(i);
        end
        head0=head;
        goeap=0;
        goprop=0;
    end

    inactuatore1=-Kq*(x(2,5)-xincontrol(5))+Ktheta*(0-(x(2,8)-
xincontrol(8)));
    inactuatore2=inactuatore2+Ki_1*Ktheta*(0-(x(2,8)-xincontrol(8)))*dt;
    inactuatore=inactuatore1+inactuatore2;
    %inactuatore4=inactuatore4+dt*exp(-0.15*dt*cont)*((1230)-(x(2,12)));
```

```
%     inactuatore4=inactuatore4+dt*exp(-0.15*dt*cont)*(0-(xdot(12)-
xdotincontrol(12)));
%     inactuatore5=inactuatore4*KHdot*1/Ktheta;
%     if inactuatore5>=18*3.141593/180
%         inactuatore5=18*3.141593/180;
%     elseif inactuatore5<=-8*3.141593/180
%         inactuatore5=-8*3.141593/180;
%     end
%     inactuatore3=Ktheta*((inactuatore5)-(x(2,8)-xincontrol(8)));
%
%     %inactuatore1=-Kq*(x(2,5));
%     inactuatore1=-Kq*(x(2,5)-
(x(2,6)*tan(x(2,9))+(0.03*x(2,1)+0.25)*((1/cos(x(2,9)+0.03491))-1)) );
%     %inactuatore1=-Kq*(x(2,5)- (x(2,6)*tan(x(2,9))+(0.0005*x(2,1)^2-
0.03*x(2,1)+0.9375)*((1/cos(x(2,9)))-1)) )...
%     inactuatore2=inactuatore2+Ki_1*inactuatore3*dt;
%     %inactuatore2=inactuatore2+Ki_1*Ktheta*(1/Ktheta*(KH*(0-(x(2,12)-
xincontrol(12))))-(x(2,8)-xincontrol(8)))*dt;
%     inactuatore=inactuatore1+inactuatore2+inactuatore3;

    %com
    %head0

   if (head0>=0 & head0<=90) & (com>=0 & com<=90)
        inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*((com*3.141593/180-xincontrol(7))-(x(2,7)-0)))-
(x(2,9)-xincontrol(9)));
   elseif (head0>=0 & head0<=90) & (com>=90 & com<=180)
        inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*((com*3.141593/180-xincontrol(7))-(x(2,7)-0)))-
(x(2,9)-xincontrol(9)));
   elseif (head0>=90 & head0<=180) & (com>=0 & com<=90)
        inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*((com*3.141593/180-xincontrol(7))-(x(2,7)-0)))-
(x(2,9)-xincontrol(9)));
   elseif (head0>=90 & head0<=180) & (com>=90 & com<=180) %& (com-head0)>0
        inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*((com*3.141593/180-xincontrol(7))-(x(2,7)-0)))-
(x(2,9)-xincontrol(9)));
   elseif (head0>=90 & head0<=180) & (com>=180 & com<=270)
        inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*((com*3.141593/180-xincontrol(7))-(x(2,7)-0)))-
(x(2,9)-xincontrol(9)));
   elseif (head0>=90 & head0<=180) & (com>=270 & com<=360)
        inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*((com*3.141593/180-xincontrol(7))-(x(2,7)-0)))-
(x(2,9)-xincontrol(9)));
   elseif (head0>=180 & head0<=270) & (com>=270 & com<=360)
       inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*(((com)*3.141593/180-xincontrol(7))-(x(2,7)-
0)))-(x(2,9)-xincontrol(9)));
   elseif (head0>=180 & head0<=270) & (com>=180 & com<=270) & (com-head0)>0
       inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*(((com)*3.141593/180-xincontrol(7))-(x(2,7)-
0)))-(x(2,9)-xincontrol(9)));
   elseif (head0>=180 & head0<=270) & (com>=0 & com<=90)
       inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*(((com)*3.141593/180-xincontrol(7))-(x(2,7)-
0)))-(x(2,9)-xincontrol(9)));
   elseif (head0>=270 & head0<=360) & (com>=0 & com<=90)
```

```
       inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*(((com)*3.141593/180-xincontrol(7))-(x(2,7)-
0)))-(x(2,9)-xincontrol(9)));
%    elseif (head0>=270 & head0<=360) & (com>=270 & com<=360) & (com-
head0)>0
%        inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*(((com)*3.141593/180-xincontrol(7))-(x(2,7)-
0)))-(x(2,9)-xincontrol(9)));
   else
       inactuatora1=dar*(x(2,6)-xincontrol(6))+Kphi*((Kpsi*(((com-
360)*3.141593/180-xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)));
   end
    %com-head0
%      if(((com-head0<=180) & (com-head0>0)) | (com-head0<-180))
% %         if com>180
% %             inactuatora1=dar*(x(2,6)-xincontrol(6))+Kphi*((Kpsi*(((com-
360)*3.141593/180-xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)));
% %         else
% %            inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*((com*3.141593/180-xincontrol(7))-(x(2,7)-0)))-
(x(2,9)-xincontrol(9)));
% %         end
% elseif ((com-head0>180) | ((com-head0<0) & (com-head0>=-180)))
% %         if com<=180
% %             inactuatora1=dar*(x(2,6)-
xincontrol(6))+Kphi*((Kpsi*(((com)*3.141593/180-xincontrol(7))-(x(2,7)-
0)))-(x(2,9)-xincontrol(9)));
% %         else
% %            inactuatora1=dar*(x(2,6)-xincontrol(6))+Kphi*((Kpsi*(((com-
360)*3.141593/180-xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)));
% %         end
% end

    %inactuatora2=inactuatora2+Ki_2*Kphi*(3.14*10/180-(x(2,9)-
xincontrol(9)))*dt;
    %inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*((60*3.14/180)-(x(2,7)-
xincontrol(7))))-(x(2,9)-xincontrol(9)))*dt;

   if (head0>=0 & head0<=90) & (com>=0 & com<=90)
       inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com)*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
   elseif (head0>=0 & head0<=90) & (com>=90 & com<=180)
       inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com)*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
   elseif (head0>=90 & head0<=180) & (com>=0 & com<=90)
       inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com)*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
   elseif (head0>=90 & head0<=180) & (com>=90 & com<=180) %& (com-head0)>0
       inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com)*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
   elseif (head0>=90 & head0<=180) & (com>=180 & com<=270)
       inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com)*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
   elseif (head0>=90 & head0<=180) & (com>=270 & com<=360)
       inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com)*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
   elseif (head0>=180 & head0<=270) & (com>=270 & com<=360)
       inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com)*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
   elseif (head0>=180 & head0<=270) & (com>=180 & com<=270) & (com-head0)>0
```

```
       inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com)*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
   elseif (head0>=180 & head0<=270) & (com>=0 & com<=90)
       inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com)*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
   elseif (head0>=270 & head0<=360) & (com>=0 & com<=90)
       inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com)*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
%    elseif (head0>=270 & head0<=360) & (com>=270 & com<=360) & (com-
head0)>0
%        inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com)*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
   else
       inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com-360)*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
    end
%     if (((com-head0<=180) & (com-head0>0)) | (com-head0<-180))
% %         if com>180
% %             inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com-
360)*3.141593/180-xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
% %         else
%
inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*((com*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
% %         end
%     elseif ((com-head0>180) | ((com-head0<0) & (com-head0>=-180)))
% %         if com<=180
% %
inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com)*3.141593/180-
xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
% %         else
%         inactuatora2=inactuatora2+Ki_2*Kphi*((Kpsi*(((com-
360)*3.141593/180-xincontrol(7))-(x(2,7)-0)))-(x(2,9)-xincontrol(9)))*dt;
% %         end
%     end

    inactuatora=inactuatora1+inactuatora2;
    inactuatorr=(drr-Kr)*(x(2,6)-
xincontrol(6))+(Kr*g0/xincontrol(1))*sin(x(2,9)-xincontrol(9));

    uce(1,1)=inactuatore;
    uce(2,1)=0;
    xcdote=Aelv*xce+Belv*uce;
    xce=xce+dt*xcdote;
    outactuatore=Celv*xce+Delv*uce;
    uca(1,1)=inactuatora;
    uca(2,1)=0;
    xcdota=Aail*xca+Bail*uca;
    xca=xca+dt*xcdota;
    outactuatora=Cail*xca+Dail*uca;
    ucr(1,1)=inactuatorr;
    ucr(2,1)=0;
    xcdotr=Arud*xcr+Brud*ucr;
    xcr=xcr+dt*xcdotr;
    outactuatorr=Crud*xcr+Drud*ucr;

    uaero(1)=uaerocontrol(1)+outactuatore;
    uaero(2)=uaerocontrol(2)+outactuatora;
    %if x(2,7)>0.001
    %    uaero(3)=uaerocontrol(3)+outactuatorr+0.2;
    %elseif x(2,7)<-0.001
```

```
    %     uaero(3)=uaerocontrol(3)+outactuatorr-0.2;
    %else
    uaero(3)=uaerocontrol(3)+outactuatorr;
    %end

    if x(2,12)<altu(p)-5 & goprop==0
        %uprop(2)=uprop(2)+(1500-x(2,12))/100;
        uprop(2)=30;
        uprop(1)=2000;
        goprop=1;
    elseif x(2,12)>altu(p)+5 & goprop==0
        %uprop(2)=uprop(2)-(1500-x(2,12))/100;
        uprop(2)=15;
        uprop(1)=1800;
        goprop=1;
    elseif x(2,12)>=altu(p)-5 & x(2,12)<=altu(p)+5
        uprop(2)=21;
        uprop(1)=1800;
        goprop=0;
    end

elseif int==3
    inactuator=-Kq*(x(2,5)-xinc0(5))+Ktheta*(3.14/180-(x(2,8)-
xinc0(8)))+Ki_1*Ktheta*(3.14/180-(x(2,8)-xinc0(8)))*dt*cont;
    if goeap==1
        xc(1,1)=0;
        xc(2,1)=0;
        xc(3,1)=0;
        goeap=0;
    end
    uc(1,1)=inactuator;
    uc(2,1)=0;
    xcdot=Aelv*xc+Belv*uc;
    xc=xc+dt*xcdot;
    outactuator=Celv*xc+Delv*uc
    uaero(1)=uaero0(1)+outactuator;
elseif int==4
end
```

elevator.m

```
if int==2
    %inactuator=-Kq*(x(2,5)-xinc0(5))+Ktheta*(3.14/180-(x(2,8)-
xinc0(8)))+Ki_1*Ktheta*(3.14/180-(x(2,8)-xinc0(8)))*dt*cont;
    inactuator1=-Kq*(x(2,5)-xinc0(5))+Ktheta*(3.14/180-(x(2,8)-xinc0(8)));
    if goeap==1
        inactuator2=0;
    end
    inactuator2=inactuator2+Ki_1*Ktheta*(3.14/180-(x(2,8)-xinc0(8)))*dt;
    inactuator=inactuator1+inactuator2;
    if goeap==1
        xc(1,1)=0;
        xc(2,1)=0;
        xc(3,1)=0;
        goeap=0;
    end
    uc(1,1)=inactuator;
    uc(2,1)=0;
    xcdot=Aelv*xc+Belv*uc;
```

```
    xc=xc+dt*xcdot;
    outactuator=Celv*xc+Delv*uc;
    uaero(1)=uaero0(1)+outactuator;
elseif int==3
    inactuator=-Kq*(x(2,5)-xinc0(5))+Ktheta*(3.14/180-(x(2,8)-
xinc0(8)))+Ki_1*Ktheta*(3.14/180-(x(2,8)-xinc0(8)))*dt*cont;
    if goeap==1
        xc(1,1)=0;
        xc(2,1)=0;
        xc(3,1)=0;
        goeap=0;
    end
    uc(1,1)=inactuator;
    uc(2,1)=0;
    xcdot=Aelv*xc+Belv*uc;
    xc=xc+dt*xcdot;
    outactuator=Celv*xc+Delv*uc
    uaero(1)=uaero0(1)+outactuator;
elseif int==4
end
```

uav.m

```
fil=get(handles.checkbox1,'Value');
if fil==0
    latu=str2num(get(handles.edit3,'String'));
    longu=str2num(get(handles.edit4,'String'));
    if (longu-long)>0 & (latu-lat)>0
        com=atan((longu-long)/(latu-lat))*180/3.141593;
    elseif(longu-long)<0 & (latu-lat)>0
        com=atan((longu-long)/(latu-lat))*180/3.141593+360;
    elseif(longu-long)>0 & (latu-lat)<0
        com=atan((longu-long)/(latu-lat))*180/3.141593+180;
    elseif(longu-long)<0 & (latu-lat)<0
        com=atan((longu-long)/(latu-lat))*180/3.141593+180;
    end
else
    set(handles.edit3,'String','');
    set(handles.edit4,'String','');
    if (longu(p)-long)>0 & (latu(p)-lat)>0
        com=atan((longu(p)-long)/(latu(p)-lat))*180/3.141593;
    elseif(longu(p)-long)<0 & (latu(p)-lat)>0
        com=atan((longu(p)-long)/(latu(p)-lat))*180/3.141593+360;
    elseif(longu(p)-long)>0 & (latu(p)-lat)<0
        com=atan((longu(p)-long)/(latu(p)-lat))*180/3.141593+180;
    elseif(longu(p)-long)<0 & (latu(p)-lat)<0
        com=atan((longu(p)-long)/(latu(p)-lat))*180/3.141593+180;
    end
    if ((longu(p)-long)<0.001 & (longu(p)-long)>-0.001 & (latu(p)-
lat)<0.001 & (latu(p)-lat)>-0.001)
        if p<K
            p=p+1;
            head0=head;
            goprop=0;
%             for i=1:12
%                 xincontrol(i)=x(1,i);
%             end
            %!!!!!!!!!!!!!!!!
            if (longu(p)-long)>0 & (latu(p)-lat)>0
```

```
                com=atan((longu(p)-long)/(latu(p)-lat))*180/3.141593;
            elseif(longu(p)-long)<0 & (latu(p)-lat)>0
                com=atan((longu(p)-long)/(latu(p)-lat))*180/3.141593+360;
            elseif(longu(p)-long)>0 & (latu(p)-lat)<0
                com=atan((longu(p)-long)/(latu(p)-lat))*180/3.141593+180;
            elseif(longu(p)-long)<0 & (latu(p)-lat)<0
                com=atan((longu(p)-long)/(latu(p)-lat))*180/3.141593+180;
            end
            %!!!!!!!!!!!!!!!!
        else
            set(handles.checkbox1,'Value',0);
            set(handles.togglebutton3,'Value',0);
            set(handles.togglebutton2,'Value',1);
            set(handles.edit2,'String','0');
            p=1;
        end
    end
end

heading;
```