Simulador de voo simples

Jean Marcelo Mira Junior
Programação 3
Universidade Federal de Santa Catarina
Joinville, Brasil
jeandemira@gmail.com

William Chiou Abe
Programação 3
Universidade Federal de Santa Catarina
Joinville, Brasil
williamabe1998@hotmail.com

***Resumo* — Trabalho final da disciplina de programação 3 consiste em elaborar um sistema de estabilização de uma aeronave baseado em sensores como giroscópio, altímetro e o tubo de pitot que dão parâmetros para decidir o comportamento dos servomotores da aeronave que devem movimentar as superfícies de comando.**

Palavras-chaves — Simulador, avião, orientação, objeto, cpp

I Introdução

Este projeto tem como objetivo solucionar o problema de um avião desde sua decolagem até o pouso. Após gerar dados de sua altura, velocidade e momento, o código deverá ser capaz de estabilizá-los para que a aeronave possa concluir sua viagem de maneira segura e confortável.

Os dados são gerados de forma randômica e após a análise do programa serão alterados para atingir os valores ideais para cada situação de voo.

Para nossa classe avião, ela possui três aparelhos de medições:

- Altímetro (Altura)
- Pitot (Velocidade)
- Giroscopio (Momento)

Para o giroscópio, existem momentos nos 3 eixos, designados:

- Pitch(Eixo X)
- Yaw(Eixo Y)
- Roll(Eixo Z)

Acompanhados por quatro instrumentos para alterar o valor dos mesmos:

- Leme (Roll)
- Aileron (Yaw)
- Acelerador (Velocidade)
- Profundor (Pitch e Altura)

Neste trabalho, serão analisados os dados de decolagem, cruzeiro e pouso. Em cada instante, ela possui a sua altura, velocidade e momento ideal para que tenha plena funcionalidade durante o percurso. Ao observar que as informações não estão de acordo com o ideal, o programa deve ser capaz de através de parâmetros passados pelo usuário estabilizar a aeronave utilizando os conceitos de programação 3. Todo código está comentando demonstrando a lógica utilizada na implementação, a seguir serão ressaltados pontos chaves da lógica.

II Desenvolvimento

II.A Funcionamentos e Abordagem

A ideia base do projeto é a estabilização de uma aeronave através da leitura de dados, após a leitura o programa é capaz de estabilizar e posicionar a aeronave em determinadas condições que o usuário determinar através da lógica de programação orientada a objetos.

O arquivo de dados de decolagem, cruzeiro e pouso contém na primeira coluna dados de altura, na segunda coluna dados de velocidade, na terceira coluna os dados do momento em pitch, na quarta coluna dados de momento em roll e na última e quinta coluna dados de momento em yaw.

Para a análise deste projeto específico consideramos que o momento ideal e almejado deve ser zero para todos os eixos. Nas informações de decolagem não foi imposto nenhuma velocidade mínima nem máxima, nenhuma altura mínima e uma altura máxima de 1120 m. Nas informações de cruzeiro foi imposta uma velocidade mínima 216 km/h, uma velocidade máxima de cruzeiro de 222 km/h, altura mínima de 900 m e uma altura máxima de 1000 m. E por fim nas informações de pouso não foi imposto nenhuma velocidade mínima, uma velocidade máxima de 200 km/h, nenhuma altura mínima e uma altura máxima de 800 m.

Por questões de facilitar a identificação dos conceitos chaves do trabalho vamos aplicar a cor **vermelha** para demonstrar (**conceitos chaves**) da programação orientada a objeto. E algumas figuras de funções do código serão estruturadas de modo a facilitar a visualização neste texto, mas mantendo sua estrutura nos respectivos arquivos.

II.B Gerador de Dados

A análise se baseia na leituras de arquivos de dados que são gerados através de um programa denominado gerador.cpp, o programa que gera os dados têm como modelo base o avião Cessna 172 Skyhawk que é uma aeronave monomotor americana de quatro lugares, com dois tipos de asa, alta e fixa, fabricada pela marca Cessna Aircraft Company.

Para gerar os dados da análise foram utilizados parâmetros da aeronave Cessna 172 Skyhawk (valores fictícios), como altura máxima suportada de 1270 m, velocidade de cruzeiro 222 km/h, velocidade máxima 302 km/h, comprimento 8,28 m, envergadura 11 m, altura 2,72 m e área das asas 16,2 m². Além disso foi pensando em gerar momentos em K.N/m através da área das superfícies da aeronave, esse momentos titulados de pitch, roll e yaw são causados durante o voo por forças externas ao avião como por exemplo o vento, chuva entre outros.

Iniciando a geração de dados a partir da decolagem criando o arquivo “dados-decolagem.txt” respeitando a altura máxima de 1270 m e a velocidade máxima 302 km/h visando sempre a buscar o máximo de velocidade possível para sair do chão, após isso foi criado o arquivo de “dados-cruzeiro.txt” que contém os dados de voo do avião tendo como velocidade de cruzeiro 222 km/h, esta velocidade determina a economia de combustível da aeronave. E por fim foi determinado os dados de pouso nomeados de "dados-pouso.txt" este e todos os outros dados respeitam a velocidade, altura e momentos máximos de cada processo de voo da aeronave Cessna 172 Skyhawk.

II.C Entendendo a Main.cpp

Iniciando a main.cpp com uma função intitulada de "leArquivos" que faz a leitura dos arquivos de dados e coloca estes dados na classe Aviao. Neste projeto utilizaremos o namespace SimuladorDeVooSimples que na main.cpp foi simplificada e titulada de svs. Após isso criamos um objeto da classe Aviao denominado aeronave e a partir do construtor padrão inicializamos como é possível ver na figura 1.

```
svs::Aviao aeronave("Cessna 172 Skyhawk", "Cessna Aircraft Company", 0.0, 0.0, 0.0, 0.0, 0.0);
```

Figura 1: Criando objeto da classe Aviao

Primeiro iniciamos com modelo, marca, inclinação do servomotor do profundor, leme, aileron esquerdo, aileron direito e por fim a inclinação do servomotor do acelerador. Depois criamos um objeto da classe Aeromodelo figura 2.

```
svs::Aeromodelo miniAeronave("Albatroz", "Incoaer", "Azul", "Transmissor 6ch", 0.0, 0.0, 0.0, 0.0, 0.0, 1400.55);
```

Figura 2: Criando objeto da classe Aeromodelo

Primeiro iniciamos com modelo, marca, cor, marca do transmissor, inclinação do servomotor do profundor, leme, aileron esquerdo, acelerador e por fim o preço. Além dessas, outras funções que são importantes e merecem ser ressaltadas são as funções de estabilizar a altura, velocidade e momentos. O primeiro termo dos métodos de estabilização é onde começa a análise da função, o método de alocar os dados tem formato de um vetor e portanto é necessário informar quando começa cada arquivo. O segundo termo de cada função é a altura, velocidade ou momentos máximos. E o último termo dos métodos é a altura, velocidade e momentos mínimos.

```
aeronave.estabilizaAltura(0, 1000, 900);
aeronave.estabilizaVelocidade(255, 222, 216);
aeronave.estabilizaMomentos(1255, 0, 0);
```

Figura 3: Chamando os métodos de estabilização do avião.

II.D Entendendo dados.cpp

A classe dados.cpp é uma aplicação clara de template, a classe conta com os atributos giroscópio[3] que contém os dados dos momentos (pitch, roll e yaw), e o altimetro contém os dados da altura e pitot que contém os dados da velocidade. Além disso, contém os métodos de acesso, iniciar e mudar os atributos. Esta classe é responsável pelos dados do voo. Toda classe é um exemplo de template (**Template**).

II.E Entendendo aviao.h e aviao.cpp

O arquivo aviao.h contém a classe Aviao com seus atributos privados e públicos além de seus métodos que podemos ver figura 4 que é um exemplo simples do encapsulamento da classe (**Encapsulamento**).

```
class Aviao
{
private:
    string modelo;
    string marca;
    vector<Dados<float>> dadosDoModelo;
    float servoProfundor;
    float servoLeme;
    float servoAileronEs;
    float servoAileronDi;
    float servoAcelerador;

public:
    // Sobrecarga de operador ==
    bool operator==(const Aviao &) const;

    // Métodos para mudar as variaveis
    void setModelo(string);
    void setMarca(string);
    void setServoProfundor(float);
    void setServoLeme(float);
    void setServoAileronEs(float);
    void setServoAileronDi(float);
    void setServoAcelerador(float);
    void insereDados(Dados<float> temp);
```

Figura 4: Atributos da classe Aviao.

Os atributos contém os dados do modelo modelo, marca, um vetor de dados (**Associação**), o servomotor do profundor, leme, aileron esquerdo, aileron direito e acelerador. Contendo os métodos de acesso, iniciar e mudar os atributos. As três principais funções desta classe são as que estabilizam a aeronave figura 5.

```
void estabilizaAltura(int i, float max, float min);
void estabilizaVelocidade(int i, float max, float min);
void estabilizaMomentos(int i, float max, float min);
```

Figura 5: Métodos da classe Aviao de estabilização.

O funcionamento na essência da estabilização do avião tem como princípio verificar se os servos estão na posição zero ou seja se a superfície de controle está alinhada com a estrutura do avião, após verificar e alinhar os servos e consequentemente as superfícies de controle o programa é capaz de ler os dados da classe Aviao e achar a taxa de variação constante de um instante de tempo para outro, ou seja ele consegue pegar a variação de altura, velocidade e momento esse fragmento de código pode ser visto na figura 6.

```
setServos();
float nivel = (dadosDoModelo[i + 1].getQualquerDado() - dadosDoModelo[i].getQualquerDado());
```

Figura 6: Nível e alinhamento das superfícies de comando.

Após isto, a função percorre todo o vetor de dados do avião verificando se há alguma condição de máximo ou mínimo estipulado pelo usuário para altura, velocidade e momentos. Se ao percorrer o vetor a função achar algum dado diferente do esperado ele faz a correção através de um laço de repetição while que corrige de instante de tempo em instante de tempo ou seja o vetor vai de 0 até i -1 sendo 0 o instante de tempo inicial e i -1 o instante de tempo final. E esta função é capaz de estabilizar mesmo que os momentos fiquem negativos demais ou positivos demais por interações a mais no decorrer da estabilização. Por questão de exemplificação a figura 7 traz um trecho da implementação da função que estabiliza a altura e é base para todas as outras funções de estabilização.

```
// Métodos da classe que estabiliza o avião na altura
void Aviao::estabilizaAltura(int i, float max, float min)
{
    setServos(); // Verifica se os servos estão na posição inicial (0.0) onde a superfície de coamndo
                 // enta sem inclinação
    int acesso = 0;
    // Verifica a taxa de variação
    float nivel = (dadosDoModelo[i + 1].getAltimetro() - dadosDoModelo[i].getAltimetro());
    // Se a taxa for negativa faz o módulo
    if (nivel < 0)
        nivel *= -1;
    // Percorre os dados para corrigir para o ideal
    for (int j = i; j < (int)dadosDoModelo.size(); j++)
    {
        // Se o dados forem menores que o mínimo entra e ajusta
        if (dadosDoModelo[j].getAltimetro() < min)
        {
            // Questão de acesso para ajustar o nível do profundor
            if (acesso == 0)
            {
                // Ajusta o nível do profundor
                decrementaProfundor(nivel);
                acesso++;
            }
            // Faz o ajuste para aumentar a altura
            while (dadosDoModelo[j].getAltimetro() < min)
                dadosDoModelo[j].movimentaAltimetro(servoProfundor);
        }
        // Zera o acesso e o profundor para poder ter outro movimento
        acesso = 0;
        servoProfundor = 0;
        // Se o dados forem maiores que o máximo entra e ajusta
        if (dadosDoModelo[j].getAltimetro() > max)
        {
            // Questão de acesso para ajustar o nível do profundor
            if (acesso == 0)
            {
                // Ajusta o nível do profundor
                incrementaProfundor(nivel);
                acesso++;
            }
            // Faz o ajuste para diminuir a altura
            while (dadosDoModelo[j].getAltimetro() > max)
                dadosDoModelo[j].movimentaAltimetro(servoProfundor);
        }
        // Zera o acesso e o profundor para poder ter outro movimento
        acesso = 0;
        servoProfundor = 0;
    }
}
```

Figura 7: Método de estabilizar a altura.

Os arquivos "aviao.h" e "aviao.cpp" também trazem implementações de polimorfismo (**Polimorfismo**) tanto no

construtores padrão quando o usuário não instancia o objeto e construtor no qual o usuário insira os objetos, como pode ser visto na figura 8.

```
// Construtor padrão
Aviao() : modelo{"DESCONHECIDO"}, marca{"DESCONHECIDA"}, servoProfundor{0.0}, servoLeme{0.0},
                  servoAileronEs{0.0}, servoAileronDi{0.0}, servoAcelerador{0.0} {};

// Construtor que inicializa os parâmetros
Aviao(string modelo, string marca, float servoProfundor, float servoLeme, float servoAileronEs,
      float servoAileronDi, float servoAcelerador);
```

Figura 8: Exemplo de polimorfismo nos construtores.

Além deste exemplo de polimorfismo também foi empregado outra função de polimorfismo (**Polimorfismo**) denominada de "imprimirEspecificacoes" que mostra na tela do usuário as especificações do avião, como pode ser visto na figura 9.

```
// Imprime todos os dados do avião aplicando os conceitos de polimorfismo
virtual void imprimirEspecificacoes() const;
```

Figura 9: Exemplo de polimorfismo entre duas classes.

Este método tem polimorfismo pois a seguir será inserido a classe Aeromodelo que tem herança (**Herança**) da classe Aviao e nas duas classes temos os métodos com o mesmo nome. Na classe Aviao também é possível verificar a implementação de sobrecarga de operador == (**Sobrecarga de operador**), que é utilizado para comparar dois objetos a fim de verificar se as especificações são idênticas como é possível ver na figura 10.

```
// Sobrecarga de operador ==
bool Aviao::operator==(const Aviao &aviaoDeComparacao) const
{
    if (modelo == aviaoDeComparacao.modelo && marca == aviaoDeComparacao.marca &&
        servoProfundor == aviaoDeComparacao.servoProfundor &&
        servoLeme == aviaoDeComparacao.servoLeme &&
        servoAileronEs == aviaoDeComparacao.servoAileronEs &&
        servoAileronDi == aviaoDeComparacao.servoAileronDi &&
        servoAcelerador == aviaoDeComparacao.servoAcelerador)
        return true;
    else
        return false;
}
```

Figura 9: Exemplo de sobrecarga de operador ==.

II.F Entendendo aeromodelo.h e aeromodelo.cpp

A classe Aeronave é um exemplo claro de herança (**Herança**) como é possível ver na figura 10. Essa classe é baseada em um avião de controle remoto, além do que ela herda da classe Aviao também contém os atributos cor, transmissor e preço.

```
// Classe Aeromodelo herda classe Aviao
class Aeromodelo : public Aviao
{
private:
    float preco;              // Valor do aeromodelo
    string cor;               // Cor predominante do aeromodelo
    string modeloTransmissor; // Modelo de transmisor do aeromodelo
```

Figura 10: Classe Aeromodelo que herda a classe Aviao.

Além disso na classe Aeromodelo também é explorado a o assunto de polimorfismo (**Polimorfismo**) com o método denominada de "imprimirEspecificacoes", como pode ser visto na figura 11, que mostra na tela do usuário as especificações do aeromodelo, e tem duplicidade no classe avião assim justificando o polimorfismo.

```
// Imprime todos os dados do aeromodelo aplicando
// os conceitos de polimorfismo
virtual void imprimirEspecificacoes() const;
```

Figura 11: Método para mostrar as especificações aplicando polimorfismo.

III Discussão

Após a análise do programa, percebe-se que ele estabiliza a altura, velocidade e momento de acordo com os parâmetros considerados ideais dados pelo usuário. A seguir as figuras 12,13 e 14 demonstram uma parte dos dados do "dados-decolagem.txt","dados-cruzeiro.txt" e "dados-pouso.txt" gerados antes de utilizar nossa solução para o problema, respectivamente:

```
0 0 165 248.4 0
5 200.402 170 253.4 0
10 200.803 175 258.4 0
15 201.205 180 263.4 0
20 201.606 185 268.4 0
25 202.008 190 273.4 0
30 202.409 195 278.4 0
35 202.811 200 283.4 0
40 203.213 205 288.4 0
45 203.614 210 293.4 0
50 204.016 215 298.4 0
55 204.417 220 303.4 0
60 204.819 225 308.4 0
65 205.22 230 313.4 0
70 205.622 235 318.4 0
75 206.024 240 323.4 0
80 206.425 245 328.4 0
85 206.827 250 333.4 0
90 207.228 255 338.4 0
95 207.63 260 343.4 0
100 208.031 265 348.4 0
105 208.433 270 353.4 0
110 208.835 275 358.4 0
115 209.236 280 363.4 0
120 209.638 285 368.4 0
125 210.039 290 373.4 0
130 210.441 295 378.4 0
135 210.843 300 383.4 0
140 211.244 305 388.4 0
145 211.646 310 393.4 0
```

Figura 12: Fragmento do dados de decolagem.

```
1270 302 -1.65 -248.4 -413.4
1269 301.5 -2.65 -249.4 -414.4
1268 301.5 -3.65 -250.4 -415.4
1267 301 -4.65 -251.4 -416.4
1266 301 -5.65 -252.4 -417.4
1265 300.5 -6.65 -253.4 -418.4
1264 300.5 -7.65 -254.4 -419.4
1263 300 -8.65 -255.4 -420.4
1262 300 -9.65 -256.4 -421.4
1261 299.5 -10.65 -257.4 -422.4
1260 299.5 -11.65 -258.4 -423.4
1259 299 -12.65 -259.4 -424.4
1258 299 -13.65 -260.4 -425.4
1257 298.5 -14.65 -261.4 -426.4
1256 298.5 -15.65 -262.4 -427.4
1255 298 -16.65 -263.4 -428.4
1254 298 -17.65 -264.4 -429.4
1253 297.5 -18.65 -265.4 -430.4
1252 297.5 -19.65 -266.4 -431.4
1251 297 -20.65 -267.4 -432.4
1250 297 -21.65 -268.4 -433.4
1249 296.5 -22.65 -269.4 -434.4
1248 296.5 -23.65 -270.4 -435.4
1247 296 -24.65 -271.4 -436.4
1246 296 -25.65 -272.4 -437.4
1245 295.5 -26.65 -273.4 -438.4
1244 295.5 -27.65 -274.4 -439.4
1243 295 -28.65 -275.4 -440.4
1242 295 -29.65 -276.4 -441.4
1241 294.5 -30.65 -277.4 -442.4
```

Figura 13: Fragmento dos dados de cruzeiro.

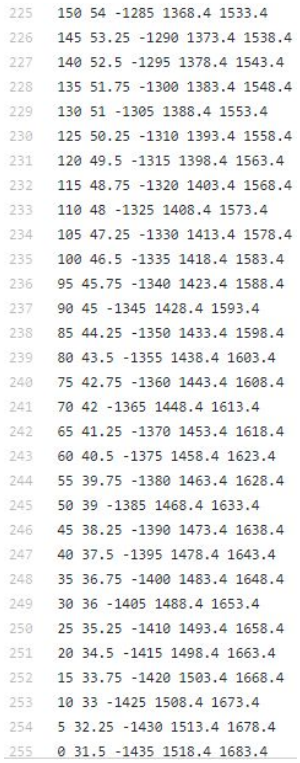

Figura 14: Fragmento dos dados de pouso.

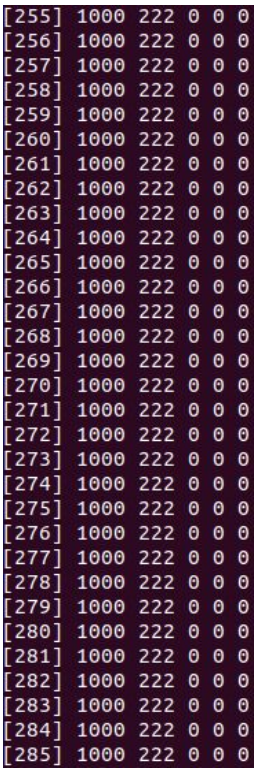

Figura 16 :Fragmento dos dados de cruzeiro (corrigido).

E as figuras 15, 16 e 17 representam os dados após a correção feita pelo nosso código:

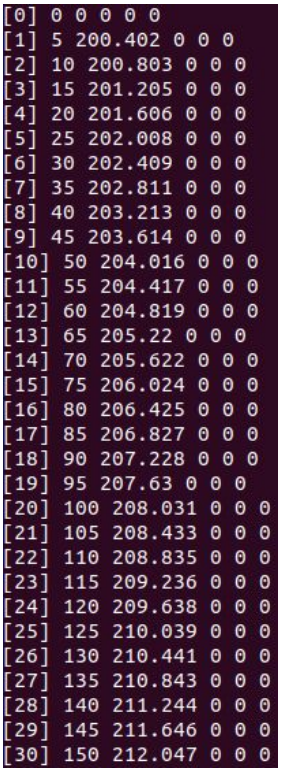

Figura 15: Fragmento dos dados de decolagem (corrigido).

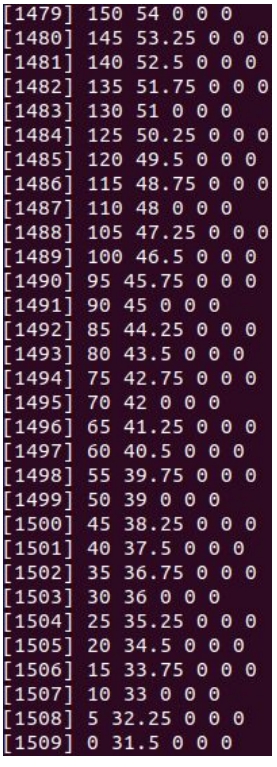

Figura 17: Fragmento dos dados de pouso (corrigido).

É possível perceber que o os dados são estabilizados de acordo com o mínimo e máximo fornecido pelo usuário através da “main.cpp” como podemos ver na figura 18:

```
// Abre os dados de decolagem e adiciona no novo avião
ifstream arquivoDecolagem;
arquivoDecolagem.open("Gerador-de-dados/dados-decolagem.txt");
leArquivos(arquivoDecolagem, aeronave);

aeronave.estabilizaAltura(0, 1120, 0); // Estabiliza a Altura (
aeronave.estabilizaMomentos(0, 0, 0);  // Estabiliza os Momento

//Abre os dados de cruzeiro e adiciona no novo avião
ifstream arquivoCruzeiro;
arquivoCruzeiro.open("Gerador-de-dados/dados-cruzeiro.txt");
leArquivos(arquivoCruzeiro, aeronave);

aeronave.estabilizaAltura(255, 1000, 900);    // Estabiliza a A
aeronave.estabilizaVelocidade(255, 222, 216); // Estabiliza a V
aeronave.estabilizaMomentos(255, 0, 0);       // Estabiliza os

//Abre os dados de cruzeiro e adiciona no novo avião
ifstream arquivoPouso;
arquivoPouso.open("Gerador-de-dados/dados-pouso.txt");
leArquivos(arquivoPouso, aeronave);

aeronave.estabilizaAltura(1255, 800, 0);    // Estabiliza a Alt
aeronave.estabilizaVelocidade(1255, 200, 0); // Estabiliza a Ve
aeronave.estabilizaMomentos(1255, 0, 0);     // Estabiliza os M
aeronave.imprimirDadosDados();               // Imprime os dado
```

Figura 18: Dados de mínimo e máximo utilizado no projeto.

IV CONCLUSÃO

Aplicando os conceitos de c++ estudado durante o semestre na disciplina de programação 3, foi possível implementar um simulador de voo simples que cumpri com toda ideia de programação orientada a objeto. Tendo assim uma aplicatividade prática.

Referências

1 WIKIPEDIA. Aircraft principal axes. Disponível em: https://en.wikipedia.org/wiki/Aircraft_principal_axes. Acesso em: 31 out. 2020.

2 SILVA, Túlio Dapper e. Instrumentação de um veículo aéreo não tripulado para análise de sua performance. 2017. 48 f. TCC (Graduação) - Curso de Engenharia de Controle e Automação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2017. Disponível em: https://lume.ufrgs.br/bitstream/handle/10183/172738/001059842.pdf;jsessionid=6088500D3F5A15B86E0E6DD05B137684?sequence=1. Acesso em: 31 out. 2020.

3 WIKIPÉDIA. Cessna 172. Disponível em: https://pt.wikipedia.org/wiki/Cessna_172. Acesso em: 31 out. 2020.