Emulação de um indicador ILS

Trabalho experimental E-10

Carolina Serra	Tiago Oliveira	João Gonçalves
80765	81024	81040

Grupo 4

Sistemas Aviónicos Integrados

MEAer

Instituto Superior Técnico

Janeiro de 2019

Conteúdo

1 Introdução

O trabalho consiste no processamento de dados de um simulador de voo e sua apresentação num indicador ILS semelhante ao de uma aeronave. O indicador é construído com recurso à biblioteca gráfica SDL, versão 2.0.9, em linguagem C.

O simulador debita um conjunto de dados de voo em tempo real, que são enviados via rede para um número arbitrário de clientes, cujo programa descrito é um exemplo. Na figura 1 indica-se o conceito da ligação em rede usada. Ao longo deste documento, pode referir-se ao simulador como servidor, e a qualquer programa que use os dados enviados por ele como cliente. As comunicações são detalhadas na secção 2.

O objectivo é construir um programa robusto que exiba toda a informação relevante, e que permita fácil configuração. Para isso, devem ser representados os indicadores de desvio do *Localizer* e *Glideslope*, assim como os alertas dos *Marker Beacons*. O utilizador interage com o programa apenas seleccionando o canal ILS da pista em que pretende aterrar, como é o geral para este sistema Os dados recebidos do simulador não são os sinais ILS, mas a posição da aeronave, logo o comportamento do sistema ILS é emulado, usando informação das pistas possíveis, que estão inseridas num ficheiro. A emulação das referências do ILS e configurações das pistas é descrita na secção 3, e na secção 4 detalha-se o conteúdo da *interface* gráfica.

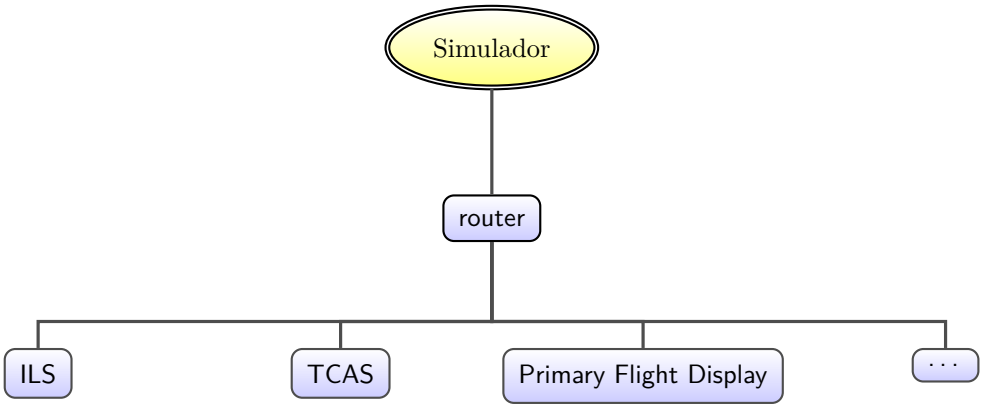

Figura 1: Ligação em rede.

2 Comunicação

2.1 Protocolo

Os dados são provenientes do simulador de voo *X-plane*. São transmitidos em tempo real, em protocolo UDP sobre IPv4. Escolhe-se o protocolo UDP, pois este é o mais usado para *streaming*, onde ter os dados mais recentes é o mais importante. Para esta aplicação, não é problemático receberem-se pacotes fora de ordem, duplicados ou até serem perdidos alguns. A outra vantagem é que sem protocolos de *handshake*, o programa toma um papel passivo na rede – o servidor não é afectado de qualquer forma pelo programa.

A ligação é feita numa rede local. Para o envio de dados, o servidor só precisa de conhecer o endereço IP e porto onde o programa se fixa. O porto é configurado no arranque, passado pela linha de comandos.

Os dados enviados em cada pacote são um *array* de *floats* com 4 bytes cada, enviado para todos os clientes. Desta forma, nem todos os dados são relevantes para cada cliente. O servidor codifica os valores em formato *big endian*, que é a norma para transmissão de valores em rede. O programa foi desenvolvido numa arquitectura *little endian*, logo os dados são convertidos à entrada.

2.2 *Multi-threading*

O programa tem duas *threads*, implementadas com a biblioteca `pthread` (POSIX):

- `main`:

  Onde são feitos todos os cálculos e é controlado o *display* gráfico. Quando uma nova amostra está disponível, os dados são processados para obter a referência de ILS mais recente.

- `reception_thread`:

  Onde é aberto o *socket* para leitura dos dados. Esta `thread` está constantemente à escuta de dados novos, e sinaliza a `main` quando existe uma nova amostra.

A comunicação entre *threads* é feita com memória partilhada (por variáveis globais), e a sincronização com um *mutex*. O papel do *mutex* é garantir que os dados estão a ser acedidos apenas por uma *thread* ao mesmo tempo.

A espera por novos dados é feita de forma activa, ou seja, a cada ciclo de amostragem, `main` verifica se existem novos dados. Esta escolha é feita de modo a que o *display* se mantenha responsivo mesmo que não estejem a chegar dados.

3 Programa

3.1 Configuração das Pistas dos Aeroportos

As pistas onde se pretende realizar uma aterragem de precisão com ILS devem ser carregadas para um ficheiro com a estrutura exemplificada de seguida:

LPPT; RWY03; 202.728484; 38º45’59.15"N; 9º08’38.04"W; 100.3; 3.0; 0.0; 0.6; 4.2; 109.1;

Cada um dos parâmetros corresponde respetivamente à:

- Identificação do aeroporto;
- Identificação e orientação da pista;
- Orientação da pista (em relação ao true north) do ponto de vista da aterragem, em graus;
- Latitude da cabeceira de pista em graus, minutos, segundos e décimas de segundo;
- Longitude da cabeceira de pista em graus, minutos, segundos e décimas de segundo;
- Altitude da cabeceira de pista em metros;
- Glide slope ou ladeira de aproximação em graus;
- Localização do Inner Marker em milhas náuticas;
- Localização do Middle Marker em milhas náuticas;
- Localização do Outer Marker em milhas náuticas;
- Frequência do ILS disponível na pista em MHz.

No ficheiro utilizado nas demonstrações constam informações das pistas 03 e 21 do Aeroporto de Lisboa e da pista 17 do Aeroporto do Porto. As informações foram recolhidas das cartas aeronáuticas, correspondendo portanto aos parâmetros reais. Por exemplo, para a pista 03 do Aeroporto de Lisboa, retiramos a informação das coordenadas do threshold do manual da NAV Portugal [AIS18] cujo excerto relevante apresentamos na Figura 2, e a informação dos marker beacons e da ladeira de aproximação retiramos das cartas aeronáuticas da Jeppesen [San06] cujo excerto relevante apresentamos na Figura 3.

2 DADOS DE REFERÊNCIA AD

Lat 384627N
Long 0090803W

Elevação:. 114m / 374FT
THR RWY03 elevação: 100.3m
THR RWY 21 elevação 105.7m
THR RWY17 elevação: 113m
THR RWY 35 elevação 101m

Figura 2: Informação da localização do threshold da pista 03 do Aeroporto de Lisboa.

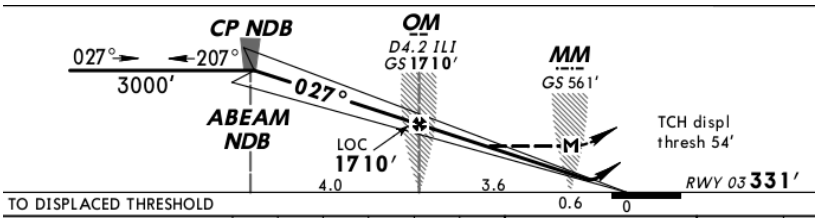

Figura 3: Excerto da carta de aproximação ILS da pista 03 do Aeroporto de Lisboa.

3.2 Conversão de Coordenadas GPS (Geodésicas) para ENU

A posição da aeronave proveniente do simulador vem expressa em coordenadas GPS, ou seja, em formato latitude(ϕ), longitude (λ) e altitude (h), segundo o Elipsóide Internacional de Referência *WGS84*. Este elipsóide tem as seguintes características [Wik18]:

$$\text{Semi-eixo maior } a = 6378137m; \tag{1}$$

$$\text{Inverso do achatamento } 1/f = 298.257223563. \tag{2}$$

De forma a determinar a posição da aeronave em relação à pista, que é necessária para sabermos se o avião está na cobertura do ILS e para obter os ângulos de afastamento da trajetória de aproximação ideal, decidimos converter as coordenadas para um referencial ENU (*East-North-Up*). Para tal, temos que utilizar um referencial ECEF (*Earth Centered, Earth Fixed*) como referencial intermédio.

A conversão de coordenadas GPS para coordenadas ECEF [Val18] consegue-se com base conjunto de equações 3 a 6, que recorrem aos parâmetros 1 e 2 do Elipsóide Internacional de Referência *WGS84*.

$$N(\phi) = \frac{a}{\sqrt{1 - f(2-f)\sin^2\phi}} \tag{3}$$

$$x_A^{ECEF} = (N+h)\cos\phi\sin\lambda \tag{4}$$

$$y_A^{ECEF} = (N+h)\cos\phi\sin\lambda \tag{5}$$

$$z_A^{ECEF} = (N(1-f)^2+h)\sin\lambda \tag{6}$$

A conversão de coordenadas ECEF para ENU [Val18] obtém-se através da matriz 7, tendo em conta que (X_A, Y_A, Z_A) correspondem às coordenadas ECEF da aeronave e que (ϕ_R, λ_R, h_R) e (X_R, Y_R, Z_R) correspondem respectivamente às coordenadas geodésicas e às coordenadas ECEF do threshold da pista.

$$\begin{bmatrix} X_A^{ENU} \\ Y_A^{ENU} \\ Z_A^{ENU} \end{bmatrix} = \begin{bmatrix} -\sin\lambda_R & \cos\lambda_R & 0 \\ -\sin\phi_R\cos\lambda_R & -\sin\phi_R\sin\lambda_R & \cos\phi_R \\ \cos\phi_R\cos\lambda_R & \cos\phi_R\sin\phi_R & \sin\phi_R \end{bmatrix} \begin{bmatrix} X_A - X_R \\ Y_A - Y_R \\ Z_A - Z_R \end{bmatrix} \tag{7}$$

Desta forma, as coordenadas da aeronave ficam expressas num referencial local cartesiano ENU (*East-North-Up*) centrado no threshold da pista.

3.3 Distância ao Threshold da Pista

Na sequência da secção anterior, a distância da posição da aeronave ao threshold da pista corresponderá à distância da aeronave à origem do referencia ENU e pode ser obtida pela equação 8.

$$d = \sqrt{X_{ENU}^2 + Y_{ENU}^2 + Z_{ENU}^2} \tag{8}$$

O alcance de um sistema ILS, como será analisado nas secções posteriores, corresponde a $18NM$. Para distâncias da aeronave à pista superiores a este valor, sabemos que o indicador ILS a bordo não receberá informação da antena.

3.4 Cobertura do Localizer

O localizer é o sistema de guiamento lateral do ILS, que permite aos pilotos alinharem a aeronave com o eixo de pista durante a aproximação e a aterragem. A cobertura garantida pelo localizer é apresentada na Figura 4.

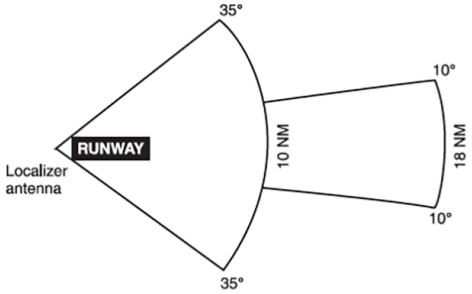

Figura 4: Cobertura de um Localizer.

Para determinar se uma aeronave se encontra dentro desta área de cobertura, recorre-se às coordenadas ENU da mesma. A orientação α da aeronave relativamente ao Norte no ponto de threshold da pista, se tomarmos em conta o esquema da Figura 5, é obtida pelo arcotangente do quociente entre as suas coordenadas x e y, como expresso na equação 9.

$$\alpha = \arctan \frac{X_{ENU}}{Y_{ENU}} \tag{9}$$

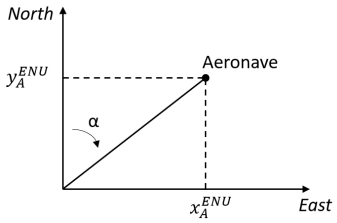

Figura 5: Esquema representativo da posição de uma aeronave no plano East-North centrado na pista.

Conclui-se portanto que a aeronave está na cobertura do localizer sempre que:

- A distância da aeronave à pista seja inferior a $18NM$ e o ângulo α apresente um desfasamento de até 10^o em relação à orientação magnética da pista;
- A distância da aeronave à pista seja inferior a $10NM$ e o ângulo α apresente um desfasamento de até 35^o em relação à orientação magnética da pista.

3.5 Cobertura do Glide Slope

Por sua vez, o glide slope é o sistema de guiamento vertical do ILS, que permite aos pilotos seguirem a ladeira de aproximação à pista recomendada. A cobertura de um sistema glide slope no plano horizontal é apresentada na Figura 6, e no plano vertical na Figura 7.

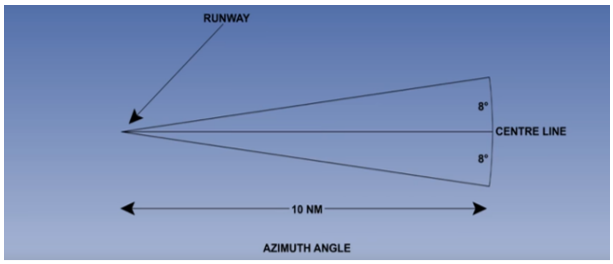

Figura 6: Cobertura no plano horizontal do Glide Slope.

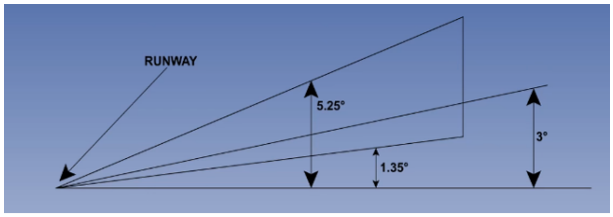

Figura 7: Cobertura no plano vertical do Glide Slope.

No plano horizontal existe cobertura quando a distância da aeronave à pista é inferior a $10NM$ e quando o ângulo α calculado na secção anterior pela equação 9 não apresente um desfasamento de até 8^o em relação à orientação magnética da pista.

Para inferir se existe cobertura no plano vertical, é necessário calcular o ângulo γ de aproximação da aeronave em relação ao threshold da pista, que tomando em conta o esquema da Figura 8, é obtido pelo arcoseno do quociente entre a coordenada z e a distância d da aeronave à pista, como expresso na equação 10.

$$\alpha = \arcsin \frac{Z_{ENU}}{d} \tag{10}$$

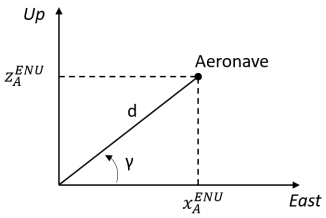

Figura 8: Esquema representativo da posição de uma aeronave no plano East-Up centrado na pista.

Conclui-se que aeronave está na cobertura do plano vertical do glide slope quando o ângulo γ de aproximação à pista tem um desfasamento entre -1.65º e +2.25º em relação à ladeira de aproximação ideal (que geralmente corresponde a 3º).

3.6 Movimento dos Ponteiros do Indicador

A agulha vertical do indicador ILS indica-nos o desfasamento lateral da posição da aeronave em relação ao eixo de pista. O movimento do ponteiro vertical tem em conta os seguintes aspetos:

- Quando a aeronave se encontra à direita do eixo de pista, a agulha encontra-se no lado esquerdo do indicador;
- Quando a aeronave se encontra à esquerda do eixo de pista, a agulha encontra-se no lado direito do indicador;
- Cada traço corresponde a um desfasamento de 0.5^{o} em relação ao eixo de pista;
- O erro de fim de escala são 2.5^{o}.

Por sua vez a agulha horizontal indica-nos o desfasamento em relação à ladeira de aproximação recomendada e o seu movimento tem em conta os seguintes aspetos:

- Quando a aeronave se encontra acima da ladeira de aproximação ideal, a agulha encontra-se na metade inferior do indicador;
- Quando a aeronave se encontra abaixo da ladeira de aproximação ideal, a agulha encontra-se na metade superior do indicador;
- Cada traço corresponde a um desfasamento de 0.14^{o} em relação à ladeira de aproximação recomendada;
- O erro de fim de escala corresponde a 0.7^{o}.

As agulhas indicam, portanto, como piloto deve corrigir o curso da aeronave. Uma situação em que as agulhas vertical e horizontal estejam respetivamente à direita e abaixo do centro do indicador, requer que o piloto corrija o curso da aeronave para a direita e para baixo, de forma a alinhar a aeronave com o eixo de pista e com a ladeira de aproximação ideal.

3.7 Passagem nos Marker Beacons

Um sistema ILS é também composto por dois ou três Marker Beacons que têm a função de fornecer ao piloto informação sobre a distância a que a aeronave se encontra do ponto de aterragem. A localização aproximada dos beacons em relação ao threshold da pista é apresentada na Figura 9.

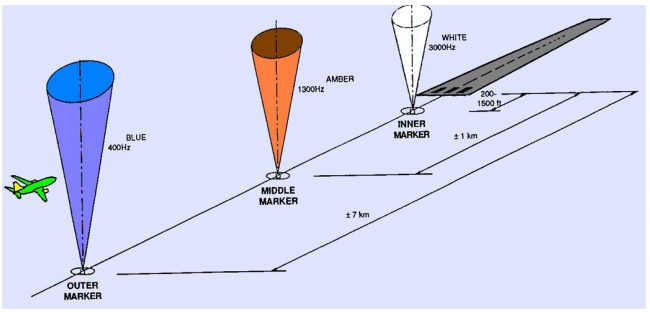

Figura 9: Representação dos Marker Beacons e da distância aproximada a que se encontram da pista.

Neste trabalho optamos por utilizar as distâncias reais. Para tal, e como referido na secção 3.1, fizemos um levantamento dos parâmetros dos Marker Beacons através da informação disponível nas cartas aeronáuticas. Desta vez apresentamos como exemplo, na Figura 10, a carta Jeppesen de aproximação da pista 21 do Aeroporto de Lisboa [San06]. A informação dos Marker Beacons foi inserida no ficheiro utilizado como base de dados dos aeroportos.

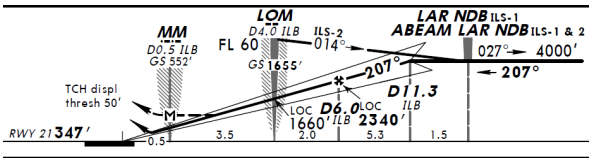

Figura 10: Excerto da carta de aproximação da pista 21 com ILS do Aeroporto de Lisboa, onde é possível adquirir informação sobre a localização real dos beacons.

4 Interface Gráfica

A interface gráfica do indicador ILS foi desenvolvida com recurso à biblioteca SDL e em linguagem C. O resultado final apresenta-se na Figura 11.

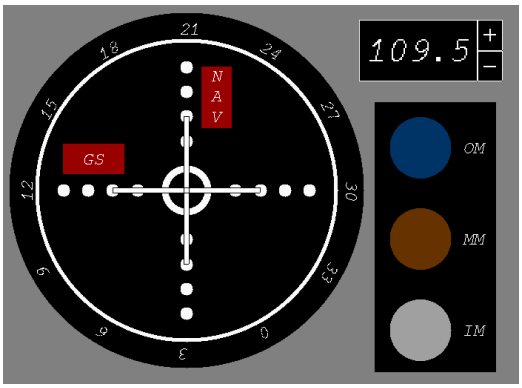

Figura 11: Interface gráfica desenvolvida para o Indicador ILS.

O indicador propriamente dito corresponde a um CDI (*Course Deviation Indicator*). Um exemplo de um CDI comercial é apresentado na Figura 12. Neste tipo de indicador, são exibidas as correções de curso a serem efetuadas pelo piloto. Se as agulhas vertical e horizontal estão respetivamente à direita e abaixo do centro do indicador, o piloto deve corrigir o curso da aeronave para a direita e para baixo, de forma a alinhar a aeronave com o eixo de pista e com a ladeira de aproximação ideal.

Um indicador CDI também incorpora um indicador de *heading*. Quando utilizado como instrumento de aproximação à aterragem com ILS, o indicador de *heading* é automaticamente ajustado para a orientação da pista, através da informação recebida das estações solo. Isto significa que o manípulo OBS (*Omni Bearing Selector*), utilizado para selecionar as radiais quando o instrumento está sintonizado à frequência de uma estação VOR, não tem utilidade quando o CDI está sincronizado a um ILS.

Na interface gráfica desenvolvida também incluímos um display da frequência em que o CDI está sintonizado, idêntica às incorporadas nos pilotos automáticos. Por fim, incluímos também um recetor e indicador de Marker Beacons idêntico ao da Figura 13. A função deste aparelho é emitir um sinal visual intermitente (isto é, piscar) quando a aeronave se encontra dentro da cobertura de um Marker Beacon.

Figura 12: Indicador CDI.

Figura 13: Set de Marker Beacons.

5 Funcionalidades

O indicador desenvolvido é composto por 8 elementos, enumerados na Figura 14.

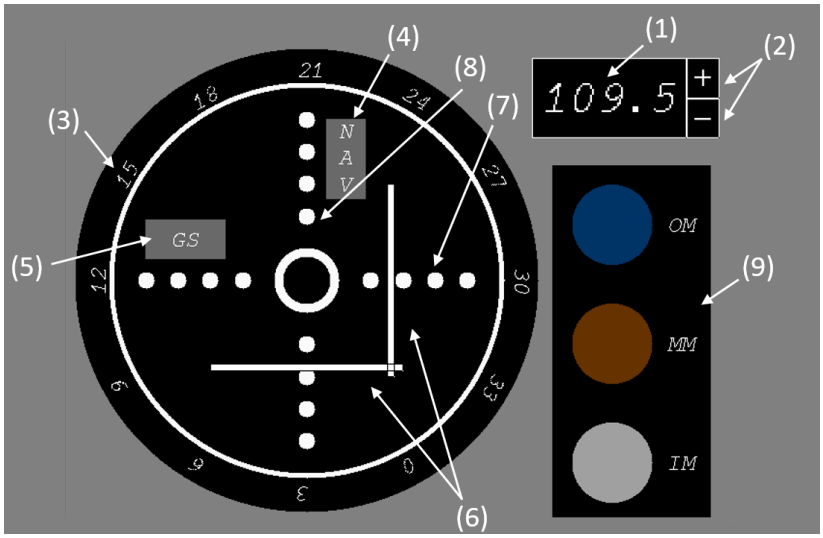

Figura 14: Interface gráfica desenvolvida para o Indicador ILS.

Estes indicadores correspondem respetivamente ao:

1. **Display da Frequência Selecionada:**

   Permite ao piloto inferir a frequência a qual o CDI está sintonizado. De forma a que o indicador ILS receba e interprete a informação proveniente do estação disponível na pista do aeroporto, não basta que a aeronave se encontre na área de cobertura. É também necessário que o indicador esteja sintonizado na frequência correspondente a esse sistema ILS. Neste trabalho utilizamos e carregamos para a base de dados dos aeroportos as frequências reais, disponíveis por exemplo nas cartas aeronáuticas. Por exemplo, para a pista 21 do Aeroporto de Lisboa, a frequência do sistema ILS corresponde a 109.5 MHz [San06]. Isto significa que, no programa desenvolvido, o indicador só recebe informação útil dos sistemas ILS disponíveis nesta pista quando sintonizado nos 109.5MHz e quando a aeronave se encontra na zona de cobertura. As frequência de cada pista encontram-se disponíveis no ficheiro utilizado como base de dados e foram obtidas das cartas aeronáuticas, como já referimos na secção 3.1.

2. **Botões de Seleção de Frequência:**

   Permitem ao piloto sintonizar o CDI para uma frequência distinta. Semelhantes aos disponíveis em alguns auto pilotos comerciais.

3. **Indicador de Heading:**

   O indicador de heading que programamos ajusta-se automaticamente à orientação da pista sintonizada. Apresenta, portanto, um comportamento semelhante a um indicador CDI real, como documentado na secção 4.

4. **Bandeira de Estado NAV:**

   A bandeira de estado NAV, quando exibida a vermelho como na Figura 11, indica que o sinal recebido da antena *localizer* disponível na pista não tem qualidade suficiente para navegação, ou por uma anomalia no transmissor ou porque a aeronave não está na zona de cobertura da antena do *localizer*. Isto significa que quando a bandeira se encontra a cinzento (que corresponde a apagada), a aeronave encontra-se na zona de cobertura e a receber informação útil da estação-terra, pelo que as indicações da agulha vertical exibidas no CDI podem ser utilizadas para a aproximação à pista.

5. **Bandeira de Estado GS:**

   Tem um funcionamento semelhante à bandeira de estado NAV, mas em vez de avaliar a qualidade proveniente da antena do *localizer* avalia a qualidade da antena do *glide slope*.

6. **Agulha vertical e horizontal:**

   A agulha vertical indica o desvio no plano lateral, em graus, da posição da aeronave em relação ao eixo de pista. A posição da agulha vertical obedece aos seguintes critérios:

   - Quando a aeronave se encontra à direita do eixo de pista, a agulha encontra-se no lado esquerdo do indicador;
   - Quando a aeronave se encontra à esquerda do eixo de pista, a agulha encontra-se no lado direito do indicador.

   A agulha horizontal indica o desvio vertical, em graus, da posição da aeronave em relação à ladeira de aproximação ideal. A posição desta agulha obedece aos seguintes critérios:

   - Quando a aeronave se encontra acima da ladeira de aproximação ideal, a agulha encontra-se na metade inferior do indicador;
   - Quando a aeronave se encontra abaixo da ladeira de aproximação ideal, a agulha encontra-se na metade superior do indicador.

7. **Escala Horizontal:**

   A escala horizontal está calibrada de acordo com os valores das especificações de um indicador CDI sintonizado a uma estação ILS com *localizer*:

   - Cada traço corresponde a um desfasamento de 0.5^{o} em relação ao eixo de pista;
   - O erro de fim de escala são 2.5^{o}.

8. **Escala Vertical:**

   A escala vertical está também calibrada de acordo com os valores das especificações de um indicador CDI sintonizado a uma estação ILS com *glide slope*:

   - Cada traço corresponde a um desfasamento de 0.14^{o} em relação à ladeira de aproximação recomendada;
   - O erro de fim de escala corresponde a 0.7^{o}.

9. **Painel dos Marker Beacons:**

   Neste painel sempre que a aeronave se encontrar na cobertura dos marker beacons o piloto recebe um aviso visual, que corresponde ao piscar da respetiva luz indicadora. Na interface gráfica teve-se os seguintes cuidados, de forma a respeitar as convenções utilizadas nestes dispositivos:

   - OM que corresponde ao Outer Beacon está representado com a cor azul e pisca com a menor frequência.
   - MM que corresponde ao Middle Beacon está representado com a cor Ambar e pisca com uma frequência intermédia.
   - IM que corresponde ao Inner Beacon está representado com a cor Branca e pisca com a maior frequência.

6 Testes e Resultados

Para testarmos todas as funcionalidades e nuances do nosso trabalho realizamos dois tipos de testes:

- Testes com trajetórias de aterragem obtidas através do Google Maps com posterior interpolação entre os pontos recolhidos. Estas trajetórias têm a particularidade de exigir um constante movimento das agulhas e passam na zona de cobertura de todos os marker beacons. Estes testes podem ser realizados a qualquer momento e encontram-se disponíveis na pasta "*stub 2*"do zip que acompanha este relatório. Estes testes também simulam todas as fases do programa, incluindo a receção de dados.
- Testes com o simulador X-Plane que permitem ao piloto fazer a aproximação à pista através da leitura da informação proveniente do indicador desenvolvido. O simulador modela toda a dinâmica do avião e do ambiente envolvente, pelo que estes testes são de grande fidelidade. Nestes testes existe efetivamente comunicação entre dois computadores distintos e o piloto tem total liberdade no movimento da aeronave.

Em ambos os tipos de testes obtivemos um correto funcionamento do indicador. Tivemos o cuidado de analisar todos os aspetos desenvolvidos:

- A comunicação e receção de dados;
- O comportamento da frequência de sintonização do CDI e o funcionamento do display e dos respetivos botões;
- O ajuste do indicador de heading para a orientação da pista à qual o CDI está sintonizado;
- O movimento dos ponteiros e a calibração da escala;
- O comportamento das flags de aviso NAV e GS;
- E o funcionamento dos marker beacons nas distâncias à pista expressas nas cartas aeronáuticas.

Após um conjunto considerável de testes em diferentes condições e pistas e onde avaliamos sistematicamente cada uma das peculiaridades e comportamentos enumerados, concluímos que o indicador CDI funciona de acordo com o esperado.

Referências

[AIS18] NAV Portugal AIS - Aeronautical Information Services. *Manual VFR Lisboa AD*. 2018. URL: https://www.nav.pt/docs/AIS/aerodromos/%20%5C%5C%20lisboaa7bd57cda4b86c298a71ff00009f255e.pdf?sfvrsn=34 (acedido em 20/01/2019).

[San06] Jeppesen Sanderson. *Lisbon Airport LPPT (Lisbon)*. 2006. URL: http://www.pht-formation.fr/ops/SID/20STAR/LPPT-Libonne.pdf (acedido em 20/01/2019).

[Wik18] Wikipedia. *World Geodetic System*. 2018. URL: https://en.wikipedia.org/wiki/World_Geodetic_System (acedido em 20/01/2019).

[Val18] Rafael Vázquez Valenzuela. «Fundamentos de Navegación Aérea Tema 2: Modelos de la superficie terrestre. Geodesia y cartografía. Rutas aéreas.» Em: (2018).

[Had04] Ibrahim Haddad. «UNIX systems programming: communication, concurrency and theory». Em: *Linux Journal* 2004.118 (2004), p. 15.

A Documentação do código

A.1 position Struct Reference

Relative Position to a referential.

`#include <ils.h>`

Data Fields

- double x
- double y
- double z

A.1.1 Detailed Description

Relative Position to a referential.

A.1.2 Field Documentation

x `double position::x`
X-offset in meter

y `double position::y`
Y-offset in meter

z `double position::z`
Z-offset in meter

A.2 position_gps Struct Reference

Geographical Position.

`#include <ils.h>`

Data Fields

- float latitude
- float longitude
- float altitude

A.2.1 Detailed Description

Geographical Position.

A.2.2 Field Documentation

altitude `float position_gps::altitude`
Altitude, in meter

latitude `float position_gps::latitude`
Latitude, in radian

longitude `float position_gps::longitude`
Longitude, in radian

A.3 runway Struct Reference

runway data

`#include <ils.h>`

Data Fields

- double or_local
- double orientacao
- double latitude_thr
- double longitude_thr
- double altitude_thr
- double x_ecef_thr
- double y_ecef_thr
- double z_ecef_thr
- double gs
- double im
- double mm
- double om
- double freq

A.3.1 Detailed Description

runway data

A.3.2 Field Documentation

altitude_thr `double runway::altitude_thr`
Runway threshold height, in meter

freq `double runway::freq`
Runway ILS frequency, in MHz

gs `double runway::gs`
Glide slope, in radian

im `double runway::im`
Inner marker location, in meter

latitude_thr `double runway::latitude_thr`
Runway threshold latitude, in radian

longitude_thr `double runway::longitude_thr`
Runway threshold longitude, in radian

mm `double runway::mm`
Middle marker location, in meter

om `double runway::om`
Outer marker location, in meter

or_local `double runway::or_local`
Runway orientation (magnetic)

orientacao `double runway::orientacao`
Runway orientation (true)

x_ecef_thr `double runway::x_ecef_thr`
Runway threshold x-position (ECEF)

y_ecef_thr `double runway::y_ecef_thr`
Runway threshold y-position (ECEF)

z_ecef_thr `double runway::z_ecef_thr`
Runway threshold z-position (ECEF)

A.4 ils.h File Reference

Header file for the ILS auxiliary functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <GL/gl.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
```

Include dependency graph for ils.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct runway
  runway data
- struct position_gps
  Geographical Position.
- struct position
  Relative Position to a referential.

Macros

- #define **PI** 3.14159
- #define **DEG_to_RAD** (PI/180.0)
- #define **RAD_to_DEG** (180.0/PI)
- #define **NM_to_m** 1852
- #define **max_num_rwys** 10
- #define **fps** 30
- #define **w_width** 700
- #define **w_height** 500

Functions

- void import_info_runways (char *file, struct runway *rwy, int *num_rwys)
  imports runway data from text file to rwy array
- void runway_coordinates_to_ecef (struct runway *rwy, int num_rwys)
  converts runway GPS coordinates into Earth Centered, Earth Fixed coordinates
- void detect_sel_runway (struct runway *rwy, int num_rwys, int sel_freq, int *sel_rwy)
  identifies the runway that corresponds to the selected frequency
- void coordenadas_gps_to_ecef (struct position_gps p, struct position *p_ecef)
  converts GPS coordinates into Earth Centered, Earth Fixed coordinates

- void coordenadas_ecef_to_enu (struct runway *rwy, int sel_rwy, struct position p_ecef, struct position *p_enu)

  converts Earth Centered, Earth Fixed coordinates into East North Up coordinates relative to the runway
- void distance_to_runway (struct position p_enu, double *dist_rwy)

  determines the distance between the aircraft and the runway
- void in_localizer (struct position p_enu, struct runway *rwy, int sel_rwy, int *in_loc, double *loc_ang)

  checks if aircraft in position p_enu is captured by the localizer
- void normalizar_angulo (double *alpha)

  *normalizes angle to the interval [0, 2*pi]*
- void in_glide_slope (struct position p_enu, struct runway *rwy, int sel_rwy, double dist_rwy, int *in_gs, double *gs_ang, double loc_ang)

  checks if aircraft in position p_enu captures the glideslope
- void movimento_ponteiro_localizer (double loc_ang, double *x_sum_pt)

  Moves the localizer marker.
- void movimento_ponteiro_glide_slope (double gs_ang, double *y_sum_pt)

  Moves the glideslope marker.
- void in_markers (struct position p_enu, struct runway *rwy, int sel_rwy, double loc_ang, int *im_on, int *mm_on, int *om_on)

  checks if aircraft in passing the markers
- void draw_indicator (SDL_Renderer *renderer)

  draws the base of the ILS indicator
- void draw_circle (SDL_Renderer *renderer, float x0, float y0, float raioext, float raioint)

  draws a circle centered in (x0, y0) with an outer radius of raioext and an inner radius of raioint
- void draw_CDI (SDL_Renderer *renderer, double x_sum_pt, double y_sum_pt)

  draws the Course Deviation Indicator on the ILS indicator
- void draw_beacons (SDL_Renderer *renderer, int im_on, int mm_on, int om_on, int *b_on)

  draws marker beacon lights either off or blinking
- void draw_text (SDL_Renderer *renderer, int x, int y, float angle, int size, char *text, TTF_Font *font)

  Draws text on the graphical window.
- void draw_compass (SDL_Renderer *renderer, TTF_Font *font, float course)

  Draws the compass around the marker.
- void draw_state (SDL_Renderer *renderer, int nav_flag, int gs_flag, TTF_Font *font)

  draws GS and NAV flags on the ILS indicator
- void write_freq (SDL_Renderer *renderer, float frequency, TTF_Font *font)

  writes the selected frequency on the top right corner of the window
- int botao (SDL_Renderer *renderer, int *sel_freq)

  defines buttons to change the selected frequency
- void name_markers (SDL_Renderer *renderer, TTF_Font *font)

  draws a black box where the markers will be placed and writes each marker's identification by their right side,

A.4.1 Detailed Description

Header file for the ILS auxiliary functions.

Author

João Gonçalves, Tiago Oliveira, Carolina Serra

Date

23 Jan 2019

Defines data structures and functions to operate on streamed data and generate ILS reference signals. Also contains procedures to draw them on a SDL renderer.

A.4.2 Function Documentation

botao() `int botao (`
`SDL_Renderer * renderer,`
`int * sel_freq )`

defines buttons to change the selected frequency

Draws two boxes next to the frequency, a + in the top box and a - in the bottom When the mouse button is pressed, gets the mouse position and compares it to the boxes' coordinates Adds or removes 0.1 to the selected frequency depending on which box is clicked Also polls for a window closing event, returning the state

Parameters

renderer	pointer to renderer
sel_frec	pointer to the selected frequency

Returns

0 if the 'x' was pressed, 1 otherwise

coordenadas_ecef_to_enu() `void coordenadas_ecef_to_enu (`
`struct runway * rwy,`
`int sel_rwy,`
`struct position p_ecef,`
`struct position * p_enu )`

converts Earth Centered, Earth Fixed coordinates into East North Up coordinates relative to the runway

Parameters

rwy	pointer to the ordered array of runways
sel_rwy	position of the selected runway in the array
p_ecef	aircraft coordinates in the Earth Centered, Earth Fixed referential
p_enu	aircraft coordinates in the East North Up referential, relative to the runway

coordenadas_gps_to_ecef() `void coordenadas_gps_to_ecef (`

```
struct position_gps p,
struct position * p_ecef )
```

converts GPS coordinates into Earth Centered, Earth Fixed coordinates

Parameters

p	desired point's GPS coordinates
p_ecef	pointer to aircraft coordinates in the Earth Centered, Earth Fixed referential

detect_sel_runway() `void detect_sel_runway (`

```
struct runway * rwy,
int num_rwys,
int sel_freq,
int * sel_rwy )
```

identifies the runway that corresponds to the selected frequency

For each runway in the rwy array checks if the runway frequency is equal to the selected frequency If yes, gives the value of the position of the selected runway in the array to the variable sel_rwy If none of the runways in the array have the desired frequency, sel_rwy is set to -1

Parameters

rwy	pointer to the ordered array of runways
num_rwys	total number of runways in the array
sel_freq	frequency selected in the ILS system
sel_rwy	pointer to the position of the selected runway in the array

distance_to_runway() `void distance_to_runway (`

```
struct position p_enu,
double * dist_rwy )
```

determines the distance between the aircraft and the runway

Uses the formula d=sqrt(x^2+y^2+z^2) to determine the distance Uses relative coordinates of the aircraft to the runway

Parameters

p_enu	aircraft coordinates in the East North Up referential, relative to the runway
dist_rwy	pointer to the runway distance

draw_beacons() `void draw_beacons (`

```
SDL_Renderer * renderer,
int im_on,
int mm_on,
int om_on,
int * b_on )
```

draws marker beacon lights either off or blinking

Draws three circles, corresponding to each marker light, in dark colours to represent the lights turned off Counts the number of cycles, restarting the count after half a second Checks if each marker is on, if yes, compares the cycle to a designated time for each marker, in order

to get the markers lights to blink at different speeds At that time, the cycle count is restarted so the lights will be turned off on the next cycle At half of that time, it is drawn a circle in a bright colour, on top of the previously drawn marker light, to represent the light turning on

Parameters

renderer	pointer to renderer
im_on	integer that defines whether or not the inner marker is activated
mm_on	integer that defines whether or not the middle marker is activated
om_on	integer that defines whether or not the outter marker is activated
b_on	integer that defines whether or not the current marker's light is on

draw_CDI() `void draw_CDI (`
`SDL_Renderer * renderer,`
`double x_sum_pt,`
`double y_sum_pt )`

draws the Course Deviation Indicator on the ILS indicator

Draws a vertical line deviated x_sum_pt pixels from the vertical axis of the indicator Draws a horizontal line deviated y_sum_pt pixels from the horizontal axis of the indicator

Parameters

renderer	pointer to renderer
x_sum_pt	number of pixels that the vertical CDI should be deviated from the center of the indicator
y_sum_pt	number of pixels that the horizontal CDI should be deviated from the center of the indicator

draw_circle() `void draw_circle (`
`SDL_Renderer * renderer,`
`float x0,`
`float y0,`
`float raioext,`
`float raioint )`

draws a circle centered in (x0, y0) with an outer radius of raioext and an inner radius of raioint

Uses the equations x=x0+r∗cos(teta) and y=y0+r∗sin(teta) to draw every point of the desired circle, with raioint<r<raioext and 0<teta<2∗PI The larger the radius is, the more angles need to be drawn in the circle to get it completely filled, but that can also severely impact our program's speed For that reason, the step by which the teta angle is increased in each cycle is inversely related to the current radius, with a 1/8 factor that was obtained through trial an error to get the biggest step possible (minimizing time) but a completely filled large circle

Parameters

renderer	pointer to renderer
x0	horizontal coordinate of the circle's center
y0	vertical coordinate of the circle's center
raioext	exterior radius of the circle
raioint	interior radius of the circle

draw_compass() `void draw_compass (`
`SDL_Renderer * renderer,`
`TTF_Font * font,`
`float course )`

Draws the compass around the marker.

Draws the numbers 0 3 6 9 12 15 18 21 24 27 30 33 on a circle, which represent the tens part of the full 360 degrees. Used to navigate on a given radial, which in this case is the runway heading. Calls draw_text() internally.

Parameters

renderer	pointer to the renderer object
font	pointer to the font object
course	angle to be left on top

draw_indicator() `void draw_indicator (`
`SDL_Renderer * renderer )`

draws the base of the ILS indicator

Clears the renderer, adding a grey background to the window Draws a black circle that will be the background of the indicator Draws a white small circunference in the center that will be the first angle marker Draws 3 white dots up, down, right and left of the circunference, equally spaced between them, that will be other angle markers

Parameters

renderer	pointer to renderer

draw_state() `void draw_state (`
`SDL_Renderer * renderer,`
`int nav_flag,`
`int gs_flag,`
`TTF_Font * font )`

draws GS and NAV flags on the ILS indicator

Draws two boxes on the ILS indicator Writes GS on the horizontal box, and NAV on the vertical one Depending if the aircraft is in range, the boxes are red or grey

Parameters

renderer	pointer to renderer
nav_flag	integer that defines whether the NAV flag should be ON or OFF
gs_flag	integer that defines whether the GS flag should be ON or OFF
font	pointer to the font which will be used to write the flags

draw_text() `void draw_text (`
`SDL_Renderer * renderer,`
`int x,`
`int y,`

```
float angle,
int size,
char * text,
TTF_Font * font )
```

Draws text on the graphical window.

Uses a font object to render text in the display.

Parameters

renderer	pointer to the renderer
x	x-position, where the text should start (counting from the upper left corner)
y	y-position, where the text should start (counting from the upper left corner)
angle	rotation of the text from the horizontal
size	scaling factor
text	string to be rendered
font	pointer to the font object

import_info_runways() `void import_info_runways (`

```
char * file,
struct runway * rwy,
int * num_rwys )
```

imports runway data from text file to rwy array

Reads each line of the text document, storing the data in the runway array Counts the number of lines in the text documents (i.e. the number of runways), putting it in the num_rwys variable

Parameters

file	pointer to the name of the text file with runway data
rwy	pointer to the ordered array of runways
num_rwys	pointer to the total number of runways in the array

in_glide_slope() `void in_glide_slope (`

```
struct position p_enu,
struct runway * rwy,
int sel_rwy,
double dist_rwy,
int * in_gs,
double * gs_ang,
double loc_ang )
```

checks if aircraft in position p_enu captures the glideslope

Compares the position of the arcraft to the glideslope's coverage If aircraft is within the covered area of the glideslope, in_gs is set to 1

Parameters

p_enu	aircraft coordinates in the East North Up referential, relative to the runway
rwy	pointer to the ordered array of runways
sel_rwy	position of the selected runway in the array
dist_rwy	distance between aircraft and runway

Parameters

in_gs	pointer to integer that defines whether or not the aircraft is captured by the localizer
gs_ang	angle between runway direction and aircraft's vertical position
loc_ang	angle between runway direction and aircraft's horizontal position

in_localizer() `void in_localizer (`
`struct position p_enu,`
`struct runway * rwy,`
`int sel_rwy,`
`int * in_loc,`
`double * loc_ang )`

checks if aircraft in position p_enu is captured by the localizer

Compares the position of the arcraft to the localizer's coverage If aircraft is within the covered area of the localizer, in_loc is set to 1

Parameters

p_enu	aircraft coordinates in the East North Up referential, relative to the runway
rwy	pointer to the ordered array of runways
sel_rwy	position of the selected runway in the array
in_loc	pointer to integer that defines whether or not the aircraft is captured by the localizer
loc_ang	angle between runway direction and aircraft position

in_markers() `void in_markers (`
`struct position p_enu,`
`struct runway * rwy,`
`int sel_rwy,`
`double loc_ang,`
`int * im_on,`
`int * mm_on,`
`int * om_on )`

checks if aircraft in passing the markers

Compares the position of the aircraft to the position of each marker and their respective coverage. If the aircraft is at the outer marker coverage om_on is set to 1. In the same way, if the aircraft is at middle marker mm_on is set to 1 and if the aircraft is at the inner marker im_on is set to 1

Parameters

p_enu	aircraft coordinates in the East North Up referential, relative to the runway
rwy	pointer to the ordered array of runways
sel_rwy	position of the selected runway in the array
loc_ang	angle between runway direction and aircraft position
im_on	pointer to the integer that defines whether or not the aircraft is passing the inner marker
mm_on	pointer to the integer that defines whether or not the aircraft is passing the middle marker

Parameters

om_on	pointer to the integer that defines whether or not the aircraft is passing the outer marker

movimento_ponteiro_glide_slope() `void movimento_ponteiro_glide_slope (`
`double gs_ang,`
`double * y_sum_pt )`

Moves the glideslope marker.

Parameters

gs_angle	glideslope angle
y_sum_pt	pointer to the marker coordiates, to be changed

movimento_ponteiro_localizer() `void movimento_ponteiro_localizer (`
`double loc_ang,`
`double * x_sum_pt )`

Moves the localizer marker.

Parameters

loc_angle	Localizer angle
x_sum_pt	pointer to the marker coordiates, to be changed

name_markers() `void name_markers (`
`SDL_Renderer * renderer,`
`TTF_Font * font )`

draws a black box where the markers will be placed and writes each marker's identification by their right side,

Parameters

renderer	pointer to renderer
font	pointer to the font which will be used to write the marker's identification

normalizar_angulo() `void normalizar_angulo (`
`double * alpha )`

normalizes angle to the interval [0, 2∗pi]

Parameters

alpha	pointer to the angle to be normalized, in radians

runway_coordinates_to_ecef() `void runway_coordinates_to_ecef (`
`struct runway * rwy,`
`int num_rwys )`

converts runway GPS coordinates into Earth Centered, Earth Fixed coordinates
Completes the structure of each runway in the array with the obtained coordinates

Parameters

rwy	pointer to the ordered array of runways
num_rwys	pointer to the total number of runways in the array

write_freq() `void write_freq (`
`SDL_Renderer * renderer,`
`float frequency,`
`TTF_Font * font )`

writes the selected frequency on the top right corner of the window
Converts frequency float value to a string Draws a black box on the top right corner of the window Displays the string on the box

Parameters

renderer	pointer to renderer
frequency	value of the selected frequency to be shown
font	pointer to the font which will be used to write the selected frequency

A.5 reception_thread.h File Reference

Data reception handler.

```
#include <pthread.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <math.h>
#include "ils.h"
```

Include dependency graph for reception_thread.h: This graph shows which files directly or indirectly include this file:

Functions

- float float_swap (float value)
  Reverses endianess of a floating point value.
- void * reception_thread (void *ptr)
  Data reception thread.

Variables

- pthread_mutex_t **m**

- int **ready**
- struct position_gps **position**

A.5.1 Detailed Description

Data reception handler.

Author

João Gonçalves, Tiago Oliveira, Carolina Serra

Date

23 Jan 2019

Provides a thread to listen on a UDP socket for incomming data, and store it appropriately. Coded inline.

A.5.2 Function Documentation

float_swap() `float float_swap (`
`float` *value* `)`

Reverses endianess of a floating point value.

Network transmission is by defauld big-endian, while most machines are little-endian.

Parameters

value	floating point value (4 bytes)

Returns

same floating point value with reversed endianess

reception_thread() `void* reception_thread (`
`void *` *ptr* `)`

Data reception thread.

Opens a UDP socket to listen for incomming data and store it on a global data structure. Converts the endianess of values based on the macro BIGENDIAN being defined or not. Calls exit() in case there is an error opening or binding the socket (common cause is the port already being in use). Runs indefinately.

Parameters

ptr	pointer to an uint32_t port number

Returns

Not used