TÉCNICO LISBOA

Flight Management System and Dead-Reckoning Navigation

Design of a Flight Management System and implementation of dead-reckoning navigation

Pedro Afonso **66277**
João Manito **73096**
Daniel de Schiffart **81479**

Instituto Superior Técnico
Integrated Master's Degree in Aerospace Engineering
Integrated Avionic Systems

2018/2019

Contents

Abstract

For the first laboratory of the course of Integrated Avionic Systems the objective was to design a simple version of a *Flight Management System* and use it to simulate a navigation across a series of pre-defined waypoints across a sphere-shaped earth. Further on, the development focused on the study of dead-reckoning navigation, its implementation within the Flight Management System and the comparison of a simulation with this feature against the original simulation. The final part of the laboratory shifted the focus to possible errors within the acquisition of flight velocity within the flight and ways to reduce these errors to obtain more accurate navigation. The entire work was to be implemented in C code and use a basic interface of both terminal and text-files for input and output of information.

Part I
Route Distance

The first part of this laboratory project was to develop the basic functions to allow for basic simulated navigation. With the final objective of this part being to determine the total distance of a path comprised of a series of waypoints, the work was split into different components to allow for streamlined development.

1 Definition of Coordinates on a Spherical Earth

On a spherical Earth, the coordinates are given in latitude and longitude in relation to a reference equator and meridian, while altitude is given as the difference between a certain point's distance to the center of the Earth and the spherical Earth's radius. The value of the Earth's radius was defined as being 6378000 meters.

With this environmental information, the obvious approach for saving the data of waypoints and positions in the code would be with a definition of a C structure. Our code implements a `struct` with the `typedef` name of `Coord`, as represented in the file `lab1.h`.

```
15 typedef struct {
16     double latitude;
17     double longitude;
18     double altitude;
19 } Coord ;
```

2 Distance Determination

To determine the distance between two coordinates on a spherical earth, the first objective is to determine the distance of the shortest great-circle path. As we assume that the altitude (and therefore the radius) of the path is constant for each segment of the flight, this can be done using the radial distance between the two points and multiplying it by the radius of the path. Using ϕ as latitude and λ as latitude, the radial distance θ between points 1 and 2 is given by equation 1.

$$\theta = \cos^{-1}\left(\cos\phi_2 \cos\left(\lambda_1 - \lambda_2\right)\cos\phi_1 + \sin\phi_2 \sin\phi_1\right) \tag{1}$$

Defining the Earth radius as R and the height of any coordinate as h, we can finally obtain the distance of the great-circle path between two coordinates as d, which is given by equation 2.

$$d = \theta \times (R + h_1) \tag{2}$$

These equations have been implemented as C functions in the file `lab1.c` as visible below.

```
double coord_dist(Coord coord1, Coord coord2){
    /* Accepts a coord struct.
     * Returns the great circle distance between coordinates in meters. */

    double ortho = acos(cos(coord2.latitude) * cos(coord1.longitude - coord2.longitude)
    ↪  * cos(coord1.latitude) + sin(coord2.latitude) * sin(coord1.latitude));

    return (ortho * (EARTH_RADIUS + coord1.altitude));
}
```

Applying this to a sequencial set of waypoints is a matter of iteration and cumulative sums to determine the total distance. However, an issue is raised when the altitude is not constant between points, as we consider the altitude constant between any two waypoints using the value of the first point of any segment. This will lead to some error in the measurement of altitudes, as sudden jumps of altitude will distort the distance travelled from reality by a little bit. The focus of this discussion is illustrated by figure 1 in two dimensions.

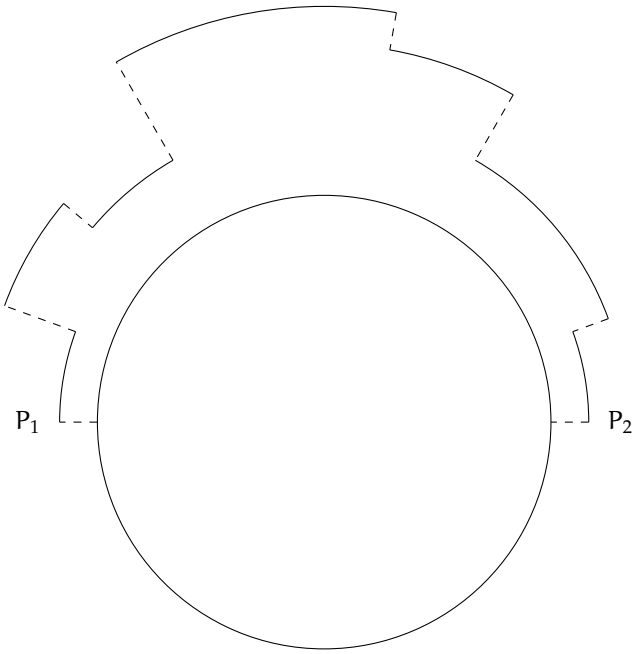

Figure 1: Simulation of the implemented distance calculation algorithm in 2D. Notice the separation of segments, the constant altitude in each of them and the irrealistic jumps of altitude between each of them.