CS-Note* 23 Tommy Ingulfsen
July 30, 2002

Pushing Tin Project - Second Design Draft

1 Introduction

This report is an algorithmic overview rather than a design. The picture to have in mind here is that of an 'X' - two crossing airways within a radar scope. There are 4 entry and exit points for aircraft, situated at each "end" of the 'X'. These are called *gates* here.

2 Overview of system

The application is divided into three distinct parts. The Generator, the Scheduler and the safety-critical ACDS (Automatic Collision Detection System). It might be that this will change in the future. For example, the Generator and the Scheduler could come together into one.

The Generator and Scheduler will be written in ordinary Ada, while SPARK will be used for the ACDS. There will obviously be a SPARK boundary between the ACDS and the Scheduler, since it is the Scheduler that interfaces with the ACDS.

The three system parts are intended to be kept as separate as possible.

Two different data structures are used to represent the planes. The Generator and the Scheduler maintains a "transponder" view, with callsign, direction and other information stored. The ACDS has a simpler view, the "raw" radar picture. This is simply an array of current aircraft co-ordinates and the previous co-ordinates.

The reason for having several views of the same situation is that the various parts have different information needs. It seems a bit chaotic to have everything in one.

- The Generator: Generates new aircraft at random times. Gives them a "random" access point into the radar scope and a flight path that is a straight line.
- The Scheduler: Uses the Generator to feed the ACDS with raw radar data for analysis.
- The ACDS: Analyses the data it gets to see if any flights are on conflicting courses. Reports back to the Scheduler the result of the processing.

3 The Generator

This relies heavily on random number generation, using the Ada.Numerics.Discrete_Random library package.

3.1 Algorithm - Generator

With probability P_n introduce a new aircraft into the system.

If a new aircraft is introduced, randomly choose an entry point within a gate, and an exit point at the diagonally opposite gate. Assign a direction to the plane based upon the entry and exit co-ordinates. Add the necessary information to the aircraft data structure.

*Notes in this series are for ϵ-baked ideas, for $1 \geq \epsilon \geq 0$. Only exceptionally should they be cited or distributed outwith the NuSPADE project (EPSRC GR/R24081).

Remove any aircraft that have passed their exit points.

For all aircraft: With probability P_a change altitude.
Update the data structure with the new co-ordinates of all aircraft.

4 The Scheduler

The Scheduler keeps track of time. It calls the Generator to get new co-ordinates of aircraft and passes them on to the ACDS for analysis. When there is a danger of collision, the Scheduler notifies the user in some way.

This subsystem has the form of a loop. It may seem to be infinite, but in practice some limitation will be set to avoid the program running forever.

4.1 Algorithm - Scheduler

begin loop

Call Generator to get new state.

Pass the new state to the ACDS for processing.

If the result indicates potential hazards, inform the application user.

Wait a few seconds.

End loop.

5 The Automatic Collision Detection System

The ACDS will be almost entirely mathematical. The theory is based upon vectors and line segments. The following assumptions are made:

1. Speed is constant.
2. Movement is linear within the next time slot.
3. All aircraft move within the next time slot.

Using the current and previous snapshots of the situation in the sky, the ACDS will attempt to determine where every aircraft is the next time. The result of that is then used to create vectors for each plane, as line segments. The shortest distance between these lines is then found. The resulting distance vector is decomposed into its horizontal and vertical components. These components are used to ascertain if a separation standard will be breached.

5.1 Algorithm - ACDS

Using the new co-ordinates and the previous ones, determine which old aircraft belongs to which new point. This is analogous to drawing a line.

With the information derived in the previous step, determine what the next picture will look like. This assumes that movement is linear.

For all aircraft
determine minimum distance to every other within next time slot

If the minimum distance is below the safe separation criteria, then flag an error. This step involves decomposing the distance into its horizontal and vertical components.

6 Comments

The algorithms are not ready to be coded yet, since they clearly need refinement. This is just a proposal. I am particularly interested in comments on the division of the program into three distinct parts and the assumptions made for the ACDS algorithm.

A clearer picture of how the system will work is emerging. I envisage that the next step will be to adjust (or discard) this proposal, and refine it towards a design that is ready to be implemented in the not too distant future.

A couple of data structures have been written, but they are obviously not finished yet. See the appendix.

A aircrafts.ads - code

```
-- File "aircrafts.ads", Data structure to hold data on an aircraft.
-- This is for use by the part of the application that deals with the
-- user interface => holds more data than the structure for the ACDS
-- (see file "acds_aircrafts.ads").

-- Author: Tommy Ingulfsen, July 2002

package Aircrafts is
    subtype String_Length is Integer range 1..6;
    subtype Max_Collisions is Integer range 1..15;
    subtype System_Size is Integer range 1..100;

    -- Callsigns for aircraft are 6-char strings
    subtype Callsign is String(String_Length);

    -- Callsign for an aircraft on collision course
    subtype On_Collision is String(String_Length);

    -- Can have up to 15 aircraft on the collision course with "this"
    type Collision_Array is array(Max_Collisions) of On_Collision;

    -- A single aircraft
    type One_Aircraft is
       record
           Call: Callsign;
           Colliding_Aircraft: Collision_Array;
           X: Integer;
           Y: Integer;
           Z: Integer;
           Delta_X: Integer;
           Delta_Y: Integer;
           Delta_Z: Integer;
           Altitude: Integer;  -- really need this?
       end record;

    -- We work with arrays of aircraft
    type Aircraft is array(System_Size) of One_Aircraft;
end Aircrafts;
```

B acds_aircrafts.ads - code

```
-- File "acds_aircrafts.ads" - data structure for holding information on
-- an aircraft. This is for use within the Automatic Collision Detection
-- System only. It therefore holds considerably less information than
-- the full Aircrafts package (See "aircrafts.ads").

-- Author: Tommy Ingulfsen, July 2002.

package ACDS_Aircrafts is

    -- Do not permit any more than 15 collisions for 1 aircraft
    subtype Max_Collisions is Integer range 1..15;

    -- Store up to 100 aircraft in the system
    subtype Max_Aircraft is Integer range 1..100;

    -- Holds the indices of other aircrafts that may collide with "this"
    type Index_Array is array(Max_Collisions) of Integer;

    -- Type for a single aircraft
    type One_Aircraft is
       record
           X: Integer;
           Y: Integer;
           Z: Integer;
           Previus_X: Integer;
           Previus_Y: Integer;
           Previus_Z: Integer;
           Colliding_Aircrafts: Index_Array;
       end record;

    -- We will work with an array of aircraft repr. system
    type Aircraft is array(Max_Aircraft) of One_Aircraft;

end ACDS_Aircrafts;
```