Air Collision Avoidance System

Abstract

This document describes the design, implementation and validity of an aircraft collision avoidance system (ACAS). ACAS is a system that that is designed to avoid mid-air collision between two aircrafts. The project provides a toy yet a complete model of the ACAS in 2D plane with various simplifying assumptions. The report discusses the problem with designing ACAS, its implementation details including assumptions, specifications, requirement satisfaction, discussion of design of different controllers and provides a proof of correctness. It uses a synchronous model as the framework for the implementation. [1]

I. Introduction

An aircraft collision avoidance system executes independently of any ground-based equipment or air-traffic control to mitigate any threat of mid-air collision.

This document is a report of a simple model of an air collision avoidance system that is designed to handle two aircrafts. The goal of the project is to ensure that an aircraft travels from source to destination without colliding with the other aircraft in its path.

Section II summarizes the problem and points out the underlying assumptions and design constraints. Section III is the core of this report. It contains most of the details required to understand the model. This section offers explanation about extra assumption that were to be made to design ACAS. It establishes the interfaces by defining its input-output specifications and the data structure used for communication between aircraft and the aircraft controller. Continuing the design further, the safety and liveliness requirements are formally laid out. The following subsection on the design and implementation of the simple remote controller and the complete remote controller provides a pseudocode of the algorithm and a brief explanation about the working of the two controllers. Finally, a proof of correctness is also presented as a subsection. It proves how the two requirements of the controller by running through different scenarios.

II. Problem Description

An aircraft controller navigates an aircraft from source to destination without collision. The controller receives the information from its own aircraft and the aircraft in its vicinity. It uses this information to navigate the aircraft. A description of the problem with underlying assumptions and design constraints are listed below:

1. The aircraft flies in a 2D plane and its source and destination are integer-values points.
2. The aircraft flies with a constant velocity of 1km/minute along either of the two dimensions.
3. The aircraft can update its direction every 1 minute.
4. The aircraft flies with a constant velocity of 1km/minute along either of the two dimensions.
5. The controller can exchange message with another aircraft when it is in its vicinity. The vicinity is a 2km distance of aircraft from each of its four sides. It can make decisions based on the messages received from the other flights.
6. The collision region is 0.5 km from every side of the aircraft. Presence of aircrafts in the same region is considered a collision.

The problem is to ensure that no two aircrafts should be in that region.

III. Details

A. Assumptions

Most of the constraints are mentioned in the problem description section. However, there are assumptions that are made while implementing the system as well:

1. Since we are dealing with integer values rather than taking 0.5 km on all sides of the aircraft the meaning of collision for implementation of the given design is having the same coordinates at the same time.
2. The problem mentions that the sources of the aircraft cannot be the same. This can be expressed as the sources of two aircrafts should be at least a unit distance.
3. The program equates execution of each loop as equivalent to the given rate of update.
4. Each unit distance in the problem refers to a km of the original design requirement.

B. Input-Output Specifications

- Inputs

The aircraft controller accepts the following inputs from its own and the other aircraft:

- Current position of own aircraft (x1, y1)
- Destination of own aircraft (dx1, dy1)
- Current direction of own aircraft (direction1)
- Current position of own aircraft (x1, y1)
- Destination of own aircraft (dx1, dy1)
- Current direction of own aircraft (direction2)

All of them are of integer types. 0, 90, 180 and 270 are the integers used to represent east, north, west and south respectively.

- Outputs

The controller provides an output of a string type to indicate whether the aircraft should move straight, left or right. “L”, “R” and “S” indicate left, right and straight respectively.

The output of the controller is the input to the aircraft and vice-versa

C. Requirements

- Safety Requirements

The safety requirements of a system ascertain that nothing bad ever happens. In the design of aircraft collision avoidance system, the safety requirement is that for any given execution at any given round the coordinates of the two aircrafts will never be the same i.e.

$$x1 \neq x2 \wedge y1 \neq y2$$

where x1 and y1 are the coordinates of aircraft1 and x2 and y2 are coordinates of aircraft2.

- Liveness Requirements

The liveness requirements in a system ascertain that something good eventually happens. In the design of ACAS, the arrival of aircraft eventually to its destination is that. The liveness requirement can be expressed as

$$\Diamond(x1 = dx1) \wedge \Diamond\ (y1 = dy1)$$

The means that eventually the x coordinate of the source and the y coordinate of the destination would be eventually equal.

D. Design of a simple controller

A simple controller is responsible for guiding the aircraft from the source to the destination in the shortest possible way. It assumes that there will be no obstacles in the aircraft’s way to the destination. The controller sends the messages to the aircraft form of left, right or straight commands.

The working of the controller is as follows. The controller turns the aircraft initially until it is aligned to one of the two dimensions of the destination. Once, that happens it continues to move forward until it meets one coordinate i.e. the aircraft keeps moving until either its x or y coordinate becomes equal to the x or y coordinate of the destination. From there on, based on the destination’s direction it turns left or right and continues its journey until both the coordinates of the aircraft are equal to the destination coordinates.

A pseudo code of the algorithm looks like the one on the right-hand side of the page in Fig.1

E. Design of a complete controller

The responsibility of the complete controller is to guide the aircraft in the presence of other aircraft from source to destination in the shortest possible way. The complete controller incorporates the simple controller’s algorithm to make predictions about the next location of aircrafts based on their current position. The controller turns the aircrafts in the directions such that they are aligned to their destination. The design of the complete controller has two aspects.

The first one is the driver of controller which is essentially keeping a global view of the two aircrafts. It checks if the aircrafts are in vicinity and drives the system. In a real world, this would be equivalent to a high frequency radio which would cause controller to receive signals when the other aircraft is in vicinity. The driver is also responsible for fetching expected commands and then sending them to the respective aircrafts. For the two aircraft problem, in situations where next step of both aircrafts can cause a collision then for conflict resolution an aircraft with a higher

```
while aircraft has not reached destination do
    get current location coordinates;
    get destination coordinates;
    get current direction of the aircraft;
    if direction is east then
        if src x < dest x then
            move forward;
        else if src y ≤ dest y then
            turn left;
        else
            turn right;
    else if direction is west then
        if src x > dest x then
            move forward;
        else if src y ≤ dest y then
            turn right;
        else
            turn left;
    else if direction is north then
        if src y < dest y then
            move forward;
        else if src x ≤ dest x then
            turn right;
        else
            turn left;
    else if direction is south then
        if src y > dest y then
            move forward;
        else if src x ≤ dest x then
            turn left;
        else
            turn right;
end
```

Fig 1: Simple controller algorithm

priority becomes the decision-maker whereas the other aircraft is rotated in its position. The choice of

```
if If the other aircraft is in vicinity then
    get aircraft1's current location coordinates;
    get aircraft2's current location coordinates;
    get aircraft1's next location coordinates;
    get aircraft2's next location coordinates;
    if will collide then
        if this controller's aircraft has priority then
            if anticipated position of this aircraft and current position of
             the other aircraft are same then
                if resolution flag is activated then
                    move forward;
                else if If y coordinates of both aircrafts are same then
                    set resolution flag;
                    if y coordinate of this aircraft < y coordinate of the
                     destination of the other aircraft then
                        turn right;
                    else
                        turn left;
                else if If x coordinates of both aircrafts are same then
                    set resolution flag;
                    if x coordinate of this aircraft < x coordinate of the
                     destination of the other aircraft then
                        turn left;
                    else
                        turn right;
            else
                Execute the simple controller command;
        else if If y coordinates of both aircrafts are same then
            if y coordinate of this aircraft < y coordinate of the
             destination of the other aircraft then
                turn right;
            else
                turn left;
        else if x coordinates of both aircrafts are same then
            if x coordinate of this aircraft < x coordinate of the
             destination of the other aircraft then
                turn left;
            else
                turn right;
    else
        Execute the simple controller command;
```

priority is random and is handled by the driver. From the pseudocode, it maintains various checks depending upon the current state of the system. The following is a pseudo code of the driver of the complete controller.

```
while both aircrafts have not reached destination do
    get aircraft1's current location coordinates;
    get aircraft2's current location coordinates;
    if in vicinity and no aircraft has reached destination then
        get command for aircraft1 when aircraft2 is in vicinity;
        get command for aircraft2 when aircraft1 is in vicinity;
        send command to aircraft1;
        send command to aircraft2;
    else if aircraft1 has reached destination then
        get command for aircraft2 when aircraft1 is not present;
        send command to aircraft2;
    else if aircraft2 has reached destination then
        get command for aircraft1 when aircraft2 is not present;
        send command to aircraft1;
    else
        get command for aircraft1 when aircraft2 is in vicinity;
        get command for aircraft2 when aircraft1 is in vicinity;
        send command to aircraft1;
        send command to aircraft2;
end
```

Fig 2: Pseudo code of driver of aircraft controller

The actual command functionality of the controller is what determines the controller's output. The controller behaves like a simple controller when aircrafts are not present in each other's vicinity. The command functionality is activated in the presence of aircrafts within a distance of 2 units. With regards to the working of the controller there are three cases that are identified during a controller's execution.

F. Proof of Correctness

The proof of correctness requires that the controller should ensure that the aircraft eventually reaches its destination and that the aircrafts will never collide. If we can prove that the aircrafts will never collide, and that the simple controller always takes the aircraft to its destination then it is implied that the complete controller will also satisfy both the requirements. Let us go through the simple remote controller first. Looking at the pseudo code of the simple controller, a loop runs continuously until the aircraft has reached the destination. The loop represents each round of execution. If the aircraft is set in a direction, then the controller checks if moving forward in that direction would result in moving towards the destination. If not, then it rotates it such that it is turned towards the destination. Once the rotation happens then the aircraft reaches the destination as shown in the design of simple controller's section. Since it is very straightforward, with not complexities the liveness requirement is always satisfied here.

Now for collision, let us consider different scenarios and organize them in the order of conflict and resolution.

1) Case I

- Conflict

  In the first case, there can be a situation when the two aircrafts are at a distance of 1 unit (1km) facing each other. The anticipated next locations of both the aircrafts are the current position of each other.

- Resolution

  In this case, the prioritized aircraft turns in the direction opposite to the direction of the other aircraft's direction, so it moves farther away from the trajectory of the other aircraft. While the other aircraft also turns left or right decided based on how it can be farther from the destination of the other aircraft. In the next step, the prioritized moves forward in the turned direction whereas the non-prioritized aircrafts turn back to the initial direction it was in when the conflict happened. Now the two aircrafts are two units of Euclidean distance away from each other. Again, in the next step the two aircrafts are separated by at least a unit distance. They move forward in their trajectory and do not collide.

2) Case2

- Conflict

  The next step of both aircrafts will cause a collision they are not currently facing each other, and one aircraft's current coordinates are not equal to the other's next coordinates. This

implies that they are perpendicular to each other.

- Resolution
  When this conflict happens, the prioritized aircraft takes its optimal path and moves into the position where the conflict was supposed to happen. The other aircraft rotates in the

Fig 3. The actual command pseudocode

direction which is farther from the prioritized aircraft's destination thus moving out of its trajectory.

3) Case 3

- Conflict
  When the destination of both the aircrafts are same in the next step.
- Resolution
  In this case, again the prioritized aircraft makes its optimal move. Since it has already reached its destination the other aircraft is managed by the driver to choose the optimal path and thus making it reach its destination.

The safety requirement is also ensured by a safety monitor which goes into an error mode if ever the collision happens. In other cases, the controller uses simple remote algorithm which has already been proven to satisfy the liveness and the safety requirement.

IV. Conclusion

The implementation is an instance of the synchronous model in that the components like aircraft and the aircraft controller have inputs and outputs across sequence of rounds and they execute reactions through each of them. The complete controller satisfies the requirements of the system. The implementation fixes priority beforehand based on random choice. It would be interesting to see if the priority could be changed dynamically and if the system would still function correctly since the basis of conflict resolution is priority. Also, halting the non-prioritized aircraft is an effective but not an optimal solution. Tweaking the controller's algorithm, a little bit can reduce the number of steps required for non-prioritized aircraft to reach its destination.

The report's organization is partly inspired by Ankit Mishra and Nikhilesh Behera's work.[2]

The entire project with report and code is available at https://github.com/SKShah36/ACAS

References

[1] R. Alur, *Principles of Cyber-Physical Systems*. The MIT Press, 2015.

[2] A. Mishra and N. Behera, "AircraftController/CIS-540-Project-1.pdf at master · mishra14/AircraftController." [Online]. Available: https://github.com/mishra14/AircraftController/blob/master/CIS-540-Project-1.pdf. [Accessed: 12-Dec-2019].