Aircraft Collision Avoidance Controller Design Based on Matlab

Manyao Peng
Computer Science Department, Vanderbilt University, Nashville, TN, 37203 USA
manyao.peng@vanderbilt.edu

Abstract—**The problem is that when there are two aircrafts in the airspace, they are very possible to collide with each other, so naturally we are thinking about a controller to avoid the collision. The goal of this project is to design an aircraft controller that navigates the aircraft from source to destination in a 2-D plane while ensuring that it does not collide with other aircraft in its path. Followed from this report, the three aircrafts problem is more complex. Software used is MATLAB.**

I. Introduction

This report demonstrated a solution of how to design a controller, which has same algorithm for both aircraft, that ensures each aircraft reaches its destination in as small as time as possible while ensuring collision avoidance. In this problem, the sources and destinations are provided as parameters and the designed controller should work with all such source-destination pairs[1].

II. Input/Output

The controller gets information about the current location and the target location of the aircraft. Further, it also receives messages from aircraft in its vicinity and uses this information to navigate the aircraft and avoid collision with other aircraft. Data structure containing the above-mentioned information is a variable called 'in' with the following fields in it

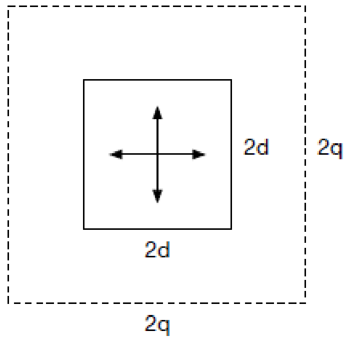
- (x,y) - They denote the current position of the aircraft.
- (xd,yd) - They denote the destination position of the current aircraft.
- Theta - It denotes the direction of movement of the aircraft.
- m – Incoming message from the other aircraft which contains (x,y,xd,yd,theta) of the other aircraft in its vicinity.

The controller generates an output value which indicates if the aircraft should move forward, move right or move left.

III. Assumptions

Here clarify the backgrounds and assumptions of the problem:

1. The aircraft can fly in a 2-D plane. Its source and destination are integer-valued points in the plane.
2. The aircraft flies with a constant velocity v = 1 km/min along either X-axis or Y-axis.
3. The aircraft controller can update direction of flight every k = 1 min. Note that, it can decide to either fly straight or rotate left or right by 90 degrees but not turn back.
4. At the beginning of every k = 1 minutes
   - The controller can exchange messages with any aircraft in a square region with side length 2q = 4 km (communication zone) in the vicinity of the aircraft as shown in figure below
   - Based on the messages received and the current state of the aircraft, the controller can update the direction of flight.



Figure(1)

5. Collision Avoidance: Each aircraft has a square region with side length 2d = 1 km (danger zone) around it such that no aircraft should enter this region at any time during the flight as shown in figure(1) above.
6. The source locations are distinct locations for different aircraft.
7. Once an aircraft reaches its destination, it is no longer a threat and collision avoidance is not required for this aircraft. Also, the aircraft stops sending messages to the other aircraft in its vicinity.
8. Further, we assume that the airspace consists only of two aircraft that start at time t = 0.

IV. Requirements

The controller should guide the aircraft from source to destination with the following 2 requirements:

Liveness requirement: Controller should guide the aircraft from source to destination eventually.

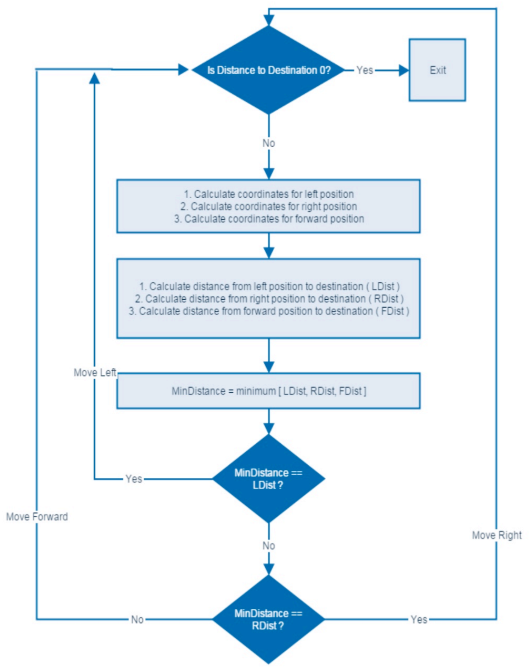
- At all times, the controller should bring the aircraft closer to the destination using the shortest path algorithm implemented.
- Once the aircraft reaches its destination the controller should not move the aircraft.

Safety requirement: The controller should avoid collisions between the aircraft along the whole process.

- The controller should check for incoming messages from the other aircraft when it is in the communication zone.
- The controller should read the incoming messages to predict the next move of the other aircraft.
- The controller should avoid guiding the aircraft to the predicted position of the other aircraft or their vicinity actively.
- At any step there must exist one valid move for both the aircraft based on our collision avoidance algorithm.

V. Design of Simple Controller

The simple controller guides the aircraft from source to destination as quickly as possible. To ensure this we use the shortest path algorithm, that is, in every step, we calculate the next position for left, right, forward direction, and then calculate their distance to the destination, then we sort the three paths and choose the minimum one as the next move. [2]


Is Distance to Destination 0?
Yes
Exit
No
1. Calculate coordinates for left position
2. Calculate coordinates for right position
3. Calculate coordinates for forward position
1. Calculate distance from left position to destination (LDist)
2. Calculate distance from right position to destination (RDist)
3. Calculate distance from forward position to destination (FDist)
MinDistance = minimum [LDist, RDist, FDist]
Move Left
MinDistance == LDist ?
Yes
No
Move Forward
Move Right
MinDistance == RDist ?
No
Yes

Figure(2)

The following code shows the process of calculate the distance from left, right and straight position to the destination.

```
if( in.theta == 0 || in.theta == 360)
        xF = in.x+1;
        yF = in.y;
        xR = in.x;
        yR = in.y-1;
        xL = in.x;
        yL = in.y+1;
    elseif(in.theta == 90)
        xF = in.x;
        yF = in.y+1;
        xR = in.x+1;
        yR = in.y;
        xL = in.x-1;
        yL = in.y;
    elseif(in.theta == 180)
        xF = in.x-1;
        yF = in.y;
        xR = in.x;
        yR = in.y+1;
        xL = in.x;
        yL = in.y-1;
    elseif(in.theta == 270)
        xF = in.x;
        yF = in.y-1;
        xR = in.x-1;
        yR = in.y;
        xL = in.x+1;
        yL = in.y;
    end

    currentDistanceF = abs(in.xd-xF)+abs(in.yd-yF);
    currentDistanceR = abs(in.xd-xR)+abs(in.yd-yR);
    currentDistanceL = abs(in.xd-xL)+abs(in.yd-yL);
```

Further, I used a 2-d array with dimensions 3x2 to store and sort these distances. Here the first column contains the distance and the second column has the value which tells whether that distance is left, right or straight distance.
1 - denotes left
-1 - denotes right
0 - denotes straight

The code used to initialize the matrix is

```
    matrix1(1,1) = currentDistanceL;
    matrix1(2,1) = currentDistanceR;
    matrix1(3,1) = currentDistanceF;

    matrix1(1,2) = 1;
    matrix1(2,2) = -1;
    matrix1(3,2) = 0;
```

So, the matrix looks like this initially when it is not sorted

Left distance	1
Right distance	-1
Forward distance	0

The following code sorts the matrix with an increasing order of the distance

```
for i=1:3
  for j=(i+1):3
     if ( matrix1(i,1) >  matrix1(j,1))
        temp=matrix1(i,1);
        matrix1(i,1)=matrix1(j,1);
        matrix1(j,1) = temp;

        temp=matrix1(i,2);
        matrix1(i,2)=matrix1(j,2);
        matrix1(j,2) = temp;
     end
  end
end
```

So, after sorting, we can get the minimum distance from matrix (1,1), and the corresponding value of direction is matrix (1,2). So, we can just return the matrix (1,2) as the navigation of next move in this case.

VI. Complete Controller with collision avoidance

The complete controller has to guide the aircraft from source to destination using the shortest path algorithm without any collision.

This solution gives higher priority to aircraft 1 if the first preferences of both the aircraft result in a collision. For example, for aircraft-1 taking a left is the shortest path and after taking left assume it would reach position A. for aircraft-2 taking a right is the shortest path and assume it would come to position A as well. In this case, the algorithm assigns priority to aircraft 1, so aircraft 1 still goes on the first choice which is taking a right turn while aircraft 2 has to change its choice to the second shortest path.

This idea is implemented by using a persistent variable 'count'. When it is the first time, it is initialized to 1. And after that, every time we execute the controller function, it will increment by one. So, it means that when count is odd, it is the aircraft 1, and we just go to the simple controller process, and when the count is even but not in the communication zone, the aircraft 2 goes on with the simple controller process as well. But if the count is even and there is a message, then we need to make sure there's no collision next step.

The following is code in the controller.m:

```
persistent count;

if isempty(count)
  count = 1;
end

out.val = getNextMove(in, in.m, count);
count = count + 1 ;
state.mode = out.val;
```

Since we know that aircraft 1 is always given its first choice, we only need to change the course of aircraft 2 whenever it is in vicinity of aircraft 1. This is facilitated by the following code, where we check if message is not empty and if aircraft_no is even.

```
if( ~isempty(message) && mod(aircraft_no,2)==0)
        if( message.theta == 0 || message.theta == 360)
          xF = message.x+1;
          yF = message.y;
          xR = message.x;
          yR = message.y-1;
          xL = message.x;
          yL = message.y+1;
        elseif(message.theta == 90)
          xF = message.x;
          yF = message.y+1;
          xR = message.x+1;
          yR = message.y;
          xL = message.x-1;
          yL = message.y;
        elseif(message.theta == 180)
          xF = message.x-1;
          yF = message.y;
          xR = message.x;
          yR = message.y+1;
          xL = message.x;
          yL = message.y-1;
        elseif(message.theta == 270)
          xF = message.x;
          yF = message.y-1;
          xR = message.x-1;
          yR = message.y;
          xL = message.x+1;
          yL = message.y;

      end
    otherDistanceF = abs(message.xd - xF)+abs(message.yd - yF);
    otherDistanceR = abs(message.xd - xR)+abs(message.yd - yR);
    otherDistanceL = abs(message.xd - xL)+abs(message.yd - yL);
```

As shown in the simple controller, we go ahead and make a 2-array called matrix2 which contains the sorted distances to the destination of aircraft 2.

```
    matrix2(1,1) = otherDistanceL;
    matrix2(2,1) = otherDistanceR;
    matrix2(3,1) = otherDistanceF;


    matrix2(1,2) = 1 ;
    matrix2(2,2) = -1 ;
    matrix2(3,2) = 0;

        for i=1:3
          for j=(i+1):3
            if ( matrix2(i,1) > matrix2(j,1))
              temp=matrix2(i,1);
              matrix2(i,1)=matrix2(j,1);
              matrix2(j,1) = temp;

              temp=matrix2(i,2);
              matrix2(i,2)=matrix2(j,2);
              matrix2(j,2) = temp;
            end
          end
        end
```

Now we have the sorted distances in the array matrix2. Further, we go ahead and check if the "to be taken first choice" of aircraft 2 collides with the "already taken first choice" aircraft 1.

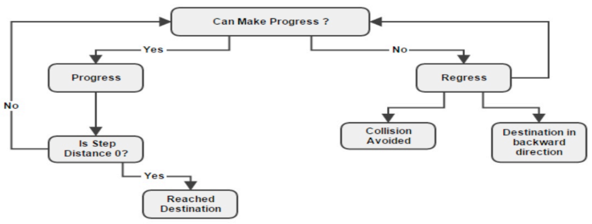
This is facilitated by the following code

```
for i=1:3

  if(NoCollision(matrix1(i,2), in, matrix2(i,2), in.m))
            dir_value = matrix1(i,2);
            break;
        end
 end
```

So, in the above code we make a call to a new function "NoCollision.m" which basically loops through the array and returns an option for aircraft 2 that does not collide with the aircraft 1. This decision is then returned to the controller which guides the planes left, right, or straight accordingly.

VII. Proof Of Correctness



Figure(3)

This part shows the proof of correctness by addressing the requirements of Liveness and Safety.

1. Liveness
   - At each step both the aircraft make progress towards their respective destinations. This is ensured by the shortest-path algorithm explained in the simple controller part. In each call to controller this algorithm returns a direction for the aircraft to take.
   - The aircraft never enter a state of deadlock where they move around in circles. This is ensured because I have given priority to aircraft 1 in case when the shortest path of both the aircraft causes collision. So, this makes sure that aircraft 1 takes its shortest path while aircraft 2 alters is path to the next shortest path. This way the algorithm is also optimized because both the aircraft do not avoid the shortest path, only aircraft 2 changes its path to the next shortest one. So, this is the proof of correctness for liveness.

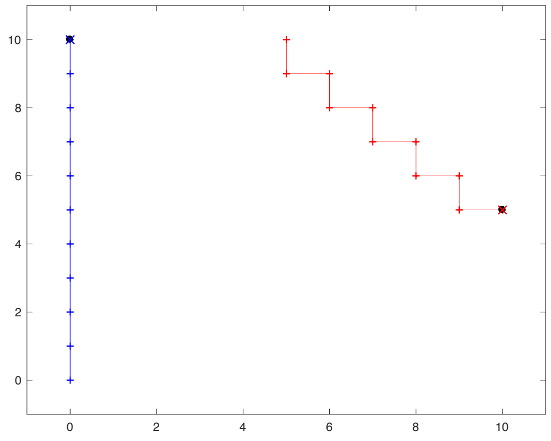
2. Safety
   - At each step if there are incoming messages from the aircraft in vicinity we check it in our code to avoid collision. In case of aircraft 2 we go and check if the shortest path for aircraft 2 collides with the shortest path of aircraft 1, and since I have given priority to aircraft 1, aircraft 2 alters its path thus avoiding collision. So, this is the proof of correctness for safety.

VIII. Results

The following are some results of some test cases from matlab simulation.

1. runSimulation([0, 0], [0, 10], [5, 10], [10, 5], 40)

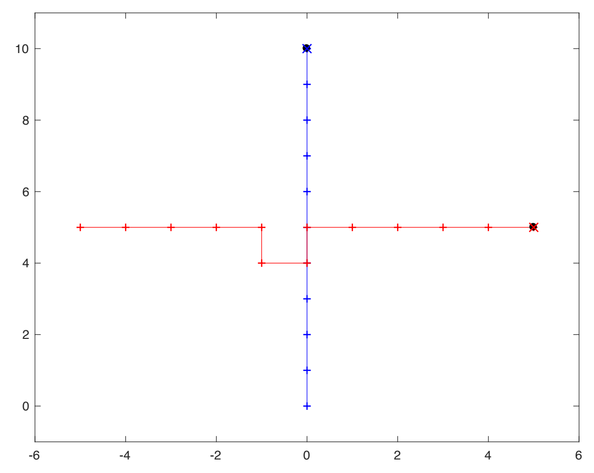
   Both of them reach destination without cross.



Figure(4)

2. runSimulation([0, 0], [0, 10], [-5, 5], [5, 5], 40)

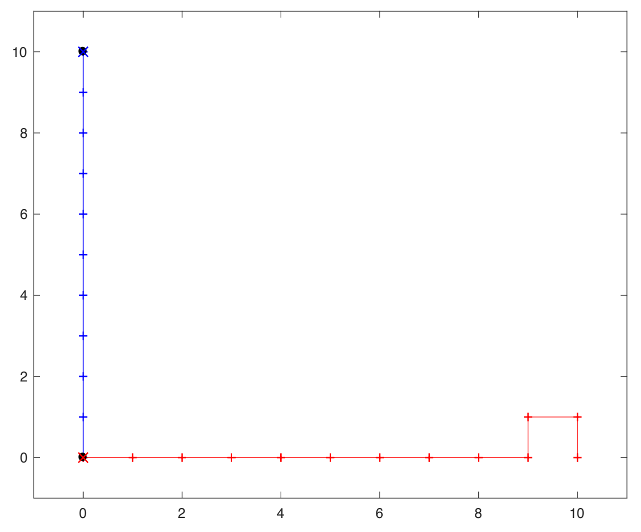
   From the figure below, we can see that since aircraft 1(blue line) has the highest priority it follows its shortest path while aircraft 2 (red line) changes its path at (-1,5) to go to (-1,4) instead of (0,5) and thus avoids collision.



Figure(5)

3. runSimulation([0, 0], [0, 10], [10, 0], [0, 0], 40)

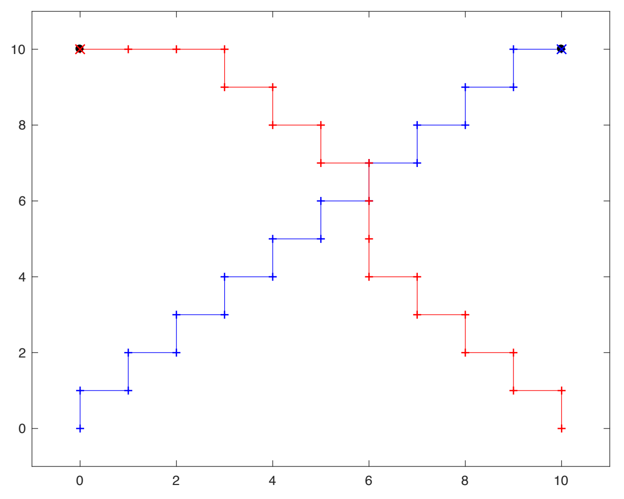
   There is no collision avoidance as their paths do not cross, so both reach destination by quickest path. Since initial theta is 0, which is along the x+ direction, so for aircraft, it need to change its direction to the destination by turn left, left, left, right, because the aircraft can't turn back.



Figure(6)

4. runSimulation([0, 0], [10, 10], [10, 0], [0, 10], 40)

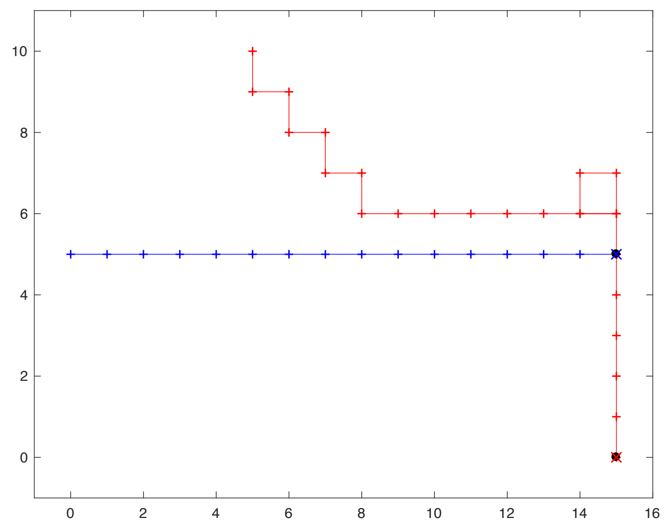
   The figure shows that aircraft 1 (blue line) follows its shortest path since it has higher priority while aircraft-2(red line) changes its path at (6,5) to go to (6,6) instead of (5,5) and thus avoids collision.



Figure(7)

5. runSimulation([0, 5], [15, 5], [5, 10], [15, 0], 40)

   The figure below shows that since aircraft 1(blue line) has the highest priority it follows its shortest path while aircraft 2 (red line) changes its path at (15,6) to go to (15,7) instead of (15,5) and thus avoids collision.



Figure(8)

The following are the results of the 3 different test benches. You can find the instruction of running test cases and test bench from readme.txt.

```
>> [numPassed, numFailed, log] = testbench(3, 100)

numPassed = 22174
numFailed = 0
log = []

>> [numPassed, numFailed, log] = testbench(4, 40)

numPassed = 116859
numFailed = 0
log = []
```

```
>> [numPassed, numFailed, log] = testbench(3, 25
numPassed = 22174
numFailed = 0
log = []
```

REFERENCES

[1] https://brightspace.vanderbilt.edu/d2l/le/content/16188/viewContent/567807/View

[2] https://github.com/mishra14/AircraftController