Free Flight in aircraft with the use of neural networks and non-linear programming

by

Chrisavgi Kontogeorgou

Diploma Thesis

Presented to the Department of Electronic Engineering of
Technical University of Crete, GREECE
in Partial Fulfilment
of the Requirements
for the Degree of

Diploma in Electronic and Computer Engineering

Technical University of Crete, Hellas(Greece)

January 2008

Abstract

In recent years there has been a great effort to convert the existing Air Traffic Control system to a novel system known as Free Flight. Free Flight is based on the concept that increasing international airspace capacity will grant more freedom to individual pilots during the enroute flight phase, thereby giving them the opportunity to alter flight paths in real time. Under the current system pilots must request, then receive permission from air traffic controllers to alter flight paths. Understandably the new system allows pilots to gain the upper hand in air traffic. At the same time, however, this freedom increase pilot responsibility. Pilots face a new challenge in avoiding the traffic shares congested air space. In order to ensure safety, an accurate system, able to predict and prevent conflict among aircrafts is essential. There are certain flight maneuvers that exist in order to prevent flight disturbances or collision and these are graded in the following categories : vertical, lateral and airspeed. This work focuses on airspeed maneuvers and tries to introduce a new idea for the control of Free Flight.

.

Contents

1 Introduction **7**

1.1 Select the most appropriate system for conflict avoidance in Free Flight [1] 7

1.2 The purpose of this diploma thesis 8

1.3 Structure of thesis 8

2 Problem Approach **9**

2.1 Basic idea of scenario 9

2.1.1 Requirements for aircraft tracking in the sphere 11

2.1.2 Equations for collision scenarios 11

2.1.3 Collection Of Collision Data 14

3 Non-Linear programming **17**

3.1 General about non-linear programming 17

3.1.1 GAMS software package 19

3.1.2 Problem Example 20

3.1.3 Optimum Velocity Changes with the help of Gams . . 21

3.1.4 Cases with 3 aircraft 36

3.1.5 Recommended Solutions for unsolvable cases with the presented algorithm (Which account for 0.3% of the total number of cases in our experiment). 38

3.1.6 Comparison with another algorithm 38

4 Neural Networks **47**

4.1 Fundamentals of Neural Network 47

4.1.1 Connections between units 48

4.1.2 Activation and output rules 49

4.1.3 Network topologies 50

4.1.4 Training of artificial neural networks 51

4.1.5 Terminology 51

4.1.6 Networks with threshold activation function 52

4.1.7 Perceptron learning rule and convergence theorem . . . 54
4.1.8 Networks with linear activation functions:The delta rule 55
4.1.9 Back Propagation 56
4.1.10 Multi layer feed-forward networks 57
4.1.11 Back propagation algorithm 58
4.2 Conflict avoidance with neural networks 58
4.2.1 Construction of the neural network 61
4.2.2 Evaluation of the neural network 63
4.2.3 Neural networks vs Non-Linear programming 65
4.3 Appendix A-Codes 67
4.3.1 Matlab Code-Representation of the problem 67
4.3.2 Gams-Conflict resolution 74
4.3.3 Matlab Code Neural Network 80
4.3.4 Side programs 82

List of Figures

2.1 The control sphere. 10
2.2 P1 (P3) is the insertion point, P2 (P4) the escape point in the sphere of the first (second) aircraft. 11
2.3 The angles that determine the position of the aircraft in the 3D space. 12
2.4 The control sphere created by MATLAB code. 15

3.1 Position of each aircraft at every time slice 23
3.2 Distances between aircraft position 24
3.3 With the velocity changes the aircraft pass through the crucial point at different time slices 25
3.4 Case1 Example1 28
3.5 Case2 Example1 29
3.6 Case2 Example2 30
3.7 Case3 Example1 31
3.8 Case3 Example1 after conflict resolution 31
3.9 Case3 Example2 32
3.10 Case3 Example2 after conflict resolution 33
3.11 Case3 Example2 34
3.12 Case3 Example3 35
3.13 Case3 Example3 after conflict resolution 35
3.14 Conflict Example with three aeroplanes 37
3.15 Conflict Resolution with three aeroplanes 37
3.16 Cone sections between two moving spheres 39
3.17 The two non parallel straight lines tangent to the safety discs of radius d/2 for two aircraft 39
3.18 Four airplanes and one conflict. 41
3.19 Four airplanes after conflict avoidance. 41
3.20 Five airplanes and one conflict. 42
3.21 Five airplanes after the conflict avoidance. 43
3.22 Five airplanes and two conflicts. 43
3.23 Five airplanes after the conflict avoidance. 44

3.24 Six airplanes and two conflicts. 44
3.25 Six airplanes after the conflict avoidance. 45

4.1 The basic components of neural network. 48
4.2 Various activation function for a unit. 50
4.3 Single layer network with one output and two inputs. 52
4.4 Geometric representation of the discriminant function and the weights. 54
4.5 The percepton 55
4.6 A multi-layer network with l layers of units. 57
4.7 Feed Forward Back Error Propagation Neural Network with two hidden layers.The first hidden layer has 50 neuron while the second 40 neurons. 59
4.8 Tan-Sigmoid Transfer Function 61
4.9 Tan-Sigmoid Transfer Function (Wp+b) 61
4.10 Optimum Velocity changes and failure 64
4.11 Error through training in function with the epochs 66

Chapter 1

Introduction

1.1 Select the most appropriate system for conflict avoidance in Free Flight [1]

Dealing with aircraft means dealing with real time systems. An accurate real time system requires the minimum possible response time so that the system functions well.

In dealing with this specific problem we can focus on two basic areas in the creation of an accurate control system based on airspeed maneuvers: **optimum (minimum) velocity changes** [2] and **quick response** of the system. Initiating slight changes in velocity requires a minimum time to change the velocity. Furthermore, having an immediate response by the system, means a quick calculation of the changes. If those calculation times are minimum then we also minimize the total process time which is a basic principle in real time systems.

Considering the importance of response time in relation to the prevention of air traffic conflict we conclude that the most appropriate way to calculate new velocities is to use a neural network, knowing that an already trained neural network has very short response time.

Neural networks [3] can be characterized as computational models with particular properties such as the ability to adapt or learn, to generalize, as well as to cluster or organize data, and whose operation is based on parallel processing. The intriguing question however, is to what extent does the neural approach prove to be better suited to certain applications rather than other models.

1.2 The purpose of this diploma thesis

The goal of this work is the creation of a neural network that can predict the optimal velocity change of two aircraft in order to avoid an imminent conflict.

First and foremost, collision cases were gathered for the creation of the specific neural network. This task was managed with Matlab code which created random flights with true velocities. In this way, a great number of cases are gathered but many of those cases were relevant to the problem, so the method of sample deletion is used. Sample ejection omits cases that are irrelevant to a problem.

The next step is the creation of a nonlinear program in GAMS [4] tool used to define the new velocities of each aircraft in order to avoid conflict. The function of this program is to find the minimum velocity changes. Furthermore, it is important to mention that for each conflict case there is a unique file in GAMS [4], giving solution to the problem as well as a specific file keeping the velocity changes. Because of the great amount of data two side programs in C++ were used in order to edit these files (GAMS, velocity) in a few seconds while manually it would have taken days for this work.

The final step is the collection of data to train the neural network [3]. Thus it was decided that the inputs of the neural network, known as training set, would consist of initial and final positions in the control sphere, as well as initial velocities. The desired output, known as the target set, of the neural network would be the velocity changes [2].

1.3 Structure of thesis

After the introduction part the four chapters follow:

Chapter2 : Approach of the problem
Chapter3 : Non-linear programming
Chapter4 : Neural Networks
Chapter5 : Appendix A

Chapter 2

Problem Approach

Free Flight is the new air traffic system that is meant to replace the existing national airspace system. It is a system that gives the pilot freedom to change a flight path in real time although burdening him with the risk of safety. In Free Flight many aircraft might share the same airspace as they do not follow specific flight paths as in the current system. Thus the new system requires the pilot to ensure a safe flight [5]. Earlier efforts have been made in order to define the appropriate control system in Free Flight. Collision avoidance and more specifically the maneuvers that are used, can be grouped into the following categories: vertical, lateral and airspeed. The primary work is focused on conflict detection and resolution in Free Flight domain by using airspeed maneuver [6].

2.1 Basic idea of scenario

Matlab code is used, in order to create the collision cases. First of all, a true scenario is represented for the Free Flight problem. There is a sphere with a radius equal to 108 kilometers, termed here as the control zone [7].

The object of our study is the case of two aircraft flying in this sphere. Each airplane has random initial and final configuration points in the control sphere. We consider the traffic of the two aircraft linear with stable velocities in order to simplify the problem. When the two objects are closer than 9 kilometers we consider that a conflict happens.

Our study involves collision cases while, non-crash cases are ignored. The conflict scenarios are created as follows:

First of all, we create a sphere with radius of 108 kilometers and two aircraft randomly distributed in it. The first aircraft (as well as the second) has random initial points $P_{aircraft1}(x_i, y_i, z_i)$ and final points $P_{aircraft1}(x_f, y_f, z_f)$

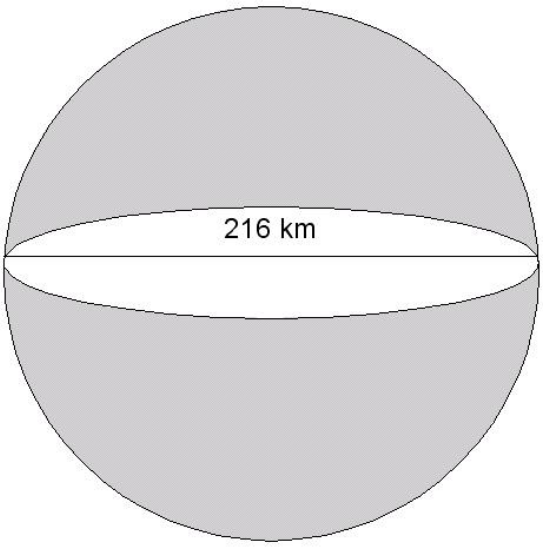

Figure 2.1: The control sphere.

in the sphere.

Furthermore, to determine each aircraft in a unique way, we use the two spherical angles , θ, ϕ. Consider that for the two dimension space we need only one angle to define the direction of an object. Thus for the 3 dimension space we need 2 angles. Where

$$-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2} \tag{2.1}$$

and

$$0 \leq \phi < 2\pi \tag{2.2}$$

To resume, we have the position of an object totally determined by the point:

$$P_{aircraft1}(x_i, y_i, z_i, \theta_i, \phi_i) \tag{2.3}$$

We want to find a relationship between the two objects so we consider the route as a function of time. Thus the configuration of an aircraft at time t is:

$$P_{aircraft1}(x_t, y_t, z_t, \theta_t, \phi_t) \tag{2.4}$$

To test whether the objects collide, we check the distance between them during a specific period of time.

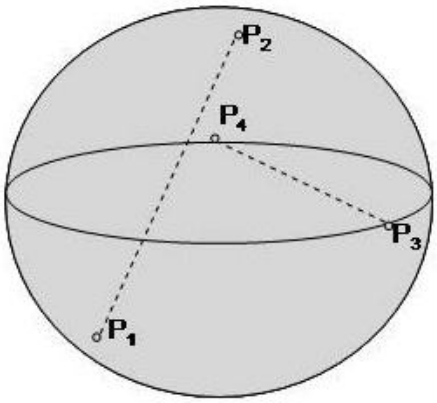

Figure 2.2: P1 (P3) is the insertion point, P2 (P4) the escape point in the sphere of the first (second) aircraft.

2.1.1 Requirements for aircraft tracking in the sphere

The question that arises from the above scenario is how the system will track the aircraft and find their position in the control sphere. The answer is simple; it is the same as in the current system of tracking, via radar [8]. To begin with, there are two categories of radar: the embedded radar [9] in each aircraft and the ground radar [10] that is located mainly near airports. Civil aircraft have radar that can cover distances between 3 km and 20 km. There are also some categories of fighter aircraft that have radar able to cover distances in the range of 100 km. Furthermore, ground radar can cover distances greater than 600 km. In the statement of our problem, we have a control sphere with a radius of 108 km. The worst case scenario concerning the greatest distance two aircraft may have is 216 km, equal to the length of the diameter of the sphere, so we need radar that can cover this distance.

2.1.2 Equations for collision scenarios

The equations [11] expressing the move from the initial point to the final are:

$$\begin{cases} x = x_1 + a_1\lambda \\ y = y_1 + a_2\lambda \\ z = z_1 + a_3\lambda \end{cases} \tag{2.5}$$

Where

$$a = (a_1, a_2, a_3) = (x_2 - x_1, y_2 - y_1, z_2 - z_1) \tag{2.6}$$

and λ is a variable.

Having random values for the two input angles ϕ_i and θ_i and for the output angles ϕ_f and θ_f in the sphere, we create the initial and final points as follows:

$$x = Rsin(\phi)cos(\theta) \tag{2.7}$$

$$y = Rsin(\phi)sin(\theta) \tag{2.8}$$

$$z = cos(\phi) \tag{2.9}$$

Where θ, ϕ angles are shown in the figure below:

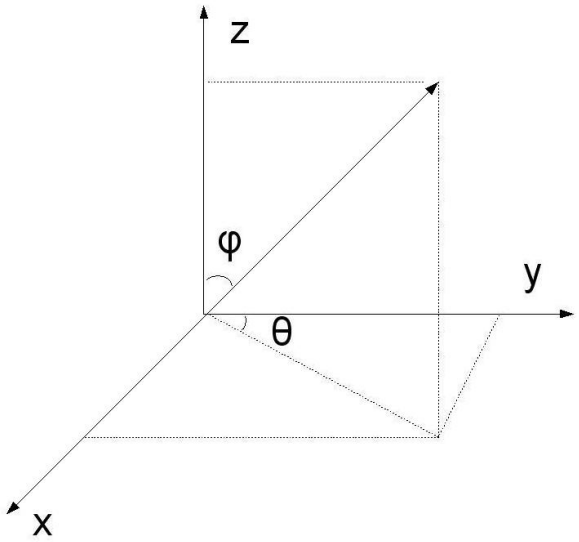

Figure 2.3: The angles that determine the position of the aircraft in the 3D space.

Considering that the the two aircraft follow a linear path with constant velocities we get the following equation.

$$s = u_1 t \tag{2.10}$$

On the other hand we have that the length **s** covered by an object in a flight is equal to :

$$s = \sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2} \tag{2.11}$$

considering the mathematical type of distance.

By equating the (2.10) and (2.11) we get:

$$u_1 t = \sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2} \tag{2.12}$$

If we replace term $(x - x_1)$ with its equal from equation (2.5) $a_1\lambda$ we finally have:

$$u_1 t = \lambda\sqrt{(a_1)^2 + (a_2)^2 + (a_3)^2} \tag{2.13}$$

and if we set the equation: $u_1 t = \lambda|a|$ analyzing the above equation we have:

$$\lambda = \frac{u_1 t}{|a|} \tag{2.14}$$

Furthermore we can replace the term λ with it's equal in equation (2.5) and finally take the equation below:

$$x = x_1 + \frac{a_1 u_1}{|a|} t \tag{2.15}$$

$$y = y_1 + \frac{a_2 u_1}{|a|} t \tag{2.16}$$

$$z = z_1 + \frac{a_3 u_1}{|a|} t \tag{2.17}$$

With these functions we know where an aircraft is positioned at the moment t. So we know the distance between the two aircraft at the specific time t. The distance between the two aircraft at the time t is:

d=$\sqrt{((x_i(t) - x_j(t))^2 + (y_i(t) - y_j(t))^2 + (z_i(t) - z_j(t))^2}$

2.1.3 Collection Of Collision Data

We construct 4000 collision cases using a Pentium 4 at 2,3 GHz. The running time is approximately four months. The data are selected via a random number generator, uniformly. Some of those data present cases where collision is unavoidable using the velocity control because either they consist of head-on collision or they are in parallel routes, where the distance between each other is less than 9 kilometers, or they possess almost the same initial points in the sphere. These cases represent 0.3% of the total number. The remaining cases are solved using non-linear programming. The information we get consists of the initial and final configuration points in the sphere, including the polar θ, ϕ angles. To begin with, when a collision occurs, two basic files are created: the colloutx.txt file that contains the initial and final parameters x, y, z in the control zone and the angles.txt file that contains the initial and final angles for the two aircraft. The code stops running only when our data reach the number of 2000 or if we force it to stop.

Procedure of data collection

In this section the procedure of the data collection is explained. We actually want to simulate real collision scenarios during free flight. The scenarios are created through a MATLAB code. In the beginning of this program the control environment, where the two aircraft fly, is created. This environment is a sphere with a diameter of 216 km.

Each aircraft covers a distance in this sphere. This distance is defined by two points in the three dimension space, the initial and the final point in the sphere. Simply by connecting this points with a straight line we get the distance that each aircraft will cover. We create these points with the help of the polar angles θ, ϕ as it is shown in the above equations:

$$x = Rcos(\theta)sin(\phi)$$

$$y = Rsin(\theta)sin(\phi)$$

$$z = cos(\phi)$$

Since we want to have random points we actually select random polar angles θ, ϕ. We need four angles for each aircraft, $\theta_{in}, \theta_{fin}, \phi_{in}, \phi_{fin}$ for the initial and final points. This is achieved by the use of the function rand() that originally generates values from 2^{-53} to $1 - 2^{-53}$. We want to have values for the angle θ from -1.57 to 1.57 because $\theta = [-\frac{\pi}{2}, \frac{\pi}{2}]$ and for ϕ from 0 to

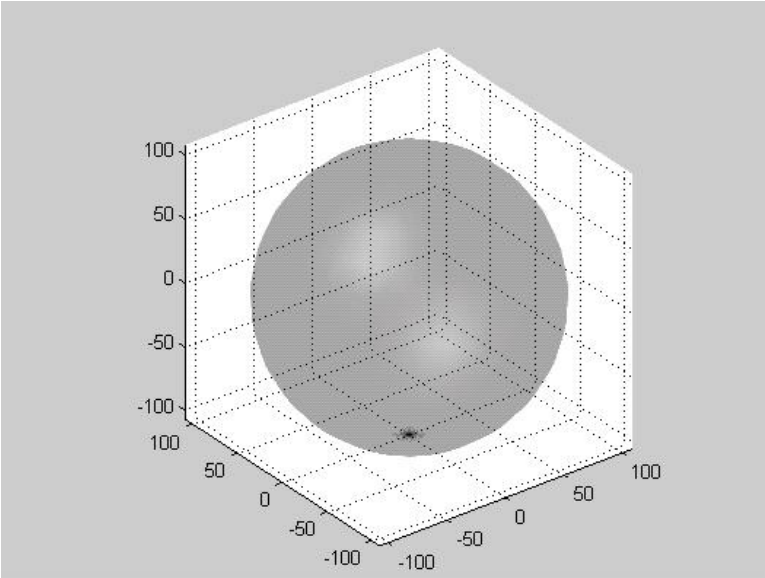

Figure 2.4: The control sphere created by MATLAB code.

6.28 because $\phi = [0, 2\pi)$. Thus we multiply the given number from function rand() with an appropriate value so as to get the desired range of values. The velocity for each aircraft is defined in the beginning of the program. From the program we get two categories of scenarios: those where conflict occurs and those where conflict does not occur. The route of each aircraft is created step by step. The aircraft are moving straight with constant velocities, thus the distance is given by the equation $s = ut$. Given a value for t and consider the velocity as a constant, we can measure the current distance covered as well as the position at a specific time t. Knowing the position of each aircraft, we can check for every time t the distance between the two aircraft and evaluate, if the two aircraft are going to conflict. The two aircraft conflict, when their between distance at time t is less than 9 km. All these scenarios are schematically represented by a video created during the execution of the program. The initial and final points as well as the routes and the conflict points are shown in this video. In case of conflict we represent the conflict points with blue color. Moreover two files are saved for a conflict case. The first file contains the initial and final coordinates of aircraft. The name of the file is collout.txt and in the first (second) column has the coordinates for the first (second) aircraft. The other file named angles.txt contains in the first (second) row the angles $(\theta_{in}, \theta_{fin}, \phi_{in}, \phi_{fin})$ for the first (second) aircraft. Cases without conflict are not saved but they encountered instead for statistic

reasons. There is a variable named counter that is increased for every new case. It was noticed that the total amount of cases was 80.000 and only the 5% were conflict cases. The program was running for 18 hours per day for 120 days . Thus we had an average of 34 conflict cases per day. We choose to make a program that generates both conflict and non-conflict cases because we wanted to approach a real scenario. Finally it is good to mention that some data of a percentage of 0.2% was generated twice. To find these data we used the command diff of program cygwin. In every new step we were comparing the new file with the previous ones. In case the new file already existed, we simply discarded it. Thus all data collected are unique.

Chapter 3

Non-Linear programming

3.1 General about non-linear programming

Having the function [12] :

$minf(x), x\epsilon X$

Where

- $f : R \rightarrow R$ is a continuous (and usually differentiable) function of n variables
- $X = R^n$ or X is a subset of R^n with a "continuous" character.
    1. If $X = R^n$, the problem is called unconstrained
    2. If f is linear and X is polyhedral, the problem is a linear programming problem. Otherwise it is a nonlinear programming problem.

Linear and nonlinear programming have traditionally been treated separately. Their methodologies have gradually come closer.

A constrained non-linear programming problem deals with the search for a maximum (or minimum) of a function $f(x)$ of n variables $x = (x_1, x_2, ..., x_n)$ subject to a set of inequality constraints $g_j(x) \leq 0, (g_j(x) = 0, j = 1, 2, ..., p)$, and is denoted as:

$$\text{Maximize } f(x) \text{subject to } g_j(x) \leq b_j, j = 1, 2, ..., m$$

If any of the functions f(x), h(x), g(x) is non-linear, then the above formulation is called a constrained non-linear programming problem. The functions f(x), h(x), g(x) can take any form of non-linearity, and it is assumed that

they satisfy continuity and differentiability requirements. No algorithm that will solve every specific problem fitting this format is available. However, substantial progress has been made for some important special cases of this problem by making various assumptions about these functions, and research is continuing very actively. Closely related to the idea of non-linear programming are the notions of convex sets as well as convex and concave functions. We will briefly define these notions below [9]:

Convex set definition:

A set $S \epsilon R^n$ is said to be convex if the closed line segment joining any two points x_1 and x_2 of the set S, that is ,$(1-\lambda)x_1 + \lambda x_2$ belongs to the set S for each λ such that $0 \leq \lambda \leq 1$.

Convex function definition:

Let S be the convex subset of R^n, and $f(x)$ be a real valued function defined on S. The function $f(x)$ is said to be convex if for $x_1, x_2 \epsilon S$, and $0 \leq \lambda \leq 1$, we have $f[(1-\lambda)x_1 + \lambda x_2 \leq (1-\lambda)f(x_1) + \lambda f(x_2)]$.
This inequality is called Jensen's inequality after the Danish mathematician who first introduced it.

Concave function definition:

Let S be a convex subset of R^n, and $f(x)$ be a real valued function defined on S. The function $f(x)$ is said to be concave if for any $x_1, x_2 \epsilon S$,and $0 \leq \lambda \leq 1$, we have:$f[(1-\lambda)x_1 + \lambda x_2 \geq (1-\lambda)f(x_1) + \lambda f(x_2)]$.

In simpler terms, a convex function is always "curving upward" (or not at all) and a concave function is always "curving downward" (or not at all). If a non-linear programming problem has no constraints, the objective function being concave guarantees that a local maximum is a global maximum. Similarly, the objective function being convex ensures that a local minimum is a global minimum. If there are constraints, then one more condition will provide this guarantee, namely, that the feasible region is a convex set. In essence, a convex set is simply a set of points such that, for each pair of points in the collection, the entire line segment joining these two points is also in the collection.

In general, the feasible region for a non-linear programming problem is a convex set whenever all the $g_j(x)$ [for the constraints $g_j(x) \leq b_j$ are convex. The subject of non-linear programming is a very large one and is constantly updated and reviewed.

3.1.1 GAMS software package

GAMS [4] provides a high-level language for the compact representation of large and complex models, allowing changes to be made in model specifications simply and safely. Also it allows unambiguous statements of algebraic relationships and permitting model descriptions that are independent of solution algorithms.

The design of GAMS incorporated ideas drawn from relational database theory and mathematical programming and attempted to merge these ideas to suit the needs of strategic modelers. Relational database theory provides a structured framework for developing general data organization and transformation capabilities. Mathematical programming provides a way of describing a problem and a variety of methods for solving it. The following principles were used in designing the system:

- All existing algorithmic methods should be available without changing the user's model representation. Introduction of new methods, or of new implementations of existing methods, should be possible without requiring changes in existing models. Linear, nonlinear, mixed integer, mixed integer nonlinear optimizations and mixed complementarity problems can currently be accommodated.

- The optimization problem should be expressible independently of the data it uses. This separation of logic and data allows a problem to be increased in size without causing an increase in the complexity of the representation.

- The use of the relational data model requires that the allocation of computer resources be automated. This means that large and complex models can be constructed without the user having to worry about details such as array sizes and scratch storage.

Basic Components of GAMS

Inputs

- ***Sets Declaration***
  Assignment of members

- ***Data(Parameters,tables,Scalars)***
  Declaration
  Assignment of values

- ***Variables***
  Declaration
  Assignment of type
- ***Assignment of bounds and/or initial values(optional)***
- ***Equations***
  Declaration
  Definition
- ***Model and Solve statements***
- ***Display statement(optional)***

Outputs

- ***Echo Print***
- ***Reference Maps***
- ***Equation listings***
- ***Status reports***
- ***Results***

3.1.2 Problem Example

To understand the use of GAMS a simple problem is presented below [13]: In the transportation problem, we are given the supplies at several plants and the demands at several markets for a single commodity. And we are given the unit costs of shipping the commodity from plants to markets. The question is: how much shipment should be between each plant and each market so as to minimize total transport cost? The algebraic representation of this problem is usually presented in a format similar to the following.

Indices: i =plants
j =markets

Given Data: a_i =supply of commodity of plant i (in cases)
b_j =demand for commodity at market j (cases)
c_{ij} =cost per unit shipment plant i and market j (euro/case)

Decision variables: X_{ij} =amount of commodity to ship from plant i to market j (cases)
Where $x_{ij} \geq 0$,for all i,j

Constraints: Observe supply limit at plant i : $\sum_j x_{ij} \leq a_i$, for all i cases
Satisfy demand at market j : $\sum_i x_{ij} b_j$

Objective Function: Minimize $\sum_i \sum_j c_{ij} x_{ij}$

Note that this simple example reveals some modeling practices that we regard as good habits in general and that are consistent with the design of GAMS. First, all the entities of the model are identified (and grouped) by type. Second, the ordering of entities is chosen so that no symbol is referred to before it is defined. Third, the units of all entities are specified and fourth, the units are chosen to a scale such that the numerical values to be encountered by the optimizer have relatively small absolute orders of magnitude. The names of the types of entities may differ among modelers. For example, economists use the terms "exogenous variable" and "endogenous variable" for "given data" and "decision variable", respectively. In GAMS, the terminology adopted is as follows: indices are called sets, given data are called parameters, decision variables are called variables, and constraints and the objective function are called equations. The GAMS representation of the transportation problem closely resembles the algebraic representation above. The most important difference, however, is that the GAMS version can be read and processed by the computer.

3.1.3 Optimum Velocity Changes with the help of Gams

Basic ideas of the problem

The main decision variable that determines whether or not we have conflict is the distance between the two aircraft. If this distance is less than 9 km then we have conflict. Based on this fact, the problem is approached as a function of time. Thus the solution for the avoidance of conflict lies in keeping the distance between the two objects greater than 9 km during the entire route. First of all, we sectioned the time of the route into a number of

parts. To begin with we know that the aircraft fly with a speed equal to 15 km per minute and that the control sphere has a diameter of 216 km. It is easily understandable that the greatest distance that can be covered is 216 km and this can be done in 14.4 minutes. Whether the segment of time is a minute, half a minute or a quarter of a minute depends on how accurate we need or want to be. In the specific solution, the amount of the time is a quarter of a minute. Having decided how to split time, we now require that, for all these segments of time, the distance between the two objects should be greater than 9 km. According to this, the velocities of the two collision objects will change until the desired ones are given.

Algorithm for GAMS

The velocity change occurs when two aircraft are flying in specified directions and must avoid a conflict by implementing a velocity magnitude change. Each aircraft can change its velocity by a quantity q that can take positive or negative values. It is understandable that aircraft can not make exorbitant changes in their velocities, instead they make limited ones. More specifically, there exist upper and lower bounds of velocities for each aircraft given by the following equation:

$$U_{i,min} \leqslant U_i \leqslant U_{i,max} \tag{3.1}$$

The velocity bounds differ from aeroplane to aeroplane. The new velocity that comes from the old one augmented by a value of q (negative or positive), must not exceed the minimum and maximum bounds. This can be represented by the following equation:

$$U_{i,min} \leqslant U_i + q_i \leqslant U_{i,max} \tag{3.2}$$

The usual bounds for a commercial aircraft must not exceed 1% of their nominal velocity. We choose stricter bounds in our case, 0.5%. Consequently q has lower and upper bounds as follows

$$-0.99 \leqslant q_i \leqslant 0.99 \tag{3.3}$$

In our algorithm we consider that two aircraft conflict if the distance between them is less than 9 km. The distance between them is given by the following equation:

$$d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

As mentioned before, if we manage to change the velocities in a way that expands the distance between them more than 9 km, we will have conflict

resolution. The basic issue is that there are an infinite number of values that can manage this resolution but actually only one set will be optimum according to the hierarchy defined in our algorithm. Therefore, we want to achieve $min|q_i|$, $i = 1, 2$.

Furthermore in non-linear programming there are some constraints. Our algorithm, is based on a very simple idea: By taking the route of the aircraft as a function of time we can split the route into time slices. An example shown in the figure reveals the idea.

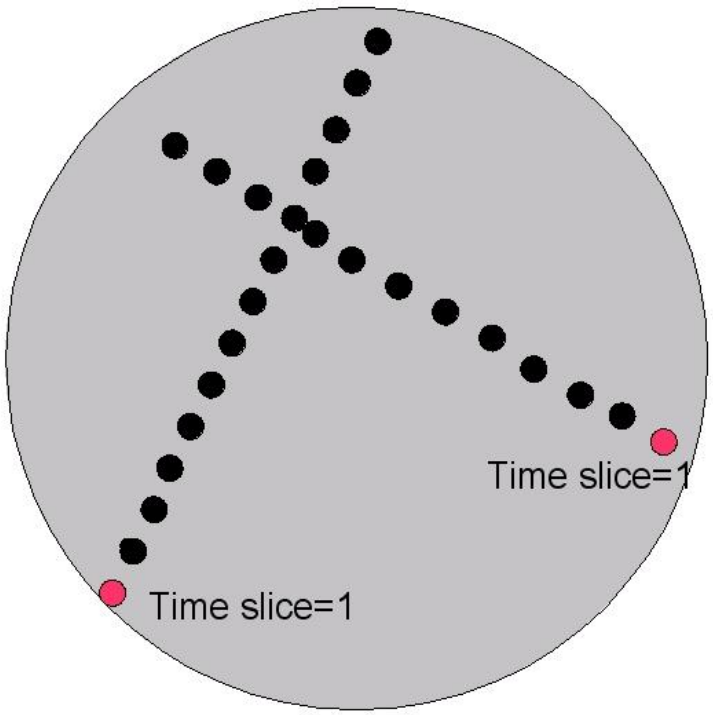

Figure 3.1: Position of each aircraft at every time slice

In the above figure, we see the positions of the aircraft in time slice 1 in red. We examine, for each time slice, the distance between the two current positions d_{12}. Whenever $d_{12} \leq d_{col}$, a collision occurs. Therefore, if for all these positions, all the distances are made greater than 9 km, then the conflict is successfully avoided.

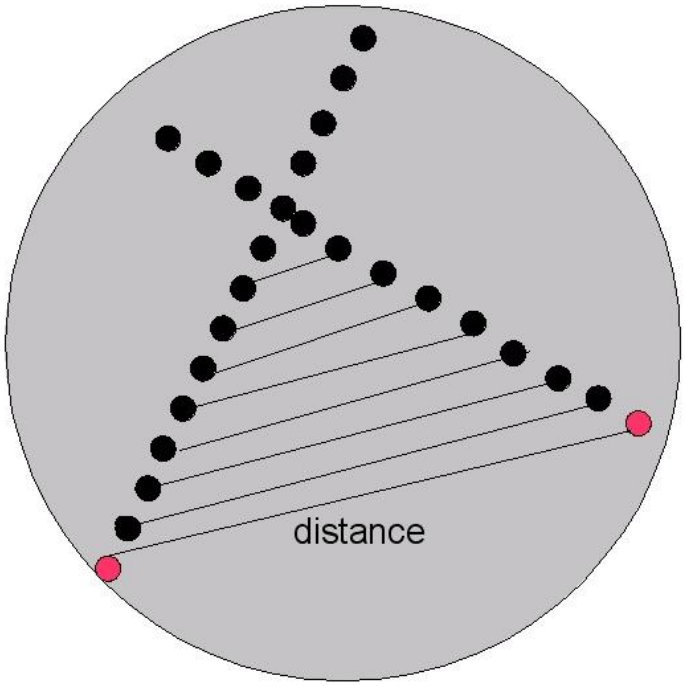

Figure 3.2: Distances between aircraft position

Thus, the constraints in our non-linear program according to 3.10 are the following:

$$dis(1, i, j) > 9 \tag{3.5}$$

and

$$dis(2, i, j) > 9 \tag{3.6}$$

and

$$dis(3, i, j) > 9 \tag{3.7}$$

and

$$dis(4, i, j) > 9 \tag{3.8}$$

and

.

.

.

$$d(n, i, j) > 9 \tag{3.9}$$

The maximum time n in our problem is equal to 50. Consider that the max distance, each aircraft can travel is 216 km and with a velocity of 15km/minute, it needs 14,4 minutes in total. So by taking almost a quarter of minute as our time slice, we need at most 50 slices.

At the end of each GAMS program we get the optimum velocity changes. By applying these changes, actually we decrease the velocity for one of the two aircraft and increase the other's. As a result, the aircraft passes the crucial conflict point at another time slice and thus the conflict is avoided.

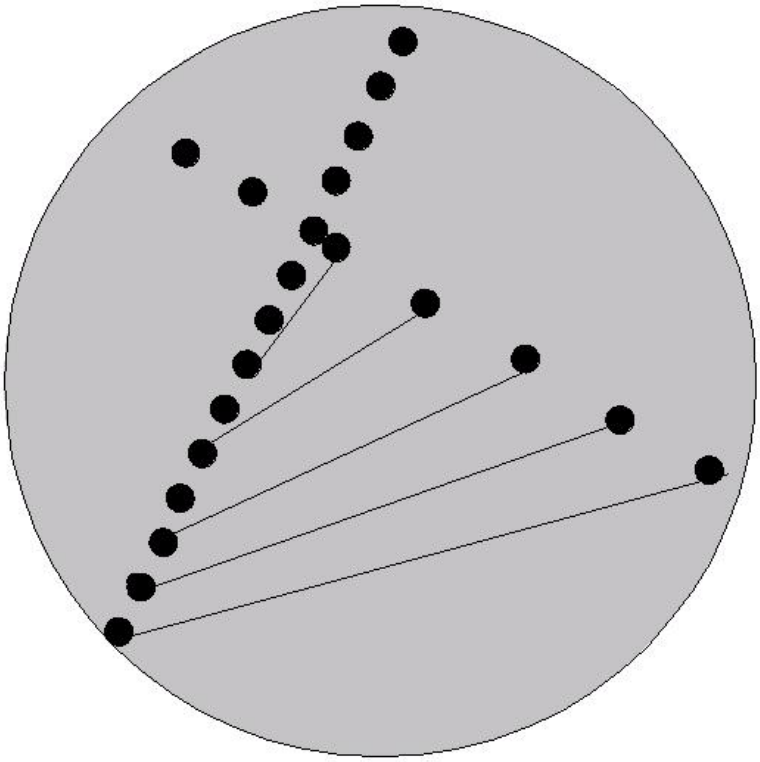

Figure 3.3: With the velocity changes the aircraft pass through the crucial point at different time slices

Implementation of the problem

GAMS is an important tool in non linear programming. It is used to compute the minimum velocity change of two aircraft that are about to collide, in order to avoid this from happening.

The algebraic representation of conflict avoidance problem is represented in the format below :

Indices i=number of aircraft
t=timeslice

Given Data: *Coordinates:*
$x_i = x$ coordinate for the i aircraft(while entering the sphere in Km)
$y_i = y$ coordinate for the i aircraft(while entering the sphere in Km)
$z_i = z$ coordinate for the i aircraft(while entering the sphere in Km)
$x_{fin} = x$ coordinate for the i aircraft(while exiting the sphere in Km)
$y_{fin} = y$ coordinate for the i aircraft(while exiting the sphere in Km)
$z_{fin} = z$ coordinate for the i aircraft(while exiting the sphere in Km)

Decision Variables: f equals to $\sum(i, q_i)$

Constraints: q_i where q is belong in the set [-0.9,0.99]

Objective Function: Considering that:

$A_1(j) = x_{final}(i) - x_i$

$A_2(j) = y_{final}(i) - y_i$

$A_3(j) = z_{final}(i) - z_i$

$u(i)$ are the initial velocities of the two aircraft

$q(i)$ are the velocity changes

number(t)=0,1,2,3,...,50

$distance(i) = \sqrt{(A_1(i))^2 + (A_2(i))^2 + (A_3(i))^2}$

considering that:

$B_x(i) = x(i) + \frac{A_1(i)}{distance(i)})(u(i) + q(i))0,5number(t)$
$B_y(i) = y(i) + \frac{A_2(i)}{distance(i)})(u(i) + q(i))0,5number(t)$
$B_z(i) = z(i) + \frac{A_3(i)}{distance(i)})(u(i) + q(i))0,5number(t)$

Then the objective function will be:

$$dis_t(i,j) = \sqrt{(B_x(j) - B_x(i))^2 + (B_y(j) - B_y(i))^2 + (B_z(j) - B_z(i))^2} \tag{3.10}$$

The complexity of the algorithm based on the aircraft number i is $O(i^2)$ and that is because we have n aircraft then we are going to take complexity $\binom{i}{2}25 = \frac{i!}{(i-2)!*2!*25} = \frac{i(i-1)}{2}$.

Collision scenarios

There are three basic categories of collision cases.

1. Cases in which the two aircraft are conflicting straight forward
2. Cases in which the two aircraft enter the control sphere from almost the same point
3. And last but not least all other cases of conflict

In the first two categories there is no solution to the problem while in the third category all cases can be solved. In the specific problem it was desired to select only cases from category three and ignore the other two categories. Nevertheless proposals will be given in the end of the chapter for categories one and two. Scenarios for each case are given below.

Case1:

Aircraft$_1$	**Initial**	**Final**
Angleθ	0.14965	4.79630
Angleϕ	4.8592	5.1159
x	-24.218	24.400
y	-14.809	14.638
z	-42.402	42.113
Aircraft$_2$	**Initial**	**Final**
Angleθ	4.7800	0.14900
Angleϕ	5.1060	4.8590
x	24.4840	-24.200
y	14.500	-14.799
z	-41.99	-42.450

The conflict scheme is given below:

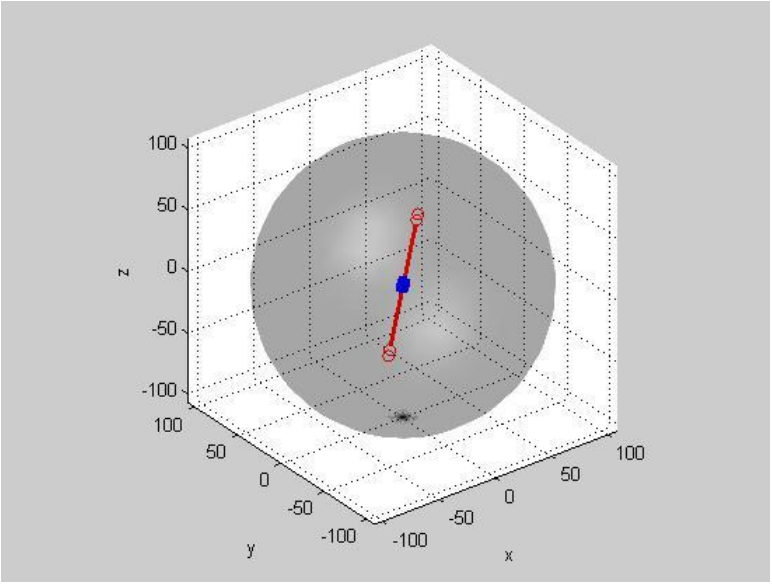

Figure 3.4: Case1 Example1

Case2:
Example1:

Aircraft$_1$	**Initial**	**Final**
Angleθ	5.4066	5.2772
Angleϕ	4.3041	1.2738
x	-63.42	55.281
y	76.181	-87.231
z	-42.875	31.603
Aircraft$_2$	**Initial**	**Final**
Angleθ	2.2871	5.1811
Angleϕ	1.9524	0.3153
x	-65.813	-15.129
y	75.599	-29.876
z	-40.217	102.68

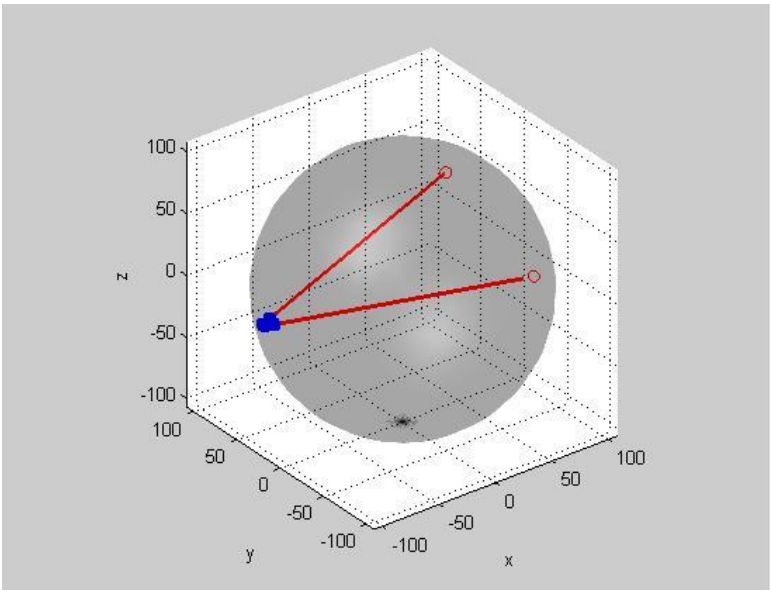

Figure 3.5: Case2 Example1

Example2:

Aircraft$_1$	**Initial**	**Final**
Angleθ	4.6026	5.3576
Angleϕ	2.5832	2.7386
x	-6.2674	25.469
y	-56.875	-33.838
z	-91.597	-99.35
Aircraft$_2$	**Initial**	**Final**
Angleθ	4.6466	3.6438
Angleϕ	2.6422	1.948
x	-3.4001	-88.012
y	-51.609	-48.33
z	-94.81	-39.777

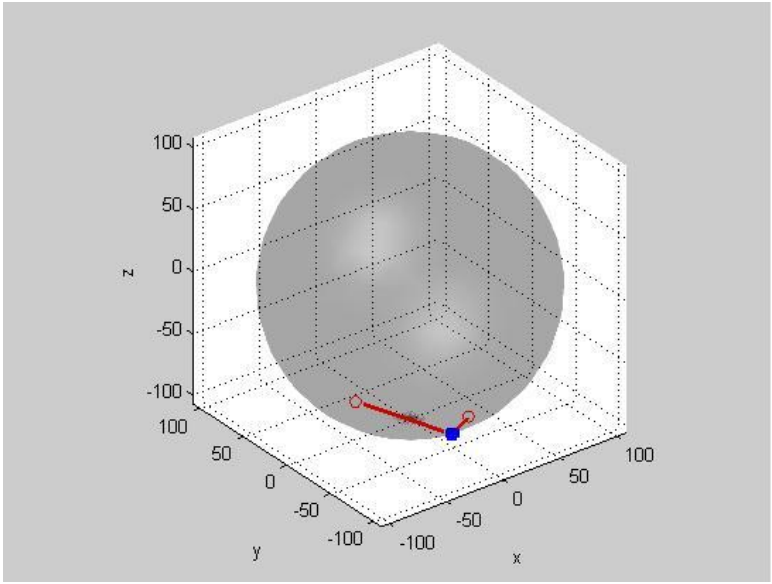

Figure 3.6: Case2 Example2

Case3:
Example1:

Aircraft$_1$	**Initial**	**Final**
Angleθ	4.2087	2.0148
Angleϕ	1.5943	2.301
x	-52.114	-33.968
y	-94.561	71.402
z	-2.5359	-73.566

Aircraft$_2$	**Initial**	**Final**
Angleθ	0.6791	2.1375
Angleϕ	5.1478	2.6372
x	-76.198	-28.02
y	-61.507	44.038
z	45.55	-94.549

In this category every conflict has an optimum solution.

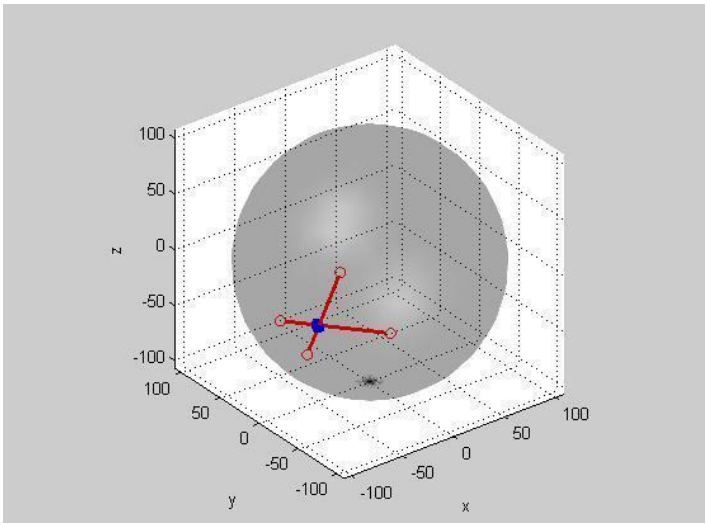

Figure 3.7: Case3 Example1

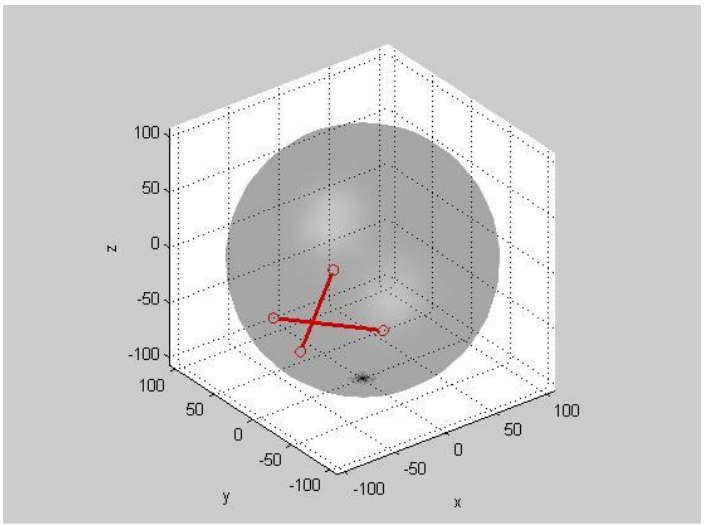

Figure 3.8: Case3 Example1 after conflict resolution

The velocity changes are: for the first aircraft 0.023 and for the second -0.021.

Example2:

Aircraft$_1$	**Initial**	**Final**
Angleθ	4.9967	1.2665
Angleϕ	2.0376	2.1732
x	27.053	26.664
y	-94.575	84.901
z	-48.6	-61.195
Aircraft$_2$	**Initial**	**Final**
Angleθ	5.4515	2.4885
Angleϕ	2.0537	2.1577
x	64.431	-71.418
y	-70.695	54.644
z	-50.149	-59.812

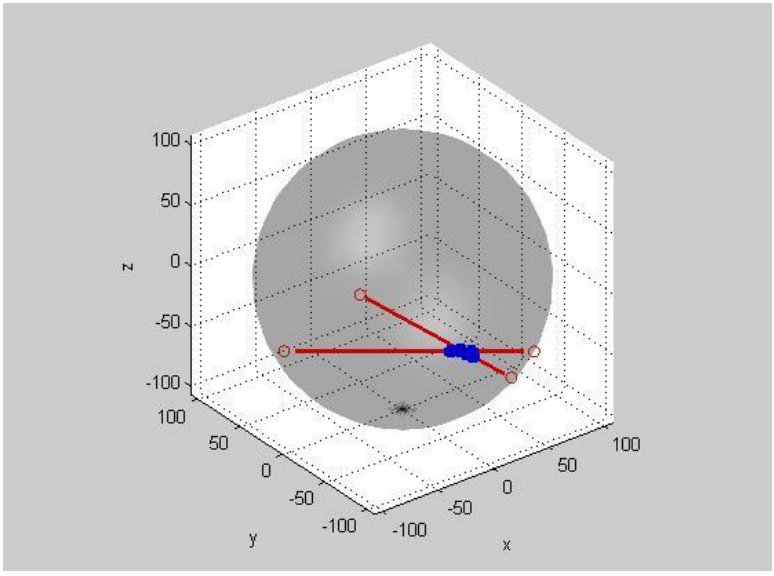

Figure 3.9: Case3 Example2

And after the conflict resolution the new scheme is the above:

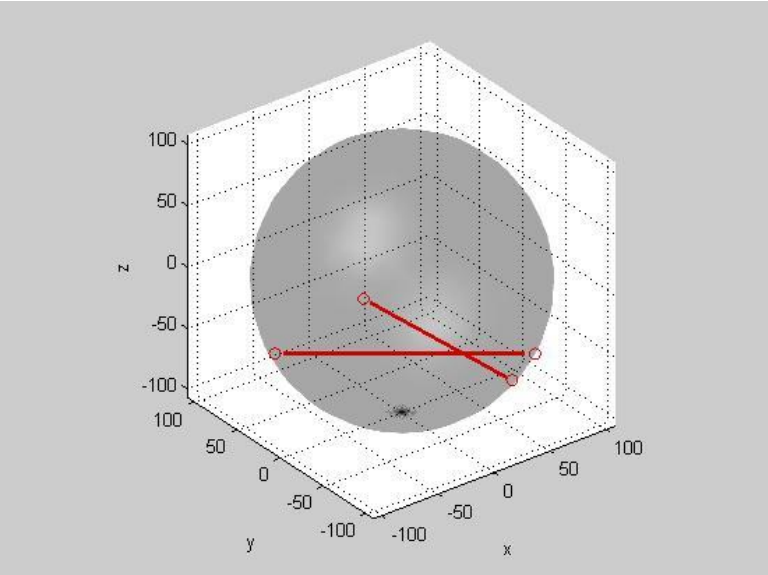

Figure 3.10: Case3 Example2 after conflict resolution

It is important to mention that the solution for the conflict is optimum with accuracy of 2 decimals. Thus, if we change the second decimal of one solution we will have conflict. The velocity change for the first aircraft is -0.634 while for the second aircraft is 0.565 . Now if we set the velocity change for the first aircraft to be 0.555 we can see in the following scheme that conflict happens again.

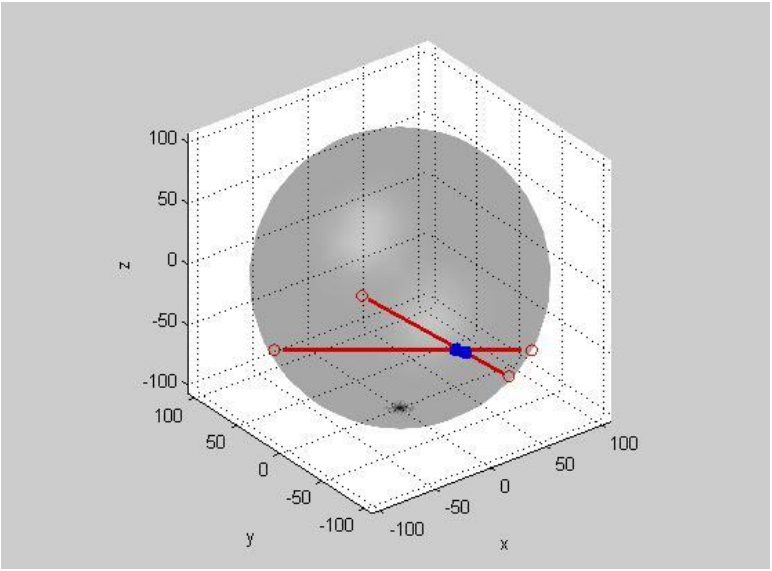

Figure 3.11: Case3 Example2

Example3:

Aircraft$_1$	**Initial**	**Final**
Angleθ	4.7468	0.007718
Angleϕ	4.2593	4.2593
x	3.6801	-97.099
y	-106.77	-0.74943
z	15.858	-47.278

Aircraft$_2$	**Initial**	**Final**
Angleθ	1.2038	3.9212
Angle ϕ	4.42	6.096600
x	-37.103	14.25
y	-96.53	14.084
z	-31.134	106.13

The velocity changes are -0.042 and 0.113 for each aircraft.

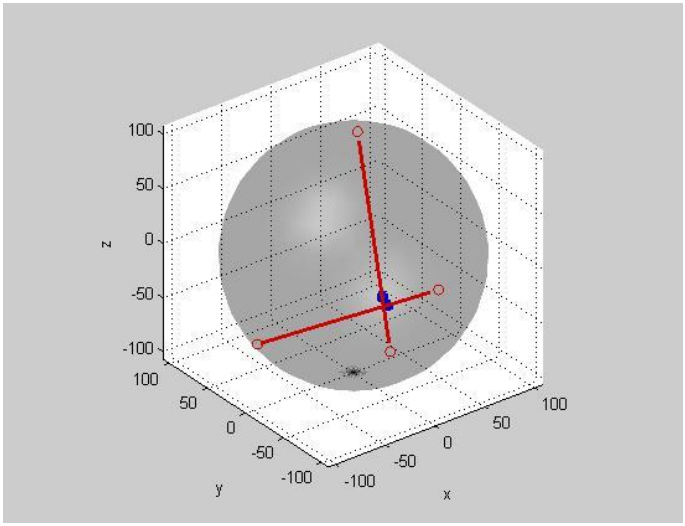

Figure 3.12: Case3 Example3

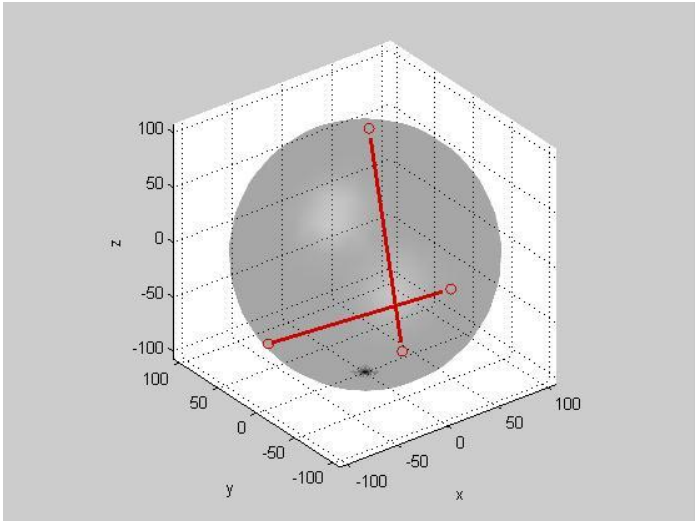

Figure 3.13: Case3 Example3 after conflict resolution

Similarly if we change the second velocity change to 0.103 the conflict will not be solved.

3.1.4 Cases with 3 aircraft

The algorithm used for the conflict solution of two aircraft applies also in cases with more aircraft. It is noticed that for three and four airplanes the algorithm is fast enough. For more than four airplanes the problem complexity is increased but still we take results fast. We set out an example for three aircraft.

Aircraft$_1$	**Initial**	**Final**
x	3.2643	16.535400
y	-16.0633	93.59
z	106.749	-51.2973
Aircraft$_1$	**Initial**	**Final**
x	-0.39850	20.0728
y	1.4585	27.527401
z	107.99	-102.4857
Aircraft$_1$	**Initial**	**Final**
x	-103.034	15.146100
y	-17.2012	-102.261
z	27.4251	31.2734

In this example collision happens between two aircraft and we have to change the velocities so that not only the initial conflict but also any possible new conflict with the third aircraft to be avoided. Before the conflict resolution we had the scheme below. The optimum velocity changes for each aircraft are : - 0.9 for the first airplane, 0.761 for the second and 0 for the third.

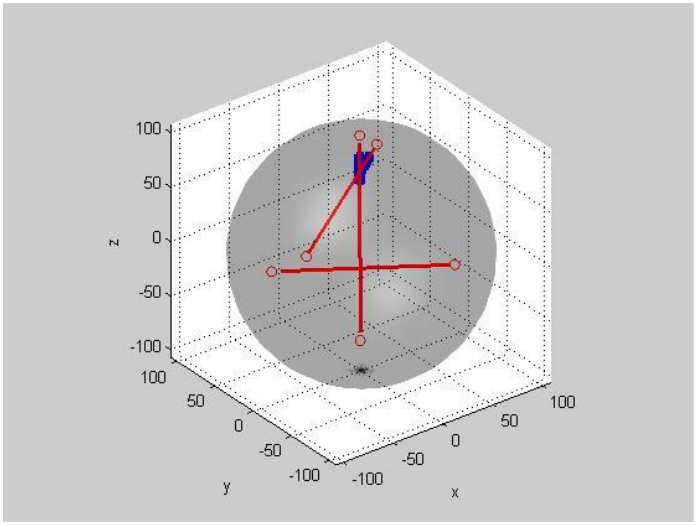

Figure 3.14: Conflict Example with three aeroplanes

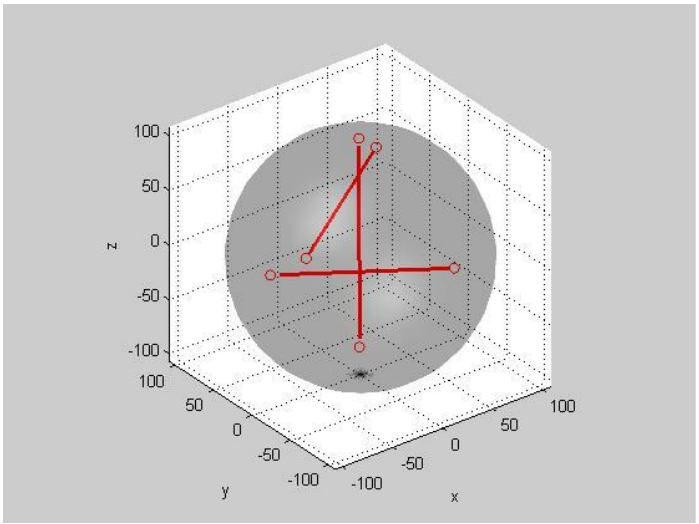

Figure 3.15: Conflict Resolution with three aeroplanes

3.1.5 Recommended Solutions for unsolvable cases with the presented algorithm (Which account for 0.3% of the total number of cases in our experiment).

As it was mentioned above, there are two cases of conflict in which the specific algorithm is not able to provide a solution. Considering the fact that aircraft conflict resolution is a serious problem that should be handled with responsibility, we have to provide solutions for all conflicts. Thus we present below solutions for two different cases.

- Solution for straight forward conflict: A lot of research has already been made for this case with the most prevalent solution the direction change for the aircraft. A simple idea is changing the angles θ or ϕ each time. For example when a conflict is imminent a system will oblige the aircraft to change its direction. A good idea would be, only one from a pair of aircraft to change its direction so that, the possibility for another conflict with another aircraft to be avoided.

- Solutions for aircraft entering in the control sphere from almost the same point: A simple proposal for this issue is to have sequential control spheres or a bigger control sphere. Then either the algorithm for velocity or direction change can be applied.

However an optimum system may have as a default solution the velocity change and when the above categories appear, the direction change can be selected as a second option.

3.1.6 Comparison with another algorithm

Many algorithms appeared that solve the velocity change problem. A very good algorithm is also the following : It considers that each aircraft is vested with a sphere and conflict happens if aircraft 1 and its safety sphere intersect, the cone generated by aircraft 2, or vice versa, as we can see in the scheme 3.1.6.

We refer to the above case, by examining the motion of two spheres in 2-dimensions because of symmetry, in the plane defined by vector $\mathbf{s}$, which represents the distance of two spheres' centers, and by u_{12} of relative speed of motion among the 2 flying aircraft. (Refer to Figure 3.1.6) [2].

Consider now the two non-parallel straight lines that are tangent to the discs of both aircraft. Let a be the angle between the first straight line and the horizontal axis, and ω be the angle between the vector u_{12} of relative

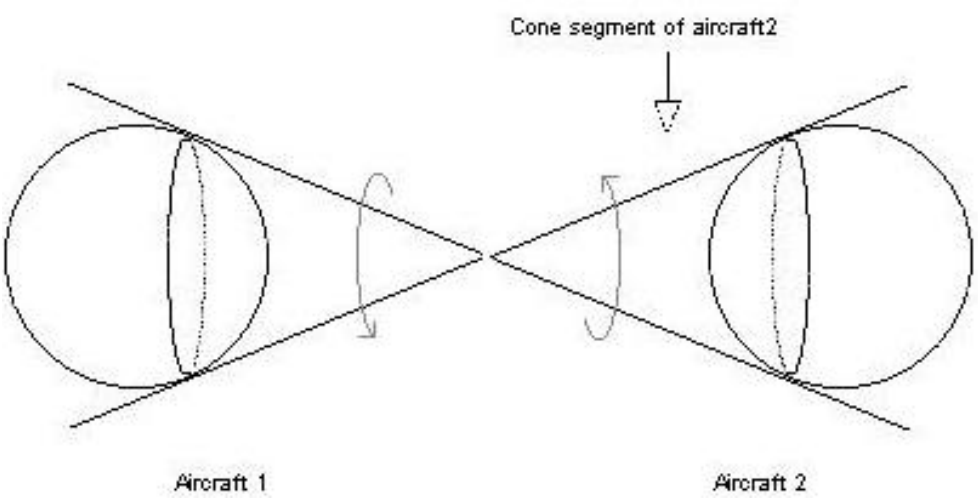

Figure 3.16: Cone sections between two moving spheres

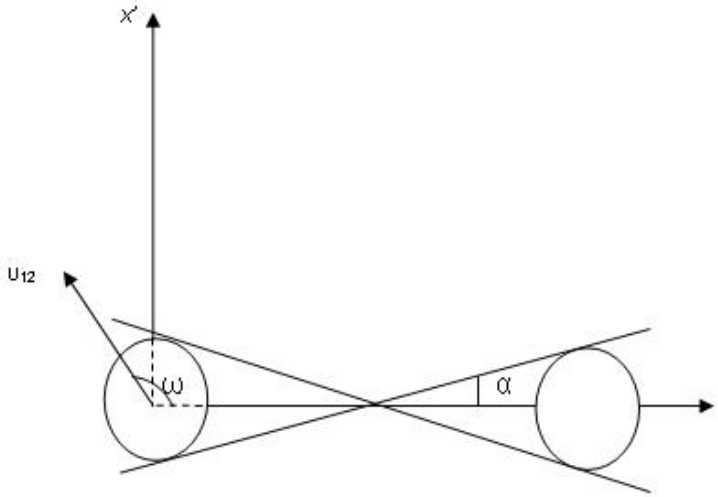

Figure 3.17: The two non parallel straight lines tangent to the safety discs of radius d/2 for two aircraft

speed and the vector **s** which represents the distance of the two spheres centers. If ω is the angle between vectors u_{12} and **s** we have

$$cos\omega = \frac{u_{12} * s}{|u_{12}| * s|} \tag{3.11}$$

where

$$s = \Big\{x2 - x1, y2 - y1, z2 - z1\Big\} \tag{3.12}$$

Since

$$tan\omega = \frac{\pm\sqrt{1 - cos\omega^2}}{cos\omega}$$

in the case of positive sign no conflict occurs if:

$$\frac{\sqrt{1 - (\frac{u_{12}*s}{|u_{12}|*|s|})^2}}{\frac{u_{12}*s}{|u_{12}|*|s|}} \geq tan(a)$$

(3.14)

or

$$\frac{\sqrt{1 - (\frac{u_{12}*s}{|u_{12}|*|s|})^2}}{\frac{u_{12}*s}{|u_{12}|*|s|}} \leq tan(-a) \tag{3.15}$$

To obtain non conflict constraints for n aircraft we need to consider the non conflict conditions described by 3.14 and 3.15 for all possible pairs of aircraft. Let's consider the pair of aircraft (i, j). We have to distinguish between two possible cases: $u_i * s < 0$ and $u_i * s > 0$ and also $tan_\omega \leq tan_{-a}$ $tan_a = \frac{\frac{d}{2}}{\frac{A_{ij}}{2}} = \frac{d}{A_{ij}}$ where A_{ij} is the distance between the two aircraft i and j. So, we obtain the following groups of constraints:

Case1: $u_{ij} * s < 0$ and tan_ω has positive sign

Case2: $u_{ij} * s > 0$ and tan_ω has positive sign

Case3: $u_{ij} * s < 0$ and tan_ω has negative sign

Case4: $u_{ij} * s < 0$ and tan_ω has negative sign

Each case has two subcases. Only one of these subcases of constraints will be used in our model for each instance. Because this cases are or cases, only one will happen each time we introduce boolean variables to have the total set of constraints.

From the complexity examination we can see that the two algorithms do not have any difference because the complexity is affected by the number of aircraft. Thus for our algorithm we have complexity equal to $O(n^2)$, as the case for our algorithm. Considering the above mentioned algorithm, we have complexity also equal to $O(n^2)$. If we examine the airplanes by pairs we have : $\binom{n}{2}8 = \frac{n!}{(n-2)!*2!*8} = \frac{n(n-1)}{2}$

So it is better to examine the two algorithms in their ability to find the optimum changes. For this reason we give some examples of conflict. We will refer to the algorithms by the names algorithm1 and algorithm2 were algorithm1 is our algorithm and algorithm2 is S.Kodaxakis algorithm.

Example1: Four airplanes distributed in the sphere and one pair involved in a conflict.

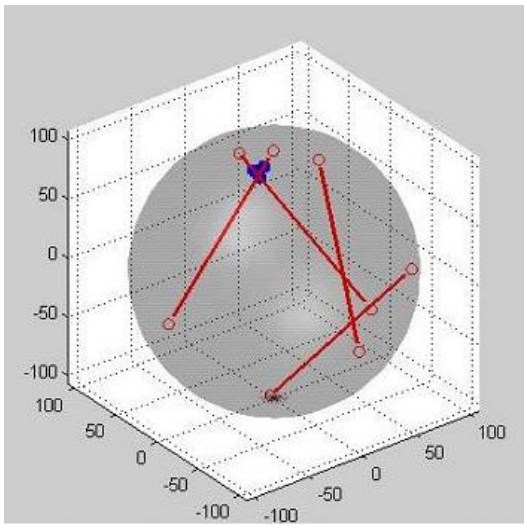

Figure 3.18: Four airplanes and one conflict.

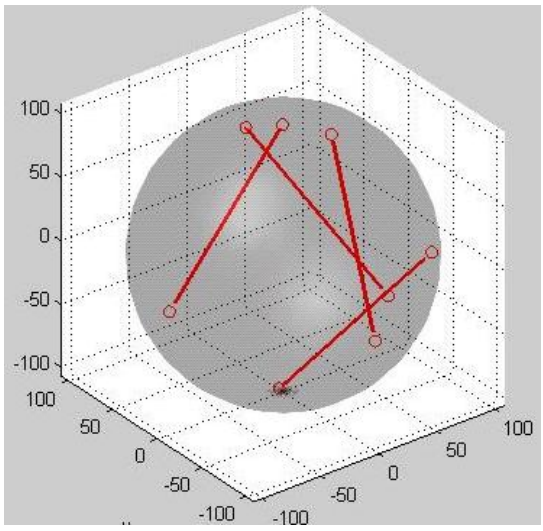

Figure 3.19: Four airplanes after conflict avoidance.

Both algorithms are able to solve the conflict. Thus it is interesting to check their advantages and disadvantages. The solution that each algorithm gives is the following.

Algorithms	**Aircraft1**	**Aircraft2**	**Aircraft3**	**Aircraft4**
Algorithm1	0.432	0	-0.596	0
Algorithm2	0.66	0.66	-0.397	0.66

We notice that our algorithm gives optimum velocities and it is not necessary for all aircraft to change their velocities. We may have optimum changes but we have bigger computation time. Thus algorithm2 give, results in 0.008 seconds in contrast to our algorithm that runs in 0.012 seconds.

Example2: The following example is five aircraft in the sphere and one conflict.

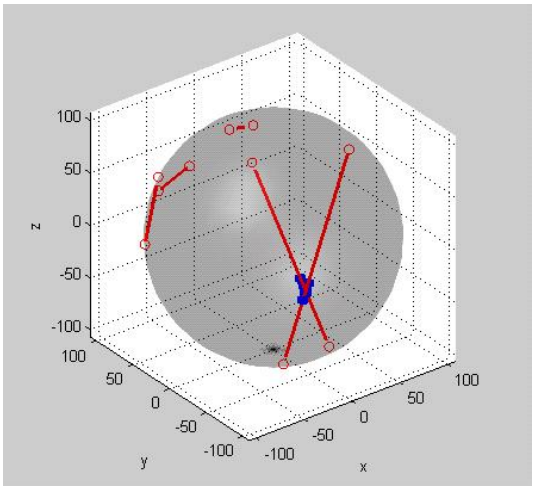

Figure 3.20: Five airplanes and one conflict.

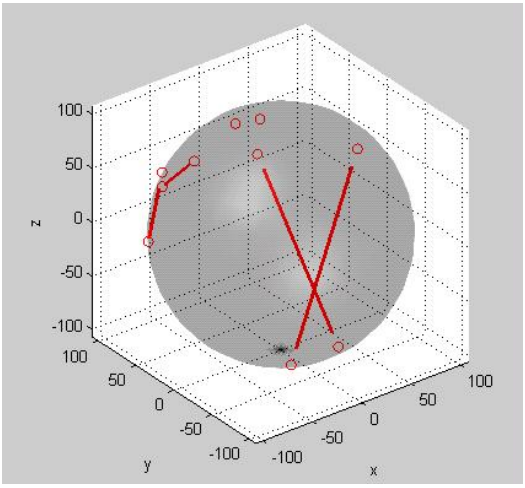

Figure 3.21: Five airplanes after the conflict avoidance.

And the recommended solutions of the two algorithms are:

Algorithms	**Aircraft1**	**Aircraft2**	**Aircraft3**	**Aircraft4**	**Aircraft5**
Algorithm1	0.896	0	0	0	0
Algorithm2	0.66	0.66	0.66	0.66	-0.196

Example3:

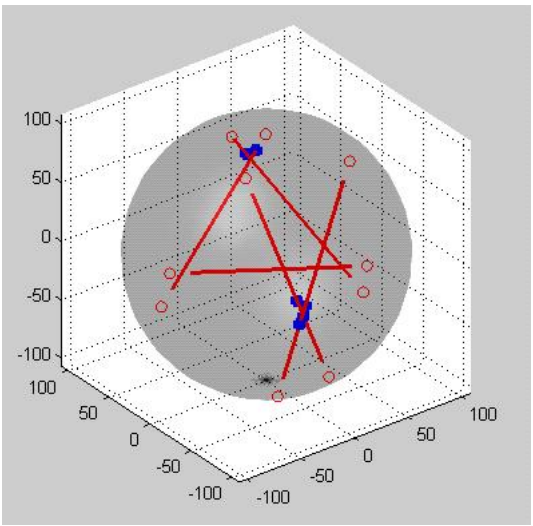

Figure 3.22: Five airplanes and two conflicts.

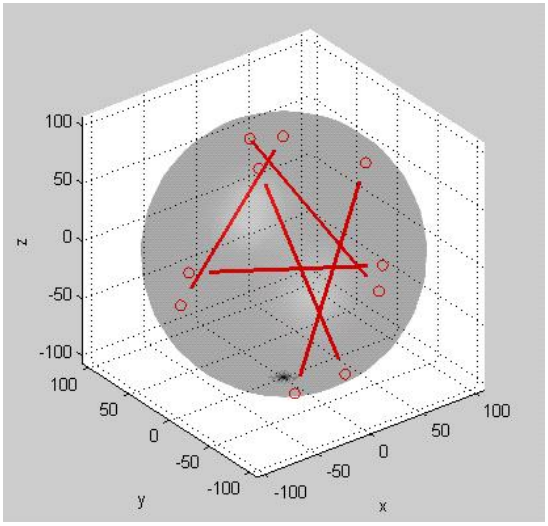

Figure 3.23: Five airplanes after the conflict avoidance.

Algorithms	**Aircraft1**	**Aircraft2**	**Aircraft3**	**Aircraft4**	**Aircraft5**
Algorithm1	0.86	0	0.19	0	0
Algorithm2	0.66	0.66	-0.397	0.66	-0.196

.

Example4:

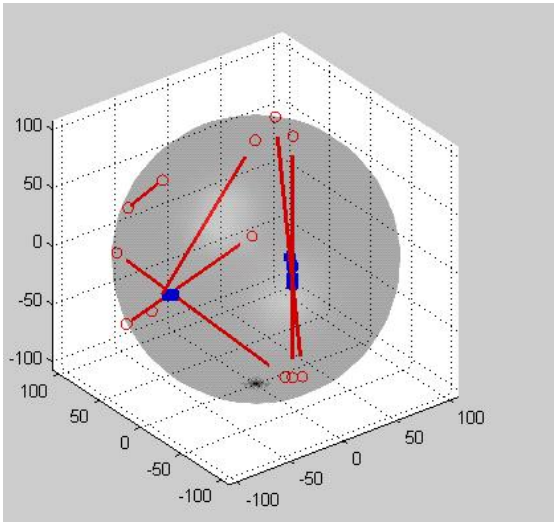

Figure 3.24: Six airplanes and two conflicts.

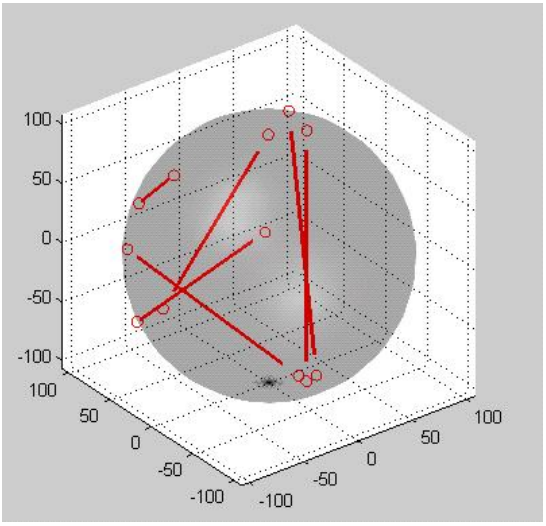

Figure 3.25: Six airplanes after the conflict avoidance.

Algorithms	**Air1**	**Air2**	**Air3**	**Air4**	**Air5**	**Air6**
Algorithm1	0	0.3	0	0.43	0	0
Algorithm2	0.364	0.66	0.66	0.136	0.66	0.66

.

One disadvantage for algorithm2 is that it does not calculate the angles that are used(a,ω), instead they are calculated separately and this makes the total computation time bigger. If those extra calculations are inserted into the algorithm we will manage to decrease the total response time. Another disadvantage is that the algorithm does not provide optimum velocity changes as someone can notice from the above examples. Let's mention that the two algorithms were examined and compared through an amount of examples that reached 40. Our disadvantage is the execution time of the algorithm. In algorithm2 the execution time is 0.008 but in our case 0.012 sec.

.

Chapter 4

Neural Networks

4.1 Fundamentals of Neural Network

An artificial network consists of a pool of simple processing units which communicate by sending signals to each other over a large number of weighted connections [3]. A set of major aspects of a parallel distributed model can be distinguished :

1. a set of processing units ('neurons,' 'cells').

2. a state of activation y_k for every unit, which equivalent to the output of the unit; connections between the units. Generally each connection is defined by a weight w_{jk} which determines the effect which the signal of unit j has on unit k.

3. a propagation rule, which determines the effective input s_k of a unit from its external inputs.

4. an activation function F_k, which determines the new level of activation based on the effective input $s_k(t)$ and the current activation $y_k(t)$ (i.e., the update); an external input (bias, offset) j_k for each unit.

5. a method for information gathering (the learning rule), an environment within which the system must operate, providing input signals and (if necessary)error signals.

Each unit performs a relatively simple job: receive input from neighbors or external sources and use this to compute an output signal which is propagated to other units. Apart from this processing, a second task is the adjustment of the weights. The system is inherently parallel in the sense that many units can carry out their computations at the same time. Within

neural systems it is useful to distinguish three types of units: input units (indicated by an index i) which receive data from outside the neural network, output units (indicated by an index o) which send data out of the neural network, and hidden units (indicated by an index h) whose input and output signals remain within the neural network. During operation, units can be updated either synchronously or asynchronously. With synchronous updating, all units update their activation simultaneously; with asynchronous updating, each unit has a (usually fixed) probability of updating its activation at a time t, and usually only one unit will be able to do this at a time. In some cases the latter model has some advantages. The figure below shows the fundamental components of neural network:

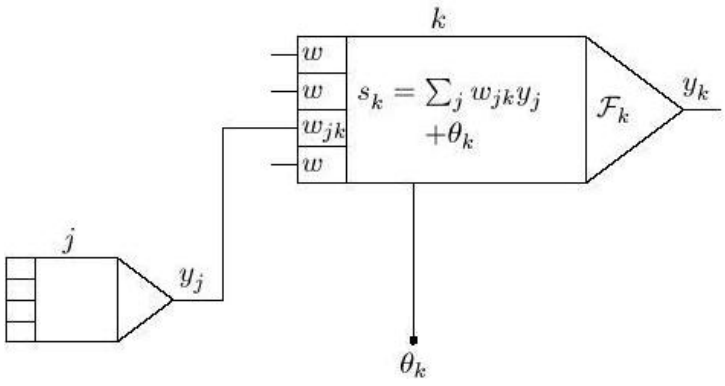

Figure 4.1: The basic components of neural network.

4.1.1 Connections between units

In most cases we assume that each unit provides an additive contribution to the input of the unit with which it is connected. The total input to

unit k is simply the weighted sum of the separate outputs from each of the connected units plus a bias or offset term

j_k: $s_k(t) = \sum_j \omega_{jk}(t) y_j(t) + \theta_k(t)$.

The contribution for positive ω_{jk} is considered as an excitation and for negative ω_{jk} as inhibition. In some cases more complex rules for combining inputs are used, in which a distinction is made between excitatory and inhibitory inputs. We call units with a propagation rule sigma units.A different propagation rule, introduced by Feldman and Ballard (Feldman Ballard, 1982),is known as the propagation rule for the sigma-pi unit:

$s_k(t) = \sum_j \omega_{jk}(t) \prod y_{j_m}(t) + \theta_k(t)$

Often, the y_{jm} are weighted before multiplication. Although these units are not frequently used,they have their value for gating of input, as well as implementation of lookup tables (Mel, 1990).

4.1.2 Activation and output rules

We also need a rule which gives the effect of the total input on the activation of the unit. We need a function F_k which takes the total input $s_k(t)$ and the current activation $y_k(t)$ and produces a new value of the activation of the unit k:

$y_k(t+1) = F_k(y_k(t), s_k(t))$.

Often, the activation function is a nondecreasing function of the total input of the unit:

$y_k(t+1) = F_k(s_k(t)) = F_k(\sum_j \omega_{jk}(t) y_j(t) + \theta_k(t))$

although activation functions are not restricted to nondecreasing functions. Generally, some sort of threshold function is used: a hard limiting threshold function (a sgn function), or a linear or semi-linear function, or a smoothly limiting threshold . For this smoothly limiting function often a sigmoid (S-shaped) function like

$y_k = F(s_k) = \frac{1}{1+e^- s_k}$

is used. In some applications a hyperbolic tangent is used, yielding output values in the range [-1, +1].

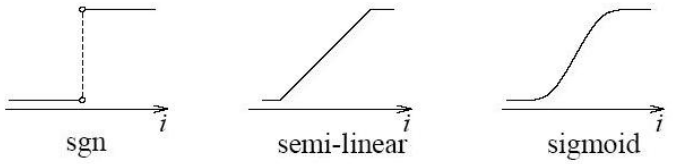

Figure 4.2: Various activation function for a unit.

In some cases, the output of a unit can be a stochastic function of the total input of the unit. In that case the activation is not deterministically determined by the neuron input, but the neuron input determines the probability p that a neuron get a high activation value:

$p(y_k \leftarrow 1) = \frac{1}{1+e^- s_k/T}$

,in which T (cf. temperature) is a parameter which determines the slope of the probability function. In all networks we describe we consider the output of a neuron to be identical to its activation level.

4.1.3 Network topologies

In the previous section we discussed the properties of the basic processing unit in an artificial neural network. This section focuses on the pattern of connections between the units and the propagation of data.As for this pattern of connections, the main distinction we can make is between:

- Feed-forward networks, where the data flow from input to output units is strictly feedforward. The data processing can extend over multiple (layers of) units, but no feedback connections are present, that is, connections extending from outputs of units to inputs of units in the same layer or previous layers.

- Recurrent networks that do contain feedback connections. Contrary to feed-forward networks, the dynamical properties of the network are important. In some cases, the activation values of the units undergo a relaxation process such that the network will evolve to a stable state in which these activations do not change anymore. In other applications,

the change of the activation values of the output neurons are significant, such that the dynamical behaviour constitutes the output of the network (Pearlmutter, 1990).

4.1.4 Training of artificial neural networks

A neural network has to be configured such that the application of a set of inputs produces (either 'direct' or via a relaxation process) the desired set of outputs. Various methods to set the strengths of the connections exist. One way is to set the weights explicitly, using a priori knowledge. Another way is to 'train' the neural network by feeding it teaching patterns and letting it change its weights according to some learning rule. We can categorize the learning situations in two distinct sorts.

These are:

- **Supervised learning** or Associative learning in which the network is trained by providing it with input and matching output patterns. These input-output pairs can be provided by an external teacher, or by the system which contains the network (self-supervised).
- **Unsupervised learning** or Self-organization in which an (output) unit is trained to respond to clusters of pattern within the input. In this paradigm the system is supposed to discover statistically salient features of the input population. Unlike the supervised learning paradigm, there is no a priori set of categories into which the patterns are to be classified; rather the system must develop its own representation of the input stimuli.

4.1.5 Terminology

Output vs. activation of a unit. Since there is no need to do otherwise, we consider the output and the activation value of a unit to be one and the same thing. That is, the output of each neuron equals its activation value [15].

Bias, offset, threshold. These terms all refer to a constant (i.e., independent of the network input but adapted by the learning rule) term which is input to a unit. They may be used interchangeably, although the latter two terms are often envisaged as a property of the activation function. Furthermore, this external input is usually implemented (and can be written) as a weight from a unit with activation value 1.

Number of layers. In a feed-forward network, the inputs perform no computation and their layer is therefore not counted. Thus a network with

one input layer, one hidden layer, and one output layer is referred to as a network with two layers. This convention is widely though not yet universally used.

Representation vs. learning. When using a neural network one has to distinguish two issues which influence the performance of the system. The first one is the representational power of the network, the second one is the learning algorithm. The representational power of a neural network refers to the ability of a neural network to represent a desired function. Because a neural network is built from a set of standard functions, in most cases the network will only approximate the desired function, and even for an optimal set of weights the approximation error is not zero. The second issue is the learning algorithm. Given that there exist a set of optimal weights in the network, is there a procedure to (iteratively) find this set of weights?

4.1.6 Networks with threshold activation function

A single layer feed-forward network consists of one or more output neurons o, each of which is connected with a weighting factor ω_{io} to all of the inputs i [5]. In the simplest case the network has only two inputs and a single output, as sketched in figure 18 (we leave the output index o out). The input of the neuron is the weighted sum of the inputs plus the bias term.

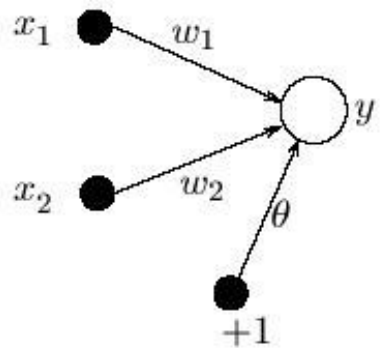

Figure 4.3: Single layer network with one output and two inputs.

The output of the network is formed by the activation of the output neuron, which is some function of the input :

$y = F(\sum_{i=1}^{2} \omega_i x_i + \theta)$

The activation function F can be linear so that we have a linear network, or nonlinear. In this section we consider the threshold (or Heaviside or sgn) function:

$$F(s) = \begin{cases} 1, & \text{if } s > 0; \\ -1, & \text{otherwise.} \end{cases}$$

The output of the network thus is either +1 or -1, depending on the input. The network can now be used for a classification task: it can decide whether an input pattern belongs to one of two classes. If the total input is positive, the pattern will be assigned to class +1, if the total input is negative, the sample will be assigned to class -1. The separation between the two classes in this case is a straight line, given by the equation:

$\omega_1 x_1 + \omega_2 x_2 + \theta = 0$

The single layer network represents a linear discriminant function. A geometrical representation of the linear threshold neural network is given in figure 14 equation, can be written as

$x_2 = -\frac{\omega_1}{\omega_2} x_1 - \frac{\theta}{\omega_2}$

and we see that the weights determine the slope of the line and the bias determines the 'offset' , i.e. how far the line is from the origin. Note that also the weights can be plotted in the input space: the weight vector is always perpendicular to the discriminant function.

Now that we have shown the representational power of the single layer network with linear threshold units, we come to the second issue: how do we learn the weights and biases in the network? We will describe two learning methods for these types of networks: the 'perceptron' learning rule and the 'delta' or 'LMS' rule. Both methods are iterative procedures that adjust the weights. A learning sample is presented to the network. For each weight the new value is computed by adding a correction to the old value. The threshold is updated in a same way:

$\omega_i(t+1) = \omega_i(t) + \Delta\omega_i(t)\ \theta(t+1) = \theta(t) + \Delta\theta(t)$

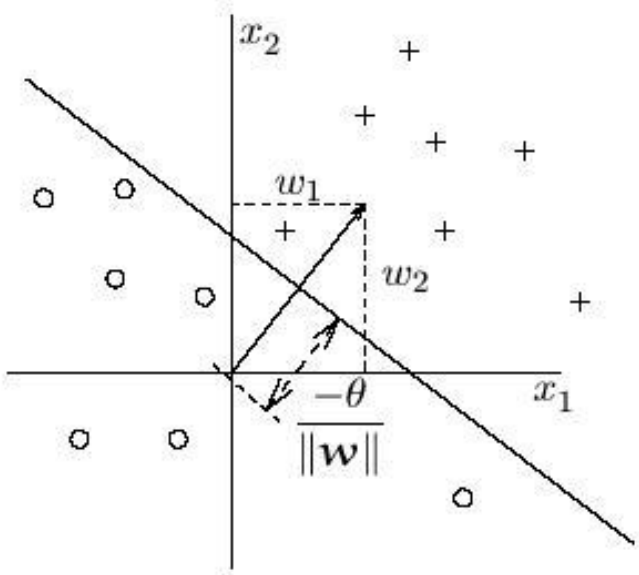

Figure 4.4: Geometric representation of the discriminant function and the weights.

4.1.7 Perceptron learning rule and convergence theorem

Suppose we have a set of learning samples consisting of an input vector x and a desired output d(x). For a classification task the d(x) is usually +1 or -1. The perceptron learning rule is very simple and can be stated as follows:

1. Start with random weights for the connections
2. Select an input vector x from the set of training samples
3. If $y \neq d(x)$ (the perceptron gives an incorrect response), modify all connections w_i according to: $D\omega_i = d(x)x_i$
4. Go back to 2

Note that the procedure is very similar to the Hebb rule; the only difference is that, when the network responds correctly, no connection weights are modified. Besides modifying the weights, we must also modify the threshold θ. This θ is considered as a connection w_0 between the output neuron and

a 'dummy' predicate unit which is always on: $x_0 = 1$. Given the perceptron learning rule as stated above, this threshold is modified according to:

$$\Delta_\theta = \begin{cases} 0, & \text{if the perceptron responds correctly;} \\ \text{d(x)}, & \text{otherwise.} \end{cases}$$

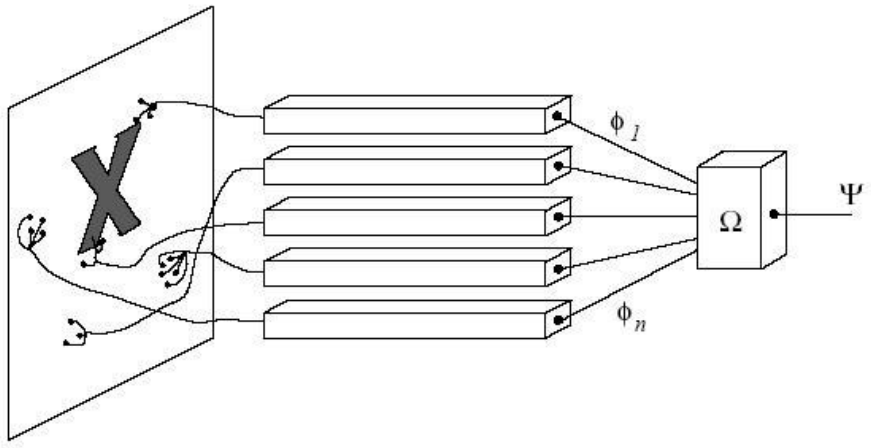

Figure 4.5: The percepton

4.1.8 Networks with linear activation functions:The delta rule

For a single layer network with an output unit a linear activation function the output is:

$$y = \sum_j = \omega_j x_j + \theta$$

Such a simple network is able to represent a linear relationship between the value of the output unit and the value of the input units. By thresholding the output value, a classifier can be constructed (such as Widrow's Adaline), but here we focus on the linear relationship and use the network for a function approximation task. In high dimensional input spaces the network represents a (hyper)plane and it will be clear that also multiple output units may be defined. Suppose we want to train the network such that a hyper plane is fitted as well as possible a set of training samples consisting of input values x^p and desired (or target) output values d^p. For every given input sample, the output of the network differs from the target value d^p by $(d^p - y^p)$, where y^p is the actual output for this pattern. The delta-rule now uses a cost- or error-function based on these differences to adjust the

weights. The error function, as indicated by the name least mean square, is the summed squared error. That is, the total error E is defined to be simply given by:

$E = \sum_p E^p = \frac{1}{2} \sum_p (d^p - y^p)^2$

where the index p ranges over the set of input patterns and E^p represents the error on pattern p. The LMS procedure finds the values of all the weights that minimise the error function by a method called gradient descent. The idea is to make a change in the weight proportional to the negative of the derivative of the error as measured on the current pattern with respect to each weight:

$\Delta_p \omega_j = -\gamma \frac{\vartheta E^p}{\vartheta \omega_j}$

where γ is a constant of proportionality. The derivative is:

$\frac{\vartheta E^p}{\vartheta \omega_j} = \frac{\vartheta E^p}{\vartheta y^p} \frac{\vartheta y^p}{\vartheta \omega_j}$

Because of linear units :

$\frac{\vartheta y^p}{\vartheta \omega_j} = x_j$

$\frac{\vartheta E^p}{\vartheta y^p} = -(d^p - y^p)$

such that:

$\Delta_p \omega_j = \gamma \delta^p x_j$

where $\delta^p = (d^p - y^p)$ is the difference between the target output and the actual output for pattern p. The delta rule modifies weight appropriately for target and actual outputs of either polarity and for both continuous and binary input and output units. These characteristics have opened up a wealth of new applications.

4.1.9 Back Propagation

In this chapter we will focus on feed-forward networks with layers of processing units. The central idea behind this solution is that the errors for the units of the hidden layer are determined by back-propagating the errors of the units of the output layer. For this reason the method is often called

the back-propagation learning rule. Back-propagation can also be considered as a generalization of the delta rule for non-linear activation functions and multilayer networks.

4.1.10 Multi layer feed-forward networks

A feed-forward network has a layered structure. Each layer consists of units which receive their input from units from a layer directly below and send their output to units in a layer directly above the unit. There are no connections within a layer. The Ni inputs are fed into the first layer of Nh,1 hidden units. The input units are merely 'fan-out' units; no processing takes place in these units. The activation of a hidden unit is a function F_i of the weighted inputs plus a bias. The output of the hidden units is distributed over the next layer of Nh,2 hidden units, until the last layer of hidden units, of which the outputs are fed into a layer of No output units (see figure 4.1.10). Although back-propagation can be applied to networks with any number of layers, just as for networks with binary units it has been shown that only one layer of hidden units suffices to approximate any function with finitely many discontinuities to arbitrary precision, provided the activation functions of the hidden units are non-linear (the universal approximation theorem). In most applications a feed-forward network with a single layer of hidden units is used with a sigmoid activation function for the units.

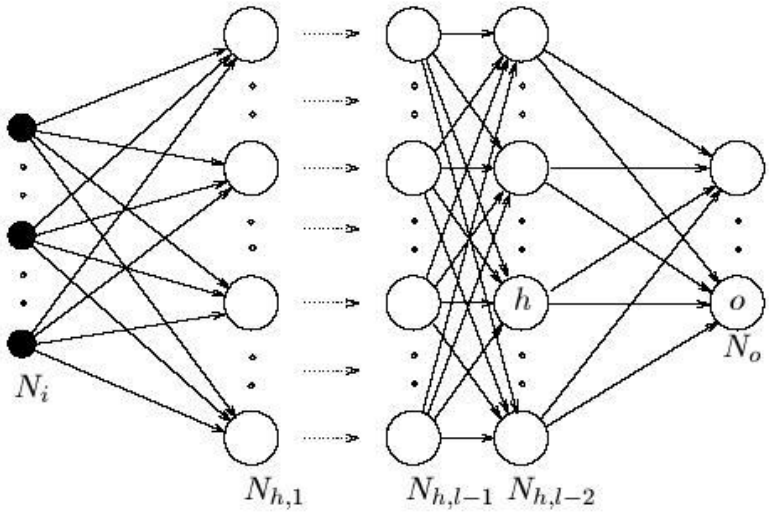

Figure 4.6: A multi-layer network with l layers of units.

4.1.11 Back propagation algorithm

The algorithm of feed forward back error propagation is given below:

```
Repeat
{
for (pno=0;pno<N_Patterns;pno++)
{
/* Forward pass */
Apply Input[pno] to input units;
Compute Y[pno] at output units ;
/* Backward pass */
For each layer, starting at output
{
For each unit in this layer
{
Compute the error at this unit
For each weight to this unit
{
Compute Dw
Apply Dw
}
}
}
}
Increment epoch counter
Compute total error
}
until (total error small enough or
epoch count exceeded)
```

Where epoch is the number of repentance of training with different inputs every time and Δ_w is the arithmetic difference between old and new weights.

4.2 Conflict avoidance with neural networks

In our problem the neural network that is selected is a back propagation neural network with two hidden layers. One input layer with twelve inputs

and one output layer with two outputs. The twelve inputs are ordered by the way they are written in the train file.

Inputs:

- The initial positions of the first aircraft $x_{in1}, y_{in1}, z_{in1}$
- The initial positions of the second aircraft $x_{in2}, y_{in2}, z_{in2}$
- The final positions of the first aircraft $x_{fin1}, y_{fin1}, z_{fin1}$
- The final positions of the second aircraft $x_{fin2}, y_{fin2}, z_{fin2}$

The first hidden layer has 50 Neurons and the second has 40 neurons. We decide to choose these numbers of neurons for each layer, as this neural network provides less training and testing error.

Outputs:

- The velocity change for the first aircraft $_1$
- The velocity change for the second aircraft $_2$

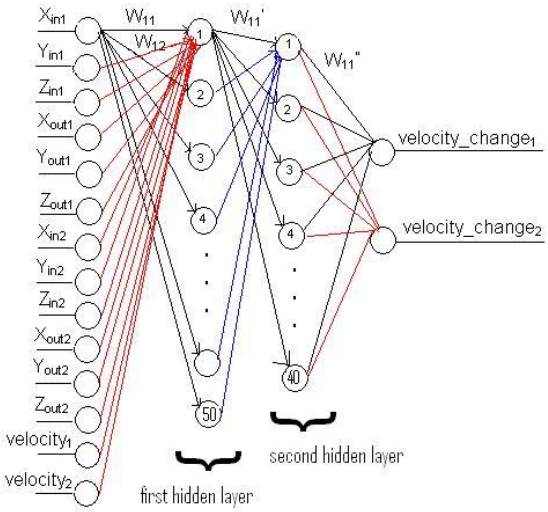

Figure 4.7: Feed Forward Back Error Propagation Neural Network with two hidden layers.The first hidden layer has 50 neuron while the second 40 neurons.

The outputs of our network are not quantized, they take continuous values. Thus we need big of training set. To minimize the training set we can follow one of the two techniques :

1. In the problem that we examine we have 2 aircraft that are about to crash and we want to avoid the conflict. An easy way to decrease the training data but also to keep our solution accurate is to give in each case the relative position of the two aircraft. For example the reference system could be the position of the first aircraft and not the beginning of the 3 axes x, y ,z of the 3D system. With this model the number of the inputs decreasing in 6 instead of 12.

2. The second proposal and the one that it is followed in the specific thesis is to limit the cases in a small part in our sphere by taking advantage of the symmetry of it. In this way, a small training set would be enough.

We finally use 3500 data for training and 600 data for testing. The reliability of the network is examined by the method of cross validation. Cross-validation is a method to estimate the generalization error based on "resampling". In k-fold cross-validation, data is divided into k subsets of (approximately) equal size. The net is trained k times, each time leaving out one of the subsets from training. In the beginning, the data are separated in ten parts (k=10) then each time 8 different parts are chosen to be the training data and the two remaining parts to be the testing data. The neural network is trained with 10 slightly different set of data. Then the final training and testing error is estimated by taking the average of all the errors resulted from each time we trained the network.

The number of epochs that was essential for the training of the neural network was 1000. The transfer function that was used in every unit was the hyperbolic tangent sigmoid function.

For example, the above function calculates the output vector **a** of a layer of hyperbolic tangent sigmoid neurons given a network input vector **p**, the layer's weight matrix **W** and bias vector **b**, and denote by

$a = tansig(W_p + b)$.

This transfer function takes the input vector of any value between plus and minus infinity, and squashes the output into the range -1 to 1, according to the expression:

$a = \frac{e^n - e^- n}{e^n + e^- n}$

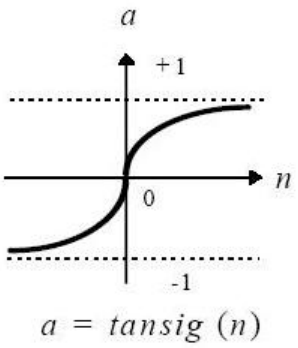

Figure 4.8: Tan-Sigmoid Transfer Function

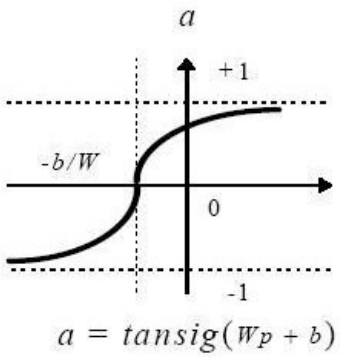

Figure 4.9: Tan-Sigmoid Transfer Function (Wp+b)

4.2.1 Construction of the neural network

In this section the procedure followed, in order to create the neural network for collision avoidance in Free Flight, is explained. The program is written in Matlab and is described in the following four steps below.

Step1 : Inputs and Outputs of the neural network

The non-linear program already gave solutions for every solvable conflict case we gathered. Thus we have a system that takes as inputs the initial and

final positions in the sphere of the each aircraft $(x_{in}, y_{in}, z_{in}, x_{fin}, y_{fin}, z_{fin})$, the initial velocities for each aircraft and as outputs provides the velocity changes so that the conflict to be avoided. With those data we construct a neural network, acting as a global interpolator system, that, after training, may provide answers to unknown inputs. It has as inputs the basic information such as positions in the sphere for each aircraft and provides the velocity changes as outputs in order for an imminent conflict between two aircraft to be avoided. Thus we train the neural network and use it in order to provide solutions for unknown conflict cases. Thus the data used in nonlinear program as well as their outputs for each case are used as training data for the neural network. They are gathered in a file named train.txt. Each row in the file represents a conflict case. The file consists of 16 columns where the first 14 represent the input of the neural network and columns 15 and 16 the desired outputs. The total number of rows is equal to the total number of conflict training cases (4000). To create the training file we need to gather data from 2 different files (collout.txt, sol.txt) that sum up to $2*4000 = 8000$. Because of the great amount of information a side program in C++ is created in order to build the train.txt file. Each time this file is executed the user provides two numbers. Thus if for example, numbers 1 and 100 are provided, data from cases 1 to 100 will be written in the train.txt file automatically. Having the train.txt file we can save the inputs that will be needed for the neural network in a array called **input** and the desired output in a array named **target**. Moreover an array named PR that contains the number of columns define the number of neurons in the first layer of the neural network.

Step2 : Construction of the neural network

The neural network in our problem is a feed forward back error propagation with full connectivity. In Matlab there is a command that builds a feed forward fully connected neural network named **newff**. The command newff has as attributes the number of neurons of the first, hidden and output layer and the transfer function for each layer. Finally the training function is defined. In this problem the training function that we use is **trainlm**. Trainlm is a network training function that updates weight and bias values according to Levenberg-Marquardt optimization [17]. Construction of a neural network means to create the layers, the neurons, the connectivity between the neurons of each layer and the weights of each connectivity. It is usually recommended initial weights to be selected at random.

Step3 : Training

In this step the neural network is trained for all conflict cases.The total number of conflict cases used in the training set is 3500 while the test set consists of 600 cases. After training it provides two outputs: the velocity changes for both aircraft. These velocity changes try to approach the optimum velocity changes. The absolute difference of their values is the training error. In training procedure there is a parameter named epochs and its value can be given before the beginning of the training. A training ends if our goal or a restriction (set by us) is achieved. A goal is usually the training error. If the training error for example turns equal to 0.001 then the training stops. A restriction can be the number of epochs, for example we can set that if the epochs turns 500 and the goal is not yet achieved then the training stops.

Step4 : Errors

The most important factors for the evaluation of a neural network is the training and the testing error. Training error must be extremely low, this means that the neural network is well trained for the input data. Testing error actually evaluate the neural network. Testing is greater than training error. The testing error is calculated by the absolute difference of a the output with the desired output for cases not used in training.

4.2.2 Evaluation of the neural network

Following the above mentioned states we train the feed forward back propagation neural network. In a first approach we get a testing error 6% that is a considerable error. In this part of the thesis we have the following issues:

- First of all it would be satisfying to have a testing error 0,1% but in such big range of values a 6% could be considered as satisfactory.
- Secondly the issue of the highest importance in this study is the accuracy of the system, especially while this study is related with aircraft where there is no tolerance in errors. Also the solutions taken from GAMS are optimum for each case and as it was mentioned above if the second decimal of one of the solutions is changed we are going to have conflict again. But the basic thing that was not mentioned until now is what we mean by using the word change: decrease or increase? The answer is simple because if the solution is negative, by giving a more

negative solution nothing will happen but by increasing the solution it is a fact that a conflict will occur. Similarly if we have a new velocity bigger than the old one if we increase this velocity even more nothing will be happened.

Sometimes the optimum velocity is not acceptable because if we slightly change the optimum solution we will fail in solving the problem. As we can see in the following scheme there are two optimum velocities, one for the first aircraft and one for the second.

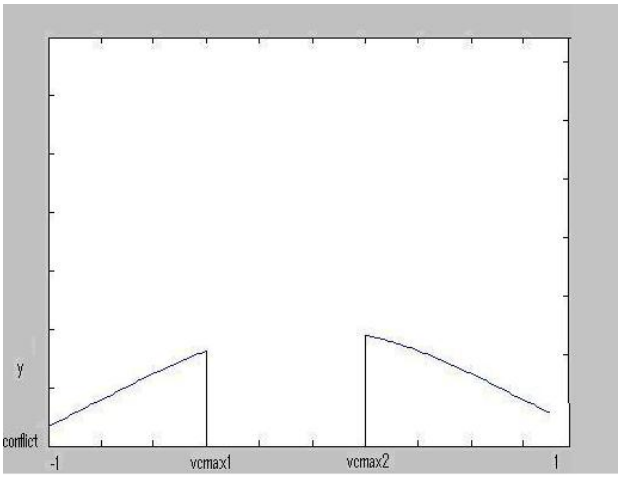

Figure 4.10: Optimum Velocity changes and failure

Assume the case where the first aircraft has to decrease its original velocity and the second to increase its velocity. Because we have optimum velocity changes if we give at the negative solution a more negative value, we still avoid the conflict but if we give a solution less negative than the optimum we will present a conflict. As we want a safe belt in our system we will adjust the solution with a value of 0,12. We either add or subtract this value depending on the category of solution as it was mentioned above. This approach makes the system more accurate and the testing error of 6% acceptable.

Two matrices with error information for 10 different training sets used through cross validation are represented below.

Set/Error	Max Training Er	Min Training Er	Average Training Er
set1	2.35	$57 * 10^{-5}$	$13 * 10^{-4}$
set2	0.487	$62 * 10^{-6}$	10^{-4}
set3	0.82	$15 * 10^{-6}$	$18 * 10^{-4}$
set4	2.007	$17 * 10^{-8}$	$9 * 10^{-4}$
set5	0.03	$9 * 10^{-6}$	$4,7 * 10^{-4}$
set6	0.17	$23 * 10^{-6}$	10^{-3}
set7	1.14	$74 * 10^{-7}$	$12,3 * 10^{-4}$
set8	0.626	$33 * 10^{-6}$	$11.98 * 10^{-4}$
set9	0.333	$12 * 10^{-6}$	$3.2 * 10^{-4}$
set10	0.507	$16.3 * 10^{-7}$	$6.81 * 10^{-4}$

Set/Error	Max Testing Er	Min Testing Er	Average Testing Er
set1	12,01	0, 12	6.16
set2	22	1.87	6.53
set3	15.74	0.034	5.25
set4	18.27	0.078	5.89
set5	9.87	0.14	5.93
set6	14.076	0.002	6.01
set7	18.48	0.08	5.78
set8	10	0.15	4.99
set9	15.56	0.04	6.24
set10	12.84	0.105	5.9

.

We can see below a picture that Matlab code produces during the training and shows the training error.

4.2.3 Neural networks vs Non-Linear programming

We saw that non-linear programming helps us to solve the collision cases. The solutions we got were used in order to train the neural network. Actually the solutions were the desired outputs of the neural network known as targets. The basic criterion for us to evaluate the two methods is the execution time and the optimum velocity changes. We can compute in GAMS tool the execution time by a file that is produced with the name file.lst. In MATLAB tool we can compute the execution time with the help of two commands named tic and toc. Tic command is placed at the begging of the code that we want to examine and toc at the end. It is noticed that the GAMS code has execution time 0.012 while matlab code for an already trained neural network

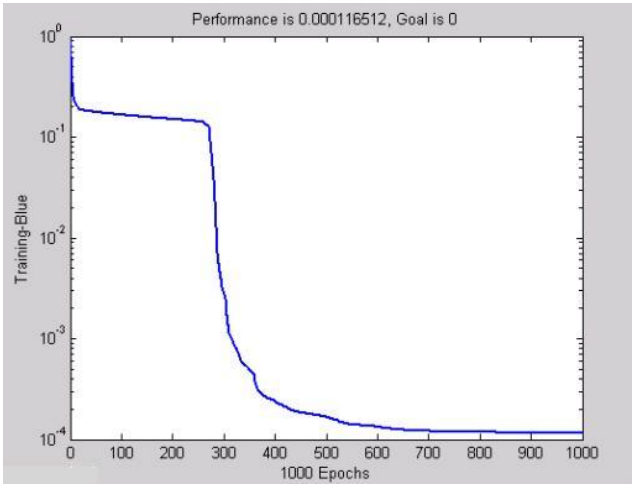

Figure 4.11: Error through training in function with the epochs

has execution time 2.5340 sec. Obviously the GAMS code performed better than the MATLAB, but this is not the best way to solve collision cases. First of all GAMS code gives solutions only for one case in each pass while MATLAB can give solutions for hundred of cases simultaneously. In this point of view MATLAB code prevails. Furthermore it is good to mention that GAMS gives the optimum velocities, while MATLAB gives solutions approaching the optimum. It is desirable at the most of the cases, as it was explained previously, to have almost optimum solutions. To sum up MATLAB code is offering a good response time when we are dealing with lots of aircraft but GAMS tool is definitely the best way to calculate solutions for only two aircrafts. It is up to us which of the two methods to select.

4.3 Appendix A-Codes

4.3.1 Matlab Code-Representation of the problem

```
number=1;
 rand('state',sum(100*clock));
while number<2000,
v=[15,15];

tslice=0.5;
d=9;
clear collout;
clear coll phi1 theta1 phi2 phi5 theta2 theta5 phi3 theta3 dist distance;
clear x R;
clear y z;
clear X Y Z;
close all; % close any opened figures
clc;
n=max(size(v));
opengl autoselect
R=108;
phis=linspace(0,2*pi,30)';
thetas=linspace(0,2*pi,30)';
for N=1:1:size(phis,1)
    for M=1:1:size(thetas,1)
        xs(N,M)=R*sin(phis(N))*cos(thetas(M));
        ys(N,M)=R*sin(phis(N))*sin(thetas(M));
        zs(N,M)=R*cos(phis(N));
    end
end
figure
surf(xs,ys,zs)
hold on
axis equal
shading interp
colormap('gray')
alpha(0.1)
caxis([R+100,R+101]);
lighting gouraud
camlight left
%-----------------------------------------------
```

```
X=[];
Y=[];
Z=[];



collout=[];

    counter=0;
while size(collout)==0,

  counter=counter+1;
    if(counter==20)
        break;
    end


 tyxaioi1=rand(1,8);
phi1=2*pi*tyxaioi1(1,1);
theta1=2*pi*tyxaioi1(1,2);
   % shmeia eisodou 1ou aeroplanou  (1h sthlh)

   x(1,1)=R*sin(phi1)*cos(theta1);
    y(1,1)=R*sin(phi1)*sin(theta1);
    z(1,1)=R*cos(phi1);

   phi3=2*pi*tyxaioi1(1,3);
theta3=2*pi*tyxaioi1(1,4);

% shmeia eksodou 1ou aeroplanou  (1h sthlh)

x(2,1)=R*sin(phi3)*cos(theta3);
    y(2,1)=R*sin(phi3)*sin(theta3);
    z(2,1)=R*cos(phi3);

  distance=sqrt((x(2,1)-x(1,1))^2+(y(2,1)-y(1,1))^2+(z(2,1)-z(1,1))^2);
```

```
  if distance<d
      counter=0;
      continue;
  end

phi2=2*pi*tyxaioi1(1,5);
 theta2=2*pi*tyxaioi1(1,6);
 theta5=2*pi*tyxaioi1(1,7);
 phi5=2*pi*tyxaioi1(1,8);

 % shmeia eksodou 1ou aeroplanou (2h sthlh)
    x(1,2)=R*sin(phi5)*cos(theta5);
    y(1,2)=R*sin(phi5)*sin(theta5);
    z(1,2)=R*cos(phi5);

    % shmeia eksodou 2ou aeroplanou (2h sthlh)
   x(2,2)=R*sin(phi2)*cos(theta2);
    y(2,2)=R*sin(phi2)*sin(theta2);
    z(2,2)=R*cos(phi2);

     % ypologismos twn A

 for N=1:1:2
    A1(N,1)=x(N,2)-x(N,1);
    A2(N,1)=y(N,2)-y(N,1);
    A3(N,1)=z(N,2)-z(N,1);
    A(N,1)=sqrt(A1(N,1)^2+A2(N,1)^2+A3(N,1)^2);

    %xronoi tou kathe aeroplanou
    T(N,1)=A(N,1)/v(1,N);

 end

%Tmax=max(T);
Tmax=max(T);
t=0:tslice:Tmax;
t=t';
X=[];
Y=[];
Z=[];
for N=1:1:n
```

```
    for M=1:1:size(t,1)
        X(M,N)=x(N,1)+(A1(N,1)*v(N)/A(N,1))*t(M,1);
        Y(M,N)=y(N,1)+(A2(N,1)*v(N)/A(N,1))*t(M,1);
        Z(M,N)=z(N,1)+(A3(N,1)*v(N)/A(N,1))*t(M,1);
    end
end


tot=factorial(n)/(factorial(n-2)*factorial(2));
coll=[];
for N=1:1:size(X,1)

    TESTER=0;
    Pair=[0,0];
    for M=1:1:size(X,2)

    if sqrt((X(N,M)-x(M,1))^2+(Y(N,M)-y(M,1))^2+(Z(N,M)-z(M,1))^2)<A(M,1)
      if N==1 | N==size(X,1)
       plot3(X(N,M),Y(N,M),Z(N,M),'ro','linewidth',1);
      else
      if M==1
       plot3(X((2:N),M),Y((2:N),M),Z((2:N),M),'g-','linewidth',2);
      else
       plot3(X((2:N),M),Y((2:N),M),Z((2:N),M),'r-','linewidth',2);

      end
    end
    else
     plot3(x(M,2),y(M,2),z(M,2),'ro','linewidth',1);
     end
    end
    I=1;
    J=2;
    for K=1:1:tot
     if J<M

      dist=sqrt((X(N,I)-X(N,J))^2+(Y(N,I)-Y(N,J))^2+(Z(N,I)-Z(N,J))^2);
           if dist<d
              TESTER=1;
              Pair(1,1)=I;
              Pair(1,2)=J;
```

```
            plot3(X(N,I),Y(N,I),Z(N,I),'b*','linewidth',4);
            plot3(X(N,J),Y(N,J),Z(N,J),'b*','linewidth',4);
            if size(coll,2)==0;
               coll=Pair;
            else
            checker=0;
            steper=1;
               for L=1:1:size(coll,2)/2
                  if Pair(1,1)~=coll(1,steper) | Pair(1,2)~=coll(1,steper+1)
                     checker=1;
                  else
                     checker=0;
                  end
                  steper=steper+2;
               end
               if checker==1
                  coll(1,end+1)=Pair(1,1);
                  coll(1,end+1)=Pair(1,2);
               end
            end
         end
         J=J+1;
      else
         dist=sqrt((X(N,I)-X(N,J))^2+(Y(N,I)-Y(N,J))^2+(Z(N,I)-Z(N,J))^2);
         if dist<d
            TESTER=1;
            Pair(1,1)=I;
            Pair(1,2)=J;
            plot3(X(N,I),Y(N,I),Z(N,I),'b*','linewidth',4);
            plot3(X(N,J),Y(N,J),Z(N,J),'b*','linewidth',4);
            if size(coll,2)==0;
               coll=Pair;
            else
            checker=0;
            steper=1;
               for L=1:1:size(coll,2)/2
                  if Pair(1,1)~=coll(1,steper) | Pair(1,2)~=coll(1,steper+1)
                     checker=1;
                  else
                     checker=0;
                  end
```

```
                    steper=steper+2;
                end
                if checker==1
                    coll(1,end+1)=Pair(1,1);
                    coll(1,end+1)=Pair(1,2);
                end
            end
        end
        I=I+1;
        J=I+1;
    end
end
pause(0.1);
   Mo(N)=getframe;
end
xlabel('x')
ylabel('y')
zlabel('z')
collout=[];
steper=1;
for N=1:1:size(coll,2)/2
    collout(1,steper)=coll(1,steper);
    collout(1,steper+1)=coll(1,steper+1);
    collout(2,steper)=x(coll(1,steper),1);
    collout(2,steper+1)=x(coll(1,steper+1),1);
    collout(3,steper)=y(coll(1,steper),1);
    collout(3,steper+1)=y(coll(1,steper+1),1);
    collout(4,steper)=z(coll(1,steper),1);
    collout(4,steper+1)=z(coll(1,steper+1),1);
    collout(5,steper)=x(coll(1,steper),2);
    collout(5,steper+1)=x(coll(1,steper+1),2);
    collout(6,steper)=y(coll(1,steper),2);
    collout(6,steper+1)=y(coll(1,steper+1),2);
    collout(7,steper)=z(coll(1,steper),2);
    collout(7,steper+1)=z(coll(1,steper+1),2);
    steper=steper+2;
end
intactout=[];
C=1;
for N=1:1:size(x,1)
    checker=0;
```

```
    for M=1:1:size(coll,1)
        for K=1:1:size(coll,2)
            if N==coll(M,K)
                checker=1;
            end
        end
    end
    if checker==0
       intactout(1,C)=N;
       intactout(2,C)=x(N,1);
       intactout(3,C)=y(N,1);
       intactout(4,C)=z(N,1);
       intactout(5,C)=x(N,2);
       intactout(6,C)=y(N,2);
       intactout(7,C)=z(N,2);
       C=C+1;
    end

end
end
    if counter<20
    f=num2str(number);
    d=['collout' f '.txt'];
    e=['angles' f '.txt']

     angles(1,1)=phi1;
     angles(1,2)=phi5;
     angles(1,3)=theta1;
     angles(1,4)=theta5;
     angles(2,1)=phi3;
     angles(2,2)=phi2;
     angles(2,3)=theta3;
     angles(2,4)=theta2;
    dlmwrite(d,collout,'\t')
    dlmwrite(e,angles,'\t')
    dlmwrite('intactout.txt',intactout,'\t')
    end
    if counter<20
    number=number+1;
    end
end
```

4.3.2 Gams-Conflict resolution

```
*Number of aircraft
Set i   "aircraft"         /1 * 2/;
Set t   "timeslice"         /1 * 50/;
alias (i,j);

Parameters msd,A1(i),A2(i),A3(i),distance(i);
Parameters count(i)
/ 1 1
  2 2
   /;

*x-coordinates in Km
Parameters x(i)

/1 -43.467
 2 27.736/;




*y-coordinates in Km
Parameters y(i)




/1 -9.018
2 -40.555/;


*z-coordinates in Km
Parameters z(i)
```

```
/1 98.455
2 96.177/;


*initial heading angles in rads
Parameters theta(i)



/1 0.205
2 5.312   /;




*initial heading angles in rads
Parameters phi(i)



/1 5.860
2 0.472/;




*initial velocities in Km/min
Parameters u(i)
/ 1 15
  2 15

/;

Parameters xfin(i)
```

```
/1 34.039
2 -13.797/;



Parameters yfin(i)



/1 -49.472
2 -25.928/;



Parameters zfin(i)



/1 89.766
2 103.930/;



Parameters number(t)
/1   1
 2   2
```

3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25
26	26
27	27
28	28
29	29
30	30
31	31
32	32
33	33
34	34
35	35
36	36
37	37
38	38
39	39
40	40
41	41
42	42
43	43

```
44   44
45   45
46   46
47   47
48   48
49   49
50   50
 /;

*minimum safe distance in Km
msd=9;


A1(i)=xfin(i)-x(i);
A2(i)=yfin(i)-y(i);
A3(i)=zfin(i)-z(i);

d(i)=sqrt( A1(i)*A1(i)+A2(i)*A2(i)+A3(i)*A3(i)  );



Variable f,q(i) ;


q.up(i)=0.99;
q.lo(i)=-0.9;


Equations
metabolh,velocity1(i),velocity2(i),dis(i,j,t);



metabolh..     f=e=sum(i,sqr(q(i)));



velocity1(i)..  u(i)+q(i)       =l=15.99;
velocity2(i)..  u(i)+q(i)       =g=14.1;
```

```
dis(i,j,t).. (sqrt
(sqr(
((x(j)+(A1(j)/distance(j))*(u(j)+q(j))*0.5*number(t))
- (x(i)+(A1(i)/d(i))*(u(i)+q(i))*0.5*number(t)))
)
+sqr(
((y(j)+(A2(j)/d(j))*(u(j)+q(j))*0.5*number(t))
- (y(i)+(A2(i)/d(i))*(u(i)+q(i))*0.5*number(t)))
)
+sqr(
((z(j)+(A3(j)/d(j))*(u(j)+q(j))*0.5*number(t))
- (z(i)+(A3(i)/d(i))*(u(i)+q(i))*0.5*number(t))
)
)

)-9.01  ) $ (count(i)<count(j))=g=0;
```

```
Model nlc /all/ ;
option domlim=10;
option sysout=on;
option nlp=conopt2;
option mip=cplex;
option rminlp=conopt2;
option minlp=dicopt;
option sysout=on;
*
*solve relaxed model
*
*solve nlc using rminlp minimizing f;


solve nlc using nlp minimizing f;
```

4.3.3 Matlab Code Neural Network

```
close all;
clear all;
load train3.txt;

num_of_data=3596;
nl1=55;
nl2=45;
%nl3=20;

input=train3(1:num_of_data,1:12);
target=train3(1:num_of_data,13:14);

for i=1:12
 PR(i,1)=min(input(:,i));
 PR(i,2)=max(input(:,i));
end
%PR
A=input';
B=target';

net=newff(PR,[nl1 nl2 2],{'tansig'  'tansig' 'tansig' 'tansig' },'trainlm');

net.trainParam.epochs=500;
net.trainParam.show = 1;
net.trainParam.lr = 0.1;
net=train(net,A,B);

y2=sim(net,input');
i=1:num_of_data;
for j=1:num_of_data
    diaf1(j)=B(1,j)-y2(1,j);
    diaf2(j)=B(2,j)-y2(2,j);
    adiaf1(j)=abs(diaf1(j));
    adiaf2(j)=abs(diaf2(j));
    ades1(j)=abs(B(1,j));
    ades2(j)=abs(B(2,j));

    training_error(j)=(max( adiaf1(j),adiaf2(j) ))*100;
```

```
end


training_error
sum=0;
for j=1:num_of_data
    sum=sum+training_error(j);
end
av_training_error=sum/num_of_data;

max_training_error=max(training_error);
max_training_error
av_training_error

testing_data=500;
input1=train3(num_of_data+1:num_of_data+testing_data,1:12);
target1=train3(num_of_data+1:num_of_data+testing_data,13:14);

y3=sim(net,input1');
k=1:testing_data;
%plot(k,target1','o',k,y3,'*')

 C=input1';
D=target1';

 for j=1:testing_data
    testing_error(j)=(max   (( abs(   D(1,j)-y3(1,j) ) ) , (abs( D(2,j)-y3(2,j) )))
 end
    max_testing_error=max(testing_error);

%testing_error
max_testing_error
sum=0;
for j=1:testing_data
    sum=sum+testing_error(j);
end

av_testing_error=sum/testing_data;

av_testing_error
```

```
dlmwrite('err.txt',testing_error,'\t')
dlmwrite('iw.txt',net.IW{1,1},'\t')
dlmwrite('b.txt',net.b,'\t')
dlmwrite('lw21.txt',net.LW{2,1},'\t')
dlmwrite('lw32.txt',net.LW{3,2},'\t')
```

4.3.4 Side programs

Write the Gams file

```
#include <iostream.h>
#include <string.h>
#include <stdio.h>

int  main()
{
int num1,num2,local,mod;
char file1[15],temp[4];
char file2[15],file3[15];
    float xin1,yin1,zin1,theta1,phi1,xin2,yin2,zin2;
    float xout2,yout2,zout2,xout1,yout1,zout1,theta2,phi2,local1,dv1,dv2=0.0;
FILE *fp1;
FILE *fp2;
FILE *fp3;
cout<<"Enter the first number:";

cin>>num1;
cout<<endl<<"Enter the second number:";
cin>>num2;
strcpy(file1,"");
strcpy(file2,"");
strcpy(file3,"");
strcpy(temp,"");


for(int i=num1;i<=num2;i++)
{

      strcat(file1,"collout");
  strcat(file2,"angles");
```

```
  strcat(file3,"testing");
     if(i<10)
     {
 temp[0]=i+48;
 temp[1]='\0';

 }
 if(i>9 && i<100)
 {
   temp[0]=(i/10)+48;
   temp[1]=(i % 10)+48;
   temp[2]='\0';

 }
 if(i>99)
 {
  temp[0]=(i/100)+48;
  mod=i%100;
  temp[1]=(mod/10)+48;
  temp[2]=(mod % 10)+48;
  temp[3]='\0';
 }
  strcat(file1,temp);
 strcat(file1,".txt");

strcat(file2,temp);
strcat(file3,temp);
 strcat(file2,".txt");
  strcat(file3,".gms");

   if((fp1=fopen(file1,"r"))==NULL)
      cout<<"Error opening file";
    fseek( fp1, 0L, SEEK_SET );

    fscanf(fp1,"%d",&local);
    fscanf(fp1,"%d",&local);
    fscanf(fp1,"%f",&xin1);

    fscanf(fp1,"%f",&xin2);

     fscanf(fp1,"%f",&yin1);
```

```
fscanf(fp1,"%f",&yin2);

   fscanf(fp1,"%f",&zin1);

fscanf(fp1,"%f",&zin2);



fscanf(fp1,"%f",&xout1);

fscanf(fp1,"%f",&xout2);

 fscanf(fp1,"%f",&yout1);

fscanf(fp1,"%f",&yout2);

   fscanf(fp1,"%f",&zout1);

fscanf(fp1,"%f",&zout2);




 fclose(fp1);

fp2=fopen(file2,"r");
fseek( fp2, 0L, SEEK_SET );
fscanf(fp2,"%f",&phi1);

fscanf(fp2,"%f",&local1);
fscanf(fp2,"%f",&theta1);

fscanf(fp2,"%f",&local1);
fscanf(fp2,"%f",&phi2);

fscanf(fp2,"%f",&local1);
fscanf(fp2,"%f",&theta2);
```

```
 fclose(fp2);

fp3=fopen(file3,"r+");
fseek( fp3, 245L, SEEK_SET );
fprintf(fp3,"/1 ");
fprintf(fp3,"%3.3f\n",xin1);

fprintf(fp3,"2 ");
fprintf(fp3,"%3.3f",xin2);
fprintf(fp3,"/;");



fseek( fp3, 60L, SEEK_CUR);
fprintf(fp3,"/1 ");
fprintf(fp3,"%3.3f\n",yin1);

fprintf(fp3,"2 ");
fprintf(fp3,"%3.3f",yin2);
 fprintf(fp3,"/;");


 fseek( fp3, 55L, SEEK_CUR);
fprintf(fp3,"/1 ");
fprintf(fp3,"%3.3f\n",zin1);

fprintf(fp3,"2 ");
fprintf(fp3,"%3.3f",zin2);
fprintf(fp3,"/;");

 fseek( fp3, 70L, SEEK_CUR);
fprintf(fp3,"/1 ");
fprintf(fp3,"%3.3f\n",theta1);

fprintf(fp3,"2 ");
fprintf(fp3,"%3.3f   ",theta2);
fprintf(fp3,"/;");

  fseek( fp3, 70L, SEEK_CUR);
  fprintf(fp3,"/1 ");
  fprintf(fp3,"%3.3f\n",phi1);
```

```
      fprintf(fp3,"2 ");
      fprintf(fp3,"%3.3f",phi2);
      fprintf(fp3,"/;");



    fseek( fp3, 110L, SEEK_CUR );
    fprintf(fp3,"/1 ");
    fprintf(fp3,"%3.3f\n",xout1);
    fprintf(fp3,"2 ");
    fprintf(fp3,"%3.3f",xout2);
    fprintf(fp3,"/;");



    fseek( fp3, 42L, SEEK_CUR);
    fprintf(fp3,"/1 ");
    fprintf(fp3,"%3.3f\n",yout1);
    fprintf(fp3,"2 ");
    fprintf(fp3,"%3.3f",yout2);
     fprintf(fp3,"/;");


     fseek( fp3, 42L, SEEK_CUR);
    fprintf(fp3,"/1 ");
    fprintf(fp3,"%3.3f\n",zout1);
    fprintf(fp3,"2 ");
    fprintf(fp3,"%3.3f",zout2);
   fprintf(fp3,"/;");


   fclose(fp3);
     strcpy(file1,"");
strcpy(file2,"");
strcpy(temp,"");
strcpy(file3,"");
}



return 0;
```

```
}
```

Write the solution files

```
#include <iostream.h>
#include <string.h>
#include <stdio.h>


int  main()
{
int num1,num2,flag,counter,mod;
char local;
char file1[15],temp[4];
char file2[15];
char str[50]="";
float sol1,sol2,sol3;
FILE *fp1;
FILE *fp2;

cout<<"Enter the first number:";

cin>>num1;
cout<<endl<<"Enter the second number:";
cin>>num2;


strcpy(file1,"");
strcpy(file2,"");

counter=0;
for(int i=num1;i<=num2;i++)
{
flag=1;
strcpy(file1,"");
strcpy(file2,"");
strcpy(temp,"");
strcpy(str,"");
sol1=sol2=sol3=0.0;
strcat(file1,"testing");
strcat(file2,"sol");
```

```
 if(i<10)
      {
  temp[0]=i+48;
  temp[1]='\0';

  }
  if(i>9 && i<100)
  {
    temp[0]=(i/10)+48;
    temp[1]=(i % 10)+48;
    temp[2]='\0';

  }
  if(i>99)
  {
   temp[0]=(i/100)+48;
   mod=i%100;
   temp[1]=(mod/10)+48;
   temp[2]=(mod % 10)+48;
   temp[3]='\0';
  }
strcat(file1,temp);
strcat(file1,".lst");
strcat(file2,temp);
strcat(file2,".txt");
 if((fp1=fopen(file1,"r"))==NULL)
       cout<<"Error opening file";
 fseek( fp1, 0L, SEEK_SET );
strcpy(str," ");



while(strcmp(str,"VAR"))
{

fscanf(fp1,"%s",str);

}
```

```
for(int k=0;k<14;k++)
{
  fscanf(fp1,"%s",str);

}

fscanf(fp1,"%s",str);
if(str[0]=='.')
{

   flag=0;
   fp2=fopen(file2,"w");
   fprintf(fp2,"0");
   fclose(fp2);
   continue;
}

if((!strcmp(str,"-0.900")) ||  (!strcmp(str,"0.990")) )
{

  flag=0;
  sol1=atof(str);
}


  sol1=atof(str);
for(int k=0;k<4;k++)
  fscanf(fp1,"%s",str);

fscanf(fp1,"%s",str);
if (flag==0)
{

if((strcmp(str,"0.990" )) &&  (strcmp(str,"-0.900" )))
{


   flag=1;

}
else
```

```
{
  fp2=fopen(file2,"w");
   fprintf(fp2,"0");
   fclose(fp2);
  continue;

}

}

sol2=atof(str);

fclose(fp1);



if(flag==1)
{
counter++;
fp2=fopen(file2,"w");

fprintf(fp2,"%f   ",sol1);
fprintf(fp2,"%f",sol2);
fclose(fp2);
}
strcpy(file1,"");
strcpy(file2,"");
strcpy(temp,"");
strcpy(str,"");
sol1=sol2=sol3=0.0;
}
cout<<endl;
cout<<"*****************************************\n";
cout<<counter<<" "<<"situations were solved"<<endl;
cout<<"Press any key and enter to exit:";

cin>>local;


}
```

Write the training file

```
#include <iostream.h>
#include <string.h>
#include <stdio.h>

int  main()
{
int num1,num2,counter,local,mod;
char in;
char file1[15],temp[4];
char file2[15],file3[15];
    float xin1,yin1,zin1,xout1,zout1,yout1,xin2,yin2,zin2,xout2,yout2,zout2
    float local1,dv1,dv2=0.0;
float phi1,phi2,theta1,theta2;
FILE *fp1;
FILE *fp2;
FILE *fp3;
FILE *fp4;
cout<<"Enter the first number:";

cin>>num1;
cout<<endl<<"Enter the second number:";
cin>>num2;
strcpy(file1,"");
strcpy(file2,"");
strcpy(file3,"");
strcpy(temp,"");

fp3=fopen("train3.txt","a");
counter=0;
for(int i=num1;i<=num2;i++)
{
  strcpy(file1,"");
strcpy(file2,"");
strcpy(temp,"");
strcpy(file3,"");
      strcat(file1,"collout");
 strcat(file2,"angles");
   strcat(file3,"sol");
      if(i<10)
```

```
      {
  temp[0]=i+48;
  temp[1]='\0';

  }
  if(i>9 && i<100)
  {
    temp[0]=(i/10)+48;
    temp[1]=(i % 10)+48;
    temp[2]='\0';

  }
  if(i>99)
  {
   temp[0]=(i/100)+48;
   mod=i%100;
   temp[1]=(mod/10)+48;
   temp[2]=(mod % 10)+48;
   temp[3]='\0';
  }
   strcat(file1,temp);
  strcat(file1,".txt");
 // cout<<file1<<endl;
 strcat(file2,temp);

 strcat(file3,temp);

  strcat(file2,".txt");

   strcat(file3,".txt");


fp4=fopen(file3,"r");
     fscanf(fp4,"%f",&dv1);

     if(dv1==0)
     {
       fclose(fp4);
       continue;

     }
```

```
 fscanf(fp4,"%f",&dv2);

 fclose(fp4);
if((fp1=fopen(file1,"r"))==NULL)
   cout<<"Error opening file";
 fseek( fp1, 0L, SEEK_SET );

 fscanf(fp1,"%d",&local);
 fscanf(fp1,"%d",&local);

 fscanf(fp1,"%f",&xin1);
 fscanf(fp1,"%f",&xin2);
 fscanf(fp1,"%f",&yin1);
 fscanf(fp1,"%f",&yin2);
    fscanf(fp1,"%f",&zin1);

 fscanf(fp1,"%f",&zin2);
 fscanf(fp1,"%f",&xout1);
 fscanf(fp1,"%f",&xout2);
 fscanf(fp1,"%f",&yout1);
 fscanf(fp1,"%f",&yout2);
 fscanf(fp1,"%f",&zout1);
 fscanf(fp1,"%f",&zout2);

  fclose(fp1);

 if((fp2=fopen(file2,"r"))==NULL)
   cout<<"Error opening file";
 fseek( fp2, 0L, SEEK_SET );
 fscanf(fp2,"%f",&phi1);
 fscanf(fp2,"%f",&local);
 fscanf(fp2,"%f",&theta1);
 fscanf(fp2,"%f",&local);

 fscanf(fp2,"%f",&phi2);
 fscanf(fp2,"%f",&local);
 fscanf(fp2,"%f",&theta2);
 fscanf(fp2,"%f",&local);
 fclose(fp2);
```

```
     fprintf(fp3,"%3.3f   ",xin1);
fprintf(fp3,"%3.3f   ",yin1);
fprintf(fp3,"%3.3f   ",zin1);
fprintf(fp3,"%3.3f   ",phi1);
 fprintf(fp3,"%3.3f   ",xout1);

     fprintf(fp3,"%3.3f   ",yout1);
     fprintf(fp3,"%3.3f   ",yout2);
     fprintf(fp3,"%3.3f   ",zout1);



fprintf(fp3,"15   ");
fprintf(fp3,"%3.3f   ",xin2);
fprintf(fp3,"%3.3f   ",yin2);
fprintf(fp3,"%3.3f   ",zin2);
fprintf(fp3,"%3.3f   ",xout2);
fprintf(fp3,"%3.3f   ",yout2);
fprintf(fp3,"%3.3f   ",zout2);

fprintf(fp3,"15   ");


fprintf(fp3,"%3.3f   ",dv1);
fprintf(fp3,"%3.3f   ",dv2);
fprintf(fp3,"\n");
    counter++;


     strcpy(file1,"");
strcpy(file2,"");
strcpy(temp,"");
strcpy(file3,"");
}




fclose(fp3);
cout<<endl;
cout<<"*****************************************\n";
```

```
cout<<counter<<" "<<"situations were added to train.txt"<<endl;
cout<<"Press any key and enter to exit:";

cin>>in;
return 0;
}
```

Bibliography

[1] http://www.aiaa.org

[2] M.A.Christodoulou and S.G. Kodaxakis,"Automatic Commercial Aircraft Collision Avoidance in Free Flight: The Three Dimensional Problem" , *IEEE Transactions on Intelligent Transportation Systems*, vol.7, No.2, pp. 242-249, 2006

[3] Ben Krose, Patrick van der Smagt, *An introduction to Neural Networks* Eighth Edition, chapter2 p16-18, 1996

[4] Gams site: www.gams.com

[5] www.faa.gov

[6] Krozel, J.Peters, M.Seagull Techno, "Strategic conflict detection and resolution for free flight", Decision and Control, *Proceedings of the 36th IEEE Conference*, 10-12-07

[7] I.Kodaxakis, *Free Flight in Commercial aircraft: A new algorithm for automatic conflict avoidance in 3-D space*, Masters Thesis, Department of Electronic and Computer Engineering, Technical University of Crete, 2005

[8] Merril Skolnik, *Introduction to radar systems*, McGraw-Hill Company,INC, Tokyo

[9] http://ams.allenpress.com

[10] http://md1.csa.com/partners/viewrecord.php

[11] Earl W.Swokowski, *Calculus with analytic Geometry second edition*,chapter 14 p.680

[12] C.A.Floudas, *Non-Linear and Mixed-Integer Optimization*, Eighth Edition, 1996

[13] Richard E.Rosenthal, *A Gams tutorial*

[14] D.P.Bertsekas, *Non-Linear Programming*, Boston M: Athena Sientific, 2005

[15] Ben Krose, Patrick van der Smagt, *An introduction to Neural Networks*, Eighth Edition, chapter2 p20, 1996

[16] Ben Krose, Patrick van der Smagt, *An introduction to Neural Networks*, Eighth Edition, chapter3 p23-24, 1996

[17] Gill P. R., Murray W. and Wright M. H., *The Levenberg-Marquardt Method*, chapter4.7.3 in Practical Optimization, London: Academic Press, pp. 136-137, 1981.