國立雲林科技大學電機工程系

獎　狀

(111)雲科大電機字第 0056 號

本系 翟侑群、李佳衡 同學，參加 111 學年度實務專題-自動化與系統控制組競賽分組，以『船艦目標運動分析演算法及人機介面』獲得完成證明。

特頒此狀 以資鼓勵

電機工程系主任 曾萬存

中 華 民 國 111 年 11 月 23 日

YunTech

第一章 緒論

1.1 動機與研究目的

隨著各國軍事科技日新月異，我國理所當然不能落後他國，夙夜匪懈的強化自我，從過往到現代，戰爭的形態已經由冷兵器戰鬥演變至科技戰，而範圍也觸及陸海空三方面，而台灣四面環海，再加上間隔台灣海峽有著對面中國的長期武力威脅，使得我國必須增加海軍的防衛實力，而政府的國艦國造計畫，大力研發屬於我國的潛艦，本組員們在有著同島一命使命在我，為了保家衛國的國防事業出一份心力，致力於研究潛艦雷達應用之開發，所以我們決定要使用撰寫程式的方式來模擬潛艦的雷達系統，傳統艦船雷達是透過經驗老到的雷達手不斷聽聲納來判斷目標船艦之方位、航向後推演速度與航向角，有感於經驗傳承之不易，真正厲害的雷達手人力不足，若能透過電腦來幫助估測，必能更早的洞悉先機

1.2 研究方法

本次專題利用美國The MathWork公司出品之Matlab軟體強大的數學運算能力來研究演算法與開發，之後將演算法轉移至Microsoft公司所推出之Visual Studio軟體環境下模擬目標艦船的運動模式，以及利用事先做好方便使用者操作之人機介面觀察模擬結果。本次研究選用之高斯牛頓迭代法可視作本次演算法的主軸，其基本構想是使用泰勒級數展開式去近似地代替非線性迴歸模型，然後通過多次迭代多次修正迴歸係數，使迴歸係數不斷通過逼近非線性迴歸模型的最佳迴歸係數，最後使原模型的殘差平方和達到最小，本次研究透過此特性，先藉由雷達偵測到的位置，不斷推演我方欲獲取之目標船艦後續資訊，最後得到目標之實際參數

第二章 演算法與船艦運動模擬之原理

2.1 Gauss-Newton Algorithm

本次研究使用Gauss-Newton法搭配Jacobian矩陣和三角函數的特性來開發演算法，選擇使用Gauss-Newton法的契機是有鑑於機器從現實面獲得之輸入通常為類比訊號，其與輸出數值的關係會為非線性，得到精確解非常困難，只能求出近似解，而近似解的最佳求法大多是使用逐次線搜索的迭代法，反覆迭代直到滿足預定的精度要求。相較牛頓法收斂不穩定且計算困難，高斯牛頓法捨棄黑森矩陣的二階偏導數，直接利用在計算梯度 ∇f 時已得到J(x)簡化了計算，之後不斷迭帶以獲得最接近現實面目標船艦之數值，搭配三角函數可進行航速、方向角與位置的計算，最後透過隨機模擬目標船艦之運動來檢視演算法的實際可行性。

$$y_i^{(0)} \approx \sum_{k=0}^{p-1} D_{ik}^{(0)} b_k^{(0)} + \varepsilon_i \qquad (i=1, 2, \cdots, n)$$

$$\boldsymbol{Y}^{(0)} \approx \boldsymbol{D}^{(0)} \boldsymbol{b}^{(0)} + \boldsymbol{\varepsilon}$$

$$\boldsymbol{Y}_{n\times p}^{(0)} = \begin{bmatrix} y_1 - f(x_1, g^{(0)}) \\ \vdots \\ y_n - f(x_n, g^{(0)}) \end{bmatrix}, \boldsymbol{D}_{n\times p}^{0} = \begin{bmatrix} D_{10}^{(0)} \cdots & D_{1p-1}^{(0)} \\ \vdots & \vdots \\ D_{n0}^{(0)} \cdots & D_{np-1}^{(0)} \end{bmatrix}, \boldsymbol{D}_{p\times 1}^{0} = \begin{bmatrix} b_0^{(0)} \\ \vdots \\ b_{p-1}^{(0)} \end{bmatrix}$$

$$\boldsymbol{b}^{(0)} = (\boldsymbol{D}^{(0)\mathrm{T}} \boldsymbol{D}^{(0)})^{-1} \boldsymbol{D}^{(0)\mathrm{T}} \boldsymbol{Y}^{(0)}$$

$$\mathrm{SSR}^{(s)} = \sum_{i=1}^{n} [y_i - f(x_i, g^{(s)})]^2$$

圖表 1牛頓高斯法簡述圖

2.2 Jacobian Array

在向量分析中，雅各比矩陣是函數的一階偏導數以一定方式排列成矩陣。其重要性在於，如果函數f:R如果函數f:$\mathbb{R}^n \to \mathbb{R}^m$在點X可微分的話，在點X的雅各比矩陣即為該函數在該點的，在點X的雅各比矩陣即為該函數在該點的最佳線性逼近，也代表雅可比矩陣是單變數實數函數的微分在向量值多變數函數的推廣，在這種情況下，雅可比矩陣也被稱作函數 f在點x的微分或者導數假設某函數從 f:$\mathbb{R}^n \to \mathbb{R}^m$，從 X ∈ $\mathbb{R}^n$映射到向量f(x)∈$\mathbb{R}^m$，其雅可比矩陣是一m×n的矩陣，換句話講也就是從$\mathbb{R}^n$到$\mathbb{R}^m$的線性映射，其重要意義在於它表現了一個多變數向量函數的最佳線性逼近。因此，雅可比矩陣類似於單變數函數的導數。此函數f的雅可比矩陣J為m×n的矩陣，一般由以下方式定義：

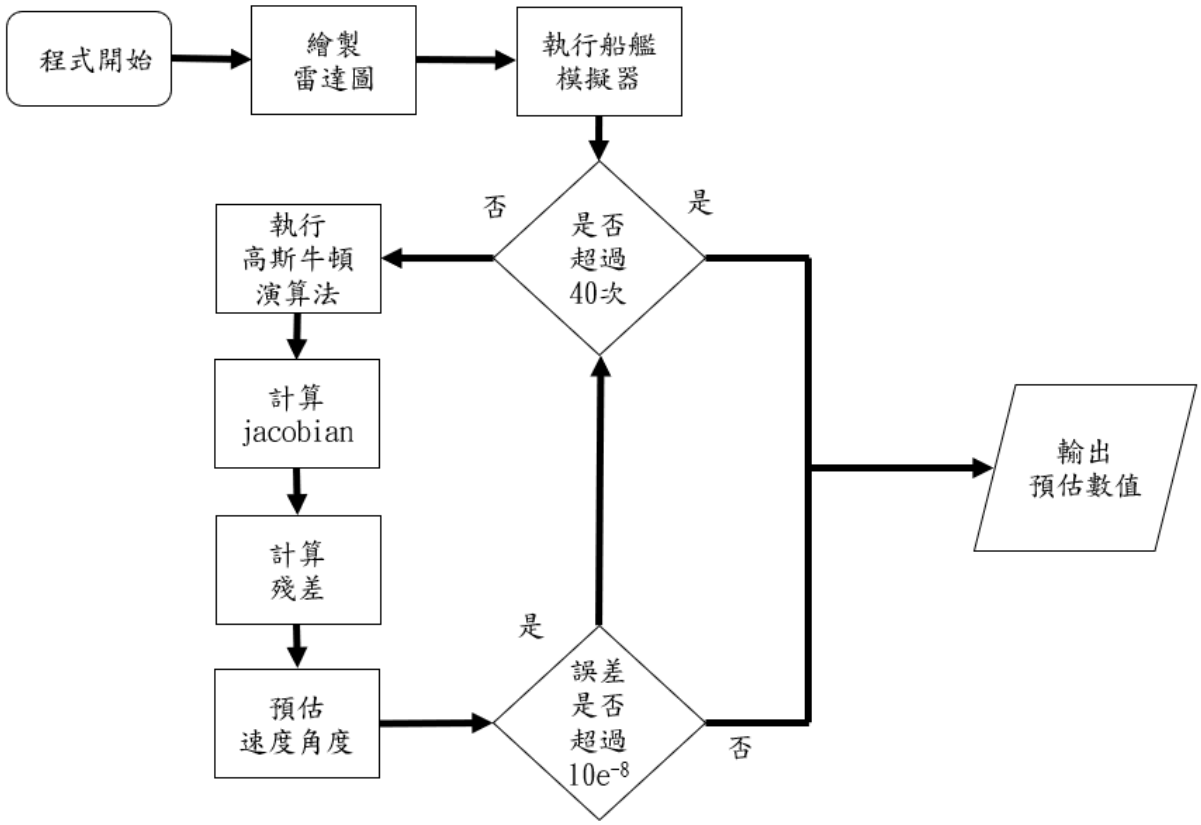
$$\mathbf{J}(x_1, \ldots, x_n) = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_n} \end{bmatrix}.$$

圖表 2雅各比矩陣

第3章　船艦目標運動分析與人機介面實務

在本次實務專題中，我們使用C#來當作使用的工具，藉由C#進行介面的設計及程式的書寫，先進行雷達的繪製、建立船艦模擬器、再進行高斯牛頓法的迴圈運算，得到對對方船艦的預估數值，最後在將運行結果執行在人機介面上。

下圖為本次專題流程圖

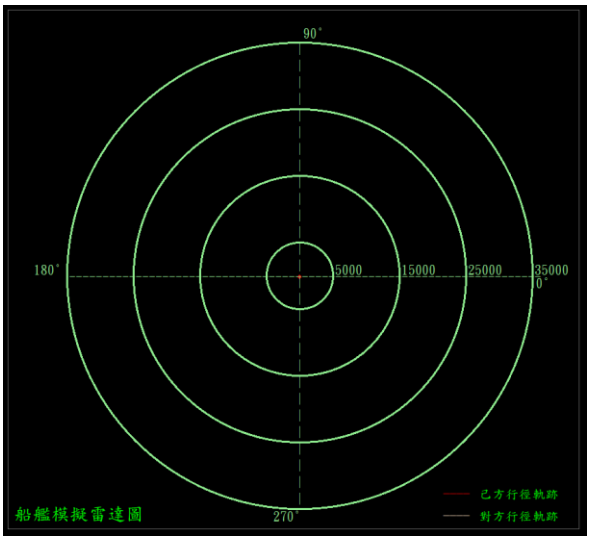


圖表 3流程圖

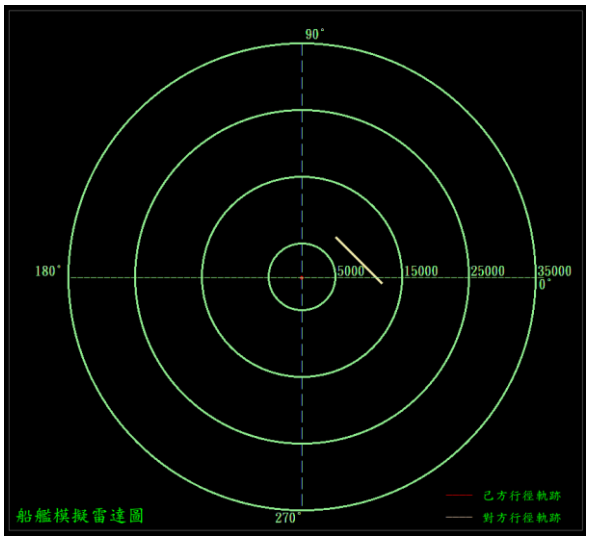
3.1 人機介面開發實務

(一)主介面設計

主介面的設計方向是以黑色基底搭配螢光色的色調，接近現實雷達圖的樣式，左方顯示的圖圈為雷達圖，紅色代表以方之行徑軌跡，米色代表對方之行徑軌跡，利於辨明船艦模擬器中己方和對方的行徑軌跡。



圖表 4主介面



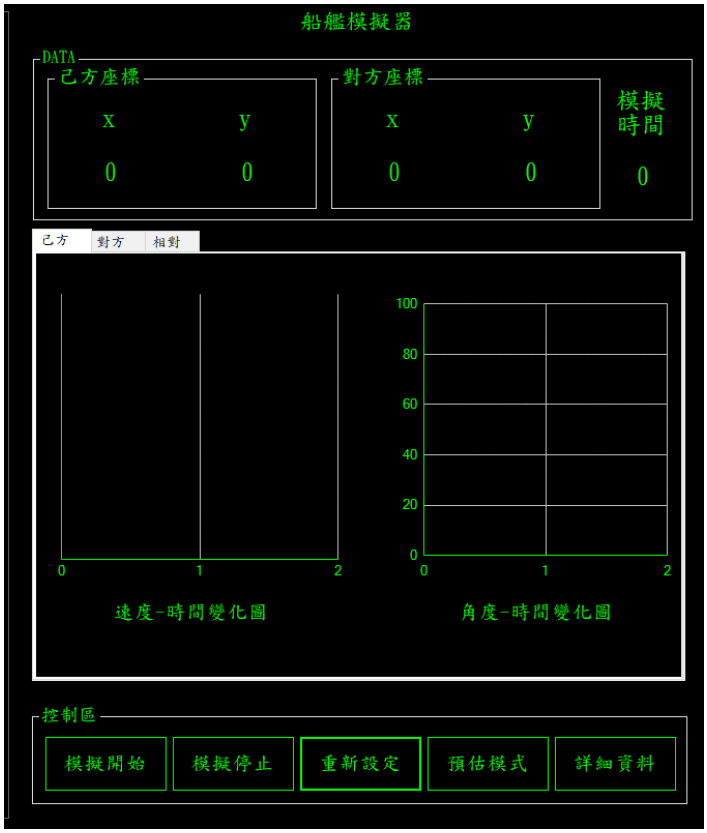
圖表 5主介面執行

(二)控制介面設計

1. 右上方簡易的標示重點資訊座標位置及當前模擬時間。

2. 右中標示己方、對方和相對的速度及角度變化圖，並隨著時間做變化，利用笛卡爾坐標系更利於觀察變化的情形。

3. 右下控制區則共有5顆按鈕，分別控制開始、停止、重設、開啟預估模式及查看更詳細的資料。



圖表 6控制介面

(三)主程式進行

設定變數外，還設定了對方船艦的資料，此項資料為猜測數值，猜測對方的初始位置、速度及方向，所以設定任何值都是可以的，目的是在於給程式初始值，方便程式進行後續運算。

```
x0[0] = initialPose[0] + iniRangEr * Math.Cos(iniBeta); //猜測對方x軸初始位置
double[] y0 = new double[1];
y0[0] = initialPose[1] + iniRangEr * Math.Sin(iniBeta); //猜測對方y軸初始位置
double x0log = x0[0];
double y0log = y0[0];
double v = Math.Sqrt(Math.Pow(initialPose[2], 2) + Math.Pow(initialPose[3], 2)
v = v - 10; //猜測對方初始速度
double theta = Math.Atan(initialPose[3] / initialPose[2]);
theta = theta + 1.5; //猜測對方初始行進方向
```

圖表 7在主程式設定變數

(四)重複進行高斯牛頓法的疊代

Bearlog是相對角度、xown、yown是己方的座標位置，在重複進行疊代之前先對相對角度及位置做取樣，會隨著迴圈次數越多，取樣數據會越多，所預估之值會更加接近實際之值。

```
for (w = 1; w <= N; w++)
{
    a_test[0, w - 1] = a_n[0, 0];
    a_test[1, w - 1] = a_n[1, 0];
    double[] BearData = new double[w * Ndata];
    for (int a = 0; a < w * Ndata; a++)
        BearData[a] = Bearlog[beingM + a * 2];
    for (int a = 0; a < w * Ndata; a++)
        xown[a] = own_xlog[beingM + a * 2];
    for (int a = 0; a < w * Ndata; a++)
        yown[a] = own_ylog[beingM + a * 2];
    if (w >= 2)
        a_n = unknowns;
    GaussNewton(a_n, BearData, x0, y0, beingM, xown, yown);
```

圖表 8執行迴圈

3.2船艦模擬實務

(一)船艦模擬器

意為定義己方以及對方艦船運動情形的模擬器，是實際情況的演示，所以我們可以在此得知從0秒開始，一直到時間結束為止的船艦運動情形，包括座標位置，移動速度，航向角等等，當然也可以得知相對角度、距離等等。提供在預估模式中所需要的資料，預估結束之後再以這些資料加以比對。

```
public void shipMotion_TarOwnturn()
{
    int Ts = 960;
    int dt = 1, i = 0;//, j = 10;
    int n_own = 125, ni_own = 1, n_ship = 45, ni_ship = 5, n = 480, ni = 1;
    double Vel = 20 * 0.5144;
    double Vel_own = 6 * 0.5144;
    //double Vel_ship = 20 * 0.5144;
    double turnRateO = 2 * Math.PI;// 0.0349;
    double turnRate = 1 * Math.PI;// 0.01745;
    simulationTime = 0;
        initialPose[0] = 5000;
        initialPose[1] = 6000;
        initialPose[2] = 7.274714565;//Vel * Math.Cos(-1 / 4 * 180);
        initialPose[3] = -7.274714565;// Vel * Math.Sin(-1 / 4 * 180);
        ship_x = initialPose[0];
        ship_y = initialPose[1];
        shipvel_x = initialPose[2];
        shipvel_y = initialPose[3];
        own_x = 0;
        own_y = 0;
        ownvel_x = Vel_own * Math.Cos(0.5 * Math.PI);
        ownvel_y = Vel_own * Math.Sin(0.5 * Math.PI);
```

圖表 9設定船艦模擬器參數

(二)船艦模擬器運行

用while迴圈模擬船艦運動情形，是以己方不動，對方定速定向的方式進行模擬。

1. 己方不動、對方定速定向駛

```
own_x = 0;// own_x + ownvel_x;
own_y = 0;// own_y + ownvel_y;
ship_x = ship_x + shipvel_x;
ship_y = ship_y + shipvel_y;
```

圖表 10船艦運動模擬

2. 將值儲存進陣列之中，方便在預估模式取值

```
own_xlog[i] = own_x;
own_ylog[i] = own_y;
own_courselog[i] = Math.Atan(ownvel_y / ownvel_x);
ship_xlog[i] = ship_x;
ship_ylog[i] = ship_y;
shipvel_xlog[i] = shipvel_x;
shipvel_ylog[i] = shipvel_y;
ship_courselog[i] = Math.Atan(shipvel_y / shipvel_x);
rel_xlog[i] = rel_x;
rel_ylog[i] = rel_y;
```

圖表 11船艦運動模擬

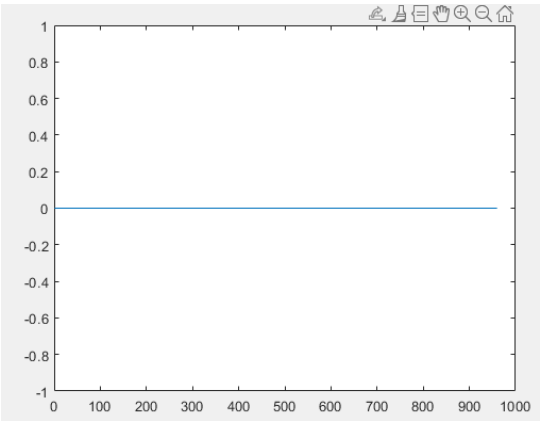
3. 計算相對角度及距離，並使模擬時間+1，重複進行模擬

```
measurement = Math.Atan2(rel_y, rel_x);
Bearlog[i] = measurement;
Ranglog[i] = Math.Sqrt(Math.Pow(rel_x, 2) + Math.Pow(rel_y, 2));
simulationTime++;
i++;
```

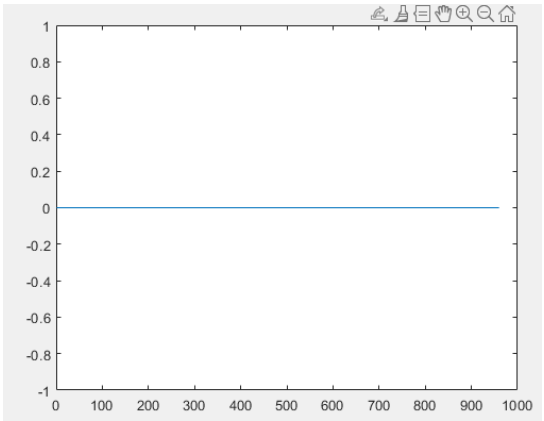
圖表 12船艦運動模擬

(三)以matlab執行之行情

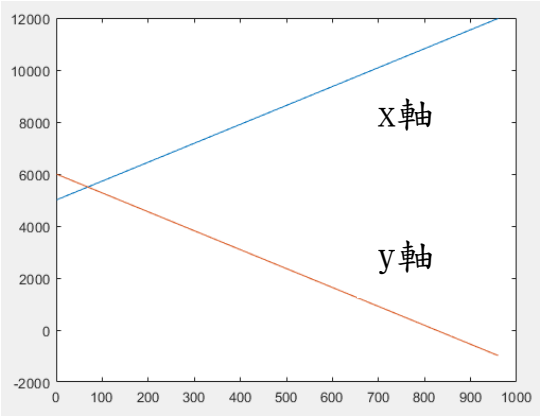
將以上程式用matlab先行做測試，可清楚得知己方與設定值一樣是定點不移動的，對方是以x軸增加、y軸減少的方向進行移動，相對角度及相對距離都有所變化。



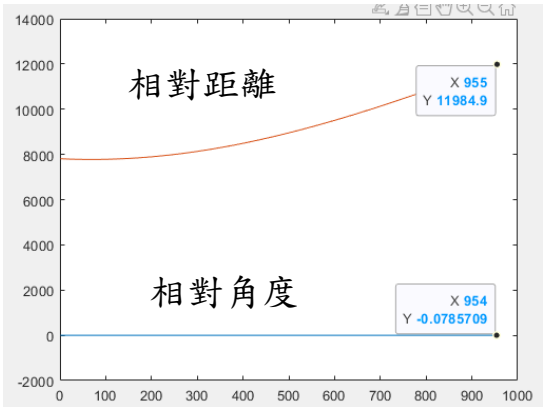
圖表 14己方x軸座標變化



圖表 13己方y軸座標變化



圖表 16對方x y座標軸變化



圖表 15相對角度及距離變化

3.3 演算法開發與實務

在此章節中，我將介紹前面章節所提到的高斯牛頓法的實際應用。在實際的船艦上是聲納來量測其他船艦的位置，依靠所聽到的螺旋槳聲音知道船方位，而螺旋槳的轉速可以讓我們得知船的種類，去猜測對方的最高速度，因此在實際的下，我們所能依靠的數據就只有依聲音大小分辨方位的相對角度及猜測的速度，在此演算法中，我們將以角度作為核心，預估出對方船艦實際航行速度及航行方向。

(一)獲取資料

取得(a, beta, x0, y0, xowm, yown)等資料進行運算，a為一個2*1的向量，負責儲存在主程式猜測的速度及角度，beta是從3.2節船艦模擬器中取得的相對角度，x0, y0是在主程式猜測的對方位置，xown, yown是己方船艦的位置。

```
public void GaussNewton(double[,] a,double[] beta,double[] x0,double[] y0,int bm ,double[] xown,double[] yown)
{
    int beginMoment = bm;
    int m = beta.Length;
    int n = a.Length;// k = 0;// a.Length, k = 0;
    int p = n;
    double maxstep = 500, delT = 2, tol = 10e-8, S=0;
    double[] T = new double[1000];
    double[,] J = new double[m,n];
    double[,] JT = new double[n,m];
    double[,] Jz = new double[n,n];
    double[,] Jzinv = new double[n, n];
    double[,] g = new double[n, m];//Jz\JT
    double[,] g1 = new double[2, 1];//g*r
    double[,] r = new double[m, 1];//殘差
```

圖表 17設定高斯牛頓法參數

（二）計算jacobian矩陣

下列所示演算法的部分，主要是在進行a = aold - 0.01 * $(J^T * J)^{-1} * J^T$ * r的疊代，因此我們將上述所提到的資料代入函式df，計算jacobian矩陣及轉至矩陣J^T。

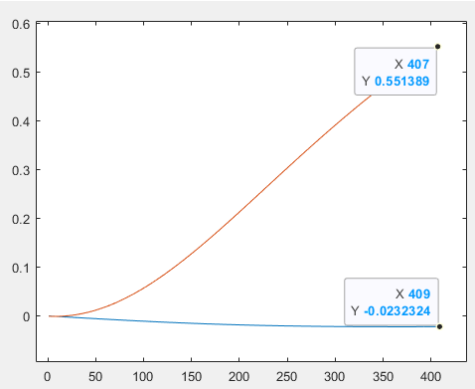
```
for (int u = 1; u < maxstep; u++)
{
    S = 0;
    for (int i = 0; i <= m-1; i++)
    {
        for (int j = 0; j <= n-1; j++)
        {
            J[i, j] = df(x0[0], y0[0], xown[i], yown[i], T[i], a[0, 0], a[1, 0], j);
            JT[j, i] = J[i, j];
        }
    }//計算jacobian及JT
```

圖表 18計算jacobian矩陣

```
public double df(double x0, double y0, double xown, double yown, double T, double a1, double a2, double index)
{
    double value;
    switch (index)
    {
        case 0:
            value = ((T * Math.Sin(a2)) / (x0 - xown + T * a1 * Math.Cos(a2)) - (T * Math.Cos(a2) * (y0 - yown + T * a1 * Math.Sin(a2))) /
                Math.Pow((x0 - xown + T * a1 * Math.Cos(a2)), 2)) / (Math.Pow((y0 - yown + T * a1 * Math.Sin(a2)), 2) / Math.Pow((x0 - xown + T * a1 * Math.Cos(a2)), 2) + 1);
          //value = ((T * Math.Sin(a2)) / (x0 - xown + T * a1 * Math.Cos(a2)) - (T * Math.Cos(a2) * (y0 - yown + T * a1 * Math.Sin(a2))) / Math.Pow((x0 - xown + T * a1 * Math.

            return value;//break;
        case 1:
            //value = ((T * a1 * Math.Cos(a2)) / (x0 - xown + T * a1 * Math.Cos(a2)) + (T * a1 * Math.Sin(a2) * (y0 - yown + T * a1 * Math.Sin(a2))) / Math.Pow((x0 - xown + T
            value = ((T * a1 * Math.Cos(a2)) / (x0 - xown + T * a1 * Math.Cos(a2)) + (T * a1 * Math.Sin(a2) * (y0 - yown + T * a1 * Math.Sin(a2))) /
                Math.Pow(x0 - xown + T * a1 * Math.Cos(a2), 2)) / (Math.Pow(y0 - yown + T * a1 * Math.Sin(a2), 2) / Math.Pow(x0 - xown + T * a1 * Math.Cos(a2), 2) + 1);
            return value;//break;
        default:
            return 0;
    }
}//jacobia
```

圖表 19計算df函式



圖表 20由matlab計算jacobian之值

(三)計算殘差

```
for (int i = 0; i < m; i++)
{
    r[i, 0] = beta[i] - Math.Atan((y0[0] - yown[i] + T[i] * a[0, 0] * Math.Sin(a[1, 0])) /
        (x0[0] - xown[i] + T[i] * a[0, 0] * Math.Cos(a[1, 0])));

    r1[i,0] = r[i,0];
}//計算殘差
```

圖表 21計算殘差r

(四) $Jz = -J^T * J$

```
for (int i = 0; i <= n - 1; i++)
{
    for (int j = 0; j <= p - 1; j++)
    {
        for (int k = 0; k <= m - 1; k++)
        {
            Jz[i, j] = Jz[i, j] + -JT[i, k] * J[k, j];

        }
    }
}//Jz = -JT * J;
```

圖表 22令$Jz = -J^T * J$

(五)反矩陣

```
{
    double det = Jz[0, 0] * Jz[1, 1] - Jz[0, 1] * Jz[1, 0];
    Jzinv[0, 0] = Jz[1, 1] / det;
    Jzinv[1, 0] = -Jz[1, 0] / det;
    Jzinv[0, 1] = -Jz[0, 1] / det;
    Jzinv[1, 1] = Jz[0, 0] / det;
}//反矩陣
```

圖表 23計算jacobian矩陣之反矩陣

（五）$g = (Jz)^{-1}J^{T}$

```
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
    {
        for (int k = 0; k < p; k++)
        {
            g[i, j] = (g[i, j] + Jzinv[i, k] * JT[k, j]);
        }
    }
}//g = Jz\JT;
```

圖表 24令$g = (Jz)^{-1}J^{T}$

（六）$a = aold - 0.01 * (J^{T} * J)^{-1} * J^{T} * r$

```
for (int i = 0; i <= n - 1; i++)
{
    for (int j = 0; j <= p - 2; j++)
    {
        a[i, j] = aold[i, j] -  g1[i, j];
    }
}//a[i,j] = aold[i,j] - 0.01*g1[i,j];
```

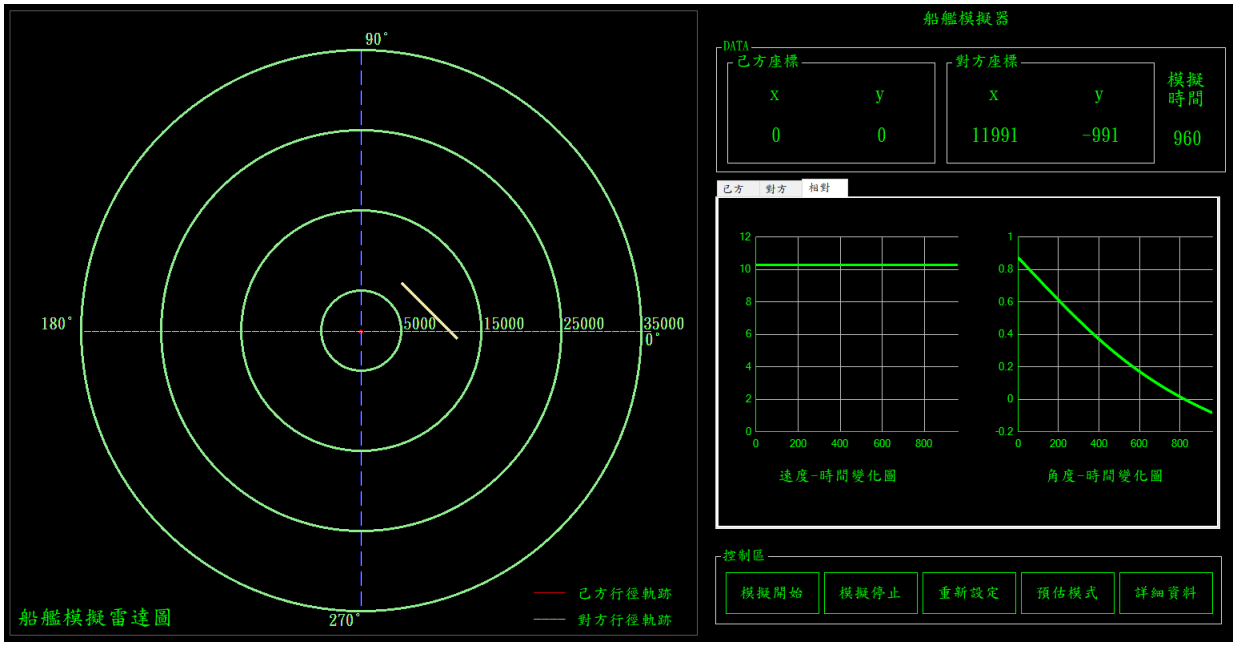
圖表 25得出新的角度速度存至a

至此為計算1次，只要持續計算將會得到接近實際值的角度及速度，這就是以高斯牛頓法來預估對方船艦實際運行角度速度的演算法。

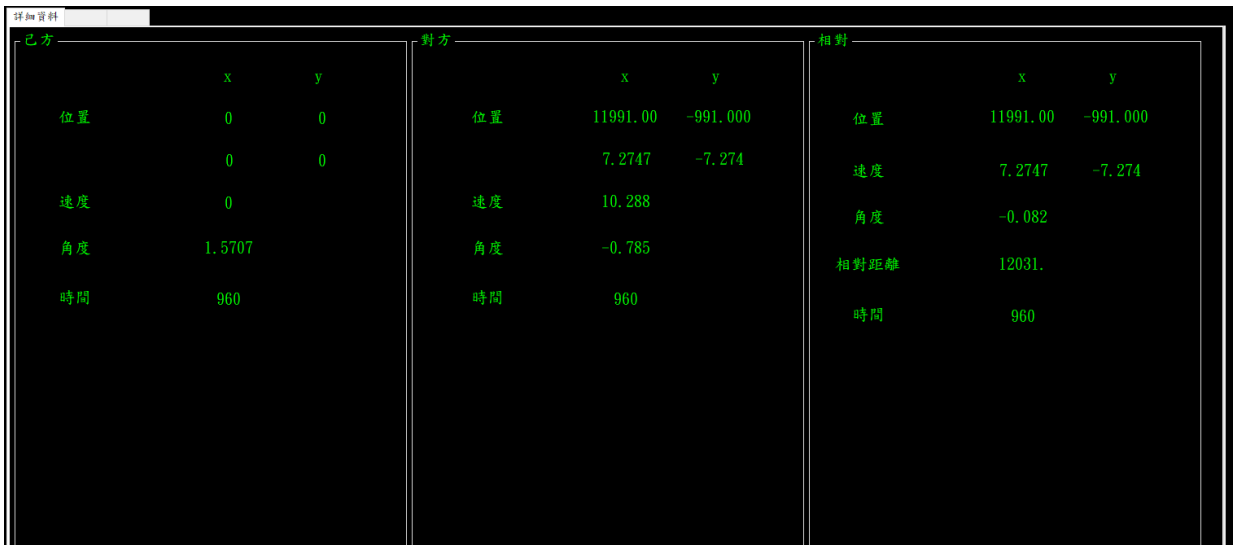
第四章 實務模擬結果

4.1船艦模擬結果

如圖表26所示，紅色原點即為己方位置，米色線條為對方船艦的移動軌跡，由右方的對話方塊得知雙方的座標位置，也可得知雙方相對速度及相對角度。



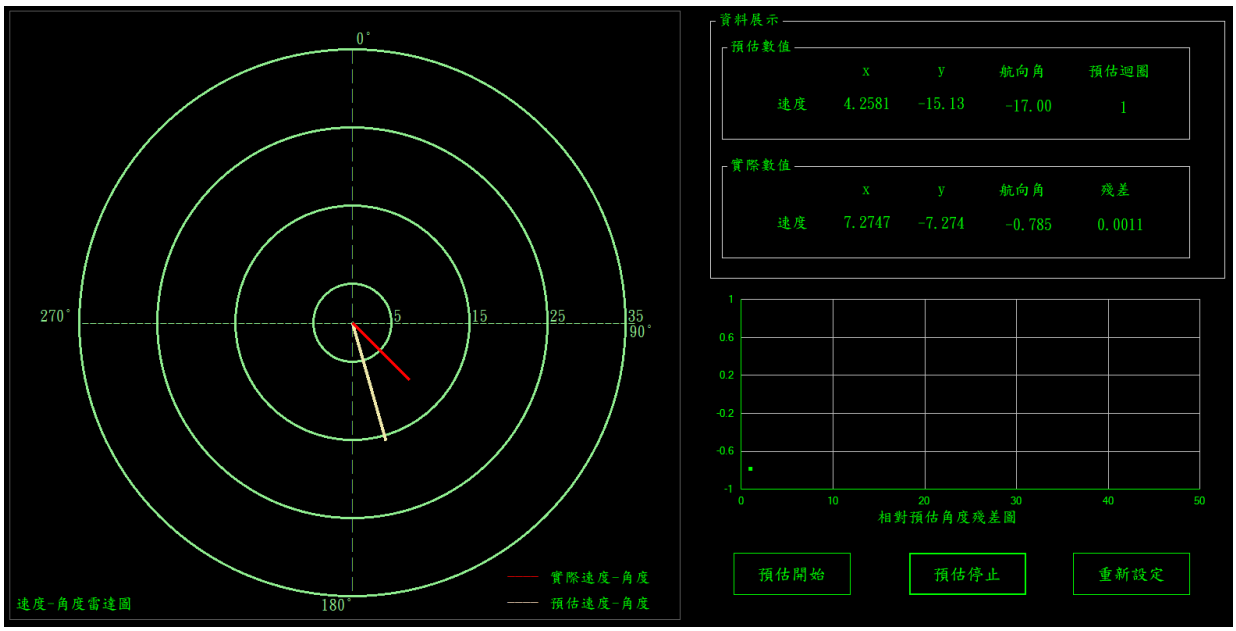
圖表 26船艦模擬器



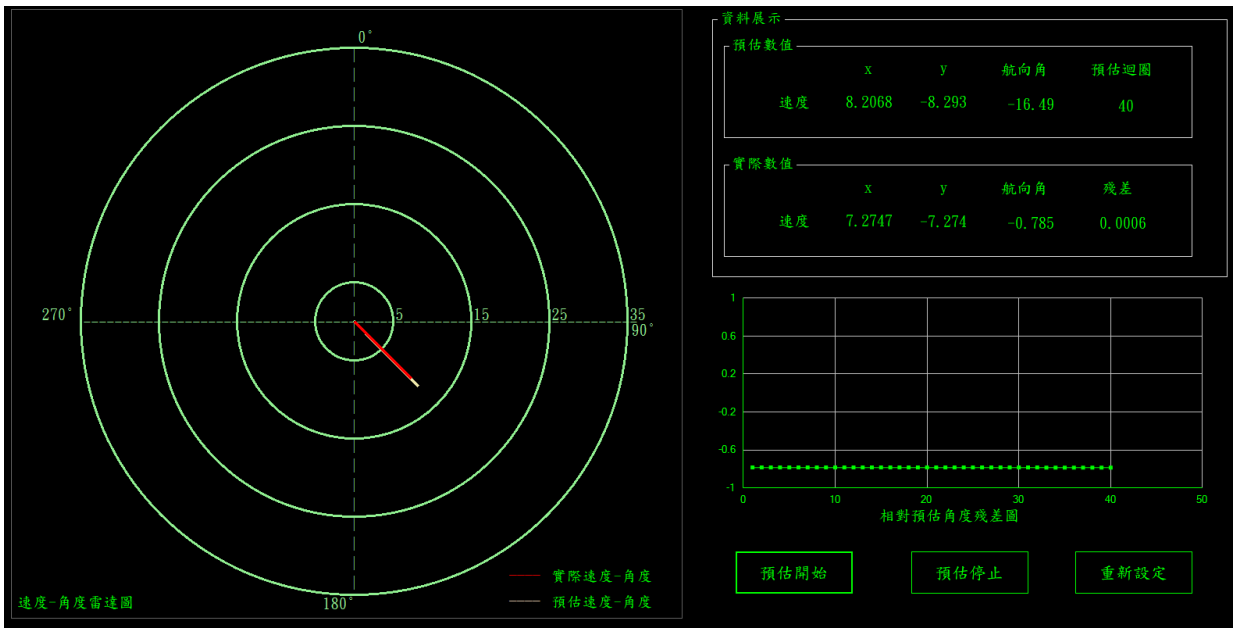
圖表 27詳細資料

4.2演算法運行結果

預估第一次的值通常是不準確的，由圖表28可知預估出來的結果(米色)與實際結果(紅色)相差甚遠，但在預估了40次之後可以發現，預估線非常接近實際線。



圖表 28最初預估結果



圖表 29最終預估結果

第五章 結論與未來研究展望

(一)結論

我們從2021年的12月開始作專題，直到現在花了將近一年的時間在製作，起初C#及matlab都是不曾接觸過的軟體，更別說寫演算法了，在這段時間以來我們遇到了各種困難，在此真的非常感謝老師及碩班學長們支援與建議，實在使我們受益良多，從最初的連介面都不會畫，一步一步學習，學語法、學陣列、學繪雷達圖等等，一直到最後200、300行程式，3個介面中的每個字元每行程式都出自於自己之手，那成就感是令人感到無比的驕傲，還記得當初連如何讓matlab開始運作都不知道，遇到不懂的就開始問，沒人能問的時候就查資料，到現在都會有其他朋友來詢問我們C#、matlab該怎麼用，讓我們著實感受到自己以經成長了。

(二)未來展望

在本次專題中仍有許多我們無法完成的部分

1. 己方與對方初始參數自訂義
2. 己方與對方有更多元的運動模式
3. 預估值與實際值之間仍有不小誤差
4. 無法預估己方與對方之間的距離

參考文獻

- [1]船艦模擬器與高斯牛頓演算法-陳聖化 教授
- [2]數值優化之高斯-牛頓法（Gauss-Newton）-程式人生
  ttps://www.796t.com/content/1548912065.html
- [3]高斯牛頓法-台部落
  https://www.twblogs.net/a/5eedee7533e47b02063c400b
- [4]牛頓法 高斯牛頓法-Cheng Wei's Blog
  https://scm_mos.gitlab.io/algorithm/newton-and-gauss-newton/
- [5]MATLAB程式設計：入門篇
  http://mirlab.org/jang/books/matlabprogramming4beginner/
- [6]Gauss-Newton Method 高斯牛頓法（最佳化簡介）-HMOO 讀書筆記
  https://www.hmoonotes.org/2020/06/intro-gauss-newton.html
- [7]Visual C# 程式設計完全解析-資訊種子研究室著
- [8]Introduction to Linear Algebra (3rd Ed.) by Gilbert Strang.
  Introduction to Linear Algebra (3rd Ed.) by Gilbert Strang.
- [9] K.Chakrabarty, S. Iyengar, H. Qi, and E. Cho, "Grid covera ge for surveillance and target location in distributed sensor n etworks," IEEE Transactions on Computers, vol. 51, no. 12, pp. 1228-1453,Dec 2002.