基於 Wifi RTT 的室內定位系統

簡介

此系統以 Wifi RTT 技術估計定位目標至各參考點距離，並將估計距離以 regressor 機器學習模型校正至更準確的距離，最後以 Gradient Descent 技術進行 Maximal Likelihood 計算出定位距離，根據模擬實驗平均距離誤差在 2 公尺之內。

相關研究

Wifi RTT：

以 Wifi 資料封包往返兩個收發器的時間差進行距離估計，由於採用往返時間，避免了因不同機器時鐘誤差產生的距離估計誤差。

Random Forest 機器學習模型：

生成大量的較小型 Decision Tree，各自對訓練資料集的隨機子集合進行訓練，預測時總和所有預測結果為此模型結果。有點「陪審團」的概念，能夠很有效的避免 over fitting，且對大部分機器學習議題都能達到優秀的精準度。

Maximal Likelihood：

原為來自統計學的概念，本質為透過 try and error 的方式對可觀測但未知的目標函數找出最符合的擬合函數，以此進行隨機變數的 regress。
到定位技術的領域則衍生為：透過各種方式根據已知但可預期存在誤差的「定位目標至各參考點距離」來估計「最有可能的目標所在位置」。

Gradient Descent：

為一種機器學習的方式，根據對於模型的輸出估計與真實資料的已知知識，制定計算其間誤差的「誤差函數」，接著對誤差函數進行偏微分取得梯度，並不斷根據梯度方向往誤差函數較低的方向調整輸出估計，直到誤差函數足夠低。
此方法的優勢為不需要訓練資料，且數學理論基礎直觀而明確，理論上可取得最精準的估計結果；缺點除了每次估計都要重新整個流程、要求速度會嚴重影響準度之外，還有「可能取得區域最佳而非全域最佳」、「接近最佳時震盪」、「學習過程可能之字形跳躍下降而非效能最優」等問題。

Random Forest regressor features

estRange: RTT 估計的距離。

rssi: RTT 收發過程中的信號強度指標，反映距離或該次信號收發當下的連線品質。

burstDev: RTT 分多個 burst (短時間內一小波多次收發)，同 burst 估計距離的標準差。

rssiDiff: 以最多 9 筆資料(9 個 burst)為 sliding window，將該筆資料 rssi 減去時間序上附近資料的平均 rssi。這個 feature 的意義是除去 rssi 受距離的影響，單純反映該次信號收發當下的連線品質。

Gradient Descent 細節

error function 推導：

$(x-x_1)^2+(y-y_1)^2+(z-z_1)^2={r_1}^2$

$\rightarrow x^2-2x_1x+{x_1}^2+y^2-2y_1y+{y_1}^2+z^2-2z_1z+{z_1}^2={r_1}^2$

$\rightarrow x^2-2x_1x+y^2-2y_1y+z^2-2z_1z={r_1}^2-{x_1}^2-{y_1}^2-{z_1}^2$

Let $\begin{cases}C_1={x_1}^2+{y_1}^2+{z_1}^2-{r_1}^2\\ C_2={x_2}^2+{y_2}^2+{z_2}^2-{r_2}^2\end{cases}$, then $\begin{cases}x^2-2x_1x+y^2-2y_1y+z^2-2z_1z+C_1=0\\ x^2-2x_2x+y^2-2y_2y+z^2-2z_2z+C_2=0\\ \dots\end{cases}$

Single AP error(s): $\ln((x^2-2x_1x+y^2-2y_1y+z^2-2z_1z+C_1)^2+1)$

Dual APs error(s): $\ln\left(\left((x_2-x_1)2x+(y_2-y_1)2y+(z_2-z_1)2z+C_1-C_2\right)^2+1\right)$

N 台 AP 有 N 個 Single AP errors、N(N-1)/2 個 Dual APs errors，全部相加即誤差函數。

Single AP error(s)偏微分：

$\ln((x^2-2x_1x+y^2-2y_1y+z^2-2z_1z+C_1)^2+1)'_x$

$=\frac{2(x^2-2x_1x+y^2-2y_1y+z^2-2z_1z+C_1)2x-2x_1}{(x^2-2x_1x+y^2-2y_1y+z^2-2z_1z+C_1)^2+1}$, yz 同理

Dual APs error(s)偏微分：

$\ln\left(\left((x_2-x_1)2x+(y_2-y_1)2y+(z_2-z_1)2z+C_1-C_2\right)^2+1\right)'_x$

$=\frac{2\left((x_2-x_1)2x+(y_2-y_1)2y+(z_2-z_1)2z+C_1-C_2\right)2(x_2-x_1)}{\left((x_2-x_1)2x+(y_2-y_1)2y+(z_2-z_1)2z+C_1-C_2\right)^2+1}$, yz 同理

將所有誤差的偏微分根據加法原則 xyz 分別總和，得到誤差函數的梯度 xyz 分量函數。

基於誤差的 decay learning rate：

傳統的 decay learning rate 是依據訓練時間(epoch)進行遞減，此研究選擇根據前一次輸出誤差來調整 learning rate。如此一來遠離最佳點時側重速度，接近最佳點側重準度，十分直觀而有效。1/gl*log(e/10+1)，其中 gl 為梯度向量長度；e 為誤差函數輸出。

Dual Jumping Window Convergence：

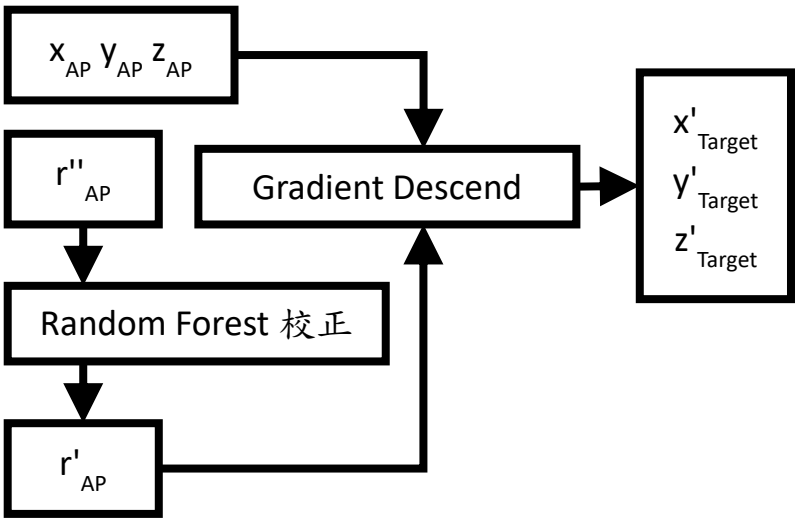
此誤差函數代表「估計定位目標位置到各參考點距離」與「輸入的定位目標位置到各參考點距離」之間的平方誤差。由於後者已經可預期存在誤差，即連輸入的距離都無法決定一定位點，此應用中的誤差函數輸出及梯度向量長度都不可能收斂至零。

因此絕大部分傳統收斂判斷方式都不堪使用，本研究改採自創方式：

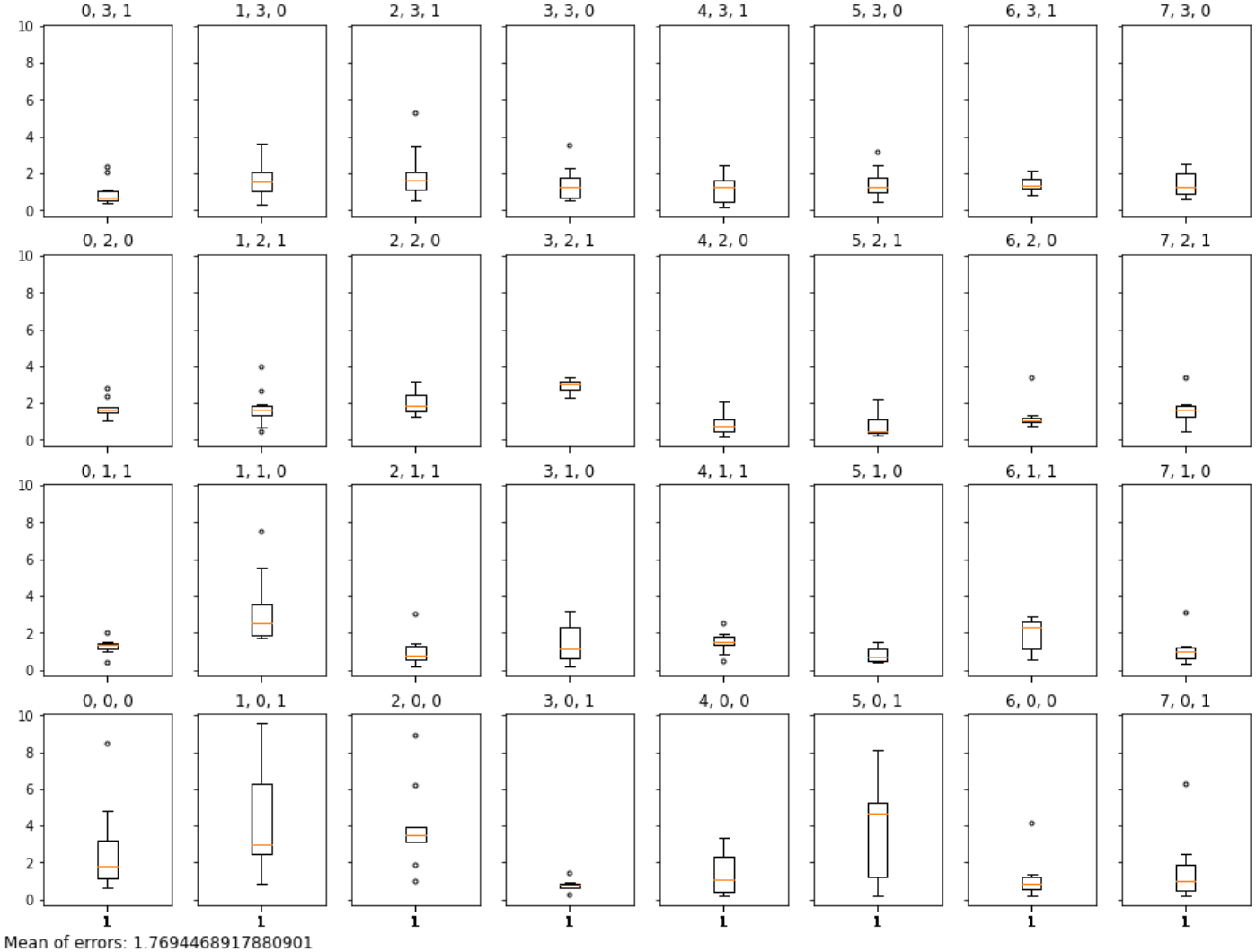
制定三個 hyper parameters：interval、window、small，每過 interval 次 epoch，就會將「最近 window 次 epoch 的平均誤差」與「interval 次之前取另一塊 window 次 epoch 的平均誤差」進行比較，若差距小於 small 則結束 Gradient Descent。

此方法更加全面性(robust)，直接針對收斂的本質意義「誤差無法再降低」進行判斷。

系統架構



結果



Mean of errors: 1.7694468917880901

結論

此定位系統根據模擬實驗平均距離誤差在 2 公尺之內，系統的兩項最大優勢為「不須重複訓練」以及「參考點(如 AP 等)可隨意設置」。校正模型只需進行一次大量資料採集訓練後便可以沿用到任何場所的應用，Gradient Descent 也使用不須訓練資料的方式；而在任何場所隨意設置多個參考點後，只需將參考點的座標輸入系統產生座標空間即可。整體而言此定位系統在應用上彈性更佳。

額外

Colab：

https://colab.research.google.com/drive/1rJ15P4locbVVmQJgsXI7B6cjxq8ncBzl?usp=sharing

root 資料夾：

https://drive.google.com/drive/folders/1RAnMj9jM282AFLcG8Ndsp2KdgL3haeYg?usp=sharing

備註 1：

之前提到 Gradient Descent 的部分改成先輸入 AP 位置，再重複輸入定位目標與 AP 距離，這部分的優化因為受限於由 Python 呼叫 Java 的架構無法平行處理(.jar 整個跑完才回到 Python 繼續執行)，所以無法使用，還是要每次呼叫 Gradient Descent 時重複輸入 AP 位置。但相較於 Gradient Descent 大量的迴圈，這部分的 overhead 影響不大。

備註 2：

我發現 Gradient Descent 中，之前提到的加入「base」那個方法其實把輸入再放更大也一樣會出問題，沒有太大的作用。真正聰明的方式是做 feature scaling，但我們要應用到定位上不能這麼做。而我之前測試的那些資料在實際定位應用根本也用不到(例如到 100000 公尺)，所以這部分直接拔掉了，改為另外針對這次的應用調整 hyper parameters (例如 learning rate 等等)。

以下是最終版本的 Java code：

```
package maximal_likelihood;
import java.util.Scanner;
public class Gradient_Descent_positioning {
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);
		int ap = scanner.nextInt();
		double[] x = new double[ap+1];
		double[] y = new double[ap+1];
		double[] z = new double[ap+1];
		double[] r = new double[ap+1];
		double [] c = new double[ap+1];
		for(int i = 1; i <= ap; i++) {
			x[i] = scanner.nextDouble();
			y[i] = scanner.nextDouble();
			z[i] = scanner.nextDouble();
			c[i] = x[i]*x[i] + y[i]*y[i] + z[i]*z[i];
		}
		double[][] xx2 = new double[ap+1][ap+1];
		double[][] yy2 = new double[ap+1][ap+1];
		double[][] zz2 = new double[ap+1][ap+1];
		double[][] cc = new double[ap+1][ap+1];
		for(int i = 1; i <= ap; i++) {
			for(int j = 1; j <= ap; j++) {
				xx2[i][j] = (x[i] - x[j])*2;
				yy2[i][j] = (y[i] - y[j])*2;
				zz2[i][j] = (z[i] - z[j])*2;
				cc[i][j] = c[i] - c[j];
			}
		}
		for(int i = 1; i <= ap; i++) {
			r[i] = scanner.nextDouble();
```

```
        }
        double small = 0.000001;
        int limit = 1000000;
        double rate = 0.0;
        double tx = 0, ty = 0, tz = 0;
        double gx = 0, gy = 0, gz = 0, gl = 1;
        double e = 0;
        final int window=1000,interval=10000;
        double[] eh1 = new double[window];
        double[] eh2 = new double[window];
        int t = 0;
        do {
            rate = 1/gl*Math.log(e/10+1);
            tx -= gx*rate;
            ty -= gy*rate;
            tz -= gz*rate;
            gx = 0;
            gy = 0;
            gz = 0;
            e = 0;
            for(int i = 1; i <= ap; i++) {
                double raw = tx*tx - 2*x[i]*tx + ty*ty - 2*y[i]*ty + tz*tz - 2*z[i]*tz + (c[i] - r[i]*r[i]);
                double share = 2*raw/(raw*raw + 1);
                gx += share*2*(tx - x[i]);
                gy += share*2*(ty - y[i]);
                gz += share*2*(tz - z[i]);
                e += Math.log(raw*raw + 1);
                for(int j = 1; j <= ap; j++) {
                    if(i < j) {
                        raw = xx2[i][j]*tx + yy2[i][j]*ty + zz2[i][j]*tz + (cc[j][i] + r[i]*r[i] - r[j]*r[j]);
                        share = 2*raw/(raw*raw + 1);
                        gx += share*xx2[i][j];
                        gy += share*yy2[i][j];
                        gz += share*zz2[i][j];
                        e += Math.log(raw*raw + 1);
                    }
                }
            }
            e /= (ap + ap*(ap-1)/2);
            gl = Math.sqrt(gx*gx + gy*gy + gz*gz);
            t++;
            eh1[t%window] = e;
            if(t%interval == 0) {
                double m1=0,m2=0;
                for(int i=0;i<window;i++) {
                    m1+=eh1[i];
                    m2+=eh2[i];
                }
                m1/=window;
                m2/=window;
                if(Math.abs(m2-m1) < small) {
                    break;
                }
                for(int i=0;i<window;i++) {
                    eh2[i] = eh1[i];
                }
            }
        }while(t < limit);
        System.out.println(tx + " " + ty + " " + tz);
    }
}
```