國 立 中 央 大 學

資 訊 工 程 學 系

碩 士 論 文

整合 MIAT 方法論與大型語言模型於系統設計：

以指紋比對系統為案例研究

Integration of MIAT Methodology and Large Language Models for System Design:

A Case Study of Fingerprint Matching System

研 究 生：饒珮以

指導教授：陳慶瀚 博士

中 華 民 國 114 年 1 月

中文摘要

本研究以指紋特徵比對系統為驗證案例，探討結合 MIAT 方法論系統設計與大型語言模型的系統設計和高階合成方法。MIAT 方法論著重在系統的階層式模組化的架構設計，以及演算法的離散事件建模，最後合成可維護、可擴展的系統程式碼。在方法論的最後階段，我們引進大型語言模型生成每個離散事件狀態所驅動所需要的程式碼。為了驗證這個方法，我們設計了指紋特徵比對系統的階層模組，包括旋轉排除模組、線段方向排除模組及特徵比對模組。實驗部分，基於 FVC2004 指紋資料集進行系統性能測試，透過不同比對演算法如 Minutia 與 FLANN 的比較，驗證了系統在準確性與運行效率上的表現。實驗結果顯示在模組化設計下，各模組的獨立性和可靠性得到了充分驗證。大型語言模型在程式碼生成中的應用，提升了開發效率並降低人為錯誤的風險。此研究不僅對於指紋辨識技術的進一步發展具有啟發性，也提供了一種創新的系統設計方法，適用於其他高複雜度的軟體開發領域。

關鍵字：指紋比對；MIAT 方法論：離散事件建模；高階合成；大型語言模型 (LLM)

Abstract

This research uses a fingerprint feature matching system as a validation case to explore system design and high-level synthesis methods that combine MIAT methodology with Large Language Models (LLMs). The MIAT methodology focuses on hierarchical modular architecture design and discrete event modeling of algorithms, ultimately synthesizing maintainable and scalable system code. In the final stage of the methodology, we introduce LLMs to generate the necessary code driven by each discrete event state. To validate this approach, we designed hierarchical modules for the fingerprint feature matching system, including rotation elimination module, segment direction elimination module, and feature matching module. For experimentation, we conducted system performance testing based on the FVC2004 fingerprint dataset, comparing different matching algorithms such as Minutia and FLANN to validate the system's performance in terms of accuracy and operational efficiency. The experimental results demonstrated that under the modular design, the independence and reliability of each module were thoroughly validated. The application of LLMs in code generation improved development efficiency and reduced the risk of human error. This research not only provides insights for further development of fingerprint recognition technology but also offers an innovative system design method applicable to other complex software development domains.

謝 誌

本論文之完成，首先衷心感謝恩師慶瀚教授的悉心指導與鼓勵。碩士期間，從文獻的探討、研究方向的選擇、觀念架構之建立、實驗方法的設計，以迄本文之撰寫，吾師不斷地予以指導與啟迪，更對初稿逐字斧正，使得本論文得以順利完成，師恩浩瀚，永銘五內。此外，慶瀚教授開設的嵌入式系統設計課程讓我受益良多，課程內容層層深入，清楚闡述方法論，使我在研究過程中獲得寶貴的知識與啟發。承蒙口試老師林明義教授、謝昇憲教授許多寶貴的建議與指正，謹致以最深的謝意。

同時，我也要感謝彥廷同學與如珊同學，在最後口試前匆忙的日子裡，彼此相互督促與鼓勵，共同努力完成論文。

當然，我的雙親與家人的鼓勵與支持是功不可沒的。尤其是在撰寫論文最後階段的晨昏顛倒、日以繼夜的日子裡，沒有他們的悉心照料與打氣，我肯定是無法撐過去的。衷心感謝他們無所求的付出

目錄

中文摘要 i
Abstract ii
目錄 iv
圖目錄 vii
表目錄 ix
第一章、 緒論 1
1.1 研究背景 1
1.2 研究動機與目的 2
1.3 研究架構 3
第二章、 指紋比對技術回顧 4
2.1 指紋特徵 4
2.1.1 全域特徵 4
2.1.2 局部特徵 5
2.1.3 次級特徵 6
2.2 指紋比對演算法 8
2.2.1 Image correlation 演算法比對 9
2.2.2 Phase 演算法比對 12
2.2.3 Skeleton 演算法比對 14
2.2.4 Minutia 演算法比對 17
2.2.5 FLANN 演算法比對 22
第三章、 可重構指紋特徵比對系統設計 25
3.1 MIAT 方法論 25
3.1.1 IDEF0 階層式模組化設計 26
3.1.2 Grafcet 離散事件建模 27

3.2 LLM 輔助生成 Grafcet action 程式碼 29
3.3 可重構指紋特徵比對系統架構設計 31
3.2.1 指紋比對模組(A4) 32
3.2.2 線段旋轉排除模組(A41) 33
3.2.3 線段方向排除模組(A42) 34
3.2.4 特徵點比對模組(A43) 35
3.4 離散事件建模 36
3.3.1 指紋特徵比對離散事件模型 37
3.3.2 線段旋轉排除離散事件模型 38
3.3.3 線段方向排除離散事件模型 39
3.3.4 特徵點比對離散模型 40
第四章、 可重構指紋特徵比對實驗 41
4.1 實驗環境 41
4.2 資料描述 41
4.3 基於 LLM 的 Grafcet 框架生成方法：實驗與應用分析 43
4.3.1 LLM 模型 43
4.3.2 Grafcet Action 生成流程 43
4.3.3 生成結果驗證 45
4.4 可重構指紋特徵比對驗證 49
4.4.1 指紋特徵點擷取 49
4.4.2 Minutia 比對實驗 51
4.4.3 FLANN 比對實驗 54
4.5 結果比較 57
4.6 實驗結果分析 60
第五章、 結論 61

5.1 結論 61
5.2 未來展望 62
參考文獻 63
附錄一 生成 Grafcet 對話流程 72
附錄二 可重構指紋比對 Grafcet Action 75

圖目錄

圖 2.1 Minutia 類型-核心和三角形奇異點.....5
圖 2.2 Minutia 類型： (a) 山脊末端 (b) 山脊分岔.....5
圖 2.3 脊線 6
圖 2.4 (a)脊線數量為 2 (b)脊線數量為 1 7
圖 2.5 脊線曲率方向 (a)凹形 (b)凸形 7
圖 2.6 將樣本圖像 t 對應到來源圖像 f.....10
圖 2.7 相位比對流程圖.....13
圖 2.8 (a)區域座標系統(Base-ridge) (b)正特徵點 (c)負特徵點 15
圖 2.9 骨架對應的區域骨架描述器.....15
圖 2.10 匹配對(Matching pairs)與其相關參數.....18
圖 2.11 ps、qs為中心的符合條件匹配對結果 19
圖 2.12 KD 樹示意圖.....23
圖 3.1 MIAT 系統設計方法論 25
圖 3.2 IDEF0 功能模組.....26
圖 3.3 IDEF0 表示 MIAT 方法論架構 27
圖 3.4 Grafcet 範例.....28
圖 3.5 (a)Grafcet 狀態表示 (b)Grafcet 行為方塊 29
圖 3.6 指紋辨識系統 IDEF0 模組階層規劃.....31
圖 3.7 指紋比對模組(A4).....32
圖 3.8 線段旋轉排除模組(A41).....33
圖 3.9 線段方向排除模組(A42).....34
圖 3.10 特徵點比對模組(A43).....35
圖 3.11 指紋辨識系統離散事件建模.....36
圖 3.12 指紋比對離散事件建模.....37

圖 3.14 線段方向排除離散事件建模.....39
圖 3.15 特徵比對離散事件建模.....40
圖 4.1 每個資料庫指紋圖片範例.....42
圖 4.2 (a)輸入 Prompt 產生狀態 0-4 (b)實際產生狀態 0-4 的程式碼.....45
圖 4.3 (a)輸入狀態 40-43 的 Prompt (b)實際產生狀態 40-43 的程式碼.....46
圖 4.4 執行修正 Prompt.....47
圖 4.5 產生修改後的程式碼.....47
圖 4.6 架構流程圖簡易.....48
圖 4.7 指紋比對模組 0 至 410 狀態的高階軟體合成狀態轉移結果.....48
圖 4.8 指紋比對系統操作介面.....49
圖 4.9 選取樣本特徵集(DB1_102_2)與測試特徵集(DB1_102_5).....50
圖 4.10 確認系統成功正確讀取特徵點數據.....50
圖 4.11 Minutia 比對實驗成功結果.....51
圖 4.12 Minutia 比對實驗失敗結果.....52
圖 4.13 線段與線段參數.....53
圖 4.14 FLANN 比對實驗成功結果.....55
圖 4.15 FLANN 比對實驗失敗結果.....56
圖 4.16 指紋比對相似度統計圖.....58
圖 4.17 指紋比對時間統計圖.....59
附圖 1 翻譯中文 Grafcet.....72
附圖 2 架構邏輯確認.....72
附圖 3 生成對應的程式碼.....73
附圖 4 子系統邏輯敘述確認.....73
附圖 5 結合子系統與主架構程式.....74

表目錄

表 4.1 實驗環境規格 41

表 4.2 FVC2004 資料集說明 42

表 4.3 Minutia 比對實驗結果紀錄.....53

表 4.4 FLANN Matching 實驗結果紀錄.....57

第一章、　緒論

1.1　研究背景

指紋辨識技術作為一種基於生物特徵的安全驗證方法，憑藉其唯一性、穩定性和便捷性[1]，廣泛應用於金融交易、智慧設備解鎖以及邊境管制等多個領域[2]，可以通過指紋比對來進行身份認證和識別來保護個人財產及私人訊息[3]。然而，隨著大規模數據處理需求的增加以及應用場景的多樣化，指紋辨識系統面臨了效率與準確性兼顧的挑戰。

為了提高指紋比對的效率和準確性，近年來，研究人員開始探索自動化指紋比對技術。隨著計算機視覺、機器學習等技術的不斷發展，自動化指紋比對技術也在不斷進步[4]。目前指紋比對技術已經實現了高度自動化，可以在數秒內完成大量的指紋比對且準確率高[5]。

指紋比對是通過比較待辨識與數據庫中的指紋特徵，判斷是否為同一個人的過程。常用的指紋比對演算法包括 Qi 與 Zhao 等人使用 Hough Transform 來進行指紋特徵比對[6]和 Lee 和 Choi 提出區域性對齊比對演算法來降低指紋影象因指紋扭轉和牽引造成變形而產生非線性變形的參數[7]。

指紋比對系統需處理龐大的數據量，包括已紀錄和待辨識的指紋資料，並進行複雜運算。由於這些數據量非常龐大，因此需要進行高效的存儲和管理方式[8]。為了提高指紋比對效率，Peralta、García、Benitez 與 Herrera 提出了一種指紋比對分解方法[9]，即將指紋圖像分解為多個小區域，然後對每個小區域進行指紋比對，該方法使用了 Apache Hadoop 和 Apache Spark 分散式叢集架構，系統只需定義區域性結構和聚合函數，即可達到良好的可擴展性和高效性。該研究進一步驗證了系統框架的優越性。

在軟體開發層面，系統設計的複雜性也給開發效率與維護性帶來了壓力[10]。傳統系統設計方法難以有效應對功能模組的快速擴展和升級需求，特別是在面對高複雜度的生物辨識系統時。MIAT 方法論的引入為此提供了一種有效解決方案。該方法論以階層式模組化為核心，通過將系統分解為易於管理的小型模組，實現功能模組的高獨立性與易擴展性。

此外，人工智慧技術的快速發展，特別是大型語言模型(LLMs)(如 ChatGPT)在程式碼生成方面展現了令人印象深刻的效能。通過分步進行程式碼生成，可以逐步引導 LLMs 分析和實現程式邏輯，從而提高程式碼的正確性、品質和可維護性。這些模型將自然語言提示轉換為可執行程式碼的能力，對軟體開發實踐產生重大影響，顯著減少手動編碼工作和人為錯誤的可能性[11]。

1.2 研究動機與目的

系統架構的設計和優化變得越來越重要，本研究提出了一種基於模組化的指紋特徵比對系統設計方法，並結合大型語言模型(LLMs)在自動生成程式方面的卓越能力[12]。這種方法能幫助系統開發人員更高效地進行開發和維護。

本論文首先將針對指紋特徵比對系統進行研究，探討系統架構模組化的設計思路和好處。模組化的系統架構使得不同模組之間相互獨立，可以單獨維護和更新，而不影響整個系統的運行[13]。當需要增加新功能或修改現有功能時，只需針對相應模組進行修改，無需重構整個系統[14]，從而大幅簡化開發和維護工作。這樣可以提高系統的可靠性和穩定性，減少系統故障對用戶帶來的影響。模組化的系統架構也可以讓不同的開發人員專注於不同的功能模組開發，提高開發效率[15]。然而，模組之間的接口定義清晰，不同開發人員可以獨立開發並協作完成，提高效率[16]。此外，模組化設計支持模組重用，減少開發時間和成本，縮短軟體開發生命週期。

MIAT 方法論強調階層式模組化設計，提供了將複雜功能分解為易於管理的模組化解決方案的可能性。同時，人工智慧技術的快速進步，大型語言模型進行程式碼生成可以顯著提高程式碼的準確性和開發效率，同時降低人為錯誤和安全漏洞的風險，從而大幅提升軟體開發的質量和效率[17]。

基於此，本研究旨在結合 MIAT 方法論與大型語言模型，開發一套創新的高效系統設計框架，並以指紋特徵比對系統作為驗證案例，透過不同比對演算法如 Minutia 與 FLANN 的比較，探討這種方法在高複雜度軟體開發中的應用價值。研究目標包括驗證模組化設計在提升系統獨立性與可靠性方面的效果，探索大型語言模型在程式碼生成中對開發效率和準確性的貢獻，評估結合 MIAT 與 LLM 方法後系統在準確性與運行效率上的整體性能，並進一步分析該設計框架在其他高複雜度軟體開發場景中的適用性。本研究期待能在指紋辨識技術的應用以及系統設計理論的拓展方面實現突破，為未來相關領域的研究與實踐提供有價值的創新思維。

1.3 研究架構

本研究以結合 MIAT 方法論與大型語言模型(LLMs)的創新系統設計方法為核心，旨在探討其在指紋特徵比對系統開發中的應用價值。整體架構採用系統性的方法，從設計理論到實驗驗證，逐步揭示該方法的潛在優勢與挑戰。

此研究架構分為五章節，第一章緒論說明研究背景、動機與目的，並介紹研究架構。第二章為回顧與此指紋比對有關的現行技術的參考文獻。第三章細說研究方法，使用 MIAT 方法論設計流程、程式碼生成方法以及實驗設計的具體步驟。第四章為系統高階合成，主要為驗證整合 MIAT 方法論與大型語言模型於系統設計，並以指紋比對系統為案例研究，展示並分析系統性能測試結果。第五章為回顧研究結果與本實驗貢獻，並提出在研究中所遇到的限制與未來研究方向。

第二章、　指紋比對技術回顧

指紋比對因其各種特徵而流行，例如：易於收集資料，易於訪問眾多來源，且十根手指可用於收集。指紋的使用有十九世紀的記錄，弗朗西斯·高爾頓爵士定義了指紋的特徵點，這些特徵點被稱為高爾頓點(Galton points)，且高爾頓點亦是建立指紋識別系統的基礎，高爾頓點的子集用於表示指紋影像，這些被稱為特徵點(Minutia)[18]。

2.1　指紋特徵

指紋特徵中，脊線呈現為黑色線條，相反之谷地則呈現為脊線之間的白色區域[19]。指紋特徵大致可分為三類，分別為全域特徵、局部特徵、次級特徵，這些特徵在指紋識別系統中扮演著重要角色。

全域特徵提供了指紋的基本形狀訊息，局部特徵則提供了詳細的識別訊息，而次級特徵在高精度識別中發揮重要作用。通過結合這些特徵，可以有效地提高指紋識別的準確性和穩定性[20]。

2.1.1　全域特徵

全域特徵是指紋圖像中最直觀的特徵，這些特徵描述了指紋的整體形狀和紋理結構。它們在指紋識別的初步分類階段中起著至關重要的作用，因為它們能夠迅速將指紋分類為幾個基本類型，從而減少比對的搜尋範圍。

其中特徵形成了一種特殊的脊線和谷地圖案，稱為奇異點或單點(Singular Points)[21]，可進一步分為三種類型：環狀、三角和螺旋。重要的是核心點及三角點。核心點被定義為最內側脊線上的最重要點，而三角點則被定義為三種不同

趨勢流匯聚的中心點如圖 2.1 所示。因此，這些特徵為指紋分類、指紋比對和指紋對齊提供了有用和關鍵的訊息[22][23]。

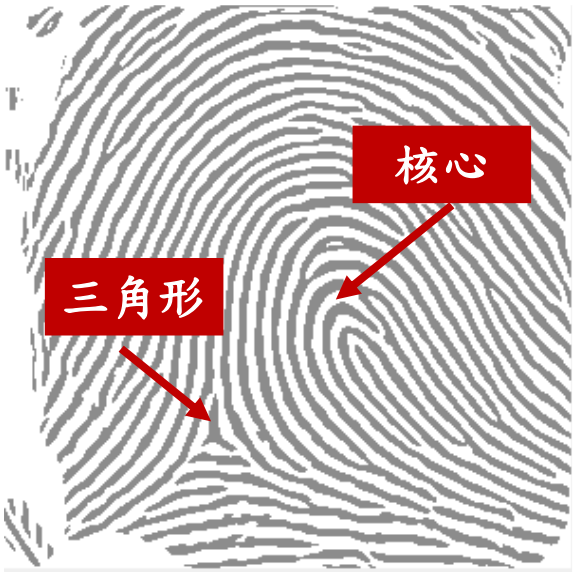


圖 2.1 Minutia 類型-核心和三角形奇異點

2.1.2 局部特徵

在局部層面上，稱為細節特徵的脊線特徵是用於指紋比對的最廣泛使用的特徵。雖然細節特徵有幾種類型，主要考量兩種類型的細節特徵[24][25]，這兩種是最突出的脊線特徵：脊線末端指的是脊線突然終止的點，如圖 2.2 (a)所示，脊線分叉指的是脊線分岔或分散成分支脊線的點，如圖 2.2(b)所示。

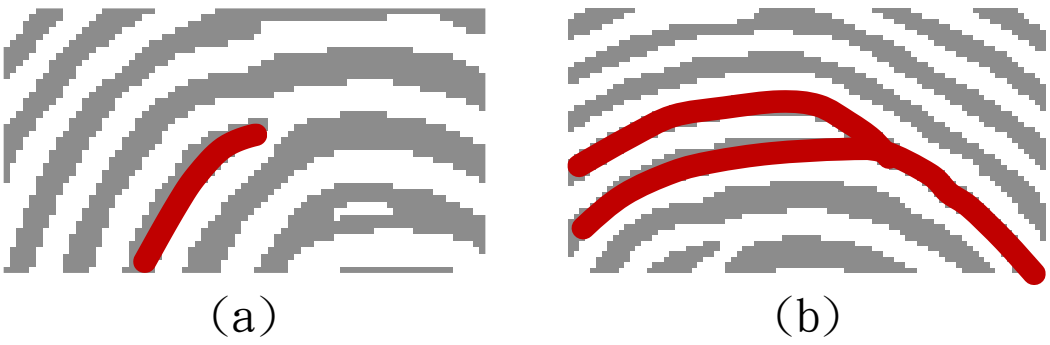


圖 2.2 Minutia 類型： (a) 山脊末端 (b) 山脊分岔

2.1.3 次級特徵

指紋特徵的次級特徵是指在主要特徵的脊線終點和分叉點之外更細微且更具體的特徵，這些次級特徵在指紋比對和識別過程中也起著重要作用。以下是一些常見的次級特徵[26]：

1. 脊線方向

每個脊線的方向[27]，包括脊線的起點方向、終點方向以及脊線的整體走向如圖 2.3 所示。由特徵點定義為中心，從特徵點延伸出一條垂直軸，並且垂直於所有指紋線條結構，而此脊線垂直軸可以克服扭曲的指紋圖片。

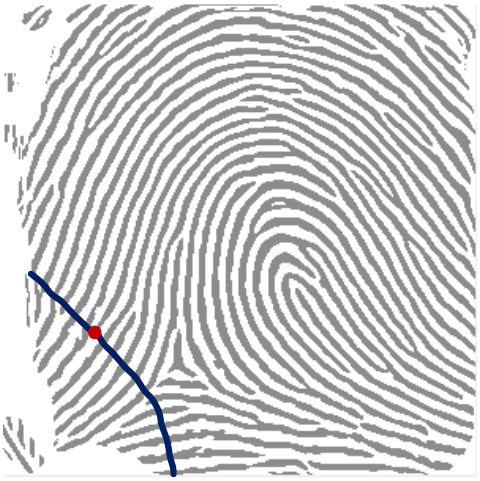


圖 2.3 脊線

2. 脊線數量

脊線數量[26](Ridge Count)是透過計算沿垂直軸的脊數來計算的，直到軸與附著的脊相遇。脊線數量的定義是垂直軸經過的山脊數量。如果已經包含的脊線或脊線沒有附加的特徵點時，則不會新增脊線數量，而兩特徵點(u、v)之間的脊線數量，如圖 2.4 所示。

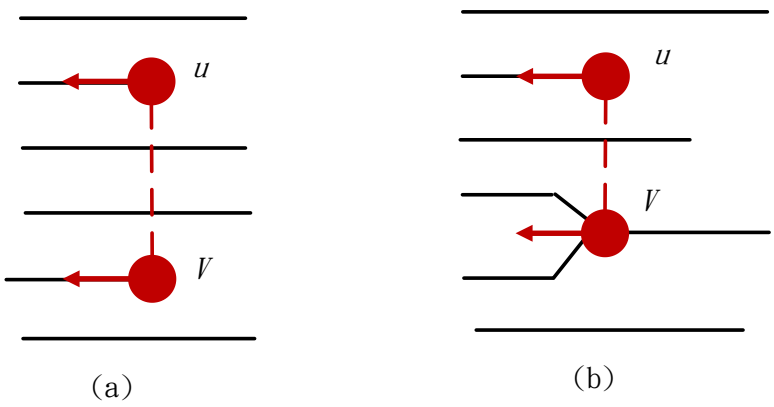


圖 2.4 (a)脊線數量為 2 (b)脊線數量為 1

3. 脊線曲率

脊線的彎曲程度即為脊線曲率[28]，為了在脊線模式中使用更多資訊進行比對，需考慮脊線曲率方向(Ridge Curvature Direction)。這是因為即使脊線數量和脊線長度值非常相似，脊線圖案的形狀也可能不同。例如，有兩條曲線的長度相同，但一條是凹形，另一條是凸形，如圖 2.5 所示。如果只考慮曲線的長度，就無法區分曲線，因此在這種情況下，如果需要納入曲率方向，才可以分辨出它們是不同的。

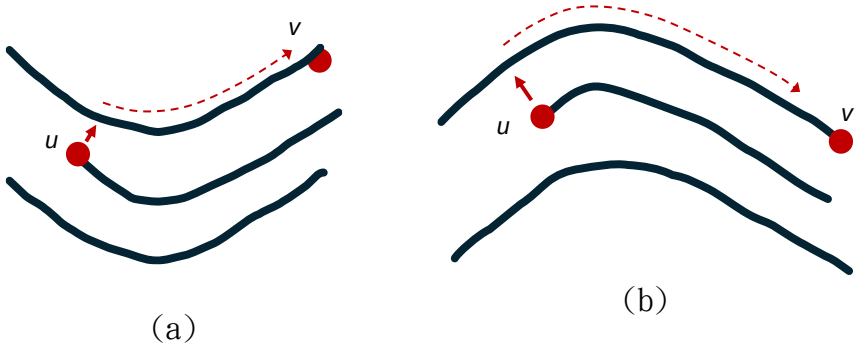


圖 2.5 脊線曲率方向 (a)凹形 (b)凸形

2.2 指紋比對演算法

指紋比對在驗證和辨識問題中是一個至關重要的步驟[29]。該算法通常比較兩個指紋圖像，並回傳一個實數區間的相似度或二元輸出(比對成功或比對失敗)。指紋比對模組計算兩個指紋之間的比對分數時，來自同一手指的指紋應該很高，而來自不同手指的指紋應該很低[30]。但現實往往不是，許多因素使指紋比對為一個困難的問題：其中圖像雜訊、皮膚狀況、扭曲、旋轉、位移等等，造成同一手指的不同影像之間具有很大的差異，然而另一個為來自不同手指的兩個圖像之間具有很大的相似性[31]。

大多數指紋比對演算法採用四種方法之一[30]：Image correlation 演算法比對、Phase 演算法比對、Skeleton 演算法比對和 Minutia 演算法比對。其中最廣泛的技術是 Minutia 的比對。

在過去的二十年中，已有多種機器學習和深度學習方法應用於指紋分類，指紋分類方法需要使用機器學習、深度學習以及仿生模型等預處理技術，以提高指紋分類的準確性[32]。指紋分類演算法在減少指紋識別系統的搜尋時間和計算複雜度方面非常重要[33]。隨著科技進步，傳統的指紋識別系統易受假指紋攻擊，導致安全問題，因此在 Liu 與 Li 等人研究[34]使用 FLANN(Fast Library for Approximate Nearest Neighbors)演算法，擷取測試的內部指紋特徵點並比對指紋特徵集來以認證身份，與傳統算法相比，這種方法難以模仿，且具有較高的安全性。在 Mishra 與 Dehuri[35]的實驗中使用粒子群演算法(Particle Swarm Optimization)連結 FLANN 在真實收集的指紋中，透過提出的 FLANN-PSO 演算法，在不同參數和指紋的不同角度特徵測試其準確性，並獲得了 98%的準確率，準確性和均方誤差(MSE)方面的結果顯示出相較於其他算法的顯著改善。

接下來的小節將詳細介紹各指紋比對的主要概念。指紋比對過程分為兩個部分：特徵擷取和比較。比對的部分是指將一個指紋圖像與另一個指紋圖像進

行比較以確定它們是否相同的過程。比對原理是基於指紋圖像中的紋理特徵進行比較，通過將兩個指紋圖像的紋理特徵進行對齊，然後比較它們之間的差異程度來確定它們是否配對成功。一般會使用指紋辨識演算法將指紋圖像轉換為數據表示，並且將數據進行比較，以確定它們是否相同。

2.2.1 Image correlation 演算法比對

影像比對是指在同一場景下，比對兩張或多張影像的幾何關係的過程。這些影像可以在不同時間、不同角度或使用不同成像感測器所獲得。通過比對影像的共同特徵，識別並比對它們之間的相似性，從而進行影像配對[36]。其通常是用於圖形識別、圖形比對、目標跟蹤等應用領域。

一般而言，相關方法是一種可以顯示變量對(Pairs of variables)之間相關性強度的技術，以結果的相關係數值來表示，範圍從-1.0 到+1.0，接近+1 的範圍表示一個變量與另一個變量之間的關係越密切。它在圖形識別中展示了良好的應用效果。其中，正規化交叉相關比對法(Normalized Cross Correlation)廣泛應用於機器視覺的工業檢測，包括複雜圖像中的瑕疵檢測。

Insankeovilay 與 Prasarn [37]研究中，設 f 為測試圖像(來源圖像)，t 為樣本圖像，透過尋找樣本圖像 t 與測試圖像 f 任何部分的相似性。在這種情況下，圖像 t 的大小小於或等於 f 時。測量相似性或不匹配的一種簡單方法是對特定區域內的樣本圖像 t 和測試圖像 f 取絕對差，如下圖 2.6 所示。

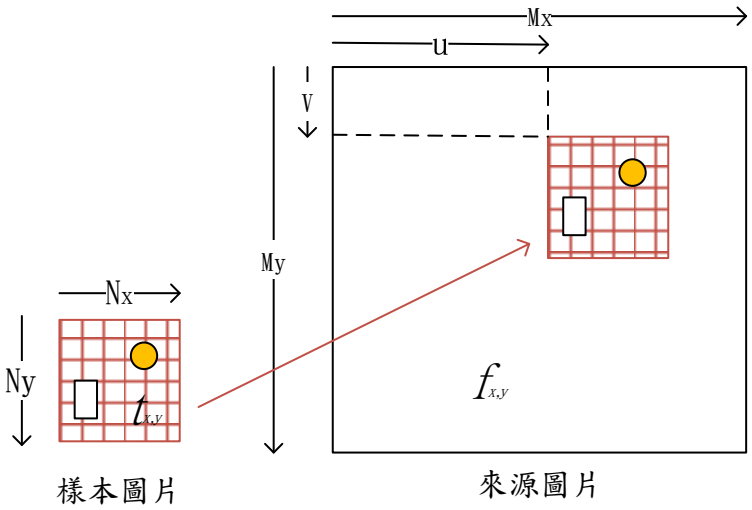


圖 2.6 將樣本圖像 t 對應到來源圖像 f

實驗中來源圖像 f(測試圖像)與樣本圖像 t(數據庫)的比較步驟分為三個：

1. 將兩個圖像裁剪成 128x128 像素大小(核心點位於中心)。
2. 將每個圖像分成 4 個子圖像，每個子圖像為 64x64 像素。
3. 使用 NCC 樣本來比對相應的子圖像。每個區域的比對分數相加，分數高於設定值(閾值)的圖像則比對成功。

其 NCC 比對相關說明如下：

樣本圖像 t 和來源圖像 f 在每個維度上偏移 u 和 v 的區域進行差異平方和的計算，因此可以得到以下公式(2.1)：

$$d_{f,t}^2(u,v) = \sum_{x,y}[f(x,y) - t(x-u, y-v)]^2 \tag{2.1}$$

從上述公式延伸出公式(2.2)：

$$d_{f,t}^2(u,v) = \sum_{x,y}\begin{bmatrix} f^2(x,y) - 2f(x,y)t(x-u,y-v) \\ +t^2(x-u,y-v) \end{bmatrix} \tag{2.2}$$

而$\sum_{x,y}[t^2(x-u,y-v)]$因為從樣本圖像來，所以是固定值，且$\sum_{x,y}[f^2(x,y)]$也是固定的。

因此相似性的交叉相關公式表示為：

$$c(u,v) = \sum_{x,y} [f(x,y)t(x-u,y-v)] \tag{2.3}$$

然而直接使用公式(2.3)會導致圖像強度可能在不同區域間變化的問題，而使獲得的結果不一致。為避免此類問題，需考慮均值(means)和方差(variances)，因此定義公式(2.4)稱為正規化交叉相關(NCC)。

正規化交叉相關公式：

$$\gamma(u,v) = \frac{\sum[f(x,y) - \bar{f}_{u,v}][t(x-u,y-v) - \bar{t}]}{\sqrt{\sum_{x,y}[f(x,y) - \bar{f}_{u,v}]^2 \sum_{x,y}[t(x-u,y-v) - \bar{t}]^2}} \tag{2.4}$$

$\bar{f}_{u,v}$與$\bar{t}$分別表示為：

$$\bar{f}_{u,v} = \frac{1}{N_x N_y} \sum_{x=u}^{u+N_x-1} \sum_{y=v}^{v+N_y-1} f(x,y)$$

$$\bar{t} = \frac{1}{N_x N_y} \sum_{x=u}^{u+N_x-1} \sum_{y=v}^{v+N_y-1} t(x,y)$$

簡單來說，圖像相關比對是指將兩個指紋圖像進行重疊，通過比較不同對齊方式下對應像素的相關性來計算它們之間的相似度。當比對區域相對較小時，圖像相關比對能夠提供良好的計算效果，但這種技術會消耗大量的存儲和計算資源。然而，相關性比對的往往較難以獲得理想的結果，主要是由於圖像的全局結構、亮度和對比度發生了不良的變化，而這些變化通常取決於指紋的扭曲和皮膚狀態[31]。

2.2.2 Phase 演算法比對

典型的指紋識別方法採用基於特徵的影像比對，從註冊的指紋影像和輸入指紋影像中擷取特徵，主要為指紋脊末端和指紋脊分岔，並通過對比這些特徵在兩幅圖像中對應的特徵點數量來識別有效的指紋圖像[38]。然而，由於特殊的面板條件，一些人無法透過基於特徵的方法識別指紋，因為影像處理很難擷取特徵點。擁有如此困難指紋的人的比例因種族、性別、年齡、工作組合等而異，但佔總人口的百分之一到百分之五可能屬於這一類別。為了解決這個問題，Ito、Morita、Aoki[39]等人提出了一種使用基於相位的影像比對的高效指紋識別演算法，使用給定影像的二維離散傅立葉變換(Discrete Fourier Transforms)中的相位分量的影像比對技術。其相位相關函式(Phase-correlation function)的相位影像比對的演算法，由四個步驟組成分別為核心檢測、旋轉和位移對齊、公共區域擷取和、指紋比對，如圖 2.7 所示。

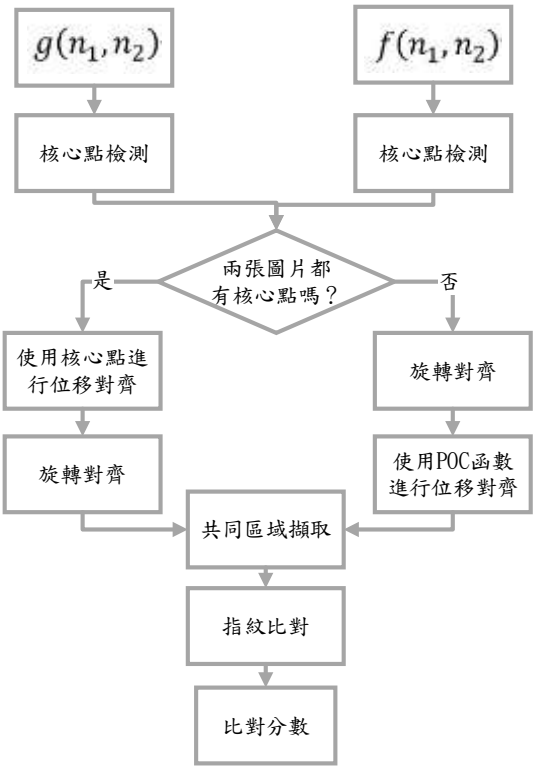


圖 2.7 相位比對流程圖

首先先將註冊指紋影像$f(n_1,n_2)$和輸入指紋影像$g(n_1,n_2)$對齊兩個影像，接著對兩指紋進行旋轉和位移。在兩個指紋影像都有核心的情況下，使用核心的位置對齊指紋影像之間的平移位移，在角度範圍為$-40^\circ \le \theta \le 40^\circ$且角度間隔為$1^\circ$的情況下，對於已註冊的指紋影像$f(n_1,n_2)$，計算其旋轉後的影像$f_\theta(n_1,n_2)$的BLPOC(Band-Limited Phase-Only Correlation)值，其中影像旋轉採用雙三次插值法(bicubic interpolation)。通過評估旋轉後的已註冊影像$f_\theta(n_1,n_2)\left(-40^\circ \le \theta \le 40^\circ\right)$與輸入影像$g(n_1,n_2)$之間的相似性，可以確定輸入影像相對於註冊影像的旋轉角度θ。

但當$f(n_1,n_2)$或$g(n_1,n_2)$中有一者缺少其核心時，使用上述方法對旋轉進行正規化。接下來，對旋轉正規化(Rotation-normalized)的圖像$f_\theta(n_1,n_2)$和輸入圖像$g(n_1,n_2)$進行平移位移。平移位移可以通過$g(n_1,n_2)$和$g(n_1,n_2)$之間的 POC 函數(Phase-Only Correlation function)的峰值位置獲得。將獲得已正規化的註冊圖像和輸入圖像，分別表示為 $f'(n_1,n_2)$和$g'(n_1,n_2)$。

接下來的步驟是擷取兩幅圖像$f'(n_1,n_2)$和$g'(n_1,n_2)$的重疊區域。此過程提高了指紋比對的準確性，因為兩幅圖像的非重疊區域會成為 BLPOC 函數中的不相關雜訊成分。為了檢測已註冊圖像$f'(n_1,n_2)$和輸入圖像$g'(n_1,n_2)$中的有效指紋區域，因此檢查的n_1軸突出畫素值和n_2軸突出畫素值，只有相同大小的公共有效圖像區域$f''(n_1,n_2)$和$g''(n_1,n_2)$會被擷取，用於後續的圖像比對步驟。最後計算兩個擷取出的圖像$f''(n_1,n_2)$和$g''(n_1,n_2)$之間的 BLPOC 函數$r_{f''g''}^{K_1K_2}(n_1,n_2)$，並評估比對分數。由於彈性指紋變形可能會產生多個相關峰值，因此定義兩個圖像之間的比對相似度分數為 BLPOC 函數$r_{f''g''}^{K_1K_2}(n_1,n_2)$的最高兩個峰值之和，如公式(2.5)。

比對相似度分數其計算公式如下：

$$r_{fg}^{K_1K_2}(n_1,n_2)=\frac{1}{L_1L_2}\sum_{K_1,K_2}^{'}R_{FG}(K_1,K_2)\times W_{L_1}^{-k_1n_1}W_{L_2}^{-k_2n_2} \tag{2.5}$$

$$n_1=-K_1\ldots K_1, n_2=-K_2\ldots K_2$$

$\sum'_{K_1,K_2}$　表示$\sum_{k_1=-K_1}^{K_1}\sum_{k_2=-K_2}^{K_2}$

2.2.3 Skeleton 演算法比對

骨架化(skeletonization)是用於增強比對的一個常見工具，在電腦視覺和影像辨識中用於比對骨架或中軸表示的形狀的技術。物體的骨架是物體的一個較為簡化的版本，能夠捕捉到其幾何和拓撲特徵。Skeleton Matching 通過比較兩個形狀的骨架圖的拓撲來判斷它們是否相似。在[40]中，Feng 等人透過脊線來加強特徵點的比對，以避免錯誤比對。Sha 等人[41]將脊數用作特徵點的補充訊息。Octant[42]使用八個方向上最近的特徵點的脊數來改善特徵點比對。在 EFS 格式[43]中，NIST 提出了一個標準化的擴展特徵集，包括骨架、點和毛孔。

大多數作者沒有明確考慮變形，而是透過比對特徵點誤差的小位移來處理[44]，當特徵點位置在影像中位置被定義時，變形會影響這兩個二維坐標，因此在 Bohné 和 Despiégel[44]的研究中提出了一種指紋局部骨架描述器(Local Skeleton Descriptor)的方法，該描述器在特定坐標系統中編碼區域內特徵點的位置，在系統中僅通過使用脊數(Ridge count)使一個坐標對變形影響。兩點之間的脊數定義為連接這兩點的線段所跨越的脊線數量。當連接兩點的線段接近於脊線流(ridge flow)的正交時，任何變形都不太會影響脊數。

研究中先定義基脊(Base ridge)如圖 2.8 (a)所示，並將特徵點分為端點和正特徵點如圖 2.8 (b)與負特徵點如圖 2.8 (c)所示，在描述器中每個特徵點都表示為 M=(x,y,type)，因此最後每個描述器呈現的特徵點骨架如圖 2.9 所示，正特徵點用藍色星星表示，負特徵點用紅色十字架表示。

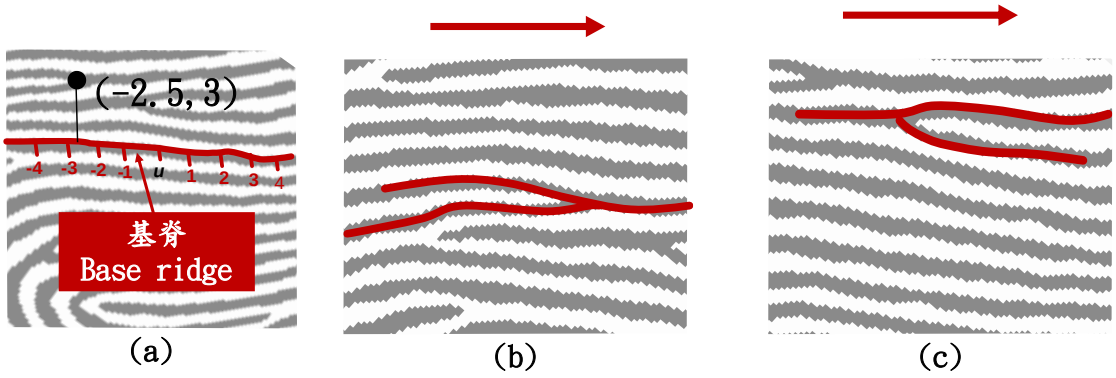


圖 2.8 (a)區域座標系統(Base-ridge) (b)正特徵點 (c)負特徵點

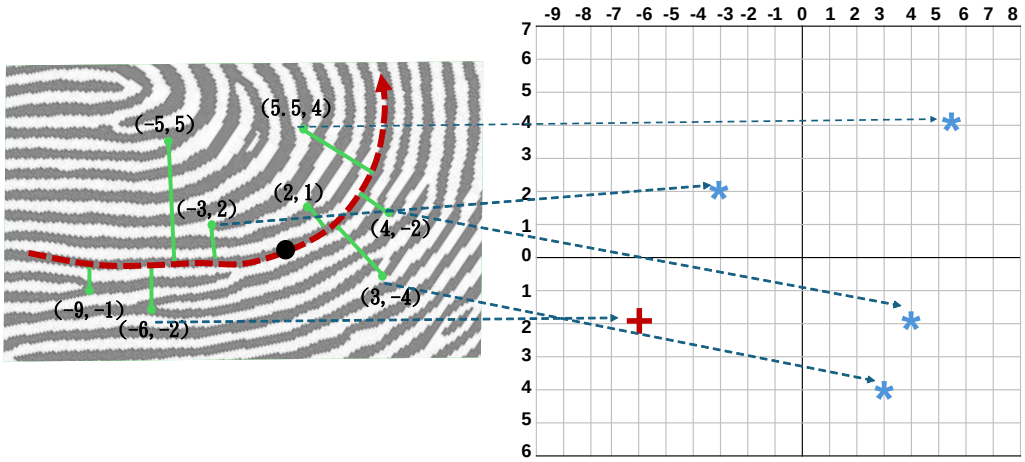


圖 2.9 骨架對應的區域骨架描述器

骨架比對演算法由三個階段組成，第一階段透過演算法將指紋特徵點的影像進行註冊，局部描述器沿紋線流向約有 200 像素長，變形可近似為線性延伸，變換參數為：s(線性延伸)、tx(沿紋線流向平移)和 ty(垂直紋線流向平移)。最佳轉換(Transformation)的難點在於未知的比對特徵點對(Minutia pairs)，因此採用概率方法，尋找最佳重疊最多特徵點集的轉換。

接著從兩個指紋中選擇 N 對對應的點，使用霍夫轉換來找出最佳參數 (s, tx, ty)。每對比對的特徵點在離散參數空間中對(s, tx, ty) 進行計算，如公式(2.6)。

$$Vote(\log(s), tx, ty) = \sum_{\substack{M^i_{src}M^k_{src} \in \ minutiae\ from\ search\ print \\ M^j_{ref}M^l_{Ref} \in \ minutiae\ from\ reference\ print \\ with\ type^i_{Src} = type^j_{Ref}\ and\ type^k_{Src} = type^l_{ref}}} P(\log(s), tx, ty | M^i_{src}, M^j_{ref}, M^k_{Src}, M^l_{Ref}) \quad (2.6)$$

兩對特徵點：

$$\left(M^i_{src}, M^j_{ref}\right)、(M^k_{Src}, M^l_{Ref})$$

s、tx 分別表示：

$$\begin{cases} x^i_{Src} = s \times x^j_{Ref} - tx \\ x^k_{Src} = s \times x^l_{Ref} - tx \end{cases} \Longrightarrow \begin{cases} s = \dfrac{x^i_{Src} - x^k_{Src}}{x^j_{Ref} - x^l_{Ref}} \\ tx = x^i_{Src} - s \times x^j_{Ref} \end{cases}$$

若計算後，每對兩個特徵點類型相同時，則比對成功成功機率較高為公式(2.7)

$$P(match(M^i_{Src}, M^j_{Ref})) \propto \begin{cases} e^{-\frac{(y^i_{src} - y^j_{ref})^2}{2\sigma_y^2} - \frac{(x^i_{Src} - x^j_{Ref})^2}{2\sigma_x^2}}, & if\ type^i_{Src} = type^j_{Ref} \\ 0\ otherwise \end{cases} \quad (2.7)$$

第二階段使用動態規劃(Dynamic programming)在搜尋指紋和參考指紋間找到一對特徵，其成本矩陣公式如公式(2.8)。在尋找特徵點集 A 時，使其在比對成本總和最小的情況下沒有特徵點被重複使用的情況，這個最小值用來計算相似度分數如公式(2.9)。

成本矩陣：$Cost_{ij} = 1 - P(match(M^i_{Src}, M^j_{Ref}))$ (2.8)

相似度分數：$Score = -\min_{A}\left(\sum_{(i,j)\in A} Cost_{ij}\right)$ (2.9)

下限：$(i,j) \in A \implies \neg\exists(i,k) \in A\, with\ k \neq j.$

第三階段為計算兩指紋的全域性相似性得分，因此合併了從許多局部描述器中計算出的分數。使用指紋中每個特徵點計算一個描述器，以產生選擇清單。接著透過描述器中的局部特徵點密度來計算每個描述器的機率，選擇機率最高的描述器(d)，將其從選擇清單中移除，若機率 p(d)低於閥值或選擇的描述器數量超過N，則結束。

計算兩個描述器之間的相似性分數完，最後才是輸出分數的平均值。其每一描述器(d')計算公式如(2.10)：

$$p^{k+1}(d') = p^{k}(d') \times \left(1 - e^{-\frac{dst(d,d\prime)^2}{2\sigma^2}}\right) \quad (2.10)$$

在實驗最後結果說明，骨架演算法的錯誤接受率大約是特徵點比對器的兩倍。 這種準確性的差異可能主要是因為骨架演算法不使用定向資訊，它只執行許多區域性比對，不像特徵點比對器那樣使用全域一致性的特徵。

2.2.4 Minutia 演算法比對

特徵點比對是通過比較指紋的細節特徵，也就是指紋留下的細小痕跡(minutia)，來識別兩個指紋是否相同。指紋圖像由脊紋和谷紋構成，形成獨特的結構，從不同指尖脊紋產生的影像中擷取結構，有兩種類型的指紋，分為平面或傾斜。平面指紋包括指尖和第一個指節之間的中心區域影像，而傾斜影像中，脊紋存在於手指的兩側，因此被稱為滾動影像[45]。

然而，Singh 等人[46]提出了一種新的指紋特徵檢測演算法，該演算法改善由指紋影象中存在的掃瞄指紋機造成的扭曲、旋轉、位移或雜訊產生的不良特徵

點問題。 為了解決這個問題，他們設計了一種有效的特徵擷取策略，可以擷取真正的特徵，只過濾和保留可以進一步用於指紋比對的關鍵特徵點。

Yin、Pan 等人[47]也提出採集過程中手指方向、位置、角度、壓力的變化，指紋特徵點產生了旋轉、平移、變形的問題，為了解決此問題，需要先將特徵點進行對齊，才可依照特徵點的對應關係來比對其相似度。因此此研究分為三個步驟，首先擷取指紋特徵點，並將特徵點表示為四維特徵($p = \{x, y, \theta, t\}$)，x 、 y 表示特徵點位置，θ表示特徵點方向($\theta \in [0, 2\pi]$)，t 表示特徵點型別(端點或分岔點)。特徵點p_a與特徵點p_b的脊線數(Ridge lines)表示為$\gamma(p_a, p_b)$，而所有脊線數矩陣則表示為$\Gamma[c, d] = \gamma(p_c, p_d)$。

第二步驟特徵點對齊後，透過局部比對方法來計算給定的指紋特徵和資料庫特徵之間的局部比對關係。 使用匹配對(Matching pair)結構的區域性特徵相鄰描述器(Minutia neighborhood descriptor)，建立了中心點及其周圍點之間的空間和方向關係，這種結構只取決於兩個特徵點和脊線資訊之間的相對位置關係，因此不會將指紋平移或旋轉角度內入計算。兩個排序的特徵點及其相應的脊線矩陣，分別為測試特徵集(Template minutia) $P = \{p_1, \ldots, p_n\}$及其脊線矩陣ΓP，以及樣本特徵集(Sample minutia)$Q = \{q_1, \cdots, q_m\}$及其脊線矩陣ΓQ。

比對的特徵點變數說明如圖 2.0 所示：

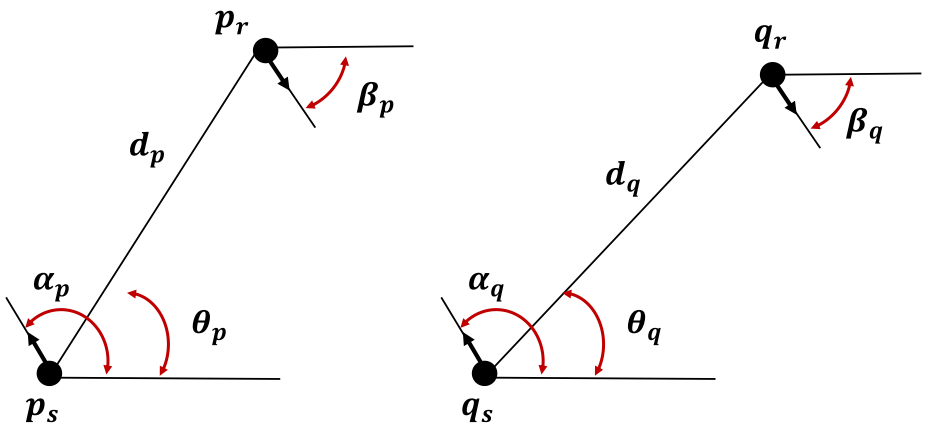


圖 2.10 匹配對(Matching pairs)與其相關參數

中心點：p_s 、q_s

中心點的周圍點：p_r 、q_r

因此可以分別組成測試特徵集對(Template minutia pair)$\{p_s, p_r\}$、樣本特徵集對(Sample minutia pair)$\{q_s, q_r\}$

兩對之間距離：d_p 、d_q

中心點的定向角度：α_p 、α_q

周圍點的定向角度：β_p 、β_q

中心點和周圍點的連線與正 X 軸方向之間的角度：

$\theta_p = arctan\frac{y_{ps}-y_{pr}}{x_{ps}-x_{pr}}$、$\theta_q = arctan\frac{y_{qs}-y_{qr}}{x_{qs}-x_{qr}}$

研究中考慮量化誤差，包含手指硬體感測器變形、感測器本身感應問題，為了消除不良特徵點，因此只將符合公式(2.11)條件的特徵點納入至集合中，

特徵點選取條件：

$$\begin{cases} |\gamma_p - \gamma_q| < T_\gamma \\ |d_p - d_q| < T_d \\ |(\beta_p - \alpha_p) - (\beta_q - \alpha_q)| < T_{\alpha\beta} \\ |(\beta_p - \theta_p) - (\beta_q - \theta_q)| < T_{\beta\theta} \end{cases} \tag{2.11}$$

符合條件的特徵點集：$A = \{p_s, q_s, p_r, q_r\}$

其結果範例如圖 2.1。最後符合局部比對結果 A 特徵點集($\{A_1, A_2, \ldots, A_k\}$)結果輸出為矩陣。

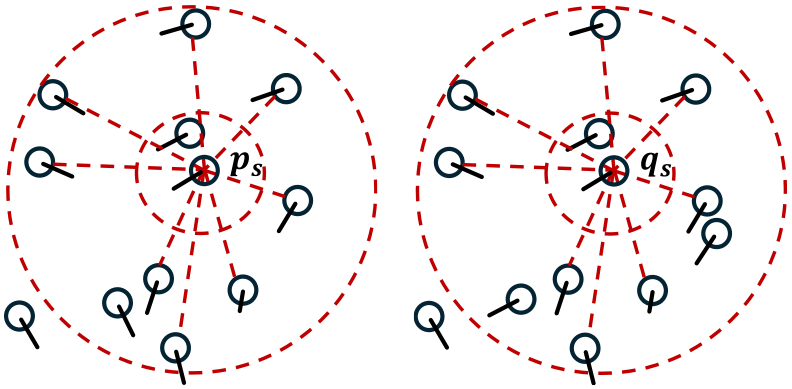


圖 2.11 p_s 、q_s為中心的符合條件匹配對結果

第三步驟建立指紋拓撲關係(Topological relationship)和建立特徵點之間相關性的方法。 然而，拓撲關係計算指紋的全域性相似性，以定量評估兩個特徵點之間的相似性，如公式(2.12)。

相似度分數：

c_p為測試特徵點中心鄰近的周邊點數量

c_q為樣本特徵點中心鄰近的周邊點數量

$c(p_i, q_j)$為特徵點p_i與特徵點q_j 為中心點的全域性匹配對數量

$$
\begin{aligned}
&v(p_s, q_s, p_r, q_r) \\
&\quad = min\left(\frac{c_p}{c(p,q)}, 1\right) \cdot min\left(\frac{c_q}{c(p,q)}, 1\right) \cdot \Delta\gamma_{pq} \cdot \Delta t_{pq} \\
&\quad \Delta\gamma_{pq} = \begin{cases} 1\ , & if\ \gamma_p = \gamma_q \\ 0.7\ , & otherwise \end{cases} \\
&\quad \Delta t_{pq} = \begin{cases} 1\ , & if\ t_{rp} = t_{rq} \\ 0.7\ , & otherwise \end{cases}
\end{aligned}
\tag{2.12}
$$

在全域相似性計算中，設計了三個子指標，從不同角度評估兩個特徵點的相似性，最後計算指紋的整體相似性，其計算步驟如下：

第一步驟依據匹配對數量的分數：正負全域性匹配對的數量差異很大，使匹配對的數量成為評估指紋相似性的重要指標。使用具有特定參數的 S 型函式(Sigmoid function)來計算匹配對數量的分數，參考公式(2.13)，參數是根據來自正負樣本的大量比對資料。

$$S_{num} = Z(n_{pairs}, \mu_n, \tau_n) \tag{2.13}$$

n_{pairs}表示全域性匹配對的數量

sigmoid 函式表示為 Z ，並且定義參數μ_n、τ_n

因此 sigmoid 表示如下(2.14)：

$$Z(n,\mu,\tau)=\frac{1}{1+e^{-\left(\frac{v-\mu}{\tau}\right)}} \tag{2.14}$$

第二步驟依據局部相似度的分數：局部相似性分數(LSS)方法按降序對所有全域性匹配對的相似性分數進行排序，如公式(2.16)。

$$\begin{aligned} &O_{lss}=\{(p_{sc},q_{sc},p_{rc},q_{rc})\},\\ &c=1,\ldots,n_p,1\le p_{sc}\ne p_{rc}<m,1\le q_{sc}\ne q_{rc}\le n \end{aligned} \tag{2.15}$$

而n_p元素的平均分數是指紋的區域性相似性得分S_{lss}，如公式(2.16)。

$$S_{lss}=Z\left(\frac{\sum_{(p_s,q_s,p_r,q_r)\in O_{ks}}V[p_s,q_s,p_r,q_r]}{n_p},\mu_l,\tau_l\right) \tag{2.16}$$

第三步驟最大叢集分數(Cluster Score)：在正指紋比對中，連線全域性匹配對的樣本特徵點對和測試特徵點對構成覆蓋指紋整個重疊區域的最大叢集，為了識別指紋中的叢集，隨意從任何特徵點開始，並使用深度第一搜尋(Depth-First Search)演算法來遍歷所有全域性匹配對。未遍歷的全域性匹配對不屬於叢集，可以排除。

構成最大叢集的全域性匹配對的相似性分數集表示為：

$$O_{cluster}=\{(p_s,q_s,p_r,q_r)\},1\le p_{sc}\ne p_{rc}<m,1\le q_{sc}\ne q_{rc}\le n$$

計算重疊區域內指紋比對性能的分數是最大叢集的分數(2.17)。

$$\begin{aligned} S_{cluster}=&\frac{min(p_{cov},q_{cov})}{min(p_{num},q_{num})}\\ &\times Z\left(\frac{\sum_{(p_s,q_s,p_r,q_r)\in O_{cluster}}\Gamma[p_s,q_s,p_r,q_r]}{n_{pairs}},\mu_c,\tau_c\right) \end{aligned} \tag{2.17}$$

最後的最終相似值：最後，從全域性比對關係中提取各種參數，以計算指紋的相似性分數，參考公式(2.18)。

$$S=S_{lss}\times S_{cluster}\times S_{num} \tag{2.18}$$

2.2.5 FLANN 演算法比對

FLANN (Fast Library for Approximate Nearest Neighbors) 是一種快速近似鄰近搜尋演算法，設計用於大規模高維數據集的快速比對[48]。它在機器學習和計算機視覺中廣泛應用，特別是在圖像檢索和特徵比對領域。

如 Vijayan[49]。研究顯示將快速近似鄰近搜索演算法(FLANN)應用於生物辨識中的特徵比對，結合 SIFT 技術，提升了人臉特徵檢測的準確性與效率。FLANN 在處理高維數據時展現出卓越的比對能力，透過多棵隨機化 KD 樹的算法，顯著減少了錯誤比對，特別是在特徵點方向的比對上，FLANN 能有效處理不同頭部姿勢所造成的特徵點偏移，提升比對準確率。

又如Wang 與 Li[50]研究中提供一個高效的相似圖像檢索系統，針對從不同角度拍攝但場景或物體相同的相似影像，基於內容的影像檢索方法摘取不同影像的像素特徵作為計算影像之間相似性的線索。實驗中使用了 INRIA Holidays 數據集與 ZuBuD 數據集，透過比較圖片顏色的特徵來計算相似性。實驗過程中，丟棄了查詢影像中相似度小於閾值 0.24 的影像。這些研究展示了在處理高維圖像特徵數據時的高效性與準確性，適用於多樣化的圖像檢索場景。

Gowandy、Halim 與 Santoso 也在研究中[51]提出了一種基於指紋識別與射頻識別(RFID)的電子安全投票系統，旨在提高投票系統的安全性與效率。在指紋辨識中，分析了兩種比對算法：暴力比對(Brute force matcher)和快速近似最近鄰庫(FLANN matcher)。FLANN 比對器中使用的算法包括分層 k-means 樹算法和隨機化 k-d 樹算法。分層 k-means 樹算法使用優先佇列，根據每個 k-means 區域與查詢之間的距離，按順序擴展搜索範圍。同時，隨機化 k-d 樹算法通過選擇分割維度構建隨機樹，在樹的每一層將數據分為兩部分，並選擇數據方差最大的維度進行分割，與暴力比對相比，FLANN 比對在處理大型數據集時運行速度更快 。FLANN 比對在處理大規模數據集時的效率與準確性均優於暴力比對。此外，該

系統通過 RFID 標籤與指紋圖像的聯合驗證，顯著縮短了指紋比對的處理時間，增強了整體系統的實用性。

其比對流程可以分為四個主要階段[52] [53]：首先是數據預處理階段，系統會收集待比對的指紋特徵點數據，並將這些圖片中擷取出的特徵點轉換為高維度特徵向量。

第二階段是索引建立階段，系統會構建索引結構，通常多維度的特徵點使用多個 KD 樹的集合來進行。KD 樹(K-dimension tree)存儲 K 維空間中，主要進行快速檢索的樹形數據結構，KD 樹是一種二元樹，表示對 K 維空間的劃分，使用垂直於坐標軸的超平面將 K 維空間切分，形成一個 K 維矩形區域，每個節點對應一個 K 維矩形區域，如圖 2.12 輸入(a)黃色特徵點為樹根(root)，用分割維度線以上的為左子樹，線以下的為右子樹，接著不斷劃分，即如輸出(b)圖示分為 K 個維度

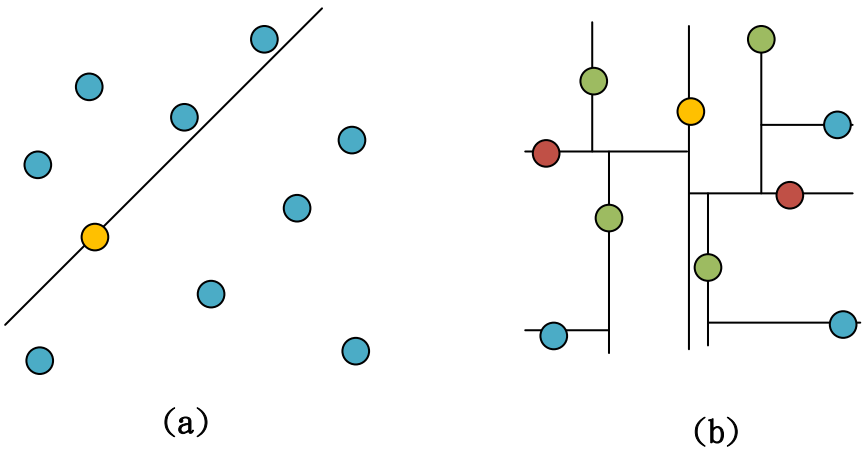


圖 2.12 KD 樹示意圖

第三階段為鄰近距離搜尋階段，搜索過程從樹的根節點開始，比較查詢向量 q 與每個子節點的中心。透過 Hamming 距離最小的子節點繼續搜索，重複此過程直到達到葉節點，然後將葉節點中的所有點加入候選集。接下來進行 Hamming 距離計算，對於篩選出的每個候選項 p，計算其與查詢向量 q 的 Hamming 距離

公式如(2.19)，使用 XOR 操作可以通過位運算高效實現，提高計算速度。最後，系統根據計算出的距離對候選項進行排序，選擇距離最小的 k 個作為最終的 k 近鄰，從而完成整個近似最近鄰搜索過程。

p 表示資料集$P = \{p_1, \ldots, p_n\}$中的一個點(存儲在索引中的一個特徵向量)。

q 表示查詢點(想要找到其最近鄰的特徵向量)。

$$d_{hamming}(p, q) = \sum_{i=0}^{n-1} (p_i \oplus q_i) \tag{2.19}$$

第四階段在結果過濾階段，演算法參考 Lowe 在[54]中提出使用距離比率測試來嘗試消除錯誤比對。計算考慮中的關鍵點的兩個最近比對之間的距離比率，當該值低於某個閾值時，則認為這是一個好的比對。這個比率可以幫助區分模糊比對(兩個最近鄰之間的距離比率接近 1)和區分良好的比對。最後，系統會輸出比對結果，傳回最佳比對的列表，並提供相似度得分與距離。

FLANN 演算法犧牲一些精確度來換取更快的搜索速度，用戶可以根據需求調整速度和準確性之間的平衡[55]。由於其演算法優點，多實驗將其比對方法列入指紋比對研究中。

第三章、　可重構指紋特徵比對系統設計

本章主要基於 MIAT 實驗室提出的方法論執行可重構指紋特徵比對系統架構設計，第一小節主要說明 MIAT 方法論如何將一個複雜的系統，透過問題所需的理論、方法、模型、技術、工具以邏輯化的 IDEF0 階層式流程進行設計，且針對每個功能獨立的模組進行 Grafcet 圖形流程建立離散事件模型，使問題導向解答，第二小節將介紹將複雜的指紋特徵比對系統進行階層模組化設計，因應現今消費性電子硬體產品週期縮短[56]，使複雜的系統為可重構的模組化設計，解決日新月異不同的硬體設備所導致架構重構或修改，第三小節為每個設計的模組進行離散事件建模與說明。

3.1 MIAT 方法論

MIAT 方法論基於 Top-Down 設計方式，階層式模組化的功能架構設計，配合 Grafcet 圖形流程建立離散事件模型，再根據組成的法則實現軟體高階合成[57]，MIAT 架構示意如圖 3.1。綜整以上可知方法論在設計複雜系統有良好的成效，複雜系統通常涉及多領域的知識，包括軟硬體介面設計與結合及團隊合作協調，對於知識管理也有極大的幫助，透過模組的切割使個別工作降低重複性，使架構可以為各種新產品提出快速開發流程的解決方案。

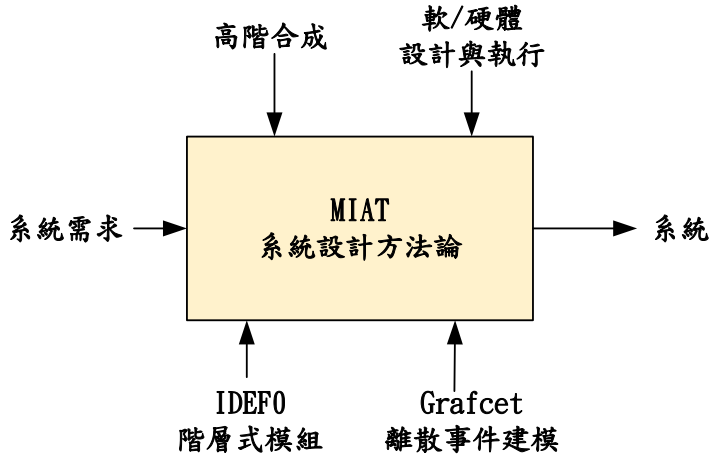


圖 3.1 MIAT 系統設計方法論

本論文將指紋特徵比對系統用此方法執行方法論系統設計，以解決開發流程可能面臨介面協調困難及研發控管不易問題，從而實現適應硬體規格變化的需求，所需改變指紋特徵比對演算法，並設計出可重構指紋特徵比對系統架構。

3.1.1 IDEF0 階層式模組化設計

IDEF0 是一種功能建模的方法，大多應用於複雜系統的設計、分析。其階層式模組化方法提供了一種有效的架構，用於系統功能的分解和組成。

首先逐層分解系統的主要功能，使每個子功能均能被詳細描述和分析。這種 Top-Down 的分解過程有助於理解系統的複雜性，並提供一種系統化的方法來管理各個功能模組。系統的每個功能模組可以進一步分解為更小的子模組。這種逐層細化的過程使每個功能模組的細節逐漸展現，從而達到全面描述系統功能的目的。每個功能模組都通過 IDEF0 標準符號表示其輸入(Input)、輸出(Output)、控制(Control)和機制(Mechanism)如圖 3.2。這些方塊定義了功能模組的邊界和內部結構，並都會以代號標記，由最高層 A0 依序向下命名 A1、A2、A3…，有助於精確描述模組的功能和相互關係。

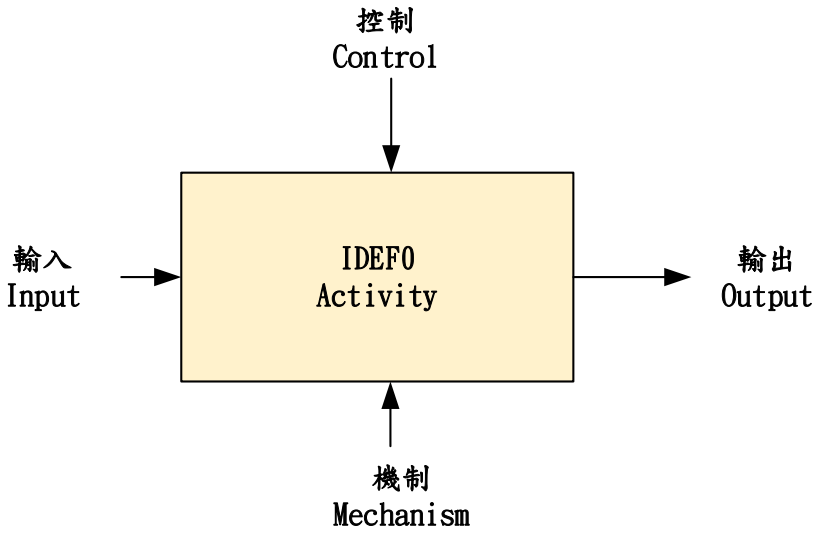


圖 3.2 IDEF0 功能模組

IDEF0 強調模組化設計，如圖 3.3 為例使用 IDEF0 表示 MIAT 方法論架構，每個模組作為獨立的單元進行開發和測試。這種設計方法提高了系統的靈活性和可維護性，便於對系統進行修改和擴展。此外，每個功能模組具有明確的邊界和定義，確保系統運行的穩定性和可靠性。

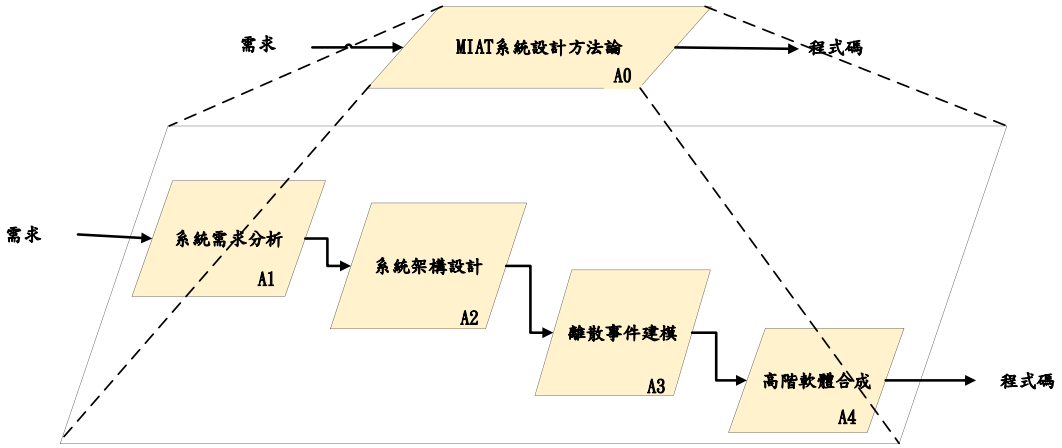


圖 3.3 IDEF0 表示 MIAT 方法論架構

階層式模組化設計有助於團隊分工合作，各模組之間的定義清晰明確，使來自不同領域的專家能夠有效合作，進而提高系統設計的整體效率，並由於模組化設計的特點，系統的各部分可以獨立開發和測試，縮短了開發周期，同時也便於後續的系統擴展和升級，為各種新產品的快速開發提供解決方案。

3.1.2 Grafcet 離散事件建模

離散事件建模能夠詳細模擬和分析複雜系統中的事件和狀態變化，分析系統中事件的發生和這些事件對系統狀態的影響，因為這些事件在時間軸上是離散的，即在特定的時間點發生，而非連續發生，通過這種建模技術，可以深入理解系統行為。

Grafcet 是一種用於描述和設計控制系統。通過其圖形化和結構化的表示方法，主要組成為狀態(State)、轉移條件(Condition)、有向性的連結(Directed connection)，狀態是指離散系統所處的具體情況，行為描述了系統在該狀態下執行的動作，轉移條件則是系統從一個狀態轉換到下一個狀態所需的條件，簡易 Grafcet 範例為如圖 3.4。

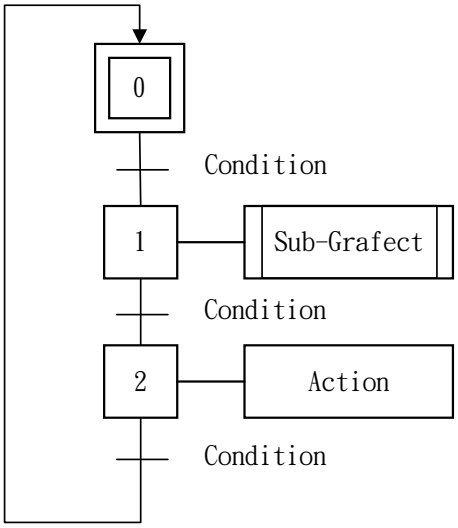


圖 3.4 Grafcet 範例

Grafcet 狀態可分為三類分別為初始狀態(Initial state)、執行狀態(Inactive state)、閒置狀態(Active state)如 圖 3.5。初始狀態使用雙實線方塊呈現，執行狀態為當下工作執行狀態，使用單實線方塊並由黑點標示，閒置狀態為未執行狀態，使用單實線並無黑點標示。其中階層化設計中，行為方塊(Action)及子系統(Sub-Grafcet)分別表示狀態方塊執行動作或者執行內容較為複雜則可向下劃分子系統，使系統架構更為清晰。

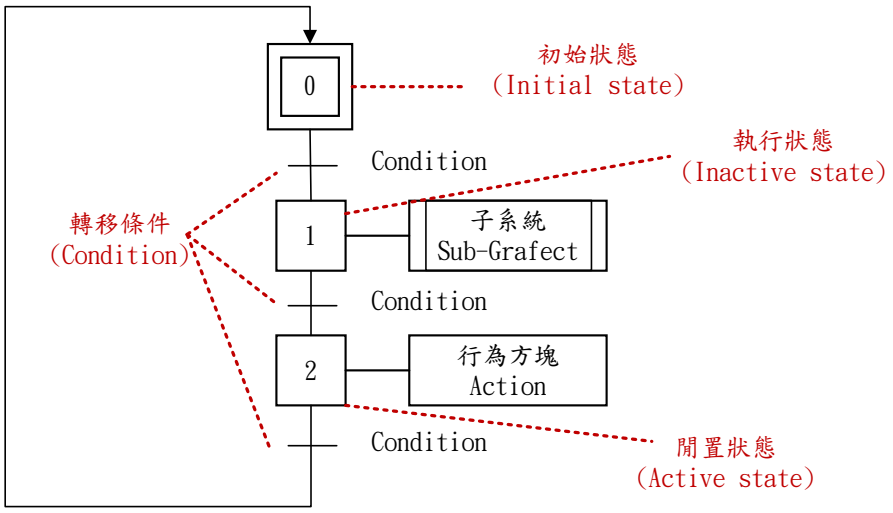


圖 3.5 (a)Grafcet 狀態表示 (b)Grafcet 行為方塊

綜整上述使用 Grafcet 進行離散事件建模可以使系統設計更易於分析，這有助於理解和溝通行為，並結構化設計可以使系統更有條理，減少錯誤或遺漏的可能性。

3.2 LLM 輔助生成 Grafcet action 程式碼

大型語言模型(Large Language Models, LLMs)是基於深度學習技術的自然語言處理模型，擁有數十億參數，能夠理解並生成自然語言文本。這些模型利用龐大的語料庫進行訓練，通過學習語言的結構和語義來進行文本生成和理解。LLM 的發展促使其在多個領域中取得了突破性進展，特別是在自動生成程式碼方面，展現了極大的潛力和應用價值。

大型語言模型的出現代表了自然語言處理領域的重要進步。以 GPT(Generative Pre-trained Transformer)系列模型為代表，ChatGPT o1 模型是 OpenAI 推出的新一代大型語言模型，採用大規模強化學習與「思考鏈」(Chain of Thought)技術進行訓練，模型在生成答案前執行多步推理，並在輸出中顯示完整的推理過程[58]。該模型旨在複雜推理任務中提升準確性，還使其回答具備更

強的可解釋性，同時增強模型對安全性與指令遵從的能力。在訓練過程中，o1 模型利用多元數據來源，包括公開的網頁數據、科學文獻，以及合作夥伴提供的專有數據集，確保模型的知識覆蓋範圍廣泛且專業。此外，OpenAI 開發了嚴格的數據處理管道，通過篩選、拒絕有害內容和安全分類器過濾，降低訓練數據中個人信息或敏感內容的風險[59]。

這些模型的強大性能主要來自於兩個方面：一是龐大的訓練數據，涵蓋了各種主題和語言；二是深度神經網絡架構，允許模型學習複雜的語言模式和結構。通過這些優勢，LLM 在各種自然語言處理任務中，如文本生成、機器翻譯、問答系統等，都展現出了卓越的性能。

然而，自動生成程式碼是 LLM 的一個重要應用領域，旨在將自然語言描述轉換為可執行的程式碼。這一技術在提高開發效率、減少錯誤以及促進創新方面具有重要意義。

LLM 還能生成高質量的程式碼，提高可讀性和可維護性。Liu 等人(2023)的研究系統性地評估了 ChatGPT 在生成程式碼方面的表現，發現其在正確性、可理解性和安全性方面具有較好的性能[60]。LLM 生成的程式碼不僅能提高效率，還能減少人為錯誤。Huang 等人(2023)提出了一個針對程式碼生成任務的偏見評估框架，並指出 LLM 生成的程式碼中存在一定比例的偏見，但通過適當的策略可以有效減少這些偏見[61]。

3.3 可重構指紋特徵比對系統架構設計

本論文研究主要提出可依照使用者需求選擇特徵比對演算法的模組化系統架構，透過 MIAT 方法論進行 IDEF0 模組階層規劃、Grafcet 離散事件建模，進而整合至指紋辨識系統(A0)中，如圖 3.6，完成使用者可配和各場景及需求切換不同演化法。

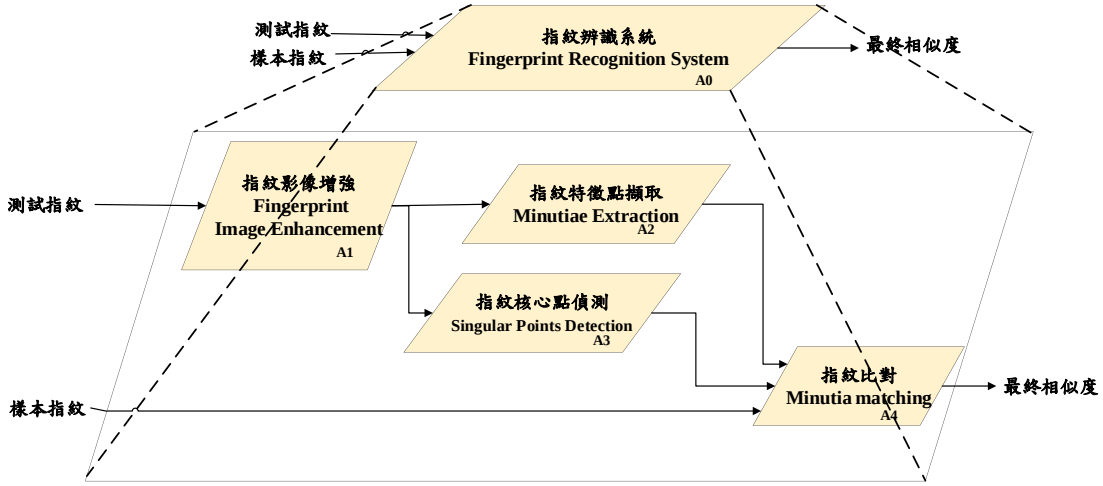


圖 3.6 指紋辨識系統 IDEF0 模組階層規劃

指紋特徵比對模組為指紋辨識系統(A4)模組中的一個模組，並進一步分層為三個子系統，分別為旋轉排除、線段方向排除、特徵比對模組，首先上層輸入樣本特徵及測試特徵，進行旋轉排除(A41)演算法排除後，將解析傳遞予線段方向排除(A42)進行演算法排除，經處理後傳至特徵比對(A43)進行比對分析，最終取得相似度。

3.3.1 指紋比對模組(A4)

本研究中，使用 IDEF0 技術來進行指紋比對模組(A4)規劃，如圖 3.7 所示。該子模組包括三個功能模組，分別是線段旋轉排除(A41)、線段方向排除(A42)和特徵點比對(A43)。

首先，線段旋轉排除(A41)功能模組負責根據指紋的旋轉特徵進行排除。這一過程的輸入是指紋圖像資料和相關的旋轉特徵數據，輸出為已經排除不符合旋轉條件的指紋數據。接下來是線段方向排除(A42)功能模組。該模組對指紋線段的方向和特徵進行分析和排除。輸入為指紋圖像資料和相關的線段特徵數據，輸出為已經排除不符合條件的線段數據。

最後是特徵點比對(A43)功能模組。該模組根據前述排除過程得到的指紋數據進行詳細的特徵比對。輸入為已經排除不符合條件的指紋數據和線段數據，輸出為最終的相似度值。

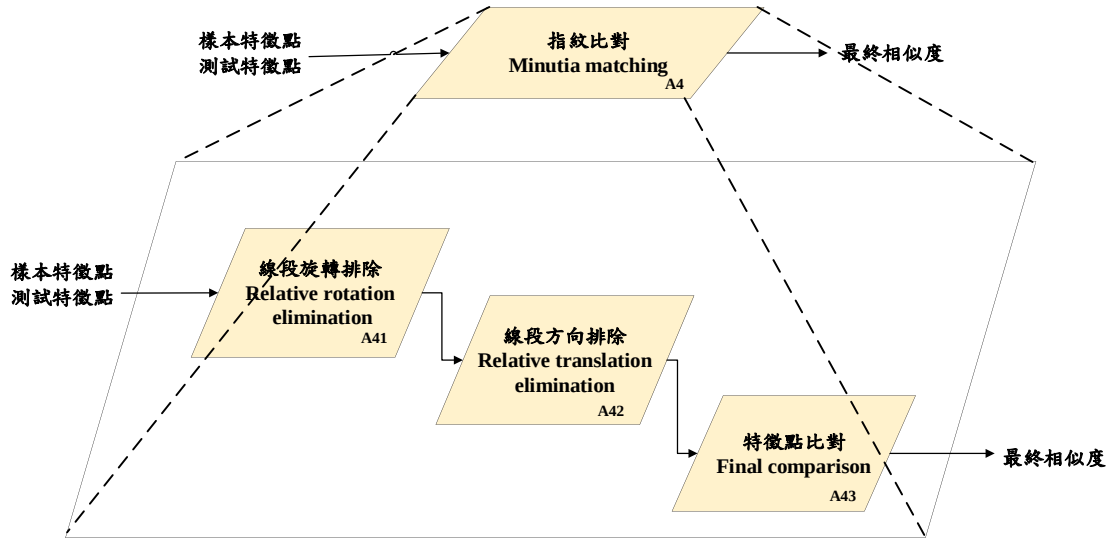


圖 3.7 指紋比對模組(A4)

3.3.2 線段旋轉排除模組(A41)

本研究使用 IDEF0 來規劃旋轉排除模組如圖 3.8 所示，該子模組分別為七個功能模組為兩線段相似值比較(A411)、特徵點種類(A412)、特徵點曲率(A413)、特徵點的平均脊密度(A414)、構成旋轉直方圖(A415)、旋轉確認(A416)、旋轉排除(A417)。

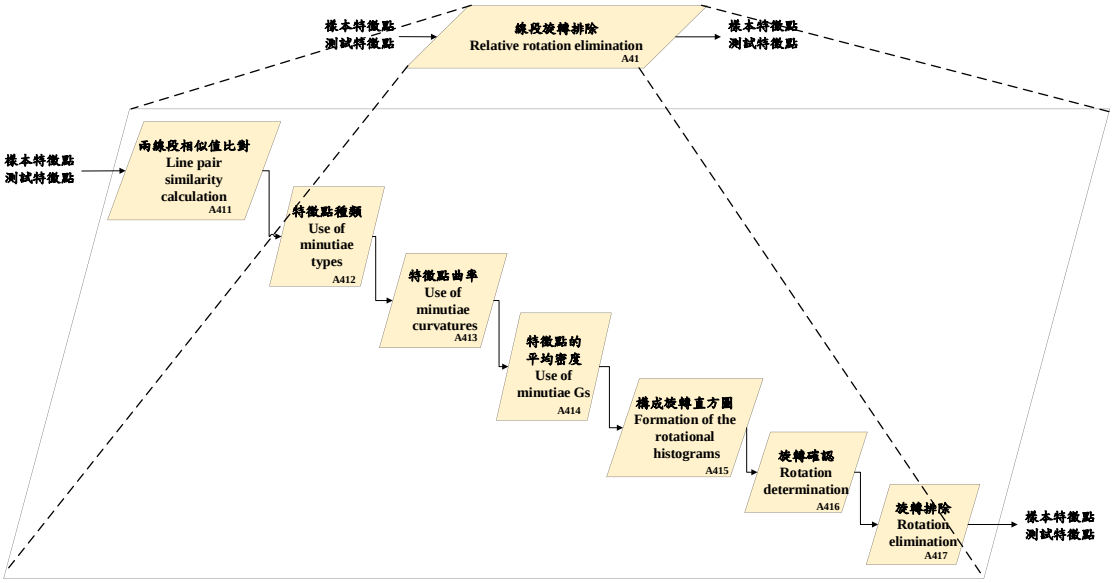


圖 3.8 線段旋轉排除模組(A41)

在兩線段相似比較(A411)功能模組中，系統比較兩條指紋線段的相似性。這一步的輸入為指紋線段資料，輸出則是相似性評估結果。接下來，進行特徵點種類(A412)的識別和分類。系統輸入指紋特徵點資料，通過系統識別並分類指紋中的特徵點類型，輸出為特徵點類型列表，依據特徵點識別標準進行控制。隨後，系統計算特徵點的曲率(A413)。這一步驟以指紋特徵點資料為輸入，通過讀取特徵點的曲率數值進行分類。特徵點的平均脊密度(A414)計算是下一步驟。系統讀取特徵點資料，分類出特徵點的平均脊密度值。

在完成上述計算後，系統根據特徵點和線段相似性結果，構成旋轉直方圖(A415)。這一步的輸入是相似性評估結果和特徵點資料，輸出則為旋轉直方圖。

旋轉確認(A416)功能模組負責確認是否尚有未處理的特徵點。系統以旋轉直方圖為輸入，輸出確認的旋轉點，通過比對系統進行確認。

最後，旋轉排除(A417)模組排除不符合條件的旋轉點。確認的旋轉點作為輸入，系統進行排除過程，輸出剩餘的旋轉點，由排除模組實現，並依據排除標準進行。

3.3.3 線段方向排除模組(A42)

線段方向排除模組(A42)如圖 3.9 該子模組分別為四個功能模組為線段方向差異測試(A421)、構成平移直方圖(A422)、線段判斷(A423)、線段排除(A424)。在線段方向差異測試(A421)功能模組中，系統進行兩條指紋線段的方向差異測試。這一過程的輸入是指紋線段資料，輸出為方向差異測試結果。接著，根據方向差異測試的結果，系統在構成平移直方圖(A422)模組中生成平移直方圖。這一步驟的輸入是方向差異測試結果，輸出則為平移直方圖。

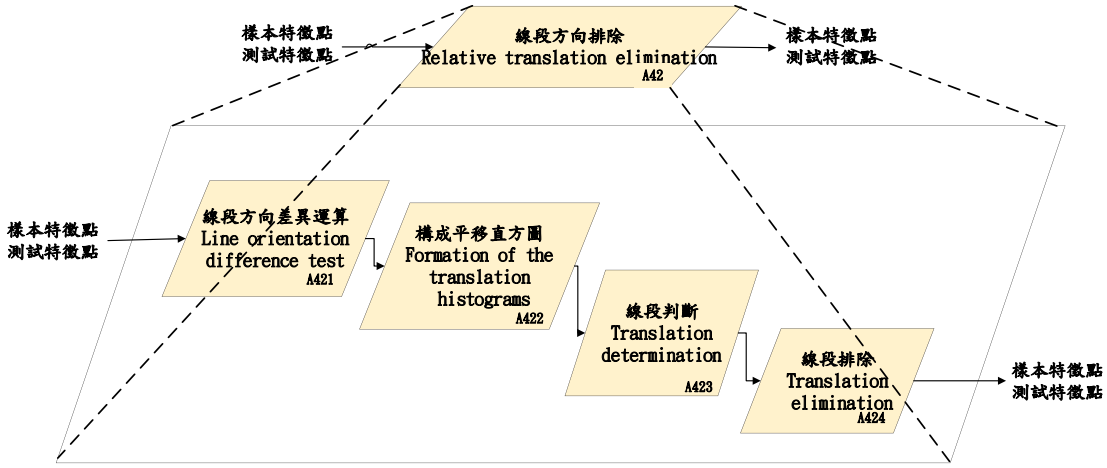


圖 3.9 線段方向排除模組(A42)

隨後，在線段判斷(A423)功能模組中，系統根據平移直方圖進行線段判斷。系統以平移直方圖為輸入，輸出為線段判斷結果。該過程由判斷系統實現，並依據判斷標準進行控制。

最後，在線段排除(A424)功能模組中，系統根據判斷結果排除不符合條件的線段。確認的線段判斷結果作為輸入，系統進行排除過程，輸出為剩餘的線段。該功能由排除模組實現，並依據排除標準進行控制。

3.3.4 特徵點比對模組(A43)

特徵點比對(A43)如圖 3.10 該子模組分別為三個功能模組為線段端點差異測試(A431)、不良特徵點比對排除(A432)、相似度計算(A433)。

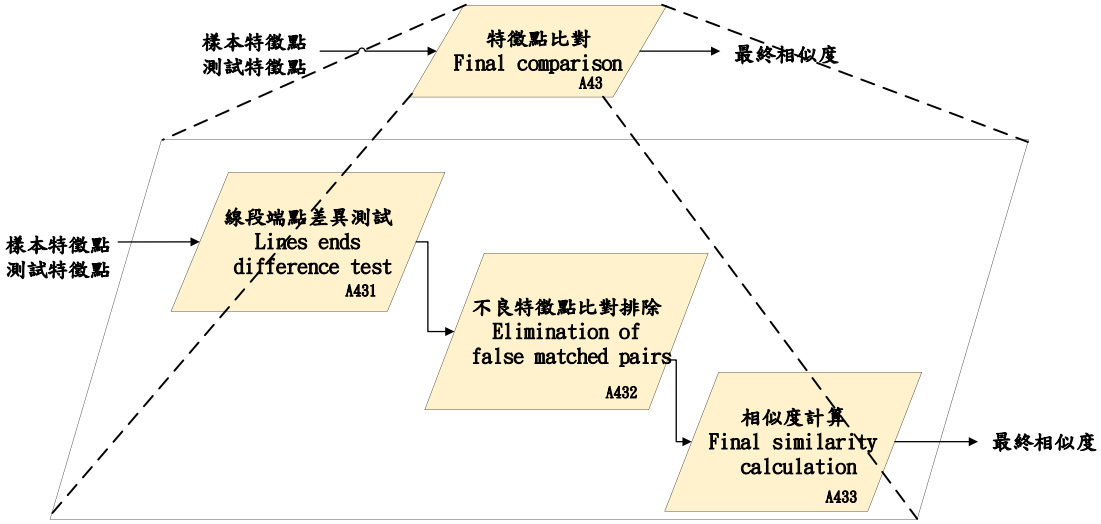


圖 3.10 特徵點比對模組(A43)

首先，在線段端點差異測試(A431)功能模組中，系統對兩條指紋線段的端點差異進行測試。這一過程的輸入是指紋線段資料，輸出為端點差異測試結果。

接下來，不良特徵點比對排除(A432)功能模組負責根據端點差異測試結果，消除不符合條件的錯誤比對。系統輸入端點差異測試結果，輸出為不良特徵點比對排除後的結果。這一步驟依據錯誤比對排除標準進行控制，由比對排除模組來實現。

最後，在相似度計算(A433)功能模組中，系統根據不良特徵點比對排除後的結果，計算指紋之間的相似度。系統輸入不良特徵點比對排除的結果，輸出為相似度值。相似度計算過程依據相似度計算標準進行。

3.4 離散事件建模

在本研究中，使用離散事件模型來描述指紋特徵比對的過程，如圖 3.11 所示。首先，系統輸入測試指紋並對其進行影像增強處理。這一步在離散事件模型中被建模為一個初始事件，該事件完成後，會觸發兩個平行的事件：指紋核心點偵測和指紋特徵點擷取。指紋核心點偵測和指紋特徵點擷取是並行進行的，這意味著這兩個過程可以同時執行而不互相干擾。在核心點處理過程中，系統識別並處理指紋的核心點，這對後續的比對至關重要。特徵點擷取過程則負責識別和擷取指紋中的細節特徵點，如特徵點和端點。

當指紋核心點偵測和指紋特徵點擷取均完成後，這兩個平行事件會觸發下一個關鍵事件，即指紋比對。指紋比對是整個指紋辨識系統過程的最後一步，在這一步中，系統將樣本指紋與測試指紋的特徵點進行詳細比對計算並輸出相似度數值。該數值表示樣本指紋與測試指紋之間的相似程度。

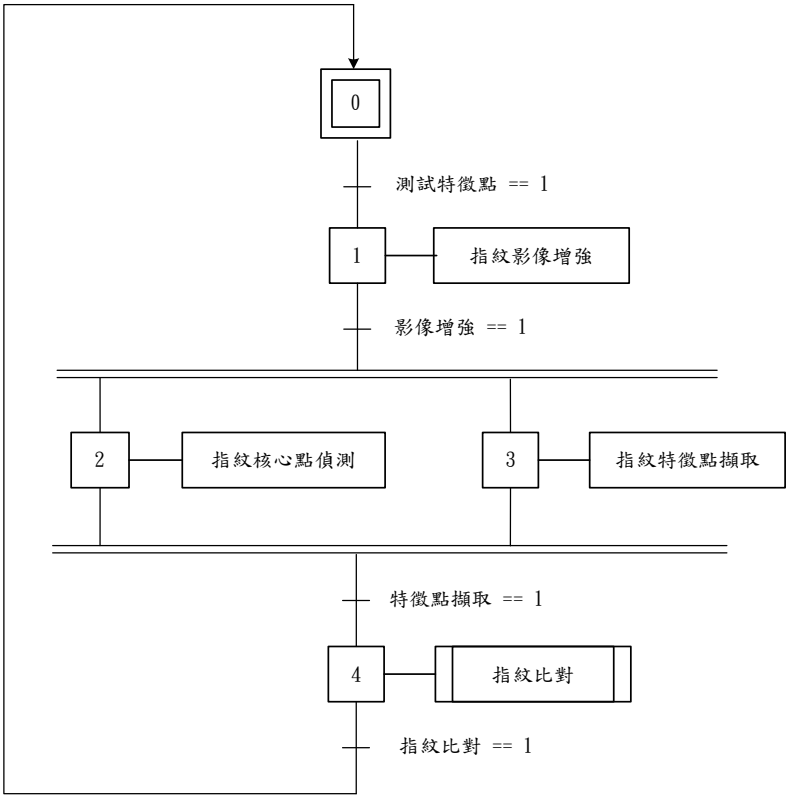


圖 3.11 指紋辨識系統離散事件建模

3.4.1 指紋特徵比對離散事件模型

指紋比對在進行子系統擴充建立離散事件模型，如圖 3.12 所示，首先，將樣本指紋及測試指紋輸入系統。隨後對這兩組指紋進行旋轉角度的比對，排除因旋轉角度不同而不配對的指紋，以減少角度偏差導致的誤差。這一過程在離散事件模型中被建模為一個事件，該事件在角度比對完成時觸發，並輸出比對結果。旋轉比對完成後，系統進一步對指紋的線段方向進行比對排除。這一過程在離散事件模型中被建模為另一個事件，該事件在線段方向比對完成時觸發，進而輸出比對結果。這一過程旨在通過精細比對指紋的線段特徵，提高比對的準確度。在線段方向比對完成後，系統會對指紋的特徵進行詳細比對。最終計算並輸出指紋之間的相似度數值。該數值表示樣本指紋與測試指紋之間的比對程度，並被模型中的最後一個事件所觸發。

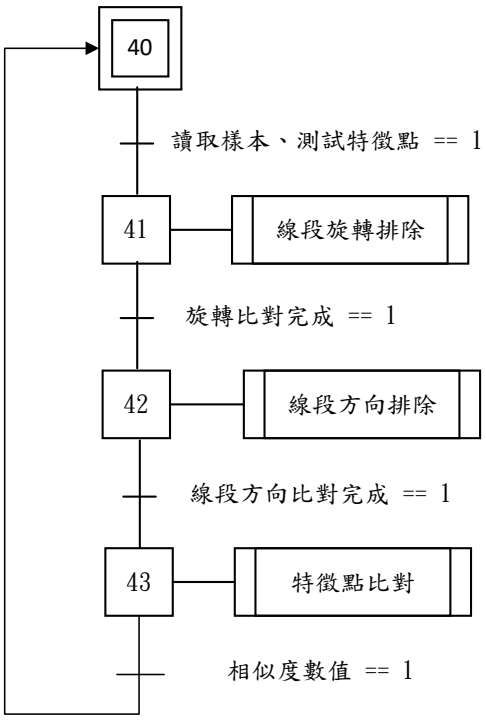


圖 3.12 指紋比對離散事件建模

3.4.2 線段旋轉排除離散事件模型

將指紋特徵比對過程中的旋轉角度比對排除子系統進行離散事件建模，如圖 3.13 所示，首先計算兩線段指紋線段的相似性，這一過程被建模為一個初始事件，完成後觸發特徵點的種類計算，識別並分類不同類型的特徵點。系統計算特徵點的曲率，確定指紋紋路的彎曲程度，接著計算特徵點平均脊密度，這些計算結果有助於更精確地進行指紋特徵比對，系統將這些結果用於建立旋轉直方圖。該事件完成後驗證是否尚有旋轉點未計算，若存在未計算的旋轉點，系統將重複上述計算過程，直至所有旋轉點計算完成。確認所有旋轉點的計算完成，在進行旋轉角度判斷，以排除不符合條件的旋轉點。這一過程確保只保留最符合條件的旋轉點，從而提高比對的準確性。

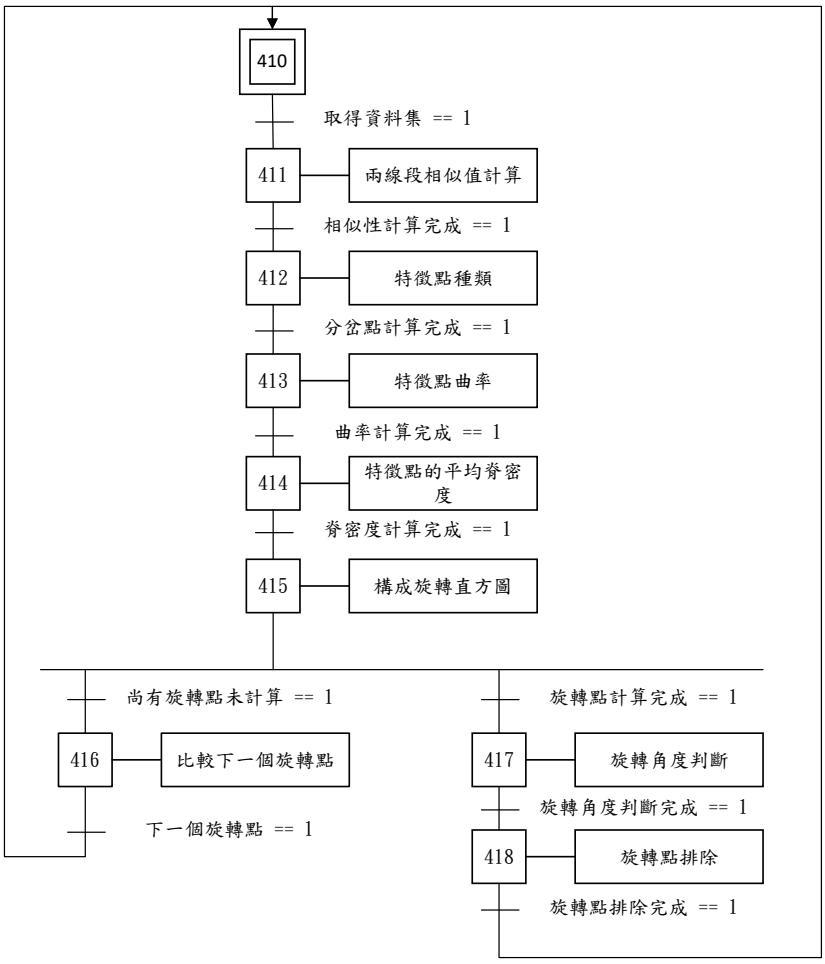


圖 3.13 旋轉排除離散事件建模

3.4.3 線段方向排除離散事件模型

首先，系統對兩條指紋線段的方向差異進行運算，完成後觸發下一個事件，即構成平移直方圖。根據方向差異的計算結果，該直方圖用於分析和比對指紋特徵的線段方向。

接著，系統檢查是否存在尚未運算的線段。如果存在未運算的線段，系統將重複方向差異運算和構成平移直方圖的過程，直至所有線段的運算完成。這一過程在離散事件模型中被建模為一個迭代過程，確保對所有指紋線段進行全面的分析和比對，並當所有線段的運算完成後，系統進行線段比對排除。

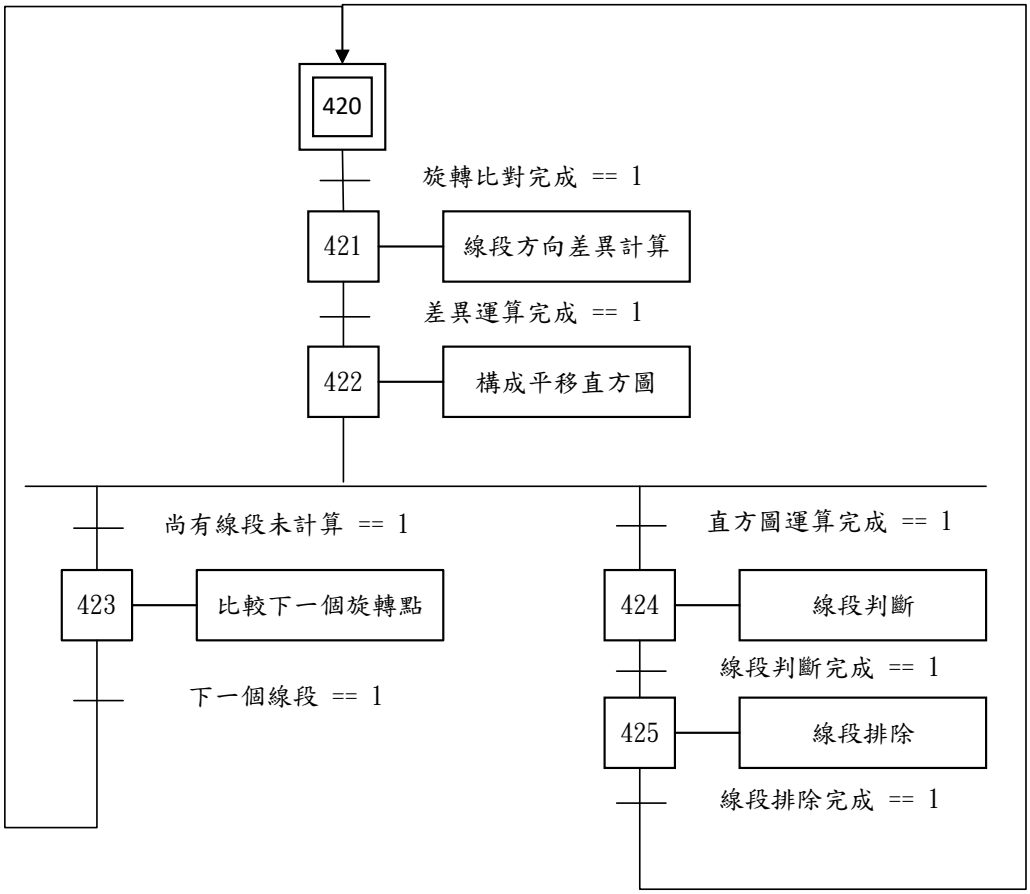


圖 3.14 線段方向排除離散事件建模

3.4.4 特徵點比對離散模型

首先針對對兩條指紋線段的端點差異進行測試，這一步被建模為初始事件，完成後觸發不良特徵點比對排除事件。不良特徵點比對排除過程中，系統根據端點差異測試的結果，排除那些不符合條件的線段，以減少比對過程中的誤差和錯誤配對。

接下來，檢查是否存在尚未計算的線段端點，如果存在未計算的線段端點，系統將重複端點差異測試和不良特徵點比對排除的過程，直至所有線段端點計算完成，確保對所有指紋線段端點進行全面的分析和比對。

當所有線段端點計算完成後，系統進行相似度計算。此步驟根據前述不良特徵點比對排除的結果，計算樣本指紋與測試指紋之間的相似度。這一過程被建模為最終事件，並輸出相似值，該相似值表示指紋之間的比對程度。

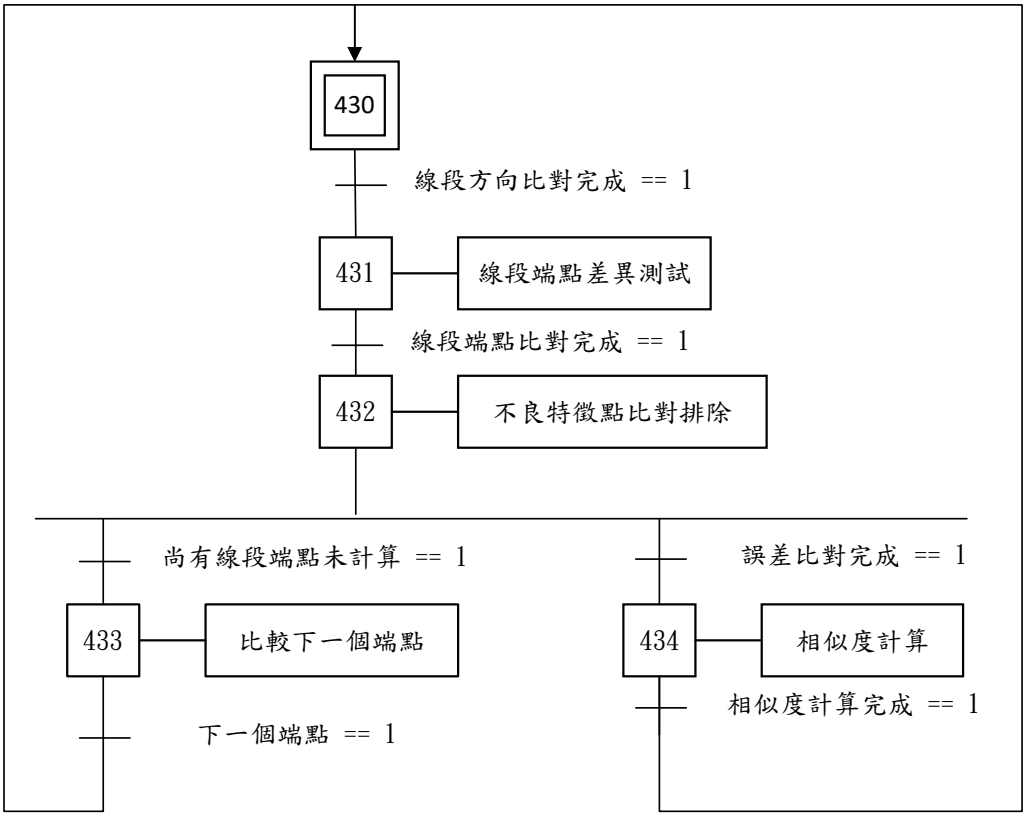


圖 3.15 特徵比對離散事件建模

第四章、　可重構指紋特徵比對實驗

本章首先介紹實驗環境、所使用的 FVC2004 資料庫、評估方法、指紋特徵比對方法。最後，將進行實驗結果的分析，對指紋比對的性能進行評估，並驗證指紋特徵比對架構的有效性。本章旨在通過實驗驗證指紋比對系統中模組化設計的性能，並測試大語言模型(LLM)生成系統架構程式碼的準確性與效率。實驗目的是確認系統可快速更換演算法及指紋影像情況下的穩定性與表現。

4.1　實驗環境

本研究的實驗環境如表 4.1，建立在 Windows 作業環境，編譯器使用 Visual Studio 提供了多樣的工具和功能，能夠支援建立 C++主控台應用程式。應用程式的編譯和測試工作也都在這個環境中完成。在數據庫方面，使用的是 Liao[62]研究中的指紋特徵點擷取程式，將圖片中的特徵點擷取出來，並且輸出成 CSV 檔。

表 4.1 實驗環境規格

CPU	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
記憶體	16 GB
作業系統	Microsoft Windows 10
編譯器	Visual studio

4.2　資料描述

FVC2004(Fingerprint Verification Competition 2004)是指紋識別領域內的重要資料集之一，旨在為學術界和工業界提供一個標準化的平台，用於評估指紋識別算法的性能。FVC2004 資料集包含四個不同的資料庫(DB1、DB2、DB3、DB4)，每個資料庫包含不同影像品質和類型的指紋圖像[63]，如表 4.2。

表 4.2 FVC2004 資料集說明

資料庫	感應器型別	圖片解析度	圖片尺寸
DB1	光學感測器”V300”	640x480	500dpi
DB2	光學感測器”U.are.U 4000”	328x364	500dpi
DB3	熱掃描感測器	300x480	512dpi
DB4	合成指紋圖片	288x384	約 500dpi

FVC2004 資料集的特點包括多樣性、真實性、合成數據和標準化。該資料集包含了來自不同掃描儀類型和不同品質的指紋圖像，提供了豐富的多樣性。前三個數據庫是真實的指紋圖像，真實反映了在實際應用中的表現，例如按壓感測器垂直位置的不同、手指壓力不同、手指乾燥程度，而第四個數據庫由合成指紋圖像組成，如圖 4.1 所示。每個資料庫庫包含 10 位指紋志願者(編號 101~編號 110)，每個志願者提供每隻手指 8 個指紋印，共 80 個指紋圖片，並且每個圖片都提供標準化的格式和命名規則，方便比較不同演算法的性能。

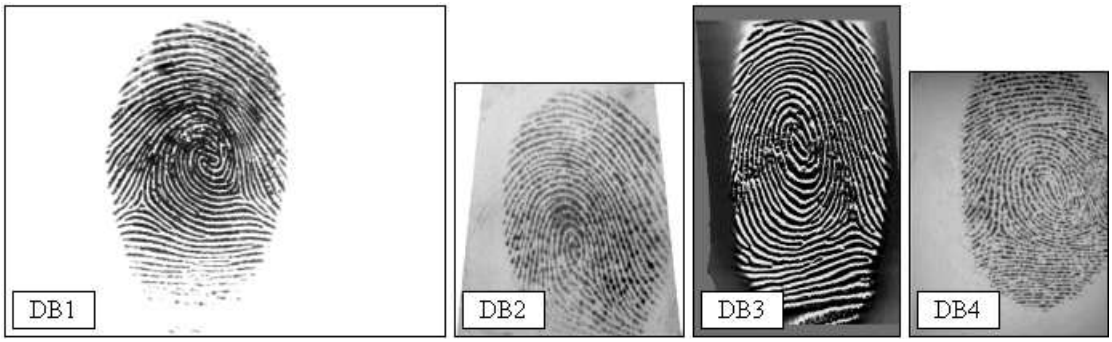


圖 4.1 每個資料庫指紋圖片範例

4.3 基於 LLM 的 Grafcet 框架生成方法：實驗與應用分析

本節探討大語言模型(LLM)輔助生成 Grafcet action 的實驗過程及結果，特別是用於生成指紋特徵比對的架構。隨著人工智能技術的發展，LLM 的自動程式碼生成能力已被廣泛研究與應用[64]。LLM 擁有學習大量程式碼數據的能力，能夠根據不同的需求生成高效、正確且符合語法規則的程式碼[65]。在本研究中，測試 LLM 自動生成程式碼是否能夠達到與人工編寫程式碼相同的結果，並分析其在實驗中的準確性、效率和應用可行性。

4.3.1 LLM 模型

在多種 LLM 中，本研究選擇了 GPT 作為程式碼生成的工具。GPT 基於其在理解語言邏輯和生成結構化內容方面的優勢。GPT 模型能夠有效理解多步驟流程的描述，並將其準確轉化為程式碼結構[66]。這一點在 Grafcet 流程的生成中尤為重要，因為該流程包含大量的條件判斷與狀態轉移需求。GPT 能夠根據簡明的指令生成符合 Grafcet 規格的 C++程式碼，並支援使用者進行二次編輯或優化。

4.3.2 Grafcet Action 生成流程

為了將指紋比對系統的 Grafcet 流程圖轉換為符合需求的 C 程式碼，本研究運用 MIAT 方法論設計了一套 Grafcet 的流程，整個生成流程被劃分為輸入格式定義和生成流程設計兩個部分，以確保程式碼的邏輯準確性及生成過程的高效性，以下將按照各項設計策略的架構逐步說明。

輸入格式定義此階段主要是為 GPT 提供生成程式碼所需的邏輯規範，確保模型能精確理解 Grafcet 圖形的流程和各步驟的狀態轉移條件。首先 Grafcet 圖

形描述格式根據 Grafcet 圖形設定，將每個步驟的狀態和轉換條件進行結構化描述，並提供給 GPT 作為參考格式。這樣的描述格式不僅定義了程式碼所需的資料結構，也使 GPT 能準確掌握圖形中的轉換邏輯。

接著步驟轉換條件描述針對 Grafcet 的各步驟轉換條件進行詳細定義，例如在「旋轉角度比對」階段設定的閾值及「線段方向比對」的偏差範圍等。這些條件為 GPT 提供了步驟間狀態切換的依據，使其能在程式碼生成過程中，準確設定各步驟的狀態判斷。GPT 根據這些描述生成程式碼中的 if-else 判斷條件，以確保每個步驟的執行符合 Grafcet 圖形的要求，並符合流程的設計邏輯。

生成流程設計是 GPT 生成程式碼的具體步驟，透過前處理、Prompt 對話設計和結果後處理方法，確保生成結果。在前處理步驟時，GPT 生成程式碼之前，先對 Grafcet 圖形進行結構分析，將每個步驟的轉換條件與動作需求具體化，並轉化為清晰的提示詞，讓 GPT 能準確把握圖形邏輯。前處理還包括將每步驟的狀態切換需求定義為條件式格式，便於 GPT 在程式碼生成時依循這些條件進行判斷。

第二步驟 Prompt 對話設計，為每個步驟設計特定的 Prompt，並將 Grafcet 的邏輯結構嵌入對話，使 GPT 能理解每步邏輯的需求和條件。對話設計遵循 if-else 判斷結構的基礎邏輯，讓 GPT 針對每個步驟的判斷條件生成程式碼，從而實現正確的狀態轉移。

第三步驟結果處理方法，在 GPT 生成程式碼後，會對生成結果進行處理，檢查程式碼的邏輯準確性與每步需求是否符合 Grafcet 圖形。例如，若發現 GPT 生成的程式碼在某些閾值設定上存在不足，會進一步優化 Prompt 重新生成。此外，要求 GPT 自動添加註解，詳細說明每個步驟的邏輯目的與條件判斷，進一步增強程式碼的可讀性與維護性，詳見附錄一。

4.3.3 生成結果驗證

在本研究中，為了確保程式碼符合預期邏輯並具備實際功能，設計了兩個驗證階段，邏輯正確性驗證與功能性驗證。這兩個驗證階段相輔相成，分別對程式碼的內部邏輯和整體功能進行全面檢查，保證最終系統的運行可靠性。邏輯正確性驗證側重於程式碼內部邏輯的一致性與準確性。該過程包括兩個主要方面：狀態轉換檢查與條件判斷驗證。

狀態轉換檢查對程式碼中的每個步驟進行檢查，確保狀態轉換符合 Grafcet 圖的預期設計。具體而言，狀態轉移過程中應符合 Grafcet 中的流程邏輯，從一個狀態進入下一個狀態時，需要符合其設置的觸發條件。若在某狀態中缺少必要的轉換或出現無效轉換，將會對此進行記錄並修正，以保證程式碼的邏輯嚴謹性。

條件判斷驗證在每個條件判斷部分，檢查 if-else 結構是否正確地執行對應的條件邏輯。例如，在指紋旋轉比對的步驟中，確認 GPT 生成的程式碼是否正確地執行旋轉角度比對並設置適當的閾值條件。每個條件的閾值或範圍應符合系統設計需求，以避免錯誤判斷或非預期結果。條件判斷驗證確保每一判斷邏輯均符合系統需求並能正確驅動狀態轉換。實際 Prompt 執行結果如下圖 4.2。

(a) (b)

圖 4.2 (a)輸入 Prompt 產生狀態 0-4 (b)實際產生狀態 0-4 的程式碼

在 GPT 產生狀態 4 之後，接續下 40 的 Prompt，並且產生的程式碼順利融合進前述狀態 4 子涵式內，如下圖 4.3。

(a) (b)

圖 4.3 (a)輸入狀態 40-43 的 Prompt (b)實際產生狀態 40-43 的程式碼

功能性驗證在程式碼內部邏輯驗證的基礎上進行，重點在於整體功能是否符合預期，主要通過整合測試方法來進行驗證。在整合測試中，將各步驟的程式碼組合在一起，並模擬實際指紋比對的情境輸入，觀察系統在不同輸入情況下的運行結果。每次測試需檢查系統是否能正確完成各比對階段的狀態轉換並產生準確的比對結果。若整合測試發現功能性問題如下圖 4.4，則將對 Prompt 和生成的程式碼進行調整，重新生成直至達到功能性驗證的標準，如圖 4.5。

等等我發現有錯誤，在415到416或417的時候需要判斷輸入的變數

感謝提醒！在狀態 **415** 中，需要根據輸入條件判斷是進入 **416** 還是 **417**，以下是修正的部分程式碼，更新狀態 **415** 的邏輯並正確處理條件判斷。

更新後的程式碼（修正 415 至 416/417 的判斷）

```
// 狀態 41 子流程狀態機
void grafcet41()
{
    if (X410 == 1 && get_data_410 == 1)
    {
        X410 = 0;
        X411 = 1;
        get_data_410 = 0;
        return;
    }
```

圖 4.4 執行修正 Prompt

```
        X410 = 0;
        X410 = 1;
        rotation_point_exclusion_completed_410 = 0;
        return;
    }
}
```

修正後的條件邏輯

1. **狀態 415**：
   - 如果 `rotation_points_remaining_for_calculation_415 == 1`，進入狀態 **416**。
   - 如果 `rotation_point_calculation_completed__415 == 1`，進入狀態 **417**。
2. **狀態 416**：
   - 處理下一個旋轉點後，回到 **410**。
3. **狀態 417**：
   - 確定旋轉角度後進入 **418**。

圖 4.5 產生修改後的程式碼

透過以上步驟將離散事件的系統架構，結合 GPT 產生的程式碼，完成高階合成的程式碼，架構流程圖如圖 4.6。

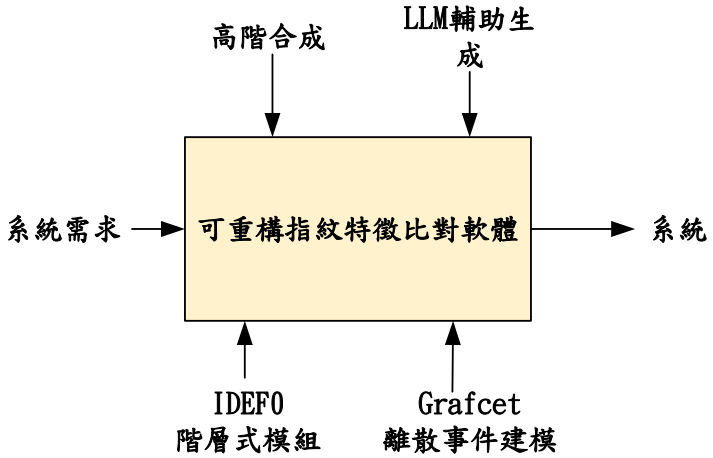


圖 4.6 架構流程圖簡易

下圖 4.7 內容展示了指紋比對模組軟體中 0 至 410 狀態的高階軟體合成程式碼及其狀態轉移結果，其他系統架構相關的程式碼詳見 附錄二，其中包含高階軟體合成的完整實現。完成高階合成後，將進一步進行指紋比對軟體的驗證工作，以確保各狀態的運行邏輯與功能的完善性。

```
Grafcet simulation starting...
State 0: Initializing...
State 1: Performing image enhancement...
State 2: Core point processing...
State 3: Feature point extraction...
State 4: Entering sub-process...
State 40: Reading sample/test feature points...
State 4: Entering sub-process...
State 41: Performing rotation exclusion...
X410 = 1,X411 = 0,X412 = 0,X413 = 0,X414 = 0,X415 = 0,X416 = 0,X417 = 0,X418 = 0
State 410: Getting data...
State 4: Entering sub-process...
State 41: Performing rotation exclusion...
X410 = 0,X411 = 1,X412 = 0,X413 = 0,X414 = 0,X415 = 0,X416 = 0,X417 = 0,X418 = 0
State 411: Calculating line similarity...
State 4: Entering sub-process...
State 41: Performing rotation exclusion...
X410 = 0,X411 = 0,X412 = 1,X413 = 0,X414 = 0,X415 = 0,X416 = 0,X417 = 0,X418 = 0
State 412: Calculating bifurcation point...
State 4: Entering sub-process...
State 41: Performing rotation exclusion...
X410 = 0,X411 = 0,X412 = 0,X413 = 1,X414 = 0,X415 = 0,X416 = 0,X417 = 0,X418 = 0
State 413: Calculating curvature...
State 4: Entering sub-process...
State 41: Performing rotation exclusion...
X410 = 0,X411 = 0,X412 = 0,X413 = 0,X414 = 1,X415 = 0,X416 = 0,X417 = 0,X418 = 0
State 414: Calculating ridge density...
```

圖 4.7 指紋比對模組 0 至 410 狀態的高階軟體合成狀態轉移結果

4.4 可重構指紋特徵比對驗證

實驗目的在於驗證指紋特徵比對架構的準確性和可重構性，因此本研究採用了 Minutia 比對與 FLANN (Fast Library for Approximate Nearest Neighbors)比對技術，通過將擷取的特徵點輸入不同的比對演算法中，來驗證系統在不同比對方法下的差異表現。

4.4.1 指紋特徵點擷取

依據 FVC2004 數據庫所產生的特徵點集，這些特徵點集內容包含X座標、Y座標、方向與特徵點種類和曲率，作為比對的基礎數據。實驗開始時，從指紋比對程式介面中如圖 4.8，選取兩組特徵點作為比對輸入資料如圖 4.9 所示，分別定義為測試特徵集(Test template)和樣本特徵集(Sample template)。

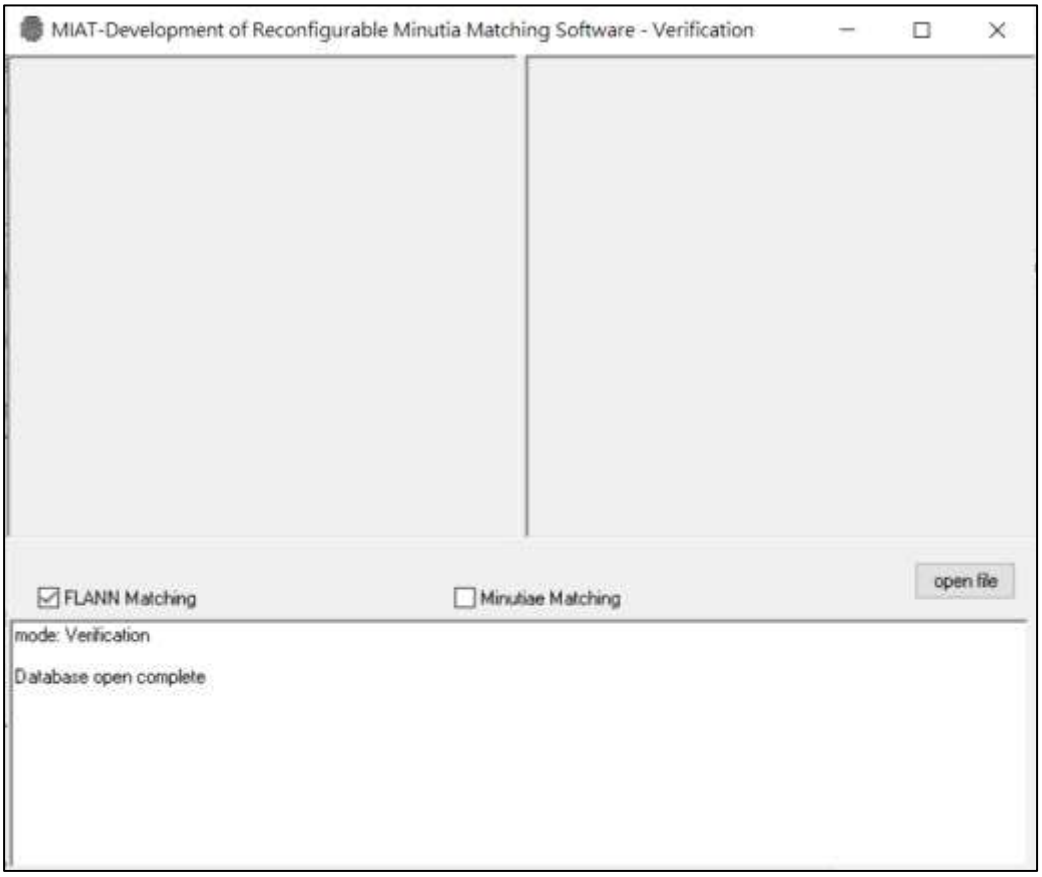


圖 4.8 指紋比對系統操作介面

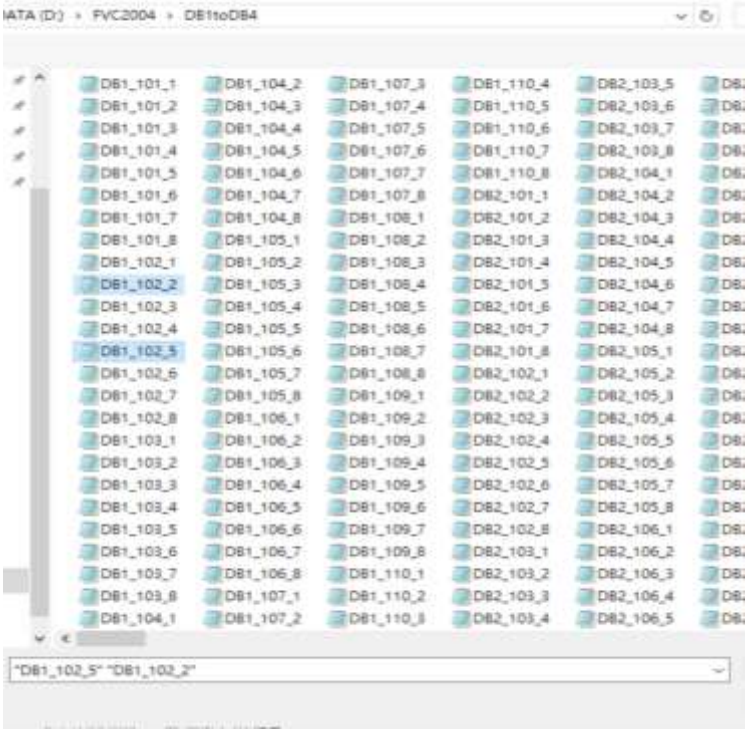


圖 4.9 選取樣本特徵集(DB1_102_2)與測試特徵集(DB1_102_5)

這些資料集輸入至比對系統中進行讀取和比對。當系統成功讀取特徵點後，對應的指紋特徵點會顯示在界面上如圖 4.10，便於用戶觀察和驗證讀取是否完整，並為後續的指紋比對實驗提供參考。

圖 4.10 確認系統成功正確讀取特徵點數據

4.4.2 Minutia 比對實驗

本論文旨在提出了一種可重構指紋特徵點比對架構，以提高指紋比對的準確性和效率。為驗證此框架的可行性，在實驗中使用 Minutia 演算法比對，因為 Minutia 比對為目前最廣泛的[67]。

Minutia 實驗結果如下圖 4.11，在系統中勾選 Minutia Matching 功能後，選擇樣本特徵集 DB1_102_2 與測試特徵集 DB1_102_5 進行比對，系統隨即顯示比對結果，相似度分數為 154。根據 Minutia 演算法的說明，相似度分數介於 30 到 1000 之間，皆表示兩者存在部分特徵點相似，可判斷為同一隻手指。

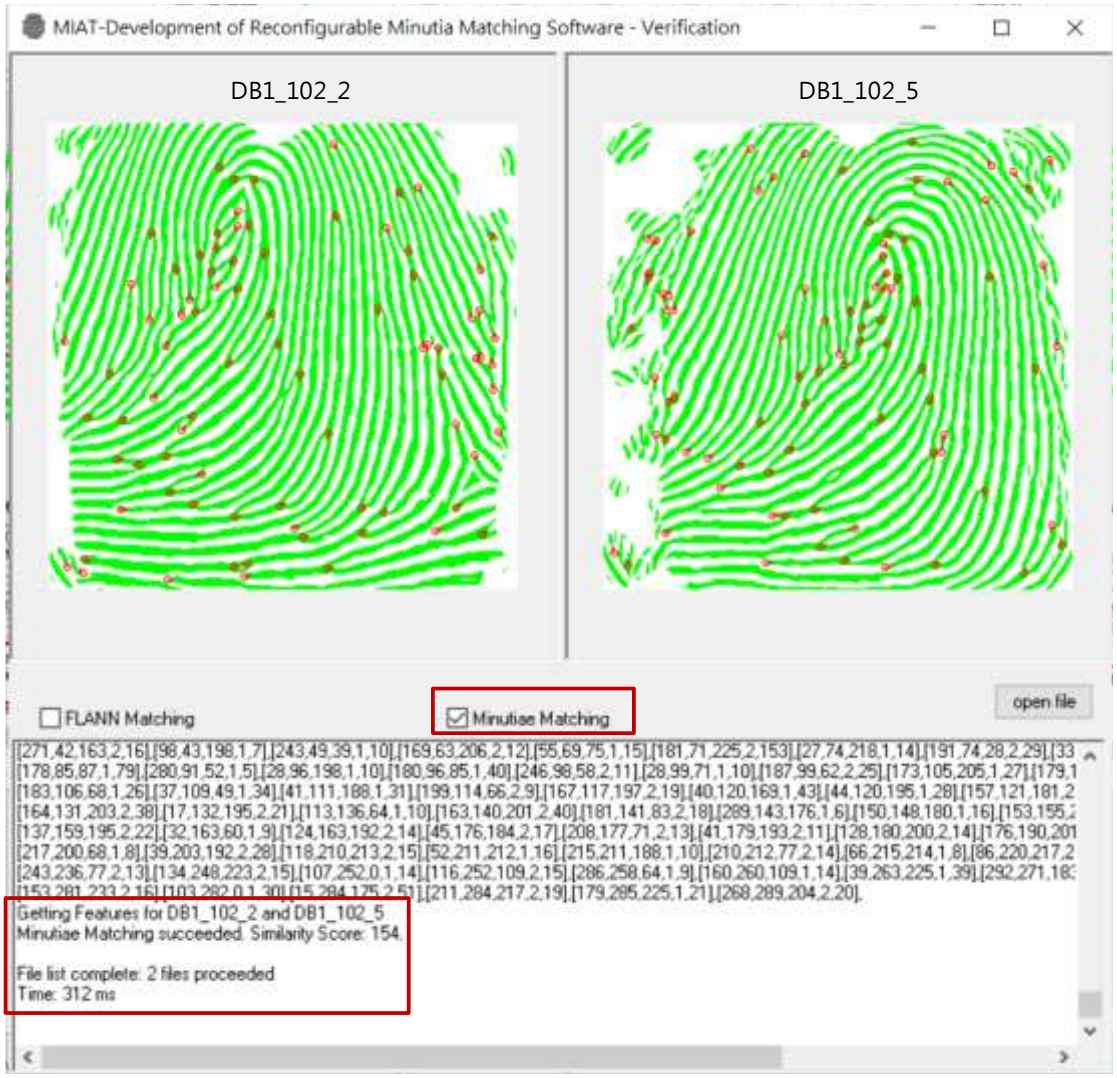


圖 4.11 Minutia 比對實驗成功結果

選擇樣本特徵集 DB1_101_1 與測試特徵集 DB1_101_6 進行比對時，系統顯示相似度分數為 0，表示兩者之間沒有任何相同的特徵點，據此可判定為不同的手指，結果如下圖 4.12。

圖 4.12 Minutia 比對實驗失敗結果

在比對的演算法中，會比對兩個指紋的樣板及回傳兩指紋的相似度(Similarity)。比對過程首先擷取測試樣本與樣本樣板中的特徵線段，並對其進行初步的參數計算。接著，透過計算特徵線段的相似度(Pair's similarity)，判斷兩線

段是否進一步進入門檻篩選步驟。相似度的值是基於線段長度(D)、角度 (α1 和 α2)等參數加權平均後得出，用以評估特徵線段的比對程度，參數如下圖 4.13 說明。

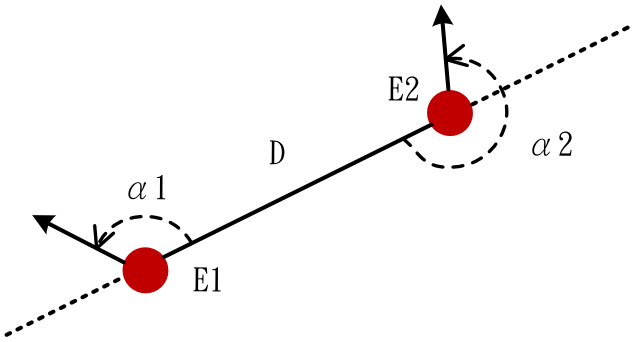


圖 4.13 線段與線段參數

在旋轉與平移處理階段，演算法會根據測試樣板的特徵方向進行適當的旋轉與位移。旋轉處理的目的是找出最佳旋轉角度，使測試樣板與樣本樣板的相似度最大化。特徵線段的方向(α)以及角度(α1 和 α2)是旋轉處理的重要依據；平移處理則以特徵線段的端點位置(E1 和 E2)為核心，排除不合理或相似度過低的樣板。

最終，比對通過旋轉與平移處理的特徵線段後，演算法將統合所有有效線段的相似度值，計算出整體樣板的最終相似度(Similarity)，實驗紀錄 60 組如下表 4.3。

表 4.3 Minutia 比對實驗結果紀錄

Group	Input1	Input2	Minutia Similarity	Minutia Time
1	DB1_102_1	DB1_102_5	169	547ms
2	DB1_101_1	DB1_101_6	0	281ms
3	DB4_108_7	DB4_108_8	0	625ms
4	DB1_101_6	DB1_101_7	113	360ms

5	DB1_101_1	DB1_101_2	0	250ms
6	DB4_103_7	DB4_103_8	61	672ms
…	…	…	…	…
57	DB2_107_7	DB2_107_8	302	766ms
58	DB3_103_5	DB3_103_6	0	1s485ms
59	DB2_108_5	DB2_108_6	0	985ms
60	DB2_108_6	DB2_108_7	0	1s16ms

4.4.3 FLANN 比對實驗

基於 FLANN 演算法特性與優點，本研究將其作為測試可重構指紋比對架構更換之模組。FLANN 指紋特徵點比對系統實驗步驟，首先，演算法的目的是在兩組指紋特徵點之間找到相鄰的對應點，並衡量相似性[68]。特徵點包括位置 (x, y)和方向(Direction)。

在實驗中，準備兩組指紋特徵點集為 DB1_102_2 與 DB1_102_5，並在系統中勾選 FLANN Matching 功能。後端會將資料分別標記為 F_1(DB1_102_2)和 F_2(DB1_102_5)，對於每個在F_1中的特徵點$f1_i$，找到F_2中距離最近的特徵點$f2_j$。為此，需設定 FLANN 比對器的參數，包括算法和搜尋精度。本次實驗選擇 KD 索引樹(FLANN_INDEX_KDTREE)進行最近鄰查找，並設置維度樹的數量參數(Trees=5)以及遞迴索引樹的檢查次數參數(Checks=50)，以平衡查找速度與準確度。

接著執行 FLANN 比對程式，根據設定進行 KNN (K-Nearest Neighbors)比對，將每個特徵點與目標影像中最接近的兩個特徵點進行比對。接下來進行比對結果的篩選，參考 Lowe 實驗[54]的測試閾值設定為 0.7，過濾掉距離過大的比對結果，保留高品質的特徵點對。最終的相似性計算是依據所有最近距離的平均值來進行計算，並得到相似值 0.3604，實驗結果如圖 4.14 所示。

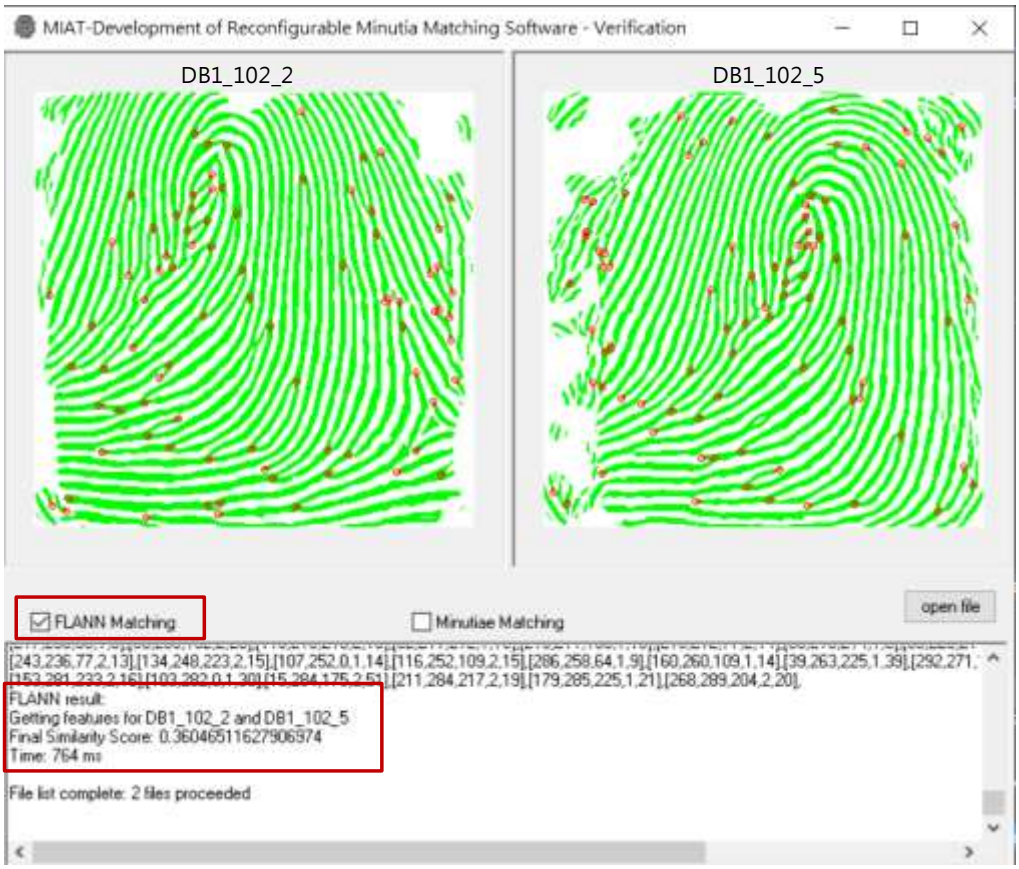


圖 4.14 FLANN 比對實驗成功結果

然而，在選擇第二組實驗特徵點集 DB1_101_1 和 DB1_101_6 時，系統得出的相似值為 0，這表示兩組數據間無鄰近的特徵點，因此無法比對成功，實驗結果如圖 4.15 所示。

圖 4.15 FLANN 比對實驗失敗結果

此外，實驗中共隨機選擇 60 組數據進行比對，詳細實驗紀錄如表 4.4 所示。

表 4.4 FLANN Matching 實驗結果紀錄

Group	Input1	Input2	FLANN Similarity	FLANN Time
1	DB1_102_1	DB1_102_5	0.348837209	995ms
2	DB1_101_1	DB1_101_6	0	770ms
3	DB4_108_7	DB4_108_8	0	1s103ms
4	DB1_101_6	DB1_101_7	0	1s43ms
5	DB1_101_1	DB1_101_2	0	1s128ms
6	DB4_103_7	DB4_103_8	0	1s87ms
	…	…	…	…
57	DB2_107_7	DB2_107_8	0.3888888	1s53ms
58	DB3_103_5	DB3_103_6	0.482758621	1s21ms
59	DB2_108_5	DB2_108_6	0	1s11ms
60	DB2_108_6	DB2_108_7	0.396039604	1s49ms

4.5 結果比較

本實驗採用了 *Minutia Matching* 和 *FLANN Matching* 兩種指紋比對技術，隨機選擇 60 組共 120 個指紋特徵點集進行比對驗證。每組指紋樣本包含不同的特徵點數量和指紋特徵點分布，透過這些測試，從比對準確性、性能表現兩個評估指標來評估整體系統的效果。

比對準確性為本實驗中的核心指標，用於衡量 Minutia Matching 與 FLANN Matching 的結果準確度。在 60 組指紋測試中，兩者的準確率和錯誤率皆被記錄

與分析。本研究圍繞「可重構指紋特徵比對」的主題，旨在驗證設計架構的靈活性與適應性。實驗採用了基於指紋特徵點的 Minutia 比對與 FLANN 比對，兩者在理論基礎與實現方法上存在顯著差異。Minutia 比對依賴細節點(如分叉點與終止點)的位置和方向進行比對，適用於細節完整的高質量指紋圖像。而 FLANN 比對基於特徵點的全局搜索，對圖像模糊或細節缺失具有一定的容忍度。這種方法學差異使其成為檢驗架構可重構性的理想組合。

實驗將數據分為三組：第一組是 Minutia 和 FLANN 同時比對成功或失敗；第二組是 Minutia 比對成功而 FLANN 比對失敗；第三組則是 FLANN 比對成功而 Minutia 比對失敗。實驗結果顯示如下圖 4.16，Minutia 和 FLANN 的比對結果一致性最高(61.7%)，即兩者同時比對成功或均比對失敗。這表明在指紋圖像有相同特徵點條件下，兩種方法能夠共同提供準確的比對結果，架構能同時支持兩種比對模式並保持結果的一致性與穩定性。此外，26.7%的數據中僅 Minutia 比對成功，反映出其對局部特徵點的敏感性強於 FLANN，適合局部細節完整但全局特徵不足的指紋圖像。而僅有 11.7% 的數據顯示 FLANN 比對成功但 Minutia 比對失敗。

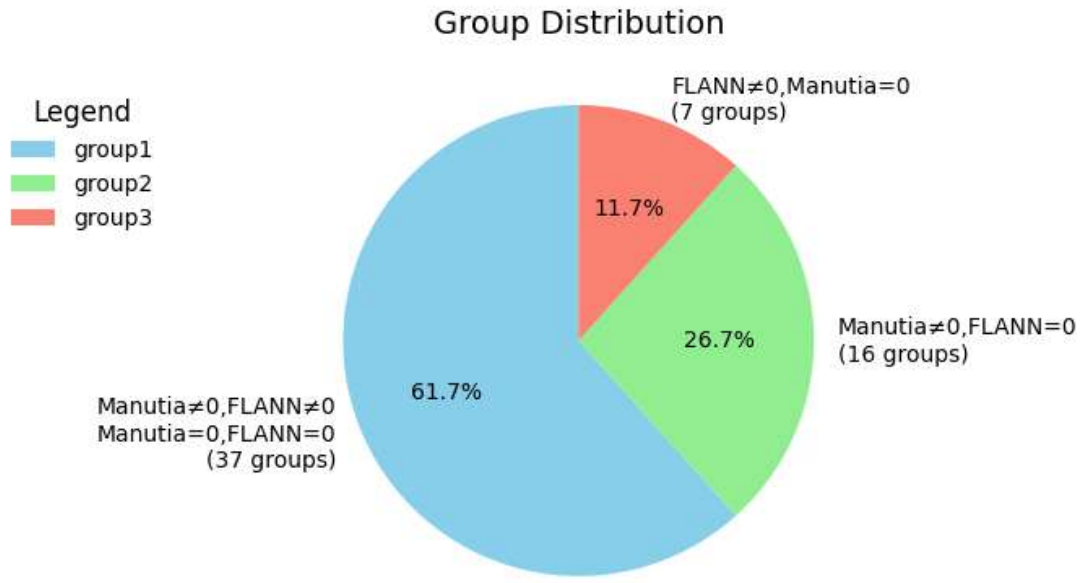


圖 4.16 指紋比對相似度統計圖

針對不同比對模組的性能表現，本實驗比較了每種方法在運行速度差異，在運算時間上，Minutia Matching 的性能優勢更為明顯。然而，FLANN 和 Minutia 兩種比對方法的處理時間比較中進一步分析，根據下圖 4.17 時間數據的統計圖顯示，整體而言，Minutia 比對的執行時間相較於 FLANN 展現了一定的優勢。Minutia 方法在指紋特徵點比對中通常只針對較小的數據集進行比對，而 FLANN 主要基於快速最近鄰搜尋，因此對於高維特徵數據的處理效率會因數據量增加而降低。

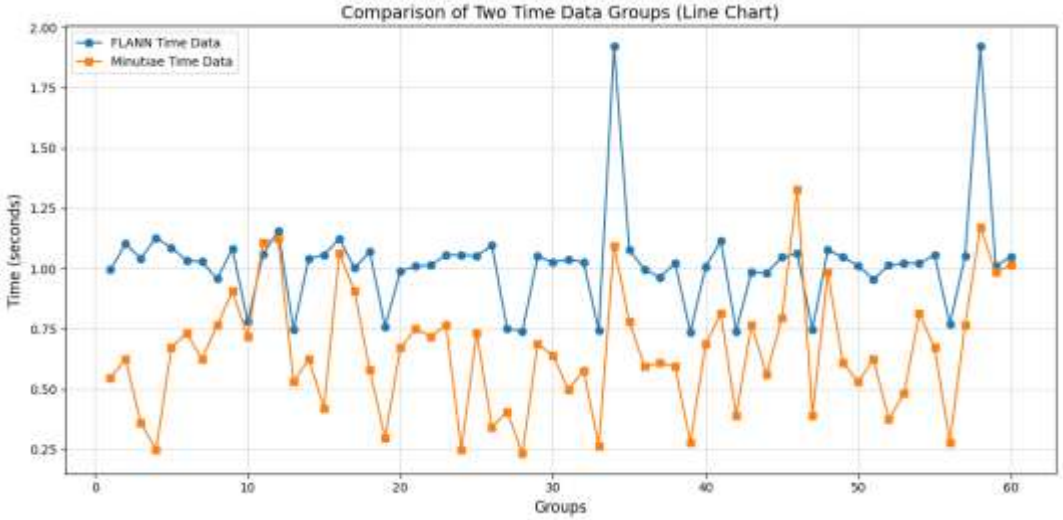


圖 4.17 指紋比對時間統計圖

另外，從數據圖中可觀察到，部分數據組的執行時間相對較長，且在兩組演算法數據中皆有出現。由此可知，對於相同組的特徵點資料，使用不同的比對演算法時，皆需額外的運算時間來確保演算的精度。因此，未來在進行指紋比對研究時，可以針對此類指紋進一步探討和優化相關運算方法。

綜合分析表明，設計的可重構指紋比對架構成功驗證了其在不同比對模式下的適應能力。架構能夠根據數據情境靈活切換比對策略，充分體現了其重構能力，特別是在多模態數據中展現了架構的靈活性。這一結果證明該架構具有良好的穩定性與靈活性，可在多樣化的任務需求中應用。同時，實驗也揭示了未來的優化方向，包括提升比對效率與應用範圍，以進一步檢驗架構的通用性與可擴展

性，靈活性使系統在面對新指紋比對技術或更高效的算法時，能以低成本的方式升級以提高比對效能。

4.6 實驗結果分析

本研究為驗證設計的可行性與可重構性，透過 FLANN 與 Minutiae 兩種比對演算法進行系統切換與驗證，分別評估其執行結果與運算時間。實驗中觀察到系統能在相同特徵點資料下保持運算準確性，並靈活切換演算法，進一步證明了設計的高適應性與可重構性。

實驗結果顯示，透過 MIAT 方法論整合大型語言模型所設計的指紋特徵比對系統，在不同的比對演算法下皆能維持運算效能，並透過 LLM 輔助生成 Grafcet 架構設計，進一步提升了開發效率。此外，系統在可重構設計上的優勢，使其能夠針對未來的運算瓶頸進行靈活優化，進一步奠定了指紋比對研究的技術基礎並拓展其應用潛力。整體實驗結果證明了該設計方法在實現高效性與靈活性上的可行性，並為後續指紋比對技術的優化和擴展提供了重要的參考與驗證依據。

第五章、　結論

5.1　結論

本研究結合 MIAT 方法論與大型語言模型(LLMs)，提出了一種創新的高效系統設計方法，並以指紋特徵比對系統為案例進行驗證。研究從系統的階層模組化設計出發，利用 MIAT 方法論分解並構建了旋轉排除模組、線段方向排除模組及特徵比對模組。隨後，在方法論的高階合成階段，引入 LLM 自動生成 Grafcet action 的程式碼，實現了高效的離散事件驅動系統開發。通過實驗測試和數據分析，研究展示了該方法在系統準確性、效率及維護性上的優勢。

實驗結果表明，基於 MIAT 方法論的模組化設計，顯著提升了系統的穩定性、功能獨立性和維護的便捷性。此外，透過比較 Minutia 與 FLANN 演算法的運行結果，評估系統在處理速度和比對精度上的優劣勢。實驗同時關注模組化設計對系統架構穩定性的影響，通過多組指紋比對測試驗證了其對準確性與速度的支持，滿足了指紋比對的實際應用需求。其中大型語言模型在程式碼生成中的應用，顯著提升了開發效率並降低了人為錯誤風險。實驗證實，LLM 能根據 Grafcet 模型需求生成高準確性的程式碼，大幅縮短開發時間，尤其在處理離散事件驅動程式設計時展現出強大的適應能力。

本研究的主要貢獻在於首次將 MIAT 方法論與 LLM 結合應用於指紋辨識系統設計，並驗證了該方法在提升系統開發效率、準確性和可維護性方面的可行性與有效性。這一創新設計方法不僅為指紋辨識技術的進一步發展提供了新的解決方案，也為其他高複雜度軟體開發的架構設計和實施提供了參考價值。

5.2 未來展望

本研究結合 MIAT 方法論與大型語言模型(LLMs)，提出了一種創新的系統設計方法，並成功驗證了其在指紋特徵比對系統中的應用價值。然而，隨著技術的進一步發展和應用需求的多樣化，未來仍有許多值得探索的方向。

未來研究可針對更廣泛的指紋比對應用場景進行探討，隨著軟硬體協同設計的趨勢，未來研究可以探索 MIAT 方法論與嵌入式硬體平台的深度結合，為物聯網(IoT)設備或邊緣計算系統提供高效的軟硬體集成解決方案。並在此基礎上，結合 LLM 自動生成硬體描述語言程式碼，可能進一步提升嵌入式系統設計的效率和精確性。

參考文獻

[1] A. K. Jain, A. Ross and S. Prabhakar, “An introduction to biometric recognition,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 1, pp. 4-20, Jan. 2004.

[2] K. Han and X. Li, “Application of Partial Differential Equation Method in Fingerprint Image Enhancement,” *2021 IEEE/ACIS 19th International Conference on Software Engineering Research, Management and Applications (SERA)*, Kanazawa, Japan, pp. 33-38, 2021.

[3] E. N. Bifari and L. A. Elrefaei, “Automated Fingerprint Identification System based on weighted feature points matching algorithm,” *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Delhi, India, pp. 2212-2217, 2014.

[4] V. Kakani, V. H. Nguyen, B. P. Kumar, H. Kim, V. R. Pasupuleti, “A critical review on computer vision and artificial intelligence in food industry,” *Journal of Agriculture and Food Research*, vol. 2, no. 100033, 2020.

[5] A. K. Jain, S. S. Arora, K. Cao, L. Best-Rowden and A. Bhatnagar, “Fingerprint Recognition of Young Children,” *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 7, pp. 1501-1514, July 2017.

[6] J. Qi, Z. Shi, X. Zhao and Y. Wang, “A novel fingerprint matching method based on the Hough transform without quantization of the Hough space,” *Third International Conference on Image and Graphics (ICIG'04)*, pp. 262-265, December 2004.

[7] D. Lee, K. Choi and J. Kim, “A robust fingerprint matching algorithm using local alignment,” *2002 International Conference on Pattern Recognition*, vol. 3, pp. 803-806, August 2002.

[8] P. D. Gutiérrez, M. Lastra, F. Herrera and J. M. Benítez, “A High Performance Fingerprint Matching System for Large Databases Based on GPU,” *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 1, pp. 62-71, Jan. 2014.

[9] D. Peralta, S. García, J. M. Benitez, F. Herrera, “Minutiae-based fingerprint matching decomposition: Methodology for big data frameworks,” *Information Sciences*, vol. 408, pp. 198-212, 2017.

[10] M.Y. Lin, S.H. Hsieh, C.H. Chen, C.H. Lin, "High-Performance Chip Design with Parallel Architecture for Magnetic Field Imaging System," *IEEE Transactions on Instrumentation and Measurement*, vol. 73, no. 9502313, pp. 1-13, 2024.

[11] J. Li, G. Li, Y. Li and Z. Jin, “Structured chain-of-thought prompting for code generation,” 2023, *arXiv:2305.06599*.

[12] L. Murr, M. Grainger, D. Gao, “Testing LLMs on Code Generation with Varying Levels of Prompt Specificity,” 2023, *arXiv:2311.07599*.

[13] C. C. Hsieh, C. Y. Liu, P. Y. Wu, A. P. Jeng, R. G. Wang, C. C. Chou, “Building information modeling services reuse for facility management for semiconductor fabrication plants,” *Automation in Construction*, vol. 102, pp. 270-287, 2019.

[14] Z. Xu, F Xi, L Liu, and L. Chen, “A Method for Design of Modular Reconfigurable Machine Tools,” *Machines*, vol. 5, no. 1, 2017.

[15] O. David, J. C. Ascough, W. Lloyd, T. R. Green, K. W. Rojas, G.H. Leavesley, L.R. Ahuja, “A software engineering perspective on environmental modeling framework design: The Object Modeling System,” *Environmental Modelling & Software*, vol. 39, pp. 201-213, 2013.

[16] G. Fylaktopoulos, M. Skolarikis, I. Papadopoulos, G. Goumas, A. Sotiropoulos, I. Maglogiannis, “A distributed modular platform for the development of cloud

based applications," *Future Generation Computer Systems*, vol. 78, no. 1, pp. 127-141, 2018.

[17] J. Wang, L. Cao, X. Luo, Z. Zhou, J. Xie, A. Jatowt, Y. Cai, "Enhancing Large Language Models for Secure Code Generation: A Dataset-driven Study on Vulnerability Mitigation," 2023, *arXiv:2310.16263*.

[18] V. J. Rathod, N. C. Iyer and Meena S M, "A survey on fingerprint biometric recognition system," *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, Greater Noida, India, pp. 323-326, 2015.

[19] J. Bohné and V. Despiégel, "Fingerprint skeleton matching based on local descriptor," *2009 IEEE 3rd International Conference on Biometrics: Theory,Applications, and Systems*, Washington, DC, USA, pp. 1-6, 2009.

[20] C.H. Chen, C.S. An, C.Y. Chen, "Fingerprint Quality Assessment Based on Texture and Geometric Features," *Journal of Imaging Science and Technology, Journal of Imaging Science and Technology*, vol.64, no.4, pp. 40403-1-40403-7(7), Jul. 2020.

[21] C.H. Chen, C.Y. Chen, and T.M. Hsu, "Singular Points Detection in Fingerprints Based on Poincare Index and Local Binary Patterns," *Journal of Imaging Science and Technology*, vol. 63, no.3, pp.030401-1-7, Jun. 2019.

[22] S. Bakheet and A. Al-Hamadi, "Robust hand gesture recognition using multiple shape-oriented visual cues," *J Image Video Proc*, vol. 26, pp. 1-18, 2021.

[23] Kalyani Mali and Samayita Bhattacharya, "Fingerprint Recognition Using Global and Local Structures," *International Journal on Computer Science and Engineering*, vol. 3, no. 1, pp. 161-172, Jan. 2011.

[24] D. Petrovska-Delacretaz, G. Chollet, and B. Dorizzi. Guide to Biometric Reference Systems and Performance Evaluation. Springer, Dordrecht, 2009. 3.

[25] S. Bakheet and A. Al-Hamadi, “A framework for instantaneous driver drowsiness detection based on improved HOG features and naïve Bayesian classification,” *Brain Sci.*, vol. 11, no. 2, pp. 240, Feb. 2021.

[26] V. Gudkov and D. Lepikhova, “Fingerprint Model Based on Fingerprint Image Topology and Ridge Count Values,” *2018 Global Smart Industry Conference (GloSIC)*, Chelyabinsk, Russia, pp. 1-5, 2018.

[27] C. C. Liao and C. T. Chiu, “Fingerprint recognition with ridge features and minutiae on distortion,” *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Shanghai, China, pp. 2109-2113, 2016.

[28] H. Choi, H. K. Choi and J. Kim, “Fingerprint Matching Incorporating Ridge Features With Minutiae,” *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 2, pp. 338-345, 2011.

[29] C.H. Chen, C.T. Liu, “Person Re-Identification Microservice over Artificial Intelligence Internet of Things Edge Computing Gateway,” *Electronics*, vol.10, no.18, pp. 2264, 2021.

[30] A. K. Jain, J. Feng and K. Nandakumar, “Fingerprint Matching,” *Computer*, vol. 43, no. 2, pp. 36-44, Feb. 2010.

[31] D. Peralta et al., “A survey on fingerprint minutiae-based local matching for verification and identification: Taxonomy and experimental evaluation,” *Information Sciences*, vol. 315, pp. 67-87, Sep. 2015.

[32] K. N. Win, K. Li, J. Chen, P. Fournier-Viger and K. Li, ”Fingerprint classification and identification algorithms for criminal investigation: A survey, “ *Future Generation Computer Systems*, vol. 110, pp. 758-771, 2020.

[33] H. Wu, Q. Liu and X. Liu, “A review on deep learning approaches to image classification and object segmentation,” *Comput. Mater. Continua*, vol. 1, no. 2, pp. 1-5, 2018.

[34] B. Liu, Z. Li, et al. “Fingerprint reconstruction and recognition based on the three-dimensional structure of fingertip skin,” *5th Optics Young Scientist Summit (OYSS 2022)*, vol. 12448, 2022.

[35] A. Mishra, S. Dehuri, “A Novel Hybrid FLANN-PSO Technique for Real Time Fingerprint Classification,” *Medico-Legal Update*, vol. 19, no. 2, pp. 740-746, 2019.

[36] Y. Zheng and P. Zheng, “Image Matching Based on Harris-Affine Detectors and Translation Parameter Estimation by Phase Correlation,” *2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP)*, Wuxi, China, pp. 106-111, 2019.

[37] S. Insankeovilay, P. Prasarn and S. Choomchuay, “Fingerprint matching with cross correlation and minutiae scores,” *The 4th 2011 Biomedical Engineering International Conference*, Chiang Mai, Thailand, pp. 174-177, 2012.

[38] D. Maltoni, D. Maio, A. Jain and S. Prabhakar, Handbook of Fingerprint Recognition, USA, NY, New York:Springer-Verlag, 2009.

[39] K. Ito, A. Morita, T. Aoki, T. Higuchi, H. Nakajima and K. Kobayashi, “A fingerprint recognition algorithm using phase-based image matching for low-quality fingerprints,” *IEEE International Conference on Image Processing 2005*, Genova, Italy, pp. 11-33, 2005.

[40] J. Feng, Z. Ouyang, A. Cai, “Fingerprint matching using ridges,” *Pattern Recognition*, vol. 39, no. 11, pp. 2131-2140, 2006.

[41] L. Sha, F. Zhao, X. Tang, "Minutiae-based Fingerprint Matching Using Subset Combination," *ICPR*, vol. 4 pp. 566-569, 2006.

[42] ANSI-INCITS 378-2004, "Information Technology - Finger Minutiae Format for Data Interchange," 2004.

[43] ANSI/NIST-ITL 1-2007, "Data Format for the Interchange of Fingerprint, Facial, Other Biometric Information," 2007.

[44] J. Bohné and V. Despiégel, "Fingerprint skeleton matching based on local descriptor," *2009 IEEE 3rd International Conference on Biometrics: Theory,Applications*, and Systems, Washington, DC, USA, pp. 1-6, 2009.

[45] K. Nirmalakumari, H. Rajaguru and P. Rajkumar, "Efficient Minutiae Matching Algorithm for Fingerprint Recognition," *2019 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, Sathyamangalam, India, pp. 1-5, 2019.

[46] K. Singh, K. Kaur and A. Sardana, "Fingerprint feature extraction," *Int. J. Comput. Sci. Technol.*, vol. 2, pp. 237-241, 2011.

[47] J. Yin, S. Pan, X. Li, S. Yu, Y. Xue and J. Zhou, "Research on Fingerprint Recognition Algorithm Based on Minutiae Matching Pairs," *2023 3rd International Conference on Electronic Information Engineering and Computer (EIECT)*, Shenzhen, China, pp. 484-493, 2023.

[48] I. Kovač, P. Marák, "Openfinger: Towards a combination of discriminative power of fingerprints and finger vein patterns in multimodal biometric system, " *Tatra Mountains Mathematical Publications*, vol. 77, no. 1, pp. 109-138, 2020.

[49] V. Vijayan and P. Kp, "FLANN Based Matching with SIFT Descriptors for Drowsy Features Extraction," *2019 Fifth International Conference on Image Information Processing (ICIIP)*, Shimla, India, 2019, pp. 600-605.

[50] Y.R. Wang, H.X. Li, and L.K. Wang, “Similar image retrieval using color histogram in HSV space and SIFT descriptor with FLANN,” *Foundations of Intelligent Systems: Proceedings of the Eighth International Conference on Intelligent Systems and Knowledge Engineering*, vol. 277, pp.1085-1093, 2014.

[51] R. W. Gowandy, A. Halim, & H. Santoso, ”ELECTRONIC VOTING SYSTEM BASED ON FINGERPRINT RECOGNITION AND RADIO FREQUENCY IDENTIFICATION,” *International Journal of Engineering Development And Research*, vol. 2, no. 4, pp. 3850-3854, 2017.

[52] M. Muja and D. G. Lowe, “Scalable Nearest Neighbor Algorithms for High Dimensional Data,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2227-2240, Nov. 1, 2014.

[53] M. Muja and D. G. Lowe, “FLANN—Fast library for approximate nearest neighbor user manual,” *Proc. Int. Conf. Comput. Vis. Theory Appl. (VISAPP)*, pp. 1-29, Jan. 2013.

[54] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International journal of computer vision*, vol. 60, no. 2, pp. 91-110, 2004.

[55] A. Mishra, N. Singh, S. R. Samal and S. Dash, “Biogeography Based Optimized Hybrid Chebyshev FLANN for Fingerprint Classification,” *2023 1st International Conference on Circuits, Power and Intelligent Systems (CCPIS)*, Bhubaneswar, India, pp. 1-4, 2023.

[56] Y. T. Fang, and H. Rau. “Optimal consumer electronics product take-back time with consideration of consumer value,” *Sustainability*, vol. 9, no. 3, pp. 385, 2017.

[57] C.H. Chen, M.Y. Lin, X.C. Guo, “High-level Modeling and Synthesis of Smart Sensor Networks for Industrial Internet of Things,” *Computers & Electrical Engineering*, vol.61, pp.48–66, Jul. 2017.

[58] M.H. Temsah, A. Jamal, K. Alhasan, A.A. Temsah, K.H. Malki, “OpenAI o1-Preview vs. ChatGPT in Healthcare: A New Frontier in Medical AI Reasoning,” *Cureus*, vol. 16, no. 10, pp.70640, 2024.

[59] A. Jaech, A. Kalai, et al., “OpenAI o1 System Card,” 2024, *arXiv:2412.16720*.

[60] Z. Liu, Y. Tang, X. Luo, Y. Zhou and L. F. Zhang, “No Need to Lift a Finger Anymore? Assessing the Quality of Code Generation by ChatGPT,” in IEEE Transactions on Software Engineering, vol. 50, no. 6, pp. 1548-1584, Jun. 2024.

[61] D. Huang, Q. Bu, J. Zhang, X. Xie, J. Chen, H. Cui, “Bias assessment and mitigation in LLM-based code generation,” 2023, *arXiv:2309.14345*.

[62] Y.H. Liao, “Fingerprint Minutiae Detection and Extraction Algorithm with Software High-Level Modeling and Synthesis,” M.S. thesis, Dept. of Computer Science and Information Engineering, National Central Univ., Taoyuan, Taiwan, 2024. [Online]. Available: https://ir.lib.ncu.edu.tw:444/thesis/view_etd.asp?URN=110552019&fileName=GC110552019.pdf

[63] R. Cappelli, D. Maio, D. Maltoni, J. L. Wayman and A. K. Jain, “Performance evaluation of fingerprint verification systems,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 1, pp. 3-18, Jan. 2006.

[64] L. Murr, M. Grainger, D. Gao, “Testing LLMs on Code Generation with Varying Levels of Prompt Specificity,” 2023, *arXiv: 2311.07599*.

[65] A. Buscemi, “A comparative study of code generation using chatgpt 3.5 across 10 programming languages,” 2023, *arXiv:2308.04477*.

[66] F. A. Sakib, S. H. Khan and A. H. M. R. Karim, “Extending the Frontier of ChatGPT: Code Generation and Debugging,” *2024 International Conference on*

Electrical, Computer and Energy Technologies (ICECET, Sydney, Australia, pp. 1-6, 2024.

[67] R. Cappelli, M. Ferrara and D. Maltoni, "Minutiae-Based Fingerprint Matching," *Cross Disciplinary Biometric Systems*, vol. 37, pp. 117-150, 2012.

[68] B.Y. Satria, A. Bejo and R. Hidayat, "Fingerprint Enhancement using Iterative Contextual Filtering for Fingerprint Matching," *2021 9th International Conference on Information and Communication Technology (ICoICT)*, pp. 186-191, 2021.

附錄一 生成 Grafcet 對話流程

首先將原先中文擬定的 Grafcet 翻譯成英文，如附圖 1。

附圖 1 翻譯中文 Grafcet

接著，將繪製英文的 Grafcet 圖片給 ChatGPT 請他敘述後，確認邏輯是否正確如附圖 2。

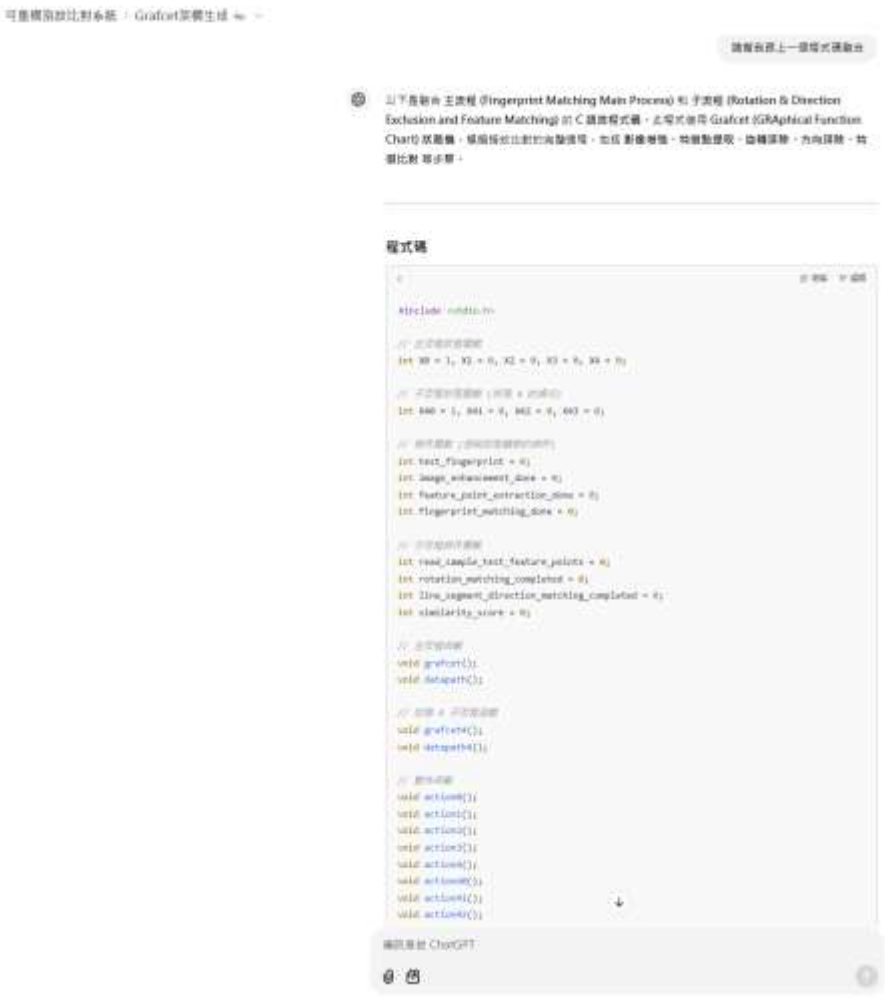
附圖 2 架構邏輯確認

接著，確認邏輯無誤後，依照此生成對應程式碼如附圖 3。

附圖 3 生成對應的程式碼

接著，若有子系統並重複上述流程，確認邏輯如附圖 4 後，再將他與主架構程式結合如附圖 5。

附圖 4 子系統邏輯敘述確認



附圖 5 結合子系統與主架構程式

附錄二 可重構指紋比對 Grafcet Action

```
#include <stdio.h>

// 主流程狀態變數
int X0 = 1, X1 = 0, X2 = 0, X3 = 0, X4 = 0;

// 狀態 4 的子流程變數
int X40 = 1, X41 = 0, X42 = 0, X43 = 0;

// 狀態 41 的子流程變數
int X410 = 1, X411 = 0, X412 = 0, X413 = 0, X414 = 0, X415 = 0, X416 = 0, X417 = 0, X418 =
0;

// 狀態 42 的子流程變數
int X420 = 1, X421 = 0, X422 = 0, X423 = 0, X424 = 0, X425 = 0;

// 狀態 430 的子流程變數
int X430 = 1, X431 = 0, X432 = 0, X433 = 0, X434 = 0;

// 主流程條件變數
int initial0 = 0;
int image_enhancement_1 = 0;
int feature_extraction_2_3 = 0;
int feature_point_matching_4 = 0;

// 狀態 4 子流程條件變數
int read_sample_test_feature_points_40 = 0;
int rotation_matching_completed_41 = 0;
int segment_direction_matching_completed_42 = 0;
int similarity_score_43 = 0;

// 狀態 41 子流程條件變數
int get_data_410 = 0;
int line_similarity_calculation_completed_411 = 0;
int bifurcation_point_calculation_completed_412 = 0;
int curvature_calculation_completed_413 = 0;
int ridge_density_calculation_414 = 0;
```

```
int rotation_points_remaining_for_calculation_415 = 0;
int rotation_point_calculation_completed__415 = 0;
int next_rotation_point_416 = 0;
int rotation_angle_determination_completed_417 = 0;
int rotation_point_exclusion_completed_418 = 0;

int user_input_415=0;//輸入假設是否判斷完成
int user_input_422=0;
int user_input_432=0;


// 狀態 42 子流程條件變數
int rotation_matching_completed_420 = 0;
int difference_computation_completed_421 = 0;
int histogram_computation_completed_422 = 0;
int segments_remaining_for_computation_422 = 0;
int next_segment_423 = 0;
int segment_determination_completed_424 = 0;
int segment_exclusion_completed_425 = 0;

// 狀態 430 子流程條件變數
int segment_direction_matching_completed_430 = 0;
int line_end_comparison_completed_431 = 0;
int error_matching_completed_432 = 0;
int line_ends_remaining_for_computation_432 = 0;
int next_line_end_433 = 0;
int similarity_calculation_completed_434 = 0;

void grafcet();         // 主流程狀態機
void grafcet4();        // 狀態 4 子流程狀態機
void grafcet41();       // 狀態 41 子流程狀態機
void grafcet42();       // 狀態 42 子流程狀態機
void grafcet43();
void datapath();        // 主流程動作
void datapath4();       // 狀態 4 子流程動作
void datapath41();      // 狀態 41 子流程動作
void datapath42();      // 狀態 42 子流程動作
void datapath43();
```

```
// 動作函數
void action0();
void Image_Enhancement();
void Core_Point_Processing();
void Minutia_Extraction();
void Minutia_Matching();
void action40();
void Line_Rotation_Exclusion();
void Line_Orientation_Exclusion();
void Minutia_Matching_Similarity();
void action410();
void Line_Similarity_Calculation();
void Minutia_Type_Calculation();
void Minutia_Curvature_Calculation();
void Minutia_Average_Ridge_Density_Calculation();
void Build_Rotation_Histogram();
void Compare_Next_Rotation_Point();
void Rotation_Angle_Determination();
void Rotation_Point_Exclusion();
void action420();
void Line_Orientation_Difference_Calculation();
void Build_Translation_Histogram();
void Compare_Next_Translation_Point();
void Line_Determination();
void Line_Exclusion();
void action430();
void Line_Endpoint_Difference_Test();
void False_Minutia_Matching_Exclusion();
void Compare_Next_Endpoint();
void Similarity_Calculation();

int main()
{
    printf("Grafcet simulation starting...\n");
    while (1)
    {
        datapath(); // 執行當前狀態動作
```

```
            grafcet();  // 主流程狀態轉換
        }
        return 0;
}

// 主流程動作
void datapath()
{
    if (X0 == 1)
        action0();
    if (X1 == 1)
        Image_Enhancement();
    if (X2 == 1)
        Core_Point_Processing();
    if (X3 == 1)
        Minutia_Extraction();
    if (X4 == 1)
        Minutia_Matching();
}

// 狀態 4 子流程動作
void datapath4()
{
    if (X40 == 1)
        action40();
    if (X41 == 1)
        Line_Rotation_Exclusion();
    if (X42 == 1)
        Line_Orientation_Exclusion();
    if (X43 == 1)
        Minutia_Matching_Similarity();
}

// 狀態 41 子流程動作
void datapath41()
{
    if (X410 == 1)
        action410();
```

```
    if (X411 == 1)
        Line_Similarity_Calculation();
    if (X412 == 1)
        Minutia_Type_Calculation();
    if (X413 == 1)
        Minutia_Curvature_Calculation();
    if (X414 == 1)
        Minutia_Average_Ridge_Density_Calculation();
    if (X415 == 1)
        Build_Rotation_Histogram();
    if (X416 == 1)
        Compare_Next_Rotation_Point();
    if (X417 == 1)
        Rotation_Angle_Determination();
    if (X418 == 1)
        Rotation_Point_Exclusion();
}
// 狀態 42 子流程動作
void datapath42()
{
    if (X420 == 1)
        action420();
    if (X421 == 1)
        Line_Orientation_Difference_Calculation();
    if (X422 == 1)
        Build_Translation_Histogram();
    if (X423 == 1)
        Compare_Next_Translation_Point();
    if (X424 == 1)
        Line_Determination();
    if (X425 == 1)
        Line_Exclusion();
}

void datapath43()
{
    if (X430 == 1)
        action430();
```

```
    if (X431 == 1)
        Line_Endpoint_Difference_Test();
    if (X432 == 1)
        False_Minutia_Matching_Exclusion();
    if (X433 == 1)
        Compare_Next_Endpoint();
    if (X434 == 1)
        Similarity_Calculation();
}
// 主流程狀態機
void grafcet()
{
    if (X0 == 1 && initial0 == 1)
    {
        X0 = 0;
        X1 = 1;
        initial0 = 0;
        return;
    }

    if (X1 == 1 && image_enhancement_1 == 1)
    {
        X1 = 0;
        X2 = 1;
        image_enhancement_1 = 0;
        return;
    }

    if (X2 == 1 && feature_extraction_2_3 == 1)
    {
        X2 = 0;
        X3 = 1;
        feature_extraction_2_3 = 0;
        return;
    }

    if (X3 == 1 && feature_point_matching_4 == 1)
    {
```

```
            X3 = 0;
            X4 = 1;
            feature_point_matching_4 = 0;
            return;
        }

        if (X4 == 1)
        {
            datapath4();
            grafcet4();
            if (X40 == 1 && X41 == 0 && X42 == 0 && X43 == 0)
            {
                X4 = 0;
                X0 = 1;
            }
            return;
        }
}

// 狀態 4 子流程狀態機
void grafcet4()
{
    if (X40 == 1 && read_sample_test_feature_points_40 == 1)
    {
        X40 = 0;
        X41 = 1;
        read_sample_test_feature_points_40 = 0;
        return;
    }

    if (X41 == 1)
    {
        datapath41();
        grafcet41();
        return;
    }
```

```
    if (X42 == 1 )
    {

        if (rotation_matching_completed_420 == 0)
        {
            rotation_matching_completed_420 = 1; // 初始化條件
        }
        datapath42();  // 呼叫狀態 42 的動作
        grafcet42();   // 處理 42 的子流程狀態

        return;
    }

    if (X43 == 1)
    {
        datapath43();  // 執行 430 子流程動作
        grafcet43();   // 更新 430 子流程狀態
        return;
    }
}

// 狀態 41 子流程狀態機
void grafcet41()
{
    if (X410 == 1 && get_data_410 == 1)
    {
        X410 = 0;
        X411 = 1;
        get_data_410 = 0;
        return;
    }

    if (X411 == 1 && line_similarity_calculation_completed_411 == 1)
    {
        X411 = 0;
        X412 = 1;
        line_similarity_calculation_completed_411 = 0;
        return;
```

```
}

if (X412 == 1 && bifurcation_point_calculation_completed_412 == 1)
{
    X412 = 0;
    X413 = 1;
    bifurcation_point_calculation_completed_412 = 0;
    return;
}

if (X413 == 1 && curvature_calculation_completed_413 == 1)
{
    X413 = 0;
    X414 = 1;
    curvature_calculation_completed_413 = 0;
    return;
}

if (X414 == 1 && ridge_density_calculation_414 == 1)
{
    X414 = 0;
    X415 = 1;
    ridge_density_calculation_414 = 0;
    return;
}

if (X415 == 1)
{
    if (rotation_points_remaining_for_calculation_415 == 1)
    {
        X415 = 0;
        X416 = 1;
        rotation_points_remaining_for_calculation_415=0;
    }
    else if (rotation_point_calculation_completed__415 == 1)
    {
        X415 = 0;
        X417 = 1;
```

```
            rotation_point_calculation_completed__415=0;
        }
        return;
    }

    if (X416 == 1 && next_rotation_point_416 == 1)
    {
        X416 = 0;
        X410 = 1;
        next_rotation_point_416 = 0;
        return;
    }

    if (X417 == 1 && rotation_angle_determination_completed_417 == 1)
    {
        X417 = 0;
        X418 = 1;
        rotation_angle_determination_completed_417 = 0;
        return;
    }

    if (X418 == 1 && rotation_point_exclusion_completed_418 == 1)
    {
        X418 = 0;
        X410 = 1;
        rotation_point_exclusion_completed_418 = 0;
        X41 = 0;
        X42 = 1;
        return;
    }
}

// 狀態 42 子流程狀態機邏輯
void grafcet42()
{
    if (X420 == 1 && rotation_matching_completed_420 == 1)
    {
        X420 = 0;
```

```
    X421 = 1;
    rotation_matching_completed_420 = 0; // 重置條件
    return;
}

if (X421 == 1 && difference_computation_completed_421 == 1)
{
    X421 = 0;
    X422 = 1;
    difference_computation_completed_421 = 0; // 重置條件
    return;
}

if (X422 == 1)
{
    if (segments_remaining_for_computation_422 == 1)
    {
        X422 = 0;
        X423 = 1;
        segments_remaining_for_computation_422=0;
    }
    else if (histogram_computation_completed_422 == 1)
    {
        X422 = 0;
        X424 = 1;
        histogram_computation_completed_422=0;
    }
    return;
}

if (X423 == 1 && next_segment_423 == 1)
{
    X423 = 0;
    X420 = 1; // 回到 420 處理下一段
    next_segment_423 = 0; // 重置條件
    return;
}
```

```
    if (X424 == 1 && segment_determination_completed_424 == 1)
    {
        X424 = 0;
        X425 = 1;
        segment_determination_completed_424 = 0; // 重置條件
        return;
    }

    if (X425 == 1 && segment_exclusion_completed_425 == 1)
    {
        X425 = 0;
        X420 = 1; // 回到 420 開始新的循環
        segment_exclusion_completed_425 = 0; // 重置條件
        X42 = 0;
        X43 = 1;
        return;
    }
}

void grafcet43()
{
    if (X430 == 1 && segment_direction_matching_completed_430 == 1)
    {
        X430 = 0;
        X431 = 1;
        segment_direction_matching_completed_430 = 0; // 重置條件
        return;
    }

    if (X431 == 1 && line_end_comparison_completed_431 == 1)
    {
        X431 = 0;
        X432 = 1;
        line_end_comparison_completed_431 = 0; // 重置條件
        return;
    }

    if (X432 == 1)
```

```
    {
        if (line_ends_remaining_for_computation_432 == 1)
        {
            X432 = 0;
            X433 = 1;
            line_ends_remaining_for_computation_432=0;
        }
        else if (error_matching_completed_432 == 1)
        {
            X432 = 0;
            X434 = 1;
            error_matching_completed_432=0;
        }
        return;
    }

    if (X433 == 1 && next_line_end_433 == 1)
    {
        X433 = 0;
        X430 = 1; // 回到 430 處理下一段
        next_line_end_433 = 0; // 重置條件
        return;
    }

    if (X434 == 1 && similarity_calculation_completed_434 == 1)
    {
        X434 = 0;
        X430 = 1; // 子流程完成後返回 430
        X40 = 1;
        X41 = 0;
        X42 = 0;
        X43 = 0;
        similarity_calculation_completed_434 = 0; // 重置條件
        return;
    }
}
```

```
// 動作函數
void action0() {
    printf("State 0: Initializing...\n");
    initial0 = 1;
}
void Image_Enhancement() {
    printf("State 1: Performing image enhancement...\n");
    image_enhancement_1 = 1;
    }
void Core_Point_Processing() {
    printf("State 2: Core point processing...\n");
    feature_extraction_2_3 = 1;
    }
void Minutia_Extraction() {
    printf("State 3: Feature point extraction...\n");
    feature_point_matching_4 = 1;
    }
void Minutia_Matching() {
    printf("State 4: Entering sub-process...\n"); }
void action40() {
    printf("State 40: Reading sample/test feature points...\n");
    read_sample_test_feature_points_40 = 1;
    }
void Line_Rotation_Exclusion() {
    printf("State 41: Performing rotation exclusion...\n");
    rotation_matching_completed_41 = 1;
    printf("X410 = %d,X411 = %d,X412 = %d,X413 = %d,X414 = %d,X415 = %d,X416 =
%d,X417 = %d,X418 = %d \n",
    X410 ,X411 ,X412 ,X413 ,X414 ,X415 ,X416 ,X417 ,X418);
}
void Line_Orientation_Exclusion() {
    printf("State 42: Performing segment direction exclusion...\n");
    segment_direction_matching_completed_42 = 1;
    printf("X420 = %d,X421 = %d,X422 = %d,X423 = %d,X424 = %d,X425 =
%d\n",X420 ,X421 ,X422 ,X423 ,X424 ,X425 );
}
void Minutia_Matching_Similarity() {
    printf("State 43: Performing feature matching...\n");
```

```
    similarity_score_43 = 1;
    printf("X430 = %d,X431 = %d,X432 = %d,X433 = %d,X434 =
%d\n",X430 ,X431 ,X432 ,X433 ,X434 );
}
void action410() {
    printf("State 410: Getting data...\n");
    get_data_410 = 1; }
void Line_Similarity_Calculation() {
    printf("State 411: Calculating line similarity...\n");
    line_similarity_calculation_completed_411 = 1; }
void Minutia_Type_Calculation() {
    printf("State 412: Calculating bifurcation point...\n");
    bifurcation_point_calculation_completed_412 = 1; }
void Minutia_Curvature_Calculation() {
    printf("State 413: Calculating curvature...\n");
    curvature_calculation_completed_413 = 1; }
void Minutia_Average_Ridge_Density_Calculation() {
    printf("State 414: Calculating ridge density...\n");
    ridge_density_calculation_414 = 1; }

void Build_Rotation_Histogram() {
    printf("State 415: Constructing rotational histograms...\n");

    printf("Enter 1 for remaining points or 2 for completed: ");
    scanf("%d", &user_input_415);
    if (user_input_415 == 1)
    {
        rotation_points_remaining_for_calculation_415 = 1;
    }
    else if (user_input_415 == 2)
    {
        rotation_point_calculation_completed__415 = 1;
    }
}
void Compare_Next_Rotation_Point() {
    printf("State 416: Comparing next rotational point...\n");
    next_rotation_point_416 = 1;
}
```

```
void Rotation_Angle_Determination() {
    printf("State 417: Determining rotation angle...\n");
    rotation_angle_determination_completed_417 = 1;
}
void Rotation_Point_Exclusion() {
    printf("State 418: Excluding rotation point...\n\n");
    rotation_point_exclusion_completed_418 = 1;
    rotation_matching_completed_41=0 ;
}
// 動作函數
void action420() {
    printf("State 420: Performing rotation matching...\n");
    rotation_matching_completed_420 = 1;
}
void Line_Orientation_Difference_Calculation() {
    printf("State 421: Computing segment direction differences...\n");
    difference_computation_completed_421 = 1;
}
void Build_Translation_Histogram() {
    printf("State 422: Constructing transformation histogram...\n");
    // int user_input;
    printf("Enter 1 for remaining segments, 2 for completed histogram: ");
    scanf("%d", &user_input_422);
    if (user_input_422 == 1)
        segments_remaining_for_computation_422 = 1;
    else if (user_input_422 == 2)
        histogram_computation_completed_422 = 1;
}
void Compare_Next_Translation_Point() {
    printf("State 423: Comparing next segment...\n");
    next_segment_423 = 1;
}
void Line_Determination() {
    printf("State 424: Determining segment...\n");
    segment_determination_completed_424 = 1;
}

void Line_Exclusion() {
```

```
    printf("State 425: Excluding segment...\n\n");
    segment_exclusion_completed_425 = 1;
    segment_direction_matching_completed_42 = 0;
}

void action430() {
    printf("State 430: Matching segment directions...\n");
    segment_direction_matching_completed_430 = 1;
}

void Line_Endpoint_Difference_Test() {
    printf("State 431: Testing line end differences...\n");
    line_end_comparison_completed_431 = 1;
}

void False_Minutia_Matching_Exclusion() {
    printf("State 432: Excluding defective feature points...\n");
    printf("Enter 1 for remaining line ends, 2 for error matching complete: ");
    scanf("%d", &user_input_432);

    if (user_input_432 == 1)
        line_ends_remaining_for_computation_432 = 1;
    else if (user_input_432 == 2)
        error_matching_completed_432 = 1;
}

void Compare_Next_Endpoint() {
    printf("State 433: Comparing next line end...\n");
    next_line_end_433 = 1;
}

void Similarity_Calculation() {
    printf("State 434: Calculating similarity...\n\n");
    similarity_calculation_completed_434 = 1;
    similarity_score_43 = 0;
}
```