靜宜大學
資訊工程學系

畢 業 專 題 成 果 報 告 書

遠距魚缸

學生：

資工四A	411154604	范振恆
資工四A	411154484	王睿璟
資工四A	411180671	高鼎恩
資工四B	411154620	黎志軒
西文四A	411012886	翁欣惠

指導教授：滕元翔　教授

西 元 二0二五 年 十二 月

書背格式

靜宜大學

資訊工程學系

遠距魚缸

西元二〇二五年十二月

靜宜大學資訊工程學系
專題實作授權同意書

本人具有著作財產權之論文全文資料，授予靜宜大學資工系，為學術研究之目的以各種方法重製，或為上述目的再授權他人以各種方法重製，不限地域與時間，惟每人以一份為限。授權內容均無須訂立讓與及授權契約書。依本授權之發行權為非專屬性發行權利。依本授權所為之收錄、重製、發行及學術研發利用均為無償。

指導教授：滕元翔 教授

學生簽名:	學號:	日期:西元	年	月	日
學生簽名:	學號:	日期:西元	年	月	日
學生簽名:	學號:	日期:西元	年	月	日
學生簽名:	學號:	日期:西元	年	月	日
學生簽名:	學號:	日期:西元	年	月	日

指導教師簽章 _____

西 元 二0二五 年 十二 月 五 日

靜宜大學資訊工程學系
專題實作指導教師確認書

茲確認專題書面報告之格式及內容符合本系之規範

畢業專題實作名稱：遠距魚缸

畢業專題實作分組名單：　共計 _5_ 人

組員姓名	學號
范振恆	411154604
王睿璟	411154484
高鼎恩	411180671
黎志軒	411154620
翁欣惠	411012886

指導教師簽章 _____

西 元　二０二五 年 十二 月 五 日

目錄

誌謝

本專題的完成，離不開指導老師滕元翔教授的細心指導與寶貴建議，從專題構思到系統設計，都提供了重要的方向與協助。感謝學校提供良好的設備與完整的資源，使我們得以專注進行專題研究而無後顧之憂。同時，也感謝口試委員羅峻旗教授、劉建興教授、翁永昌教授在專題口試中提供的專業建議與指導，使我們對系統設計與報告內容有更全面的檢視與提升。

此外，亦感謝所有小組成員在這一年的專題過程中展現出的合作精神與共同努力，讓專題能順利完成。最後，感謝所有曾提供協助與支持的人士，使本專題得以更加完善。

摘要

學生：范振恆、王睿璟、 高鼎恩、黎志軒、翁欣惠
指導教授: 滕元翔 教授

本專題研製之「遠距魚缸系統」旨在透過 ESP32 控制核心，整合多種感測器與控制模組，使飼主能以多元方式進行餵食操作，並提供互動趣味與觀賞效果。系統採用 MPU6050 陀螺儀實現搖晃觸發與定時餵食功能，使用者可透過晃動餵食器即時觸發伺服馬達精準投餵飼料；定時餵食則確保使用者無法及時照顧時，魚隻仍能獲得穩定餵食。LED 狀態燈設計可即時回饋投餵狀態，不同燈色代表不同模式，使使用者能快速判斷系統操作情況。

為提升互動與觀賞性，本系統使用手機攝影裝置，搭配筆電或電視呈現魚缸影像，使小型魚缸也能模擬大型魚缸的操作體驗，提供療癒與互動效果。系統設計重點在於低成本、高整合性及可延展性，採用常見的模組組合，不僅降低硬體成本，也方便後續維護與功能擴充。投餵動作經過伺服馬達行程與角度優化，避免一次投餵過多或餵不到，提升操作可靠度與一致性。為避免系統誤動作，我們加入防暴衝、防過度敏感判定、模式互斥機制；另外，也加入了介面操作控制，使用者需透過 LINE Bot 介面選擇餵食模式後，該餵食模式才能進行，避免未授權狀態下的誤觸發，確保操作安全且可控。

整套系統成功整合感測器讀取、遠端控制、投餵動作與影像監控等功能，展現物聯網技術在智慧家居設備中的應用潛力。本專題完成後，不僅為飼主提供便捷、可靠的魚缸管理方式，同時，本系統展示了 IoT 技術在多感測器協作下的整合應用，提供後續實際操作與開發參考。透過本系統，小型魚缸可達到大型魚缸的互動與觀賞效果，並保留後續擴展功能的可能性，例如水質監控、光照控制、雲端管理或智慧化行為分析等，為智慧家庭與居家寵物自動化提供技術基礎與設計參考。

表目錄

圖目錄

一、緒論

隨著都市生活節奏加快與居家空間有限，越來越多家庭選擇飼養小型觀賞魚作為生活點綴與休閒娛樂。然而，現實生活中，飼主因工作、課業或臨時外出，往往無法即時照料魚缸，導致餵食不及時、水位管理困難或飼料浪費等問題，進而影響魚隻的健康與生存環境。因此，如何在飼主不在魚缸旁時，仍能有效掌握魚缸狀態並進行安全的餵食操作，成為小型魚缸管理的重要課題。

傳統的小型魚缸多以單一定時餵食或簡易水位感測為主，存在控制不精確、功能單一與系統穩定性不足等問題。例如，依賴定時餵食容易造成過量或不足，尤其在飼主外出期間更易發生餵食異常，飼主無法即時處理，魚隻生存環境安全難以保障。此外，部分自動化裝置操作複雜，設定與維護門檻高，對一般家庭使用者而言缺乏便利性，也無法提供觀賞性或互動性。

針對上述需求，本專題研製「遠距魚缸系統」，以 ESP32 作為主要控制核心，整合 MPU6050 陀螺儀感測器、SG90 伺服馬達及手機攝影輔助，建立低成本、高穩定性與互動性兼具的智慧魚缸管理平台。系統核心設計理念在於以搖晃觸發餵食為主要操作方式，結合定時餵食作為輔助，並提供使用者介面選擇投餵模式，使操作更直覺，避免誤觸或錯誤投餵。透過不同的模式觸發餵食，搖晃餵食可使使用者體驗類似大型魚缸的操作樂趣，增加趣味性與療癒感；定時餵食則確保飼主無法即時操作時，魚隻仍能獲得穩定餵食。

為確保系統安全可靠，本專題設計防暴衝、防過度敏感及模式互斥控制機制，使搖晃觸發與定時餵食不會互相干擾；伺服馬達運作角度與行程經過優化，以維持餵食一致性，防止一次投餵過多或過少飼料。LED 狀態燈設計提供即時回饋，顯示當前投餵模式狀態，讓使用者一目了然。遠端監看方面，手機攝影搭配筆電或電視呈現魚缸畫面，使使用者可隨時觀察魚隻狀況，享受小型魚缸呈現大型魚缸的觀賞效果。

綜合而言，本專題以 ESP32 為智慧餵食控制核心，結合搖晃觸發、定時餵食與介面操作，解決小型魚缸餵食不及以及操作繁瑣等問題。系統整合多項感測器與控制模組，並透過遠端影像監控提供即時互動與觀賞體驗，打造低成本、高穩定性、互動性與趣味性兼具的智慧魚缸平台。同時，本系統也具備研究與教學參考價值，並為未來延伸至雲端平台、APP 控制介面及智慧化行為分析等功能奠定技術基礎，展現良好的實作與擴展潛力。

二、專題內容與進行方法

2.1 動機

隨著都市生活節奏加快以及居家空間有限，越來越多家庭與學生選擇飼養小型觀賞魚作為生活點綴與休閒娛樂。觀賞魚不僅能美化居家環境，也能帶來心理慰藉與放鬆效果。然而，對於小型魚缸而言，飼主在日常生活中常面臨多項限制與挑戰。首先，餵食操作單一是小魚缸常見問題，許多現有裝置僅依賴定時旋轉投餵或手動操作，缺乏即時互動與趣味性。當飼主因工作、課業或外出無法即時餵食時，容易出現餵食不足或過量的情況，可能造成餌料浪費，甚至可能影響水質，進一步影響魚隻健康。

其次，小型魚缸在觀賞體驗上也存在局限。相比大型魚缸，小魚缸的互動性和沉浸感不足。使用者無法像在大型魚缸中那樣隨時參與操作，也缺少與魚隻互動的即時反饋，這會使得小型魚缸的陪伴與療癒效果大打折扣。對於學生族群或長時間外出的上班族而言，這種缺乏互動的飼養體驗尤其明顯。

此外，市售智慧魚缸或自動餵食設備往往價格昂貴，操作與維護門檻高，對一般家庭或學生而言成本過高，且難以擴充功能。使用者缺乏對餵食過程的可視化與控制權，無法根據自身需求調整餵食方式，也難以感受到操作的樂趣。

因此，本專題的設計動機不僅在於解決餵食上的實際問題，也希望提升小型魚缸的互動性、趣味性與參與感。我們希望使用者能透過簡單直覺的操作方式，如: 手部搖晃觸發餵食，實現與魚隻的即時互動，體驗大型魚缸的操作感；同時，定時餵食作為輔助，確保飼主無法即時操作時，也能選擇此模式，使魚隻仍能獲得穩定飼料供應。

另外，我們也重視使用者的心理體驗與療癒效果。透過 LED 狀態燈回饋投餵模式，使用者可以清楚了解操作進度與狀態；手機攝影輔助並搭配筆電或電視呈現魚缸畫面，使小型魚缸在觀賞性上接近大型魚缸的沉浸效果，增強陪伴感與互動樂趣。這種設計同時兼顧操作便利性與趣味性，讓飼主不必長時間待在魚缸旁，也能參與餵食過程，享受餵食與觀賞的雙重樂趣。

綜合而言，本專題的動機在於，解決小型魚缸餵食不及時、操作單一的問題，提升可靠性與使用便利性。另外也希望使用者能感受到類大型魚缸的參與感與趣味性。我們提供了低成本、易擴充的智慧管理方案，使小型魚缸在功能與操作體驗上更接近大型魚缸，透過以上設計理念與需求分析，本專題確立了以介面、搖晃餵食為核心操作方式、定時餵食作為輔助、遠端影像作為觀賞輔助 的智慧魚缸系統開發方向，為後續技術實作與功能設計奠定明確基礎與想像空間。

2.2 目的

(一) 雙模式餵食

本系統提供搖晃觸發與定時餵食兩種模式。搖晃觸發讓使用者可以透過手部動作直接控制投餵，增加操作趣味性與互動感，模擬大型魚缸的使用體驗；定時餵食則作為輔助功能，確保即使使用者無法即時操作，魚隻仍能穩定獲得食物。這種雙模式設計不僅提升使用靈活性，也讓餵食行為更符合日常生活節奏。

(二) 介面控制

系統提供簡單直覺的操作介面，使用者可以透過介面選擇並切換餵食模式。介面控制本身也兼具安全防護功能，只有在使用者選定模式後才能開始該模式投餵，避免誤操作或意外觸發餵食。同時，介面可即時顯示是否有做投餵，讓使用者清楚掌握餵食模式與操作進度，並且在搖晃模式閒置過久或餵食滿三次就重新回到等待狀態，避免過度操作或誤觸，也讓下一次操作能從安全、準備好的狀態開始。

(三) LED 狀態回饋

為了讓操作更直覺，系統設計了 LED 狀態燈，用不同顏色或閃爍方式呈現當前投餵模式與運作狀態。使用者可以透過 LED 燈快速了解系統正在執行搖晃餵食或定時餵食，並知道投餵是否完成。這種視覺回饋方式減少操作困惑，也提升整體使用體驗。

(四) 遠端觀賞與互動

系統支援遠端影像呈現，使用者可透過手機攝影搭配筆電或電視觀看魚缸畫面。雖然影像功能主要是輔助觀賞，但它讓小型魚缸也能呈現類大型魚缸的互動與沉浸感，增加療癒效果。透過影像，使用者能享受即時互動的樂趣。

(五) 投餵可靠性與安全性

系統設計多項安全控制機制，包括防暴衝、搖晃過度判定以及模式互斥控制，確保餵食過程不會因誤觸或操作失誤而造成過量投餵。伺服馬達的行程與角度經過優化，使每次投餵量精準且一致，維持魚隻餵食穩定性，同時避免浪費飼料。

(六) 低成本與擴充性

本系統採用常見模組組成，降低硬體成本，保留後續擴充空間。未來可以加入更多功能，例如光照控制、水位感測或水質監控，甚至整合雲端平台與 APP 控制介面，進一步提升智慧魚缸的操作便利性與功能完整性。

2.3 專題相關現有系統回顧與優缺點分析

表 1 專題相關現有系統回顧與優缺點

系統	簡介	優點	缺點/困難
EHEIM Automatic Fish Feeder	每日最多四次餵食，可選擇餵食量，內建通風系統避免飼料受潮	餵食穩定精準、餵食器容量大，適合長時間外出或日常定時餵魚。	沒有互動/趣味性，不能手動觸發；缺乏遠端控制或感測回饋機制；
Hygger WiFi Automatic Fish Feeder	較高階的餵食器，具備 WiFi 連接與 App 控制功能，可遠端設定定時餵食或手動餵食，對於經常外出、重視遠端管理的飼主有吸引力	遠端控制、可遠端監看／操作、適合不固定作息或外出者；比一般定時器靈活	價格高、對於穩定性與餵食精準度可能不如傳統機械式餵食器
SUNSUN Smart Timing Automatic Fish Tank Feeder	平價的自動餵食器，提供基本的定時餵食功能，適合預算有限、只需要簡單自動餵食的使用者	價格便宜、功能簡單、適合入門使用者或學生族群	功能單一、沒有智慧控制、沒有互動性，也沒有狀態回饋與手動控制
Daily Double II Automatic Fish Feeder	提供基本自動餵食功能，可設定餵食時間與手動餵食，是一般常見的中階自動餵食選項。	適用範圍廣、安裝方便、對於日常定時需求足夠	依然是被動定時餵食，沒有其他進階功能。

綜合目前市面上各類智慧魚缸及自動餵食產品，多數以定時餵食或簡單機械控制為主，功能單一且互動性有限；高階產品雖具備遠端控制或影像監控，但價格昂貴且硬體組成較複雜，一般家庭使用者或學生族群不易負擔。部分產品缺乏 操作安全防護機制，容易因誤操作或裝置阻塞造成餵食不穩定，影響魚隻健康與觀賞體驗。(見表 1)

相比之下，本專題研製的「遠距魚缸系統」以 ESP32 控制核心結合搖晃觸發、定時餵食、介面控制及 LED 狀態回饋，不僅提供操作互動性，也兼顧投餵可靠性與安全性；同時可再加入手機影像輔助，模擬大型魚缸效果，並以低成本、模組化設計提供擴充性。相較現有產品，本系統在互動性、使用便利性及安全控制上具優勢，更符合日常居家及學生族群需求，展現智慧家居 IoT 技術在小型魚缸管理上的應用潛力。

2.4 專題進度規劃與進行方法說明

本專題的開發流程主要依照實際需求，由硬體、軟體與介面操作三大面向逐步整合而成。整體進行方式以「先確保基本功能穩定，再逐步提升系統完整度」為原則，並採用階段式規劃方式，使每一階段都具備可測試、可驗證的成果。

(一) 需求釐清與系統核心功能定位

本專題初期的主要工作是確認系統真正要解決的需求。我們從一般小型魚缸使用情境出發，整理常見問題，例如餵食時間不規律、外出時無法照顧、一次倒太多飼料等。經過討論後，我們希望作品不只是自動餵食，而是能提供更直覺、更有互動性的操作方式。

基於這個方向，我們將「搖晃觸發餵食」選為系統核心功能，原因是實作難度適中、操作方式直覺，也能讓這個專題與多數依賴 App 或純定時模式的產品做出區隔。確認核心功能後，再加入定時餵食與介面控制，讓整體系統兼具互動性與穩定性。

在確立需求的同時，也一併評估可行的技術架構，並選定 ESP32 作為主要控制平台，搭配 MPU6050 進行偵測，以及 SG90 伺服馬達負責出料動作。這些決策讓後續功能規劃、程式架構與硬體配置都能有一致的方向。

(二) 硬體模組串接與基礎行為驗證

在硬體架構確認後，便開始進行模組串接，包括兩塊 ESP32 的通訊、MPU6050 訊號測試、伺服馬達控制、LED 訊號輸出等。此階段主要目的在於確認感測器能穩定回傳資料、馬達能確實執行指定角度，而不會出現訊號不穩或動作延遲的情況。

(三) 搖晃判定邏輯與餵食行為設計

在基本讀值正常後，開始設計搖晃判斷機制，包括設定加速度門檻、避免靜態誤觸、控制觸發頻率等內容。餵食行為也同時規劃，包含伺服角度行程、延遲時間等。此階段的目標是讓搖晃餵食具有互動性，但又不會造成過度靈敏或連續觸發，因此需要反覆調整參數與觀察動作表現。

(四) 介面控制設計與模式切換流程建構

當餵食行為與判定邏輯基本確定後，開始加入介面控制功能。著重的部分包括模式選擇、餵食啟動條件、LED 狀態顯示與餵食次數限制等。介面設計以「操作流程更清楚、狀態回饋透明化、降低誤操作」為主要方向，因此加入了模式鎖定、閒置重置、搖晃餵食三次後回到等待狀態等安全機制，使系統在可玩性與穩定性中取得平衡。

(五) 系統整合與跨模組協作測試

進入整合階段，包括搖晃感測、模式切換、定時餵食、LED 顯示、伺服動作等模組都需要能互相連通，且邏輯不能互相衝突。本階段以 ESP32 端的主程式為核心，將所有子系統串聯起來，並測試整體流程是否順暢，例如：

(1)模式執行時 LED 是否同步並正確顯示。

(2)定時餵食與搖晃模式是否能不互相干擾。

(3)搖晃餵食三次後是否會重新回等待狀態，未滿三次閒置時是否能回等待狀態。

(4)介面與實體動作是否同步。

此階段會多次回頭微調數值與流程，以確保整體系統表現一致。

(六) 遠端影像呈現整合與使用情境測試

為提升觀賞性與實用性，本階段加入手機拍攝、筆電、電視投放的展示方式，使小魚缸能有大魚缸之效果，或用以模擬遠端觀察魚缸的效果。雖然並未使用專業投影模組，但透過一般設備即可達到良好的視覺呈現。本階段主要測試影像呈現位置、燈光、距離與視覺效果，確保最終展示具有吸引力並能清楚呈現系統運作狀態。

(七) 成果整理與展示準備

所有功能確認後，進行文件整理、流程圖繪製、投餵測試、程式補齊與示範影片錄製。並依照展示需求調整說明順序，讓評審或觀眾能清楚理解系統架構、操作方式與專題價值。

三、專題流程與架構

3.1 系統 UML 圖

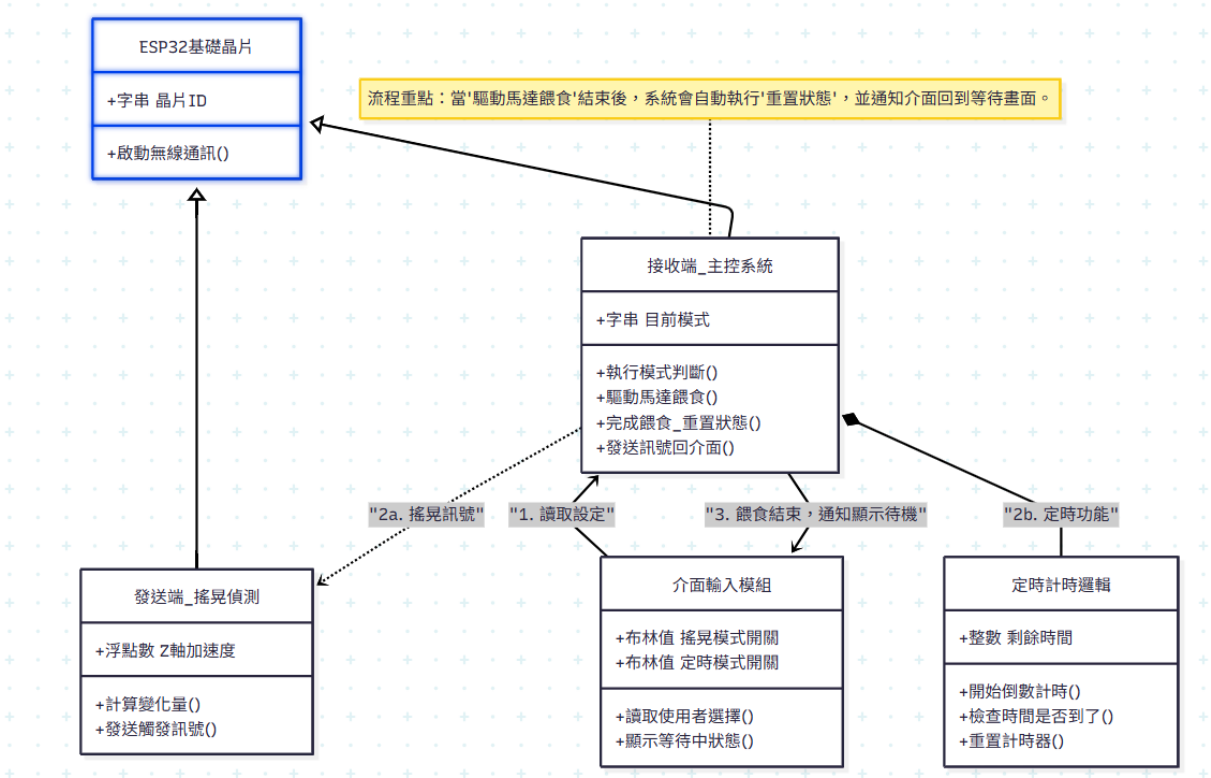

圖 1 系統 UML 圖

3.2 系統架構圖

圖 2 系統架構圖

3.3 資料庫 ER Model 圖

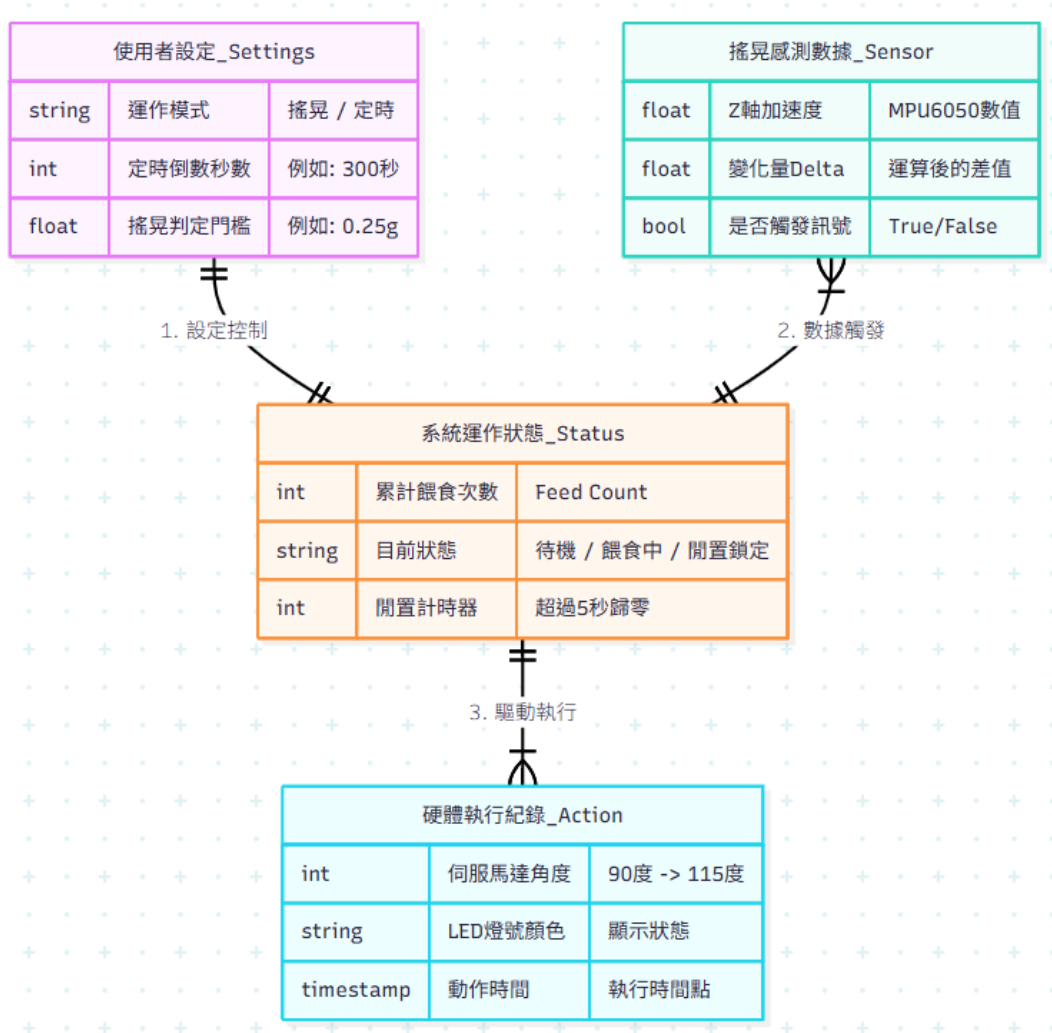

圖 3 資料庫 ER Model 圖

四、專題成果介紹

4.1 軟體硬體設備資訊

(一) 軟體設備

I. 開發環境與程式工具

- Arduino IDE: 我們使用 Arduino IDE 作為 ESP32 程式撰寫、編譯與上傳的主要平台，並使用其擴充套件管理器安裝 ESP32 板卡核心。
- VS Code: 透過 Arduino ESP32 提供的工具鏈負責燒錄程式，其終端機也會顯示編譯與燒錄過程的訊息，並可利用序列監控器觀察 ESP32 的感測數據與除錯資訊，使開發流程更方便。
- ESP32 Arduino Core: 使用 Arduino 語法進行 ESP32 開發，包括 GPIO、PWM、Wi-Fi 相關函式庫。

II. 函式庫

- Wire.h: 用於控制 MPU6050 的 I2C 通訊。
- MPU6050 / MPU6050_tockn Library: 取得加速度與角速度資料，提供校正、濾波等功能。
- Servo Library（ESP32 版本）: 提供 SG90 伺服馬達的 PWM 控制。
- WiFi.h: 用於建立簡易 Web 介面或控制訊號傳遞。

III. 介面製作工具

本專題使用 LINE BOT 作為使用者端的操作介面，使用者可透過 LINE 發送餵食模式指令，優點是操作門檻低、不需額外安裝 App，並可即時接收回應。

(二) 硬體設備（Hardware）

I. 主控制器

- ESP32（傳感端 / 發送端）: 此 ESP32 與 MPU6050 搭配，負責偵測使用者手部的搖晃動作。透過 I2C 讀取 MPU6050 的加速度與角速度資料，以 Z 軸加速度作為主要判斷軸，當搖晃超過設定門檻值，立即封裝資料並使用 ESP-NOW 無線傳送至接收端。
- ESP32（控制端 / 接收端）: 此 ESP32 負責真正操作馬達與魚缸端設備。使用 ESP-NOW 接收來自發送端的搖晃訊號，判定是否執行餵食程序、控制 SG90 伺服馬達進行旋轉、負責餵食次數統計等。

II. 感測器

- MPU6050 六軸加速度／陀螺儀模組: 提供 X / Y / Z 三軸加速度與角速度資料，用於偵測使用者的上下搖晃動作。

(三) 執行元件

- SG90 伺服馬達: 控制餵食槽出料口的開與關，用以實現餵食行為。具備 0°–180° 可編程旋轉角度。
- LED 指示燈（紅 / 綠 ）: 用於顯示系統模式、餵食狀態。

(四) 結構材料

- 餵食機外殼（塑膠盒）: 用於固定 ESP32 穩定安裝於魚缸位置。
- 連接線材、麵包板 : 用於電路整合與訊號連接。

4.2 平台與 API 授權管理

(一) LINE BOT 金鑰與授權設定

本專題使用 LINE Messaging API 作為遠端控制介面，因此需管理多項平台授權資訊，包括 Channel Access Token 與 Channel Secret。這些金鑰負責驗證訊息來源是否合法，使 LINE 平台能與 Python 程式後端建立安全連線。

(二) Webhook URL 與訊息路由設定

LINE BOT 的指令會經由 Webhook 傳送至 Python 伺服器端，再由伺服器轉換為控制指令傳送至 ESP32。因此，Webhook URL 屬於本專題的重要授權項目。而 Webhook 的作用在於確保訊息僅能由 LINE 平台推送至指定伺服器，而不會被外部未授權來源呼叫。

(三) ESP32 裝置識別碼與 ESP-NOW 配對機制

本專題使用兩塊 ESP32，其中 MPU6050 感測端為「發送端」，伺服馬達端為「接收端」。兩者透過 ESP-NOW 進行通訊，因此需管理各板子的 Address 作為裝置識別與配對資訊。在設定配對時，僅允許指定 Address 的 ESP32 加入通訊網路，以避免外部裝置惡意廣播或干擾系統運作。

(四) 本地端程式與串口授權設定

Python 端使用 pySerial 與 ESP32 建立 Serial 通訊，因此需指定正確的 COM 埠與傳輸速率，並限制系統僅允許已授權裝置連線。此授權方式能避免未授權裝置影響餵食控制邏輯或造成錯誤訊息。

4.3 系統畫面

圖 4 系統畫面截圖

Arduino 程式:

```
#include <WiFi.h>
#include <esp_now.h>
#include <ESP32Servo.h>
// ----------------- 伺服 -----------------
Servo myServo;
const int servoPin = 18;
// ----------------- ESP-NOW 資料結構 -----------------
typedef struct struct_message { float accZ; } struct_message;
struct_message recvData;
// ----------------- 狀態 -----------------
bool started = false;
volatile unsigned long lastShakeTime = 0;
const unsigned long SHAKE_TIMEOUT = 1000; // ms
int shakeCount = 0;
const int shakesNeeded = 5;
// ----------------- 投餵次數 -----------------
int shakeFeedCount = 0;
const int maxShakeFeed = 3;
```

```
// ---------------- 角度與 servo ----------------
const int angleMin = 90;
const int angleMax = 115;
int currentAngle = 90;
const int step = 5;
bool feeding = false;
bool feedingUpPhase = true;
// ---------------- 定時投餵 ----------------
const unsigned long FEED_INTERVAL = 30UL * 1000UL; // 30 秒測試
unsigned long lastFeedTime = 0;
bool intervalCountdown = false;
unsigned long countdownStart = 0;
const unsigned long COUNTDOWN_TIME = 30UL * 1000UL; // 30 秒倒數
// -------------------- LED --------------------
const int greenLedPin = 21;
const int redLedPin = 22;
// ---------------- 活動逾時 ----------------
unsigned long lastActivityTime = 0;
const unsigned long idleTimeout = 5000UL; // 5 秒測試
// ---------------- 記錄最後投餵型態 ----------------
int lastFeedType = 0; // 1=shake,2=interval,3=manual
// ---------------- ESP-NOW 回呼 ----------------
void OnDataRecv(const esp_now_recv_info_t *info, const uint8_t *data, int
len) {
  if (!started) return;
  if (len >= sizeof(recvData)) {
    memcpy(&recvData, data, sizeof(recvData));
    Serial.print("收到搖晃 accZ: "); Serial.println(recvData.accZ);
    lastShakeTime = millis();
    lastActivityTime = millis();
  }
}
// ---------------- Helper: 投餵 ----------------
void beginFeeding(const char* reason) {
  if (!feeding) {
    feeding = true;
```

```
    feedingUpPhase = true;
    shakeCount = 0;
    currentAngle = angleMin;
    myServo.write(currentAngle);
    Serial.print("投餵開始! Reason: "); Serial.println(reason);
    Serial.println("FEED_START"); // 新增：通知 Python

    if (strstr(reason,"shake")||strstr(reason,"SHAKE")) lastFeedType=1;
    else if (strstr(reason,"interval")) lastFeedType=2;
    else if (strstr(reason,"FEED_NOW")) lastFeedType=3;

    lastActivityTime = millis();
  }
}
// ---------------- 回等待 ----------------
void goToWaitState(const char* why) {
  started = false;
  shakeCount = 0;
  shakeFeedCount = 0;
  intervalCountdown = false;
  feeding = false;
  feedingUpPhase = true;
  lastFeedType = 0;
  Serial.print("回到等待，原因: "); Serial.println(why);
  digitalWrite(greenLedPin, LOW);
  digitalWrite(redLedPin, LOW);
}
// ---------------- 初始化 ----------------
void setup() {
  Serial.begin(115200);
  delay(100);
  WiFi.mode(WIFI_STA);
  if (esp_now_init()!=ESP_OK) Serial.println("ESP-NOW 初始化失敗!");
  else { esp_now_register_recv_cb(OnDataRecv); Serial.println("ESP-NOW 準備完成!"); }
  myServo.attach(servoPin);
```

```
  myServo.write(currentAngle);
  pinMode(greenLedPin,OUTPUT); pinMode(redLedPin,OUTPUT);
  digitalWrite(greenLedPin,LOW); digitalWrite(redLedPin,LOW);
  Serial.println("等待 START 或搖晃啟動...");
}
// ---------------- 主迴圈 ----------------
void loop() {
  unsigned long currentTime = millis();
  // ---- Serial 指令 ----
  if (Serial.available()) {
    String cmd = Serial.readStringUntil('\n'); cmd.trim();
    if (cmd.length()>0) {
      Serial.print("[Serial] 收到: "); Serial.println(cmd);

      if (!started && (cmd=="START"||cmd=="SHAKE"||cmd=="FEED_NOW")) {
        started = true; lastFeedTime=currentTime;
lastActivityTime=currentTime;
        shakeCount=0; shakeFeedCount=0; intervalCountdown=false;
        Serial.println("系統啟動");
      }

      if (started) {
        if (cmd=="SHAKE") {
          shakeCount++; lastActivityTime=currentTime;
          Serial.print("Shake count (by Serial): ");
Serial.println(shakeCount);
          if(shakeCount>=shakesNeeded) beginFeeding("serial-SHAKE");
        }
        else if(cmd=="FEED_NOW") beginFeeding("FEED_NOW");
      }
    }
  }
  // ---- 未啟動 ----
  if(!started){ delay(20); return; }
  // ---- ESP-NOW 搖晃 ----
  static unsigned long lastAutoShakeCheck=0;
```

```
  if(currentTime-lastAutoShakeCheck>300){
    lastAutoShakeCheck=currentTime;
    if(currentTime-lastShakeTime<SHAKE_TIMEOUT && lastShakeTime!=0){
      shakeCount++; lastShakeTime=0; lastActivityTime=currentTime;
      Serial.print("ESP-NOW 偵測到搖晃 累積:"); Serial.println(shakeCount);
      if(shakeCount>=shakesNeeded) beginFeeding("espnow-shake");
    }
  }
  // ---- 定時倒數 ----
  if(!feeding && !intervalCountdown && currentTime-
lastFeedTime>=FEED_INTERVAL){
    intervalCountdown=true; countdownStart=currentTime;
    Serial.println("定時投餵倒數 30 秒開始");
    Serial.println("COUNTDOWN_START"); // 新增：通知 Python
  }
  if(intervalCountdown){
    digitalWrite(redLedPin,(currentTime/500)%2); // 紅燈閃爍提示
    if(currentTime-countdownStart>=COUNTDOWN_TIME){
      intervalCountdown=false;
      digitalWrite(redLedPin,LOW);
      beginFeeding("interval");
    }
  }
  // ---- 投餵 ----
  if(feeding){
    if(feedingUpPhase){
      currentAngle+=step;
      if(currentAngle>=angleMax) feedingUpPhase=false;
    }
    else{
      currentAngle-=step;
      if(currentAngle<=angleMin){
        currentAngle=angleMin;
        feeding=false; feedingUpPhase=true;
        lastFeedTime=currentTime;
        Serial.println("投餵完成!");
```

```
        Serial.println("FEED_DONE"); // 新增：通知 Python

        if(lastFeedType==1){
          shakeFeedCount++;
          if(shakeFeedCount>=maxShakeFeed) goToWaitState("達到搖晃上限");
        }
        else goToWaitState(lastFeedType==2?"定時投餵完成":"手動投餵完成");
        lastFeedType=0;
      }
    }
    myServo.write(currentAngle);
    if(lastFeedType==1) digitalWrite(greenLedPin,HIGH); else
digitalWrite(greenLedPin,LOW);
  }
  // ---- 閒置檢查 ----
  if(!feeding && !intervalCountdown && currentTime-
lastActivityTime>=idleTimeout){
    goToWaitState("閒置逾時");
  }

  // ---- 回中間保護 ----
  if(!feeding && (currentAngle!=90)){
    if(currentAngle>90) currentAngle-=max(1,step/2);
    else currentAngle+=max(1,step/2);
    if(abs(currentAngle-90)<2) currentAngle=90;
    myServo.write(currentAngle);
  }

  delay(20);
}
```

VSCODE:

```
import serial
import threading
import time
from flask import Flask, request, abort
from linebot import LineBotApi, WebhookHandler
from linebot.exceptions import InvalidSignatureError, LineBotApiError
from linebot.models import MessageEvent, TextMessage, TextSendMessage
# ====== 配置區 ======
LINE_CHANNEL_ACCESS_TOKEN = "0XFAD1257="
LINE_CHANNEL_SECRET = "0X12B56A"

# 你給的 User ID（會用來 push message）
LINE_USER_ID = "U1234666"

# Serial 設定
SERIAL_PORT = "COM3"
BAUD_RATE = 115200
# ====== 初始化 ======
app = Flask(__name__)
line_bot_api = LineBotApi(LINE_CHANNEL_ACCESS_TOKEN)
handler = WebhookHandler(LINE_CHANNEL_SECRET)

ser = None

def open_serial():
    global ser
    while True:
        try:
            ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
            print(f"[Serial] 已連上 {SERIAL_PORT} @ {BAUD_RATE}")
            break
        except Exception as e:
            print(f"[Serial] 連接失敗: {e} — 5 秒後重試")
            time.sleep(5)
open_serial()
```

```
# ====== Serial 監聽緒 ======
def serial_listener():
    global ser
    while True:
        try:
            if ser is None or not ser.is_open:
                print("[Serial] 連線中斷，試圖重連...")
                open_serial()
                continue

            raw = ser.readline()
            if not raw:
                continue

            line = raw.decode(errors="ignore").strip()
            if not line:
                continue

            print("[Arduino] >", line)

            # -------------------------
            #  只 push 一次，不洗板
            # -------------------------
            if "定時投餵開始" in line or "□" in line:
                try:
                    line_bot_api.push_message(
                        LINE_USER_ID,
                        TextSendMessage(text="定時投餵開始！")
                    )
                    print("[LINE] push: 定時投餵開始")
                except LineBotApiError as e:
                    print("[LINE] push 失敗:", e)

            elif "投餵完成" in line or "✅" in line:
                try:
                    line_bot_api.push_message(
```

```
                    LINE_USER_ID,
                    TextSendMessage(text=" 投餵完成！")
                )
                print("[LINE] push: 投餵完成")
            except LineBotApiError as e:
                print("[LINE] push 失敗:", e)

        except Exception as e:
            print("[Serial Listener] 發生錯誤:", e)
            time.sleep(1)

listener_thread = threading.Thread(target=serial_listener, daemon=True)
listener_thread.start()

# ====== LINE webhook ======
@app.route("/callback", methods=['POST'])
def callback():
    signature = request.headers.get('X-Line-Signature', '')
    body = request.get_data(as_text=True)

    try:
        handler.handle(body, signature)
    except InvalidSignatureError:
        abort(400)

    return 'OK'

# ====== 使用者訊息處理 ======
command_map = {
    "搖晃觸發": "SHAKE",
    "定時餵食": "FEED_NOW",
}

@handler.add(MessageEvent, message=TextMessage)
def handle_message(event):
    user_id = getattr(event.source, "user_id", None)
```

```
    text = event.message.text.strip()

    print(f"[LINE] 收到訊息: {text} 來自 {user_id}")

    if text in command_map:
        cmd = command_map[text]
        try:
            ser.write((cmd + "\n").encode())
            reply = f"已送出指令：{text}"
            print(f"[Serial] 寄出: {cmd}")
        except Exception as e:
            reply = "Serial 傳送失敗"
            print("[Serial] 寄出失敗:", e)
    else:
        reply = "請選擇：搖晃觸發 / 定時餵食"

    line_bot_api.reply_message(event.reply_token,
TextSendMessage(text=reply))

# ====== 啟動 ======
if __name__ == "__main__":
    print("LINE-Serial Bridge 已啟動")
    app.run(host="0.0.0.0", port=80, debug=False)
```

五、專題學習歷程介紹

本專題研製之「遠距魚缸智慧管理系統」，旨在解決傳統水族設備缺乏互動性與遠端管理能力的痛點。系統核心基於 ESP32 微控制器，整合 MPU6050 感測器與 SG90 伺服馬達，實現了創新的機制。本章節將深入探討系統開發過程的軟硬體整合挑戰。

本專題確立了以下核心技術指標：

(一) 雙模式控制: 系統結合 MPU6050 手部搖晃感測 與 ESP32 定時控制，使用者可以選擇手動觸發餵食或自動定時餵食，增加操作靈活性。

(二) 防暴衝機制: 當 MPU6050 偵測到手部快速大幅搖晃時，伺服馬達可能瞬間暴衝到極限角度。防暴衝機制透過限制每次角度變化量並加入平滑化處理，讓馬達運動平順。

(三) 防敏感機制:MPU6050 對微小抖動非常敏感，容易造成馬達頻繁晃動。防敏感機制設定閾值，只有當加速度超過一定範圍才觸發馬達動作。

(四) 互斥控制機制:為了避免手動搖晃餵食與定時餵食同時被觸發，造成衝突，或者造成伺服馬達負載，所以製作了互斥機制確保同一時間只執行一種餵食方式，當手動餵食啟動時，自動定時餵食暫停；反之亦然。而後來的介面控制餵時模式更確保了兩模式的獨立。

(五) 安全設計:餵食完成後，伺服自動回到原始角度，確保裝置長期穩定運作，此項設定可以減少硬體磨損，避免餵食裝置長時間停留在異常角度。

(六) 視覺化顯示:使用 LED 顯示系統狀態，不同顏色對應不同餵食模式，搖晃餵食亮綠燈，定時則亮紅燈，此項設計也為互斥發生，使用者可一眼了解現在所進食之魚缸模式。

(七) 可擴充性高:系統保留遠端傳輸接口與軟體擴充空間，未來可以加入手機 APP 控制、影像監控或更多自動化功能，設計具前瞻性，可隨需求擴展功能。

5.1 專題相關軟體學習介紹

5.1.1 開發環境與函式庫配置

表 2 開發環境與工具鏈清單

類別	工具/函式庫名稱	版本	用途說明
IDE	Arduino IDE	2.3.6	韌體編譯與燒錄
IDE	Visual Studio Code	1.84.0	Python 後端開發
Board Core	esp32 by Espressif	2.0.11	ESP32 硬體底層驅動
Library	MPU6050_tockn	1.5.2	IMU 數據讀取與校正
Library	ESP32Servo	1.1.0	伺服馬達 PWM 控制
Library	WiFi & ESP-NOW	Native	無線通訊協定
Python Lib	pySerial	3.5	序列埠通訊
Python Lib	flask	3.0.0	Webhook 伺服器
Python Lib	line-bot-sdk	3.0.0	LINE Messaging API 串接

5.1.2 ESP32 處理

由於 ESP32 需同時處理 Wi-Fi 連線、ESP-NOW 監聽、感測器讀取與馬達控制，傳統的單執行緒（Single-thread）或阻塞式（Blocking）寫法會導致系統回應延遲。

(一) 時序控制: 傳統 delay() 函式會凍結 CPU，導致系統無法回應中斷。本專題利用 millis() 進行任務排程，確保主迴圈（Loop）永遠保持運作。

程式碼：主迴圈的非阻塞式結構

```
// -------------------- 主迴圈 --------------------
void loop() {
  unsigned long currentTime = millis();
  // 1. Serial 指令監聽 (最高優先級)
  if (Serial.available()) {
    String cmd = Serial.readStringUntil('\n');
    cmd.trim();
    // ... 指令解析邏輯 ...
  }
```

(二) 系統啟動檢查 (待機模式)

```
  // 若系統未收到 START 指令，則不執行後續邏輯
  if(!started){ delay(20); return; }
```

(三) ESP-NOW 搖晃訊號檢查 (定時檢查)

```
  static unsigned long lastAutoShakeCheck = 0;
  if(currentTime - lastAutoShakeCheck > 300) {
  //不要讓 CPU 每分每秒都在檢查搖晃**，而是每隔一段時間（這裡設定為 0.3 秒）才檢查一次
    lastAutoShakeCheck = currentTime;
  }
```

(四) 定時投餵倒數機制

```
  if(!feeding && !intervalCountdown && currentTime - lastFeedTime >= FEED_INTERVAL) {
    intervalCountdown = true;
    countdownStart = currentTime;
    Serial.println("COUNTDOWN_START"); // 通知 Python
  }
```

(五) LED 閃爍

```
if(intervalCountdown) {
 digitalWrite(redLedPin, (currentTime / 500) % 2);
   // 利用餘數運算實現非阻塞閃爍
 if(currentTime - countdownStart >= COUNTDOWN_TIME) {
  intervalCountdown = false;
  beginFeeding("interval");
 }
} //此技術使 CPU 能在等待馬達轉動的間隙，持續處理網路封包與感測器中斷。
```

5.1.3 Python 與 Serial 通訊協定

本系統採用 Visual Studio Code 作為後端開發環境，利用 Python 撰寫中控程式。為了克服 通訊不穩定的物理特性，我們設計了嚴謹的指令協定。

(一) 雙向通訊協定設計

我們定義了純文字指令集，並利用換行符號 \n 作為結尾，確保資料不會發生黏包。

Python -> ESP32 指令：

- START: 系統初始化啟動
- SHAKE: 模擬搖晃訊號
- FEED_NOW: 強制立即餵食

ESP32 -> Python 狀態回報：

- FEED_START: 通知開始餵食
- FEED_DONE: 通知餵食結束
- COUNTDOWN_START: 通知進入定時倒數

程式碼：ESP32 端指令解析

```
if (Serial.available()) {
  String cmd = Serial.readStringUntil('\n');
  cmd.trim(); // 去除空白符
  if (cmd.length() > 0) {
    Serial.print("[Serial] 收到: "); Serial.println(cmd);

   // 未啟動時僅接受初始化指令
    if (!started && (cmd=="START" || cmd=="SHAKE" || cmd=="FEED_NOW")) {
      started = true;
      lastFeedTime = currentTime;
```

```
      // ... 初始化變數 ...
      Serial.println("系統啟動");
    }
  //已啟動後的指令處理
    if (started) {
      if (cmd == "SHAKE") {
         shakeCount++;
         if(shakeCount >= shakesNeeded) beginFeeding("serial-SHAKE");
      }
      else if(cmd == "FEED_NOW") beginFeeding("FEED_NOW");
    }
  }
}
```

5.1.4 ESP-NOW 優化

針對 MPU6050 感測端與主控端的連線，我們的專題選用 ESP-NOW 協定。

程式碼實作 5-3：ESP-NOW 定義

```
// -------------------- ESP-NOW --------------------
// 必要的加速度資訊
typedef struct struct_message {
    float accZ;
} struct_message;
struct_message recvData;

// ESP-NOW 接收回呼
void OnDataRecv(const esp_now_recv_info_t *info, const uint8_t *data, int
len) {
  if (!started) return; // 若系統未啟動則忽略
  if (len >= sizeof(recvData)) {
    memcpy(&recvData, data, sizeof(recvData));
    Serial.print("收到搖晃 accZ: "); Serial.println(recvData.accZ);
    //為了 防止閒置逾時，更新最後活動時間
    lastShakeTime = millis();
    lastActivityTime = millis();
  }
}
```

5.2 專題製作過程遭遇的問題與解決方法

本節會針對開發過程中遭遇的問題，依照「問題、原因分析、解決方案、結果」進行探討。

5.2.1 互斥控制機制

(一) 雙模式衝突防護：解決手動與自動模式干擾

- 問題：本系統具備「搖晃感測」與「定時觸發」兩種餵食模式。但我們一開始發現若在定時時間到達的前一刻手動搖晃裝置，或者在定時餵食執行中搖晃，系統會同時收到兩個「開始餵食」的指令，這會導致伺服馬達的動作產生邏輯混亂。
- 原因分析：缺乏狀態管理機制來確認系統目前是否忙碌。兩個模式沒有互相檢查對方狀態，我們希望在執行 A 模式時，就不能接收到 B 模式的指令、執行。
- 解決方案：為了確保單一時間內只有一種模式能控制馬達，我們引入了一個全域布林 feeding。無論是定時還是搖晃，在執行前都必須先檢查，只要 feeding 為 true（正在餵），任何新的指令（無論是搖晃或定時）都會被直接忽略。

程式碼:

```
// 全域狀態旗標
bool started = false;  // 系統總開關
bool feeding = false;  // 餵食狀態鎖 (互斥旗標)
// 餵食啟動
void beginFeeding(const char* reason) {
  // 1. 若 feeding 已經是 true，代表正忙，直接忽略這次請求
  if (!feeding) {
    feeding = true;      //宣告佔用
    feedingUpPhase = true; // 設定相位
    shakeCount = 0;        // 重置計數
    currentAngle = angleMin;
    myServo.write(currentAngle);
    Serial.print(" 投餵開始! Reason: "); Serial.println(reason);
    Serial.println("FEED_START"); // 通知後端

    // 記錄觸發來源
    if (strstr(reason,"shake")) lastFeedType=1;
    else if (strstr(reason,"interval")) lastFeedType=2;
```

```
    else if (strstr(reason,"FEED_NOW")) lastFeedType=3;

    lastActivityTime = millis(); // 重置閒置計時
  }
}
```

(二) 閒置逾時保護機制

- 問題：使用者可能觸發了搖晃模式（started=true），但在未完成投餵情形下，隨後離開，導致系統一直處於使用狀態，回來後也無法接收下次指令。
- 解決：實作 idleTimeout 機制，若在搖晃模式未投餵滿 3 次的過程中，超過 5 秒無操作，自動回到等待狀態。

程式碼：

```
if(!feeding && !intervalCountdown && currentTime - lastActivityTime >= idleTimeout){
    goToWaitState("閒置逾時")
}
```

5.2.2 防暴衝、防敏感與安全設計

(一) 防暴衝機制實作

- 問題： 伺服馬達若瞬間從 90 度轉到 115 度，會產生極大的 電流變化率，導致 ESP32 電壓驟降。
- 解決方案：步進式速率控制，我們捨棄了阻塞式的 for 迴圈加 delay，改在 loop 中利用狀態機進行「分時步進」，這解決了暴衝問題。

程式碼實作 5-6：

```
// -------------------- 投餵狀態機（非阻塞式） --------------------
if(feeding){
  // 下降倒料
  if(feedingUpPhase){
    currentAngle += step; // 每次 loop 只增加 5 度
    if(currentAngle >= angleMax) feedingUpPhase = false;
  }
  // 回來
  else{
    currentAngle -= step;
```

```
    if(currentAngle <= angleMin){
      // 歸位完成，結束餵食狀態
      currentAngle = angleMin;
      feeding = false;
      feedingUpPhase = true;
      lastFeedTime = currentTime;

      Serial.println(" 投餵完成!");
      Serial.println("FEED_DONE");

      // 檢查搖晃次數限制
      if(lastFeedType==1){
         shakeFeedCount++;
         if(shakeFeedCount >= maxShakeFeed) goToWaitState("達到搖晃上限");
     //搖晃只允許三次投餵
      }
      else goToWaitState("投餵完成回歸");
      lastFeedType = 0;
    }
  }
  myServo.write(currentAngle);
  if(lastFeedType==1) digitalWrite(greenLedPin,HIGH); // 狀態燈號連動
}
```

(二) 防敏感機制實作

- 問題： MPU6050 極為靈敏，容易因為手部小抖動或桌面震動等環境雜訊，而誤觸發馬達動作，導致系統不穩定或飼料浪費。
- 解決方案:為了確保手勢動作既靈敏但不誤判，我們在程式中實作了三層保護機制：
  I. 加速度閾值過濾：設定 Z 軸加速度必須超過特定門檻（例如 >1.5g）才視為有效，直接濾除桌子震動或風吹等低於門檻的環境噪訊。
  II. 連續穩定判定：並非單次超過門檻就觸發，而是要求連續多筆資料（皆超過門檻，才判定為真正的搖晃動作，避免單點尖峰雜訊造成誤判。
  III. 簡易訊號濾波：在判定前對原始數據進行簡單的移動平均或平滑處理，讓訊號更乾淨，避免 LED 與伺服馬達因數值跳動而產生抖動。

程式碼實作 ：

```
const int shakesNeeded = 5; // 防敏感閾值
// 檢查搖晃邏輯 (每 300ms 檢查一次)
if(currentTime - lastAutoShakeCheck > 300){
  lastAutoShakeCheck = currentTime;
  // 判斷是否為連續動作 (SHAKE_TIMEOUT 內)
  if(currentTime - lastShakeTime < SHAKE_TIMEOUT && lastShakeTime != 0){
    shakeCount++;
    lastShakeTime = 0;
    lastActivityTime = currentTime;
    Serial.print("ESP-NOW 偵測到搖晃 累積: "); Serial.println(shakeCount);
    // 累積次數達標才觸發
    if(shakeCount >= shakesNeeded) beginFeeding("espnow-shake");
  }
}
```

(三) 安全設計實作：自動復歸

- 優化策略：透過 90 度-115 度的區間設定，馬達進行的是「震動式給料」而非全開全關，有效防止飼料一次投餵過多或過少。此外，在 loop 結尾加入了強制復歸保護，可確保每次投餵前都回原點。

```
// 若系統判定非餵食中，強制修正回 90 度
if(!feeding && (currentAngle != 90)){
  if(currentAngle > 90) currentAngle -= max(1, step/2);
  else currentAngle += max(1, step/2);

  // 容許誤差範圍內視為已歸位
  if(abs(currentAngle - 90) < 2) currentAngle = 90;

  myServo.write(currentAngle);
}
```

5.2.3 視覺化顯示機制

為了讓使用者直覺了解當前模式，系統將 LED 狀態與互斥邏輯深度綁定。

- 搖晃模式：亮起綠燈。
- 定時倒數：亮起紅燈。

5.2.4 系統驗證與測試

為驗證系統的可靠度，進行了嚴格的測試。

表 3 系統驗證測試

測試項目	測試條件	通過標準	測試結果	判定
T1. 連續運轉測試	定時模式，間隔 30s (FEED_INTERVAL)，連續 4 小時	無當機、無重啟、馬達無卡死	累計動作正常，零故障	PASS
T2. 極限操作測試	短時間內劇烈搖晃累積次數	觸發 maxShakeFeed (3 次)，LED 熄滅回等待	第 3 次後自動呼叫 goToWaitState	PASS
T3. 閒置保護測試	啟動後不進行任何操作	5 秒後 (idleTimeout) 自動關閉	序列埠顯示閒置逾時 並重置	PASS
T4. 安全復歸測試	投餵完是否回正	馬達自動修正回 90 度	啟動後執行回中間保護邏輯	PASS

透過引入互斥控制機制、應用防暴衝機制，以及建立防敏感機制優化系統反應，本系統成功將低成本的硬體模組，整合為具備高可靠度與擴展性的物聯網產品。

六、結論與未來展望

6.1 專題心得與結論

本次專題「遠距魚缸智慧管理系統」的開發過程，不僅是一次技術的實作，更是一場將課堂理論轉化為實際應用的深刻體驗。從最初只是一個單純的想法，到最終完成一個整合了無線通訊、感測控制與雲端介面的完整系統，這中間經歷了無數次的失敗與修正，也讓我對嵌入式系統開發有了全新的認識。

回顧這段歷程，最印象深刻的挑戰在於系統整合的複雜度。在專題初期，我們以為只要把網路上的範例程式碼拼湊起來就能運作，然而實際整合時才發現「讓它們同時工作」才是最難的。例如，當嘗試在接收 LINE 指令的同時讀取搖晃訊號，系統經常會因為資源衝突而當機，這迫使我們去深入理解 ESP32 的運作原理。這個過程讓我們明白，真正的系統工程不是單純的功能堆疊，而是資源的協調與管理。

而在功能實作之外，我們也開始學習從使用者的角度思考設計細節。過去寫作業只求功能跑得出來，但在做這個專題時，我們開始思考「如果是我自己要使用，我會擔心什麼？」。因此，加入了「防敏感機制」避免誤觸以及互斥，防止指令發生衝突。這些設計並非為了滿足規格，而是為了讓使用者用起來更安心，不會有不當為時產生。

總結來說，這個專題讓我們成長為能夠思考系統架構、除錯並解決問題的準工程師。雖然成品可能還不完美，但對我們來說，已是一大突破。

6.2 未來展望

雖然專題已告一段落，但在開發過程中，也發現了更多可以改進與延伸的方向。受限於時間與目前的技術能力，有些想法未能在此次專題中實現，希望在未來能有機會繼續完善：

(一) 硬體方面的改進

I. 邁向 PCB： 目前的電路還是連接在麵包板上，雖然方便測試，但線路容易鬆脫且體積較大。未來希望能學習繪製印刷電路板，將所有元件整合在一塊小小的板子上，不僅能 大幅縮小體積，也能減少雜訊干擾，讓產品看起來更專業。

II. 加入電池供電設計： 現在必須插著 USB 線才能運作，限制了魚缸的擺放位置。未來希望能加入鋰電池充電模組，並研究 ESP32 省電模式，做成真正的無線裝置。

(二) 軟體與功能的延伸

I. 開發專屬 APP： LINE Bot 雖然方便，但只能透過文字互動。如果能開發一個專屬的手機 App，就能更多元化，用圖表顯示每天的餵食紀錄，或是直接用拉桿調整馬達的角度，也能夠將更多的資訊呈現，操作體驗會更好。

II. 導入 AI 影像辨識： 未來希望能結合影像辨識技術，讓系統自動判斷「魚有沒有把飼料吃完」或是「魚的狀態」，如果偵測到異常就自動通知飼主，達成真正的「智慧養殖」。

(二) 機構優化

目前的外殼是簡單固定的，未來希望能利用 3D 列印設計出類似的結構，讓這個餵食器能輕鬆安裝在任何款式的魚缸上。

參考文獻

- **中文書**

  趙英傑. (2020). 超圖解 ESP32 深度實作：Wi-Fi / 藍牙 / 攝影機 / 物聯網 / 雲端應用. 旗標科技.

- **中文書**

  趙英傑. (2023). 超圖解 Arduino 互動設計入門 (第五版). 旗標科技.

- **網頁資料**

  TowerPro. (n.d.). SG90 Micro Servo Data Sheet. https://www.google.com/search?q=http://www.towerpro.com.tw/product/sg90-7/

- **網頁資料**

  LINE Corporation. (2024). LINE Messaging API Reference. https://developers.line.biz/en/reference/messaging-api/