國立雲林科技大學資訊工程系

大學部實務專題

Department of Computer Science & Information Engineering
National Yunlin University of Science & Technology

Senior Design

ROS2 智慧型羽球自動收集機器人：結合機器學習導航與深度學習物件偵測

ROS2 Intelligent Shuttlecock Collection Robot：Integrating Machine Learning Navigation and Deep Learning Object Detection

高啟軒、利瑋哲、唐禮直

CHI-HSUAN KAO、WEI-JHE LI、 LI-ZHI TANG

指導教授：李建緯 博士

Advisor: JIAN-WEI LI， Ph.D.

中華民國 115 年 6 月

June 2026

115

大學部

實務專題

ROS2 智慧型羽球自動收集機器人：結合機器學習導航與深度學習物件偵測

國立雲林科技大學

資訊工程系

高啟軒

利瑋哲

唐禮直

國立雲林科技大學

資訊工程系

實務專題合格認可證明

專題學生：高啟軒、利瑋哲、唐禮直

實務專題題目：ROS2 智慧型羽球自動收集機器人：

結合機器學習導航與深度學習物件偵測

ROS2 Intelligent Shuttlecock Collection Robot：

Integrating Machine Learning Navigation and Deep Learning Object Detection

經評量合格，特此證明

指導教授簽名：

摘　要

本專題系統結合電腦視覺辨識、路徑規劃、地圖建構與嵌入式車體控制，以完成羽球之自動偵測與蒐集。整體流程分成自動與手動模式兩種狀況：在自動模式狀況中，系統首先檢查是否已經產生過地圖，若地圖存在則會透過路徑規劃演算法來規劃最佳移動路徑再去進行巡邏與羽球蒐集的動作；若無存在地圖，將進行探索和移動並透過感測器與演算法進行地圖建構，同時由 YOLOv8 物件偵測模型（You Only Look Once version 8 Object Detection Model）負責羽球的物件辨識，ROS2 （Robot Operating System 2, ROS2）負責任務的協調與自走車判斷。在手動模式中，使用者可以透過遙控器控制自走車的移動，並可以選擇是否要進行地圖的建構。無論是自動或者手動，最終皆會產生或更新地圖狀況，方便提供後續路徑上的規劃與移動的判斷；而自走車底層的移動與滾輪撿球機構則由 STM32 微控制器驅動，確保自走車能夠依靠指令完成任務。此流程使系統能夠同時實現羽球的偵測、動態路徑規劃與自動蒐集羽球，展現電腦視覺、機器人中介軟體與嵌入式控制的整合應用。

關鍵字：深度學習（Deep Learning）、機器學習（Machine Learning）、卷積神經網路（Convolutional Neural Network, CNN）、物件偵測（Object Detection）、YOLOv8 物件偵測模型（You Only Look Once version 8 Object Detection Model）、同步定位與建圖（Simultaneous Localization and Mapping, SLAM）、路徑規劃演算法（Path Planning Algorithm）、Nav2 導航框架（Navigation 2 Framework, Nav2）、機器人作業系統 2（Robot Operating System 2, ROS 2）、嵌入式系統控制（Embedded System Control）、自動羽球蒐集系統（Autonomous Shuttlecock Collection System）

致　謝

本專題《ROS2 智慧型羽球自動收集機器人：結合機器學習導航與深度學習物件偵測》能順利完成，過程中承蒙許多師長、同儕與夥伴的協助與支持，使我們得以克服研究中面臨的種種挑戰。在此謹向所有曾在專題歷程中提供指導、協力與鼓勵的人士致上最深切的謝意。

首先，衷心感謝指導教授 **李建緯老師**。李老師於研究期間全力投入指導，無論是在主題確立、研究方向釐清、硬體結構設計、系統整合規劃、演算法驗證，或是實驗流程安排上，都給予耐心而詳盡的協助。在每一次的討論中，李老師皆以扎實的專業知識、縝密的邏輯與高度的洞察力，提供寶貴且具有深度的建議，讓我們在研究遇到瓶頸時能重新思考問題並找到解決方案。老師的嚴謹態度、務實精神與對學術品質的堅持，是促使本專題得以成熟與完善的重要推力，也讓我們在過程中獲得豐富的成長。

此外，誠摯感謝實驗室各位學長姐在研究期間提供的協助與指導。從硬體設備操作、電路與機構調整、感測器安裝，到程式開發、除錯經驗分享，每一份支持都讓研究得以更順利推進。學長姐不僅提供技術層面的協助，也在遇到困難時給予我們鼓勵與建議，讓團隊能在良好、積極的氛圍中專注於系統開發與效能驗證。

同時，也感謝所有在專題期間與我們討論、提供回饋的同學與夥伴。大家在過程中共同腦力激盪、分享想法，無論是對演算法表現的觀察、場域佈局的討論、測試策略的調整，或是對資料處理與模型訓練細節的建議，都讓我們能從不同角度審視問題並進一步提升系統品質。

最後，向所有在本專題推動過程中曾付出心力的師長、朋友與家庭成員，表達深深的感謝。你們的支持、理解與鼓勵，是我們能持續投入研究、堅持完成專題的重要力量。本研究之完成，凝聚了許多人的付出與心血，在此再次致上最誠摯的謝意。

目　錄

圖目錄

表目錄

第一章 研究動機與目的

1.1 研究動機

羽球是一項需要高速度、敏捷與反應力的休閒運動，受到不同年齡的人們喜愛。無論是專業球員所進行的高強度訓練，或者是一般民眾休閒的娛樂運動，羽毛球在比賽或者練習的情況下都會產生許多掉落與各處的羽球。撿拾羽球的工作常常需要讓球員或助理反覆來回，這不僅會對體能造成額外的消耗，還會中斷了訓練中的連續性，導致降低了球員的專注與練習的效率。對於專業球員而言，持續的高效率訓練將會是是提升競技水平的一項重要的因素，而頻繁的去撿球則很容易干擾訓練。對於休閒民眾來說，頻繁的彎腰與來回走動撿球也容易影響到運動的流暢體驗感，甚至可能會增加運動傷害產生的風險。

隨著現在的智慧型機器人與人工智慧技術的發展盛況，如何透過這些新興科技與運動場域進行結合，成為了一個值得前進的方向。ROS2（Robot Operating System 2）[23]是新一代的機器人作業系統，其具備分散式的系統架構、跨平台支援能力以及多模組協作的方便性，非常適合用在需要整合感測器、影像處理與運動控制等多樣的系統。而 STM32 微控制器[47]則是以低功耗、高效能與穩定的硬體控制能力為主，廣泛的應用在各種嵌入式裝置內。將 ROS2 的高層規劃能力與 STM32 微控制器的即時控制能力結合設計，就可以形塑成一個具有智慧與穩定性的系統架構，實現羽球自動蒐集的功能。

1.2 研究目的

本專題的主要目標在於開發一套能於羽球場域中自動辨識並蒐集羽球的智慧型機器人系統，藉由結合 ROS2（Robot Operating System 2）的模組化軟體架構與 STM32 微控制器的低層硬體控制，為達成此目標，專題聚焦於以下幾個核心方向：

首先，在羽球辨識方面，利用攝影機搭配影像處理與深度學習技術，對場地上的羽球進行即時偵測與定位。由於羽球具有體積小、顏色偏白、且容易與地板或背景混淆的特性，需確保在不同光線與角度下仍能穩定偵測。辨識系統不僅能判斷羽球位置，亦需輸出相對座標資訊，提供後續的路徑規劃模組使用。

其次，在環境感知與地圖建立方面，本專題將透過 ROS2 的 RViz 工具進行場地地圖的建置與可視覺化，並搭配 SLAM（Simultaneous Localization and Mapping, SLAM）或導航套件實現定位功能。透過地圖的建立與即時更新，機器人能具備環境感知能力，理解自身位置與羽球位置之間的相對關係，進而在複雜場域中進行導航與避障。RViz 的使用不僅能輔助開發過程中的系統除錯，也有助於驗證機器人在不同情境下的運作表現。

再次，在自動導航與行為決策方面，系統將結合 ROS2 的路徑規劃模組，設計能自動規劃最短且最安全路徑的演算法，使機器人能從當前位置自主移動至羽球所在位置。導航過程需考量到羽球場上的多顆羽球分布、場地邊界，如何讓機器人快速、平穩且準確地抵達目標，是研究的重要挑戰。

最後，在撿球機構與控制整合方面，STM32 微控制器將負責底層的馬達驅動、機構控制與感測器訊號回饋，確保機器人能在到達羽球位置後順利完成蒐集動作。整體系統將形成一個「高層規劃（ROS2）、低層控制（STM32）」的雙層架構，使機器人兼具智慧決策與穩定執行的能力。

第二章 文獻回顧與討論

2.1 自動化機器人與物體偵測技術

隨著機器人技術的發展，自動化收集任務逐漸應用於各種運動領域，尤其是在羽毛球等場地運動中，機器人自動收集羽球的研究逐漸成為熱門議題。傳統的自動化系統多依賴機械式撿球，但這些系統無法靈活應對複雜環境變化，且效率有限。因此，近年來許多研究開始將視覺偵測技術引入其中，透過深度學習進行羽球的即時識別與追蹤，使機器人能更有效率整理球場。

目前，深度學習方法，特別是卷積神經網路（Convolutional Neural Network，CNN）與 YOLO 系列模型，已被廣泛應用於物體偵測領域 [7]。YOLOv8 為最新一代模型，具有更快的推論速度與更高的準確率，能在多種環境下保持穩定的偵測效果[7][9]。此外，Roboflow 平台也提供強大的資料管理與標註工具，使開發者能快速建立客製化資料集並用於模型訓練[8]，進一步提升羽球偵測的準確性。

在影像處理方面，OpenCV（Open Source Computer Vision Library）函式庫仍是重要工具，其提供多種影像前處理與特徵擷取方法，能有效協助深度學習模型進行偵測前的資料準備[11]。綜合上述技術，可為自動羽球偵測提供可靠的技術基礎。

2.2 ROS2 在機器人系統中的應用

ROS2 作為下一代機器人作業系統，具有分散式架構、模組化設計、跨平台支援與高效能通訊能力，使其成為複雜自走車系統的核心基礎。ROS2（Robot Operating System 2）能整合多種感測器（如 LiDAR、深度攝影機、IMU），並提供 SLAM（Simultaneous Localization and Mapping, SLAM）、Navigation2 （ROS 2 Navigation Stack 2）等工具[3, 4]，使機器人具備定位、建圖與自主導航功能。

在地圖建構方面，相關文獻指出，透過 ROS2 以 LiDAR 與感測器資料建構地圖，可有效反映環境中的障礙物分布，有助於後續導航與避障[12]。ROS2 的模組化也使 YOLOv8 偵測結果能與導航系統整合，例如透過 TF 座標系轉換、topic 訂閱與發布，建立「偵測 → 定位 → 移動」的自動化流程。

因此，ROS2 在本專題中扮演統一平台的角色，將物件辨識、路徑規劃與移動控制完整串連，形成具備智慧與即時反應能力的自主羽球收集系統。

2.3 嵌入式控制與實時反應

嵌入式控制器是自走車底層運作的關鍵元件，負責電機驅動、感測器資料讀取、即時反應與安全性管理等任務。STM32 微控制器因其低功耗、高效能、即時性與穩定性，被廣泛應用於各類嵌入式機器人系統中。本專題使用 STM32 微控制器執行差速驅動車體的運動控制，並以 PWM 驅動左右輪馬達，結合編碼器回饋以達到平穩且準確的控制。

差速驅動模型（Differential Drive）是目前最常見的移動式機器人底盤運動模型，其透過左右輪的角速度差來實現前進、倒退與轉彎，並有完整的運動方程式與理論基礎。哥倫比亞大學的相關教材清楚說明了差速驅動的運動方程式與幾何關係，常被用於教學與機器人開發[16]。本研究即依據此模型設計 STM32 微控制器的移動控制邏輯，使機器人能確實追蹤 ROS2 導航指令。

此外，為了讓 AI 開發版（NVIDIA Jetson AGX Xavier）[48]執行深度學習模型（YOLOv8），許多開發者參考 NVIDIA 官方論壇與社群資源，完成 PyTorch 與 TorchVision 的安裝 [1, 2]。這些文獻提供 Jetson 平台與深度學習運算的實用指引，使模型能以邊緣運算的方式即時執行，提升整體系統反應速度。

第三章 車體與控制設計

3.1 車體架構設計

3.1.1 主體車架

主體車架我們經討論後使用質地相較輕盈的 2020 歐規鋁擠作為我們車體的主要車架，這樣對於馬達也能相對減少負擔，並提高整體移動效率。

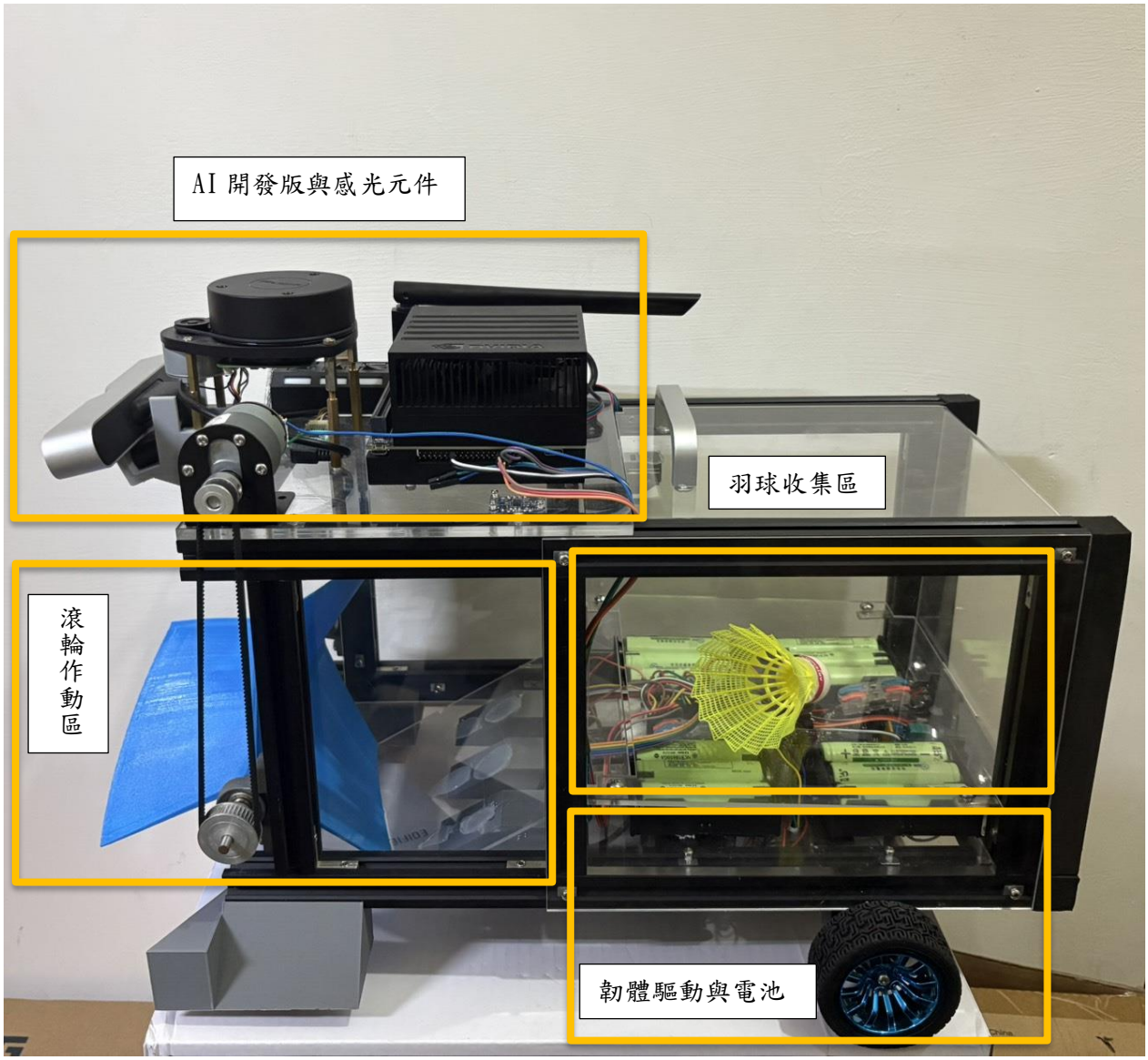


圖 1、架構分配圖

整體架構大致分為四個區域：

（一）、AI 開發版與感測元件（前半段上層區域）

作為主要程式計算機放置於上層偏中間位置以更好配置重心，其餘空間擺置光達、加速感測器、深度攝影頭等主要感測設備以及滾軸馬達。

（二）、滾輪作動區（前半段下層區域）

滾輪主要運動區域，設計擺放用以蒐集羽球的機構。

（三）、羽球收集區（後半段上層區域）

此區域空間最大，用以收集羽球使用，配有一壓克力上蓋作為開口。

（四）、韌體驅動與電池組（後半段下層區域）

放置電池、韌體驅動版、遙控模組及滾輪 PWM 控制器等動作裝置，背面放置馬達驅動器、電壓轉換模組等需要散熱的設備和編碼器馬達。

3.1.2 3D 列印與壓克力設計

我們使用 FreeCad 設計軟體設計並繪製我們所需要的各項支撐件及壓克力大小；並使用較堅硬且具承重能力的聚乳酸（Polylactic Acid, PLA）材質製作前輪支撐、羽球收集機構斜板支撐以及開關支架等支撐部件，因為其質地較硬的關係所以可以幫助我們承重，此外我們也使用此材質印製我們的遙控器外殼，另外我們使用質地較軟且具形變恢復能力的熱塑性聚氨酯（Thermoplastic polyurethane, TPU）材質製作我們滾輪，以便收集羽球使用。

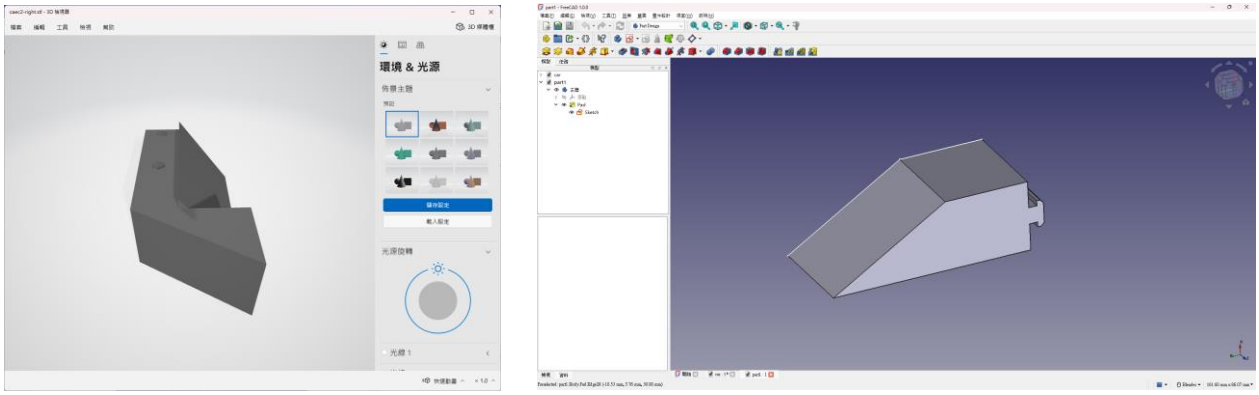


圖 2、前輪支撐與斜板支撐設計圖

由於想要達到透明車身的外觀以便觀察整體車體架構及羽球收集狀況，我們使用 2mm 的壓克力板作為側板用以完整車體以及羽球上蓋，3mm 用以作為羽球收集機構斜板以及羽球收集區的承重板，以及 5mm 的壓克力用於 AI 開發版與上層區域的承重板。

透明壓克力厚度	作用及使用區域
2mm	側板、羽球上蓋
3mm	羽球收集機構斜板、羽球收集區承重板
4mm	前半段上層區域的承重板

表 1、壓克力厚度應用表

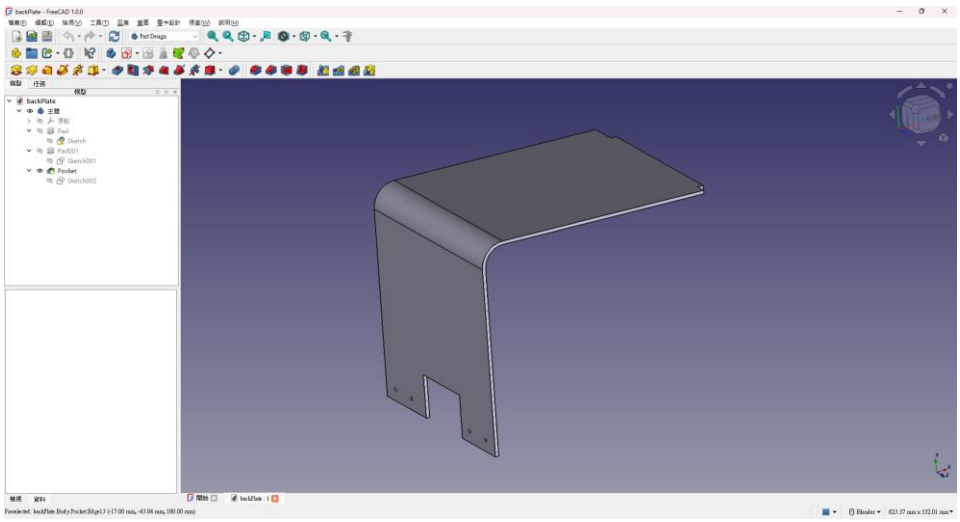


圖 3、FreeCad 上蓋設計

3.1.3 滾輪與驅動設計

由於要順利撿起羽球需要相對大的扭力，我們一樣使用與移動動力單元相同動力的編碼器馬達，使用降壓模組並設有額外的 PWM 控制模組控制其輸出；使用常見的 6mm 皮帶來帶動光軸上的齒輪組，這邊特意使用相較於馬達齒輪 2:1 的齒比來增加扭矩。

滾輪的主要架構使用光軸、軸承座、限位環以及同步輪作為主要作動機構，另外使用熱塑性聚氨酯（TPU）所印製的滾輪作為主體。

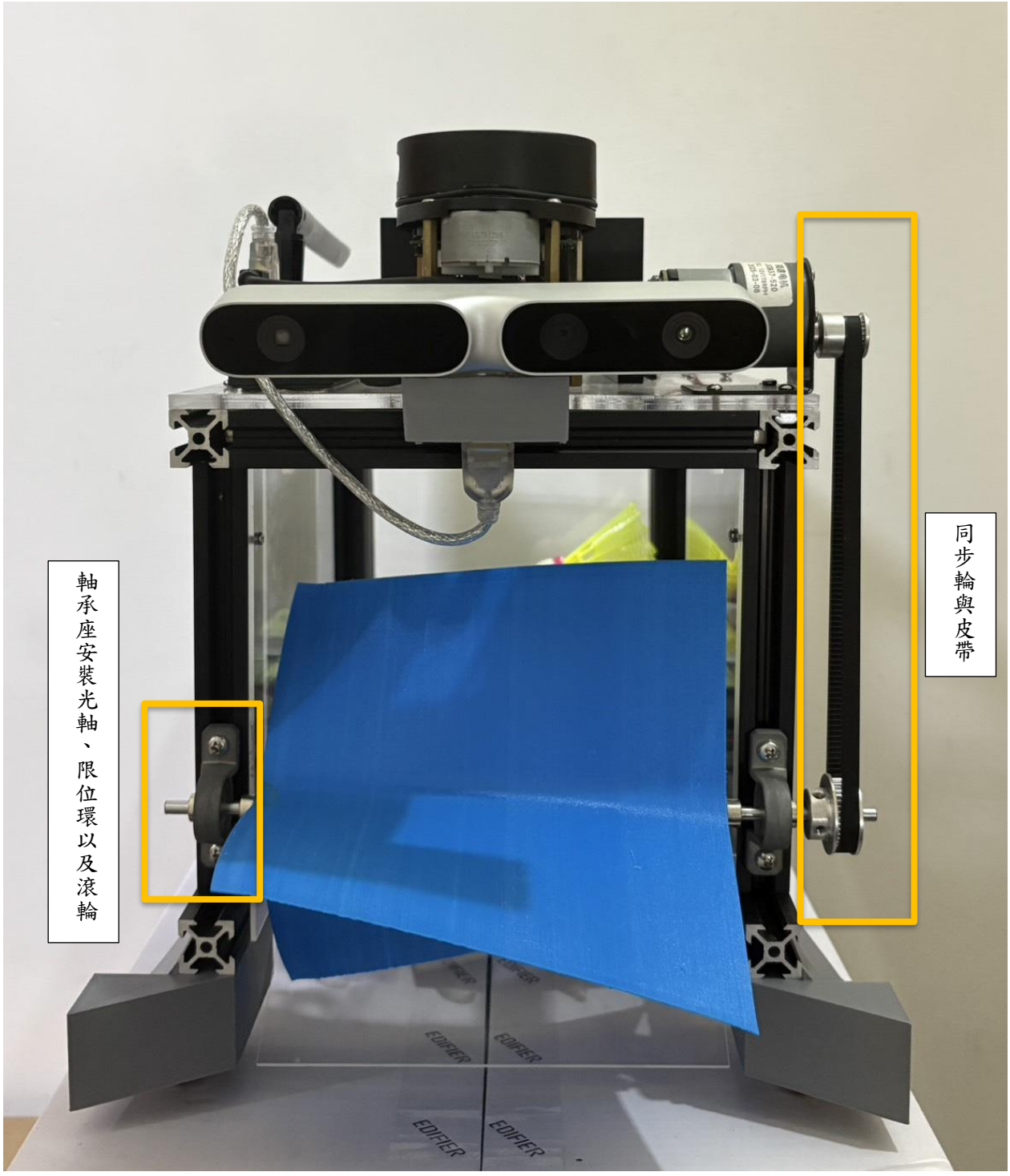


圖 4、滾輪機構實圖

3.2 電力與控制系統

3.2.1 動力單元與 STM32 微控制器控制

動力單元我們使用兩組 3 顆的 18650 電池作為電力來源，並使用電壓模組來適當調整輸出及穩壓，並且分配給兩顆移動用的編碼器馬達，以及給予 STM32 微控制器供電。

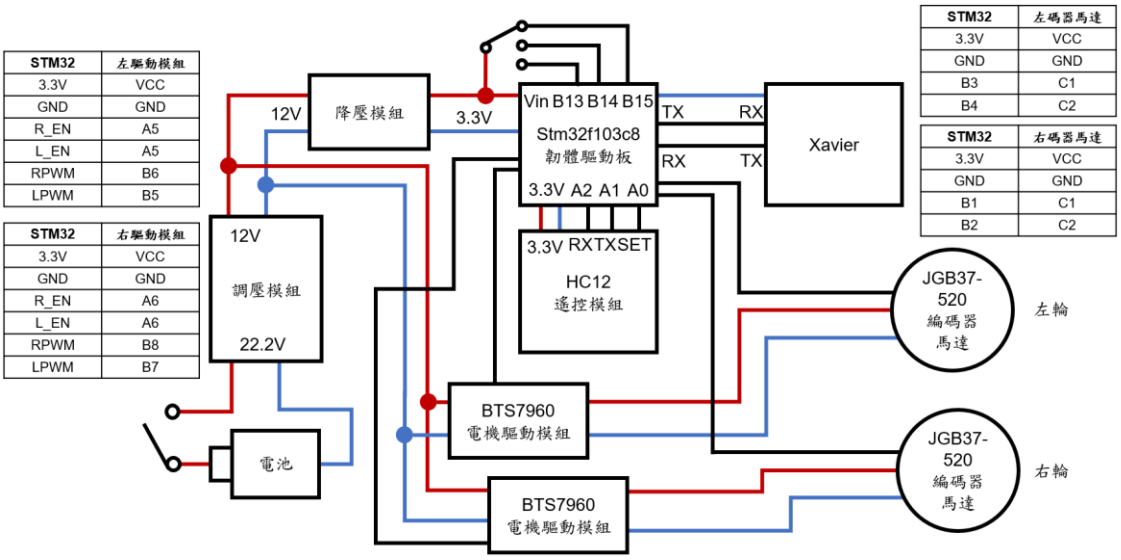


圖 5、ROS2 自主移動平台之底層運動控制電路架構

我們應用「微算機原理及應用實習」課程所教導的 STM32 微控制器編碼及控制，用於遙控模式、編碼器馬達控制以及與 AI 開發版溝通，接收其移動控制指令。

3.2.2 AI 開發版模組與運算分工

AI 開發版主要依靠兩組 3 顆 18650 電池作為電力來源，並藉由電壓模組來適當調整輸出及穩壓，同時分配給滾輪馬達。相關感測元件如深度攝影頭、光達等元件直接連接 AI 開發版的接孔進行供電。

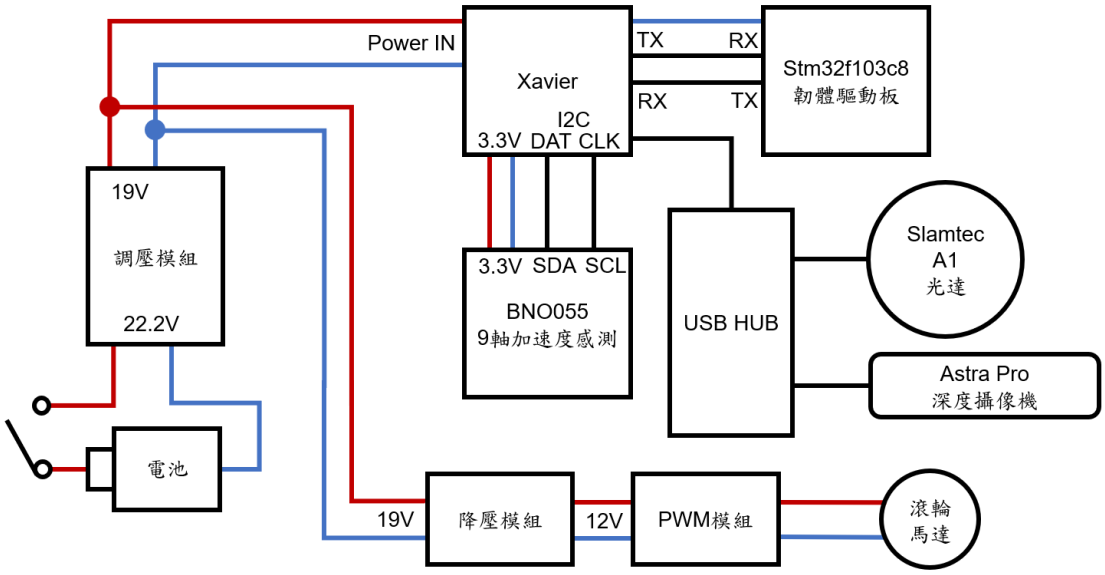


圖 6、ROS2 自主移動平台之感測器、控制器與電源供應架構

我們應用「ROS 機器人作業系統」課程所教導的 ROS 運作原理和撰寫方法，負責機器學習、深度學習等主要運算及物件辨識，並且使用 SLAM 等 ROS 工具來實現地圖及導航等功能。

3.3 遙控器設計

3.3.1 外型設計

於網路上找尋許多樣本後，我們決定自主設計遙控器外殼，同時計算容積以確保裝得下所有的控制元件，並且為了方便日後維護及拆裝我們使用螺絲和熱熔羅母作為鎖固方法。

基於極小的空間，我們需要合理配置每樣元件都放得進去並分配重量避免握持時的重心不穩，因為我們除了將體積較大的主控板與搖桿模組放於前方位置外，其餘元件放置後半段以降低整體重心，保持握持最好位置。

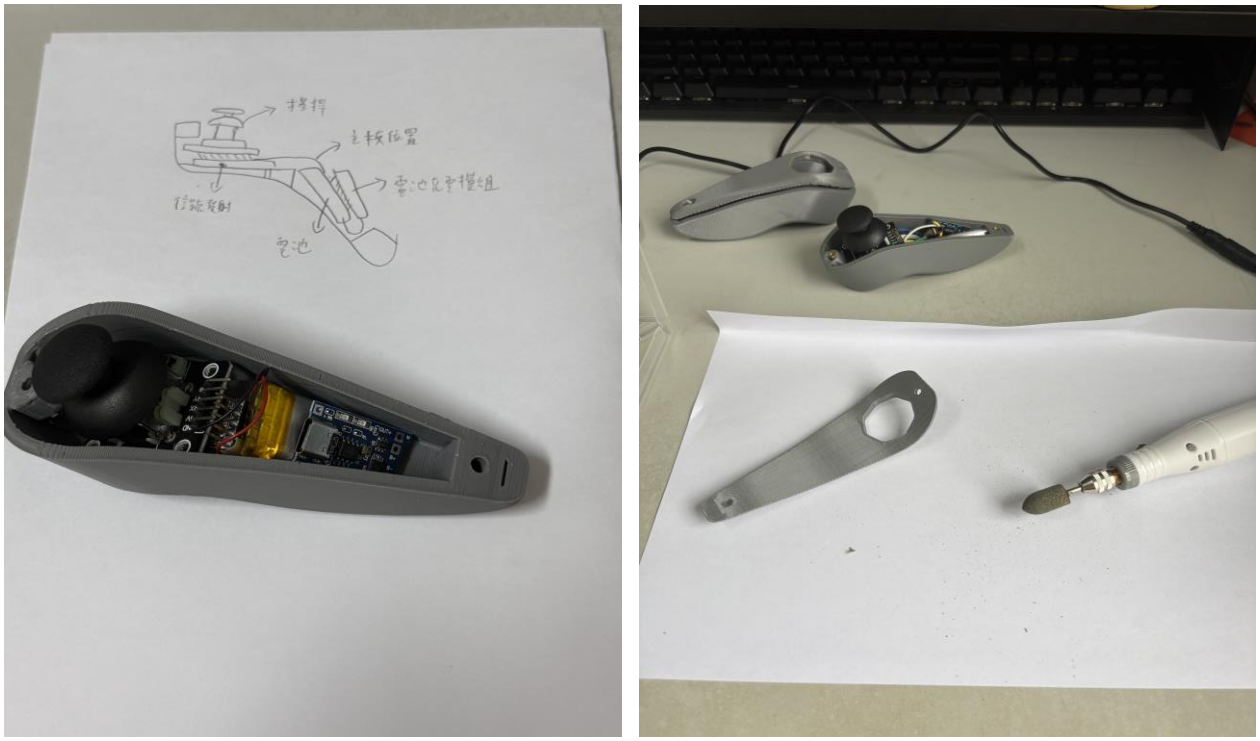


圖 7、擺放草圖與打磨外殼

3.3.2 電路設計

為了能夠支撐足夠長的遙控時間，我們使用容量 400mA 的聚合物鋰電池，並搭配專用充放電模組，以手工銲接方法使用杜邦線將每個元件連結起來：搖桿模組、主板、HC12 遙控模組、專用充放電模組、鋰電池以及總開關。

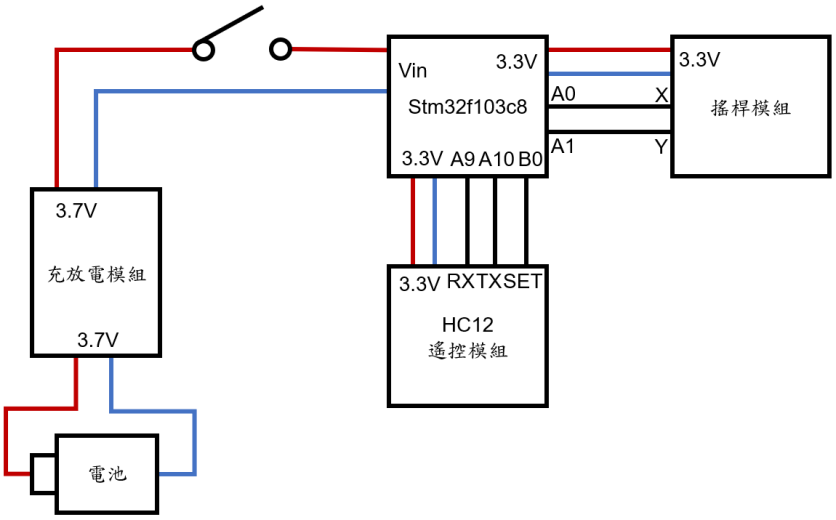


圖 8、遙控器電路圖

第四章 軟體架構與程式設計

4.1 系統流程設計

4.1.1 系統流程圖

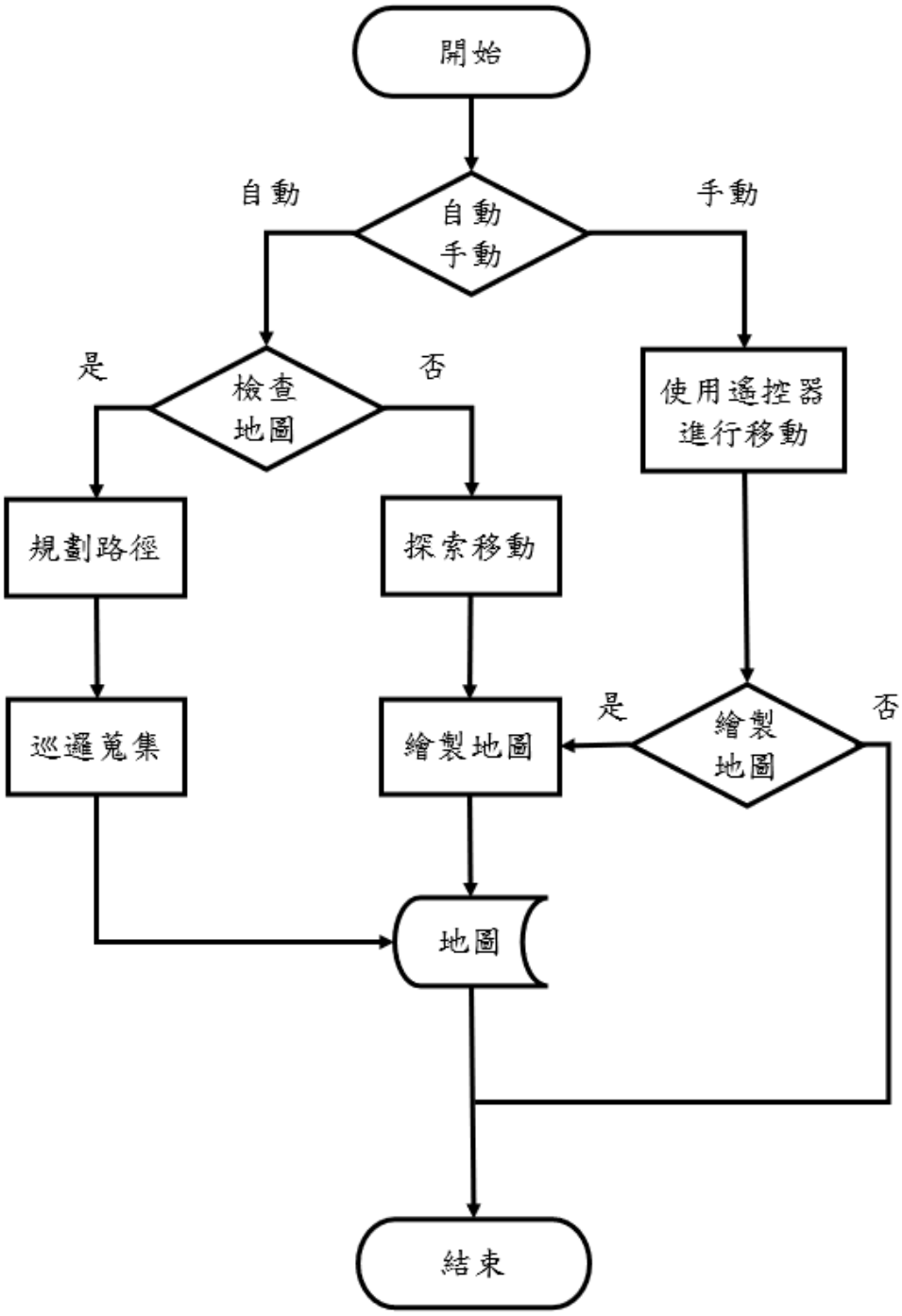


圖 9、系統流程圖

4.1.2 STM32 韌體控制流程圖

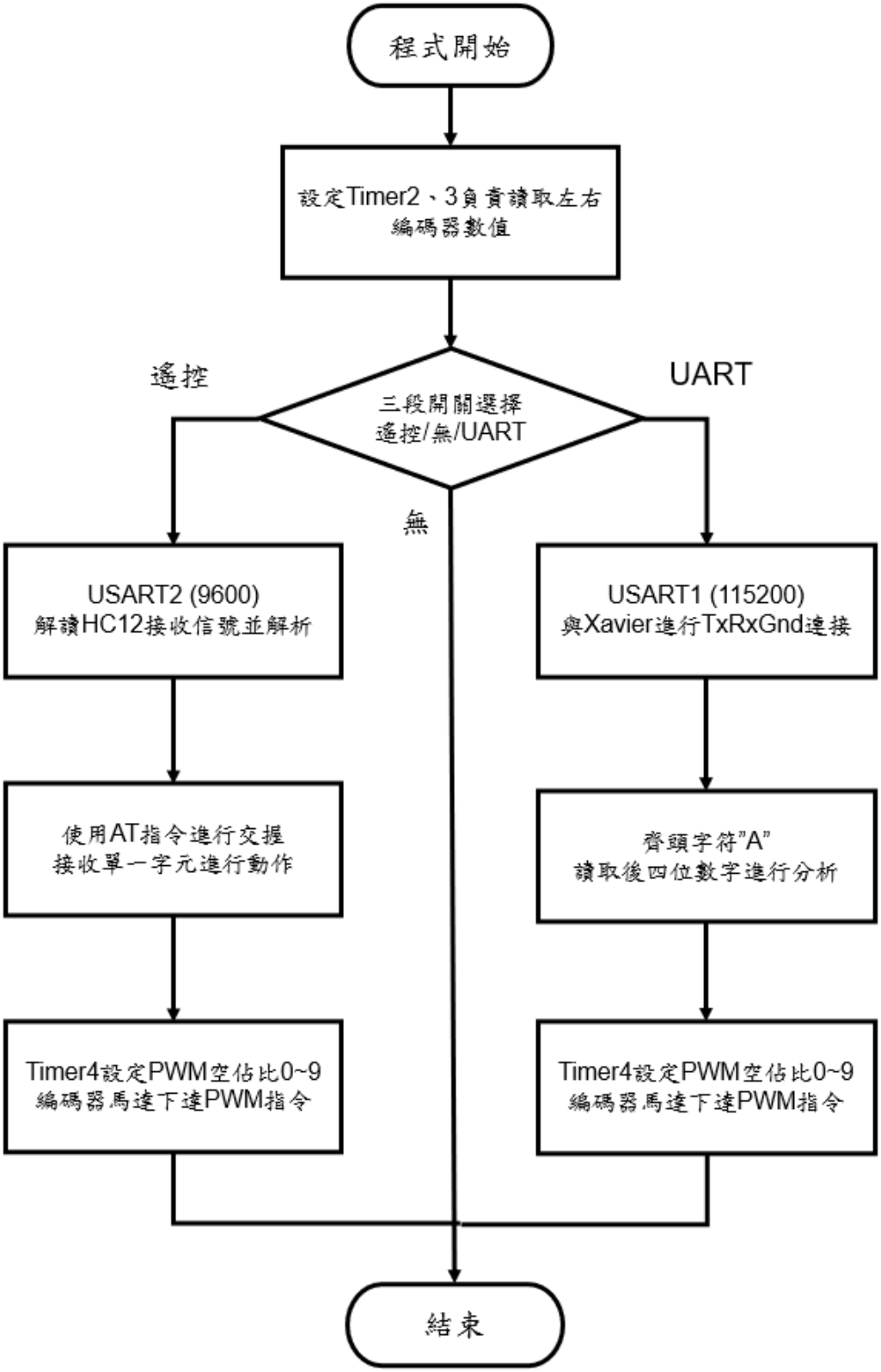


圖 10、STM32 韌體控制流程圖

4.1.3 AI 開發版運算及檔案控制流程圖

我們於電源開啟後，使用服務檔的功能，預先將光達以及深度攝像機的驅動程式，這樣一來我們可以省去開啟步驟，其次我們將檔案以 launch 檔的形式整理，這樣可以一個指令開啟所有的 node，大致分為 auto.launch.py、slam_mapping.launch.py 以及 nav_patrol.launch.py 以 auto.launch.py 作為主要指令，再判斷地圖是否存在，分別啟用剩餘兩個 launch 檔案。

檔案名稱	作用和作動方法
Yolo_mark	啟用物件辨識，在/map 中標記物體並發布位置
Serial_out	訂閱/robot-control2，為輸出加上齊頭字元，於 0.5 秒內多次輸出指令，並於結尾發出停止信號
Yolo_patrol	訂閱光達資料，針對自走車前方範圍偵測障礙，輸出移動指令，並從地圖 Frontier 未記錄區域，最後輸出移動位置及標記
Nav_patrol	等待/map 資訊，使用機器學習來建立巡邏點，啟用 nav 依序巡邏，遇到羽球標記則切換目標

表 2、檔案作用表

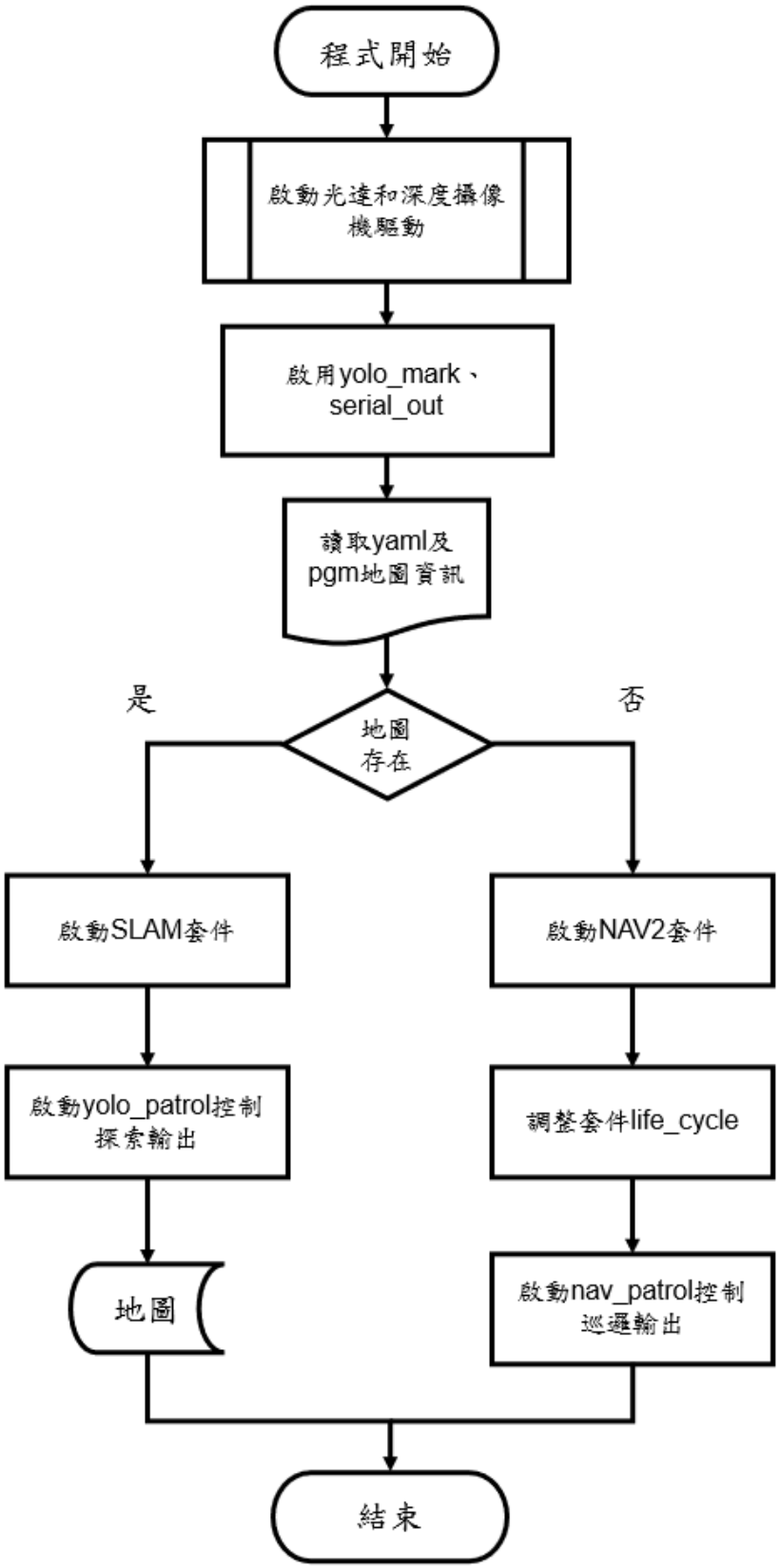


圖 11、AI 開發版（NVIDIA Jetson AGX Xavier）運算及檔案控制流程圖

4.2 STM32 韌體設計

4.2.1 編碼器馬達驅動

（一）、系統初始化

在系統啟動時，先進行 UART 設定與 GPIO 初始化，確保通訊與馬達啟動安全：

```
char* setCmd = "AT+C010\r\n";
HAL_UART_Transmit(&huart2, (uint8_t*)setCmd, strlen(setCmd), HAL_MAX_DELAY);
HAL_Delay(100);

HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);
HAL_Delay(50);
//-----------------------------------------------------
uint16_t l=0,r=0;


uint16_t CLL[3][3] = {
    {  65,   100,   100},
    {  0,    0,    100},
    {  0,    0,    0}
};
uint16_t CRL[3][3] = {
    {  100,   100,   65},
    {  100,   0,    0},
    {  0,    0,    0}
};
uint16_t CLR[3][3] = {
    {  0,    0,    0},
    {  100,   0,    0},
    {  65,   100,   100}
};
uint16_t CRR[3][3] = {
    {  0,    0,    0},
    {  0,    0,    100},
    {  100,   100,   65}
};
uint16_t c_l[]={0,0,0,0,0,0,30,47,65,82,100};
uint16_t c_r[]={100,82,65,47,30,0,0,0,0,0,0};


//
```

圖 12、初始化左右輪 PWM 對照表程式碼

```
void setPWM(uint8_t selector, uint8_t level, uint32_t ch_on, uint32_t ch_off) {
    int l = 0;
    __HAL_TIM_SET_COMPARE(&htim4, ch_off, 0);  // ??????

    switch(level) {
        case '0': l = 0; break;
        case '1': l = 20; break;
        case '2': l = 30; break;
        case '3': l = 40; break;
        case '4': l = 50; break;
        case '5': l = 60; break;
        case '6': l = 70; break;
        case '7': l = 80; break;
        case '8': l = 90; break;
        case '9': l = 100; break;
    }

    __HAL_TIM_SET_COMPARE(&htim4, ch_on, l);
}
```

圖 13、字元對應速度程式碼

CLL，CRL，CLR，CRRCLL， CRL， CLR， CRRCLL，CRL，CLR，CRR：左右輪不同模式的正反轉 PWM 查表

cl，crc_l， c_rcl，cr：速度等級對應的 PWM 占空比曲線

此初始化確保控制程式可以透過矩陣索引快速設定不同方向與速度：

$$PWM_{L,f} = CLL[m][v]， \quad PWM_{L,r} = CLR[m][v]$$

$$PWM_{R,f} = CRL[m][v]， \quad PWM_{R,r} = CRR[m][v]$$

（二）、PWM 控制與馬達模型

PWM（Pulse Width Modulation）可用於調節直流馬達的有效電壓：

$$Vmot = D \cdot Vcc， \quad D = \frac{t_{on}}{T}， \quad D \in [0，1]$$

其中：

- Vmot：馬達端電壓
- Vcc：供電電壓
- D：PWM 佔空比
- t_{on}：高電平持續時間
- T：PWM 週期

對應程式實作：__HAL_TIM_SET_COMPARE（&htim4， ch_on， level）；

直流馬達動力方程式[19]:

$$J\frac{d\omega(t)}{dt} + b\,\omega(t) = K_t\frac{V_{mot}(t) - K_e\,\omega(t)}{R}$$

其中 J 為轉子慣量，b 為黏滯摩擦係數，Kt 與 Ke 分別為力矩常數與反電動勢常數，R 為馬達內阻，$\omega(t)$為轉速。

（三）、編碼器回饋:

編碼器計算角位移與角速度[20]：

$$\theta(t) = \frac{2\pi}{N_{ppr}}\sum N_{enc}(t) \text{ ，} \omega(t) = \frac{d\theta(t)}{dt}$$

其中 Nppr為編碼器每轉脈衝數，Nenc(t)為累積編碼器脈衝數。速度誤差定義為目標轉速與實際轉速之差：

程式實務:

- update_encoders（）：每循環讀取 TIM 計數器差值，累加到全域編碼器變數。
- HAL_TIM_PeriodElapsedCallback（）：處理計數器溢位，防止速度或位移計算錯誤。
- debug_encoders（）：將累積脈衝透過 UART 輸出，方便調試。

```
void update_encoders() {
    uint16_t now1 = __HAL_TIM_GET_COUNTER(&htim2);
    uint16_t now2 = __HAL_TIM_GET_COUNTER(&htim3);

    int16_t delta1 = (int16_t)(now1 - last_cnt1);
    int16_t delta2 = (int16_t)(now2 - last_cnt2);

    encoder1 += delta1;
    encoder2 += delta2;

    last_cnt1 = now1;
    last_cnt2 = now2;
}
void debug_encoders() {
    char buffer[128];
    sprintf(buffer, "Enc1 = %" PRId64 ", Enc2 = %" PRId64 "\r\n", encoder1, encoder2);
    HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);
}
```

圖 14、編碼器讀取與更新程式碼

（四）、閉迴路控制

速度誤差[21]：

$$e(t) = \omega_{target} - \omega_{measured}$$

- ω_{target}：希望馬達達到的速度（目標速度）
- $\omega_{measured}$：實際馬達的速度，由編碼器回饋計算
- $e(t)$：當前速度誤差，越大表示馬達與目標速度偏差越大

PID 控制 PWM[21]：

$$D(t) = K_p\, e(t) + K_i \int e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$$

P 項（Proportional）：$K_p e(t)$

- 根據當前誤差直接計算控制量
- 誤差越大，PWM 調整越多 → 快速反應

（Integral）：$Ki \int e(\tau) d\tau$

- 累積誤差，消除長期偏差
- 例如速度總是比目標慢，I 項會慢慢把 PWM 提高

D 項（Derivative）：$K_d \frac{de(t)}{dt}$

- 預測誤差變化趨勢
- 避免過衝，讓系統更平滑

程式實務:

```
D = Kp*e + Ki*sum_e + Kd*（e - e_last）;
__HAL_TIM_SET_COMPARE（&htim4， ch_on， D）;
```

4.2.2 模式切換邏輯

我們使用一顆實體的三段開關還操作模式選擇，並且直接連接 STM32 主板，使用軟體設計內建上拉電阻，以讀取高低信號並切換模式。

模式	針腳號碼	電位條件
1	B13	HIGH
2	B14	HIGH
3	B15	HIGH
0	其他	無

表 3、模式切換邏輯表

模式介紹

（一）、模式 1

為空信號模式，默認等待按鈕切換，作為檢修或待命模式。

（二）、模式 2

手動模式，MCU 透過 UART2（9600）來接收來自 HC12 遙控模組的字元指令，並依照對應指令對馬達下達移動指令。

從指令中取得兩個索引 cc1 和 cc2，用於選擇預先定義的馬達 PWM 表格，控制左右輪的運動軌跡。

字元	指令
A	向左
D	向右
W	往前
S	往後
P	停止

表 4、手動模式指令配對表

控制原理

馬達 PWM 值來自四個矩陣

$$\mathrm{CLL[i][j]}，\mathrm{CRL[i][j]}，\mathrm{CLR[i][j]}，\mathrm{CRR[i][j]}$$

曲線控制

選取表格

$$\mathrm{PWMchannel} = \mathrm{PWM_table[i][j]}$$

左右輪實際速度：

$$v_{left} = \frac{PWM_{CLL[i][j]}}{100} \cdot v_{max}，v_{right} = \frac{PWM_{CRR[i][j]}}{100} \cdot v_{max}$$

這種方式可以形成不同的預設運動曲線，使機器人在行進時達到平滑運動效果。

程式實務

```
if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_14)){//mode2----------------------------------------ca
  memset(rxBuf, 0, sizeof(rxBuf));


  CRL_= CRL[ccl][cc2];
  CLR_= CLR[ccl][cc2];
  CLL_= CLL[ccl][cc2];
  CRR_= CRR[ccl][cc2];
  __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_1,CRR[ccl][cc2]);
  __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_2,CRL[ccl][cc2]);
  __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_3,CLR[ccl][cc2]);
  __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_4,CLL[ccl][cc2]);

  HAL_Delay(1);

}
```

圖 15、模式二程式碼

（三）、模式 3

自動模式，STM32 微控制器連接 AI 開發版的 RxTx 進行傳輸，使用 UART1 傳送編碼器資料，同時接收五位元的即時控制指令。

其中位元 0 為字元 A 作為齊頭使用

位元	數字	對象	指令
1	0、1	右輪	0:向前、1:向後
2	0~9		轉換 PWM 空佔比
3	0、1	左輪	0:向前、1:向後
4	0~9		轉換 PWM 空佔比

表 5、UART 指令配對表

控制原理

PWM 設定公式

$$PWMchannel = Level \cdot \frac{PWMmax}{100}$$

方向選擇

Dir = 0，正轉;1，反轉

馬達速度控制

$$v_{left} = \frac{Level_L}{9} \cdot v_{max}，v_{right} = \frac{Level_R}{9} \cdot v_{max}$$

這種模式可以實現即時操作或遠程調整行進軌跡，擁有穩定、快速的特性。

程式實務

```
if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_15)){//mode3----------------------------------------ro

  char txBuf[64];
  memset(txBuf, 0, sizeof(txBuf));
  //sprintf(txBuf, "E1:%" PRIu64 " E2:%" PRIu64 "\r\n", encoder1, encoder2);
  //HAL_UART_Transmit(&huart1, (uint8_t*)txBuf, strlen(txBuf), 100);


  memset(rxBuf, 0, sizeof(rxBuf));
  if (HAL_UART_Receive(&huart1, rxBuf, sizeof(rxBuf)-1, 1000) == HAL_OK) {

    if (rxBuf[0] == '0')
        setPWM(rxBuf[0], rxBuf[1], TIM_CHANNEL_1, TIM_CHANNEL_2);
    else if (rxBuf[0] == '1')
        setPWM(rxBuf[0], rxBuf[1], TIM_CHANNEL_2, TIM_CHANNEL_1);

    if (rxBuf[2] == '0')
        setPWM(rxBuf[2], rxBuf[3], TIM_CHANNEL_3, TIM_CHANNEL_4);
    else if (rxBuf[2] == '1')
        setPWM(rxBuf[2], rxBuf[3], TIM_CHANNEL_4, TIM_CHANNEL_3);
  }

  HAL_Delay(1);


}
```

圖 16、模式三程式碼

4.2.3 遙控器邏輯

系統分工架構如下：

1. 搖桿：提供前後、左右方向控制訊號。
2. ADC（模數轉換器）：將搖桿的類比電壓訊號轉換為數位值。
3. DMA（直接記憶體存取）：自動將 ADC 數值存入記憶體，減少 CPU 輪詢負擔。
4. UART：將計算後的控制訊號傳送給外部裝置（自走車）。
5. GPIO：LED 顯示模組狀態或運作指示。

搖桿數值與前後左右映射：

搖桿輸出電壓範圍為 0~3.3V，STM32 ADC 為 12-bit，對應數值範圍為 0~4095。

假設 ADC 值為 Vx，Vy，對應左右與前後控制信號 RL，FB：

映射公式：

$$FB = \frac{4095 - V_y}{4040} \times 2$$

$$RL = 2 - \frac{4095 - V_x}{4040} \times 2$$

FB 用於前後速度或方向

- 搖桿向前推時，Vy 減小 → FB 增大 → 前進
- 搖桿向後拉時，Vy 增大 → FB 減小 → 後退

RL 用於左右轉向

- 搖桿向右推，Vx 增大 → RL 減小 → 右轉
- 搖桿向左推，Vx 減小 → RL 增大 → 左轉

UART 控制訊號

計算完成後的 FB 與 RL 值會被格式化成字串：

```
sprintf(msg, "%u,%u\n", fb, rl);
HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
```

圖 17、FBRL 程式碼

我們使用 HC12 遙控發射模組，用以與自走車上的另一塊模組進行連線，並且使用 AT 指令進行交握，最後使用普通模式傳送指令字元。

HC-12 與 STM32 的通信模式

1. 普通模式（透傳）
   - STM32 發送資料 → HC-12 無線發送到另一 HC-12 → 接收端 MCU 讀 UART
   - 適合傳送遙控指令，例如 "FB，RL\n"
2. AT 模式（指令設定）

- SET 腳位低 → 模組進入 AT 模式
- 可以透過 UART 發送指令設定模組參數
- 設定完成 → SET 腳位高 → 回到透傳模式

```
uint8_t txBuf[] = "AT\r\n";
uint8_t rxBuf[10] = {0};

HAL_UART_Transmit(&huart1, txBuf, sizeof(txBuf)-1, 100);
HAL_UART_Receive(&huart1, rxBuf, sizeof(rxBuf), 200);

if (rxBuf[0] == 'O' && rxBuf[1] == 'K') {
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);  // ????
    HAL_Delay(200);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_Delay(200);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);  // ????
    HAL_Delay(200);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
    HAL_Delay(200);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);



} else {
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);    // ????
}
//--------------------

  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_Delay(50);

  char* setCmd = "AT+C010\r\n";
  HAL_UART_Transmit(&huart1, (uint8_t*)setCmd, strlen(setCmd), HAL_MAX_DELAY);
  HAL_Delay(100);

  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
  HAL_Delay(50);
```

圖 18、AT 切換程式碼

LED / GPIO 指示

- GPIOC PIN13：LED 指示 UART 模組初始化與運作狀態。
- GPIOB PIN0：控制模組的上電或重置。
- 透過 LED 閃爍可以提示模組是否成功初始化。

4.3 AI 開發版程式設計

4.3.1 探索模式邏輯

在使用者或是 auto_mode.launch.py 啟動 slam_mapping.launch.py 後，會依序啟動

檔案或指令	工作原理
自走車模型發布	發布客製 xarco 模型
RVIZ	用於顯示地圖與各個 topic 資訊
關節發布	為模型發布第一次的關節位置和 TF
編碼器控制關節檔案	讀取編碼器數值移動已發佈的模型
EKF 檔案	IMU 進行位移校正
移動指令檔案 yolo_patrol_node.py	計算地圖探索狀態，並發布下個位移點以及控制指令[23]
地圖儲存檔案	每十秒儲存已繪製地圖，在巡邏完節點關閉時會再儲存一次並確認地圖儲存狀態

表 6、探索模工作對表

主要移動指令檔案運作邏輯

1. 系統功能簡介

這份程式 （YoloPatrolNode） 的目的是讓移動式機器人在室內或羽球場環境中，自動進行 探索與巡邏，具備以下特點：

- 使用 LiDAR（雷射測距儀） 判斷前方是否暢通
- 結合 SLAM 地圖 （Occupancy Grid） 來產生 frontier 目標點[22]
- 若 LiDAR 偵測到前方有障礙物，會嘗試更換目標
- 若所有方法皆無可行目標，則結束探索 （mapping finished）
- 使用 TF 獲取機器人在地圖中的位置與朝向
- 發布 cmd_vel 給機器人底盤控制，並同步轉換成串列通訊格式給 STM32
- 在 RViz 顯示當前目標點 （Marker）

2. 程式碼邏輯流程

整體動作邏輯如下：

（1） 初始化

- 建立 ROS2 Node
- 訂閱：

- /map → 獲取當前地圖（Occupancy Grid）
- /scan → 獲取 LiDAR 雷射數據

- 發布：
  - /cmd_vel → 控制機器人移動
  - /robot_control2 → 串列控制訊息（給 STM32）
  - /goal_marker → RViz 目標點顯示

（2）主循環（explore_loop）

- 檢查是否有新的地圖與雷射數據
- 從 TF 查詢機器人位置（rx，ry，yaw）
- 若當前目標達成或尚未設定 → 選擇新的目標點：

1. LiDAR 前方自由空間檢查 → 嘗試在前方 path_distance 找一個乾淨區域
2. 左右方向嘗試 → 若正前方有障礙，嘗試±90°
3. Frontier（邊界）探索 → 從地圖找未知區域，並確認 LiDAR 前方沒有障礙
4. 無可行目標 → 宣告 mapping finished，結束任務

（3）路徑檢查（LiDAR 判斷）

程式使用 LiDAR 的角度範圍來檢查前方是否有障礙物。

公式：

- 前方角度範圍
- $\theta = \arctan\left(\frac{W}{2D}\right)$ [24]
  - W：路徑寬度（機器人寬度）
  - D：要檢查的距離
- 檢查條件

$$r(\theta) > D \quad \forall\theta \in [-\theta，+\theta]$$

若在此角度範圍內所有 LiDAR 量測值$r(\theta)$ 都大於距離 D，代表前方乾淨，可以前進。[24]

（4） Frontier 探索邏輯

在地圖（Occupancy Grid）中尋找未知區域（-1），若其鄰近有已知空間（0），則判定為 Frontier 點。[22]

條件：

- 若（x， y）為未知格點（-1），且其鄰居中存在已知格點（0），則[22]：

$$(x，y) \in \text{Frontier}$$

Frontier 會被轉換到地圖座標系，用來當作候選目標點。

（5） 移動控制

控制方式分為旋轉與直行：

■ 計算目標角度[25]：

$$\alpha = \arctan 2\left(y_{goal} - y_{robot}，x_{goal} - x_{robot}\right)$$

■ 計算角度誤差：

$$e_\theta = \alpha - \theta_{robot}$$

■ 若$|e_\theta| > 0.3 \rightarrow$ 原地旋轉

■ 否則 → 前進

（6） 串列控制轉換

為了讓 STM32 微控制器控制馬達，將速度轉換成左右輪指令[26]：

■ 左輪：

$$v_L = v - \frac{\omega B}{2}$$

■ 右輪：

$$v_R = v + \frac{\omega B}{2}$$

■ v：線速度

- ω：角速度
- B：輪距

再轉換成指令格式，例如：

11 → 向後 1 檔

02 → 前進 2 檔

4.3.2 巡邏模式邏輯

在使用者或是 auto_mode.launch.py 啟動 nav2_patrol.launch.py 後，會依序啟動

檔案或指令	工作原理
Navegation2	導航及建立地圖的檔案包
RViz	用於顯示地圖與各個 topic 資訊
lifecycle_manager	管理並分別啟用 Nav2 各項工具
自走車模型發布	發布客製 xarco 模型
關節發布	為模型發布第一次的關節位置和 TF
編碼器控制關節檔案	讀取編碼器數值移動已發佈的模型
EKF 檔案	IMU 進行位移校正
init_pointer	預設機器人方向及位置，以便定位安排任務
移動指令檔案 nav2_patrol.py	計算地圖探索狀態，並發布下個位移點以及控制指令

表 7、巡邏模工作對表

主要移動指令檔案運作邏輯

1. 系統簡介

自動化巡邏與任務中斷系統，讓自走車能夠依照機器學習生成路徑進行巡邏，並在巡邏過程中根據外部事件（羽球目標點/badminton_point）動態中斷原本任務，轉而前往新的目標點完成收集，之後再返回繼續巡邏。

2. 設計目標如下：

1. 高覆蓋率巡邏：利用地圖資訊自動生成巡邏點，並優化路徑順序以減少巡邏冗餘。
2. 即時反應能力：在巡邏過程中能即時偵測環境目標（羽球），中斷巡邏完成任務後自動返回原路徑。
3. 演算法最佳化：結合 K-means 聚類（K-Means Clustering）聚類與基因演算法（GA）進行巡邏點生成與路徑排序。
4. 精確運動控制：透過差速驅動運動模型、速度指令轉換與里程計融合，確保導航精度。

3. 架構與動作邏輯

程式碼核心為 PatrolPathMarkerNode ROS2 節點，功能流程如下：

1. 初始化與參數設定
   - 宣告與取得參數，例如巡邏點數量 num_points、角點半徑 corner_radius、巡邏點閃爍次數 blink_times。
   - 建立 Publisher/Subscriber，包含地圖（/map）、巡邏 marker（patrol_path_marker）與羽球目標（/badminton_point）。
2. 地圖回調（Map Callback）
   - 收到 OccupancyGrid 後進行 可行走區域分析[27]：

$$\text{safe\_area} = \neg(\text{obstacle\_mask} \oplus B)$$

其中 $\oplus B$ 表示使用結構元素 B 進行二值膨脹，確保機器人與障礙物保持安全距離。

   - 角點偵測：利用 Shi-Tomasi goodFeaturesToTrack 找出地圖角落，作為巡邏關鍵點。
   - K-means 聚類：對可行走區域隨機取樣，使用 K-means 生成候選巡邏點。
3. 巡邏點合併與路徑優化
   - 合併角點與 K-means 生成點，形成完整巡邏點集合。
   - 基因演算法（GA）用於路徑順序最佳化：
     - 適應度函數：

$$f(P) = \frac{1}{\sum_{i=1}^{N-1} d(p_i, p_{i+1}) + \lambda \sum_{i=1}^{N-2} \Delta\theta_i^2}$$

     - $d(p_i, p_{i+1})$：點間歐式距離
     - $\Delta\theta_i$：連續點間路徑轉角
     - λ：角度懲罰係數，若啟用強制 90 度轉向則增大 λ
     - 基因演算法（GA）流程：選擇（Selection）→ 交配（Crossover）→ 突變（Mutation）→ 收斂判斷

4. Marker 與閃爍動畫
   - 透過 visualization_msgs/Marker 將巡邏點與路徑在 RViz 顯示，並透過閃爍效果標示巡邏順序。

5. 導航控制（Nav2 Integration）
   - 使用 BasicNavigator 發送巡邏點目標，實時監控回饋，計算距離：

$$d = \sqrt{(x_{target} - x_{robot})^2 + (y_{target} - y_{robot})^2}$$

   - 判斷到達後更新巡邏點索引，若接收到羽球目標中斷巡邏，完成任務後返回原點繼續巡邏。

4. 核心演算法與數學推導

（一）、K-means 聚類

（1） 方法原理

K-means 是一種經典非監督式學習（Unsupervised Learning）演算法，用於將資料分群，使群內樣本相似、群間差異大。目標函數為最小化群內平方距離[28]：

$$J = \sum_{i=1}^{k} \sum_{x_j \in C_i} |x_j - \mu_i|^2$$

- C_i：第 i 群
- μ_i：第 i 群的中心
- x_j：安全區域內的像素點

演算法流程：

1. 資料準備
   - 從 OccupancyGrid 地圖中提取安全區域（free cells），作為 K-means 的輸入特徵空間。
   - 將地圖中的每個像素座標 (x_{pixel}, y_{pixel})視為二維特徵向量。
2. 初始化中心點
   - 隨機選擇 k個像素作為初始群中心μi。
3. 資料分配
   - 將每個像素點分配到距離最近的群：

$$\text{assign } x_j \text{ to cluster } i = \text{argimin}|x_j - \mu_i|^2$$

4. 更新群中心
   - 計算每個群的新中心點為群內所有點的平均值[28]：

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

5. 重複分配與更新
   - 直到群中心收斂或最大迭代次數達成。

（2） 在巡邏系統的應用

- 目的：將整個可行走區域分成均勻的巡邏區域，避免巡邏點過於集中或疏散。
- 輸出：K 個群中心作為候選巡邏點。
- 工程優化：
  - 若可行區域過大，隨機抽樣 5000 點加速 K-means 訓練。
  - 可自動計算 k 值，依據安全面積大小與單點負載面積（area_per_point）[29]：

$$k = \text{clip}\left(\left\lfloor \frac{\text{safe\_area}}{\text{area\_per\_point}} \right\rfloor, \text{min\_points}, \text{max\_points}\right)$$

（二）、基因演算法（Genetic Algorithm，GA）

（1）方法原理

基因演算法（GA）屬於進化式學習（Evolutionary Optimization），透過模擬自然選擇過程優化解的品質。

- 編碼（Chromosome Encoding）：每條染色體為巡邏點序列 $P = (p_1, p_2, \ldots, p_N)$
- 適應度函數（Fitness Function）：綜合路徑長度與轉角限制[30]：

$$f(P) = \frac{1}{\sum_{i=1}^{N-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} + \lambda \sum_{i=1}^{N-2} \Delta\theta_i^2}$$

其中$\Delta\theta_i$為連續向量間角度變化[30]：

$$\Delta\theta_i = \arccos \frac{(p_{i+1} - p_i) \cdot (p_{i+2} - p_{i+1})}{|p_{i+1} - p_i||p_{i+2} - p_{i+1}|}$$

- 選擇（Selection）：挑選適應度較高的染色體進入交配池。
- 交配（Crossover）：以順序交叉（Order Crossover）生成子代，保持巡邏點唯一性。
- 突變（Mutation）：隨機交換兩個巡邏點位置，增加探索能力。
- 迭代（Iteration）：重複選擇→交配→突變直到收斂或達最大世代。

（2）在巡邏系統的應用

- 目的：優化巡邏點順序，縮短巡邏路徑並降低急轉彎次數，提升巡邏效率與穩定性。
- 結果：系統生成的巡邏序列經過基因演算法（GA）優化後，不僅覆蓋整個安全區域，還避免路徑重複和不合理轉向。
- 參數調整：
  - λ 控制轉角懲罰，可實現強制 90 度轉向或鬆散轉向策略。
  - 初始族群大小與世代數可根據地圖大小與巡邏點數量調整。

（三）、K-means 聚類 + 基因演算法（GA）的協同應用

1. 使用 K-means 生成均勻分布巡邏點，確保安全區域全部被覆蓋。
2. 將角點偵測出的關鍵點加入巡邏點集合，保留環境結構特徵。
3. 使用基因演算法（GA）對巡邏點進行路徑排序，兼顧路徑長度與角度限制。
4. 導航控制器根據最終巡邏點順序執行任務，並可中斷完成即時任務（羽球收集），再返回繼續巡邏。

（四）、YOLO 羽球偵測 + 深度計算

YOLO 網路輸出 Bounding Box 與 類別機率[31]：

$$\mathrm{P(badminton)} = \sigma(\mathrm{C}) \cdot \sigma\left(\mathrm{p_{obj}}\right) \cdot \max(\mathrm{p_{class}})$$

將偵測框中心投影到深度影像[32]：

$$\mathrm{Z} = \mathrm{Depth}\left(\mathrm{u}，\mathrm{v}\right)，\quad \mathrm{X} = \frac{(\mathrm{u} - \mathrm{c_x})}{\mathrm{f_x}} \cdot \mathrm{Z}，\quad \mathrm{Y} = \frac{\left(\mathrm{v} - \mathrm{c_y}\right)}{\mathrm{f_y}} \cdot \mathrm{Z}$$

- u，v：影像座標（pixel）
- $\mathrm{c_x}$，$\mathrm{c_y}$：相機主點（principal point）
- $\mathrm{f_x}$，$\mathrm{f_y}$：相機焦距（pixel）
- Z：深度值（Depth）
- X， Y：相機座標系的水平與垂直座標

轉換為相機座標後，透過 TF（camera → base_link → map）得到羽球的 map frame 座標。

5. 中斷（Interrupt）機制

巡邏任務運行時，機器人需具備「事件驅動」能力：

1. 正常巡邏 → 接收 YOLO 訊號 （/yolo/detection）
2. 若有羽球被偵測 → 觸發中斷，暫停當前 Nav2 目標
3. 插入新的導航目標 （羽球座標） → 前往回收
4. 回收完成 → 自動恢復巡邏路線

公式上可以視為一個 有限狀態機 FSM：

$$\mathrm{S} = \{\mathrm{Patrol}，\mathrm{Interrupt}，\mathrm{Fetch}，\mathrm{Resume}\}$$

轉換規則：

$$\text{Patrol} \rightarrow \text{Interrupt} \rightarrow \text{Fetch} \rightarrow \text{Resume} \rightarrow \text{Patrol}$$

6. 自走車運動模型
   - 採用差速驅動（Differential Drive）：
   - 輪速轉換[34]：

$$v_L = \frac{2v - \omega L}{2R}，v_R = \frac{2v + \omega L}{2R}$$

   - 位姿更新[34]：

$$\begin{cases} xt+1 = xt + v\cos\theta\Delta t \\ yt+1 = yt + v\sin\theta\Delta t \\ \theta t+1 = \theta t + \omega\Delta t \end{cases}$$

   - 到達判斷[33, 35]：

$$\text{arrived} \Leftrightarrow \sqrt{\left(x_{target} - x_{robot}\right)^2 + \left(y_{target} - y_{robot}\right)^2} \leq r_{threshold}$$

第五章 視覺辨識與深度學習

5.1 羽球偵測與定位

5.1.1 演算法選擇與比較

羽球偵測是自動撿球機器人的核心技術之一，必須同時具備高準確率、快速反應能力與穩定性。由於羽球體積小、顏色偏白且容易與場地背景混合，因此在演算法選擇上需特別謹慎。本研究評估了兩大類演算法：傳統影像處理方法與深度學習方法。[39, 40]

演算法類型	方法	優點	缺點	適用場景
傳統影像處理	HSV 色彩分割	實現簡單，運算量小	對光線、背景敏感，易誤判	簡單背景、單物件環境
傳統影像處理	邊緣檢測與輪廓分析	運算量低，可快速部署	難應對多物件與場地變化	小型或封閉場域
深度學習	卷積神經網絡（CNN）/ YOLO	偵測準確率高，可辨識多物件	需要大量標註資料與運算資源	複雜場地、多羽球、多光線環境

表 8、演算法比較表

傳統影像處理方法適合環境單一且即時性要求高的場景，但精度受限，而深度學習方法在複雜場地、多人或多羽球同時存在的情況下更穩定，且容易與 ROS2 系統整合進行自動導航。

考量以上，本專題選用 YOLOv8 深度學習模型[7]進行羽球偵測，搭配 Roboflow 平台建立資料集[8]，透過 ROS2（Robot Operating System 2, ROS 2）將偵測座標即時輸出給導航模組，達成精準、穩定的自動撿球功能。

5.1.2 模型訓練與參數設置

1. 資料集準備

1. 使用 Roboflow 平台挑選不同光線、角度及距離下的羽球圖像。
2. 每張圖像使用 Bounding Box 標註羽球位置，建立完整資料集。
3. 資料集包含：
    - 單顆羽球與多顆羽球的場景
    - 不同背景與光照條件
    - 不同角度和距離的拍攝影像

2. 模型訓練流程

1. 將標註完成的資料集輸入 YOLOv8 模型。

2. 將資料集拆分為：

    - 訓練集（Train Set）：80%用於模型訓練
    - 驗證集（Validation Set）：20%用於監控訓練過程與調整超參數

3. 超參數設置

- 學習率（Learning Rate）：控制權重更新幅度，設置 0.01。
- 批次大小（Batch Size）：影響訓練穩定性與運算效率，使用 16 張一批次
- 訓練輪數（Epochs）：確保模型充分學習特徵，為了兼顧訓練時間與硬體效能，本專題設定 50 輪
- 損失函數（Loss Function）：採用 YOLOv8 預設的多任務損失函數，包括分類、定位與置信度損失。

4. 模型驗證與調整

1. 在訓練過程中使用驗證集監控 準確率（Precision）、召回率（Recall）與 mAP（mean Average Precision）[36]。
2. 在訓練期間模型也多次未達預期表現，通過以下狀況來進行持續調整，以便達到更好狀況：
    - 增加資料集數量或多樣性
    - 調整資料增強策略
    - 微調學習率或其他超參數

5. 訓練結果輸出

- 訓練完成後，將生成的 YOLOv8 模型權重檔（.pt）部署到 AI 開發版，並透過 ROS2 節點進行即時處理。
- 使用攝影頭，搭配 ROS2 運算結果，獲取羽球距離等資訊[37]。

5.2 深度學習模型應用

5.2.1 視覺辨識流程

1. 影像捕捉

- 使用相機拍攝羽球場地內的羽球特徵。
- 攝影角度需涵蓋整個場地，並確保光線均勻，以利模型穩定偵測。

2. 資料標註

- 在 Roboflow 平台挑選不同光線、角度及距離下的羽球圖像。
- 對每張影像進行 Bounding Box 標註，建立完整的訓練資料集。
- 資料集包含單顆與多顆羽球的場景，以及不同背景與光照條件。

3. 模型訓練

- 將標註完成的資料集輸入 YOLOv8 模型進行訓練。
- 訓練完成後生成權重檔（.pt），用於模型部署與推理。

4. 模型部署

- 將訓練完成的模型部署至 AI 開發版。
- 透過 ROS2 節點接收攝影機影像進行即時推理。

5. 整合

- ROS2 計算座標後，結合場地地圖資訊，運算機器人最佳路徑。
- STM32 控制馬達與撿球機構，自動移動至羽球位置並完成撿球任務。

5.2.2 模型測試

1. 模型測試

- 在羽球場地中，使用深度攝影頭捕捉即時影像，並透過 ROS2 節點進行推理。
- 模型輸出羽球座標後，由導航模組控制 STM32 馬達與撿球機構，使機器人自動移動至羽球位置。

- 記錄模型在不同光線、角度及背景下的偵測準確率和輸出延遲，確認模型可即時產生可用座標。

2. 性能驗證

- 驗證模型能正確辨識單顆與多顆羽球，並在各種場地條件下保持穩定偵測。
- 評估推理速度是否符合即時導航需求，確保機器人可順暢移動而不因延遲產生錯誤動作。

3. 系統整合測試

- 將模型與 ROS2 導航系統及 STM32 微控制器控制模組整合[38]，進行完整自動撿球任務測試。
- 驗證系統在完成自動導航與撿球任務時，偵測與控制的協同運作是否穩定。
- 測試結果提供系統效能基準，用於後續可能的系統優化或功能擴展。

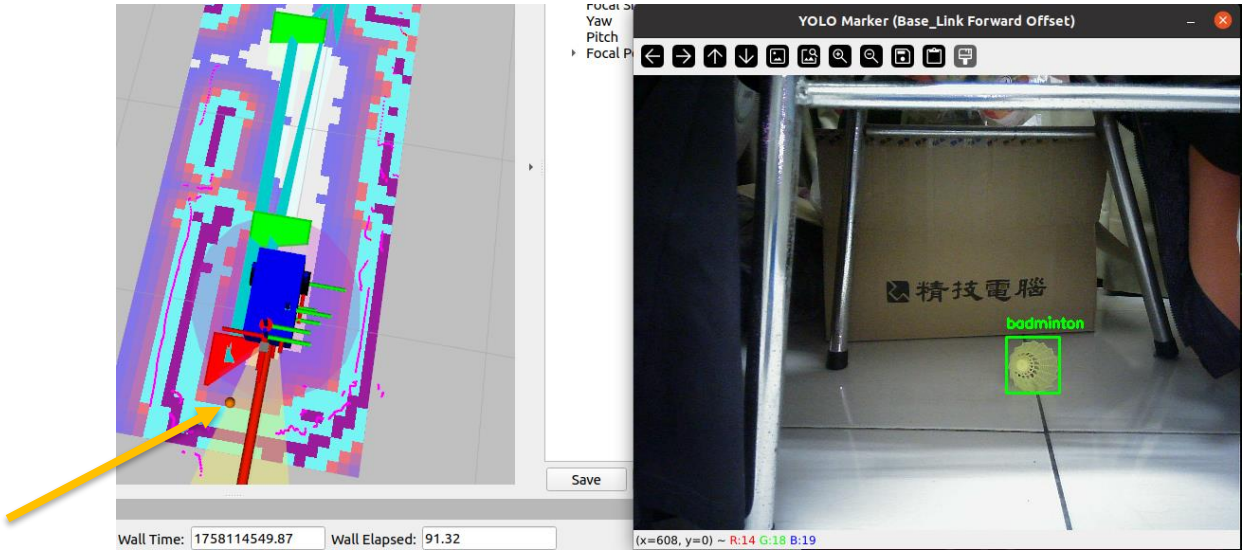


圖 19、羽球辨識應用與位置發送

第六章 巡邏與機器學習

6.1 巡邏點生成與排序

6.1.1 演算法選擇與比較

在自主移動機器人（Autonomous Mobile Robot， AMR）的任務中，巡邏點的生成與排序是一個關鍵問題。

- 生成（Point Generation）： 如何從地圖中挑選代表性點，使機器人能覆蓋環境。
- 排序（Point Ordering）： 如何決定巡邏點的拜訪順序，使得路徑高效且完整[42、43]。

這兩個子問題，實際上對應到：

- 聚類（Clustering）與取樣（Sampling）問題
- 旅行商問題 （Traveling Salesman Problem， TSP）[17、18、30]

因此，我們將介紹不同演算法組合並進行比較，最終選擇最適合的方案。

（一）、巡邏點生成演算法比較

1. 隨機取樣（Random Sampling, RS）[41]

原理：在地圖的安全區域內隨機選取點作為巡邏點。
公式：
對於安全區域集合 S，生成 n 個點：

$$P_i \sim \text{Uniform}(S)， \quad i = 1，2， \ldots，n$$

優點：簡單快速，易於實現。
缺點：可能分布不均，部分區域過密或過疏，巡邏效率低。

2. 均勻網格（Grid Sampling, GS）[42]

原理：將地圖劃分為固定大小的網格，每格選取中心點作為巡邏點。
公式：
對地圖長寬 W， H，網格間距 d：

$$Pi，j =（x_0 + i \cdot d，y_0 + j \cdot d），i = 0， \ldots，[W/d]，j = 0， \ldots，[H/d]$$

優點：分布均勻，覆蓋性好。
缺點：忽略地圖障礙物，生成點可能位於不可行走區域。

3. K-means 聚類（K-Means Clustering, K-means）[43]

原理：將安全區域像素聚類成 k 群，每群中心作為巡邏點。
公式：
最小化群內平方距離：

$$\arg\min_{C} \sum_{i=1}^{k} \sum_{x \in C_i} |x - \mu_i|^2$$

其中C_i為第 i群，μ_i為群中心。
優點：生成均勻分布點，適合多區域覆蓋。
缺點：需要指定群數 k，對初始值敏感。

4. DBSCAN（Density-Based Spatial Clustering of Applications with Noise, DBSCAN）[44]

原理：基於密度的聚類演算法，將密集區域的像素聚為群。
公式：
對半徑ϵ和最小樣本數 MinPts：

$$\text{core point} \Leftrightarrow \left|\{x_j : |x_j - x_i| \leq \epsilon\}\right| \geq \text{MinPts}$$

優點：能自動識別群數，忽略噪點。
缺點：對ϵ和 MinPts 敏感，生成點數不易控制。

5. Mean-Shift（Mean-Shift Clustering, Mean-Shift）[45]

原理：通過平滑核密度函數（Kernel Density Estimation, KDE）尋找密度最大位置作為群中心。
公式：
對每個點x，更新：

$$x_{t+1} = \frac{\sum_i K\left(\frac{x_i - x_t}{h}\right) x_i}{\sum_i K\left(\frac{x_i - x_t}{h}\right)}$$

其中K為核函數，h為帶寬（bandwidth）。
優點：自動找到群數，不需預先指定。
缺點：計算量大，帶寬選擇影響結果。

（二）、巡邏點排序演算法比較

1. 最近鄰（Nearest Neighbor, NN）[46]

原理：從起點出發，每次選擇距離最近的未訪點。
公式：

$$p_{next} = \arg\min_{q \in U} dist(p_{current}, q)$$

其中U為未訪問點集合。
優點：計算簡單，速度快。
缺點：易陷入局部最優，路徑可能較長。

2. 旅行商問題（Traveling salesman problem ，TSP）精確解[17]

原理：解旅行推銷員問題，找出最短巡邏路徑。
公式：

$$\min \sum_{i=1}^{n} d(P_i, P_{i+1}), \quad P_{n+1} = P_1$$

優點：路徑最短，最優解。
缺點：計算量隨點數呈指數增長，對大地圖不實用。

3. 基因演算法（Genetic Algorithm, GA）[13、14]

原理：模擬生物進化，通過選擇、交配與突變尋找最優路徑。
公式：
適應值（fitness）：

$$f(route) = \sum_{i=1}^{n} d(P_i, P_{i+1}) + penalty$$

優點：可處理大規模問題，接近最優解。
缺點：需要調參（種群數、迭代次數），結果帶隨機性。

4. 螞蟻演算法（Ant Colony Optimization， ACO）[18]

原理：模擬螞蟻釋放費洛蒙，選擇路徑概率與費洛蒙濃度相關。
公式：
轉移概率：

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{k \in allowed} \tau_{ik}^{\alpha} \cdot \eta_{ik}^{\beta}}$$

其中τ_{ij}為費洛蒙，$\eta_{ij} = 1/d_{ij}$為啟發值。
優點：適合大規模、多路徑問題，能全局探索。
缺點：收斂慢，需調參，易陷入局部最優。

（三）、演算法組合與表較

組合	生成方法	排序方法	優點	缺點
隨機 +最近鄰（NN）	快速	簡單	適合快速測試	覆蓋率差
網格 +最近鄰（NN）	覆蓋完整	簡單排序	適合小地圖	計算量大
K-means 聚類 +基因演算法（GA）	均勻分布	近似最優排序	平衡效率與品質	計算量中等
DBSCAN + ACO	適合不規則區域	可跳脫局部最小	高適應性	時間消耗大
Mean-Shift + 基因演算法（GA）	自動決定群數	最佳化排序	智能化高	計算最重

表 9、演算法組合比較表

測驗綜合表現後，我們選擇 K-means 聚類+ 基因演算法（GA）的組合作為巡邏點生成與排序演算法，主要原因在於兩者互補。K-means 聚類能夠自動將安全區域劃分成均勻分布的群集，生成的巡邏點數量可控且覆蓋完整，避免盲區或過於集中。而基因演算法（GA）作為排序演算法，能有效地解決巡邏點路徑優化問題，接近全局最短路徑，並可調整參數控制收斂速度。兩者搭配下，不僅保證巡邏點分布合理，也在計算效率與路徑品質之間取得平衡，適合中大型地圖上的自主巡邏。

6.1.2 參數設置

巡邏點生成系統主要由 K-means 聚類與基因演算法（GA）排序構成。K-means 聚類用於分析地圖中的安全區域，將自由空間劃分為多個群集，並以群集中心作為巡邏點，保證點位均勻分布並避免偏僻區域。聚類參數如帶寬（bandwidth）會影響巡邏點密度，帶寬過大可能造成點位稀疏，過小則產生過多冗餘點。

生成巡邏點後，基因演算法（GA）用於優化路徑順序，使總路徑長度最短並符合可行性。系統可啟用 90°轉向限制，在適應度函數中對每三個連續點形成的角度進行計算，若偏離 80°~100°，施加懲罰，確保路徑轉向平滑、安全。基因演算法（GA）的族群大小、迭代次數及突變率影響最終路徑品質與計算效率，需依地圖規模調整以平衡收斂速度與路徑最優性。

參數名稱	說明	機器學習影響	可能造成的問題
num_points	指定巡邏點數量	K-means 聚類中心數量	設定過大：聚類中心過多，部分巡邏點可能落在安全邊界或偏僻位置；設定過小：巡邏點數量不足，安全區域覆蓋不完全
area_per_point	每個巡邏點應覆蓋的安全區域面積，用於自動計算巡邏點數量	間接影響 K-means 聚類中心數量	面積過大 → 點數過少，部分角落未被覆蓋；面積過小 → 點數過多，計算量增加，基因演算法（GA）收斂時間長
min_points / max_points	限制生成巡邏點的上下限	保護 K-means 聚類結果在合理範圍內	若 min_points 過小，巡邏點不足 → 部分安全區域未被巡邏；若 max_points 過大，巡邏點過多 →基因演算法（GA）排序運算量增加
enforce_90deg_turn	是否強制巡邏路徑角度接近 90°	不影響 K-means 聚類，但影響基因演算法（GA）排序結果	強制角度可能使路徑變長，或迫使巡邏點排序偏離最短路徑[13、43]
strict_90deg_limit	90°角度容忍範圍	與 K-means 聚類無關	容忍度過小可能導致基因演算法（GA）找不到合理路徑

表 10、參數設置表

6.2 機器學習應用

6.2.1 機器學習流程

在機器學習應用前我們需要先從地圖中提取/map 相關資訊，同時啟用 nav2 套件中的 global_costmap 和 local_costmap 兩個地圖膨脹數據[34、35]，確保與牆壁保持安全距離，再來自動生成 K-means 聚類所需的 Npoints，並交給遺傳算法（GA）排序。

在收到巡邏點後，我們需要先確認是否設置 enforce_90deg_turn 以及 strict_90deg_limit 來確保路徑不會太難以行駛，最後將限制交給 GA 生成最佳巡邏路徑並發布給自走車控制指令。

（1）、地圖資料前處理

- 從 ROS 2 /map 主題取得 OccupancyGrid 地圖資料。[34]
- 將地圖轉換為二值安全區域矩陣 $S(y, x)$：

$$S(y,x) = \begin{cases} 1 \text{ 障礙物} \\ 0 \text{ 空地} \end{cases}$$

- 將安全區域像素座標轉換為二維點集：

$$X_{safe} = \{(x_i, y_i) \mid S(y_i, x_i) = 0\}$$

- 若某個巡邏點落在障礙物附近，會透過鄰域檢查微調：

$$(x_{safe}, y_{safe}) = \arg \min_{(x_j, y_j) \in N_r(x_i, y_i)} \text{距離}\left((x_i, y_i), (x_j, y_j)\right)$$

（2）、自動計算巡邏點數量

為確保巡邏點密度合理，根據安全區域面積自動計算巡邏點數量：

$$N_{points} = \min\left(\max\left(\text{min\_points}, \frac{A_{safe}}{\text{area\_per\_point}}\right), \text{max\_points}\right)$$

其中：

- $A_{safe} = N_{safe} \cdot r^2$
  - N_{safe} 為安全像素數量
  - r 為地圖解析度（每個像素對應地面長度，單位 m^2）
- area_per_point 為每個巡邏點希望覆蓋的面積
- min_points / max_points 為上下限

程式實作範例：

這段程式會依據地圖大小與安全區域面積自動決定巡邏點數量。

```
570     def auto_compute_num_points(self, map_data: OccupancyGrid):
571         resolution = map_data.info.resolution
572         data = np.array(map_data.data, dtype=np.int8).reshape((map_data.info.height, map_data.info.width))
573         obstacle_mask = (data > 0)
574         dilation_iters = max(1, int(0.3 / resolution))
575         dilated_obstacle = binary_dilation(obstacle_mask, iterations=dilation_iters)
576         safe_area = np.logical_not(dilated_obstacle)
577         safe_cells = np.count_nonzero(safe_area)
578         free_area_m2 = safe_cells * (resolution ** 2)
579         estimated = int(max(self.min_points, min(self.max_points, math.floor(free_area_m2 / self.area_per_point))))
580         estimated = max(1, estimated)
581         safe_pixels = np.argwhere(safe_area)
582         if len(safe_pixels) < estimated:
583             estimated = max(1, len(safe_pixels))
584         self.get_logger().info(f"Auto num_points: {estimated} (safe_cells={safe_cells}, free_area={free_area_m2:.2f} m²)")
585         return estimated
```

圖 20、地圖資料程式碼

（3）、K-means 聚類生成巡邏點[43]

1. K-means 聚類生成中心點

- 將安全區域座標作為特徵向量X_{safe}進行 K-means 聚類：

$$\{C_1，C_2，\ ...\ ，C_N\} = \text{K-means}(X_{safe}，N_{points})$$

- 聚類中心$C_i = (x_i，y_i)$即為候選巡邏點。

2. 聚類中心微調

- 聚類中心可能落在障礙物或邊界附近，程式會在中心附近的安全像素中選擇最近的一個。
- 若鄰域內無安全像素，則在整個安全區域中找到最近的安全點作為巡邏點。
- 這步確保每個巡邏點都落在安全可行區域。

3. 座標轉換

- 將聚類中心的像素座標轉換為 ROS 世界座標，用於導航。
- 每個巡邏點會保存成 Point（x， y， z） 格式，方便機器人使用。

程式實作範例：

```python
    def generate_kmeans_points(self, map_data, num_points):
        width = map_data.info.width
        height = map_data.info.height
        resolution = map_data.info.resolution
        origin = map_data.info.origin.position
        data = np.array(map_data.data, dtype=np.int8).reshape((height, width))
        obstacle_mask = (data > 0)
        dilation_iters = max(1, int(0.3 / resolution))
        dilated_obstacle = binary_dilation(obstacle_mask, iterations=dilation_iters)
        safe_area = np.logical_not(dilated_obstacle)
        safe_pixels = np.argwhere(safe_area)
        if len(safe_pixels) < num_points:
            self.get_logger().warn("Safe area too small for requested points, reducing num_points.")
            num_points = max(1, len(safe_pixels))
        sample_limit = 5000
        if len(safe_pixels) > sample_limit:
            idx = np.random.choice(len(safe_pixels), sample_limit, replace=False)
            kmeans_input = safe_pixels[idx]
        else:
            kmeans_input = safe_pixels
        n_clusters = min(num_points, len(kmeans_input))
        kmeans = KMeans(n_clusters=n_clusters, n_init=10, random_state=0)
        kmeans.fit(kmeans_input)
        centers = kmeans.cluster_centers_
        patrol_points = []
        for cy, cx in centers:
            cy_int, cx_int = int(round(cy)), int(round(cx))
            if not safe_area[cy_int, cx_int]:
                neighbors = safe_pixels[np.linalg.norm(safe_pixels - np.array([cy_int, cx_int]), axis=1) < 3]
                if len(neighbors) == 0:
                    dists = np.linalg.norm(safe_pixels - np.array([cy_int, cx_int]), axis=1)
                    ni = np.argmin(dists)
                    cy_int, cx_int = safe_pixels[ni]
                else:
                    cy_int, cx_int = neighbors[0]
            wx = origin.x + cx_int * resolution
            wy = origin.y + cy_int * resolution
            patrol_points.append(Point(x=float(wx), y=float(wy), z=0.0))
        return patrol_points
```

圖 21、K-means 聚類程式碼

（4）、基因演算法（GA）路徑排序與直角限制[13, 14, 15]

為了得到一條合理且短的巡邏路徑，使用 基因演算法（GA）進行路徑排序。核心數學公式：

1. 總路徑長度：

$$L_{path} = \sum_{i=1}^{N} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

2. 碰撞懲罰：

$$f_{collision} = 1000 \cdot \text{collision\_penalty}$$

若路徑經過障礙物，則懲罰極大，避免生成不可行路徑。

3. 直角限制懲罰（啟用 enforce_90deg_turn）：

$$f_{angle} = 10 \sum_{i=1}^{N} \max\left(0，|\theta_i - 90°| - \delta\right)$$

其中：

- θ_i為連續三個巡邏點的夾角：

$$\theta_i = \arccos\left(\frac{(P_{i-1} - P_i) \cdot (P_{i+1} - P_i)}{|P_{i-1} - P_i| \cdot |P_{i+1} - P_i|}\right)$$

- δ為容忍角度（程式中對應 strict_90deg_limit，例如 10°）
- f_{angle} 會加入 GA 適應度，促使生成的路徑角度接近 90°，利於狹窄走道巡邏。

4. 基因演算法（GA）適應度函數：

$$F(path) = L_{path} + f_{collision} + f_{angle}$$

基因演算法（GA）使用選擇、交叉、變異操作，迭代尋找適應度最小的巡邏路徑。

程式實作範例：

```
627     def optimize_path(self, points, enforce_90deg=True, strict_limit=10):
628         if len(points) <= 2:
629             return points
630         unvisited = points.copy()
631         path = [unvisited.pop(0)]
632         while unvisited:
633             last = path[-1]
634             dists = [math.hypot(p.x-last.x, p.y-last.y) for p in unvisited]
635             min_idx = np.argmin(dists)
636             path.append(unvisited.pop(min_idx))
637         return path
```

圖 22、基因演算法（GA）程式碼

```python
    # ============ 巡邏點產生 ============
    def generate_patrol_points(self, msg: OccupancyGrid):
        width = msg.info.width
        height = msg.info.height
        resolution = msg.info.resolution
        origin = msg.info.origin.position
        data = np.array(msg.data, dtype=np.int8).reshape((height, width))

        obstacle_mask = (data > 0)
        dilation_iters = max(1, int(0.3 / resolution))
        dilated_obstacle = binary_dilation(obstacle_mask, iterations=dilation_iters)
        self.safe_area = np.logical_not(dilated_obstacle)

        # 角落檢測
        corners = cv2.goodFeaturesToTrack((self.safe_area*255).astype(np.uint8),
                                          maxCorners=100,
                                          qualityLevel=0.01,
                                          minDistance=3,
                                          blockSize=3,
                                          useHarrisDetector=False)
        if corners is None:
            corners = np.empty((0,1,2))
        corners = corners.reshape(-1, 2)

        filtered_corners = self.filter_corners_by_radius_priority(corners, int(self.corner_radius))
        if len(filtered_corners) > self.corner_point_num:
            filtered_corners = filtered_corners[:self.corner_point_num]

        self.corner_points = []
        for c in filtered_corners:
            wx = origin.x + c[0] * resolution
            wy = origin.y + c[1] * resolution
            self.corner_points.append(Point(x=float(wx), y=float(wy), z=0.0))

        # KMeans
        num_pts = self.auto_compute_num_points(msg) if self.num_points is None else self.num_points
        self.kmeans_points = self.generate_kmeans_points(msg, num_pts)

        # 合併
        combined_points = self.kmeans_points + self.corner_points
        self.get_logger().info(f'合併巡邏點與角點，共 {len(combined_points)} 個點，進行路徑優化')

        self.patrol_points = self.optimize_path(combined_points,
                                                enforce_90deg=self.enforce_90deg_turn,
                                                strict_limit=self.strict_90deg_limit)

        self.get_logger().info(f'產生總共 {len(self.patrol_points)} 個巡邏點')
```

圖 23、巡邏點生成主程式碼

6.2.2 測試與優化

為了驗證機器學習的巡邏點於真實場景的生成表現，我們分別在小型室內空間（250x80cm）以及學校體育館的羽球場進行試驗，分別測驗其對於路徑規劃與生成巡邏點的數量、位置與排序進行測試。

1. 小型環境測試（租屋處 250 x 80cm 走道）

由於不可能頻繁攜帶自走車至羽球場，因此我們先在房間進行小範圍測試。此環境狹小、障礙物較多，可移動空間有限。

同時因為空間狹小，可以更方便我們檢測其轉彎角度及路徑選擇是否有最佳化。

- 巡邏點生成：由於地圖過小，演算法自動套用最低限制，僅生成 4 個巡邏點，避免產生偏僻或過密的點位。

- 移動精度：在此空間下，路徑較短，便於測試演算法正確性。結果顯示機器人能以誤差小於 10cm 到達目標點，證明了路徑生成的可靠性。

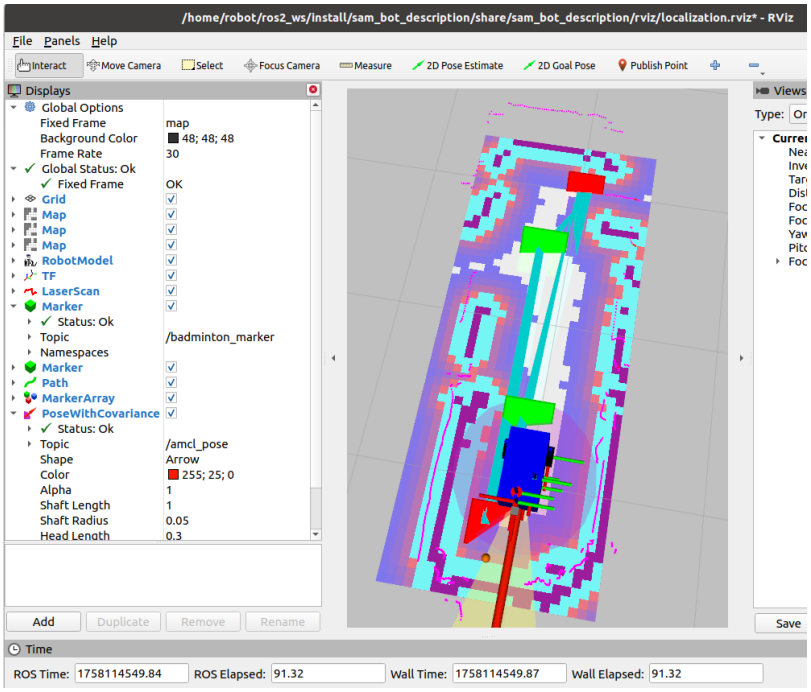


圖 24、室內生成與規劃

- 優化措施：在測試中發現，若 band_width 設定過大，巡邏點容易偏向房間邊角；若過小，則巡邏點數量不足。因此我們將 band_width 動態調整為地圖對角線長度的 1~2%，使點分布更合理。

2. 實際環境測試（羽球場）

完成小規模驗證後，我們將自走車搬至羽球場進行實測。羽球場屬於開放式環境，標線尺寸 13.4 × 6.1m，場地由數個羽球場組成，更能檢驗演算法在中大型地圖下的可行性。

- 巡邏點生成：在羽球場，K-means 聚類自動生成 22 個巡邏點，均勻分布於球場範圍。
- 路徑排序與優化：
  - 初始隨機路徑長度偏長，轉彎過多。
  - 經 基因演算法（GA）優化後，總路徑長度縮短，且大部分轉彎保持直角（80~100 度），更符合機器人底盤運動特性。
- 優化措施：

在羽球場實測中，我們觀察到 K-means 聚類能生成均勻巡邏點，但部分點位仍可能接近場邊或造成冗餘移動。

為優化分布，我們調整 band_width，並透過基因演算法（GA）強化路徑排序，減少不必要繞行與急轉彎，使路徑長度縮短約 25%，轉角更平滑。

num_points 自動化設置使巡邏點能夠隨著地圖大小動態匹配，sample_limit 平衡運算效率與代表性，基因演算法（GA）的種群與迭代次數設定加快收斂並接近全域最優解。

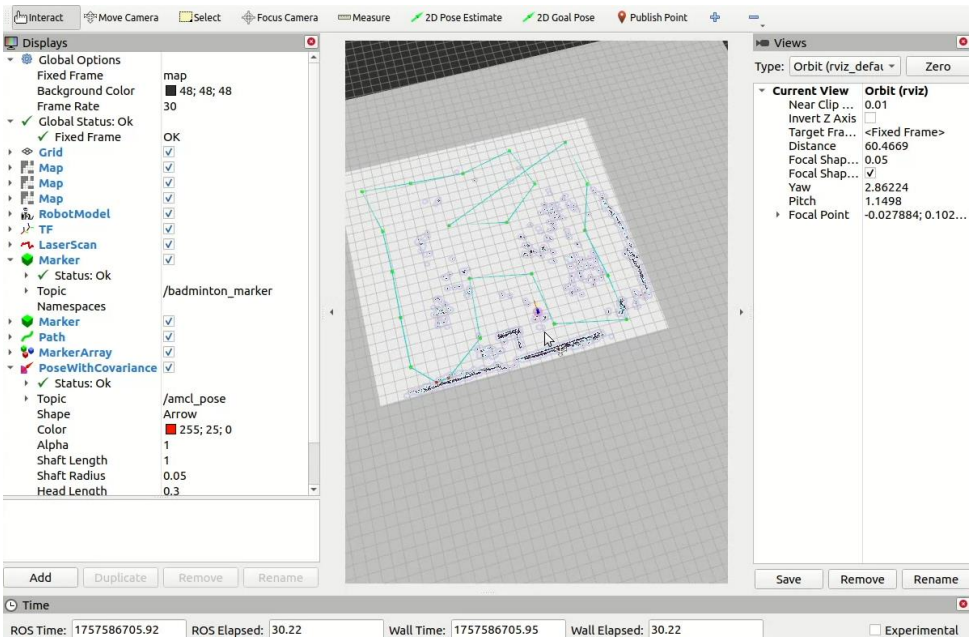


圖 25、羽球場巡邏點生成與規劃

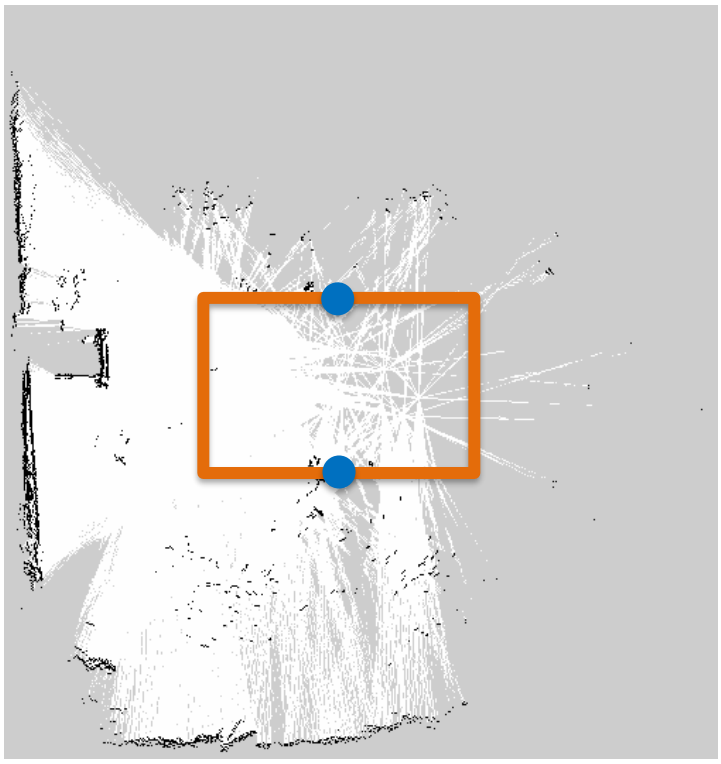


圖 26、半場掃描羽球場地圖

- 球桿:
- 球場:

第七章 問題描述與解決

7.1 STM32 與 AI 開發版齊頭問題

在本專案中，STM32 與 AI 開發版之間的通信扮演著重要角色。在最初的實作中，我們設計了使用四位數字（例如 0010）來控制 STM32 微控制器的左右轉動。但在實際運行過程中，我們發現，由於接收過程中的數據丟失或錯誤，這種純粹的數字傳輸方式經常無法正確傳遞指令，導致控制精度的下降。

為了解決這一問題，我們進行了數據格式的修改，將數字前面添加了固定的齊頭碼（例如 A0010），變為帶有特定格式的數字串。這樣的改動使得 STM32 微控制器在接收來自 AI 開發版的指令時，能夠確保數據的完整性和準確性，避免了因數據丟失或錯誤而產生的控制問題。

具體而言，齊頭碼使接收系統能夠更準確地識別每個數據包的開始，並提高了通信的穩定性。此方法有效避免了原先僅傳送純數字時出現的通信錯誤，進一步增強了 STM32 微控制器與 AI 開發版之間的協同工作。

7.2 路徑規劃無法使用 nav_followpoint，改用 nav_pose

在進行路徑規劃時，我們原本的計畫是使用 ROS 中的 nav_followpoint 模塊來實現精確的點對點導航。nav_followpoint 的目的是根據預定的目標點，精確控制機器人沿著指定路徑移動，並且能夠動態調整路徑以避免障礙物，這對於進行高精度的導航至關重要。然而，在實施過程中，我們發現系統的韌體無法有效處理 nav_followpoint 模塊，這可能是由於軟硬體的兼容性問題，或者是系統資源分配不足，導致無法實現預期的導航精度和穩定性。

這個問題對整體系統運行造成了嚴重影響，特別是在需要高精度路徑跟蹤的應用場景中。為了解決這個問題，我們決定將導航模式切換為 nav_pose，這是一個能夠根據當前位置和目標位置進行位置跟蹤的模塊。nav_pose 是一種較為簡單的導航方法，它能夠根據機器人的當前姿態（位置與角度）來調整路徑，並進行穩定的運行。

改用 nav_pose 的影響與調整：

- 兼容性問題解決： 使用 nav_pose 有效避開了原本在 nav_followpoint 中出現的韌體兼容性問題，並能夠保持系統的穩定運行。
- 路徑規劃算法調整： 雖然 nav_pose 解決了系統的兼容性問題，但由於其基於位置和角度的控制方式不同於 nav_followpoint，我們不得不對原有的路徑規劃算法進行調整。這包括修改控制邏輯，讓機器人根據當前的定位信息和目標位置進行最短路徑計算，而不是直接跟蹤預設的路徑點。

- 精度與穩定性的提升： 這一改動大大提升了系統的穩定性，尤其是在面對複雜地形或動態環境時，nav_pose 模塊能夠更好地進行即時的調整，從而提高了導航過程中的精確度。

7.3 取樣 map 無法直接使用 map_server 的內容

在本專案中，map_server 被用來提供機器人運行所需的地圖數據。在理想情況下，map_server 會在啟動時發布一次地圖，並持續提供地圖更新，以便機器人能夠基於最新的地圖進行運作。然而，在實際運行中，我們發現，map_server 並不會持續發布地圖，僅會在啟動時發佈一次，這對於需要持續進行地圖取樣的系統來說，造成了不小的挑戰。

這樣的設計問題意味著，當我們需要使用地圖進行導航或路徑規劃時，無法直接從 map_server 中獲得最新的地圖數據。這一情況在動態環境中尤其棘手，因為機器人可能需要根據實時的地圖更新來進行決策和規劃，而無法依賴靜態的地圖。

為了解決這個問題，我們改變了地圖取樣的方式。具體來說，當 map_server 發佈地圖後，我們不再依賴即時的數據流，而是將 map_server 發布的最新地圖保存在一個緩存中。在每次需要地圖數據時，我們直接從緩存中獲取最後一次發佈的地圖數據。

這樣的改動有效解決了無法實時從 map_server 獲取最新地圖的問題，並確保機器人能夠在每次執行導航或路徑規劃時，使用到最後一次有效發布的地圖數據。此外，我們實現了一個檢查機制，確保每當有新的地圖發佈時，系統會自動更新緩存中的地圖數據，從而確保地圖的準確性。

第八章 結論與未來展望

8.1 研究成果總結

此專題成功研製出一套結合 ROS2（Robot Operating System 2, ROS 2）機器人軟體、電腦視覺與嵌入式控制的智慧型羽球自動收集機器人系統。整體系統具備自動模式與手動模式兩種運作方式：在自動模式下，機器人會先檢查是否已存在場地地圖，若有則使用路徑規劃演算法生成最佳巡迴路線並自主巡邏以蒐集散落的羽球；若無地圖，則進入探索模式，透過感測器資料與 SLAM（Simultaneous Localization and Mapping, SLAM）即時建構環境地圖。同時，深度學習視覺模型負責即時辨識場地上的羽球，ROS2 則協調各模組執行任務並下達移動與蒐集決策。手動模式下，使用者可利用遙控器直接控制車體運動，並且可選擇是否啟動地圖構建功能，以支援後續的自動巡迴與定位。無論自動或手動模式，系統運作最終都會產生或更新環境地圖，為後續路徑規劃與導航決策提供依據；而整體流程確保機器人在運動過程中同步完成羽球偵測、定位與蒐集任務，充分展現本系統在電腦視覺、機器人軟體架構與嵌入式控制整合上的應用價值。

在硬體架構方面，系統採用了雙層次控制設計：底層以 STM32 微控制器作為核心，負責驅動雙輪差速底盤的移動以及前方滾輪撿球機構的執行，確保車體動作的即時性與穩定性。高層則由 AI 開發版嵌入式運算平台構成，大量計算任務如 ROS2 資訊處理、SLAM 建圖與導航規劃、以及 YOLOv8 深度學習模型推理均由 AI 開發版負責處理。兩者透過串列埠 UART 進行資料通訊：STM32 定期回傳編碼器等感測數據供定位與里程計計算，同時接收 AI 開發版經 ROS2 決策後下達的速度控制與蒐集動作指令，形成穩健的「高層規劃—低層控制」雙層控制架構。此外，在機構設計上，車體採用輕量化的鋁合金框架搭配多處 3D 列印與壓克力部件，確保結構強度與減重並行。機器人配備雙驅動馬達（含編碼器）、一組高扭力滾輪撿球裝置、鋰電池電源模組，以及 2D LiDAR、深度攝影機和 IMU 等感測器，用於環境感知與自主避障。整合上述軟硬體技術，本專題達成了羽球的自動偵測與回收目標，打造出一套能自主巡迴羽球場並穩定執行撿球任務的機器人系統。

8.2 實驗與結果討論

為驗證本系統的效能，我們分別對視覺辨識模組、導航巡邏演算法以及整體整合表現進行了實驗測試與結果分析。首先，在羽球物件辨識實驗中，我們使用深度攝影機擷取即時畫面並透過 ROS2（Robot Operating System 2, ROS 2）將影像輸入訓練好的 YOLOv8 模型進行推理。結果顯示該模型在不同光線、拍攝角度及場地背景下皆能偵測出場地中的羽球，且輸出座標資訊的延遲極低，滿足即時性要求。這證實了本系統視覺模組的可靠性：在各種實際條件下，YOLOv8 模型均能提供精準的羽球位置座標，支援後續導航與蒐集決策而不會因延遲造成機器人動作錯誤。

其次，在自主巡邏與路徑規劃部分，實驗選擇了小型室內環境與標準羽球場兩種場景來測試演算法的有效性。在狹小的室內走道環境（約 2.5 × 0.8 公尺）下，系統基於即時建構的地圖自動生成了 4 個巡邏點（受限於空間大小的最低點數）進行覆蓋巡迴。機器人依序導航這些目標位置，實測到達各點的定位誤差均在 10 公分以內，路徑跟隨精度良好，顯示所提巡邏點生成與路徑規劃策略在小尺度環境下具有高度可靠性。隨後在標準羽球場（13.4 × 6.1 公尺，開放式空間）進行測試，演算法透過對整個自由空間的分析自動產生了 22 個均勻分布於場地的巡邏點，基本覆蓋了球場內所有區域。經由基因演算法（GA）對巡邏點訪問順序進行優化後，機器人總巡迴路徑長度較隨機順序大幅縮短（約降低了 25%），且大部分轉彎角度維持在直角範圍（約 80°～100°）。優化後的路徑不僅明顯減少了不必要的繞行，亦使轉向更平滑符合車體運動特性，驗證了所設計巡邏演算法在中大型地圖下的可行性與效率。

最後，整體系統的整合測試結果亦證明了各子模組協同工作的穩定性。我們將物件辨識、導航規劃以及底盤控制模組進行端對端集成，在實地自動撿球任務中觀察機器人的表現。當機器人巡邏過程中攝影機偵測到羽球目標時，系統能即時中斷原定巡邏路線，透過 ROS2 導引底盤精確移動至目標位置並啟動 STM32 控制的機械蒐集裝置將羽球回收；完成撿球後，機器人可以自行返回先前中斷的位置繼續未完成的巡邏任務。連續多次實驗順利完成了指定區域的羽球蒐集任務。上述結果說明本專題所構建的系統在真實環境中達到了預期的功能目標，各項技術指標均符合專題需求。

8.3 未來改進方向

1、提升視覺辨識精度與環境適應性：

可考慮持續擴充羽球影像資料集，納入更多樣化的場地材質、光源條件及羽球顏色/狀態，以強化深度學習模型在各種情境下的穩健性。此外，未來亦可嘗試引入更先進的物件偵測或物件辨識模型，提升對較遠距離或部分遮蔽羽球的辨識率，確保視覺模組在更嚴苛環境中依然表現良好。

2、強化導航演算法與多目標蒐集策略：

在軟體上，可進一步改進巡邏路徑規劃與導航演算法。針對場地中同時存在的多顆羽球目標，未來可開發更加智能的多目標行為決策策略，讓機器人能根據羽球位置分布及距離遠近，自主判斷蒐集順序或劃分區域逐一蒐集，以提高整體任務效率。

3、增進擴充性：

在大型體育館部署多台撿球機器人協同工作，以縮短集體訓練後的回收時間，或將本系統技術移植應用到網球、高爾夫等其他球類的自動撿球裝置上。

參考文獻

[1] NVIDIA Developer Forum, “PyTorch for Jetson,” Mar. 27, 2019. [Online]. Available: https://forums.developer.nvidia.com/t/pytorch-for-jetson/72048

[2] N. Vo Dong, “How to Easily Install PyTorch on Jetson Orin Nano running JetPack 6.2,” NinjaLABO Blog, Feb. 27, 2025. [Online]. Available: https://ninjalabo.ai/blogs/jetson_pytorch.html

[3] YouTube, “Easy SLAM with ROS using slam_toolbox,” [Online Video]. Available: https://www.youtube.com/watch?v=ZaiA3hWaRzE

[4] YouTube, “Making robot navigation easy with Nav2 and ROS!,” [Online Video]. Available: https://www.youtube.com/watch?v=jkoGkAd0GYk

[5] YouTube, “五分鐘學會 PID 控制,” [Online Video]. Available: https://www.youtube.com/watch?v=nQXyNE-xJ2k

[6] YouTube, “如何下載 VMware Workstation Pro！完整教學！,” [Online Video]. Available: https://www.youtube.com/watch?v=LCOvSQPzW5g

[7] Ultralytics, “YOLOv8 Documentation,” [Online]. Available: https://docs.ultralytics.com/

[8] Roboflow, “Build Custom Object Detection Models,” [Online]. Available: https://roboflow.com/

[9] 泓宇, “重頭開始教你如何用 YOLOv8 訓練自己的資料集,” Medium, Jan. 15, 2024. [Online]. Available: https://henry870603.medium.com/重頭開始教你如何用 YOLOv8 訓練自己的資料集-ab5425746bd0

[10] T. Huang, “機器學習：集群分析 K-means Clustering,” Medium, Apr. 27, 2018. [Online]. Available: https://chih-sheng-huang821.medium.com/機器學習-集群分析-k-means-clustering-e608a7fe1b43

[11] YouTube, “OpenCV Course – Full Tutorial with Python,” [Online Video]. Available: https://www.youtube.com/watch?v=oXlwWbU8l2o

[12] D. Nirwan, “Occupancy Grid Mapping with Webots and ROS2,” Towards Data Science, Aug. 3, 2021. [Online]. Available: https://towardsdatascience.com/occupancy-grid-mapping-with-webots-and-ros2-a6162a644fab

[13] DataCamp, “Genetic Algorithm: Complete Guide With Python Implementation,” Jul. 29, 2024. [Online]. Available: https://www.datacamp.com/tutorial/genetic-algorithm-python

[14] H. Cheng, “Python — 基因演算法 (Genetic Algorithm, GA) 求解最佳化問題,” Medium, Aug. 29, 2021. [Online]. Available: https://medium.com/hunter-cheng/python-基因演算法-genetic-algorithm-ga-求解最佳化問題-b7e6d635922

[15] D. Deng, “演化式學習的基礎 — 基因演算法,” Medium, Mar. 25, 2024. [Online]. Available: https://medium.com/@deborah.deng/演化式學習的基礎-基因演算法-6e1a1a068c1b

[16] Dept. of Computer Science, Columbia Univ., “CS W4733 Notes – Differential Drive Robots,” Sept. 16, 2013. [Online]. Available: https://www.cs.columbia.edu/~allen/F17/NOTES/icckinematics.pdf

[17] The University of Hong Kong Algorithms Library, “Traveling salesman problem (TSP),” [Online]. Available: https://i.cs.hku.hk/~hkual/Notes/Greedy/Traveling%20salesman%20problem%20(TSP).html

[18] 邱秉誠, “以 Python 實作蟻群演算法並解決旅行商人問題,” Medium, Jan. 26, 2021. [Online]. Available: https://medium.com/qiubingcheng/以 python 實作蟻群最佳化演算法-ant-colony-optimization-aco-並解決 tsp 問題-上-b8c1a345c5a1

[19] “DC Motor Speed: System Modeling,” Control Tutorials for MATLAB and Simulink (CTMS), University of Michigan. [Online]. Available: https://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SystemModeling

[20] “Measuring Speed and Position with the QEI Module,” Application Note AN93002A, Microchip Technology Inc., Jul. 1, 2005. [Online]. Available: https://ww1.microchip.com/downloads/en/appnotes/93002a.pdf

[21] “DC Motor Speed: PID Controller Design,” MATLABSolutions Control System Documentation. [Online]. Available: https://www.matlabsolutions.com/documentation/control-system/dc-motor-speed-pid-controller-design.php

[22] A. Topiwala, P. Inani, and A. Kathpal, “Frontier Based Exploration for Autonomous Robot,” University of Maryland Robotics Center, Jul. 2018. [Online]. Available: https://arxiv.org/pdf/1806.03581

[23] K. M. Han and Y. J. Kim, “Autoexplorer: Autonomous Exploration of Unknown Environments Using Fast Frontier-Region Detection and Parallel Path Planning,” IROS 2022. [Online]. Available: https://graphics.ewha.ac.kr/autoexplorer/IROS22_3683.pdf

[24] R. Roy et al., “Path Planning and Motion Control of Indoor Mobile Robot Using LiDAR Sensor Data,” Sensors & Materials, vol. 36, no. 5, pp. 1961-1976, May 2024. [Online]. Available: https://sensors.myu-group.co.jp/sm_pdf/SM3647.pdf

[25] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics, MIT Press, 2005. [Online]. Available: https://www.probabilistic-robotics.org

[26] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, Introduction to Autonomous Mobile Robots, 2nd ed., MIT Press, 2011. [Online]. Available: https://github.com/correll/Introduction-to-Autonomous-Robots

[27] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 4th ed. Pearson, 2018. [Online].Available: https://www.mdpi.com/2220-9964/4/4/2821

[28] “k-means clustering,” Wikipedia, 2020. [Online]. Available: https://en.wikipedia.org/wiki/K-means_clustering

[29] GeeksforGeeks, “Determining the Number of Clusters in Data Mining,” Feb. 13, 2022. [Online].Available: https://www.geeksforgeeks.org/determining-the-number-of-clusters-in-data-mining/

[30] Y. Tao et al., “A Path-Planning Method for Wall Surface Inspection Robot Based on Improved Genetic Algorithm,” Electronics, vol. 11, no. 8, p. 1192, 2022. [Online].Available: https://doi.org/10.3390/electronics11081192

[31] D.-K. Kim, “You Only Look Once: The Core of Deep Learning-Based Object Detection,” Medium, Jan. 9, 2025. [Online].Available: https://medium.com/@kdk199604/you-only-look-once-the-core-of-deep-learning-based-object-detection-50e28b615b4a

[32] Open3D, “create_point_cloud_from_depth_image,” Open3D Documentation, 2019. [Online].Available: http://www.open3d.org/docs/0.7.0/python_api/open3d.geometry.create_point_cloud_from_depth_image.html

[33] Cuemath, “Euclidean Distance Formula,” Cuemath.com. [Online].Available: https://www.cuemath.com/euclidean-distance-formula/

[34] Open Robotics, “Wheeled Mobile Robot Kinematics – ROS 2 Control Documentation,” 2025. [Online].Available: https://control.ros.org/rolling/doc/ros2_controllers/doc/mobile_robot_kinematics.html#differential-drive-robot

[35] MathWorks, “Path Following for a Differential Drive Robot,” MATLAB Documentation, accessed Nov. 2025. [Online].Available: https://www.mathworks.com/help/robotics/ug/path-following-for-differential-drive-robot.html

[36] Ultralytics, “Mean Average Precision (mAP) in Object Detection,”

Ultralytics Blog, Aug. 28, 2025. [Online]. Available: https://www.ultralytics.com/blog/mean-average-precision-map-in-object-detection

[37] Intel RealSense, “Projection, Texture-Mapping and Occlusion with Intel® RealSense Depth Cameras,”

RealSense Documentation, 2020. [Online]. Available: https://dev.realsenseai.com/docs/projection-texture-mapping-and-occlusion-with-intel-realsense-depth-cameras

[38] Open Robotics, “ROS 2 Perception Pipeline,” ROS 2 Documentation, 2024. [Online]. Available: https://docs.ros.org/

[39] G. Moreira, S. A. M., F. N. d. S. and M. C., “Benchmark of Deep Learning and a Proposed HSV Colour Space Model,” Agronomy, vol. 12, no. 2, 356, 2022. [Online]. Available: https://www.mdpi.com/2073-4395/12/2/356

[40] N. Hassan, K. W. Ming and C. K. Wah, “A Comparative Study on HSV-based and Deep Learning-based Object Detection Algorithms for Pedestrian Traffic Light Signal Recognition,” Proc. 2020 3rd Int. Conf. Intelligent Autonomous Systems (ICoIAS), Singapore, Feb. 26-29, 2020. [Online]. Available: https://www.researchgate.net/publication/338229433_A_Comparative_Study_on_HSV-based_and_Deep_Learning-based_Object_Detection_Algorithms_for_Pedestrian_Traffic_Light_Signal_Recognition

[41] K. Y. Chan, “Master Thesis Guidelines,” National Taiwan University, 2025, p. 9. [Online]. Available: https://www.lib.ntu.edu.tw/img/tulblog/HELP/Master_20250520_Kuei_Yuan_Chan.pdf

[42] Lx_ROS, “ROS K-means 巡邏點應用,” CSDN. [Online]. Available: https://blog.csdn.net/lx_ros/article/details/134796198

[43] Chih-Sheng Huang, “機器學習：集群分析 K-means Clustering,” Medium. [Online]. Available: https://chih-sheng-huang821.medium.com/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E9%9B%86%E7%BE%A4%E5%88%86%E6%9E%90-k-means-clustering-e608a7fe1b43

[44] AXK51013, “不要再用 K-Means！DBSCAN 詳細解說,” Medium. [Online]. Available: https://axk51013.medium.com/%E4%B8%8D%E8%A6%81%E5%86%8D%E7%94%A8k-means-

%E8%B6%85%E5%AF%A6%E7%94%A8%E5%88%86%E7%BE%A4%E6%B3%95dbscan%E8%A9%B3%E8%A7%A3-a33fa287c0e

[45] Tomohiro Liu, “機器學習筆記系列 81 — 均值偏移 Mean-Shift,” Medium. [Online]. Available: https://tomohiroliu22.medium.com/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E5%AD%B8%E7%BF%92%E7%AD%86%E8%A8%98%E7%B3%BB%E5%88%97-81-%E5%9D%87%E5%80%BC%E5%81%8F%E7%A7%BB-mean-shift-7084a2ddb014

[46] Tomohiro Liu, “機器學習筆記系列 99 — 神經網路 Neural Network,” Medium. [Online]. Available: https://tomohiroliu22.medium.com/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E5%AD%B8%E7%BF%92%E7%AD%86%E8%A8%98%E7%B3%BB%E5%88%97-99-%E7%A5%9E%E7%B6%93%E7%B6%B2%E7%B5%A1-neural-network-1e7177542995

[47] “Day2: 初識 STM32 – iT 邦幫忙,” iT Help, Oct. 27, 2023. [Online]. Available: https://ithelp.ithome.com.tw/articles/10264112?sc=hot

[48] "Jetson AGX Xavier 系列", NVIDIA, [Online]. Available: https://www.nvidia.com/zh-tw/autonomous-machines/embedded-systems/jetson-agx-xavier/. nvidia.com

附　錄

附錄 A：使用材料與軟體清單

A.1 電子元件與控制模組

- STM32F103C8T6：**韌體驅動核心板**
- PCA9685：**PWM 擴充板**
- JGB37-520：**行走動力系統**
- DRV8871：**馬達驅動電路**
- 9 軸 IMU：移動位置感測（SLAM）
- **HC12 遙控模組**：遠程無線通訊模組
- **PWM 調速模組**：電機調速
- **TP4056 離電池充放電模組**：充電管理系統

A.2 結構與機械零件

- 2020 客製鋁擠型材：**車架結構**
- 20 型 L 直角連接件：**車架連接**
- GT2 同步輪：**動力傳動**
- 光軸：**撿球滾輪軸**
- 蘑菇頭搖桿模組：**手動控制裝置**

A.3 感測器與影像模組

- **深度攝像機（Astra Pro 3D）**：羽球偵測
- **光達（Slamtec A1）**：自動導航感測

A.4 計算與開發平台

- NVIDIA Jetson Xavier Developer Kit：**高層運算與深度學習推理平台**

A.5 使用軟體

- **FreeCad**：用於繪製 3D 設計圖（.stl）和雷射切割檔案（.dxf）
- **VMware Workstation Pro**：虛擬機平台，用於撰寫並測試機器學習生成巡邏點
- **Ubuntu 20.04.06**：作為系統平台
- **Text Editor**：Linux 環境下撰寫程式碼
- **RViz2**：用於檢視機器人的運動與狀態
- **jtop**：檢視 GPU 耗能
- **Roboflow**：用於分析羽球圖片並建立資料集

系統需求規格書

System Requirement Specification

專 題 名 稱	ROS2 智慧型羽球自動收集機器人：結合機器學習導航與深度學習物件偵測
發 展 者	B11117009 高啟軒 B11117008 利瑋哲 B11117002 唐禮直
專題指導教師	李建緯 博士

目錄

1. 系統概述（System Description）

智慧羽球自動收集機器人系統（Intelligent Badminton Shuttlecock Collecting Robot System ，IBSCR System， Version 1.0.0）是一套高度整合的智慧化羽球收集系統，採用 ROS 2 （Robot Operating System 2） 為核心，結合感測器資料處理、即時建圖、物件辨識、自主導航與羽球收集功能，能自主巡邏場地、偵測落地羽球並完成收集任務。

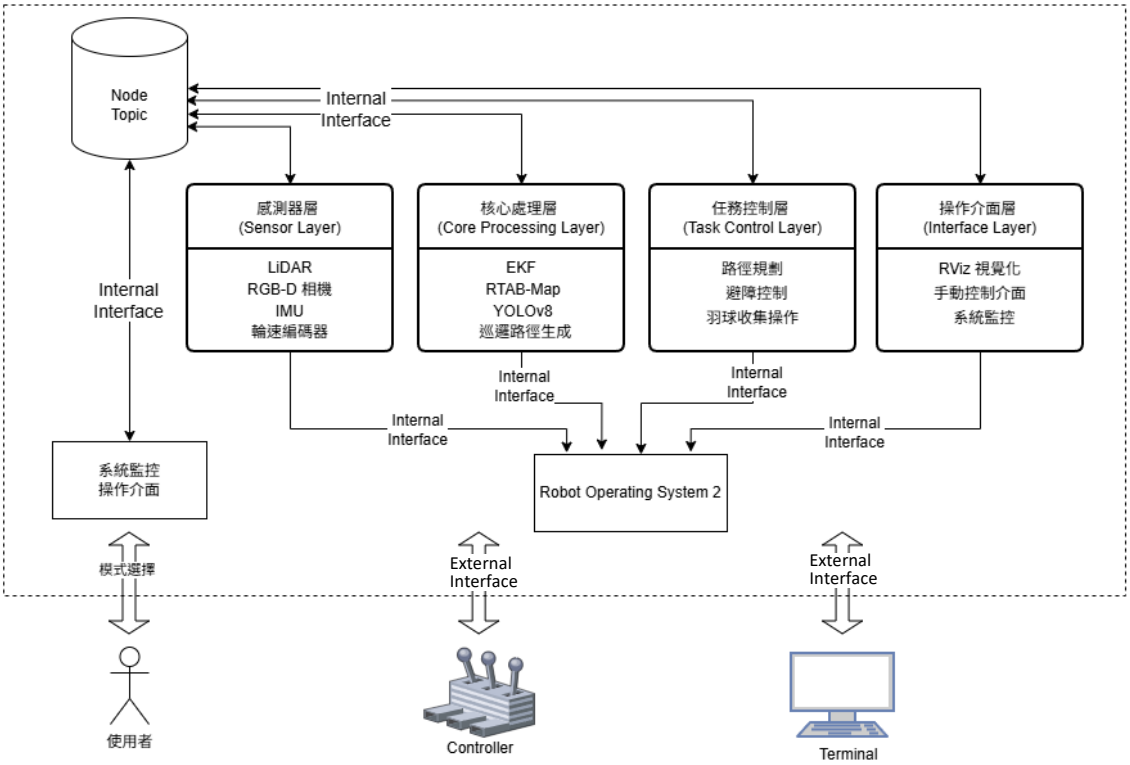


Figure 1. IBSCR 系統架構圖

1.1 多層架構設計 （Multi-layer Architecture）

本系統採 多層式 （Multi-layer） 設計架構，將不同功能模組分層管理，以提升系統的可擴充性、可維護性與可靠性：

1. 感測層 （Sensor Layer）
    - 收集來自 LiDAR、Astra RGB-D 相機、BNO055 IMU、輪速編碼器（Encoder） 的資料。
    - 負責資料前處理、校正與融合，提供定位與環境感知資訊。
2. 核心處理層 （Core Processing Layer）
    - 負責資料整合、演算法運算與決策。

- 包含 EKF（Extended Kalman Filter）感測器融合、RTAB-Map 建圖、YOLOv8 羽球辨識與 3D 座標估算、巡邏路徑生成演算法（K-means 聚類 + 基因演算法（GA））。

3. 任務控制層（Task Control Layer）
    - 控制自主巡邏與羽球收集任務。
    - 負責路徑規劃、避障、任務分派與收集操作。
4. 操作與介面層（Interface Layer）
    - 提供使用者與系統的交互介面，包括 RViz 可視化、終端手動控制（Keyboard / Joystick）與系統監控。
    - 使用者可即時觀察地圖、機器人位姿、任務狀態，並可手動干預。

1.2 系統設計理念

- 模組化（Modular Design）：各功能模組獨立封裝，透過 ROS2 Topic / Service / Parameter 交換資料，方便後續擴充感測器或演算法。
- 彈性擴充（Flexible & Scalable）：多層架構可針對不同層級修改或升級，不影響其他層運作。
- 即時性（Real-time Processing）：核心演算法與任務控制層支援即時資料處理，確保導航與收集精準性。
- 可靠性（Reliability & Maintainability）：各層分工明確，可快速定位錯誤並維護。

1.3 系統模組概覽

整個系統包含六主要子系統模組：

1. 感測器管理子系統（Sensor Management， SM 1.1.0）

    整合 LiDAR（RPLIDAR A1）、Astra RGB-D 相機、BNO055 IMU、輪速編碼器（Encoder）的資料。

    提供 定位資訊（Odometry & IMU Fusion），使用 EKF（Extended Kalman Filter）將多源感測器融合。

    ROS2 節點：lidar_a1_launch.py、sensor_control.py、ekf_filter.py、serial_listener.py、keyboard_controller.py（測試移動功能）。

2. 導航與建圖子系統（Navigation & Mapping， NM 1.2.0）

使用 RTAB-Map 進行 SLAM（Simultaneous Localization and Mapping），生成地圖並更新機器人即時位姿。

使用光達數據，並利用 Frontier（邊界）探索發送目標位置，支援 Nav2 自主導航，包括全局路徑規劃與局部避障。

地圖資料格式：robot_map.pgm / robot_map.yaml。

使用套件：navigation2（Nav2）、rtabmap_ros。

ROS2 節點：探索建圖 yolo_patrol_node.py。

3. 物件辨識子系統（Object Detection， OD 1.3.0）

利用 YOLOv8 演算法辨識羽球目標，並結合深度影像推算 3D 空間位置。

支援多目標追蹤，與導航模組協同完成收集任務。

ROS2 節點：模型（.pt）以及 yolo_map_marker.py。

使用套件：OpenCV、Pytorch（啟用 GPU 加速）。

4. 自主巡邏與收集控制子系統（Autonomous Patrol & Collect Control， PCC 1.4.0）

根據地圖生成 巡邏路徑（Patrol Path），可使用 K-Means 聚類 + Genetic Algorithm（GA）進行巡邏點最佳化。

負責羽球收集流程控制：導航至目標 → 執行收集 → 返回巡邏路線。

ROS2 節點：自動巡邏 nav2_patrol.py。

5. 系統監控與操作介面子系統（System Monitoring & Operation， SMO 1.5.0）

提供 RViz 視覺化操作介面：地圖、路徑、感測器狀態與機器人位姿。

支援 手動操作介面（Keyboard / CLI Control）與自動任務切換。

系統啟動透過 robot_start.sh 腳本統一開機啟動感測元件各節點。

程式啟動可使用指令：全自動判別（主啟動）auto_mode.launch.py/手動建圖 slam_mapping.launch.py/自主巡邏 nav2_patrol.launch.py 實現全自動或自行選擇模式執行任務，launch 檔案會向下啟動所有已加入的子節點與套包。

6. 生命週期節點子系統（LifeCycle Management， LC 1.6.0）

生命週期節點子系統（LifeCycle Node Subsystem）是整個系統的運行狀態管理核心，負責監控各個模組的生命狀態、資源使用與節點間依賴關係。
系統中各主要 ROS 2 節點皆遵循 LifeCycle Node 設計模式，包含狀態 unconfigured、inactive、active、finalized 等階段。

此模組可監控並協調各子系統啟動順序，例如先啟動感測器、再啟動導航與辨識節點，以確保依賴關係正確；同時在異常狀態下可安全重啟或關閉節點，提升整體穩定性與維護性。

1. 主要功能：節點狀態監控、模組啟停管理、故障恢復。
2. 主要套件：rclcpp_lifecycle、lifecycle_manager （Nav2 內建模組）。
3. 主要節點：lifecycle_manager_navigation、ifecycle_manager_sensors。
4. 狀態流程：unconfigured → inactive → active → finalized。
5. 功能特色：可自動偵測依賴模組狀態並觸發重啟機制，支援遠端狀態查詢與控制。

2. 功能需求（Functional Requirements）

IBSCR-F-001	整合多種感測器資料，提供定位與航向資訊。
IBSCR-F-002	即時建圖與地圖更新，支援 SLAM。
IBSCR-F-003	自主導航與避障功能。
IBSCR-F-004	羽球物件辨識與 3D 座標推算。
IBSCR-F-005	自動生成巡邏路徑，支援多種策略（zigzag， spiral，random）。
IBSCR-F-006	羽球收集任務自動執行。
IBSCR-F-007	系統狀態與地圖資料可視化（RViz）。
IBSCR-F-008	任務資料與地圖自動記錄、儲存。
IBSCR-F-009	手動操作與系統除錯介面。
IBSCR-F-010	系統啟動、關閉與異常偵測機制。

3. 模組介面需求（Module Iterface Requirement）

IBSCR-I-001	感測器資料接收介面，資料格式需統一 （Odometry， IMU，Camera）。
IBSCR-I-002	使用者操作介面，支援鍵盤與 GUI 控制。
IBSCR-I-003	導航模組參數傳遞介面（目標點、路徑、速度限制）。
IBSCR-I-004	影像資料格式轉換與同步，供 YOLO 與建圖使用。

IBSCR-I-005	模組狀態透過 ROS2 Topic / Service 發布與訂閱。

4. 非功能需求（Non-functional Requirements）

IBSCR-N-001	感測器資料更新頻率 ≥ 30Hz。
IBSCR-N-002	導航決策與避障反應時間 ≤ 0.5 秒。
IBSCR-N-003	系統啟動至任務可執行時間 ≤ 60 秒。
IBSCR-N-004	地圖資料載入與儲存時間 ≤ 10 秒。

5. 設計與實作限制（Design and Implementation Constrains）

IBSCR-D-001	開發環境 ROS2 Galactic + Python 3.8~3.10。
IBSCR-D-002	建圖與導航使用 RTAB-Map + Nav2。
IBSCR-D-003	感測器支援 Astra RGB-D、RPLIDAR A1、BNO055、Encoder。
IBSCR-D-004	物件辨識使用 YOLOv8。
IBSCR-D-005	系統執行於 AI 開發版（NVIDIA Jetson AGX Xavier）平台。

6. 操作概念（Operational Concept）

智慧羽球自動收集機器人系統（IBSCR System）

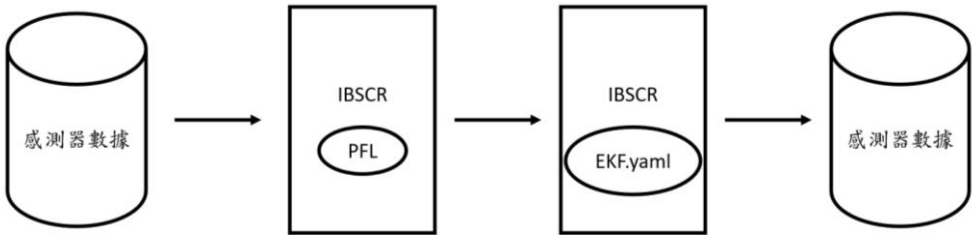
主要提供自動化巡邏、羽球偵測、目標收集與地圖管理等功能，整體運作流程如以下所述。

當系統啟動時，AI 開發版會自動開啟 ROS 2 環境並啟動各感測模組，包括 LiDAR 雷射測距儀、Astra RGB-D 深度相機、BNO055 姿態感測器（IMU）以及 編碼器（Encoder）。

此時感測層（Sensing Layer）會開始持續蒐集環境資訊與機器人姿態資料。

所獲得的感測資料經由資料處理層（Perception & Fusion Layer）進行同步與濾波，並透過 EKF 節點進行多源資料融合，生成實時位置與方向估計資訊。

這些資料被傳送至決策層（Decision Layer）以作為巡邏與偵測任務的依據，如圖示。

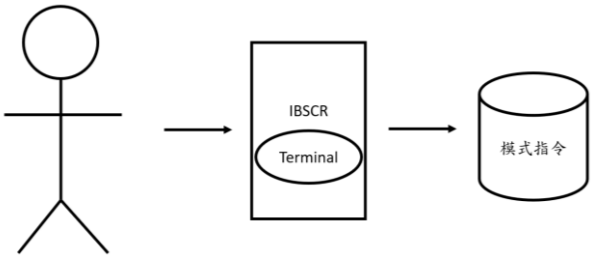


系統啟動後，操作人員可透過終端控制介面（Terminal Interface）選擇「自動/探索/巡邏模式」三種模式中任一項啟動檔案執行任務。

自動模式下，ROS2 系統會檢測地圖存檔位置是否存在已儲存的地圖檔案，並啟動探索建圖模式或是巡邏模式。

探索建圖模式被啟動，系統即會利用 RTAB-Map 建立環境地圖，並以 Nav2 框架 執行即時導航與避障任務，或是關閉 yolo_patrol_node.py 關閉自動探索並搭配實體遙控器或是 keyboard_controller.py 搭配鍵盤移動機器人，同時建立地圖。

巡邏模式期間，物件辨識模組（Object Detection Subsystem）將持續啟動 YOLOv8 物件辨識模型，即時判斷畫面中是否出現羽球。若辨識到羽球影像，系統會自動取樣其深度資料，並根據相機內部參數計算羽球於世界座標系中的三維位置。



一旦目標位置確定，決策層會根據當前地圖資訊生成收集路徑，導航子系統（Navigation Subsystem）會以最短路徑與避障演算法控制機器人前往目標。
到達羽球位置後，執行層（Execution Layer）會控制馬達與收集機構啟動收集程序。

收集動作完成後，機器人將自動回到巡邏主路線，並繼續進行下一段區域巡邏，確保整個場地皆被覆蓋。

整個巡邏與收集過程中，所有感測資料、位姿資訊及任務狀態皆被即時記錄於系統資料庫中，以供後續統計分析與報告使用。

在作業結束時，系統會自動儲存目前地圖、收集紀錄及路徑資訊，供操作人員於下一次啟動時載入，達到「任務連續執行與學習更新」的效果。

7. 使用者介面設計（User Interface Design）

（1） RViz 控制介面

顯示地圖、路徑、感測器狀態、機器人姿態以及已訂閱的節點內容狀態。

提供圖像式操作介面，可以手動設置機器人起始位置。

（2） 終端控制介面 / 手動控制說明

終端機啟動任務指令：

[自動啟動] → 啟動 auto_mode.launch.py 節點自動判斷建圖或巡邏。

[開始建圖] → 啟動 slam_mapping.launch.py 節點自主探索未知區域。

[導航巡邏] → 啟動 nav2_patrol.launch.py 節點自主巡邏與收集。

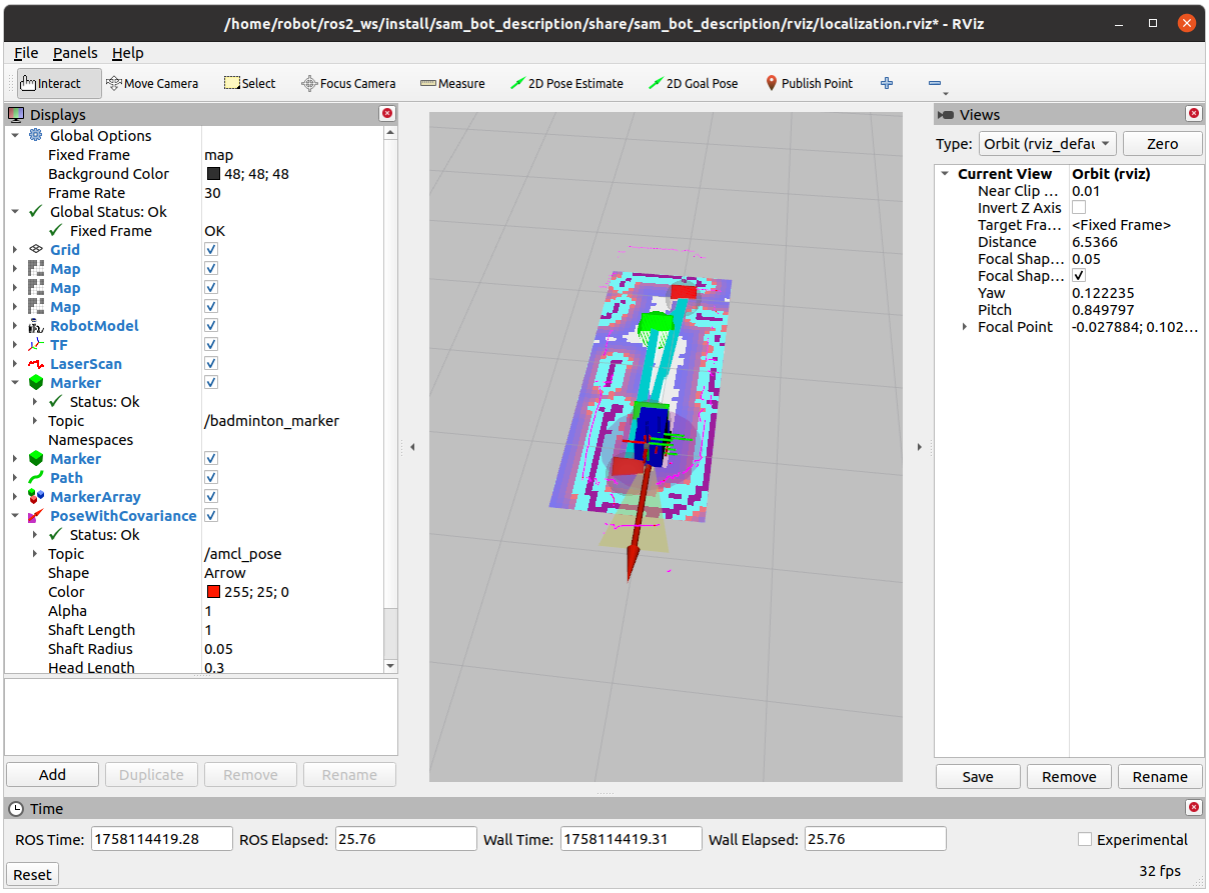


Figure 2. RViz 控制介面

手動控制：

- 主要控制方式：實體搖桿 （Physical Joystick）
  - 使用者可透過搖桿直接操作機器人，控制 STM32 韌體驅動馬達（PWM） 及方向，實現前進、後退、旋轉與速度調整。
  - 搖桿訊號透過 UART 傳送至 STM32，STM32 根據指令控制馬達驅動。
  - 此方式為實際任務中主要的手動控制方法，適用於場地測試或特定任務干預。
- 測試控制方式：鍵盤 （Keyboard Control， CLI）
  - 鍵盤控制主要用於軟體測試或系統除錯。
  - 使用者可透過 keyboard_controller.py 節點模擬搖桿操作，發送 ROS2 指令至控制模組。
  - 此方式不作為實際任務主控，但便於功能驗證與開發調試。