2024年臺灣國際科學展覽會
優勝作品專輯

作品編號　100035

參展科別　工程學

作品名稱　基於深度學習與自動化技術降低桌球硬體需求之開發

得獎獎項　耶魯科學與工程學會獎

就讀學校　臺北市立建國高級中學
指導教師　柯坤呈、劉翠華
作者姓名　陳泓菖、柯又恩

關鍵詞　深度學習、自動化技術、桌球機器人教練

作者簡介

我們是目前就讀建國中學高二普通班的陳泓菖（左）和柯又恩（右）。基於對科學、工程的興趣與熱忱，因而投入大量精力於本次的專題探究。過程中，我們從理論到實踐，遇到了大大小小的問題，也一步一步地找到解決方案，沉浸在不斷的試錯與學習中，從中得到了不少的樂趣與成就感。感謝所有辛苦指導我們的教授、老師與學長們，無償地給予我們的幫助，讓我們有機會能夠向更遠的道路前進。

摘要

桌球機器人技術長期以來一直受到技術和成本挑戰的制約，限制了其廣泛應用。深度學習和自動化技術的發展提供了新機會，特別是長短時記憶（LSTM）模型的應用，可有效強化桌球軌跡時域資訊預測結果，並同時減少硬件要求，提高成本效益。

此研究強調軟體創新，透過程式控制與機器視覺，達到多目標辨識與硬體協作之功用，同時使用易得之零部件構建硬體，進而降低成本。不僅如此，本研究亦克服多個技術挑戰，包括相機校正、空間定位、乒乓球追蹤、模型訓練、機器人控制等相關議題，並在最後完成建造低成本桌球協作機器人，從而達到技術突破與實務應用之可行性，為機器人技術在娛樂、體育等領域帶來新發展潛能。總結而言，本研究透過深度學習和自動化技術之應用，為該領域帶來學術和實際創新價值，開創未來之嶄新發展可行性，並提供實務應用創新展現。

ABSTRACT

The table tennis robot technology has long been constrained by technical and cost challenges, limiting its widespread application. The advancement of deep learning and automation technologies has offered new opportunities, particularly the application of Long Short-Term Memory (LSTM) models, which effectively enhance the prediction of table tennis ball trajectories in the temporal domain while simultaneously reducing hardware requirements, thus improving cost-effectiveness.

This study emphasizes software innovation, achieving multi-object recognition, and hardware collaboration through program control and computer vision. It utilizes readily available components to construct hardware, thereby reducing costs. Furthermore, this research addresses various technical challenges, including camera calibration, spatial localization, ping-pong ball tracking, model training, and robot control. Ultimately, it culminates in the creation of a low-cost table tennis collaborative robot, demonstrating the feasibility of technological breakthroughs and practical applications. This development holds great potential for the integration of robot technology in entertainment, sports, and various other domains. In summary, this study, through the application of deep learning and automation technologies, brings both academic and practical innovation value to the field, paving the way for new possibilities in future development and showcasing innovative, practical applications.

壹、前言

一、研究動機

桌球是一項很需要技巧的運動，往往要在雙方實力相當的時候才能有較好的體驗，因此找到一個好球友並不是一個簡單的事。但如果，球友不是人類呢？現在市面上有各式各樣的桌球發球機，可以朝不同位置，射出不同速度、旋轉方向的球，而且精度其實也不錯，但是發球機往往只是把蒐集好的一袋球射出去，而不是把球打回去，並不符合桌球的規則，也少了點樂趣。

2013 年，Omron 開始了一個桌球機器人教練專案 Forpheus[1]，成為世界上第一個桌球機器教練，可以用兩個 80fps 的相機預測出球的軌跡，並透過設計的手臂連接球拍對人類打向機器的球做出反擊。到目前為止已經有六代的 Forpheus。我們初次看到 Forpheus 的時候可以說是嘆為觀止，有一股想要自己動手做一台的衝動。進一步研究發現 Omron 有公開部份的實現方法，不幸的是其對於設備的規格要求偏高，有許多都是工業級的設備，主要有高速攝影機以及高階的擊球機構，成為我們建構桌球教練機器人最大的阻礙。

然而，隨著深度學習 LSTM 架構的興起，我們認為其對於桌球軌跡預測的任務再適合不過了。人類的眼睛完全比不上高速攝影機，但是有時卻能比機器更準確地預測桌球的軌跡，我們認為背後的祕密在

於我們大腦處理訊息的方式，並非如機器般按照死板的演算法，而是如深度學習般靈活的運用每一筆輸入資料進行合理且迅速的預測。

於是我們希望可以透過我們設計的基於 LSTM 的新方法，降低桌球機器教練對硬體設備的需求，用身邊簡單的設備(手機相機、57步進馬達、伺服馬達、線性滑軌.....)，做出一台低配版桌球機器人，讓一般人也能體驗到與機器人對打桌球，甚至有能力的人也可以自己做一台。

二、研究目的與研究問題

為做出一台可以與人對打桌球的機器人，所需解決的問題有以下，作為本研究之研究目的：

(一) 透過 AprilTag 獲取相機位置和姿態與單應性矩陣

(二) 以 OpenCV 獲取球的像點座標

(三) 透過球的像點座標、單應矩陣與相機位置求經過相機和球的世界座標直線

(四) 運用 LSTM 模型對球進行動態三維重建、預測

(五) 製作可以快速接收指令並擊球的機器人

三、文獻回顧

(一) 小孔成像模型

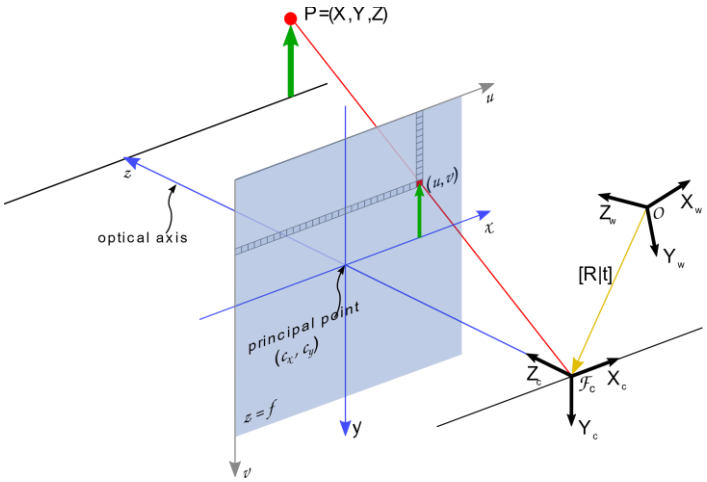


圖 1 小孔成像模型的坐標系[2]

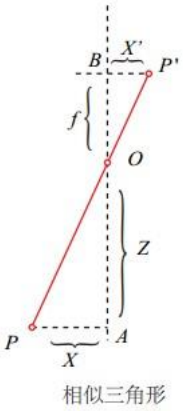


圖 2 小孔成像模型形成之相似三角形[3]

根據圖 2 形成之相似三角形，並加上齊次座標的概念，則可以將相機座標系下的點投影到像素平面(像點)，並可以用式 1 的矩陣乘積表示，左側為像點（uv），右側為相機座標系下的點（XYZ），中間的方陣就是相機內參矩陣。

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \qquad \text{式 1}$$

式 1 中的 f 取決於相機焦距和像素單位與相機座標系單位的比例關係，而 c 則是取決於相機座標系與像素座標系的原點的平移。因此，相機內參會隨著不同的相機構造有所差異，但是不會因為相機的位置與姿態而有所改變，一個相機只會有一個固定的相機內部參數。

相機座標系到世界座標系的轉換需要經過旋轉、平移，可由式 2 表示：

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \quad \text{式 2}$$

r 是旋轉矩陣、t 是平移向量，加入齊次座標的概念可表示為式(3)：

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad \text{式 3}$$

中間的方陣就是相機外部參數矩陣，會隨著相機位置與姿態不同而有改變；相反的，也可以透過相機外參得知相機的位置與姿態。

透過結合內外參，就可以將世界座標投影到相機中實際拍到的像點。這會使變換後的點失去深度訊息(從三維座標變換維二維)，因此變換是不可逆的，但是可以透過攝影變換(單應性)把拍攝到的像點映射到世界座標係下的一個平面上。攝影變換是一個二維平面到另一個二維平面的變換，加入齊次座標的概念，該變換可以用一個 3*3 的單應性矩陣 H(式 4)表示。

$$\begin{pmatrix} x'_i \\ y'_i \\ 1 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \quad \text{式 4}$$

而單應矩陣加了一個比例因子後依然會和原本的單應矩陣等價

$$x'_i = \frac{ah_{11}x_i + ah_{12}y_i + ah_{13}}{ah_{31}x_i + ah_{32}y_i + ah_{33}} = \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}} \quad \text{式 5}$$

$$y'_i = \frac{ah_{21}x_i + ah_{22}y_i + ah_{23}}{ah_{31}x_i + ah_{32}y_i + ah_{33}} = \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}} \quad \text{式 6}$$

因此，在一般的情況下，會加入 h33 = 1 的限制條件，單應性矩陣有 8 個自由度。剩下的 8 個未知數，可以由四組對應的(x,y)和(x',y')

求得。

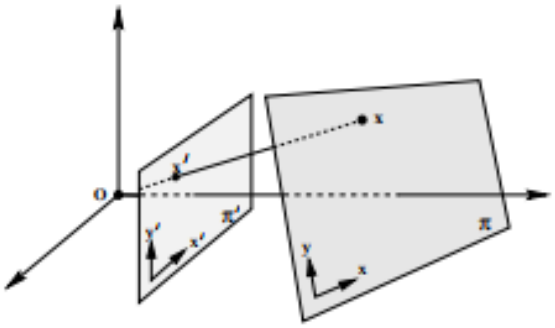
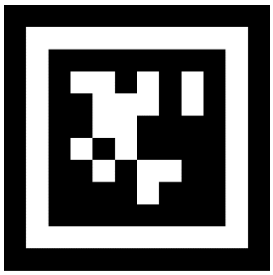


圖 3 單應性變換[4]

(二) AprilTag 定位法[5,6]



TAG36H11 - 0

圖 4 AprilTag (OpenMV 生成)

AprilTag 常用在相機校準、AR、機器人定位上。使用方法：將 AprilTag 影印下來，並放在場景中的一個平面上，然後對該 tag 進行攝影，透過一系列影像處理演算法，可以獲取該 tag 的四個角落在該影像的像點座標以及該 tag 所攜帶的 ID 號碼。

我們可以設一個基於 AprilTag 的座標系，x、z 軸分別平行於 AprilTag 的兩個邊，y 軸指出 AprilTag，原點在 AprilTag 的中心，並測量 AprilTag 的實際長度為單位。這樣就可以得到 AprilTag 的四個角落在 y=0 平面上四個點與在攝影平面拍攝到的四個像點的對應關係。透過四個相對應的點，可以求得兩個平面的單應性矩陣。

結合單應性矩陣、相機內參與 AprilTag 的實際長度，就可以求得相機位置與姿態[7]：把 AprilTag 座標系(三維)下 z = 0 平面下的點透過旋轉矩陣 R 和平移向量 T 變換到相機座標系(三維)，再將相機座標系投影到像素座標系(二維)，其變換如下：

$$s\begin{bmatrix}u\\v\\1\end{bmatrix} = M\begin{bmatrix}R_{T_k}^{C} & T_{T_k}^{C}\\ 0^T & 1\end{bmatrix}\begin{bmatrix}X_{T_k}\\Y_{T_k}\\Z_{T_k}\\1\end{bmatrix} \qquad \text{式 7}$$

M 為相機內參、s 為比例因子。其中因為 z = 0 平面上的點的 z 座標皆為 0，因此可以把旋轉矩陣中的第三列跟 z = 0 平面上的點的 z 座標刪除，可得下式 8：

$$s\begin{bmatrix}\tilde{u}\\\tilde{v}\\1\end{bmatrix} = M\begin{bmatrix}r_{11} & r_{12} & t_{13}\\ r_{21} & r_{22} & t_{23}\\ r_{31} & r_{32} & t_{33}\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}X_{T_k}\\Y_{T_k}\\1\end{bmatrix} \qquad \text{式 8}$$

可以把上式中的相機內參矩陣與截斷的外參矩陣相乘，得到的 3*3 矩陣與單應矩陣是等價的，可藉由以下等式求得：

$$\begin{aligned} & s\begin{bmatrix}h_{11} & h_{12} & h_{13}\\ h_{21} & h_{22} & h_{23}\\ h_{31} & h_{32} & h_{33}\end{bmatrix}\\ &= \begin{bmatrix}f_x & 0 & c_x\,0\\ 0 & f_y & c_y 0\\ 0 & 0 & 1\;0\end{bmatrix}\begin{bmatrix}r_{11} & r_{12} & t_{13}\\ r_{21} & r_{22} & t_{23}\\ r_{31} & r_{32} & t_{33}\\ 0 & 0 & 1\end{bmatrix} \qquad \text{式 9}\\ &= \begin{bmatrix}f_x r_{11}+c_x r_{31} & f_x r_{12}+c_x r_{32} & f_x t_{13}+c_x t_{33}\\ f_y r_{21}+c_y r_{31} & f_y r_{22}+c_y r_{32} & f_y t_{23}+c_y t_{33}\\ r_{31} & r_{32} & t_{33}\end{bmatrix}\end{aligned}$$

式 9 中 h、f、c 已知，r、t、s 未知，先求出 r、t，在藉由旋轉矩陣行

向量模長等於一的特性計算出 s，最後，被截斷的 r_{13}、r_{23}、r_{33}、可藉由旋轉矩陣正交的特性求得，即可得出完整的相機位置與姿態。

AprilTag 有很多種不同的 family，每個 family 對應到一個解碼/解碼方式、辨識成功率，不同的 family 能夠攜帶的信息量也不一樣，各個 family 有著其適合的場合，其中，tag36h11 是最普遍使用的，也是適用性最高的一個 family，其可攜帶的 ID 範圍從 0 到 586。

(三) 循環神經網路 (Recurrent Neural Network, RNN)[8]

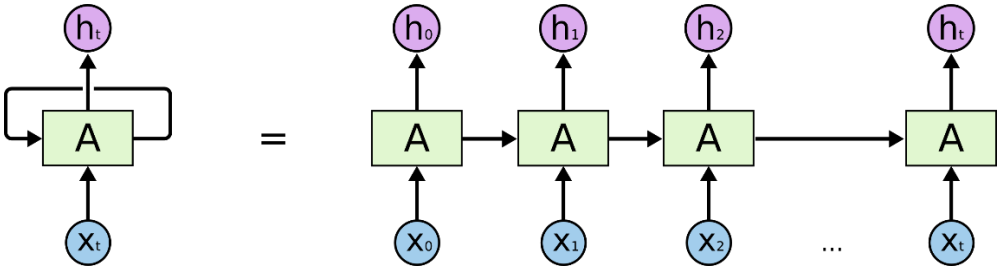


圖 5 RNN 示意圖[9]

循環神經網路是一個遞歸的模型，會將上一次的輸出結合新的輸入，經過神經網路產生出新的輸出。透過其遞歸的特性，RNN 的輸入、輸出可以是多個向量的序列，每一次的輸出除了會參考該次的輸入，還會額外參考上一次的輸出，使模型具有"記憶"。

但是，在一個序列輸入中，前面的資料對後面的輸出的影響力會隨著距離而慢慢減弱，導致 RNN 只能有"短期"的記憶，難以完成需要"長期記憶"的工作。

(四) 長短期記憶網絡 (Long Short-Term Memory, LSTM)[10]

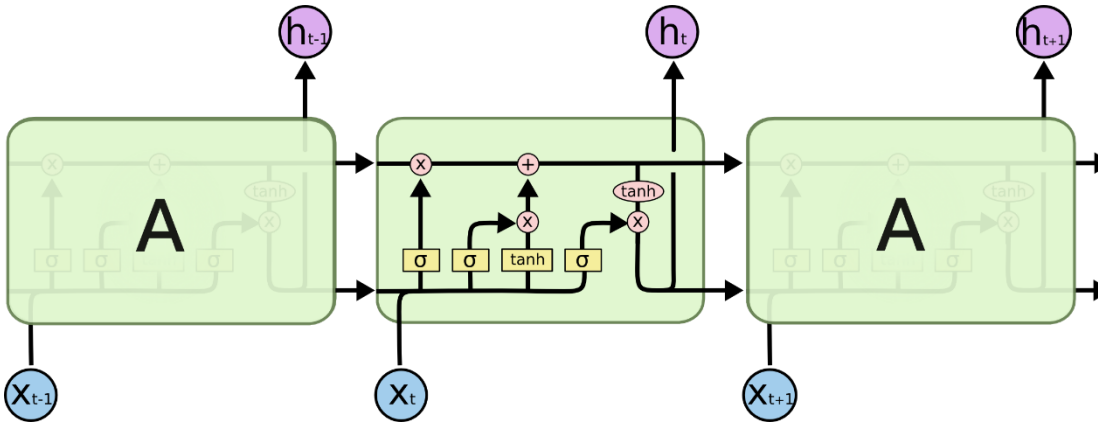


圖 6 LSTM 示意圖 [9]

LSTM 是 RNN 的一種變形，新增了單元狀態(cell state)的概念，並設計出不同的門(gate)控制單元狀態，進而解決傳統 RNN 對於長期記憶任務的問題。在 LSTM 中，會同時參考上一次的輸出、上一次的單元狀態還有新的輸入，並產生出新的輸出、單元狀態。

貳、研究設備及器材

一、硬體

(一) 機器人

(1) 57 步進馬達(57BYGH56)*2

(2) 伺服馬達(LD-27MG)

(3) 步進/伺服馬達固定架

(4) MGW12-H 線性滑軌 (140cm)

(5) Raspberry Pi 4 Model B

(6) 手機支架*2

(7) 皮帶、皮帶輪、螺絲、軸承

(8) 自研發木板拍面 (雷射切割 3mm 木板)

(9) 2040 鋁擠 (150cm)

(二) 電腦

(1) CPU：intel Core i7-12700H 2.30GHz

(2) GPU：NVIDIA GeForce RTX 3070 8GB GDDR6

(3) RAM：24GB DDR5-4800

(4) 儲存：1536GB SSD

(三) 鏡頭（手機）

(1) iPhone 12 pro max

(2) iPhone 13

二、軟體

(一) DroidCam (IOS Server: 1.9.3/Windows Client: 6.5.2)

(二) Visual Studio Code

(三) Python 3.10.5

(1) PyTorch 1.13.0(gpu)

(2) PyBullet 3.2.5

(3) matplotlib 3.5.2

(4) NumPy 1.22.4

(5) OpenCV 4.6.0.66

參、研究方法或過程

對於球的實時追蹤與軌跡預測，已有不少針對該問題進行的研究，主要將問題分成三個子問題：

1. 追蹤球在相機拍攝到的影像中的二維座標
2. 對球的二維座標進行三維重建(二維像素平面到三維世界座標)
3. 從已知的球的位置中(三維座標)預測球的落點、軌跡

對應的解決方法有下：

1. 計算機視覺算法；深度學習
2. 單一攝影機：面積；雙攝影機：幾何約束
3. 建立物理模型；深度學習

對於追蹤球的問題，其他研究所使用的解決方法主要可以分成兩類：傳統的機器視覺與深度學習。傳統的機器視覺解法，大致上包括顏色偵測、邊緣檢測、動態檢測、圖形檢測，在[11]中所使用的方法更是將追蹤準確率經過一些參數的微調後提升到 96%，本研究中參考了[11]的追蹤球方法；而深度學習運用在追蹤球的問題上，各個研究所用的方法五花八門，主要有捲積神經網路(CNN)[12]、循環神經網路，若要以深度學習完成軌跡追蹤的任務，前期的資料蒐集、模型建構會比用傳統機器視覺算法複雜不少，而後期在程式的運行上也很難避免的容易產生效能問題。

而三維重建的方法主要可以分為單相機視覺[13]與多相機視覺[11,12]，多相機視覺的三維重建大致上是將多個相機視角所拍攝到的影像，基於幾何條件約束對一個點重建出桌球的三維座標，多相機視覺雖然對預測精度可以有效提升，但往往會衍伸出多部相機的同步問題;而單相機視覺的三維重建在文獻[13]中是透過拍攝影像的乒乓球面積換算深度訊息，這種方法雖相對雙相機視覺降低了不少複雜度，也解決了兩個相機間同步的問題，但是面積計算、快門時間所造成的影像殘影往往會對計算結果造成更大的誤差，在精確度上，可能不及雙相機視覺。

至於軌跡預測，傳統的方法是以物理學模型，對三維重建後的桌球世界座標序列進行曲線的擬和，預測出球未來的軌跡；而[11]中深度學習得方式則是主要透過長短期記憶網絡(LSTM)模型進行，透過給定已知的球的三維世界座標序列作為輸出，相對於傳統的方法更加簡單、直接，並且可以透過訓練資料的增加而使模型更加準確，有較佳的穩健性。

由於本實驗的目標是要降低桌球機器人的硬體配置需求，所以我們必須面對降低硬體配置所會造成的誤差，也要留更多的機器反應時間給低配版的手臂完成擊球的動作。首先，我們打算用手機相機取代高速攝影機，迎面而來的就是較低的幀率與較高的延遲還有

兩相機的同步問題，這會造成兩個相機之間幀與幀的時間差擴大，對於高速移動的物體，用傳統方法進行三維重建可能會產生較大的誤差，於是，我們設計了一個基於 LSTM 模型預測球的路徑的新方法(圖 7)，透過將三維重建與軌跡預測兩個任務由一個神經網路處理，並將時間差、誤差加入訓練資料中，讓模型可以學到處理時間差、誤差的方法。

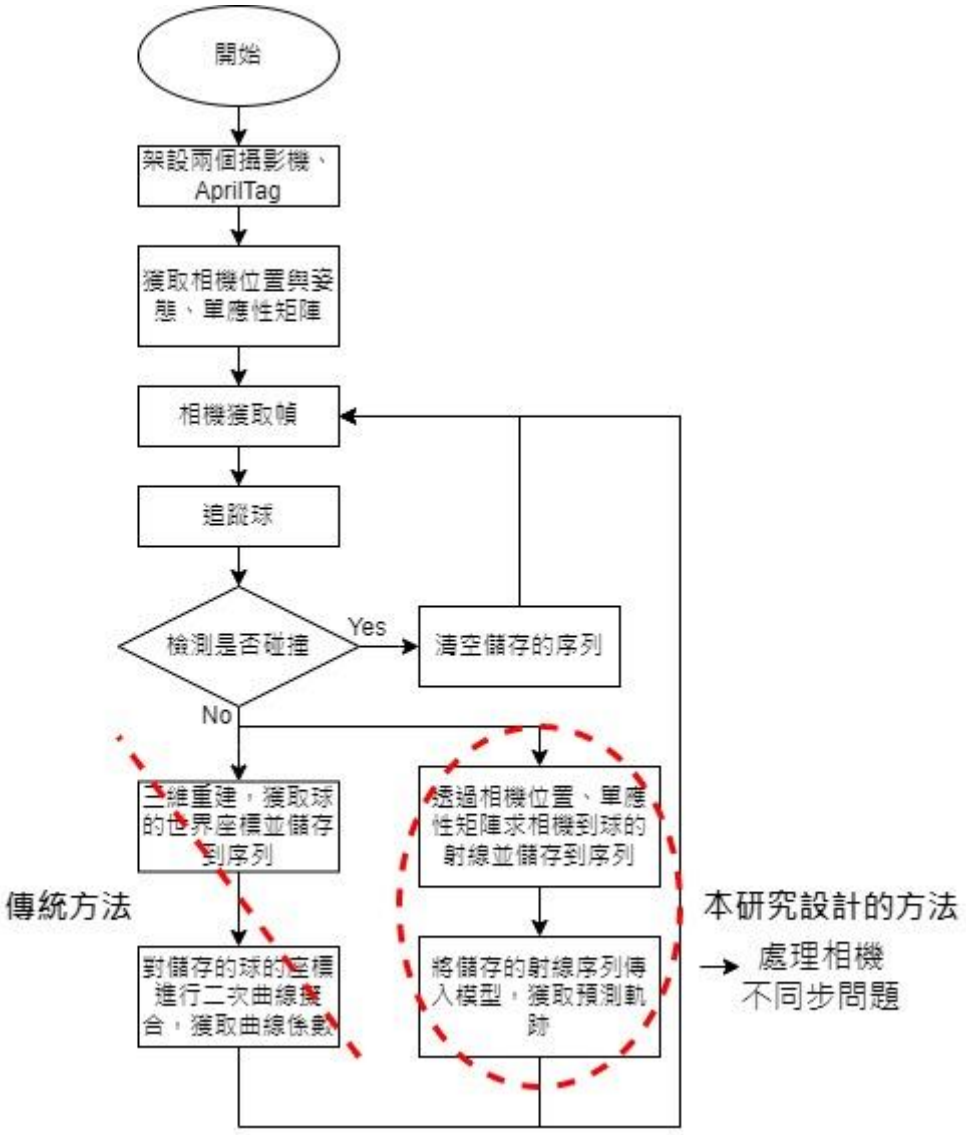


圖 7 實驗流程圖

(一) 相機定位

1. 獲取相機內參：以 OpenCV 的 Camera Calibration[14]方法，透過張正友標定法獲取各個相機內參矩陣。

2. AprilTag 定位：

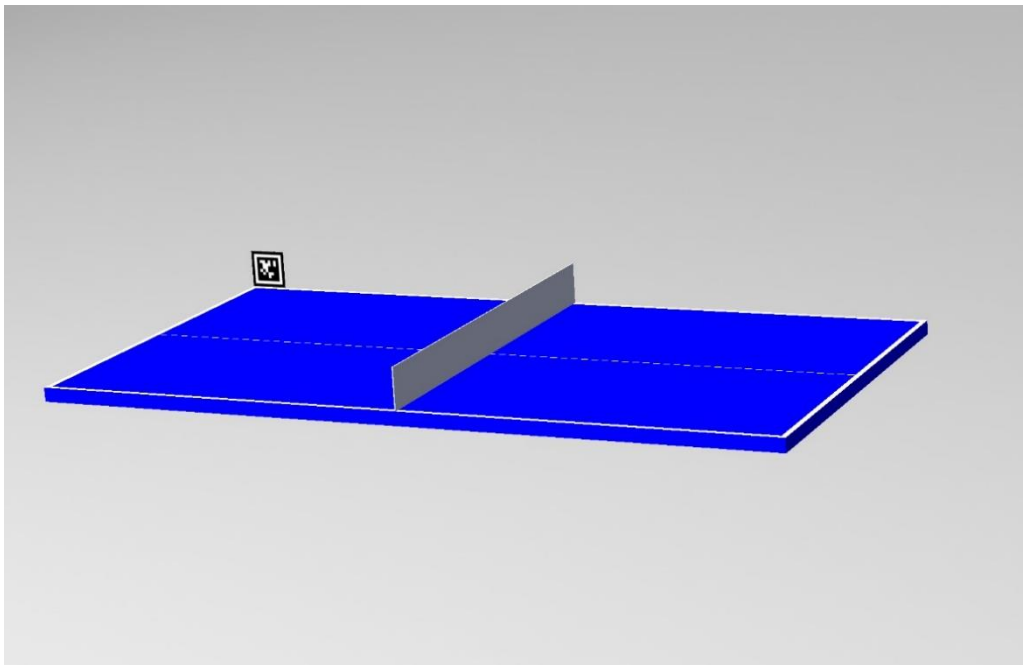


圖 8 AprilTag 放置位置圖 (由 SolidWorks 產生)

將一個 AprilTag 垂直放置於桌球桌的一個角落，並定義桌面中點為世界座標系的原點，x 軸指向桌面長邊，z 軸指出桌面，y 軸指向桌面短邊。接著，將兩個相機分別架在面對 AprilTag 的另一側(如圖 8 視角位置)。將兩個相機拍攝到的影像藉由 python 模組”pupil-apriltags” 的 Detector 找到 AprilTag 的四個角落，並計算出相機位置姿態與所拍攝圖像到 y=1 平面上的單應性矩陣。

(二) 追蹤球

1. 目的：找出乒乓球在拍攝幀的中心點位置

2. 方法：

(1) 顏色追蹤：找出和桌球顏色相近的像素。

(2) 動態追蹤：找出顏色有變化的像素。在以往的文獻中是將相機獲取到的新的幀與第一幀相減，再做二值化。我們也有試著以 xor 運算子代替相減進行上述的動態追蹤，但是我們發現若使用的是躁點較大的相機，會造成動態追蹤後的影像破碎如圖 10，於是我們設計了新的動態追蹤演算法，刪除靜止不動的目標。

(3) 篩選桌球位置：從多個可能的結果中篩選出最有可能為桌球的目標。一開始我們單純用面積大小來篩選，只考慮和設定面積最接近的結果，但這產生了兩個問題：第一是桌球的遠近對面積的影響大，第二是有時只追蹤到球的一小部分，使面積過小，於是我們在上一幀中有追蹤到球時，優先考慮和上一幀中心座標最相近的。另外，由於球拍或人的顏色和桌球相近，且也在移動，常常會使程式誤認為是桌球，於是我們加入了面積閾值的限制，並手動畫出一個範圍如圖 11，只考慮範圍內的

目標，將人或是球拍排除在外。

以下圖 9 是我們最終追蹤球程式的演算法：

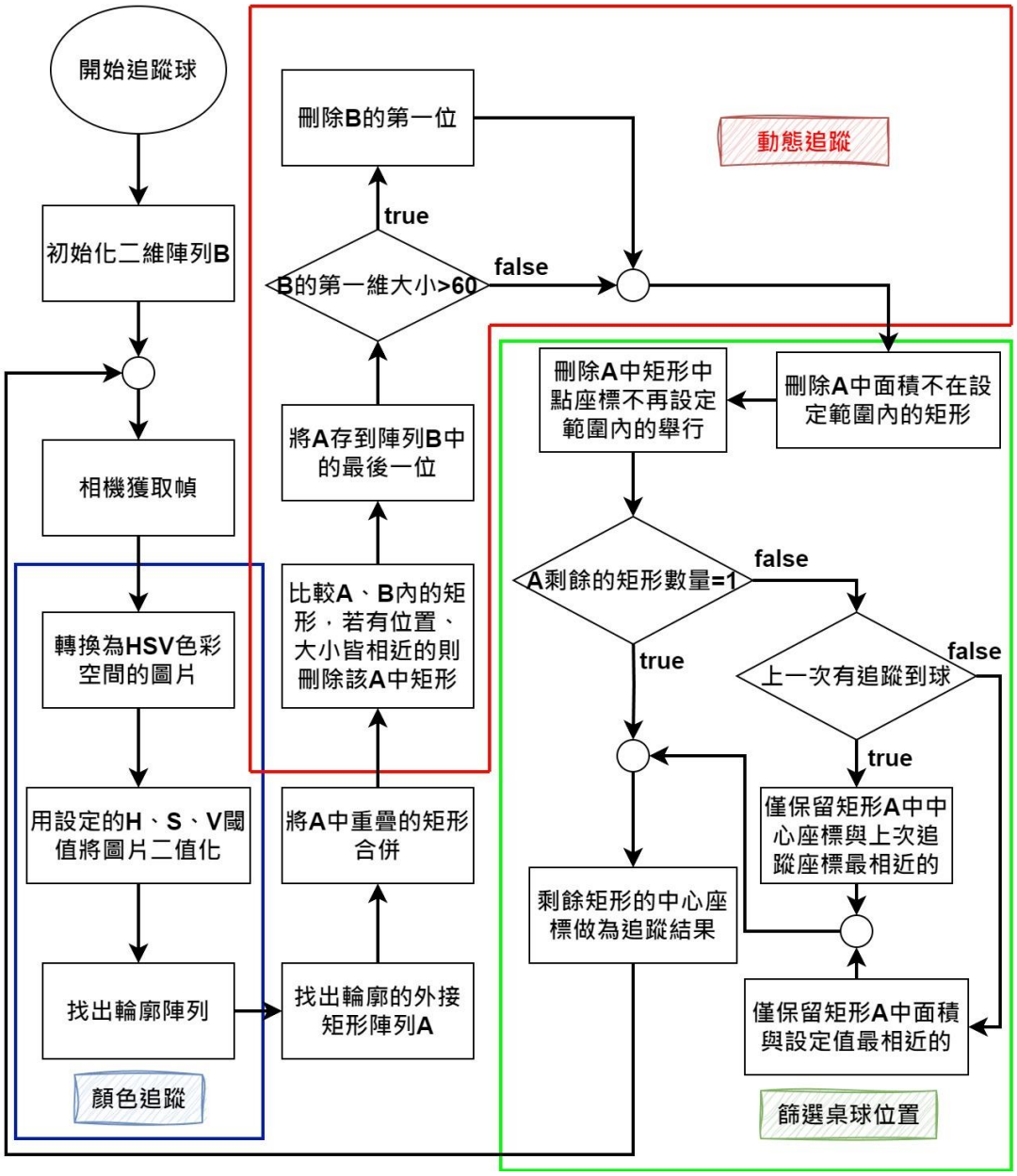


圖 9 桌球追蹤演算法流程圖

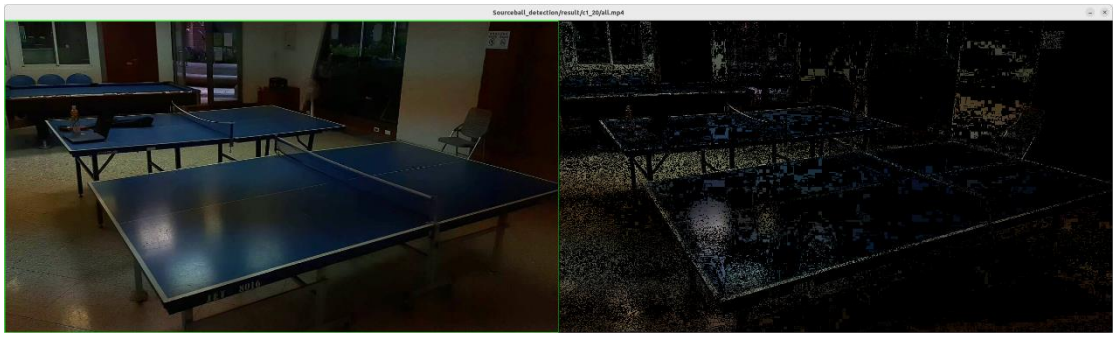


圖 10 破碎的動態追蹤結果(右方為動態追蹤結果)

圖 11 手動畫出桌球追蹤範圍(只考慮多邊形內的目標)

(三) 軌跡預測

1. 深度學習模型設計：

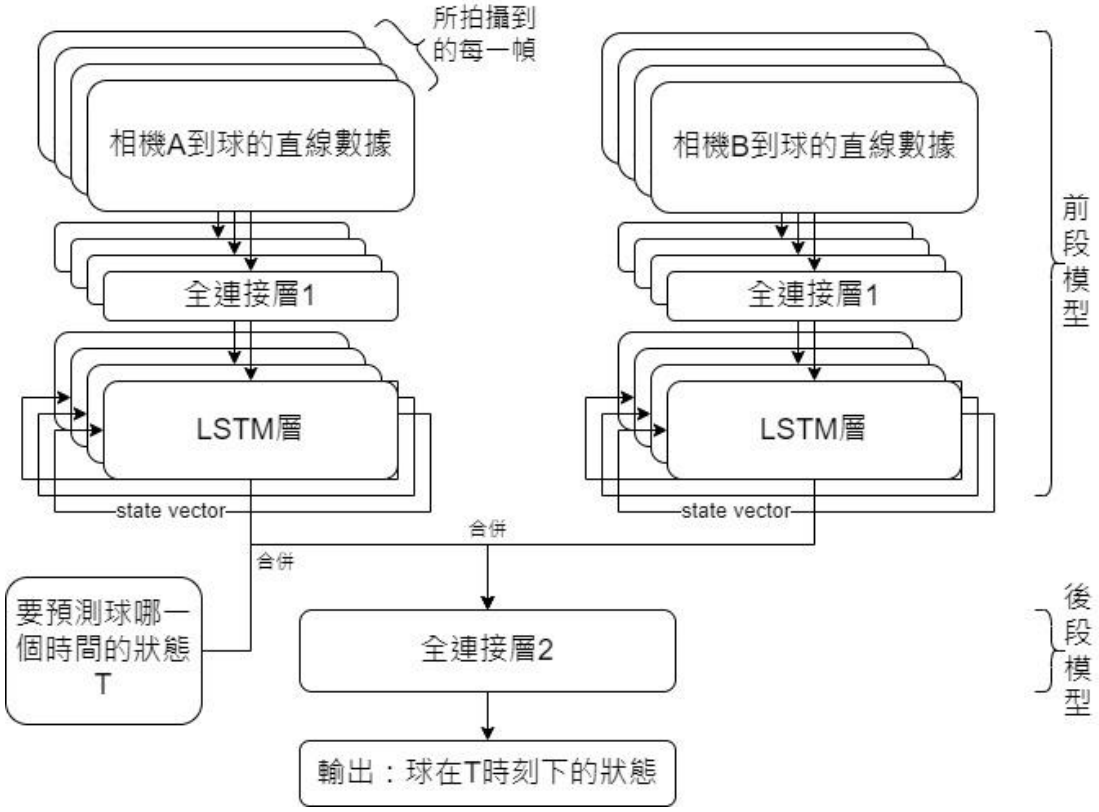


圖 12 模型架構圖

我們設計的軌跡預測模型分為兩個部分：前段與後段。前段包含 LSTM，將兩組經過相機和球的世界座標直線序列分別傳入前段模型中，取出最後 LSTM 輸出的兩個 vector 並合併為一個 vector。將該 vector 與所求的未來的時間點 t 做為後段模型的輸入，最後由後段模型輸出所求時間點下預測中球的世界座標，可將後段模型視為預測出來的球的軌跡的時間函數。綜上所述，整體模型(結合前段、後段模型)的輸入包含兩組經過相機和球的直線序列、所求的時間點陣列；輸出包含預測的球的座標陣列(與輸入的所求的時間點陣列相對應)。其中，

時間點從第一個相機第一幀的時間開始算，單位為毫秒；一條三維直線因模型輸入的標準化問題，是以兩個三元一次聯立方程式(式 10)

$$\begin{cases} y = ax + b \\ z = cx + d \end{cases} \qquad \text{式 10}$$

中的 a、c(直線斜率)經過 arc tangent 轉換為弧度 rxy、rxz，加上直線上的一個三維座標 x,y,z 表示。

綜上所述，模型的輸入可整理為：

$$((F_{11}, F_{12}, \dots F_{1n}), (F_{21}, F_{22}, \dots F_{2m}), T) \qquad \text{式 11}$$

其中$F_{ij} = \left(rxy_{ij},\ rxz_{ij}, x_i, y_i, z_i\right)$代表直線；$T = (t_1, t_2, \dots t_k)$為要預測的時間點序列，模型的輸出則為：

$$(P_1, P_2, \dots P_k) \qquad \text{式 12}$$

其中$P_k = (x_k, y_k, z_k)$，代表桌球當下的位置。

在實際應用中，兩個經過相機和球的直線序列長度會隨著時間增加，對軌跡的預測也會隨著序列長度改變實時的更新。

而若球的運動方向發生改變(可能是擊球了或是有其他外力介入)，則將該序列清空，重新進行新的一筆軌跡預測。

2. 深度學習模型訓練資料生成：

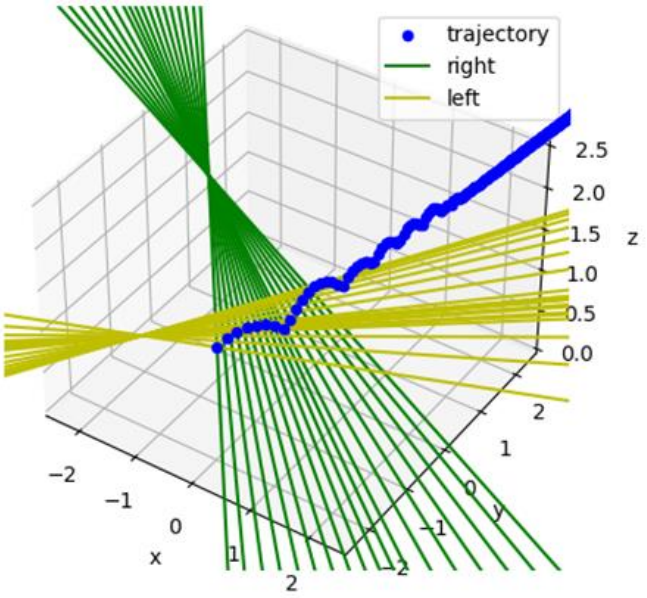


圖 13 單次桌球移動路徑模擬

透過基於 Bullet Physics SDK 的 Python 模組 PyBullet 模擬球受到重力、桌面支撐力影響下的軌跡，並隨機兩個視角，分別以大約 30fps(加入些許時間誤差)的頻率對球的位置進行異步的採樣，分別蒐集兩組經過相機和球的三維直線序列，並在其中加入些微的距離誤差模擬現實中的情形。接著，以固定的取樣頻率生成一段固定長度的時間陣列做為所求的時間點陣列。結合上面得到的兩組經過相機和球的直線序列、所求的時間點陣列就成了訓練資料輸入。最後，將所生成的時間陣列帶入 PyBullet 一開始生成的球的軌跡，得到球的真實位置作為訓練資料的答案。

3. 實際預測流程與模型推理：

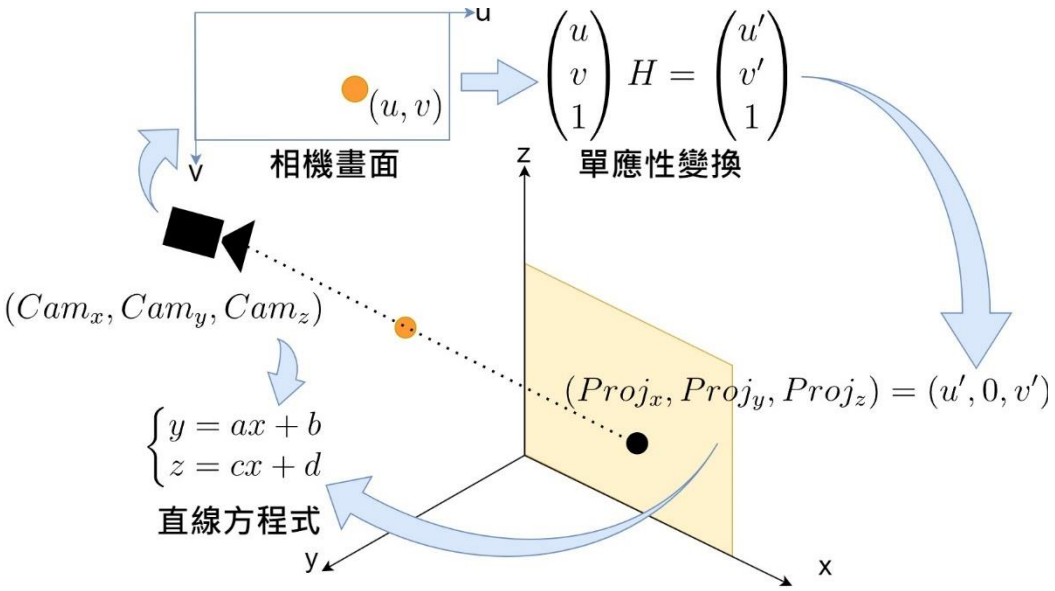


圖 14 空間與平面座標變換圖

要透過訓練好的模型進行球的軌跡預測，首先要獲取兩組經過相機和球的直線序列。相機的座標已在相機定位的時候求得；而桌球的位置，可以透過在「相機定位」中所求得的單應性矩陣對在「追蹤球」中所得到的球在畫面中的座標進行單應性變換，並將所得的座標(基於 y=0 平面下的二維座標)轉換為世界座標(設 y=0)，即可求得球投影到世界座標係下 y=0 平面上的點座標。藉由上面得到的兩個點，經過一些代數計算，就可以得到一條經過相機和球的直線的四個係數。

將兩個相機每收到新的一幀進行上述的運算，然後把得到的新的資料加到兩組經過相機和球的序列之中，並傳入前段模型，更新要給後段模型作為輸入的 vector。透過將該 vector 傳入後段模型，就可以得到預測的球的軌跡，持續對預測的軌跡進行修正。

(四) 機器手臂

1. 機器設計：

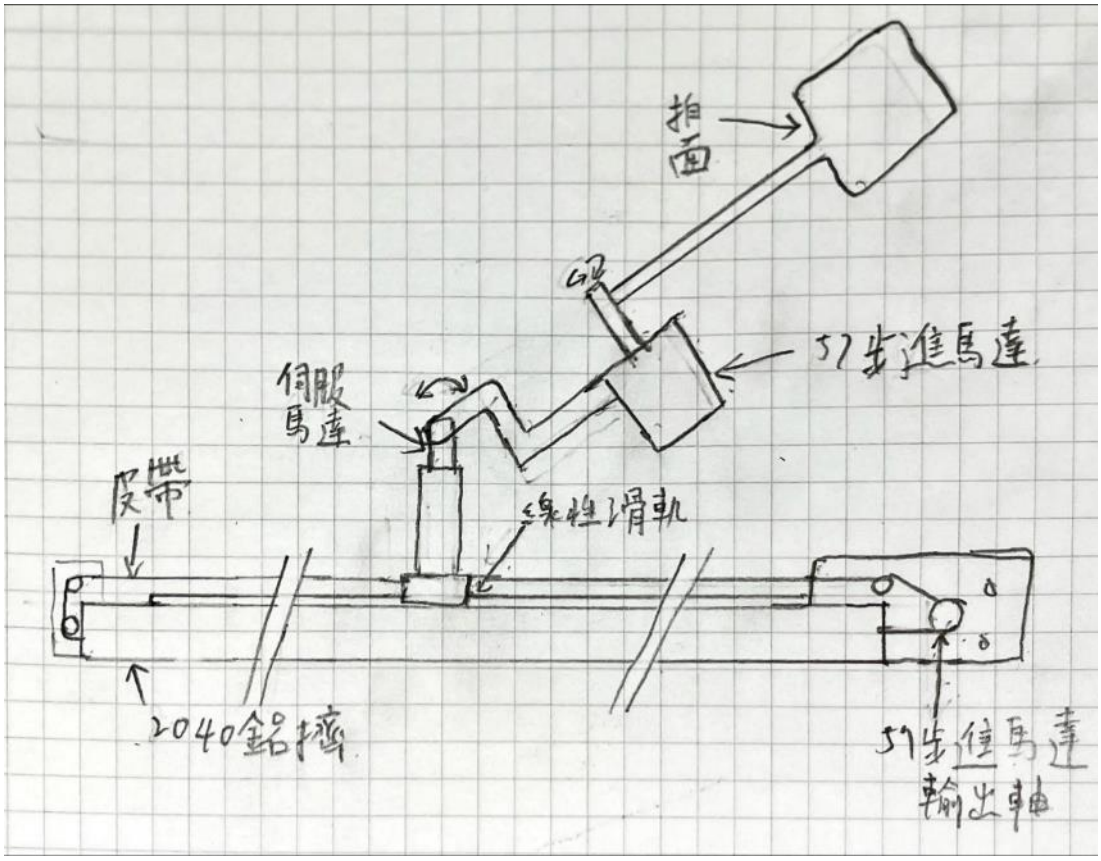


圖 15 擊打機構設計概念圖

上圖為本研究設計之擊打機構，可將其分為上半部的手臂機構與下半部的滑軌機構。

上半部的手臂機構有兩個自由度。第一個自由度由上圖中的伺服馬達控制，可以將手臂進行超過 180 度的選轉，進而配合下半步的線性滑軌瞄準不同高度的球，經果初步的計算，大約會需要 20kg · cm 以上的力矩才夠驅動手臂，所以本研究會使用 1-2 顆 240 度、力矩約 20kg · cm 的伺服馬達；第二個自由度由手臂上的 57 步進馬達控制，執行揮拍的動作，根據網路上的資料，桌球正手拍最快速率大約 7m/s，

之所以用 57 步進馬達，是因為其帶來的大約 1.2N · m 的力矩與大約 600rpm 的轉速，只要拍面到馬達的長度大於 15 公分就可以達到 7m/s 的速率，而拍面與拍面到馬達連接部分的材質因為 57 步進馬達 1N · m 的力矩可能不夠驅動真正的球拍，我們打算選用 3mm 木板代替。

下半部則是使用 MGW-12 線性滑軌，以 57 步進馬達連接皮帶帶動放置在桌球桌的底邊。

2. 機器控制

當軌跡預測的資料在電腦端計算完成，電腦將機器人要執行的動作以網路傳給 Raspberry Pi，控制負責左右移動及負責揮拍的步進馬達和調整手臂仰角的伺服馬達。

貳、研究結果與討論

一、AprilTag 獲取相機位置

我們用 DroidCam 以網路連接的方式將手機鏡頭拍攝的即時影像串流到電腦端，並利用 OpenCV 實現的張正友標定法(calibrateCamera 函式)透過不同角度拍攝棋盤的影像，計算出我們的相機在不同解析度(DroidCam 支援 640 x 480 至 1920 x 1080)下的相機內參。再將帶有 AprilTag 的灰階圖片傳給名為 pupil-apriltags 的 Python 庫的 Detector.detect 函式，找出畫面中的 AprilTag 並計算出相機外參的旋轉矩陣 R 與平移矩陣 T，其關係式如下：

$$P_{cam} = R\ P_{at} + T \qquad \text{式 13}$$

P_{cam} 是相機座標系下的點、P_{at} 是 AprilTag 座標系下的點，把相機在相機座標系下的位置 P_{cam}=(0,0,0)帶入上式即得

$$P_{at} = -R^{-1}T \qquad \text{式 14}$$

P_{at} 就式相機在世界座標系下的座標。然而，AprilTag 座標系和我們最初定義的世界座標系有些許的差異，變換後的程式碼如下：

```
def calculateCameraPosition(cameraMatrix:np.ndarray, frame_gray, tagSize=APRILTAG_SIZE) :
    try :
        detector = Detector()
        results = detector.detect(frame_gray,
                                  estimate_tag_pose=True,
                                  camera_params=(cameraMatrix[0][0],cameraMatrix[1][1],cameraMatrix[0][2],cameraMatrix[1][2]),
                                  tag_size=tagSize)
        if len(results) == 1:
            res:Detection = results[0]
            position = np.matmul(np.linalg.inv(res.pose_R), -res.pose_t)
            return equ.Point3d(position[0][0] + (tagSize/2) - 2.74/2, -position[2][0] - 1.525/2, -position[1][0] + (tagSize/2))
        else:
            return None
    except Exception as e:
        print(e)
        return None
```

圖 16 座標變換程式碼

二、追蹤球

(一) 獲取桌球顏色(HSV)範圍

在不同的環境下，影像中桌球的顏色會有些許的不同，因此我們設計了程式讓使用者可以在使用前微調追蹤的顏色範圍，使用介面如下：

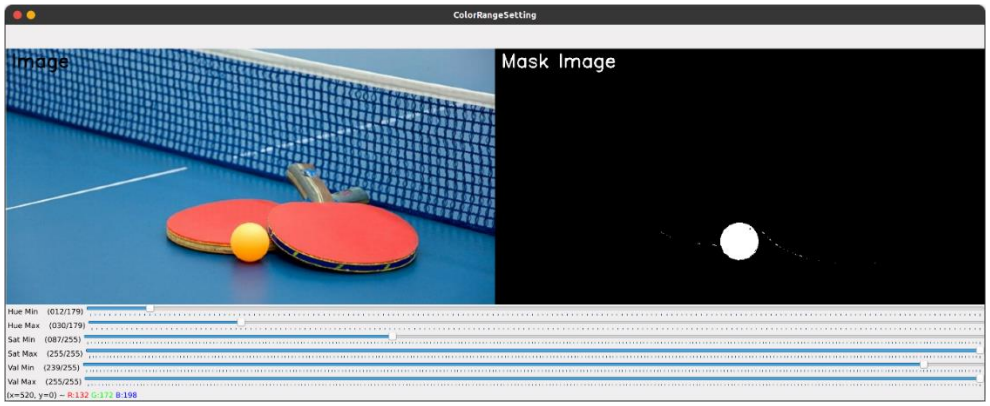


圖 17 獲取桌球顏色範圍

(二) 獲取桌球位置

圖 18 桌球追蹤畫面

我們運用上文提到的追蹤球邏輯設計追蹤球的程式。另外，我們錄了一段 640*480/30fps 和一段 1920*1080/60fps 我們在打桌球的影片，用來檢視追蹤球的效果，並以 LabelImg 手動標出球在畫面中的位置(如圖 19)。標好後，我們用此資料分別找出兩個分辨率下球的外框平均像素面積並評估追蹤球的效果。

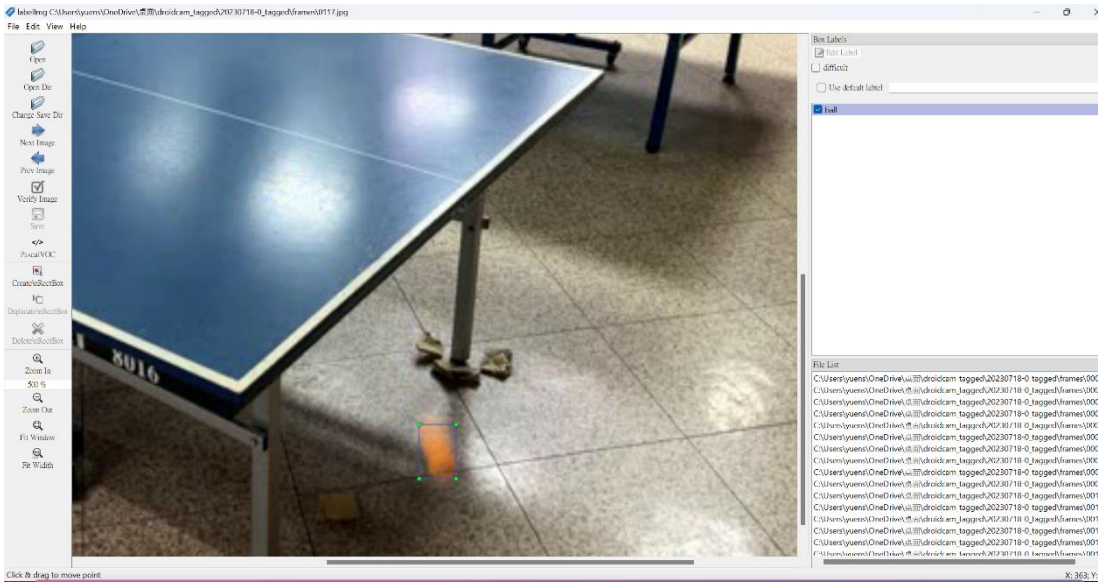


圖 19 手動標註桌球位置

至於評估追蹤球效果的方法我們將其分為兩類，分別是：偵測的準確率以及偵測座標準確度。

偵測的準確率我們用四個像度評估，如下表所示：

表 1 準確度評估方式

	有標註的幀(T)	**沒有標註的幀(F)**
有偵測到的幀(P)	TP	FP
沒有偵測到的幀(N)	TN	FN

而偵測座標的準確度我們是取 TP 的幀中，標記的矩形中心與偵測的矩形中心的幾何距離，並取其平均值 D_m 和標準差 D_s(像素)。

以下是我們評估的結果：

表 2 桌球追蹤效果

	TP	TN	FP	FN	總幀數	D_m(px)	D_s(px)
640*480/30fps	171	216	6	606	777	1.91	1.69
1920*1080/60fps	350	8	2	321	671	3.14	2.08

從上表可以看出我們追蹤球的程式在 FP 上的表現不錯，對輸入到模型來說是一件好事，不會給模型太多錯誤的資料。然而 640*480/30fps 的 TN 比 1920*1080/60fps 高了不少，我們認為可能是在低幀數的情況下每一幀的曝光時間太長，快速移動的物體產生殘影所致(如圖 19)，我們也認為此結果可能跟我們選擇的片段、不同鏡頭參數的特性有關，再調整一些桌球追蹤程式的參數後，應該會有更好的效果。

三、軌跡預測

(一) 生成訓練資料

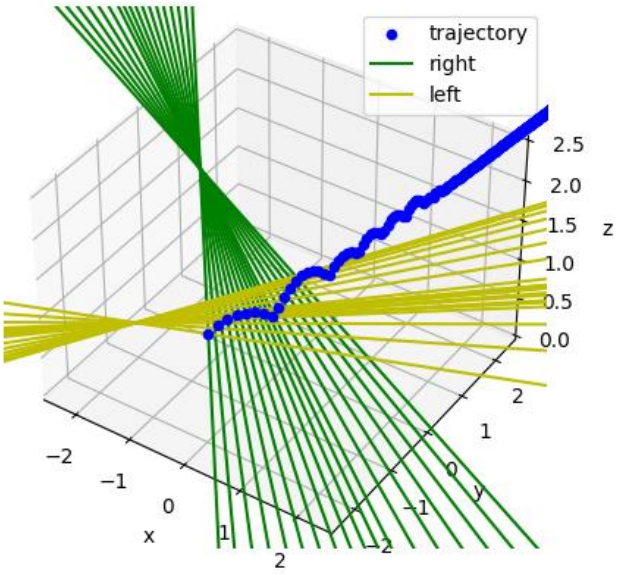


圖 20 單次模擬

上文提到模型的輸入與輸出如下：

輸入：

$$((F_{11}, F_{12}, \dots F_{1n}), (F_{21}, F_{22}, \dots F_{2m}), T) \quad \text{式 15}$$

$$F_{ij} = \left(rxy_{ij},\ rxz_{ij}, x_i, y_i, z_i\right) \quad \text{式 16}$$

$$T = (t_1, t_2, \dots t_k) \quad \text{式 17}$$

輸出：

$$(P_1, P_2, \dots P_k) \quad \text{式 18}$$

$$P_k = (x_k, y_k, z_k) \quad \text{式 19}$$

至於球的模擬，我們假設 z=0 的平面為桌面，球會在桌面中心以上長寬高 3x2x1(m)的範圍隨機生成，以隨機的初速(x、y 方向-5~5 m/s；z 方向 0~3 m/s)進行模擬；相機的位置則是在球的生成範圍以外長寬高 5x4x1.5 的範圍隨機生成兩個。

再來是要將模擬的結果整理成模型的輸入與輸出。我們的做法是先得出需要知道球的位置的時間點，包括：

1. 模型輸入資料中兩個相機拍攝的時間點
2. 模型輸出資料中要預測的時間點

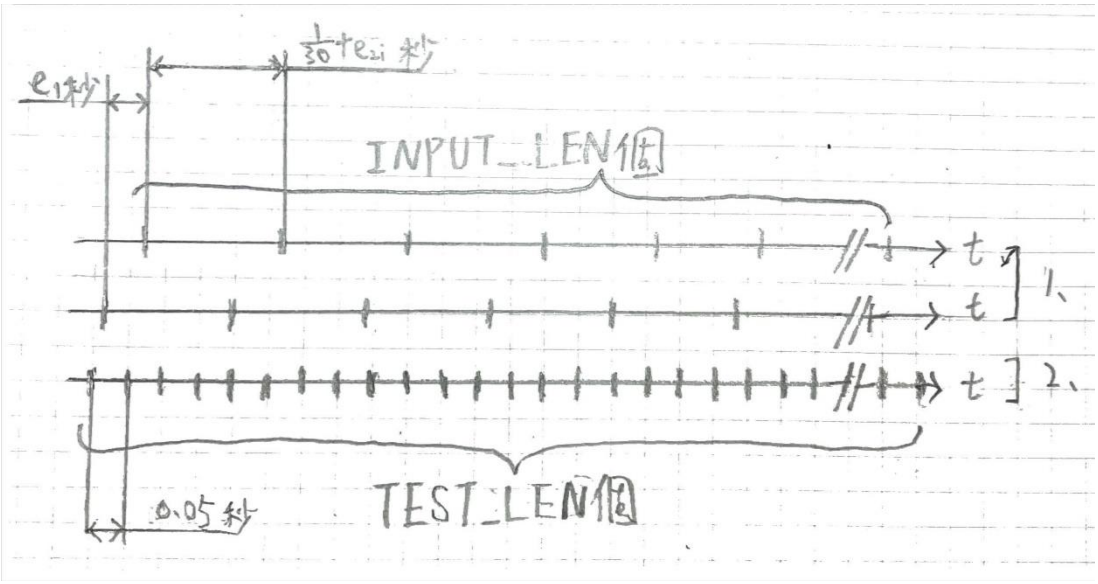


圖 21 時間點示意圖

如上圖所示，其中

$$e_1 \sim |N(0,0.01)| \qquad \text{式 20}$$

$$e_{2i} \sim N(0,0.005) \qquad \text{式 21}$$

N 代表常態分佈，e_1、e_{2i} 分別代表兩相機拍攝時間的系統誤差與隨機誤差，INPUT_LEN 代表最多相機已經拍到幾幀，可以丟到模型裡模擬，TEST_LEN 代表最多要預測多久之後的球的狀態。接著，從頭開始模擬(我們以 1/600s 做為模擬步長)並檢查模擬時間是否達到上述三個時間點的其中一個。

若模擬的時間達到相機拍攝一幀的時間點，則將球的位置與相機位置的直線以上文提到直線的表示法儲存為該相機的”幀資料”，其中，相機位置座標的三個分量分別隨機加入一個誤差，並且同一個相機的誤差值不會隨著每一幀有所改變(視其為相機位置的測量誤差，由於

不會每一幀都測量一次，所以不會隨著每一幀而改變)，而球的位置則和相機一樣隨機加入一個有相同分布的隨機誤差，並且誤差值會隨著每一幀重新取樣(視其為追蹤球所造成的誤差，由於每一幀都會追蹤一次，所以誤差值需要重新取樣)。

若模擬的時間達到要預測的時間點，則將當下球的位置儲存為"預測資料”(因為為理想的輸出值，所以不施加誤差)。

如此，便完成了一次的模擬，為了考慮追蹤球可能會有幾幀遺失，在每一次模擬後我們為各個相機以相同的方法隨機生成 5 個不同的區間 C(代表追蹤球有找到球的區間)，並將兩個相機在各自區間內的”幀資料”與完整的”預測資料”整理為一筆訓練資料，即可得 5 筆訓練資料。區間 C 的隨機方法如下圖所示：

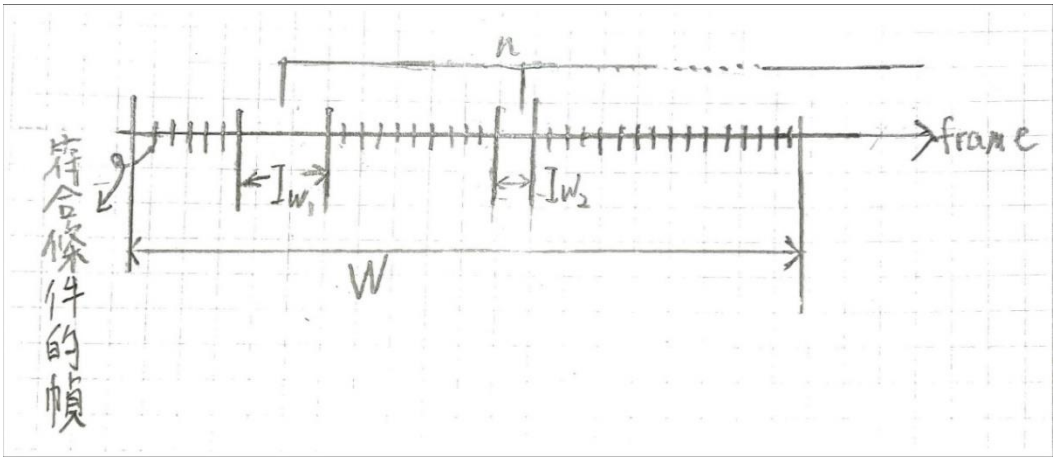


圖 22 區間 C 示意圖

$$C = W \cap I'_{w1} \cap I'_{w2} \cdots \cap I_{wn} \quad \text{式 22}$$

$$W = [0, w]$$
$$w \in \{x | 3 \leq x \leq INPUT\_LEN \text{且} x \in \mathbb{Z}\} \quad \text{式 23}$$

$$I_{wn} = [beg_n, beg_n + i_{wn}], \quad \text{式 24}$$

$$beg_n \in \{x | 3 \leq x \leq INPUT\_LEN \text{且} x \in \mathbb{Z}\}, i_{wn} \sim round(N(3,2)) \quad \text{式 25}$$

Round 表示四捨五入，R 則表示常態分佈

根據上述，可以得出：若輸入的 t_k<在 C 內的最後一幀*1/fps，則該次的輸出 P_k 可以看做是重建桌球已經行進過的軌跡，反之則為預測桌球未來的軌跡。TEST_LEN 和 INPUT_LEN 的大小也因此會很大成度的影響生成資料的分佈。本次研究，我們設計了三組模擬參數 A 預測組、B 擬合組、C 理想預測組(所有誤差接忽視)，並分別模擬了 200000 次球的軌跡，獲取了 1000000 筆訓練資料三組模擬參數如下：

表 3 A、B、C 資料集生成參數

資料集	A	B	C
TEST_LEN	250	100	250
INPUT_LEN	40	100	40
SHUTTER_RANDOM_ERROR_STD(秒)	0.005		0
SHUTTER_SYSTEMATIC_ERROR_STD(秒)	0.01		0
CAMERA_POSITION_ERROR_STD(公尺)	0.05		0
BALL_POSITION_ERROR_STD(公尺)	0.05		0
INPUT_IGNORE_AREA_MEAN	3		0
INPUT_IGNORE_AREA_STD	2		0
INPUT_IGNORE_WIDTH_MEAN	4		0
INPUT_IGNORE_WIDTH_STD	3		0

另一方面，我們也有試著在 PyBullet 進行桌球運動模擬的時候應用球的旋轉，但是我們卻發現旋轉的有無並未對球的軌跡產生影響。於是我們先暫時不在模擬時應用球的旋轉。將來，我們希望能夠以不同的方式進行旋球的模擬(例如用 VPython 進行模擬)，以預測旋球的軌跡。

(二) 模型訓練

我們將模型分成前後兩段：前段包含全連接層 1 與 LSTM 層，後段包含全連接層 2。以下是我們用 PyTorch 實現向前傳播的程式碼：

```
def forward(self, X1:torch.Tensor, X1_len:torch.Tensor, X2:torch.Tensor, X2_len:torch.Tensor, T:torch.Tensor):
    x1_batch_size = len(X1)
    x2_batch_size = len(X2)
    # 輸入全連接層1
    X1 = self.mlp1(X1.view(-1,self.input_size)).view(x1_batch_size, -1, self.mlp1_out)
    X2 = self.mlp1(X2.view(-1,self.input_size)).view(x2_batch_size, -1, self.mlp1_out)

    # 輸入LSTM
    X1 = X1.transpose(0, 1)
    X2 = X2.transpose(0, 1)
    X1_seq, self.llstm_hidden_cell = self.lstm(X1, self.llstm_hidden_cell)
    X2_seq, self.rlstm_hidden_cell = self.lstm(X2, self.rlstm_hidden_cell)

    # 擷取LSTM最後一次的輸出
    X1_len_ind = X1_len - 1
    X2_len_ind = X2_len - 1
    X1_ind = X1_len_ind.view(1, x1_batch_size, 1).expand(1, x1_batch_size, self.lstm_out)
    X2_ind = X2_len_ind.view(1, x2_batch_size, 1).expand(1, x2_batch_size, self.lstm_out)
    X1 = X1_seq.gather(0, X1_ind).view(1, x1_batch_size, self.lstm_out)
    X2 = X2_seq.gather(0, X2_ind).view(1, x2_batch_size, self.lstm_out)

    # 合併前段模型輸出
    X1 = X1.transpose(0, 1)
    X2 = X2.transpose(0, 1)
    X = torch.cat((X1, X2), 2)

    # 合併的結果複製k份，k為輸入T的長度，再與要預測的時間點合併
    X = X.repeat(1, T.shape[1], 1)
    X = torch.cat((X, T.view(x1_batch_size, T.shape[1], 1)), 2)

    # 輸入全連接層2
    res = self.mlp2(X.view(-1, self.lstm_out * 2 + 1)).view(x1_batch_size, T.shape[1], self.output_size)
    return res
```

圖 23 前後段模型傳播程式碼

其中 mlp1(全連接層 1)、lstm、mlp2(全連接層 2)是抽象的成員，讓我們可以靈活的透過 class 的繼承實做出不同大小的模型，我們實作出了 small、medium、big、large 模型，各層詳細的大小定義如下所示：

1. small

```
ISEFWINNER_SMALL(
  (mlp1): Sequential(
    (0): Linear(in_features=5, out_features=50, bias=True)
    (1): ReLU()
    (2): Linear(in_features=50, out_features=40, bias=True)
    (3): Tanh()
    (4): Linear(in_features=40, out_features=40, bias=True)
    (5): Tanh()
    (6): Linear(in_features=40, out_features=15, bias=True)
  )
  (lstm): LSTM(15, 40, num_layers=4)
  (mlp2): Sequential(
    (0): Linear(in_features=81, out_features=45, bias=True)
    (1): ReLU()
    (2): Linear(in_features=45, out_features=30, bias=True)
    (3): Tanh()
    (4): Linear(in_features=30, out_features=20, bias=True)
    (5): Tanh()
    (6): Linear(in_features=20, out_features=3, bias=True)
  )
)
```

2. medium

```
ISEFWINNER_MEDIUM(
  (mlp1): Sequential(
    (0): Linear(in_features=5, out_features=100, bias=True)
    (1): ReLU()
    (2): Linear(in_features=100, out_features=80, bias=True)
    (3): Tanh()
    (4): Linear(in_features=80, out_features=80, bias=True)
    (5): Tanh()
    (6): Linear(in_features=80, out_features=50, bias=True)
  )
  (lstm): LSTM(50, 60, num_layers=6)
  (mlp2): Sequential(
    (0): Linear(in_features=121, out_features=90, bias=True)
    (1): ReLU()
    (2): Linear(in_features=90, out_features=60, bias=True)
    (3): Tanh()
    (4): Linear(in_features=60, out_features=50, bias=True)
    (5): Tanh()
    (6): Linear(in_features=50, out_features=30, bias=True)
    (7): Tanh()
    (8): Linear(in_features=30, out_features=3, bias=True)
  )
)
```

3. big

```
ISEFWINNER_BIG(
  (mlp1): Sequential(
    (0): Linear(in_features=5, out_features=140, bias=True)
    (1): ReLU()
    (2): Linear(in_features=140, out_features=120, bias=True)
    (3): Tanh()
    (4): Linear(in_features=120, out_features=100, bias=True)
    (5): Tanh()
    (6): Linear(in_features=100, out_features=90, bias=True)
  )
  (lstm): LSTM(90, 70, num_layers=7)
  (mlp2): Sequential(
    (0): Linear(in_features=141, out_features=90, bias=True)
    (1): ReLU()
    (2): Linear(in_features=90, out_features=90, bias=True)
    (3): Tanh()
    (4): Linear(in_features=90, out_features=90, bias=True)
    (5): Tanh()
    (6): Linear(in_features=90, out_features=60, bias=True)
    (7): Tanh()
    (8): Linear(in_features=60, out_features=60, bias=True)
    (9): Tanh()
    (10): Linear(in_features=60, out_features=3, bias=True)
  )
)
```

4. large

```
ISEFWINNER_LARGE(
  (mlp1): Sequential(
    (0): Linear(in_features=5, out_features=200, bias=True)
    (1): ReLU()
    (2): Linear(in_features=200, out_features=150, bias=True)
    (3): Tanh()
    (4): Linear(in_features=150, out_features=150, bias=True)
    (5): Tanh()
    (6): Linear(in_features=150, out_features=90, bias=True)
    (7): Tanh()
    (8): Linear(in_features=90, out_features=70, bias=True)
  )
  (lstm): LSTM(70, 60, num_layers=8)
  (mlp2): Sequential(
    (0): Linear(in_features=121, out_features=130, bias=True)
    (1): ReLU()
    (2): Linear(in_features=130, out_features=120, bias=True)
    (3): Tanh()
    (4): Linear(in_features=120, out_features=100, bias=True)
    (5): Tanh()
    (6): Linear(in_features=100, out_features=60, bias=True)
    (7): Tanh()
    (8): Linear(in_features=60, out_features=60, bias=True)
    (9): Tanh()
    (10): Linear(in_features=60, out_features=3, bias=True)
  )
)
```

再來是模型的訓練，首先，為了找出各項參數對模型訓練結果的影響，我們進行了非常多不同參數、模型大小的模型訓練，起初，因為 batch_size 設定大小不足，模型的 loss 不減反增，直到 batch_size 設定到大於 16 之後，模型才開始慢慢地收斂。

以下是我們以 learning rate=0.001，Adam 優化器及不同 batch_size 訓練 8 個 epoch 的 loss 曲線：

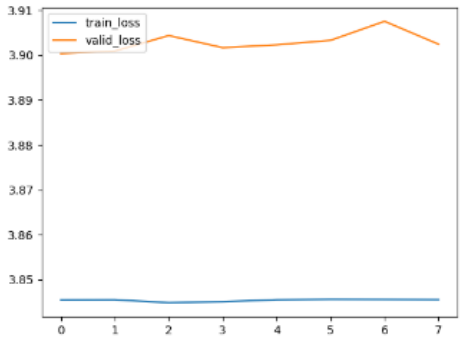


圖 24 batch_size=4 的 loss 曲線圖

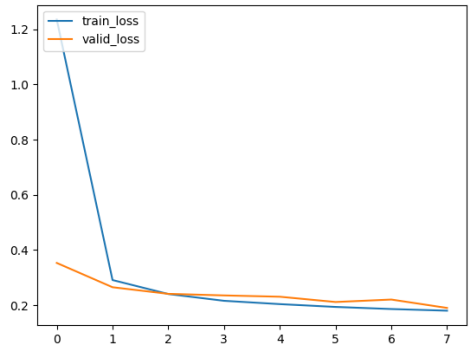


圖 25 batch_size=16 的 loss 曲線圖

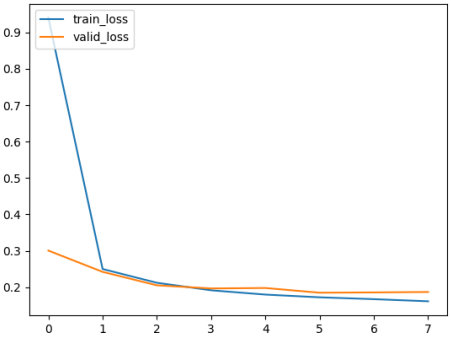


圖 26 batch_size=64 的 loss 曲線圖

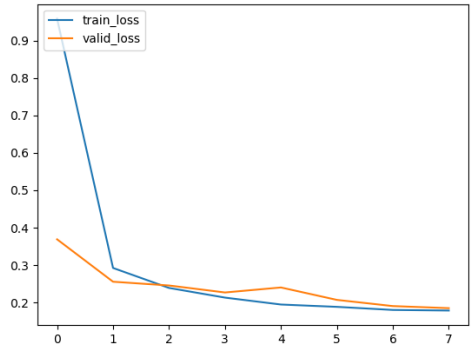


圖 27 batch_size=256 的 loss 曲線圖

再來是我們以 batch_size=64，Adam 優化器及不同 learning rate 訓練 8 個 epoch 的 loss 曲線：

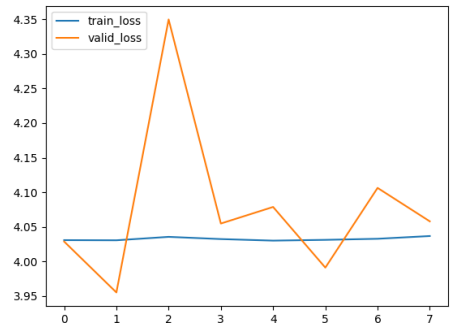


圖 28 Learning rate=0.1 的 loss 曲線圖

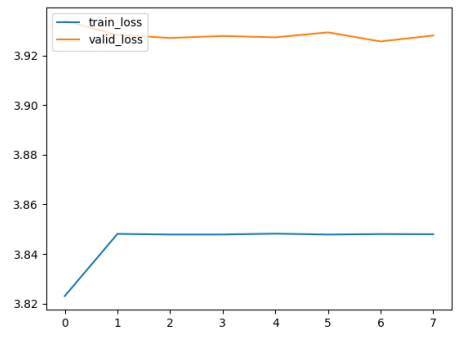


圖 29 Learning rate=0.01 的 loss 曲線

圖

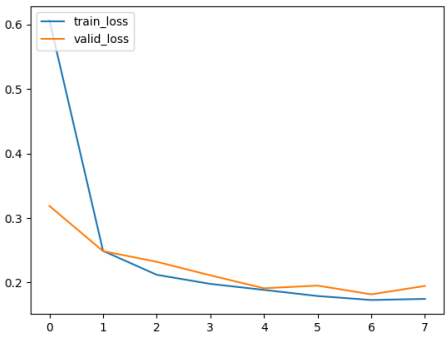


圖 30 Learning rate=0.001 的 loss 曲線圖

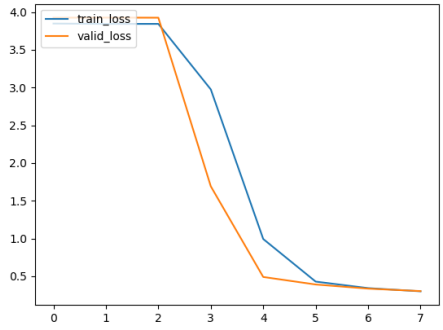


圖 31 Learning rate=0.0001 的 loss 曲線圖

找到了較佳的訓練參數後，我們用 medium 模型針對上文生成的 A、B、C 組訓練資料做訓練，想要探討應用不同資料集的成效如何，訓練參數為：

- 優化器：Adam
- Batch size：64
- Learning rate：0.001
- Step scheduler：step size=8
- Epoch：30
- Loss function：MSE loss

以下是三組 medium 模型在訓練時的 loss 曲線：

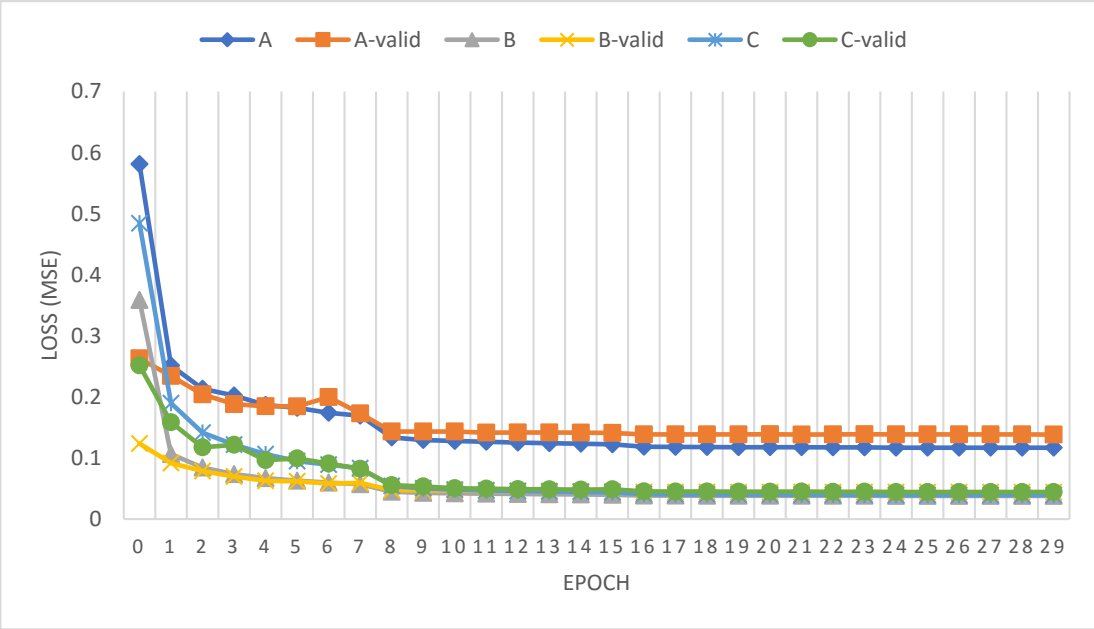
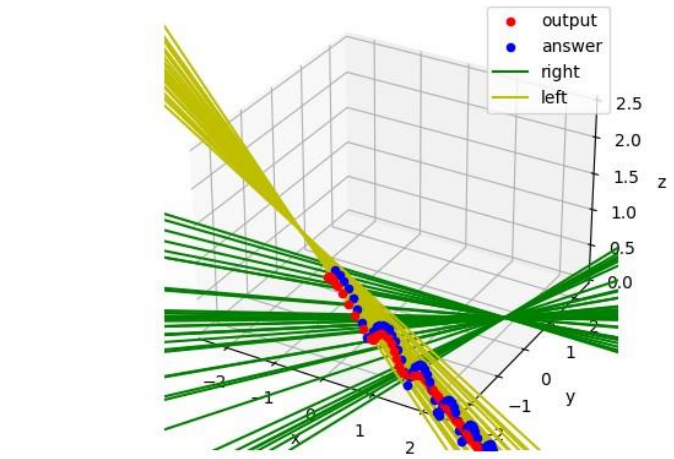


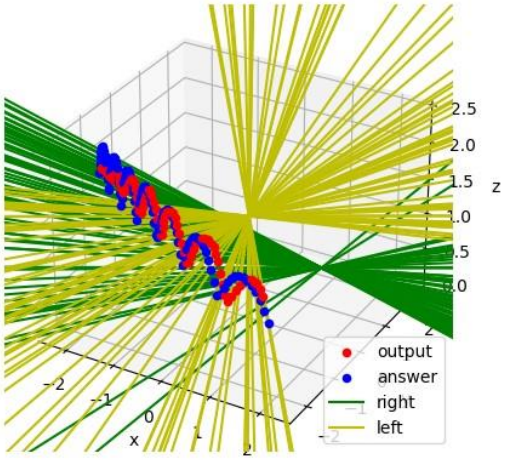
圖 32 medium 模型在 A、B、C 三個資料集中訓練的 Loss 曲線

以測試資料來看，A 模型的 Loss 大約能達到 0.13，B、C 模型則是在 0.04 左右。以下為依序為 A、B、C 模型對某一筆資料的預測圖：



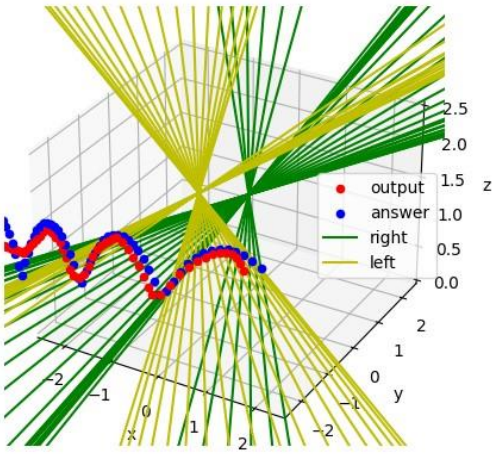
MSE loss: 0.014099556021392345

圖 33 A 模型對一筆資料預測圖



MSE loss: 0.033213421702384 95

圖 34 B 模型對一筆資料預測圖



MSE loss: 0.04814844951033592

圖 35 C 模型對一筆資料預測圖

訓練好後，再分析三組訓練好的模型分別對 A、B、C 資料集的測試資料的 loss，結果如下：

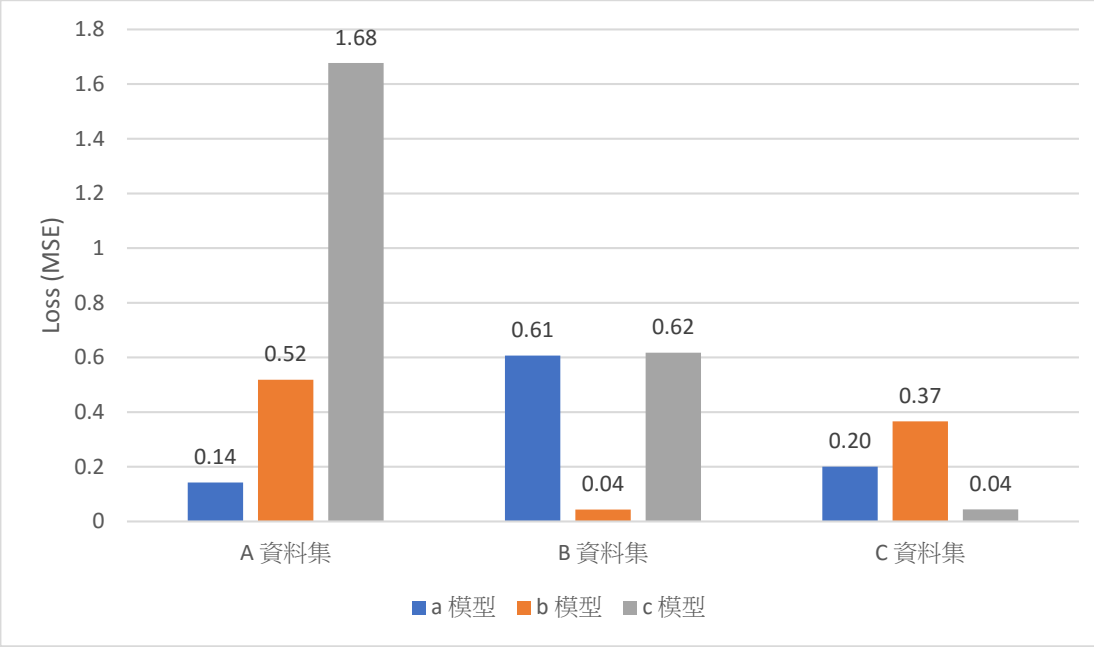


圖 36 三組訓練好的模型分別對 A、B、C 資料集的測試資料的 loss

其中，a 模型是指以 A 資料集訓練的模型，以此類推。

從上圖 36 我們發現了兩件事：

1. 太長的 INPUT_LEN 與 TEST_LEN 並不會使在其訓練的模型有更佳、更通用的預測能力，設定一組適當(符合應用情形)的值會使預測的效果最好。
2. 我們應用的誤差對結果的影響是很大的。

第一點是由 a 模型在 A、B 資料集的 Loss 發現的：A 資料集相比 B 資料集需要在更少的輸入(INPUT_LEN:A<B)下得到更多的輸出(TEST_LEN:A>B)，原本認為 a 模型的"預測能力"應該會比 B 模型好，a 模型在 B 資料集上的 Loss 表現應該也會不錯，結果恰恰相反，b 模型在 A 資料集上的表現反而還比較好，且兩者的 loss 皆不小，進一步思考後，我們認為我們訓練的模型不是在找一個通用的軌跡預測算

法，只能較好的處理和我們設定的 TEST_LEN 和 INPUT_LEN 分布相近的預測工作。

第二點則是由 c 模型在 A 資料集和 a 模型在 C 資料集的 Loss 看出來的：a 模型可以很好的適應 C 資料集，但 c 模型則完全無法應用在 A 資料集上，可見在資料集中加入誤差的重要性，我們也有點擔心我們所加入的誤差是否和真實應用情形所產生的誤差相近。我們也發現 a 模型在 C 資料集上的 Loss 比 c 模型在 C 資料集上的 Loss 大了一點，因為 A 資料集跟 C 資料集差在有無加入誤差，C 資料集的"分布"應該被包含在 A 資料集中，但是卻有如此的結果，我們認為在模型參數一樣的情況下，a 模型為了應對誤差的加入，只能降低預測對於沒有誤差的資料的準確度，成為更加通用的模型。

最後，我們根據 1. 的結論，以接近實際應用情形為原則，創建了用以下參數模擬生成的訓練資料，並重新訓練出最終的模型：

```
TEST_LEN = 50
INPUT_LEN = 40
SHUTTER_RANDOM_ERROR_STD = 0.005
SHUTTER_SYSTEMATIC_ERROR_STD = 0.03
CAMERA_POSITION_ERROR_STD = 0.05
BALL_POSITION_ERROR_STD = 0.05
INPUT_IGNORE_AREA_MEAN = 3
INPUT_IGNORE_AREA_STD = 3
INPUT_IGNORE_WIDTH_MEAN = 4
INPUT_IGNORE_WIDTH_STD = 3
```

(三) 模型推理與部署

```
if result is not None :
    x, y, w, h = result
    # 在畫面中畫出偵測到的矩形
    self.drawDirection(frame, x, y, h, w, 1)
    if self.homography_matrix is not None and self.camera_position is not None:
        # 單應性變換
        ball_in_world = np.matmul(self.homography_matrix, np.array([x+w//2, y+h//2, 1]))
        # 獲取投影點
        projection = equ.Point3d(ball_in_world[0]-2.74/2, 0, ball_in_world[1])
        # 將投影點和相機座標連成直線
        line = equ.LineEquation3d(self.camera_position, projection)
```

圖 37 座標變換程式

```
class LineCollector:
    def __init__(self) :
        self.bounceTimestamp = 0
        self.lines:List[Tuple[int, int, int, int, int]] = []
    def put(self, x, y, z, rxy, rxz) :
        if self.checkHit(x, y, z, rxy, rxz) :
            self.lines.clear()
            return False
        else :
            if len(self.lines) == 0 :
                self.bounceTimestamp = time.time()
            if len(self.lines) >= Constants.SIMULATE_INPUT_LEN:
                self.lines.pop(0)
            self.lines.append((x, y, z, rxy, rxz))
            return True
    def checkHit(self, x, y, z, rxy, rxz) :
        pass
    def clear(self) :
        self.lines.clear()

def rad_minus(a, b) :
    return (a - b + np.pi) % (2 * np.pi) - np.pi

class LineCollector_hor(LineCollector) :
    def __init__(self):
        super().__init__()
    movement = None
    last_rxy = None
    def clear(self):
        self.last_rxy = None
        self.movement = None
        super().clear()
    def checkHit(self, x, y, z, rxy, rxz):
        if self.movement == 1:
            if rad_minus(rxy, self.last_rxy) < 0:
                self.movement = None
                self.last_rxy = None
                return True
        elif self.movement == -1 :
            if rad_minus(rxy, self.last_rxy) > 0:
                self.movement = None
                self.last_rxy = None
                return True
        elif self.movement == None and self.last_rxy is not None:
            self.movement = 1 if rxy - self.last_rxy > 0 else -1
        self.last_rxy = rxy
        return False
```

圖 38 維護直線序列的程式

目前模型推理與部署已完成的部分如下：

1. 將追蹤球的結果結合相機位置、單應性矩陣進行座標轉換後得到經過相機和球的直線方程式(如圖 37)。
2. 透過 multiprocessing 函式庫開啟額外的兩個進程同時進行兩個相機的追蹤球與座標變換的工作，並與主進程溝通。
3. 主進程收到資料後，藉由接收到的直線的 xz 平面的斜率改變趨勢判斷球是否受外力干擾(運動方向改變)，並維護經過相機和球的直線序列(如圖 38)。
4. 進行模型推理並處理輸出，並將其用 matplotlib 庫畫出模型的輸入與輸出。

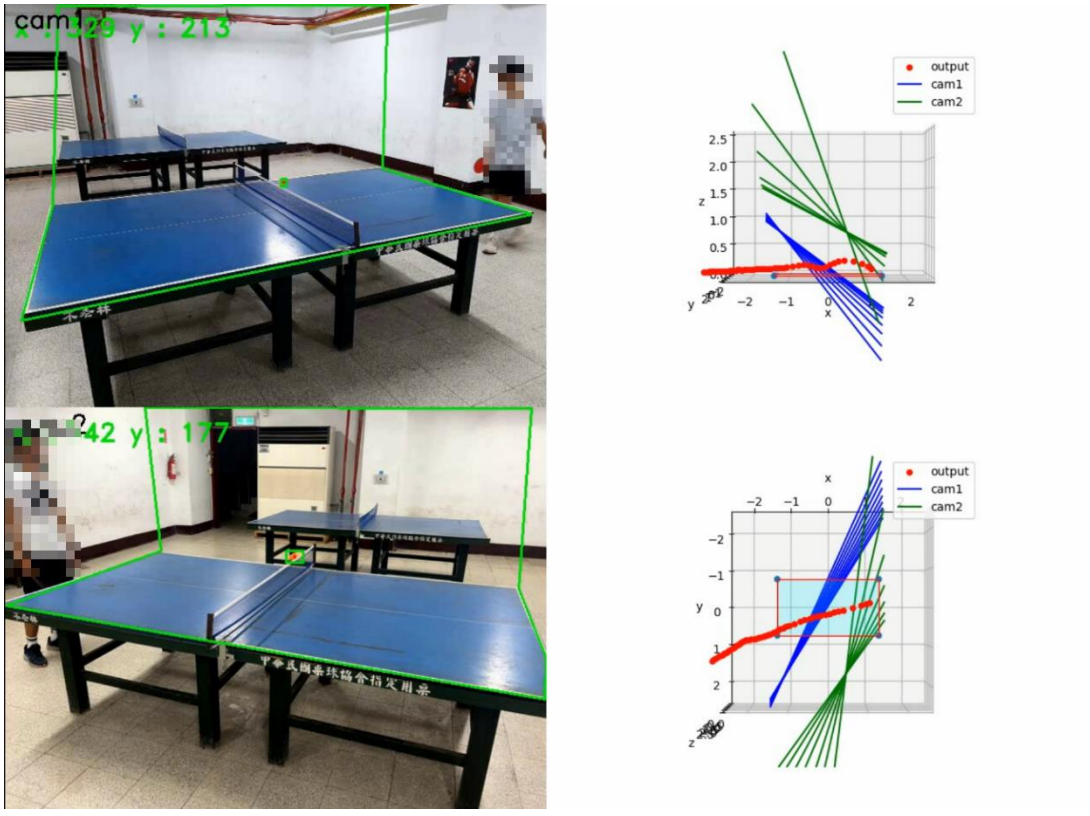


圖 39 軌跡預測視覺化結果

接著，我們用兩個相機錄製一段同步的打桌球的影片，並帶入上面寫到的程式，最終的視覺化結果如圖 39，右方紅色的軌跡為模型所預測的桌球的軌跡。然而，我們卻發現在視覺化結果中兩相機的直線序列(圖 39 中的藍、綠色直線)有時重疊的部分不多，進一步排除是兩相機同步的問題後(透過減去兩部影片中球彈跳的時間點的幀數差，大約有 10 幀)，上述的情形依舊。最終，我們在檢查單應性變換時發現了問題所在：我們用從 AprilTag 獲取的單應性矩陣對畫面中的桌球桌長度進行測量(真實的桌球桌長度應在 2.74 公尺上下)，卻發現其中一台相機測得的桌長為 2.07 公尺，另一台相機測得的則是 2.40 公尺。我們認為可能是由兩個因素造成了這項結果：

1. 相機分辨率太小(只有 640x480)
2. AprilTag 實體太小(邊長約 16 公分)

我們目前想到的解決方案有：

1. 印出更大的 AprilTag 進行校正
2. 將相機定位的參照物由 AprilTag 改為較大的桌球桌面

四、機器手臂

(一) 機構設計

我們透過 SolidWorks 設計出更加詳細的機器人細節，包含各個零件是如何固定在一起的、決定要用甚麼材質等等，以下是我們設計

的機構圖與其爆炸圖：

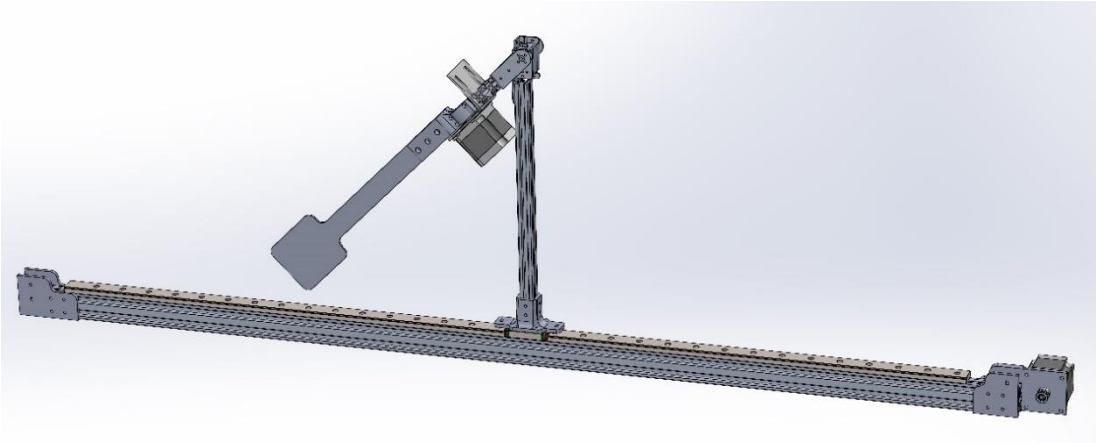


圖 40 機器手臂 CAD 圖(研究者自行繪製 [15]、[16]、[17]、[18]、[19])

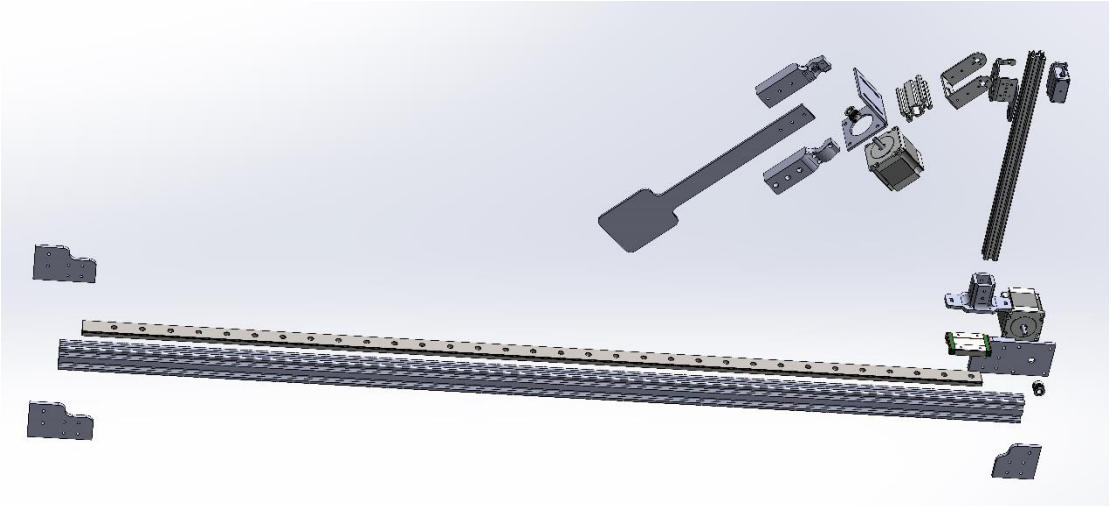


圖 41 機器手臂爆炸圖(研究者自行繪製[15]、[16]、[17]、[18]、[19])

此機構的設計和上文研究方法裡提到的設計概念非常相似，是將原本較模糊的構想加以實體化，其中也不乏稍作修改。同樣將其分為上半部的手臂機構與下半部的滑軌機構。

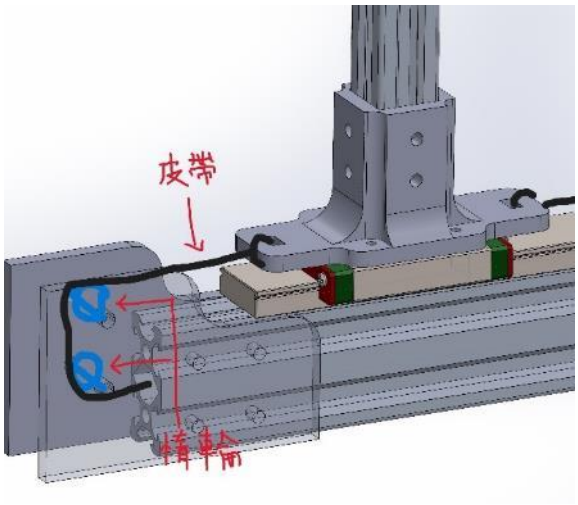


圖 42 滑軌傳動方法(研究者自行繪製 [15]、[16]、[17]、[18]、[19])

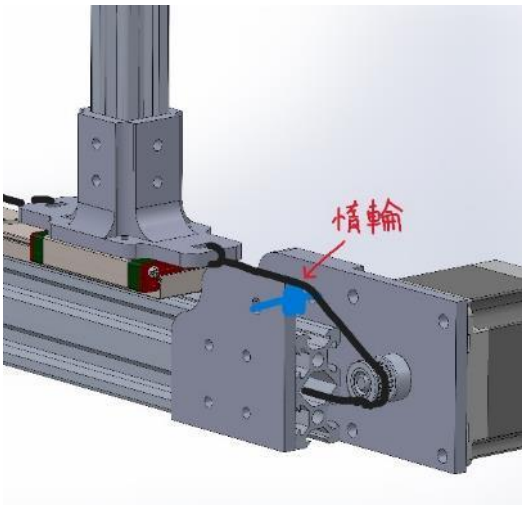


圖 43 滑軌傳動方法(研究者自行繪製：[15]、[16]、[17]、[18]、[19])

滑軌機構的設計如上圖所示，以 2GT 皮帶穿過 2040 的歐規鋁擠型以 57BYGH56 的步進馬達帶動手臂在 1400mm MGW12H 的線性滑軌上左右移動。其中，安裝惰輪和馬達的零件、連接滑塊與手臂的零

件我們打算利用 3D 列印，並設計導角以分散應力。

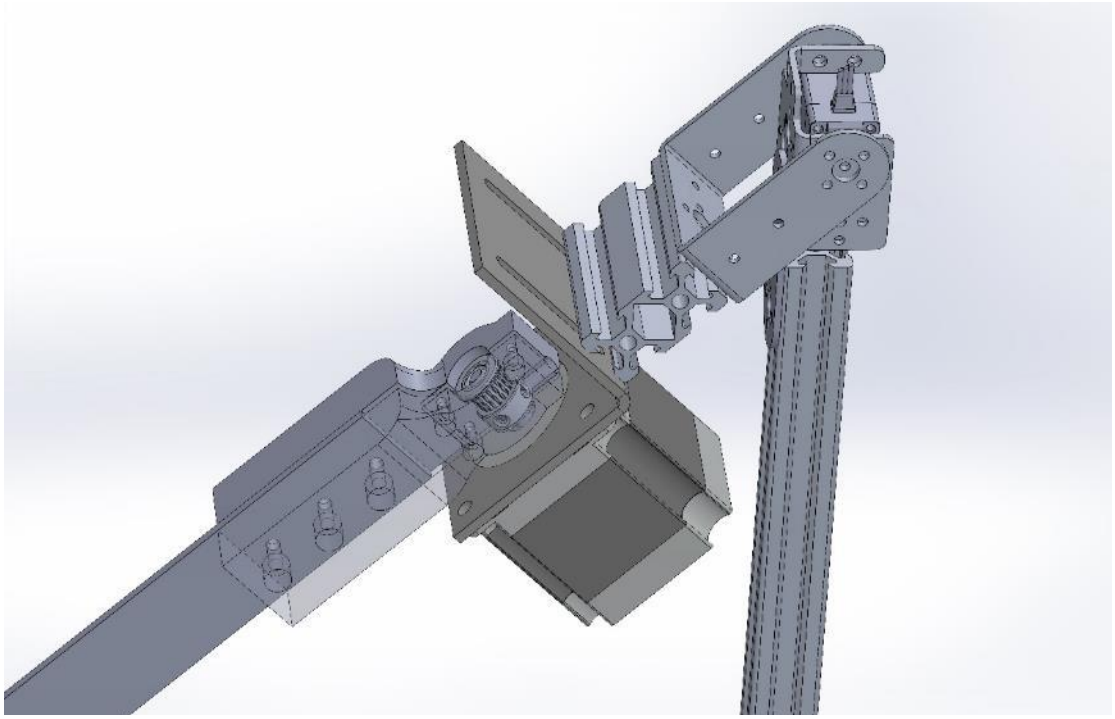


圖 44 關節(研究者自行繪製 [15]、[16]、[17]、[18]、[19])

手臂機構的設計如上圖所示，用能夠轉 270 度的 LD27MG 伺服馬達控制仰角，57BYGH56 步進馬達控制揮拍。其中，拍面我們打算用 3mm 的雷射切割木板，連接 57 步進馬達與拍面的機構，我們設計是用兩片一端可以吻合 2GT 皮帶輪，另一端可以用螺絲鎖固拍面的零件結合而成，並打算用 3D 列印製作。而圖中反著裝的 57 步進馬達支架是為了不要讓手臂在調整仰角的時候與下方 2020 鋁擠型發生干涉，使其能夠完整的旋轉 270 度，並且能夠透過調整鎖固位置讓拍面運動的軌跡和伺服馬達的旋轉軸在同一個平面上，方便程式設計。

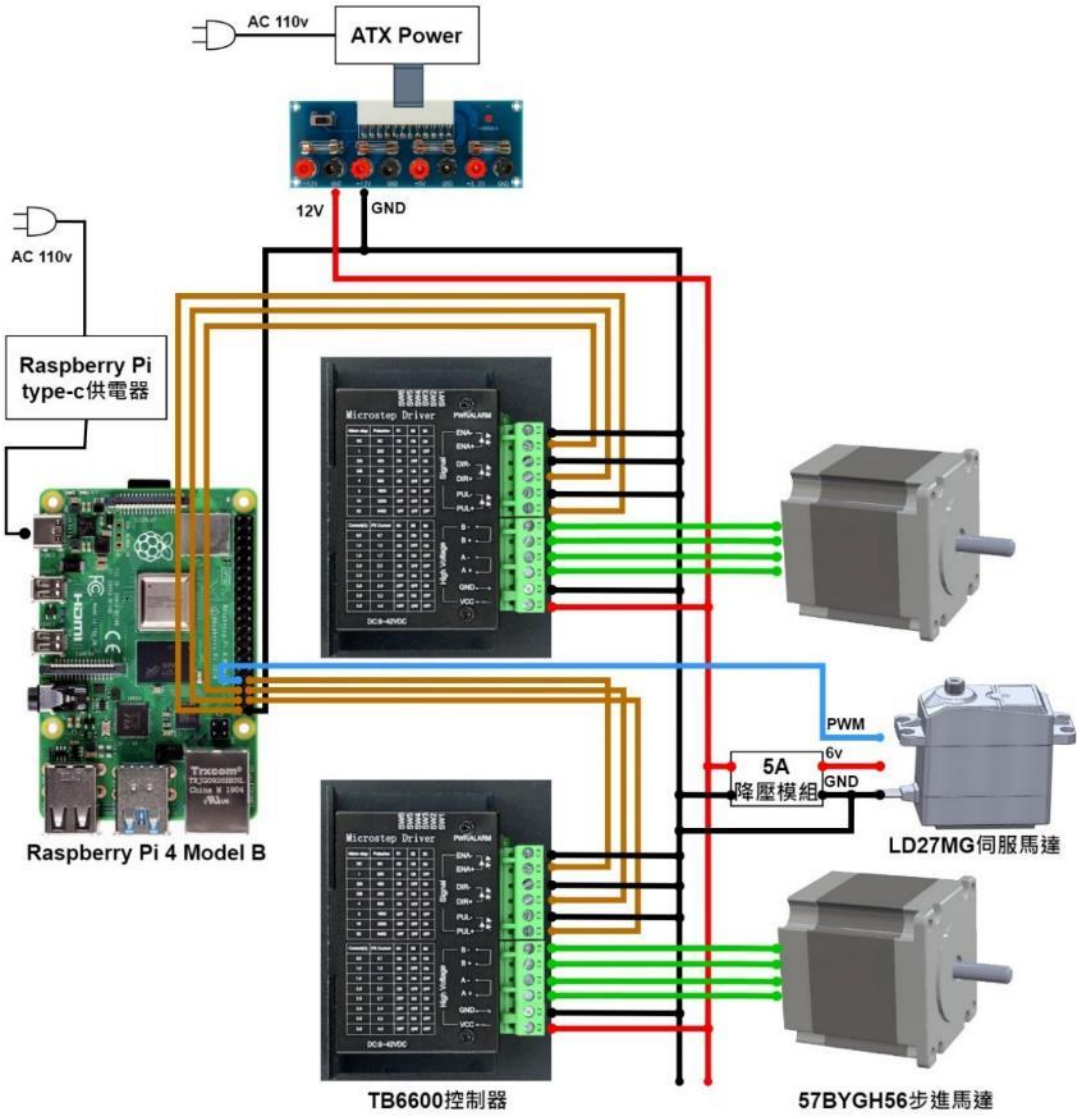


圖 45 接線圖 (研究者自行繪製，[17]、[18]、[20]、[21])

另外，我們也設計出了各個馬達與控制器、電子零件的線路接法如圖，我們用電腦主機作為變壓器輸出 12v 穩定的電源供應兩顆步進馬達，再以降壓板將電壓降至 6v 左右單獨為伺服馬達供電。

(二) 組裝與控制

1. 橫向移動(步進馬達)

我們使用 Raspberry Pi 接上 TB6600 並以 1 細分驅動 57BYGH56 步進馬達(步距角 1.8°)，用 Python 中的 RPI.GPIO 函式庫控制信號輸出。然而，原本預計要連接滑軌與上方機構的 3D 列印件在初次測試時斷裂，如圖 46。經過分析，我們認為主要的原因有兩點：

(1) 當機器手臂要進行折返時，並不會減速，而導致 3D 列印零件因為慣性承受太大的力量。

(2) 我們觀察了斷裂的零件，發現其密度很低。

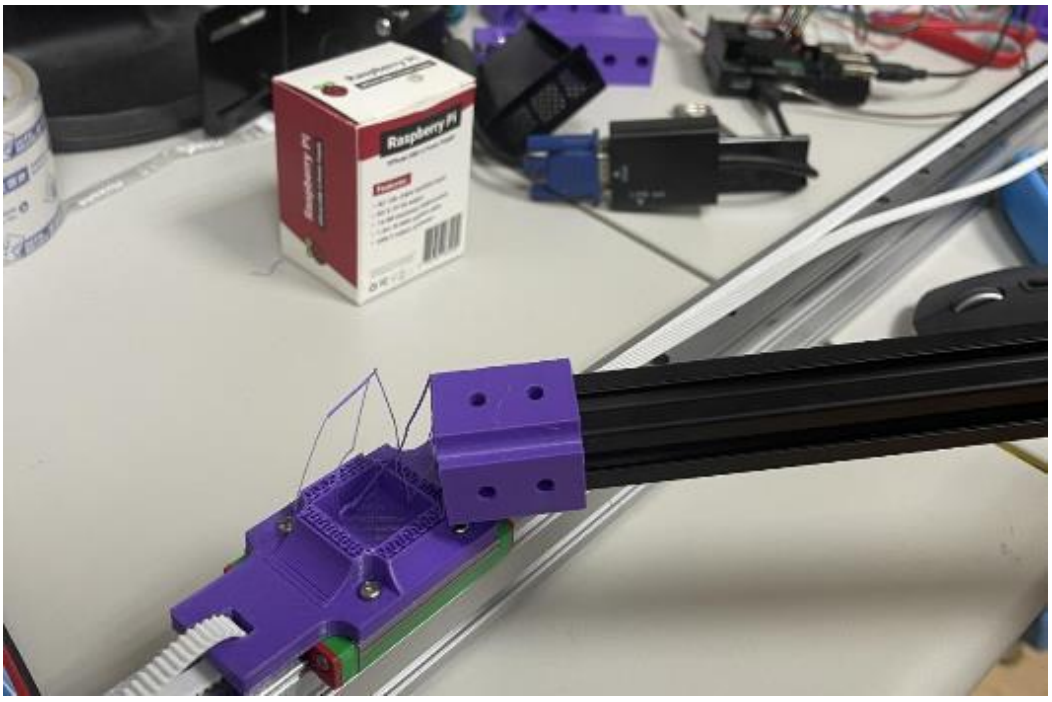


圖 46 連接線性滑軌與機器手臂之零件斷裂

於是我們更改了該連接部件的設計如下圖 47 所示：

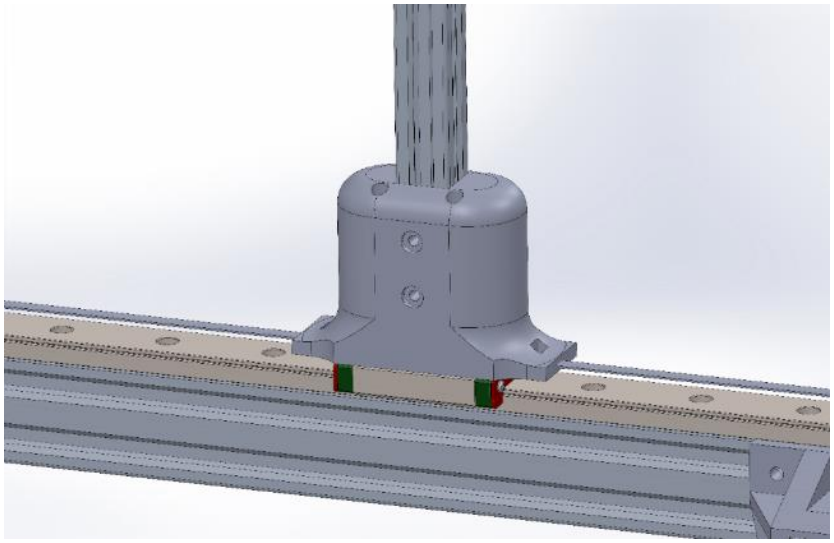


圖 47 新零件設計(研究者自行繪製 [15]、[16]、[17]、[18]、[19])

並設計一個名為 Stepper 的 class 繼承 threading.Thread，以絕對位置的方始移動機器手臂，使 Raspberry Pi 能夠不斷地接收桌球軌跡預測的結果更改手臂位置。也因應(1)，在移動時應用加速及減速，讓步進馬達在每一步可以加上設定的加速度，直到達到設定的最高速度，使機器手臂能以更高的速度移動，並且如果遇到折返或停下的狀況，手臂也不會因為慣性而導致丟步，進而造成往後的控制的誤差。最終，我們的手臂能夠以最高每秒 1000 步(80cm/s)的速率左右移動。

2. 揮拍(步進馬達)。

與橫向移動相同，我們使用 Raspberry Pi、TB6600 1 細分驅動 57BYGH56 步進馬達並以 Python 設計了一個 hit_ball 函式，當桌球的預測位置接近機器手臂時，步進馬達將帶動拍面進行來回一次的擺動，做出擊球的動作，而在擺動時同樣會有加速度及減速度，來防止丟步

造成的拍面偏移，揮拍速度最快時可以達到每秒 700 步(拍面約 6.5m/s)的速率。

3. 調整拍面高度(伺服馬達)

我們使用 PWM 來控制伺服馬達，使其可以在 0-270 度之間擺動，調整拍面的高低。進一步測試後發現當伺服馬達到達定點之後，並不會在定位保持靜止，而是在定位附近來回不停地抖動，起初我們懷疑是 Raspberry Pi 的供電不穩造成的問題，於是我們嘗試使用不斷電系統(Uninterruptible Power Supply)對 Raspberry Pi 進行供電，經過測試，伺服馬達的顫抖幅度有減少，但依然會有抖動的情況發生。

最終，我們完成了整體機構的組裝：

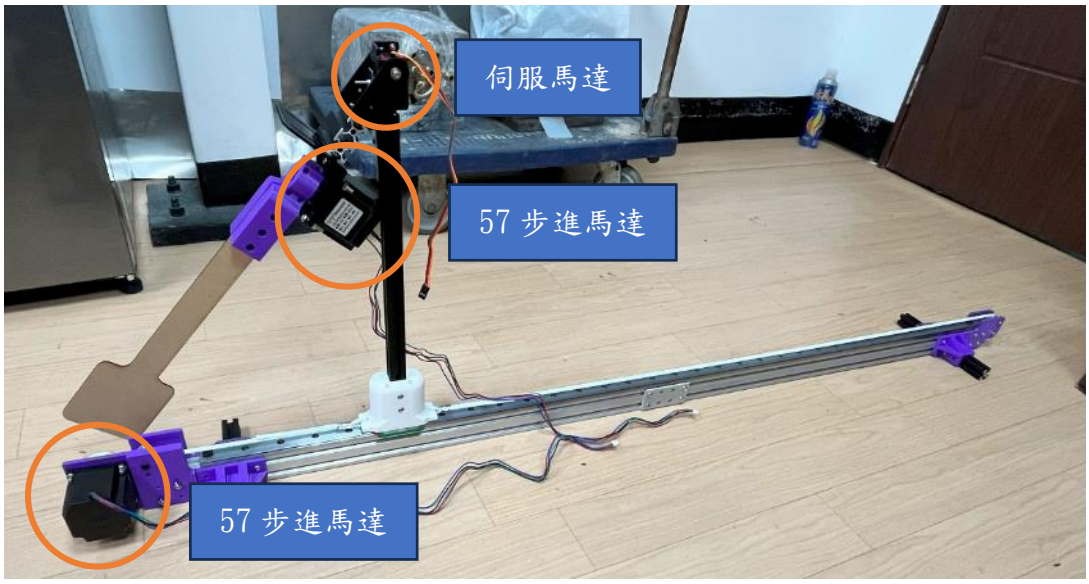


圖 48 機器手臂組裝完成圖

肆、研究結論與應用

一、研究結論

本研究完成了相機校正、空間定位、乒乓球追蹤、模型訓練、機器人控制，雖尚未達成回擊桌球的目標，但在各個項目上皆有很大的突破，特別是軟體的部分，我們已能夠用成本比高速攝影機低了不少的手機鏡頭對桌球的軌跡做出預測。機器手臂硬體的部分也能夠迅速地移動到指定的位置擊球。期望在不久的將來能夠完成這項專案。以下我們為各個項目重點說明：

首先是桌球追蹤部分，我們依序透過顏色追蹤、動態追蹤、篩選桌球位置準確地找出桌球在畫面中的位置，其中我們發現：

1. 若相機的躁點大，使用”將靜止的目標刪除”的效果會比[11]用的兩幀相減好。
2. 在篩選桌球位置時，先刪除面積和座標不在設定範圍內的目標，再用”和上次追蹤結果座標最近”或”面積和設定值最近”篩選桌球可以有效地消除人或拍面的干擾。

再來是軌跡預測的部分，我們透過 PyBullet 模擬桌球軌跡生成訓練資料，進行模型訓練以及空間定位與部署，並能夠預測出球的軌跡。其中我們發現：

1. 生成訓練資料時，太長的 INPUT_LEN 與 TEST_LEN 並不會

使在其訓練的模型有更佳、更通用的預測能力，設定一組適當(符合實際應用情形)的值會使預測的效果最好。

2. AprilTag 進行空間定位時，實體目標物大小以及相機的分辨率大小皆不能太小，我們使用約 16.4 公分的 AprilTag 在 640x480 的分辨率下產生了不可忽視的誤差。

最後是機器手臂控制的部分，目前已能透過網路通訊接收信號、迅速地計算並控制滑塊移動到指定的位置，配合伺服馬達調整仰角、步進馬達控制擊球。其中我們發現：

1. 伺服馬達在承受較大的力矩時，容易發熱並產生抖動。
2. 步進馬達控制時適當的進行加速與減速能夠有效地防止丟步並能達到更高的最大轉速，最終使我們的手臂移動一整條滑軌的距離只費時約 1.9 秒。

雖然整體系統尚未整合在一起，並且達到即時的人機互動使用體驗，但根據我們目前各個部分的成果，我們有信心在不久的將來，我們能夠完成整體的系統，從偵測桌球、軌跡預測到機器手臂做出回擊的動作，整個過程可以達到及時地人機對打以及高精準度，提供使用者良好的使用體驗。

二、未來應用

本研究拋棄了許多以往的桌球教練機器人所使用的高階硬體設備，包含高速攝影機以及較昂貴的擊球機構，並研發價格較親民的解決方案，為桌球教練機器人的商品化提供可能性，甚至桌球愛好者也能夠自行搭建。

並且，我們提出的基於深度學習的軌跡預測方法除了用來預測桌球的軌跡，更是能應用在其他物體的三維重建上。首先，過去的方法中，要進行多相機的三維重建勢必需要較高幀率、品質的鏡頭，並在時間上進行精確的同步，但若是用本研究所採用的重建方法，即可有效的解決這方面的問題；再來，我們預測的物體是既快速又小的乒乓球，在軌跡預測的任務中算是困難度很高的，若最後預測的效果不錯，想必也能在其他的三維重建任務上有很好的表現，小到乒乓球，大到飛彈、天體都是有可能的應用範圍。

伍、參考資料

[1] A. Kyohei, N. Masamune, Y. Satoshi, The ping pong robot to return a ball precisely, Omron Tech. 51 (2020) 1–6.

[2] VPI - Vision Programming Interface: Pinhole Camera Model, https://docs.nvidia.com/vpi/appendix_pinhole_camera.html (accessed October 21, 2023).

[3] SLAM 入门之视觉里程计(2)：相机模型（内参数，外参数） - Brook_icv - 博客园, https://www.cnblogs.com/wangguchangqing/p/8126333.html (accessed October 21, 2023).

[4] 图像的变换, Weis Blog. (2017). https://VVingerfly.github.io/2017/08-13-CV-ImageTransformation/index.html (accessed October 21, 2023).

[5] E. Olson, AprilTag: A robust and flexible visual fiducial system, in: 2011 IEEE Int. Conf. Robot. Autom., 2011: pp. 3400–3407. https://doi.org/10.1109/ICRA.2011.5979561.

[6] J. Wang, E. Olson, AprilTag 2: Efficient and robust fiducial detection, in: 2016 IEEERSJ Int. Conf. Intell. Robots Syst. IROS, 2016: pp. 4193–4198. https://doi.org/10.1109/IROS.2016.7759617.

[7] G. Zhenglong, F. Qiang, Q. Quan, Pose Estimation for Multicopters Based on Monocular Vision and AprilTag, in: 2018 37th Chin. Control Conf. CCC, 2018: pp. 4717–4722. https://doi.org/10.23919/ChiCC.2018.8483685.

[8] D.E. Rumelhart, G.E. Hinton, R.J. Williams, others, Learning internal representations by error propagation, (1985).

[9] Understanding LSTM Networks -- colah’s blog, https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed October 21, 2023).

[10] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory, Neural Comput. 9 (1997) 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

[11] H. Li, H. Wu, L. Lou, K. Kühnlenz, O. Ravn, Ping-pong robotics with high-speed vision system, in: 2012 12th Int. Conf. Control Autom. Robot. Vis. ICARCV, 2012: pp. 106–111. https://doi.org/10.1109/ICARCV.2012.6485142.

[12] H. Li, S.G. Ali, J. Zhang, B. Sheng, P. Li, Y. Jung, J. Wang, P. Yang, P. Lu, K. Muhammad, L. Mao, Video-based table tennis tracking and trajectory prediction using convolutional neural networks, Fractals. 30 (2022) 2240156. https://doi.org/10.1142/S0218348X22401569.

[13] 謝宗倫，洪晟翔，李正宇，單相機高速運動物體 3D 軌跡重建的新方法, (2010).

[14] Z. Zhang, A flexible new technique for camera calibration, IEEE Trans. Pattern

Anal. Mach. Intell. 22 (2000) 1330–1334. https://doi.org/10.1109/34.888718.
[15] GT2 Timing Pulley 20 Tooth | 3D CAD Model Library | GrabCAD, https://grabcad.com/library/gt2-timing-pulley-20-tooth-3 (accessed October 21, 2023).
[16] HiWin MGW 12H linear guide rail | 3D CAD Model Library | GrabCAD, https://grabcad.com/library/hiwin-mgw-12h-linear-guide-rail-1 (accessed October 21, 2023).
[17] Lewansoul LD-27MG | 3D CAD Model Library | GrabCAD, https://grabcad.com/library/lewansoul-ld-27mg-1 (accessed October 21, 2023).
[18] NEMA 23 - 57BYGH56-401A | 3D CAD Model Library | GrabCAD,https://grabcad.com/library/nema-23-57bygh56-401a-1 (accessed October 21, 2023).
[19] Servo Bracket Mount Stand - Standard size x5 | 3D CAD Model Library | GrabCAD, https://grabcad.com/library/servo-bracket-mount-stand-standard-size-x5-1 (accessed October 21, 2023).
[20] raspberry-pi-4-product-brief.pdf, https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf (accessed October 21, 2023).
[21] TB6600 Stepper Motor Driver, https://www.dfrobot.com/product-1547.html (accessed October 21, 2023).

【評語】100035

1. 本作品利用視覺追蹤、深度學習與自動控制設計製作桌球機器人，內容豐富，符合科學精神，在運動訓練領域有應用潛力。

2. 題目建議應更精準地說明作品之主題。

3. 建議資料之分析可以強化，包含訓練資料之取得、桌球定位誤差之分析、桌球速度之估測與位置追蹤，機器人控制動作之動態等。