國立臺北大學資訊工程學系專題報告

非接觸式街機遊戲 – Ninja Strike Master

專題組員：陳柏翰、陳俊良、何定綸、吳少森

專題編號：PRJ-NTPUCSIE-111-03

執行期間：2022年09月至2023年05月

一. 摘要

本專題使用 Unity 開發了一個非接觸式街機遊戲，透過筆電的鏡頭取得連續的影像，將此連續影像利用 OpenPose[1]產生玩家的身體關鍵點，並利用 FCN(Fully Connected Network)進行訓練及預測不同丟球姿勢產生的落點，最後藉由預測的落點進行遊戲。

這套系統成功實現了一個基於機器學習的非接觸式街機遊戲，並為未來發展更多相關於這類遊戲和應用提供了一個有趣且具有潛力的範例。

二. 簡介

(一) 研製背景

在電子遊樂場中的有一種遊戲機台，以投擲塑膠球擊中螢幕上的目標作為遊戲進行方式(Fig. 1)，經過我們的觀察主要存在以下三個問題。第一，玩家不小心把球丟出機台外，導致塑膠球的遺失和環境整理的負擔增加。第二，在遊戲結束後，接觸過的每顆球上會殘留大量細菌，產生衛生問題。第三，塑膠球經常因為損壞需要更換，增加機台的維護成本。由於上述三個原因，我們決定設計一個非接觸式的遊戲系統。

Fig. 1: 遊戲機台

(二) 研究目標

我們希望設計一個遊戲系統，遊戲過程中玩家不需拿著實體的球，只需做出丟球的姿勢即可命中螢幕上的目標。玩家丟球姿勢的連續影像經由筆電的鏡頭當作輸入，透過 OpenPose 產生身體關鍵點，再將這些關鍵點當作輸入用機器學習的方式取得落點位置，藉此判斷是否擊中螢幕上的目標。

(三) 主要預期效益

玩家在做出丟球姿勢後，系統能計算出在螢幕上的落點，從而避免使用實體塑膠球，以減少病毒傳播風險，也降低業主維護成本。

三. 專題進行方式

(一) 系統環境

模型訓練使用 Python 3.10.11作為開發環境，並使用深度學習框架 Tensorflow 2.12.0來進行訓練。

遊戲運行需要有一台有鏡頭的筆電，並配合投影機來呈現遊戲畫面(Fig. 2)。

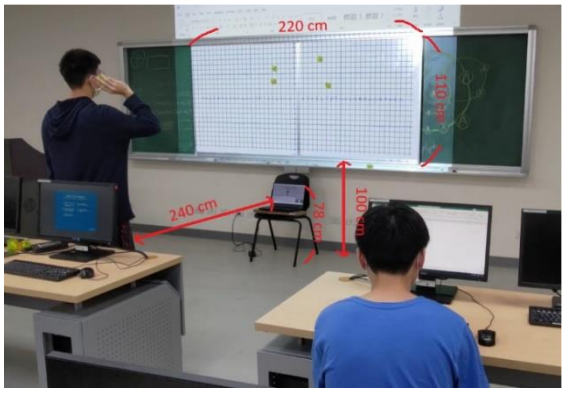

Fig. 2: 環境配置: 筆電高度: 78 cm、玩家與筆電距離: 240cm、遊戲畫面高度: 100cm、遊戲畫面大小: 220cm × 110 cm

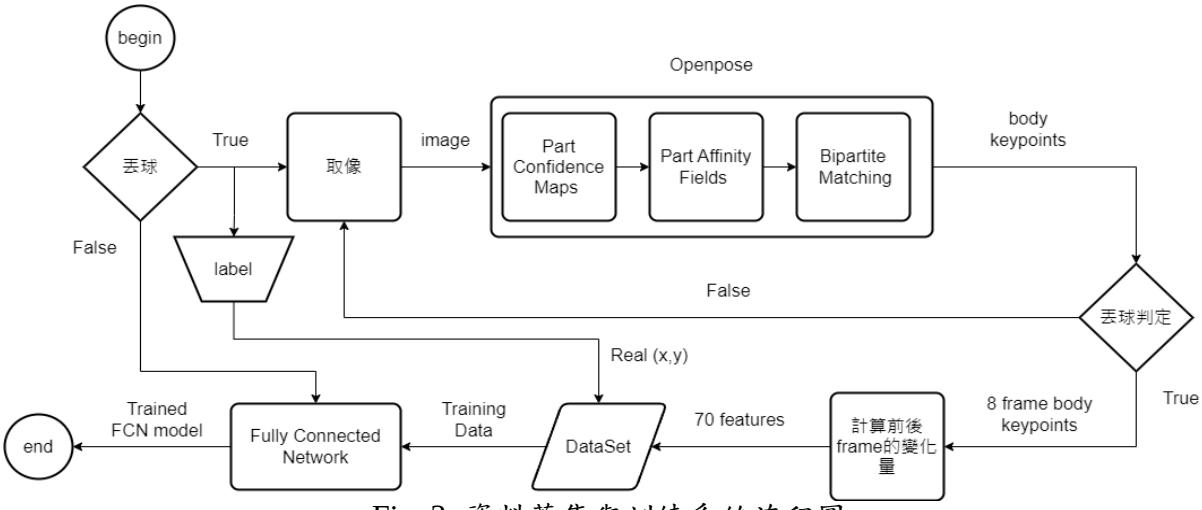

Fig. 3: 資料蒐集與訓練系統流程圖

系統的硬體配置如下，CPU: Intel Core i7-9750H 2.6GHz、記憶體: 16GB、鏡頭 fps(frames per second): 30、鏡頭解析度: 640×480 pixel。

系統的軟體配置如下，unity version: 2018.4.0f1(64-bit)、unity scripting runtime version: C# 6.0、unity api compatibility level: C# 4.0、openpose unity plugin: v1.0.0alpha-1.5.0。

(二) 系統架構

本專題可大致分成兩個系統: 1.資料蒐集與模型訓練系統，用來得出 FCN。2.遊玩系統，玩家實際遊玩時的系統，包括 UI 及計分方式。

1. 資料蒐集與訓練

資料蒐集架構如 Fig. 3所示，實驗人員丟球，Openpose 同時透過攝影鏡頭取得每一幀的影像，並將該影像處理成5個選取關鍵點(Fig. 4)，每一幀都會執行丟球判定，並從判定幀往前取8幀，經過資料預處理後，加上人工標註的實際落點作為訓練資料，每筆訓練資料為70 × 1的 vector，training dataset 共508筆，並使用 FCN 模型進行訓練。

1.1. Openpose

Openpose[1]的執行流程如 Fig. 5所示，將影像傳入 Openpose 後，它會先對攝影鏡頭所擷取到的畫面進行人體部位偵測，建立出2D 的 confidence maps，其代表著 Openp-ose 對人體各部位預測的位置。

同時，它也計算出各部位之間的 PAFs (Part Affinity Field)，產生2D 的向量場，其代表各個單獨部位該往哪個方向去連接，進而形成肢體。

最後，將 confidence maps 跟 PAF maps 做結合，透過 bipartite matching 的方式，將最有可能形成肢體的組別，也就是信心分數的最高相連接，最終形成一張完整的人體姿態圖(Fig. 4)。

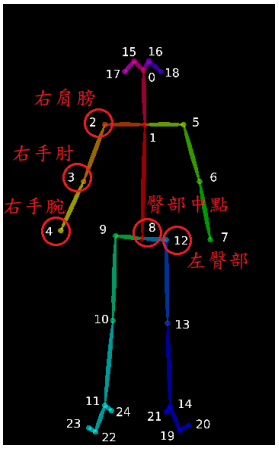

Fig. 4: Openpose 25個身體關鍵點[2]: 5個圈選點為選取關鍵點

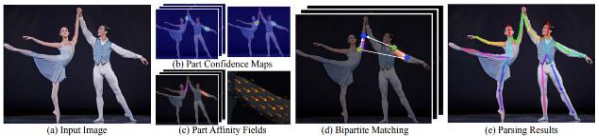

Fig. 5: Openpose 生成身體關鍵點流程[1]

1.2. 丟球判定

丟球是一種右手腕迅速遠離右肩膀的動作，而且右手腕應該全程高於右肩膀，所以我們的丟球判定主要是在觀察右手腕與右肩膀的距離，但由於當玩家丟球的目標接近鏡頭時，右手腕與右肩膀距離的變化並不明顯(Fig. 6)，所以我們還把右手肘與右肩膀的距離也納入考慮。

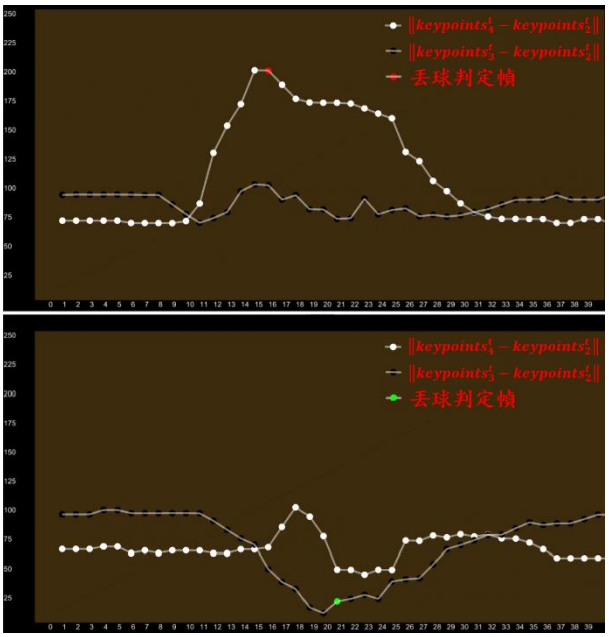

Fig. 6: 距離折線圖: 橫軸每一格為一幀，縱軸為距離。**Top:** 目標遠離鏡頭時的丟球趨勢。**Bottom:**目標接近鏡頭時的丟球趨勢。

經過觀察後，我們總結出正常的丟球動作需要依序經歷以下4種狀態(Fig. 7)，分別是 other, ready, throwing 以及 cool down。玩家初始狀態會在 other，當其通過準備判定(判定1)時，會進入 ready 狀態，而當其通過變化判定(判定2)時，會進入 throwing 狀態，當其首次未通過變化判定且通過出球判定(判定3)時，會被判定為丟球結束幀，並進入 cool down 狀態，在倒數15幀後，才會回到 other 狀態。

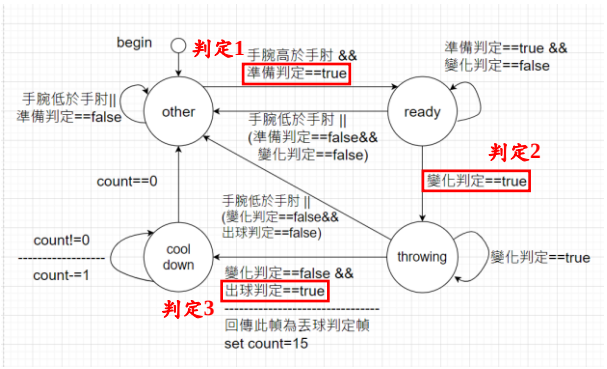

Fig. 7: 丟球判定狀態圖: 分隔線上下分別代表條件和行為。

各判定的條件如下列所述，其中 $keypoints_i^t$ 代表第 t 幀、編號 i 的關鍵點，關鍵點編號如 Fig. 4的編號。

判定1: 準備判定

$$\|keypoints_4^t - keypoints_2^t\| < \textit{手腕與肩膀準備閥值}$$
$$and$$
$$\|keypoints_3^t - keypoints_2^t\| > \textit{手肘與肩膀準備閥值}$$

判定2: 變化判定

$$\|keypoints_4^t - keypoints_2^t\| - \|keypoints_4^{t-1} - keypoints_2^{t-1}\| > 0$$
$$or$$
$$\|keypoints_3^t - keypoints_2^t\| - \|keypoints_3^{t-1} - keypoints_2^{t-1}\| < 0$$

判定3: 出球判定

$$\|keypoints_4^t - keypoints_2^t\| > \textit{手腕與肩膀出球閥值}$$
$$or$$
$$\|keypoints_3^t - keypoints_2^t\| < \textit{手肘與肩膀出球閥值}$$

在我們的環境配置中(Fig. 2)，手腕與肩膀準備閥值設為75，手肘與肩膀準備閥值設為55，手腕與肩膀出球閥值設為100，手肘與肩膀出球閥值設為40。

1.3. 資料預處理

我們將資料處理成逐幀的位移量，來避免玩家在遊玩的過程中，因為偏離正中間的位置，導致落點預測失敗的問題，以下是資料預處理的 pseudo code：

```
input: frames
numFrames=length(frames)
numKeypoints=length(frames[0])
result=Array((numFrames-1)*numKeypoints)
for i from 0 to numFrames-1 do
    for j from 0 to numKeyponts do
        variation=frames[i+1][j]-frames[i][j]
        append variation to result
    end for
end for
output: result
```

1.4. 訓練模型

我們的目標是利用我們蒐集來的資料集，來進行丟球落點的預測。

我們認為這個問題是一種具有複合性質的問題，因為我們的訓練資料包含了許多資訊，如時間序列、空間位置，以及關鍵點間的相關性等等。在此情況下，我們決定先採用 FCN 來解決此問題。

FCN 是一種全面性的模型，其每一個神經元都與前一層的所有神經元全連接，使得每一個神經元都能參考到輸入的全部資訊，從而發現 features 之間複雜的關係。

在模型層數選擇上，我們透過10-fold cross-validation 來計算 validation loss，最終根據結果選擇了 hidden layer 為3層的模型，其大小依序為256, 64, 256 (Fig. 8)

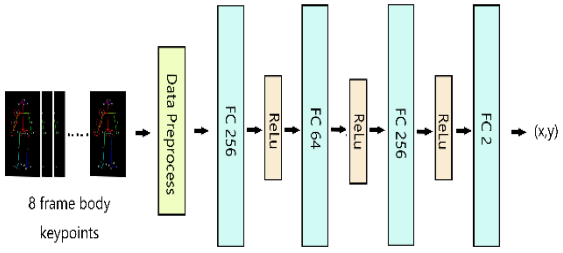

Fig. 8: FCN Model 架構圖

2. 遊玩系統

此遊戲的進行流程如 Fig. 9所示，系統讓玩家透過丟球選擇模式，兩模式會在一定時間後結束，並讓玩家選擇繼續與否。

2.1. 選擇模式

遊戲中分成兩個模式，玩家可以透過向左或向右丟來選擇，練習模式可讓玩家在正式遊玩前熟悉系統計算的落點，一般模式則是讓玩家正式開始遊戲。

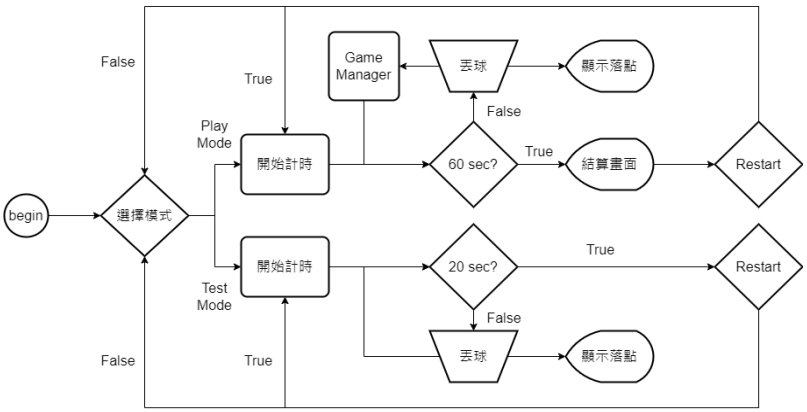

Fig. 9: 遊戲系統流程圖

2.2. Game manager

此程序負責遊戲進行中的座標轉換與數據統計。

(1) 座標轉換

每一球透過 FCN 預測的座標組，進行座標轉換至 Unity 的座標系。

$$Unity.coordinate = 9.1 \times FCN.predict$$

在 Unity 中的遊戲畫面大小是 400×200，在訓練系統中的丟球範圍是 44×22，因此轉換的比例為9.1。

(2) 數據統計

計算擊中不同目標的球數與得分。遊戲內有兩種目標，命中正確目標依照範圍得1或2分，擊中錯誤目標扣1分(Fig. 10)。

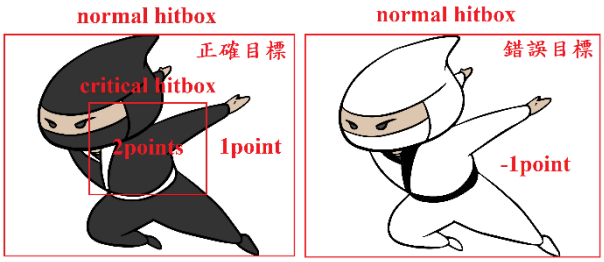

Fig. 10: 得分區塊示意圖

2.3. 結算畫面

遊戲結束後，會顯示當局統計數據(Fig. 11)，並在5秒後選擇是否重新開始。

Fig. 11: 結算畫面: CriticalHit、NormalHit、BombHit、Miss 分別是得2分、得1分、扣1分、未命中任何目標的球數。

$$\textbf{\textit{HitRate}} = (CriticalHit + NormalHit) \div (CriticalHit + NormalHit + BombHit + Miss)$$

$$\textbf{\textit{Score}} = CriticalHit \times 2 + NormalHit \times 1 - BombHit \times 1$$

(三) 主要困難與解決辦法

1. FPS 過低

一開始的方案是將遊戲畫面呈現在筆電螢幕上，並使用手掌的關鍵點作為訓練資料，但由於同時捕捉身體與手掌會導致 Openpose 計算量龐大，使 FPS 過低(Fig. 12)，難以在小於1秒的丟球過程中取得足夠的資訊，導致訓練結果不理想。

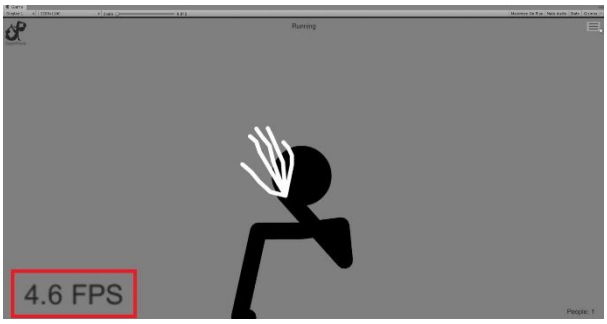

Fig. 12: 舊版執行畫面

為了解決這個問題，我們取消了手掌的計算，FPS 因此從4.6上升至19.7(Fig. 13)，同時為了在只有身體計算時也能取得足夠資訊，將距離拉遠並使用投影機放大遊戲畫面(Fig. 2)，使玩家丟球姿勢的變化更大。

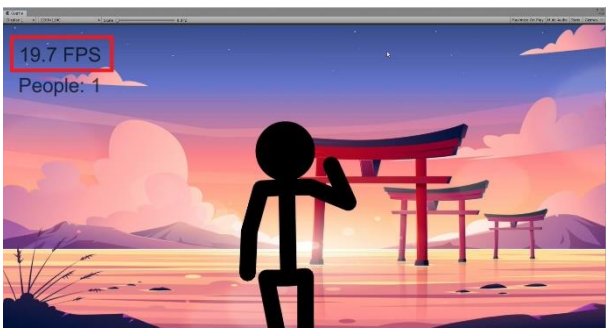

Fig. 13: 新版執行畫面

2. 訓練資料蒐集困難

我們蒐集資料的方式是將吸盤球丟至白板上，搭配上人工觀察實際落點，並記錄在資料檔中，如 Fig. 2之配置圖。

訓練環境選用一間有投影設備的教室。由於此實驗會製造不小的聲響，加上需配合組內成員平時上課的時間，因此只能選擇休假日的半夜進行實驗。

實驗過程中，吸盤球並不如我們所想的能牢固地吸附在白板上，時常不到一秒就掉落，有時甚至吸不上去，有效資料僅30~40%，這導致訓練員需丟1000~2000次才能拿到預想的訓練資料量。

第一次實測系統後，發現系統的左與右區分得不夠好(Fig. 14)，所以我們更改實驗環境配置，將鏡頭的位置從左斜前方改成正對玩家(Fig. 15)。由於角度變動，關鍵點之間的關係會隨之改變，需要重新蒐集訓練資料。

效率上的低落、時段上的限制，以上種種原因，造成我們最終花費1個多月的時間才將訓練資料蒐集完畢。

Fig. 14: 鏡頭在左斜前方的配置，左右差別不明顯。**Left:** 向左丟。**Right:** 向右丟。

Fig. 15: 鏡頭正對玩家的配置，較能區分左右。**Left:** 向左丟。**Right:** 向右丟。

四. 主要成果與評估

在遊戲系統完成，並將訓練模型的參數引入遊戲後，我們4人各自測試了3局，最終的平均命中率是63.8%，考慮到遊戲本身的變化性，加上實際丟球時的誤差，我們認為此平均命中率已足夠讓玩家有良好的遊戲體驗。

	總球數（顆）	總得分（分）	命中數（顆）	得分命中率（%）
玩家1	104	80	73	70.2
玩家2	129	107	87	67.4
玩家3	131	83	74	56.5
玩家4	144	87	90	62.5
累計	508	357	324	-
平均	127	89.3	81	63.8

TABLE 1: 丟球結果: 每筆數據為3局的統計

五. 結語與展望

此專題透過 Openpose、Tensorflow、Unity 等多個開發工具，完成了一個基於機器學習的非接觸式街機遊戲，可以在兼顧衛生之餘，同時降低機台維護成本。

未來我們期望能增加更多的訓練資料，使玩家也能用左手遊玩此機台，並針對不同的丟球習慣與姿勢，設計出丟球起始幀統一的判定標準，與現有的丟球結束幀做結合，使系統能更準確的預測出落點。

六. 銘謝

感謝指導教授在專題中的用心指導和指引研究方向，在每周的會議上針對目前的研究內容提供寶貴的建議和指正，協助我們完成專題。

七. 參考文獻

[1] Z. Cao, G. Hidalgo, T. Simon, S. -E. Wei and Y. Sheikh, “OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields,” in *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 1, pp. 172-186, Jan. 2021.

[2] OpenPose body keypoints, https://cmu-perceptual-computing-lab.github.io/openpose/web/html/doc/md_doc_02_output.html

[3] OpenPose, https://github.com/CMU-Perceptual-Computing-Lab/openpose

[4] OpenPose's Unity Plugin, https://github.com/CMU-Perceptual-Computing-Lab/openpose_unity_plugin

[5] M. Nielsen, “Neural Networks and Deep Learning,” http://neuralnetworksanddeeplearning.com/chap4.html, Dec. 2019.

[6] S. Haykin, “Neural Networks and Learning Machines,” 3rd edition, Pearson Prentice Hall, USA, pp.198-199, 2009

[7] Keras, Keras: The Python deep learning API, https://keras.io/

[8] Ninja picture, designed by 588ku, https://zh.pngtree.com/freepng/cartoon-hand-drawn-pattern-japanese-ninja-handsome-ninja-illustration-darts_3931767.html?sol=downref&id=bef

[9] Background picture, designed by Pikisuperstar - Freepik.com, https://www.freepik.com/free-vector/set-torii-gates-water_9925874.htm

[10] Sound effects, created by Taira Komori, https://taira-komori.jpn.org/freesoundtw.html