生醫實驗期末專題
第二組

Air-Ring Gesture Recognition: Improving Workflow with Shortcut Keys

陳祈安	柯岱佑	翁證騏
B08901210	B07901149	B08901009

1 Introduction

非實體接觸的控制一直是現在的趨勢，比起過去的按鈕、觸控螢幕，已有許多替代方案能提供更有彈性、更便利的控制方式，如電腦視覺（Computer Vision, CV）在諸多應用都已達到能商用的水準，但電腦視覺也存在許多難以克服的問題，如對隱私的侵害、照相機視角遮蔽的局限。因此，在某些應用中，電腦視覺不會是最好的選擇。

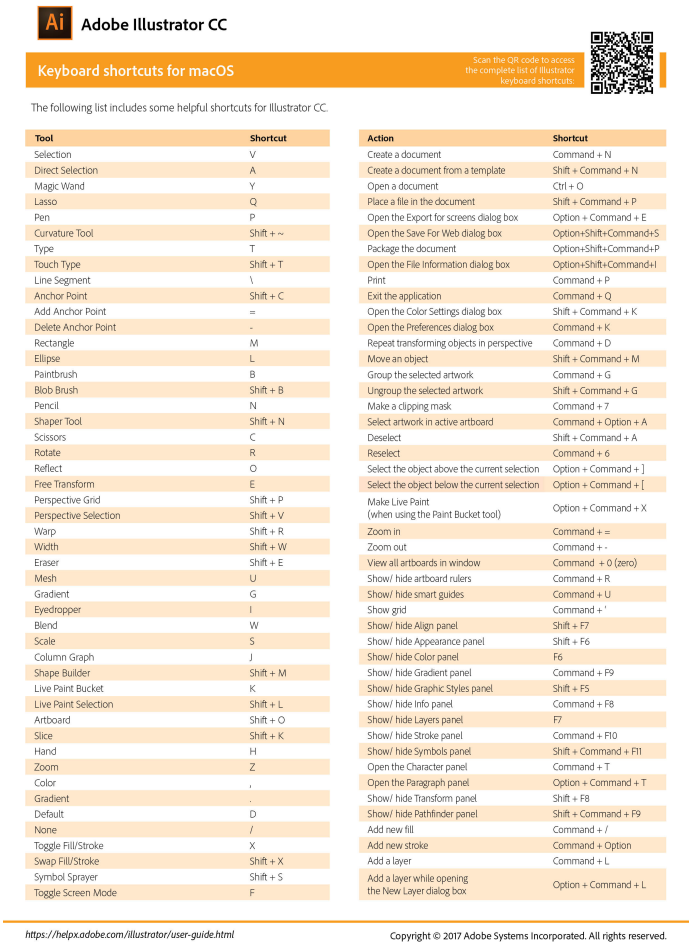
參考過去的資料，我們發現使用九軸感測器（LSM9DS1）在手勢辨識（Gesture recognition）上可以達到令人滿意、不輸 CV 的準確率。九軸感測器由三軸加速度感應器（Accelerometer）、三軸陀螺儀（Gyroscope）、三軸磁場感測器（Magnetometer）構成，手勢執行時，通常伴隨明顯的移動和轉動，同手勢在不同人之間，透過適當的資料處理，也能捕捉到類似的特徵（pattern）。因此，利用九軸感測器固定在身體部位上，理應能打造出不侷限特定用戶（user-independent）的系統，取代 CV 進行非實體觸碰的控制。

此專題中，我們將九軸感測器以指環的形式固定在手指上，作為一個戒指形式的商品雛形（Air-Ring），並定義出八種能輕易操作的手勢（U, D, L, R, O, V, Z, N），蒐集超過 5000 筆資料後，使用 CNN (Convolution Neural Network) 架構訓練，配合套件 PyAutoGUI，實現即時的手勢辨識進行鍵盤控制。在訓練資料上，達到 95% 以上的準確率，並能在 60 秒內，正確辨識 30 次不同的手勢。

2 Motivation

精準的手勢控制是諸多應用的基礎，我們希望在完成穩定、即時的辨識系統後，藉由手勢進行鍵盤控制，能不實際觸摸到裝置的情況下，使用手勢隔空控制鍵盤觸發快捷鍵，達到提升工作效率的效果。

市面上各種軟體的功能越來越複雜，快捷鍵表越來越長（例：Fig. 1），幾乎沒有人能完整記下所有快捷鍵，許多快捷鍵也往往需要同時按下超過三個鍵才能觸發，在工作時常有少一隻手用的困擾。因此，若能在工作前快速設定接下來所用軟體的特定快捷鍵，就有機會使用這套系統，讓工作流程獲得改善。



Adobe Illustrator CC

Keyboard shortcuts for macOS

Scan the QR code to access the complete list of Illustrator keyboard shortcuts:

The following list includes some helpful shortcuts for Illustrator CC.

Tool	Shortcut
Selection	V
Direct Selection	A
Magic Wand	Y
Lasso	Q
Pen	P
Curvature Tool	Shift + ~
Type	T
Touch Type	Shift + T
Line Segment	\
Anchor Point	Shift + C
Add Anchor Point	=
Delete Anchor Point	-
Rectangle	M
Ellipse	L
Paintbrush	B
Blob Brush	Shift + B
Pencil	N
Shaper Tool	Shift + N
Scissors	C
Rotate	R
Reflect	O
Free Transform	E
Perspective Grid	Shift + P
Perspective Selection	Shift + V
Warp	Shift + R
Width	Shift + W
Eraser	Shift + E
Mesh	U
Gradient	G
Eyedropper	I
Blend	W
Scale	S
Column Graph	J
Shape Builder	Shift + M
Live Paint Bucket	K
Live Paint Selection	Shift + L
Artboard	Shift + O
Slice	Shift + K
Hand	H
Zoom	Z
Color	,
Gradient	.
Default	D
None	/
Toggle Fill/Stroke	X
Swap Fill/Stroke	Shift + X
Symbol Sprayer	Shift + S
Toggle Screen Mode	F

Action	Shortcut
Create a document	Command + N
Create a document from a template	Shift + Command + N
Open a document	Ctrl + O
Place a file in the document	Shift + Command + P
Open the Export for screens dialog box	Option + Command + E
Open the Save For Web dialog box	Option+Shift+Command+S
Package the document	Option+Shift+Command+P
Open the File Information dialog box	Option+Shift+Command+I
Print	Command + P
Exit the application	Command + Q
Open the Color Settings dialog box	Shift + Command + K
Open the Preferences dialog box	Command + K
Repeat transforming objects in perspective	Command + D
Move an object	Shift + Command + M
Group the selected artwork	Command + G
Ungroup the selected artwork	Shift + Command + G
Make a clipping mask	Command + 7
Select artwork in active artboard	Command + Option + A
Deselect	Shift + Command + A
Reselect	Command + 6
Select the object above the current selection	Option + Command +]
Select the object below the current selection	Option + Command + [
Make Live Paint (when using the Paint Bucket tool)	Option + Command + X
Zoom in	Command + =
Zoom out	Command + -
View all artboards in window	Command + 0 (zero)
Show/ hide artboard rulers	Command + R
Show/ hide smart guides	Command + U
Show grid	Command + '
Show/ hide Align panel	Shift + F7
Show/ hide Appearance panel	Shift + F6
Show/ hide Color panel	F6
Show/ hide Gradient panel	Command + F9
Show/ hide Graphic Styles panel	Shift + F5
Show/ hide Info panel	Command + F8
Show/ hide Layers panel	F7
Show/ hide Stroke panel	Command + F10
Show/ hide Symbols panel	Shift + Command + F11
Open the Character panel	Command + T
Open the Paragraph panel	Option + Command + T
Show/ hide Transform panel	Shift + F8
Show/ hide Pathfinder panel	Shift + Command + F9
Add new fill	Command + /
Add new stroke	Command + Option
Add a layer	Command + L
Add a layer while opening the New Layer dialog box	Option + Command + L

https://helpx.adobe.com/illustrator/user-guide.html

Fig. 1. Adobe Illustrator, Keyboard Shortcuts for MacOS

此外，鍵盤控制也能在特定場合客製化符合特定用途，如使用 Netflix 看劇時，搭配外掛套件 Video Speed Controller (Fig. 2)，也能用手勢控制觸發鍵盤，做到像遙控器般的應用。結合以上，如果我們能讓裝置夠輕便，控制準確度、延遲夠低，配戴上負擔不高，就能配合每個軟體和外掛套件，提供一個高度彈性的裝置，讓用戶依據自己需求完成不同的任務。

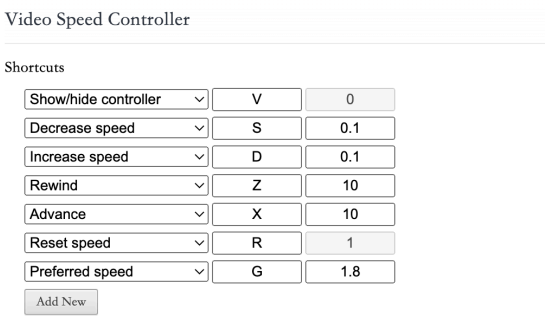


Fig. 2. Video Speed Controller 控制畫面

3 System Overview

為了達到高度彈性，我們寫了一個簡單的前端網站（https://chian-chen.github.io/BELab-Final/）方便用戶進行快捷鍵設定，完成的系統流程大略如下圖（Fig. 3）表示。

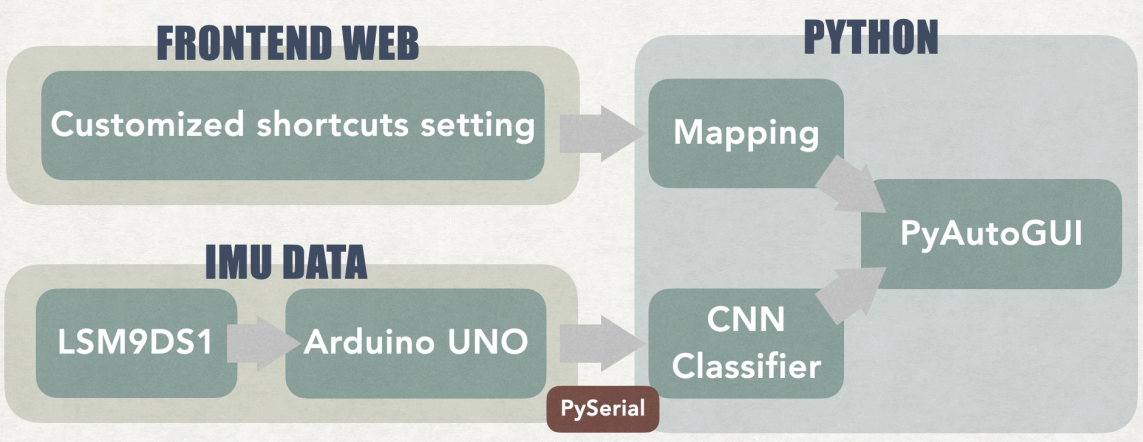


Fig. 3. System Flowchart

首先，由用戶依據自己的需求，使用前端網站進行設定，將我們定義的手勢對應到特定的快捷鍵後生成設定的.json 檔，隨後將產生的檔案放到 python 程式執行的路徑；九軸感測器（IMU）蒐集到的資料，則藉由套件（https://github.com/arduino-libraries/Arduino_LSM9DS1）和 Arduino UNO 連結，再使用 PySerial 連結 Python 和 Arduino，隨後透過訓練的 CNN 模型產生結果，配合設定後 Mapping 成特定的快捷鍵組合，送到 PyAutoGUI 完成手勢到鍵盤控制。

4 Data Collection

我們將九軸感應器利用醫療透氣膠帶與矽膠指環固定在手指上，蒐集手指移動時產生的三軸加速度，以 60Hz 取樣蒐集 150 點的加速度時域訊號（費時 2-3 秒）。手勢的動作有 8 種，分別為：上、下、左、右、N、Z、V 與 O。為了達成實際投入工作的模擬，我們的系統必須將所有非手勢的訊號都歸類為雜訊，例如：手部靜止、使用鍵盤或觸控板、拿取物品等常見的晃動。因此，我們添加了額外的第 9 種類別：雜訊（Noise）。

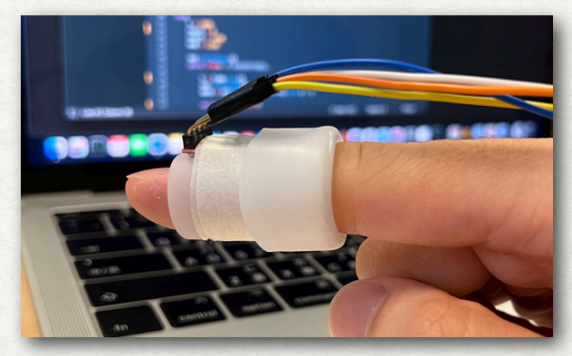


Fig. 4. 配戴裝置示意圖

最初我們使用九軸感測器的六軸進行實驗，但由於磁軸的取樣頻率和另外兩者不同（Fig. 5），再加上購入的九軸感測器在陀螺儀發生了故障，因此，我們最終只採用九軸裡的三軸加速度計進行資料蒐集和辨識。

nr	setAccelODR(nr)	setGyroODR(nr)	setMagnetODR(nr)
0	Gyro&Accel off	Gyro off	0.625 Hz
1	10 Hz	10 Hz	1.25 Hz
2	50 Hz	50 Hz	2.5 Hz
3	119 Hz	119 Hz	5 Hz
4	238 Hz	238 Hz	10 Hz
5	476 Hz	476 Hz	20 Hz
6	don't use	don't use	40 Hz
7			80 Hz
8			400 Hz

Fig. 5. Sample rate for the LSM9DS1

此外，為了降低不同使用者所產生的訓練資料的差異，我們對每個類別的手勢都有明確的定義。使用者可以想像一個虛擬的九宮格在前方，並將手指依序移動到相對應的位置，定義的手勢和執行時的步驟如附圖（Fig. 6），蒐集到的資料型態則如 Fig. 7 所示。

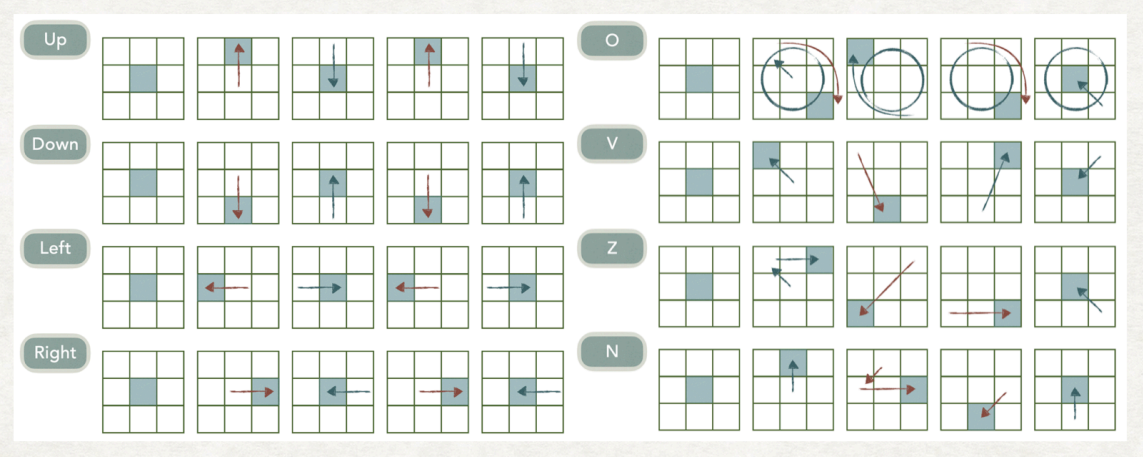


Fig. 6. 手勢執行步驟，紅色箭頭標示處為主要實施加速度的位置

為了避免模型 over-fitting 與增加更多元的訓練資料以提高模型精準度，在蒐集完訓練資料後我們會再進行資料增強（Data Augmentation）產生更多資料。對於每筆手勢的的訓練資料，我們會以一定權重加上蒐集到或利用 Gaussian distribution 隨機產生的雜訊。對於雜訊類別，我們利用 Gaussian distribution 隨機產生的雜訊代表靜止狀態的雜訊（Fig. 8）。

在實作即時辨識演算法時，模型會不斷將信號當作輸入進行辨識，容易造成模型將接收到手勢信號前半段後，就進行錯誤判斷。所以我們隨機從蒐集到的雜訊資料取 50 點（Fig. 8 左圖紅色部分），再加上經過前處理的訓練資料前 30 點（Fig. 8 左圖藍色部分），連接在一起成新的雜訊資料。

我們總共蒐集了約 5700 筆訓練資料，其中每個手勢資料約有 450 筆，雜訊資料約 2000 筆。可參考 github repo 中的資料夾 data 和 imgs，分別有原始資料（.npz）和視覺化後的圖片（.png）。

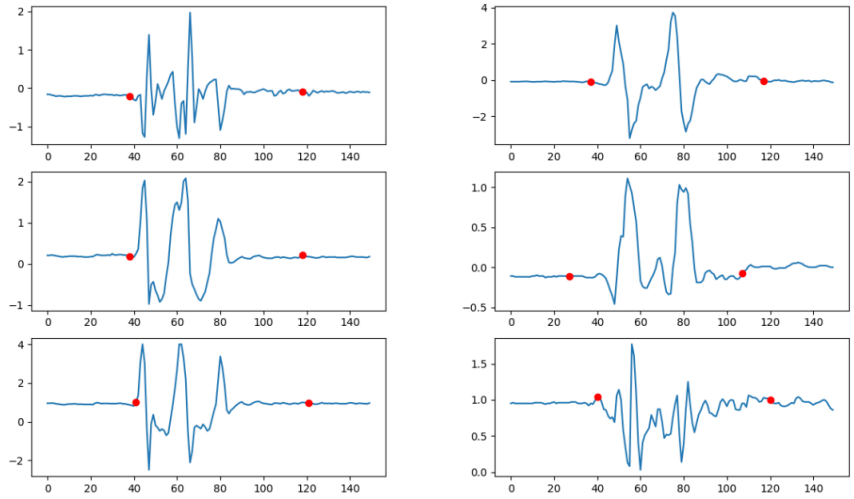


Fig. 7. Data Sample, 左邊為手勢 Up, 右邊為手勢 Right

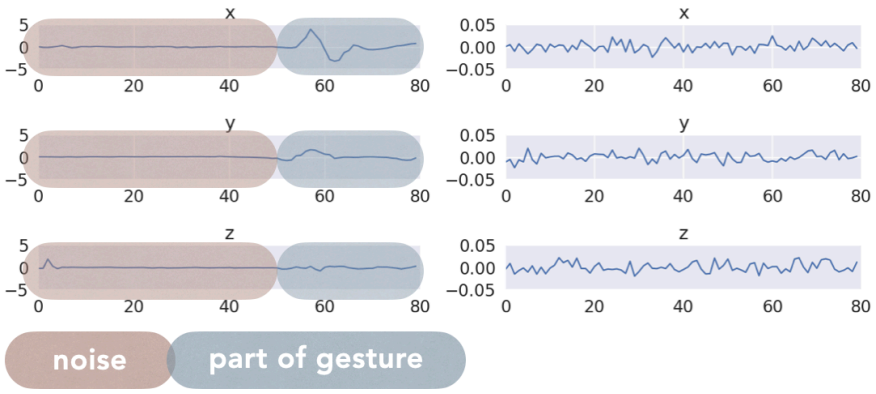


Fig. 8. Data Augmentation, 左邊為雜訊混合資料的方式，右邊為 Gaussian distribution Noise

5 Preprocessing

在蒐集資料時，為了方便使用者操作動作與完整涵蓋手勢信號，我們將信號的長度設為 150 個點。但實際輸入到模型辨識不需要這麼長的信號且其中許多值都是沒有變化的。故進行前處理的目的是為了找出有重要意義的信號，並且減少輸入維度（Dimension reduction），避免訓練資料在高維空間過於稀疏，進而導致模型表現不佳。

以下圖的原始信號為例，可以觀察到蒐集到的信號中有意義的部分只占一部分，許多信號都是對辨識沒有幫助的雜訊。為了找出信號中富有資訊的區間，我們利用固定長度為 80 點的滑動窗口（Sliding window）去尋找能量 $p_{i,j}$ 最大的窗口，能量的定義為信號點平方的總和。

$$p_{i,j} = \sum_{k=i}^{j} p_k^2$$

以下圖（Fig. 9）為例，演算法所找到的窗口是兩個紅點的區間，明顯並不能夠代表這個信號的區間。原因是當手勢靜止時，感測器蒐集到的信號並不是 0，而是一段非零且變化極小的信號，進而造成最大能量的區間沒有找到信號變化最劇烈的地方。

為了解決這個問題，我們將信號減掉整個信號的平均值讓處理後的信號的平均變成 0，再進行前述滑動窗口尋找信號最大能量區間的演算法。從下圖（Fig. 9）的橘色信號與綠色點可以看到減掉平均後，滑動窗口找到的區間是信號中變化最大的部分。代表我們成功將輸入到模型的信號長度從 150 點減少到 80 點。

信號變成縮短 80 點後，我們可以直接用時域的信號當作模型的輸入、或是對時域的信號進行一維的小波轉換再輸入進模型。根據我們的實驗結果兩者的表現相差不大，因此選擇以不需額外處理的時域訊號當作輸入。

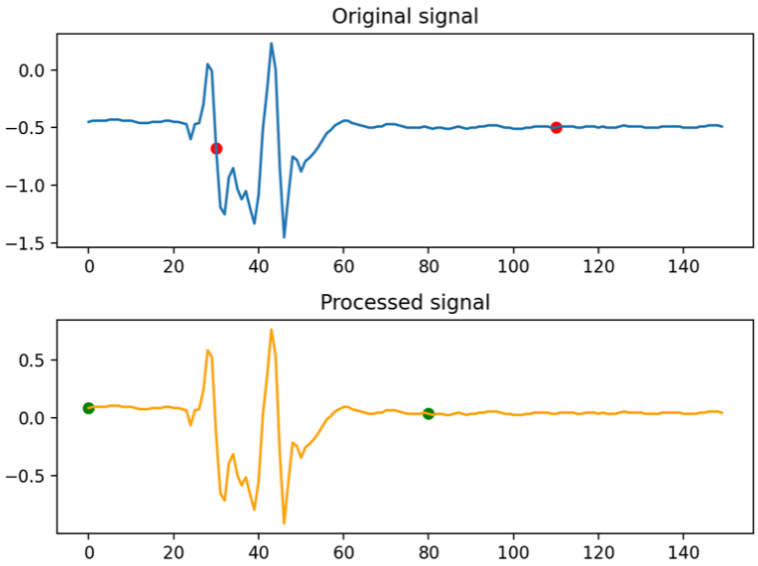


Fig. 9

6 Model

手勢辨識的模型主要採用兩種類型的模型，分別為 SVM 與 CNN，其中 CNN 模型的準確度較好，可以達到超過 97% 的準確率。使用 SVM 時，會將三維 X, Y, Z 軸的手勢信號連接在一起，並對它進行 PCA 降維成 10 維，再將所有資料進行訓練。在一開始資料只有蒐集數百筆時，SVM 的辨識正確率可以到達 84%，相比同樣資料量時的 CNN 模型的正確率只有不到 80%。但是當資料蒐集到 5000 筆以上時，SVM 模型的正確率卻只有不到 50%，而 CNN 的模型正確率可以穩定超過 95%。

CNN 模型能夠透過 filter 找出更大範圍的局部特徵，在許多任務上皆有成功的表現。在這個實驗中的 CNN 模型一開始會利用 1x16 的 Filter 先對每個軸的加速度進行運算並產生 96 個 Feature map，再將所有所有 feature map 的資訊全部拉直，再利用數層 Linear layer 進行分類。得到輸出結果後，再利用 cross entropy 函數計算模型輸出的 loss，並利用 Adam optimizer 進行模型參數調整，總共訓練 10 epochs。為了避免模型 over-fitting，除了利用 Data augmentation 增加訓練資料

外，也同時使用更少的模型參數與加上 Dropout layer 層。Model 的架構如 Fig. 10 所示。

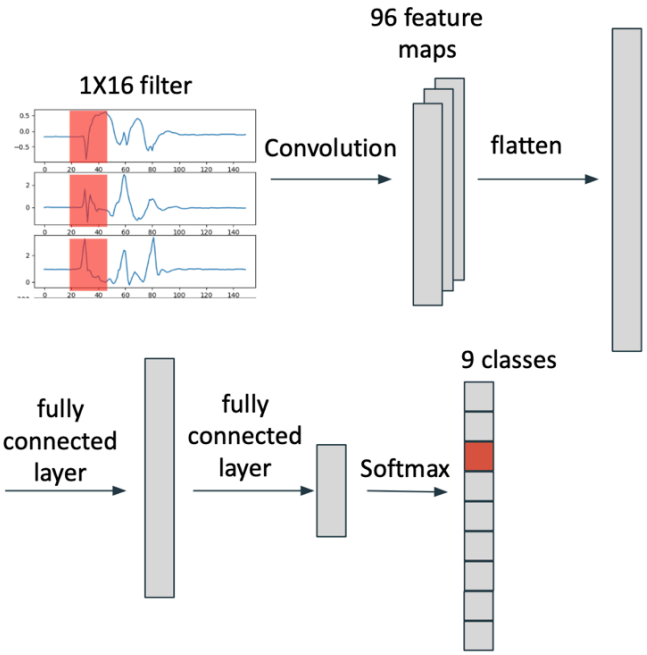


Fig. 10. Model Architecture

CNN 模型在測試資料的準確度可以到達 97% 左右，從測試結果的 Confusion matrix (Fig. 11) 可以看到模型對於每個 Class 的正確率都十分地高，但有時候會個別將左和右、Z 和 N 的手勢混淆，觀察這些手勢的信號可以發現他們有些相似度，可能是因為不同使用的手指用力的程度不一造成模型辨識錯誤。

7 Real-time Algorithm

我們不斷儲存最新的 150 筆資料，並且在每次蒐集到 1 筆資料後，在這 150 筆資料中找出能量最大的 80 個點，並將這 80 個點輸入進 model，得到一個分類的結果。而如果結果不為雜訊，我們就執行相對應的快捷鍵設定。

Real-time 資料蒐集：

三個不斷更新、蒐集最新 150 個點的 queue（分別蒐集 x, y, z 方向的加速度，個別歷時約 2.5 秒）

流程：

1. 從 150 個點的 queue 找到能量最大的 80 點連續片段（此處和上述訓練模型 training, testing data 的蒐集所作之操作相同）。
2. 找到的 x, y, z 三個 80 點連續片段，若有任何一個是屬於該 queue 的最後 5 個連續片段其中之一，則我們視其為可能的未完成手勢動作，並因此不做以下操作，直接視為雜訊。

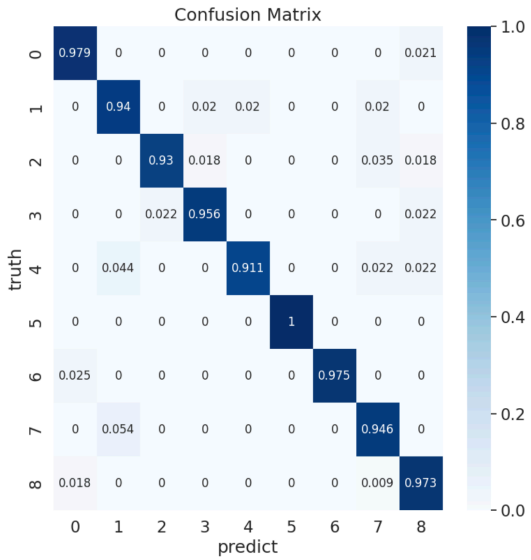


Fig. 11. Confusion Matrix

```
x = deque([0] * 150, maxlen=150)
y = deque([0] * 150, maxlen=150)
z = deque([0] * 150, maxlen=150)
```

Fig. 12. queue, 蒐集 x, y, z 軸的資料

(150 個點的 queue 共有 150 - 80 = 70 個 80 點連續片段可以選擇，最後 5 個連續片段指的是這 70 種可能的最後 5 個。至於 5 這個數字的決定是於測試時，權衡正確率與延遲的結果。)

3. 若三個 80 點連續片段的能量都未超過設定的閾值，則同樣不做以下操作，並視為雜訊。
4. 將這 80 個點輸入進 model，得到一個分類的結果（結果會是某一種手勢動作或雜訊）。
5. 若結果不為雜訊，則執行相對應預先設定好的快捷鍵設定，並清空 queue，以避免重複執行相同快捷鍵設定的情況。

細節說明：

1. 快捷鍵：
   快捷鍵操作我們使用了 python 的 PyAutoGUI module，而手勢動作對應到快捷鍵由 user 設定，並使用 dictionary 存取。在得到手勢動作後，我們從 dictionary 中找出其對應的快捷鍵，並利用 PyAutoGUI hotkey 的 function 操作。

2. queue 的重新初始化：
   當我們清空 queue 後，從測試實驗中，我們發現 model 容易在收了一定數量的 data 後，誤判雜訊為手勢動作。實際做測試畫圖找其中原因後，我們發現重新初始化 queue 為 0 不太合理，因為 IMU 的 data 會有一 offset，並且每個軸都不太一樣。因此，為正確初始化 queue 以避免因 offset 或雜訊而導致的錯誤判斷，我們清空 queue 後，先接續蒐集 10 個 data（每一軸）。在蒐集這 10 個 data 的期間，我們不會輸入 queue 進去 model，而這也不會影響實

際的操作或造成延遲，因 10 個 data 約為 0.2 秒，而手勢動作持續的時間會大於這個時間。蒐集 10 個 data 後，我們先計算這 10 個 data 的平均，配合預先設定的標準差數值（0.01）產生 140 個 Gaussian 雜訊 data，並把這 140 個 data 填到 queue 的前面，配合蒐集到的 10 個 data，構成新的 queue，完成 queue 的初始化。

8 Evaluation

我們採用兩種不同的方式呈現結果，分別是測試手勢判斷速度，和真實使用情況的模擬，並分別呈現在展示影片（https://youtu.be/3UDJmE8ajxY ）中的 Part2 和 Part3。

8.1 速度評估（Speed Testing）

在速度的評估上，我們借用打字網站（https://monkeytype.com/）的客製化設定，測試完成三十次正確手勢判斷需要花費的時間。手勢（Up, Down, Left, Right, O, V, Z, N）分別取字首對應到（U, D, L, R, O, V, Z, N），畫面如圖（Fig. 13）所示。

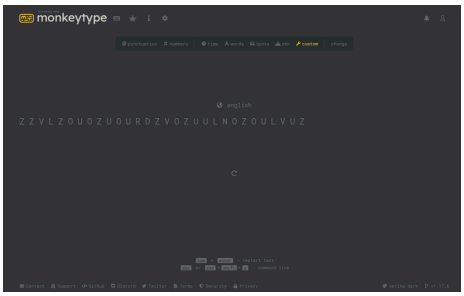


Fig. 13. 利用 Monkeytype custom 設定測試速度

在發生手勢動作前，data 理應擁有較小的能量，因此在我們的演算法中，如能量沒有超過設定的閾值，是不會被輸入進 model 中的。因此，在沒有觸發手勢時的延遲只包括 data 的前處理、資料經過 model 的時間。若有判斷到手勢動作，時間便會再加上 PyAutoGUI 模組控制鍵盤的時間。我們量測各功能軟體執行的時間。前處理、簡單的邏輯判斷和模型判斷基本上都是可忽略的。主要的時間延遲發生在 PyAutoGUI 快捷鍵的執行。嘗試尋找其他操作快捷鍵的模組後，我們發現 PyAutoGUI 已經是相對快的方式，其他的模組有些能提供更複雜的功能，但相對地，執行速度更慢，不符合我們的需求，因此在軟體上對速度的優化暫時無法克服。

最終大約能在 60 秒前後完成 30 個正確的手勢判讀，也就是大約兩秒一個。此速度和我們在資料蒐集時所採取的措施密切相關，由於在蒐集資料時，我們採用約 60Hz 的速率取 150 個資料點（2-3 sec）作為訓練資料，因此在即時系統上，如果沒有用類似的方式取資料，往往會在正確率上大打折扣，難以透過改進演算法進一步突破其限制。

Table 1: 程式執行速度

資料處理（雜項）	<0.001s
模型判斷（CNN Model）	0.003 - 0.005s
快捷建操作（PyAutoGUI）	0.1s

8.2 實際使用模擬

我們使用軟體 Photoshop 作為示範的工具，從影片的第二部分可以看到，分別使用了五組快捷鍵。選用的都是使用上不那麼普遍的快捷鍵。可以看到，在實際使用時，我們的系統確實能抵抗正常工作的雜訊避免誤判，在速度和準確度上也有不錯的成果。快捷鍵和手勢的對應如下表(table 2)。

Table 2: Shortcuts Mapping for Photoshop Demo

Gesture Up	command + shift + E
Gesture V	command + shift + S
Gesture O	command + shift + U
Gesture N	command + O
Gesture Z	command + U

9 問題與討論

1. 是否考慮不同使用者的手勢的動作與速度的差異性？

由於我們所設定的幾個手勢有高度的類似性，例如向左的手勢在順時針方向旋轉後其實就是向上的手勢，因此如上述，我們清楚定義了手勢動作的執行力道與方向。

另外，手勢速度的差異性的確是我們一開始在設計演算法時有思考到的部份。因此，我們曾經試過分別取 sliding window 後 150、100、50 個點，且各自去做前處理與模型判斷，最後三者所得到的非雜訊結果再進行投票（若票數相同，則優先順序為 150 > 100 > 50）。針對 150、100（>=80）個點的 data，我們都是在這些點中取能量最大的連續 80 個點。對於 50（< 80）個點，我們則是 resample 他們到 80 個點。

然而，從實驗結果來看，我們發現 50 個點的 data 容易被誤判為手勢。而造成這樣的現象，我們認為和資料本身的不完全有關，亦即從 50 個點 resample 到 80 個點和蒐集真正的 80 個點還是有差異。另外，我們也發現撇除掉上述的誤判，大部分的投票結果幾乎和 150 個點所產生的結果吻合。綜合我們認為我們測試時的手勢已經足夠快速及上述結果，我們最終捨棄掉這個方法，也能達到很高正確率。

2. 為何使用九軸感測器而非 EMG 電路？

在實驗二接 EMG 電路時我們發現不同人的肌電訊號強弱差異很大，剛好組員們的肌電信號都很弱，造成在實驗二測試電路時狀況不斷。當我們決定要做手勢辨識時，考量到手指肌肉產生的肌電訊號可能更小，我們決定採用不需要用到肌電訊號的解決方案：九軸感測器，如此便能夠讓所有使用者都能夠容易地使用這個應用。

3. 為何在訓練時模型正確率很高但實際使用時仍會出現辨識錯誤？

這是由於訓練資料與實際產生資料的分佈不一樣，蒐集訓練資料時手部的位置是固定的，但實際使用時手部可能順、逆時針的角度旋轉，造成雖然是同樣的手勢卻辨識錯誤。此外，由於 CNN

模型的能力對於此任務而言十分強大，造成訓練時容易發生 over-fitting 問題，導致在訓練資料和測試資料都表現很好卻在實際測試時表現稍不如預期。若要改進模型正確度可蒐集更多手部旋轉角度的訓練資料，讓模型的訓練資料更多元，模型更不容易發生 over-fitting。

10 Conclusion

本專題使用九軸感測器（LSM9DS1）連接 Arduino UNO 開發板，製造了手勢控制鍵盤裝置 Air-Ring 的雛型。藉由九軸感測器蒐集移動時產生的加速度信號，並將信號經過前處理後輸入訓練好的 CNN 模型進行手勢辨識。在經由不同的實驗與嘗試，不斷改善使延遲速度和準確率上升後，我們在 CNN Model 上達到超過 95% 的準確度，在速度方面，達到每 60 秒可連續辨識正確 30 個手勢。

利用我們撰寫的前端網站，可方便的產生快捷鍵與模型的辨識結果的對應關係，讓使用者在任何工作軟體都可以使用手勢提升效率，增加工作的生產力。目前在時間上的瓶頸主要來自於資料蒐集的方式，若能改善則速度和準確率都有希望更上一層樓。

11 Appendix

分工表

Table 3: 工作分配表

統籌、專題構想和時程規劃	陳祈安
Arduino 手勢資料收集軟硬體	陳祈安
器材採購	陳祈安
手勢信號收集	柯岱佑、陳祈安、翁證騏
信號前處理	柯岱佑
手勢辨識模型	柯岱佑
PyAutoGUI 整合	翁證騏
即時辨識模組	翁證騏
手勢設定前端網站	陳祈安
DEMO 影片	陳祈安
報告撰寫	柯岱佑、陳祈安、翁證騏
報告整合	陳祈安
報告投影片	陳祈安
問題與討論	柯岱佑、翁證騏

程式碼、Demo 影片、前端網頁

Github: https://github.com/chian-chen/BELab
Demo 影片: https://youtu.be/3UDJmE8ajxY
Frontend Web: https://chian-chen.github.io/BELab-Final/

References

[1] Andrey Ignatov. Real-time human activity recognition from accelerometer data using convolutional neural networks. *Applied Soft Computing*, 62:915–922, 2018.

LSM9DS1:

- https://cdn.sparkfun.com/assets/learn_tutorials/3/7/3/LSM9DS1_Datasheet.pdf
- https://github.com/FemmeVerbeek/Arduino_LSM9DS1
- https://learn.sparkfun.com/tutorials/lsm9ds1-breakout-hookup-guide/all
- https://github.com/sparkfun/SparkFun_LSM9DS1_Arduino_Library

PySerial/ PyAutoGUI documentation:

- https://pypi.org/project/pyserial/
- https://www.mathworks.com/help/supportpkg/arduinoio/ug/find-arduino-port-on-windows-mac-and-linux.html
- https://pyautogui.readthedocs.io/en/latest/

Others:

- http://cc.ee.ntu.edu.tw/~ultrasound/belab/term_project/Group7/final_reprt_G7.pdf
- https://learn.sparkfun.com/
- https://www.biometricupdate.com/201801/this-ring-uses-gesture-recognition-to-write-words-and-numbers
- https://swf.com.tw/?p=1188&fbclid=IwAR3kguoYJDWyvA7fybZRm8fOZz0lmJnv13t9mzQMa4MRST8QOaOa_P16QXk
- https://link.springer.com/chapter/10.1007/978-3-319-27707-3_19
- ChatGPT