中華民國第 57 屆中小學科學展覽會

作品說明書

高級中等學校組　電腦與資訊學科

第一名

052508

神來之手-自動繪圖機設計與探討

學校名稱：嘉義市私立嘉華高級中學

作者：	指導老師：
高三 朱雁丞 高三 柯昱任 高三 謝佳峻	李威璋

關鍵詞：自動化控制、路徑規劃、影像處理

得獎感言

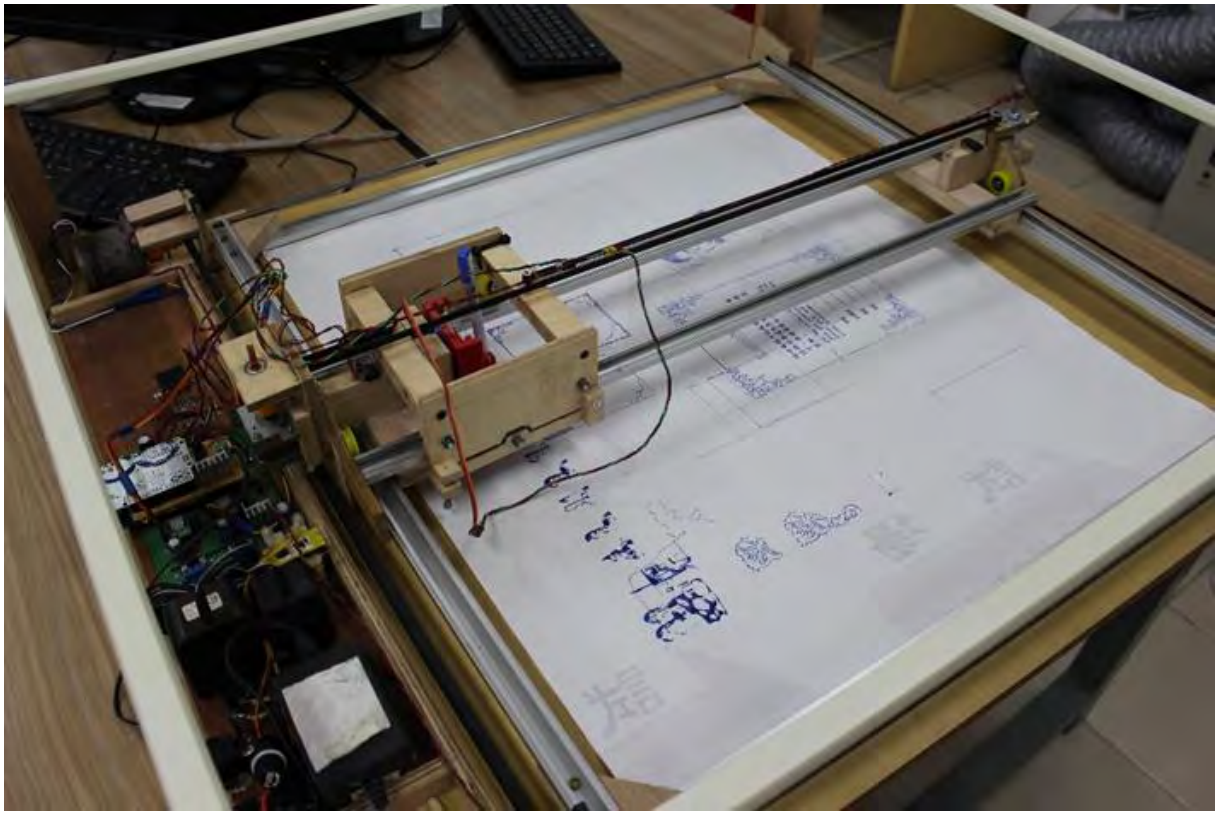
破繭成蝶，迎向未來科學研究之路

自國中以來到高三畢業參展六年了，每年參展都有不同感觸。今年再度憑著一股熱誠與挑戰的精神，用僅剩的高中時光放手一搏，不在高中時光留下任何遺憾，終於榮獲「電腦與資訊學科組」第一名的佳績。

在這六年的時光，經過時間的磨練，自己的經驗與知識不斷地增加，每一次的專題製作都讓我涉獵了更廣泛的領域，給我進入科學界的初步認識，同時也體認到科學的魅力與知識的遼闊無邊。從撰寫自己的第一支程式開始，到現在可以開發應用程式，這都是在製作專題的過程中查閱無數資料所獲得的能力，這些過程讓我成長，最終才使得自己今天足以站立在這個舞台。讓我收穫更多的是自己動手做的「Maker」精神，專題完成後所獲得的成就感與滿足感只有在經歷過漫長的 Debug 過程才能體會，在這幾年中也因此遇見了許多志同道合的夥伴，一起討論研究、互相共鳴。

一開始的動機只是新奇有趣，試著想想讓機器運筆是一件多麼酷的事情！為了達成目標我們實作了「自動繪圖機」，並在撰寫圖片轉換路徑演算法的過程中發現，現代工業中有許多機器使用到的技術概念與此相似，於是展開一連串的實驗分析，希望能藉由我們的研究解決問題。

日常生活中有很多事物等著我們去探索，但卻僅有極少人有那個熱情與毅力會去發掘它們，當我們發現新的事物或是新的問題時要大膽地去思考，並主觀詮釋、客觀求證，藉由探索世界萬物運行真理的過程中發現自然的偉大及奧妙，發現前人智慧與現代科技的進步。



自動繪圖機成品

團體合照

摘要

一個好的路徑規劃演算法應該為機器爭取最少的運作時間、盡量減少機器傷害，並且同時達到最大目標效果。為了找出最佳演算法，本研究嘗試各種不同的演算法，並試圖在不同的幾何圖像中表現其最佳路徑特性，並融合各種不同演算法的特性，希望藉此找到更具有效率之演算法。於實驗中本研究藉由模擬軟體進行初步分析，再以繪圖機做實際比對分析，找出各種可能影響輸出成品之因素，並且找出不同演算法對於不同幾何圖案的相適性關係。

本研究之特色在於所使用之軟體、韌體、硬體皆由本研究開發而成，能以最直接、最簡單、未經過優化的原始狀態進行分析，以深入探討演算法對於繪出圖形之影響因素。

壹、 研究動機

隨著現代科技蓬勃發展，越來越多的工作交由電腦或機器人完成，舉凡文書處理、圖像列印、工業模型製造、CNC、3D列印機等等。機器的工作效率取決於前置圖形運算工作與動力機械的運作是否能發揮最高效率。舉例來說，機器在填滿一個面的過程中，若演算法充滿轉折並且必須不斷起筆來改變位置形成「無效移動」，將會增加失步與錯位的機率並且浪費許多無謂的時間。

基於上述原因，本研究期望能探究不同圖形與不同演算法之相適性關係，期望能解決現代許多機械面臨之難題。

本研究使用Arduino Mega作為自動繪圖機的運算處理核心，運用A4988步進馬達驅動晶片控制X軸與Y軸之馬達，並透過Arduino IDE撰寫此專題之Firmware，該Firmware將會讀取SD卡之路徑資料檔，並依據該檔案之路徑控制機器座標。

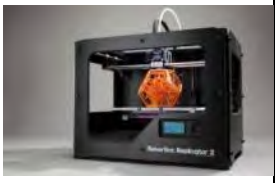
本研究使用Microsoft Visual Studio Express撰寫Windows Form程式，透過C#撰寫「影像分析工具」(pictureModify)與「VcodeViewer」，前者負責多種演算法處理之運算處理，後者用於預覽輸出之路徑檔。

貳、研究目的

綜合上述討論，本研究目的歸納如下：

1. 製作出能夠透過步進馬達精確控制位置的座標系統裝置。
2. 撰寫能夠依據指令移動座標軸馬達的Arduino Firmware。
3. 透過C#撰寫多種演算法進行圖像資料處理並建構介面供使用者操作(影像分析工具)。
4. 透過C#撰寫路徑預覽工具方便檢視輸出路徑並檢測該演算法之運算效果
5. 找出在不同的演算法對於不同幾何圖形之最佳相適性。
6. 分析不同演算法處理圖像時的特色。
7. 推測不同演算法發揮最大效益之時機。

參、研究設備及器材

			
Arduino	步進馬達	A4988	蜂鳴器
			
電阻	洞洞板	電烙鐵	焊錫
Visual Studio			
Microsoft Studio C#	Arduino IDE	3D列印機	筆記型電腦

肆、 研究過程與方法

一、 研究架構

本研究架構由以「影像分析工具」進行不同演算法之分析、以「Vcode Viewer」進行輸出路徑之模擬並探討，最後用「自製繪圖機」實際描繪圖形。以下將分則說明詳細內容。

(一)、 以「影像分析工具」進行不同演算法之分析：

藉由此工具以「循序掃描法」、「重複邊緣法」、「深度優先搜尋法」、「廣度優先搜尋法」、「邊緣DFS搜尋法」，分別對不同圖片進行路徑運算，並輸出於##_AreaPath.txt，以下介紹實驗樣本。

將不同的圖片以相同方式壓縮至200*200像素大小，本實驗使用「Circle」、「CircleHollw」、「Square」、「Bubble」、「Alphabet」、「multipleCircle」、「Odd」、「Unrule」、「Lena」、「Einstein」、「Bill」等11種不同圖片 。

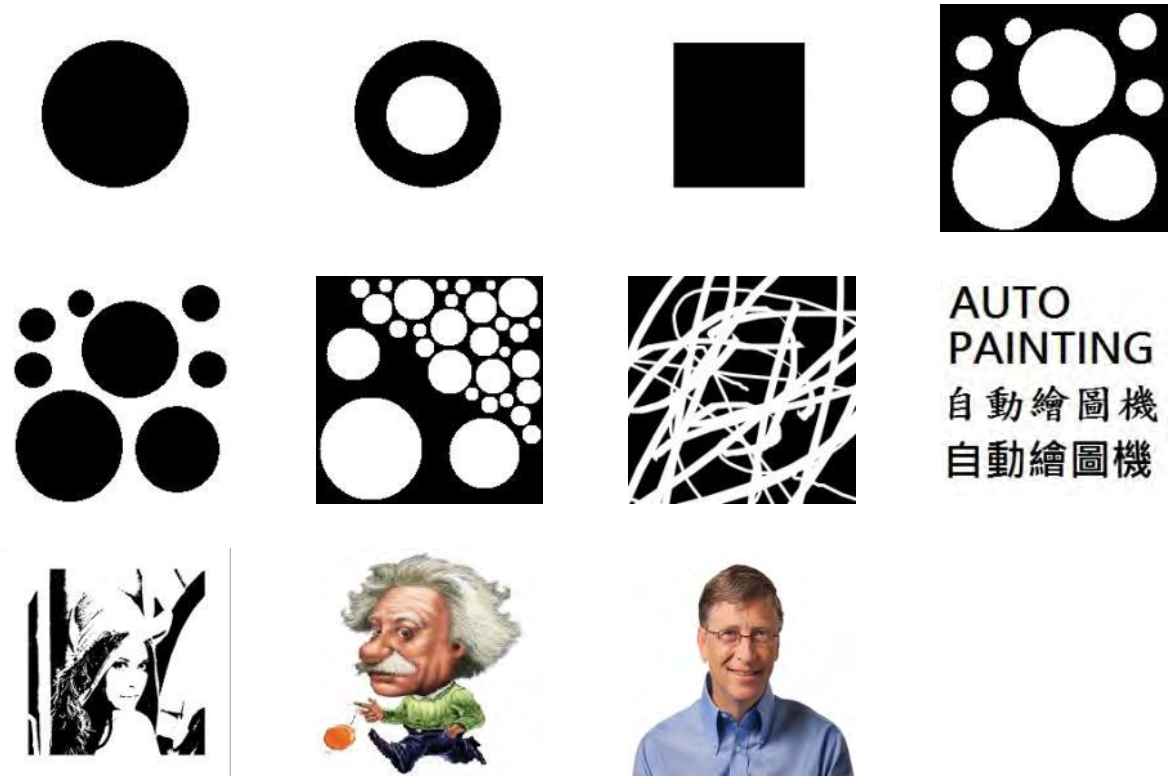
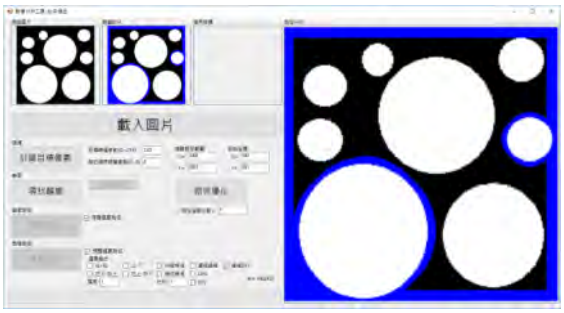
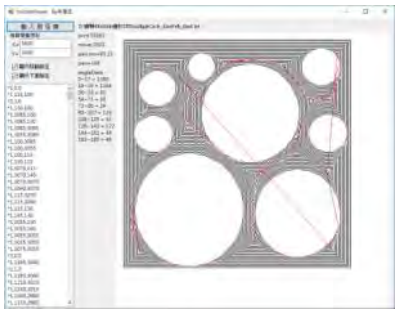


圖5-1、實驗樣本

(二)、　用「VcodeViewer」取得屬性資料並逐一比較，分析不同演算法對於





不同圖形之運算之效果。

圖4-1、影像分析工具　　圖4-2、Vcode Viewer

(三)、　藉由「VcodeViewer」檢視所輸出的路徑資料，並藉此歸納出不同演算法對於不同圖形之特殊表現，以下說明屬性資料。

1. 移動效率：為下筆路徑長與全部路徑長的比值，值越大代表移動效率越高，代表機器將能以最少之運作動作完成任務。
2. 填充率：由下筆路徑長除以對該圖片之所有運算之路徑長最大值以計算填充率，填充率越高代表該演算法對於圖形有越高的填充效果。
3. 旋轉角度統計：紀錄路徑行進角度變化值，並以π/10為一單位進行統計，旋轉角度越多，代表機器需要改變移動方向次數越多。
4. 畫筆狀態改變次數：紀錄畫筆狀態改變之次數。

(四)、　用「自製繪圖機」實際描繪圖形：

將輸出的路徑資料輸入機器使其畫出完整圖形，以展現輸出成果並比對理論輸出與實際輸出之相異。

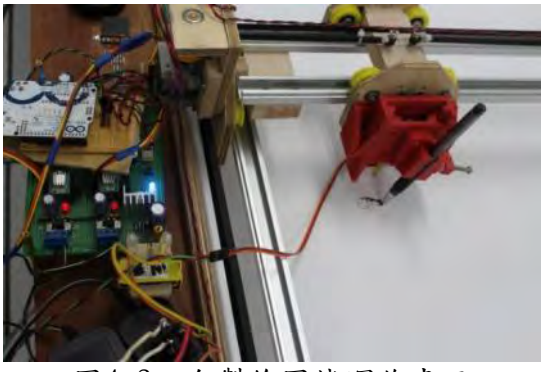


圖4-3、自製繪圖機運作畫面

二、 自動繪圖機製作

(一)、 直角坐標滑輪系統

本研究利用鋁窗滑輪與鋁合金製作軌道，並以木工完成零件之固定。此自動繪圖機以H型結構(圖4-4)進行X、Y軸之操控，透過控制步進馬達之步數達到座標定位的功能。

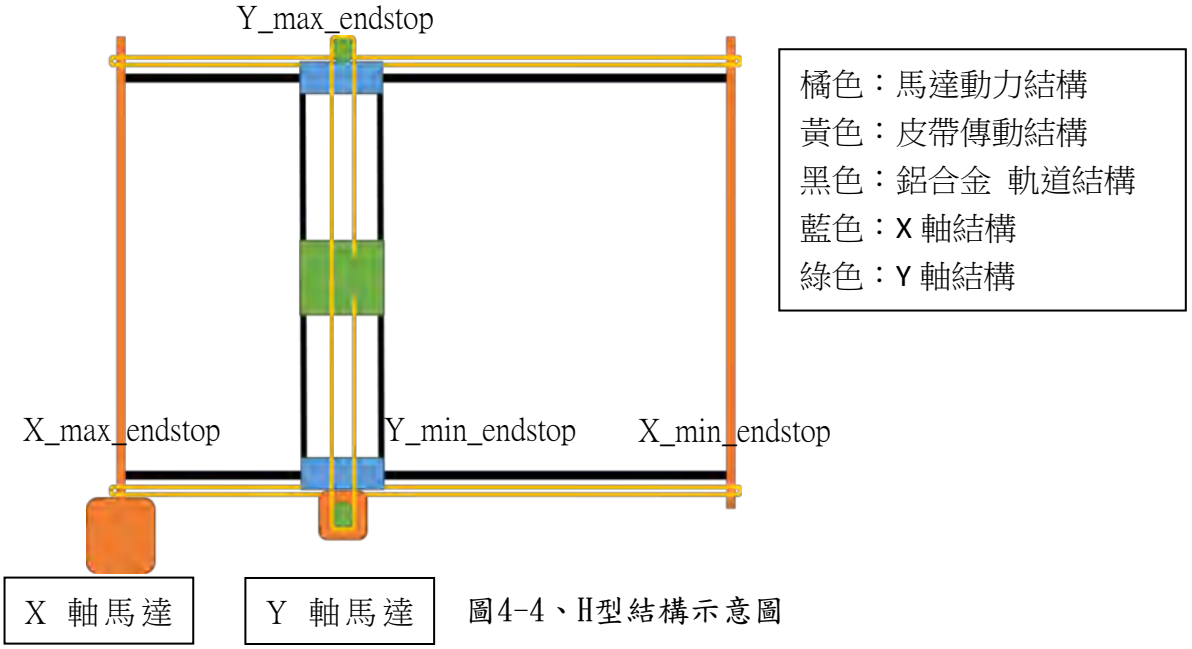


圖4-4、H型結構示意圖

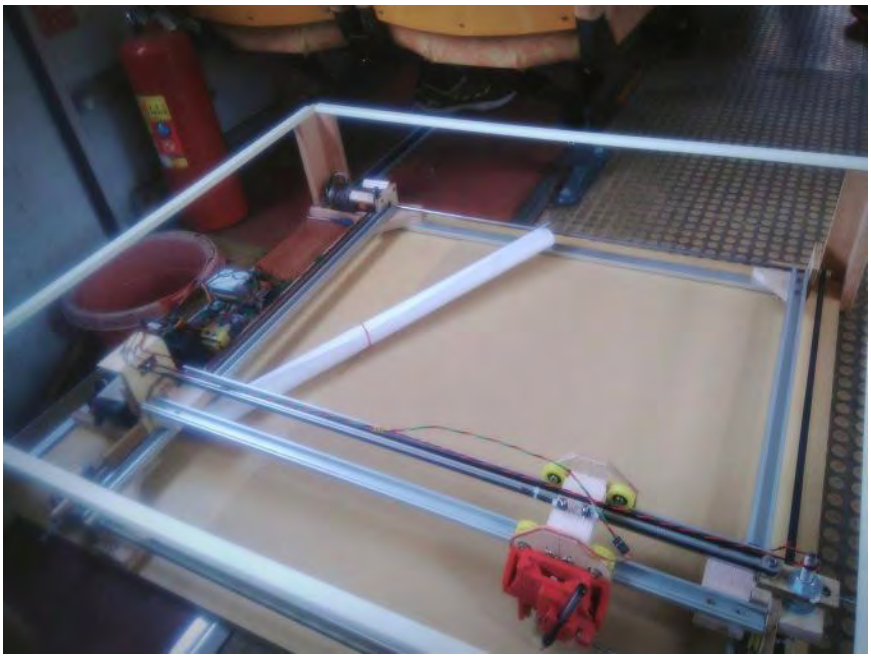


圖4-5、自動繪圖機成品圖

(二)、 A4988步進馬達控制晶片

步進馬達的控制需要調控隨時間變化的輸出1A、2A、1B、2B四個腳位(圖4-6)，此晶片可協助開發者省去類比信號輸出與功率放大之實作、提供最高32微步之類比驅動，使步進馬達藉由此晶片進行高精度之控制，並簡化為只需輸入Step、Direction、Enable三個數位資料輸入便可操控。以下分別說明三者之功能：

1. Direction：控制馬達旋轉方向，例如：高電壓為正轉，低電壓為逆轉。
2. Step：當此腳位輸入正緣信號時，步進馬達往該Direction移動一步。
3. Enable：當此腳位為低電壓時馬達由晶片供電控制，否則為自由轉動。

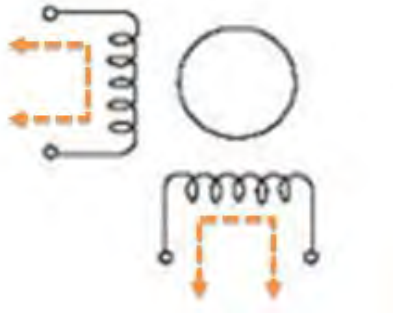


圖4-6、步進馬達內部構造示意圖

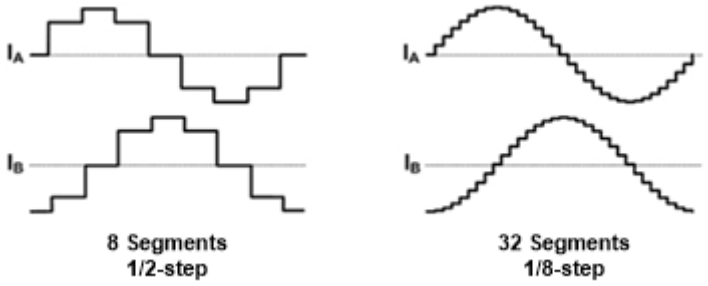


圖4-7、8微步與32微步驅動示意圖

(三)、 Arduino Mega 2560與控制電路

利用微電腦Arduino Mega 2560(圖4-8)進行SD卡資料讀取與控制步進馬達。電路架構如圖4-9。

圖 4-8、Arduino Mega 2560

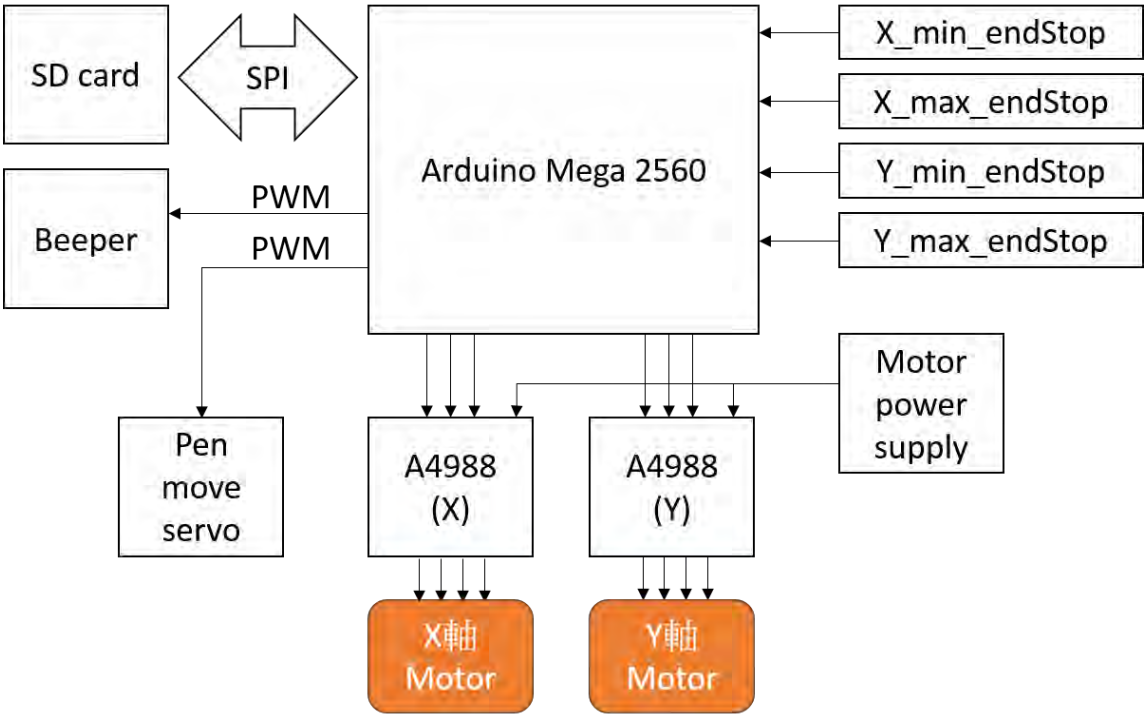


圖 4-9、控制電路核心架構

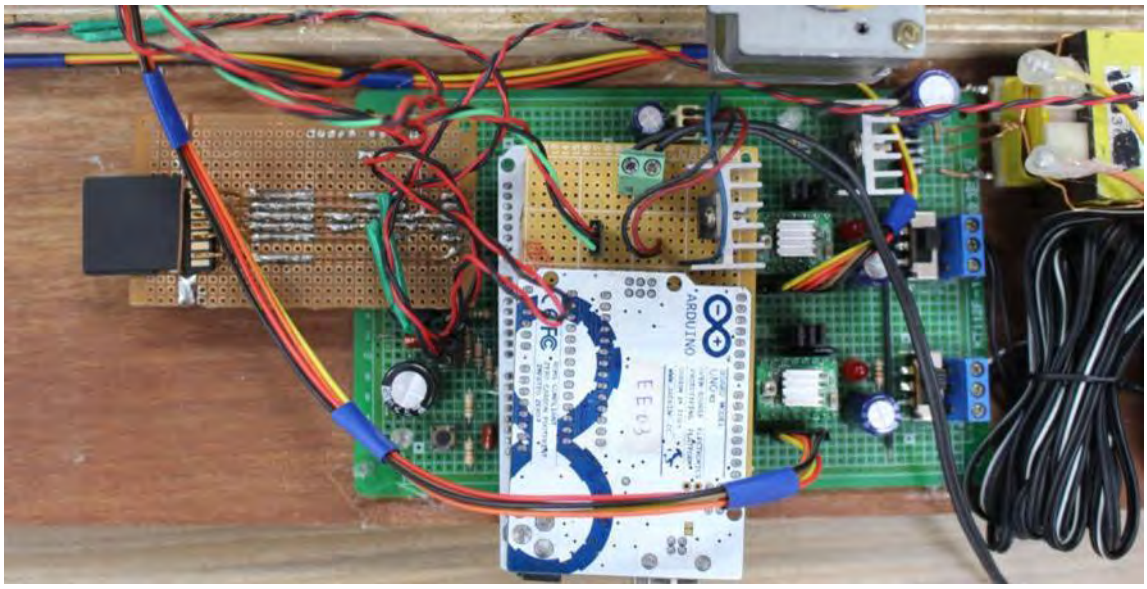


圖4-10、控制電路成品圖

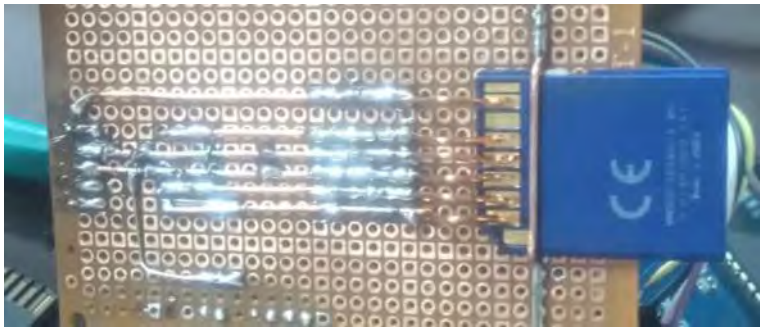


圖 4-11、SD 卡模組成品圖

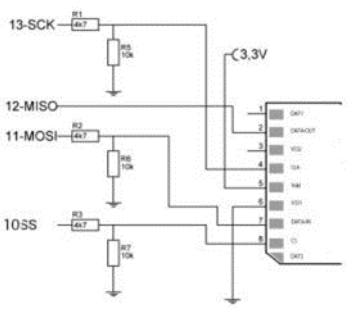


圖 4-12、SD 卡電路圖

(四)、　下筆起筆結構設計

在高速移動下的環境，畫筆的固定裝置必須達到穩固、快速、安全之基本條件，於是本研究透過AutoDesk 123D繪製筆架(圖4-13)，並用以3D列印技術印製零件。

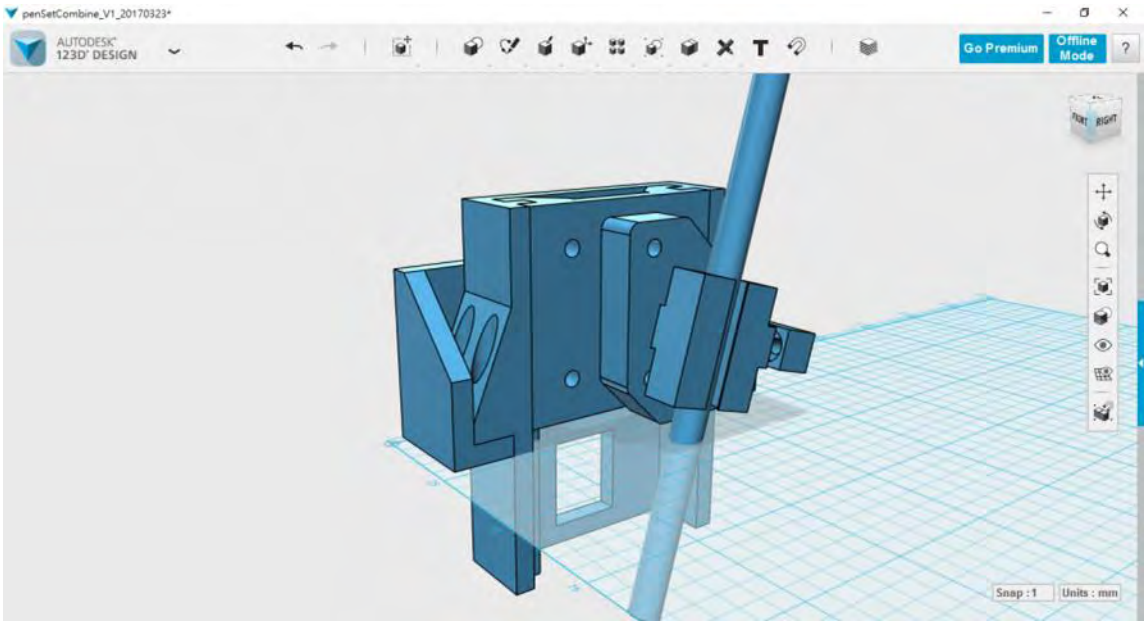


圖4-13、筆架3D結構圖

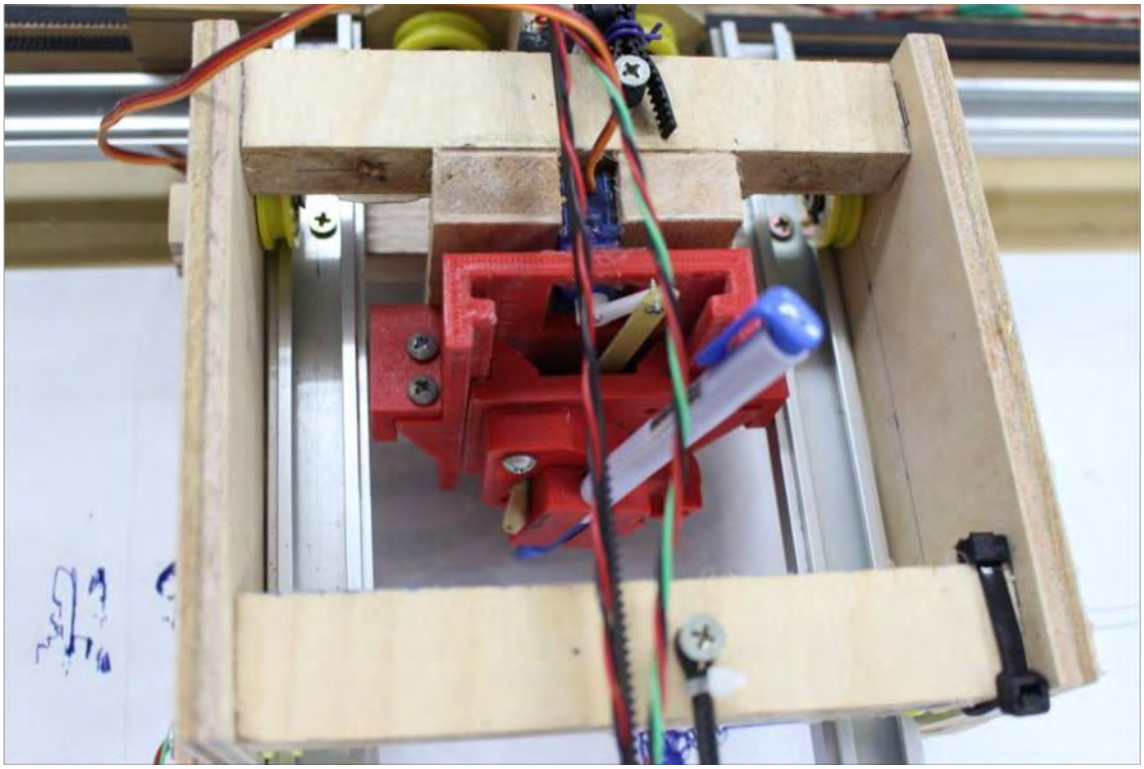


圖4-14、筆架3D列印成品

(五)、 Vcode 指令

為使電腦輸出之路徑檔案可供Arduino讀取，必須在兩者之間建立可互相流通的共通語言。此「Vcode」以一行為一指令單位，最多可包含兩個引數，並分別以逗號區隔，其中包含此自動繪圖機專屬指令集，以下介紹重要指令。

1. M,*x*,*y* ：M為move縮寫，代表移動到點(*x*,*y*)。
2. V,*speed*：V代表將機器速度改變為*speed*。
3. H：H為home縮寫，代表回到原點(0,0)。
4. A,*status*：A代表控制起筆與下筆，若*status*為1為下筆，若為0為起筆。
5. D,*time*：D為delay的縮寫，代表使機器暫停*time*毫秒。

(六)、 Arduino Firmware

此程式分為兩大主要目的：

1. 讀取資料：讀取SD卡data.txt內的Vcode資料並解譯指令。
2. 執行指令：從記憶體讀取指令並依據指令執行。

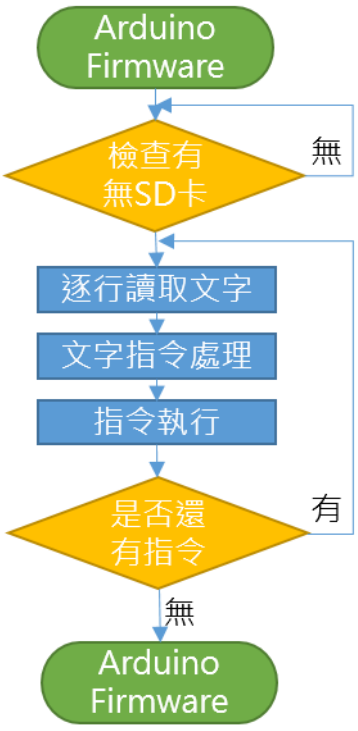


圖4-15、程式流程圖

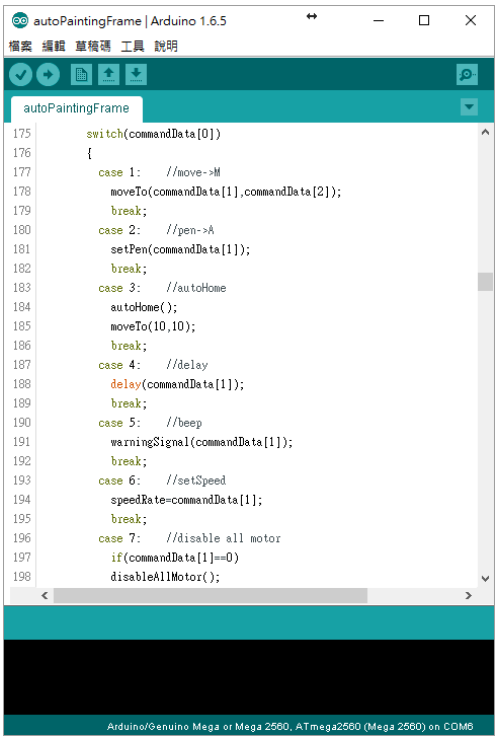


圖4-16、指令執行程式碼

三、 撰寫「影像分析工具」程式

(一)、 架構介紹

1. 計算目標像素(goalColor)：利用灰度轉換公式Gray = (R*30 + G*59 + B*11 + 50) / 100取得原始圖片灰階值，再依據所輸入的灰階標準進行篩選。
2. 尋找輪廓：計算是否為輪廓方法如下，計數單位像素四周goalColor個數，若其值為1~7代表其為輪廓的一部份。
3. 規劃輪廓路徑：沿著第二點所搜尋到的輪廓延伸便可得到輪廓路徑，此演算法於「演算法介紹」將詳細說明。
4. 規劃面積路徑：點此以選擇用以下不同演算法進行路徑規劃。

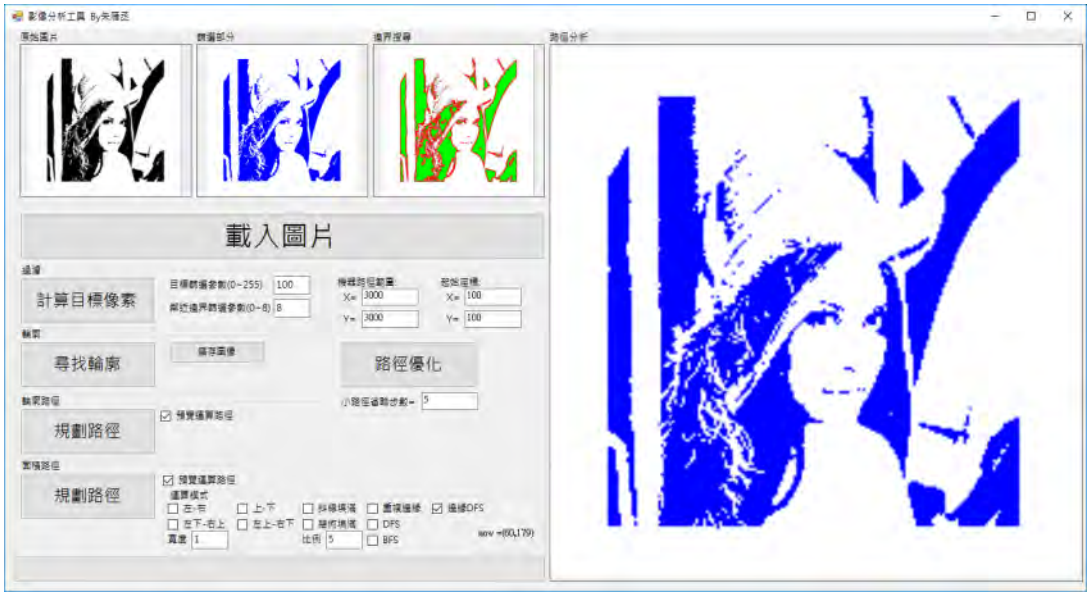


圖4-17、影像分析工具

(二)、 演算法介紹

1. 循序掃描法：此方法以掃描法遍歷圖片陣列，當遇到goalColor時便寫入下筆指令，反之下達起筆指令。本演算法依掃描方向至不同再分為四種：

   (1) 左右型

   (2) 上下型

(3) 主對角線

(4) 副對角線

2. 重複邊緣：

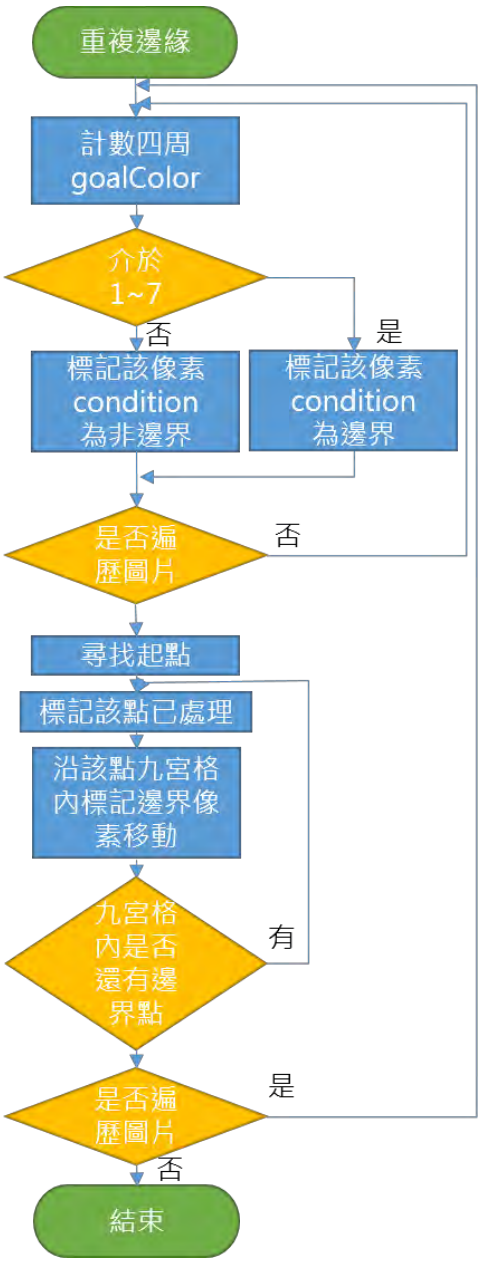
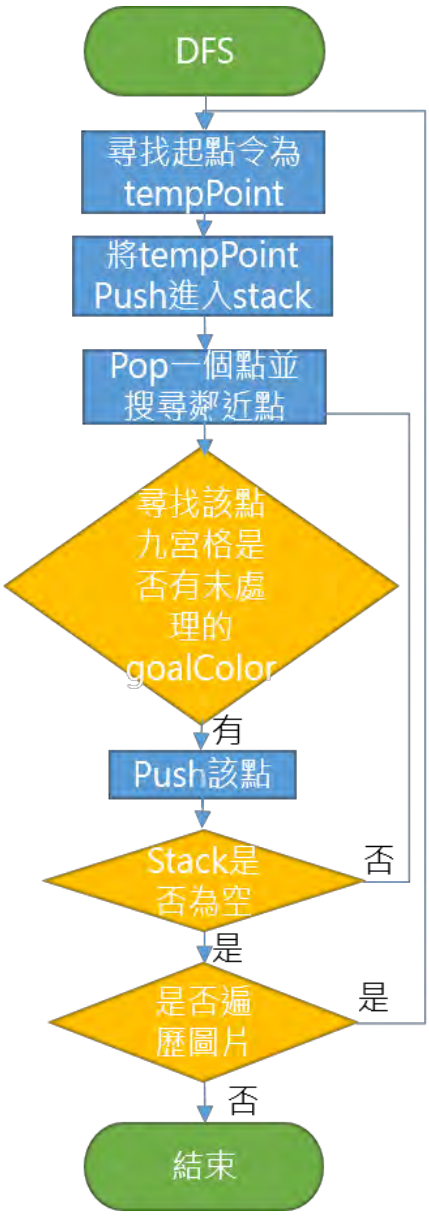


圖 4-18、重複邊緣流程

藉由重複進行邊緣搜尋逐步深入圖形內部，以達到填滿效果。流程圖見圖4-17。該演算法遍歷整張圖片，並嘗試以每個點作為起點進行邊緣搜尋，當getNextPoint()回傳值為(-1,-1)時表示已完全遍歷也代表演算法完成。邊緣搜尋分為兩大部分，其一為找出圖片內的邊緣點，該方法藉由遍歷整張圖片，搜尋單位點四周goalColor個數為1~7者並標記之，這些點即為邊緣。其二為邊緣路徑走訪，此演算法將會記錄自身前進像素方向與目標方向，並利用特定方向優先的DFS完成該部分的演算法。

3. DFS：



深度優先搜尋法，此演算法將會以深度為優先往下探索(圖4-19)，本研究採用堆疊(Stack)來實作。

圖 4-19、DFS 流程圖

4. BFS：

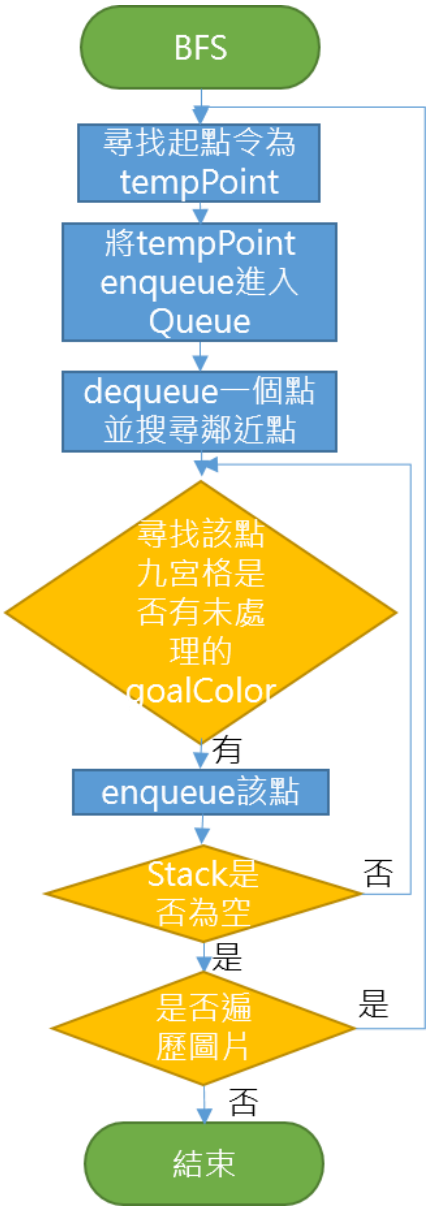


圖 4-20、BFS 流程圖

廣度優先搜尋法，此演算法將會以廣度為優先側向探索(圖4-20)，本研究採用佇列(Queue)來實作。

四、　撰寫VocdeViewer路徑模擬程式

為模擬自動繪圖機畫筆運作，本研究以C#撰寫路徑模擬程式，並以該程式讀取「影像分析工具」產生之路徑檔，並用與Arduino Firmware相同的資料讀取演算法、執行演算法模擬機器路徑運行，並記錄「下筆路徑長」、「起筆路徑長」、「移動效率」、「填充率」、「旋轉角度統計」、「下筆次數」等資料，以判斷該演算法對於不同圖形之相適性。

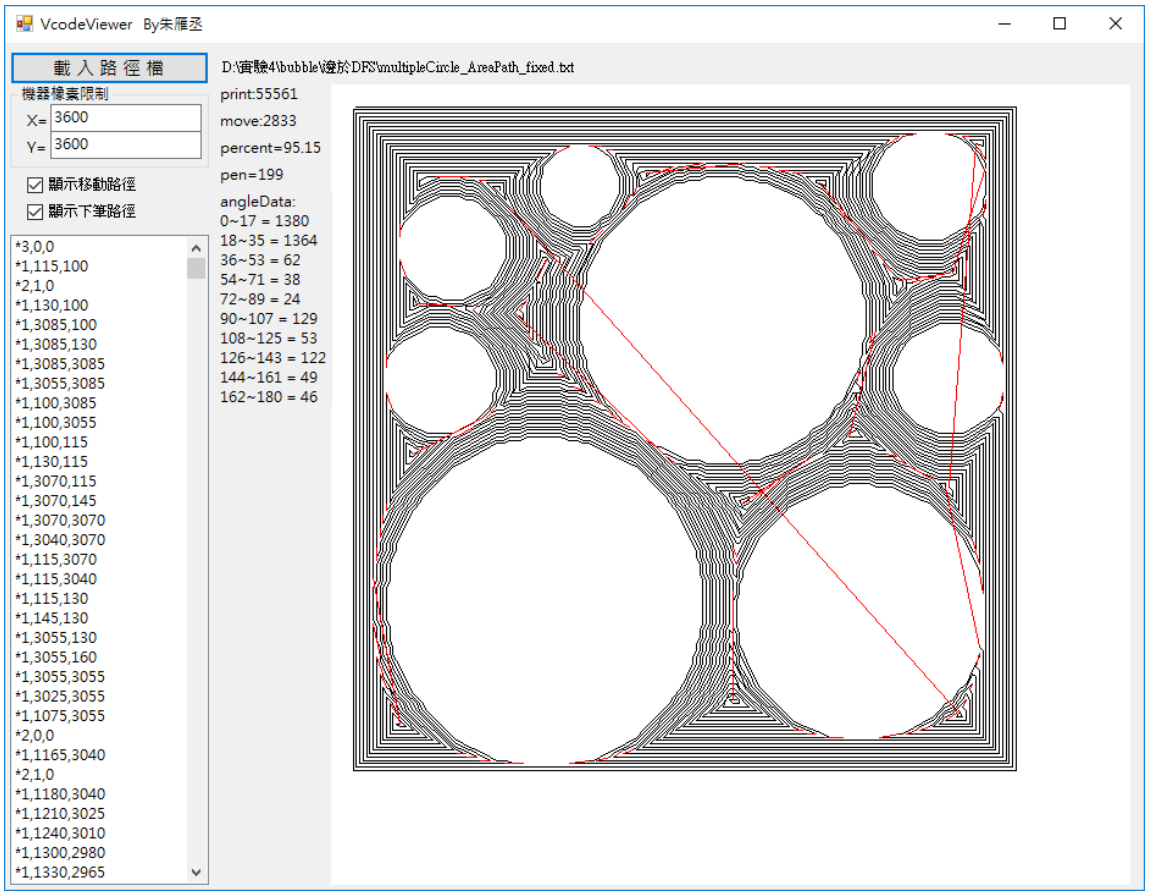


圖4-21、VcodeViewer

伍、 研究結果與討論

實驗一、各種演算法對於不同圖片之分析比較

本實驗使用「循序掃描法」、「重複邊緣法」、「DFS」、「BFS」對前述11種圖片進行路徑規劃，再藉由路徑模擬工具取得得演算法特性，並觀察其路徑分析演算法特色，最後觀察，實驗數據參照表1、2、3、4。

實驗小結：

(一) 循序掃描法

1. 連續實心圖形有最佳表現，但遇到不連續圖形或空心圖形效率會大幅下降。
2. 上下型及左右型填充率為所有演算法中最低，但斜向掃描卻是所有演算法中最高。
3. 上下及左右型演算法在角度統計的資料中，有最佳表現。
4. 斜向掃描演算法產生最多畫筆狀態改變次數，推測其與填充密度相關。

(二) 重複邊緣演算法

1. 重複邊緣演算法提升了簡單不連續圖片 、簡單空心圖片的移動效率。
2. 重複邊緣演算法在下筆次數的表現次佳，僅次於DFS。
3. 重複邊緣演算法在角度表現較不出色，產生大量90度~107度的移動。

(三) DFS

1. DFS相較於掃描法和重複邊緣有更高的移動效率
2. DFS之填充率僅次於斜向掃描法
3. DFS在角度表現最差，造成多次方向變換而損傷機器或造成失步的現象。

(四) BFS

1. BFS演算法於移動效率、填充率皆為最差

實驗二、邊緣DFS演算法對於不同圖片之分析比較

實驗一發現「重複邊緣演算法」與「DFS」演算法皆僅次於掃描法有優秀之表現，故結合兩者之特色設計「邊緣DFS演算法」以期能融合兩者之優點。

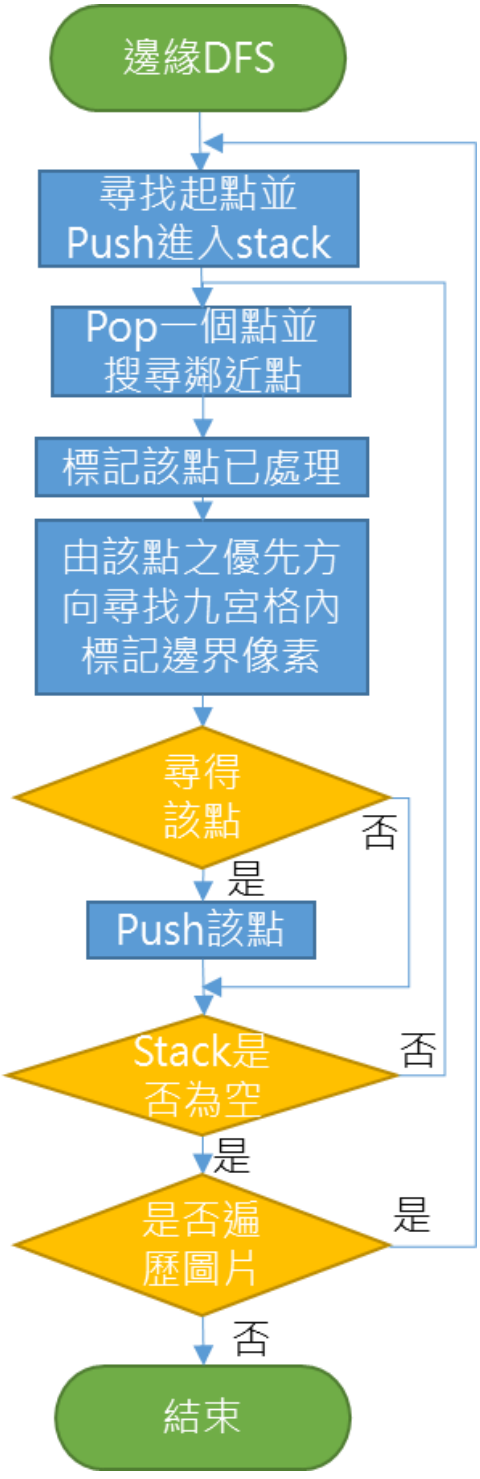


圖 4-20、邊緣 DFS 流程圖

邊緣DFS：在實驗過上述演算法後，發現若能綜合重複邊緣演算法與DFS之特色，便能產生另一高效率之規畫路徑演算法，因此設計此融合重複邊緣與DFS之演算法(圖4-20)。該演算法之重點為，在取得邊緣後沿著邊緣做DFS運算，便能使路徑有方向性的移動。

表1、移動效率總資料

移動效率	上下	左右	主對角	副對角	重複邊緣	DFS	BFS	邊緣 DFS
Circle	98.26%	98.27%	98.32%	98.29%	97.10%	99.47%	54.39%	100.00%
circleHollow	69.04%	69.03%	69.41%	69.39%	95.83%	97.10%	36.75%	98.16%
square	98.91%	98.75%	98.16%	98.15%	99.53%	99.30%	57.15%	100.00%
Bubble	47.69%	47.66%	47.83%	47.95%	92.40%	96.03%	22.60%	95.16%
multiple	76.16%	76.17%	80.33%	78.28%	94.83%	97.69%	28.12%	98.36%
odd	47.79%	47.76%	47.95%	48.07%	89.52%	92.25%	22.01%	94.03%
unrule	53.84%	53.69%	53.47%	57.68%	85.01%	88.67%	28.01%	91.11%
Capter	31.66%	37.70%	28.76%	29.79%	72.11%	76.30%	14.62%	80.25%
girl	51.68%	41.50%	49.75%	50.12%	75.06%	83.67%	23.42%	87.98%
Einstein	32.80%	38.52%	37.26%	39.03%	62.32%	72.81%	19.83%	78.49%
Bill	48.77%	43.00%	49.48%	51.93%	74.44%	78.43%	24.81%	87.36%

表2、填充率總資料

填充率	上下	左右	主對角	副對角	重複邊緣	DFS	BFS	邊緣 DFS
Circle	69.51%	69.51%	100.00%	100.00%	69.01%	80.30%	68.13%	78.17%
circleHollow	69.53%	69.51%	100.00%	99.99%	68.63%	79.57%	66.27%	79.67%
square	69.48%	69.53%	100.00%	100.00%	69.17%	76.61%	68.29%	69.49%
Bubble	69.95%	69.90%	99.50%	100.00%	68.49%	77.06%	63.78%	76.82%
multipleCircle	69.49%	69.53%	100.00%	100.00%	68.48%	80.05%	46.66%	78.16%
odd	69.95%	69.91%	99.53%	100.00%	67.63%	75.89%	60.85%	78.57%
unrule	69.81%	69.74%	99.78%	100.00%	66.37%	77.63%	45.38%	78.50%
Capter	69.56%	69.47%	99.87%	100.00%	62.66%	74.22%	35.20%	70.76%
girl	69.49%	69.46%	99.97%	100.00%	62.66%	78.74%	56.36%	77.11%
Einstein	69.38%	69.60%	99.94%	100.00%	58.45%	73.69%	45.24%	76.49%
Bill	69.59%	69.46%	99.74%	100.00%	62.66%	77.18%	52.90%	78.12%

表3、旋轉角度統計總資料

旋轉角度統計	上下	左右	主對角	副對角	重複邊緣	DFS	BFS	邊緣 DFS
Circle	2323.42	2323.75	2665.6	2666.37	16582.2	16257.4	2678.56	5856.77
circleHollow	1757.9	1757.78	2037.2	2037.65	10799.9	10232.7	2243.11	4524.24
square	2505.64	2541.35	3124.84	3062.71	2525.96	17303.4	2909.11	2525.81
Bubble	3515.54	3515.69	4241.46	4250.54	11895.7	18575.7	5278.9	6404.12
multiple	3695.62	3708.98	4387.56	4343.06	26815.7	26317.7	20262.7	9599.29
odd	3569.24	3566.87	4280.75	4290.43	770934	17563.1	5760.18	7066.72
unrule	3870.92	3857.7	4637.66	4627.79	20016.4	23274.1	20985.8	10864.3
Capter	1786	1629.38	2250.32	2318.74	5011.06	8245.95	8905.66	3755.84

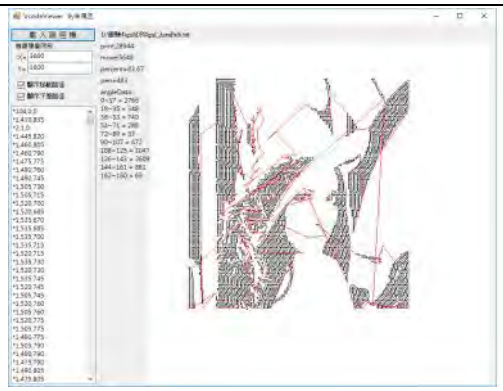
girl	1927.93	2040.46	2597.53	2531.87	470010	13122.2	5619.12	5376.53
Einstein	1228.62	1352.73	1496.96	1693.64	4770.04	5037.02	4500.06	3455.07
Bill	1407.19	1542.24	1720.89	1877.91	5659.51	7527.65	4378.19	3835.67

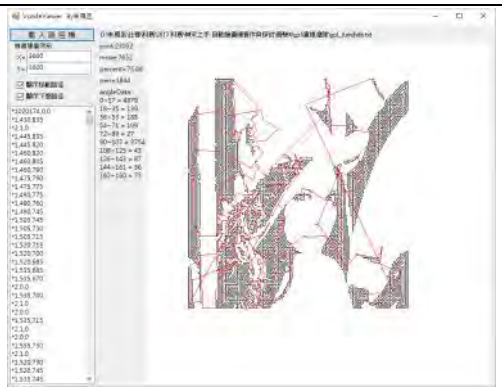
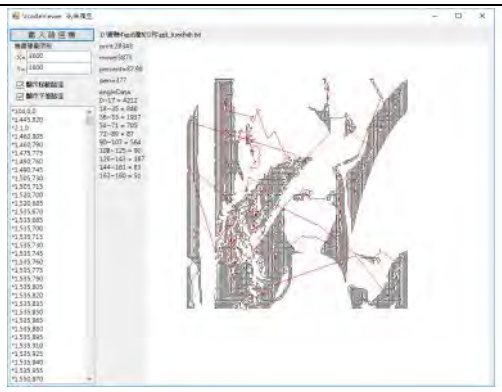
表4、畫筆狀態改變次數總資料

下筆次數	上下	左右	主對角	副對角	重複邊緣	DFS	BFS	邊緣 DFS
Circle	258	258	366	366	194	46	254	2
circleHollow	402	396	566	566	234	88	426	44
square	230	254	482	482	140	78	248	2
Bubble	816	818	1152	1152	802	308	1698	206
multiple	814	816	1150	1150	620	156	6838	16
odd	1464	1432	2098	2098	1278	434	2524	274
unrule	2826	2646	4474	3234	2022	792	6766	486
Capter	2750	2950	4710	4512	1424	624	3484	432
girl	1608	2608	2938	2636	1846	484	1720	378
Einstein	2010	1634	2274	2284	1554	566	1424	402
Bill	1330	1560	1966	1730	1214	438	1370	262

邊緣DFS演算法相較於「重複邊緣」與「DFS」而言有更少量的角度變化，可能減少該演算法對機器造成失步的影響，以「Lena」(表5)作為舉例說明。

表5、Lena之「DFS」、「重複邊緣」、「邊緣DFS」模擬路徑資料

	路徑模擬工具模擬圖	屬性資料	角度統計資料
D F S		print= 28944 move= 5648 pen= 484 percent=83.67	angleData: 0~17 = 2766 18~35 = 348 36~53 = 740 54~71 = 288 72~89 = 33 90~107 = 472 108~125 = 1047 126~143 = 2609 144~161 = 881 162~180 = 69

重複邊緣		print= 23032 move= 7652 pen= 1846 percent=75.06	angleData: 0~17 = 4879 18~35 = 139 36~53 = 188 54~71 = 109 72~89 = 27 90~107 = 3754 108~125 = 43 126~143 = 87 144~161 = 56 162~180 = 73
邊緣DFS		print= 28343 move= 3873 pen= 378 percent=87.98	angleData: 0~17 = 4212 18~35 = 846 36~53 = 1937 54~71 = 705 72~89 = 87 90~107 = 564 108~125 = 90 126~143 = 187 144~161 = 83 162~180 = 51

實驗小結：

1. 邊緣DFS移動效率相較於「重複邊緣」、「DFS」對於所有圖形皆有進步，為所有演算法中最高者。
2. 邊緣DFS填充率相較於「DFS」略低，但高於「重複邊緣」。
3. 邊緣DFS旋轉角度統計相較於「重複邊緣」與「DFS」皆大幅下降。
4. 邊緣DFS畫筆狀態改變次數大為進步，為所有演算法中最低者。

實驗三、路徑優化對不同演算法產生之路徑影響

實驗二發現各演算法所產生之路徑並非其最佳化之表現，容易產生多鋸齒、連續直線點、冗餘點等等問題，於是在「影像分析工具」內加入「路徑優化」功能，並分析此功能對各演算法產生之效果。

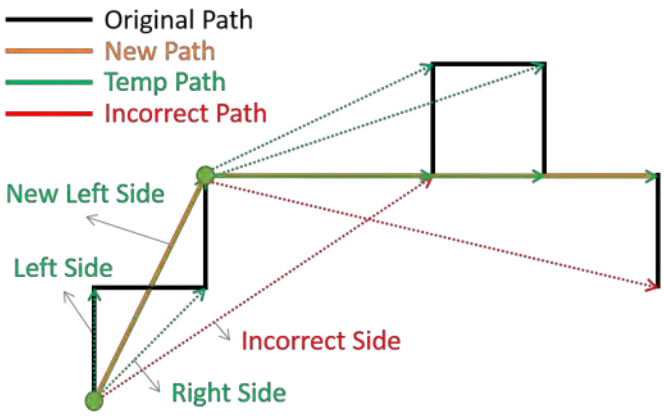


圖21、鋸齒消除演算法之概念

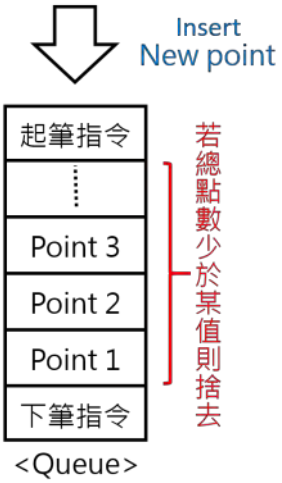


圖22、冗餘點消除演算法之概念

表6、路徑優化後之移動效率變化量

移動效率	上下	左右	主對角	副對角	重複邊緣	DFS	BFS	邊緣 DFS
Circle	0.00%	0.00%	-0.03%	-0.03%	2.15%	-0.20%	0.00%	0.00%
circleHollow	0.00%	0.00%	-0.50%	-0.50%	1.03%	-0.51%	-0.05%	-0.05%
square	0.00%	0.00%	-0.03%	-0.02%	0.09%	0.02%	0.02%	0.00%
Bubble	-0.29%	-0.27%	-1.30%	-1.56%	1.70%	-0.80%	1.06%	-0.01%
multiple	0.00%	0.00%	0.10%	-0.34%	1.75%	-0.74%	50.25%	-0.07%
odd	-0.38%	-0.27%	-1.46%	-1.46%	0.98%	-0.46%	1.36%	0.09%
unrule	-0.13%	0.15%	-1.70%	-0.71%	1.72%	-2.05%	32.49%	-0.14%
Capter	-1.07%	-0.42%	-4.03%	-2.48%	7.20%	-4.02%	13.34%	0.50%
Lena	7.47%	-1.40%	-1.69%	0.42%	6.43%	-3.13%	9.01%	-0.16%
Einstein	3.87%	12.77%	3.52%	17.76%	10.56%	-0.38%	22.74%	0.97%
Bill	0.66%	17.26%	-0.65%	16.43%	7.74%	-0.61%	9.92%	0.74%

表7、路徑優化後之填充率變化量

下筆路徑長	上下	左右	主對角	副對角	重複邊緣	DFS	BFS	邊緣 DFS
Circle	1.70%	1.70%	-0.02%	0.00%	-20.04%	-28.37%	1.68%	-2.64%
circleHollow	1.64%	1.64%	-0.04%	0.01%	-18.25%	-24.87%	1.56%	-3.56%
square	1.71%	1.71%	0.00%	0.00%	1.71%	-29.50%	1.71%	1.70%

Bubble	2.44%	2.48%	0.50%	-0.03%	-5.73%	-20.54%	2.75%	0.12%
multipleCircle	1.67%	1.67%	0.00%	-0.01%	-20.47%	-28.76%	1.26%	-2.67%
odd	2.08%	2.23%	0.12%	0.00%	-7.13%	-18.93%	2.19%	-0.83%
unrule	2.42%	2.51%	-0.11%	0.00%	-11.85%	-23.50%	1.03%	-2.67%
Capter	2.25%	4.09%	-1.24%	0.00%	-4.54%	-22.55%	-4.37%	-2.43%
girl	2.32%	1.87%	-0.04%	0.00%	-8.59%	-29.88%	1.32%	-4.09%
Einstein	3.65%	5.03%	0.06%	-0.33%	-4.67%	-12.51%	6.49%	-1.11%
Bill	1.38%	2.63%	-0.73%	0.00%	-9.08%	-24.43%	2.80%	-3.59%

表8、路徑優化後之旋轉角度統計變化量

旋轉角度統計	上下	左右	主對角	副對角	重複邊緣	DFS	BFS	邊緣 DFS
Circle	81.68%	81.67%	77.87%	77.89%	85.15%	72.35%	68.17%	72.02%
circleHollow	73.97%	74.03%	69.15%	69.19%	82.21%	66.38%	53.61%	75.22%
square	85.11%	83.78%	74.91%	75.23%	82.91%	97.77%	70.93%	83.12%
Bubble	77.22%	77.15%	65.69%	66.02%	78.04%	75.62%	48.10%	71.23%
multiple	80.72%	80.42%	73.67%	74.70%	85.05%	72.07%	85.31%	69.92%
odd	73.41%	73.75%	62.08%	62.01%	75.04%	75.05%	45.87%	64.38%
unrule	69.41%	69.31%	57.96%	62.03%	71.35%	67.54%	56.63%	58.28%
Capter	40.80%	39.37%	24.66%	23.42%	54.46%	67.52%	24.37%	44.89%
Lena	63.93%	54.09%	46.83%	49.65%	63.68%	73.81%	32.09%	54.29%
Einstein	48.81%	46.46%	45.91%	41.06%	58.06%	55.92%	52.63%	48.01%
Bill	58.04%	47.92%	50.67%	44.59%	61.62%	67.26%	39.39%	51.58%

表9、路徑優化後之畫筆狀態改變變化量

下筆次數	上下	左右	主對角	副對角	重複邊緣	DFS	BFS	邊緣 DFS
Circle	0.00%	0.00%	0.00%	0.00%	29.90%	0.00%	0.00%	0.00%
circleHollow	0.00%	0.00%	0.00%	0.00%	60.68%	22.73%	0.00%	18.18%
square	0.00%	0.00%	0.41%	0.41%	18.57%	0.00%	0.00%	0.00%
Bubble	0.00%	0.00%	0.00%	0.00%	49.13%	4.55%	5.42%	2.91%
multiple	0.00%	0.00%	0.70%	0.52%	47.42%	8.97%	85.52%	0.00%
odd	0.00%	0.00%	1.33%	0.10%	51.96%	6.45%	10.54%	8.03%
unrule	7.71%	1.44%	11.58%	9.89%	47.38%	14.90%	57.40%	9.88%
Capter	11.71%	2.44%	13.04%	10.06%	62.36%	21.15%	22.56%	17.13%
Lena	23.26%	17.10%	18.24%	20.18%	64.25%	30.17%	13.14%	10.58%
Einstein	33.73%	33.54%	38.43%	39.75%	65.64%	34.98%	41.29%	21.89%
Bill	33.38%	21.03%	27.98%	26.13%	62.77%	37.44%	25.69%	15.27%

路徑優化演算法將鋸齒修為較平滑的折線，以「Bubble」邊緣DFS作為舉例說明(表10)。此演算法亦修掉DFS產生之大量折線，以「Bubble」DFS作為舉例說明(表11)。

表10、「Bubble」邊緣DFS之原始與優化之差別

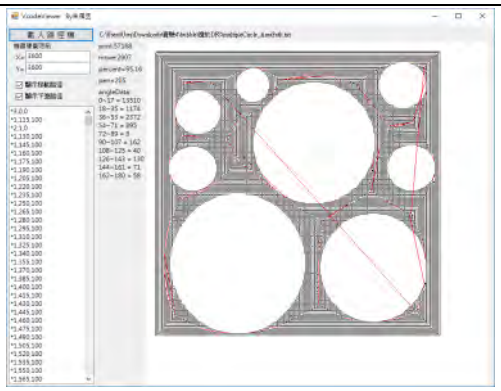
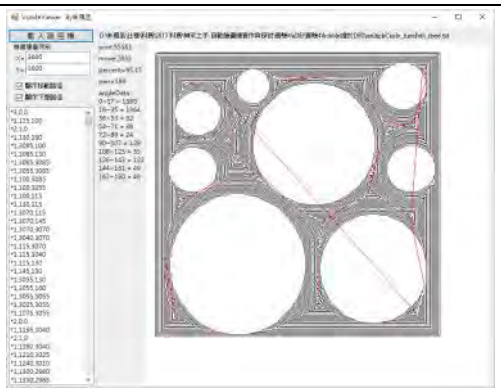
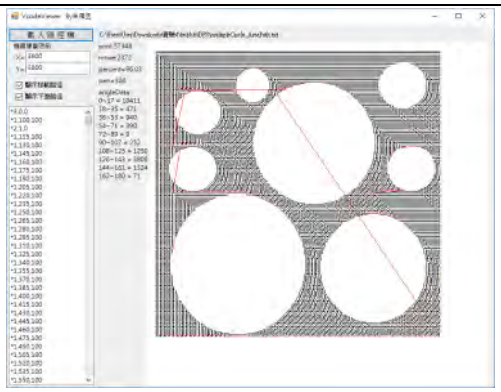
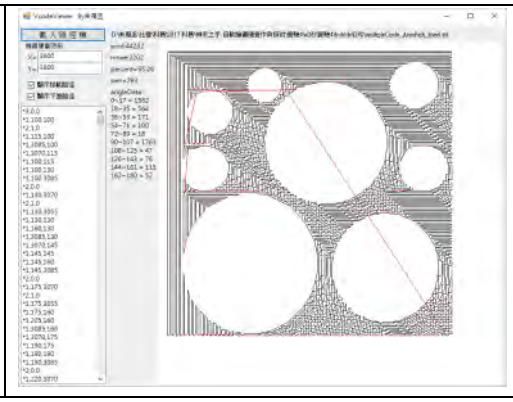
	路徑模擬工具模擬圖	屬性資料	角度統計資料
原始		print= 57168 move= 2907 pen= 206 percent=95.16	angleData: 0~17 = 13510 18~35 = 1174 36~53 = 2372 54~71 = 895 72~89 = 8 90~107 = 162 108~125 = 40 126~143 = 130 144~161 = 71 162~180 = 58
優化		print= 55561 move= 2833 pen= 200 percent=95.15	0~17 = 1380 18~35 = 1364 36~53 = 62 54~71 = 38 72~89 = 24 90~107 = 129 108~125 = 53 126~143 = 122 144~161 = 49 162~180 = 46

表11、「Bubble」DFS之原始與優化之差別

	路徑模擬工具模擬圖	屬性資料	角度統計資料
原始		print= 57348 move= 2372 pen= 308 percent=96.03	angleData: 0~17 = 10411 18~35 = 471 36~53 = 940 54~71 = 390 72~89 = 9 90~107 = 252 108~125 = 1250 126~143 = 3806 144~161 = 1324 162~180 = 71

優 化		print= 44232 move= 2202 pen= 294 percent=95.26	0~17 = 1582 18~35 = 564 36~53 = 171 54~71 = 100 72~89 = 18 90~107 = 1763 108~125 = 47 126~143 = 76 144~161 = 115 162~180 = 52

實驗小結：

(一) 移動效率

1. 此演算法對「BFS」造成大量路徑消失，因此造成「BFS」出現極端值。
2. 此演算法對「重複邊緣演算法」產生之大量鋸齒填充進行優化後，大幅減少鋸齒數，因此有較佳之效果。
3. 越複雜之圖形於此路徑優化演算法有越佳之效果。

(二) 填充率

1. 此演算法對「BFS」造成大量路徑消失，因此造成「BFS」出現極端值。
2. 此演算法對「上下掃描法」、「左右掃描法」與「BFS」皆有進步。
3. 因「DFS」產生之大量鋸齒填充路徑被優化，故填充率大幅下降。

(三) 旋轉角度統計

1. 以此演算法做優化後，對於所有圖形旋轉角度皆大幅減少。
2. 對於「Capter」、「Lena」、「Einstein」、「Bill」等複雜圖形，此演算法對其產生之優化效果有限。

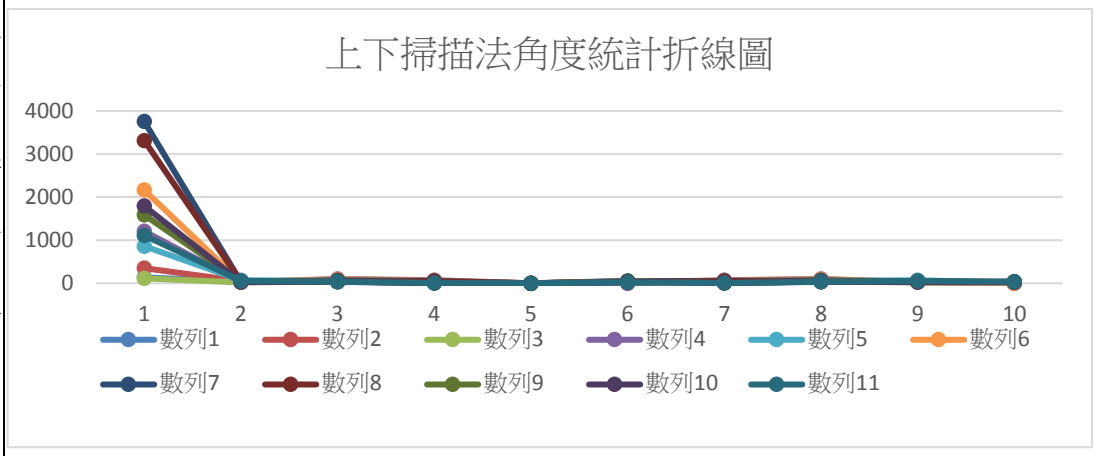
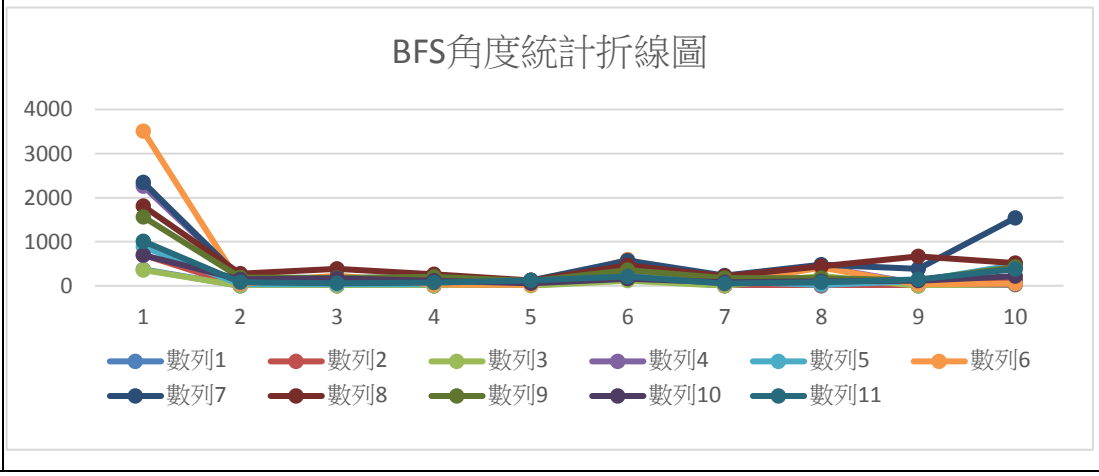
(四) 畫筆狀態改變次數

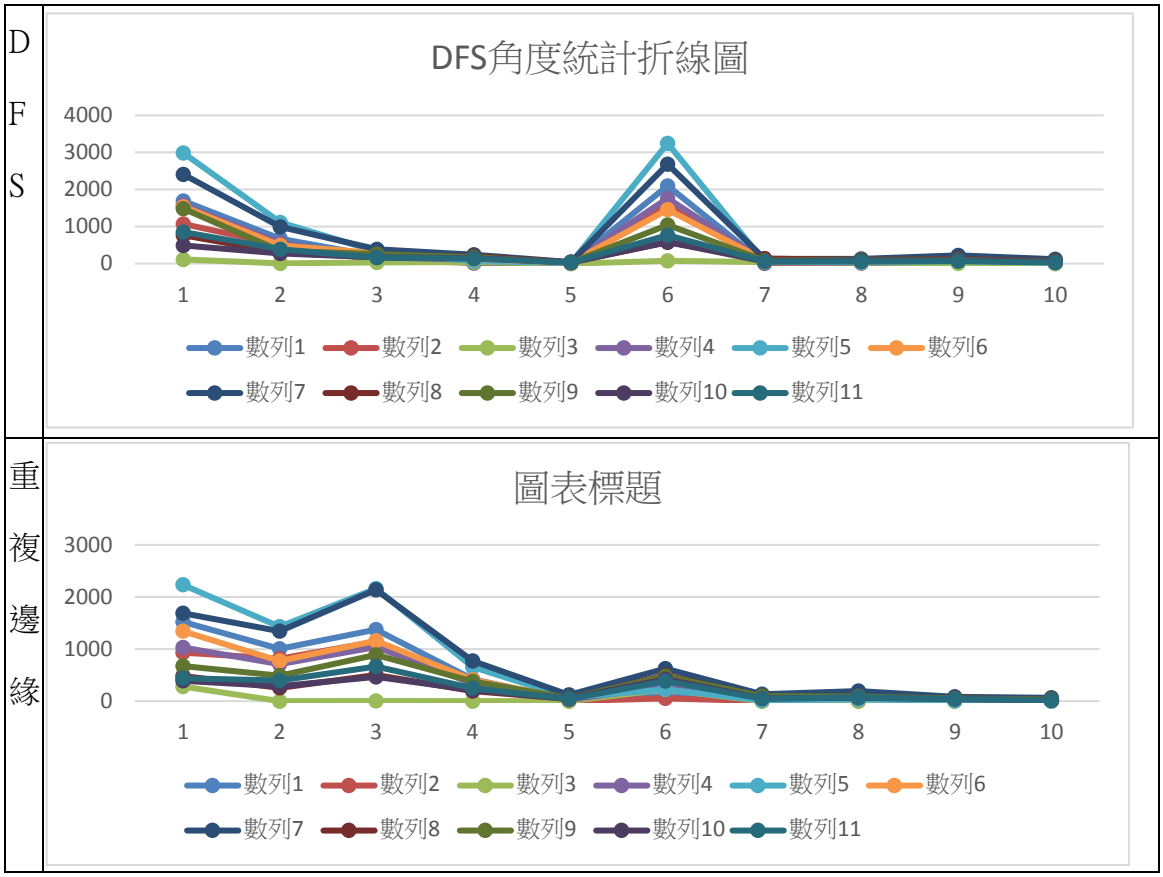
1. 以此演算法做優化後，大量冗餘點被捨去，因此對複雜圖形如「Unrule」、「Capter」、「Lena」、「Einstein」、「Bill」皆有不錯效果，但也可能造成原圖片失真。
2. 以此演算法做優化後，「重複邊緣」之效果顯著，推測其可能原因為其大幅修飾掉描點時所餘留下的冗餘點，因此大幅增進其效能。

實驗四、不同演算法於實際繪出之表現差異討論

實際藉由自動繪圖機繪出圖形時，發現掃描法均有較高之穩定性，而變形量普遍為「掃描法」普遍最低，「邊緣DFS」次之，而「DFS」再次之，「BFS」最後。列印時間則為「邊緣DFS」為最快。此外，在觀察列印過程時發現，大量失步多發生於繪製大量鋸齒路線時，由角度統計折線圖中，發現可能的失步率與角度變化之關係，在失步率最大的「BFS」中，相較於他者有較大量之大角度旋轉，在失步率次之的「DFS」與「重複邊緣」中，有較多的中角度旋轉；而掃描法的旋轉角度以小角度居多。此處以「上下掃描法」、「BFS」、「DFS」、「重複邊緣」為例(表12)。

表12、不同演算法之角度統計差異

	角度統計折線圖
上下掃描法	
BFS	



圖例：	數列3：square	數列6：odd	數列9：girl
數列1：Circle	數列4：Bubble	數列7：unrule	數列10：Einstein
數列2：circleHollow	數列5：multipleCircle	數列8：Capter	數列11：Bill

此外，由觀察繪出之成品，「DFS」會往固定方向偏移，然而，「邊緣DFS」會在完成部分區塊後，再進行其他區塊的繪圖工作，因此其偏移方向較無規律。以「Odd」之「DFS」與「邊緣DFS」比較之(圖23、圖24)。

圖23、「Odd」之「DFS」繪出結果

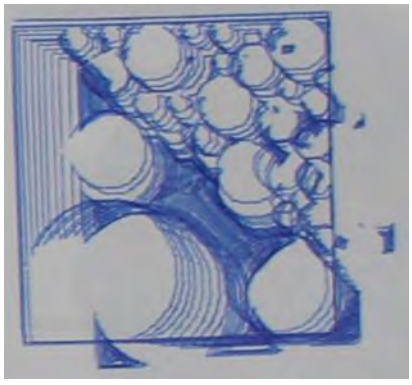


圖24、「Odd」之「邊緣DFS」繪出結果

陸、 討論與結論

一、由最後運算結果顯示，在移動效率、畫筆狀態改變次數等考量下，表現最優秀者為「邊緣DFS」；然而，實際繪圖時，「掃描法」卻因為旋轉角度大多分布於小角度，使得繪出圖片的變形量較低。

二、若目標圖形為簡單連續圖形，除了「BFS」的各演算法之移動效率皆可達到極高之表現，但若圖形過於分散或簍空，將大幅降低掃描法之移動效率。

三、若成品要求高「填充率」可選擇「斜向掃描演算法」，其能擁有最大填充之表現，且該特性不隨圖形複雜度而改變。

四、於旋轉角度統計的資料中發現，無論是何種圖形「掃描法」普遍擁有極佳之表現，代表該演算法可以盡量減少機器大幅改變行進方向之頻率。

五、對於複雜圖形，僅有「重複邊緣」、「DFS」、「邊緣DFS」能維持較高之移動效率。

六、「路徑優化演算法」對「旋轉角度統計」資料有大幅度改進，且經「VcodeViewer」預覽之路徑也較為平順，顯示該演算法對消去冗餘點之運作成效。

七、「BFS」對於本實驗所採用之圖形皆有最低之效果，因此推論「BFS」較不適合用於圖像路徑規劃。

八、「邊緣DFS」繪製圖形所耗費的時間最少，若能解決失步問題便能間得高效率與高品質的問題。

九、發現大多數失步皆發生於大量鋸齒或折返等路徑，未來開發路徑分析演算法應盡量避免此類路徑之產生。

柒、 參考文獻

[1] MSDN Stack<T>類別。取自
https://msdn.microsoft.com/zh-tw/library/3278tedw(v=vs.110).aspx

[2] MSDN Queue 類別。取自
https://msdn.microsoft.com/zh-tw/library/system.collections.queue(v=vs.110).aspx

[3] 二值化維基百科。取自
http://zh.wikipedia.org/wiki/%E4%BA%8C%E5%80%BC%E5%8C%963D

[4] 演算法筆記。取自 http://www.csie.ntnu.edu.tw/~u91029/

[5] 3D Printer 韌體原始碼解析心得 (以 Marlin 為對象)。取自
https://www.slideshare.net/roboard/3d-printer-marlin

【評語】052508

本作品主要針對自動繪圖機的繪圖方式找出有效以及具效率的路徑規劃演算法來移動繪圖針，使得繪圖時間短並且效果好。

作品之完成度高，且針對不同演算法有具體的效能分析與比較。

其研究成果可以很快的應用至實際系統。

作品海報

01 研究動機

隨著現代科技蓬勃發展，越來越多的工作交由電腦或機器人完成，舉凡文書處理、圖像列印、工業模型製造、CNC、3D 列印機等等。機器的工作效率取決於前置圖形運算工作與動力機械的運作是否能發揮最高效率。舉例來說，填滿一個面的過程中，若演算法充滿轉折並且必須不斷起筆來改變位置形成「無效移動」，將會增加失步與錯位的機率並且浪費許多無謂的時間。

一個好的路徑規劃演算法應該為機器爭取最少的運作時間、盡量減少機器傷害，並且同時達到最大目標效果。為了找出最佳演算法，本研究嘗試各種不同的演算法，並試圖在不同的圖像中表現其最佳路徑特性，並找出不同演算法對於不同圖案的相適性關係。

02 研究目的

綜合上述討論，本研究目的歸納如下：

一、製作出能夠透過步進馬達精確控制位置的座標系統裝置。

二、撰寫能夠依據指令移動座標軸馬達的 Arduino Firmware。

三、 以 C#撰寫多種演算法進行圖像資料處理並建構介面供使用者操作。

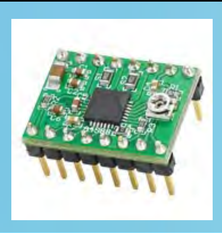
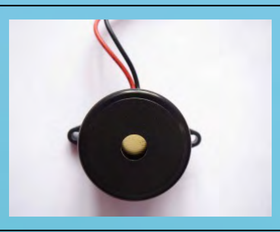
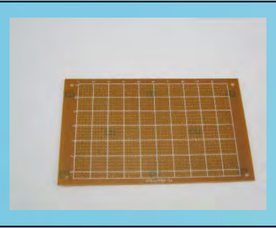
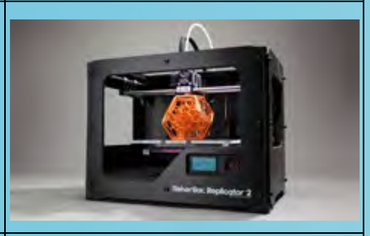
四、 以 C#撰寫路徑預覽工具方便預覽路徑並檢測該演算法之運算效果。

五、 找出不同的演算法對於不同幾何圖形之最佳相適性。

六、 分析不同演算法處理圖像時的特色。

七、 推測不同演算法發揮最大效益之時機。

03 研究設備及器材

		
Arduino Mega 2560	步進馬達	A4988
		
蜂鳴器	電阻	洞洞板
Visual Studio		
Visual Studio C#	Arduino IDE	3D printer

04 研究過程與方法

一、自動繪圖機製作

(一)、直角坐標滑輪系統

本研究利用鋁窗滑輪與鋁合金製作軌道，並以木工完成零件之固定。此自動繪圖機以 H 型結構(圖 1)進行 X、Y 軸之操控，透過控制步進馬達之步數達到座標定位的功能。

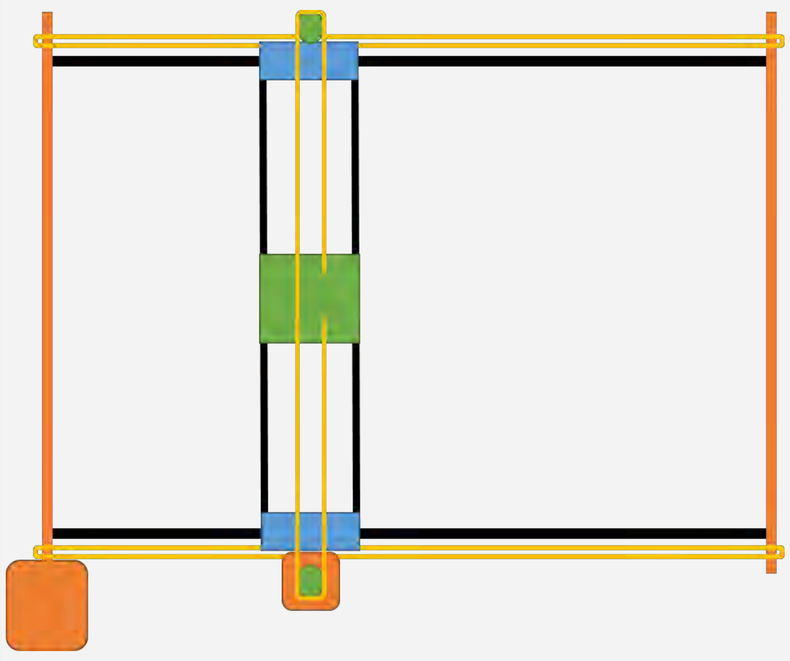


圖 1、自動繪圖機結構概念圖

(二)、A4988 步進馬達控制晶片

步進馬達的控制需要對 1A、2A 、1B、2B 四個腳位調控隨時間變化的輸出，此晶片可協助開發者省去類比信號輸出與功率放大之實作，提供最高 16 微步之類比驅動，使步進馬達藉由此晶片進行高精度之控制。

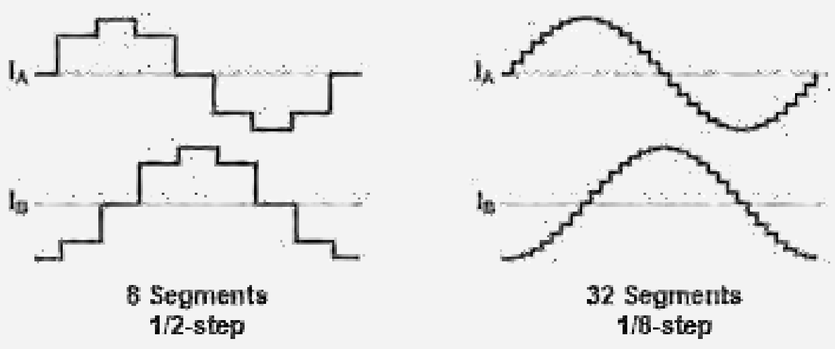


圖 2、8 微步與 32 微步電壓-時間關係圖

(三)、Arduino Mega 2560 與控制電路

利用微電腦 Arduino Mega 2560 進行 SD 卡資料讀取與控制步進馬達，電路架構圖如圖 3。

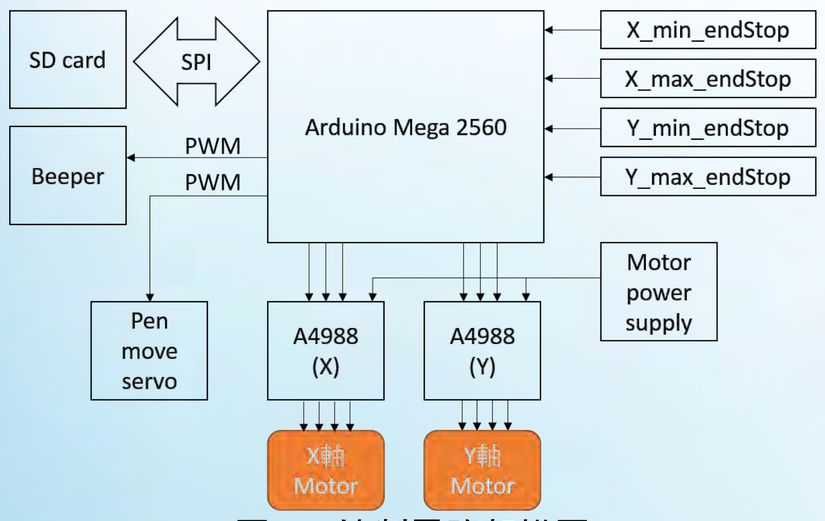


圖 3、控制電路架構圖

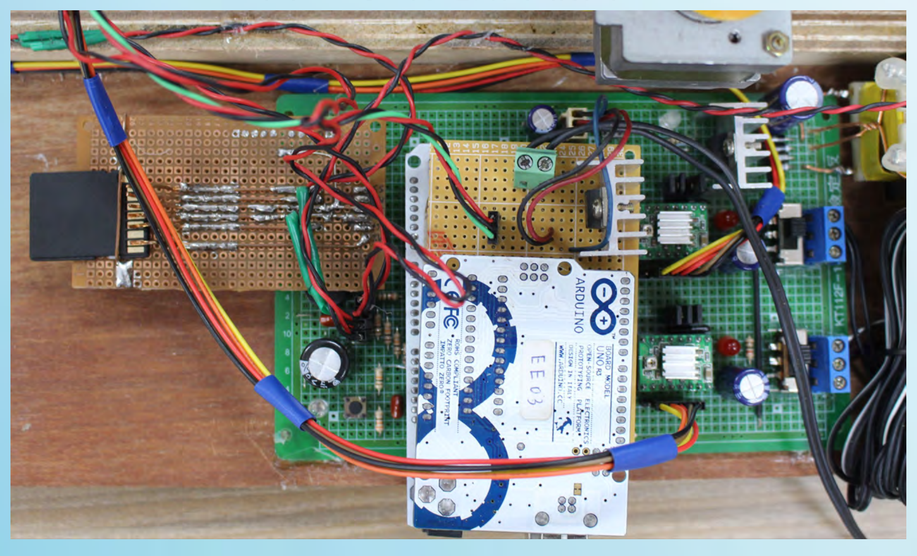


圖 4、控制電路成品

(四)、下筆起筆結構設計

以 AutoDesk 123D 繪製筆架(圖 5)，並以 3D 列印技術印製零件。

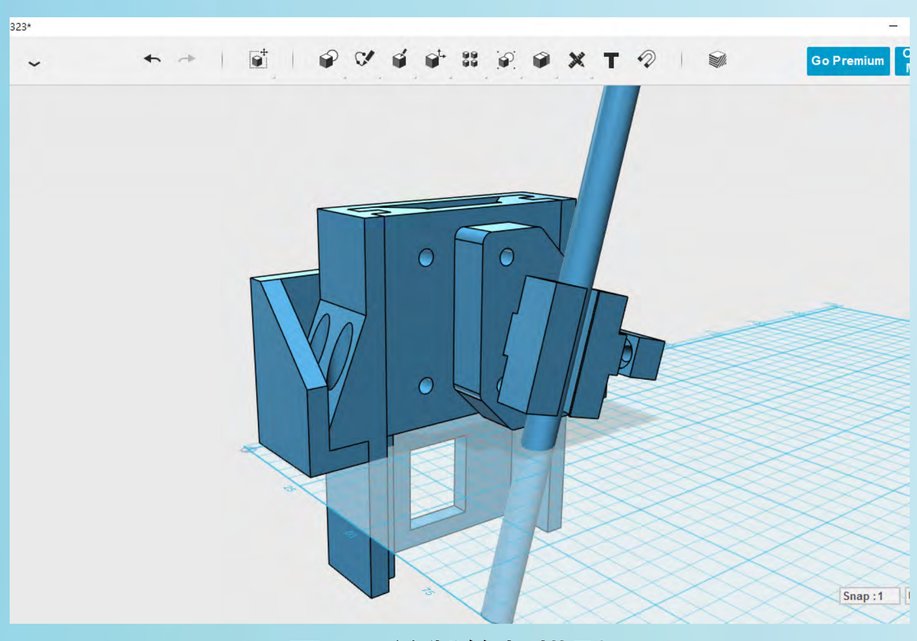


圖 5、繪製筆架模型

(五)、Vcode 指令

為使電腦輸出之路徑檔案可供 Arduino 讀取，必須在兩者之間建立可互相溝通的共通語言。此「Vcode」以一行為一指令單位，引數間以逗號區隔，其中包含此自動繪圖機專屬指令集。

(六)、Arduino Firmware

此程式分為兩大主要目的：

1. 讀取資料：讀取 SD 卡內的 Vcode 資料並解譯指令。
2. 執行指令：根據指令內容執行對應工作。

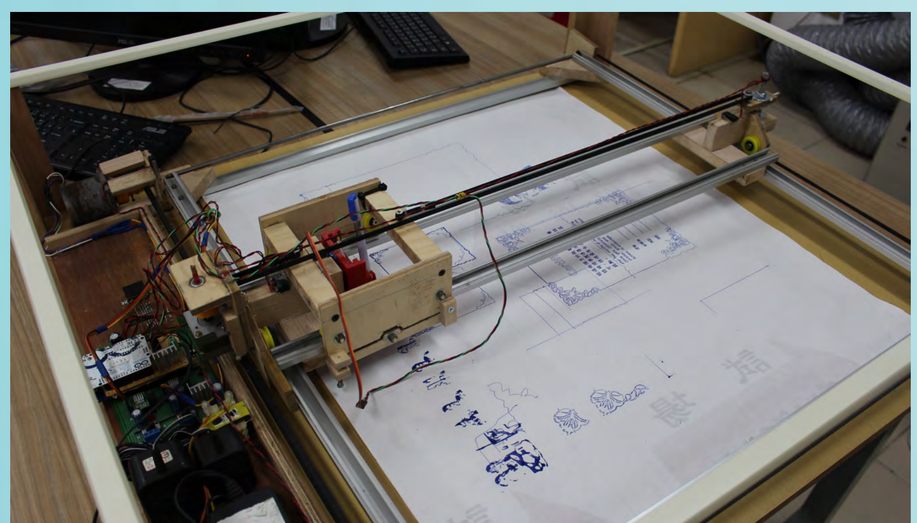


圖 6、自動繪圖機成品

二、撰寫路徑分析演算法

撰寫影像分析工具(圖 1)，以「循序掃描法」、「重複邊緣演算法」、「DFS 演算法」、「BFS 演算法」、「邊緣 DFS 演算法」，進行路徑分析。

圖 7、影像分析工具

(一)、主要架構介紹

1. 計算目標像素(goalColor)：利用灰度轉換公式 Gray = (R*30 + G*59 + B*11 + 50) / 100 取得原始圖片灰階值，再依據所輸入的灰階標準進行篩選。
2. 規劃面積路徑：以此功能選用不同演算法進行路徑規劃。
3. 路徑優化：以此功能進行多鋸齒修正、連續直線點省略與冗餘點捨去。

(二)、演算法介紹

1. 循序掃描法：以掃描法遍歷圖片陣列。本演算法依掃描方向之不同再分為四種：
   (1)左右型 (2)上下型 (3)主對角線 (4)副對角線
2. 重複邊緣演算法：重複進行邊緣檢測與描線，演算法如圖 8。
3. DFS 演算法：深度優先搜尋法，本研究採用堆疊(Stack)實作，演算法如圖 9。
4. BFS 演算法：廣度優先搜尋法，本研究採用佇列(Queue)實作，演算法如圖 10。
5. 邊緣 DFS 演算法：結合重複邊緣與 DFS 之特色，設計能以邊緣為優先深入方向之演算法，演算法如圖 11。

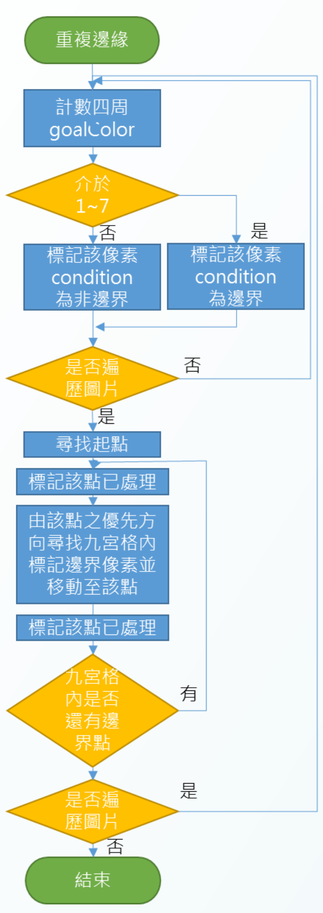


圖 8、重複邊緣演算法

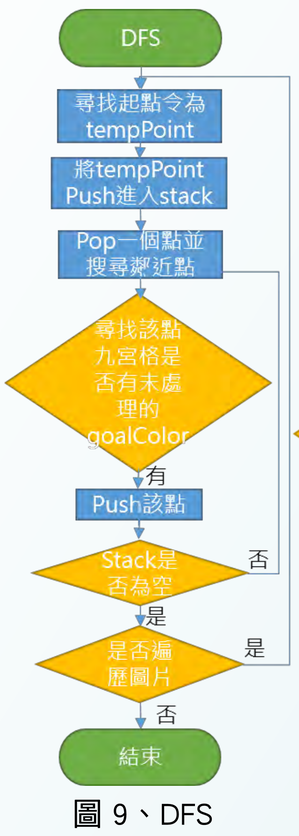


圖 9、DFS

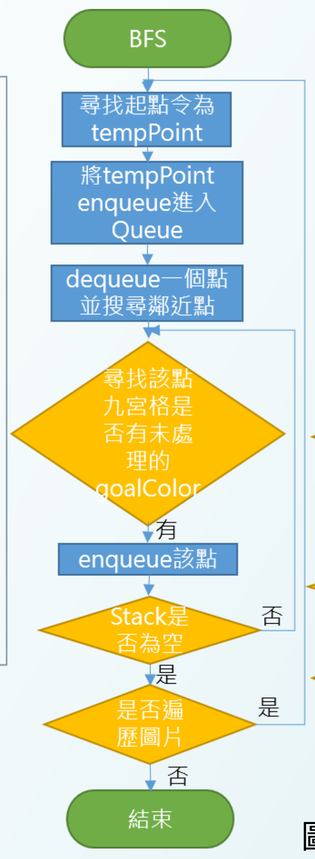


圖 10、BFS

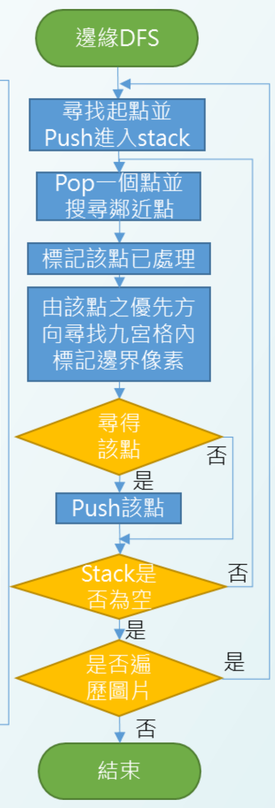


圖 11、邊緣 DFS

四、撰寫「VocdeViewer」路徑模擬程式

為模擬自動繪圖機畫筆運作，本研究以C#撰寫路徑模擬程式，讀取由「影像分析工具」產生之路徑檔，並用與Arduino Firmware相同的資料讀取演算法、執行演算法，模擬機器路徑運行，並記錄圖片之資料屬性，歸納出不同演算法對於不同圖形之特殊表現。

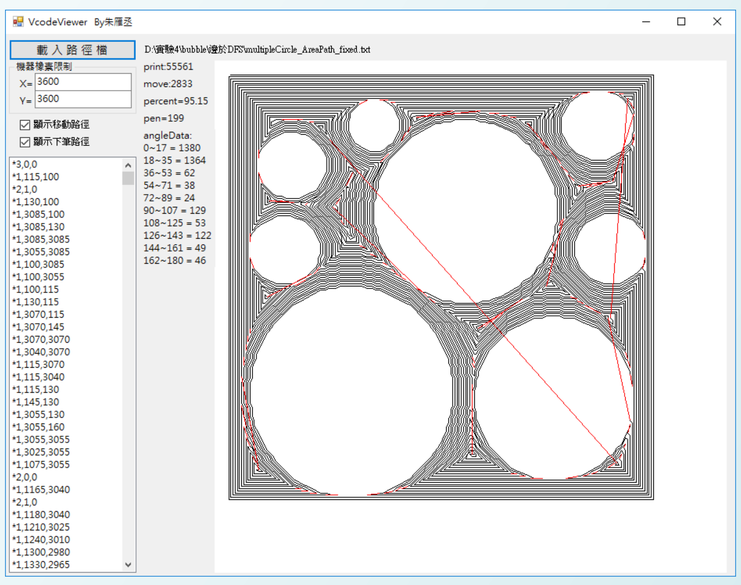


圖 12、Vcode Viewer

五、實驗進行與結果分析

(一)分析不同演算法之模擬路徑特性。
(二)比較由VcodeViewer獲得之屬性資料歸納演算法特性。
(三)演算法分析與優化，探討優化之結果。
(四)實際以自動繪圖機繪出並比較理論與實際差異，並推論差異原因。
(五)探討不同演算法應用於不同圖片之相適性關係。

05 研究結果與討論

一、 樣本描述

將不同的圖片以相同方式壓縮至 200*200 像素大小，本研究使用「Circle」、「Hollw」、「Square」、「Bubble」、「Multiple」、「Odd」、「Unrule」、「Capter」、「Lena」、「Einstein」、「Bill」等 11 種不同圖片。

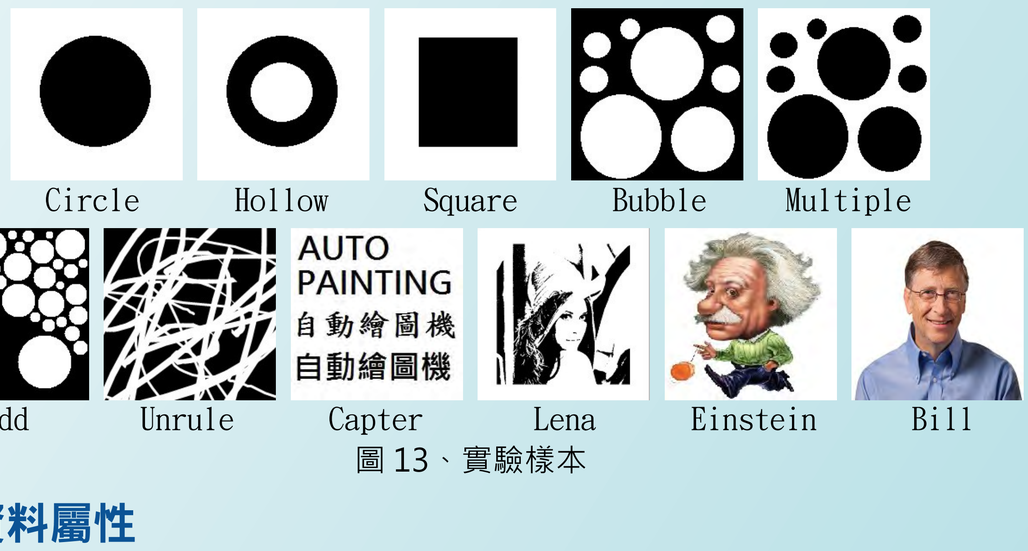


圖 13、實驗樣本

二、資料屬性

1.移動效率：為下筆路徑長與總路徑長的比值，值越大代表移動效率越高。
2.填充率：由下筆路徑長除以該圖片之所有運算路徑長最大值以計算填充率。
3.旋轉角度統計：紀錄路徑行進角度變化值，並以度數為單位進行統計。
4.畫筆狀態改變次數：紀錄畫筆狀態改變之次數。
5.預估耗時：藉由總路徑長、平均速度、畫筆改變次數、畫筆改變時間預估耗時。

實驗一、路徑模擬圖分析與優化

本實驗使用「循序掃描法」、「重複邊緣法」、「DFS」、「BFS」、「邊緣 DFS」對上述 11 種圖片進行分析，並產生路徑模擬資料，觀察路徑後發現容易產生多鋸齒、連續直線點、冗餘點等等問題，於是在「影像分析工具」內加入「路徑優化」功能，並分析此功能對各演算法之影響。

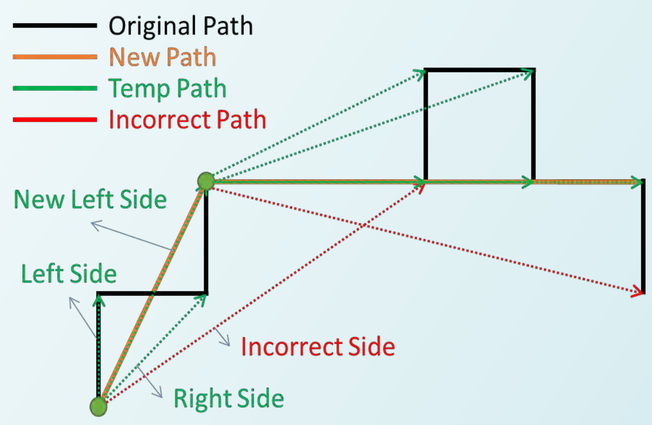


圖 14、冗餘點省略演算法

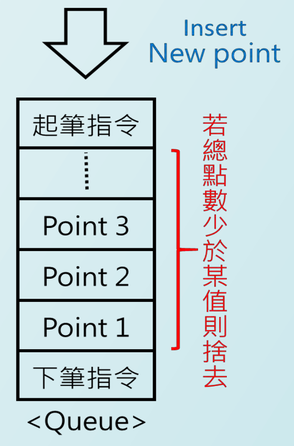


圖 15、冗餘點省略演算法

路徑優化演算法將鋸齒修為較平滑的折線，以「Bubble」之「邊緣 DFS 演算法」路徑模擬圖(圖 16、圖 17)舉例說明。

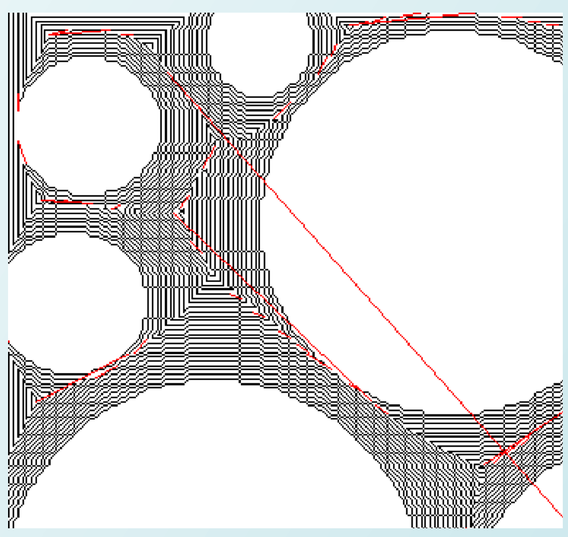


圖 16、Bubble_AreaPath.txt

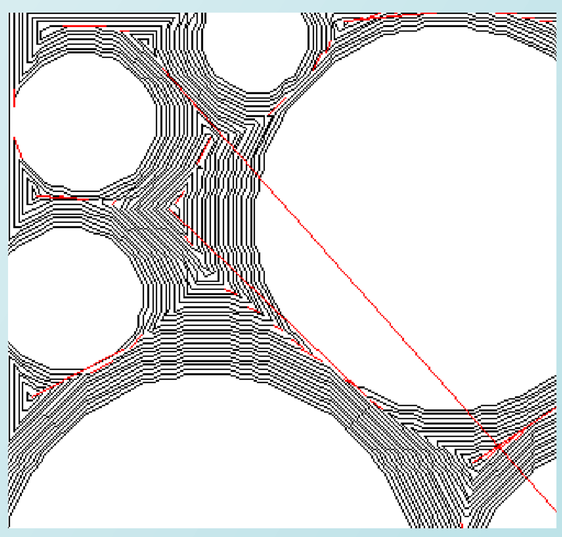


圖 17、Bubble_AreaPath_fixed.txt

實驗小結：

1.實驗發現越複雜之圖形使用此優化演算法有越佳之移動效率修正成果，推測其可能原因為複雜圖形容易產生較多的落單冗餘點 致使此演算法大量削減之。
2.此演算法對於「重複邊緣演算法」、「重複邊緣演算法」此類易產生大量鋸齒路徑之演算法有極佳之效果，大幅減少各分級層之角度數量，但對於「DFS 演算法」產生之大量大角度鋸齒優化則較為有限。

實驗二、不同演算法之特性分析比較

本實驗藉由路徑模擬工具產生之資料屬性與模擬圖分析不同演算法之特性。

表 1、各演算法之移動效率屬性資料

移動效率	Circle	Hollow	square	Bubble	multiple	odd	unrule	Capter	Lena	Einstein	Bill	平均
上下	98.26%	69.04%	98.91%	47.55%	76.16%	47.61%	53.77%	31.32%	55.54%	34.07%	49.09%	60.12%
左右	98.27%	69.03%	98.75%	47.53%	76.17%	47.63%	53.77%	37.54%	40.92%	43.44%	50.42%	60.32%
主對角	98.29%	69.06%	98.13%	47.21%	80.41%	47.25%	52.56%	27.60%	48.91%	38.57%	49.16%	59.74%
副對角	98.26%	69.04%	98.13%	47.20%	78.01%	47.37%	57.27%	29.05%	50.33%	45.96%	60.46%	61.92%
重複邊緣	99.19%	96.82%	99.62%	93.97%	96.49%	47.63%	86.47%	77.30%	40.92%	68.90%	80.20%	80.68%
DFS	99.27%	96.60%	99.32%	95.26%	96.97%	91.83%	86.85%	73.23%	81.05%	72.53%	77.95%	88.26%
BFS	54.39%	36.73%	57.16%	22.84%	42.25%	22.31%	37.11%	16.57%	25.53%	24.34%	27.27%	33.32%
邊緣 DFS	100.00%	98.11%	100.00%	95.15%	98.29%	94.11%	90.98%	80.65%	87.84%	79.25%	88.01%	92.04%

表 2、各演算法之角度統計屬性資料

角度統計	Circle	Hollow	square	Bubble	multiple	odd	unrule	Capter	Lena	Einstein	Bill	平均
上下	24382	26218	21375	45889	40834	54385	67842	60578	39843	36035	33831	41019
左右	24401	26160	23625	46025	41617	53650	67842	56601	53677	41494	46023	43737
主對角	33795	36008	44916	83368	66181	92997	111710	97143	79136	46394	48639	67299
副對角	33781	35968	43464	82759	62950	93382	100673	101739	73046	57193	59624	67689
重複邊緣	141051	110113	24730	149684	229714	53650	328521	130743	53677	114622	124439	132813
DFS	257518	197095	22131	259484	421120	251114	432821	153478	196885	127222	141207	223643
BFS	48847	59622	48450	156989	170595	178661	521537	385906	218623	122124	152046	187582
邊緣 DFS	93882	64228	24433	105571	165434	144209	259718	118596	140797	102918	106402	120562

表 3、各演算法之填充率屬性資料

填充率	Circle	Hollow	square	Bubble	multiple	odd	unrule	Capter	Lena	Einstein	Bill	平均
上下	70.69%	70.67%	70.67%	71.66%	70.65%	71.40%	71.50%	71.12%	71.10%	71.92%	70.54%	71.08%
左右	70.69%	70.65%	70.72%	71.64%	70.69%	71.47%	71.49%	72.31%	70.77%	73.11%	71.29%	71.35%
主對角	99.98%	99.96%	100.00%	100.00%	100.00%	99.64%	99.67%	98.63%	99.93%	100.00%	99.01%	99.71%
副對角	100.00%	100.00%	100.00%	99.97%	99.99%	100.00%	100.00%	100.00%	100.00%	99.67%	100.00%	99.97%
重複邊緣	55.18%	56.10%	70.35%	64.56%	54.46%	71.47%	58.50%	59.82%	70.77%	55.72%	56.97%	61.26%
DFS	57.52%	59.79%	54.01%	61.23%	57.03%	61.53%	59.38%	57.48%	55.21%	64.47%	58.33%	58.73%
BFS	69.28%	67.30%	69.45%	65.53%	47.25%	62.18%	45.85%	33.66%	57.11%	48.17%	54.38%	56.38%
邊緣 DFS	76.10%	76.83%	70.67%	76.92%	76.07%	77.92%	76.40%	69.04%	73.96%	75.64%	75.31%	74.99%

表 4、各演算法之下筆次數屬性資料

下筆次數	Circle	Hollow	square	Bubble	multiple	odd	unrule	Capter	Lena	Einstein	Bill	平均
上下	258.00	402.00	230.00	816.00	814.00	1464.00	2608.00	2428.00	1234.00	1332.00	886.00	1133.82
左右	258.00	396.00	254.00	818.00	816.00	1432.00	2608.00	2878.00	2162.00	1086.00	1232.00	1267.27
主對角	366.00	566.00	480.00	1152.00	1142.00	2070.00	3956.00	4096.00	2402.00	1400.00	1416.00	1731.45
副對角	366.00	566.00	480.00	1152.00	1144.00	2096.00	2914.00	4058.00	2104.00	1376.00	1278.00	1594.00
重複邊緣	136.00	92.00	114.00	408.00	326.00	1432.00	1064.00	536.00	2162.00	534.00	452.00	659.64
DFS	46.00	68.00	78.00	294.00	142.00	406.00	674.00	492.00	338.00	368.00	274.00	289.09
BFS	254.00	426.00	248.00	1606.00	990.00	2258.00	2882.00	2698.00	1494.00	836.00	1018.00	1337.27
邊緣 DFS	2.00	36.00	2.00	200.00	16.00	252.00	438.00	358.00	338.00	314.00	222.00	198.00

表 5、各演算法之預估耗時屬性資料

預估耗時	Circle	Hollow	square	Bubble	multiple	odd	unrule	Capter	Lena	Einstein	Bill	平均
上下	106.40	149.60	101.23	331.90	304.16	526.25	865.31	777.29	406.26	428.76	291.73	389.90
左右	106.40	147.80	108.51	332.50	304.79	516.69	865.62	904.87	697.31	349.05	395.12	429.88
主對角	150.81	210.81	189.97	468.01	422.99	743.39	1305.02	1305.72	778.15	455.82	461.15	590.17
副對角	150.83	210.84	189.97	468.00	426.06	751.32	983.06	1291.50	687.16	442.76	413.25	546.79
重複邊緣	63.23	44.02	66.06	162.10	134.29	224.53	361.38	177.46	218.19	171.37	148.42	161.00
DFS	37.16	37.94	47.93	125.35	80.62	160.69	244.83	164.50	120.59	122.68	95.70	112.54
BFS	127.54	179.71	129.21	647.60	369.29	839.14	941.63	853.14	511.20	278.15	341.39	474.36
邊緣 DFS	31.28	32.99	32.48	106.72	54.83	123.66	183.76	125.83	125.12	107.39	82.04	91.46

「邊緣 DFS 演算法」相較於「重複邊緣演算法」與「DFS 演算法」有更豐富的角度變化，能夠產生更細膩的路徑，以「Lena」(圖 18、19、20)作為舉例說明。

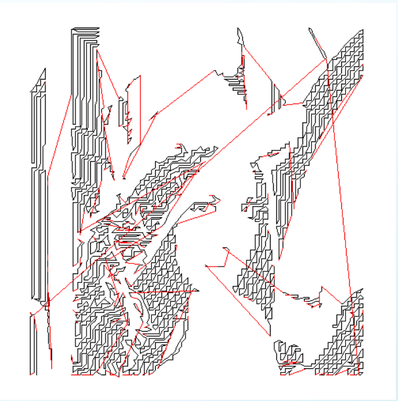


圖 18、DFS 演算法

圖 19、重複邊緣演算法

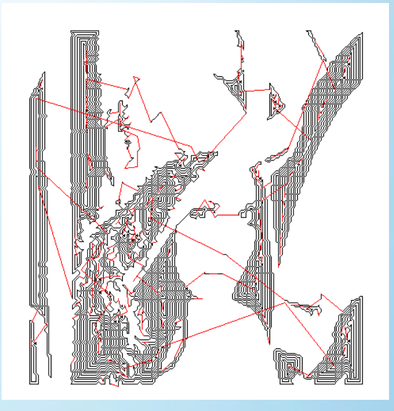


圖 20、邊緣 DFS 演算法

實驗小結：

(一)、循序掃描法

1.連續實心圖形有最佳表現，但遇到不連續圖形或空心圖形效率會大幅下降。
2.斜向掃描法之填充率為所有演算法中最高，預估耗時亦為所有演算法中最久。
3.斜向掃描演算法為所有演算法中產生較多畫筆狀態改變次數者。
4.掃描法因為下起筆集中於邊緣區域，可能造成邊緣不平整與不連續問題。
5.不同掃描方向於不同方向有不同程度之移動效率，應用該種演算法前先判斷目標圖形之目標點分布可提升效率。

(二)、重複邊緣演算法

1.重複邊緣演算法提升了簡單不連續圖片、簡單空心圖片的移動效率。
2.重複邊緣演算法於描邊時會不規律的改變繞行方向。

(三)、DFS 演算法

1.DFS 演算法相較於掃描法和重複邊緣有更高的移動效率。
2.DFS 演算法在角度表現最差，且路徑優化改善效果較有限，可能造成多次方向變換損傷機器或失步的機會。

(四)、BFS 演算法

1.BFS 演算法於移動效率、填充率皆為最差。

(五)、邊緣 DFS 演算法

1. 邊緣 DFS 演算法移動效率相較於「重複邊緣」、「DFS」，對於所有圖形皆有進步，為所有演算法中最高者，且畫筆改變次數為所有演算法中最低者，推測亦因此有最短的預估耗時，能達到最高效率。
2. 邊緣 DFS 演算法旋轉角度統計相較於「重複邊緣」與「DFS」皆大幅下降。
3. 邊緣 DFS 演算法能將區域失步位移控至於較相近位置，於不連續圖形之表現相對於其他演算法有更大優勢，因其較能維持圖形之完整性。

實驗三、實際繪出圖形比較分析

實際以自動繪圖機繪出圖形時，發現「循序掃描法」均有較低之變形量。列印時間則以「邊緣 DFS 演算法」為最快，「斜向掃描法」最慢。此外，在觀察列印過程時亦發現，大量失步多發生於繪製大量鋸齒路線時，由角度統計折線圖中分析失步率與角度變化之關係，在失步率最大的「BFS 演算法」中，相較於他者有較多之大角度旋轉，在失步率次之的「DFS 演算法」與「重複邊緣演算法」中，有較多的中角度旋轉；而「循序掃描法」的旋轉角度以小角度居多。

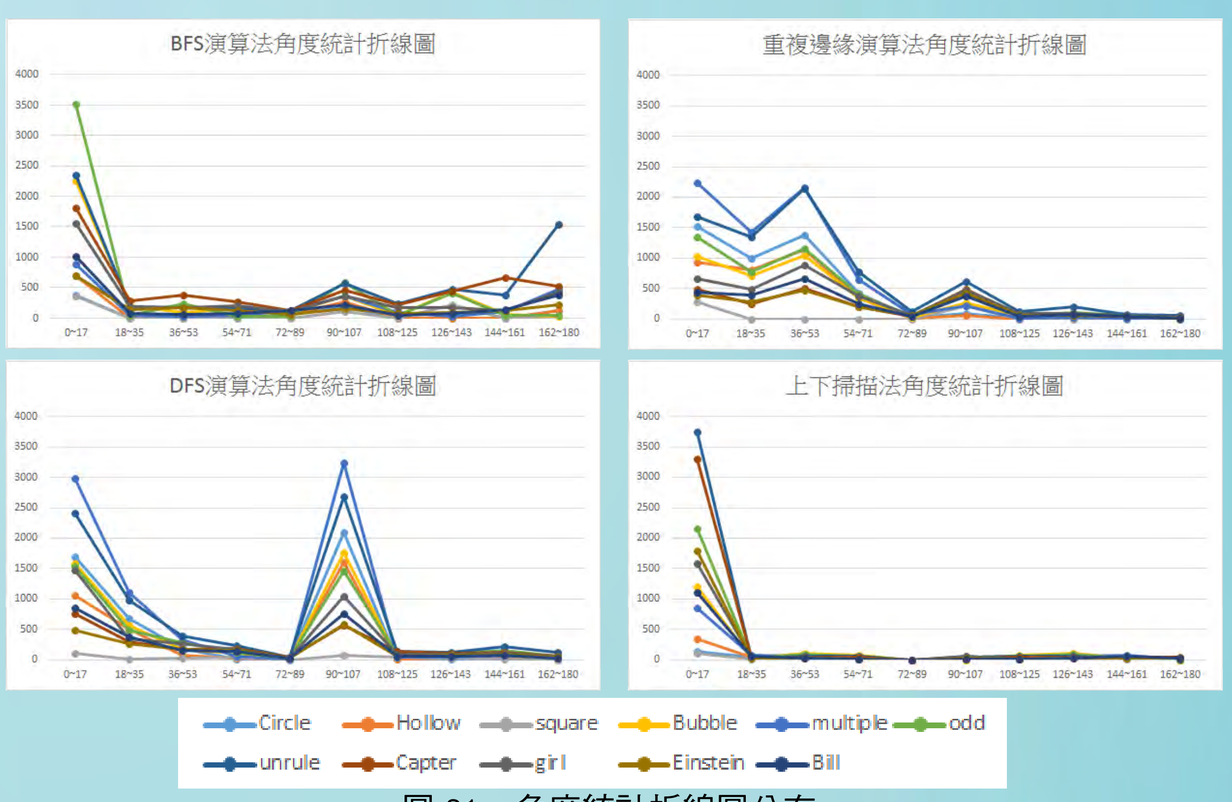


圖 21、角度統計折線圖分布

利用路徑屬性資料以及相似度問卷調查進行相關係數計算，更確認角度統計資料與相似度之關係；而預估耗時主要與下筆次數與總路徑長相關。

表 6、資料屬性相關係數

相關係數	總路徑長	移動效率	角度統計	下筆次數	預估耗時	相似度
移動效率	-0.82					
角度統計	-0.31	0.22				
下筆次數	0.86	-0.70	-0.43			
預估耗時	0.88	-0.70	-0.39	0.97		
相似度	0.09	0.15	-0.71	0.30	0.28	
填充率	0.54	-0.07	-0.49	0.30	0.59	0.58

觀察「DFS 演算法」與「邊緣 DFS 演算法」對於空心圖形繪出之成品，發現「DFS 演算法」失步方向較為固定，而「邊緣 DFS 演算法」失步方向較為分散，推測可能原因與該演算法探索方式相關。「邊緣 DFS 演算法」會在完成部分區塊後，再進行其他區塊的繪圖工作，因此偏移方向相對較無規律，以「Bubble」為例(圖 22、23)。

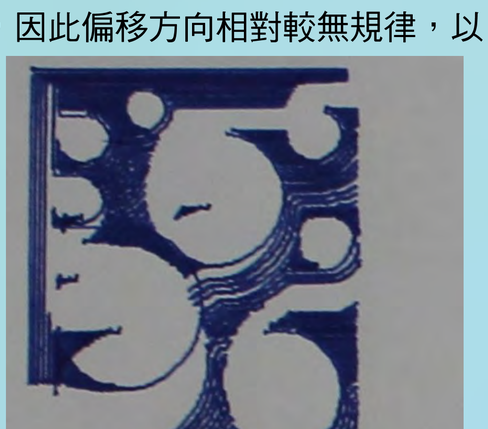


圖 22、DFS 演算

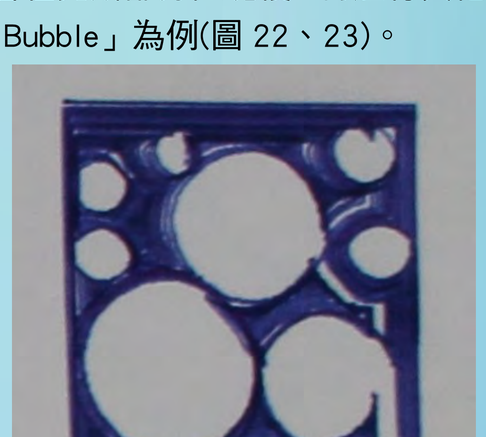


圖 23、邊緣 DFS 演算法

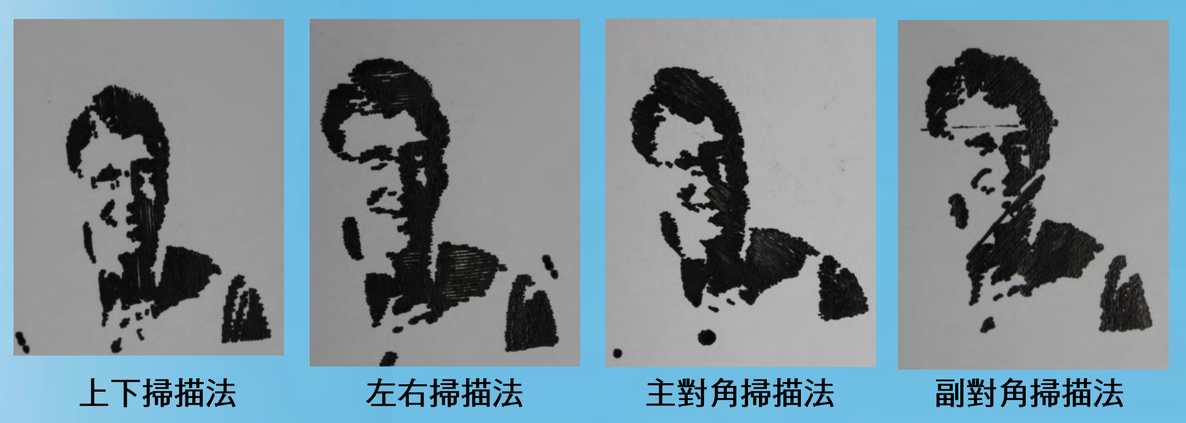


圖 24、自動繪圖機圖形「Bill」不同演算法實際繪圖結果

06 討論

各演算法特色：

一、直橫向掃描法：花費時間較久、移動效率較低，但失步率較小，位移方向較為規律，然而因下起筆多集中於邊緣，因此會造成邊緣不平整。適合需要高精度成品需求之任務，例如精密零件製作、加工或雕刻。

二、斜向掃描法：雖然有花費時間最久、移動效率較低與邊緣不平整問題，但填充率較高，位移量較直橫向掃描法略多，適合需要高密度成品需求之任務，例如雷雕機等加工點面積較小之機器。

三、重複邊緣演算法：由於描線方向較不規律，使空心圖形變形方向較為分散，若於不連續圖形有較好之表現。

四、DFS 演算法：由於失步方向較單一造成位移累加，因此失步量較多，即使為簡單幾何圖形也可能造成嚴重失步。

五、BFS 演算法：在各屬性資料考量下較不適合應用於此類專案。

六、邊緣 DFS 演算法：因區域優先完成的性質，於不連續圖形變形量較不顯著。花費時間最少且失步率偏低，主要集中在相同區域，可在效率與品質間兼顧。若應用於 3D 列印機、CNC 加工機等須同時兼顧效率與品質等機器類型將可達到最大效益。

07 研究結果

1. 最後運算結果與實際繪圖顯示，在移動效率、填充率、旋轉角度統計、畫筆狀態改變次數、繪圖耗時與演算法特性等多種考量下，於此自動繪圖機表現最優秀者為「邊緣 DFS 演算法」。
2. 若目標圖形為簡單連續圖形，除了「BFS 演算法」，「重複邊緣演算法」、「DFS 演算法」、「邊緣 DFS 演算法」皆能維持較高之移動效率，但若圖形過於分散或鏤空，將大幅降低「循序掃描法」之移動效率。
3. 若成品要求高「填充率」可選擇「斜向掃描演算法」，其擁有最大填充率，且該特性不隨圖形複雜度而改變。
4. 於旋轉角度統計的資料中發現，無論是何種圖形「循序掃描法」普遍有極佳之表現，而「邊緣 DFS 演算法」次之，代表該演算法可以減少機器大幅改變行進方向之頻率，減低失步率與機器耗損。
5. 「路徑優化演算法」對「旋轉角度統計」資料普遍有大幅度改進，且經「VcodeViewer」預覽之路徑也較為平順，顯示該演算法之運作成效，但對於「DFS 演算法」所產生的大量大角度鋸齒僅能有限度改善。
6. 「BFS 演算法」對於本實驗所採用之圖形普遍有最差之效果，以自動繪圖機繪出之結果亦因大量失步造成圖像變形，因此就目前情況推論「BFS 演算法」較不適合用於圖像路徑規劃。
7. 機器運作之失步原因與移動方向變化的角度大小有密切之關係，因此欲使機器有最佳效果應優先注意演算法產生鋸齒之嚴重程度。
8. 於圖 24 的比對中，發現各種演算法於描繪圖形時皆有特色，對於不同種類之圖形亦有不同程度之相適性，未來開發演算法時，若能利用本研究歸納之演算法特色，便能減少開發新演算法所需耗費之時間與成本。

未來發展：

一、 未來若開發能夠因應於相同工作件，針對不同區塊不同特性之需求，而應用不同演算法產生最合適路徑之程式，便能使成品達到最佳之效果。例如，3D 列印機中的支撐、內部填充、外圈填滿等，需求各不相同。支撐需要速度快，卻不要求最高精度；外圈填滿需要極高精度；而內部填充則介於兩者之間。

二、 構想「邊緣 DFS 演算法」時，將「DFS 演算法」與「重複邊緣演算法」特色結合，得到更佳的演算法。未來若能多方面結合不同演算法之特性，將能發掘演算法之無限潛力。

08 參考文獻

[1] MSDN。Stack<T> 類別【部落格文字資料】。取自 https://msdn.microsoft.com/zh-tw/library/3278tedw(v=vs.110).aspx

[2] Roboard(民 103 年 6 月 21 日)。3D Printer 韌體原始碼解析心得【部落格文字資料】。取自 https://www.slideshare.net/roboard/3d-printer-marlin

[3] 國立臺灣師範大學資訊工程學系。演算法筆記【部落格文字資料】。取自 http://www.csie.ntnu.edu.tw/~u91029/index.html