東海大學資訊工程學系

專題報告書

Arduino 電路圖像識別與偵錯

Arduino circuit image recognition and debugging

指導教師：周忠信

專題成員：s05310162 張子威

中華民國 111 年 6 月 10 日

誌謝

首先，我們於此衷心的感謝指導老師一直以來的莫大幫助，很榮幸可以成為老師的專題學生。最讓我們深刻的是老師的隨和與耐心，每當我們向老師提出疑

同時，老師總是盡心的講解分析。更甚者，當我們於一個問題多有糾結時，老師會解釋幾遍以利我們理解。簡而言之，老師對我們的幫助不僅是提供研究的方向與方法，更是以身作則的教育我們分享知識時應有的謙遜隨和之態度。

其次，我們要感謝所有資工系乃至東海的老師。我們從剛入學時的懵懂無知，到現在對計算機各領域架構有較清楚的認識，可以說是各位老師的辛勤付出為我們打開了探索計算機領域的大門。老師們循循然善誘人的風采，令人難以忘卻。

最後，再次感謝指導老師和其他諸位老師的栽培，學生們無以為報。

摘要

Arduino 開源嵌入式平臺目前在世界上廣範流行，而且往往是軟體開發的初學者或者興趣愛好者第一個接觸到的程式開發平臺。所以對於 Arduino 的學習方法對於軟體學習和推廣至關重要。

目前已有的 Arduino 學習輔助系統往往針對與原理和接法的教學，只有少數涉及到交互性的學習和糾錯，因此這方面的 Arduino 學習輔助系統比較欠缺。目前已有的類似系統常常是需要電腦和外接設備輔助和向導幫助使用者學習，或者完全是電腦模擬沒有操作感。

其次學生如果想要取得進步，不可以依賴這些向導工具，而是需要藉助自己的思考，而我們這個軟體的作用，則是像一個老師一樣，批改學生的作業，檢查學生成果是否正確。老師是如何檢查的，看到電路圖，再看看學生的接綫，來判斷是否接對，我們就是用圖像識別來模擬這個過程。

爲此本專題會打造一個僅僅使用攝像頭就可以對 Arduino 進行接綫糾錯軟體，這樣初學者可以不藉助額外的設備就可以對自己的 Arduino 進行糾錯，老師教導學生時也可以不用總是把精力耗費在糾錯上。從而增加自學者的學習效率，降低教師的工作

關鍵詞：Arduino，卷積神經網路 (Convolutional Neural Network)，電子元件識別(Electronic component identification)，導綫識別(Wire Identification)，邊緣檢測(Edge detection)，接綫電路糾錯(Wiring circuit error correction)

目錄

圖目錄

第四章 實證分析

第一章 緒論

1.1 研究背景

隨著各類電腦成本的降低和其在各行各業的廣範使用，越來越多的人開始學習計算機知識，尤其是各種嵌入式設備，比如 Arduino[1]開源嵌入式平臺，這是目前非常熱門的一款嵌入式設備開發系統，也是初學者和業餘愛好者第一款接觸的電腦設備。所以其很時候被用作於初學者對於計算機知識的學習和實作方面上，市面上也有大量的輔助學習的書籍。

對於使用 Arduino 的初學者來說，無論是有老師或者助教輔導的學生還是自學者，在實際操作過程中，經常有綫路接完執行發生錯誤或者無法執行的情況，甚至不慎導致設備損壞。因爲教學書籍和視頻能提供的接綫圖，但是接綫過程還是初學者獨自完成，所以錯誤往往發生在接綫過程中。

學生還可以通過向老師詢問來解決這個問題，但是無疑會耗費老師大量時間而無法解決更重要的問題，自學者就只能通過一邊又一遍的檢查才有可能能解決一個可能很輕微的連接錯誤。這對初學者的學習熱情是非常大的考驗，而學習熱情是初學者最重要的東西。

本文將通過圖像識別的方法，使用者可以通過相機拍攝的方式，讓軟體識別出圖中接錯的導綫或者元件，並給予一定提示，從而大大減少耗時耗力的檢查過程，加快初學者的學習進度和效率。

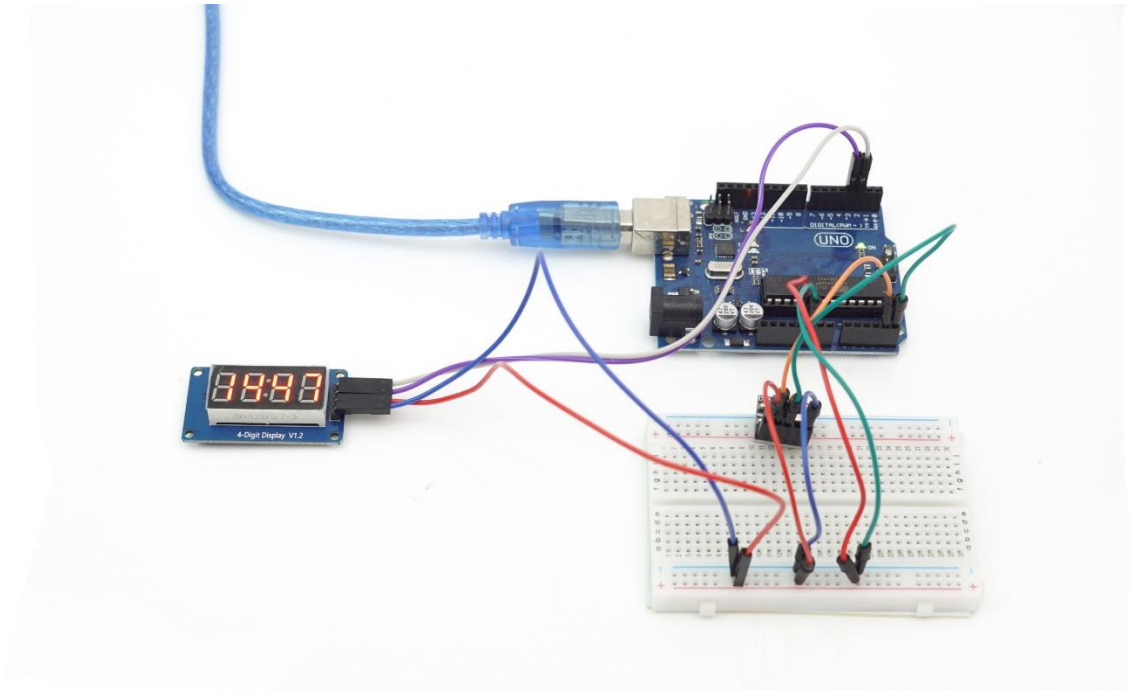

圖 1.1 常見的 Arduino 接綫電路

1.2 研究動機

以往 Arduino 的學習需要花費金錢找老師或者花費時間精力自學，這也是大多數人認爲其難度高的原因。市面上已有的一些 Arduino 接綫輔助產品，比如說吳德彥等的[1]CurrentViz 感測和可視化麵包板電路，其通過實體外接設備偵測麵包板目前的電流情況，也可以變相發現大部分綫路接錯的情況，這樣非常直觀可以省去測電流的時間，但是需要一定的經驗，不適用於初學者，因爲只能使用他們提供的麵包板和設備，適用範圍也比較少。

Daniel Drew 的[2] Toastboard：無處不在的儀器儀錶和麵包板電路的自動檢查也是類似設備，其通過其所提供的特殊麵包板設備可以偵測其上面的接綫情況，並反應到電腦軟體上與使用者交互，檢查接綫情況。這種對話模式快速準確但是需要外接專用設備和電腦，比較繁瑣昂貴，並且也只能使用設備自帶的麵包板，使用起來缺乏擴展性和實用性，並且無法偵測 Arduino 板上的接綫情況，會有遺漏。。

所以我們希望研究出一款可以通過簡單設備，比如手機，通過照相功能就可以隨時偵測出自己的 Arduino 設備接綫情況的軟體。這樣可以

使用自己設備，不需要額外購買偵測設備。

不需要電腦，用手機即可知道結果。

適用於大部分情況，不會有麵包板限制。

幫助初學者學習，簡單易用，提升學習效率。

1.3 研究目的

1.3.1 研究步驟

研究分爲 4 大步驟:

導綫位置偵測

電子元件位置偵測

與正確位置資訊對比

標出接錯導綫和元件

1.3.2 解決問題

可以解決 3 大問題:

1. 隨時隨地地使用

無需外接設備和電腦，只需要手機就可以隨時隨地的使用軟體，增加可以應用的場景和使用者便利性。

2. 增加初學者學習效率

通過本專題的軟體之應用，可以使用拍照的方式與正確的接綫方式做對比找出錯誤的接綫，節省學習時間，增加學習效率。

3. 輔助老師教學

老師上課時候也可以使用此軟體讓學生自主檢查接綫情況，讓老師的精力不必放在檢查綫路上，可以用於更重要的地方。

1.3.3 關係機構

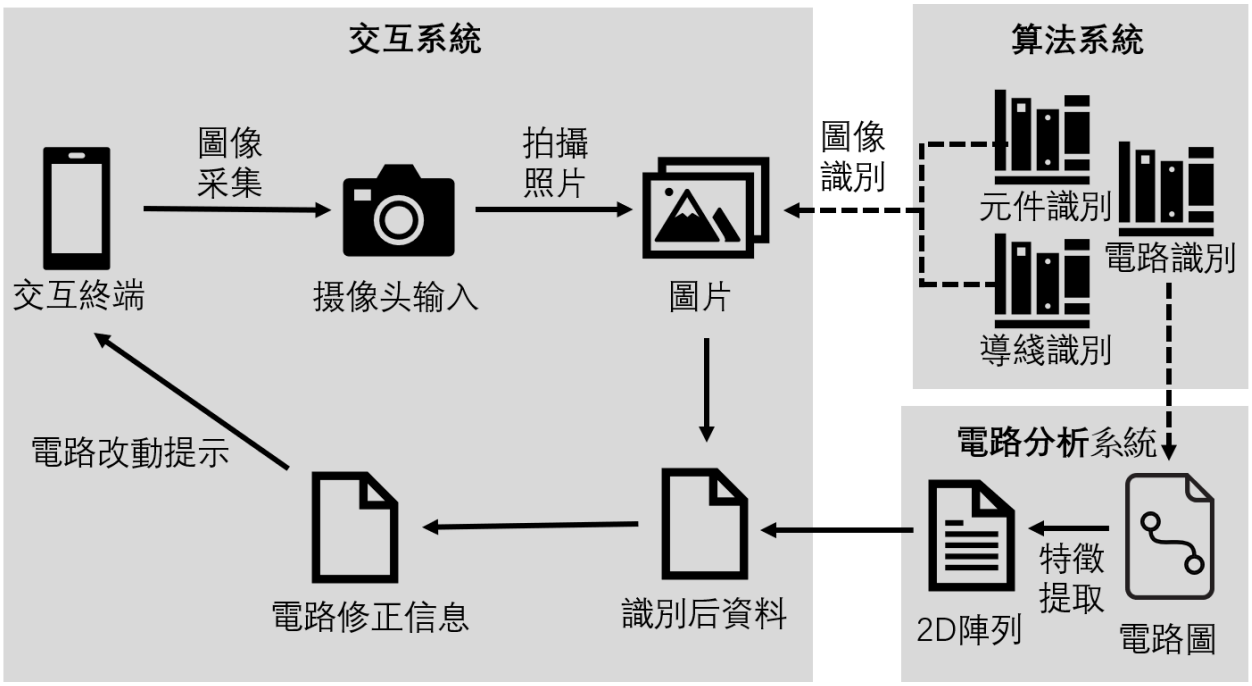

圖 1. 2 關係結構圖

第二章 相關文獻回顧

2.1 捲積神經網路

1989 年，一種新的神經網路 (Neural Networks)，稱為卷積神經網路[5] (Convolutional Neural Network) LeCun 等人[1]被報導，這在機器視覺 (Machine Vision) 相關任務中顯示了巨大的潛力。CNN 通過感受野（Receptive Field）和權值的共用減少了神經網絡需要訓練的參數個數，將複雜問題簡化。而且，在大部分情況下，降維不會影響處理結果，還可以加速訓練，防止網絡過擬合。

其中的感受野是 CNN 中每層的特徵圖（Feature Map）上的圖元點在原始圖像中映射區域的大小，也就是高層的特徵圖中的圖元點受原圖多大區域的影響。在目標檢測任務時，我們通過高層的特徵圖檢測大物體，底層的特徵圖檢測小物體。下圖中，黃色區域的感受野就是 7，綠色區域的感受野為 5。

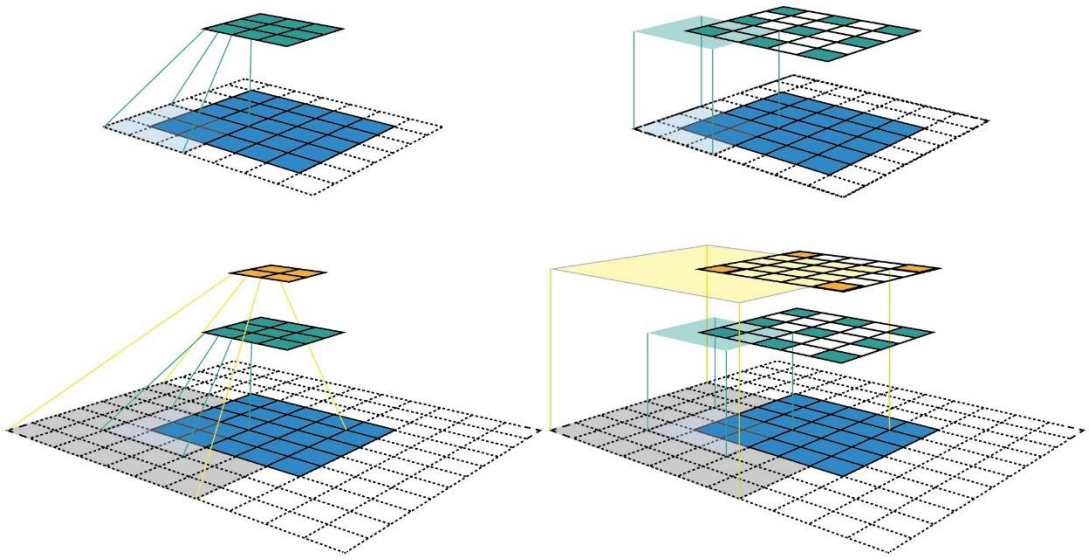

圖 2.1 感受野

本專題使用 CNN 來對圖片進行邊緣檢測以及刪除交叉點和垃圾點，以檢測出導綫位置。

2.2 邊緣檢測

邊緣檢測（英語：Edge detection）是影像處理和電腦視覺中的基本問題，邊緣檢測的目的是標識數位圖像中亮度變化明顯的點。圖像屬性中的顯著變化通常反映了屬性的重要事件和變化。這些包括（i）深度上的不連續、（ii）表面方向不連續、（iii）物質屬性變化和（iv）場景照明變化。 邊緣檢測是影像處理和電腦視覺中，尤其是特徵檢測中的一個研究領域。

邊緣檢測操作是基於亮度的二階導數。這實質上是亮度梯度的變化率。在理想的連續變化情況下，在二階導數中檢測過零點將得到梯度中的局部最大值。另一方面，二階導數中的峰值檢測是邊線檢測，只要圖像操作使用一個合適的尺度表示。如上所述，邊線是雙重邊緣，這樣我們就可以在邊線的一邊看到一個亮度梯度，而在另一邊看到相反的梯度。這樣如果圖像中有邊線出現的話我們就能在亮度梯度上看到非常大的變化。為了找到這些邊線，我們可以在圖像亮度梯度的二階導數中尋找過零點。

如果 I（x） 表示點 x 的亮度，I′ ′（x） 表示點 x 亮度的二階導數，那麼：

$$I''(x) = 1 \cdot I(x-1) - 2 \cdot I(x) + 1 \cdot I(x+1).$$

同樣，許多演算法也使用卷積遮罩快速處理圖像資料：

+1 −2 +1

2.2.1 自我調整閾值邊緣檢測

相較於使用一個全域值作為閾值，自我調整閾值可以適用於更多情況，例如，如果圖像在不同區域具有不同的照明條件。該演算法根據圖元周圍的小區域確定圖元的閾值。因此，我們為同一圖像的不同區域獲得了不同的閾值，這對於具有不同照明的圖像提供了更好的結果。其計算閾值有 2 種方法。

一是鄰域面積的平均值減去常數 C，二是鄰域值減去常數 C 的高斯加權和。通過實驗我們發現，方法二，即閾值是鄰域值減去常數 C 的高斯加權和對於陰影等雜訊剔除更好，曲綫綫段顯示更完整，所以本專題使用方法二。

2.2.2 Canny 邊緣檢測

[2]Canny 邊緣檢測運算元是澳洲電腦科學家約翰・坎尼（John F. Canny）于 1986 年開發出來的一個多級邊緣檢測演算法。方法步驟如下。

首先進行降噪。任何邊緣檢測演算法都不可能在未經處理的原始資料上很好地處理，所以第一步是對原始資料與高斯平滑範本作卷積，得到的圖像與原始圖像相比有些輕微的模糊（blurred）。這樣，單獨的一個圖元雜訊在經過高斯平滑的圖像上變得幾乎沒有影響。

然後尋找圖像中的亮度梯度。圖像中的邊緣可能會指向不同的方向，所以 Canny 演算法使用 4 個 mask 檢測水準、垂直以及對角線方向的邊緣。原始圖像與每個 mask 所作的卷積都存儲起來。對於每個點我們都標識在這個點上的最大值以及生成的邊緣的方向。這樣我們就從原始圖像生成了圖像中每個點亮度梯度圖以及亮度梯度的方向。

再其次是在圖像中跟蹤邊緣。較高的亮度梯度比較有可能是邊緣，但是沒有一個確切的值來限定多大的亮度梯度是邊緣多大又不是，所以 Canny 使用了滯後閾值。

滯後閾值需要兩個閾值——高閾值與低閾值。假設圖像中的重要邊緣都是連續的曲線，這樣我們就可以跟蹤給定曲線中模糊的部分，並且避免將沒有組成曲線的雜訊圖元當成邊緣。所以我們從一個較大的閾值開始，這將標識出我們比較確信的真實邊緣，使用前面匯出的方向資訊，我們從這些真正的邊緣開始在圖像中跟蹤整個的邊緣。在跟蹤的時候，我們使用一個較小的閾值，這樣就可以跟蹤曲線的模糊部分直到我們回到起點。

一旦這個過程完成，我們就得到了一個二值圖像，每點表示是否是一個邊緣點。

一個獲得亞圖元精度邊緣的改進實現是在梯度方向檢測二階方向導數的過零點

$$L_x^2\, L_{xx} + 2\, L_x\, L_y\, L_{xy} + L_y^2\, L_{yy} = 0$$

它在梯度方向的三階方向導數滿足符號條件

$$L_x^3\, L_{xxx} + 3\, L_x^2\, L_y\, L_{xxy} + 3\, L_x\, L_y^2\, L_{xyy} + L_y^3\, L_{yyy} < 0$$

其中 $L_x, L_y \ldots L_{yyy}$ 表示用高斯核平滑原始圖像得到的尺度空間表示 L 計算得到的偏導數。用這種方法得到的邊緣片斷是連續曲線，這樣就不需要另外的邊緣跟

蹤改進。滯後閾值也可以用於亞圖元邊緣檢測。

第三章 專題報告主體

3.1 研究方法

3.1.1 導綫識別

3.1.1.1 導綫邊緣識別

導綫識別主要通過邊緣識別的方法得以實現。第一道程式為邊緣檢測，本專題使用了 opencv 中的 2 種邊緣檢測的方法。

首先拍攝原圖並對元件進行遮擋處理。

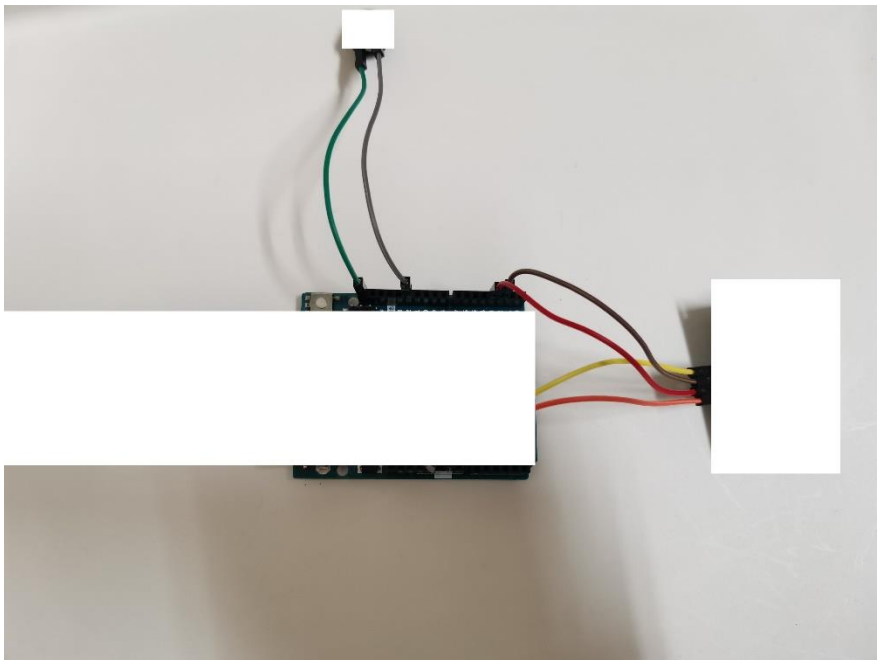

圖 3.1 原始拍攝處理後圖像

第一種為使用高斯濾波器和自我調整閾值進行圖像閾值處理，高斯濾波器相對於另一種均值濾波器，即簡單的對於圖片的一小部分取均值，可以更有效的過濾不需要的細節部分。

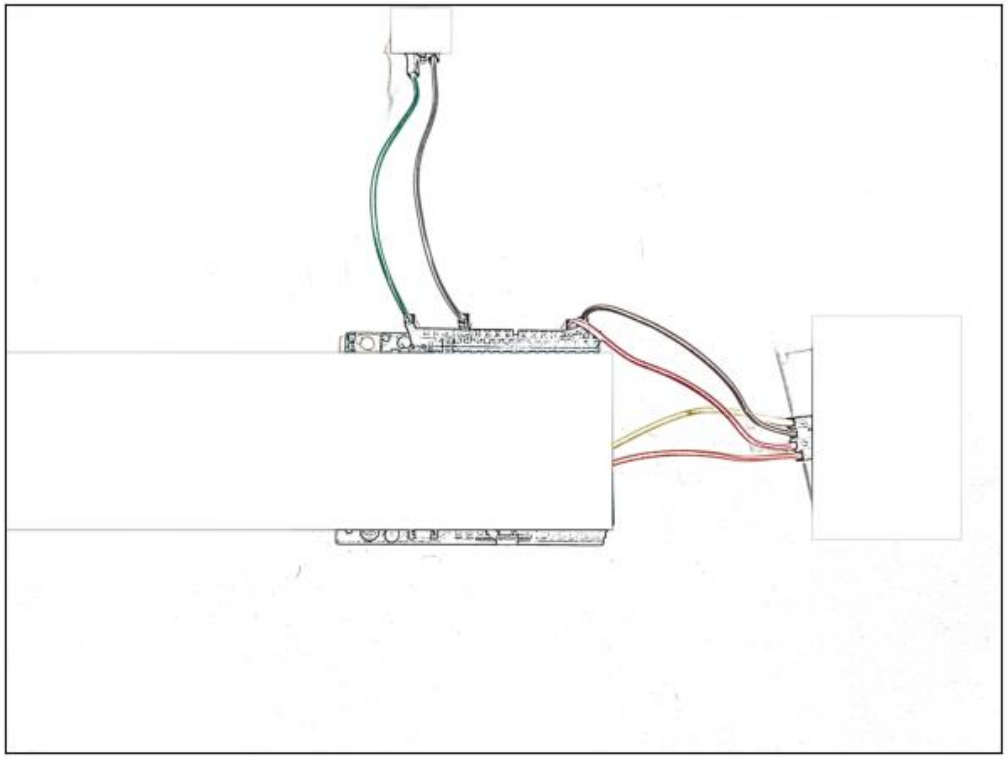

圖 3.2 相對簡單的均值濾波器

而高斯濾波可以對整幅影像進行加權平均的過程，每一個畫素點的值，都由其本身和鄰域內的其他畫素值經過加權平均後得到。高斯濾波的具體操作是：用一個範本（或稱卷積、掩模）掃描影像中的每一個畫素，用範本確定的鄰域內畫素的加權平均灰度值去替代範本中心畫素點的值。

二維高斯分佈：

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

先將圖片轉為灰階，再將將圖片做模糊化，以此降噪減少雜訊，再將處理完的圖像進行一般圖自適應高斯二值化(提前模糊降噪)，得到一個值為 0-255 的矩陣，再將矩陣反轉即為所需邊緣矩陣，填入白色背景即為其圖像。

```
[[255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 ...
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [  0 255 255 ... 255 255 255]]
```

圖 3.3 邊緣矩陣

最終得到使用高斯濾波器的自適應閾值邊緣檢測後圖像

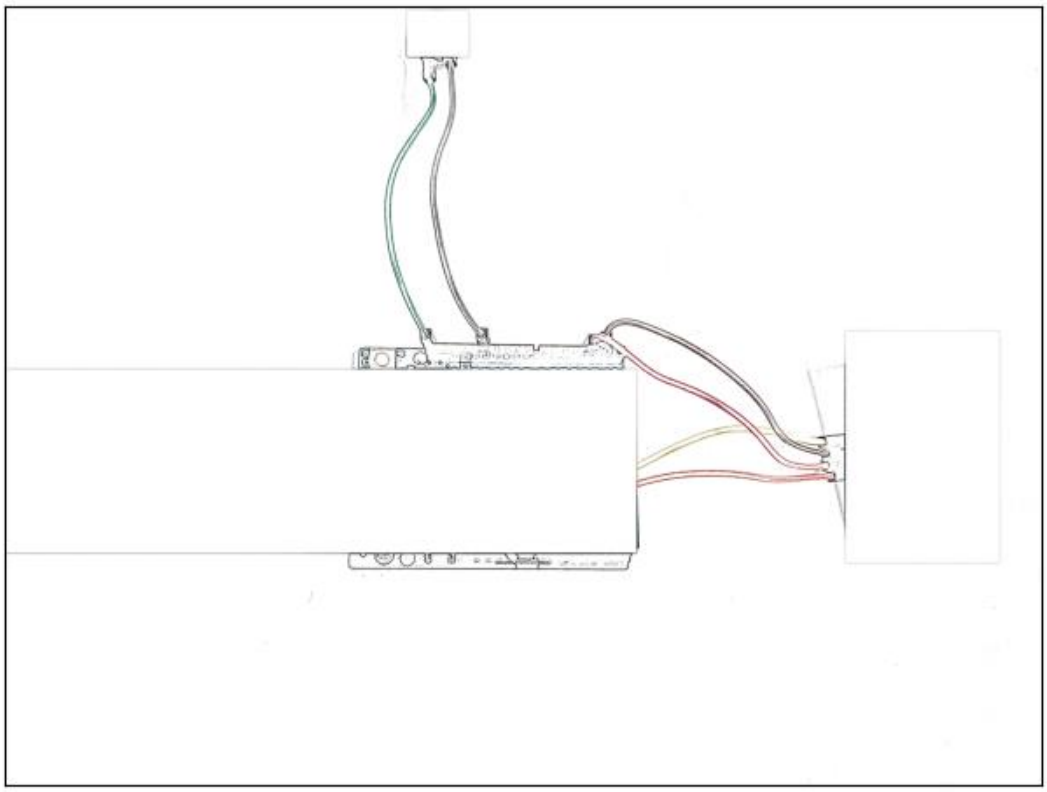

圖 3.4 自適應閾值邊緣檢測後圖像

第二種為 canny 邊緣檢測演算法，首先獲取灰度圖像，再處理 GaussianBlur，使用 Canny 邊緣檢測並反轉即得到邊緣矩陣。

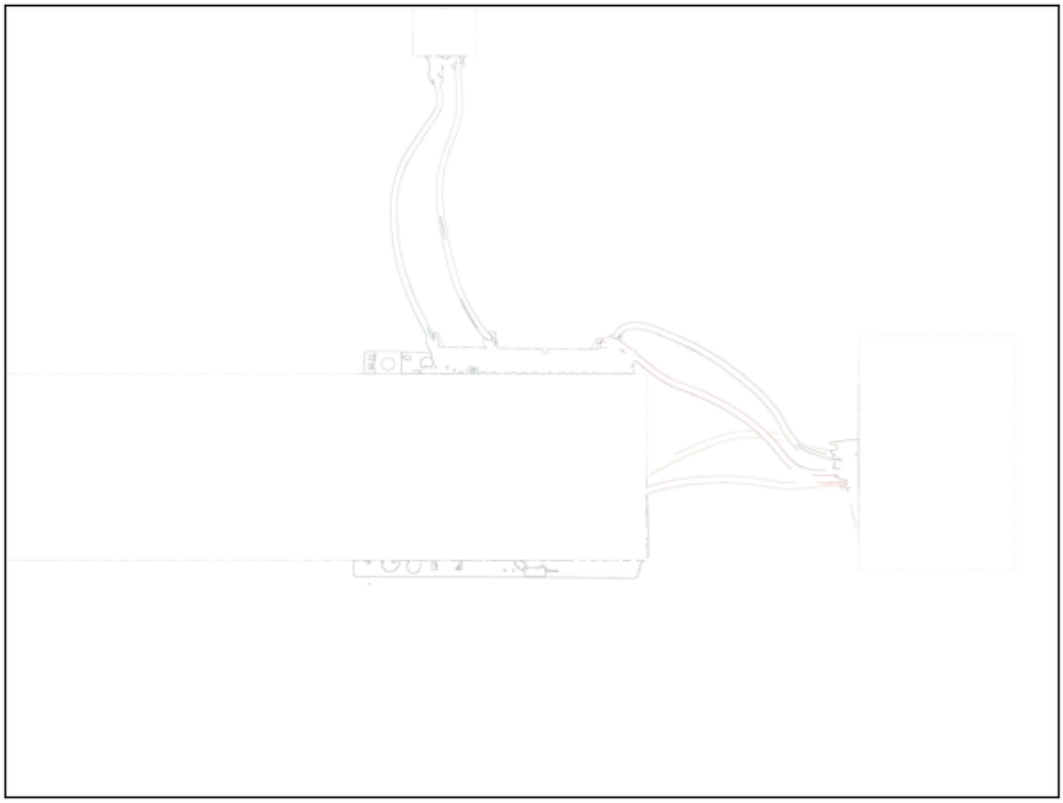

圖 3. 5 canny 邊緣檢測

取得邊緣檢測矩陣後，我們需要對其進行向量化，找出其中的邊緣綫條並繪製出來。對於其中的邊緣點，我們首先會計算其與其他點的距離並記錄下來，我們會根據一定的閾值找出目標點附近的所有點，這些點即為其相鄰點，當在一定閾值內再也找不到任何相鄰點時，一條邊緣曲綫的點就被遍歷完成了。通過遍歷整個矩陣即可檢測出照片中所有的曲綫。

然後對曲綫進行加工，刪除綫之間的交叉點，防止不同的綫之間的合併，去除環路，只偵測曲綫去除封閉圖形，刪除過短的綫條和過粗和纏繞的綫條，經過篩選，最終可得到偵測出來的導綫。

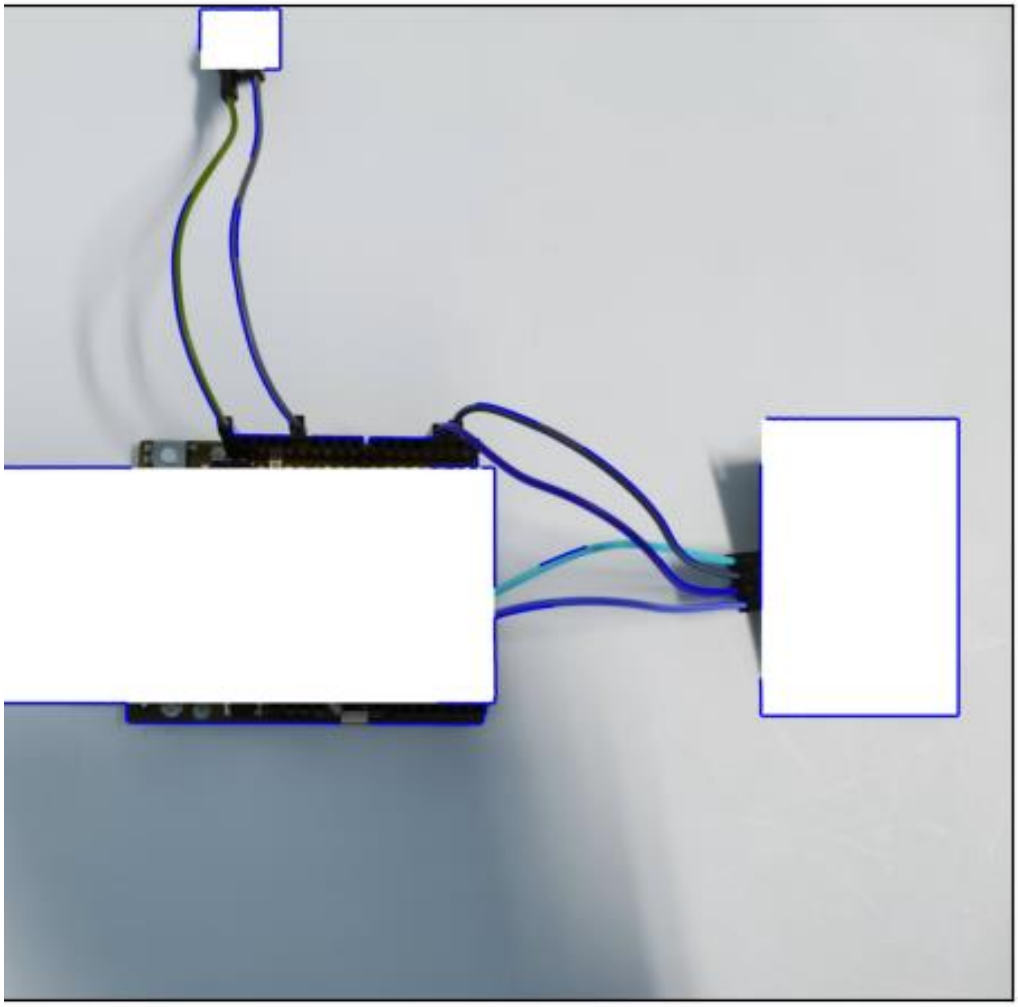

圖 3.6 自適應閾值導綫偵測

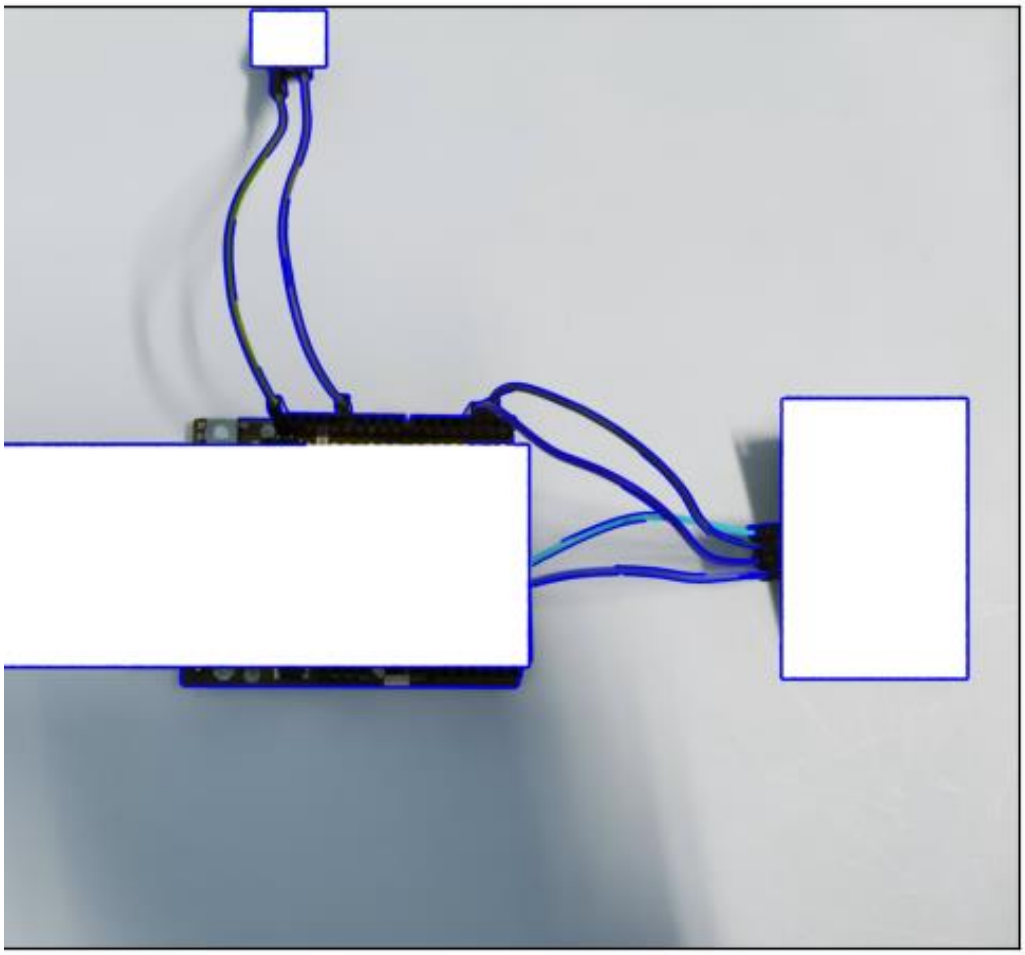

圖 3. 7 canny 導綫偵測

3.1.1.2 導綫端點識別

在識別導綫之後，需要對導綫端點進行位置判斷，從而找出導綫的介面是否正確。

我們首先需要將照片縮小灰化，然後將之骨架化，使用捲積矩陣

[[1, 1, 1],

[1, 10, 1],

[1, 1, 1]]

對圖像設置閾值，白色圖元閾值為 0，黑色圖元閾值為 10，將圖像進行捲積。這樣找出的是黑白色的邊緣綫條和空白背景，當一個黑色邊緣點的反覆運算方向不再有更多黑色邊緣點時，這個邊緣點即爲曲綫端點。由於這樣的邊緣點可能會重複，我們將彼此比較接近的曲綫端點進行合併。迴圈遍歷端點坐標，對於每個端點，產生一個標籤然後建立端點坐標的字典，存儲標籤和共用該標籤的所有質心 - 使用 groupsMask 用於在不同點之間通過 傳播標籤 flood-filling。在內部，dictionary 我們將存儲質心簇標籤、質心總和的累積以及累積多少質心的計數，因此我們可以生成最終平均值。遍歷整個字典中的值，值大於 1 即為最終曲綫端點。

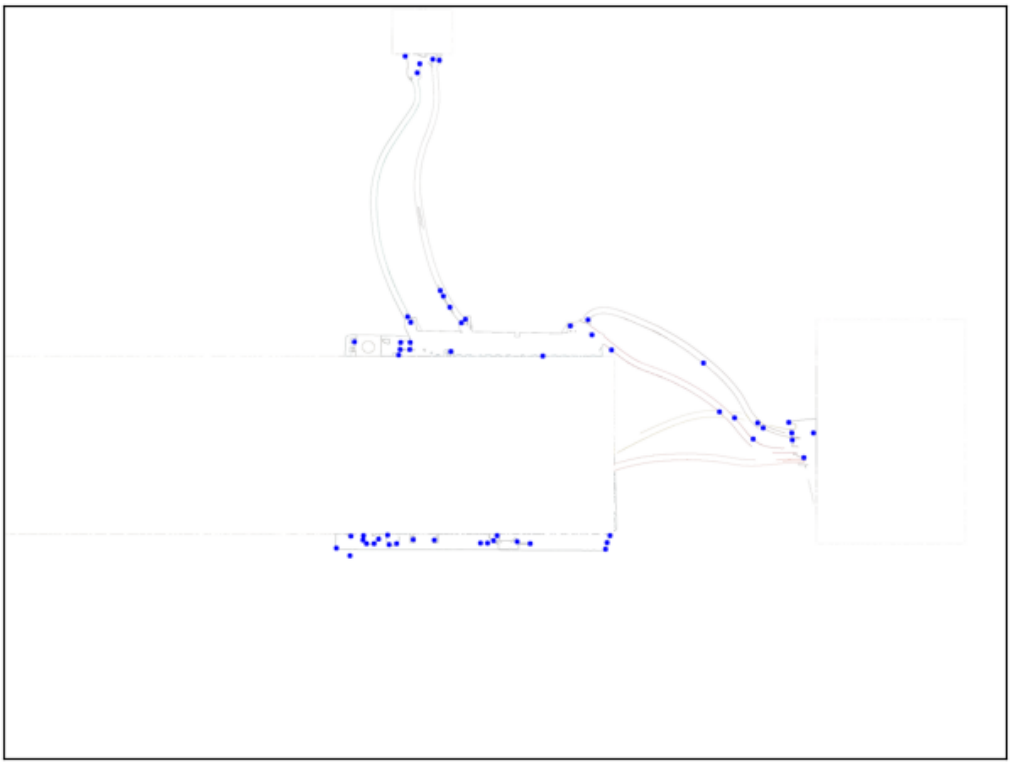

圖 3.8 曲綫端點

主要端點檢測演算法如下：

算法 5：endpoint 端點算法

功能：找出邊緣檢測圖像的邊緣端點位置。
概述：
Img: 邊緣檢測后的 RGB 圖像陣列
thinning(1): 將圖像中的綫條取粗度為 1 的中心綫骨架化
filter2D(binaryImage, -1, h): 對圖像做捲積，捲積核為 h
where(條件)：輸出滿足條件的點坐標

```
endpoint(img):
    width = int(img.shape[1] * 0.5)
    height = int(img.shape[0] * 0.5)
    resizedimage = img.resize(img, width, height)        // 算法 2：resize 改變圖像大小算法
    grayscaleimage = resizeimage.togray()
    skeleton = grayscaleimage.thinning(1)
    binaryimage = threshold(skeleton, 128, 10)     // 算法 3：threshold 圖片二值化算法
    h = array([[1, 1, 1],
               [1, 10, 1],
               [1, 1, 1]])
    imgfiltered = filter2D(binaryImage, -1, h)        // 以 h 為捲積核對圖像進行捲積
    endpointsmask = where(imgFiltered == 110, 255, 0)     // 得到端點坐標，捲積核與綫段端點的值為 x = 110。
    rgbmask = cvtColor(rgbMask, COLOR_GRAY2BGR) // 將灰色圖像轉化成 RGB 彩色
    maxkernel = getStructuringElement(MORPH_RECT, (3, 3))     // 生成形狀為矩形的內核
    groupsmask = morphologyEx(groupsMask, MORPH_DILATE, maxKernel)   // 將端點坐標膨脹，當點相互接近時，其會膨脹成一個區域，消除過於接近的點
    finalpoints = centroidsDictionary(groupmask, endpointsmask)    //    算    法    4    ：centroidsdictionary 質心算法
    return finalpoint
```

端點像素的值的算法：
捲積核與綫段端點的對應像素值為 x=110，根據捲積核對應像素的值公式

$$\mathrm{dst}(x,y) = \sum_{\substack{0 \le x' < \mathrm{kernel.cols} \\ 0 \le y' < \mathrm{kernel.rows}}} \mathrm{kernel}(x',y') \times \mathrm{src}(x + x' - \mathrm{anchor.x}, y + y' - \mathrm{anchor.y})$$

可得，其端點像素的值為 100+10。100 為錨點與捲積核中心點乘積 10*10，因爲對邊緣進行骨架化處理，邊緣寬度為 1，所以每個端點相鄰點只有一個為 10，其餘均為 0。

圖 3.9 endpoint 端點演算法

算法 2：resize 改變圖像大小算法

功能：改變圖像大小，保持信息不丟失

```
Img：輸入圖像
shape[0]：  讀取 img 長度
shape[1]：  讀取 img 寬度
resize(img, width, height):
    dst -> zeros(width, height, 3)    //三維 0 矩陣
    for c in range(3):
        for i in range(width):
            for j in range(height):
                x = (img.shape[0]/width)*i   /
                y = (img.shape[1]/height)*j
                dst[i,j,c] = img[round(x),round(y),c]
    return img
```

圖 3.10 resize 改變圖像大小演算法

算法 3：threshold 圖片二值化算法

功能：圖片二值化

概述：這是簡要說明中心思想的算法，實際還會考慮到捲積核，但核心思想是將所有小於 b1 的像素值設為 0，大於 b1 的像素值設爲 b2.

```
threshold (skeleton, b1, b2):
    for i in skeleton:
        if i < b1:
            i = 0
        if i > b1:
            i = b2
    return skeleton
```

圖 3.11 threshold 圖像二值化演算法

算法 4：centroidsdictionary 質心算法

功能：計算膨脹區域的質心，使用字典 centroidsDictionary 儲存標簽和使用該標簽的所有質心

概述：把原來 endpointsmask 中的端點代入膨脹端點后的圖中，將同一膨脹區域的端點做算數平均值，得到質心坐標

```
centroidsdictionary(groupsMask, endpointsmask):
    新建字典 centroidsDictionary
    使用連通器 Centroids 找到所有 endpointsmask 中點的位置信息
    for pixelValue in Centroids :
        pixelValue = groupsMask[cy, cx]      // 將 endpointsmask 對應到 groupsMask
        如果 pixelValue 不存在字典，加入字典
        否則，將 pixelValue 與 groupsMask 坐標相加，總共加了 i 次
    for x in centroidsDictionary:
        centroidsDictionary 中坐標值 / i
    return centroidsDictionary 中所有坐標點
```

圖 3.12 centroidsdictionary 質心演算法

使用 canny 邊緣檢測后的圖像，通過 endpoint 端點演算法，最終得到初步的端點坐標。

3.1.1.3 導綫端點定位和糾錯

通過算法對上面產生的初步端點坐標進行定位，可以得到精確的端點坐標。

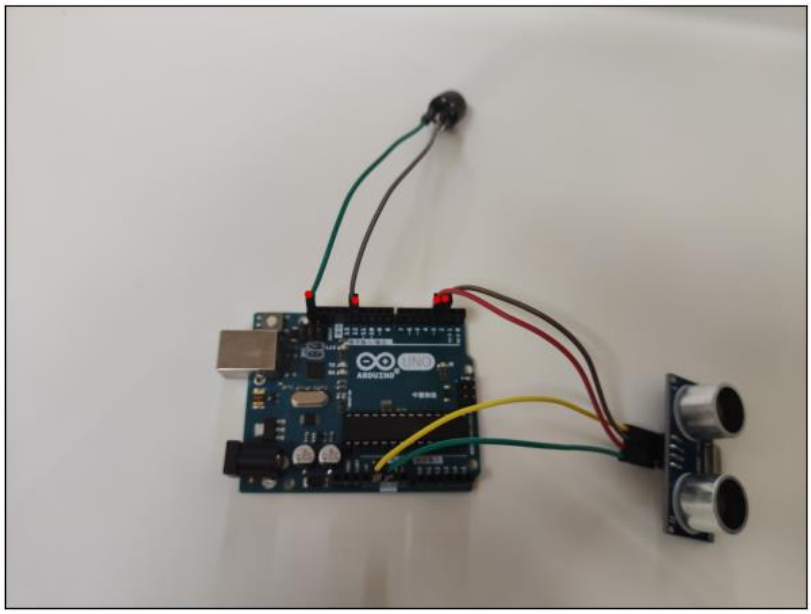

圖 3.13 精確端點坐標

演算法如下：

算法 7：point_detect 針脚定位算法

```
功能：將簡化后的偵測到的端點位置轉化成針脚位置
輸入：
points：18 針脚一側端點位置，由算法 6：端點簡化算法得到，以下均同上
points_2:：14 針脚一側端點位置
xy: 18 針脚一側針脚位置坐標
xy2：14 針脚一側針脚位置坐標
point_detect(points, points_2 ,xy ,xy2)
    final_point = []     // 針脚附件點坐標
    final_point_2 = []
    sit = []   // 偵測出的針脚編號，其對應導綫接口
    sit_2 = []
    y = (xy[0][1] + xy[1][1]) / 2   // 18 針針脚所在的縱坐標
    y_2 = (xy2[0][1] + xy2[1][1]) / 2   // 14 針針脚所在的縱坐標
    x = ((xy[1][0] - xy[0][0]) / 18 + (xy[2][0] - xy[0][0]) / 10) / 2    // 18 針針脚所在的單個針脚寬度
    x = ((xy2[1][0] - xy2[0][0]) / 14 + (xy2[2][0] - xy2[0][0]) / 6) / 2     // 14 針針脚所在的單個針脚寬度
    for i in len(points):       // 設定 18 針脚處端點取值範圍
        if xy[1][0] > points[i][0] > xy[0][0]-10 and y - 185 < points[i][1] < y:
        final_point.append(points[i])
    14 針脚處同理，得到 final_point_2
    for i in len(final_point):
        temp = final_point5[i][0] - xy[0][0]
        k = 0.06 * (final_point5[i][0]-img.shape[0]/2    )     // 偏離中心點修正參數，也就是斜率，立體圖形偏離中心點均會傾斜，需要修正
            temp = temp + k*( final_point5[i][1]-img.shape[1]/2)*0.023
        if temp > 10 * x:    # 針脚間中间空出的小格
            temp = temp - 0.5 * x
        sit.append(temp // x)       # 算出是第几个
    print('18 針一側的導綫接口為：')
    sit = list(set(sit))          // 消除相同項
    print(sit)
    14 針脚計算方法類似，得到 sit_2
    sit_fin = []     // 最終的完整導綫接口
    for i in sit:
        sit_fin.append(i)
    for i in sit_2:
        sit_fin.append(i + 18)
    return sit_fin
```

圖 3.14 point_detect 針脚定位演算法

這個算法通過對於圖像特定區域的端點選擇，以及對端點位置的坐標偏移矯正（因爲立體圖形偏移中心點直綫會有傾斜角度），定位端點屬於哪一個 arduino 接脚針孔，從而判定導綫接口位置。

此演算法的輸入，points：簡化后的端點位置，由前面提到的一系列演算法得到。包括自適應邊緣檢測算法，canny 邊緣檢測算法，環路交叉點垃圾綫清除算法，Ramer-Douglas-Pecker 綫條簡化算法，直綫過濾器算法等。其輸入為 finalpoint：由算法 5：endpoint 端點算法得到，輸出即為 points。

而簡化后即可得到完整端點坐標，在這個例子裏是：

[0.0, 5.0, 14.0, 15.0, 26.0, 27.0]

通過偵錯演算法，可以把得到的完整端點坐標和下文的電路圖資料庫內的專屬 2D 陣列表進行比對，得到最終糾錯結果。

算法 1：偵錯算法

```
功能：根據生成修正建議和修正建議圖
輸入：
final_ans: 完整導綫接口坐標，由 point_detect 針脚定位算法得到
data：接綫圖 2D 陣列字典
xy：定位的 arduino 板接脚位置坐標

point2final(final_ans, data, xy):
    for datas in data:
        if datas = 'arduino':
            arduino_p = data[datas]
        else:
                element_p.append(line)      //將其餘的字典索引值即接脚名稱存入數列
                                            element_p
            for i in len(data[line]):
                if data[line][i]:
                    excel_dict[line] = i          //將值為 1 的數列即與導綫相連的接脚的
                                            編號 i 存入字典 excel_dict，key 為 line 也就是
                                            接脚名稱字典的這是存正確接綫的字典
    correct_p = execl_dict        // 生成正確接脚字典
    for excel_dicts in excel_dict:
        for i in len(excel_dict)
            如果 final_ans 中有值和字典 excel_dict 中的值不一樣
                輸出接口錯誤提示
                將錯誤接脚移入錯誤接脚字典 worry_p
                將錯誤接脚移出正確接脚字典 correct_p
    輸出所有正確接脚的信息
    輸出"你需要將（所有錯誤接脚）移動到（所有正確接脚）"的糾錯提示
```

圖 3. 15 偵錯演算法

3.1.2 電路圖資料庫

使用者可以選擇我們提供的的不同 arduino 接綫圖自我練習，當他們選定想要練習的接綫圖后，算法根據不同的 arduino 接綫圖，在資料庫裏尋找對應的 2D 陣列資料表，再將其和之前照片識別出的完整導綫接口坐標一起，算出錯誤的接綫和糾正方法。

address	data
arduino_cir\DS1302_LCD.png	DS1302_LCD_2d.csv
arduino_cir\heart_rate.png	leart_rate_2d.csv
arduino_cir\LED.png	LED_2d.csv
arduino_cir\MPU6050.png	MPU6050_2d.csv
arduino_cir\rfid.png	RFID_2d.csv
arduino_cir\softkeyboard.png	softkeyboard_2d.csv
arduino_cir\ultrasound.png	ultrasound_2d.csv
arduino_cir\ultrasound2.png	ultrasound2_2d.csv
arduino_cir\wet.png	wet_2d.csv
arduino_cir\wet2.png	wet2_2d.csv
arduino_cir\wifi.png	wifi_2d.csv
arduino_cir\wifi2.png	wifi2_2d.csv
arduino_cir\bluetooth.png	bluetooth_2d.csv
arduino_cir\rfid_display.png	rfid_display_2d.csv

圖 3. 16 接綫圖資料庫

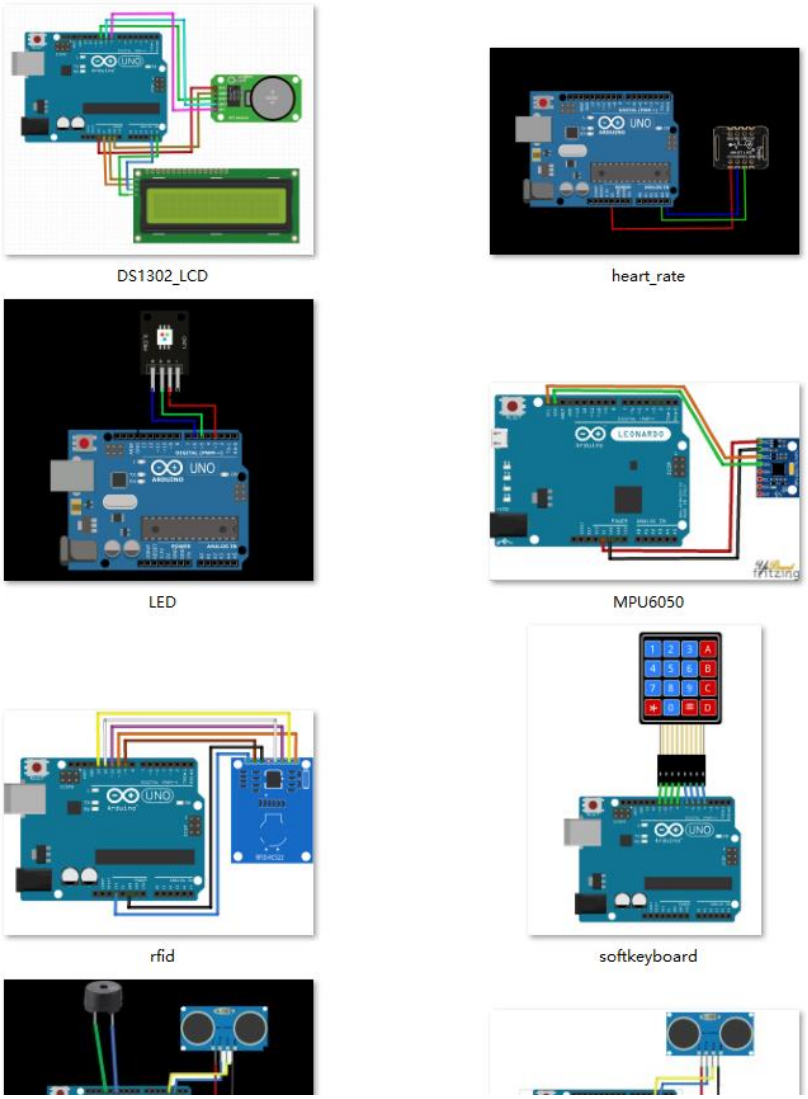

圖 3. 17 各種不同的 arduino 練習圖

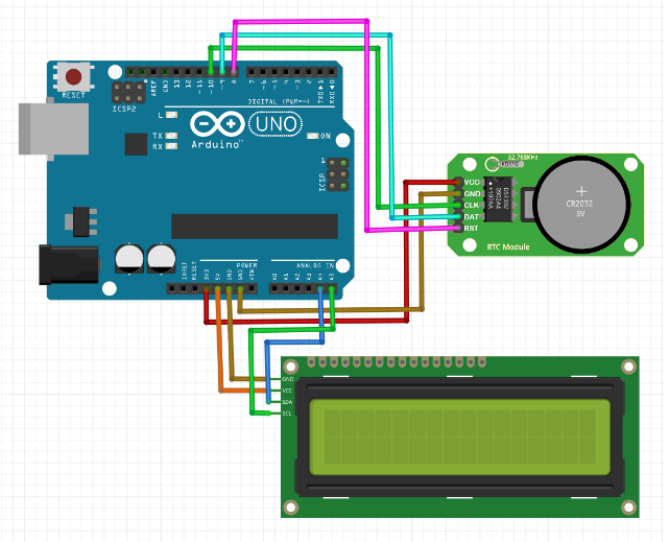

圖 3. 18 DS1302_LCD，一個 LCD 時鐘的 arduino 練習圖

以下是這個練習圖對應的 2D 陣列圖。

arduino	VCC(time)	GND(time)	CLK(time)	DAT(time)	RST(time)	GND(LCD)	VCC(LCD)	SDA(LCD)	SCL(LCD)
arduino_r1 (SCL)	0	0	0	0	0	0	0	0	0
arduino_r2 (SDA)	0	0	0	0	0	0	0	0	0
arduino_r3 (AREF)	0	0	0	0	0	0	0	0	0
arduino_r4 (GND)	0	0	0	0	0	0	0	0	0
arduino_r5 (13)	0	0	0	0	0	0	0	0	0
arduino_r6 (12)	0	0	0	0	0	0	0	0	0
arduino_r7 (11)	0	0	0	0	0	0	0	0	0
arduino_r8 (10)	0	0	1	0	0	0	0	0	0
arduino_r9 (9)	0	0	0	1	0	0	0	0	0
arduino_r10 (8)	0	0	0	0	1	0	0	0	0
arduino_r11 (7)	0	0	0	0	0	0	0	0	0
arduino_r12 (6)	0	0	0	0	0	0	0	0	0
arduino_r13 (5)	0	0	0	0	0	0	0	0	0
arduino_r14 (4)	0	0	0	0	0	0	0	0	0
arduino_r15 (3)	0	0	0	0	0	0	0	0	0
arduino_r16 (2)	0	0	0	0	0	0	0	0	0
arduino_r17 (1<-TX)	0	0	0	0	0	0	0	0	0
arduino_r18 (0->RX)	0	0	0	0	0	0	0	0	0
arduino_l1 (A5)	0	0	0	0	0	0	0	0	1
arduino_l2 (A4)	0	0	0	0	0	0	0	1	0
arduino_l3 (A3)	0	0	0	0	0	0	0	0	0
arduino_l4 (A2)	0	0	0	0	0	0	0	0	0
arduino_l5 (A1)	0	0	0	0	0	0	0	0	0
arduino_l6 (A0)	0	0	0	0	0	0	0	0	0
arduino_l7 (vin)	0	0	0	0	0	0	0	0	0
arduino_l8 (GND)	0	1	0	0	0	0	0	0	0
arduino_l9 (GND)	0	0	0	0	0	1	0	0	0
arduino_l10 (5V)	0	0	0	0	0	0	1	0	0
arduino_l11 (3.3V)	1	0	0	0	0	0	0	0	0
arduino_l12 (RESET)	0	0	0	0	0	0	0	0	0
arduino_l13 (IOREF)	0	0	0	0	0	0	0	0	0
arduino_l14 (NONE)	0	0	0	0	0	0	0	0	0

圖 3. 19 2D 陣列 DS1302_LCD_2d.csv

3.1.3 電子元件識別

這裏使用 python 和 tensorflow[4]捲積神經網路來訓練電子原件的識別。因爲我們需要實時識別電路，並且是應用在 Android 可移動設備上的應用，所以我們使用輕量級的預訓練神經網絡模型(pre-trained-models)，TensorFlow 2 Detection Model Zoo 上的 ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8[4]。這個模型的好處是反應速度快，Speed (ms)=19，COCO mAP= 20.2，很好的兼顧了速度和準確性兩方面需求。

我們先是配置 tensorflow 訓練環境，用 Anaconda 安裝 Python3.7.3，然後再安裝 Microsoft Visual Studio，C++ Build Tools，CUDA 10.1，CUDNN 7.6.5 for NVIDIA，Protocol Buffers v3.14.0 和 Model Garden for TensorFlow。然後，我們根據上面的 mobilenet 預訓練模型重訓練出可以偵測出 LED 燈和導綫的模型檔。我們選取並且處理了超過 500 張 320*320 的圖片來訓練模型，並用 anaconda 環境下的 jupyter notebook 來編譯運行 python 訓練代碼。

最後，模型使用一些 gzip 命令進行壓縮，使其適合包含在具有小存儲要求的 android 應用程式中。因此，此過程的最終結果生成了一個模型檔，該檔可用於對通過設備相機拍攝的電子元件圖像進行分類。

使用 python 和 tensorflow 捲積神經網路來訓練電子原件的識別。預訓練模型為 ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8。

```
Instructions for updating:
Use fn_output_signature instead
INFO:tensorflow:Step 100 per-step time 1.293s
I1122 14:07:22.741656 14032 model_lib_v2.py:700] Step 100 per-step time 1.293s
INFO:tensorflow:{'Loss/classification_loss': 1.1358845,
 'Loss/localization_loss': 0.4963225,
 'Loss/regularization_loss': 0.15589651,
 'Loss/total_loss': 1.7881036,
 'learning_rate': 0.0319994}
I1122 14:07:22.741656 14032 model_lib_v2.py:701] {'Loss/classification_loss': 1.1358845,
 'Loss/localization_loss': 0.4963225,
 'Loss/regularization_loss': 0.15589651,
 'Loss/total_loss': 1.7881036,
 'learning_rate': 0.0319994}
```

圖 3.20 初次訓練準確率

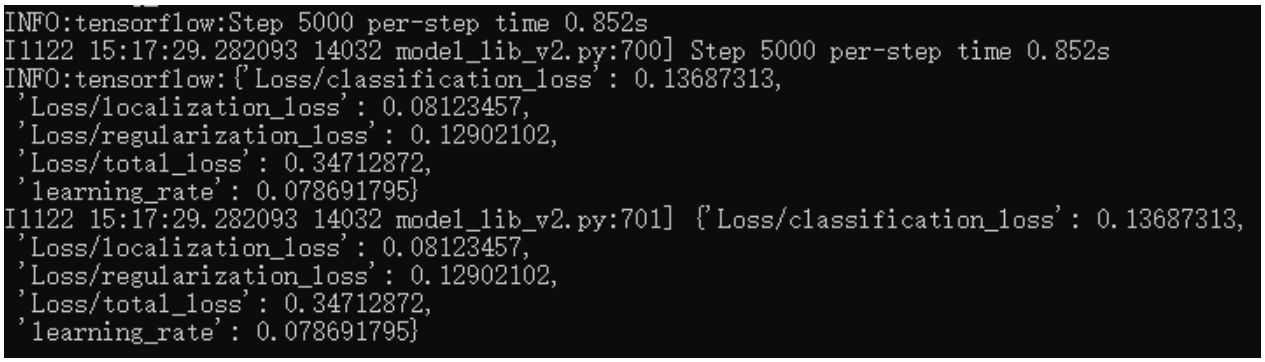

圖 3. 21 5000 次訓練後準確率

目前針對 LED 燈進行訓練，在使用 200 張照片，訓練訓練步驟 5000 次後，精確度可以達到 92%。

第四章 實證分析

4.1 模型評估

本專題使用多種 arduino 接綫圖的正確與錯誤接法進行多次實測，以證明本方法的準確性。由於拍照環境光，角度等變量的不同，會對圖像有不同的處理方法。

由於缺乏專業的采光設備，大部分使用者的拍攝條件是暗光條件，所以暗光條件下的高角度拍攝和低角度拍攝會是軟體面臨的主要使用環境，也是我們需要解決問題的地方。

4.1.1 對於暗光的處理

由於暗光下拍攝出的照片細節上比較模糊，并且在底面會有陰影，這會對偵測造成挑戰。

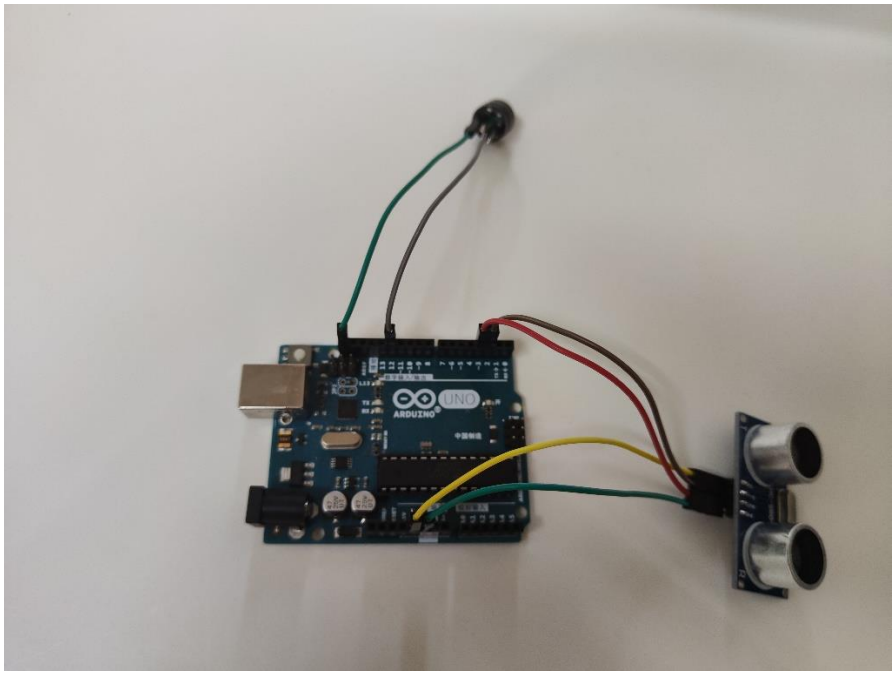

圖 4. 0.1 暗光下的 arduino 電路板接綫圖

爲此我們使用 canny 邊緣檢測的方法偵測綫條，消除無用信息。對於綫條進行去除環路和去除垃圾綫的初步處理簡化綫條。

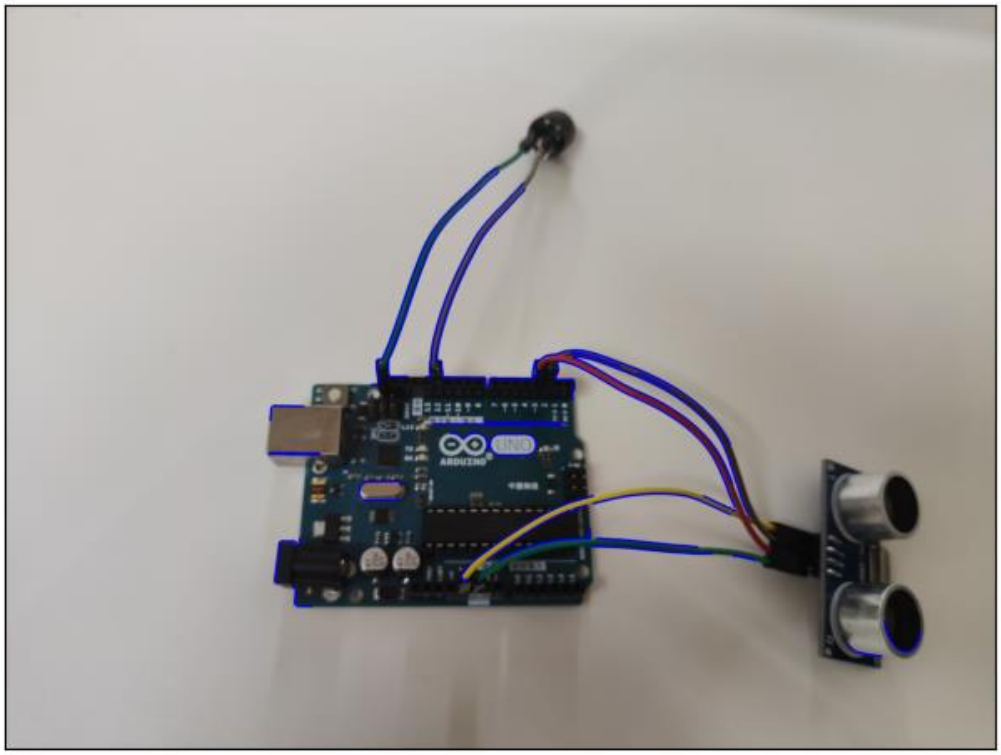

圖 4. 0.2 暗光下 canny 邊緣檢測出的綫條（藍綫）

爲了找出其中的導綫，我們需要使用直綫過濾器去除其中的直綫。

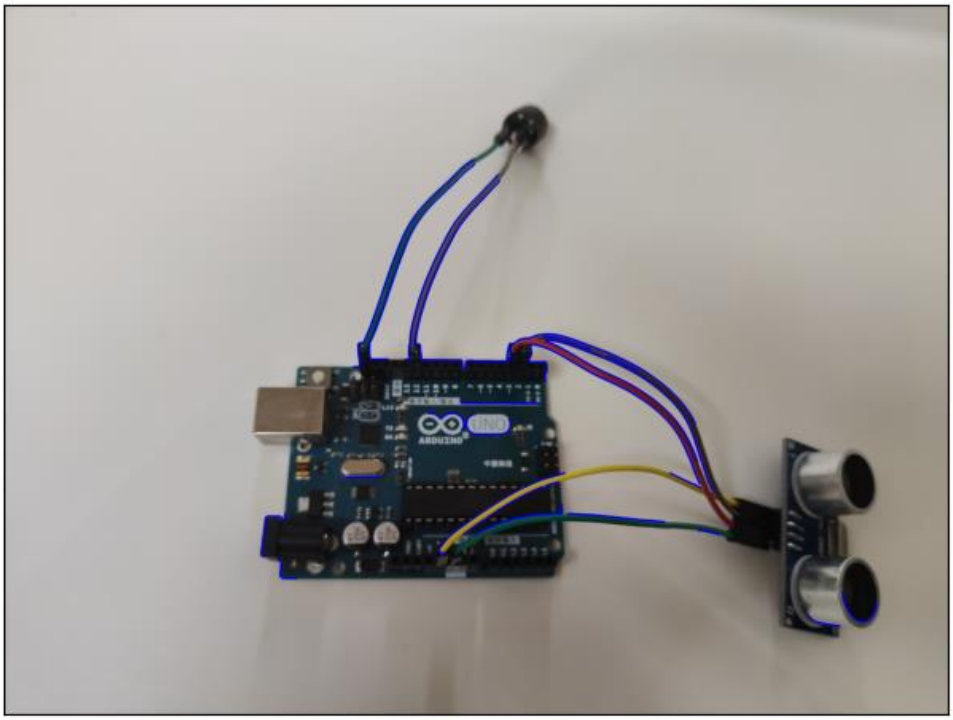

圖 4. 0.3 暗光下去除直綫后的綫條（藍綫）

然後我們需要使用端點偵測的方法，找出綫條的兩端，這就是導綫的兩個接口，

也是其連接元件的地方。

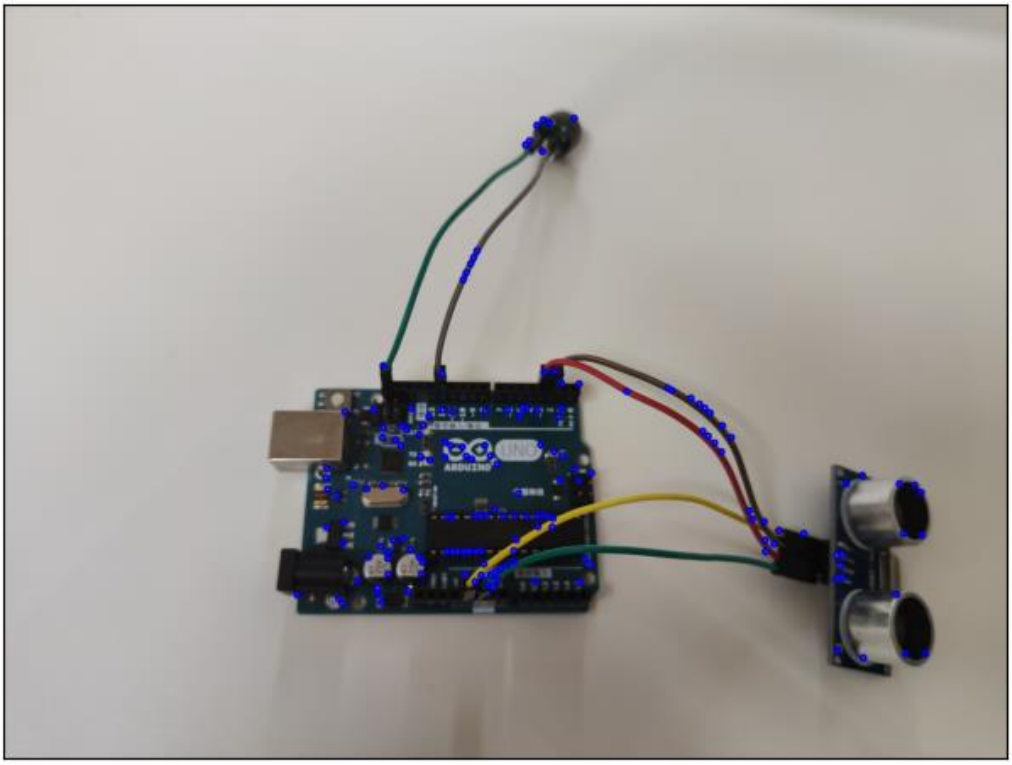

圖 4. 0.4 暗光下導綫的端點（藍色）

Canny 邊緣檢測雖然能找到幾乎所有的導綫，但是也會帶入大量無關的綫條，經過之前步驟去除大部。

因爲值滤波器和自适应阈值方法產生的邊緣檢測綫條雖然無法檢測出所有的導綫，并且不完整，但是不會有那麼多我們不需要的綫條。所以餘下部分我們會結合中值滤波器和自适应阈值方法產生的邊緣檢測綫條兩方結合對比得出更精確的綫條，再通過對於插口位置的設定進一步排除多餘綫條。

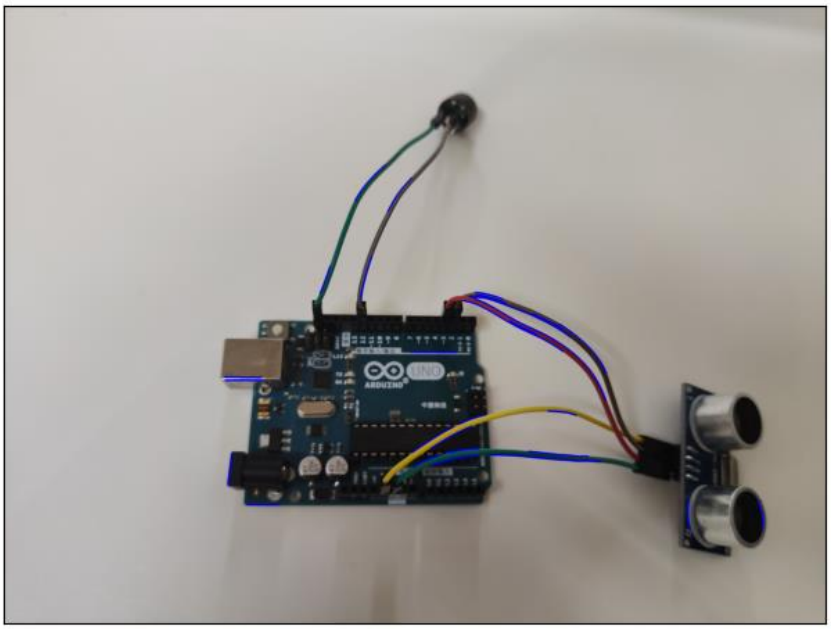

圖 4. 0.5 暗光下自适应阈值邊緣檢測的綫條

4.1.2 高拍攝角度下的模型

在暗光環境下完成邊緣檢測后，我們可以計算出 arduino 板和導綫的接綫位置。在高角度拍攝情況下，圖片立體度會下降，我們需要解決元件重叠以及接綫位置不明顯的問題。

首先，由於 arduino 板内元件對圖像的干擾，我們先解決 arduino 板 18 針接脚側的導綫與 arduino 接口一側的偵測。

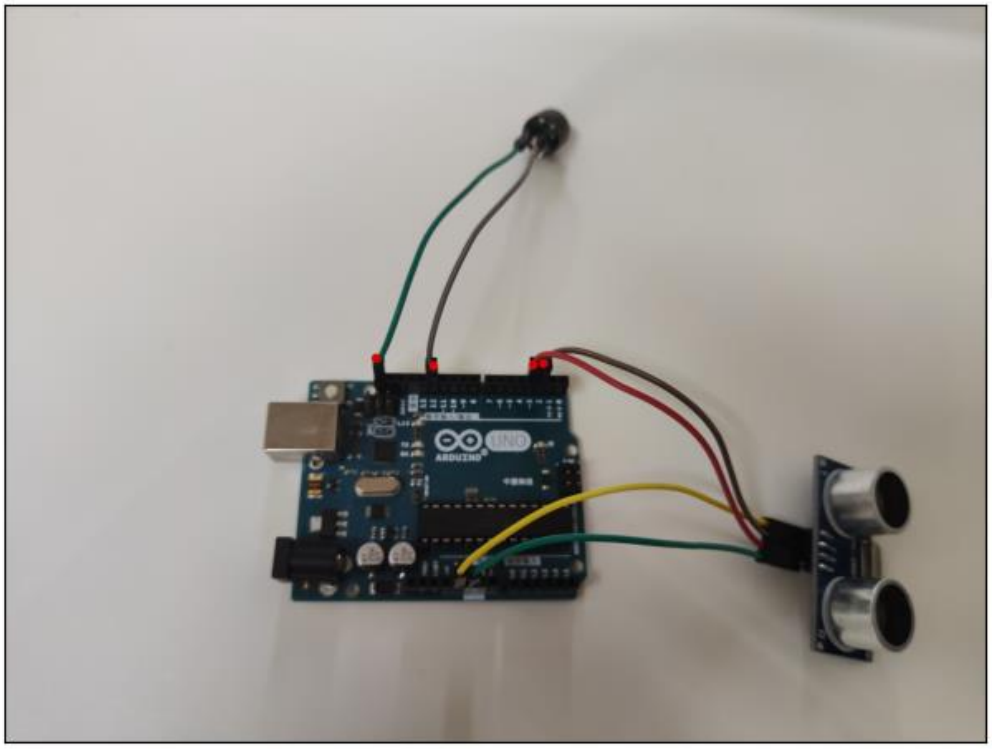

圖 4.0.6 高拍攝角度下 18 針接脚側端點位置（紅點）

接脚上的四個導綫與 arduino 板接口的位置如下，方向為從左往右。

```
[0.0, 5.0, 14.0, 15.0]
```

圖 4.0.7 端點位置（從左往右，開始為 0）

下面我們要解決 arduino 板 14 針接脚方向的兩個接口的偵測，我們需要另一個角度的拍攝，即將電路板旋轉 180 度。

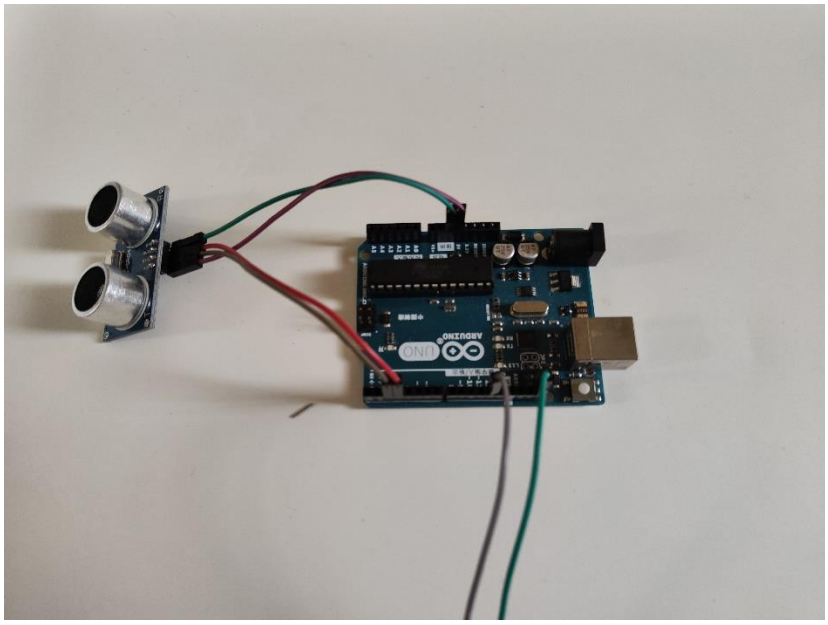

圖 4. 0.8 旋轉 180 度高拍攝角度下 arduino 電路板接綫圖

經過處理后可以得到定位后的端點位置。

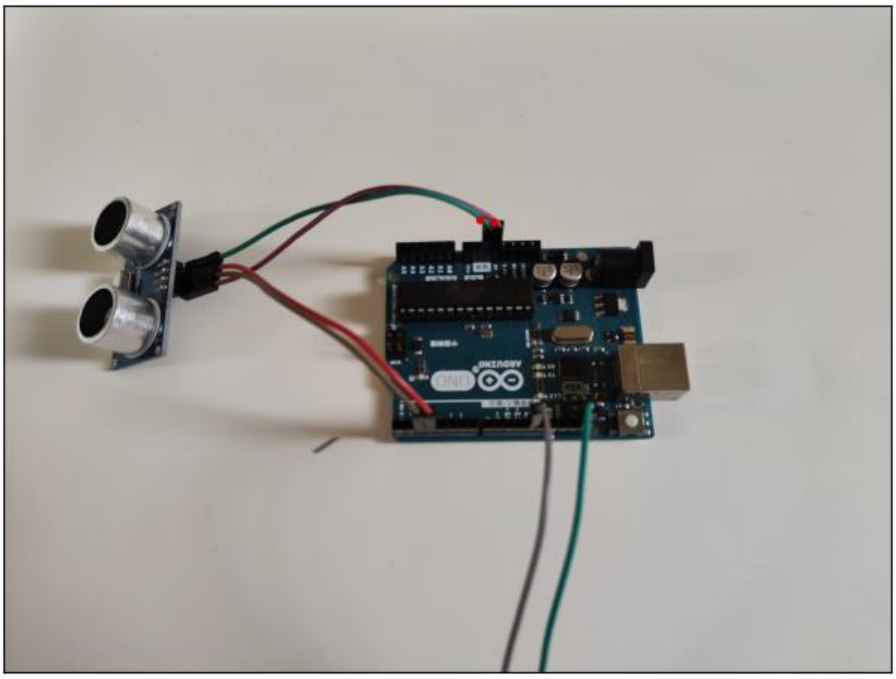

圖 4. 0.9 高拍攝角度下 14 針接脚側端點位置（紅點）

從中可以分析出端點的位置，即為導綫與 arduino 板接綫的孔位。

```
[8.0, 9.0]
```

圖 4. 0.10 端點位置（從左往右，開始為 0）

對兩者進行整合，得到完整的端點位置信息。

```
[0.0, 5.0, 14.0, 15.0, 26.0, 27.0]
```

圖 4.0.11 完整端點位置（從左往右，開始為 0）

4.1.3 低拍攝角度下的模型

上面一節討論了在高拍攝角度下的可行性，這裏再來處理低拍攝角度的綫路偵測。低拍攝角度下需要考慮到低拍攝角度下接脚的垂直直綫邊緣會不會被刪除的問題，以及導綫會遠離接口而導致的角度偏差，造成導綫端點水平位移。

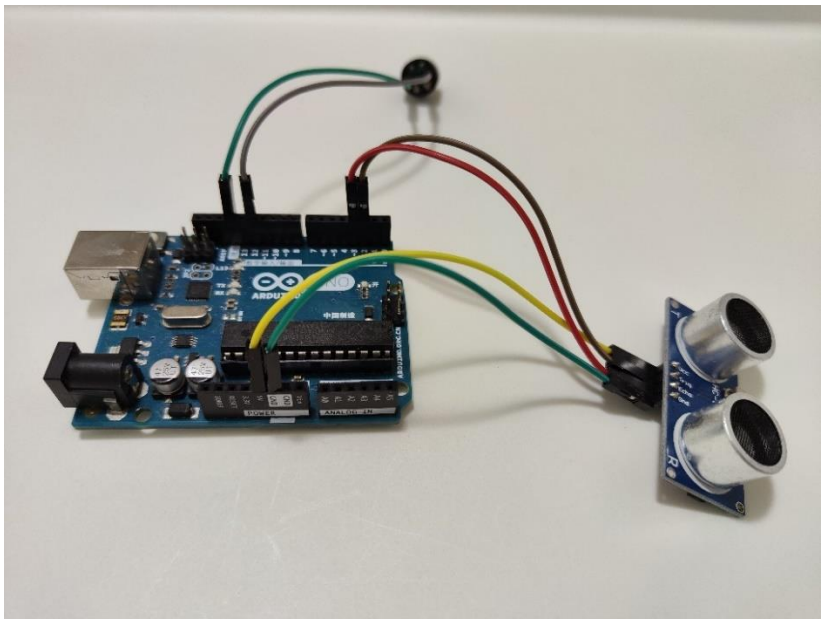

圖 4. 0.12 低拍攝角度下的 arduino 電路板接綫圖

首先找到 arduino 板 18 針接脚側的導綫與 arduino 接口一側的導綫端點。

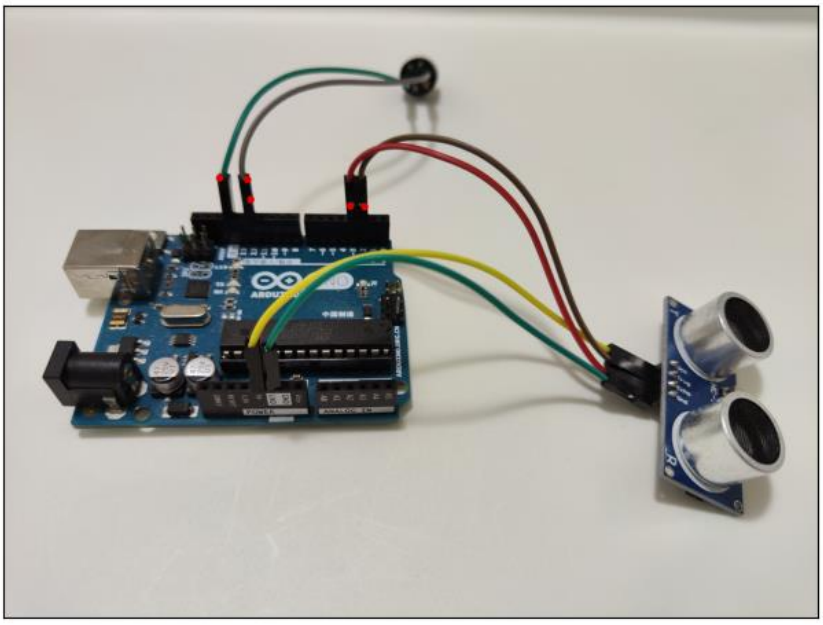

圖 4. 0.13 低拍攝角度下 18 針接脚側的導綫端點（紅點）

對低角度這個變量進行調試糾正后，我們就可以找到導綫端點位置。

```
[3.0, 5.0, 14.0, 15.0]
```

圖 4.0.14 端點位置（從左往右，開始為 0）

下面我們要解決 arduino 板 14 針接脚方向的兩個接口的偵測，翻轉 arduino 板。

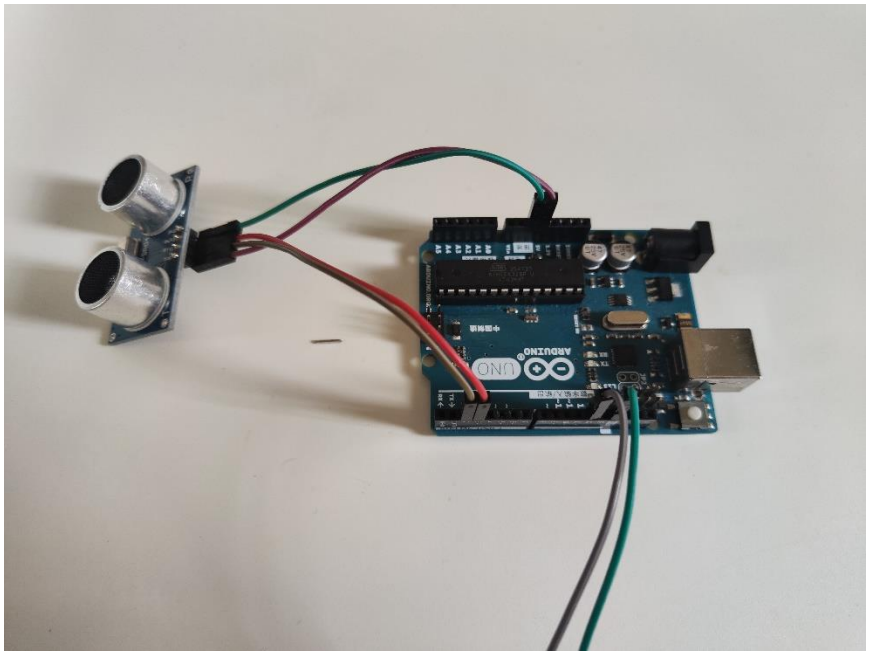

圖 4.0.15 低拍攝角度下的 14 針接脚側接綫圖

找出 14 針接脚側的導綫與 arduino 接口一側的導綫端點。

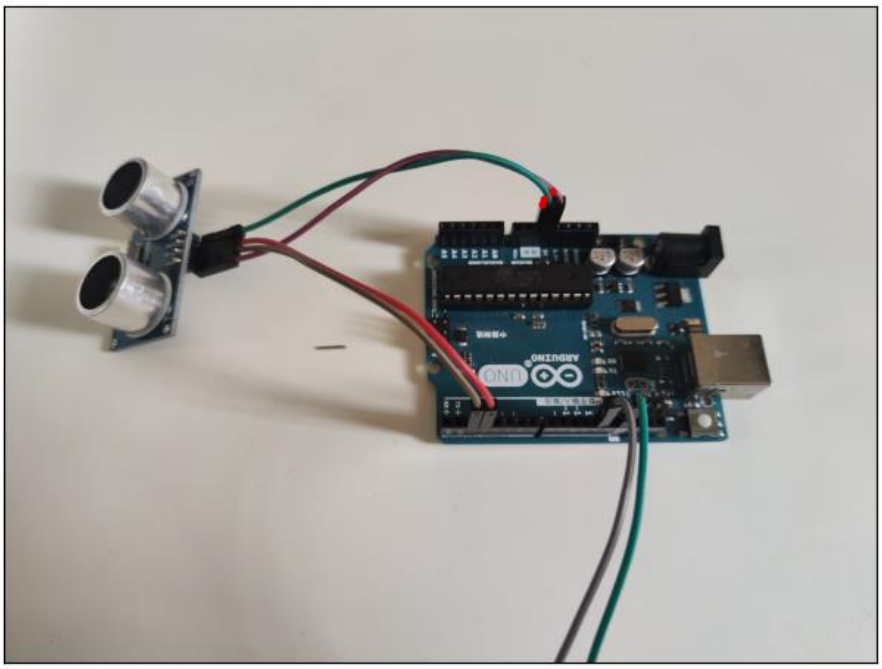

圖 4.0.16 低拍攝角度下 14 針接脚側的導綫端點（紅點）

對低角度這個變量進行調試糾正，找到導綫端點位置。

```
[8.0, 9.0]
```

圖 4.0.17 端點位置（從左往右，開始為 0）

對兩者進行整合，得到完整的端點位置信息。

```
[3.0, 5.0, 14.0, 15.0, 26.0, 27.0]
```

圖 4.0.18 完整端點位置（從左往右，開始為 0）

4.1.4 arduino 接口偵錯

找到導綫連接接口位置以後，就可以找出其接綫錯誤了。我們的接綫圖如下:

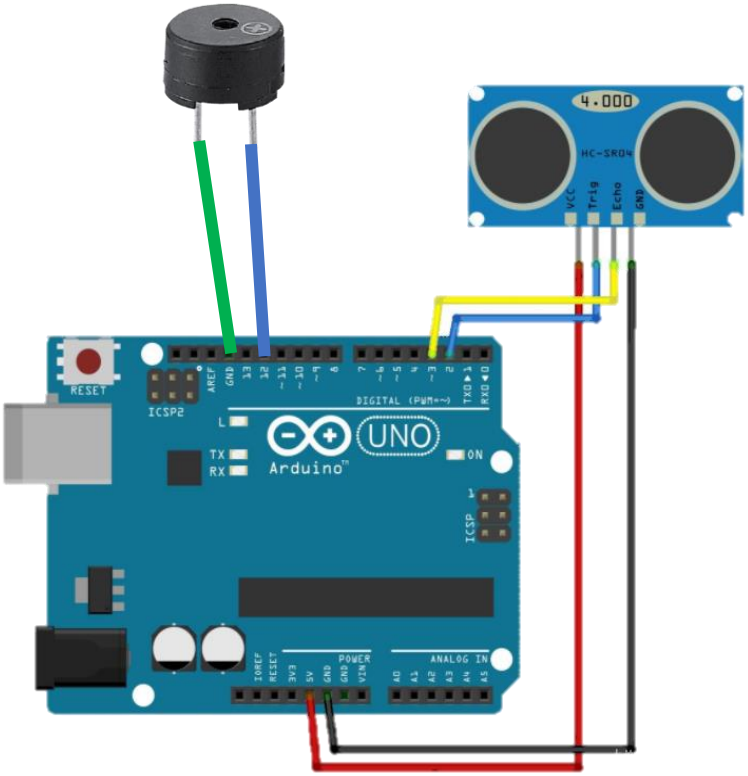

圖 4. 0.19 Arduino 電路板接綫圖

這是一個超聲波距離傳感器的接綫圖，這是 arduino 新手很常見的聯係項目之一。我們將它轉化為 2D 陣列以表現出接綫情況：

arduino	audio_p	audio_n	ultrasound_ucc	ultrasound_trig	ultrasound_echo	ultrasound_gnd
arduino_r1 (SCL)	0	0	0	0	0	0
arduino_r2 (SDA)	0	0	0	0	0	0
arduino_r3 (AREF)	0	0	0	0	0	0
arduino_r4 (GND)	1	0	0	0	0	0
arduino_r5 (13)	0	0	0	0	0	0
arduino_r6 (12)	0	1	0	0	0	0
arduino_r7 (11)	0	0	0	0	0	0
arduino_r8 (10)	0	0	0	0	0	0
arduino_r9 (9)	0	0	0	0	0	0
arduino_r10 (8)	0	0	0	0	0	0
arduino_r11 (7)	0	0	0	0	0	0
arduino_r12 (6)	0	0	0	0	0	0
arduino_r13 (5)	0	0	0	0	0	0
arduino_r14 (4)	0	0	0	0	0	0
arduino_r15 (3)	0	0	0	0	1	0
arduino_r16 (2)	0	0	0	1	0	0
arduino_r17 (1<~TX)	0	0	0	0	0	0
arduino_r18 (0~>RX)	0	0	0	0	0	0
arduino_l1 (A5)	0	0	0	0	0	0
arduino_l2 (A4)	0	0	0	0	0	0
arduino_l3 (A3)	0	0	0	0	0	0
arduino_l4 (A2)	0	0	0	0	0	0
arduino_l5 (A1)	0	0	0	0	0	0
arduino_l6 (A0)	0	0	0	0	0	0
arduino_l7 (vin)	0	0	0	0	0	0
arduino_l8 (GND)	0	0	0	0	0	0
arduino_l9 (GND)	0	0	1	0	0	0
arduino_l10 (5V)	0	0	0	0	0	1
arduino_l11 (3.3V)	0	0	0	0	0	0
arduino_l12 (RESET)	0	0	0	0	0	0
arduino_l13 (IOREF)	0	0	0	0	0	0
arduino_l14 (NONE)	0	0	0	0	0	0

圖 4. 0.20 接綫 2D 陣列圖

將之前得到的導綫所插接口和圖中對比，可以找出錯誤的接綫並予以糾正。以之前的高拍攝角度下的模型爲例。

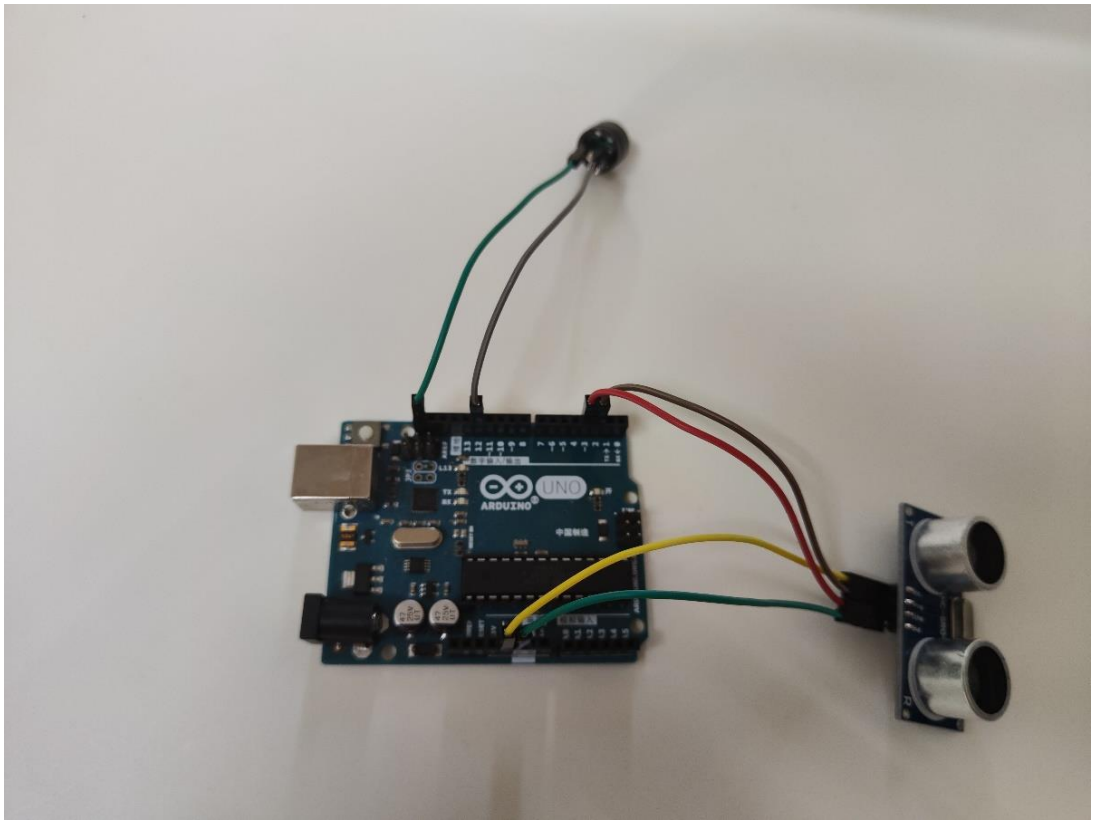

圖 4. 0.21 待偵測的 arduino 接綫圖

```
[0.0, 5.0, 14.0, 15.0, 26.0, 27.0]
```

圖 4. 0.22 完整端點位置（從左往右，開始為 0）

通過完整的端點位置信息與接綫 2D 陣列圖的比較，得出蜂鳴器接脚錯誤的信息。

```
audio_p is not correct
audio_n is correct
ultrasound_ucc is correct
ultrasound_trig is correct
ultrasound_echo is correct
ultrasound_gnd is correct
you need put the (['arduino_r1 (SCL)']) into (['arduino_r4 (GND)'])
```

圖 4. 0.23 接脚錯誤信息和改正方法

如圖可以知道我們需要將 arduino 板 18 針側的 SCL 孔位中的接脚移動到 18 針側 GND 接脚處，即可解決問題，并且生成圖像解決方案，更加清晰直觀。

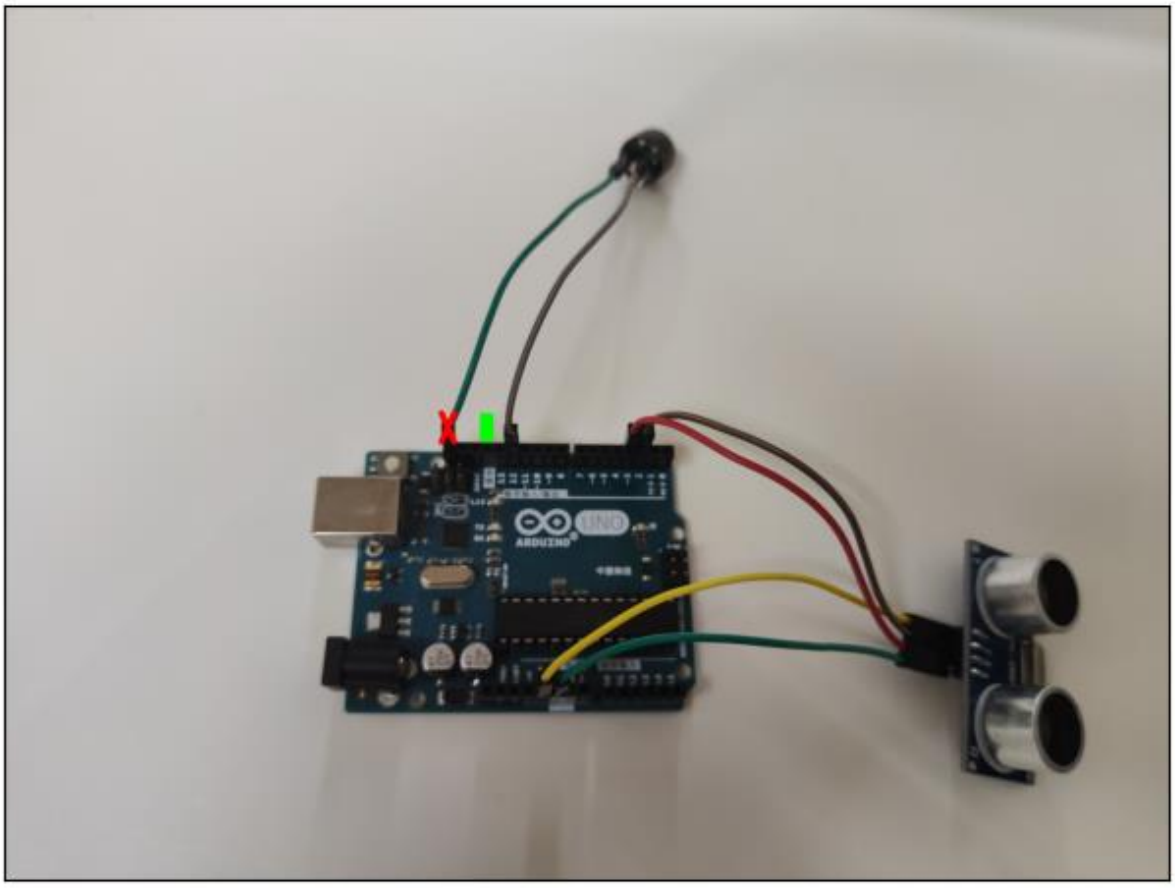

圖 4. 0.24 修正建議圖

如圖可以看到外面只需要把紅色叉叉地方的接腳拔出插到綠色矩形所提示的接腳出即可，配合文字提示，可以非常便捷地對錯誤接綫電路進行修改。

4.2 模型測試

這裏使用一些更複雜的 arduino 接綫電路進行測試，測試模型的普適性和穩定性。

4.2.1 Arduino 溫濕度計搭配 1602 LCD 測試

濕度計是 arduino 初學者到新手過度中很常用的練習項目，其接綫相對於超聲波測距儀複雜很多，容易犯錯。

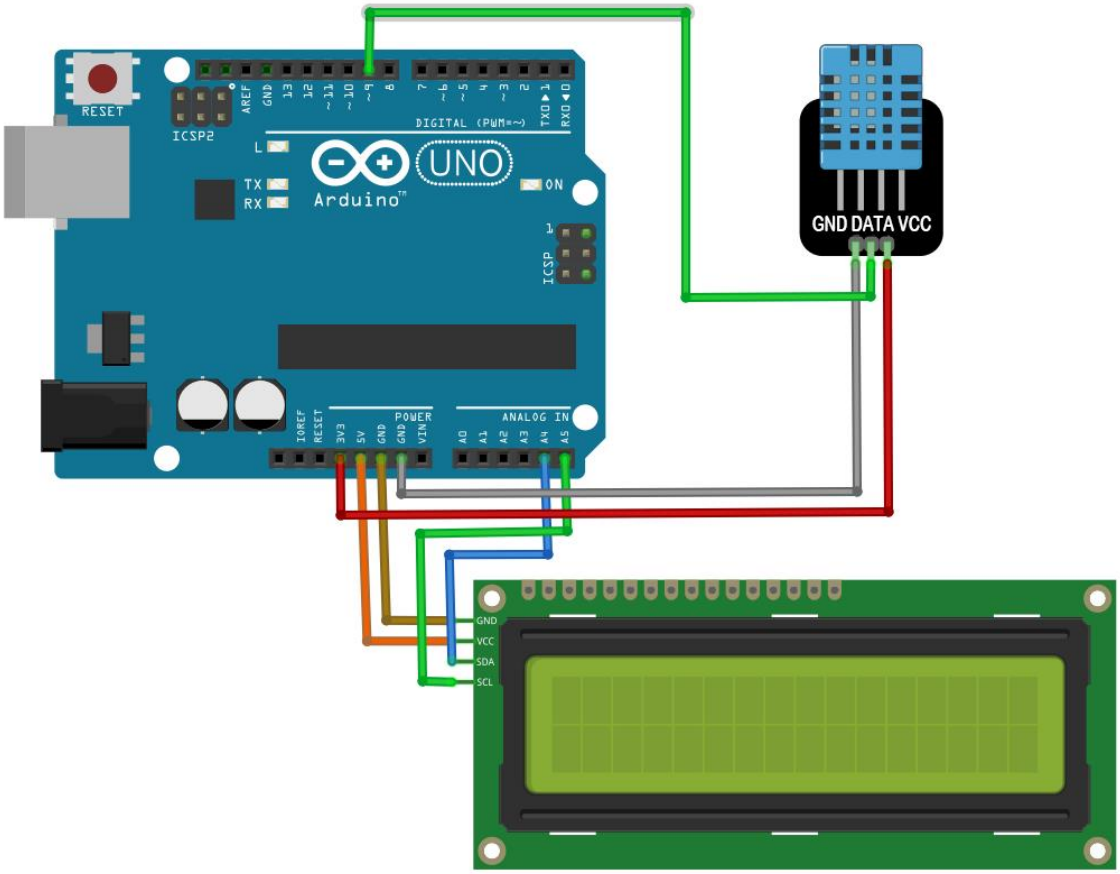

圖 4. 0.25 Arduino 溫濕度計綫路圖

我們根據綫路圖生成 2D 陣列圖

arduino	GND(濕度計)	DATA(濕度計)	VCC(濕度計)	GND(LCD)	VCC(LCD)	SDA(LCD)	SCL(LCD)
arduino_r1 (SCL)	0	0	0	0	0	0	0
arduino_r2 (SDA)	0	0	0	0	0	0	0
arduino_r3 (AREF)	0	0	0	0	0	0	0
arduino_r4 (GND)	0	0	0	0	0	0	0
arduino_r5 (13)	0	0	0	0	0	0	0
arduino_r6 (12)	0	0	0	0	0	0	0
arduino_r7 (11)	0	0	0	0	0	0	0
arduino_r8 (10)	0	0	0	0	0	0	0
arduino_r9 (9)	0	1	0	0	0	0	0
arduino_r10 (8)	0	0	0	0	0	0	0
arduino_r11 (7)	0	0	0	0	0	0	0
arduino_r12 (6)	0	0	0	0	0	0	0
arduino_r13 (5)	0	0	0	0	0	0	0
arduino_r14 (4)	0	0	0	0	0	0	0
arduino_r15 (3)	0	0	0	0	0	0	0
arduino_r16 (2)	0	0	0	0	0	0	0
arduino_r17 (1<~TX)	0	0	0	0	0	0	0
arduino_r18 (0~>RX)	0	0	0	0	0	0	0
arduino_l1 (A5)	0	0	0	0	0	0	1
arduino_l2 (A4)	0	0	0	0	0	1	0
arduino_l3 (A3)	0	0	0	0	0	0	0
arduino_l4 (A2)	0	0	0	0	0	0	0
arduino_l5 (A1)	0	0	0	0	0	0	0
arduino_l6 (A0)	0	0	0	0	0	0	0
arduino_l7 (vin)	0	0	0	0	0	0	0
arduino_l8 (GND)	1	0	0	0	0	0	0
arduino_l9 (GND)	0	0	0	1	0	0	0
arduino_l10 (5V)	0	0	0	0	1	0	0
arduino_l11 (3.3V)	0	0	1	0	0	0	0
arduino_l12 (RESET)	0	0	0	0	0	0	0
arduino_l13 (IOREF)	0	0	0	0	0	0	0
arduino_l14 (NONE)	0	0	0	0	0	0	0

圖 4. 0.26 濕度計 2D 陣列圖

然後執行程序，程序標記 2 組端點位置，整合后最終會得到完整端點位置。下面我們對其不同的接綫進行測試。

4.2.2.4 接法 1

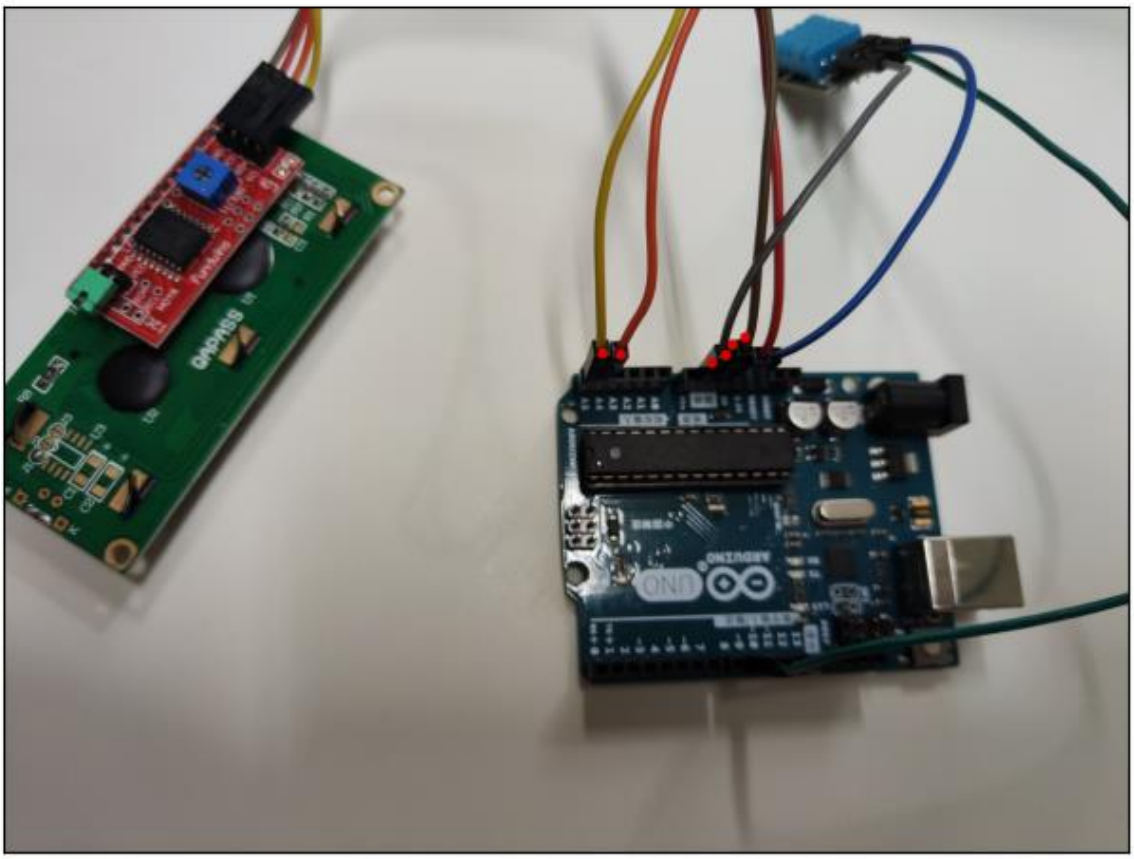

圖 4. 0.27 arduino 板 14 針腳處端點位置

```
[0.0, 1.0, 7.0, 8.0, 9.0, 10.0]
```

圖 4. 0.28 arduino 板 14 針腳處端點

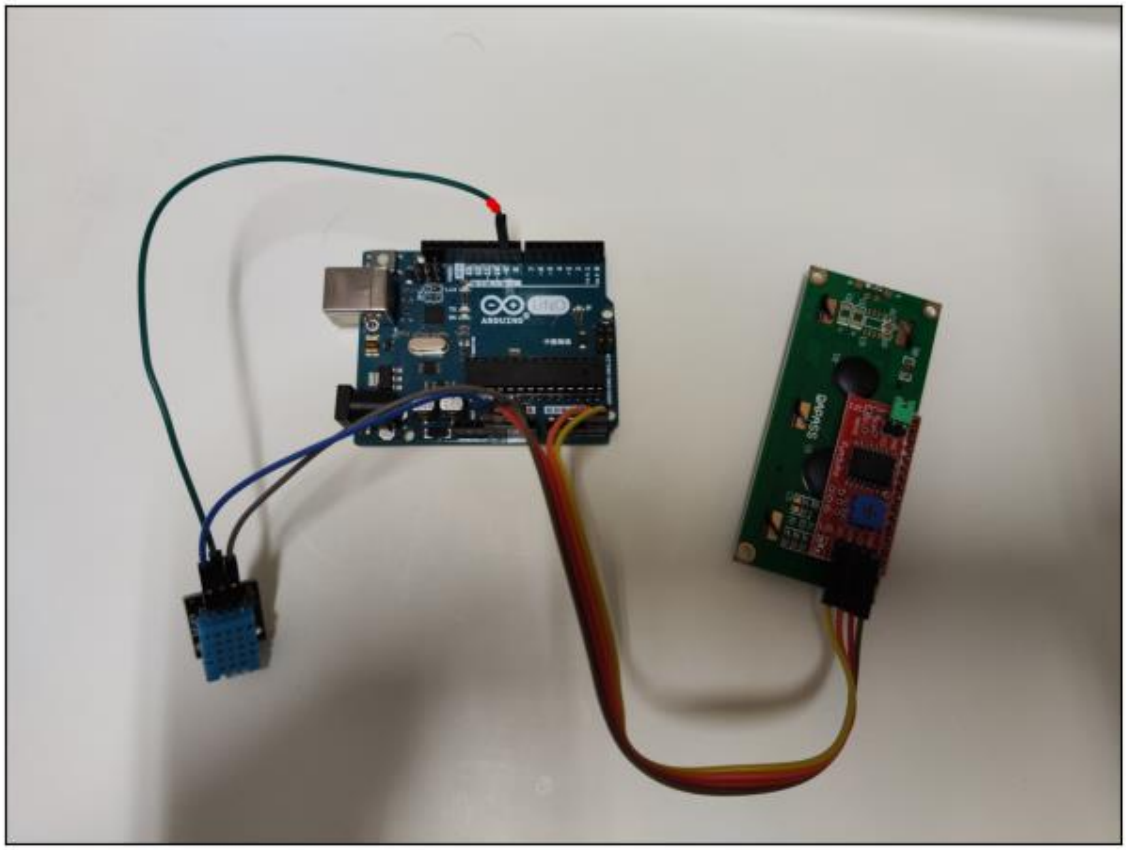

圖 4. 0.29 arduino 板 18 針脚處端點位置

```
[8.0]
```

圖 4. 0.30 arduino 板 14 針脚處端點

```
[8.0, 18.0, 19.0, 25.0, 26.0, 27.0, 28.0]
```

圖 4. 0.31 完整端點

```
GND(hygrometer) is correct
DATA(hygrometer) is correct
VCC(hygrometer) is correct
GND(LCD) is correct
VCC(LCD) is correct
SDA(LCD) is correct
SCL(LCD) is correct
you need put the ([]) into ([])
```

圖 4. 0.32 接脚錯誤信息和改正方法

可以看到這次接綫完全正確，不需要修改。

4.2.1.2 接法 2

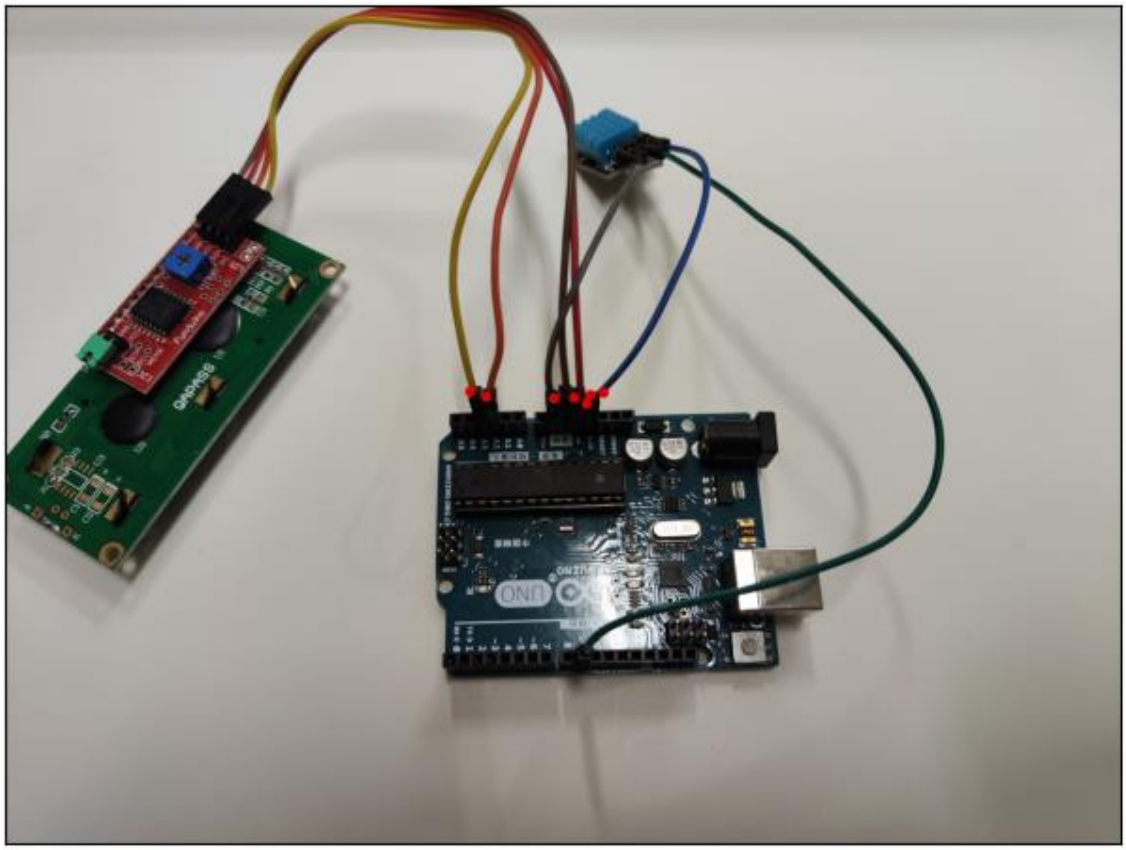

圖 4. 0.33 arduino 板 14 針腳處端點位置

```
[2.0,  3.0,  7.0,  8.0,  9.0,  10.0]
```

圖 4. 0.34 arduino 板 14 針腳處端點

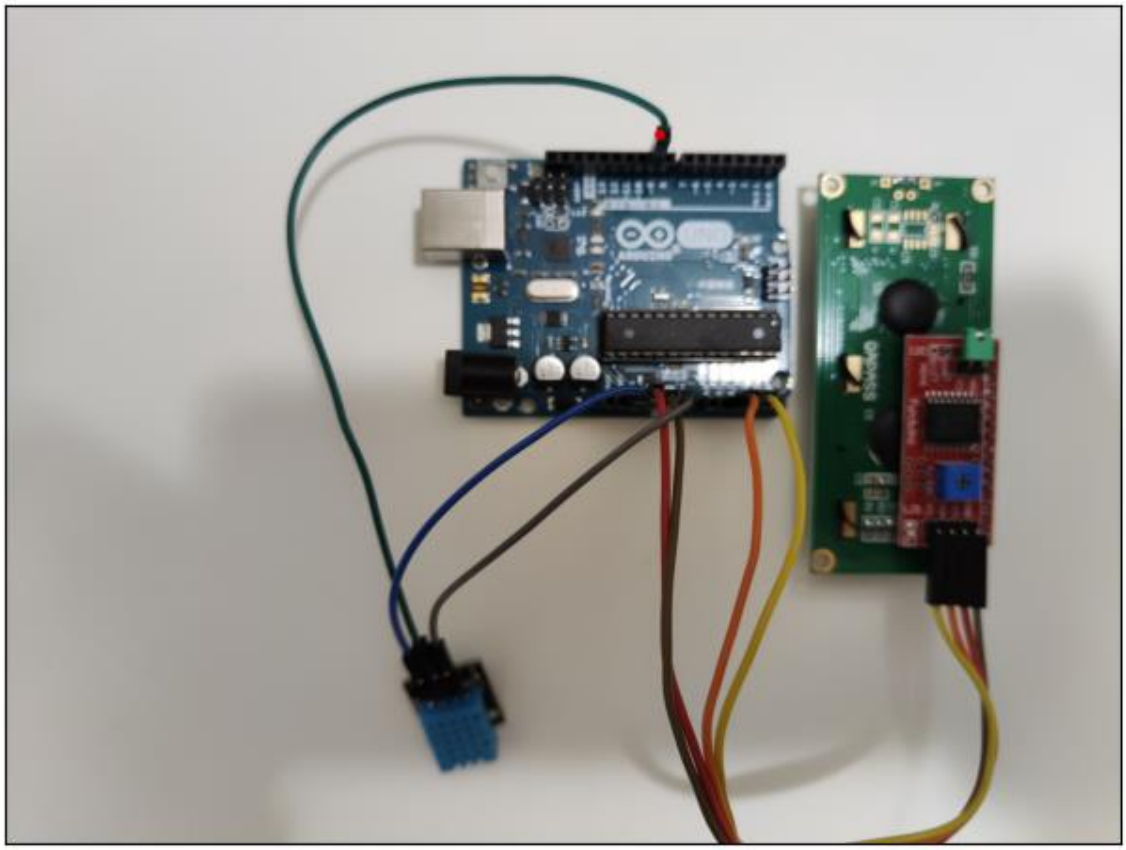

圖 4. 0.35 arduino 板 18 針腳處端點位置

```
[8.0]
```

圖 4. 0.36 arduino 板 18 針腳處端點

```
[8.0, 20.0, 21.0, 25.0, 26.0, 27.0, 28.0]
```

圖 4. 0.37 完整端點

```
GND(hygrometer) is correct
DATA(hygrometer) is correct
VCC(hygrometer) is correct
GND(LCD) is correct
VCC(LCD) is correct
SDA(LCD) is not correct
SCL(LCD) is not correct
you need put the (['arduino_13 (A3)', 'arduino_14 (A2)']) into (['arduino_12 (A4)', 'arduino_11 (A5)'])
```

圖 4. 0.38 接腳錯誤信息和改正方法

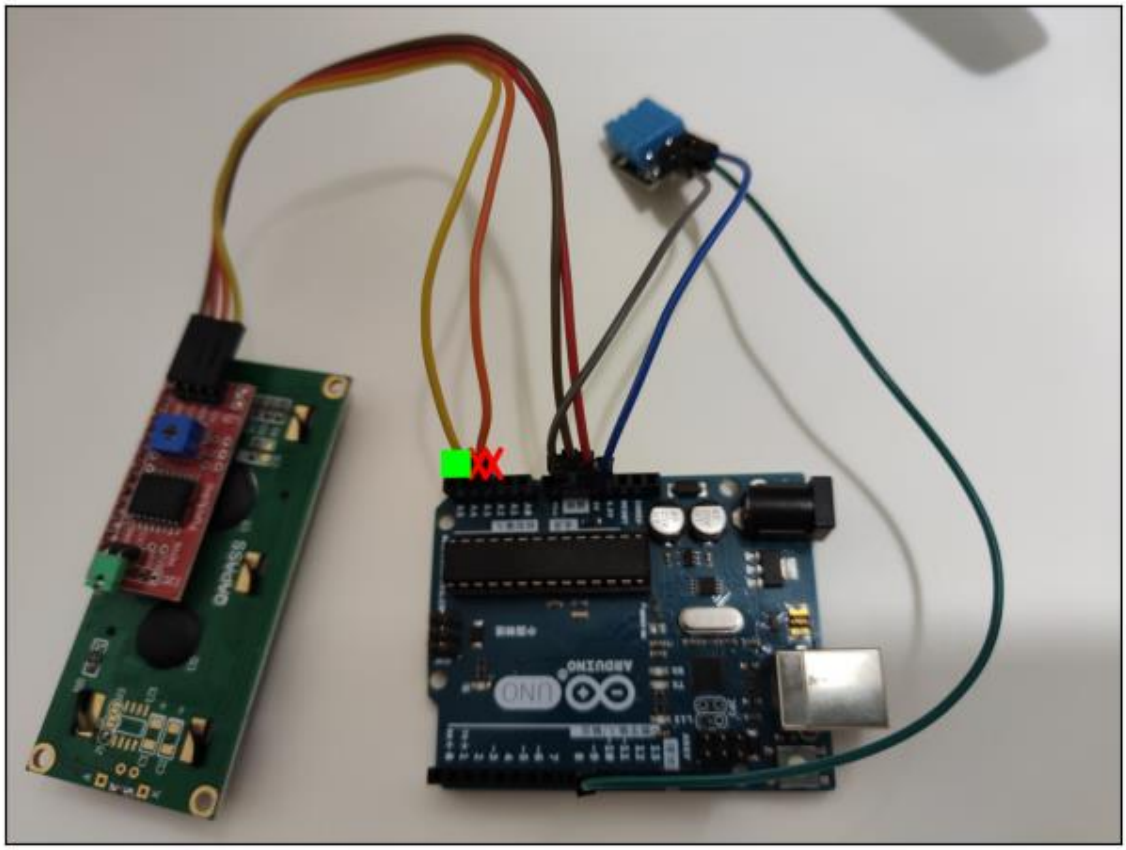

圖 4. 0.39 修正建議圖

如圖可以發現我們需要改動 2 個接腳位置，配合文字的改正方法，也可以輕鬆解決這個問題。

第五章 結論

本文通過對 arduino 電路板的一系列圖像處理，證明了 arduino 接綫電路可以通過圖像處理的方式偵測很多種電路的接綫情況，并且對其進行糾錯和修改提示。在絕大多數測試中軟體均可完美執行，并且精確地給出錯誤報告和修改建議。本算法也考慮到了使用者使用環境的問題，解決了導綫和元件雜亂遮擋、燈光昏暗、拍攝角度等使用上的變量對本算法效能的影響，很大程度上適應了很大一部分的使用情況。本算法可以識別 arduino 板上的，同樣也可以用於各種元件上的導綫接點識別，配合使用圖像識別的方法識別元件位置，進而實現電路的完整識別與偵錯。

由於時間和技術和專題規模等原因，本算法只做到了 arduino 區域的識別，但是元件區域的方法是一樣的。目前無法檢測出元件上的一些接脚錯誤，比如 arduino 上的 A 接脚原本應該和元件上的 A 接脚相連但是錯誤接到了 B，但是就像上述的一樣，這是可以通過對元件區域的識別來完善。我們也會在日後的研究中對算法進行進一步的添加完善。

參考文獻

網頁

[1]. Arduino:

https://zh.wikipedia.org/wiki/Arduino

[2]. Canny 邊緣檢測:

https://zh.wikipedia.org/wiki/Canny%E7%AE%97%E5%AD%90

[3]. MobileNets:

https://link.springer.com/article/10.1007/s10462-020-09825-6

[4]. TensorFlow:

https://zh.wikipedia.org/wiki/TensorFlow

[5]. 卷積神經網路 (CNN):

https://ieeexplore.ieee.org/abstract/document/6795724

[6]. 手繪電路圖的機器識別:

https://ieeexplore.ieee.org/abstract/document/860185

期刊論文

[1]. 吳德彥、沉浩平等，CurrentViz : Sensing and Visualizing Electric Current Flows of Breadboarded Circuits， *"UIST '17: Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology,"* Pages 343–349，October 2017

[2]. Daniel Drew, Julie L. Newcomb 等，The Toastboard: Ubiquitous Instrumentation and Automated Checking of Breadboarded Circuits *"UIST '16: Proceedings of the 29th Annual Symposium on User Interface Software and Technology,"* Pages 677–686, October 2016