自動還原魔術方塊之智慧手臂實現

林洵
國立成功大學
台南, 台灣

王偉丞
國立成功大學
台南, 台灣

蕭天鴻
國立成功大學
台南, 台灣

關鍵字
影像處理，Arduino 開發版、魔術方塊、步進馬達

研究動機

每個人在成長過程中或多或少會玩過魔術方塊這種益智遊戲，而很多人對於解魔術方塊的方法不外乎就是背公式，照著公式一步一步的還原。既然只需要靠背公式就能夠還原魔術方塊，那麼就應該有辦法能夠讓機器來代替我們做這件事。所以我們想要利用我們所會的一點工程方面的知識來製作一個能夠自動判斷並且還原的機器手臂。

研究目的

以協助初學魔術方塊者，幫助其研究公式，包括從一開始一步一步慢慢解到之後有速解法都可以。

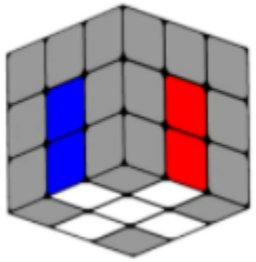
研究方法

將研究方法分為三個部分。第一部分是運用影像處理之技術，辨識出魔術方塊之各面顏色，並且能夠分辨出每一個色塊的相對位置。第二部分為程式運算部分，將影像處理完後之結果作為此部分之輸入，寫出演算法來得出要還原魔術方塊的每一個步驟。第三部分為馬達控制，利用六個馬達抓住魔術方塊每個面，將運算出來之解法步驟輸入至 arduino 開發版來控制馬達。

影像處理部分使用的程式語言是 Python 以及 OpenCV，先讀取出照片的 RGB 值來分辨出不同顏色，並且將每一個方塊位置座標化。

演算法部分有兩種，第一種是以正常人在還原一顆魔術方塊的做法-CFOP

C:cross 選擇一面為基準面，先將基準面之十字還原。

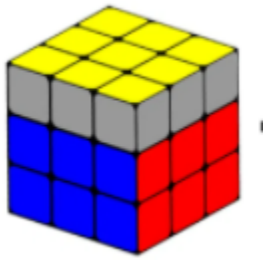


F: F2L(First two Layer)

將前兩層之魔術方塊還原

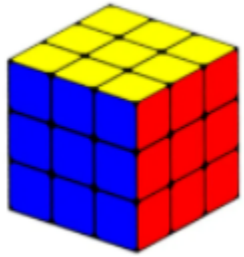
O:OLL(Orientation of last layer)

將基準面對面的面還原。



P:PLL(Permutation of last layer)

將第三層之邊塊及角塊交換已完成還原魔術方塊



第二種演算法為利用 kociemba algorithm

首先是計算要轉 90 度的次數，第二個是計算轉動外層的數量。利用群論和電腦搜索。

定義三個組 G0、G1、G3

$$G_0 = \langle U, D, L, R, F, B \rangle$$

$$G_1 = \langle U, D, L^2, R^2, F^2, B^2 \rangle$$

$$G_2 = \{1\}$$

定義一個 coset space 可以找出 Gi+1/Gi，G0 是一開始亂的情況，從 coset space 中找出 G1/G0 後就可以找到 G1，以此類推就可找到 G2 也就解完的結果。

硬體部分需要一些支架來固定馬達及魔術方塊，需要先用 soliworks 進行軟體模擬再利用 3D-Printer 來列印出固定支架。

上圖為參考之硬體外觀圖形

實驗成果

影像處理:

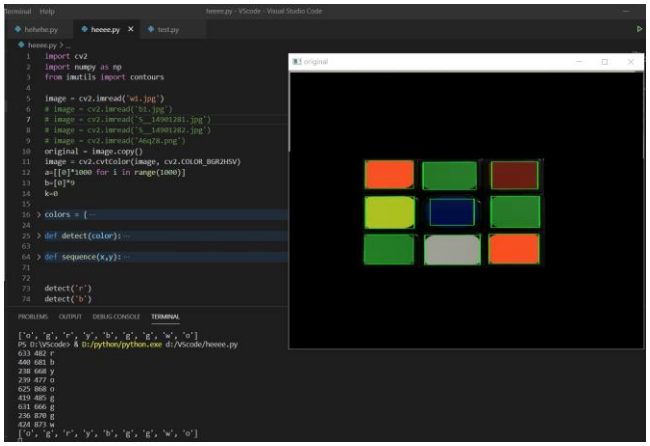
```
detect('r')
detect('b')
detect('y')
detect('o')
detect('g')
detect('w')

sequence(0,0)
sequence(1,0)
sequence(2,0)
sequence(0,1)
sequence(1,1)
sequence(2,1)
sequence(0,2)
sequence(1,2)
sequence(2,2)

print(b)

cv2.namedWindow("original",0)
cv2.resizeWindow("original", 640, 500)
cv2.imshow("original",original)
```

定義偵測函式 detect、排序函式 sequence

```
def sequence(x,y):
    for i in range(200):
        for j in range(300):
            if a[j+300*x][400+i+200*y] != 0:
                global k
                b[k]=a[j+300*x][400+i+200*y]
                k=k+1
```

```python
def detect(color):
    mask = np.zeros(image.shape, dtype=np.uint8)
    open_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (7,7))
    close_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5,5))
    lower = np.array(colors[color][0], dtype=np.uint8)
    upper = np.array(colors[color][1], dtype=np.uint8)
    color_mask = cv2.inRange(image, lower, upper)
    color_mask = cv2.morphologyEx(color_mask, cv2.MORPH_OPEN, open_kernel, iterations=18)
    color_mask = cv2.morphologyEx(color_mask, cv2.MORPH_CLOSE, close_kernel, iterations=5)

    color_mask = cv2.merge([color_mask, color_mask, color_mask])
    mask = cv2.bitwise_or(mask, color_mask)

    gray = cv2.cvtColor(mask, cv2.COLOR_BGR2GRAY)
    gray = cv2.cvtColor(mask, cv2.COLOR_BGR2GRAY)

    cnts = cv2.findContours(gray, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = cnts[0] if len(cnts) == 2 else cnts[1]

    (cnts, _) = contours.sort_contours(cnts, method="top-to-bottom")

    cube_rows = []
    row = []
    for (i, c) in enumerate(cnts, 1):
        row.append(c)
        if i % 1 == 0:
            (cnts, _) = contours.sort_contours(row, method="left-to-right")
            cube_rows.append(cnts)
            row = []
    number = 0
    for row in cube_rows:
        for c in row:
            x,y,w,h = cv2.boundingRect(c)
            cv2.rectangle(original, (x, y), (x + w, y + h), (36,255,12), 2)
            number += 1
            print(x,y,color)
            if a[x][y]==0:
                a[x][y]=color
```

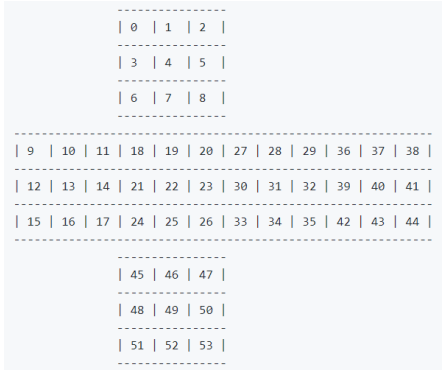
把圖片輸入程式之後，會有偵測完的顏色和那些顏色之間的相對位置。

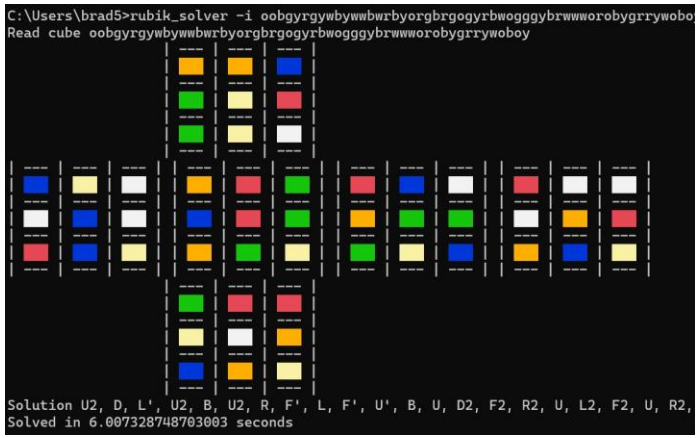


輸出結果會有顏色跟所在的座標。

演算法:

先選定魔術方塊的其中一面為基準面再將每個面攤開。用一個一維矩陣來存每個色塊代表的顏色。





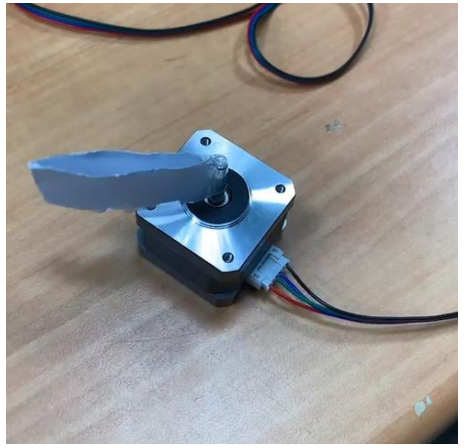
將色塊依序輸入至程式之後，得到的輸出結果就會是解法，用符號來代表各個軸的轉向。
(例如:U 代表 Up，B’ 中的”B”代表 Back
“‘ ”代表逆時針方向)

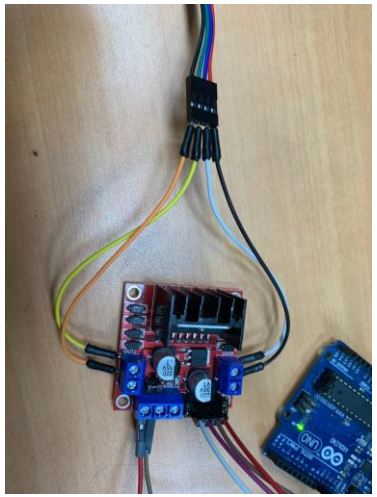
馬達控制:

利用 arduino 加上 L298N 控制馬達編寫程式碼讓馬達能精準轉動±90° 及 180° ，然後我們也有測試馬達轉速，有一定的速度。

預計使用一個 arduino 及六個 L298N 來控制馬達。

```
#include <Stepper.h>                        // degree180 ->step
// Define number of steps per revolution:
const int stepsPerRevolution = 200;
// Initialize the stepper library on pins 8 through 11:
Stepper myStepper = Stepper(stepsPerRevolution, 8, 9, 10, 11);
int speed1 = 100;
int speed2 = 0;
void setup() {
  // Set the motor speed (RPMs):
  //myStepper.setSpeed(300);

  myStepper.setSpeed(speed1);
}
void loop() {
  // Step one revolution in one direction:
  speed2 = map(analogRead(A0),0,1024,0,500);
  speed1 = 300*speed2/1024;
  Serial.print("speed1");
  //myStepper.setSpeed(speed1);
  myStepper.step(53);
```

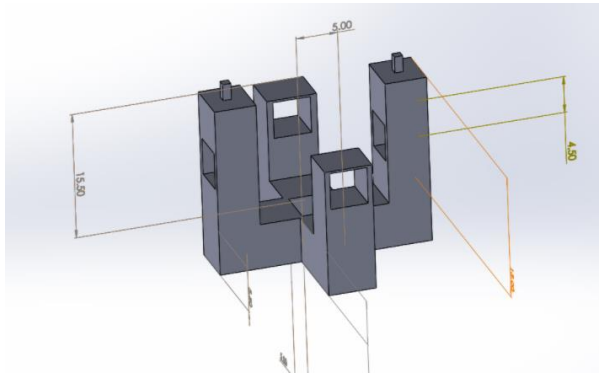


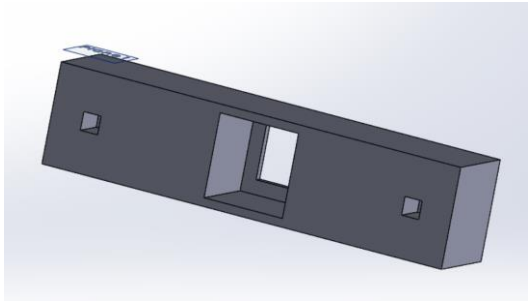


上圖為電路圖

硬體部份:

目前有用 SOLIDWORKS 畫出主架構。





未來規劃

下學期要將不同的程式語言整合(影像處理部份及演算法部份)，也要將軟、硬體之間整合，使演算法輸出結果能夠對馬達下命令。

硬體部份也要將 solidworks 模擬之結果利用 3D 列印實現。實體用鏡頭做影像辨識方面也要研究。

3月	4月	5月	6月	7月	8月
將solidworks模擬之結果利用3D列印實現				實際測試	
影像辨識			不同程式語言之整合		

結論

目前所做出的成果還非常少，不過也是有慢慢地找到方向，影像處理方面已經能夠在圖片上定位並且辨識出顏色，要將辨識結果輸出成一維矩陣讓演算法去計算出解法傳給 arduino 來控制馬達轉動魔術方塊。在各個不同的程式語言的整合也需要很多時間來完成，要花更多的時間來研究。硬體的 3D 列印部分先等系上之 3D 列印機開放使用之後就會去做，列印方面也有可能會遇到機器所能畫出的精度與我們模擬之圖形不同所以需要調整。

參考文獻

https://github.com/Wiston999/python-rubik

https://newatlas.com/jay-flatland-paul-rose-rubiks-cube-robot/41523/

https://www.arduino.cc/en/Tutorial/LibraryExamples/StepperSpeedChttps://www.speedsolving.com/wiki/index.php/Kociemba%27s_Algorithmontrol