國立中興大學

一一一學年度第一學期

數位影像處理

第一次平時作業

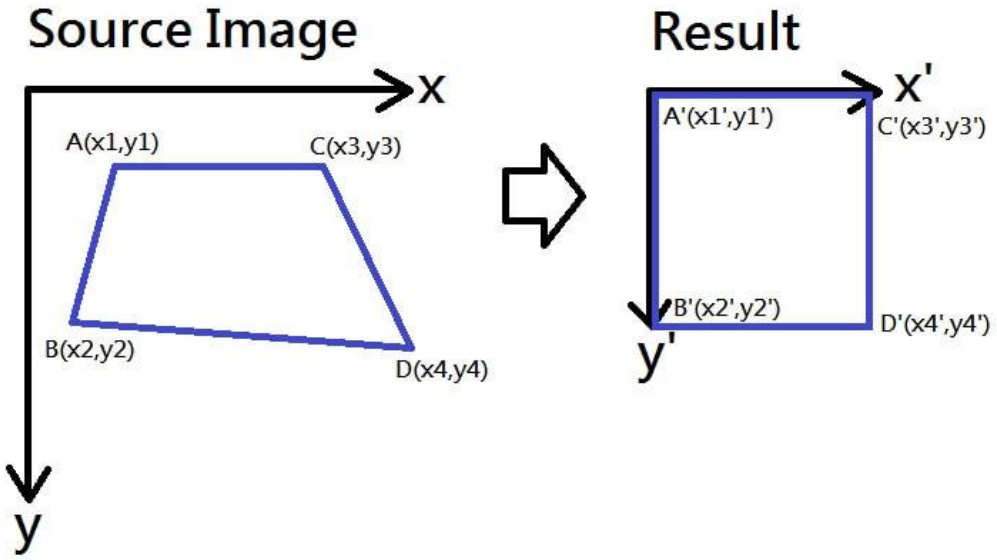
班級：資工碩士一年級
學號：7110056210
學生：丁吾心
授課教師：吳俊霖 教授
繳交日期：111 年 11 月 25 日

目錄

透視變型校正 Perspective Distortion Correction

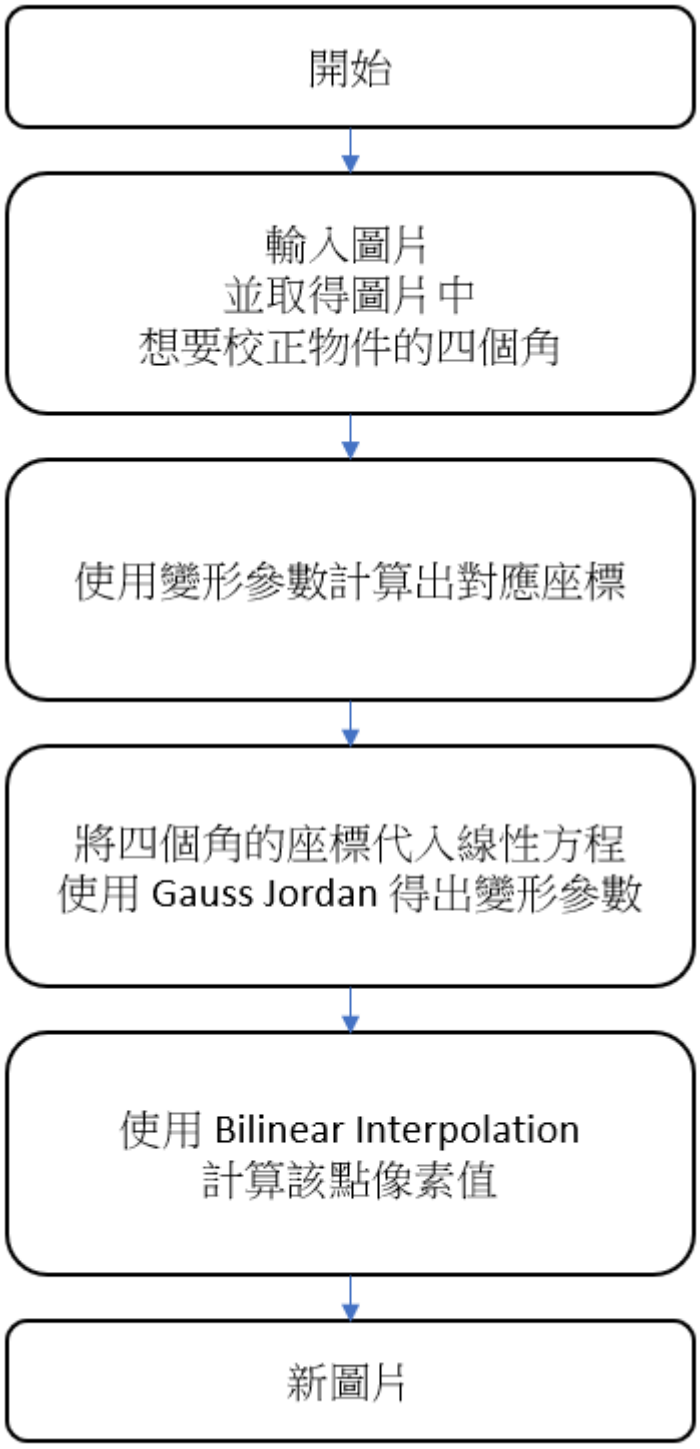
1.1 題目說明

透視變形是指一般在拍攝影片、照片的時候，你可以看到一個物體及其周圍區域與標準鏡頭中看到的相比完全不同，由於遠近特徵的相對比例變化，發生了彎曲或變形。所以通常我們在自然背景拍攝一張照片時，或多或少都會有一些透視變形存在，而透視變形是由拍攝和觀看圖像的相對距離決定的，因為成像的視角也許會比觀看物體的視角更窄或是更廣，這樣看上去的相對距離就會與所期待的不一樣。因此需做透視變形校正，將圖片中該物體校正為正視該物體的樣子。



圖一

1.2 演算法流程


開始
輸入圖片
並取得圖片中
想要校正物件的四個角
使用變形參數計算出對應座標
將四個角的座標代入線性方程
使用 Gauss Jordan 得出變形參數
使用 Bilinear Interpolation
計算該點像素值
新圖片

圖二

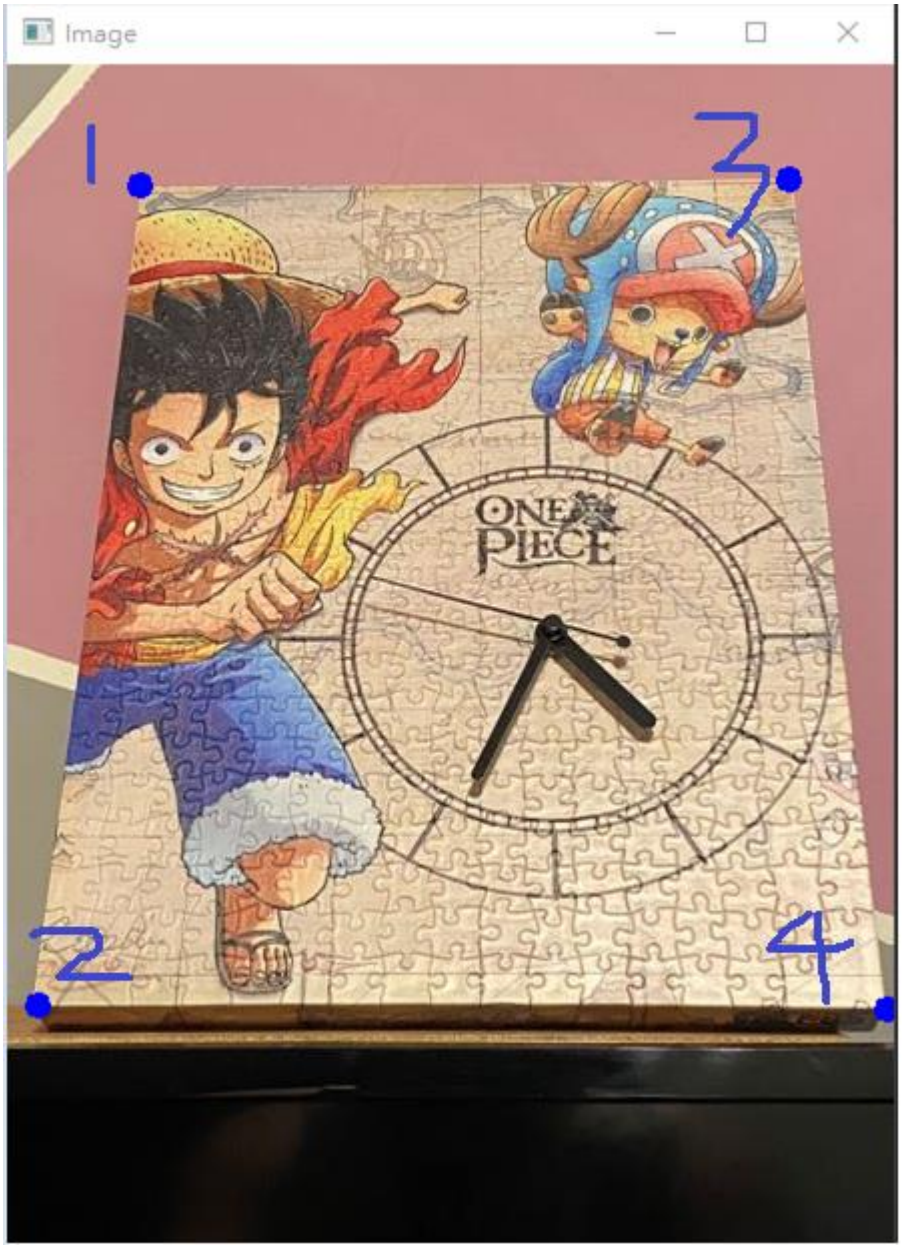
1.3 詳細流程與說明

1.3.1 圖片取得

拍攝圖片，盡量以矩形的物件為主，從不同角度拍攝

1.3.2 選取物件的四個角

依照該物件的左上、左下、右上、右下，依序選取



圖三

1.3.3 依照選取的點，計算新圖的長與寬

```
# 選取四個角落
corner_point = Get_points( ori_img )
corner_point = np.array( corner_point )
if len( corner_point ) != 4:
    print( "Error, please select 4 point to execute program" )
    exit( 1 )

# 取得新圖片的大小
new_row = Euclidean_dist( corner_point[ 0 ], corner_point[ 1 ] )
new_col = Euclidean_dist( corner_point[ 0 ], corner_point[ 2 ] )
new_size = [ [ 0, 0 ], [ new_row - 1, 0 ], [ 0, new_col - 1 ], [ new_row - 1, new_col - 1 ] ]
a = np.zeros( ( ARR_SIZE, ARR_SIZE ), dtype = np.float64 )
b = list( corner_point.T.reshape( -1 ) )
Set_matrix( a, new_size, ARR_SIZE )
```

圖四

1.3.4 計算線性方程

```
def Gauss_jordan( a_copy, b_copy, max_size ):

    i, j, p, s = 0, 0, 0, 0
    a_temp = a_copy.copy()
    ret_solu = b_copy.copy()
    temp = np.zeros( ( max_size, max_size ), dtype = np.float64 )

    for p in range( max_size - 1 ):
        s = p
        r = math.fabs( a_temp[ p, p ] )
        for i in range( p, max_size ):
            if r < math.fabs( a_temp[ i, p ] ):
                r = math.fabs( a_temp[ i, p ] )
                s = i
        if s != p:
            a_temp[ [ p, s ] ] = a_temp[ [ s, p ] ]
            ret_solu[ p ], ret_solu[ s ] = ret_solu[ s ], ret_solu[ p ]
        for i in range( p + 1, max_size ):
            temp[ i, p ] = a_temp[ i, p ] / a_temp[ p, p ]
            for j in range( p, max_size ):
                a_temp[ i, j ] = a_temp[ i, j ] - temp[ i, p ] * a_temp[ p, j
            ret_solu[ i ] = ret_solu[ i ] - ( temp[ i, p ] * ret_solu[ p ] )

    if a_temp[ p, p ] < 0.0001:
        print( "Error!Can not find the solution!" )
        exit( 1 )

    for i in range( max_size - 1, -1, -1 ):
        u = 0
        for j in range( i + 1, max_size ):
            u = u + a_temp[ i, j ] * ret_solu[ j ]
        ret_solu[ i ] = ( ret_solu[ i ] - u ) / a_temp[ i, i ]

    return ret_solu
```

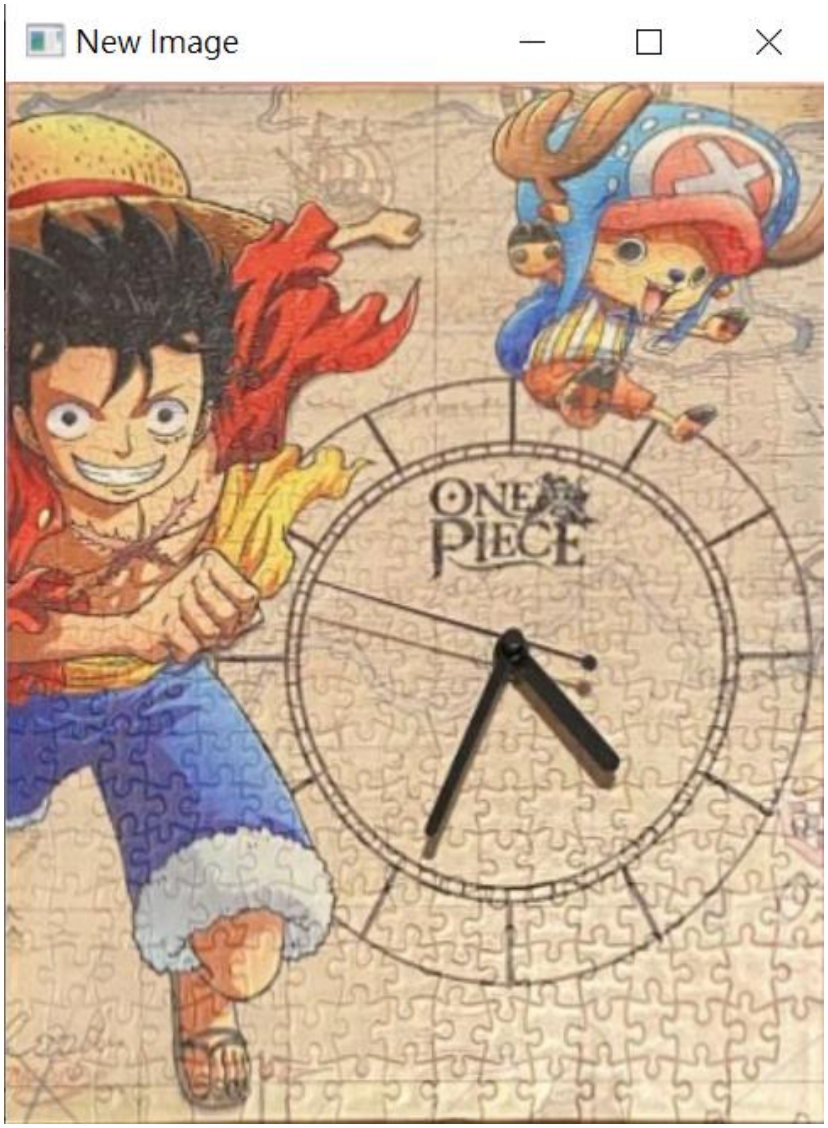
圖五 Gauss Jordan 解線性方程

```python
# 透視變形轉換
def Transform( img_temp, solu_temp, r_temp, c_temp ):
    ret_img = np.zeros( ( r_temp, c_temp, 3 ), dtype = np.uint8 )
    for i in range( r_temp ):
        for j in range( c_temp ):
            float_y = solu_temp[ 0 ] * i + solu_temp[ 1 ] * j + solu_temp[ 2 ] * i * j + solu_temp[ 3 ]
            float_x = solu_temp[ 4 ] * i + solu_temp[ 5 ] * j + solu_temp[ 6 ] * i * j + solu_temp[ 7 ]
            y = math.floor( float_y )
            x = math.floor( float_x )
            v = float_y - y
            u = float_x - x

            # Bilinear Interpolation
            ret_img[ i ][ j ] = ( ( 1 - u ) * ( 1 - v ) * img_temp[ x ][ y ]) + ( u * ( 1 - v ) * img_temp[ x + 1 ][ y ]
                    v * ( 1 - u ) * img_temp[ x ][ y + 1 ] ) + ( u * v * img_temp[ x + 1 ][ y + 1 ] )
    return ret_img
```

圖六 變形並做 Bilinear Interpolation

1.4 結果圖



圖七

1.5 程式碼

```
import cv2
import numpy as np
import math
import os
from os import walk

ARR_SIZE = 8

def Set_matrix( a_temp, corner_temp, arr_size ):
    for i in range( 0, arr_size ):
        if i < 4:
            a_temp[ i, 0 ] = corner_temp[ i ][ 0 ]
            a_temp[ i, 1 ] = corner_temp[ i ][ 1 ]
            a_temp[ i, 2 ] = corner_temp[ i ][ 0 ] * corner_temp[ i ][ 1 ]
            a_temp [i, 3 ] = 1
            print("result=    \n", a_temp[i, 0], a_temp[i, 1], a_temp[i, 2],
a_temp[i, 3])
        else:
            a_temp[ i, 4 ] = corner_temp[ i - 4 ][ 0 ]
            a_temp[ i, 5 ] = corner_temp[ i - 4 ][ 1 ]
            a_temp[ i, 6 ] = corner_temp[ i - 4 ][ 0 ] * corner_temp[ i -
4 ][ 1 ]
            a_temp[ i, 7 ] = 1

def Mouse_handler( event, x, y, flags, data ):
    if event == cv2.EVENT_LBUTTONDOWN and len( data[ 'points' ])
< 4:
        cv2.circle( data[ 'img' ], ( x, y ), 3, ( 255, 0, 0 ), 5 )
        cv2.imshow( "Image", data[ 'img' ] )
        print( "get points: ( x, y ) = ({}, {})".format( x, y ) )
        data[ 'points' ].append( ( x, y ) )

def Get_points( im ):
    ret_data = { 'img': im.copy(), 'points': [] }
```

```
    cv2.namedWindow( "Image", 0 )
    h, w, dim = im.shape
    cv2.resizeWindow( "Image", w, h )

    cv2.imshow( 'Image', im )
    cv2.setMouseCallback( "Image", Mouse_handler, ret_data )
    cv2.waitKey()
    return ret_data[ 'points' ]

def Gauss_jordan( a_copy, b_copy, max_size ):

    i, j, p, s = 0, 0, 0, 0
    a_temp = a_copy.copy()
    ret_solu = b_copy.copy()
    temp = np.zeros( ( max_size, max_size ), dtype = np.float64 )

    for p in range( max_size - 1 ):
        s = p
        r = math.fabs( a_temp[ p, p ] )
        for i in range( p, max_size ):
            if r < math.fabs( a_temp[ i, p ] ):
                r = math.fabs( a_temp[ i, p ] )
                s = i
        if s != p:
            a_temp[ [ p, s ] ] = a_temp[ [ s, p ] ]
            ret_solu[ p ], ret_solu[ s ] = ret_solu[ s ], ret_solu[ p ]
        for i in range( p + 1, max_size ):
            temp[ i, p ] = a_temp[ i, p ] / a_temp[ p, p ]
            for j in range( p, max_size ):
                a_temp[ i, j ] = a_temp[ i, j ] - temp[ i, p ] * a_temp[ p,
j ]
            ret_solu[ i ] = ret_solu[ i ] - ( temp[ i, p ] * ret_solu[ p ] )

    if a_temp[ p, p ] < 0.0001:
        print( "Error!Can not find the solution!" )
        exit( 1 )
```

```
    for i in range( max_size - 1, -1, -1 ):
        u = 0
        for j in range( i + 1, max_size ):
            u = u + a_temp[ i, j ] * ret_solu[ j ]
        ret_solu[ i ] = ( ret_solu[ i ] - u ) / a_temp[ i, i ]

    return ret_solu

# 透視變形轉換
def Transform( img_temp, solu_temp, r_temp, c_temp ):
    ret_img = np.zeros( ( r_temp, c_temp, 3 ), dtype = np.uint8 )
    for i in range( r_temp ):
        for j in range( c_temp ):
            float_y = solu_temp[ 0 ] * i + solu_temp[ 1 ] * j +
solu_temp[ 2 ] * i * j + solu_temp[ 3 ]
            float_x = solu_temp[ 4 ] * i + solu_temp[ 5 ] * j +
solu_temp[ 6 ] * i * j + solu_temp[ 7 ]
            y = math.floor( float_y )
            x = math.floor( float_x )
            v = float_y - y
            u = float_x - x

            # Bilinear Interpolation
            ret_img[ i ][ j ] = ( ( 1 - u ) * ( 1 - v ) * img_temp[ x ][ y ])
+ ( u * ( 1 - v ) * img_temp[ x + 1 ][ y ]) + (
                        v * ( 1 - u ) * img_temp[ x ][ y + 1 ] ) + ( u * v *
img_temp[ x + 1 ][ y + 1 ] )
    return ret_img

# 用原圖計算新圖的歐式距離
def Euclidean_dist( x, y ):
    ret_dist = ( abs( x - y ) ** 2 ).sum() ** ( 1 / 2 )
    return int( ret_dist )

if __name__ == '__main__':
    print( "DIP Homework_1~~~\n" )
```

```
def main():

    # 取得所有圖片
    picture_name = []
    now_path = os.getcwd()
    for root, dirs, files in walk( now_path ):
        for file in files:
            if '.jpg' in file:
                picture_name.append( file )

    # Run 全部圖片
    for cnt in range( len( picture_name ) ):

        # 讀取圖片
        ori_img = cv2.imread( picture_name[ cnt ], cv2.IMREAD_COLOR )

        # 選取四個角落
        corner_point = Get_points( ori_img )
        corner_point = np.array( corner_point )
        if len( corner_point ) != 4:
            print( "Error, please select 4 point to execute program" )
            exit( 1 )

        # 取得新圖片的大小
        new_row = Euclidean_dist( corner_point[ 0 ], corner_point[ 1 ] )
        new_col = Euclidean_dist( corner_point[ 0 ], corner_point[ 2 ] )
        new_size = [ [ 0, 0 ], [ new_row - 1, 0 ], [ 0, new_col - 1 ], [ new_row - 1, new_col - 1 ] ]
        a = np.zeros( ( ARR_SIZE, ARR_SIZE ), dtype = np.float64 )
        b = list( corner_point.T.reshape( -1 ) )
        Set_matrix( a, new_size, ARR_SIZE )

        # 解方程式
        solu = Gauss_jordan( a, b, ARR_SIZE )

        # 開始轉換
```

```
        new_img = Transform( ori_img, solu, new_row, new_col )

        # 輸出
        cv2.imshow( 'New Image', new_img )
        cv2.imwrite( 'output_' + picture_name[ cnt ], new_img )
        cv2.waitKey()
        cv2.destroyAllWindows()

main()
```

1.6 心得分享

這次的實做我覺得很有趣，是我第一次做影像處理，以前都認為只要用套件，很快就可以完成了，但這次自己在實做的過程，對此大大的改觀，現在雖然有很多套件可以用，但若沒有自己動手做一次，很難去理解其中的精隨，也不會去想有沒有什麼更好的方法，很感謝教授的指導，也謝謝辛苦改作業的學長姊。