機器視覺作業報告

使用 ArUco 增強實境

姓名:黎芷柔
學號:00957103
日期:2023/5/5

方法

- 二維做法之想法，演算步驟

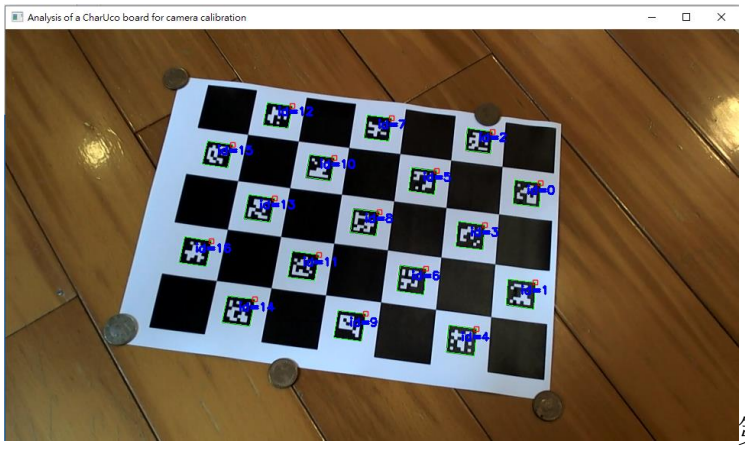
1. 使用 ChArUco Board 做相機校正

```
#使用ChArUco Board做相機校正
cap = cv2.VideoCapture('CharUco_board.mp4')

totalFrame   = int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) #擷取影片總幀數
frameWidth   = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))//2 #擷取影片一半大小的寬
frameHeight  = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))//2 #擷取影片一半大小的高

arucoParams  = aruco.DetectorParameters_create()
#創建 aruco 檢測器參數對象，可以用來設置偵測器的參數
arucoParams.cornerRefinementMethod = aruco.CORNER_REFINE_SUBPIX
#使用子像素級別的角點檢測演算法，以提高角點檢測的準確性
arucoDict    = aruco.Dictionary_get(aruco.DICT_6X6_250)
#加載了預定義的ArUco標記字典，6x6表示每個標記都是一個6x6的方陣
#250表示這個字典中一共包含了250個這樣的標記，每個標記都有一個唯一的ID

# 必須描述ChArUco board的尺寸規格
gridX        = 5 # 水平方向5格
gridY        = 7 # 垂直方向7格
squareSize   = 4 # 每格為4cmX4cm
charucoBoard = aruco.CharucoBoard_create(gridX,gridY,squareSize,squareSize/2,arucoDict)
#gridX x gridY 的 ChArUco board
#squareLength:每個格子的邊長
#markerLength:示每個 ArUco marker 的邊長 -> ArUco marker為2cmX2cm
#arucoDict:要使用的 ArUco 字典

print('height {}, width {}'.format(cap.get(cv2.CAP_PROP_FRAME_HEIGHT),cap.get(cv2.CAP_PROP_FRAME_WIDTH)))
refinedStrategy = True #refinedStrategy 表示是否旨用检测到的角点的精确化方法
criteria        = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.00001)
#criteria 是用來控制這個迭代(求解相機內部參數時)過程的停止條件
#當最大迭代次數(cv2.TERM_CRITERIA_MAX_ITER)為100，收斂誤差小(cv2.TERM_CRITERIA_EPS)於0.00001時，就停止迭代求解內部參數
frameId         = 0
collectCorners = []
collectIds     = []
collectFrames  = []
```

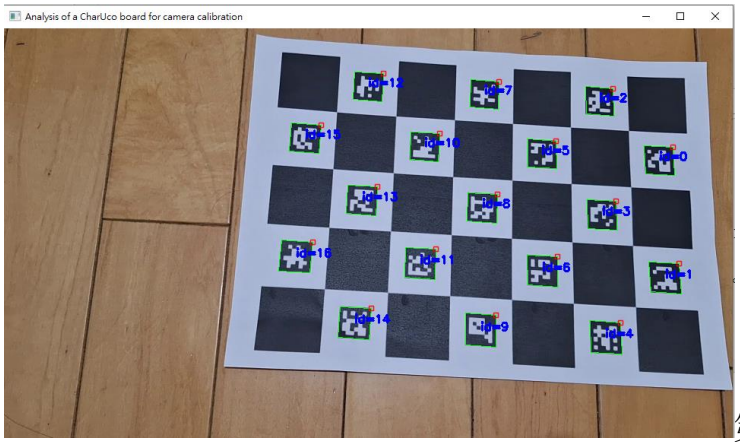
```
while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.resize(frame,(frameWidth,frameHeight)) #重塑影像大小
    (corners, ids, rejected) = aruco.detectMarkers(frame, arucoDict, parameters=arucoParams)
    #aruco.detectMarkers 函數偵測影像中的 ArUco marker
    #傳入影格、aruco 字典和偵測參數 arucoParams 作為參數
    #回傳檢測到的標記的角落點位置（corners）、對應的標記 ID（ids），以及被排除的標記（rejected）
    #角落點位置是一個 N 個元素的 List，每個元素包含標記上的角落點位置
    #標記 ID 則是一個 N 個元素的 List，每個元素是標記的整數 ID
    #被排除的標記則是沒有被偵測到的標記

    if refinedStrategy:
        corners, ids, _, _ = aruco.refineDetectedMarkers(frame,charucoBoard,corners,ids,rejected)
        #aruco.refineDetectedMarkers() 函數進行檢測到的 ArUco marker 的角點精確化
        #frame：當前的影像幀。
        #charucoBoard：包含所檢測的 ArUco marker 的 CharUco board。
        #corners：所有檢測到的 ArUco marker 的角點。
        #ids：所有檢測到的 ArUco marker 的 ID。
        #rejected：被拒絕的角點。
        #輸出為精確化後的角點、ID，以及被拒絕的角點
    if frameId % 100 == 50 and len(ids)==17: #當幀數(frameId)除以100的餘數為50，且檢測到的ArUco markers數量為17時
        #收集符合條件的角點(corners)、ID(ids)、以及圖像幀(frame)，並將它們儲存在列表中
        collectCorners.append(corners)
        collectIds.append(ids.ravel()) #ravel:用於將多維數組降為一維
        collectFrames.append(frame)

    if len(corners) > 0: #偵測到一組以上的ArUco marker
        aruco.drawDetectedMarkers(frame, corners, ids) #會在影像上繪製偵測到的 ArUco 標記的邊界框和標記 ID

    cv2.imshow('Analysis of a CharUco board for camera calibration',frame)
    if cv2.waitKey(20) != -1:
        break

    frameId += 1

cv2.destroyAllWindows()
cap.release()
```

```
height 1080.0, width 1920.0
```

注意:若是在影片還未跑完前就跳出，會導致錯誤發生。因為可能根本還沒蒐集到有效校正影像，又過早跳出可能蒐集到太少的有效校正影像，會使得計算出的參數存在誤差。



第一部實驗影片的棋盤



第二部實驗影片的棋盤

2. 使用 calibrateCameraAruco 做第一次相機校正

```
#使用calibrateCameraAruco
caliCorners=np.concatenate([np.array(x).reshape(-1,4,2) for x in collectCorners],axis=0)
#list轉numpy，並reshape成三維數組，-1 表示根據數組長度自動填缺失的那個維度大小，也就是角點組數量
#numpy.concatenate用於將兩個或多個數組沿指定軸連接在一起，其中axis=0表沿著角點組數量拼接
counter=np.array([len(x) for x in collectIds])
#counter 的長度應該等於 len(collectIds)，也就是50幀內共幾幀捕捉到17個ArUco markers
#內容皆為17(len(x))，因為棋盤中有17個ArUco markers
caliIds=np.array(collectIds).ravel()
#collectIds種成numpy再拉平成一維數組
cameraMatrixInit = np.array([[ 1000.,    0., frameWidth/2.],[    0., 1000., frameHeight/2.],[    0.,    0.,           1.]])
#相機內參初值
distCoeffsInit   = np.zeros((5,1))
#相機畸變係數初值
ret, aruco_cameraMatrix, aruco_distCoeffs, aruco_rvects, aruco_tvects = aruco.calibrateCameraAruco(caliCorners,caliIds,count
#aruco.calibrateCameraAruco() 函數，將收集到的所資料傳遞給 ArUco 相機校正函數，以獲得最終的相機校正參數
#校正後內參矩陣(aruco_cameraMatrix)、校正後畸變係數(aruco_distCoeffs)、旋轉向量(aruco_rvects)、平移向量(aruco_tvects)
print(aruco_cameraMatrix)
print(aruco_distCoeffs)
print(aruco_rvects)
print(aruco_tvects)
```

```
[[913.81070779   0.         479.5586767 ]
 [  0.         921.0914589  293.00945991]
 [  0.           0.           1.        ]]
[[ 0.06739156]
 [-0.11176159]
 [-0.00446193]
 [-0.00266387]
 [-0.01974274]]
(array([[ 1.85560181],
       [ 2.13559441],
       [-0.43460999]]), array([[ 2.02379095],
       [ 2.16232636],
       [-0.37108907]]), array([[ 1.95358902],
       [ 2.13934218],
       [-0.43973432]]), array([[ 2.07453931],
       [ 2.14265824],
       [-0.2240849 ]]), array([[ 1.84966146],
       [ 2.40870897],
       [-0.12752218]]), array([[ 2.00997394],
       [ 2.25545981],
       [-0.31581337]]), array([[ 2.03659511],
       [ 2.15591552],
       [-0.52993239]]), array([[ 2.12746156],
       [ 2.19254096],
       [-0.09196582]]), array([[ 2.22641063],
       [ 2.13735615],
       [-0.11921934]]), array([[ 2.23525175],
       [ 2.03372319],
       [-0.02535581]]), array([[ 2.23620076],
       [ 2.09170461],
       [-0.1405798 ]]))
(array([[-12.14589974],
       [-10.64405907],
       [ 56.98317687]]), array([[-13.38017857],
       [-18.22560279],
       [ 59.1431909 ]]), array([[-10.80232673],
       [ -8.92702527],
       [ 59.25512522]]), array([[-18.97385837],
       [-17.05133588],
       [ 60.35444519]]), array([[-11.66868767],
       [-17.99299082],
       [ 58.60553023]]), array([[-14.54365133],
       [-14.41334339],
       [ 54.91649119]]), array([[-9.25622858],
       [-9.73031501],
       [52.63833396]]), array([[-20.43461122],
       [-15.20900616],
       [ 53.34331024]]), array([[-17.37367361],
       [-11.22448361],
       [ 52.50109744]]), array([[-23.96197376],
       [-10.68196653],
       [ 48.76065871]]), array([[-17.63697996],
       [ -8.88850214],
       [ 51.3847817 ]]))
```

左圖為第一部實驗影片的校正結果

```
[[997.6778458    0.         496.41956738]
 [  0.         995.7852487  190.93133326]
 [  0.           0.           1.        ]]
[[ 0.15972721]
 [-0.16508154]
 [-0.01189295]
 [-0.00359419]
 [-1.36109771]]
(array([[ 1.98954356],
       [ 2.10767634],
       [-0.21762844]]), array([[ 1.64720927],
       [ 2.35216824],
       [-0.15624993]]), array([[ 2.18065759],
       [ 2.04027487],
       [-0.02112809]]), array([[2.03523242],
       [2.13719416],
       [0.01683296]]), array([[ 2.11138357],
       [ 1.95328743],
       [-0.07911607]]), array([[ 2.24614732],
       [ 1.95297195],
       [-0.018248  ]]), array([[ 1.87965026],
       [ 2.21949084],
       [-0.09203485]]), array([[ 2.84695432],
       [ 0.98362403],
       [-0.06066175]]), array([[ 2.22813342],
       [ 1.8835504 ],
       [-0.1288941 ]]), array([[ 1.85733906],
       [ 2.22381779],
       [-0.09003627]]), array([[ 2.21886935],
       [ 1.940735  ],
       [-0.09539579]]))
(array([[-7.3242466 ],
       [-8.30330922],
       [51.77017798]]), array([[-14.90689903],
       [-11.61254708],
       [ 53.17876954]]), array([[-17.94998672],
       [ -5.81683605],
       [ 49.90069386]]), array([[-17.75057996],
       [ -9.8349794 ],
       [ 59.62703731]]), array([[-19.64547442],
       [  0.79338704],
       [ 60.65548576]]), array([[-22.5325525 ],
       [ -6.64450857],
       [ 49.69363706]]), array([[-13.70769941],
       [ -0.79726882],
       [ 75.08273209]]), array([[-14.60273265],
       [  9.92498523],
       [ 65.93662096]]), array([[-15.90787502],
       [ -2.54898129],
       [ 51.60952154]]), array([[-20.29709974],
       [ -7.74298912],
       [ 57.79095798]]), array([[-17.02574376],
       [ -6.00201386],
       [ 48.9663617 ]]))
```

左圖為第二部實驗影片的校正結果

3. 使用 calibriateCameraChAruco 做第二次相機校正
   ->通常會校正的更精準

```python
#使用calibriateCameraChAruco
caliCorners=[]
caliIds    =[]
for corners, ids, frame in zip(collectCorners,collectIds,collectFrames):
    ret, charucoCorners, charucoIds = aruco.interpolateCornersCharuco(corners,ids,frame,charucoBoard,aruco_cameraMatrix,aruc
    #interpolateCornersCharuco:用於Charuco棋盤格，用於提取Charuco棋盤格上的所有角點和標記的ID
    #類似detectMarkers的功用，但拿到的訊息更多，所以可以提高相機校正的準度
    caliCorners.append(charucoCorners)
    caliIds.append(charucoIds)

ret, charuco_cameraMatrix, charuco_distCoeffs, charuco_rvects, charuco_tvects = aruco.calibrateCameraCharuco(caliCorners,cal
print(charuco_cameraMatrix)
print(charuco_distCoeffs)
print(charuco_rvects)
print(charuco_tvects)
```

```
[[888.25457479   0.         482.34262359]
 [  0.         896.25270996 305.57029901]
 [  0.           0.           1.        ]]
[[ 0.08518861]
 [-0.41241448]
 [-0.00235481]
 [-0.00228344]
 [ 1.0810133 ]]
(array([[ 1.86281848],
       [ 2.14235737],
       [-0.42323125]]), array([[ 2.03014216],
       [ 2.16949307],
       [-0.36027035]]), array([[ 1.96162848],
       [ 2.14740957],
       [-0.43240375]]), array([[ 2.0803946 ],
       [ 2.14909494],
       [-0.21471339]]), array([[ 1.84935369],
       [ 2.40862702],
       [-0.12541285]]), array([[ 2.01467574],
       [ 2.26127206],
       [-0.30519933]]), array([[ 2.04292273],
       [ 2.16299229],
       [-0.51617167]]), array([[ 2.13256338],
       [ 2.19833919],
       [-0.08449552]]), array([[ 2.23040726],
       [ 2.1419325 ],
       [-0.11295605]]), array([[ 2.23796388],
       [ 2.03550134],
       [-0.0238541 ]]), array([[ 2.24034167],
       [ 2.0957615 ],
       [-0.13390518]]))
(array([[-12.28808103],
       [-11.36223488],
       [ 55.38342961]]), array([[-13.51937209],
       [-18.97283796],
       [ 57.46384517]]), array([[-10.94765357],
       [ -9.67123961],
       [ 57.65357435]]), array([[-19.11250761],
       [-17.81793339],
       [ 58.60352057]]), array([[-11.82090471],
       [-18.72360157],
       [ 57.01211212]]), array([[-14.67393136],
       [-15.11042308],
       [ 53.36131391]]), array([[ -9.39102837],
       [-10.40138054],
       [ 51.20317467]]), array([[-20.56317736],
       [-15.90156477],
       [ 51.80201864]]), array([[-17.50117402],
       [-11.9044849 ],
       [ 51.00708341]]), array([[-24.0879699 ],
       [-11.31379008],
       [ 47.38536377]]), array([[-17.76459883],
       [ -9.55203081],
       [ 49.92334468]]))
```

左圖為第一部實驗影片的校正結果

```
[[1.08315986e+03 0.00000000e+00 4.60856255e+02]
 [0.00000000e+00 1.06930223e+03 1.45019631e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
[[ 0.30450798]
 [-2.46275413]
 [-0.0185546 ]
 [-0.01064879]
 [ 9.68481597]]
(array([[ 1.98256887],
       [ 2.11072014],
       [-0.22941994]]), array([[ 1.62621028],
       [ 2.3389176 ],
       [-0.1794903 ]]), array([[ 2.16021593],
       [ 2.03172843],
       [-0.01879415]]), array([[2.01591067],
       [2.13005345],
       [0.01906981]]), array([[ 2.08444276],
       [ 1.94100264],
       [-0.10406892]]), array([[ 2.2365219 ],
       [ 1.95527225],
       [-0.013473  ]]), array([[ 1.86661483],
       [ 2.21615948],
       [-0.11791923]]), array([[ 2.90711393],
       [ 1.00812615],
       [-0.09380966]]), array([[ 2.21598699],
       [ 1.88257553],
       [-0.14023291]]), array([[ 1.84057064],
       [ 2.22007528],
       [-0.09413467]]), array([[ 2.20517463],
       [ 1.93861956],
       [-0.09730537]]))
(array([[-5.48450129],
       [-5.95680714],
       [56.00064712]]), array([[-12.97707628],
       [ -9.1937836 ],
       [ 57.54797917]]), array([[-16.06208737],
       [ -3.5289196 ],
       [ 53.82347005]]), array([[-15.4682657 ],
       [ -7.11267616],
       [ 64.1562243 ]]), array([[-17.38109227],
       [  3.63161995],
       [ 65.65934973]]), array([[-20.63617419],
       [ -4.37952499],
       [ 53.64950859]]), array([[-10.95807136],
       [  2.73889513],
       [ 81.20355545]]), array([[-12.47038877],
       [ 13.1789378 ],
       [ 72.48278414]]), array([[-14.02473234],
       [ -0.1545249 ],
       [ 55.83774945]]), array([[-18.10655671],
       [ -5.12504681],
       [ 62.35620689]]), array([[-15.22115959],
       [ -3.77511013],
       [ 52.88366436]]))
```

左圖為第二部實驗影片的校正結果

4. 抓到要被貼上的 youtube 影片

```python
#抓影片
urls = [
    "https://youtu.be/CWp2ZDMWn2g",
    "https://youtu.be/BfyH4ROtFm0",
    "https://youtu.be/o1WXFxHXieM",
    "https://youtu.be/lzyDD8bMDKs",
    "https://youtu.be/Zs1sHB6DNR4",
    "https://youtu.be/v2HN4gd66nM"
]
videos = []
# 遍歷6個url，依次擷取最佳解析度和編碼格式，並創建影片對象
for url in urls:
    video = pafy.new(url, basic=False, gdata=False) #創建pafy對象
    #basic：如果設置為True，則只擷取基本元數據（如影片標題、作者、影片ID等）
    #basic：如果設置為False，還會擷取更詳細的數據（如影片描述、上傳日期、喜歡和不喜歡的數量、影片解析度、時長等）
    #gdata：如果設置為True，則從gdata API(google提供的，但只要用yt的話，YouTube API會載方便)擷取數據，否則從YouTube API擷取數據
    best = video.getbest(preftype="mp4") #擷取最佳的影片解析度和編碼格式
    cap_yt = cv2.VideoCapture(best.url)
    videos.append(cap_yt)
```

5. 針對影片的每幀，偵測 ArUco marker

```
#2d
cap = cv2.VideoCapture('arUco_marker.mp4')
markerSize  = 6 #6cm

frameWidth    = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))//2
frameHeight   = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))//2

arucoParams = aruco.DetectorParameters_create()
arucoParams.cornerRefinementMethod = aruco.CORNER_REFINE_SUBPIX
arucoDict   = aruco.Dictionary_get(aruco.DICT_7X7_50)

print('height {}, width {}'.format(cap.get(cv2.CAP_PROP_FRAME_HEIGHT),cap.get(cv2.CAP_PROP_FRAME_WIDTH)))

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.resize(frame,(frameWidth,frameHeight))
    # 將圖像轉換為灰度圖像並進行二值化處理
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 51, 1)
    #自適應二值化，使用到高斯濾波去除圖像中的噪聲，提高算法的準確性
    #255:當像素值超過閾值時，將被賦予的值
    #cv2.ADAPTIVE_THRESH_GAUSSIAN_C:自適應方法，對光照變化更好
    #cv2.THRESH_BINARY:二值化類型
    #25:指定區域的大小，必須為奇數
    #1:從均值或加權平均值中減去的常數值
    (corners, ids, rejected) = aruco.detectMarkers(thresh, arucoDict, parameters=arucoParams,cameraMatrix=charuco_cameraMatr
    #加了校正後的參數進行ArUco marker檢測
```

29 行的 aruco.detectMarkers 有加上兩個相機校正的參數 cameraMatrix = charuco_cameraMatrix, distCoeff = charuco_distCoeffs，提高檢測的準確性。

6. 進行貼圖及顯示檢測到的邊框、姿態等

```
    if len(corners) > 0:
        aruco.drawDetectedMarkers(frame, corners, ids)

        for i, corner in enumerate(corners):
            for pt in corner[0]:
                cv2.drawMarker(frame,tuple(pt.astype(int).ravel()),(0,0,255),cv2.MARKER_CROSS,20,2)
                #在每個標記的四個角上繪制紅色的十字形標記

            idd = ids[i] #拿到當前corner的id
            ret, yt_frame = videos[int(idd)-1].read() #-1是因為videos從0開始，corners從1開始
            if ret == False: #影片沒了就跳出
                break
            wid = yt_frame.shape[1]
            high = yt_frame.shape[0]

            pts = np.array(corner[0], np.float32) #拿到一組四個角點
            #製造四個角點從左上方開始順時鐘排列
            # 對數組進行排序，先按y再按x
            pts = pts[np.argsort(pts[:, 1])] #照pts中的第二元素升序排列
            if pts[0][0] > pts[1][0]:
                pts_copy = pts.copy()
                pts[0], pts[1] = pts_copy[1], pts_copy[0] #互換
            if pts[2][0] < pts[3][0]:
                pts_copy = pts.copy()
                pts[2], pts[3] = pts_copy[3], pts_copy[2]

            #獲得要貼過去的yt影片角點，從左上方開始順時鐘排列
            srcp=np.array([[0, 0], [wid, 0], [wid, high], [0, high]]).astype('float32')

            # 計算兩組4個點之間的透視變換矩陣（3x3）
            M = cv2.getPerspectiveTransform(srcp, pts)
            #圖像幾何變換函數，用於實現透視變換
            warped_img = cv2.warpPerspective(yt_frame,M, frame.shape[:2][::-1])#參數:待處理的原始圖像、變換矩陣M、變換後圖像的大
            #取代目標圖中的不規則四邊形區域
            #做出一個遮罩，在需要被換掉的像素填上255，其餘為0
            mask = np.zeros(frame.shape[:2], dtype=np.uint8)
            cv2.fillPoly(mask, [pts.astype(np.int32)], 255)
            frame[mask != 0] = warped_img[mask != 0] #mask中不為0的像素換成yt影片

        rvects, tvects, _ = aruco.estimatePoseSingleMarkers(corners, markerSize, charuco_cameraMatrix, charuco_distCoeffs)
        #從檢測到的單個標記（marker）的角點坐標計算其相對於相機坐標系的旋轉向量和平移向量
        #corners是檢測到的角點坐標
        #Marker尺寸
        #cameraMatrixInit是相機的內參矩陣
        #distCoeffsInit是相機的畸變參數
        for rvec,tvec in zip(rvects,tvects):
            aruco.drawAxis(frame, charuco_cameraMatrix, charuco_distCoeffs, rvec, tvec, markerSize/2)
            #繪製標記的坐標系，以可視化其方向
```

7. Show 出畫布

```
    cv2.imshow('for Q1 and Q2',frame)
    if cv2.waitKey(20) != -1:
        break

cv2.destroyAllWindows()
cap.release()
```

```
height 1080.0, width 1920.0
```

8. 兩種關閉方式:
   a. 影片播完自動關閉
   b. 按下 ESC 鍵

- 三維做法之想法，演算步驟

1. 與二維的差異處:

   原本是在 corners 的 for 迴圈中實作，三維化改成在 zip(rvects, tvects) 的 for 迴圈中實作。角點的拿取從 corner 變成 proj_pt_with_dist，然而 proj_pt_with_dist 的形狀是(4,1,2)，因此需要 reshape 成(4,2)，其他貼圖過程均沒有改變。另外 cv2.projectPoints 所需的參數 markerCorners3D 需要事先定義，並且調整其值就可以做出各種 3D 效果。

   markerSize 在程式碼中定義為 6，又(0,0)在 marker 正中心，所以 marker x、y 軸的座標範圍在 3~-3。座標點依序為:左下、右下、右上、左上。

```
print('height {}, width {}'.format(cap.get(cv2.CAP_PROP_FRAME_HEIGHT),cap.get(cv2.CAP_PROP_FRAME_WIDTH)))
markerCorners3D = np.array([[3,0,0],[-3,0,0],[-3,-3,6],[3,-3,6]],dtype=float) #調整顯示的座標
while True:
```

```
    if len(corners) > 0:

        rvects, tvects, _ = aruco.estimatePoseSingleMarkers(corners, markerSize, charuco_cameraMatrix, charuco_distCoeffs)
        #從檢測到的單個標記（marker）的角點坐標計算其相對於相機坐標系的旋轉向量和平移向量
        #corners是檢測到的角點坐標
        #Marker尺寸
        #cameraMatrixInit是相機的內參矩陣
        #distCoeffsInit是相機的畸變參數
        for i, (rvec, tvec) in enumerate(zip(rvects, tvects)):
            proj_pt_with_dist,_ = cv2.projectPoints(markerCorners3D, rvec, tvec, charuco_cameraMatrix, charuco_distCoeffs)
            #使用 cv2.projectPoints() 函數將每個標記的三維座標投影到二維圖像平面上

            idd = ids[i] #拿到當前corner的id
            ret, yt_frame = videos[int(idd)-1].read() #-1是因為videos從0開始，corners從1開始
            if ret == False: #影片沒了就跳出
                break
            wid = yt_frame.shape[1]
            high = yt_frame.shape[0]

            pts = np.array(proj_pt_with_dist, np.float32)
            pts = pts.reshape((4, 2)) #拿到一組四個角點
            #裝這四個角點從左上方開始順時鐘排列
            # 對數組進行排序，先排y再排x
            pts = pts[np.argsort(pts[:, 1])] #照pts中的第二元素升序排列
            if pts[0][0] > pts[1][0]:
                pts_copy = pts.copy()
                pts[0], pts[1] = pts_copy[1], pts_copy[0] #互換
            if pts[2][0] < pts[3][0]:
                pts_copy = pts.copy()
                pts[2], pts[3] = pts_copy[3], pts_copy[2]

            #獲得要貼過去的yt影片角點，從左上方開始順時鐘排列
            srcp=np.array([[0, 0], [wid, 0], [wid, high], [0, high]]).astype('float32')

            # 計算兩組4個點之間的透視變換矩陣（3x3）
            M = cv2.getPerspectiveTransform(srcp, pts)
            #圖像幾何變換函數，用於實現透視變換
            warped_img = cv2.warpPerspective(yt_frame,M, frame.shape[:2][::-1])#參數:待處理的原始圖像、變換矩陣M、變換後圖像的大
            #取代目標圖中的不規則四邊形區域
            #做出一個遮罩，在需要被換掉的像素填上255，其餘為0
            mask = np.zeros(frame.shape[:2], dtype=np.uint8)
            cv2.fillPoly(mask, [pts.astype(np.int32)], 255)
            frame[mask != 0] = warped_img[mask != 0] #mask中不為0的像素換成yt影片
```

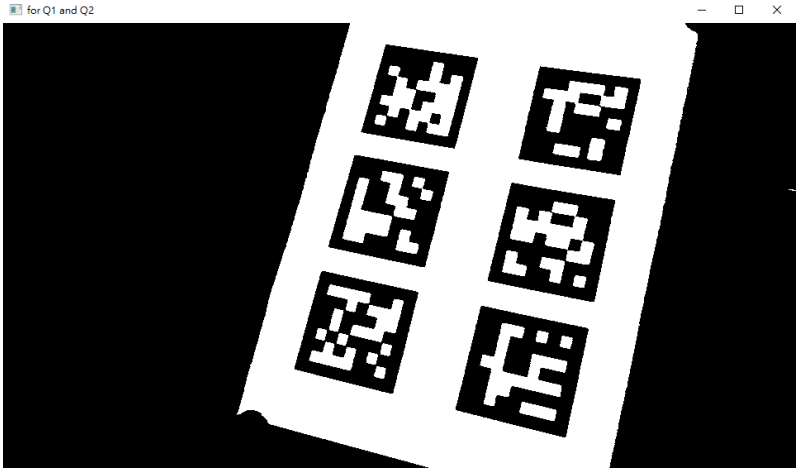
- 重要程式片段說明

1. 增強影像辨識

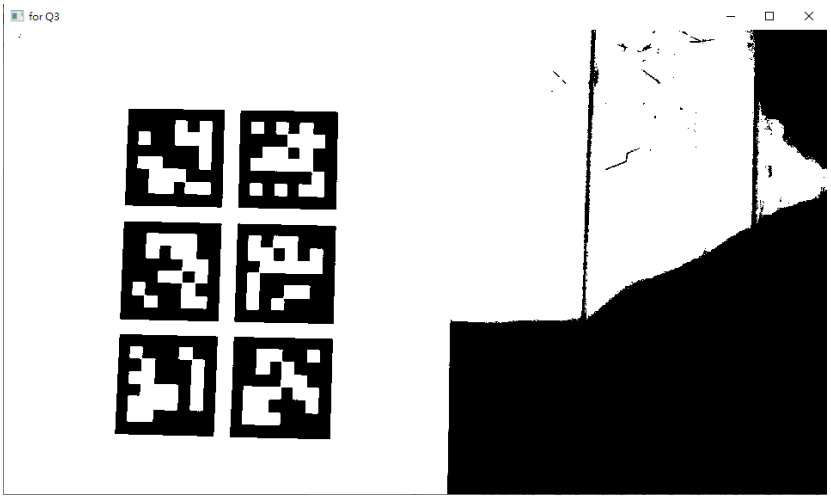
   在偵測 ArUco marker 時，經常出現明明 marker 完整卻沒偵測到的情況，因此使用二值化影像處理將圖像增強，黑白分明的圖可以很好的偵測 marker。cv2.THRESH_BINARY 方法是當像素值超過閥值時，會被歸類在白色，反之為黑色。嘗試過兩種二值化法:

   a. threshold

      thr, thresh = cv2.threshold(src, thresh, maxval, cv2.THRESH_BINARY) 簡單的全局二值化，其中 thresh 閥值調了很久，且換成不同影片時，又要找新的閥值。當 thresh 變得更高時，將有更多的像素被歸為黑色，比較多像素值會被歸類在黑色。Q3 使用的影片，陰影比 Q1、Q2 的還重，所以 Q3 的閥值會比較小，更多的像素值應歸類在白色。



第一部實驗影片 :thresh=115 效果



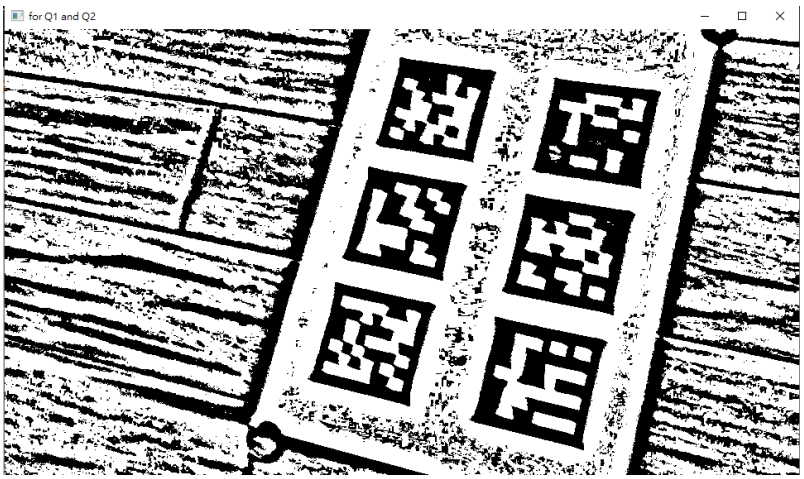
第二部實驗影片 :thresh=95 效果

b. adaptiveThreshold

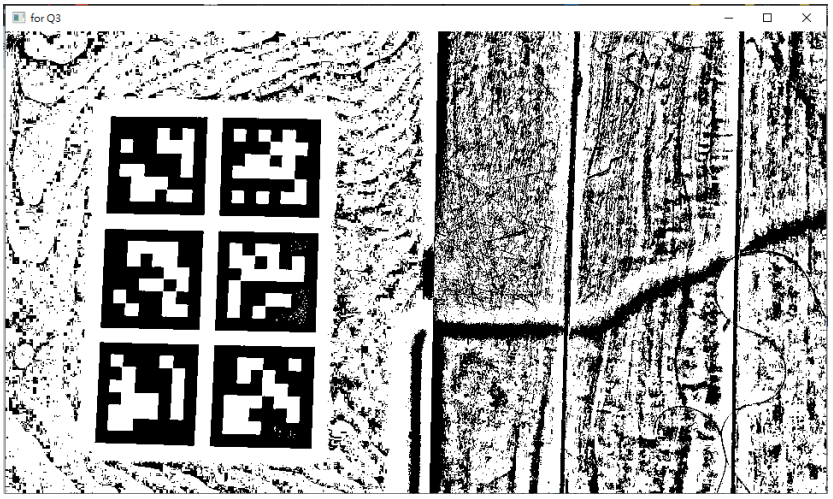
thresh = cv2.adaptiveThreshold(src, maxval, adaptiveMethod, cv2.THRESH_BINARY, blockSize, C) 自適應二值化，使用到高斯濾波去除圖像中的噪聲，提高算法的準確性。

參數 blockSize 指定了局部區域的大小，blockSize 越大時，算法就會考慮更廣泛的區域進行二值化，因此對圖像的整體光照變化較不敏感，因此可能會導致細節部分的丟失，因為兩部影片拍起來都時亮時暗的，所以這裡的值我給的很大。

參數 C 控制閾值的調整程度。正負號代表了二值化閾值的偏移方向，C 為正數時，閾值會相應地減小，從而得到更多的白色區域，反之當 C 為負數時，閾值會相應地增大，從而得到更多的黑色區域。



第一部實驗影片:blockSize=51, C=1 效果



第二部實驗影片:blockSize=51, C=1 效果

```python
    frame = cv2.resize(frame,(frameWidth,frameHeight))
    # 將圖像轉換為灰度圖像並進行二值化處理
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 51, 1)
    #自適應二值化，使用到高斯濾波去除圖像中的噪聲，提高算法的準確性
    #255:當像素值超過閾值時，將被賦予的值
    #cv2.ADAPTIVE_THRESH_GAUSSIAN_C:自適應方法，對光照變化更好
    #cv2.THRESH_BINARY:二值化類型
    #25:指定區域的大小，必須為奇數
    #1:從均值或加權平均值中減去的常數值
    (corners, ids, rejected) = aruco.detectMarkers(thresh, arucoDict, parameters=arucoParams,cameraMatrix=charuco_cameraMatr
    #加了校正後的參數進行ArUco marker檢測
```

2. 實做貼圖方法

由於 corners 給的一組角點順序會不一樣，所以要依照他們的 ids 去分配要放的 youtube 影片。

將獲得的兩組(ArUco marker、youtube 影片)四個角點照左上角開始順時針排序，如此不管拍攝角度如何旋轉都可以看到影片是正的，這點可以在第二部實驗影片中看到。getPerspectiveTransform 函式可以將這兩組角點做對應。warpPerspective 函式會做出與原影像同大小但 ArUco marker 被替換成 youtube 影片。

最後做一個 mask 標是要被換掉的區域為 255，其餘為 0，當原影像看到 mask 為 255 時，去找 warpPerspective 要填入的像素值。

```python
        for i, corner in enumerate(corners):
            for pt in corner[0]:
                cv2.drawMarker(frame,tuple(pt.astype(int).ravel()),(0,0,255),cv2.MARKER_CROSS,20,2)
                #在每個標記的四個角上繪制紅色的十字形標記

            idd = ids[i] #拿到當前corner的id
            ret, yt_frame = videos[int(idd)-1].read() #-1是因為videos從0開始，corners從1開始
            if ret == False: #影片沒了就跳出
                break
            wid = yt_frame.shape[1]
            high = yt_frame.shape[0]

            pts = np.array(corner[0], np.float32) #拿到一組四個角點
            #製造四個角點從左上方開始順時鐘排列
            # 對數組進行排序，先排y再排x
            pts = pts[np.argsort(pts[:, 1])] #照pts中的第二元素升序排列
            if pts[0][0] > pts[1][0]:
                pts_copy = pts.copy()
                pts[0], pts[1] = pts_copy[1], pts_copy[0] #互換
            if pts[2][0] < pts[3][0]:
                pts_copy = pts.copy()
                pts[2], pts[3] = pts_copy[3], pts_copy[2]

            #獲得要貼過去的yt影片角點，從左上方開始順時鐘排列
            srcp=np.array([[0, 0], [wid, 0], [wid, high], [0, high]]).astype('float32')

            # 計算兩組4個點之間的透視變換矩陣（3x3）
            M = cv2.getPerspectiveTransform(srcp, pts)
            #圖像幾何變換函數，用於實現透視變換
            warped_img = cv2.warpPerspective(yt_frame,M, frame.shape[:2][::-1])#參數:待處理的原始圖像、變換矩陣M、變換後圖像的大
            #取代目標圖中的不規則四邊形區域
            #做出一個遮罩，在需要被換掉的像素填上255，其餘為0
            mask = np.zeros(frame.shape[:2], dtype=np.uint8)
            cv2.fillPoly(mask, [pts.astype(np.int32)], 255)
            frame[mask != 0] = warped_img[mask != 0] #mask中不為0的像素換成yt影片
```

結果

結果影片 1: https://youtu.be/lYXdGg9OvUk

結果影片 1 前 36 秒為呈現第一、第二題的要求，而後則是第三題的部分。

結果影片 2: https://youtu.be/RtzvN_UF2k8

結果影片 2 前 36 秒為呈現第一部實驗影片在三維的效果，而後則是二部實驗影片在三維的效果。

較零散的每行程式碼意義都呈現在截圖中。

有關相機校正的參數顯示在二維做法之想法，演算步驟中。

第二部實驗影片為我印出 marker 後，自行拍攝的，有刻意凸顯出拍攝角度旋轉的情況。程式碼的差異，只有在 aruco 的字典使用上，我使用的為 DICT_6X6_250。

針對兩種二值化方法，若今天只有一部實驗影片且在光照沒有太大差異，可以使用 threshold，只有一個參數要調整。其餘情況使用 adaptiveThreshold 可能效果較好，函式本身還有高斯濾波可提高精確度。

二維使用 corners 會比較精準，三維使用 proj_pt_with_dist 在幾何形狀的表現好，但找出來的角點可能沒有直接使用 corners 來的準確，這就是我在二維、三維會用不同方法的原因，否則 proj_pt_with_dist 其實也可以做到結果影片 1 要的效果。

結果影片 2 在 1:20~1:22 出現影像扭曲情形，推測原因是座標太立所致現象，因為在測試不立起來，僅漂浮時不會出現此現象。

結論

這次的作業還滿有趣的，沒想到擴增實境可以這麼輕易地達成。看結果的時候，一直覺得我貼上去的影片抖得很誇張，但看朋友跑出來的效果卻感受不到很浮誇的抖動，再經過多方測試後，終於確定是因為我挑選的影片多為淺色系，造成視覺上抖動特別顯。

還未做二值化前，經常出現 marker 偵測不到的情況，百思不得其解該如何解決，與朋友討論才被點醒可以對每幀圖像做影像處理，前一份作業學了多種影像處理法在這時候派上用場。

將影片立起來更有擴增實境的感覺，只是 z 軸座標一直輕微晃動，讓影片看起來也不斷抖動，本來還不確定 cv2.projectPoints 的用途，在研究完三維後，就清楚知道他與直接使用 corners 的差別。

參考文獻