機器視覺作業報告

擴增實境遊戲

姓名: 王心妤
學號: 00657127
日期: 2019/06/09

方法

程式碼: https://github.com/creek0810/2019_mv_hw2

在這次的作業中，我將步驟分為 main、calc_projection 及 render 三個部分。

1. main

   此 function 主要用來讀取當前的視訊 frame、進行位置偵測、呼叫 calc_projection 與 render，及結果的顯示。

   位置偵測的部分，我使用 opencv 中的 aruco 模組，在偵測到 aruco marker 後，將這些 corner 傳入 calc_projection 找出包含 z 軸的 transformation matrix。再利用此 transformation matrix 進行圖像的 render。

```
camera_parameters = np.array([
    [1.08355894e+03, 0.00000000e+00, 5.73362291e+02],
    [0.00000000e+00, 1.08500947e+03, 3.45074914e+02],
    [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]
])
cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
out = cv2.VideoWriter('output.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 4, (width,height))

# load obj
obj = OBJ("./dog.obj", swapyz=True)
cap = cv2.VideoCapture(0)
# aruco var
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
parameters =  aruco.DetectorParameters_create()
```

將校正好的相機參數填上，並宣告之後要用到的變數

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

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict, parameters=parameters)

    if len(corners) >= 1:
        # get projection matrix
        projection = calc_projection(camera_parameters, corners[0])
        # draw aruco marker bound
        frame = aruco.drawDetectedMarkers(frame, corners)
        # render 3d model
        frame = render(frame, obj, projection, False)

    # show result
    frame = cv2.resize(frame, (width, height))
    cv2.imshow("frame", frame)
    out.write(frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

讀取視訊畫面，並進行偵測，若偵測到 aruco marker 則計算 projection matrix，並進行 render

2. calc_projection

此 function 主要利用內部參數矩陣及 homograpy，進行反推，得到真正的具備 z 軸的 transformation matrix。

我們先自己定義原始的 aruco marker 分別的位置，再算出，由原始位置，轉換到目前圖片上所經過的 matrix(T)

```
ori_corners = np.array([[
            [0, 0],
            [200, 0],
            [200, 200],
            [0, 200]
        ]])
T, mask = cv2.findHomography(ori_corners, corners[0], cv2.RANSAC)
```

而我們知道 T 為內部參數矩陣與外部參數矩陣進行 cross 的結果，為了求得外部矩陣，所以我們可以將式子寫為

$$\text{external} = camera\_parameters^{-1}T$$

```
external = np.dot(np.linalg.inv(camera_parameters), T)
```

而由於我們找出 T 的方法是從一平面，轉到另一平面，所以我們找到的 T 僅能知道 x 及 y 的訊息，缺乏 z 軸。

已知原始的 rotation matrix 為 homogenous transform，所以 rotation matrix 的三個向量將互相正交，之後我們將利用此特性找出缺乏的第三個方向向量。

又因為我們找出的 external 為估計出來的，所以不能保證 rotaion matrix 的第一個向量(R1)正交於第二個向量(R2)，所以我們將對此進行一些處理，如下圖所示

$$l = \sqrt{\|\mathbf{G}_1\|\|\mathbf{G}_2\|} \quad \mathbf{R}_1 = \frac{\mathbf{G}_1}{l}, \mathbf{R}_2 = \frac{\mathbf{G}_2}{l}, \boxed{\mathbf{t} = \frac{\mathbf{G}_3}{l}}$$

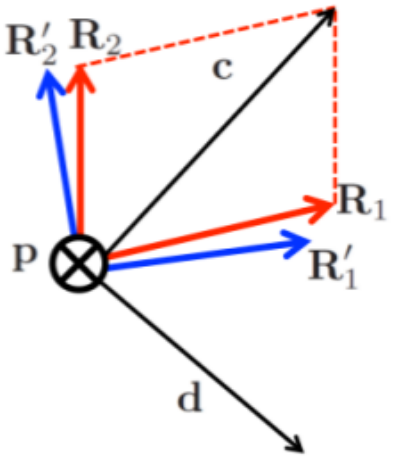
$$\mathbf{c} = \mathbf{R}_1 + \mathbf{R}_2 \,, \mathbf{p} = \mathbf{R}_1 \times \mathbf{R}_2 \,, \mathbf{d} = \mathbf{c} \times \mathbf{p}$$

$$\mathbf{R}_1' = \frac{1}{\sqrt{2}} \left(\frac{\mathbf{c}}{\|\mathbf{c}\|} + \frac{\mathbf{d}}{\|\mathbf{d}\|} \right)$$

$$\mathbf{R}_2' = \frac{1}{\sqrt{2}} \left(\frac{\mathbf{c}}{\|\mathbf{c}\|} - \frac{\mathbf{d}}{\|\mathbf{d}\|} \right)$$

$$\mathbf{R}_3 = \mathbf{R}_1' \times \mathbf{R}_2'$$

$$\boxed{\mathbf{R} = [\mathbf{R}_1' \; \mathbf{R}_2' \; \mathbf{R}_3]}$$




```
# calc external matrix and transpose
col_1 = external[:, 0]
col_2 = external[:, 1]
col_3 = external[:, 2]

# normalise vectors
l = math.sqrt(np.linalg.norm(col_1, 2) * np.linalg.norm(col_2, 2))
rot_1 = col_1 / l
rot_2 = col_2 / l
translation = col_3 / l
# compute the orthonormal basis
c = rot_1 + rot_2
p = np.cross(rot_1, rot_2)
d = np.cross(c, p)
rot_1 = np.dot(c / np.linalg.norm(c, 2) + d / np.linalg.norm(d, 2), 1 / math.sqrt(2))
rot_2 = np.dot(c / np.linalg.norm(c, 2) - d / np.linalg.norm(d, 2), 1 / math.sqrt(2))
rot_3 = np.cross(rot_1, rot_2)
```

對 R1 及 R2 進行完處理後，算出 R3。我們把 rotation matrix(處理過後的 R1、R2 及 R3)與 tranlation matrix 組合起來。將 camera_parameters 與我們算出的 transformation matrix 做 dot，得到 projection matrix。

```
projection = np.column_stack((rot_1, rot_2, rot_3, translation))

return np.dot(camera_parameters, projection)
```

3. render

此 function 主要利用算好的 projection matrix，將 3d model 貼上原圖。

我們先將 points 乘上 scale_matrix，對該 model 進行適當的縮放，並利用 perspectiveTransform 及 projection matrix 對這些 points 進行坐標轉換(貼到原圖時的位置)，最後，將算出的點圍出的區域著色(fillConvexPoly)，貼上原圖。

```
def render(img, obj, projection, model, color=False):
    """
    Render a loaded obj model into the current video frame
    """
    vertices = obj.vertices
    scale_matrix = np.eye(3) * 20
    h = 200
    w = 200

    for face in obj.faces:
        face_vertices = face[0]
        points = np.array([vertices[vertex - 1] for vertex in face_vertices])
        points = np.dot(points, scale_matrix)
        # render in the middle
        points = np.array([[p[0] + w / 2, p[1] + h / 2, p[2]] for p in points])
        dst = cv2.perspectiveTransform(points.reshape(-1, 1, 3), projection)
        imgpts = np.int32(dst)
        cv2.fillConvexPoly(img, imgpts, (137, 27, 211))

    return img
```

結果

https://www.youtube.com/watch?v=WiWQGJ9rgZA&feature=youtu.be

結論

透過實作擴增實境的遊戲，我對相機模型，及各種坐標系的轉換有了更深的理解，同時，也理解到了，線性代數的重要性(對於各個矩陣的名稱還是有點混亂，如果有寫錯的地方，還請老師指正)。
一開始有了做這個遊戲的想法後，真的很茫然，完全不清楚該怎麼實現，幸好網路上有很多的資源可以參考，才得以完成此次作品。
比較可惜的是，由於時間不夠(我自己時間規劃不夠完善)，導致沒能完成原先預期的彩色圖片功能。

參考文獻

http://www.pygame.org/wiki/OBJFileLoader
https://home.gamer.com.tw/creationDetail.php?sn=4093935
https://bitesofcode.wordpress.com/2017/09/12/augmented-reality-with-python-and-opencv-part-1/
https://bitesofcode.wordpress.com/2018/09/16/augmented-reality-with-python-and-opencv-part-2/