Problem 1

Q1-1 For each validation image, compute its camera pose with respect to world coordinate. Find the 2D-3D correspondence by descriptor matching, and solve the camera pose. Implement at least one kind of algorithm that solves a PnP problem. Briefly explain your implementation and write down the pseudo code in your report.

我主要實作 P3P+RANSAC 來計算 camera pose，方法如 pseudo code 所示。

```
def p3p_ransac(points3D,points2D, cameraMatrix, distCoeffs ,rotq_gt,tvec_gt):
    # 設定ransac參數
    max_step = 200
    camaraMatrix_inv = np.linalg.inv(cameraMatrix)
    points2D = np.append(points2D, np.ones((points2D.shape[0],1)),axis=1)
    distance = 1
    res = []
    min_r_loss = 10000000

    for i in range(max_step):
        # 隨機sample 3個點
        index_random_sample_point = np.random.choice(len(points2D), 3)
        random_sample_point_2d = points2D[index_random_sample_point]
        random_sample_point_3d = points3D[index_random_sample_point]
        # 計算sample出的3點在世界座標中的距離
        distance = np.array([np.linalg.norm(random_sample_point_3d[0] - random_sample_point_3d[1]),\
            np.linalg.norm(random_sample_point_3d[1] - random_sample_point_3d[2]),\
            np.linalg.norm(random_sample_point_3d[0] - random_sample_point_3d[2])])
        # 計算每個cosine theta
        cos_thetas = np.array([cos_theta(random_sample_point_2d[0], random_sample_point_2d[1], cameraMatrix), \
                    cos_theta(random_sample_point_2d[1], random_sample_point_2d[2], cameraMatrix),\
                    cos_theta(random_sample_point_2d[2], random_sample_point_2d[0], cameraMatrix)]).reshape(-1,3)[0]
        # 依照上課ppt之求解p3p方法，利用餘弦定理求解四次多項式之解x
        k1 = (distance[1] / distance[2]) ** 2
        k2 = (distance[1] / distance[0]) ** 2
        G = compute_G(k1,k2,cos_thetas)
        x = solve(G[4] * (x_**4) + G[3] * (x_**3) + G[2] * (x_**2) + G[1] * (x_) + G[0], x_)
        # 利用內參的inverse和sample點計算到世界座標的單位向量
        o1 = np.matmul(camaraMatrix_inv, np.transpose(random_sample_point_2d[0]))
        o2 = np.matmul(camaraMatrix_inv, np.transpose(random_sample_point_2d[1]))
        o3 = np.matmul(camaraMatrix_inv, np.transpose(random_sample_point_2d[2]))
        o1 = o1 / np.linalg.norm(o1)
        o2 = o2 / np.linalg.norm(o2)
        o3 = o3 / np.linalg.norm(o3)
        # 驗證每組解
        for sol in x:
            # 嘗試於每個解出的x中找出正確的解
            y = -(sol ** 2 - k1 - (k1 -1) * (sol**2 * (k2 - 1) - 2 * k2 * sol * cos_thetas[0] + k2)  ) / (2 * k1 * (cos_thetas[2] - sol * cos_thetas[1]))
            d1 = solve((1 + sol**2 - 2 * sol * cos_thetas[0]) * d1_ ** 2 - distance[0] ** 2, d1_)[0]
            d2 = d1 * sol
            d3 = d1 * y
            O1_temp = o1 * float(d1)
            O2_temp = o2 * float(d2)
            O3_temp = o3 * float(d3)

            norm_vec = np.cross((O2_temp - O1_temp), (O3_temp - O1_temp))
```

```
            # 利用icp求旋轉平移
            c_o = np.array([O1_temp,O2_temp,O3_temp])
            T, _ ,_= icp(random_sample_point_3d[0:3], c_o)
            T= T[0:3, :]
            # 利用得到的外在參數計算影像平面上的誤差，以利ransac運作
            l2_dis = 0
            for i in range(len(points2D)):
                a = points2D[i]
                temp = np.matmul(T, np.transpose(np.append(points3D[i],1)))
                temp = undistortion(temp, distCoeffs)
                b = np.matmul(cameraMatrix, np.transpose(temp))
                b = b / b[2]
                l2_dis +=np.linalg.norm(a-b)

            if l2_dis <min_r_loss:
                res = T
                min_r_loss = l2_dis
    # 回傳外在參數
    rotM = R.from_matrix(res[0:3, 0:3]).as_rotvec()
    return rotM, res[0:3, 3]
```

Q1-2 For each camera pose you calculated, compute the median pose error (translation, rotation) with respect to ground truth camera pose. Provide some discussion.

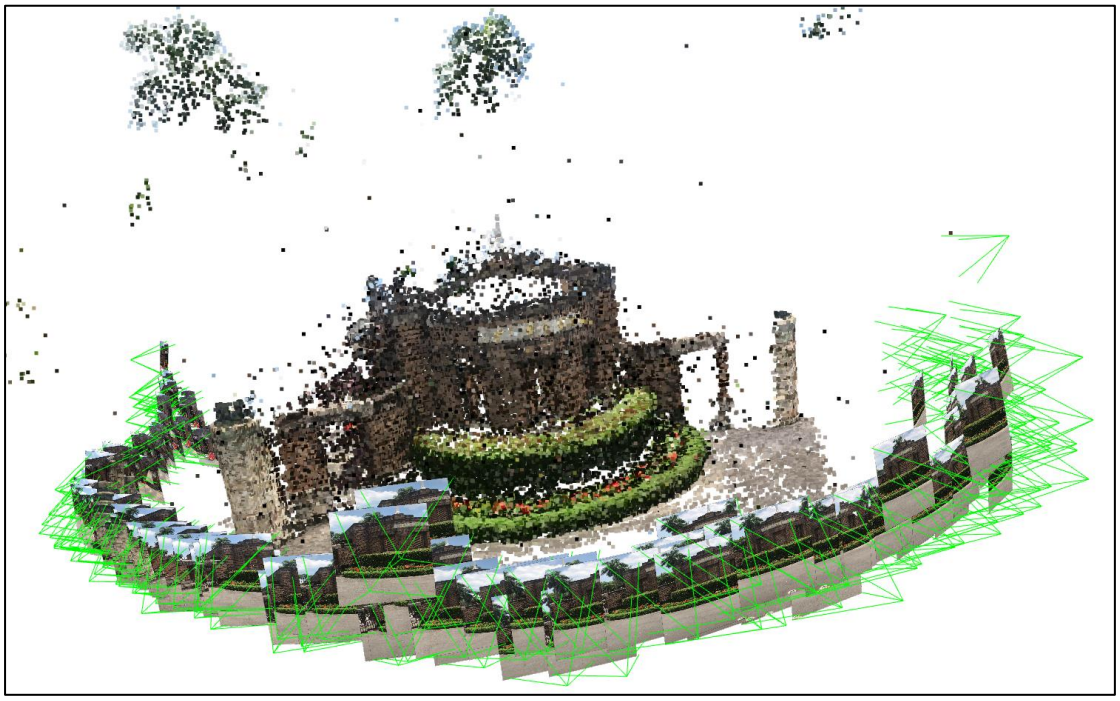
我實測了我的方法和 opencv 中的函式結果如表格所示，發現自行實作的方法與 opencv 的函式有一段落差，原因可能是 icp 的計算不夠精準。

Table of error

	rotation	translation
P3P + ransac (icp)	0.02759	0.07028
cv2.solvePnPRansac(Using P3P)	0.00628	0.00411

Q1-3 For each camera pose you calculated, plot the trajectory and camera poses along with 3d point cloud model using Open3D. Explain how you draw and provide some discussion.

這裡的方法是畫一個四邊形並將 validation set 的圖片貼在四邊形上模擬影像平面，並畫兩個三角形連接頂點組成一個金字塔，接著移動此金字塔到計算出來的位置，結果如圖所示。



Problem 2

Q2-1 With camera intrinsic and extrinsic parameters, place a virtual cube in the validation image sequences to create an Augmented Reality video. Draw the virtual cube as a point set with different colors on its surface. Implement a simply but efficient painter's algorithm to determine the order of drawing.

Demo 影片 : https://youtu.be/XUBGQ43hTVI

1. 先解壓縮: $ unzip data.zip
2. 在 data 資料夾地下建立 img_plotCube 資料夾

How to execute:
Q1-1 1-2:
$ python 2d3dmathcing.py
Q1-3:
$ python CameraPose.py
Q2:
$ python cube.py
$ python plotCube.py