Name: Melwyn D Souza
Student Number: R00209495
Date: 29/Apr/2022
Module: Machine Vision
Assignment: 2
Lecturer: Dr Christian Beder
Course: MSc in Artificial Intelligence

Task 1 (pre-processing)

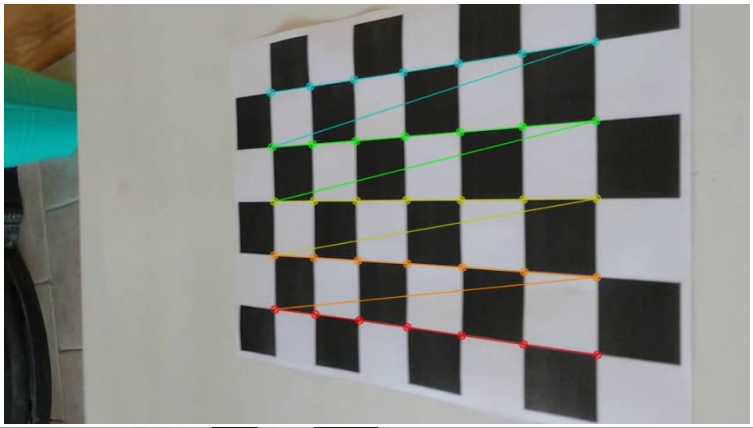
Task 1 A: Checkboard corners (subpixel accuracy)

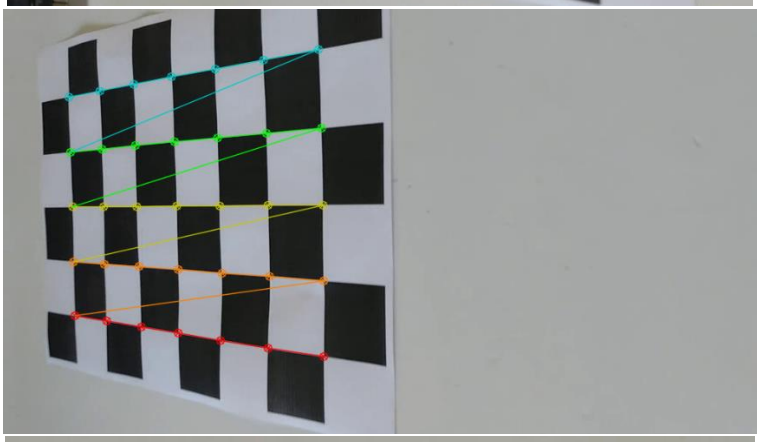
The checkboard corners are displayed to subpixel accuracy using the OpenCV tool cornweSubPix as shown below

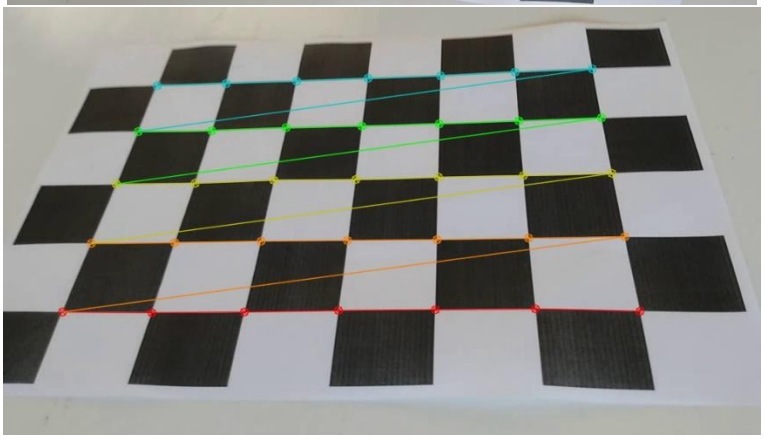
Code snippet (the line number is captured for ease of cross verification)

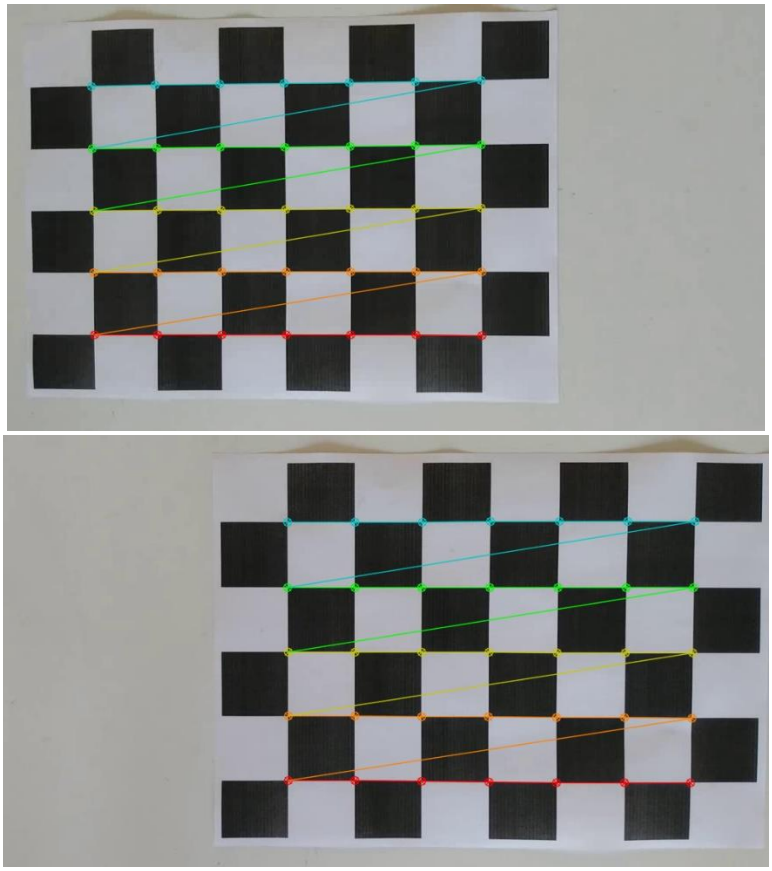
```
21      """Task 1 A - Checkboard corners with subpixel accuracy"""
22      print("TASK 1 A")
23      calib_images_loc = os.getcwd()+r'\Assignment_MV_02_calibration'
24      criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
25      for i in os.listdir(calib_images_loc):
26          rgb_image = cv2.imread(calib_images_loc+'\\'+i)
27          grey_image = cv2.cvtColor(rgb_image, cv2.COLOR_RGB2GRAY)
28
29
30          ret, corners = cv2.findChessboardCorners(grey_image, (7,5), None)
31
32          if ret:
33              image1 = copy.deepcopy(grey_image)
34              rgb_draw = copy.deepcopy(rgb_image)
35              corners2 = cv2.cornerSubPix(grey_image, corners, (11,11), (-1,-1), criteria)
36              img_pts.append(corners2)
37              obj_pts.append(cord_3d)
38
39              img = cv2.drawChessboardCorners(rgb_draw, (7,5), corners2, ret)
40              cv2.imshow("sub-pixel accuracy "+str(i), img)
41
42          cv2.waitKey(0)
43          cv2.destroyAllWindows()
44
```

Output Images as shown below:









Task 1 B:

Code Snippet

```
45      """Task 1 B - Camera Calibraton and Parameters"""
46      print("Task 1 B")
47      ret,K,d,r,t = cv2.calibrateCamera(obj_pts, img_pts, grey_image.shape[::-1], None, None)
48
49      principal_length = K[0][0] # c = f*mx
50      aspect_ratio = K[1][1]/K[0][0] # alpha = my/mx
51      principal_point = (K[0][2], K[1][2]) # x0,y0
52      skew = K[0][1]
53
54      print("Camera Calibration Matrix K:\n\n", K)
55      print("\nPrincipal Length:", principal_length)
56      print("Aspect Ratio: ", aspect_ratio)
57      print("Principal Point: ", principal_point)
58      print("Image Skew (Modern cameras skew == 0): ", skew)
59
```

Camera Calibration Matrix K:

994.92286802	0.	485.15648718
0.	953.82760941	286.17927724
0.	0.	1.

Principal Length: 994.922868018294

Aspect Ratio: 0.9586950306064169

Principal Point: (485.1564871803063, 286.1792772448046)

Image Skew (Modern cameras skew == 0): 0.0

Task 1 C: Feature Extract and KLT algorithm

There are 109 features extracted in the first frame using the OpenCV's goodFeaturesToTrack method, converted all feature points to sub-pixel accuracy

```
60      """Task 1 C - Feature tracking"""
61      video = cv2.VideoCapture('Assignment_MV_02_video.mp4')
62
63      # initialise features to track the feature points in the first frame - Done
64      ret, frame0 =video.read()
65      if ret:
66          grey_img = cv2.cvtColor(frame0, cv2.COLOR_BGR2GRAY)
67          p0 = cv2.goodFeaturesToTrack(grey_img, 200, 0.3, 7)
68          print("Total features in the first frame: {}".format(len(p0)))
69          subpixel_p0 = cv2.cornerSubPix(grey_img, p0, (11, 11), (-1, -1), criteria=criteria)
70
71          #display frame 0
72          for i in range(len(subpixel_p0)):
73              cv2.circle(frame0, (subpixel_p0[i,0,0],subpixel_p0[i,0,1]), 2, (0,0,255), 2)
74          cv2.imshow("Task 1 C Frame 0 feature points", frame0)
75          cv2.waitKey(0)
76          cv2.destroyAllWindows()
77
```

Total features in the first frame

me: 109

Task 1 D:

The KLT algorithm is used to track these features across all the frames (30 total frames, 30fps, 1 sec video length).

Code snippet

```
        """Task 1 D"""
        frame = 0
        while ret:
            ret, img = video.read()

            if not ret:
                break

            frame += 1
            old_img = grey_img
            grey_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

            if len(p0) > 0:
                p1, st, err = cv2.calcOpticalFlowPyrLK(old_img, grey_img, p0, None)

                # visualise points
                for i in range(len(st)):
                    if st[i]:
                        cv2.circle(img, (p1[i, 0, 0], p1[i, 0, 1]), 2, (0, 0, 255), 2)
                        cv2.line(img, (p0[i, 0, 0], p0[i, 0, 1]), (int(p0[i, 0, 0] + (p1[i][0, 0] -

                p0 = p1[st == 1].reshape(-1, 1, 2)
                index = index[st.flatten() == 1]

            if len(p0) < 100:
                features = cv2.goodFeaturesToTrack(grey_img, 200 - len(p0), 0.3, 7)
                new_p0 = cv2.cornerSubPix(grey_img, features, (11, 11), (-1, -1), criteria=criteria)
                for i in range(len(new_p0)):
                    if np.min(np.linalg.norm((p0 - new_p0[i]).reshape(len(p0), 2), axis=1)) > 10:
                        p0 = np.append(p0, new_p0[i].reshape(-1, 1, 2), axis=0)
                        index = np.append(index, np.max(index) + 1)

            # update tracks
            for i in range(len(p0)):
                if index[i] in tracks:
                    tracks[index[i]][frame] = p0[i]
                else:
                    tracks[index[i]] = {frame: p0[i]}

            # visualise last 20 frames of active tracks
            for i in range(len(index)):
                for f in range(frame - 20, frame):
                    if (f in tracks[index[i]]) and (f + 1 in tracks[index[i]]):
                        cv2.line(img,
                                 (tracks[index[i]][f][0, 0], tracks[index[i]][f][0, 1]),
                                 (tracks[index[i]][f + 1][0, 0], tracks[index[i]][f + 1][0, 1]),
                                 (0, 255, 0), 1)

            k = cv2.waitKey(2)
            if k % 256 == 27:
                print("Escape hit, closing...")
                break

            cv2.imshow("Feature Track Task 1 D", img)
            # cv2.waitKey()
```

Feature points are tracked across all frames and visualized as shown in the figure below


Feature Track Task 1 D

Task 2 (Fundamental Matrix)

Task 2 A:

Code Snippet

```
174         """Task 2 A - Extract feature points in frame 0 and frame 30"""
175         frame, frame0 = 0,0
176         frame30 = 30
177         images = extract_frames("Assignment_MV_02_video.mp4", [frame0, frame30])
178 
179         correspondences = [] #only common between frame 0 adn frame 30
180         for track in tracks:
181             if (frame0 in tracks[track]) and (frame30 in tracks[track]):
182                 x1 = [tracks[track][frame0][0,1],tracks[track][frame0][0,0],1]
183                 x2 = [tracks[track][frame30][0,1],tracks[track][frame30][0,0],1]
184                 correspondences.append((np.array(x1), np.array(x2)))
185                 cv2.circle(images[frame0],(tracks[track][frame0][0,0],tracks[track][frame0][0,1]), 2, (0,0,255), 2)
186                 cv2.circle(images[frame30],(tracks[track][frame30][0,0],tracks[track][frame30][0,1]), 2, (0,0,255), 2)
187 
188         cv2.imshow("Features in both 0 and 30 frame (frame 0)", images[frame0])
189         cv2.waitKey(0)
190         cv2.imshow("Features in both 0 and 30 frame (frame 30)", images[frame30])
191         cv2.waitKey(0)
192 
193         cv2.destroyAllWindows()
194 
195         print("The total feature points in frame 0 and frame 30 are:", len(correspondences))
196 
197         best_outliers = len(correspondences)+1
198         best_error = 1e100
```

Tracks visible is both first frame and last frame are extracted and the visualized as shown on the figure below, the total feature points in frame 0 and frame 30 are: 69

The common features in both first and last frame are displayed below

First frame common feature points shown in the figure below

Last Frame common feature points as shown in the figure below

Task 2 B:

Mean, Standard deviation and T matrices are calculated as shown in the code snippet below

```
200
201     """Task 2 B"""
202     #Mean, standard deviation and T
203
204     x1_mean = np.mean(np.array(correspondences)[:, 0, :2], axis=0)
205     x2_mean = np.mean(np.array(correspondences)[:, 1, :2], axis=0)
206
207     x1_std = np.std(np.array(correspondences)[:, 0, :2], axis=0)
208     x2_std = np.std(np.array(correspondences)[:, 1, :2], axis=0)
209
210
211     row1 = [1/x1_std[1], 0, -x1_mean[1]/x1_std[1]]
212     row2 = [0, 1/x1_std[0], -x1_mean[0]/x1_std[0]]
213     row3 = [0,0,1]
214     T1 = np.array([row1,row2,row3])
215
216     row1 = [1/x2_std[1], 0, -x2_mean[1]/x2_std[1]]
217     row2 = [0, 1/x2_std[0], -x2_mean[0]/x2_std[0]]
218     row3 = [0,0,1]
219     T2 = np.array([row1,row2,row3])
220
221     print("x1 Mean\n:",x1_mean)
222     print("x2_Mean\n:",x2_mean)
223     print("x1_std\n:",x1_std)
224     print("x2_std\n:",x2_std)
225     print("T1\n\n", T1)
226     print("T2\n\n", T2)
227
228
229     y1 = (np.matmul(T1,np.array(correspondences)[:, 0, :].T)).T
230     y2 = (np.matmul(T2,np.array(correspondences)[:, 1, :].T)).T
231     ind = [i for i in range(len(correspondences))]
232
```

The output from the script run for mean, std and T matrices is as follows

x1 Mean: [270.99616319 525.99152894]

x2 Mean: [245.30862106 534.25493345]

x1 std: [67.80714891 153.9172723]

x2 std: [87.9315564 238.93095979]

T1

0.006497	0.	-3.41736519
0.	0.01474771	-3.99657215
0.	0.	1.

T2

0.00418531	0.	-2.23602221
0.	0.01137248	-2.78976776
0.	0.	1.

Task 2 C/D/E/F/G:

The first fundamental matrix:

-4.97272337e-06	2.16721056e-05	-5.70188753e-03
-3.22867383e-05	-4.18846201e-06	1.36779929e-02
1.26530715e-02	-7.09512992e-03	2.30353313e+00

Running 10,000 loops to find the least number of outliers and highest inliers, also to find the lowest test statistics value, the code snippet is as shown below

```
    """Task 2 Fundamental Matrix """
    for i in range(10000):  #change to 10000

        inliers, inlier_index = [], []
        count_outliers, count_inliers  = 0,0
        accumulate_error = 0

        """Task 2 C"""

        samples_in = random.sample(ind, 8)
        samples_out = list(set(ind) - set(samples_in))
        A = np.zeros((0,9)) # for 8 pt DLT

        for y11, y22 in zip(y1[samples_in, :], y2[samples_in, :]):
            ai = np.kron(y11.T, y22.T)
            A = np.append(A, [ai], axis=0)

        """Task 2 D"""

        U,S,V = np.linalg.svd(A)
        Fcap = V[8,:].reshape(3,3).T

        U,S,V = np.linalg.svd(Fcap)

        #converting F to singular by dividing it with s[2]
        Fcap = np.matmul(U,np.matmul(np.diag([S[0],S[1],0]),V))
        F = np.matmul(T2.T, np.matmul(Fcap, T1))

        """Task 2 E"""
        cxx = np.array([[1, 0, 0],
                        [0, 1, 0],
                        [0, 0, 0]])

        for i in samples_out:                           #For remainder of points find model eq
            x1, x2 = correspondences[i]
            # gi = np.matmul(x2.T, np.matmul(F, x1))
            gi = x2.T @ F @ x1
            varianceSigma = np.matmul(x2.T, np.matmul(F, np.matmul(cxx, np.matmul(F.T, x2)))) +

            """Task 2_F"""
            Ti = gi**2/varianceSigma

            if Ti > 6.635: #outliers
                count_outliers += 1
            else: #inliers
                count_inliers += 1
                inliers.append((x1,x2))
                inlier_index.append(i)
                accumulate_error += Ti

        """Task 2 G"""
        if count_outliers<best_outliers:
            best_error = accumulate_error
            best_outliers_count = count_outliers
            best_inliers_count = count_inliers
            best_inliers_cors = inliers
            best_inliers_index = inlier_index
            best_F = F
        elif count_outliers==best_outliers:
            if accumulate_error<best_error:
                best_error = accumulate_error
                best_outliers_count = count_outliers
                best_inliers_cors = inliers
                best_inliers_index = inlier_index
```

Best Fundamental Matrix from best 8 samples after running 10,000 loops:

[[8.14409580e-06 2.84977558e-05 -1.18626803e-02]

[-2.17872077e-05 1.09711314e-06 6.18835588e-03]

[3.70371030e-03 -9.03106701 e-03 1.86973760e+00]]

Sum of inlier test statistics (accumulated_error): 48.96064022422926

Total number of inliers: 27

Total number of outliers: 34

In the figure below, the points in green are inliers and the points in red are outliers

Task 2 H:

Epipoles:

E1: [0.67311282 0.73953642 0.0022387]

E2: [0.73248212 0.68078206 0.00239407]

E1/E1[2]: [300.67082673 330.34139122 1.]

E2/E2[2]: [305.95706555 284.36199955 1.]

```
338
339         images = extract_frames("Assignment_MV_02_video.mp4", [frame0, frame30])
340         U,S,V = np.linalg.svd(F)
341         e1 = V[2,:]
342         U,S,V = np.linalg.svd(F.T)
343         e2 = V[2,:]
344
345         print("Epipoles:\n")
346         print(e1)
347         print(e2)
348         print(e1/e1[2])
349         print(e2/e2[2])
350
351         cv2.circle(images[0], (int(e1[0]/e1[2]),int(e1[1]/e1[2])), 3, (0,0,255), 2)
352         cv2.imshow('Epipoles F0', images[0])
353         cv2.waitKey(0)
354
355         cv2.circle(images[30], (int(e2[0]/e2[2]),int(e2[1]/e2[2])), 3, (0,0,255), 2)
356         cv2.imshow('Epi poles F30', images[30])
357         cv2.waitKey(0)
358
359         cv2.destroyAllWindows()
```

Epipole frame 0,

Epipole frame 30,


Epi poles F30

Task 3 (Essential Matrix)

Task 3 A

```
365     """Task 3 A"""
366     E = np.matmul(K.T, np.matmul(F,K)) #typically K matrices do not change so I assume K==K'
367     print("Essential Matrix:\n\n",E)
368     U, S, V = np.linalg.svd(E)
369
370     print("Singular values before:\n", S)
371
372     if np.linalg.det(U) < 0:
373         U[:,2] *= -1
374     if np.linalg.det(V) < 0:
375         V[2,:] *= -1
376
377     mean = (S[0]+S[1])/2
378
379     E = np.matmul(U,np.matmul(np.diag([mean, mean, 0]),V.T))
380
381
382     print("New Essential Matrix:\n\n",E)
383
```

Essential Matrix: Before making the S matrix singular

[[8.06160843 27.04393992 0.2427092]

[-20.67573117 0.99813939 -3.88005458]

[1.41261522 4.87289075 0.03632283]]

Singular values before:

[2.94948529e+01 1.98936072e+01 7.37054109e-16]

New Essential Matrix:

[[17.91994871 -15.95985371 3.84740011]

[16.6299219 18.08411864 -2.49263006]

[3.25931567 -2.84013651 0.6882211]]

Task 3 B:

The four solutions are found as shown below in the code snippet

```
    """Task 3 B"""

    Z = np.array([[0, 1, 0],
                  [-1, 0, 0],
                  [0, 0, 0]])

    W = np.array([[0, -1, 0],
                  [1, 0, 0],
                  [0, 0, 1]])

    car_speed = 50000 #50km/h
    hour = 60*60 #3600 seconds
    fps = 30 #30fps
    videolenght = 1 #video length
    beta = (car_speed/hour)*fps*videolenght

    #four possible solutions
    srtt1  = beta*np.matmul(U,np.matmul(Z,U.T))
    srtt2  = -(beta*np.matmul(U,np.matmul(Z,U.T)))
    rtt1 = np.array([srtt1[2, 1], srtt1[0, 2], srtt1[1, 0]])
    rtt2 = np.array([srtt2[2, 1], srtt2[0, 2], srtt2[1, 0]])
    rotation1 = np.matmul(U,np.matmul(W,V.T))
    rotation2 = np.matmul(U,np.matmul(W.T,V.T))
    translation1 = np.matmul(np.linalg.inv(rotation1), rtt1)
    translation2 = np.matmul(np.linalg.inv(rotation2), rtt2)

    print("Rotation matrx 1:\n", rotation1)
    print("Rotation matrx 2:\n", rotation2)
    print("Translation matrx 1:\n", translation1)
    print("Translation matrx 2:\n", translation2)
```

Rotation matrx 1:

[[-0.65421489 -0.75259862 -0.07482108]

[0.73766959 -0.65678951 0.15643249]

[-0.16687257 0.04714723 0.98485059]]

Rotation matrx 2:

[[0.67181547 0.68841636 -0.27339876]

[-0.73748341 0.65611058 -0.16011596]

[0.06915337 0.30919543 0.94848089]]

Translation matrx 1:

[20.68578765 -75.4327537 -409.25921977]

Translation matrx 2:

[-20.68578765 75.4327537 409.25921977]

Task 3 C:

```
        """Task 3 C"""
        solution_count =[]
        solution_corr_dict = {}
        i = 0
        for solution in solutions:
            t = solution[0]
            R = solution[1]
            count = 0
            pointsin3d = []
            for correspondence in inlier_correspondces: #inliers
                x1,x2 = correspondence[0], correspondence[1]

                m1 = np.matmul(np.linalg.inv(K), x1)
                m2 = np.matmul(np.linalg.inv(K), x2)

                m1 = np.array(m1)
                m2 = np.array(m2)

                m1Tm1 = np.matmul(m1.T, m1)
                m2Tm2 = np.matmul(m2.T, m2)
                m1TRm2 = np.matmul(m1.T, np.matmul(R,m2))
                tTm1 = np.matmul(t.T,m1)
                tTRm2 = np.matmul(t.T,np.matmul(R,m2))


                lambda_Mue = np.linalg.solve([[m1Tm1,-m1TRm2],[m1TRm2,-m2Tm2]], [tTm1,tTRm2])

                if (np.all(lambda_Mue>0)):
                    count += 1
                    xlamda = lambda_Mue[0] * m1
                    xMue = t + np.multiply(lambda_Mue[1], np.matmul(R, m2))
                    pointsin3d.append([xlamda,xMue])
            solution_corr_dict[i] = pointsin3d
            solution_count.append(count)
            i += 1


        bestSolIndex = np.argmax(solution_count)
```

The best solution is,

Translation; [-20.68578765, 75.4327537 , 409.25921977]

Rotation:

[[0.67181547, 0.68841636, -0.27339876],

[-0.73748341, 0.65611058, -0.16011596],

[0.06915337, 0.30919543, 0.94848089]]

The total points infront of both frames from the best soution are (the best inliers): 13

Task 3 D/E: 3d plots

```
460         """Task 3 D E"""
461         ax = Axes3D(plt.figure())
462         ax.set_xlabel('X')
463         ax.set_ylabel('Y')
464         ax.set_zlabel('Z')
465         best3dcors = solution_corr_dict[bestSolIndex]
466         x, y, z = np.array(best3dcors)[:, 0, 0], np.array(best3dcors)[:, 0, 1], np.array(best3dcors)[:, 0, 2]
467         ax.scatter3D(x, y, z, marker='o', c='green')
468 
469         ax.plot([0.], [0.], [0.], marker='+', c='red')
470 
471         ax.plot(solutions[bestSolIndex][0][0], solutions[bestSolIndex][0][1], solutions[bestSolIndex][0][2], marker='o', c='blue')
472 
473         plt.show()
```