TD05 : Solution EXERCICE 3 / 3 (RANSAC)

```
###################################
#REFERENCE IMAGE
image_ref = cv2.imread('reference_image_logo.png')
mask=cv2.imread('reference_image_logo_mask.png',0) #0: pour un channel
detector = cv2.xfeatures2d.SURF_create()
kp_ref, ds_ref = detector.detectAndCompute(image_ref,mask)
###################################
#INITIAL POSITION OF THE VIRTUAL OBJECT
corners=helper.keypoints2numpyarray(kp_ref)
K=helper.create_K()
corners_3d=helper.pixels2camera(corners,K,Z=500)
square_3d=helper.create_square(corners_3d,size=60)
square_3d_h=helper.coord2homogeneous(square_3d)

###################################
#NEW POSE
for i in range(1,14):
#for i in range(10,11):
    #Current image
    image_current = cv2.imread('pose_logo_'+str(i)+'.png')
    #DESCRIPTORS
    keypoints, descriptors = detector.detectAndCompute(image_current,None)
    #MATCHING
    bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
    matches = bf.match(descriptors,ds_ref)
    matches = sorted(matches, key = lambda x:x.distance)
    #SUBSET OF MATCHING KEYPOINTS (FROM REF AND CURRENT)
    kp_ref_subpart=[]
    kp_current=[]
    for m in matches[0:16]:
        tmp_kp=kp_ref[m.trainIdx]
        kp_ref_subpart+=[tmp_kp]
        tmp_kp=keypoints[m.queryIdx]
        kp_current+=[tmp_kp]
    corners_2d=helper.keypoints2numpyarray(kp_ref_subpart)
    corners_3d=helper.pixels2camera(corners_2d,K,Z=500)
    corners2d_current=helper.keypoints2numpyarray(kp_current)

    #TRANSFORMATION (POSE) ESTIMATION: 3D CORNERS IN REF IMAGE TO 2D CORNERS IN CURRENT IMAGE
    M_ext=helper.compute_extrinsic_matrix(corners_3d, corners2d_current, K,True)
    M_ext_ransac=helper.compute_extrinsic_matrix_ransac(corners_3d, corners2d_current, K,True)
    #PROJECTION
    P=np.dot(K,M_ext)
    registered_square=helper.world2pixels(square_3d_h,P)
    #PROJECTION RANSAC
    P=np.dot(K,M_ext_ransac)
    registered_square_ransac=helper.world2pixels(square_3d_h,P)
    #DISPLAY
    image_current=cv2.drawKeypoints(image_current,kp_current,image_current)
    cv2.polylines(image_current, [np.int32(registered_square)], True, (0, 0, 255), 4)
    cv2.polylines(image_current, [np.int32(registered_square_ransac)], True, (255, 0, 0), 4)
    cv2.imshow('Ransac (blue), no ransac (red)',image_current)
    cv2.imwrite('registration_'+str(i)+'.png',image_current)
    cv2.waitKey()
```

```
def compute_extrinsic_matrix_ransac(pts3d,pts2d,K,verbose=False):
    #Opencv "ransac" waits for float 32 type and not float 64
    pts3d=pts3d.astype(np.float32)
    pts2d=pts2d.astype(np.float32)

    dist_coef = np.zeros(4)
    #Change in OpenCV 3.1: 'strange' reshaping is required ( [N,3] to [N,1,3], and [N,2] to [N,1,2])
    #Note1: the reprojectionError must be set to "2" instead of default 8 value
    #reprojectionError: The parameter value is the maximum allowed distance between the observed and computed point projections
    #to consider it an inlier
    #In french: seuil (distance) en dessous de laquelle les points peuvent etre "inliers".
    #Note2: additional return value '_' (versus opencv 2.4)

    pts3d_tmp=pts3d.reshape(-1,1,3)
    pts2d_tmp=pts2d.reshape(-1,1,2)
    _,rvec,tvec,inliers=cv2.solvePnPRansac(pts3d_tmp, pts2d_tmp, K, dist_coef,reprojectionError=2.0)

    R,b=cv2.Rodrigues(rvec)
    M_ext=np.hstack((R,tvec))
    if verbose:
        print ("Rotation x (deg): " , np.round(rvec[0]*360.0/(2*np.pi),2))
        print ("Rotation y (deg): " , np.round(rvec[1]*360.0/(2*np.pi),2))
        print ("Rotation z (deg): " , np.round(rvec[2]*360.0/(2*np.pi),2))
        print ("Translation: ",np.round(tvec[0],2), ", " , np.round(tvec[1],2), ", "  , np.round(tvec[2],2))

        np.set_printoptions(suppress=True)
        print( "Matrice:\n", np.round(M_ext,2))
    return M_ext
```

Fonction **« compute_extrinsic_matrix_ransac »**