Kabsch

December 15, 2020

0.1 Overview

Throughout this course, we will be leveraging Google's Colab Notebooks to reinforce the concepts we have been learning in class. For an introduction into how to use collab, in case you are not already familiar with it, have a go at this overview of Colaboratory features.

0.1.1 Kabsch's Algorithm

As stated in the course notes, the Kabsch algorithm is a very versatile tool for optimally aligning two vectors to one another. In this example, we are provided with two point sets - a model set and a point (measured) set, and our goal would be to compute the optimal rotation matrix U that allows us to efficiently rotate the point set into the model set.

0.1.2 Load the Measured Point Set

For the example we are interested in, we have measured the position of an object in 3D space using a Northern Digital Inc's Polaris Camera. The points are collected as a set of three-dimensional (3D) points in space, arranged in rows of `(x,y,z)` tuples and they are as given by the `measured_points_full` function below:

```
In [9]: # Here, we are importing all the libraries we will be using in these notebook
        import os
        import numpy as np
        from os.path import join, expanduser
        import scipy.linalg as LA
```

```
In [8]: def measured_points_full():
            # these are the (x,y,z) tuples
            pre_calib = {
                '0,0.0': [-369.88531494140625, 101.30087280273438, -1960.3780517578125],
                '200,0': [-369.8937683105469, 101.32111358642578, -1960.302734375],
                '0,0.1': [-369.8780212402344, 101.32646942138672, -1960.353271484375],
                '220,0': [-369.8780212402344, 101.32646942138672, -1960.353271484375],
                '0,0.2': [-367.74957275390625, 101.65080261230469, -1953.7960205078125],
                '240,0': [-370.8532409667969, 101.074951171875, -1942.255126953125],
                '0,0.3': [-366.7646484375, 101.17594909667969, -1949.628173828125],
                '255,0': [-381.33837890625, 97.10205078125, -1920.667236328125],
                '0,0.4': [-368.0609436035156, 100.83153533935547, -1953.857177734375],
                '0,220': [-382.8047790527344, 100.34918975830078, -1944.807373046875],
```

```
    '0,0.5': [-369.7981262207031, 100.01362609863281, -1958.6396484375],
    '0,240': [-382.71600341796875, 99.87244415283203, -1945.184326171875],
    '0,0.6': [-370.24237060546875, 98.66026306152344, -1957.2281494140625],
    '0,255': [-382.71600341796875, 99.87244415283203, -1945.184326171875],
    '0,0.7': [-370.1295166015625, 98.33242797851562, -1956.1732177734375],
    }
def exp(x):
    'This function expands the array along the second dimension so that '
    return np.expand_dims(x, 1)

# sort pre-recorded points in the order.
measured_calib = np.array(([[
                            pre_calib['0,0.0'],
                            pre_calib['0,220'],
                            pre_calib['0,0.1'],
                            pre_calib['0,240'],
                            pre_calib['0,0.2'],
                            pre_calib['0,255'],
                            pre_calib['0,0.3'],
                            pre_calib['200,0'],
                            pre_calib['0,0.4'],
                            pre_calib['220,0'],
                            pre_calib['0,0.5'],
                            pre_calib['240,0'],
                            pre_calib['0,0.6'],
                            pre_calib['255,0'],
                            pre_calib['0,0.7'],
                        ]]))
"""
 As it is currently, our array has 3 dimensions. We need to reduce the size of the
 array alongthe singleton dimention for efficient matrix manipulations, hence why we
 are squeezing the matrix
"""
measured_calib_zero_centered = np.array(([0, 0, 0]))
for i in range(len(measured_calib)):
    ' find the centroid of the points '
    centered = measured_calib[i] - np.min(measured_calib, 0)
    measured_calib_zero_centered = np.vstack((measured_calib_zero_centered, centered
measured_calib_zero_centered = measured_calib_zero_centered[1:]

return measured_calib_zero_centered
```

0.1.3 Load the model set

It now behooves us to load the model set so we can begin our Kabsch computation. For this, we have them saved in a numpy array. Therefore, we will import numpy as well as associated and needed libraries necessary for our computation.

```
In [12]: model_points = np.array((
                                  [[-1755.87720294,   866.87898685,   283.0353811 ],
                                   [-1755.76266696,   866.8540598,    282.9782946 ],
                                   [-1758.9453555,    857.8363267,    296.13326449],
                                   [-1759.02853104,   865.92774874,   283.52951211],
                                   [-1777.42772925,   826.8692224,    293.38292356],
                                   [-1784.34737705,   836.7521396,    281.74652354],
                                   [-1777.77335781,   826.96331701,   292.88727602],
                                   [-1783.56649649,   836.45510137,   281.56390533],
                                   [-1783.53245361,   836.46510174,   281.54257437],
                                   [-1783.6947516,    836.55773878,   281.52364873],
                                   [-1783.58522171,   836.46979064,   281.55684051],
                                   [-1783.66230977,   836.54098015,   281.50709046],
                                   [-1783.52724697,   836.44927943,   281.56064662],
                                   [-1783.59681243,   836.52118858,   281.52347799],
                                   [-1783.44129296,   836.40624764,   281.5671847 ]]
         ))
```

0.1.4 Get the point set from the function above.

```
In [13]: point_set = measured_points_full()
```

0.2 Now, let us calculate the transformation as we described in our notes

```
In [23]: def Kabsch(P=None, Q=None, augment_Q=True, center=True):
             '''P and Q must be nX3. This rotation is accurate.
              Rotates points in P optimally to measured reference points in Q

              Params
              ======
              Q: Points to be rotated into
              augment_Q: Whether Q was recorded without the zero/home points embedded between su

              '''
             if not isinstance(P, np.ndarray) or not isinstance(Q, np.ndarray):
                 P, Q = prepro()

             # calculate the centroids
             if center:
                 'This only for computed old points'
                 q0 = np.mean(Q, 1)
                 p0 = np.mean(P, 1)

                 Q_ctr = Q - np.expand_dims(q0, 1)
                 P_ctr = P - np.expand_dims(p0, 1)
             else:
                 Q_ctr, P_ctr = Q, P
```

```
    # add the zero points to preconputed control points
    if augment_Q:
        'This only for computed old points'
        Q_aug = np.array(([[0,0,0]]))
        for i in range(Q_ctr.shape[0]-1):
                Q_aug = np.append(Q_aug, np.expand_dims(Q_ctr[i+1], 0),0)
                Q_aug=  np.append(Q_aug, np.expand_dims(Q_ctr[0], 0),0)
        Q_ctr = Q_aug[1:]


    Hmat = P_ctr.T@Q_ctr
    U, S, V = LA.svd(Hmat)
    d = np.sign(np.linalg.det(V@U.T))
    M = np.eye(3); M[-1][-1] =d
    opt_rot = V@M@U.T
    opt_trans = Q_ctr.T- opt_rot@P.T

    return opt_rot, np.mean(opt_trans, 1)
```

0.2.1 Test the algorithm

Remember that we are rotating the points in `point_set` into `model_points`. So we would go ahead and call the Kabsch function above as follows:

```
In [24]: Rot, Trans = Kabsch(model_points, point_set, augment_Q=False, center=False)

In [25]: print(Rot)

[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]


In [26]: print(Trans)

[1775.85125374 -842.66314863 -284.40256961]
```