In [1]:
```
import sys
```

In [2]:
```
!pipenv lock -r
```

```
/bin/sh: pipenv: command not found
```

In [3]:
```
!export PYTHONPATH=$(python -c "import site, os; print(os.path.join(site.USER_BASE, 'lib', 'python', 'site-packages'))"):$PYTHONPATH
```

In [4]:
```
# Courtesy by Murphy Lab, UBC.

# %%
import os
# Check if current environment is google colab.
# If so, execute following specific lines

data_root = './data/'
import sys
sys.path.append("..")
import numpy as np
import cv2
from skimage.transform import resize
from scipy import ndimage, misc
from scipy.ndimage import gaussian_filter
from scipy import stats,signal,fft
from scipy import io as sio
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from matplotlib.dates import DateFormatter
import matplotlib.patches as patches
import matplotlib.animation as manimation
from matplotlib.backends.backend_pdf import PdfPages
from mpl_toolkits import mplot3d
from mpl_toolkits.mplot3d import Axes3D
import matplotlib
# matplotlib.use('tkagg')
matplotlib.use('WebAgg')

import matplotlib.pyplot as plt


import pandas as pd
import phenograph
import io
import imageio
from IPython.core.debugger import set_trace
# from pytictoc import TicToc
# from statannot import add_stat_annotation
from pathlib import Path
from numpy import sin, linspace, pi
from pylab import plot, show, title, xlabel, ylabel, subplot
# import ffmpeg
from datetime import datetime
import glob
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
import multiprocessing as mp
import seaborn as sns
from scipy.spatial import distance
from sklearn import metrics
from pyclustertend import hopkins
from sklearn.preprocessing import scale
from scipy.ndimage import gaussian_filter,gaussian_filter1d
from rigid_transform_3D import rigid_transform_3D
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram
from pytransform3d.rotations import *
```

In [5]:
```
# %%
#@title function which returns an image as numpy array from figure
def get_img_from_fig(fig, dpi=180):
    buf = io.BytesIO()
    fig.savefig(buf, format="png", dpi=dpi)
    buf.seek(0)
    img_arr = np.frombuffer(buf.getvalue(), dtype=np.uint8)
    buf.close()
    img = cv2.imdecode(img_arr, 1)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    return img

def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram
    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count
    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)
    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)
```

In [6]:
```
#%%
#mouse wheel joints
joints = [[0,1],[1,2],[2,3],[3,4],[4,5],[5,6],[6,7],[7,8],[8,9],[9,10],
          [10,11],[11,12],[12,13],[14,15],[16,17],[17,18],[18,19],[19,20],
          [20,21],[22,23],[23,24],[24,25],[25,26],[26,27]]

# data_2d = ['LD1_1580414966_2d.csv', 'LD1_1580415036_2d.csv', 'LD1_1580415176_2d.csv',
#            'LD1_1580415664_2d.csv', 'LD1_1580416013_2d.csv', 'LD1_1580416083_2d.csv',
#            'LD1_1580416431_2d.csv', 'LD1_1580416571_2d.csv', 'LD1_1580416920_2d.csv',
#            'LD1_1580417059_2d.csv', 'LD1_1580417618_2d.csv', 'LD1_1580417687_2d.csv',
#            'LD1_1580418315_2d.csv', 'LD1_1580418873_2d.csv', 'LD1_1580419640_2d.csv']
#
# data_3d = ['LD1_1580414966_3d.csv', 'LD1_1580415036_3d.csv', 'LD1_1580415176_3d.csv',
#            'LD1_1580415664_3d.csv', 'LD1_1580416013_3d.csv', 'LD1_1580416083_3d.csv',
#            'LD1_1580416431_3d.csv', 'LD1_1580416571_3d.csv', 'LD1_1580416920_3d.csv',
#            'LD1_1580417059_3d.csv', 'LD1_1580417618_3d.csv', 'LD1_1580417687_3d.csv',
#            'LD1_1580418315_3d.csv', 'LD1_1580418873_3d.csv', 'LD1_1580419640_3d.csv']
```

In [7]:
```
data_2d = ['LD1_1580415036_2d.csv']
data_3d = ['LD1_1580415036_3d.csv']
coords_all_2d = []
coords_all_3d = []
dataset_name_2d = []
dataset_name_3d = []
```

In [8]:
```
for f_2d, f_3d in zip(data_2d,data_3d):
    coords_file = data_root + f_2d
    dataset_name_2d = coords_file
    coords_2d = pd.read_csv(coords_file,dtype=np.float, header=2)
```

In [ ]:
```
```

In [9]:
```
    coords_2d = coords_2d.values[:,1:] #exclude first column
    coords_2d = np.delete(coords_2d, list(range(2, coords_2d.shape[1], 3)), axis=1) #delete every 3rd column o
f prediction score
    coords_all_2d.append(coords_2d)

    coords_file = data_root + os.sep + f_3d
    dataset_name_3d = coords_file.split('/')[-1].split('.')[0]
    coords_3d = pd.read_csv(coords_file, header=2)
    coords_3d = coords_3d.values[:, 1:] #exclude the index column
    coords_3d = np.around(coords_3d.astype('float'), 2) #round to two decimal places
    coords_3d = gaussian_filter1d(coords_3d, 5, axis=0) #smooth the data, the points were oscillating
    coords_all_3d.append(coords_3d)
```

In [10]:

```
#%%

coords_all_2d = np.vstack(coords_all_2d) #convert to numpy stacked array
coords_all_3d = np.vstack(coords_all_3d)
x_3d = coords_all_3d[:, ::3];        y_3d = coords_all_3d[:, 1::3];        z_3d = coords_all_3d[:, 2::3];
x_2d = coords_all_2d[:, ::2];        y_2d = coords_all_2d[:, 1::2];        z_2d = np.zeros(x_2d.shape);
coords_all_3d_trans = []
for i in np.arange(x_3d.shape[0]):

    A=np.mat([x_3d[i,:], y_3d[i,:], z_3d[i,:]])
    B=np.mat([x_2d[i,:], y_2d[i,:], z_2d[i,:]])
    # Calculate rotation and translation from A to B
    ret_R, ret_t = rigid_transform_3D(A, B)
    n = x_3d.shape[1]
    # transform A to reference frame of B
    B2 = (ret_R*A) + np.tile(ret_t, (1, n))
    coords_all_3d_trans.append(np.asarray(B2.flatten(order='F')))
```

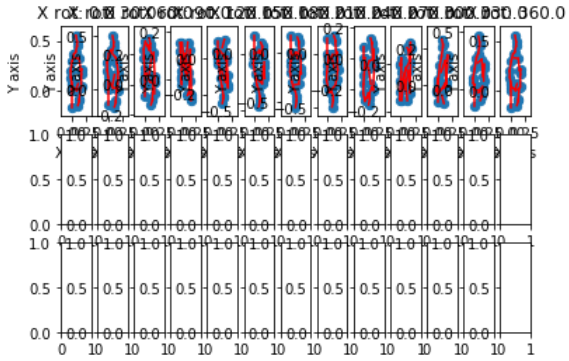
```
det(R) < R, reflection detected!, correcting for it ...
det(R) < R, reflection detected!, correcting for it ...
det(R) < R, reflection detected!, correcting for it ...
det(R) < R, reflection detected!, correcting for it ...
det(R) < R, reflection detected!, correcting for it ...
det(R) < R, reflection detected!, correcting for it ...
```

In [11]:
```
coords_all_3d_trans = np.asarray(np.vstack(coords_all_3d_trans))
# 2d projection of transformed 3d coordinates
coords_all_3d_trans_2dproj = np.delete(coords_all_3d_trans, list(range(2, coords_all_3d_trans.shape[1], 3)), a
xis=1)
k=30 # K for k-means step of phenograph
communities_2d, graph, Q = phenograph.cluster(coords_all_2d, k=k)
n_clus_2d = np.unique(communities_2d).shape[0]
communities_3d, graph, Q = phenograph.cluster(coords_all_3d, k=k)
n_clus_3d = np.unique(communities_3d).shape[0]
communities_3d_trans, graph, Q = phenograph.cluster(coords_all_3d_trans, k=k)
communities_3d_trans_2dproj, graph, Q = phenograph.cluster(coords_all_3d_trans_2dproj, k=k)
```

```
Finding 30 nearest neighbors using minkowski metric and 'auto' algorithm
Neighbors computed in 0.5019335746765137 seconds
Jaccard graph constructed in 0.6135966777801514 seconds
Wrote graph to binary file in 0.2736027240753174 seconds
Running Louvain modularity optimization
After 1 runs, maximum modularity is Q = 0.857498
After 2 runs, maximum modularity is Q = 0.859617
Louvain completed 22 runs in 1.400242567062378 seconds
PhenoGraph complete in 2.8092081546783447 seconds
Finding 30 nearest neighbors using minkowski metric and 'auto' algorithm
Neighbors computed in 0.42337965965270996 seconds
Jaccard graph constructed in 0.6936032772064209 seconds
Wrote graph to binary file in 0.3783600330352783 seconds
Running Louvain modularity optimization
After 1 runs, maximum modularity is Q = 0.798621
After 5 runs, maximum modularity is Q = 0.799845
Louvain completed 25 runs in 1.7128562927246094 seconds
PhenoGraph complete in 3.2335944175720215 seconds
Finding 30 nearest neighbors using minkowski metric and 'auto' algorithm
Neighbors computed in 0.7535905838012695 seconds
Jaccard graph constructed in 0.624932050704956 seconds
Wrote graph to binary file in 0.3549983501434326 seconds
Running Louvain modularity optimization
After 1 runs, maximum modularity is Q = 0.866071
After 4 runs, maximum modularity is Q = 0.867934
Louvain completed 24 runs in 1.4183664321899414 seconds
PhenoGraph complete in 3.1689112186431885 seconds
Finding 30 nearest neighbors using minkowski metric and 'auto' algorithm
Neighbors computed in 0.5573151111602783 seconds
Jaccard graph constructed in 0.6557719707489014 seconds
Wrote graph to binary file in 0.25798749923706055 seconds
Running Louvain modularity optimization
After 1 runs, maximum modularity is Q = 0.864395
After 3 runs, maximum modularity is Q = 0.867824
Louvain completed 23 runs in 1.3833684921264648 seconds
PhenoGraph complete in 2.8743298053741455 seconds
```

In [12]:
```
hopkins_arr = []
calinski_arr = []
q_arr = []
var_arr = [] #array to hold total variance of all points at different rotation projection
rot_angles = np.linspace(0,360,13) #array of angular rotations to be applied to the 3d data for various projec
tions
fig, ax = plt.subplots(3, rot_angles.shape[0])
#apply rotations along X-axis
for i, angle in enumerate(rot_angles):
    rot_x = matrix_from_angle(0, np.deg2rad(angle)) #generate rotations matrix for a given angle rotation
    coords_all_3d_transX = []
    #apply rotation matrix to all points frame by frame
    for fr in np.arange(coords_all_3d.shape[0]):

        A=np.mat([coords_all_3d[fr,::3], coords_all_3d[fr,1::3], coords_all_3d[fr,2::3]])
        B = rot_x * A
        coords_all_3d_transX.append(np.asarray(B.flatten(order='F')))

    coords_all_3d_transX = np.asarray(np.vstack(coords_all_3d_transX))
    # 2d project the above transformed 3d data
    coords_all_3d_transX_2dproj = np.delete(coords_all_3d_transX, list(range(2, coords_all_3d_transX.shape[1],
3)),
                                            axis=1)
    var_arr.append(coords_all_3d_transX_2dproj.var()) #calculate and append variance of points in this project
ion
    #phenograph clustering
    communities_3d_transX_2dproj, graph, Q = phenograph.cluster(coords_all_3d_transX_2dproj, k=k); q_arr.appen
d(Q)
    #calculate hopkins score for this projection and append it to the list
    h = hopkins(coords_all_3d_transX_2dproj, coords_all_3d_transX_2dproj.shape[0]); hopkins_arr.append(h)
    #calculate calinski harabasz score and append it to the list
    c = metrics.calinski_harabasz_score(coords_all_3d_transX_2dproj, communities_3d_transX_2dproj); calinski_a
rr.append(c)
    #plot the transformation projected skeleton
    x_3d = coords_all_3d_transX_2dproj[0, ::2];        y_3d = coords_all_3d_transX_2dproj[0, 1::2];
    sc = ax[0, i].scatter(x_3d, y_3d, s=40, label=str(0))
    for bone in joints:
        ax[0, i].plot([x_3d[bone[0]], x_3d[bone[1]]], [y_3d[bone[0]], y_3d[bone[1]]], 'r')
    ax[0, i].set_xlabel('X axis');        ax[0, i].set_ylabel('Y axis');
    ax[0, i].set_title('X rot: ' + str(angle))
```

```
PhenoGraph complete in 3.415166139602661 seconds
Finding 30 nearest neighbors using minkowski metric and 'auto' algorithm
Neighbors computed in 0.733910083770752 seconds
Jaccard graph constructed in 0.6258273124694824 seconds
Wrote graph to binary file in 0.4875209331512451 seconds
Running Louvain modularity optimization
After 1 runs, maximum modularity is Q = 0.803516
After 7 runs, maximum modularity is Q = 0.804702
Louvain completed 27 runs in 1.8623685836791992 seconds
PhenoGraph complete in 3.726212501525879 seconds
Finding 30 nearest neighbors using minkowski metric and 'auto' algorithm
Neighbors computed in 0.7615516185760498 seconds
Jaccard graph constructed in 0.6997153759002686 seconds
Wrote graph to binary file in 0.45431041717529297 seconds
Running Louvain modularity optimization
After 1 runs, maximum modularity is Q = 0.796794
After 6 runs, maximum modularity is Q = 0.797982
Louvain completed 26 runs in 1.8506600856781006 seconds
PhenoGraph complete in 3.7851977348327637 seconds
Finding 30 nearest neighbors using minkowski metric and 'auto' algorithm
Neighbors computed in 0.9808154106140137 seconds
Jaccard graph constructed in 0.712916374206543 seconds
Wrote graph to binary file in 0.5305912494659424 seconds
Running Louvain modularity optimization
After 1 runs, maximum modularity is Q = 0.802414
After 16 runs, maximum modularity is Q = 0.80367
After 23 runs, maximum modularity is Q = 0.805049
After 24 runs, maximum modularity is Q = 0.806832
Louvain completed 44 runs in 3.0663018226623535 seconds
PhenoGraph complete in 5.310630559921265 seconds
Finding 30 nearest neighbors using minkowski metric and 'auto' algorithm
Neighbors computed in 0.8516459465026855 seconds
Jaccard graph constructed in 0.7141101360321045 seconds
Wrote graph to binary file in 0.4100654125213623 seconds
Running Louvain modularity optimization
After 1 runs, maximum modularity is Q = 0.804572
After 5 runs, maximum modularity is Q = 0.808005
Louvain completed 25 runs in 1.750438928604126 seconds
PhenoGraph complete in 3.749793529510498 seconds
```



In [13]:
```
# apply rotations along Y-axis
for i, angle in enumerate(rot_angles):
    rot_y = matrix_from_angle(1, np.deg2rad(angle))
    coords_all_3d_transY = []
    for fr in np.arange(coords_all_3d.shape[0]):

        A=np.mat([coords_all_3d[fr,::3], coords_all_3d[fr,1::3], coords_all_3d[fr,2::3]])
        B = rot_y * A
        coords_all_3d_transY.append(np.asarray(B.flatten(order='F')))

    coords_all_3d_transY = np.asarray(np.vstack(coords_all_3d_transY))
    coords_all_3d_transY_2dproj = np.delete(coords_all_3d_transY, list(range(2, coords_all_3d_transY.shape[1],
3)),
                                            axis=1)
    var_arr.append(coords_all_3d_transY_2dproj.var())
    communities_3d_transY_2dproj, graph, Q = phenograph.cluster(coords_all_3d_transY_2dproj, k=k); q_arr.appen
d(Q)
    h = hopkins(coords_all_3d_transY_2dproj, coords_all_3d_transY_2dproj.shape[0]); hopkins_arr.append(h)
    c = metrics.calinski_harabasz_score(coords_all_3d_transY_2dproj, communities_3d_transY_2dproj); calinski_a
rr.append(c)

    x_3d = coords_all_3d_transY_2dproj[0, ::2];        y_3d = coords_all_3d_transY_2dproj[0, 1::2];
    sc = ax[1, i].scatter(x_3d, y_3d, s=40, label=str(0))
    for bone in joints:
        ax[1, i].plot([x_3d[bone[0]], x_3d[bone[1]]], [y_3d[bone[0]], y_3d[bone[1]]], 'r')
    ax[1, i].set_xlabel('X axis');        ax[1, i].set_ylabel('Y axis');
    ax[1, i].set_title('Y rot: ' + str(angle))
```

In [14]:
```
#apply rotations along Z-axis
for i, angle in enumerate(rot_angles):
    rot_z = matrix_from_angle(2, np.deg2rad(angle))
    coords_all_3d_transZ = []
    for fr in np.arange(coords_all_3d.shape[0]):

        A=np.mat([coords_all_3d[fr,::3], coords_all_3d[fr,1::3], coords_all_3d[fr,2::3]])
        B = rot_z * A
        coords_all_3d_transZ.append(np.asarray(B.flatten(order='F')))

    coords_all_3d_transZ = np.asarray(np.vstack(coords_all_3d_transZ))
    coords_all_3d_transZ_2dproj = np.delete(coords_all_3d_transZ, list(range(2, coords_all_3d_transZ.shape[1],
3)),
                                            axis=1)
    var_arr.append(coords_all_3d_transZ_2dproj.var())
    communities_3d_transZ_2dproj, graph, Q = phenograph.cluster(coords_all_3d_transZ_2dproj, k=k); q_arr.appen
d(Q)
    h = hopkins(coords_all_3d_transZ_2dproj, coords_all_3d_transZ_2dproj.shape[0]); hopkins_arr.append(h)
    c = metrics.calinski_harabasz_score(coords_all_3d_transZ_2dproj, communities_3d_transZ_2dproj); calinski_a
rr.append(c)

    x_3d = coords_all_3d_transZ_2dproj[0, ::2];        y_3d = coords_all_3d_transZ_2dproj[0, 1::2];
    sc = ax[2, i].scatter(x_3d, y_3d, s=40, label=str(0))
    for bone in joints:
        ax[2, i].plot([x_3d[bone[0]], x_3d[bone[1]]], [y_3d[bone[0]], y_3d[bone[1]]], 'r')
    ax[2, i].set_xlabel('X axis');        ax[2, i].set_ylabel('Y axis');
    ax[2, i].set_title('Z rot: ' + str(angle))
```

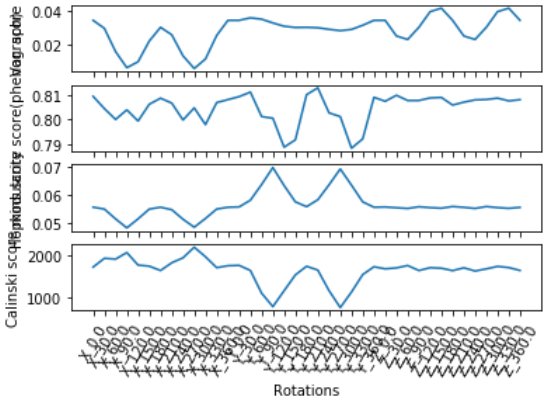
```
Neighbors computed in 0.6885528564453125 seconds
Jaccard graph constructed in 0.712367057800293 seconds
Wrote graph to binary file in 0.5103211402893066 seconds
Running Louvain modularity optimization
After 1 runs, maximum modularity is Q = 0.805073
After 2 runs, maximum modularity is Q = 0.808052
Louvain completed 22 runs in 1.6294069290161133 seconds
PhenoGraph complete in 3.5646262168884277 seconds
Finding 30 nearest neighbors using minkowski metric and 'auto' algorithm
Neighbors computed in 0.6523830890655518 seconds
Jaccard graph constructed in 0.724571704864502 seconds
Wrote graph to binary file in 0.5105772018432617 seconds
Running Louvain modularity optimization
After 1 runs, maximum modularity is Q = 0.804166
After 2 runs, maximum modularity is Q = 0.805302
After 5 runs, maximum modularity is Q = 0.806377
After 6 runs, maximum modularity is Q = 0.808595
Louvain completed 26 runs in 2.1983132362365723 seconds
PhenoGraph complete in 4.108865261077881 seconds
Finding 30 nearest neighbors using minkowski metric and 'auto' algorithm
Neighbors computed in 0.6740541458129883 seconds
Jaccard graph constructed in 0.7048666477203369 seconds
Wrote graph to binary file in 0.4109807014465332 seconds
Running Louvain modularity optimization
After 1 runs, maximum modularity is Q = 0.804319
After 2 runs, maximum modularity is Q = 0.80625
After 7 runs, maximum modularity is Q = 0.807507
Louvain completed 27 runs in 2.12532377243042 seconds
PhenoGraph complete in 3.9386355876922607 seconds
Finding 30 nearest neighbors using minkowski metric and 'auto' algorithm
Neighbors computed in 0.7842280864715576 seconds
Jaccard graph constructed in 0.697012186050415 seconds
Wrote graph to binary file in 1.1199843883514404 seconds
Running Louvain modularity optimization
After 1 runs, maximum modularity is Q = 0.804412
After 2 runs, maximum modularity is Q = 0.806903
After 3 runs, maximum modularity is Q = 0.807976
Louvain completed 23 runs in 1.8652691841125488 seconds
PhenoGraph complete in 4.4895501136779785 seconds
```

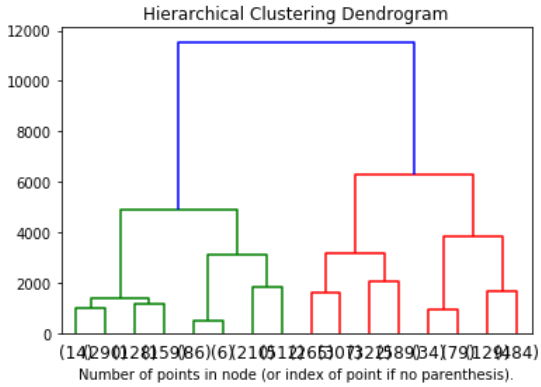
In [15]:
```
import sklearn
print(sklearn.__version__)
```

```
0.24.1
```

In [16]:
```
plt.show()
#plot modularity score, hopkins score and calinski scores for all the projected rotations transformation
xlabels = ['X_'+str(i) for i in np.around(rot_angles)]
xlabels.extend(['Y_'+str(i) for i in np.around(rot_angles)])
xlabels.extend(['Z_'+str(i) for i in np.around(rot_angles)])
fig, ax = plt.subplots(4, 1,sharex=True)
ax[0].plot(var_arr); ax[0].set_xticks(np.arange(len(xlabels))); ax[0].set_xticklabels(xlabels, rotation=60)
ax[0].set_ylabel('Var. score')
ax[1].plot(q_arr); ax[1].set_xticks(np.arange(len(xlabels))); ax[1].set_xticklabels(xlabels, rotation=60)
ax[1].set_ylabel('modularity score(phenograph)')
ax[2].plot(hopkins_arr); ax[2].set_xticks(np.arange(len(xlabels))); ax[2].set_xticklabels(xlabels, rotation=60
)
ax[2].set_ylabel('Hopkins score')
ax[3].plot(calinski_arr); ax[3].set_xticks(np.arange(len(xlabels))); ax[3].set_xticklabels(xlabels, rotation=6
0)
ax[3].set_xlabel('Rotations'); ax[3].set_ylabel('Calinski score'); plt.show()

#%% Make dendrogram
# setting distance_threshold=0 ensures we compute the full tree.
aggClust = AgglomerativeClustering(distance_threshold=0, n_clusters=None)
aggClust = aggClust.fit(coords_all_2d)
plt.figure(); plt.title('Hierarchical Clustering Dendrogram')
# plot the top three levels of the dendrogram
plot_dendrogram(aggClust, truncate_mode='level', p=3)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()

hopkins(coords_all_2d, coords_all_2d.shape[0])
hopkins(coords_all_3d, coords_all_3d.shape[0])
metrics.calinski_harabasz_score(coords_all_2d, communities_2d)
metrics.calinski_harabasz_score(coords_all_3d, communities_3d)
from sklearn.metrics import davies_bouldin_score
davies_bouldin_score(coords_all_2d, communities_2d)
davies_bouldin_score(coords_all_3d, communities_3d)
metrics.silhouette_score(coords_all_2d, communities_2d, metric='euclidean')
#%% make video of 2d pose and 3d pose side by side
# if not os.path.exists(dataset_name_2d[:-3]+"_skel.mp4"):
#     FFMpegWriter = manimation.writers['ffmpeg']
#     metadata = dict(title=dataset_name_2d[:-3] + ' 2d and 3d tracking of joints', artist='Matplotlib')
#     skel_writer = FFMpegWriter(fps=30, metadata=metadata)
#     fig = plt.figure(figsize=plt.figaspect(0.5))
#     ax1 = fig.add_subplot(121, projection='3d'); #ax1.axis("off");
#     ax2 = fig.add_subplot(122, projection='3d'); #ax2.axis("off");
#     l1 = ax1.scatter(0, 0, s=40,label='0')
#     l2 = ax2.scatter(0, 0, s=40,label='0')
#     with skel_writer.saving(fig, dataset_name_2d[:-3]+"_skel.mp4", 100):
#         for i in np.arange(0, len(communities_re)):
#             print("Frame: ", i)
#             # x_2d = coords_all_2d[:,0]
#             # y_re = coords_all_2d[:,1]
#             # plt.figure(); plt.plot(x_2d,label='x_2d'); plt.plot(y_re,label='y_re');plt.legend();plt.show()
#             x_2d = coords_all_2d[i,::2]
#             y_re = coords_all_2d[i,1::2]
#             ax1.clear()
#             sc = ax1.scatter(x_2d, y_re, s=40,label=str(communities_re[i]))
#             for bone in joints:
#                 ax1.plot([x_2d[bone[0]], x_2d[bone[1]]], [y_re[bone[0]], y_re[bone[1]]], 'r')
#             ax1.set_xlabel('X axis'); ax1.set_ylabel('Y axis'); ax1.set_zlabel('Z axis'); ax1.legend()
#             ax1.invert_yaxis(); ax1.set_xlim3d(0,640); ax1.set_ylim3d(0,320)
#             ax1.view_init(-90, -90)
#             # plt.show()
#
#             # x_3d = coords_all_3d[:,0]
#             # y_3d = coords_all_3d[:,1]
#             # z_sy = coords_all_3d[:,2]
#             # plt.figure(); plt.plot(x_3d,label='x_3d'); plt.plot(y_3d,label='y_3d');plt.plot(z_sy,label='z_
sy');plt.legend();plt.show()
#             x_3d = coords_all_3d_trans[i,::3]
#             y_3d = coords_all_3d_trans[i,1::3]
#             z_sy = coords_all_3d_trans[i,2::3]
#             ax2.clear()
#             sc = ax2.scatter3D(x_3d, y_3d, z_sy, s=40, label=str(communities_3d_trans[i]))
#             for bone in joints:
#                 ax2.plot3D([x_3d[bone[0]], x_3d[bone[1]]], [y_3d[bone[0]], y_3d[bone[1]]], [z_sy[bone[0]], z
_sy[bone[1]]], 'r')
#             ax2.set_xlabel('X axis'); ax2.set_ylabel('Y axis'); ax2.set_zlabel('Z axis'); ax2.legend()
#             ax2.invert_yaxis(); ax2.set_xlim3d(0,640); ax2.set_ylim3d(0,320); #ax2.set_zlim3d(-0.2,0.2)
#             # ax2.view_init(30, 0)
#             ax2.view_init(-90, -90)
#             # plt.show()
#
```

```
#             skel_writer.grab_frame()
#     # plt.show()
#     plt.close()
```





Out[16]: 0.2351909260575516

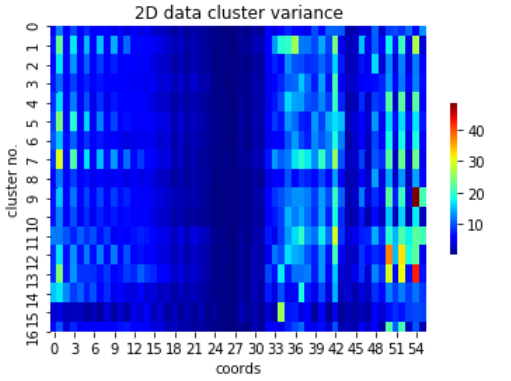
In [17]:
```
#%% make gif of mean pose of clusters in 2d data
if not os.path.exists('syn_2d_cluster_k'+str(k)+'_mean.gif'):
    res = pd.DataFrame(coords_all_2d).groupby(communities_2d).apply(lambda x: x.values)
    coords2d_clus_mean = [b.mean(axis=0) for b in res]
    coords2d_clus_mean = np.array(coords2d_clus_mean)
    coords2d_clus_std = [b.std(axis=0) for b in res]
    coords2d_clus_std = np.array(coords2d_clus_std)
    images = []
    for i in np.arange(coords2d_clus_mean.shape[0]):
        fig = plt.figure();
        ax = plt.gca()
        x_2d = coords2d_clus_mean[i, ::2]; y_2d = coords2d_clus_mean[i,1::2]
        sc = ax.scatter(x_2d, y_2d, s=40, label=str(i))
        for bone in joints:
            ax.plot([x_2d[bone[0]], x_2d[bone[1]]], [y_2d[bone[0]], y_2d[bone[1]]], 'r')
        ax.set_xlabel('X axis'); ax.set_ylabel('Y axis');  ax.legend()
        ax.set_xlim(0, 640); ax.set_ylim(0, 320); ax.invert_yaxis();
        images.append(get_img_from_fig(fig))
        plt.close()
    imageio.mimwrite('syn_2d_cluster_k'+str(k)+'_mean.gif', images,fps=1)
```

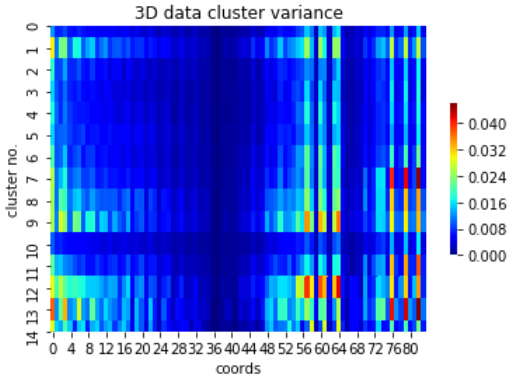
In [18]:

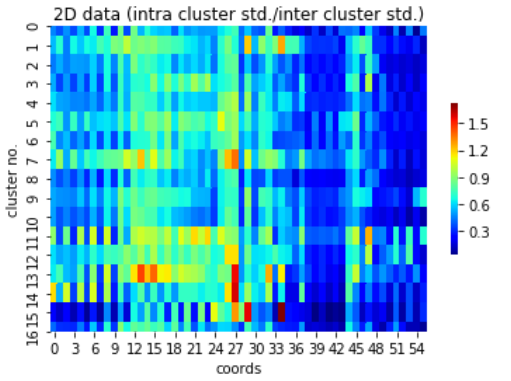
```
#%% make gif of mean pose of clusters in 3d data
if not os.path.exists('syn_3d_cluster_k'+str(k)+'_mean.gif'):
    res = pd.DataFrame(coords_all_3d).groupby(communities_3d).apply(lambda x: x.values)
    coords3d_clus_mean = [b.mean(axis=0) for b in res]
    coords3d_clus_mean = np.array(coords3d_clus_mean)
    coords3d_clus_std = [b.std(axis=0) for b in res]
    coords3d_clus_std = np.array(coords3d_clus_std)
    images = []
    for i in np.arange(coords3d_clus_mean.shape[0]):
        fig = plt.figure();
        ax = plt.axes(projection='3d')
        x_3d = coords3d_clus_mean[i, ::3];        y_3d = coords3d_clus_mean[i, 1::3];        z_3d = coords3d_c
lus_mean[i, 2::3];
        sc = ax.scatter3D(x_3d, y_3d, z_3d, s=40, label=str(i))
        for bone in joints:
            ax.plot3D([x_3d[bone[0]], x_3d[bone[1]]], [y_3d[bone[0]], y_3d[bone[1]]], [z_3d[bone[0]], z_3d[bon
e[1]]],
                      'r')
        ax.set_xlabel('X axis');        ax.set_ylabel('Y axis');
        ax.set_xlim3d(-0.2, 0.2);        ax.set_ylim3d(-0.2, 0.7);        ax.set_zlim3d(-0.2, 0.2);        ax.
invert_yaxis();
        ax.view_init(10, 0)
        ax.legend()
        images.append(get_img_from_fig(fig))
        plt.close()
    imageio.mimwrite('syn_3d_cluster_k'+str(k)+'_mean.gif', images, fps=1)
```

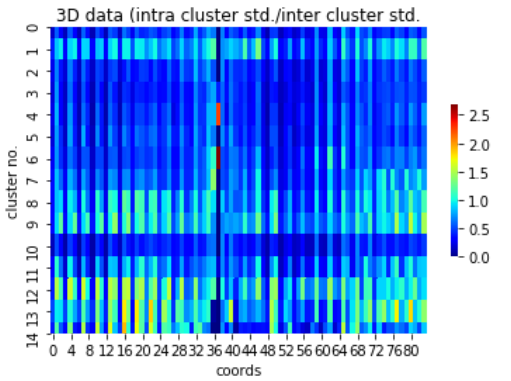
In [19]:

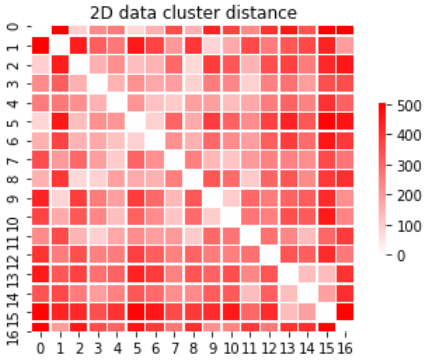
```
#%%
plt.figure(); sns.heatmap(coords2d_clus_std, cmap=cm.jet, cbar_kws={"shrink": .5});
plt.xlabel('coords');plt.ylabel('cluster no.'); plt.title('2D data cluster variance')
plt.figure(); sns.heatmap(coords3d_clus_std, cmap=cm.jet, cbar_kws={"shrink": .5})
plt.xlabel('coords');plt.ylabel('cluster no.'); plt.title('3D data cluster variance')

plt.figure(); sns.heatmap(coords2d_clus_std/coords_all_2d.std(axis=0), cmap=cm.jet, cbar_kws={"shrink": .5});
plt.xlabel('coords');plt.ylabel('cluster no.'); plt.title('2D data (intra cluster std./inter cluster std.)')
plt.figure(); sns.heatmap(coords3d_clus_std/coords_all_3d.std(axis=0), cmap=cm.jet, cbar_kws={"shrink": .5})
plt.xlabel('coords');plt.ylabel('cluster no.'); plt.title('3D data (intra cluster std./inter cluster std.')

a2 = distance.cdist(coords2d_clus_mean, coords2d_clus_mean, 'euclidean')
a3 = distance.cdist(coords3d_clus_mean, coords3d_clus_mean, 'euclidean')
plt.figure(); sns.heatmap(a2,  center=0,linewidths=.5, cmap=cm.bwr, square=True, xticklabels=np.arange(len(a2
)),
                          yticklabels=np.arange(len(a2)), cbar_kws={"shrink": .5})
plt.title('2D data cluster distance')
plt.figure(); sns.heatmap(a3,  center=0,linewidths=.5, cmap=cm.bwr, square=True, xticklabels=np.arange(len(a3
)),
                          yticklabels=np.arange(len(a3)), cbar_kws={"shrink": .5})
plt.title('3D data cluster distance')
x2d = coords2d_clus_mean[:,::2]
y2d = coords2d_clus_mean[:,1::2]
z2d = np.zeros(x2d.shape)
aa = [np.column_stack([x2d[:,i],y2d[:,i],z2d[:,i]]) for i in np.arange(0,x2d.shape[1])]
aaa = np.hstack(aa)
a33 = distance.cdist(aaa, coords3d_clus_mean, 'euclidean')
```

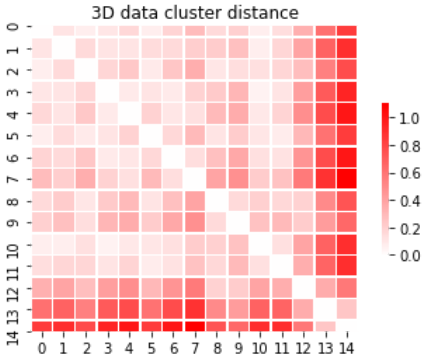

2D data cluster variance
cluster no.
coords
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
40
30
20
10


3D data cluster variance
cluster no.
coords
0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
0.040
0.032
0.024
0.016
0.008
0.000

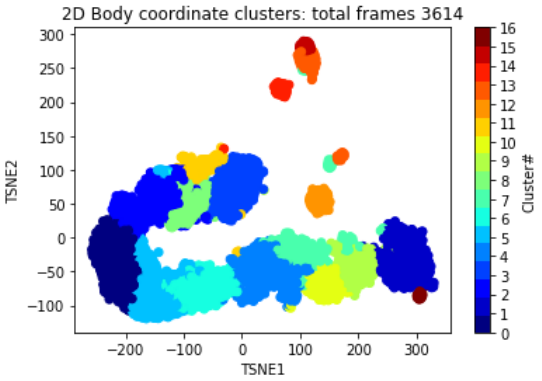

2D data (intra cluster std./inter cluster std.)
cluster no.
coords
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1.5
1.2
0.9
0.6
0.3


3D data (intra cluster std./inter cluster std.
cluster no.
coords
0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
2.5
2.0
1.5
1.0
0.5
0.0



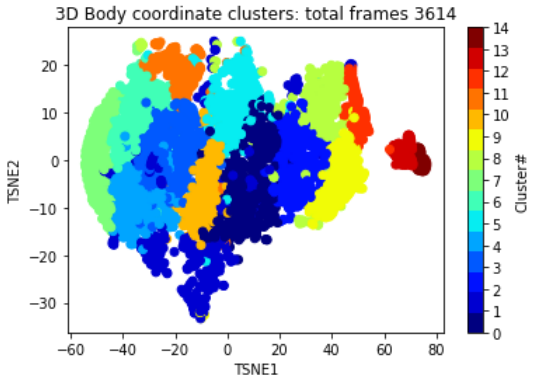


In [20]:
```
#%% tSNE plot of 2d data labeled by phenograph clusters
if not os.path.exists('syn_2d_tsne.png'):
    # tsne_model = TSNE(n_components=2, perplexity=30.0, early_exaggeration=12.0,
    #                   learning_rate=20.0, n_iter=1000, n_iter_without_progress=300,
    #                   min_grad_norm=1e-07, metric='euclidean', init='random',
    #                   verbose=0, random_state=None, method='barnes_hut',
    #                   angle=0.5)
    # tsne_model = TSNE(n_components=2,perplexity=30,random_state=0)
    tsne_model = TSNE(n_components=2, random_state=2,perplexity=100,angle=0.1,init='pca',n_jobs= mp.cpu_count
()-1)
    Y_2d = tsne_model.fit_transform(coords_all_2d)
    # cmap = matplotlib.colors.ListedColormap ( np.random.rand ( np.unique(communities_re).shape[0],3))
    cmap = plt.cm.colors.ListedColormap(plt.cm.jet(np.linspace(0,1,n_clus_2d)))
    plt.figure()
    plt.scatter(Y_2d[:,0], Y_2d[:,1],
                    c=communities_2d,
                    cmap=cmap,
                    alpha=1.0)
    plt.colorbar(ticks=np.unique(communities_2d), label='Cluster#')
    plt.xlabel('TSNE1'); plt.ylabel('TSNE2')
    plt.title('2D Body coordinate clusters: total frames ' + str(len(communities_2d)))
    plt.savefig('syn_2d_tsne.png', format='png')
    plt.show()
    plt.close()
```

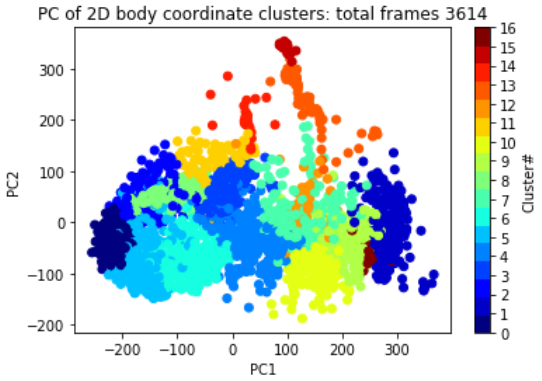


In [21]:

```
#%% tSNE plot of 3d data labeled by phenograph clusters
if not os.path.exists('syn_3d_tsne.png'):
    # tsne_model = TSNE(n_components=2, perplexity=30.0, early_exaggeration=12.0,
    #                   learning_rate=20.0, n_iter=1000, n_iter_without_progress=300,
    #                   min_grad_norm=1e-07, metric='euclidean', init='random',
    #                   verbose=0, random_state=None, method='barnes_hut',
    #                   angle=0.5)
    # tsne_model = TSNE(n_components=2,perplexity=30,random_state=0)
    tsne_model = TSNE(n_components=2, random_state=2,perplexity=100,angle=0.1,init='pca',n_jobs= mp.cpu_count
()-1)
    Y = tsne_model.fit_transform(coords_all_3d)
    # cmap = matplotlib.colors.ListedColormap ( np.random.rand ( np.unique(communities_sy).shape[0],3))
    cmap = plt.cm.colors.ListedColormap(plt.cm.jet(np.linspace(0,1,n_clus_3d)))
    plt.figure()
    plt.scatter(Y[:,0], Y[:,1],
                c=communities_3d,
                cmap=cmap,
                alpha=1.0)
    plt.colorbar(ticks=np.unique(communities_3d), label='Cluster#')
    plt.xlabel('TSNE1'); plt.ylabel('TSNE2')
    plt.title('3D Body coordinate clusters: total frames ' + str(len(communities_3d)))
    plt.savefig('syn_3d_tsne.png', format='png')
    plt.show()
    plt.close()
```



In [22]:
```
#%% PCA of 2d data
pca_model = PCA(n_components=2)  # Initializes PCA
pca_model.fit(coords_all_2d)  # Performs PCA
# pca_model.score_samples()
pca_model.explained_variance_ratio_
scores = pca_model.transform(coords_all_2d)
plt.figure()
cmap = plt.cm.colors.ListedColormap(plt.cm.jet(np.linspace(0, 1, n_clus_2d)))
plt.scatter(scores[:, 0], scores[:, 1],
            c=communities_2d,
            cmap=cmap,
            alpha=1.0)
plt.colorbar(ticks=np.unique(communities_2d), label='Cluster#')
plt.xlabel('PC1');
plt.ylabel('PC2')
plt.title('PC of 2D body coordinate clusters: total frames ' + str(len(communities_2d)))
plt.savefig('syn_2d_PC.png', format='png')
plt.show()
plt.close()

#%% was trying to plot eigen vectors of PCA, not finished yet. leave it commented or delete these.
# def draw_vector(v0, v1, ax=None):
#     ax = ax or plt.gca()
#     arrowprops = dict(arrowstyle='->',
#                       linewidth=2,
#                       shrinkA=0, shrinkB=0)
#     ax.annotate('', v1, v0, arrowprops=arrowprops)
#
# # plot data
# plt.figure();
# plt.scatter(coords_all_2d[:, 0], coords_all_2d[:, 1], alpha=0.2)
# for length, vector in zip(pca_model.explained_variance_, pca_model.components_[:, 0:2]):
#     v = vector * 3 * np.sqrt(length)
#     draw_vector(pca_model.mean_[0:2], pca_model.mean_[0:2] + v)
# plt.axis('equal');
# plt.title('Eigenvectors of 2D body coordinates: total frames ' + str(len(communities_re)))
# plt.close()
```



In [23]:
```
#%% PCA of 3d data
pca_model = PCA(n_components=2)  # Initializes PCA
pca_model.fit(coords_all_3d)  # Performs PCA
# pca_model.score_samples()
pca_model.explained_variance_ratio_
scores = pca_model.transform(coords_all_3d)
plt.figure()
cmap = plt.cm.colors.ListedColormap(plt.cm.jet(np.linspace(0, 1, n_clus_3d)))
plt.scatter(scores[:, 0], scores[:, 1],
            c=communities_3d,
            cmap=cmap,
            alpha=1.0)
plt.colorbar(ticks=np.unique(communities_3d), label='Cluster#')
plt.xlabel('PC1');
plt.ylabel('PC2')
plt.title('PC of 3D body coordinate clusters: total frames ' + str(len(communities_3d)))
plt.savefig('syn_3d_PC.png', format='png')
plt.show()
plt.close()
```