```python
# import open3d
import h5py
import cv2
import numpy as np
import ipyvolume as ipv
from sklearn.cluster import DBSCAN
import networkx as nx
import matplotlib.pyplot as plt

PATH = r"/Users/suyashsachdeva/Desktop/gsoc_data.hdf5"

with h5py.File(PATH, 'r') as f:
    image = f['X_jets'][:]
    m0 = f["m0"][:]
    pt = f["pt"][:]

image.shape
```

```
(139306, 125, 125, 3)
```

```python
ipv.figure()
points_graph = []
for c, color in enumerate(["red", "green", "blue"]):
    tracks_img = image[0, :, :, c]

    m = m0[0]
    p = pt[0]

    # Assuming the image size is 125x125 and each pixel's intensity
represents the magnitude of detection
    image_size = 125

    # Extract x, y coordinates and their magnitudes from the tracks
image
    y_coords, x_coords = np.where(tracks_img > 0)  # Get indices of
non-zero (detected) pixels
    magnitudes = tracks_img[y_coords, x_coords]  # Magnitude from
pixel intensity

    # Convert magnitudes to a z-coordinate, influenced by m0 and pt,
adjusting for visualization
    # Here, we assume a simple linear relationship for demonstration
    z_coords = np.array((m + p) * (magnitudes / np.max(magnitudes)),
dtype="int8")*4
    if color=="red":
        for c in range(len(x_coords)):
            ipv.scatter(x_coords[c], y_coords[c], z_coords[c],
marker='sphere', color=color, size=1)
            points_graph.append([x_coords[c], y_coords[c],
z_coords[c], 1, color])
    # Form the 3D points array
    else:
        points = np.vstack((x_coords, y_coords, z_coords)).T

        clustering = DBSCAN(eps=1.0, min_samples=4).fit(points)
        labels = clustering.labels_

        # Filter out noise (-1 label)
        points_filtered = points[labels != -1]
        labels_filtered = labels[labels != -1]

        # Calculate centroid and radius for each cluster
        unique_labels = set(labels_filtered)
        centroids = np.array([points_filtered[labels_filtered ==
label].mean(axis=0) for label in unique_labels])
        radii = np.array([np.sqrt((points_filtered[labels_filtered ==
label].shape[0]) / np.pi) for label in unique_labels])

        # Normalize radii for visualization purposes
        radii_normalized = radii / np.max(radii) * 2  # Scale radii
for better visualization

        ### Step 3: Visualize Clusters
        # Plot each cluster as a sphere with radius proportional to
the number of points
        for centroid, radius in zip(centroids, radii_normalized):
            ipv.scatter(centroid[0], centroid[1], centroid[2],
size=radius, marker='sphere', color=color)
            points_graph.append([x_coords[c], y_coords[c],
z_coords[c], radius, color])

ipv.xlabel('X')
ipv.ylabel('Y')
ipv.zlabel('Z')
ipv.show()
```

{"model_id":"f9cf3bbf05954dd7a93e280e4bc777a5","version_major":2,"vers
ion_minor":0}

```
from pyvis.network import Network
import numpy as np
from sklearn.cluster import DBSCAN
import networkx as nx
# Filter out noise

G = nx.Graph()
for i, point in enumerate(points_graph):
    G.add_node(i, pos=points_graph[i][:3], radius=points_graph[i][3],
color=points_graph[i][4])

# Define a threshold distance for connecting nodes
```

```
threshold_distance = 60.0

# Add edges based on distance
for i in range(len(points_graph)):
    for j in range(i+1, len(points_graph)):
        dist = np.linalg.norm(np.array(points_graph[i][:3]) -
np.array(points_graph[j][:3]))
        if dist <= threshold_distance:
            # The connection strength could be inversely proportional
to the distance
            strength = 1 / dist if dist != 0 else 1
            G.add_edge(i, j, weight=strength)



# Initialize a Pyvis network with remote CDN resources
net = Network(notebook=True, height="750px", width="100%",
cdn_resources='remote')

# Add nodes and edges from the NetworkX graph
# Ensuring all attributes are converted to JSON serializable formats
for node, attr in G.nodes(data=True):
    # try:
        net.add_node(node, title=f"Node {node}",
color=str(attr['color']),  size=int(attr['radius'])*10)


for source, target, attr in G.edges(data=True):
    width = float(attr['weight']) * 10  # Convert weight to float and
scale for visibility
    # try:
    net.add_edge(source, target, title=f"{attr['weight']:.2f}",
width=width)


# Display the network
net.show("graph.html")
```

```
graph.html
```

```
<IPython.lib.display.IFrame at 0x17ea5d730>
```