INFOMCV Assignment 3
Authors (Group number)

Summary

In order to implement this assignment, we drew on the solution code provided for Assignment 2. The voxel space was set by arbitrarily enlarging the world dimensions (length, height, depth) directly in the config.json file. We set our coordinates to 240, 80 and 240 respectively.

To create the color models, we used a single camera, namely camera 4. We chose this particular camera in order to obtain the cluster centers for offline phase because it provided an initially spread-out image of the four people. This would allow for more discernable voxel clusters of the individuals.
In order to create the color models, we used histograms, rather than Gaussian mixture models. We did this primarily because we found histograms were more straightforward to implement, but also because the histograms are more precise, albeit not as efficient. The color models were created (see 'color.py') by converting to HSV, in order to allow for changes in lighting conditions. The coordinates of the projected voxels on the image (created using a binary mask) are contained in 'project_list', with the color of each voxel saved in 'color_list'. This allows for the calculation of the histograms which was done using the cv.calcHist function from opencv. The bin values were set at 16 for 2 channels of the HSV image (namely, hue and saturation). This was done in order to sufficiently distinguish between individual color models without becoming overly computationally expensive. For calculating the distances between old and new histograms, we used the compareHist() function from opencv (see compute_distance() function in 'color.py').
We used Bhattacharyya distance to compare the color models between frames, based on the similarities of their color distributions. This is an efficient method of computing distances for our implementation because it can deal with changes in lighting by measuring the overlap between distributions.
Whilst this was our methodology, we were unable to correctly implement the cluster labeling across frames. This is something that can be re-visited in order to correctly

In order to cluster the centers, we used K-means (opencv: cv.kmeans()) with 20 iterations. To improve the robustness of identifying the clusters, we initialized the K-means with cv.KMEANS_PP_CENTERS rather than randomly locating the initial cluster centers. The combination of these flags with multiple iterations helped in avoiding local minima. We then had to match the clusters to a particular label, so that the labeling of a particular cluster would remain consistent across frames. To do so, we implemented the Hungarian algorithm (see hungarian_algorithm() in 'color.py'). This function uses the linear_sum_assignment() function on the distance matrix and effectively tracks the continuity of individuals in the video across all frames based on their color models. This is done in a way which maximizes the similarity between a given histogram and the corresponding histogram in a previous frame. The 'live_matching' function keeps track of the frames and color models in order to compare the histograms and track them across the course of the simulation. We edited the 'executable.py' file in order to realize the necessary adaptations from assignment 2 and implement the live tracking and clustering.

To build the 2D tracking map, we had to track the movement of the voxel clusters on a flat map, which plotted the trajectories of the four voxel clusters. We did this for every 25th video frame because the size of the video frames (54 x 50) was too large and computationally expensive. We implemented the matplotlib library to plot the cluster centers. As we were unable to correctly match the labels with the cluster centers, we wrote the cluster centers to a .xml file and used it to draw the resulting map (assuming a column represents the same cluster)

Link to video

https://www.youtube.com/channel/UC45WeCixZR_TwqRMfarR8ZA

Trajectory image

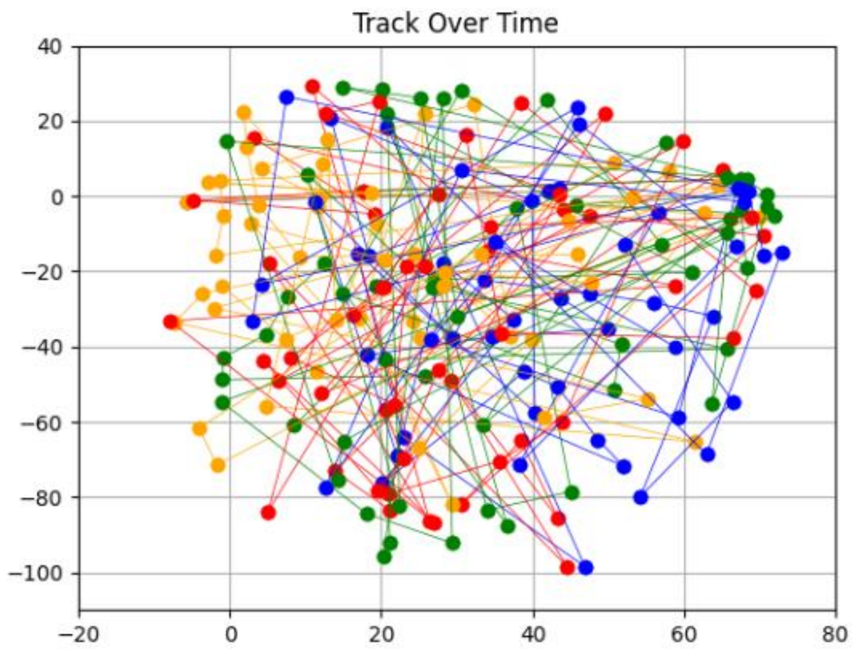

Fig.1: Trajectory Map of Voxel Cluster Centers Over Time

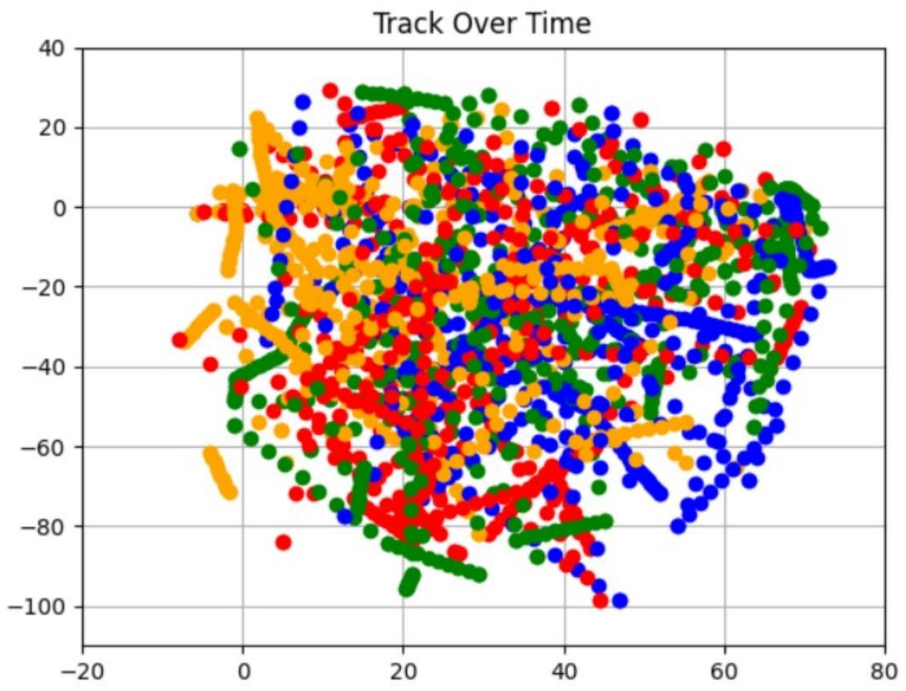

Fig2: Trajectory Map with Interpolated Points (Result of Smoothing)

Choice tasks

[Choice 4] – To implement this choice we implemented np.linspace() function to interpolate more points between cluster centers (see tracker.py; fig. 2)

[Choice 6] – We attempted to implement this choice task by directly updating the color model per frame. However, due to our inability to correctly match the clusters, we could not complete this task.