Computer Vision 22/23 Course Assignment 3 Report

Color-based Voxel Labeling

G. Picarella (2713810)

March 19, 2023

1 Camera Calibration

We calibrated the intrinsics and extrinsics parameters for each camera with the script `calibration.py` and used the same chessboard data from assignment 2.

2 Voxel space

Our voxel space is computed with the script `lookup.py` which employs parallelization and vectorization to speed-up the look-up table computation. We computed a voxel space centered at $< 12, 0, 0 >$ with world size $< 45, 32, 18 > \cdot s$ where $s = 115$ is the chessboard square size during the calibration. The chosen size guarantees that for the entire video duration every person is contained in the voxelized volume. Then, we use `build_voxel_to_pixel_lookup(...)` to compute the inverse look-up table which maps each voxel to the corresponding 2D pixel per view. This last step is not computationally expensive and can be done on-the-fly (based on the stored look-up table) right before starting the simulation.

3 Clustering

Our clustering phase is straightforward: we compute the set of voxels that will be drawn in the scene for the current frame and compute the clusters with **compute_kmeans_clusters(...)** (the detailed process is explained in the choice task 7.3).

4 Color models

Our color models are based on HSV histograms. For each camera c_i we project the voxels turned on to the i-th view: we avoid duplicate pixels by storing each pixel location in a set. Then, we consider only the pixels within the ranges $(h_{min} + h_{range} \cdot 0.1, h_{max} - h_{range} \cdot 0.4)$, $(w_{min} + wrange \cdot 0.2, w_{max} - wrange \cdot 0.2)$ as valid samples. We sample $n = 100$ colors from the current frame in the hsv space and compute one histogram per channel with $cv2.calcHist(...)$. We don't consider colours with very low value components (darker colors with $v < 20$); it has shown tangible better results during the online matching phase. The histograms are then normalized with $cv2.normalize(...)$ using the Min-Max normalization between $[0, 1]$ and stored in a dictionary mapping the i-th camera to the list of histograms per person. The process is exactly the same for both the offline and online computation of the color models, the only difference is the frame used when fetching the colors from each pixel location: for the offline part, we use the frame at index 0 while for the online part we use whatever frame at index $k > 0$.

5 Online matching

For each frame at index $k > 0$, we compute a match with **cluster_matching(...)**. In our case, the main limitations we experienced in the matching phase concern how to deal with occlusions and ghost voxels. If person k is partially or completely occluded by person j then the histogram computed with pixel colors coming from the 2D projection of voxels from person k cannot be considered reliable in the matching phase. On the other hand, if the i-th camera doesn't contain any occlusion, typically we can assume that the matching will be much more precise for that view. Based on these assumptions, we assign a bonus factor $b_i = 6$ to each camera having a foreground mask with at least 4 white distinguishable blobs. Once the bonus and color histograms are computed for each camera, we perform the actual matching phase using multiple cameras (check choice task 7.1 for more info). On the other hand, ghost voxels still remain a difficult problem to tackle especially when the ghost blobs get bigger than the actual 3D shape. In these cases, our matching showed to fail in various frames.

6 Trajectories

After each cluster without outliers is computed (check choice task 7.3 for more details), we store the 2D center location for each matched person in a dictionary mapping the person label to the list of centers. We use **draw_trajectories(...)** to draw each cluster center to a 2D window in real-time. Each cluster center is scaled by the ratio between the voxel space size and the 2d window size so that drawn trajectories are kept coherent with the real 3D space. Once the video is ended, the final raw trajectories map is stored as a jpg image. In addition, we also store the smoothed trajectories map (check choice task 7.2 for more info).

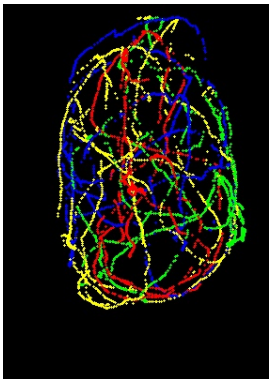

Figure 1: Raw trajectories for the entire video duration

7 Our Choice Tasks

7.1 Using multiple cameras to increase the robustness

We selected 3 cameras, in particular cam 1, 2 and 4. The main reason is that compared to camera 4, these 3 cameras don't show any occlusion for the video frame at index 0, thus allowing the computation of each color model from the same clustering. For each camera, we compute one color model per person and store it in a dictionary mapping the camera index to the list of color models. In the matching phase, for each camera c_i we compute the online histogram for each person and store it in a dictionary mapping the camera index i to the list of histograms. Next, we initialize a matrix of nn elements where $n = cameras$ and set each position k, j equal to the correlation coefficient computed between the k-th online color model and the j-th offline color model. A match M_i for the i-th camera is then found by choosing the 4 cells with the highest correlation score in the matrix. In order to guarantee a match M with unique labels, each time a pair k, j is selected, the k-th and j-th row and column is filled with $-\infty$. The match M_i is paired with its overall score $S(M_i) \cdot b_i$ which is the sum of all the correlation scores selected during the search. Then we collect the three matches (one per camera) M_1, M_2, M_3 and if M_i has the same label ordering as M_j with $i \neq j$ then M_i, M_j are merged in M_{ij} and $S(M_{ij}) = S(M_i) + S(M_j)$. If the merged list of matches contains just one element, then we consider it our final match, otherwise we sort the list of matches by score and check the distance between the first and the second matches: if $abs(S(M_1) - S(M_2))$ is lower than a user-specified threshold then the matching is considered unreliable for the current frame and no matching is displayed, otherwise the first match is returned. This final check based on S guarantees a significant reduction of false positives and false negatives throughout the entire video duration.

7.2 Smoothing of trajectories

We store for each tracked person a list containing the estimated cluster center for the entire video duration. For each person, we compute the parameterized curve representing its trajectory with the Scipy utility **interpolate.splprep(...)** and then sample 10000 points along the curve with **interpolate.splev(...)**. Finally, we draw each point with the color corresponding to the specific person.

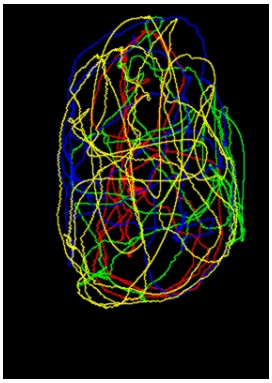

Figure 2: Smoothed trajectories for the entire video duration

7.3 Getting rid of outliers and ghost voxels

We defined a function **compute_kmeans_clusters(...)** that implements an outliers minimization strategy in each cluster. First, a clustering for the 3D scene is computed with **cv2.kmeans(...)**; For each

cluster, the distance between the cluster's center and a point in the cluster is computed; Then, we compute the mean μ and standard deviation STD of the distances array D and remove each point p_i with $d_i \in D | abs(\mu - d) > f \cdot STD$, where f is a user-specified factor. Finally, we cluster again the remaining voxels and return the final result without outliers.

8 Video

A video showing the final result of our Color-based Voxel Labeling can be found on Youtube.

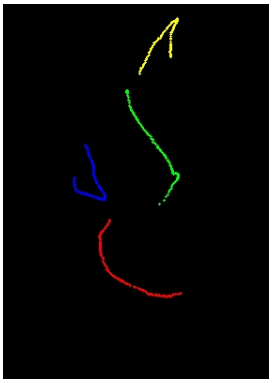

Figure 3: Raw trajectories for the Youtube video

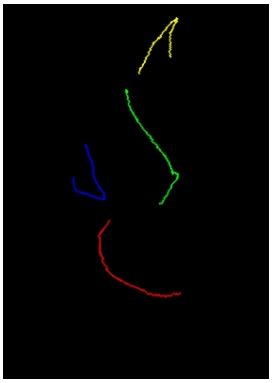

Figure 4: Smoothed trajectories for the Youtube video