Advanced Vision Assignment № 1

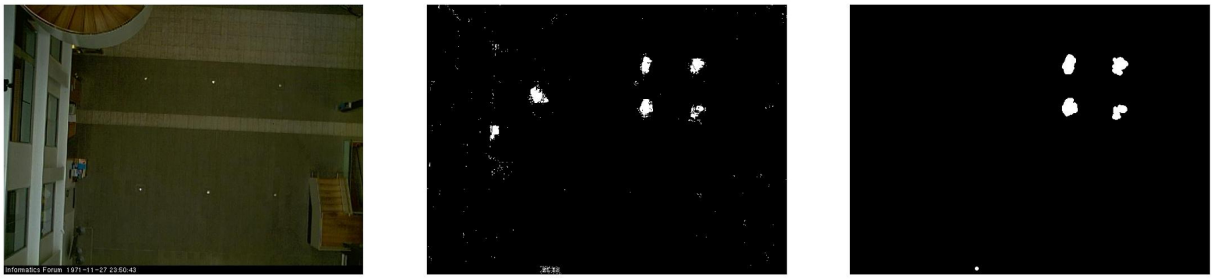
Georgi Tinchev & Ruslan Burakov, The University of Edinburgh 11/02/2015

1 Person Detector

In order to detect moving object at the scene, we have split our task into several subtasks, namely - noise removal, labelling data, static object removal, labelling - histogram computation and evaluation (bin choice, Bhattacharayya distance), tracking and evaluation.

1.1 Noise Removal

The first important observation is that we looked deeply into what exactly "close" and "open" parameters are influencing, when using "bwmorph". And we found out that they both use the same "brush" size for both erosion and dilation. This was not flexible for our approach, thus we used erode and dilate as separate actions. This behaviour allows us to utilise better segmentation of the video frames. Figure 1a is the original background of the scene. Figure 1b and 1c present the two steps taken in order to clean the image from the unnecessary noise.



(a) Original (b) Noisy (c) Noise-free

Figure 1: Background images

As presented on Figure 1b, there are 2 bigger objects, excluding the 4 dancers that we are to track. We detected a shadow at on the left of the moving person, around (125, 240) in the image coordinate frame. Our initial goal was to remove it via the algorithm for cleaning noise from binary images (remove 5 pixels-"imclose"-remove 20 pixels). However, as everything on the left is a wall, there will not be any people dancing on it, thus we cropped the image until x = 125 pixels, that also removed the object/person's shadow. Furthermore, the person in positions (150,223) was not static, thus, we tracked his trajectory manually pixel-wise and cropped a black rectangle in the area he walks. We applied this rectangle as a mask thereafter. The result is shown on Figure 1c, where only the 4 important blobs are shown. As we can see on the bottom of Figure 1c, a small dot is left even after the noise-removal algorithm. In addition, the computed binary mask produces blobs that are not completely full. An example of such is the dancer on the

bottom right. To further optimise the algorithm, we used another dilation and erosion, explained in Section 1.2.

1.2 Labelling

In order to label each of the dancers, we had to crop the 4 blobs, that can be seen on Figure 1c. Initially we considered comparing each blob on a black background of the big 480 by 640 image in multiple frames. However, this approach would be computationally inefficient when computing the Bhattacharyya distance, thus we decided to crop around each of the blobs. We considered two ways of splitting the image into blobs - drawing a circle around each blob or by cropping a box. The former finds the centre of each person(blob) and draws a circle with a specific area. The radius of the circle is calculated by the distance from the centre of the mass of the blob to the further away pixel. The latter refers to finding the topmost, leftmost, bottommost, rightmost pixels of each labelled blob and crop such a rectangle. We decided that the first approach was more time-efficient, as there was such an implementation in the AV's website.

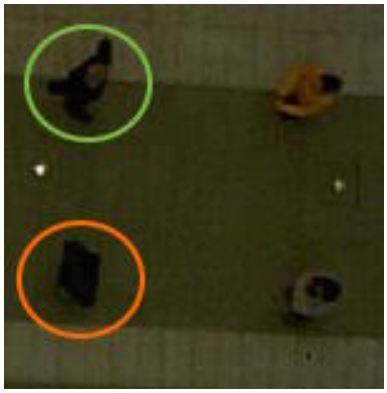
After having the 4 blobs, we have investigated the case of using 8-connectedness look-up pixels when checking the directions. When going through a 640x480 pixels, the algorithm took a noticeable amount of time. Due to inconsistencies in the detected blobs, we maintained the 8-connectedness approach, as its precision is higher. Once labelled, we order the blobs by area size and iterate through each of them in order to "threshold" certain area (150px). Everything below that threshold is not considered a person and can be safely removed. Afterwards we again apply some dilation and erosion in that order, to fill-in holes in specific dancers(bottom right one). Furthermore, we ordered each of the blobs by its area size in order to label them. The labelling itself is done in several steps. The intermediate result is shown on Figure 2.

Figure 2: Identified people with their centre of mass

1.2.1 From blobs to people

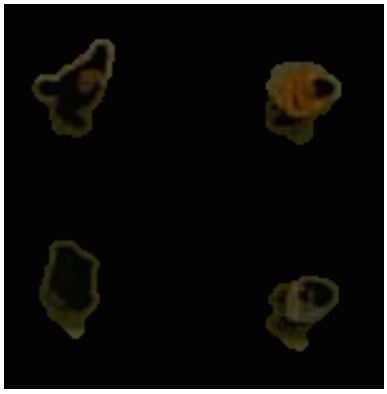
First of all, we utilise the background removing and noise reducing algorithms on the first frame of the video and retrieve the number of blobs that correspond to the number of people/dancers that will perform on the video. For each one of them we create a colour histogram. Thereafter, in each consecutive frame we extract the blobs and compare the colour histograms between people using Bhattacharyya distance. More in-depth details on the approach are given below, as the utilised method is quite tricky.

When computing the colour histograms, we could not get reliable measurements for the dancers in dark clothes. Therefore, we carried out a series of experiments in order to determine the ideal bin size. We tried adopting B. Majecka's approach [Maj09], however, it turned out to be not precise enough for our case. She concludes that people tend to wear dark clothes, which is the same conclusion we came across. However, her choice for the bin size of colour histograms was based on additional factors.



(a) People in dark clothes

(b) Hairs similar to the background



(c) Initial image cleaning

(d) Additional background removing

Figure 3

During the experiments we discovered that people who wear black clothes have very similar colours on Figure 3a (marked by orange and green circles). The only distinguishable feature between them is hair colour. One of them has black hairs and the other one has blond/red hairs. One can notice that the hair colour of the blond person is very similar to the light colour of the background, as shown on Figure 3b. Thus, if that person bends forward, his hair will be very similar to the background, and therefore both people, wearing black clothes will have a

fair amount of black and yellow/green colours in their colour distribution, making them indistinguishable. We have thought about two approaches to tackle this issue - the usage of positional information and hierarchical background subtraction. The former - because people are viewed from above they appear as concentric circles, with one bigger/outer circle representing people's clothes and inner one responsible for person's hair colour. In reality, however, people cannot be represented via perfect circles and it is not trivial to retrieve a stable zone, where the head is located. Therefore, we decided to utilise the second technique, that we named hierarchical background subtraction. It allows to iteratively get better estimate of people's contours without risking reducing them to nothingness. In this technique once we get our initial blobs we know that there is very high chance that the person is inside the blob and there is background contour around him/her. The cleaning of such image is shown in steps on Figure 3d. Relying on this information, about person's presence in the blob, we subtract background once more in order to remove background contour, as shown on Figure 3d. One might think that the person in the bottom left has disappeared, which is not really the case, however, the algorithm correctly picks out that small detail. The obtained result can be very sparse and small that is why we fill resulting pixels with convex shape using the built-in Matlab "bwconvhull" function. If nevertheless the resulting area becomes smaller than a certain threshold, we just reverse this operation. This allows us to preserve enough area in extreme cases. For instance, the guy white clothes often disappears due to the second background subtraction.

The algorithm's code is shown in Appendix 4.

The produced histograms of the above algorithm for two frames - 111, being the first and 128, randomly chosen one, can be seen on Figure 4 and 5 respectively.

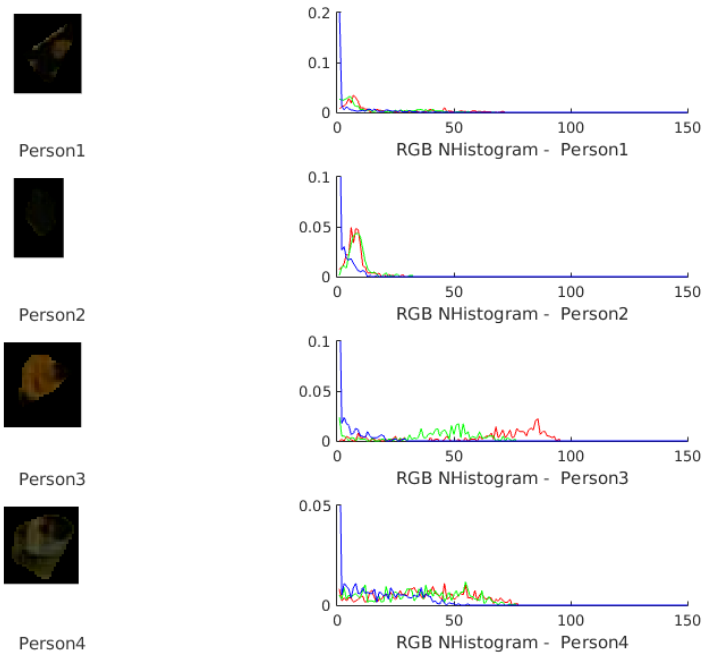


Figure 4: Histogram of frame 111 (initial) using RGB coordinates and [0:255] edges

One can also notice that the person position is swapped between different frames, as they

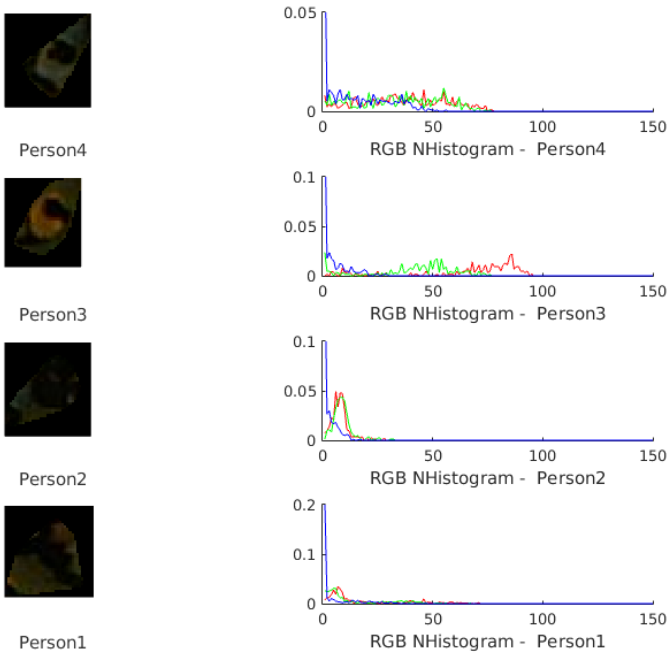


Figure 5: Histogram of frame 128 using RGB coordinates and [0:255] edges

are detected by the size of the blob and recognised thereafter, and a label is assigned. In other words, the order on Figure 4 and 5 is different, determined by the size of the convex, whereas the labelling is correct. In addition, these pictures were created using [0:255] bin size.

Figure 6 and 7 are showing the bin-sizes Majecka [Maj09] uses, applied to our problem. One could see that it miss-labels people with dark clothes (Person 1 and 2).

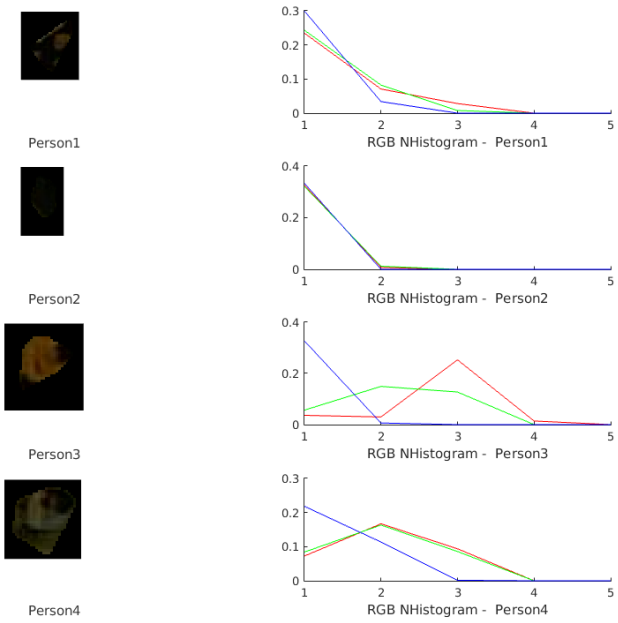


Figure 6: Histogram of frame 111 (initial) using RGB coordinates and 20, 30, 40, 166 edges

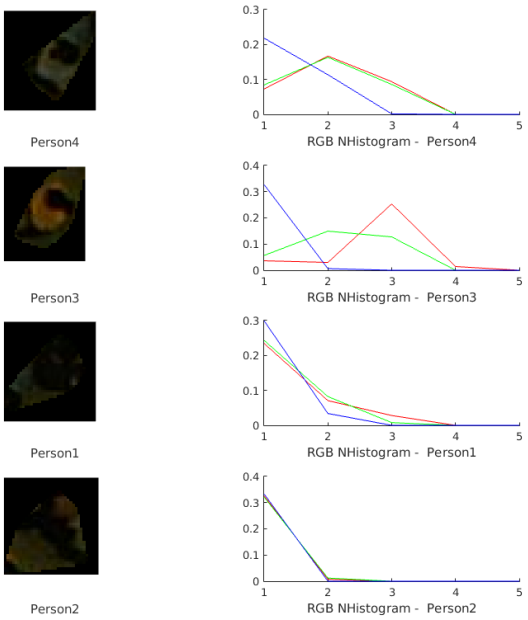


Figure 7: Histogram of frame 128 using RGB coordinates and 20, 30, 40, 166 edges

Figure 8 and 9 are showing the use of chromatic coordinates. As we can see there are mislabelling, even for people that should be clearly different. For instance, Person 1 in Figure 8 and Person 1 in Figure 9. The colour distribution there is very different, even if we use only r/g colour.

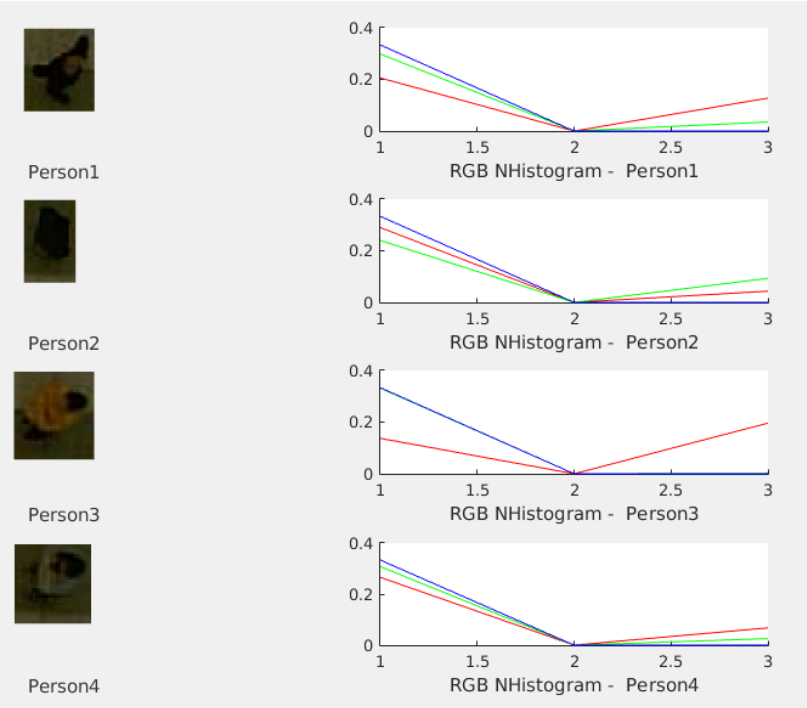


Figure 8: Histogram of frame 111 (initial) using RGB Cromatic Coordinates and 20, 30, 40, 166 edges

Finally, Figure 10 shows the final result of the labelling.

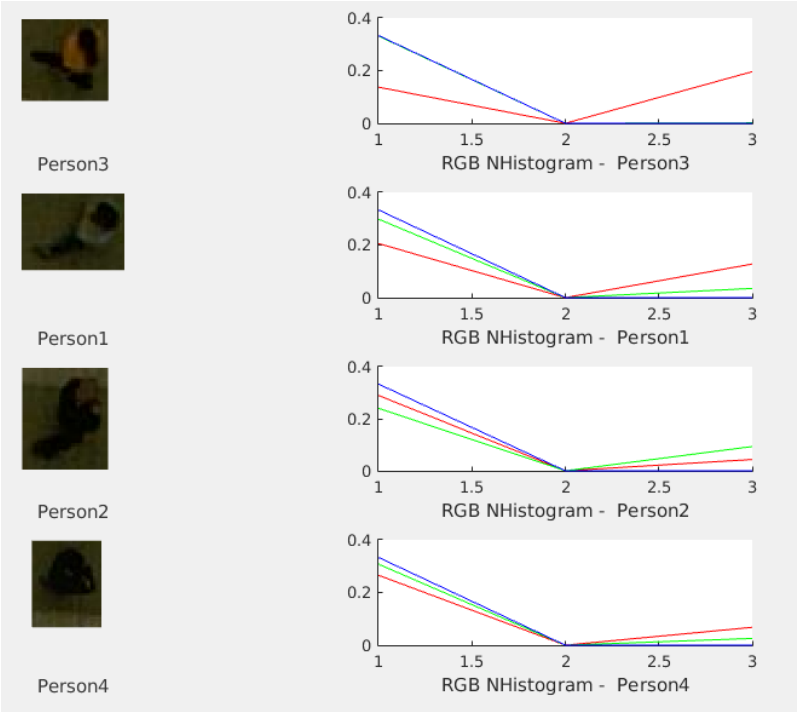


Figure 9: Histogram of frame 128 using RGB Cromatic Coordinates and 20, 30, 40, 166 edges

Figure 10: Labelled people

1.2.2 Blob Separation

As our colour histograms produced a satisfactory result, we tried using them as a tracker and pinpointing the centre of mass of each. Of course, there was a problem when people are close together, and our blob detector retrieved between 2 and 3 blobs. Therefore, we decided to split the biggest one in two. We tried 5 different methods, from which we will briefly explain the one being considered and why the others failed. The implemented approach detects the number of blobs in the picture, and until it matches the ground truth from the first frame (number of blobs detected in the first frame, in this case 4), it iteratively tries to split the bigger blob in two. It retrieves the MinorAxis of a region, and draws a black line directly on the binary image. More specifically, we create a new mask, draw a line there, invert the colours and "multiply" the line by the binary mask, producing black pixels between the blobs. With some modification of the line's width we were able to distinguish most of the blobs, to an extend where only around 20 frames were miss-labelled. The reason was that some blobs split in 3, after such a line passes through. This can happen if on the minor axis of the region at the very end there's a curve with a hole (person's arm for instance). Finally the algorithm updates its beliefs about the blobs and counts them. Thereafter, we compute the colour histograms for each blobs, using the Hierarchical Background Subtraction. The Blob Splitting algorithm is shown in Appendix 4. Several of the other approaches failed, because of the poor initial choice of blob's threshold size. Thus we adjusted the algorithm to be iterative. After implementing the above approach and trying it out we realised, it does not deal well with noise. We tried erosion and dilation, however that was not a trivial task, as it applies to all the frame, causing some of the blobs to disappear. We modified it in a way that the line in the middle is split in two, and goes closer to the centre of mass of the dancer. This small change was very influential, as it allowed people to be split even when interacting in an "armswing" (the hardest part to model via colour histograms). On all the previous approaches we got very noisy estimates and mixed computations for the "armswing" movement. Figure 11c, 11b and 11a show the process of separating patches.

(a) Patch to be separated (b) Bottom Person separated (around CoM) (c) Bottom Person separated (around CoM)

Figure 11: Blob separation

2 Gaussian smoothing

Sometimes colour histograms reacted too strongly to small pieces of background left withing person's blob, after the cleaning. We coped with that by applying prepossessing to images. We increased the contrasts of the images and smoothed them with Gaussian filter. The results of these operations can be seen on Figure 12a and can be compared to image without filtering on Figure 12b. In our runs filtering leads to more robust predictions provided by colour histograms. Note that we are using filtering only on the second step when we are doing secondary background subtraction in order to get more accurate contour of the person.

(a) Image Filtered (b) Non-filtered Image

Figure 12: Gaussian smoothing

3 Tracking

We have modelled this part relying on the given information that there are 4 persons in the field of view. Moreover, on the lab sessions it was said to us that there are only pair interactions in the dancing. Initially we considered using Condensation tracker but in our model we do not have reasonable input signal as gravity in the ball example. The reasonable hypothesises for Condensation tracker would be either person is moving (including idle state), interacting with the other dancer or there is frame rate jump and the person appears in the new place. The other problem with Condensation tracker is that it has many parameters to modify and estimate.
Instead we use simple statistical system which combines observation of people positions obtained from colour histograms and the previous position of the people. Between frames people move only a little bit. Thus, in most cases previous people positions can provide a lot of information about their current position, except for the cases of frame rate sudden jumps. In order to determine when such jump occurs and positions are no longer reliable we find minimum cumulative distance between two sets of positions: previous positions of people and current positions of blobs. This is done by calculating all possible permutations between blobs and people, which in our case is $4! = 24$ as in previous stage we have already separated initial blobs into 4. Thereafter, the case with minimum distance between matches is chosen. During the jumps we computed this metric to set threshold with which we could distinguish normal cases from

jumps. On Figure 13 4 picks correspond to the 4 jumps which can be observed during running. As it was expected their values are way above the normal ones reaching between 80 and 120 whereas usually this value does not exceed 20. Based on these observations, we determined the threshold to distinguish jumps at 63 pixels.

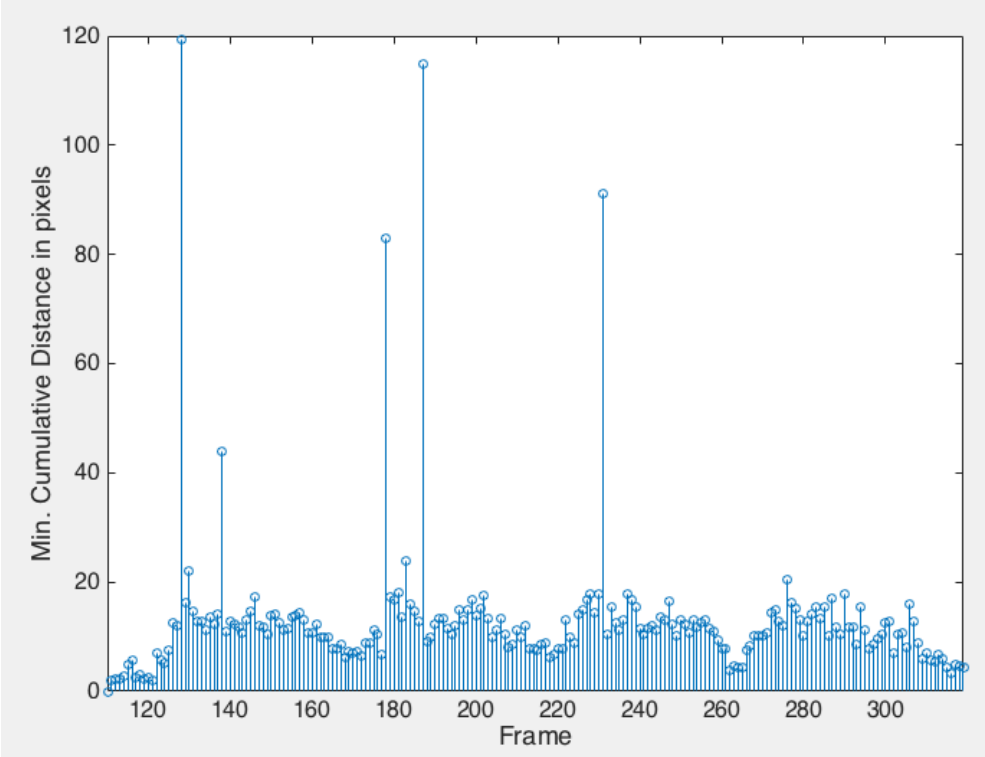


Figure 13: Minimum cumulative distance between blobs and people

Once we have the threshold with good precision, we can rely on the minimum cumulative distance assignment of people to blobs during "jump"(frame-skip). Thereafter, initialise the new position using colour histogram observations. However, such attitude has many pitfalls, for example, if after "jump", our histogram provides wrong results, then the algorithm will capture errors and propagate them further. In addition, positions are not always perfectly reliable. Therefore, for each person currently tracked and assigned to some particular blob we keep the list of predictions (to which person this blob belongs to) obtained from the colour histograms. We assume that the most of frequent person in this list must be the tracked one. If however, the opposite happens, mostly likely something went wrong and we try to reassign each person as if a jump occurred. Surprisingly, this simple method in combination with colour histograms provides correct estimation most of the time, reaching very good results - provided in Section 4. The relevant code is in the Appendix 4

4 Evaluation

When evaluating the tracker and the algorithm we are considering the following. Firstly, we have a mapped structure for each blob to each person that updates over time. Iterating through the structure and through the frames of the video we collect data about each person's X and Y

location in a vector, as well as the ground truth for that frame in a separate vector. In the same iteration, we are computing the euclidean distance between the person and its ground truth. Table 4 present our findings.

Person Identifier	# Wrong matches	Mean Within 10px
1(red)	4	4.55178
2(yellow)	16	5.7982
3(blue)	1	27011
4(green)	17	5.4929

From the video we could not identify any incorrect matches even with a big delay. We therefore belief, that it has to be some distance metric that is incorrect, perhaps in the centre of mass of each person. It is important to note that we do not have any erroneous pairings, due to the nature of our algorithm - identifying 4 people at the beginning and maintaining that throughout the run. Furthermore, in our implementation we have 4 distinct trajectories for the 4 people on the video.

In addition, we have modelled the trajectories each of the people dances, one can be seen on Figure 14 and 15. One can notice that on Figure 15, we are actually keeping the label colour on the people throughout the whole time, even for trajectories (Figure 10).

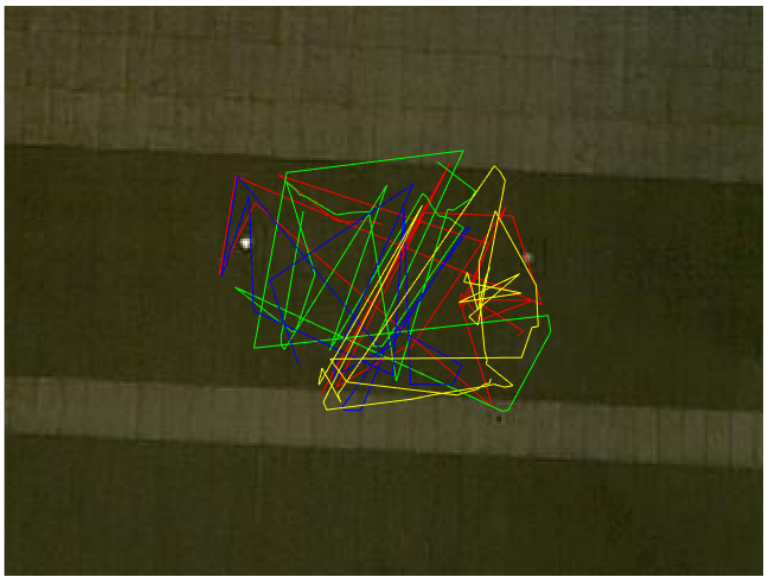


Figure 14: All trajectories of all people on the background

Figure 15: Trajectory of a person by our tracker (yellow) and by the ground truth (white)

References

[Maj09] Barbara Majecka. “Statistical models of pedestrian behaviour in the forum”. In: *Master’s thesis, School of Informatics, University of Edinburgh* (2009).

Appendices

Main function

```
function [] = main()

  global plot_original_img plot_cleaned_img plot_no_bg plot_no_guy plot_offset_img
  global plot_main_blobs plot_center_masses plot_only_blob_plots plot_person_histograms
  global plot_people_with_circles plot_masked_offset_img plot_people_binary_img
  global plot_people_img evaluation_file_name plot_trajectories_fig plot_trajectories2_fig
  plot_original_img=0; % open the original image
  plot_cleaned_img=0; % show the cleaned images
  plot_no_bg=0; % show images with background removed
  plot_no_guy=0; % cutting the stationary guy and small shadow on the left
  plot_main_blobs=0; % display 4 persons main blobs
  plot_center_masses=0; % display original images plus detection circles
  plot_only_blob_plots=0; % displays which Number of blobs after being cropped
  plot_person_histograms=0; % display the histogram of each person with the image itself
  plot_offset_img=0; % displays only the mask applied to the people
  plot_people_with_circles=1; % displays the people with circles
  plot_masked_offset_img=0; % display masked binary circles around each person
  plot_people_binary_img=0; % display background-removed blobs of people (binary)
  plot_people_img=0; % display background-removed blobs of people
  evaluation_file_name='positions1.mat'; % 4x201x2
  plot_trajectories_fig = 4; % where to plot all the trajectories
  plot_trajectories2_fig = [5,6,7,8]; % plot person1's trajectory

  %different images filters for easy detection using on color historgrams
  increase_contrast = @(I) ((I.^1.3)./(255^1.3)) .* 255;
  h = fspecial('gaussian', 5, 1);
  %h = fspecial('disk', 1.5);
  %gauss = @(I) imfilter(I, h,'replicate');
  apply_fltr = @(I) imfilter(increase_contrast(I), h,'replicate');

  % load the background image.
  background = imread('DATA1/bgframe.jpg', 'jpg');
  Imback = double(background);
  color_hist_edges = [0:255];
```

```
first_img_id = 110;
last_img_id = 319;
people_color_hists = {};
%first dimension person id, second x and y coordinates
people_pos = zeros(4, 2);
%the colors assocciated with each person
people_markers = {'r.', 'g.', 'b.', 'y.'};
person_distances = {[],[],[],[]}; % keeps the mean distances between people
person_trajectories_x = {[],[],[],[]}; % keeps the trajectories for people
person_trajectories_y = {[],[],[],[]}; % keeps the trajectories for people
truth_trajectories_x = {[], [], [], []}; % keeps the trajectories for the truth
truth_trajectories_y = {[], [], [], []}; % keeps the trajectories for the truth
min_cumulative_distances = [];
frames = [first_img_id:last_img_id];
%number of observations stored (color histograms)
NUM_TRACK_FRAMES = 10;
%To determines when the frame rate does sudden jump
JUMP_THRESHOLD = 63;
%first axis is the person id, and the second stores the last NUM_TRACK_FRAMES
%persons id estimated through color histograms (observations of our model)
last_observs = zeros(4, NUM_TRACK_FRAMES);

for img_id = frames
  fprintf('image id: %1.0f\n', img_id);

  % load image
  Im = (imread(['DATA1/frame',int2str(img_id), '.jpg'],'jpg'));
  % display image
  plot_img(plot_original_img, Im);
  Imwork = double(Im);

  cleaned_binary_img = clean_image(Imwork,Imback);
  if plot_offset_img > 0
      show_mask_img(plot_offset_img, Im, cleaned_binary_img);
  end

  %filtering background and working image for better color histograms
  filtered_bkg = apply_fltr(Imback);
  filtered_img = apply_fltr(Imwork);

```

```
75      %separating blobs to unidentified people
76      [raw_stats, cropped_images, success] = separate_people(↩
            cleaned_binary_img, uint8(filtered_img));
77      %not success in the case when separation went wrong
78      %for example, 5 blobs were detected
79      if ~success
80        continue;
81      end
82      %if success then we have 4 blobs and 4 persons
83      num_people = 4;
84      %caclulate color histograms of unidentified people
85      %also getting more precise area and position
86      [stats, blob_color_hists, people_binary_img] = extract_people(↩
            cleaned_binary_img,...
87                                                                  raw_stats↩
                                                                        ,...
88                                                                  cropped_images↩
                                                                        ,...
89                                                                  filtered_bkg↩
                                                                        ,...
90                                                                  color_hist_edges↩
                                                                        );
91
92      plot_img(plot_people_binary_img, people_binary_img);
93      if plot_people_img > 0
94          show_mask_img(people_binary_img, Im, covnex_binary_img);
95      end
96
97      if plot_center_masses > 0
98        figure(plot_center_masses);
99        plot_target_circles(stats, Im);
100     end
101
102     if plot_only_blob_plots > 0
103       plot_img(plot_only_blob_plots, cropped_images{plot_only_blob_plots})↩
              ;
104     end
105
106     %creating a map from blobs to people using color histograms
107     if img_id == first_img_id
108       %blob to person estimated through histograms distances
109       blob2person_hist_estim = zeros(num_people, 1);
110       %save initial people color hists
111       people_color_hists = cell(4, 1);
112       people_pos = cat(1, stats.Centroid);
113       for person_id=1:4
114           people_color_hists{person_id} = blob_color_hists{person_id};
```

```
115             blob2person_hist_estim(person_id) = person_id;
116           end
117         else
118           %assigns people to blobs greedily using BHATTACHARYYA color distance
119           blob2person_hist_estim = calc_greedy_matches(people_color_hists, ↩
                blob_color_hists);
120         end
121
122         %blobs positions as matrix
123         blobs_pos = cat(1, stats.Centroid);
124         %find such correspondence between previous people positions and ↩
              current
125         %blobs so the distance between such pairs is minizied. The good thing
126         %that it doesn't not rely on observations (color histograms) so by ↩
              that
127         %value it is possible to judge about framerate jumps.
128         [min_cumulative_dist, person2blob_pos_estim] = ↩
              calc_min_cumulative_dist(people_pos, blobs_pos);
129         %store them for plotting
130         min_cumulative_distances = [min_cumulative_distances, ↩
              min_cumulative_dist];
131
132         is_jump = min_cumulative_dist > JUMP_THRESHOLD;
133         %blob to person final estimation and updating observations
134         [blob2person_finl_estim, last_observs] = tracker(↩
              blob2person_hist_estim,...
135                                                          person2blob_pos_estim↩
                                                                ,...
136                                                          last_observs,...
137                                                          is_jump,...
138                                                          NUM_TRACK_FRAMES)↩
                                                                ;
139
140         if plot_people_with_circles > 0
141             %img = uint8(apply_fltr(Imwork)); %filtered image
142             plot_tracked_people(stats, Im, blob2person_finl_estim, ↩
                    people_markers);
143         end
144
145         %updating person positions
146         for blob_id = 1:4
147           person_id = blob2person_finl_estim(blob_id);
148           people_pos(person_id, :) = blobs_pos(blob_id, :);
149         end
150         %pause(1);
151
152         % compute the trajectories
```

```
    [person_distances, person_trajectories_x, person_trajectories_y, ...
        truth_trajectories_x, truth_trajectories_y] = evaluation(...
        first_img_id, img_id, stats, blob2person_finl_estim, ...
        people_markers, person_distances, person_trajectories_x, ...
        person_trajectories_y, truth_trajectories_x, ...
        truth_trajectories_y);

  end

  calculate_totals(person_distances, people_markers, frames);

  plot_trajectories(background, person_trajectories_x, ↩
     person_trajectories_y,...
      truth_trajectories_x, truth_trajectories_y, people_markers);

  figure;
  stem(frames, min_cumulative_distances);
  xlabel('Frame');
  ylabel('Min. Cumulative Distance in pixels');
end
```

Background Removal

```
% extracts people
function selected=clean_image(Imwork,Imback)
  global plot_cleaned_img plot_main_blobs plot_no_bg plot_no_guy

  % THRESHOLD - the value difference between the foreground and the
  % background
  bg_intensity_threshold = 15;

  % subtract background & select pixels with a big difference
  fore = (abs(Imwork(:,:,1)-Imback(:,:,1)) > bg_intensity_threshold) ...
     | (abs(Imwork(:,:,2) - Imback(:,:,2)) > bg_intensity_threshold) ...
     | (abs(Imwork(:,:,3) - Imback(:,:,3)) > bg_intensity_threshold);
  plot_img(plot_no_bg, fore);

  %removes the left wall completely and the shadow
  %that might appear there
  %cutting_wall_threshold = 125;
  %fore(:,1:cutting_wall_threshold,:) = 0;

  %cutting the 5th stationary person at 150, 223
  %(he is moving that is why we are removing slightly bigger area)
  fore = remove_square(fore, 170, 170, 170);
  plot_img(plot_no_guy, fore);
```

```
  foremm = imerode(fore, strel('disk', 1));
  foremm = imdilate(foremm, strel('disk', 4));

  % show the cleaned images
  plot_img(plot_cleaned_img, foremm);

  % select largest object
  labeled = bwlabel(foremm,8); % 4 - vertical, horizontal clustering
  stats = regionprops(labeled,['basic']);
  N = size(stats, 1);
  if N < 1
    success = 0;
    return %no large objects - nothing to work with
  end

  %sort by stats(i).Area
  [stats, ids] = bubble_sort(stats);
  %first one is the biggest one
  % make sure that there is at least 1 big region
  if stats(1).Area < 100
    return
  end

  %take big regions above the area_threshold
  area_threshold = 150;
  selected = labeled==ids(1);

  for i = 2 : size(stats, 1)
      if stats(i).Area < area_threshold
          %can interrupt because stats was sorted by area
          break;
      end
      selected = selected + (labeled==ids(i));
  end

  % after we combine the labels, expand them a wee bit to
  % to fill holes in people
  selected = imdilate(selected, strel('disk', 4));
  selected = imerode(selected, strel('disk', 4));
  plot_img(plot_main_blobs, selected);
end
```

```
%cutting stationary person (5th one)
%removes square region from the image (img)
%x, y - the coordinates of square center
```

```
%length - the length of the square
function [result] = remove_square(img, x, y, length)
  half = length/2;
  h = size(img, 1);
  w = size(img, 2);
  x_start = max(x - half,1);
  x_end   = min(x + half,w);
  y_start = max(y - half,1);
  y_end   = min(y + half,h);
  img(y_start:y_end, x_start:x_end,:) = 0;
  result = img;
end
```

```
function [ sorted_stats, ids] = bubble_sort(stats)
    % bubble sort (large to small) on regions in case there are more than 1
    N = size(stats, 1);
    ids = zeros(N);
    for i = 1 : N
        ids(i) = i;
    end
    for i = 1 : N-1
        for j = i+1 : N
          if stats(i).Area < stats(j).Area
            tmp = stats(i);
            stats(i) = stats(j);
            stats(j) = tmp;
            tmp = ids(i);
            ids(i) = ids(j);
            ids(j) = tmp;
          end
        end
    end
    sorted_stats = stats;
end
```

Blob Splitting

```
function [f_stats, cropped_imgs, success] = separate_people(cleaned_binary_img, real_img)
  %f_stats - final statistic.
  labeled = bwlabel(cleaned_binary_img, 8); % 8 - vertical, horizontal clustering
  stats = regionprops(labeled, 'Area', 'Centroid', 'BoundingBox', 'MajorAxisLength', 'Orientation');
```

```
5
6   f_stats = [struct(); struct(); struct(); struct()];
7   cropped_imgs = cell(4, 1);
8   num_blobs = size(stats, 1);
9
10  blobs_areas = ones(num_blobs, 2);
11  %concatinate to vector
12  blobs_areas(:,1) = cat(1,stats.Area);
13  %original indeces
14  blobs_areas(:,2) = 1:size(stats,1);
15  %ascending sorting
16  blobs_areas = sortrows(blobs_areas, -1);
17  blobs_indices = blobs_areas(:, 2);
18
19  man_id = 1;
20  %relying on the description that there are 4 persons
21  blobs_to_split = 4 - num_blobs;
22  if blobs_to_split > 2 || num_blobs > 4
23    success = 0;
24    return
25  end
26
27  for i = 1:num_blobs
28    stat_id = blobs_indices(i);
29    s = stats(stat_id);
30    [start_x, start_y, end_x, end_y] = bbox2imgsize(s.BoundingBox);
31    cropped_img = real_img(start_y:end_y, start_x:end_x, :);
32    if blobs_to_split == 0
33      %just copying existing values
34      f_stats(man_id).Area = s.Area;
35      f_stats(man_id).Centroid = s.Centroid;
36      %mask is one, everything remains
37      f_stats(man_id).Mask = 1;
38      f_stats(man_id).BoundingBox = s.BoundingBox;
39
40      cropped_imgs{man_id} = cropped_img;
41    else
42      %splitting ellipsoid which bounds two people
43      %a - above, b - below
44      %f means fokal points of the ellipsis
45      [fa, mask_a, fb, mask_b] = separate_two(size(cropped_img), s);
46      halfArea = s.Area * 0.5;
47      %saving informaton about parts
48      f_stats(man_id).Area = halfArea;
49      f_stats(man_id).Centroid = fa;
50      f_stats(man_id).Mask = mask_a;
51      f_stats(man_id).BoundingBox = s.BoundingBox;
```

```
52        cropped_imgs{man_id} = cropped_img;
53
54        man_id = man_id + 1;
55
56        f_stats(man_id).Area = halfArea;
57        f_stats(man_id).Centroid = fb;
58        f_stats(man_id).Mask = mask_b;
59        f_stats(man_id).BoundingBox = s.BoundingBox;
60        cropped_imgs{man_id} = cropped_img;
61
62        blobs_to_split = blobs_to_split - 1;
63      end
64      man_id = man_id + 1;
65    end
66    success = 1;
67  end
```

```
1  function [fa, mask_a, fb, mask_b] = separate_two(img_size, s)
2  %img - cropped image containing two people,
3  %s - stat struct containing 'Area', 'Centroid', 'BoundingBox', '↩
       MajorAxisLength', 'Orientation'
4  %return fa - global position of fist person, fb - global position of ↩
       second
5  %person. mask_a, mask_b - image offsets of size I.
6
7  %small space between separated people
8  SPACE = 2;
9
10 %y an x vice versa because image first dimension is height
11 %http://stackoverflow.com/questions/19992097/how-to-do-circular-crop-using↩
       -matlab
12 [yy, xx] = ndgrid((1:img_size(1)), (1:img_size(2)));
13 %flip orientation due to the same reason
14 teta = 180 - s.Orientation;
15 box_pos = s.BoundingBox(1:2);
16 a = cosd(teta);
17 b = sind(teta);
18 %centroid local
19 cl = s.Centroid - box_pos;
20
21 d = a * cl(1) + b * cl(2);
22 mjLen = s.MajorAxisLength;
23 %major axis vector
24 %0.5 * 0.5 because axis by half and then middle of that
25 mjAxis = [a * mjLen, b * mjLen] * 0.5 * 0.5;
```

```
26
27 %b - below, a - above
28 %mask size of I, using normalized equation of the line
29 mask_a = uint8(((xx.*a + yy.*b) - d) >  SPACE);
30 mask_b = uint8(((xx.*a + yy.*b) - d) < -SPACE);
31 %focus local
32 fla = cl + mjAxis;
33 flb = cl - mjAxis;
34 %focus global
35 fa = fla + box_pos;
36 fb = flb + box_pos;
37 end
```

Hierarchical Background Subtraction

```
1  function [stats, hists, people_binary_img] = extract_people(↩
       cleaned_binary_img, raw_stats, cropped_images, bg_img, edges)
2    %color histograms
3    hists = cell(size(raw_stats,1), 1);
4    BG_INTENSITY_THRESHOLD = 20; % used for bg removal
5    MIN_PERSON_AREA = 150;
6    % used for debugging and evaluation of the area which people take
7    people_binary_img = zeros(size(cleaned_binary_img));
8
9    for person_id = 1:size(raw_stats,1)
10     bbox = raw_stats(person_id).BoundingBox;
11     [start_x, start_y, end_x, end_y] = bbox2imgsize(bbox);
12     %applying binary mask which will separate people in one blob
13     person_mask = cleaned_binary_img(start_y:end_y, start_x:end_x) & ↩
           raw_stats(person_id).Mask;
14     %save mask value in order to preserve area if it shrinks too much
15     tmp = person_mask;
16
17     %substracting background once more in order to obtain
18     %better contour of the person and more precise color distribution
19     croped_img = double(cropped_images{person_id});
20     croped_bkg = bg_img(start_y:end_y, start_x:end_x, :);
21     croped_bkg = (abs(croped_img(:,:,1) - croped_bkg(:,:,1)) > ↩
           BG_INTENSITY_THRESHOLD) ...
22                 |(abs(croped_img(:,:,2) - croped_bkg(:,:,2)) > ↩
                       BG_INTENSITY_THRESHOLD) ...
23                 |(abs(croped_img(:,:,3) - croped_bkg(:,:,3)) > ↩
                       BG_INTENSITY_THRESHOLD);
24
25     person_mask = person_mask & croped_bkg;
26     %here we don't allow person to shrink too much
```

```
    %nnz give numbe of white pixels -> basically area
    if nnz(person_mask) < MIN_PERSON_AREA
        person_mask = tmp;
    end

    %erode and dilate with different brushes to obtain
    %better contours
    tmp = person_mask;
    person_mask = imerode(person_mask, strel('disk', 3));
    person_mask = imdilate(person_mask, strel('disk', 1));

    if nnz(person_mask) < MIN_PERSON_AREA
        person_mask = tmp;
    end

    %filling possible holes in the person
    person_mask = bwconvhull(person_mask);

    %update person parameters after filtering
    % get local center of mass
    [xcentre_local, ycentre_local] = calc_center_binary_img(person_mask);
    %switch to global coordinates using bounding box position as
    %it determines local coordinate system position
    raw_stats(person_id).Centroid = [xcentre_local, ycentre_local] + bbox(1:2);
    raw_stats(person_id).Area = nnz(person_mask);

    %save the resulting binary mask via logical addition
    %allows to see resulting people contours
    people_binary_img(start_y:end_y, start_x:end_x) = people_binary_img(start_y:end_y, start_x:end_x) | person_mask;

    %computing color histograms
    hists{person_id} =  calc_hist(person_mask, croped_img, edges);
  end
  %update people stats with new more precise area and positions
  stats = raw_stats;
end
```

```
function [hist] = calc_hist(binary_mask,  img, edges)
  %computing color histograms
  mask_vect = reshape(binary_mask, [], 1);
  non_bg_zero_ids = find(mask_vect);

  red_chnl_raw   = reshape(img(:, :, 1), [], 1);
```

```
  green_chnl_raw = reshape(img(:, :, 2), [], 1);
  blue_chnl_raw  = reshape(img(:, :, 3), [], 1);

  %roughly pixel are of the person
  num_person_pixels = numel(non_bg_zero_ids);

  red_chnl   = red_chnl_raw(non_bg_zero_ids);
  green_chnl = green_chnl_raw(non_bg_zero_ids);
  blue_chnl  = blue_chnl_raw(non_bg_zero_ids);

  histR = histc(red_chnl,edges);
  histG = histc(green_chnl,edges);
  histB = histc(blue_chnl,edges);

  %normalize each channel to 1
  histR = histR / num_person_pixels;
  histG = histG / num_person_pixels;
  histB = histB / num_person_pixels;
  hist = [histR, histG, histB];%[histR, histG, histB];
  %normalize it as whole to 1
  hist = hist ./ size(hist,2);
end
```

```
function [ xcentre, ycentre ] = calc_center_binary_img(I)
%I - binary image
%remember that image first size is height and second is width
[xx, yy] = meshgrid(1:size(I, 2), 1:size(I, 1));
weightedx = xx .* I;
weightedy = yy .* I;
xcentre = sum(weightedx(:)) / sum(I(:));
ycentre = sum(weightedy(:)) / sum(I(:));
end
```

```
function [start_x, start_y, end_x, end_y] = bbox2imgsize(bbox)
  %bbox - bounding box
  %so bounding box to image size (accounting all upper
  %and lower bounds properly)
  start_x = floor(bbox(1));
  start_y = floor(bbox(2));
  end_x = start_x + ceil(bbox(3));
  end_y = start_y + ceil(bbox(4));
end
```

Blobs to people assignment

```
function [blob2person] = calc_greedy_matches(people_color_hists, ↩
    blob_color_hists)
  %calculates greedily the best mathc of blobs to
  %people based on color histograms
  num_people = size(people_color_hists, 1);
  blob2person = zeros(num_people, 1);
  %table of matches
  %1st column - color distance (bhattacharyya)
  %2nd column - person id
  %3th column - blob id
  %4th column - presence flag
  matches = zeros(num_people * num_people, 4);
  for person_id=1:num_people
      person_hist = people_color_hists{person_id};
      %flatten matrix to 1xN vector
      person_color_vect = reshape(person_hist, 1, []);
      for blob_id=1:num_people
          blob_hist = blob_color_hists{blob_id};
          blob_color_vect = reshape(blob_hist, 1, []);
          row = (person_id - 1) * num_people + blob_id;
          color_dist = bhattacharyya(person_color_vect, blob_color_vect);
          matches(row, 1) = color_dist;
          matches(row, 2) = person_id;
          matches(row, 3) = blob_id;
          matches(row, 4) = 1;
      end
  end
  %sort rows in ascending order by color distance
  matches = sortrows(matches, 1);
  %keep finding matches until all blobs and people are connected
  while any(matches(:, 4) > 0)
      %find row with smallest distance among remaing ones
      row_id = find(matches(:,4), 1);
      person_id = matches(row_id, 2);
      blob_id = matches(row_id, 3);
      blob2person(blob_id) = person_id;
      %mark the matched pairs, remove them
      %from future matching
      same_person = matches(:,2)==person_id;
      same_blob = matches(:,3)==blob_id;
      matches(same_person, 4)=0;
      matches(same_blob, 4)=0;
  end
end
```

```
function [cumulative_dist, person2blob_pos_estim] = calc_min_cumulative_dist(people_pos, blobs_pos)
  %best_variant is 4 vector mapping person id to best blob id
  %(from distance point of view)

  %D(person_id, blob_id) - distance between person i previous position
  %and blob j current position
  D =  zeros(4, 4);
  for person_id = 1:4
   for blob_id = 1:4
     D(person_id, blob_id) = sqrt(sum((people_pos(person_id, :) - blobs_pos(blob_id, :)) .^ 2));
   end
  end

  %all possible variants - permutations between blobs and people
  variants = perms([1 2 3 4]);
  total_dists = zeros(size(variants, 1), 1);
  for variant_id = 1:size(variants, 1)
    for person_id = 1:4
      blob_id = variants(variant_id, person_id);
      total_dists(variant_id) = total_dists(variant_id) + D(person_id, blob_id);
    end
  end

  [cumulative_dist, min_variant_id] = min(total_dists);
  person2blob_pos_estim = variants(min_variant_id, :);
end
```

```
function [blob2person_finl_estim, new_observs] = tracker(blob2person_hist_estim,...
                                                         person2blob_pos_estim,...
                                                         last_observs,...
                                                         is_jump,...
                                                         NUM_TRACK_FRAMES)

  %The idea is that if for NUM_TRACK_FRAMES frames our position estimation
  %is in the good match with our observations via color histograms stored in
  %the last_observs matrix then we can folow our position estimation
  %relying on the fact that the movements are continious. However, if the
  %jump in framerate has occured then we can't rely on position estimation
  %and we use only color hystograms.
```

```
13
14    %final maping between blobs and people;
15    blob2person_finl_estim = zeros(4, 1);
16    %whether we can rely on positional estimations or not
17    reset_observ_flag = is_jump;
18    if ~reset_observ_flag
19      %no jump so person2blob_pos_estim is reliable sourse of data
20      %update our observations and ensure that positions are in good
21      %match with observations
22      for person_id = 1:4
23        blob_id = person2blob_pos_estim(person_id);
24        %observer person id via color histograms
25        person_id_from_observ = blob2person_hist_estim(blob_id);
26        %shifting old observations to the past
27        last_observs(person_id, 1:end-1) = last_observs(person_id, 2:end);
28        last_observs(person_id, end) = person_id_from_observ;
29        %determing the most frequent observation (color histograms)
30        %in order to determine how confident we are in our current position
31        observs = last_observs(person_id, :);
32        observs_without_zeros = observs(observs > 0);
33        most_freq_person = mode(observs_without_zeros);
34        if most_freq_person ~= person_id && ~isnan(most_freq_person)
35          reset_observ_flag = 1;
36          fprintf('observations for person %s diverged\n', num2str(person_id↩
              ));
37          break;
38        end
39      end
40      %in good match with observations so creating blob2person_finl_estim
41      %based on positions estimation
42      for person_id = 1:4
43        blob_id = person2blob_pos_estim(person_id);
44        blob2person_finl_estim(blob_id) = person_id;
45      end
46    end
47
48    if reset_observ_flag
49      %the big jump in positions between this and previous frames
50      %have occured. Or some persons started swapping their positions.
51      %Can't trust position estimation anymore -
52      %relying on color histograms observations only
53      fprintf('relying only on observatons\n');
54      last_observs = zeros(4, NUM_TRACK_FRAMES);
55      blob2person_finl_estim = blob2person_hist_estim;
56      for blob_id = 1:4
57        person_id = blob2person_finl_estim(blob_id);
58        %starting estimation from scratch we belive that this is we are.
```

```
59        last_observs(person_id, end) = person_id;
60      end
61    end
62    %updating observations
63    new_observs = last_observs;
64  end
```

Plotting Util

```
1  function [] = plot_img(yes_id, img)
2    if yes_id > 0
3      figure(yes_id)
4      clf
5      imshow(img)
6      %eval(['imwrite(uint8(fore),''BGONE/nobg',int2str(index),'.jpg'',''jpg↩
           '')']);
7    end
8  end
```

```
1  function [] = plot_target_circles(stats, img)
2      clf
3      imshow(img);
4      hold on;
5      for j=1:size(stats, 1)
6        centroid = stats(j).Centroid;
7        radius = sqrt(stats(j).Area/pi);
8        ang= 0:0.01:2*pi;
9        c = radius * cos(ang);%-0.97*radius: radius/3 : 0.97*radius;
10       r = radius * sin(ang);
11       cc = centroid(1);
12       cr = centroid(2);
13       plot(cc+c,cr+r,'g.');
14       plot(cc+c,cr-r,'g.');
15     end
16     hold off;
17     drawnow;
18 end
```

```
1  function [] = plot_tracked_people(stats, img, blob2person, people_markers)
2      clf
3      imshow(img);
4      hold on;
5      for blob_id=1:size(stats, 1)
```

```
6         centroid = stats(blob_id).Centroid;
7         radius = sqrt(stats(blob_id).Area/pi);
8         ang= 0:0.01:2*pi;
9         c = radius * cos(ang);%-0.97*radius: radius/3 : 0.97*radius;
10        r = radius * sin(ang);
11        cc = centroid(1);
12        cr = centroid(2);
13        person_id = blob2person(blob_id);
14
15        if person_id > 4
16          'Person ID is too big'
17        end
18        if person_id > 0
19          marker = people_markers{person_id};
20
21          %plot(cc+c,cr+r, marker);
22          plot(cc+c,cr+r, marker);
23        end
24      end
25      hold off;
26      drawnow;
27  end
```

```
1   function [] = plot_trajectories( bg_img, people_trajectories_x, ↩
        people_trajectories_y, truth_trajectories_x, truth_trajectories_y, ↩
        people_markers )
2   %PLOT_TRAJECTORIES Plots the trajectories for each person
3       global plot_trajectories_fig evaluation_file_name ↩
            plot_trajectories2_fig
4
5       % plot trajectories
6       % draw people trajectories on one image
7       disp(['Plotting all peoples trajectories on one image.']);
8       figure(plot_trajectories_fig);
9       clf;
10      imshow(bg_img);
11      hold on;
12      for person_id=1:size(people_trajectories_x, 2)
13          % plot trajectories
14          % returns [cc,cr]
15          X = people_trajectories_x{person_id};
16          Y = people_trajectories_y{person_id};
17          colour = strtok(people_markers{person_id},'.');
18          plot(X, Y, colour);
19      end
```

```
20      hold off;
21      drawnow;
22
23      input('Please press enter');
24      % get the X and Y of the ground truth
25      load(evaluation_file_name); % load the matrix
26      for person_id=1:size(people_trajectories_x, 2)
27
28          disp(['Plotting Person ', num2str(person_id) ,' trajectories on ↩
                one image.']);
29          figure(plot_trajectories2_fig(person_id));
30          imshow(bg_img);
31          hold on;
32          X = people_trajectories_x{person_id};
33          Y = people_trajectories_y{person_id};
34          colour = strtok(people_markers{person_id},'.');
35          plot(X, Y, colour);
36
37          x_truth = truth_trajectories_x{person_id};
38          y_truth = truth_trajectories_y{person_id};
39          colour = 'w';
40          plot(x_truth, y_truth, colour);
41          hold off;
42          drawnow;
43      end
44
45
46
47  end
```