Computer Vision Lab Assignment 5 Report

lijiaj@student.ethz.ch

November 2023

1 Condensation Algorithm Implementation

In this assignment, I was required to implement the CONDitional DENSity propagaTION over time (Condensation) algorithm. With the given framework code, I only needed to implement `color_histogram`, `propagate`, `observe`, `estimate`, and `resample` functions.

1.1 color_histogram function

The `color_histogram(xmin, ymin, xmax, ymax, frame, hist_bin)` function is designed to compute the normalized RGB color histogram within a specified bounding box defined by `xmin`,`ymin`, `xmax`, and `ymax` in a video frame `frame`. Specifically, it bins each color channel (R, G, B) into a specified number of bins `hist_bin`. In my implementation, I first cropped the given frame to the bounding box by `cropped_frame = frame[ymin:ymax, xmin:xmax]` and then calculated the histogram for each color channel with the help of `np.histogram` function. Finally, I further normalized the color histogram by `hist_normalized = hist / np.sum(hist)`.

1.2 propagate function

The `propagate(particles, frame_height, frame_width, params)` function is designed to propagate the particles (samples) based on the system model (Matrix A) and the system noise. In particular, I considered the twp prediction model types separately by checking the value of `params["model"]`.

If `params["model"]` was 0 (i.e. no motion just noise), the matrix A was implicitly defined as the identity matrix I. Then I looped over every particle (sample), generated Gaussian position noise by `noise = np.random.normal(0, params["sigma_position"], 2)` for every particle, and added the noise to the corresponding particle.

On the other hand, if `params["model"]` was 1 (i.e. constant velocity motion model), the matrix A was explicitly defined as in equation 1. The reason why I defined the matrix A in this way was that multiplying this matrix A by the state vector s (of length 4) was equivalent to updating the x-position and y-position by adding x-velocity and y-velocity to them separately without changing the velocities, namely constant velocity motion model. In my code, I also looped over particles in the constant velocity motion model case. Since the state vector in the constant velocity motion model had 4 elements (positions and velocities), the Gaussian noise was from `noise = np.random.normal(0, [params["sigma_position"], params["sigma_position"], params["sigma_velocity"],` params["sigma_velocity"]], 4). Then, the constant velocity motion plus Gaussian noise of particles was implemented by np.dot(A, particles[i]) + noise.

Finally, I used the np.clip function to ensure every particle's center lies inside the frame in both no-motion and constant-velocity motion cases.

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1}$$

1.3 observe function

The observe(particles, frame, bbox_height, bbox_width, hist_bin, hist_target, sigma_observe) function is designed to update the weights of the particles based on the similarity of their color histograms to the target histogram. In the implementation, I first iterated over all particles. For each particle, I refined its bounding box to make sure it was still inside the frame, computed the color histogram in the refined bounding box by calling color_histogram function, calculated the χ^2 distance between the color histogram of the particle and target color histogram by invoking the chi2_cost function, and updated the weight of the particle using the formula $\frac{1}{\sqrt{2*\pi*\sigma}} * \exp(\frac{-\chi^2}{2*\sigma^2})$. Finally, I normalized the weights with particles_w /= np.sum(particles_w).

1.4 estimate function

The estimate(particles, particles_w) function is designed to estimate the mean state of particles (samples) based on their weights. This can be easily achieved by mean_state = np.average(particles, axis=0, weights=particles_w.flatten()).

1.5 resample function

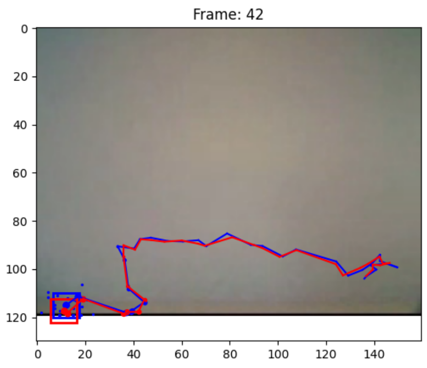
The resample(particles, particles_w) function is designed to resample particles based on their weights and return the new particle set as well as their weights. In my implementation, I first normalized the particle weights with the code of particles_w = particles_w / np.sum(particles_w) in case the input weights were not normalized. Then I resampled the particles based on their weights using np.random.choice function. After that, I normalized the weights of the resampled particles by resampled_weights /= np.sum(resampled_weights) as a good practice. In the end of the function, the resampled particles and normalized corresponding weights were returned.

2 Experiment

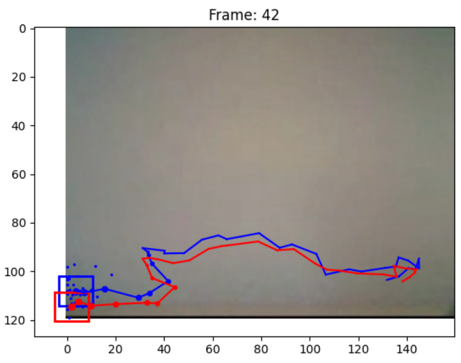
There are 3 videos provided in the assignment files. The first video **video1.avi** shows a moving hand with a uniform background. The second video **video2.avi** contains a moving hand with some clutter and occlusions. The third video **video3.avi** is about a bouncing ball.

The hand in the first video **video1.avi** was relatively easy to track. Using the no motion model, the algorithm with the given default parameters could already track the movement of the hand pretty well. However, if I decreased sigma_position value from

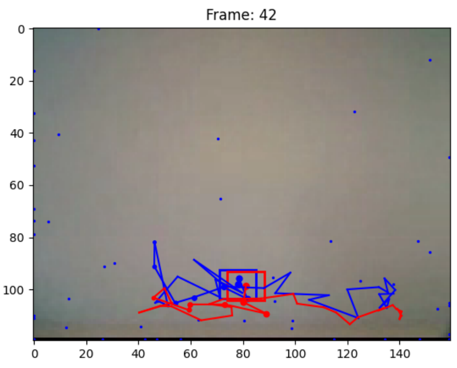
15 to 5, the tracking trajectory would be more stable. As for the constant velocity motion model, I got the desirable result by modifying the `initial_velocity` to $(-2, -1)$ and decreasing the `sigma_position` value from 15 to 5. The successful and failed tracking trajectories of `video1.avi` are shown in Figure 1.



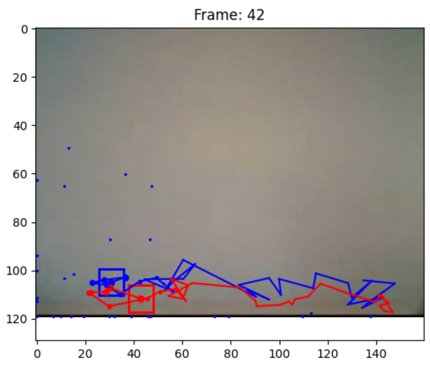
(a) No Motion Model Successful



(b) Constant Motion Model Successful



(c) No Motion Model Failed

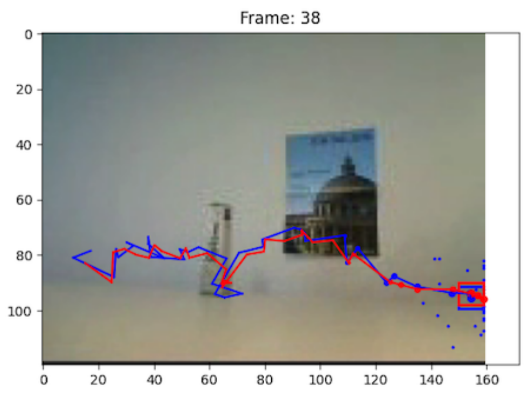


(d) Constant Motion Model Failed

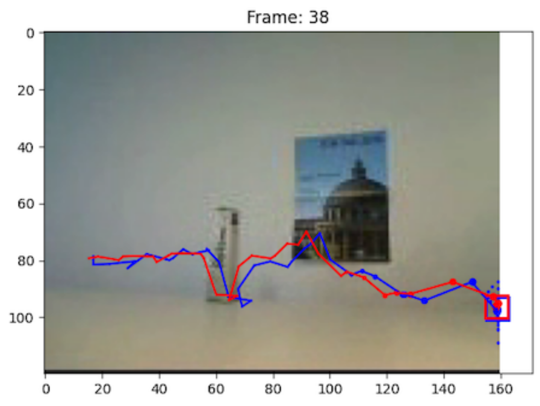
Figure 1: Tracking Trajectories of Video 1

The hand in the second video **video2.avi** was hard to track because of the occlusions and background change during its movement. With the no mition model, if we stuck to the default parameter values, it could not track the movement in the vertical direction (y direction) accurately. I obtained a better result by setting the `hist_bin` to 4 and decreasing the `sigma_position` to 10. When it comes to the constant velocity motion model, the algorithm didn't perform well with the default parameter values. Therefore, I changed the `initial_velocity` to $(2, -1)$, set the `hist_bin` to 8, and decreased `sigma_position` to 5 to achieve a much better tracking effect. The successful and failed tracking trajectories of `video2.avi` are shown in Figure 2. When experimenting with different parameter values, I found the answers to the three questions listed in the assignment sheet for the `video2.avi`. First, what is the effect of using a constant velocity motion model? The tracking trajectory of the constant velocity motion model with appropriate initial velocities was more stable and accurate than that of no motion model within first a few frames. Second, what is the effect of assuming decreased/increased system noise? Increasing the system noise, including `sigma_position` and `sigma_velocity`, allowed particles to move more randomly, which could adapt sudden movements but performed badly on this video.

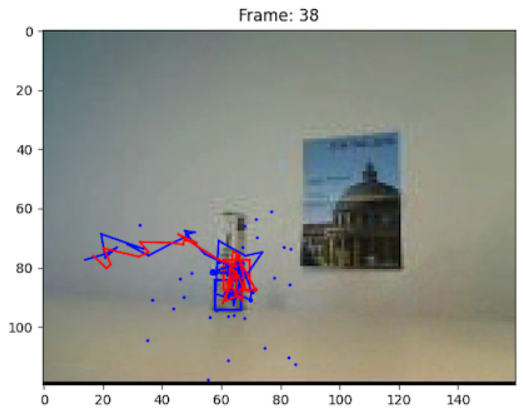
Third, what is the effect of assuming decreased/increased measurement noise? Increasing the measurement noise, namely `sigma_observe`, allowed for more variation in observations, which could lead to bad performance on this video.



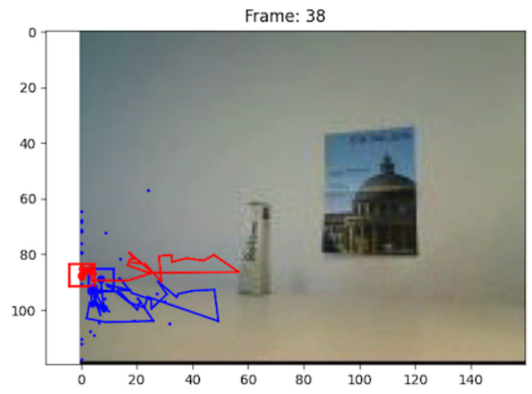
(a) No Motion Model Successful



(b) Constant Motion Model Successful



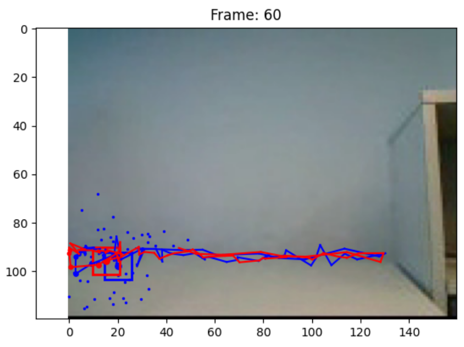
(c) No Motion Model Failed



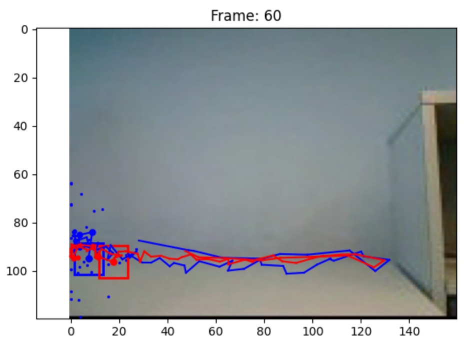
(d) Constant Motion Model Failed

Figure 2: Tracking Trajectories of Video 2

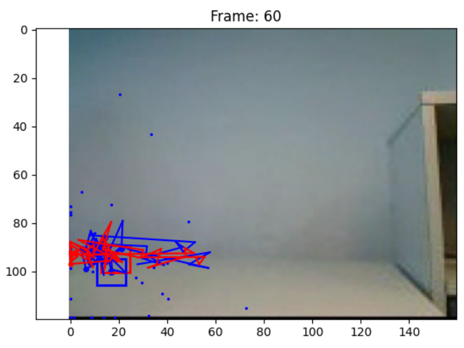
It was hard to use the best parameters tuned in the `video2.avi` to track the bouncing ball in the `video3.avi` well because they had very different environment setting. For example, there was no occlusion and background change in `video3.avi`. To obtain a satisfactory tracking effect with the no motion model, I increased the `num_particles` from 30 (default value) to 50 and decreased the `sigma_position` to 10 from 15 (default value). In terms of the constant velocity motion model case, I set the `initial_velocity` to $(1, 0)$, increased the `num_particles` to 40, and decreased the `sigma_position` to 10 from the default setting. The successful and failed tracking trajectories of `video3.avi` are shown in Figure 3. Again, when experimenting with different parameter values, I found the answers to the same three questions on the `video3.avi`. First, what is the effect of using a constant velocity motion model? I couldn't see any obvious effect of using constant motion model on the `video3.avi`. Second, what is the effect of assuming decreased/increased system noise? Increasing the system noise, including `sigma_position` and `sigma_velocity`, allowed particles to move more randomly, which could adapt sudden movements but performed badly on this video. Third, what is the effect of assuming decreased/increased measurement noise? Increasing the measurement noise, namely `sigma_observe`, allowed for more variation in observations, which could lead to bad performance on this video.



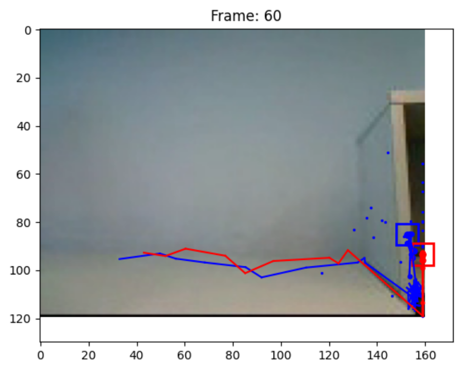
(a) No Motion Model Successful



(b) Constant Motion Model Successful



(c) No Motion Model Failed



(d) Constant Motion Model Failed

Figure 3: Tracking Trajectories of Video 3

Below are three additional questions asked at the end of the assignment sheet and my answer to them. First, what is the effect of using more or fewer particles? More particles could lead to a more accurate representation of the probability distribution at the cost of higher computational overhead. Second, what is the effect of using more or fewer bins in the histogram color model? Using fewer bins in the color histogram results in coarser histograms, which might be less sensitive to color variants. Third, what is the advantage/disadvantage of allowing appearance model updating? One disadvantage of using appearance model updating (i.e. setting `alpha` value greater than 0) is it might cause the tracking bounding box to get stuck with the occlusion.