CV: Lab 06 Writeup

Benedict Armstrong
benedict.armstrong@inf.ethz.ch

Condensation Tracker

This weeks task was to implement a condensation tracker.

Implementation

The implementation of the missing functions was relatively straight forward.

Color Histogram

As suggested in the task description i started with implementing `color_histogram()`. This is relatively simple with numpy's builtin `histogramdd()` function. The output of `histogramdd()`is then stacked into one big histogram.

Matrix A

Next step was deriving matrix a from the following equation.

$$s_t^{(n)} = As_{t-1}^{(n)} + w_{t-1}^{(n)}$$

There are two cases to consider. The first case is when we only use some noise on the particle position and ignore the velocity. In this case A is the Identity. The second case is when we use the velocity to help predict the next position. In this case

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

and every

$$s_t = (x \;\; y \;\; v_x \;\; v_y)^T$$

.

Propagation

The propagation is done by simply multiplying the particles with the matrix A and adding some noise. The noise is sampled from a gaussian distribution with mean 0 and a parameter for the variance. Noise is added both to the velocities and the positions.

Observation

Here we compute the histogram for every particle and compare it to a reference histogram. We then assign a new weight to every histogram we see based on the distance to the reference. The distance is computed using the χ^2 distance. The weights are then normalized.

If all histograms have very low weight we might hit the limit of `floats` and get `nan` values. To avoid this we check if `np.sum(particles_w) == 0` and if so we reset all weights. This usually indicates that our tracked object has left the scene.

Estimation

The estimation is done by computing the weighted mean of the particles. This is done by multiplying every particle with its weight and then summing up all particles.

Resampling

The resampling is done by sampling from the current particles with the weights as probabilities. This is done using `np.random.choice()`.

Experiment

Now for the fun part: trying it out!

To better visualize the results I manually tracked the objects to create a ground truth reference. This can then be used to calculate the difference for each run.

Video 1

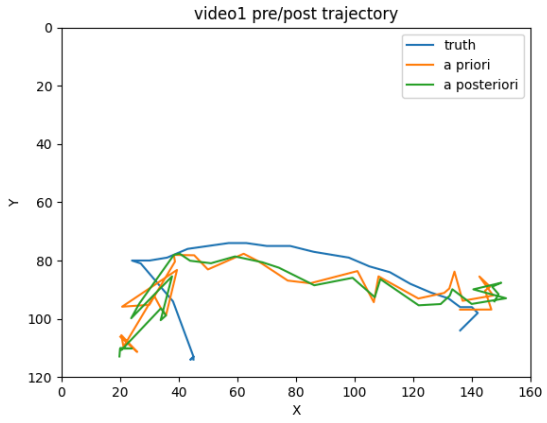

Figure 1: Trajectories pre/post vs. truth

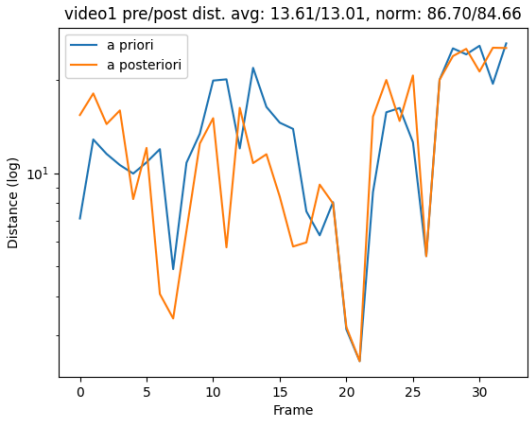

Figure 2: Distance from truth pre/post

With settings:

"hist_bin": 16,
"alpha": 0.3,
"sigma_observe": 0.2,
"model": 0,
"num_particles": 40,
"sigma_position": 15,
"sigma_velocity": 5,
"initial_velocity": (0, 0),

As we can see in the figures above we get a pretty good result. The trajectories look similar to the ground truth and the distance is pretty low. Seems like it's working. A gif of the tracking can be found on github.

Video 2

The best tracking was achieved with the following settings for video3. For the following experiments these settings were used except for the parameters varied.

"draw_plots": 1,
"hist_bin": 32,
"alpha": 0.1,
"sigma_observe": 1.5,
"model": 1,
"num_particles": 100,
"sigma_position": 20,

“sigma_velocity”: 0.5,
“initial_velocity”: (5, −10),

Both with and without the constant velocity model the tracking worked best with a high sigma position. If it is high enough some particles will see “past” the occlusion and can then track the object again.

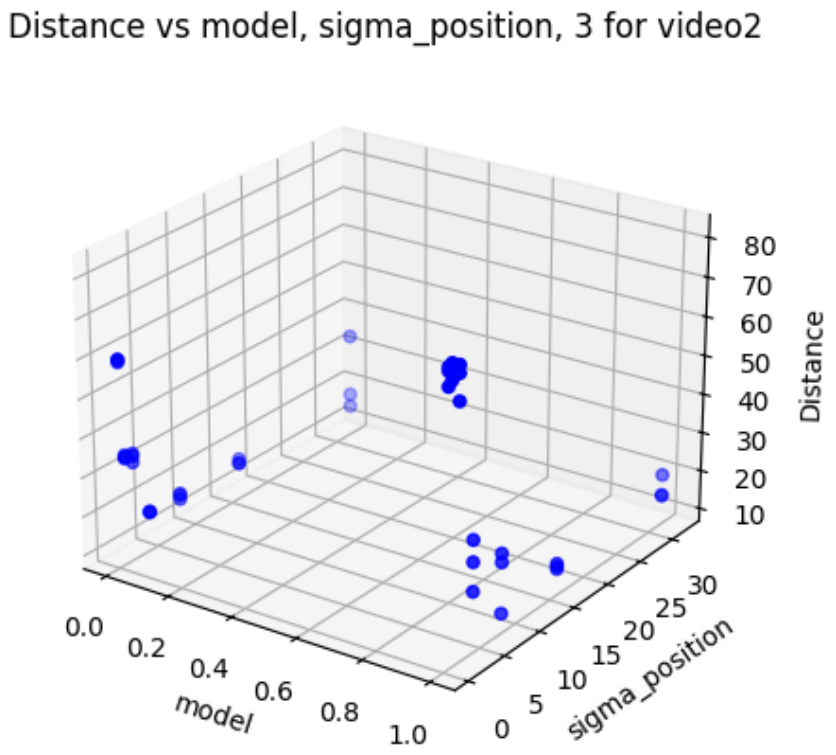

Figure 3: Average distance from truth with different sigma position

In the following graph we can see that for both models a sigma observe in the range [0.3, 2] seems to be ideal.

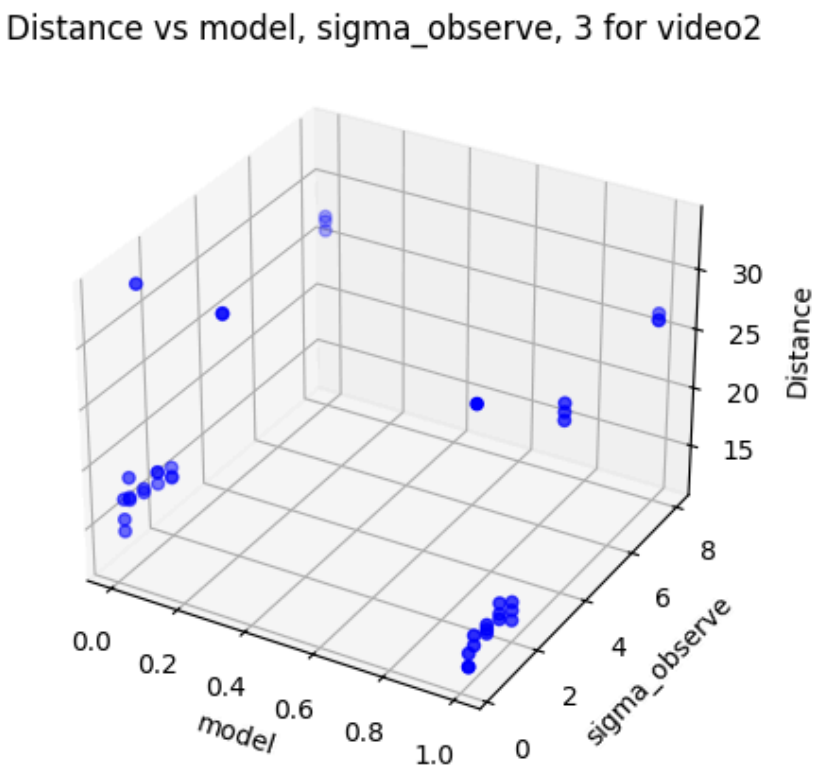

Figure 4: Average distance from truth with different sigma observe

Video 3

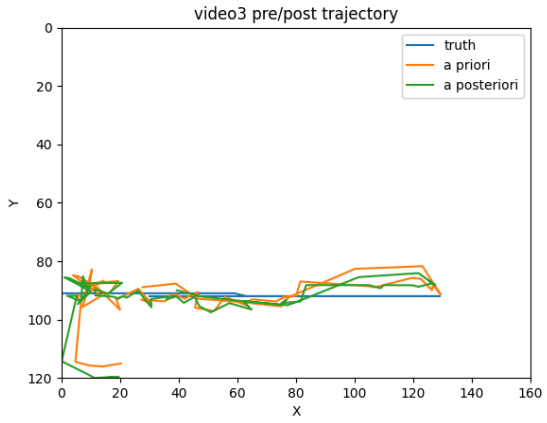

Figure 5: Trajectories pre/post vs. truth (Model 0)

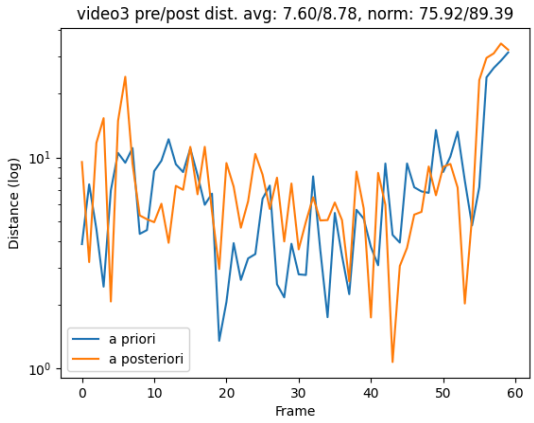

Figure 6: Distance from truth pre/post (Model 0)

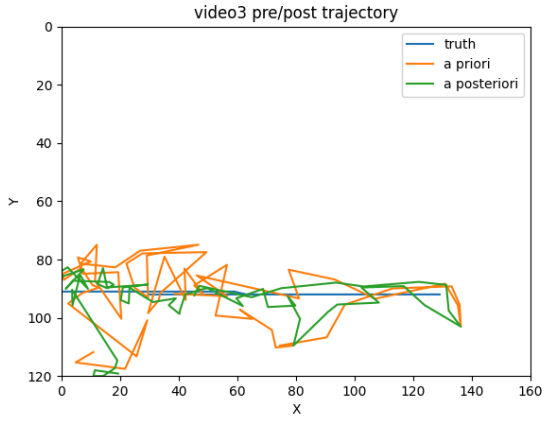

Figure 7: Trajectories pre/post vs. truth (Model 1)

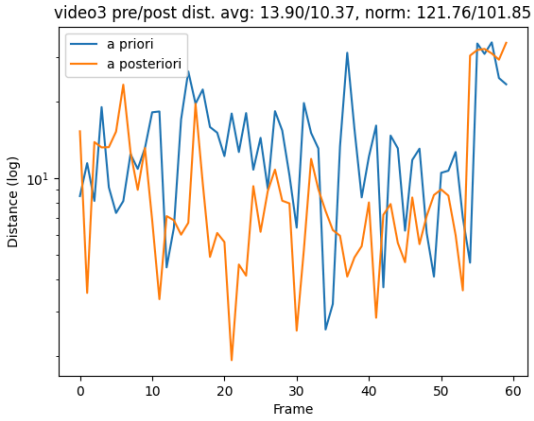

Figure 8: Distance from truth pre/post (Model 1)

With settings:

"draw_plots": 1,
"hist_bin": 16,
"alpha": 0,
"sigma_observe": 0.2,
"model": 0/1,
"num_particles": 40,
"sigma_position": 10,
"sigma_velocity": 10,
"initial_velocity": (10, 0),

As you can see adding the extra noise with the position leads to worse results. The ball is moving at a contant speed until it changes direction. The model with the velocity struggles to follow the abrupt change in direction. The model without velocity is able to follow the ball better.

Additional questions

What is the effect of using more or fewer particles?

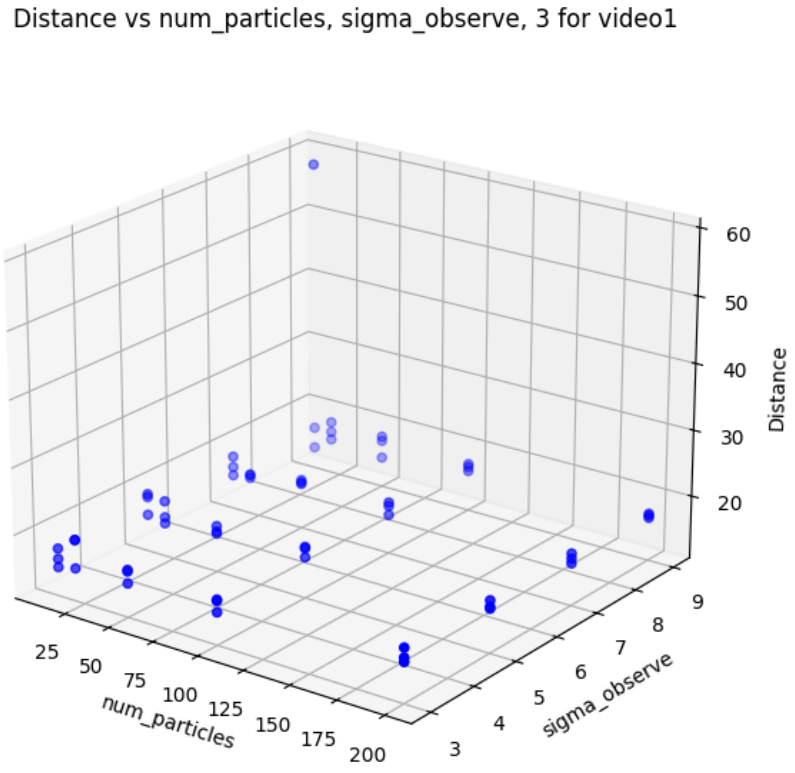

Figure 9: Average distance from truth pre/post with different number of particles and sigma observe (repeated 3 times)

Params used:

"draw_plots": 1,
"hist_bin": 32,
"alpha": 0.3,
"model": 0,
"sigma_position": 10,

As we can see the number of particles has fairly little impact on performance for video1. Using very few particles leads to unpredictable results.

What is the effect of using more or fewer bins in the histogram color model?
Using more bins seems to lead to better results:

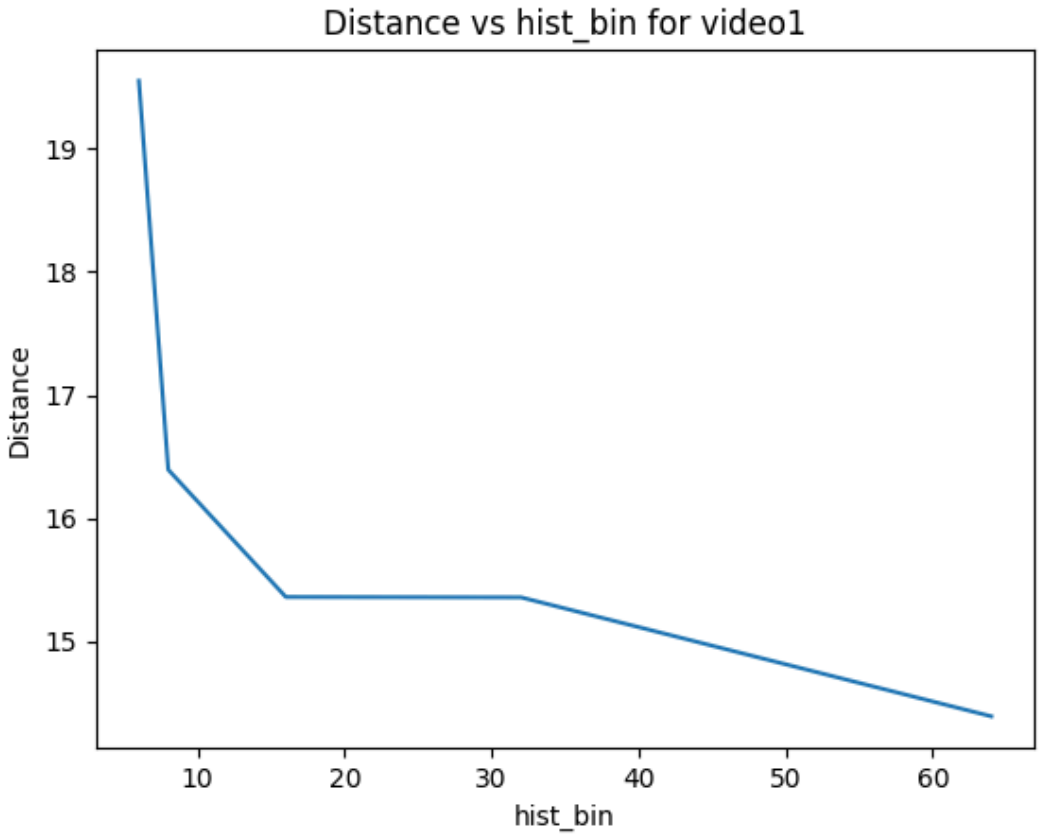

Figure 10: Distance from truth pre/post with different number of bins

Params used:

"draw_plots": 1,
"alpha": 0.3,
"sigma_observe": 0.2,
"num_particles": 50,
"sigma_position": 10, "model": 0,

What is the advantage/disadvantage of allowing appearance model updating?
Allowing the model to update is good if lighting changes (video 1 and 2) but leads to worse results for video 3 because the object is fairly constant.