Computer Vision
ETH Zurich, Fall 2023
Hardik Shah

Exercise 6: Condensation Tracker

Hardik Shah

1 Condensation Tracker Algorithm

The task was to implement the condensation tracker algorithm, a probabilistic framework widely used for object tracking in computer vision. This tracker employs particle filtering techniques, where a set of particles represents potential object states. The primary goal is to estimate the state of a tracked object over time by iteratively updating these particles based on observations and motion models.

The tracker focuses on a single object represented by bounding boxes with states defined as $x, y, \dot{x}, \dot{y}$, where (x, y) denotes the center coordinates of the bounding box, and $(\dot{x}, \dot{y})$ represents the velocities in the x and y directions. The tracking algorithm incorporates color histograms to assess the similarity between particles and the target model, utilizing the χ^2 distance for histogram comparison.

The tracking pipeline involves sampling, prediction, update, and estimation steps in each iteration. The motion model includes two prediction models: no motion (modeled by noise) and constant velocity motion. The tracker prefers particles with color histograms similar to the target model, assigning weights to particles based on the χ^2 distance. Resampling ensures that particles with higher weights are selected more frequently.

The code provides functions for essential operations in the tracker, including particle propagation, observation, estimation, and resampling. Additionally, functionalities for deriving the transition matrix A, calculating color histograms, and computing the χ^2 distance are implemented.

2 Implementation and Algorithm Outline

The functions used to implement the algorithm are described below:

1. `color_histogram(xmin, ymin, xmax, ymax, frame, hist_bin)`
   This method calculates the normalized histogram of RGB colors within a bounding box in a frame.
   (a) Extracts the region of interest from the frame using the specified bounding box coordinates.
   (b) Computes separate histograms for each color channel (R, G, B) given the number of bins `hist_bin`.

(c) Concatenates the histograms to form a single histogram, which is then normalized.

2. `derive_matrix_A(params)`
This is a custom function implemented in *propagate.py*, which derives the matrix A for the linear stochastic differential equation:

$$s_t^{(n)} = As'^{(n)}_{t-1} + w_{t-1}^{(n)} \tag{1}$$

Here s_t and s_{t-1} are particles, and A is the matrix used to update the deterministic part of the prediction model. Since we consider two types of motion models i.e. no-motion model (`model=0`) and constant velocity model (`model=1`), we will get separate definitions of A for each of them.

For the no-motion model, the position of the particle does not change and hence, if $s_{t-1} = [x, y]^T$, then we expect $s_t = [x, y]^T$. Thus, A is simply the identity matrix with dimension 2:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2}$$

For the constant velocity motion model, let's assume a constant x-velocity of $\dot{x}$, and a constant y-velocity of $\dot{y}$. Thus, if $s_{t-1} = [x, y, \dot{x}, \dot{y}]^T$ we expect $s_t = [x+\dot{x}, y+\dot{y}, \dot{x}, \dot{y}]^T$, assuming $\Delta t = 1$ between frames. So, we can define A to be:

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

* In the implementation, we actually compute $s_t = s_{t-1}A^T$, since the particles are available in a matrix of shape `Nx4` or `Nx2`, where N is the number of particles.

3. `propagate(particles, frame_height, frame_width, params)`
This method propagates particles using the derived matrix A and adds system model noise. It also clips particle positions to ensure they stay within the frame boundaries.

4. `observe(particles, frame, bbox_height, bbox_width, hist_bin, hist, sigma_observe)`
This method computes weights for particles based on the color histogram similarity between each particle's region and the target histogram.

(a) Iterates through particles and extracts the color histogram of the region around each particle.

(b) Computes the χ^2 distance between the particle histogram and the target histogram, and computes weights using:

$$\pi^{(n)} = \frac{1}{\sqrt{2\pi}\sigma}.e^{-\frac{\chi^2(\mathrm{CH}_{sn}, \mathrm{CH}_{\mathrm{target}})^2}{2\sigma^2}} \tag{4}$$

(c) Normalizes the weights.

5. `estimate(particles, particles_w)`
   This method calculates the weighted mean of the particles based on their weights. It computes the element-wise product of particles and weights. Then, it sums along the first axis (axis=0) to get the weighted sum. The result is the estimated mean state.

6. `resample(particles, particles_w)`
   This method performs resampling of particles based on their weights. Randomly samples indices from the particles array with replacement using the weights as probabilities. Creates new arrays `resampled_particles` and `resampled_particles_w` by selecting particles and weights at the sampled indices.

Here is a step-by-step overview of the implementation of the condensation tracker algorithm:

1. **User Interaction:**
   The user is prompted to draw a bounding box on the first frame of the video and the bounding box coordinates are stored.

2. **Parameters:**
   The algorithm takes various parameters, including whether to draw plots `draw_plots`, the number of histogram bins `hist_bin`, the update parameter for color histogram `alpha`, noise levels `sigma_observe, sigma_position, sigma_velocity`, the model type `model`, the number of particles `num_particles`, and the initial velocity if the model is of type 1 `initial_velocity`.

3. **Initialization:**
   The initial color histogram `hist` is obtained using the `color_histogram` function within the specified bounding box. The mean state vectors `mean_state_a_priori` and `mean_state_a_posteriori` are initialized. Particles are initialized based on the a priori mean state.

4. **Particle Filtering Loop:**
   For each frame from the specified starting frame to the last frame of the video:

   (a) Particles are propagated using the `propagate` function, and the a priori mean state is estimated using the `estimate` function.

   (b) The video frame is read, and the color histogram is observed, weights are updated for each particle using the `observe` function.

   (c) The a posteriori mean state is updated, and the color histogram model is updated based on a convex combination.

   (d) Particle resampling is performed using the `resample` function according to the weights.

2.1 Results and Observations

Tracking objects in videos using the Condensation tracker involves tuning various parameters to achieve optimal performance for specific tracking scenarios. The parameters in question include those related to the tracker's behavior, such as the number of particles,

the choice of motion model, and the standard deviations of position and velocity noise. Additionally, parameters controlling the color histogram, such as the number of bins and the observation model noise, play a crucial role.

2.1.1 Video 1

We first try the algorithm on *video1*, with the default parameters:
$\text{hist\_bin} = 16, \alpha = 0, \sigma_{obs} = 0.1, \text{num\_particles} = 30, \sigma_{pos} = 15, \sigma_{vel} = 1,$
$\text{init\_vel} = (1, 10)$. The result is observed in Figure 1.

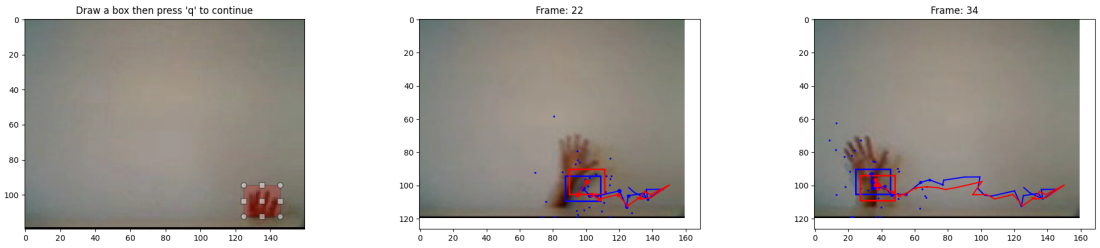


Figure 1: Hand tracking with default parameters in *video1*

We observe, that after the first couple of frames the algorithm tracks the hand instead of the fingers. This is happening mainly because the color histograms of the arm/hand and the fingers are similar and hence the former is tracked. But overall, it works quite well. We now take our experiments to *video2*.

2.1.2 Video 2

We again try with the default parameters first. The result is observed in Figure 2. We can see that this set of parameters works well for such videos with occlusions.

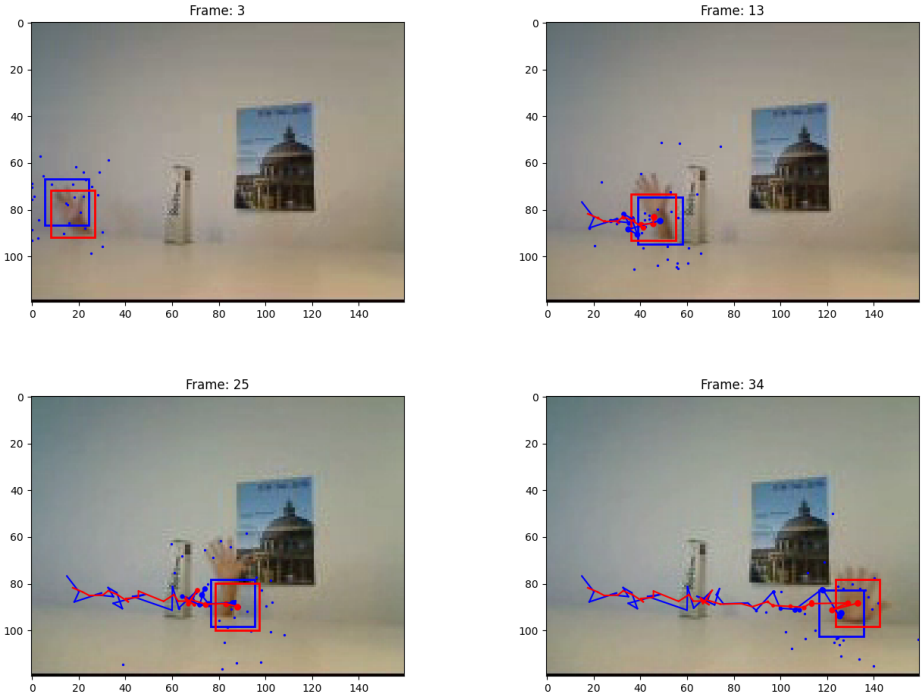


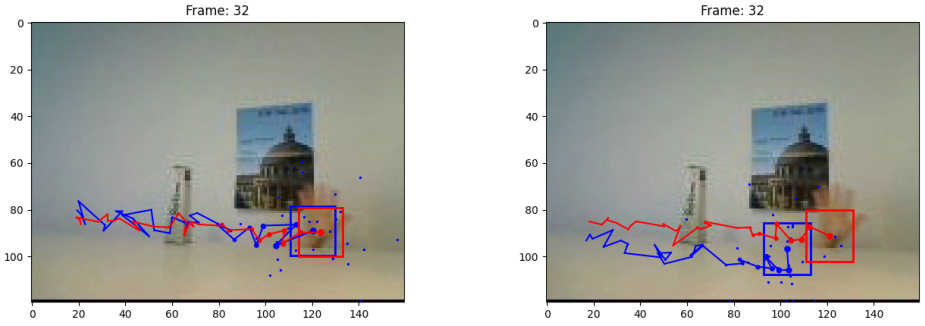
Figure 2: Hand tracking with default parameters in *video2*

We now test our algorithm over various combinations of different values of the parameters *motion_model*, $\sigma_{obs}, \sigma_{pos}, \sigma_{vel}$ according to Table 1.

Figure	**model**	**#particles**	**hist_bin**	α	σ_{obs}	σ_{pos}	σ_{vel}
3a	0	30	16	0	0.1	15	1
3b	1	30	16	0	0.1	15	1
4a	0	30	16	0	0.1	1	0.5
4b	1	30	16	0	0.1	1	0.5
5a	0	30	16	0	0.1	30	2
5b	1	30	16	0	0.1	30	2
6a	0	30	16	0	1	15	1
6b	1	30	16	0	1	15	1
7a	0	30	16	0	0.01	15	1
7b	1	30	16	0	0.01	15	1

Table 1: Experiments for *video2*

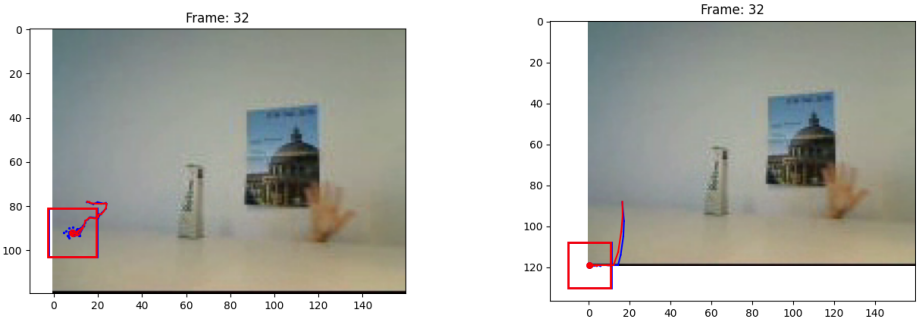
Figure 3 shows the results of using the no-motion model vs the constant velocity model. In general, we find no particular difference and both models seem to work decently well with the default parameters.



(a) (b)

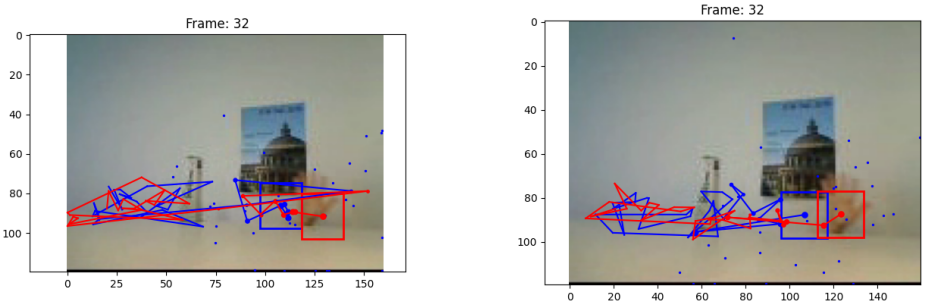
Figure 3: *Video2* σ_{obs}=0.1, σ_{pos}=15, σ_{vel}=1 (a)$model$=0, (b)$model$=1

In the Figures 4 and 5, we see the effect of changing the noise we add when we propagate the particles using Equation 1. Smaller σ_{pos} values result in less random motion added to the particles during propagation, and hence trajectories become smoother but with small incremental changes to the position. Here, this change cannot keep up with the velocity of the hand and hence these small values of σ_{pos} and σ_{vel} totally fail. Small values are well-suited for scenarios where the object's motion is relatively in a small space in the image and we expect a smooth trajectory. Larger σ_{pos} and σ_{vel} introduce more random noise during propagation, leading to more erratic trajectories. This can be seen in Figure 5 where the trajectory jumps around a lot, and the changes in position are very drastic. Thus, large values are suitable for scenarios where the object's motion is highly unpredictable or influenced by external factors.



(a) (b)

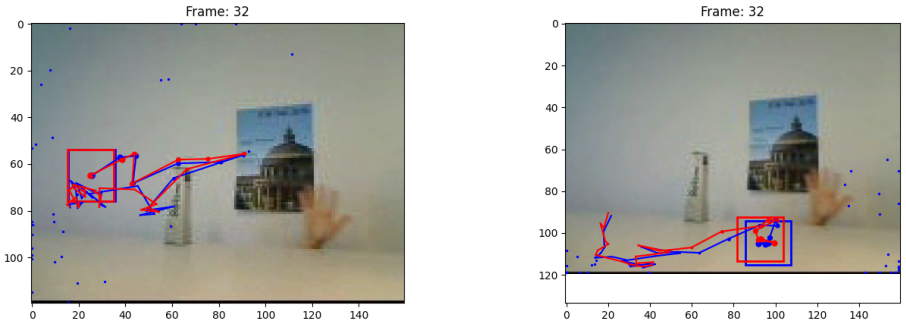
Figure 4: *Video2* σ_{obs}=0.1, σ_{pos}=1, σ_{vel}=0.5 (a)$model$=0, (b)$model$=1
Decreasing stochastic noise of the model(σ_{pos} and σ_{vel})



(a) (b)

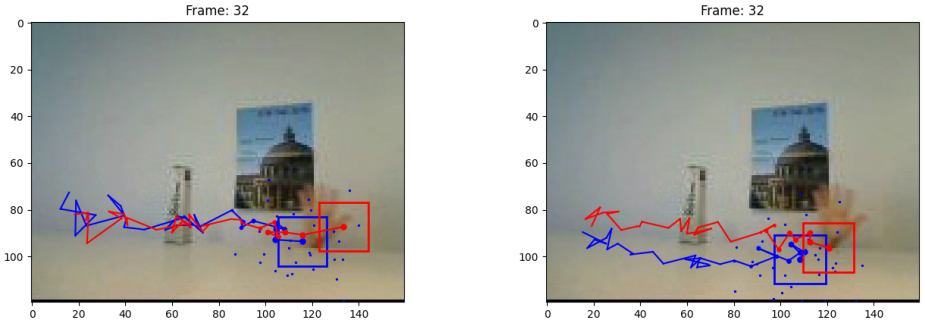
Figure 5: *Video2* σ_{obs}=0.1, σ_{pos}=30, σ_{vel}=2 (a)$model$=0, (b)$model$=1
Increasing stochastic noise of the model(σ_{pos} and σ_{vel})

In Figures 6 and 7 we see how changing σ_{obs} affects our trajectory. With smaller σ_{obs} i.e. less observation model noise color histogram observations are more precise and less influenced by random variations. This can be effective for scenarios where the object's appearance is relatively stable and consistent. In Figure 6, we lose track of the object after the occlusion because of this. With a larger value of σ_{obs}, color histograms become more tolerant to variations in object appearance, and hence the trajectory in Figure 7 looks much better for this case of occlusion. Hence, smaller values enhance the tracker's sensitivity to subtle changes in object appearance but might be more prone to tracking inaccuracies if the observed appearance deviates from the expected model. On the other hand, larger values are suitable for scenarios where the object's appearance can change significantly over time.



(a) (b)

Figure 6: *Video2* σ_{obs}=1, σ_{pos}=15, σ_{vel}=1 (a)$model$=0, (b)$model$=1
Increasing observational noise(σ_{obs})



(a) (b)

Figure 7: *Video2* σ_{obs}=0.01, σ_{pos}=15, σ_{vel}=1 (a)$model$=0, (b)$model$=1
Decreasing observational noise(σ_{obs})

So, we stick with the default parameters (as displayed in Figure 3) as they give the best trajectory among all combinations.

2.1.3 Video 3

We use the best parameters we found in the previous section, on *Video 3*, and the result is observed in Figure 8. It shows that the algorithm is able to track the ball quite successfully. However, we can tune some parameters as we see later to get a smoother trajectory.

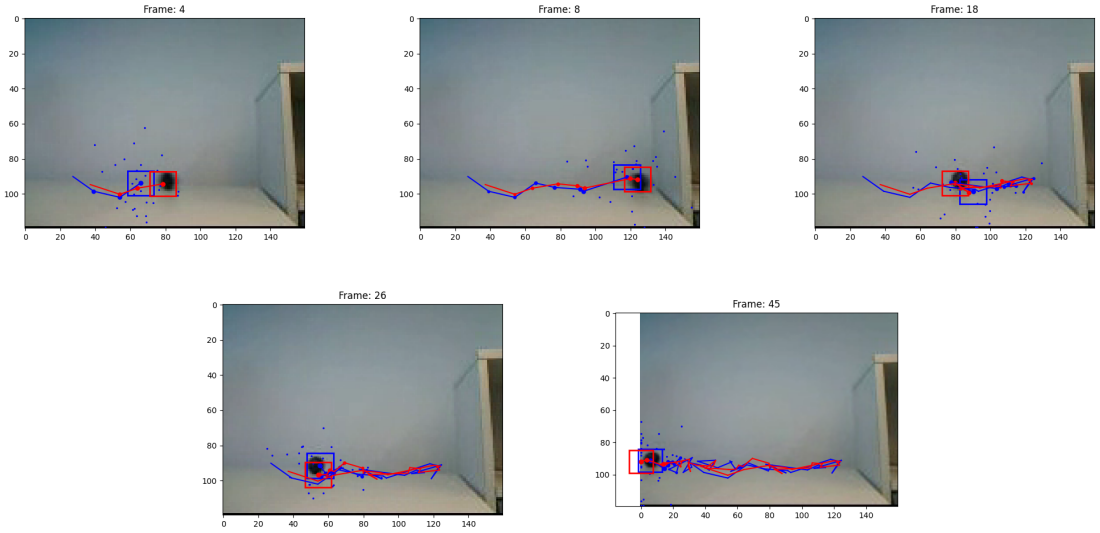


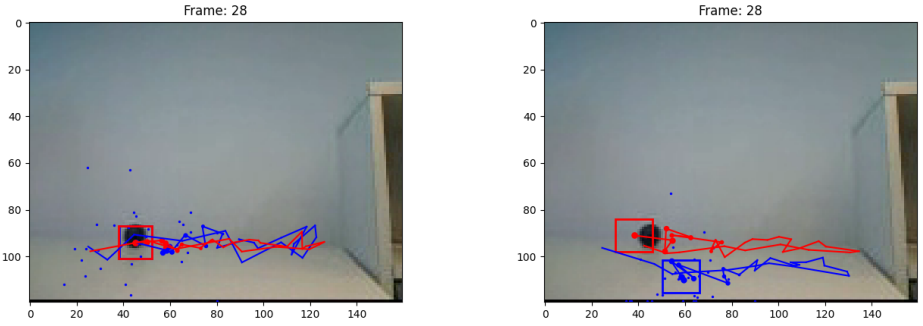
Figure 8: Hand tracking with best parameters of *video2* in *video3*

We conduct the same experiments that we conducted in the previous section, to check the influence of $motion\_model$, $\sigma_{obs}, \sigma_{pos}, \sigma_{vel}$ according to Table 2.

Figure	**model**	**#particles**	**hist_bin**	α	σ_{obs}	σ_{pos}	σ_{vel}
9a	0	30	16	0	0.1	15	1
9b	1	30	16	0	0.1	15	1
10a	0	30	16	0	0.1	1	0.5
10b	1	30	16	0	0.1	1	0.5
11a	0	30	16	0	0.1	30	2
11b	1	30	16	0	0.1	30	2
12a	0	30	16	0	1	15	1
12b	1	30	16	0	1	15	1
13a	0	30	16	0	0.05	15	1
13b	1	30	16	0	0.05	15	1

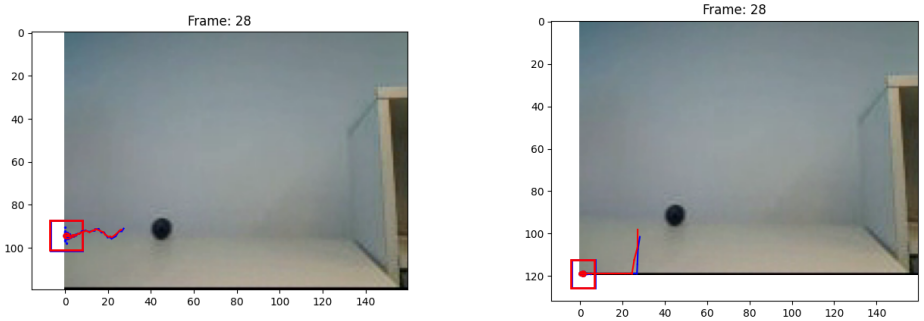
Table 2: Experiments for *video3*

Based on the figures below, we can draw similar conclusions as we did for *video2*. We observe that reducing the stochastic noise of the model in Figure 10 leads to the ball not being tracked. The changes in the particles are very small that cannot keep up with the velocity of the ball. Again, increasing the noise leads to the particles being sampled very sparsely and hence the trajectory is very erractic with the ball being tracked only for certain portions of the true trajectory in Figure 11.



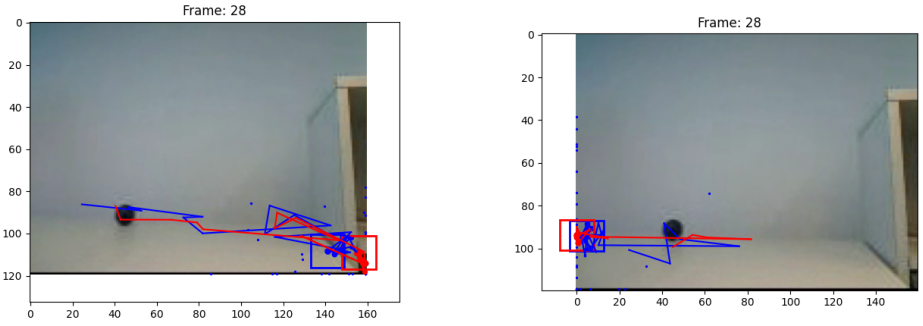
(a) (b)

Figure 9: *Video3* σ_{obs}=0.1, σ_{pos}=15, σ_{vel}=1 (a)*model*=0, (b)*model*=1



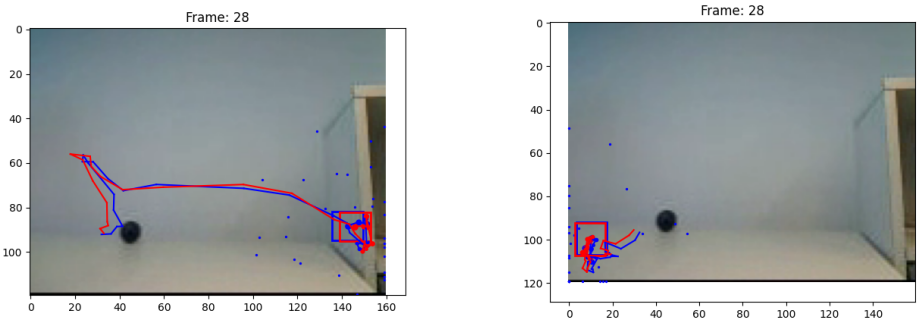
(a) (b)

Figure 10: *Video3* σ_{obs}=0.1, σ_{pos}=1, σ_{vel}=0.5 (a)*model*=0, (b)*model*=1
Decreasing stochastic noise of the model(σ_{pos} and σ_{vel})



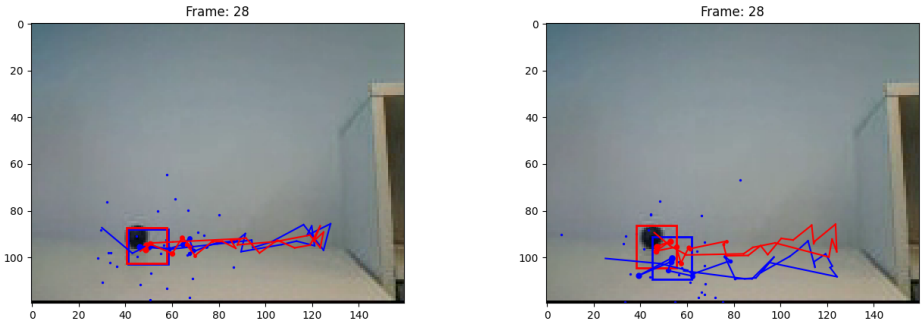
(a) (b)

Figure 11: *Video3* σ_{obs}=0.1, σ_{pos}=30, σ_{vel}=2 (a)$model$=0, (b)$model$=1
Increasing stochastic noise of the model(σ_{pos} and σ_{vel})



(a) (b)

Figure 12: *Video3* σ_{obs}=1, σ_{pos}=15, σ_{vel}=1 (a)$model$=0, (b)$model$=1
Increasing observational noise(σ_{obs})

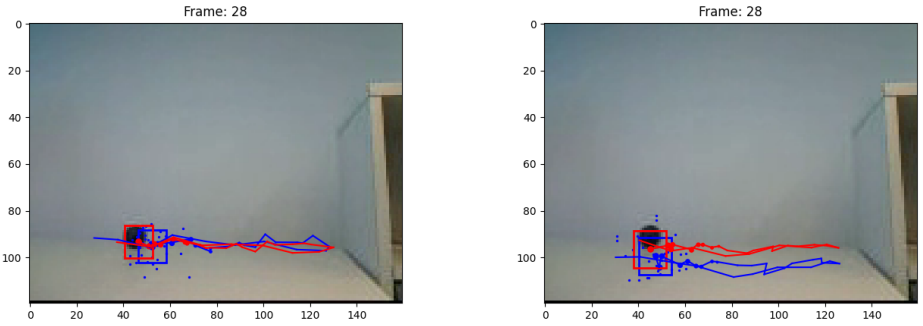


(a) (b)

Figure 13: *Video3* σ_{obs}=0.05, σ_{pos}=15, σ_{vel}=1 (a)$model$=0, (b)$model$=1
Decreasing observational noise(σ_{obs})

Now, in Figure 8, we see that the trajectory is generally decent, however we see some peaks and a sharp trajectory is tracked instead of a smooth one. From our analysis we have seen that changing σ_{pos} can help us mitigate this issue. So, we try some values for

$1 < \sigma_{pos} < 15$, essentially finetuning it to get a better trajectory than the one we currently have in Figure 8. We find that $\sigma_{pos} = 7$ is the best possible value for a smooth trajectory, the result for which we see in Figure 14. In general, we can see that the constant motion model is better and smoother.



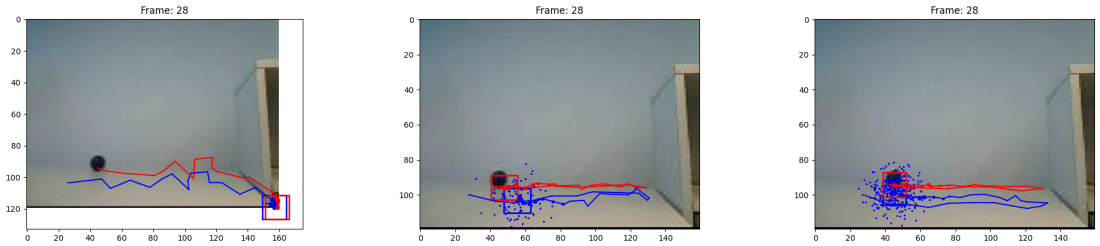
(a) (b)

Figure 14: *Video3* σ_{obs}=0.1, σ_{pos}=7, σ_{vel}=1 (a)*model*=0, (b)*model*=1 Best parameters

2.1.4 Changing other parameters

In this section we address the effect of other parameters like α, *num_particles*, *hist_bin* (number of histogram bins).
*we start with the best parameters we obtain in the previous section, with *motion_model*=1.

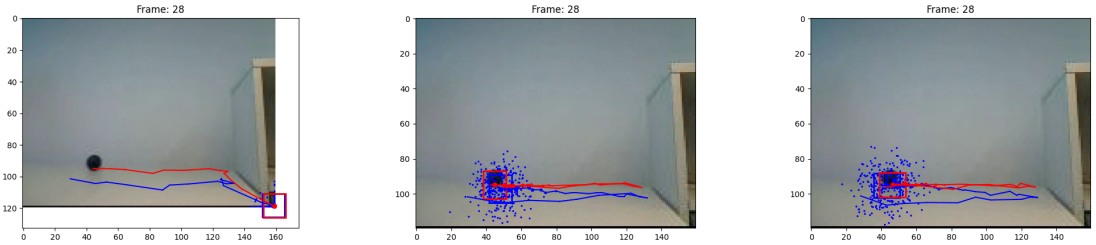
1. **Effect of number of particles:**
   In Figure 15 we observe the effects of decreasing(a) and increasing(b and c) the number of particles. Small number of particles reduce the diversity of the particle set and hence tracking may become less accurate, and the tracker may struggle to represent the true state of the object adequately. We can clearly observe this for (a). With larger number of particles, tracking accuracy improves as the tracker explores a broader range of potential states. However, the disadvantage of large values being increased computational load. Observe that in (c), the trajectory looks much better than in Figure 14. For subsequent experiments, we therefore set *num_particles* = 300.



(a) (b) (c)

Figure 15: Effect of *num_particles*: (a)*num_particles*=10 (b)*num_particles*=100 (c)*num_particles*=300

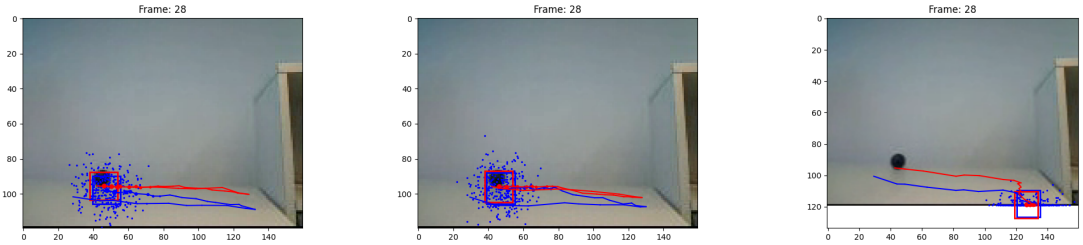
2. **Effect of number of histogram bins:**
   In Figure 16 we compare the results of decreasing(a) and increasing(a and b) the number of histogram bins. As we see in (a), smaller *hist_bin* values result in a coarser color histogram representation, the tracker captures less detailed color information and hence it's possible that the χ^2 cost may not be the least for the actual position of the ball. We see in (a) that the bottom right corner which is again, a mass of black pixels, is mistaken for the ball by the tracker because it's closer to the target histogram. With larger values, the tracker becomes more sensitive to subtle color variations, but the computational load of the algorithm increases too.



(a) (b) (c)

Figure 16: Effect of *hist_bin*: (a)*hist_bin*=8 (b)*hist_bin*=32 (c)*hist_bin*=64

3. **Effect of α:** In Figure 17, we compare the results of different α values. α is responsible for how much the target histogram is updated in each iteration. Smaller α values slow down the rate at which the target color histogram is updated, and the tracker is less responsive to changes in object appearance. If we do not expect the object appearance to change a lot, a smaller value of α is well-suited, as is the case here. The appearance of the ball is not expected to change. Larger α values lead to faster updates of the target color histogram and the tracker adapts more quickly to changes in object appearance. That is why, we see in (c), for an α value of 0.8, the tracker loses track of the ball because the target histogram get updated too quickly and is too far away from the original one after a point.



(a) (b) (c)

Figure 17: Effect of α: (a)α=0.3 (b)α=0.5 (c)α=0.8

In conclusion, we find that Figure 15(c) best depicts the trajectory our tracker can generate:
***motion_model*=1, σ_{obs}=0.1, σ_{pos}=7, σ_{vel}=1, α=0 [0.3], *num_particles*=300, *hist_bin*=16 [32].**
*values in [] denote alternatives