Lab6

1.Implementation

1.Color histogram

For each color channel(RGB), I put the pixel value into one of the three histograms(RGB), then stack the histograms into one with the size of $(3, bins)$ after normalization.

```
    #get the region of interest
    roi = frame[ymin:ymax + 1, xmin:xmax + 1]
    bin_size = (255 + 1) // hist_bin
    r_hist = np.zeros(hist_bin)
    g_hist = np.zeros(hist_bin)
    b_hist = np.zeros(hist_bin)
    #put the pixel into bins
    for y in range(roi.shape[0]):
        for x in range(roi.shape[1]):
            r_pixel, g_pixel, b_pixel = roi[y, x]
            r_hist[r_pixel // bin_size] += 1
            g_hist[g_pixel // bin_size] += 1
            b_hist[b_pixel // bin_size] += 1
    rgb_hist = np.vstack([r_hist, g_hist, b_hist])
    #normalization
    total = np.sum(rgb_hist)
    #prevent 0 divisor
    assert total != 0
    rgb_hist /= total
    return rgb_hist
```

2.Derive matrix A

Exclude the noise, the equation should hold for any situation.

$$s = As \tag{1}$$

i. No motion

To keep the position the same in every frame.

$$s = \{x, y\}^T \tag{2}$$

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

ii. Constant velocity motion

Object move as a constant velocity between frames.

$$p_t = p_{t-1} + v \times \delta t \tag{4}$$

$$s = \{x, y, \dot{x}, \dot{y}\}^T \tag{5}$$

$$A = \begin{bmatrix} 1 & 0 & \delta t & 0 \\ 0 & 1 & 0 & \delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

3.Propagation

According to the formula, using the dynamic matrix $A$ derived before:

$$S_t = S'_{t-1}A^T + W_{t-1} \tag{7}$$

Then finally clip the position of the particle into the frame size.

```
def propagate(particles, frame_height, frame_width, params):
    # time / frames
    delta_t = 1 / 30
    sigma_position = params["sigma_position"]
    sigma_velocity = params["sigma_velocity"]
    noise_position = np.random.normal(0, sigma_position, size=(particles.shape[0], 2))
    noise_velocity = np.random.normal(0, sigma_velocity, size=(particles.shape[0], 2))
    #no motion
    if params["model"] == 0:
        A_matrix = np.array([[1, 0], [0, 1]])
        noise = noise_position
    #constant velocity
    else:
        A_matrix = np.array([[1, 0, delta_t, 0], [0, 1, 0, delta_t], [0, 0, 1, 0], [0,
0, 0, 1]])
        noise = np.hstack([noise_position, noise_velocity])
    particles_new = particles @ A_matrix.T + noise
    #position clip
    particles_new[:, 0:2] = np.clip(particles_new[:, 0:2], a_min=[0, 0], a_max=
[frame_width, frame_height])
    return particles_new
```

4.Observation

I calculate the color histogram for every bounding box generated by the particles, and then calculate the Chi-Square distance between the histogram and the target histogram. Finally I use the the distance to calculate the weight using gaussian kernel(including normalization).

```
def observe(particles, frame, bbox_height, bbox_width, hist_bin, hist, sigma_observe):
    '''
    hist:initial histograms
    hist_bin:number of bins
    '''
    frame_height, frame_width = frame.shape[0], frame.shape[1]
    particles_w = np.zeros((num, 1))
    for i in range(num):
        center_x = particles[i, 0]
        center_y = particles[i, 1]
        x_min = round(center_x - bbox_width / 2)
        x_max = round(center_x + bbox_width / 2)
        y_min = round(center_y - bbox_height / 2)
        y_max = round(center_y + bbox_height / 2)
        #clip to the frame size
        x_min, x_max, y_min, y_max = np.clip([x_min, x_max, y_min, y_max], a_min=[0, 0,
0, 0], a_max=[frame_width, frame_width, frame_height, frame_height])
        hist_cur = color_histogram(x_min, y_min, x_max, y_max, frame, hist_bin)
        chi_squared_distance = chi2_cost(hist_cur, hist)
        #Chi-Square distance
        particles_w[i] = 1. / (math.sqrt(2. * np.pi) * sigma_observe) * np.exp(-0.5 *
np.square(chi_squared_distance / sigma_observe))

    #normalization
    #prevent 0 divisor
    eps = 1e-60
    particles_w /= (np.sum(particles_w) + eps)
    return particles_w
```

5.Estimation

According to the formula, I get the expectation state of all the particles:

$$E[s_t] = \sum_{n=1}^{N} \pi_t^{(n)} s_t^{(n)} \tag{8}$$

```
def estimate(particles, particles_w):
    return np.sum(particles * particles_w, axis=0)
```

6.Resampling

I calculate the CDF of the particles weight first. Then according to the CDF, I sample the corresponding particles. I change the original particle weight into uniform after getting the new particles set, because I have already taken the probability into consideration during sampling, and now set including same particles.

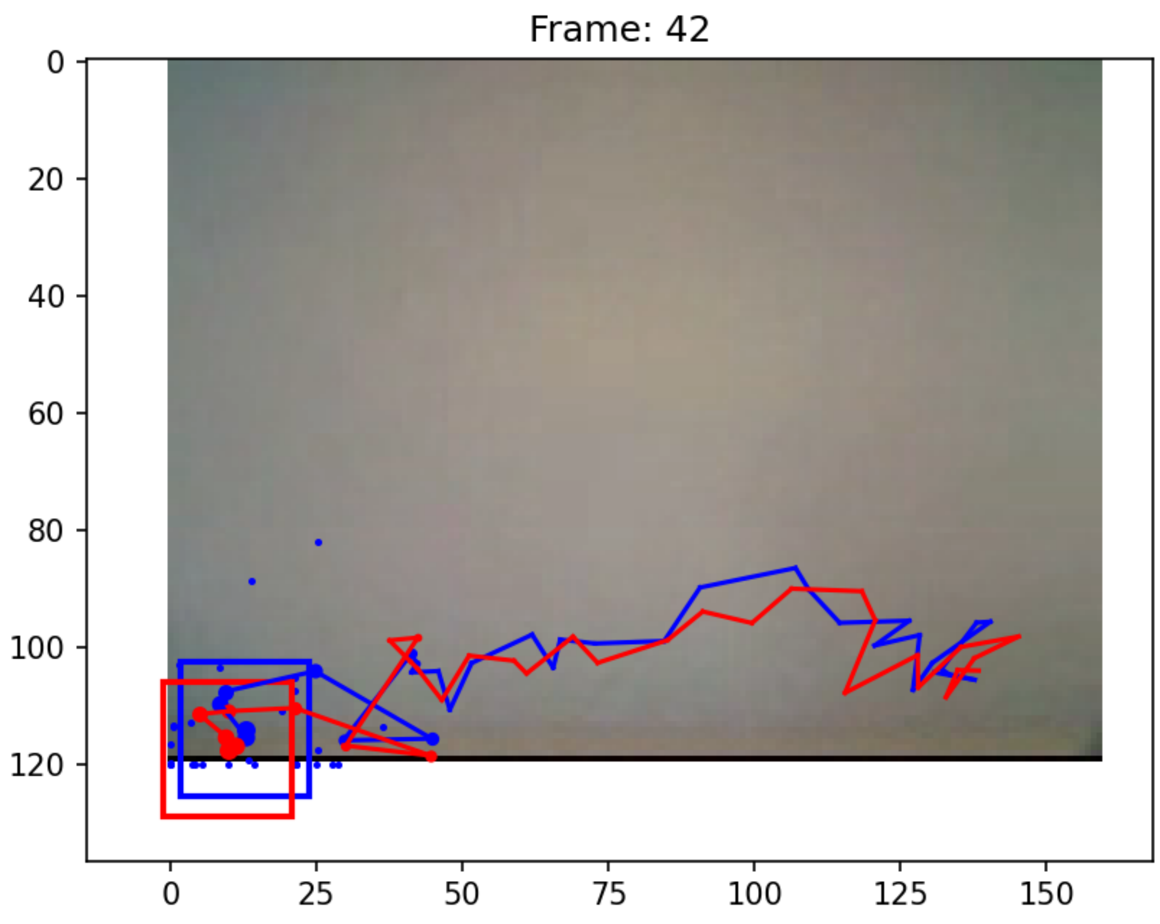
```python
def resample(particles, particles_w):
    #CDF
    cumulative_sum = np.cumsum(particles_w)
    cumulative_sum[-1] = 1.0
    #get the index randomly
    indices = np.searchsorted(cumulative_sum, np.random.rand(len(particles_w)))
    resampled_particles = particles[indices]
    #same weight after resampling
    resampled_weights = np.ones_like(particles_w) / len(particles_w)

    return resampled_particles, resampled_weights
```

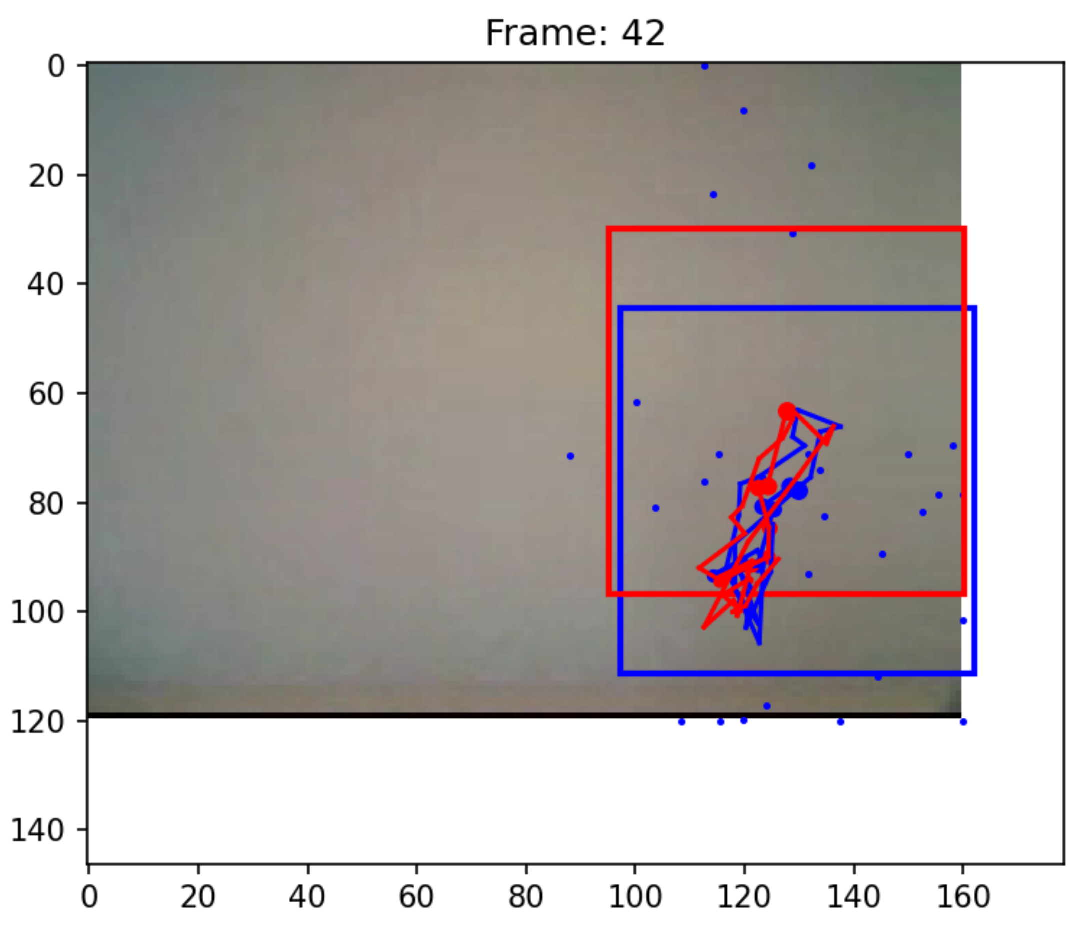
2.Experiment

1.video1

I use the default parameters set, and the model track the hand with good performance.



If I don't draw the bounding box properly(either bigger or smaller), the model may not work well.



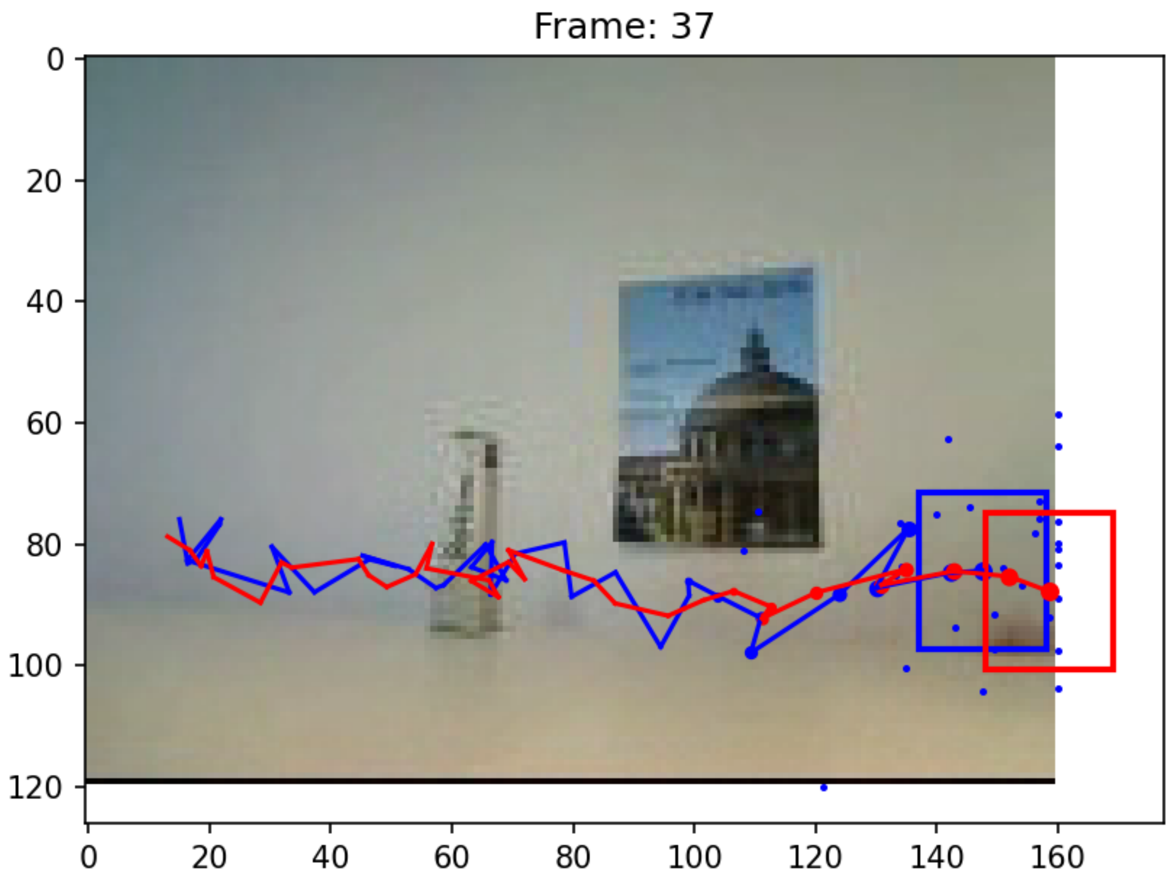
2.video2

Standard parameter is as the original ones, every time only make one parameter change and keep others the same.

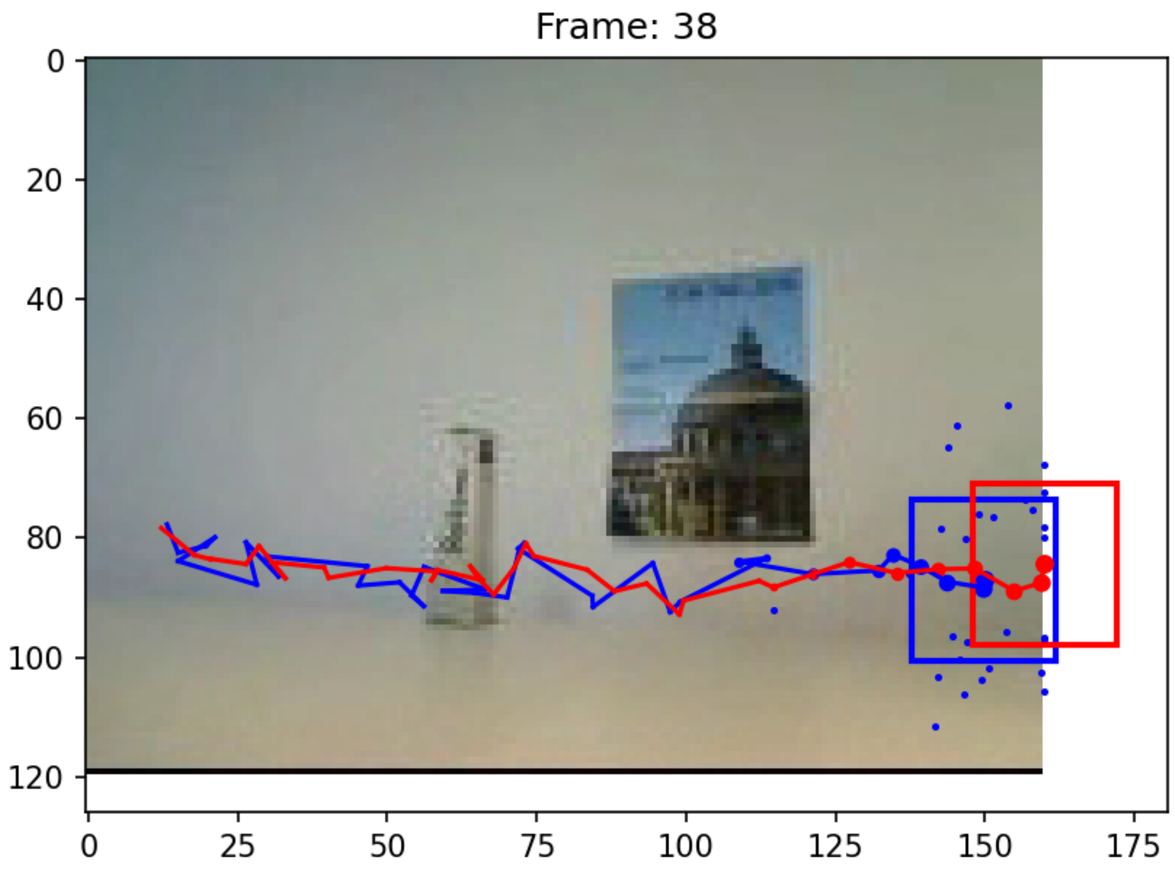
Motion model

Both the model0 and model1 can track the hand, because the hand moves relatively slow.

model0



model1:

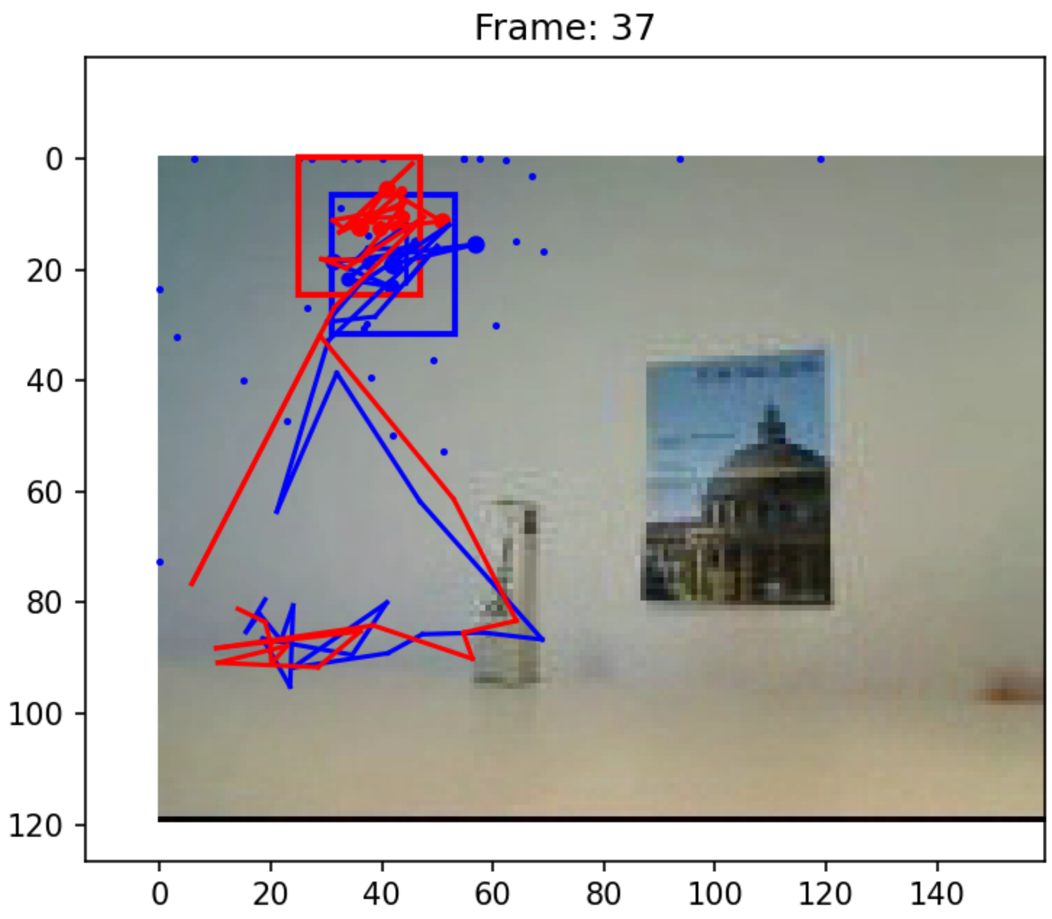


System noise

Because both model0 and model1 work well, regardless of the velocity, so I only tune the sigma_position to consider about the system noise for convenience.

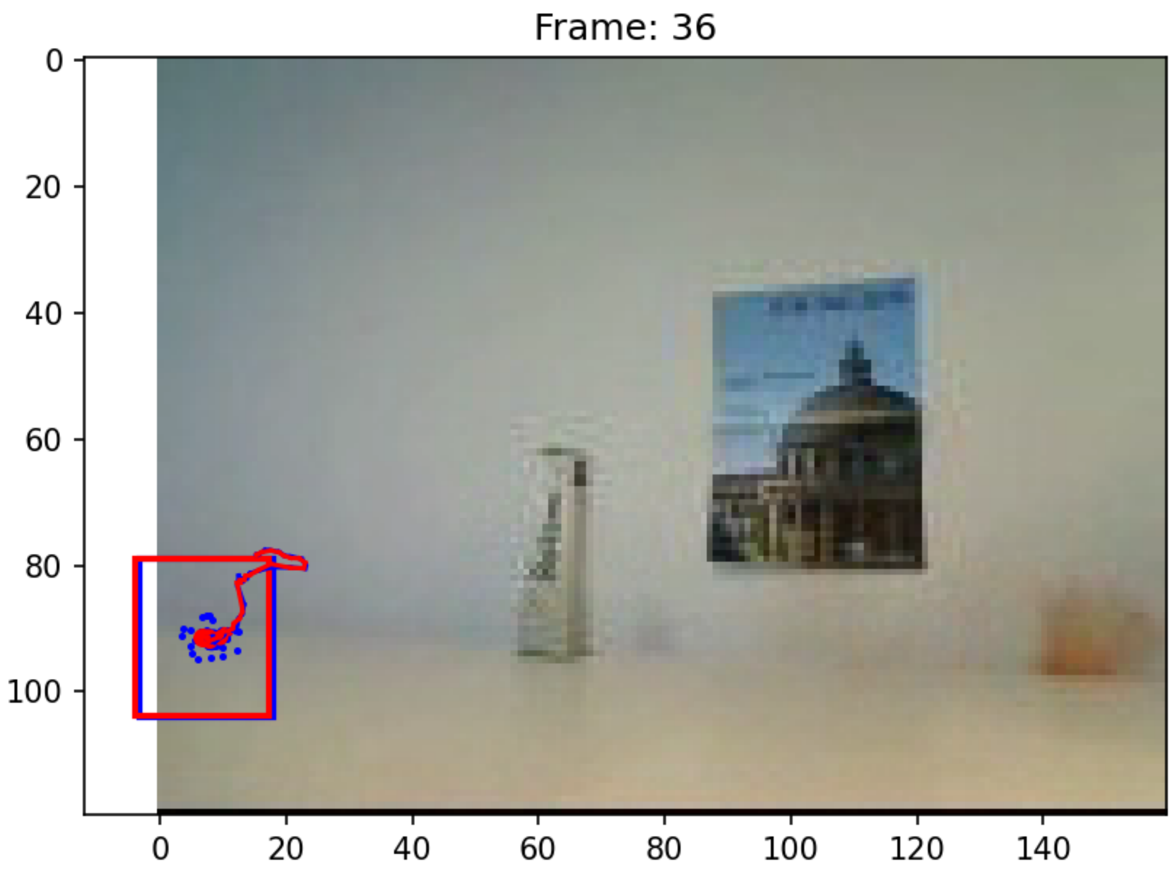
sigma_position: 30

The particle is spread very much, the model can't get the precise prediction for the object. So the model always intends to make use of the new frame input relatively, thus can be easily influenced by the observation noise. When the hand is occluded, the model lost its target, and can't use the prediction to guess where the hand is to keep tracking, leading the bounding box shift to left and up corner.



sigma_position: 1

The model don't allow the particles to explore the state space fully, the particle state changes very slowly, which means the model is too confident about its prediction, thus is relatively not sensitive to the new frame input. So the bounding box almost always stay in the original place, the particle can't capture the move of the object, thus can't track the object well.

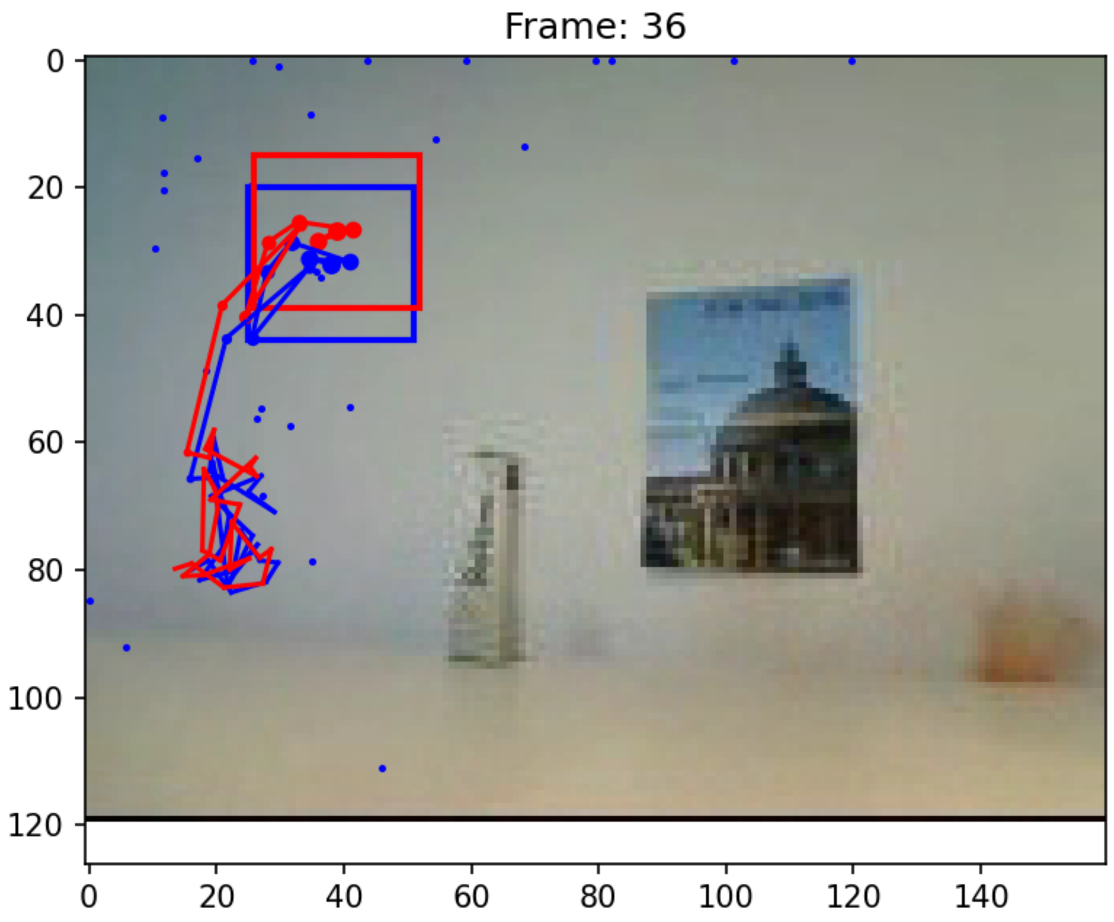


Measurement noise

The measurement noise is set by the sigma_observe in the weight calculation. If the sigma_observe value is small means the gaussian distribution is thin, so the observation with noise can only get a small weight when the observation shifts from the mean(the target value), which assumes the observation is with much less noise, vice versa. To sum up, if sigma_observe is big, means the model assumes the observation from environment inludes a lot of noise. So the robust is better, if the quality of observation is bad, but if the value is too big, which can be influenced by the unrelative particle state, make the model performs worse.

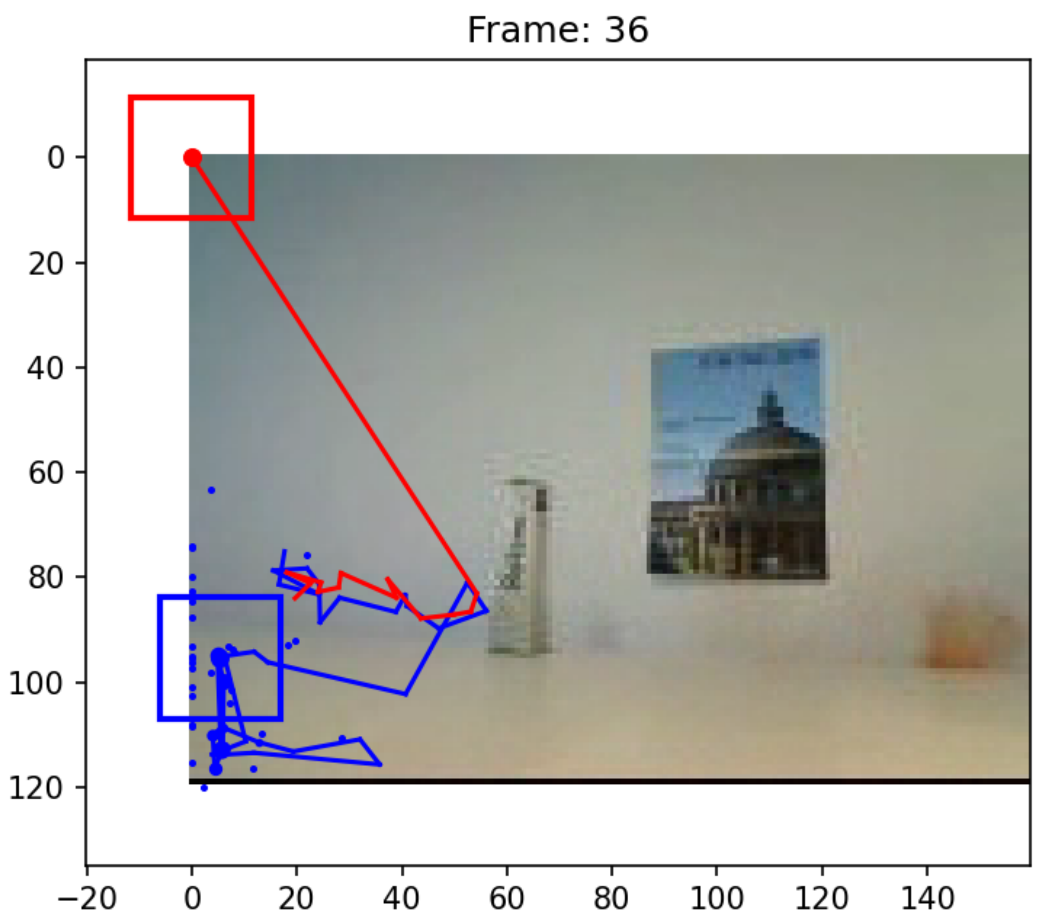
sigma_observe: 1

The model assumes the noise very big, and will be influenced by the unrelative particle state which works for exploration, making the prediction worse, leading the model lost the hand during tracking.



sigma_observe: 0.01

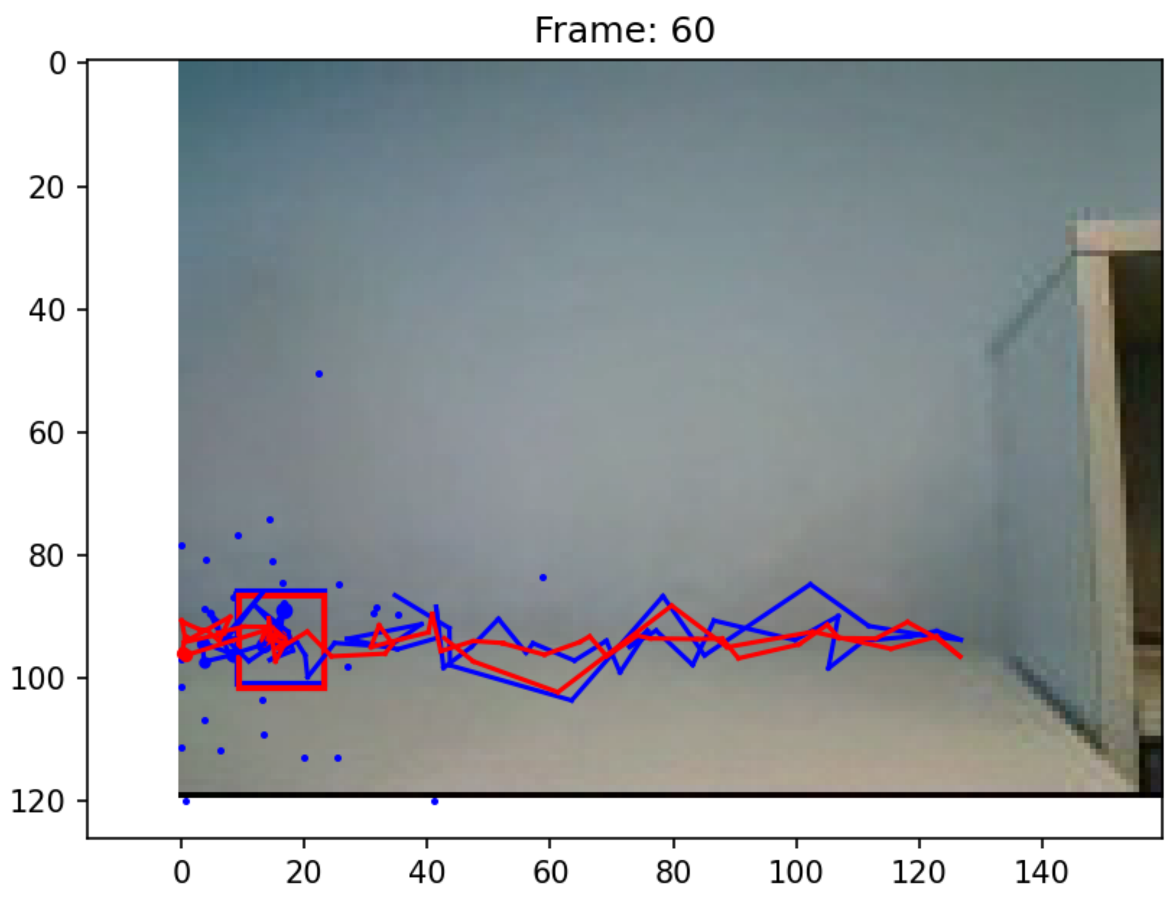
The model is too confident in the observation input, giving no tolerance for the noise of observation. The input frame should be with very high quality, and thus the model is not robut to interference. So once the hand is occluded, the model lost the hand.



3.vedio3

Using the parameters set, I can track the ball, because the parameter is best tuned in video2, except for the model, so I need to use model1.

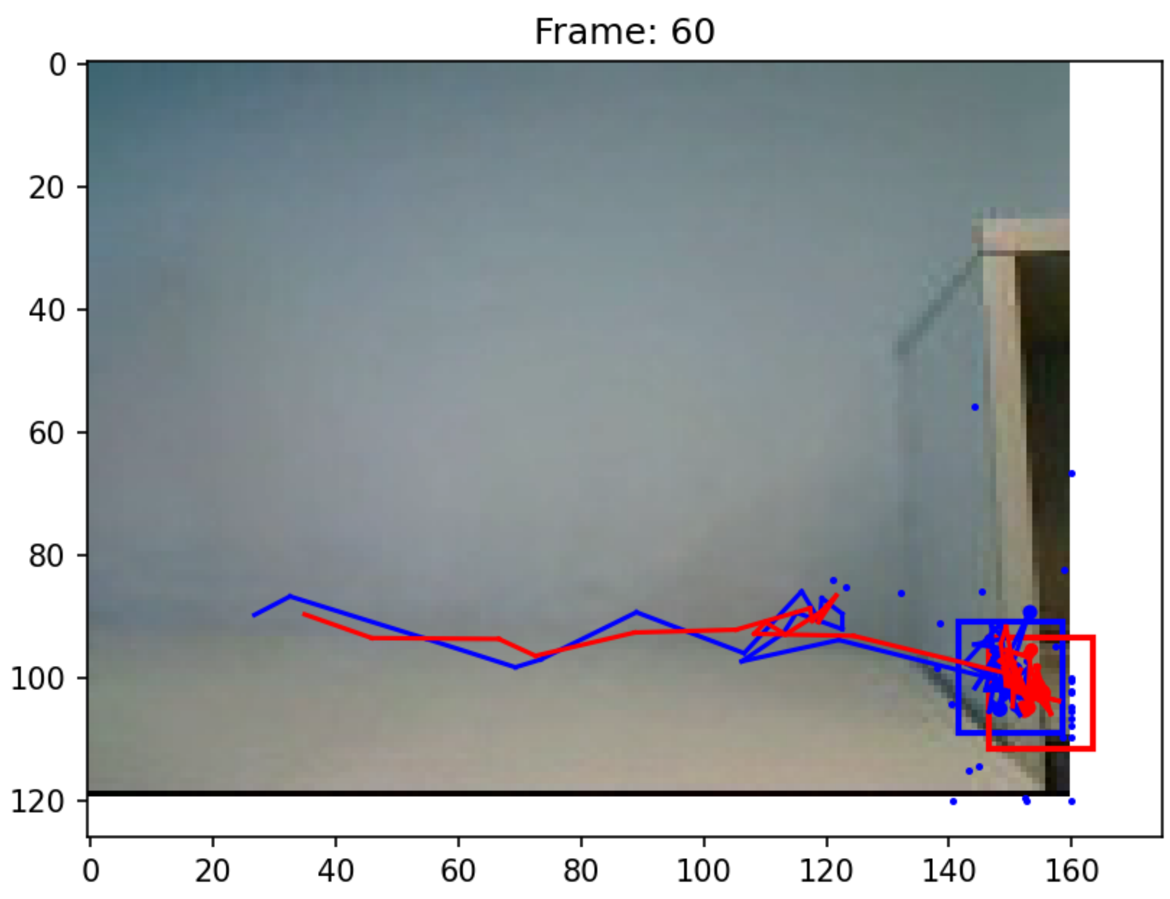
```
    params = {
        "draw_plots": 1,
        "hist_bin": 16,
        "alpha": 0,
        "sigma_observe": 0.1,
        "model": 1,
        "num_particles": 30,
        "sigma_position": 15,
        "sigma_velocity": 1,
        "initial_velocity": (1, 10)
    }
```



Motion model

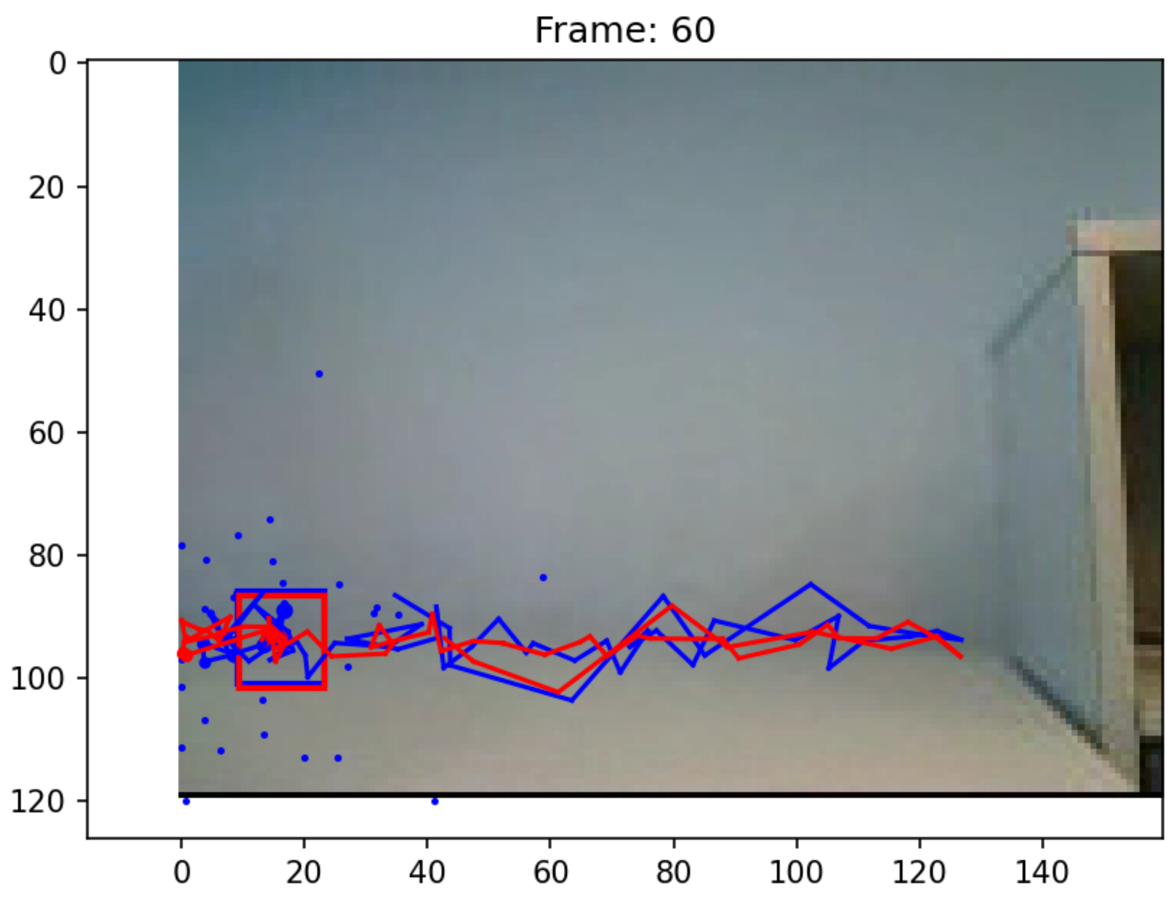
model0

The ball moves fast, if I use the no motion model, the model can't track the ball properly, especially when the ball bounces and turns back.



model1

Constant model can track the ball all the time(before and after bouncing).

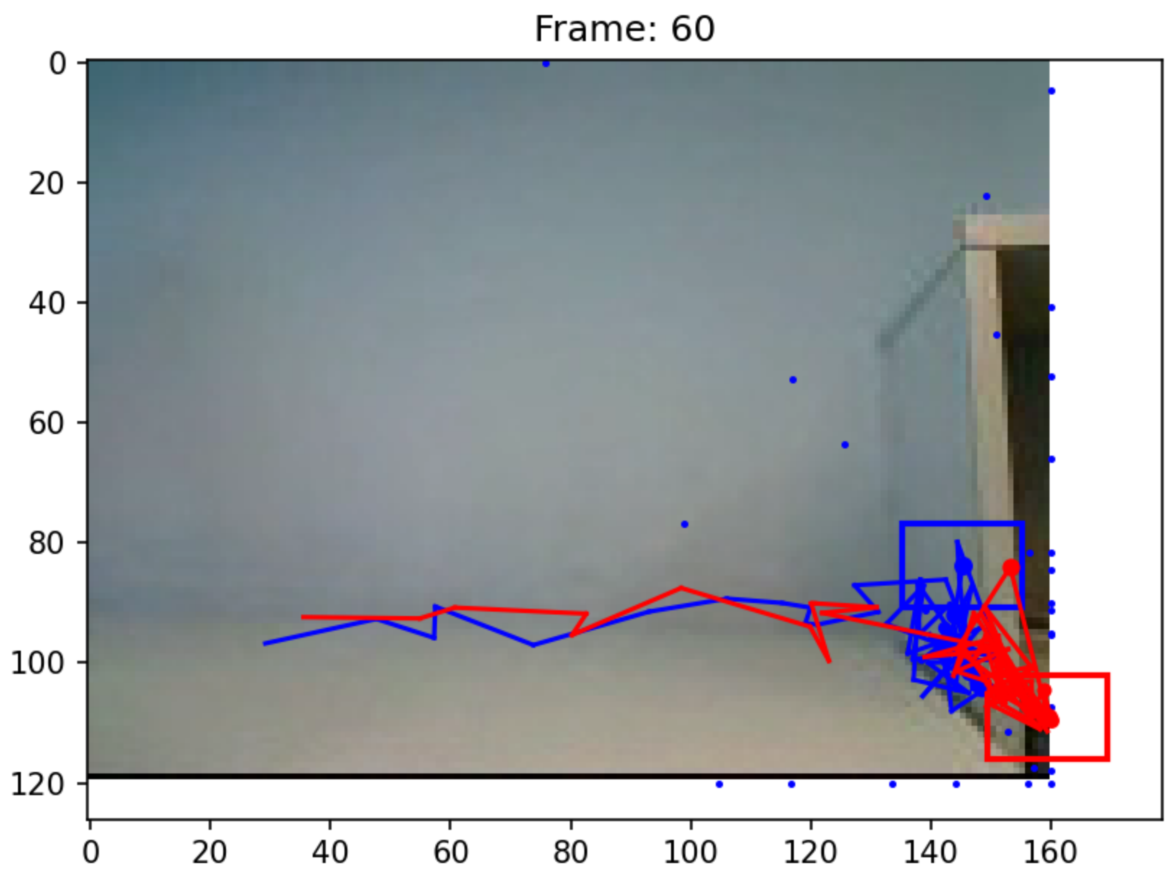


System noise

Unlike video2, for video3, I choose to use model1, system noise now includes sigma_position and sigma_velocity both. Actually, the influence to the model of sigma_position and sigma_velocity should be almost the same. To keep the influence factor less, I just tune sigma_position and keep the sigma_velocity constant.

sigma_position: 30

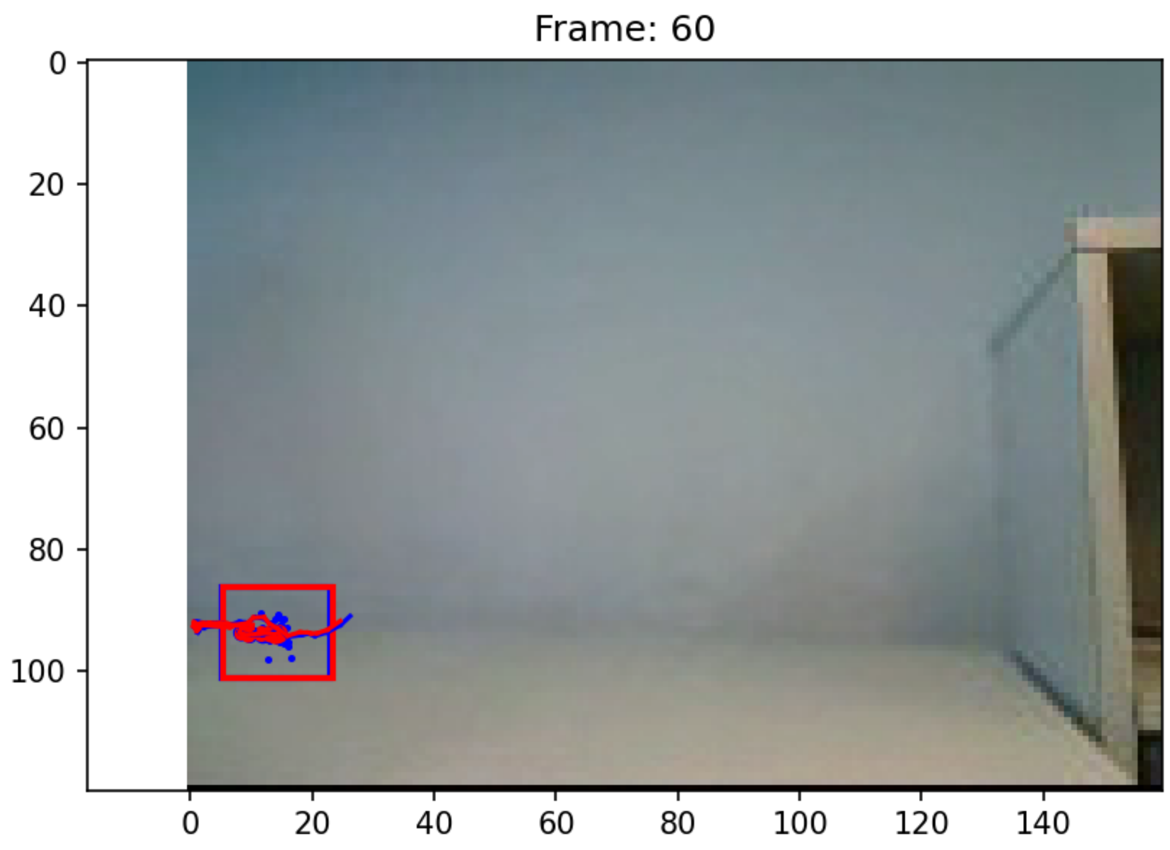
The particle is spread very much, the model can't get the precise prediction for the ball. So the model always intends to make use of the new frame input relatively, thus can be easily influenced by the observation noise. When the hand gets close to the wall and turn back, the observation includes more noise, make the model lost the object.



sigma_position: 1

Like in video2, the bounding box stay at the original place.

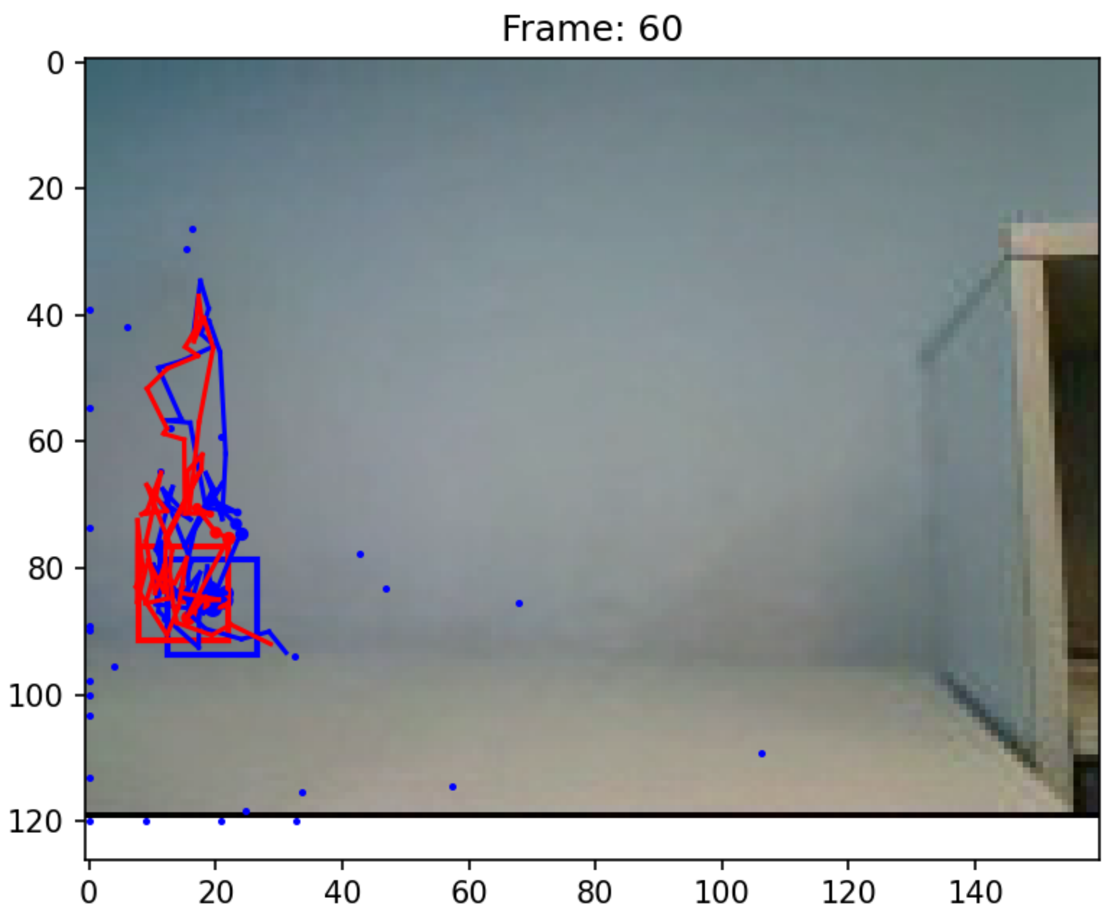
Because the model don't allow the particles to explore the state space fully, the particle state changes very slowly, which means the model is too confident about its prediction, thus is relatively not sensitive to the new frame input. So the bounding box almost always stay in the original place, the particle can't capture the move of the object, thus can't track the object well.



Measurement noise

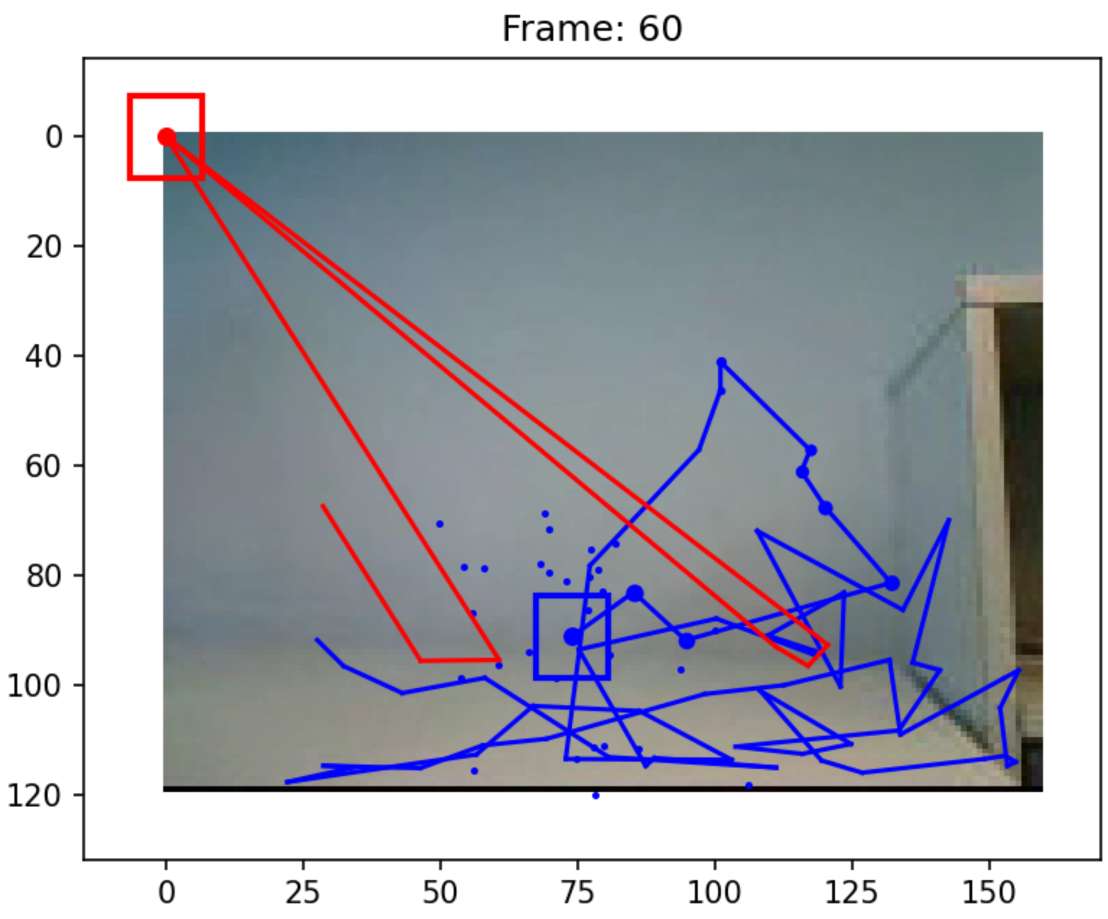
sigma_observe: 1

The model assumes the noise is very big, and will be influenced by the unrelative particle state which works for exploration (due to the bigger weight), making the prediction worse, leading the model lost the ball during tracking.



sigma_observe: 0.01

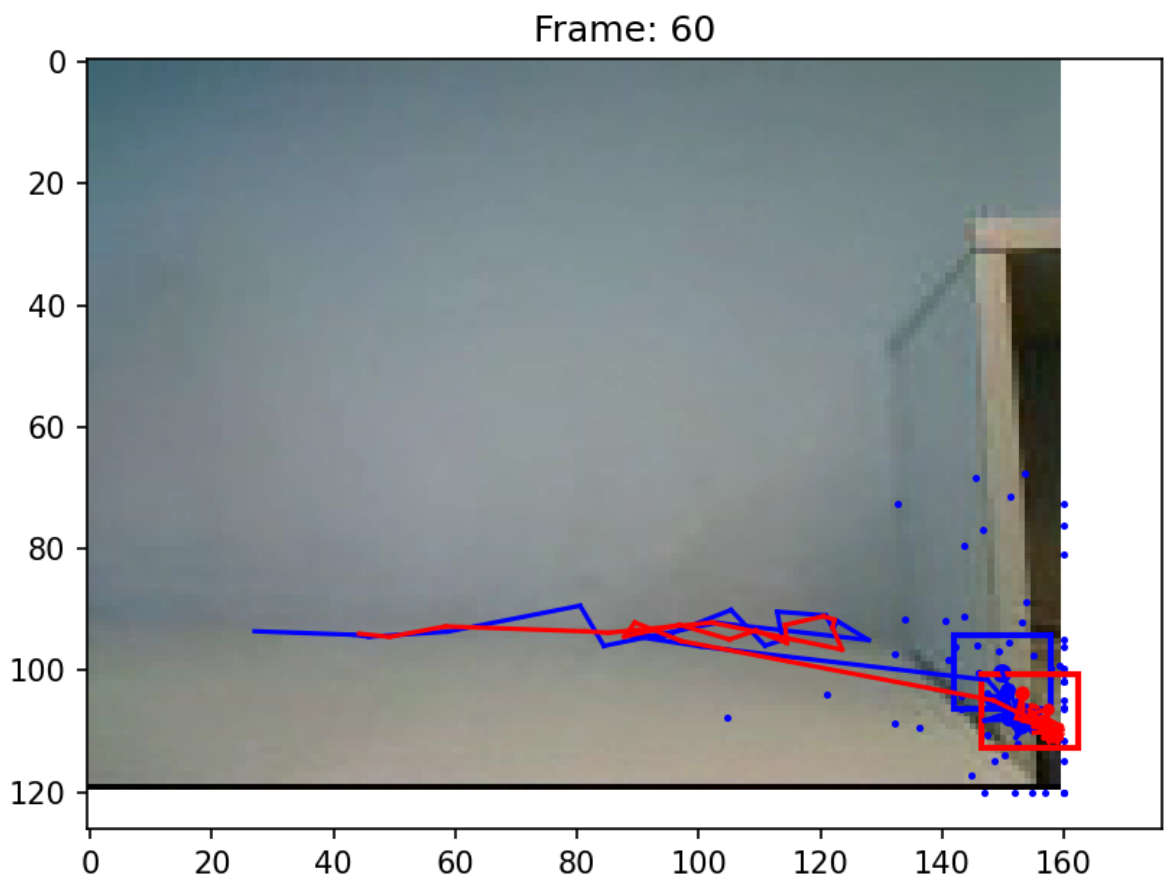
The model is too confident in the observation input, giving no tolerance for the noise of observation. The input frame should be with very high quality, and thus the model is not robuts to interference, especially when the ball gets close to the wall.



Number of particles

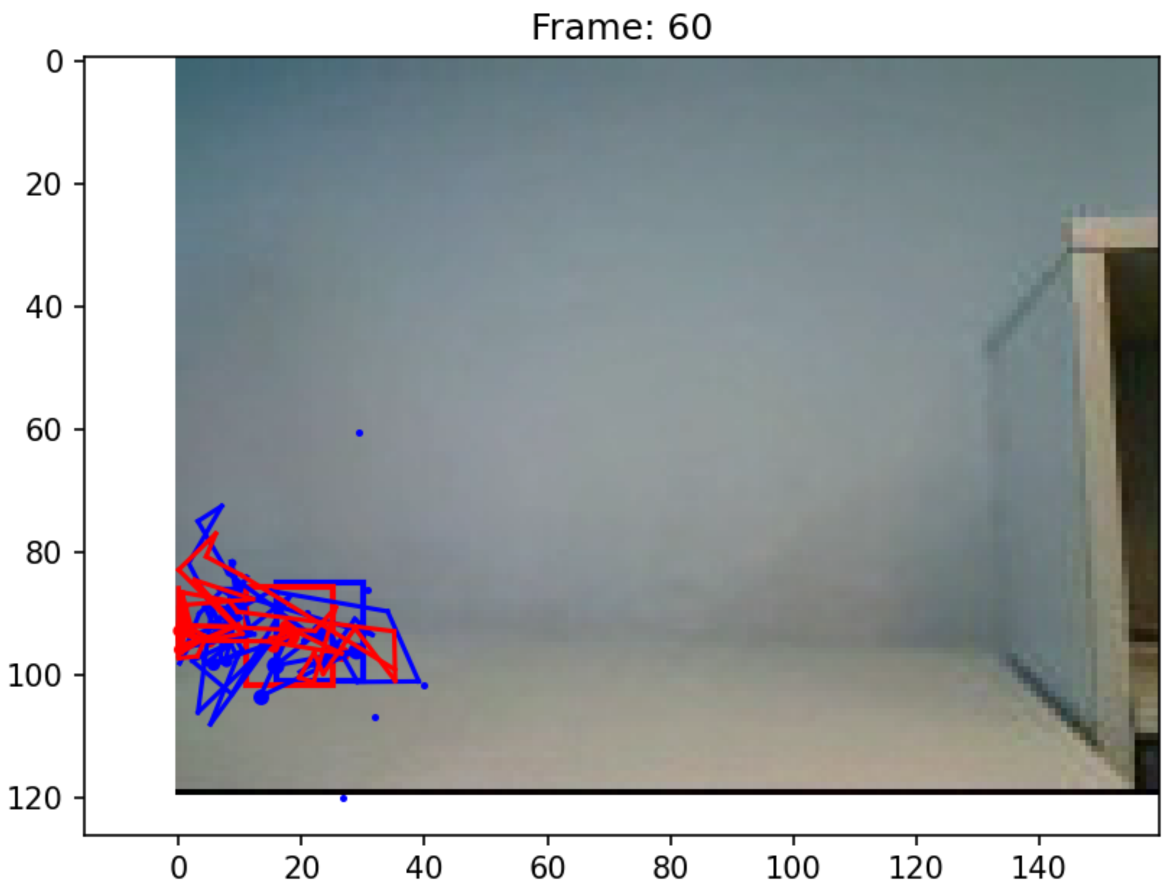
num_particles: 60

The computational power is requested more, besides, the particles state may be overfitting to the noise(i.e., use the model complexity to overfit the noise), and the expecation change slowly, thus can't track the target well if the target move quickly(e.g., bouncing).



num_particles: 10

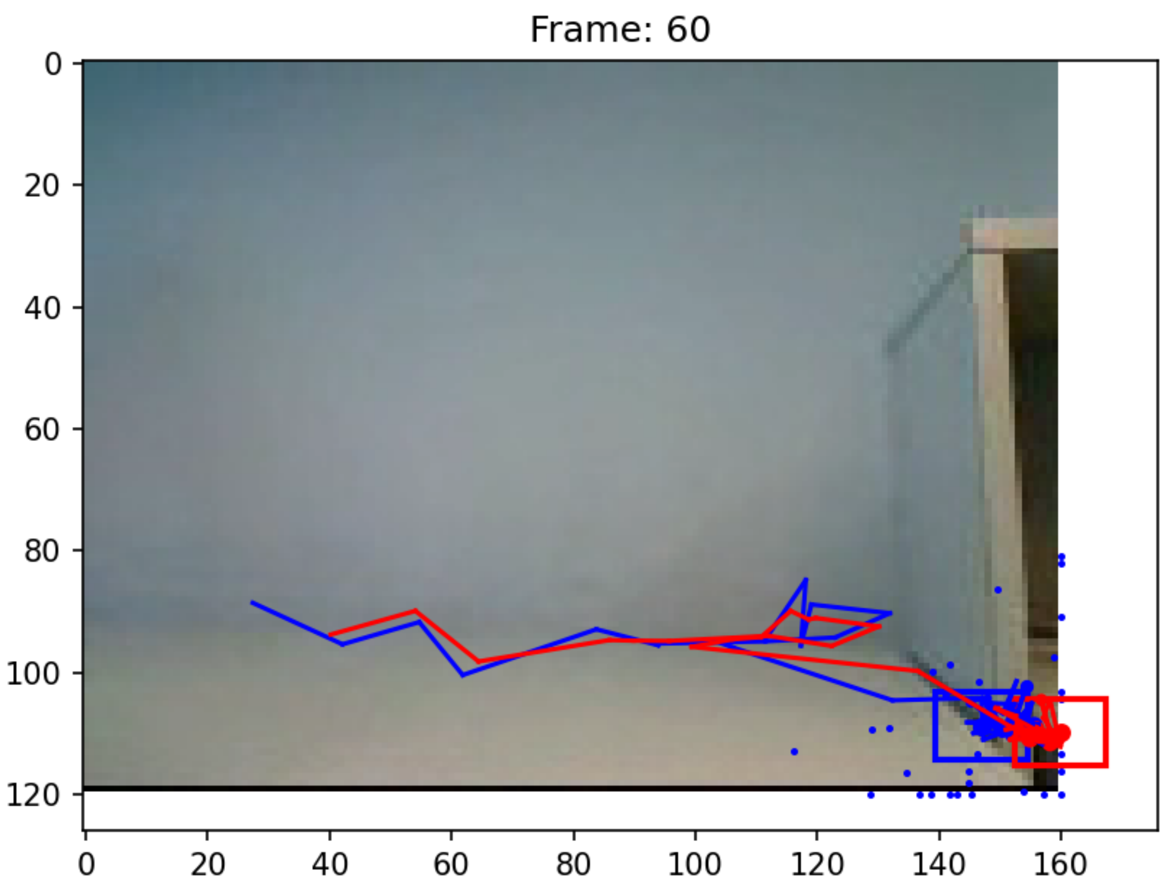
The model is easily interfered, not robuts to the noise, and can't represent the true state with very less particles.



Number of bins

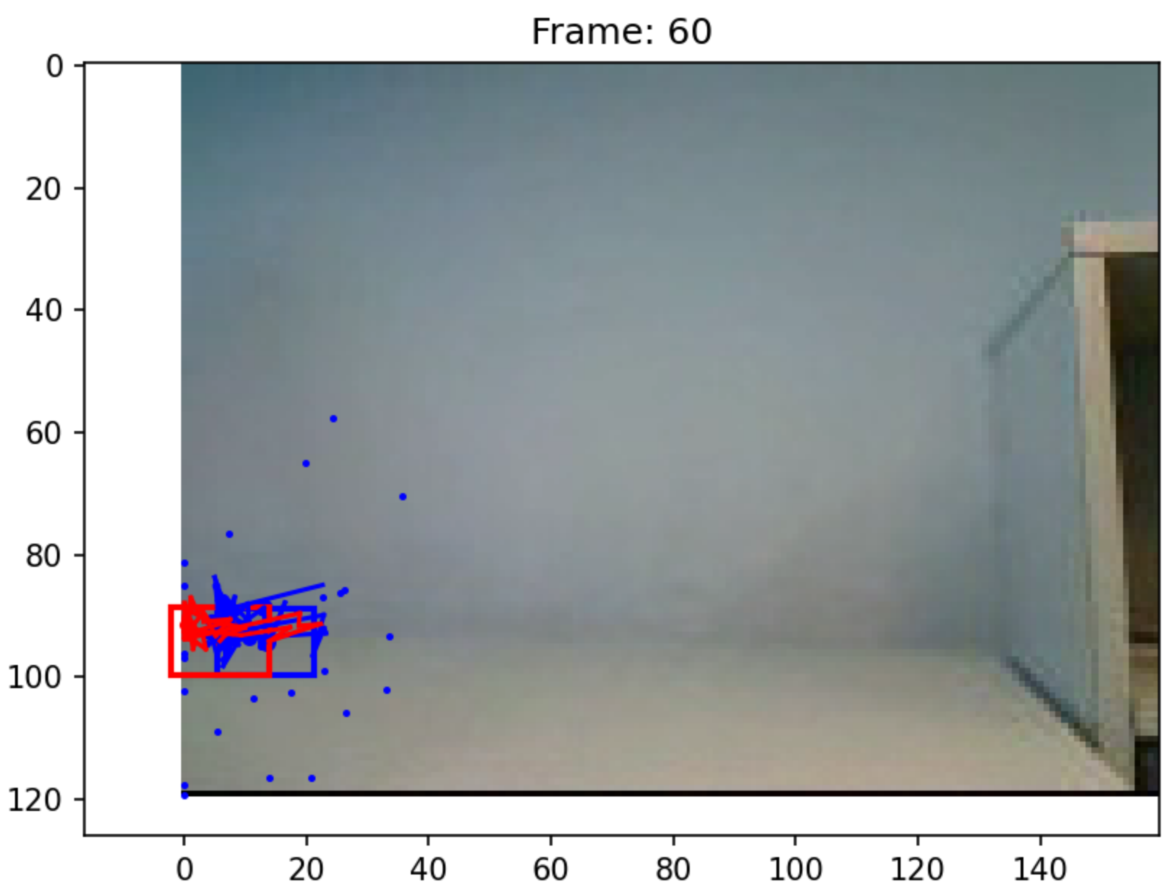
hist_bin: 48

With too many bins, the model may become too sensitive to slight color variations, which may not be significant or may be due to noise, especially when in complex layout(e.g., when bouncing), in other words, overfitting to noise.



hist_bin: 4

With too less bins, the color distribution can't be represented well, the Chi-Square distance and weight can't be calculated properly.

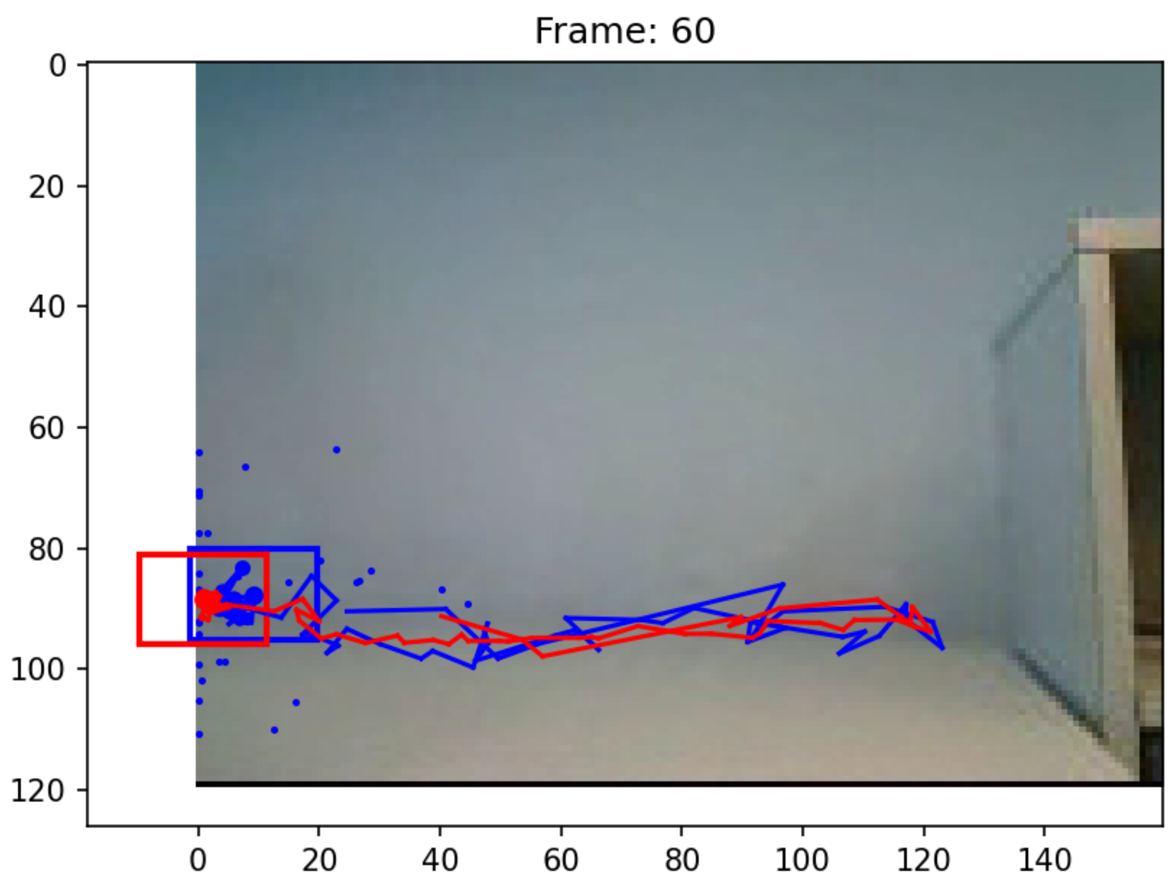


Appearance model updating

Advantage:

This allows the target to change overtime, and the model can be adapt to the changes.

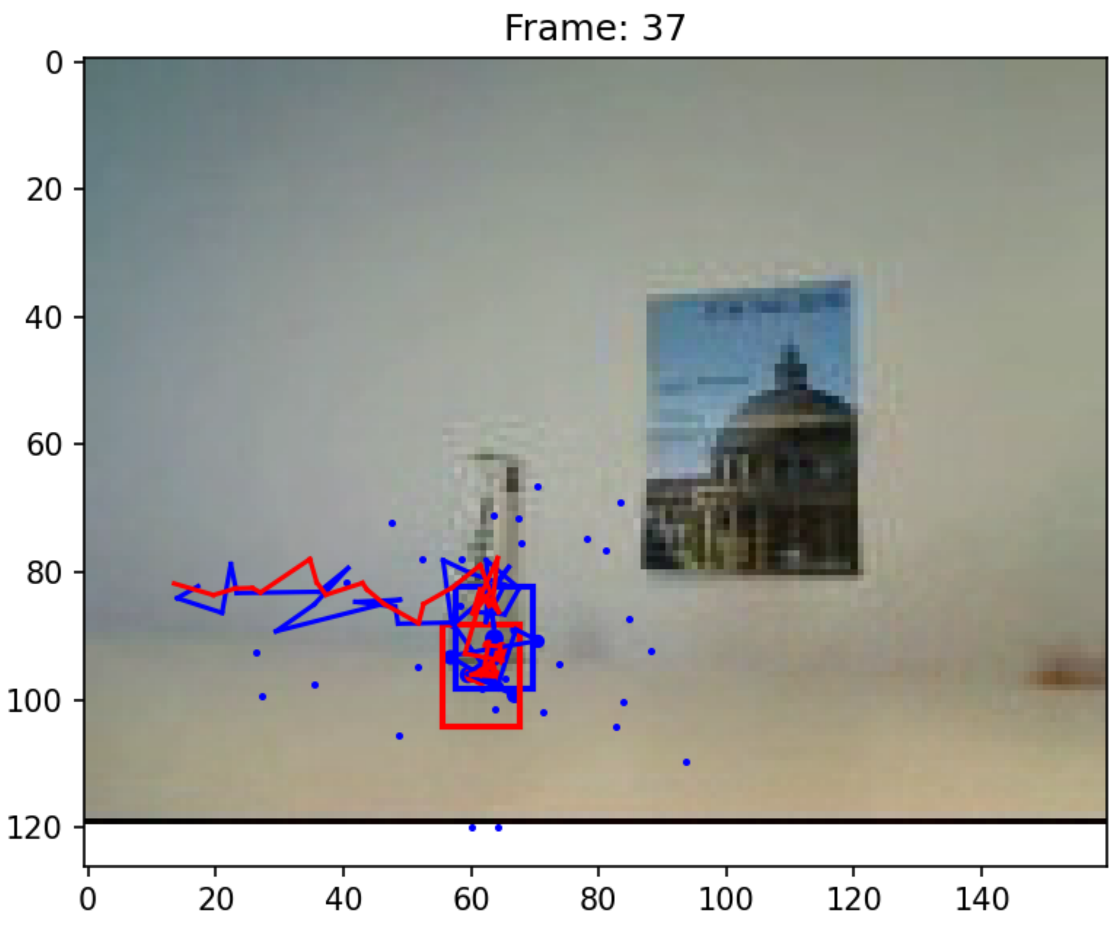
For example, now I use alpha as 0.5, even with model as model0, the ball can be tracked as well.



Disadvantage:

The tracking target may be changed slowly, leading the model to track the incorrect target.

For example, when the hand is occluded, the tracking target just change to the occlusion object in video2, using alpha as 0.5.


Frame: 37
0
20
40
60
80
100
120
0
20
40
60
80
100
120
140