Report for Lab exercise 4 for Computer Vision

Condensation Tracker Based on Color Histograms

Color Histogram

```
# First we create a color space containing hist_bin^3 bins. And we also need to
adjust position inputs to make sure they lie in the frame. For every pixel in
this small region, we judge which bin the RGB pixel should be in and sum them up
for the final histogram and then normalize it.
def color_histogram(xmin, ymin, xmax, ymax, frame, hist_bin) -> np.array:
    hist = np.zeros(hist_bin ** 3)
    divide = np.linspace(0, 255, hist_bin + 1)
    xmin = np.maximum(xmin, 0)
    ymin = np.maximum(ymin, 0)
    xmax = np.minimum(xmax, frame.shape[1])
    ymax = np.minimum(ymax, frame.shape[0])
    for x in range(math.ceil(xmin), math.floor(xmax)):
        for y in range(math.ceil(ymin), math.floor(ymax)):
            R = frame[y, x, 0]
            G = frame[y, x, 1]
            B = frame[y, x, 2]
            for i in range(hist_bin):
                if divide[i] <= R < divide[i + 1]:
                    R_index = i
                if divide[i] <= G < divide[i + 1]:
                    G_index = i
                if divide[i] <= B < divide[i + 1]:
                    B_index = i
            RGB_index = (hist_bin ** 2) * R_index + hist_bin * G_index + B_index
            hist[RGB_index] = hist[RGB_index] + 1
    return hist/sum(hist)
```

Derive matrix A & Propagation

```
# particles: [num_p, state_dim]
# A deterministic matrix: [state_dim, state_dim]

# Because the each particle is a row vector, so we change the prediction function
into:
# S_t = S_{t-1} * A + W_{t-1}
# in this case if the model is static, A is a 2 dimensional identity matrix; else
if the model is with constant velocity, A is a 4 dimensional matrix below.

# And for each particle, we should add noise according to the params. Finally
make sure they lie in the frame
def propagate(particles, frame_height, frame_width, params):
    next_state = np.zeros_like(particles)
    if params["model"] == 0:
        A = np.eye(2)
        next_state = particles.dot(A)

        for i, element in enumerate(next_state):
            next_state[i, 0] += random.gauss(0, params["sigma_position"])
            next_state[i, 1] += random.gauss(0, params["sigma_position"])

            # make sure the center lies in the frame
            next_state[i, 0] = np.maximum(0, next_state[i, 0])
            next_state[i, 0] = np.minimum(frame_width, next_state[i, 0])
            next_state[i, 1] = np.maximum(0, next_state[i, 1])
            next_state[i, 1] = np.minimum(frame_height, next_state[i, 1])
    else:
        A = [[1, 0, 0, 0],
             [0, 1, 0, 0],
             [1, 0, 1, 0],
             [0, 1, 0, 1]]
        next_state = particles.dot(A)

        for i, element in enumerate(next_state):

            next_state[i, 0] += random.gauss(0, params["sigma_position"])
            next_state[i, 1] += random.gauss(0, params["sigma_position"])
            next_state[i, 2] += random.gauss(0, params["sigma_velocity"])
            next_state[i, 3] += random.gauss(0, params["sigma_velocity"])

            # make sure the center lies in the frame
            next_state[i, 0] = np.maximum(0, next_state[i, 0])
            next_state[i, 0] = np.minimum(frame_width, next_state[i, 0])
            next_state[i, 1] = np.maximum(0, next_state[i, 1])
            next_state[i, 1] = np.minimum(frame_height, next_state[i, 1])
    return next_state
```

Observation

```
# For every particle, we need to compare it's hist with the object's hist in the
previous frame by using chi2_cost. Then we can get the weight for each particle.
(Remember the normalization)
def observe(particles, frame, bbox_height, bbox_width, hist_bin, hist,
sigma_observe):
    prob = []
    for i, element in enumerate(particles):
        xmin = element[0] - bbox_width / 2
        xmax = element[0] + bbox_width / 2
        ymin = element[1] - bbox_height / 2
        ymax = element[1] + bbox_height / 2
        hist_obs = color_histogram(xmin, ymin, xmax, ymax, frame, hist_bin)
        dist = chi2_cost(hist_obs, hist)
        prob.append(1/np.sqrt(2*np.pi)/sigma_observe*np.exp(-
(dist**2)/2/(sigma_observe**2)))
        weight = prob/sum(prob)
    return weight
```

Estimation

```python
# just calculate the weighted average of particles to the estimated next state
def estimate(particles, particles_w):
    """
    particles: size[num_particle, state_dim]
    particles_w: size[num_particle, 1]
    Return
    mean state: size[1, state_dim]
    """
    particles_w = np.array(particles_w)
    weighted_p = np.dot(particles_w.T, particles)
    return weighted_p / np.sum(particles_w)
```

Resampling

```python
# By using np.random.choice, we could get the index of resampling based on the
weights. Then we could easily get the resampled particles position and new
weights.
def resample(particles, weights):
    """
    resample the particles based on their weights,
    and return these new particles along with their corresponding weights.
    """
    particles_updated = np.zeros_like(particles)
    weights_updated = np.zeros_like(weights)
    weights = weights / sum(weights)
    idx_resample = np.random.choice(range(particles.shape[0]),
particles.shape[0], p=weights)
    for i in range(particles.shape[0]):
        particles_updated[i] = particles[idx_resample[i], :]
        weights_updated[i] = weights[idx_resample[i]]
    return particles_updated, weights_updated
```

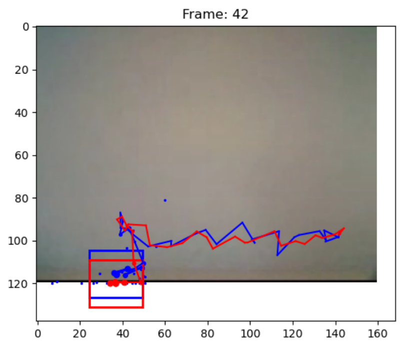
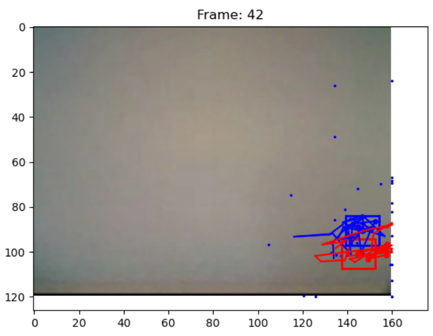
Experiments

Video1(a moving hand)

- params:

  ```
          "draw_plots": 1,
          "hist_bin": 16,
          "alpha": 0.7,
          "sigma_observe": 0.1,
          "model": 1,
          "num_particles": 30,
          "sigma_position": 15,
          "sigma_velocity": 1,
          "initial_velocity": (1, 1)
  ```

- successful and unsuccessful result:

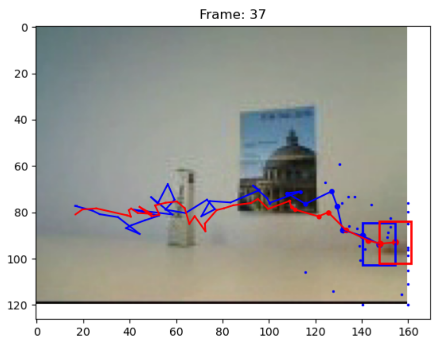
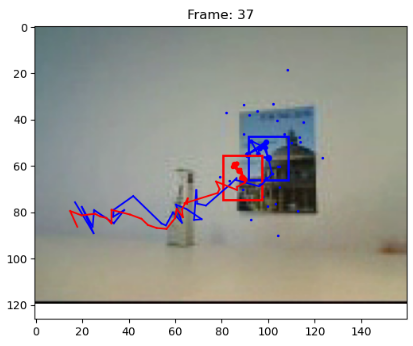
Successful	Unsuccessful
	

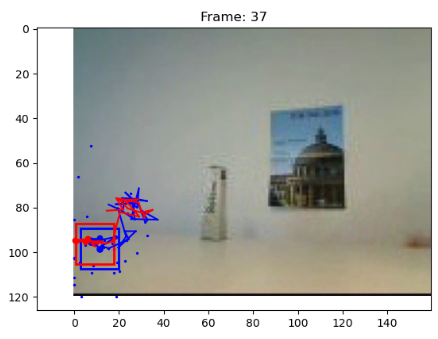
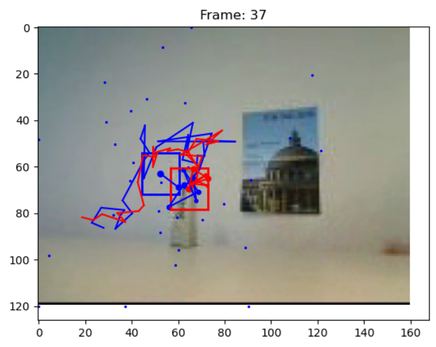
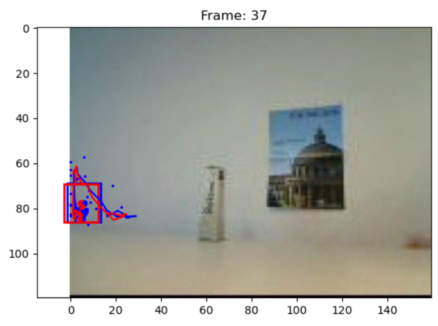
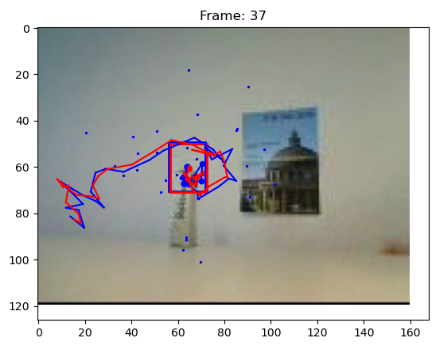
Video2(a moving hand with occlusion)

- best param

```
        "draw_plots": 1,
        "hist_bin": 16,
        "alpha": 0.7,
        "sigma_observe": 0.3,
        "model": 1,
        "num_particles": 30,
        "sigma_position": 15,
        "sigma_velocity": 1,
        "initial_velocity": (1, 1)
```

- successful and failure results : (**Figure 1.1 is SUCCESSFUL**)

constant velocity model (sigma_obs = 0.3, sigma_position = 15) SUCCESSFUL	constant velocity model (sigma_obs = 0.1)
	
static model	**constant velocity model (sigma_position=30)**

constant velocity model (sigma_obs = 0.3, sigma_position = 15) SUCCESSFUL	constant velocity model (sigma_obs = 0.1)
	
constant velocity model (sigma_position=7)	**constant velocity model (sigma_obs = 1)**
	

- Question 1: What is the effect of using a constant velocity motion model?
    - As the figure above, we can see that when using static model, the tracking box cannot follow the hand at the very beginning and intends to stay around.

      But if we change it to constant velocity motion model, the tracking box could follow the hand easily in the right direction.
- Question 2: What is the effect of assuming decreased/increased system noise?
    - Comparing figure 1.1 with 2.2 and 3.1:

      When increasing the position noise, the box tends to diverge to other places and then divide with the hand to somewhere far away.

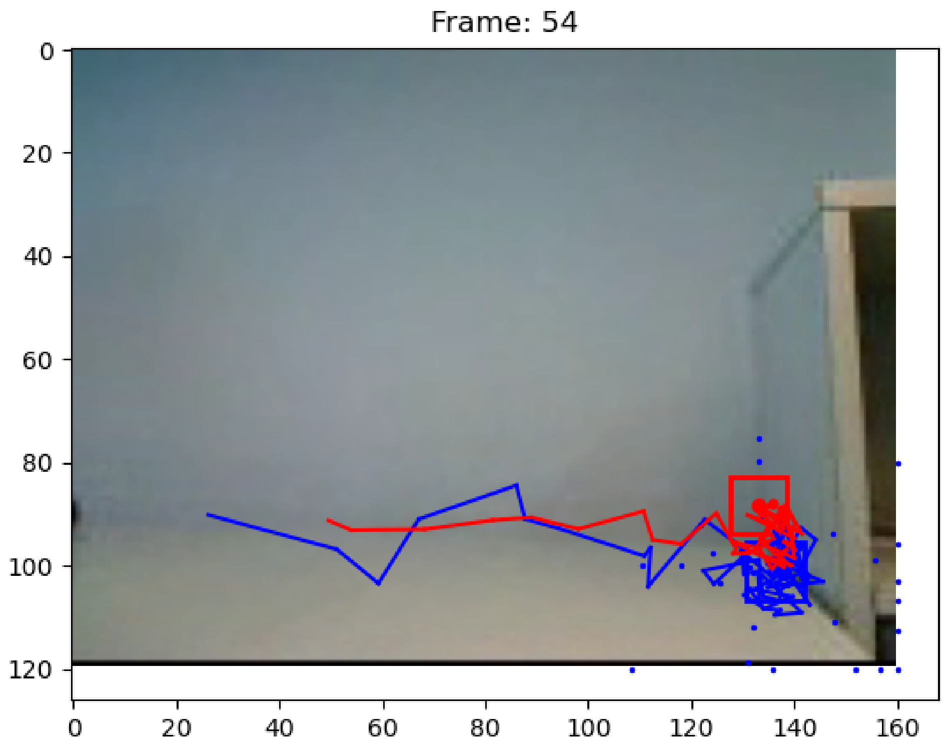
      When decreasing the position noise, the box is updated so conservatively and thus lose track of the hand and only stay at around the start position.
- Question 3: What is the effect of assuming decreased/increased measurement noise?
    - Comparing figure 1.1 with 1.2 and 3.2:

      When decrease the measurement noise, the updated weight tends to only consider the most similar particle when observing. So the box first tracks very well, but when it is occluded or near some complex background, it is easily have the wrong objective, which can be "greedy" or "myopic".

But when we increase the measurement noise so much, this box would be uninterested to the hand and consider more comprehensively. So it can lose track at the very beginning if you don't have the correct initial velocity guess. But if it tracks well at the beginning, it won't easily change objective when facing occlusion temporarily.

Video3(moving and rebounding ball)

- Use the best parameters for video2:

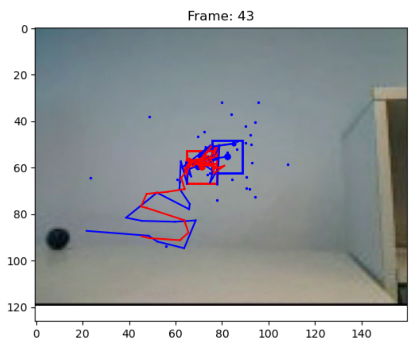
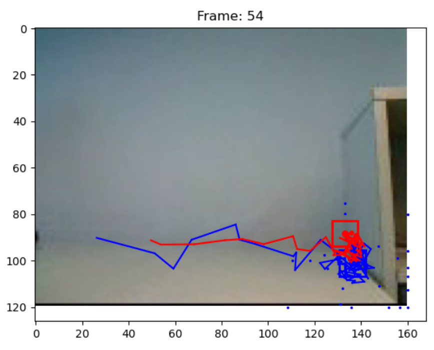


  - Can it track the ball?

    No, because when the box tracks the ball, the velocity is around (15, 0). It cannot change abruptly when the ball rebounds.

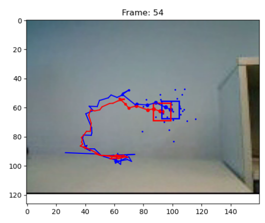
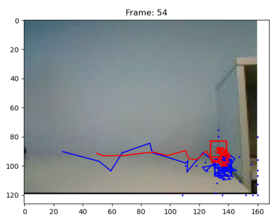
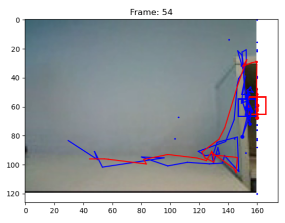
    As we can see above, the box can track perfectly when the ball is moving towards right fast. However, when it rebounds and the velocity change abruptly, the box will lose track and stay at the wall.

- Question 1: What is the effect of using a constant velocity motion model?

static model	constant velocity motion model
	

If we use static model, it would lose track easily because the velocity of the ball moves so fast. However if we use constant velocity motion model, it will at least follows the ball until touch the wall.

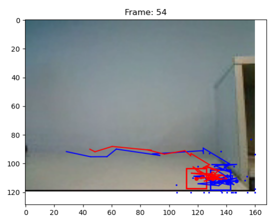
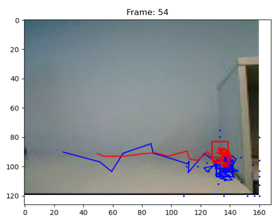
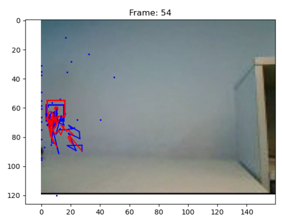
- Question 2: What is the effect of assuming decreased/increased system noise?

decrease noise(sigma_position=7)	**sigma_position=15**	**increase noise(sigma_position=30)**
 Frame: 54	 Frame: 54	 Frame: 54

Obviously, when we decrease noise, the box updates so conservatively. Because the ball move so fast, once it lose track, it will never get back to follow the ball.

On the contrary, when we increase noise, even though the ball move so fast at the very beginning, the box can easily track it regardless the initial velocity. But we can still see the box could diverge to somewhere far away from the ball.

- Question 3: What is the effect of assuming decreased/increased measurement noise?

decrease measurement noise(sigma_obs = 0.1)	**sigma_obs = 0.3**	**sigma_obs = 1**
 Frame: 54	 Frame: 54	 Frame: 54

We can see when we decrease measurement noise, the box mainly only focus on the most similar particle for updating. Even though the ball moves so fast at the very beginning, the box tracks it easily.

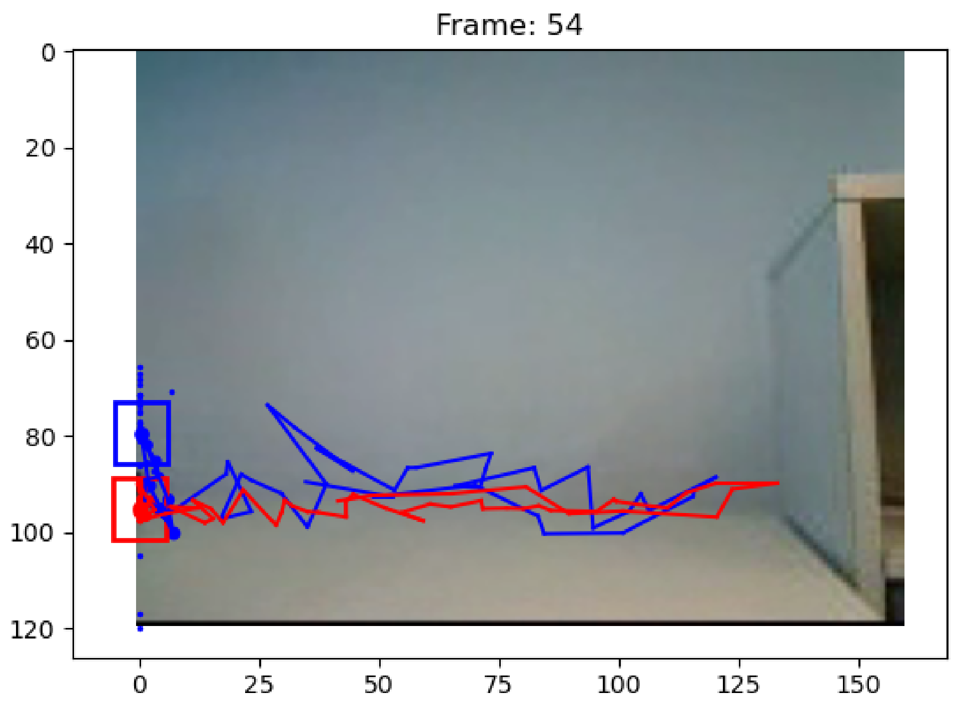
However, when we increase the measurement noise, the box wants to be comprehensive and lose tracks at the very beginning and only stay round the start position.

But I still get the perfect effect by tuning the parameters:

- best params

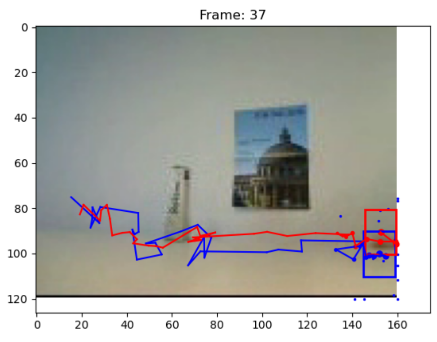
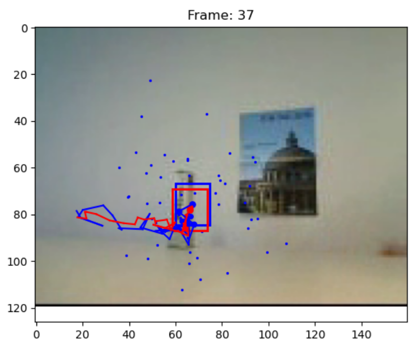
```
        "draw_plots": 1,
        "hist_bin": 16,
        "alpha": 0.1,
        "sigma_observe": 0.3,
        "model": 1,
        "num_particles": 30,
        "sigma_position": 15,
        "sigma_velocity": 4,
        "initial_velocity": (5, 0)
```

- **SUCCESSFUL RESULT**



Some more questions to consider

- Question 1: What is the effect of using more or fewer particles?

fewer particles	more particles
	

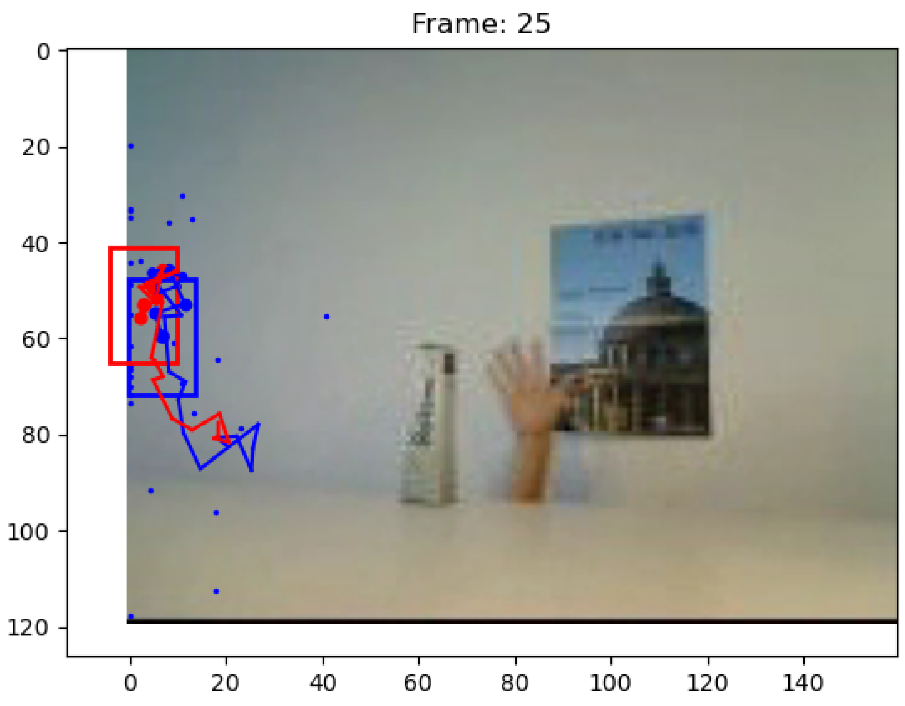
When fewer particles, the updating process will be very unsteady. Imagine that we only have samples not aligned with the objective, then it would definitely lose track and diverge.

When having more particles, the updating process will be very conservative. This is similar to the effect of increasing the measurement noise. So when the ball moves fast or having occlusion in front of hands, the box would lose track and pay attention to something else.

- Question 2: What is the effect of using more or fewer bins in the histogram color model?

  The most obvious effect is that the computation and space cost is high when we have more bins. This is because the possible numbers of color space is $histbin^3$, so we could not have more bins.

  When we have fewer bins, many colors will be treated vaguely and equally. Thus the box will easily track the wrong objective which has similar color(like the desk and the hand), it treats them the same. See figure below:

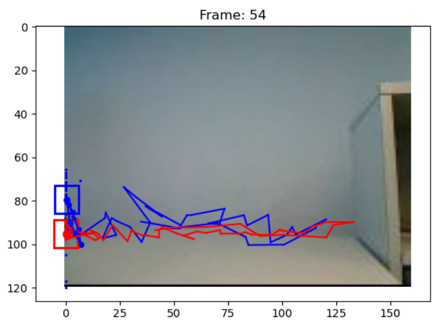
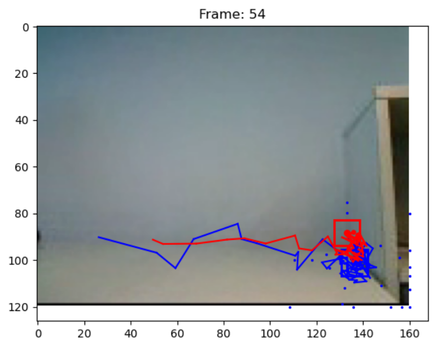
Fewer bins


- Question 3: What is the advantage/disadvantage of allowing appearance model updating?

  Advantage: Even the objective change shape or the light of the environment change, we can still track the objective perfectly as long as it changes gradually.

  Disadvantage: We cannot anchor or fix to the initial model, which means drift could happen and we track the wrong object.

no appearance model updating	appearance model updating

no appearance model updating	appearance model updating
	

The left one tracks the ball after rebounds, the right one doesn't.