Report

Implementation

1. **Colour histogram**

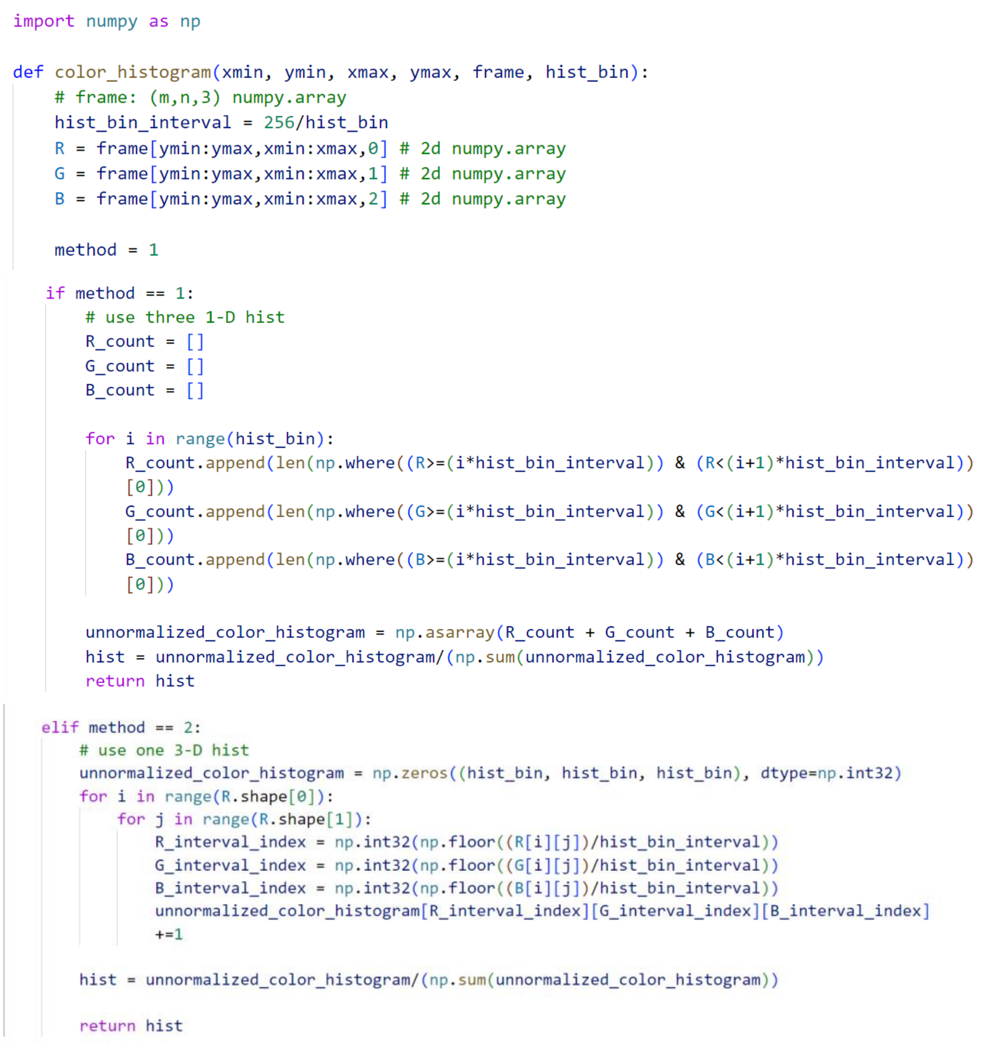
I use two methods to represent the colour histogram.

The first method is to count R, G, B channels separately. For each channel, split the whole range of colours (0 to 255) into hist_bin intervals and count the total number of the colours in each interval to build the histogram for each channel, and then concatenate them together to attain a 3×hist_bin-D vector, and then normalize it to get final histogram.

The second method is to build histogram for RGB combination for each pixel. Following the same idea as the first method, we can get a (hist_bin×hist_bin×hist_bin) D matrix and then normalize it to get the final histogram.

The difference between these two methods is the second method can recover the RGB combination for pixel, while in first method this information is discarded as each channel is counted separately. Thus, the second method is more expressive but at the cost of more storage memory required.

The code is as followed:



```python
import numpy as np

def color_histogram(xmin, ymin, xmax, ymax, frame, hist_bin):
    # frame: (m,n,3) numpy.array
    hist_bin_interval = 256/hist_bin
    R = frame[ymin:ymax,xmin:xmax,0] # 2d numpy.array
    G = frame[ymin:ymax,xmin:xmax,1] # 2d numpy.array
    B = frame[ymin:ymax,xmin:xmax,2] # 2d numpy.array

    method = 1

    if method == 1:
        # use three 1-D hist
        R_count = []
        G_count = []
        B_count = []

        for i in range(hist_bin):
            R_count.append(len(np.where((R>=(i*hist_bin_interval)) & (R<(i+1)*hist_bin_interval))
            [0]))
            G_count.append(len(np.where((G>=(i*hist_bin_interval)) & (G<(i+1)*hist_bin_interval))
            [0]))
            B_count.append(len(np.where((B>=(i*hist_bin_interval)) & (B<(i+1)*hist_bin_interval))
            [0]))

        unnormalized_color_histogram = np.asarray(R_count + G_count + B_count)
        hist = unnormalized_color_histogram/(np.sum(unnormalized_color_histogram))
        return hist

    elif method == 2:
        # use one 3-D hist
        unnormalized_color_histogram = np.zeros((hist_bin, hist_bin, hist_bin), dtype=np.int32)
        for i in range(R.shape[0]):
            for j in range(R.shape[1]):
                R_interval_index = np.int32(np.floor((R[i][j])/hist_bin_interval))
                G_interval_index = np.int32(np.floor((G[i][j])/hist_bin_interval))
                B_interval_index = np.int32(np.floor((B[i][j])/hist_bin_interval))
                unnormalized_color_histogram[R_interval_index][G_interval_index][B_interval_index]
                +=1

        hist = unnormalized_color_histogram/(np.sum(unnormalized_color_histogram))

        return hist
```

2. **Derive matrix A**

We consider two models: (i) no motion; (ii) constant velocity model.

(i) no motion model:

The model is 2-d:

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \end{bmatrix} + \begin{bmatrix} \omega 1_{t-1} \\ \omega 2_{t-1} \end{bmatrix}$$

Since there's no motion, we have: $x_t = x_{t-1}, y_t = y_{t-1}$, if there's no noise.

So we know $a = 1,\ b = 0,\ c = 0,\ d = 1$. A is an identity matrix.

(ii) constant velocity model:

The model is 4-d:

$$\begin{bmatrix} x_t \\ y_t \\ \dot{x}_t \\ \dot{y}_t \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \dot{x}_{t-1} \\ \dot{y}_{t-1} \end{bmatrix} + \begin{bmatrix} \omega 1_{t-1} \\ \omega 2_{t-1} \\ \omega 3_{t-1} \\ \omega 4_{t-1} \end{bmatrix}$$

Since velocity is constant, we have: $\dot{x}_t = \dot{x}_{t-1}, \dot{y}_t = \dot{y}_{t-1}, x_t = x_{t-1} + \dot{x}_{t-1}\Delta t$,

$y_t = y_{t-1} + \dot{y}_{t-1}\Delta t$, if there's no noise. In our case, $\Delta t = 1$.

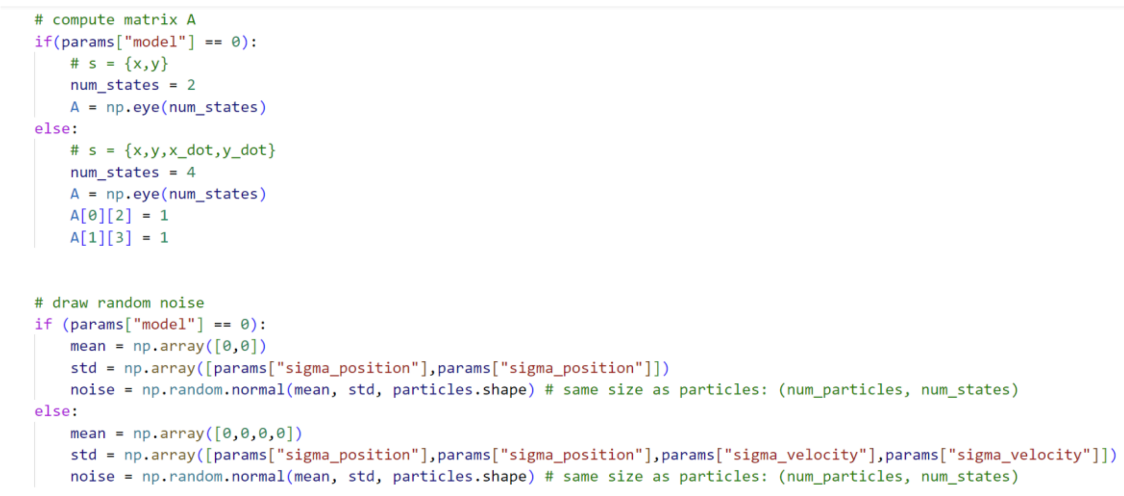
So we know $a = 1,\ b = 0,\ c = 1,\ d = 0, e = 0,\ f = 1,\ g = 0,\ h = 1,$

$i = 0,\ j = 0,\ k = 1,\ l = 0, m = 0,\ n = 0,\ o = 0,\ p = 1.$

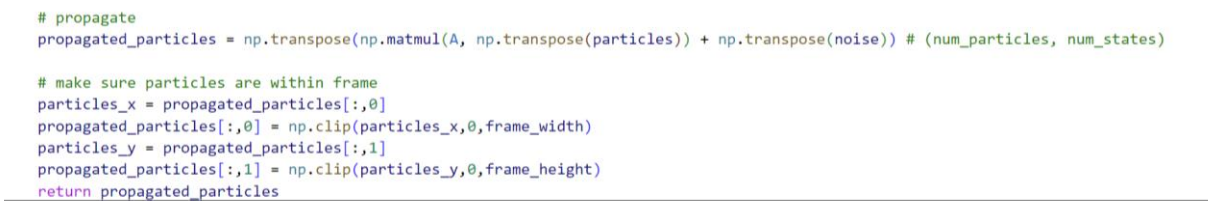
$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. **Propagation**

Use the matrix A derived above and consider the noise to be Gaussian noise, we can propagate particles via the formular above. To ensure the center of the particles lie inside the frame, I use the clip function to truncate the particles if they are outside of the frame. The code is as followed:



```
# compute matrix A
if(params["model"] == 0):
    # s = {x,y}
    num_states = 2
    A = np.eye(num_states)
else:
    # s = {x,y,x_dot,y_dot}
    num_states = 4
    A = np.eye(num_states)
    A[0][2] = 1
    A[1][3] = 1


# draw random noise
if (params["model"] == 0):
    mean = np.array([0,0])
    std = np.array([params["sigma_position"],params["sigma_position"]])
    noise = np.random.normal(mean, std, particles.shape) # same size as particles: (num_particles, num_states)
else:
    mean = np.array([0,0,0,0])
    std = np.array([params["sigma_position"],params["sigma_position"],params["sigma_velocity"],params["sigma_velocity"]])
    noise = np.random.normal(mean, std, particles.shape) # same size as particles: (num_particles, num_states)
```



```
# propagate
propagated_particles = np.transpose(np.matmul(A, np.transpose(particles)) + np.transpose(noise)) # (num_particles, num_states)

# make sure particles are within frame
particles_x = propagated_particles[:,0]
propagated_particles[:,0] = np.clip(particles_x,0,frame_width)
particles_y = propagated_particles[:,1]
propagated_particles[:,1] = np.clip(particles_y,0,frame_height)
return propagated_particles
```

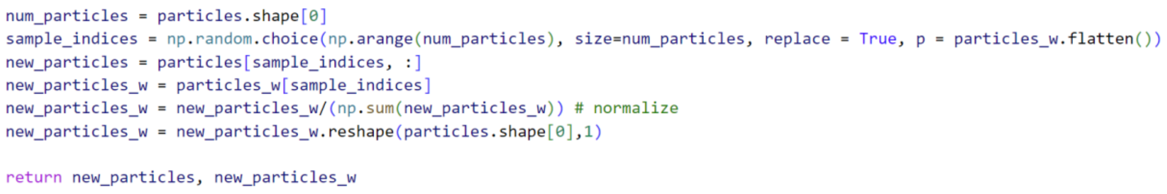
4. **Observation**
   We firstly compute the colour histograms for each particle. In this step, we need to make sure the input params xmin, ymin, xmax, ymax of color_histogram function to be within the frame to guarantee color_histogram function works well. If the bounding box of a particle is partly outside the frame (the center is surely within the frame), we only consider the part of the bounding box which is inside the frame. After having computed the histograms for each pixel, we computed their χ^2 distance to the target histogram. And use this to compute the weight for each pixel. Finally normalize the weights to make sure they sum to 1.
   The code is as followed:

```
particles_w = []

for i in range(particles.shape[0]):
    xmin = np.int32(np.max([np.floor(particles[i][0] - bbox_width/2), 0]))
    xmax = np.int32(np.min([xmin + bbox_width, (frame.shape[1]-1)]))
    ymin = np.int32(np.max([np.floor(particles[i][1] - bbox_height/2), 0]))
    ymax = np.int32(np.min([ymin + bbox_height, (frame.shape[0]-1)]))
    hist_x = color_histogram(xmin, ymin, xmax, ymax, frame, hist_bin)
    dist = chi2_cost(hist_x, hist)
    pi_i = 1/(np.sqrt(2*np.pi)*sigma_observe) * np.exp(-(dist**2)/(2*(sigma_observe**2)))
    particles_w.append(pi_i)


particles_w = np.asarray(particles_w)
particles_w = particles_w/(np.sum(particles_w)) # normalize
particles_w = particles_w.reshape(particles.shape[0],1)

return particles_w
```

5. **Estimation**
   After attaining the particles and their weights, we simply compute the weighted average as our estimation.
   The code is as followed:

```
num_states = particles.shape[1]
mean_state = []
for i in range(num_states):
    state_i = particles[:,i]
    mean_state_i = np.sum(state_i * np.transpose(particles_w))
    mean_state.append(mean_state_i)

mean_state = np.asarray(mean_state)
return mean_state
```

6. **Resampling**
   We resample the particles according to their weights. After we obtained the new particles, we need to re-normalize their weights to make sure they sum up to 1. The code is as followed:



```
num_particles = particles.shape[0]
sample_indices = np.random.choice(np.arange(num_particles), size=num_particles, replace = True, p = particles_w.flatten())
new_particles = particles[sample_indices, :]
new_particles_w = particles_w[sample_indices]
new_particles_w = new_particles_w/(np.sum(new_particles_w)) # normalize
new_particles_w = new_particles_w.reshape(particles.shape[0],1)

return new_particles, new_particles_w
```

Experiments

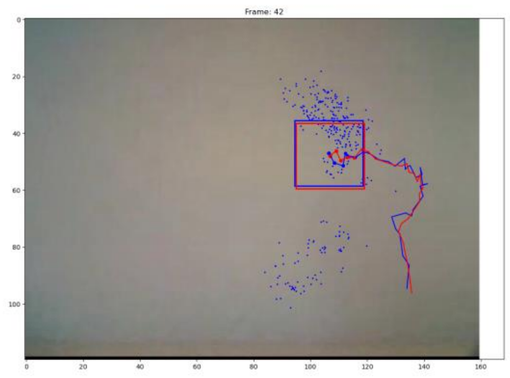
The following experiments are all based on colour histograms using the first method

(3×hist_bin-D vector).

1. **Video1**

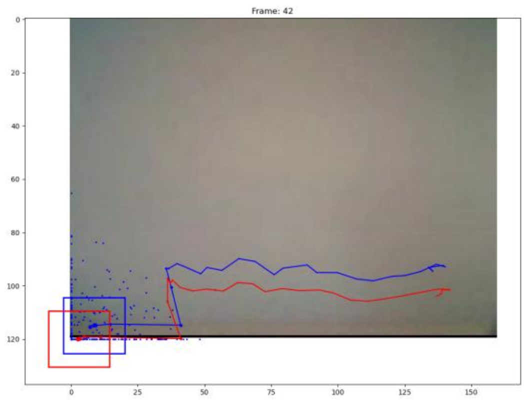
   Unsuccessful tracking example

   (hist_bin = 16, alpha = 0.2, sigma_observe = 0.1, model = 1, num_particles = 300, sigma_position = 1, sigma_velocity = 1, initial_velocity = (-1, -10)):



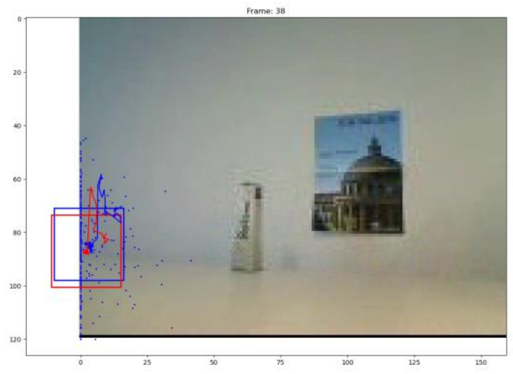
   Successful tracking example

   (hist_bin = 16, alpha = 0.2, sigma_observe = 0.1, model = 1, num_particles = 300, sigma_position = 15, sigma_velocity = 1, initial_velocity = (-1, -10)):



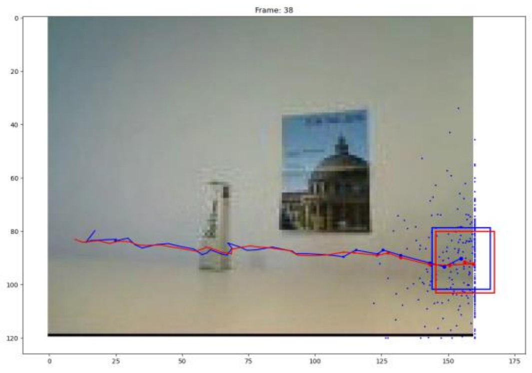
2. **Video2**
Unsuccessful tracking example:

(hist_bin = 16, alpha = 0.2, sigma_observe = 0.1, model = 1, num_particles = 300, sigma_position = 15, sigma_velocity = 1, initial_velocity = (-1, -10)):



Successful tracking example:

(hist_bin = 16, alpha = 0.2, sigma_observe = 0.1, model = 0, num_particles = 300, sigma_position = 15):



• What is the effect of using a constant velocity motion model?
It turns out that using the constant velocity motion model does not necessarily have better result than the no motion model. The key issue is the initial velocity. If the object moves exactly in the same direction as the set initial velocity, the tracker can track the object better. However, if we set our initial velocity in wrong direction, the tracker can get totally lost. Thus, if we don't have any prior information about the object velocity, simply using the "no motion model" could be a better idea.

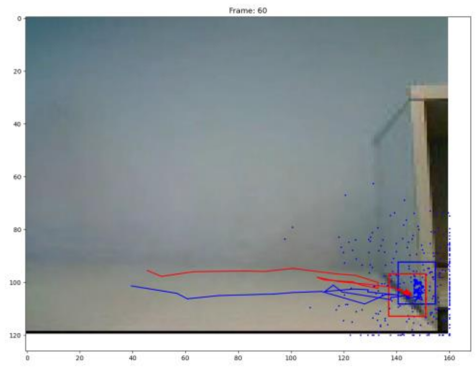
• What is the effect of assuming decreased/increased system noise?
If increase the system noise, which brings more uncertainty, the particles would spread in a large range. If decrease the system noise, the particles would be more concentrated and spread in a smaller range. However, this would lead to a greater possibility that the true object may lie outside the range covered by the particles, which could make the tracker lose its object, especially then the object moves fast.

• What is the effect of assuming decreased/increased measurement noise?
If increase the measurement noise, the Gaussian distribution will have bigger variance and thus more particles will be considered when computing the mean, which means the computed mean is "sensitive" to the outliers. Decrease the measurement noise would have opposite effect. However, it might give too much weight to only few

particles and thus, be “sensitive” to those few particles. It turns out that too big and too small measurement noise both lead to bad result. This is a parameter which should be tuned carefully as it strongly effects the update mechanism via weights computation.

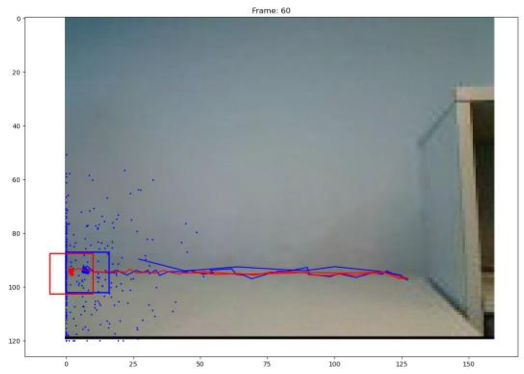
3. **Video3**
   Unsuccessful tracking example:

   (hist_bin = 16, alpha = 0.2, sigma_observe = 0.1, model = 1, num_particles = 300, sigma_position = 15, sigma_velocity = 1, initial_velocity = (10, 10)):



   Successful tracking example:

   (hist_bin = 16, alpha = 0.2, sigma_observe = 0.1, model = 0, num_particles = 300, sigma_position = 15):



   It turns out that the parameters for video2 also work for video 3. The reason might be that I use “no motion” model for both videos so it is more robust to the changing velocity directions of the object. Also, I set the model noise to a relatively large value (15), so the particles spread in a larger range and are able to cover the moving object.

   Rethink the questions in video2:
   • What is the effect of using a constant velocity motion model?
   We have similar conclusion as video2. Since the ball change its moving direction quickly, using the constant velocity motion model might lost the object. In this case, simply using the “no motion model” could be a better idea.
   • What is the effect of assuming decreased/increased system noise?
   We have similar conclusion as video2. Increase/decrease the system noise would make the particles spread in larger/smaller range. Too small system noise might lost the

object when the object goes outside of the covered range, while too big system noise would lead to inaccurate particles.

• What is the effect of assuming decreased/increased measurement noise?

We have similar conclusion as video2. However, in this case, since the colour histogram of the ball almost does not change, we can set the measurement noise to relatively small value to give more weights to the fewer particles which have similar histogram as the target histogram without being afraid of not able to track the changing target histogram.

Extra questions:

• What is the effect of using more or fewer particles?

Since we use particles to approximate distribution, according to the *law of large numbers*, the more particles we use, the more accurate our approximation is. However, more particles would also lead to larger computational cost. Thus, we need to do trade-off between accuracy and efficiency.

• What is the effect of using more or fewer bins in the histogram colour model?

The more bins we use, the more expressive our representation is. If we use too few bins, the tracker can't distinguish our object from similar patches and tend to lose the object if it moves fast or changes direction fast when there's similar patches nearby. Since the colour histogram itself is not a very expressive method to represent the object, it would be a good idea to set the number of bins to a large value, for example, 16, in our case.

• What is the advantage/disadvantage of allowing appearance model updating?

If the appearance of the object changes during the tracking process, allowing appearance model updating is a better idea, otherwise, the target we try to track will become different from the true object, which means we are essentially tracking "other thing" instead of the true object. However, we can't update the appearance in a too radical way either (set alpha to nearly 1). Because in this case, the target histogram would be essentially the histogram of the mean estimate particle, however, this mean estimate is not 100% accurate (not exactly the same as the center of the object) and could thus lead to false update. In conclusion, the value of alpha depends on many factors such as how accurate our estimate is, how fast the object changes its appearance and so on and we need to tune alpha carefully.