Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Vision
and Geometry Lab

ETH Zurich
Computer Science Department

Computer Vision

Lab Assignment - Condensation Tracker

Lara Nonino - lnonino@student.ethz.ch

December 2023

Student ID:
22-937-833

Section 1

Implementation

The first task was to implement a *Condensation tracker* based on color histogram using python. The *Condensation algorithm* is a sample-based solution of recursive Bayesian filter.

1.1 Color histogram

In the first place, the function `color_histogram.py` had to be computed. This function calculates the normalized histogram of RGB colors occurring within the bounding box defined by (xmin, xmax) (ymin, ymax) within the current video frame.
To do this, after checking that the borders of the bounding box lied inside the frame (and adjusting the values if needed), the `numpy` function `histogram` was applied to the 3 colors of the box. The 3 histograms were then normalised and stacked together. Instead of the full 256^3 possible colours in 24-bit color space, only $hist\_bin^3$ colors were considered.

```
def color_histogram(xmin, ymin, xmax, ymax, frame, hist_bin):

    hist = np.zeros((3, hist_bin))  # hist.shape = [3, hist_bin]

    x_min = max(0, round(xmin))
    y_min = max(0, round(ymin))
    x_max = min(frame.shape[1] - 1, round(xmax))
    y_max = min(frame.shape[0] - 1, round(ymax))

    color_0 = frame[y_min:y_max, x_min:x_max, 0]
    color_1 = frame[y_min:y_max, x_min:x_max, 1]
    color_2 = frame[y_min:y_max, x_min:x_max, 2]

    hist[0] = np.histogram(color_0, bins=hist_bin)[0]
    hist[1] = np.histogram(color_1, bins=hist_bin)[0]
    hist[2] = np.histogram(color_2, bins=hist_bin)[0]

    hist = hist/np.sum(hist)    # normalized histogram
    hist = hist.flatten()

    return hist
```

1.2 Derive matrix A

The next step was deriving the dynamic matrix A for the two prediction models: (i) no motion at all i.e. just noise; and (ii) constant velocity motion model.

In order to do this, it was necessary to look at how the states of the objects to be tracked were represented. The representation used was

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

where x, y represent the location of center of the bounding-box, and $\dot{x}$, $\dot{y}$ the velocities in the x and the y direction.
Since each sample evolves by the system model

$$s_t'^{(n)} = As_{t-1}^{(n)} + Bw_{t-1}^{(n)}$$

to compute A, it is possible to only look at the deterministic component of the model.
For the no motion at all model, the model will be

$$\begin{cases} x_t = x_{t-1} \\ y_t = y_{t-1} \end{cases} \qquad \begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \end{bmatrix} \qquad A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

While for the constant velocity motion model, the model will be

$$\begin{cases} x_t = x_{t-1} + \dot{x} \cdot \partial t \\ y_t = y_{t-1} + \dot{y} \cdot \partial t \\ \dot{x}_t = \dot{x}_{t-1} \\ \dot{y}_t = \dot{y}_{t-1} \end{cases} \qquad \begin{bmatrix} x_t \\ y_t \\ \dot{x}_t \\ \dot{y}_t \end{bmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \dot{x}_{t-1} \\ \dot{y}_{t-1} \end{bmatrix}$$

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

where ∂t is considered to be 1 since the time step chosen is the time between two consecutive video frames.

1.3 Propagation

Afterward, `propagate.py` was implemented so that it propagates the particles given the system prediction model and the system model noise as it follows

$$s_t'^{(n)} = As_{t-1}^{(n)} + Bw_{t-1}^{(n)}$$

Moreover, the parameter frame_height and frame_width were used to make sure that the centre of the particles lied inside the frame.

```
def propagate(particles, frame_height, frame_width, params):

    if params["model"] == 0:    # no motion
        A = np.array([[1, 0], [0, 1]])
        noise = np.array([params["sigma_position"], params["sigma_position"]])

    if params["model"] == 1:    # constant velocity motion
        A = np.array([[1, 0, 1, 0], [0, 1, 0, 1], [0, 0, 1, 0], [0, 0, 0, 1]])
        noise = np.array([params["sigma_position"], params["sigma_position"],
                          params["sigma_velocity"], params["sigma_velocity"]])

    deterministic = np.matmul(A, particles.T).T
    stochastic = noise * np.random.randn(particles.shape[0], particles.shape[1])

    particles_update = deterministic + stochastic
    particles_update[:, 0] = np.clip(particles_update[:, 0], 0, frame_width-1)
```

```
    particles_update[:, 1] = np.clip(particles_update[:, 1], 0, frame_height - 1)

    return particles_update
```

1.4 Observation

The function `observe.py` had to compute the weights for particles based on color histogram. To do so, the first step was ensuring that each particle's bounding box defined by the centre of the particle, *bbox_height* and *bbox_width* lied inside the image. The next step consisted in computing the color histogram corresponding to each particle and afterwards compute the χ^2 distance between the new histogram and the target one (the function `chi2_cost.py` was provided). At this point it was possible to compute the weights *particles_w* using the following equation

$$\pi^{(n)} = \frac{1}{\sqrt{2\pi\sigma}} \cdot e^{-\frac{\chi^2(CH_s, CH_{target})^2}{2\sigma^2}}$$

At last, *particles_w* was normalized.

```
def observe(particles, frame, bbox_height, bbox_width, hist_bin, hist, sigma_observe):

    xs_min = particles[:, 0] - bbox_width / 2
    xs_max = particles[:, 0] + bbox_width / 2
    ys_min = particles[:, 1] - bbox_height / 2
    ys_max = particles[:, 1] + bbox_height / 2

    particles_w = np.zeros((len(particles), 1))

    for i in range(len(particles)):
        hist_i = color_histogram(xs_min[i], ys_min[i], xs_max[i], ys_max[i], frame, hist_bin)
        chi_i = chi2_cost(hist_i, hist)
        particles_w[i] = (1 / (np.sqrt(2 * np.pi) * sigma_observe)) * \
                         np.exp(- (chi_i ** 2) / ((sigma_observe ** 2) * 2))

    particles_w = particles_w/sum(particles_w)

    return particles_w
```

1.5 Estimation

The `estimation.py` function estimates the mean state given the particles and their weights as

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

```
def estimate(particles, particles_w):

    mean_state = np.sum(particles*particles_w, 0)/np.sum(particles_w)

    return mean_state
```

1.6 Resampling

The last part of the implementation consisted in writing the function `resample.py` which resamples the particles based on their weights and returns the new particles along with their corresponding weights. This was done using `numpy.random.choice` through which it was possible to sample the indices of the particles according to

their weights. Once this was done it was possible to update both the particles and the particles weights and normalize them.

```
def resample(particles, particles_w):

    update_indices = np.random.choice(len(particles),
                                      size=len(particles),
                                      replace=True,
                                      p=particles_w.flatten())

    particles_update = particles[update_indices]
    particles_w_update = particles_w[update_indices] / sum(particles_w[update_indices])

    return particles_update, particles_w_update
corners[:,[0,1]] = corners[:,[1,0]]
```

Section 2

Experiments

The experiment was to track objects in three videos using the tracker code written. Each video shows a moving object in different conditions. The tracking is shown through the plotting of two colored lines and particles: the blue ones represent the mean state and the particles *a priori*, while the red ones represent the mean state and the particles *a posteriori*. In each table, an example of successful tracking will be highlighted in green and an example of unsuccessful tracking in red.

2.1 Video 1

The first video (*video1.wmv*) shows a moving hand with uniform background. The results and the corresponding parameter settings are shown below.

Figure	hist_bin	α	σ_{obs}	model	num_particles	σ_{pos}	σ_{vel}	initial_vel
2.1	16	0	0.1	0	30	15	-	-
2.2	16	0	0.1	0	500	15	-	-
2.3	5	0	0.1	0	250	15	-	-
2.4	200	0	0.1	0	250	15	-	-
2.5	16	0	0.05	0	250	15	-	-
2.6	16	0	0.8	0	250	15	-	-
2.7	16	0.1	0.1	0	250	15	-	-
2.8	16	0.5	0.1	0	250	15	-	-
2.9	16	0	0.1	0	250	12	-	-
2.10	16	0	0.1	0	250	20	-	-
2.11	16	0	0.1	1	250	15	1	(-1,-10)
2.12	16	0	0.1	1	250	15	10	(-1,-10)

Table 2.1: Parameters for *video1.wmv*

Since this video has a uniform background and the hand is fully visible and not occluded, it is possible to study how changing different parameters affects the tracking.

num_particles: from the comparison between Figure 2.1 and Figure 2.2, it stands out that a greater number of particles leads to a better tracking. This is because the continuous distribution of model movement is being approximated through a finite set of particles in Monte Carlo method. On the other side, it is not possible to choose an arbitrarily large number of particles because this would cause an increase of the computational cost.

hist_bin: from the comparison between Figure 2.3 and Figure 2.4, it stands out how a higher number of histogram bins will make distinguishing the different color shades easier. However, having a number of bins too large will make the tracker not enough robust to noise. In fact, a high number of bins will detect more shades of the colors, including the ones caused by noise.

σ_{obs}: from the comparison between Figure 2.5 and Figure 2.6, it is possible to see how changing the measurement noise affects the tracking. With a high measurement noise weights are more spread, therefore all particles are being given importance and the tracker tends to be more steady. With a lower measurement noise, the weight distribution results to be narrow and only the particles with similar histograms will be counted.

α: from the comparison between Figure 2.7 and Figure 2.8, it is possible to see how allowing appearance model updating helps the tracking. Moreover, in this particular case with no occlusion nor complex background this is particularly efficient.

σ_{pos}: from the comparison between Figure 2.9 and Figure 2.10, it is possible to see the impact that the system noise has on the tracking. With a high system noise the particles are spread on a large surface. This will help in case of occlusions, as it is possible to see in *video2.wmv*.

σ_{vel}: from the comparison between Figure 2.11 and Figure 2.12, it is possible to see how a constant velocity model with different velocity variances changes the tracking. In particular, with a large speed variance, the velocity will be corrupted by a large noise.

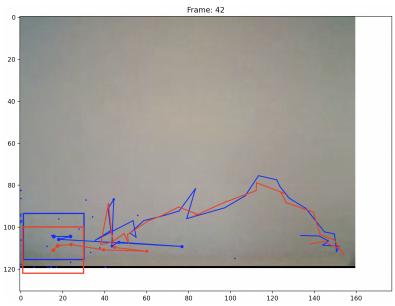

Figure 2.1

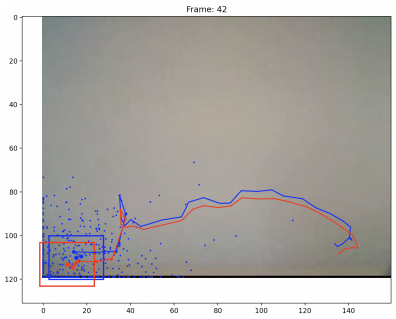

Figure 2.2

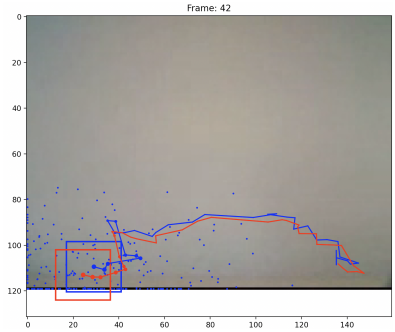

Figure 2.3

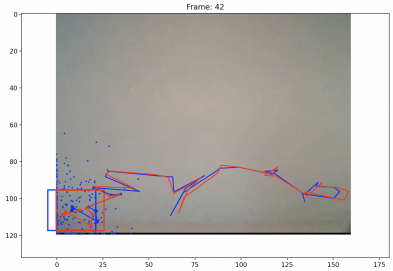

Figure 2.4

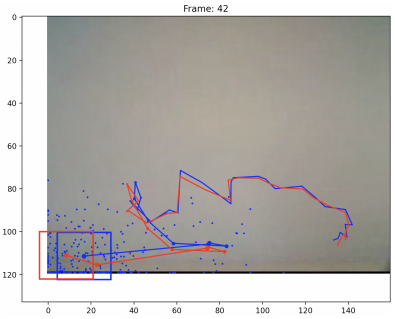

Figure 2.5

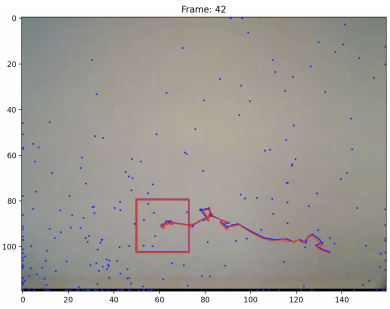

Figure 2.6

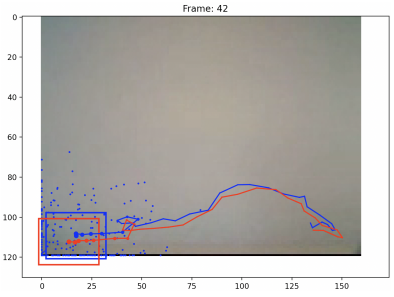

Figure 2.7

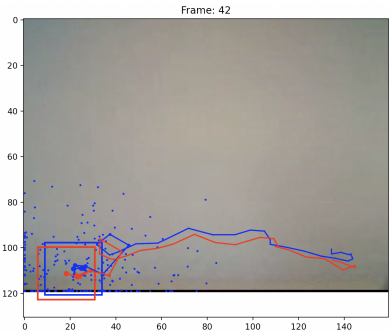

Figure 2.8

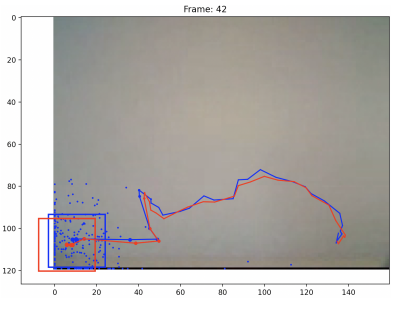

Figure 2.9

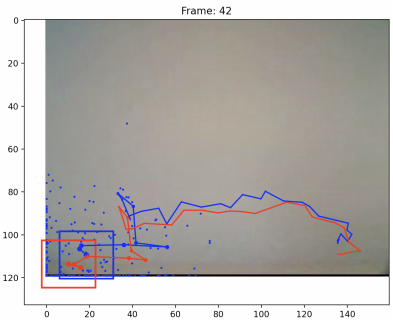

Figure 2.10

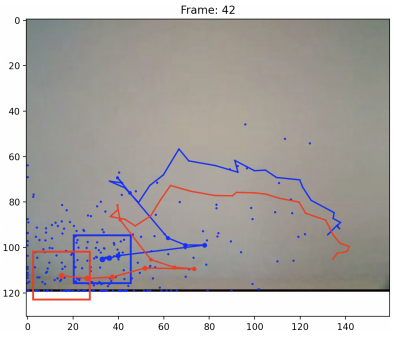

Figure 2.11

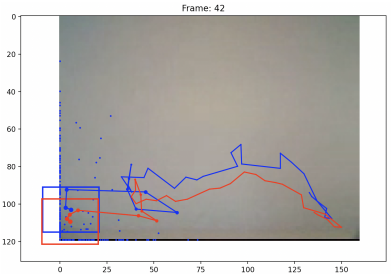

Figure 2.12

2.2 Video 2

The second video (*video2.wmv*) shows a moving hand with some clutter and occlusions. The results and the corresponding parameter settings are shown below.

Figure	hist_bin	α	σ_{obs}	model	num_particles	σ_{pos}	σ_{vel}	initial_vel
2.13	16	0	0.1	0	30	15	-	-
2.14	16	0	0.1	0	500	15	-	-
2.15	5	0	0.1	0	250	15	-	-
2.16	200	0	0.1	0	250	15	-	-
2.17	16	0	0.05	0	250	15	-	-
2.18	16	0	0.8	0	250	15	-	-
2.19	16	0.1	0.1	0	250	15	-	-
2.20	16	0.5	0.1	0	250	15	-	-
2.21	16	0	0.1	0	250	12	-	-
2.22	16	0	0.1	0	250	20	-	-
2.23	16	0	0.1	1	250	15	1	(1, 10)
2.24	16	0	0.1	1	250	15	10	(1,10)

Table 2.2: Parameters for *video2.wmv*

- From the comparison between the last two experiments (Figure 2.23 - Figure 2.24) and the other cases, it is possible to see which is the effect of using a constant velocity model. Firstly, the trajectory of the object results much smoother using a constant velocity model. This is because the model incorporates the fact that the object is moving linearly. Secondly, a higher velocity variance represents the level of uncertainty, therefore if the object does not move with constant velocity a large variance might be useful.
- As already stated for *video1.wmv*, with a high system noise the particles are spread on a large surface. This helps in case of occlusion since having particles more spread helps the tracker following the object even when is not fully visible.
- With a high measurement noise weights are more spread, therefore all particles are being given importance and the tracker tends to be more steady (an extreme measurement noise is used in Figure 2.18). With a lower measurement noise, the weight distribution results to be narrow and only the particles with similar histograms will be counted.

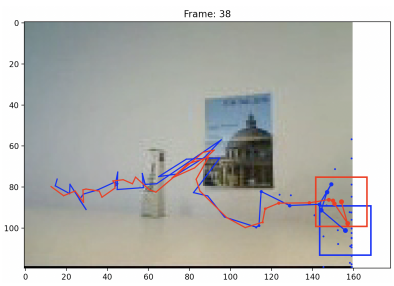

Figure 2.13

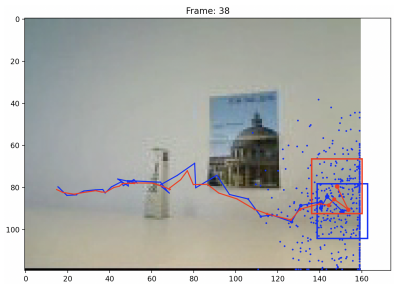

Figure 2.14

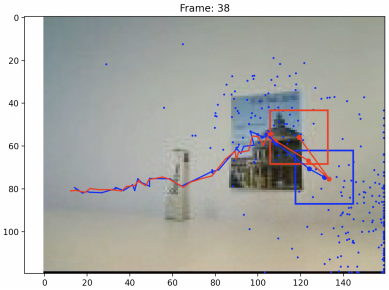

Figure 2.15

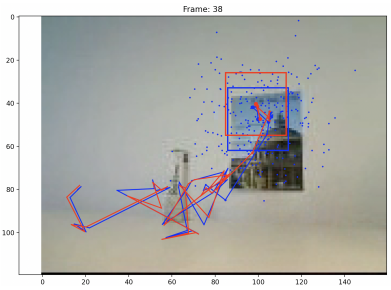

Figure 2.16

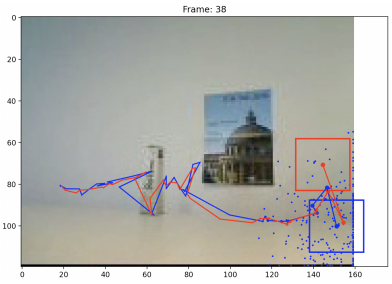

Figure 2.17

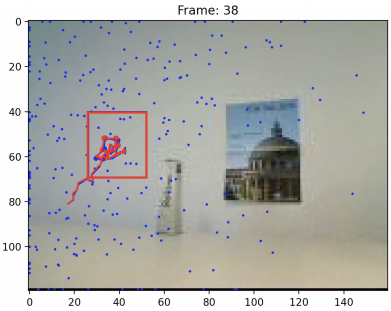

Figure 2.18

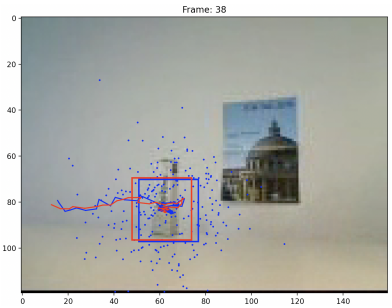

Figure 2.19

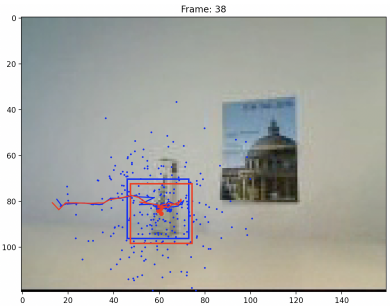

Figure 2.20

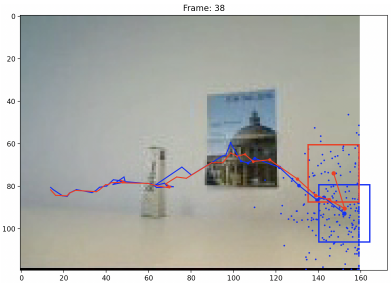

Figure 2.21

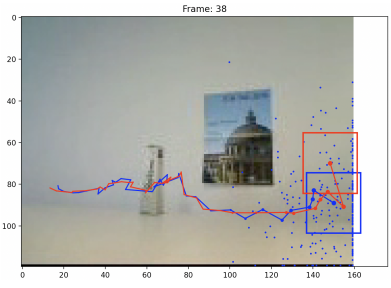

Figure 2.22

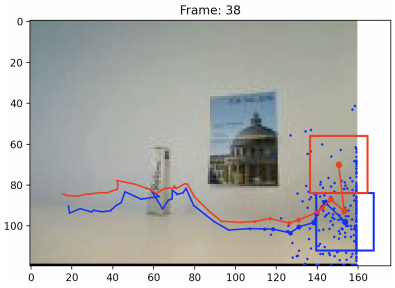

Figure 2.23

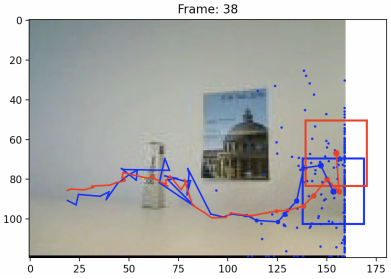

Figure 2.24

2.3 Video 3

The third video (*video3.wmv*) shows a ball bouncing. The results and the corresponding parameter settings are shown below.

Figure	hist_bin	α	σ_{obs}	model	num_particles	σ_{pos}	σ_{vel}	initial_vel
2.25	16	0	0.1	0	30	15	-	-
2.26	16	0	0.1	0	500	15	-	-
2.27	5	0	0.1	0	250	15	-	-
2.28	200	0	0.1	0	250	15	-	-
2.29	16	0	0.05	0	250	15	-	-
2.30	16	0	0.8	0	250	15	-	-
2.31	16	0.1	0.1	0	250	15	-	-
2.32	16	0.5	0.1	0	250	15	-	-
2.33	16	0	0.1	0	250	12	-	-
2.34	16	0	0.1	0	250	20	-	-
2.35	16	0	0.1	1	250	15	1	(1, 10)
2.36	16	0	0.1	1	250	15	10	(1, 10)

Table 2.3: Parameters for *video3.wmv*

- In the last two experiments (Figure 2.35 - Figure 2.36) a constant velocity model is being used. In contrast to *video2.wmv*, in this case a constant velocity model is not appropriate. This is because the bouncing ball velocity changes direction once it hits the shelf. Therefore, the no motion model with position variance would better fit this case.

- As already stated for *video1.wmv* and *video2.wmv*, with a high system noise the particles are spread on a large surface. In this video there are no occluding objects, therefore a lower value better fits the model (Figure 2.33).

- With a high measurement noise weights are more spread, therefore all particles are being given importance and the tracker tends to be more steady (Figure 2.30). With a lower measurement noise, the weight distribution results to be narrow and only the particles with similar histograms will be counted. In this video, since the ball is the only black object, and therefore has a unique histogram, a low measurement noise has a positive impact.

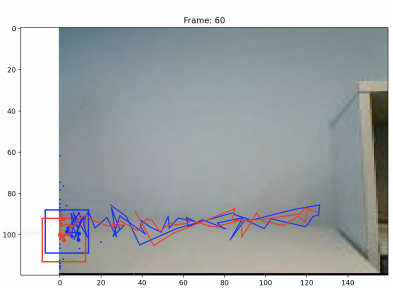

Figure 2.25

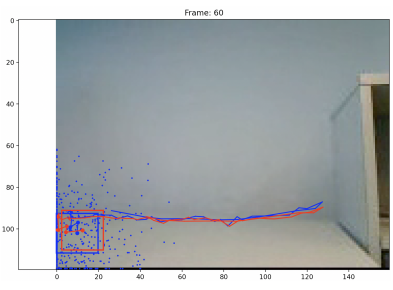

Figure 2.26

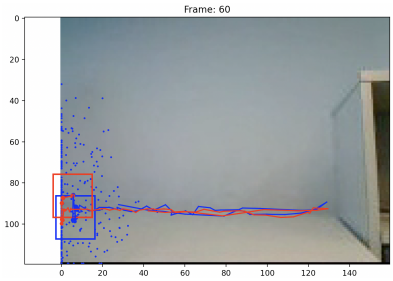

Figure 2.27

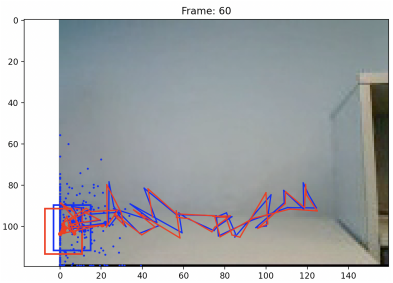

Figure 2.28

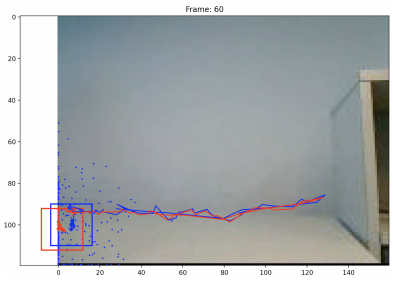

Figure 2.29

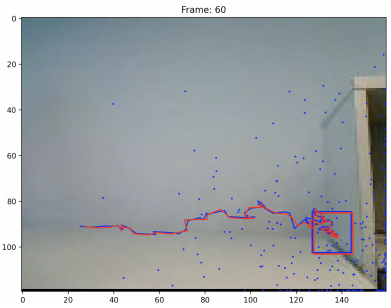

Figure 2.30

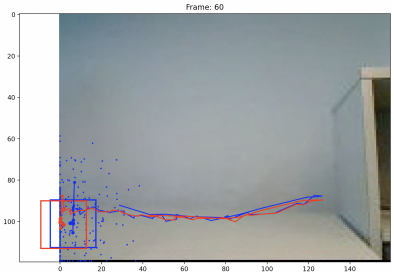

Figure 2.31

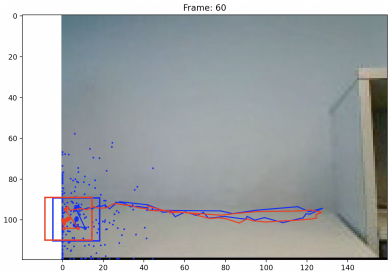

Figure 2.32

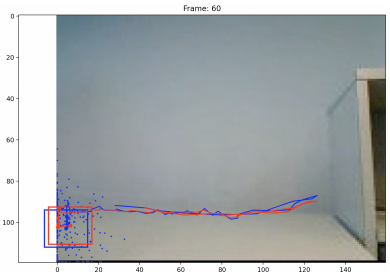

Figure 2.33

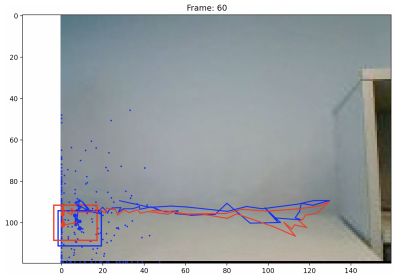

Figure 2.34

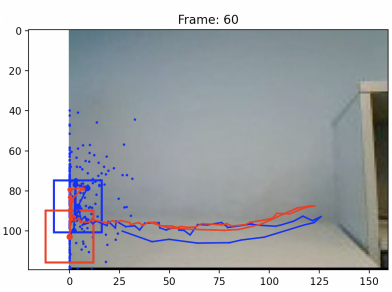

Figure 2.35

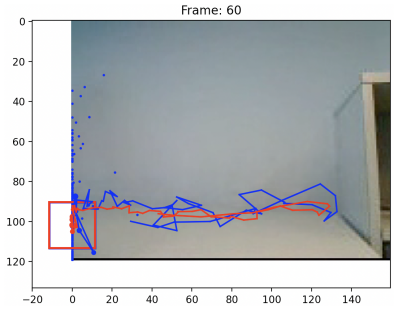

Figure 2.36

2.4 Comments on parameters

- A greater number of particles leads to a better tracking. As already stated for *video1.wmv*, this is because the continuous distribution of model movement is being approximated through a finite set of particles in Monte Carlo method. On the other side, it is not possible to choose an arbitrarily large number of particles because this would cause an increase of the computational cost.

- As already pointed out for *video1.wmv*, a higher number of histogram bins will make distinguishing the different color shades easier. However, having a number of bins too large will make the tracker not enough robust to noise. This is exactly what happened in Figure 2.28, where 200 bins were used. In contrast, Figure 2.27 shows a tracking done with only 5 bins. As expected, the result is much smoother and linear.

- Allowing appearance model updating has a positive effect when there is a changing in the illumination of the scene or in the background (as in *video2.wmv*) and the initial histogram is not representative of the object anymore. In particular, in Figure 2.32 it is possible to see how increasing the parameter α makes the shadow of the shelf have a smaller influence on the tracking. In contrast, having an update in the appearance model might lead to a negative tracking in case of occlusions, where the model might consider the wrong object.