Condensation Tracker

Computer Vision course, ETH Zurich

2023

The General Tracking Framework

Tracking methods can be used to follow a specific object through an image sequence. The quantities of an object that we want to track are described by an n-dimensional state vector $x \in X^n$. As we process image sequences, the observed scene changes over time. Accordingly, the state vector x is assumed to evolve according to a system model

$$x_t = f_{t-1}(x_{t-1}, w_{t-1}) \tag{1}$$

where f_{t-1} is the *system transition function*. As the system model may not be known exactly, $w_{t-1} \in X^w$ is a noise vector that models our uncertainty. Consequently, with the aid of equation 1, the object state at the next time can be predicted.

At discrete times, measurements $z_t \in X^m$ become available. These observations are related to the state vector by the *measurement model*.

$$z_t = h_t(x_t, v_t) \tag{2}$$

where h_t is the *measurement function*. We model inaccuracy in the measurements with another noise vector $v_t \in X^v$. The set of all observations up to a time t is denoted by $Z_t = \{z_1, ..., z_t\}$.

Generally, we are interested in finding an estimate of the state vector x_t at time t given the previous state vector x_{t-1} and the measurement z_t. The entire tracking process is therefore described by first calculating a prediction of the state vector at the next time step and then using the measurement to update the state estimation.

Recursive Bayesian Filter

We consider a statistical tracking approach known as the recursive Bayesian filter. In this approach an object is not represented by a single object state x_t but by its probability distribution $p(x_t)$. We want to calculate the prediction of the probability for the next time step, which is also called the *a priori* distribution $p(x_t|Z_{t-1})$. Given a new measurement z_t, we can then estimate the *a posteriori distribution* $p(x_t|Z_t)$. Thus, we recursively construct the *a posteriori* probability density of the state conditioned on all the measurements.

Condensation Algorithm

The CONDENSATION algorithm - CONDitional DENSity propagaTION over time - is a sampled-based solution of the recursive Bayesian filter. This tracking method was first proposed in the computer vision community by Michael Isard and Andrew Blake. We provide their paper in the release as blakeIJCV1998.pdf [1].

The underlying idea is to represent the probability distribution by a weighted sample set $S = \{(s^{(n)}, \pi^{(n)} | n = 1, ..., N\}$. Consequently, the distribution of the samples constitutes a discrete approximation of the full probability distribution. Each sample consists of an element s, which represents the state vector, and a corresponding weighting factor, the discrete sampling probability, denoted by π. The size of the sample set is denoted by N.
In analogy with the recursive Bayesian filter, the CONDENSATION algorithm consists in each iteration of first a prediction and second an update step.

Prediction. We start with a sample set S_{t-1} computed in the previous iteration step, which approximates the *a posteriori* density $p(x_{t-1}|Z_{t-1})$. Each sample then evolves by the system model

$$s_t^{\prime(n)} = As_{t-1}^n + Bw_{t-1}^{(n)} \tag{3}$$

where A defines the deterministic and B the stochastic component of the system model. The deterministic part of the equation 3 models the system knowledge, while the stochastic part allows us to model uncertainties. In this case we consider the noise to be normally distributed and hence w_{t-1} is a vector of normal random variables scaled by B so that BB^T is the covariance of the distribution. We have now generated a new sample set S_t' which approximates the *a priori* density $p(x_t|Z_{t-1})$.

Update. The new measurement zt is used to update the prediction by re-weighting each sample in the set S_t' by

$$\pi^{(n)} = p(z_t|s_t^{\prime(n)}) \tag{4}$$

We then obtain a new sample set S_t by sampling from the set S_t' with these updated weights. This new sample set approximates the new *a posteriori* density $p(x_t|Z_t)$. A particular sample $s_t^{\prime(j)}$ is drawn with replacement with probability $\pi^{(j)}$. Some elements may be chosen several times, leading to identical elements in the new sample set S_t. However, these samples will not remain identical during the following prediction steps; they will be separated due to the stochastic component of the system model. Other elements with relatively low weights may not be chosen at all.

To initialize the CONDENSATION algorithm, we start at $t = 0$ with a random sample set S_0 based on a given probability density $p(x_0)$.

CONDENSATION Tracker Based On Color Histograms (50%)

Your first task is to implement a CONDENSATION tracker based on color histograms using python. The objects to be tracked are represented by bounding boxes whose state is given as

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

where x, y represent the location of center of the bounding-box, and $\dot{x}, \dot{y}$ the velocities in the x and y direction.
In this exercise we will focus on tracking just one object and consider just two prediction models (modeled by a matrix A): (i) no motion at all i.e. just noise; and (ii) constant velocity motion model. We also assume that the width and height describing the object bounding box do not change over time.
The initial bounding box containing the object to be tracked will be provided by the user. The tracker should prefer samples (which we also equivalently call particles or bounding boxes) with color histograms (CH) similar to the target model. The similarity is assessed through the χ^2 distance, which is used to assign the weights $\pi^{(n)}$. The probability of each sample is specified by a Gaussian with variance σ.

$$\pi^{(n)} = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{\chi^2(CH_{s^n}, CH_{target})^2}{2\sigma^2}} \tag{6}$$

As already pointed out, during filtering samples with high weights may be chosen several times, leading to identical copies, while others with relatively low weights may not be chosen at all. The steps for one iteration of this particular CONDENSATION tracker are given below:
Input: Sample set S_{t-1} and the target model color histogram CH_{target}
Step 1: Sampling. Draw N samples from the set S_{t-1} with probabilities $\pi_{t-1}^{(n)}$. The new set of samples forms the set S'_{t-1}
Step 2: Prediction. Propagate each sample s'_{t-1} from the set S'_{t-1} by a linear stochastic differential equation:

$$s_t^{(n)} = As'^{(n)}_{t-1} + w_{t-1}^{(n)} \tag{7}$$

where $w_{t-1}^{(n)}$ is the stochastic component. Thus we obtain $S_t = \{s_t^{(n)} | n = 1...N\}$.
Step 3: Update. Observe the color distributions:
(3.a) for each sample $s_t^{(n)}$ compute the color histogram $CH_{s_t^{(n)}}$
(3.b) for each sample $s_t^{(n)}$ compute the χ^2 distance between its color histogram and the target color histogram
(3.c) weight each sample $s_t^{(n)}$ using eq. 6

Step 4: Estimate the mean state of the set S_t

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

After each iteration, the color histogram of the target (CH_{target}) is updated as a convex combination between the old color histogram of the target and the color histogram of the mean state:

$$CH_{target} = (1 - \alpha) \cdot CH_{target} + \alpha \cdot CH_{E[s_t]} \tag{9}$$

We provide the pipeline of the algorithm in *condensation_tracker.py*. Your task is to provide the missing functions marked in *condensation_tracker.py* by comments. Once you have done so, you will be able to run the tracker using the command: *python condensation_tracker.py*

The python code to be completed is now described.

1. Color histrogram (10%)

   ```
   hist = color_histogram(xmin, ymin, xmax, ymax, frame, hist_bin)
   ```

   This function should calculate the normalized histogram of RGB colors occurring within the bounding box defined by (xmin, xmax) (ymin, ymax) within the current video frame. The histogram is obtained by binning each color channel (R,G,B) into hist bin bins. Consequently, we essentially assume only $hist\_bin^3$ possible distinct colors instead of the full 256^3 possible colors in 24-bit color space. You may store the histogram using whichever data structure you prefer, as it is (almost) only used by the functions you will write.

2. Derive matrix A (5%)
   In this exercise we consider two prediction models: (i) no motion at all i.e. just noise; and (ii) constant velocity motion model. For both, derive the dynamic matrix A and explain how you worked them out.

3. Propagation (10%)

   ```
   particles = propagate(particles, frame_height, frame_width, params)
   ```

   This function should propagate the particles given the system prediction model (matrix A) and the system model noise represented by params.model, params.sigma position and params.sigma velocity. Use the parameter frame_height and frame_width to make sure that the center of the particle lies inside the frame.

4. Observation (10%)

   ```
   particles_w = observe(particles, frame, bbox_height, bbox_width,
                         params["hist_bin"], hist, params["sigma_observe"])
   ```

   This function should make observations i.e. compute for all particles its color histogram describing the bounding box defined by the center of the particle and bbox_height and bbox_width. These observations should be used then to update the weights *particles_w* using eq. 6 based on the χ^2 distance between the particle color histogram and the target color histogram given here as hist target. In order to compute the χ^2 distance use the provided function *chi2_cost.py*.

5. Estimation (5%)

   ```
   mean_state = estimate(particles, particles_w)
   ```

   This function should estimate the mean state given the particles and their weights.

6. Resampling (10%)

   ```
   particles, particles_w = resample(particles, particles_w)
   ```

   This function should resample the particles based on their weights (eq. 6.), and return these new particles along with their corresponding weights.

Experiments (50%)

We provide in the release three videos: video1.wmv, video2.wmv, and video3.wmv. Each video shows a moving object in different conditions:

1. *video1.wmv* shows a moving hand with a uniform background.
2. *video2.wmv* shows a moving hand with some clutter and occlusions.
3. *video3.wmv* shows a ball bouncing.

Your second task is to track the objects in these videos using the tracker code you wrote. Experiment with the three videos with various parameters (params), and determine their effect on tracking performance. Think about the different tracking applications in which different parameter settings might be appropriate.

Here are the experiments that you have to perform:

1. *video1.wmv* (10%)
   First experiment with video1.wmv. The hand in this video should be relatively easy to track, allowing you to check for bugs.

2. *video2.wmv* (20%)
   Now experiment with video2.wmv. Vary the parameters and answer the following questions for this video.

   - What is the effect of using a constant velocity motion model?
   - What is the effect of assuming decreased/increased system noise?
   - What is the effect of assuming decreased/increased measurement noise?

3. *video3.wmv* (20%)
   Now experiment with video3.wmv. Use the best parameters for video2.wmv to try to track the ball. Are you able to track the ball? Why or why not? Consider the questions from point (2) again, for video3.wmv. What are the effects of this video?

4. Hand in
   Write a report describing your implementation, the experiments you perform, and the conclusions you draw from them. Also, consider the following questions:

   - What is the effect of using more or fewer particles?
   - What is the effect of using more or fewer bins in the histogram color model?
   - What is the advantage/disadvantage of allowing appearance model updating?

   In your report please provide answers to all of the questions we ask you to consider. For each video, show plots of trajectories of the estimated mean state of the particles during tracking for an example in which tracking is successful and when it is unsuccessful.

   Submit the report with your source code (but not the data) to Moodle before the specified deadline. If you have problems using the WMV files, use the supplied AVI files instead.

Notes

- After the object disappears, your code output is irrelevant.
- There might be many ways to compute the histogram, you can use any method as long as it works with other parts of your code.
- If you have a problem loading the videos, try installing *ffmpeg* library and updating the *opencv* package. You might need the *opencv-contrib-python* package. The code is tested with *opencv-contrib-python-headless==4.6.0.66* on MacOS and *opencv-python==4.5.4.60* on Windows.
- If you have the error *OpenCV — GStreamer: missing plugin*, there might be multiple reasons but you can try the one of following:

  - For conda, use *conda install -c conda-forge gst-plugins-good.*
  - Use *opencv-contrib-python-headless* instead of opencv.

- If you have a problem with RectangleSelector, try changing matplotlib backend with *matplotlib.use('TkAgg')* (in this order TkAgg, wxAgg, Qt5Agg, Qt4Agg) after *import matplotlib.*

Reference

[1] Isard, M., Blake, A. - 'CONDENSATION - conditional density propagation for visual tracking', IJCV 1998.