COMPUTER VISION

LAB SESSION 3: VISUAL OBJECT TRACKING

MASTER IN MACHINE LEARNING FOR HEALTH

24/25

1 A particle filter for visual object tracking

Together with these guidelines the students will find an implementation of a simple particle filtering for object tracking. The filter is based on [Bagdanov et al., 2007], and is roughly described in the following paragraphs.

1.1 The state model and the particle filter

The state of a tracker is defined by the following vector X:

$$X = (x, y, w, h, \dot{x}, \dot{y}, \dot{w}, \dot{h})$$

where $\dot{a}$ stands for the derivative of the variable a with respect to time.

We propose a **particle filter** to approximate the pdf of the state x_t given the observations z_t:

$$p(x_t|z_t) \approx \sum_{i=1}^{N} w_t^i \, \delta(x_t - x_t^i)$$

where we have N particles x_t^i (samples of the distribution) indexed by i and with associated weights/importances w_t^i.

1.2 Importance Sampling and proposal distribution

We can construct a good estimate of $p(x_t|z_t)$ if we sample the particles as follows:

$$x_t^i \sim q(x_t|x_{t-1}^i, z_t)$$

$$w_t^i \propto w_{t-1}^i \frac{p(z_t|x_t^i)p(x_t|x_{t-1})}{q(x_t|x_{t-1}^i, z_t)}$$

where $p(x_t|x_{t-1})$ is the state-update model, $p(z_t|x_t^i)$ is the observation model and $q(x_t|x_{t-1}^i, z_t)$ is the proposal distribution of the particle filter.

In this very simple approach, we choose the **proposal distribution** to be equal to the state transition model (the prior of the state):

$$q(x_t|x_{t-1}^i, z_t) = p(x_t|x_{t-1})$$

Hence,

$$w_t^i \propto w_{t-1}^i \, p(z_t|x_t^i)$$

That can be easily implemented by resampling, leading to a non-iterative equation:

$$w_t^i \propto p(z_t|x_t^i)$$

1.3 State-update model

The **state-transition** is modeled through using a **Gaussian distribution**:

$$p(x_t|x_{t-1}) = \mathcal{N}(Ax_{t-1}; \boldsymbol{v}_t)$$

Which implies that the **state is updated** using a **linear model**:

$$x_t = Ax_{t-1} + v_t$$

where A is the state-transition matrix of the form. We will use an A matrix of an object that moves with **constant velocity**:

$$A = \begin{bmatrix} I_4 & I_4\Delta_t \\ 0_4 & I_4 \end{bmatrix}$$

and $I_4$is the 4x4 eye-matrix and 0_4 is the 4x4 empty matrix. We can easily see that the dynamic elements of the state (velocities in location and bounding box dimensions) are constant, whereas the static elements (location, width and height) change linearly with time.

The vector v_t introduces some noise and provides diversity during the estimation of the pdf of the state:

$$v_t = (\sigma_x, \sigma_y, \sigma_w, \sigma_h, \sigma_{\dot{x}}, \sigma_{\dot{y}}, \sigma_{\dot{w}}, \sigma_{\dot{h}})$$

1.4 Observation model

The **observation model** is **nonlinear:** we measure the observations $p(z_t|x_t^i)$ using the Color histogram Similarity (Bhattacharyya coefficient) between the object model $\boldsymbol{o_t}$ and the histograms of the particle proposals $h_t^i = h(x_t^i)$:

$$p(z_t|x_t^i) \sim BC(o_t, h\{x_t^i\}) = \int_{c \in C} \sqrt{o_t(c) h_t^i(c)}$$

where $c \in C$ are the colors in the 2D HS space (from HSV, we remove V as it is usually less representative and stable).

Particle weights are proportional to the observation model $w_t^i \propto p(z_t|x_t^i)$, and later normalized so that $\sum_{i=1}^{N} w_t^i = 1$.

1.5 Resampling

Resampling is implemented using the roulette method over the cumulative weights of the particles $c_t^i = \sum_{j \le i} w_t^j$. Details of the resampling method can be found in the lecture notes of the course.

In addition, the pseudocode of the algorithm can be found in the lecture notes and in the original paper [Bagdanov et al., 2007].

2 Implementation of the Particle Filter

2.1 Functions

The guidelines provide an implementation of the particle filter that we have described in the previous section. In particular, the students are provided with the following functions:

- **"object_tracking.py"**: main python script that implements the tracking of a visual object in a video. It initializes the tracker using the ground truth information in the first

frame, and then reads the video frame by frame and update the filter accordingly. It also computes an evaluation metric (Jaccard Index) in a per-frame basis and then averages the frame values to produce a video-level score. In addition, if verbose=true, it shows a visualization of the tracker.

- **"particle_filter.py"**: Class that implements the particle filter described in [Bagdanov et al., 2007]. The constructor method *__init__()* initializes the particle filter using the ground truth bounding box surrounding the object in the first frame, whereas the method *update()* updates the particle filter for each new frame in the video. This file also defines an auxiliary function computeHSVHistograms(), which computes a 2D histogram of a bounding box using the H and S channels of the HSV color space.
- **"metrics.py"**: defines a function *computeJI()* which computes the Jaccard index between two bounding boxes (the ground truth and the predicted). The Jaccard Index measures the ratio between the intersection and union areas formed by the two bounding boxes. Its maximum is 1, which means that the two bounding boxes are equal, and the minimum is 0, which means that they do not overlap.
- **"visualization.py"**: includes two auxiliary functions:
    - showBB(): function that shows bounding boxes overlaid on an image. Useful to visualize the results of the tracker.
    - showParticles(): function that shows particles overlaid on an image. Please, note that only x,y coordinates are considered, so that particles are represented as points.
- "**config.py**": file with the configuration of the particle filter. Currently, the values for the following parameters can be set:
    - ***K***: number of bins in each dimension of the HS histogram. The 2D histogram will finally have K^2 elements, covering all the combinations of values in both dimensions.
    - ***std_noise:*** noise standard deviation to be applied to state vector.
    - ***alpha:*** exponential factor to increase the sharpness of the weights distribution. Hence, weights w_t^i computed using the formula defined in section 1.2 will be then raised to alpha exponent and renormalized to sum to 1.
    - **prediction**: method to generate the final prediction of the tracker: two options are currently accepted: '*weighted_avg*', which computes the weighted combination of particle states; and *'max',* which sets the state to that of the particle with the maximum weight (the best particle).

In addition, for the sake of evaluation, we also include a file "**evaluateSystem.py**" that allows to evaluate a particle filter over the set of training videos.

IMPORTANT: Due to the stochastic nature of the particle filter, each execution of *object_tracking.py* provides a different realization of the filter and therefore may obtain different performance. In order to establish a fair and robust evaluation, 'evaluateSystem' function repeats each video several times (4 by default, although it can be modified) and averages the results.

2.2 Training Videos

To allow students to train or validate their system on a set of videos, we provide a reduced training set containing 4 videos:

1. Basketball
2. Biker
3. Bolt
4. Skating

For each video, we provide the frames in JPG format enumerated with 4 digits (e.g. "0000.jpg") and a file "nameVideo.mat" with the annotations in the form.

$$g_t = \left(x_g^t, y_g^t, w_g^t, h_g^t\right)$$

3 Evaluation of Lab session (IMPORTANT)

The goal of this lab session is that the students experiment with the baseline code and generate their own version of the particle filter by:

- Validating and optimizing some of the parameters of the filter.
- Extending or modifying the filter changing some of the processing blocks (e.g. the observation model, resampling process, etc.).

IMPORTANT: In order to provide a fair comparison between submissions, the submitted code will be run with N=300 particles, parameter passed as an argument for the constructor. I will check this point in the code and fix the number of particles to 300 if necessary.

IMPORTANT2: In order to provide a fair comparison, the maximum execution time of the tracking process must be 0.6 seconds per frame (with N=300), otherwise the JI will be set to 0. I understand that execution times may differ from one computer to another. To provide an estimation, please consider that the current code takes between around 0.17 seconds per frame using the laptop I will employ to evaluate. Hence you can roughly increase the execution time by a factor of 3x.

The students are asked to upload a zip file to Aula Global containing the following information:

- Their own implementations of particle_filter.py and config.py (**remember that the number of particles will be set as an argument, with N=300 particles in the evaluation**), and any other auxiliary functions/files that may be required by their code. NOTE that the submitted code MUST be compatible with the provided file object_tracking.py.

- A brief (hard limit of 1 page) summary of the optimization, design decisions or extensions/modifications over the baseline. The students shall include a table showing the performance of tested approaches.

I will evaluate the performance of the submissions over a test set with some videos of similar properties of those ones used during training. The final grade with by computed considering both the results of the challenge and the content of the report.

IMPORTANT: The deadline for the submission is March 17 at 23:59. Submissions will be done in pairs, following the organization of the course. Please, indicate the name of the students in the filename.

4 References

[Bagdanov et al., 2007] Bagdanov, A.D.; Del Bimbo, A.; Dini, Fabrizio; Nunziati, W., "Improving the robustness of particle filter-based visual trackers using online parameter adaptation," Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on , vol., no., pp.218,223, 5-7 Sept. 2007