Exercise 4: Advanced tracking

Advanced Computer Vision Methods

2024/2025

This exercise will address some more complex tracking approaches. You will implement a tracker that uses a particle filter sequential optimization and a Kalman filter motion prediction.

Submission instructions

The exercise should be submitted on-line on the course website. The submission should contain a report and the source code. Do not submit the data which was given as a part of the instructions, except you add some data that you collect from other sources. The report is the most important part of the submission, so make sure that you spend enough time on it, after you are done with coding and experiments. Note that a strict page limit of the report is **two pages maximum**. Detailed description of grading can be found at the end of the document.

The submissions should be done by the deadline. Late submissions are possible, however, a strict deadline is one week after the first one. A baseline for the late submission is 70%. After that you cannot submit the exercise anymore. The assistant will review the submissions and provide a feedback within a week after the deadline (unless stated otherwise). Each exercise must be done individually and all submissions will be checked for plagiarism. A student will be notified about the grade in the submission feedback. Passing all five exercises is required to pass this part of the course.

Assignment 1: Motion models and Kalman filter

Similarly to the previous exercises you will first do some experiments outside the tracking scenarios where it is much easier to analyze the code. For this assignment you have to implement the Random Walk (RW), Nearly-Constant Velocity (NCV), and Nearly-Constant Acceleration (NCA) models using Kalman filter. Test the implementations using some artificially generated trajectories and experiment with Kalman filter parameters. Use the following code snippet as an example:

```
N = 40
v = np.linspace(5 * math.pi, 0, N)
x = np.cos(v) * v
y = np.sin(v) * v
```

Number `N` determines the number of points (time-steps) in the trajectory. Apply the Kalman filter for different motion models to the trajectory by using the following snippet:

```
sx = np.zeros((x.size, 1), dtype=np.float32).flatten()
sy = np.zeros((y.size, 1), dtype=np.float32).flatten()

sx[0] = x[0]
sy[0] = y[0]

state = np.zeros((A.shape[0], 1), dtype=np.float32).flatten()
state[0] = x[0]
state[1] = y[0]
covariance = np.eye(A.shape[0], dtype=np.float32)

for j in range(1, x.size):
    state, covariance, _, _ = kalman_step(A, C, Q_i, R_i, np.reshape(np.array([x[j], y[↩
        j]]), (-1, 1)), np.reshape(state, (-1, 1)), covariance)

    sx[j] = state[0]
    sy[j] = state[1]
```

In the sample above a `kalman_step` function is used. The function is available in the exercise material so that you do not have to implement Kalman filter yourself. You just have to correctly determine the parameters: the system matrix `A` (also known as Φ from lectures [1]), the observation matrix `C` (also known as $\mathbf{H}$ from lectures [1]), the system noise covariance matrix `Q`, and the observation noise covariance matrix `R`. These matrices define the following equations that you know from the lectures

$$\mathbf{X}_k = A\mathbf{X}_{k-1} + W_k, \qquad W_k \sim \mathcal{N}(0, Q), \tag{1}$$

$$\mathbf{Y}_k = C\mathbf{X}_k + V_k, \qquad V_k \sim \mathcal{N}(0, R), \tag{2}$$

For the report also apply Kalman filter smoothing to some other trajectories. Try creating a rectangle or some more jagged path. Visualize the results and include the plots in the report. Describe your findings about the properties of different motion models.

Assignment 2: Particle filters

Tracking with a particle filter can be viewed at as a fusion of cross-entropy stochastic optimization and motion models. The basic idea is to transfer a set of particles (hypotheses) from one frame to another and not collapse them to a single state prediction. In a new frame these particles are propagated using a motion model, evaluated for similarity using a visual model and re-sampled based on their similarity weights. This way multiple similarly important states can be maintained for a shorter periods of time which helps to resolve some ambiguities.

In this assignment you will implement a particle filter tracker using NCV motion model. You may choose any visual model that we have implemented so far (e.g. color similarity, NCC), or use some more complex visual model. The overview of the tracking algorithm looks like this:

- **Initialize**:
    - (a) Construct a visual model of an object.
    - (b) Generate n particles at the initial position (equal weights). At this point you can simply generate samples from a Gaussian distribution around the initial position.

- **Update at frame t:**
  (a) Replace existing particles by sampling n new particles based on weight distribution of the old particles.
  (b) Move each particle using the dynamic model (also apply noise).
  (c) Update weights of particles based on visual model similarity.
  (d) Compute new state of the object as a weighted sum of particle states. Use the normalized particle weights as weights in the sum.

Sampling new particles using weight distribution can be implemented in Python in few lines:

```
particles = ... # vector of N particles (rows) and d dimensions (columns)
weights = ... # vector of weights for N particles
# resample particles
weights_norm = weights / np.sum(weights)  # normalize weights
weights_cumsumed = np.cumsum(weights_norm)  # cumulative distribution
rand_samples = np.random.rand(N, 1)
sampled_idxs = np.digitize(rand_samples, weights_cumsumed)  # randomly select N indices
particles_new = particles[sampled_idxs.flatten(), :]  # select the corresponding ↩
    samples
```

Particle filters have to be implemented correctly in order to work so it is important not to give up too soon if the first implementation does not work. Pay attention to how weights are estimated as described in [2]. For more details also consult the literature [4]. Also, note that the success of a tracker depends greatly on the type of visual model. Two successful particle filter tracker implementations are described in [3, 5].

Grading

The tasks marked with *Req.* are required to successfully complete the exercise. The number in the brackets represents number of points of other tasks while *Add.* stands for additional tasks which can bring you more than 100 points.

- (Req.) Run Kalman filtering method on a spiral curve using RW, NCV and NCA motion models and different values of parameters q and r (see exercise slides for reference). Include a similar image as that on the slides. Include also the matrices $\mathbf{x}$ (state), $\mathbf{F}$, $\mathbf{\Phi}$, $\mathbf{L}$, $\mathbf{Q}$ and $\mathbf{H}$ for all three motion models to the report (due to the page limit, you can include these on the third page as an Appendix).

- (Req.) Implement particle filter tracker using color histogram as a visual model and NCV as motion model. Run the tracker on at least five sequences from VOT and report the performance in terms of accuracy and robustness.
  For the reference: our basic implementation of a particle filter achieves average overlap 0.52 and 43 failures in total on the VOT2014 dataset. Average speed is approximately 65FPS on a standard laptop.

- (20) Run Kalman filtering on two other curves, include graphs in the report and discuss the results.

- (15) Compare particle filter tracker using different motion models (RW, NCV and NCA) in terms of accuracy and robustness. Explain how you set parameter q and show how this parameter impacts tracking performance?
- (15) What is the impact of the number of particles on a tracking performance? How accuracy, robustness and tracking speed change with different number of particles?
- (Add.) Try different colorspaces to create color histogram (RGB, HSV, Lab, YCrCb) and compare tracking performance.
- (Add.) Use different visual models in particle filter tracker (SSD, NCC, correlation filter, etc.) and compare them in terms of accuracy, robustness and speed.

References

[1] M. Kristan. Kalman filter lecture slides.

[2] M. Kristan. Particle filter lecture slides.

[3] K. Nummiaro, E. Koller-Meier, and L. Van Gool. An adaptive color-based particle filter. *Image and Vision Computing*, 21(1):99–110, 2002.

[4] Simon JD Prince and Simon Jeremy Damion Prince. *Computer vision: models, learning, and inference.* Cambridge University Press, 2012.

[5] P. Pérez, C. Hue, J. Vermaak, and M. Gangnet. Color-based probabilistic tracking. 2002.