Problem Set 7: Particle Filter Tracking

Description

In this problem set you are going to experiment with a particle filter tracker such as was described in class.

Recall that we need a variety of elements:

(1) a *model* - this is the "thing" that is actually being tracked. Maybe it's a patch, a contour, or some other description of an entity;
(2) a representation of *state* x_t that describes the state of the model at time t;
(3) a *dynamics model* $p(x_t \mid x_{t-1})$ that describes the distribution of the state at time t given the state at $t - 1$;
(4) a *measurement* z_t that some how captures the data of the current image; and finally,
(5) a *sensor model* $p(z_t \mid x_t)$ that gives the likelihood of a measurement given the state. For Bayesian-based tracking, we assume that at time $t - 1$ we have a belief about the state represented as a *density* $p(x_{t-1})$, and that given some measurements z_t at time t we update our *Belief* by computing the posterior density:

$$Bel(x_t) \propto p(z_t \mid x_t)p(x_t \mid u_t, x_{t-1})Bel(x_{t-1})$$

The Kalman filter provided an analytic method for doing this under the assumption that density representing the belief at time t, the noise component of the dynamics and the sensor model were all simple Gaussian distributions. Particle filters provide a sample-based method of representing densities that removes that restriction and also is tolerant of occasional large deviations from the well-behaved model. In this assignment, you will be implementing a particle filter to track entities in video.

Three video (.avi) files are provided in the input directory:

i. `pres_debate.avi`, which shows two candidates in a 2012 town hall debate,
ii. `noisy_debate.avi`, which is the same video overlaid with fluctuating Gaussian noise, and,
iii. `pedestrians.avi`, which shows a group of people crossing a street in London.

For each video file, there is a text file that contains the object bounding box at the first frame that you can use for initialization. The format of the text file is:

```
x y
w h
```

where (x, y) is the top-left coordinate (not center), and (w, h) is the size (width, height) of the bounding box.

What to submit

Download and unzip the `ps7` template: ps7.zip

`ps7/`

- input/ - input videos and data - **REMOVE BEFORE SUBMITTING!**
- output/ - directory containing output images and other files your code generates
- ps7.py - code for completing each part - classes, methods, functions, driver code
- *.py - supporting Python modules, any utility code
- ps7_report.pdf - a PDF file with all output images and text responses

Zip it as `ps7.zip`, leaving out the input folder, and submit on T-Square.

E.g. on Linux or Mac OS X (`-ru` = recursive, update changed files; `-x` = exclude "<path>"):

```
zip -ru ps7.zip ps7 -x "ps7/input/*"
```

***** Please do not submit the input folder for this assignment, remove it when creating your zip file. *****

Guidelines

1. Include all the required images in the report to avoid penalty.
2. Include all the textual responses, outputs and data structure values (if asked) in the report.
3. Make sure you submit the correct (and working) version of the code.
4. Include your name and GTID on the report.
5. Even if the code is not working, submit the code as the instructors can read through the algorithms to give partial credit. Comment your code appropriately.

Questions

1. Particle Filter Tracking

In class we discussed both the theory of and the implementation specifics behind particle filtering. The algorithm sketch is provided in the notes and it describes a single update of the particle filter. What is left up to the implementor are several details including what is the model, what is the state, the number of particles, the dynamics/control function $p(x_t \mid x_{t-1}, u_t)$, the measurement function $p(z_t \mid x_t)$, and the initial distribution of particles.

For this question, you will to need track an image patch template (a face) taken from the first frame of the video. For this assignment the model is simply going to be the image patch, and the state will be only the 2D center location of the patch. Thus each particle will be a (u,v) pixel location representing a proposed location for the center of the template window. We will be using a basic function to estimate the dissimilarity between the image patch and a window of equal size in the current image, the Mean Squared Error:

$$MSE(u_p, v_p) = \frac{1}{mn}\sum_{u=1}^{m}\sum_{v=1}^{n}(Template(u,v) - Image^t(u + u_p - m/2, v + v_p - n/2))^2$$

The funny indexing is just because (u, v) are indexed from 1 to M (or N) but the state $\langle u_p , v_p \rangle$ is the location of the center.

These images are color and you can do the tracking in either the full color image, or in grayscale, or even in the green channel (which is a cheap way to get luminance). If you use color, there would be an outer summation over the three color channels. If you want to use grayscale, you can use `cvtColor` in OpenCV or just use a weighted sum of R,G,B to get a luminance value (try weights of 0.3, 0.58, and 0.12).

MSE, of course, only indicates how dissimilar the image patch is whereas we need a similarity measure so that more similar patches are more likely. Thus, we will use a squared exponential equation (Gaussian) to convert this into a usable measurement function:

$$p(z_t|x_t) \propto \exp(-\frac{MSE}{2\sigma^2_{MSE}})$$

To start out, you might use a value of $\sigma_{MSE} = 10$ (for grayscale in 0-255 range) but you might need to change this value.

For the dynamics we're going to use normally distributed noise since head movements can be unpredictable and often current velocity isn't indicative of future motion; so our dynamics model is merely

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \delta_t, \text{ where } \delta_t \sim N(0, \Sigma_d), \Sigma_p = \begin{bmatrix} \sigma_d^2 & 0 \\ 0 & \sigma_d^2 \end{bmatrix}$$

which is just a fancy way to say you add a little bit of Gaussian noise to both u and v independently.

The number of particles and initial distribution are up to you to figure out. You may need to tune your parameters for these to get better tracking/performance.

In order to visualize the tracker's behavior you will need to overlay each successive frame with the following elements:

- Every particle's (u, v) location in the distribution should be plotted by drawing a colored dot point on the image. Remember that this should be the center of the window, not the corner.
- Draw the rectangle of the tracking window associated with the Bayesian estimate for the current location which is simply the *weighted* mean of the (u, v) of the particles.
- Finally we need to get some sense of the standard deviation or spread of the distribution. First, find the distance of every particle to the weighted mean. Next, take the weighted sum of these distances and plot a circle centered at the weighted mean with this radius.

You will have to produce these visuals for select frames but you will not have to submit the entire video. For reading the frames OpenCV users should look at the class `VideoCapture`. For sampling look at the function `numpy.random.multinomial`.

Note: these frames are 1280 by 720 in size. If you want to shrink them to, say, 640 by 360 that's OK, but just realize you'll need to adjust the initial bounding boxes we gave you, and render output to original frame size.

a. Implement the particle filter and run it on the `pres_debate.avi` clip. You should begin by attempting to track Romney's face. You can find the bounding box for his face for the first frame in `pres_debate.txt` - this is your template. Tweak the parameters including window size until you can get the tracker to follow his face faithfully (5-15 pixels) up until he turns his face significantly. Run the tracker and save the video frames 28, 84, and 144 with the visualizations overlaid.

   Classes, methods: Define a class **`ParticleFilter`**, with the following methods:
   - `__init__(frame, template, **kwargs)`: (constructor) Initialize particle filter object.
   - `process(frame)`: Process a frame (image) of video and update filter state.
   - `render(frame_out)`: Visualize current particle filter state.

   Two helper functions have been supplied: `get_template_rect()` - to read rectangular template bounds from a file, and `run_particle_filter()` - to run your particle filter on a video and collect output.

   Note: The helper function `run_particle_filter()` can save the template image patch and outputs generated by your `render()` method for desired frames. See template code for details.

   Output:
   - the template image patch used for tracking, as `ps7-1-a-1.png`
   - the image frame number 28 with overlaid visualizations, as `ps7-1-a-2.png`
   - the image frame number 84 with overlaid visualizations, as `ps7-1-a-3.png`
   - the image frame number 144 with overlaid visualizations, as `ps7-1-a-4.png`

b. Experiment with different dimensions for the window image patch you are trying to track. Decrease the window size until the performance of the tracker degrades significantly. Try significantly larger windows than what worked in `1-a`. Discuss the trade-offs of window size and what makes some image patches work better than others for tracking.

   Output (`Textual Response in Report`):
   - Describe 2-3 advantages of larger window size and 2-3 advantages of smaller window size

c. Adjust the σ_{MSE} parameter to higher and lower values and run the tracker.

   Output (`Textual Response in Report`):

- Discuss how changing σ_{MSE} parameter alters the results and attempt to explain why.

d. Try and optimize the number of particles needed to track the target.

Output (Textual Response in Report):
- Optimized particle number
- Discuss the trade-offs of using a larger number of particles to represent the distribution.

e. Run your tracker on `noisy_debate.avi` and see what happens. Tune your parameters so that the cluster is able to latch back onto his face after the noise disappears. Include varying σ_{MSE}. Report how the particles respond to increasing and decreasing noise. Save the video frames 14, 32, and 46 with the visualizations overlaid.

Output:
- the image frame number 14 with overlaid visualizations as `ps7-1-e-1.png`
- the image frame number 32 with overlaid visualizations as `ps7-1-e-2.png`
- the image frame number 46 with overlaid visualizations as `ps7-1-e-3.png`

Output (Textual Response in Report):
- Discuss how you managed to tune the parameters

2. Appearance Model Update

Let's say we'd now like to track Romney's left hand (the one not holding the mic). You might find that it's difficult to keep up using the naive tracker you wrote in question 1. The issue is that while making rapid hand gestures, the appearance of the hand significantly changes as it rotates and changes perspective. However, if we make the assumption that the appearance changes smoothly over time, we can update our appearance model over time.

Modify your existing tracker to include a step which uses the history to update the tracking window model. We can accomplish this using what's called an Infinite Impulse Response (IIR) filter. The concept is simple: we first find the best tracking window for the current particle distribution as displayed in the visualizations. Then we just update the current window model to be a weighted sum of the last model and the current best estimate.

$$Template(t) = \alpha\, Best(t) + (1 - \alpha) Template(t - 1)$$

where *`Best(t)`* is the patch of the best estimate or mean estimate. It's easy to see that by recursively updating this sum, the window implements an exponentially decaying weighted sum of (all) the past windows. For this assignment, you may assume `t` to be the frame number, in lieu of a time measure.

a. Implement the appearance model update feature. Run the tracker on `pres_debate.avi` and adjust parameters until you can track Romney's hand up to frame `140`. Run the tracker and save the video frames `15`, `50`, and `140` with the visualizations overlaid.

Classes, methods: Derive a class `AppearanceModelPF` from `ParticleFilter`, retaining the same interface - i.e. with methods `process()` and `render()`, as defined above. The constructor and `process()` method should transparently handle model updates. You can supply additional keyword arguments to the constructor, if needed.

Output:
- the image patch used for tracking as `ps7-2-a-1.png`
- the video frame number 15 with overlaid visualizations as `ps7-2-a-2.png`
- the video frame number 50 with overlaid visualizations as `ps7-2-a-3.png`
- the video frame number 140 with overlaid visualizations as `ps7-2-a-4.png`

b. Try running the tracker on `noisy_debate.avi`. Adjust the parameters until you are able to track the hand all the way to frame 140. Indicate what parameters you had to change to get this to work on the noisy video and discuss why this would be the case.

Output:
- the image patch used for tracking as `ps7-2-b-1.png`
- the video frame number 15 with overlaid visualizations as `ps7-2-b-2.png`
- the video frame number 50 with overlaid visualizations as `ps7-2-b-3.png`
- the video frame number 140 with overlaid visualizations as `ps7-2-b-4.png`

Output (`Textual Response in Report`):
- Discuss how you managed to tune the parameters. Indicate what parameters you had to change to get this to work on the noisy video and discuss why this would be the case.

EXTRA CREDIT QUESTIONS (3 AND 4)

3. Mean-Shift Lite

For this question you're going to try and use a different model, namely the color distribution of the patch. If the color space is RGB, and the values range from 0 to 255, you'll create a histogram by forming bins in R,G,B. For example, you might have 8 bins for each color dimension. Then for each pixel in the window you add a count to the appropriate R,G,B bins. This gives you three 8 bin histograms. This will be your model of the patch.

Now you need a way to compare the histograms of your initial patch with possible targets to yield

likelihoods for particle filter tracking. One possibility is called *Chi-Squared* $\chi 2$. To do this you should probably normalize the histograms to sum to 1.0. Then the comparison between histograms is:

$$C(p_i, q_j) = \frac{1}{2}\sum_{k=1}^{K}\frac{[h_i(k) - h_j(k)]^2}{h_i(k) + h_j(k)}$$

where there are K bins in each histogram. The sum should only be computed for bins in which at least one of the histograms bins is non zero. This metric is zero when they are identical — it's a distance. Another distance between histograms is called the Earth Mover's Distance which is a measure of the distance between two probability distributions. There is code for this online for OpenCV. As with MSE, you'll need to experiment with using either of these to generate a likelihood.

a. Use this model for running the particle filter on Romney's face like you did for question 1. That is, instead of using the MSE value as a measure of likelihood, use the difference between the histogram computed for the initial patch with the target patches. You may need to adjust the size of the initial patch. Run the tracker and save the video frames 28, 84, and 144 with the visualizations overlaid.

   Output:
   - the image patch used for tracking as `ps7-3-a-1.png`
   - the video frame number 28 with overlaid visualizations as `ps7-3-a-2.png`
   - the video frame number 84 with overlaid visualizations as `ps7-3-a-3.png`
   - the video frame number 144 with overlaid visualizations as `ps7-3-a-4.png`

b. Now try to track Romney's hand using the color histogram comparison method. Again, run the tracker and save the video frames 15, 50, and 140 with the visualizations overlaid.

   Output:
   - the image patch used for tracking as `ps7-3-b-1.png`
   - the video frame number 15 with overlaid visualizations as `ps7-3-b-2.png`
   - the video frame number 50 with overlaid visualizations as `ps7-3-b-3.png`
   - the video frame number 140 with overlaid visualizations as `ps7-3-b-4.png`

4. Incorporating More Dynamics

For this question we will work with a much more difficult video to perform tracking with, `pedestrians.avi`. For this problem, we'd like to be able to track the blond-haired woman as she crosses the road. If you try applying your adaptive tracker to this video, you will probably find that you will have difficulty dealing simultaneously with occlusion and the perspective shift as the woman walks away from the camera. Thus, we need some way of relying less on updating our appearance model from previous frames and more on a sophisticated model of the dynamics of the figure we want to track.
For this problem, expand your appearance model to include window size as another parameter. This will

change the representation of your particles.

a. Run the tracker and save the video frames 40, 100, and 240 with the visualizations overlaid. You will receive partial credit if you can show the tracking size estimate (illustrate this with the rectangle outline) up to frame 100. You will receive full credit if you can reliably track all the way to the end of the street and deal gracefully with the occlusions (reasonable tracking at frame 240).

   Output:
   - the image patch used for tracking as `ps7-4-a-1.png`
   - the video frame number 40 with overlaid visualizations as `ps7-4-a-2.png`
   - the video frame number 100 with overlaid visualizations as `ps7-4-a-3.png`
   - the video frame number 240 with overlaid visualizations as `ps7-4-a-4.png`

b. Try and optimize the number of particles needed to track the model in this video. Compare that to the number you found in problem `1-d`. Why is this number different?

   Output (`Textual Response in Report`):
   - Optimized particle number
   - Discuss why the number is different from the one found in `1-d`