Condensation Tracker

Orion Junkins

December 8, 2023

1 Introduction

This Lab presents a basic Python implementation for a CONDENSATION tracker [1]. Given a user-defined target region of interest (ROI) to track, this system

1. Samples a set of candidate points from the image according to a set of weights.
2. Predicts a propagated version of each candidate using a motion model.
3. Compares the region surrounding each candidate to the target, computing the weight of each candidate based on the degree of similarity.
4. Estimates the new state as the weighted mean of all candidates.

Objects are tracked across multiple frames by iterating through these steps.

2 Implementation Details

The implementation is decomposed into several functions defined as follows.

2.1 Color Histogram

The *color_histogram()* function takes in an image frame, the constraints of a bounding box within that frame, and a *hist_bin* number which specifies the degree of quantization for each channel. The ROI is isolated from the frame, and each color channel is binned into *hist_bin* bins. In python, this is implemented using numpy's *histogramdd()* function. This yields a numpy array of shape *(hist_bin, hist_bin, hist_bin)*, effectively reducing the color space to *hist_bin*^{3} distinct colors.

2.2 Derivation of Matrix A

The program user can specify the use of a constant velocity motion model. If no motion model is used, particle state s is specified by two values $s = \{x, y\}^T$. If the motion model is used, particle state s is specified by four values $s = \{x, y, \dot{x}, \dot{y}\}^T$ where $\dot{x}, \dot{y}$ represent the velocity in the x, y directions respectively. For either case, a matrix A is applied during propagation, modeling the expected change from one frame to the next. For the case with no motion, there is no expected change, so A is simply the 2x2 Identity Matrix:

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

In the case that a velocity model is in use, the Matrix A is defined as follows:

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

2.3 Propagation

The *propogate()* function takes a set of particles sampled from the previous timestep, S'_{t-1}, and calculates a new set of samples, S_t. It multiplies every sample by one of the A matrices specified above and adds noise. Specifically, it finds

$$S_t = \{s_t^{(n)} | n = 1...N\}$$

where

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

Here, $w_{t-1}^{(n)}$ represents a vector of random Gaussian noise with the same dimension as the state vector (either two values or four values). The noise is sampled from a Gaussian parameterized by a σ of *sigma_position* for the x, y components of the particle state, and a σ of *sigma_velocity* for the $\dot{x}, \dot{y}$ components of the particle state if a velocity model is in use. In Python, S_t is calculated using numpy's *matmul()* function and the Gaussian noise is found using numpy's *random.normal()* function.

2.4 Observation

The *observe()* function compares the ROIs surrounding each propagated particle to the target ROI and updates the weights according to the similarity. The χ^2 distance is calculated between every particle's color histogram and the target color histogram. This distance determines the overall weight for that particle according to the following formula:

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

where

$\pi^{(n)}$ is the weight of particle n.
σ is the *sigma_observe* value specified by the user.
$CH_s^{(n)}$ is the color histogram for sample n.
CH_{target} is the color histogram of the target.

In Python, weights are calculated one at a time using the provided *chi2_cost()*.

2.5 Estimation

The *estimate()* function calculates the mean state of all particles. Each particle is multiplied by its weight, and the corresponding products are summed across all particles.

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

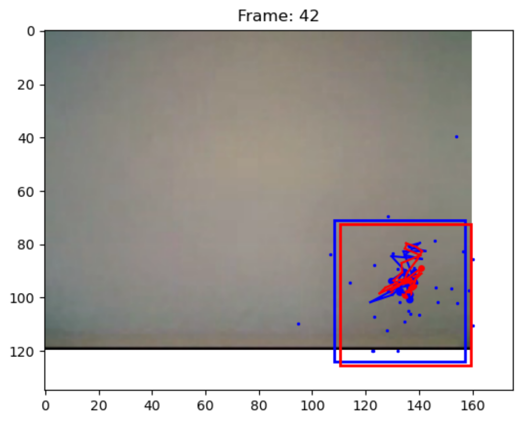
In Python, this is trivial to compute given particle and weight vectors using numpy's *sum()* function.

2.6 Resampling

The *resample()* function produces a new set of samples for the next iteration. Specifically, it produces S_{t+1} by random sampling with replacement from S_t where the probability of a particular sample is $\pi^{(n)}$. The selected samples, along with their weights, are returned. This new set is the starting point for the next iteration of the algorithm.

In Python, this is performed using numpy's *random.choice()* function. *random.choice()* generates a list of indices specifying which elements have been chosen. These corresponding elements are extracted from the particles and weights vectors, returning two new vectors.

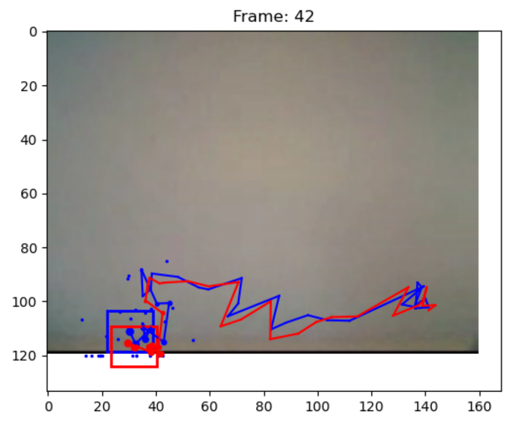
These new vectors have the same dimensions as the original and are likely to contain duplicate particles.



(a) Unsuccessful Tracking Result

Parameter	Value
hist_bins	16
alpha	0.1
sigma_observe	0.1
model	0
num_particles	30
sigma_position	15
sigma_velocity	1
initial_velocity	(1, 10)

(b) Parameters



(c) Successful Tracking Result

Parameter	Value
hist_bins	16
alpha	0.1
sigma_observe	0.1
model	0
num_particles	30
sigma_position	15
sigma_velocity	1
initial_velocity	(1, 10)

(d) Parameters

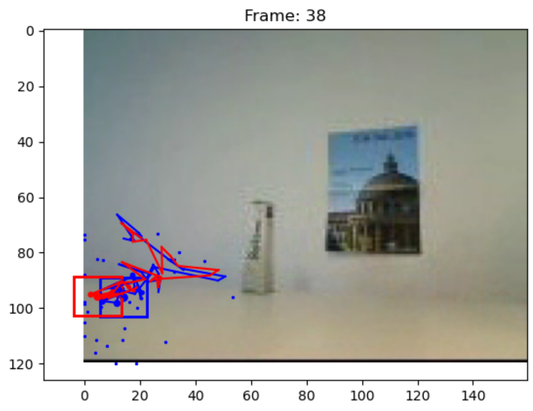
Figure 1: Video 1 Experiments

3 Experiments

The general experimental process is outlined below. Section 4 below contains specific answers to the assignment questions.

3.1 Video 1

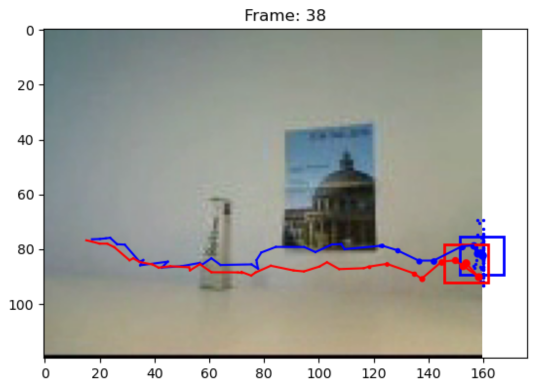
Default parameters are generally effective for *video1*. However, these results occasionally fail, particularly if the initial bounding box is poorly chosen. Figure 1 shows a successful run and a failure where an excessively large bounding box is chosen.



(a) Unsuccessful Tracking Result

Parameter	Value
hist_bins	16
alpha	0.1
sigma_observe	0.1
model	0
num_particles	30
sigma_position	15
sigma_velocity	1
initial_velocity	(1, 10)

(b) Parameters



(c) Successful Tracking Result

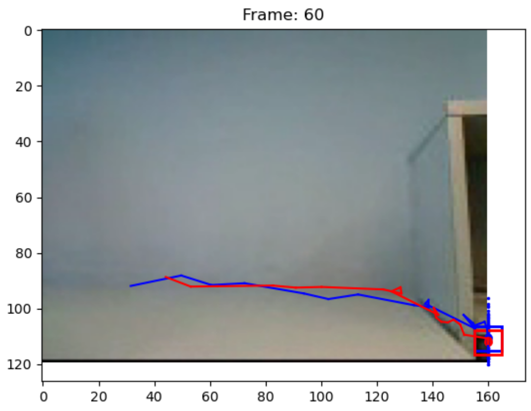
Parameter	Value
hist_bins	16
alpha	0.1
sigma_observe	0.1
model	1
num_particles	100
sigma_position	5
sigma_velocity	1
initial_velocity	(5, 0)

(d) Parameters

Figure 2: Video 2 Experiments

3.2 Video 2

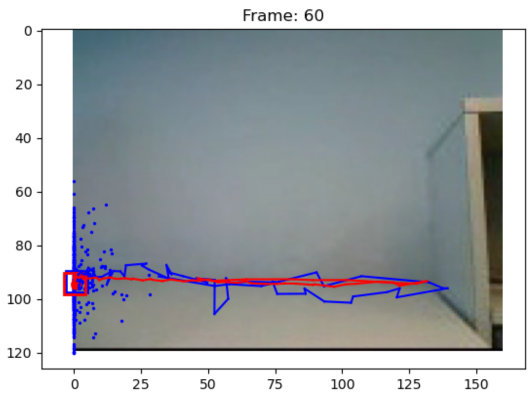
Default parameters occasionally succeed in *video2*. However, failures are frequent. The tracking fails at either the moment of occlusion or at the moment the background changes substantially (when the hand passes over the poster). Values are tuned to address these issues. Figure 2 shows both of these experiments with the tuned parameters. The *alpha* and *sigma_observe* parameters are also adjusted to improve performance.



(a) Unsuccessful Tracking Result

Parameter	**Value**
hist_bins	16
alpha	0.1
sigma_observe	0.1
model	1
num_particles	100
sigma_position	5
sigma_velocity	1
initial_velocity	(5, 0)

(b) Parameters



(c) Successful Tracking Result

Parameter	**Value**
hist_bins	16
alpha	0.05
sigma_observe	0.2
model	1
num_particles	500
sigma_position	10
sigma_velocity	3
initial_velocity	(5, 0)

(d) Parameters

Figure 3: Video 3 Experiments

3.3 Video 3

The best parameters from *video2* were ineffective for *video3*. The tracking fails at the point of rebound. This is addressed by increasing sigma_velocity, sigma_postion, and num_particles. Figure 3 shows both of these experiments.

4 Conclusions

4.1 Effect of Parameters on *video2*

What is the effect of using a constant velocity motion model? The constant velocity motion model causes particles to tend to move in the direction of expected motion during propagation. This allows the tracking of rapidly moving objects as the object is less likely to escape the particle cloud. This also allows the handling of temporary occlusions, as the particles will continue moving in the correct direction at the correct velocity and pick the object up on the other side.

Specifically for *video2*, a motion model helps overcome the occlusion; an initial velocity in the positive x direction with a relatively low *sigma_velocity* causes predictions to continue moving in the correct direction, picking up the object again once it is no longer occluded.

What is the effect of assuming decreased/increased system noise? Increasing the system noise increases the variability in particle propagation. Increasing the *sigma_position* causes the particles to cover a larger area, straying further from their initial location during propagation. This helps capture larger, unexpected motions by effectively increasing the search space of the model. But, it reduces the precision of predictions and introduces more noise to the system. Increasing *sigma_velocity* causes the particle velocities to vary more, spreading out predictions. Higher values can help capture rapid changes in velocity. But, as with *sigma_position*, this spread can introduce noise and hurt precision.

With the hand tracking in *video2*, relatively low values for both sigmas are effective as the hand moves at a relatively constant velocity without large, abrupt motions.

What is the effect of assuming decreased/increased measurement noise Increasing measurement noise, *sigma_observe*, effectively performs non-maximum-suppression. Low values dampen the variance in the χ^2 distances, yielding a relatively narrow distribution of weights; all particles have at least some impact on estimation. High values increase the variance, yielding a much wider distribution. These estimates are biased heavily toward particles with low χ^2 distances, as the weight of particles with high distances drops to negligible values.

For *video2*, increasing this value did not help as the model weights singular particles too heavily. The hand is not very distinct in color histogram from parts of the background, yielding some false positives. By over-prioritizing these false positives, the model is pulled off track. Thus, a relatively low *sigma_observe* that considers all particles in the point cloud works best for this case.

4.2 Effect of Parameters on *video3*

Why do the best parameters from *video2* fail on *video3*? The parameters were tuned for the challenges of *video2*; *video3* presents different challenges. In particular, the motion assumptions leveraged in *video2* no longer hold. Particularly at the point of rebound, the model expects the motion to remain constant and continue in the positive x direction. When the ball rapidly rebounds, all particles continue in the positive x direction and are too far away to capture it. Increasing the system noise allows this transition to be handled.

How does this video affect the answers given above? The explanations remain true, but different values are needed as the situation differs. Several key changes (discussed above and below) are made to optimize for *video3*. The parameters *sigma_velocity*, *sigma_postion*, and *num_particles* are all increased to handle rapid motion and rapid changes in motion (specifically the rebound). The parameter *alpha* (initially is decreased due to the consistent color profile of the tracked object) is decreased given the clearly defined colors and contrast.

4.3 General Effects of Parameters

What is the effect of using more or fewer particles? Increasing the number of particles generally improves performance but is more computationally expensive. If too few particles are used, performance deteriorates substantially, as it is common for no particles to cover the object. Once the model has lost the object, it is challenging to recover.

A sufficiently high *num_particles* is particularly important when large amounts of system noise are introduced as more particles are needed to cover the larger area exhaustively.

What is the effect of using more or fewer bins in the histogram color model? Reducing the number of bins allows slight color variations to be disregarded. With a large *num_bins*, two very similar colors may fall in separate bins. Meanwhile, with a smaller *num_bins*, the same two colors may be considered identical. This is desirable to some degree, as minute differences should be disregarded. However, too much of this effect results in a loss of specificity. Values between 8 and 32 yield similar results in the given videos. Below 8 and above 32, performance deteriorates. Additionally, at high values, the computation of χ^2 becomes expensive, especially if a large *num_particles* value is in use.

What is the advantage/disadvantage of allowing appearance model updating? By continuously adapting the model, it can learn a new target histogram based on observations, enabling tracking of an object in conditions different than the original capture. This is useful if the lighting conditions gradually change or the object transitions to a new background. But, this can also lead to drift as, even slightly incorrect predictions can propagate errors, resulting in a target histogram that is not representative of the tracked object.

For example, failures occur in *video2* when the hand passes over the poster. Here, an alpha value is added to overcome the background change. However, the object in *video3* has a consistent color throughout the video that is very different from the background. Here, the alpha value hurts performance and is decreased so the model does not drift from the original target.

References

[1] Isard, M., Blake, A. - 'CONDENSATION - conditional density propagation for visual tracking', IJCV 1998. 6