*Prof. Marc Pollefeys*

# Eric Tillmann Bill: Assignment 5

erbill@student.ethz.ch

## Implementation

In the following, I will briefly describe for each function my implementation.

### `color_histogram()`

In this implementation, I first extract the bounding box of the current state. Then I calculate a histogram for each color channel using `hist_bin` evenly distributed bins between 0 and 255. The final result is then obtained by concatenating these individual histograms and normalizing the resulting vector. This normalization is achieved by dividing each element by the total number of all histograms to ensure that the sum of the values in the vector is 1.

### `propagate()`

For my implementation, I devised the matrix $A$ by solving the equation:

$$s_{t+1} = A \times s_t + w_t$$

In particular, I solved the following equations for each corresponding model:
**No Motion:**

$$\begin{pmatrix} x+\epsilon_1 \\ y+\epsilon_2 \end{pmatrix} = A \times \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \end{pmatrix}, \text{ with } \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \end{pmatrix} \in \mathcal{N}(\mathbf{0}, \begin{bmatrix} \sigma^2_{\text{position}} & 0 \\ 0 & \sigma^2_{\text{position}} \end{bmatrix},$$

**Constant Velocity:**

$$\begin{pmatrix} x+\dot{x}+\epsilon_1 \\ y+\dot{y}+\epsilon_2 \\ \dot{x}+\epsilon_3 \\ \dot{y}+\epsilon_4 \end{pmatrix} = A \times \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \end{pmatrix}, \text{ with } \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \end{pmatrix} \in \mathcal{N}(\mathbf{0}, \begin{bmatrix} \sigma^2_{\text{position}} & 0 & 0 & 0 \\ 0 & \sigma^2_{\text{position}} & 0 & 0 \\ 0 & 0 & \sigma^2_{\text{velocity}} & 0 \\ 0 & 0 & 0 & \sigma^2_{\text{velocity}} \end{bmatrix},$$

Here $\mathcal{N}$ denotes the normal distribution, $\mathbf{0}$ the zero vector and I arrived at the following matrices:

$$A_{\text{No-Motion}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad A_{\text{Constant-Velocity}} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The actual implementation then simply samples the noise vector $w_t$ using `np.random.normal` with the corresponding standard deviations, calculates the new state as described above, and then clips the new coordinates of the resulting state to the frame size.

### `observe()`

In my implementation, I assign weights to each sample based on their proximity to the initial color histogram of the object. To achieve this, I first calculate the current bounding box for each sample. Then, I compute the color histogram for this box, measure the $\chi^2$-distance between this color histogram and the original color histogram of the object to be tracked, and finally, I smooth the distance with a normal distribution. In the end, I obtain a corresponding weight value for each sample. These weights are then normalized to ensure that they sum to 1, and the normalized weights are returned in the end.

### `estimate()`

This function calculates the most likely state of the object to be tracked by simply calculating a weighted average of all states (`particles`) with the corresponding weights (`particles_w`).

### `resample()`

This function takes a set of states (`particles`) and their corresponding weights (particles_-w) and performs resampling based on the state weights, with the goal of generating a new set of states that better represents the underlying distribution. In my implementation, I use the function `np.random.choice` so that I perform a random sampling with replacement and the corresponding weights. Afterward, I normalize the weights of the new samples before returning them so that they sum to 1.

## Experiments

### video1.avi

In this video, the focus is on tracking the hand in motion against a uniform background, which simplifies the tracking process. However, since the hand and arm have the same color, the tracker has difficulty distinguishing between the two objects and simply chooses the center of both as the most likely position of the hand. This issue is attributable to the design choice, of solely relying on color information, namely the color-histogram, which does not encode any structural information. See Fig. 1, for a tracking trajectory of the hand with the static model.

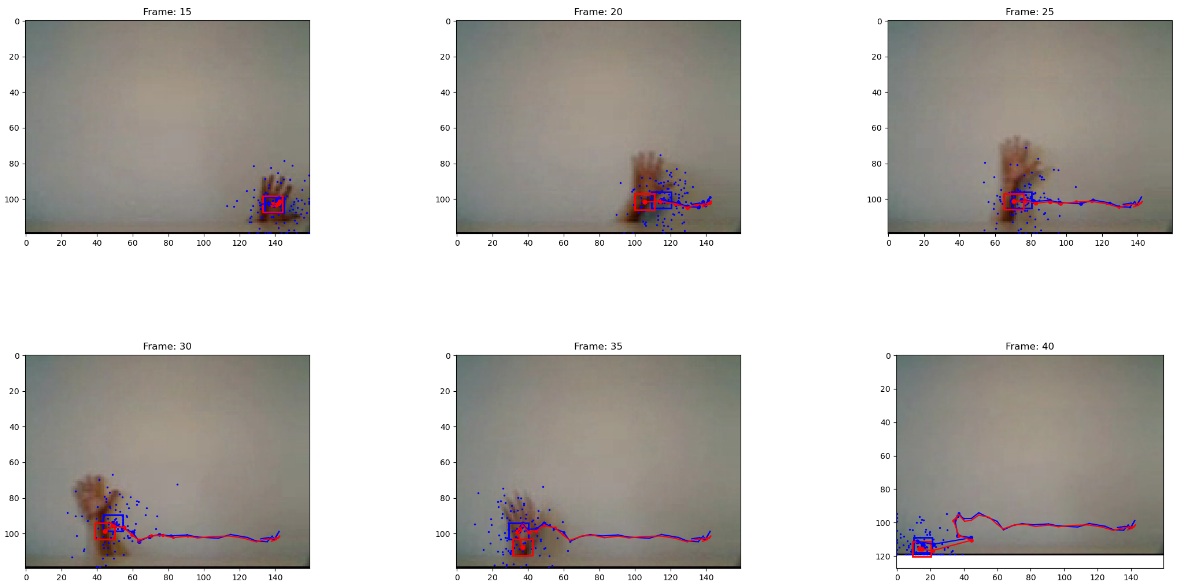


Figure 1: Experiment with video1.wmv

## video2.avi

### What is the effect of using a constant velocity motion model?

When using a model that assumes a constant velocity of the object to be tracked, the model's prior belief of the position will predict that the object will move with the same constant velocity as its previous observed state. Such that all the samples that are drawn (the particles), will be centered around the position the object would appear if it would move with the previously observed velocity. The velocity is then updated, when we update our posterior belief by reweighting every sample.

However, this assumption can lead to problems if the tracked object suddenly comes to a halt. An illustrative example is shown in Fig. 2. In this scenario, the posterior prediction of the hand incorrectly places it much further to the right than its actual position, illustrating a limitation of the constant velocity motion model.

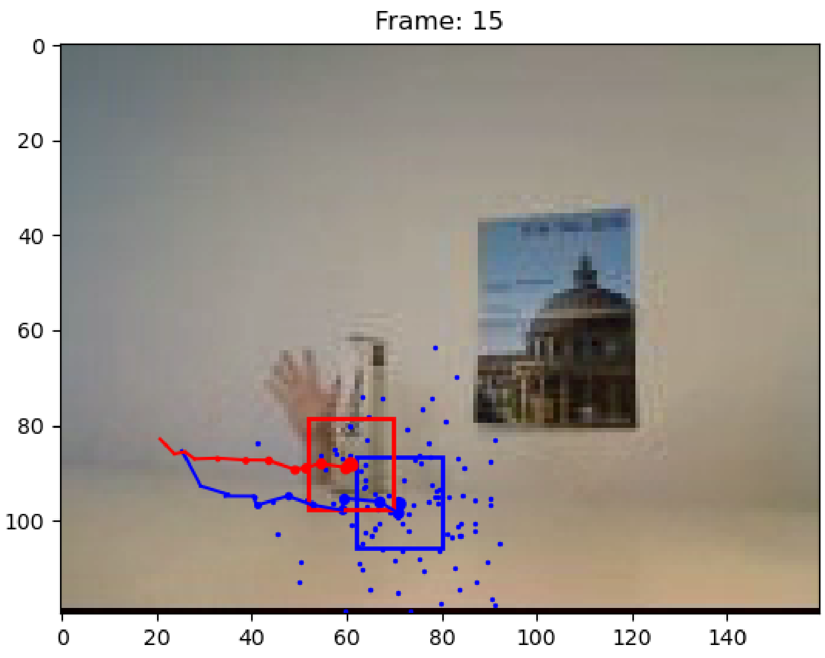


Figure 2: Caption

### What is the effect of assuming decreased/increased system noise?

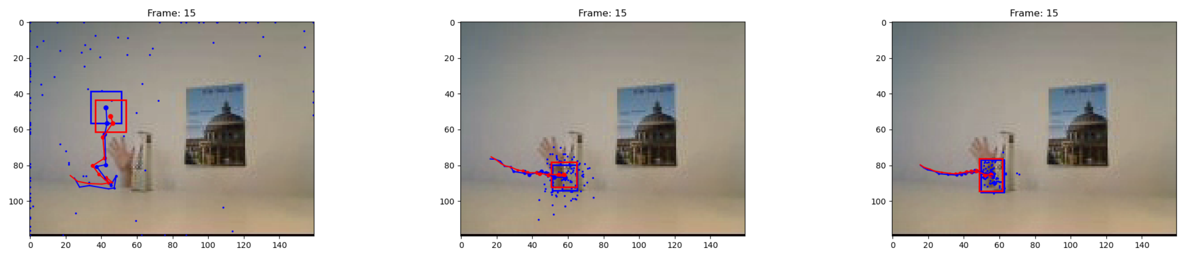
Modifying system noise parameters introduces variability in states, affecting the exploration-exploitation balance in object tracking. Increased noise promotes randomness in state predictions, encouraging exploration across potential object locations. Conversely, decreased noise encourages the exploitation of prior knowledge.

In 3 and 4, I present three scenarios: a configuration with high noise, a balanced configuration, and a configuration with minimal noise. Analyzing the impact on video2, specifically referring to Fig. 3 before the object disappears and Fig. 4 after it reappears, showcases interesting differences between the configurations.

The high noise configuration scatters samples widely across the entire frame space, emphasizing extensive exploration. Conversely, the minimal noise configuration confines all samples within the object's bounding box, demonstrating a reliance on previous knowledge.

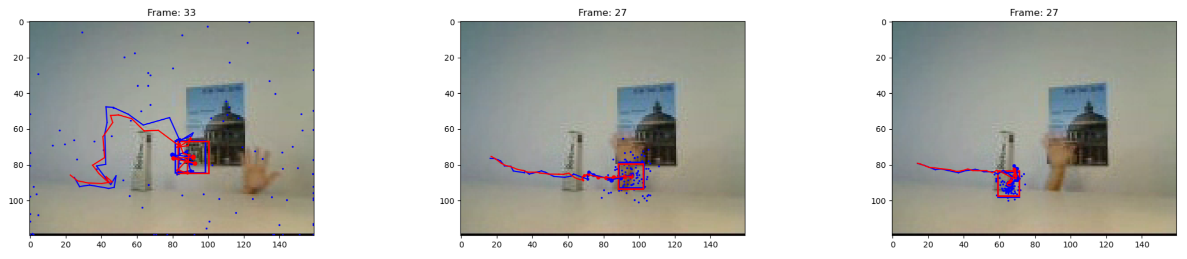
The consequence of these sample generation approaches is evident. The high noise configuration yields unstable predictions, making significant positional shifts and predicting the object to be at various locations. However, its high variance aids in easily detecting the object after reappearing. In contrast, the minimal noise configuration lacks sufficient exploration after the object disappears, relying solely on its last-known object position and getting stuck at this specific position.

In conclusion, achieving a balance in noise levels is crucial for the model to effectively explore and exploit information. The middle image in Fig. 3 and Fig. 4 illustrates a balanced parameter setting where the model successfully tracks the object.



(a) High System Noise (b) Balanced System Noise (c) Low System Noise

Figure 3: Three different configurations of system noise before the object to be tracked disappears.



(a) High System Noise (b) Balanced System Noise (c) Low System Noise

Figure 4: Three different configurations of system noise after the object to be tracked reappears.

**What is the effect of assuming decreased/increased measurement noise?**

When evaluating the similarity between two states, the color histograms for the bounding box of each state are calculated. The $\chi_2$ distance between these histograms is then calculated and subsequently reweighted using a Gaussian distribution. This Gaussian distribution is 0-centred and is parameterized with the parameter $\sigma_{\text{observe}}$. This parameter regulates the admissibility of noise in our measurements.

A higher $\sigma_{\text{observe}}$ value ensures that larger $\chi_2$-distances between two states are not excessively penalized, treating them with a value that is proportionate to their actual difference, as opposed to treating them similarly to a case where the distance is 0. Conversely, selecting a smaller value for this parameter emphasizes that states are only considered very similar if their distance is nearly 0.

If $\sigma_{\text{observe}}$ is set to a large value, the model assigns the same probability to almost all states. This has the effect, that the samples tend to spread out over the entire frame, and the tracking gets stuck. On the contrary, if $\sigma_{\text{observe}}$ is set to a very small value, the tracking is quite good up to the point where the hand disappears, as it then assigns very low probabilities to all bounding boxes that contain only parts of the obscured hand, and hence gets stuck at this point.

## video3.avi

In video3, the tracking task involves following a black ball against a uniform white background. Unlike video2, the ball maintains visibility throughout the entire sequence, simplifying the tracking process. However, the challenge lies in the ball's faster movement. To achieve accurate results, it is essential to increase the `initial_velocity`, `sigma_position`, and `sigma_velocity` parameters. A successful tracking using the constant velocity model is illustrated in Fig. 5.

Similar to video2, altering the noise parameters in video3 produces analogous effects, despite the absence of the object disappearing and reappearing. It remains essential to strike a balance in the noise levels to optimize the tracking performance.

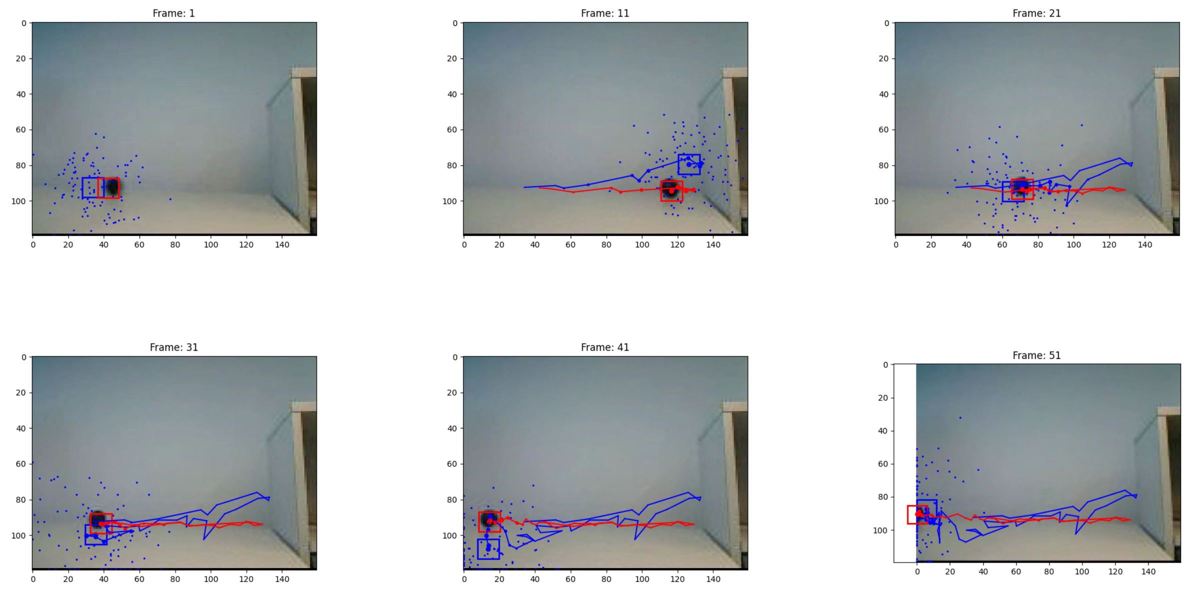


Figure 5: Experiment with video3.wmv

## General Questions

### Impact of Particle Count on Tracking

The number of samples/particles is a critical factor, influencing the information extracted from the current frame during tracking. A very low number increases randomness, making the model more susceptible to the impact of individual samples. Conversely, a higher number results in more comprehensive information retrieval, minimizing the influence of randomness. However, a larger sample count introduces increased computational demands, potentially slowing down the algorithm significantly for very large values.

### Effect of Histogram Bin Count on Tracking

The number of bins in a color histogram determines the level of detail captured. A smaller bin count merges more colors into the same bin, leading to multiple bounding boxes being mapped to similar histograms. Conversely, a higher bin count differentiates between color histograms, making them more dissimilar. Extreme cases, such as a bin count of 1 mapping all bounding boxes to the same histogram, and a maximum count of 255 resulting in the most detailed histogram, showcase the spectrum.

In practical terms, a smaller bin count is advantageous when dealing with changes in illumination during tracking, ensuring consistent mapping of the same colors. On the other hand, a higher bin count is beneficial when illumination remains constant, and the background is cluttered, allowing for effective differentiation between the object and the background.

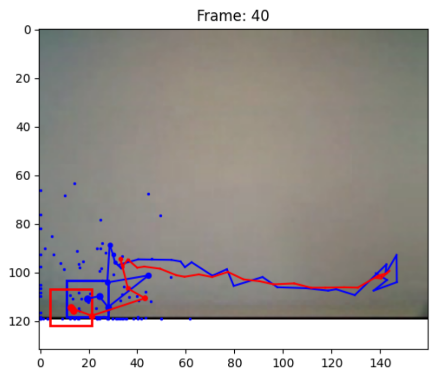
### What is the advantage/disadvantage of allowing appearance model updating?

The advantage of updating the appearance model is that the tracking can adapt to changes in the appearance of the tracked object over time. This is particularly useful in scenarios where the appearance of the object may change due to changes in lighting conditions, occlusion, or other factors. By updating the appearance model, the tracker can better capture the evolving visual characteristics of the object, resulting in improved robustness and accuracy in tracking.

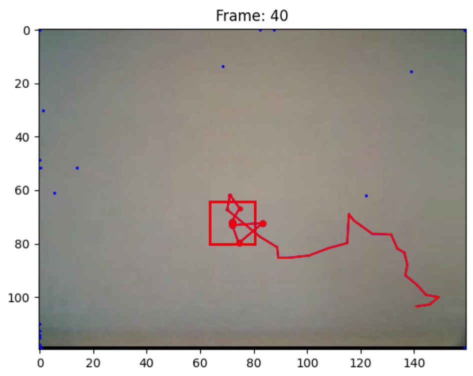
However, a potential drawback of updating the appearance model is the risk of adapting too quickly to transient changes or outliers in the object's appearance. If the model is updated too frequently or without adequate filtering, it may be sensitive to noise, leading to less reliable tracking results. A balance between the frequency and robustness of appearance model updates is critical to ensure effective adaptation without compromising the stability of the tracking algorithm.

## Examples of Tracking Performance for each Video

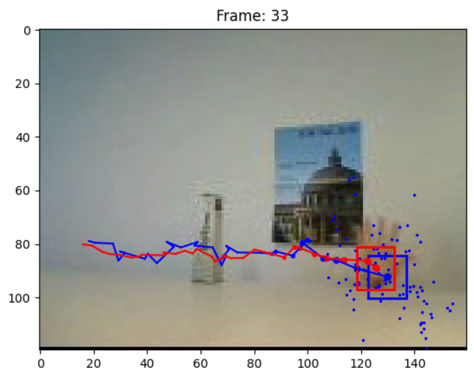
In the following figure, we have listed a complete trajectory for a successful and an unsuccessful tracking of the object for each video.



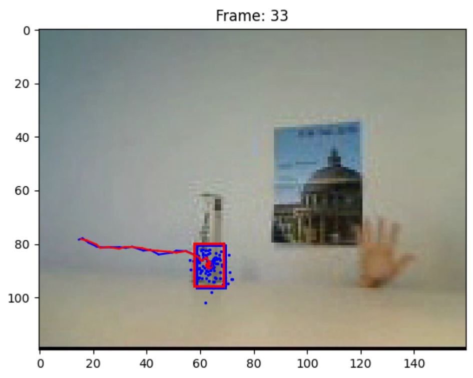
(a) Successful object tracking for video1



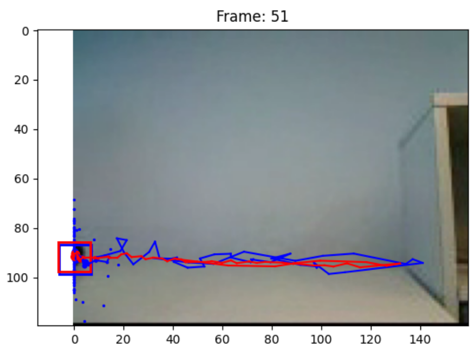
(b) Unsuccessful object tracking for video1



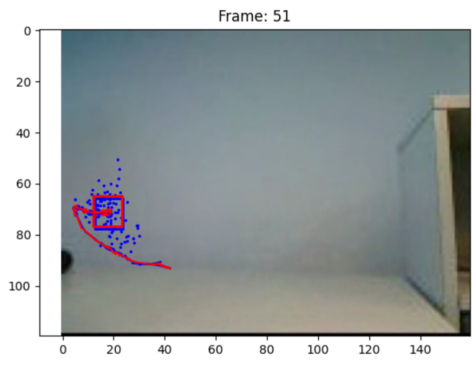
(c) Successful object tracking for video2



(d) Unsuccessful object tracking for video2



(e) Successful object tracking for video3



(f) Unsuccessful object tracking for video3