OBJECT TRACKING - A PARTICLE FILTER APPLICATION

The goal of this project is to implement an object tracking algorithm using a **particle filter**, with the aim of tracking a moving object in a sequence of frames. The particle filter belongs to a family of methodologies known as recursive Bayesian estimators. These recursive algorithms probabilistically estimate the state of a system by combining prior knowledge of the system's state with the measurements - observations obtained for this system. The two basic models we use in the application of the filter are the **prediction - dynamics model** and the **observation model**. The first describes the dynamics of the system and the mechanism by which the state evolves, while the second describes the mechanism by which we obtain and utilize state-related measurements. It is common for noise to enter both models, which in the general case is non-Gaussian. Here we encounter one of the fundamental differences between the particle filter and the Kalman filter (a sensor fusion filter which also belongs to the family of Bayesian estimation filters) as the Kalman filter assumes the noise is Gaussian, in contrast to the particle filter which is not limited by the type of noise and is applied in cases of non-Gaussian noise. In this project, I will explore the practical applications of the particle filter algorithm, focusing on the Computer Vision problem of Object Tracking.

The operation of the particle filter is as follows: It estimates the distribution that describes the state of the system in a non-parametric way, as it uses the so-called particles for this estimation, which are samples of the distribution. Each particle is characterized by a weight. This weight value represents a probability (the sum of all weights is equal to 1) and the larger this value, the better the particle describes the state of the system. The algorithm consists of the following steps:

A) **Particles initialization.** The particles are initialized —usually in a random manner if there is no initial information— and in such a way that they have the same weight. In this specific implementation (as we will see later), the particles are not initialized randomly, as we utilize some initial information to achieve a more targeted and accurate initial estimate. Initialization occurs only once, at the beginning of the application, and not at every run of the algorithm.

B) **Resampling**. At the beginning of each run of the algorithm, we have a set of particles with different weights, which set resulted from the immediately previous run. During the resampling stage, we sample this set using resampling with replacement, and the probability of selecting a particular particle equals its weight. The purpose of resampling is for particles with large weights to prevail, as they describe the state of the system more accurately. It should be noted that due to the resampling stage, the set of particles may contain multiple copies of only certain strong (big weight values) particles. That is, there will be few particles describing the distribution, which is undesirable, as we aim to have many particles-samples of the distribution in order to achieve the best possible estimatation. This phenomenon is encountered in the current implementation of the filter

and is addressed by spreading the particles that result after sampling by applying small variance Gaussian noise (so that they do not move significantly from their initial position).

C) The next step of the algorithm is the application of the **prediction model**. As we have already mentioned, the prediction model determines how the state of the system evolves. In other words, it determines how the particles are transformed during the transition from state at time t to state at time t+1. In this case, the prediction model used is a zero-mean Gaussian.

D) The final step involves taking measurements for the state of the system and incorporating this information into the overall mechanism of state estimation and prediction. The so-called **observation model** is applied, and for each particle, a weight is calculated based on the likelihood P(Yt|Xt), where Xt is the state of the system at time t, and Yt is the measurement at time t. In this specific problem, the chosen observation model is based on edge detection.

The task on hand

The video we are dealing with shows a can of cola being moved on top of a table. It can be approximately considered to be a rectangle and that is why the particles are also rectangular. We aim to track this can across all video frames. The state of the targeted object can be described by the vector (x,y,h,w) where x,y are the coordinates of the top left corner of the can and h,w are its height and width. In this case, the can of cola moves from right to left and the camera is steady so there are no differences in scale across the frames. With that in mind, we can consider the height and the width values to be constant so we end up with a 2D state vector (x,y).

Particle Filter Implementation

Let's take a closer look at the particle filter implemented for tracking the beverage can in the video seven_up.avi. Initially, the program reads the video and informs the user if this process was successful. If it wasn't, the program terminates. Next, the first frame is read and displayed on the screen. The user has the option to select and define the initial position of the object, i.e., the bounding box around the beverage can, by clicking on the upper left corner and dragging the mouse to the lower right corner. **It is important for the first click to be at the upper left corner**, as this is what the program expects, resulting in the coordinates of the first click point to be stored as the coordinates of the upper left corner of the bounding box around the initial position of the can. As mentioned before, the target model is a rectangular bounding box, in other words, the particles themselves will be rectangular. Each rectangular particle is described by the coordinates (x, y) of the upper left corner and the values of its height (h) and width (w). All particles will have the same **fixed** height and width, values which are determined by the user during the design of the

bounding box in the first frame as previously mentioned (the can maintains the same height and width throughout its movement, so the particles will have the same values, which also remain constant).

Next, the particles are initialized. Given that the user has defined the initial position of the object, the initialization is not done randomly. Instead, we utilize this information so that the particles are initialized near the object. For the generation of each particle, we take a sample from a 2D Gaussian distribution characterized by a mean vector [x, y] (where (x, y) are the coordinates of the upper left corner of the bounding box drawn by the user). This sample contains two values: the first represents the displacement of the initial bounding box along the x-axis, and the second represents the corresponding displacement along the y-axis. By displacing the initial box differently each time based on the sample obtained, a different particle is generated. Finally, the particles are drawn onto the frame, and the processed first frame is written to the output video. The first frame requires specific processing different from the process applied to the other frames of the video, which is described immediately below.

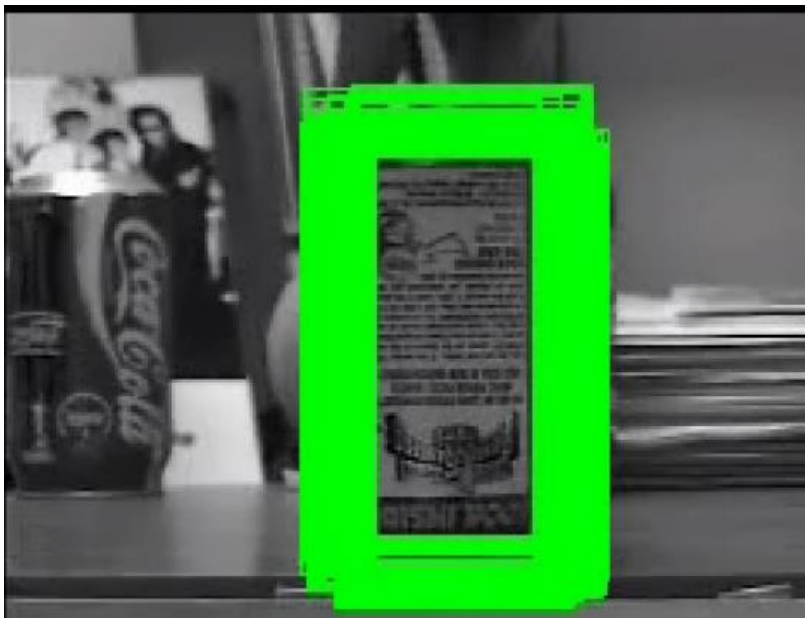
From the second frame onwards, the steps of the particle filter we described previously are applied. The particles are resampled with replacement based on their weights, and the new set of particles is spread to avoid ending up with a small number of particles. Then, the zero-mean Gaussian prediction model is applied to displace the particles. This model is common to all particles—each particle is displaced on the image the same way—because the prediction model describes the overall evolution of the system. In contrast, in the case of spreading mentioned earlier, each particle is displaced slightly around its initial position in a different way from the others.

Finally, the observation model is applied to calculate the new weights of the particles and update the old ones. The observation model is based on edge detection. Specifically, a Canny edge detector is applied to the frame, resulting in a binary image where a pixel value is 1 if it is on an edge, otherwise it is 0. The distance transform is then applied to the binary image, resulting in a new non-binary image where the value of a pixel is the Euclidean distance between itself and the nearest non-zero pixel in the edges image produced by the Canny detector. Next, we traverse all the particles and for each particle we slice the image obtained from the distance transform to get the portion of it that lies within the particle. We sum all the pixel values at this region of interest and divide by their total number to get an average distance value d, of the specific particle from the nearest edges. This value is given as an argument to the function $\exp(-2d)$ and by dividing this result by the sum of all exponentials for each particle - $\sum\exp(-2d)$ - we receive the weight of the particle. We are essentially using a softmax function to create probability values.

How is the accuracy of the tracking algorithm affected by the number of particles used and how should the covariance matrix of the prediction model be adjusted?

Tests were conducted for a range of particle numbers from 10 to 5000. It was observed that tracking was possible even with just 10 particles. However, it was also noted that as the number of particles increased, the tracking accuracy improved, and the algorithm became more robust. In other words, the result was not significantly affected by the randomness that characterizes the subprocesses occurring during the program's operation (e.g., sampling from Gaussian distributions). Conversely, when using a few particles (e.g., 10), to achieve a good result and not lose the target, the program had to run multiple times, and in most of these runs, the tracking failed. This result does not surprise us. We know that the more particles used, the better the estimation of the distribution describing the state of the system. It is important to find a good balance between the number of particles and the time it takes for the program to run. After trials, it was found that a value of 100 is a sufficient number of particles to achieve good tracking while allowing the program to run fairly quickly. The sacrifice we make in terms of tracking accuracy (due to using only 100 particles) to achieve satisfactory run times is almost insignificant, as increasing the number of particles beyond 100 results in differences in tracking quality that are very small or nonexistent. Below are snapshots of the output video (at the 13th second) for various particle counts (in all cases, the covariance matrix remains the same).

10 particles used



100 particles used

100 particles used

Regarding the covariance matrix of the prediction model, i.e., the zero-mean 2D Gaussian distribution, tests were conducted for various values of the standard deviation along the x-axis and y-axis, as well as tests for the covariance values between the two dimensions. It was found that for various small covariance values, the result remains practically unaffected. However, if the covariance increases significantly, we observe a substantial decrease in the quality and accuracy of the tracking. This should not surprise us, as the object performs a relatively linear movement almost entirely along the x-axis. Therefore, we do not expect any substantial correlation between the two spatial dimensions, and for this reason, we choose to set the covariance to zero. Regarding the values of the standard deviations of the covariance matrix, due to the limited movement along the y-axis, the standard deviation along the x-axis should be greater than the corresponding value for the y-axis, which should be kept at low levels. This fact is confirmed experimentally, and empirically it is found that a good value for the standard deviation along the x-axis is $\sigma x = 2$, while for the y-axis, $\sigma y = 1$. If the standard deviation σx is increased further ($\sigma x >= 4$), then the displacements of the particles from frame to frame are excessively large compared to the object's movement, resulting in target loss. Below we observe the influence of increasing the standard deviation and the mentioned target loss due to the large standard deviation.

100 particles, $\sigma x = 2$, $\sigma y = 1$

100 particles, σx = 5, σy = 1

The described algorithm for object tracking was implemented in Python utilizing the OpenCV and NumPy modules.