ECSE-626 Project: An Adaptive Color-Based Particle Filter

Fabian Kaelin
McGill University
Montreal, Canada
fabian.kaelin@mail.mcgill.ca

Abstract

The goal of this project was to discuss and implement a paper that deals with a problem in computer vision. The paper that was chosen presents a method for target tracking in video sequences. More specifically, Nummiaro et. al [2] present a particle filter that uses a simple linear dynamical model and a likelihood model based on color-histograms. The algorithm is adaptive in the sense that the target model is updated over time to make it more reliable and robust against changes in illumination or scale. Since the paper that presented the algorithm has already been written, this report will focus on the implementation. The limitations and weaknesses are discussed. The results obtained from the implementation are compared to the results in the paper. Finally some ideas that extend the paper are presented.

1. Introduction

The motivation for choosing this paper is to get an intuition behind the mathematics of sequential Monte Carlo methods. At the same time this report might serve other people that are learning about particle filters as an example of how particle filters can be applied. Since this work is done in the scope of a project for a course, there was a limit in complexity in the paper if we wanted to avoid simplying the original idea. The paper that was chosen was ideal because only few additional assumptions or modifications were necessary to reproduce the proposed approach.

Since the idea is not to just replicate the original paper, this report will go into more detail about the actual implementation. At the same time this allows us to point out the weaknesses of the paper and present some ideas about how it could be extended or improved. We will start off by introducing the models used and by explaining the additional assumptions that apply for this implementation. Next the particle filter is presented and the individual steps are described in more detail. The implementation is then tested on a video and the performance is compared to the results that the authors of the original paper presented. Finally a number of possible ideas are proposed that extend the scope of the original paper.

2. Models

We have to find a representation for how the target moves and we have to define a method for calculating the likelihood of a target state given an observation frame.

2.1. Dynamical Model

Before we can define the dynamics we have to define a target model

$$s = \{x, y, \widehat{x}, \widehat{y}, H_x, H_y, \widehat{a}\} \tag{1}$$

where (x,y) specify the location, ($\widehat{x}$,$\widehat{y}$) the velocity, (H_x,H_y) the size and $\widehat{a}$ a scaling change. Note that in the original paper they used an ellipse, whereas we use a rectangle for simplicity. We can define a dynamical model that predicts the state of a potential target state at the next time step.

$$s_{t+1} = A * s_t + w_t \tag{2}$$

where A is a deterministic component of the model and w_t a multivariate gaussian. At this point we assume that the model moves with constant velocity ($\widehat{x}$,$\widehat{y}$) and $\widehat{a}$ simply changes the scale of (H_x,H_y). This not a very sophisticated model could be extended to include more features like the acceleration.

2.2. Likelihood Model

If we have a predicted state s_p of where the target could be in the next frame, we would like to calculate the likelihood that s_p is actually at this location given the the next frame. This is done using the color histogram of the target rectangle. We use $8\times8\times8$ bins in the RGB space and denote $h(v_i)$ as the function that assigns the color values of pixel v_i to the corresponding bins. To increase the reliability of the distribution, we assign pixels that are further away from the center a smaller weight based on the weighting function

$$k(r_i) = \begin{cases} 1 - r_i^2 & r_i \leq 1 \\ 0 & otherwise \end{cases} \tag{3}$$

where $r_i = \frac{||v_i - c||}{b}$ is the normalized distance from the target center $c = (x, y)$ to the actual pixel v_i that we are evaluating when computing the histogram and b a normalizing constant (e.g. $b = \sqrt{H_x^2 + H_y^2}$). Because we want a probability distribution (independence of the size of the rectangle), we normalize the color distribution by a factor f

$$f = \sum_{i=1}^{I} \frac{1}{k(r_i)} \tag{4}$$

where I is the number of pixels in the rectangle. Finally the color distribution can be written as $p_s = \{p_s^{(u)}\}_{u=1...8}$ where

$$p_s^{(u)} = f \sum_{i=1}^{I} k(r_i)\delta[h(v_i) - u] \tag{5}$$

that allows to compare the similarity of the states. A popular measure to do that is the Bhattacharyya coefficient, which is defined as

$$\rho[p, q] = \sum_{u=1}^{8} \sqrt{p(u)q(u)} \tag{6}$$

for two discrete distributions p and q. ρ is 1 if p and q are identical and decreases the more different they are. Finally we define a distance d between two distributions (and two states for that matter) as the Bhattacharyya distance.

$$d = \sqrt{1 - \rho[p, q]} \tag{7}$$

We will use this measure to determine the likelihood of a proposed target state given a new observation frame.

3. Particle Filter

Now we have defined the models needed to conceptualize a particle filter based on color historgrams. In this section we will first give an outline of the different steps executed during the algorithm. Then each of those steps will be explained in detail.

We want to track a target in a video sequence, therefore we have a target state X_t at time t. If we denote our observation frames as $Z_t = \{z_1, ..., z_t\}$, the goal of our particle filter is to find the posterior distribution $p(x_t, |Z_t)$ for every new frame z_t that we observe. As in all sequential Montre Carlo methods, we approximate the posterior distribution by a set of particles $S = \{(s^{(n)}, \pi^{(n)})|n = 1.....N\}$. Every particle $s^{(n)}$ is represented by the defined parameters $\{x, y, \widehat{x}, \widehat{y}, H_x, H_y, \widehat{a}\}$ and has weight $\pi^{(n)}$ associated with it. This is a probability distribution, therefore the weights must satisfy

$$\sum_{n=1}^{N} \pi^{(n)} = 1 \tag{8}$$

Consequently, the best approximation for the posterior at each time step is the mean state of all our particles

$$E(S) = \sum_{n=1}^{N} \pi^{(n)} s^{(n)} \tag{9}$$

Note that at this point we assume that we have a target that we want to track, and we denote the the color distribution of the target as q. In the section 'Initialization' we will talk about different ways we can define or find the original target X_0.

Now we have all the background to formulate the steps of the algorithm that we will use iteratively at each time step.

1. Resampling the particles to avoid degeneracy

2. Propagate each particles according to our dynamical model

3. Update the weight $\pi^{(n)}$ of each particle according to our likelihood model

4. Estimate the posterior state $p(x_t, |Z_t)$ of the target given the new frame z_t

5. Adapt the target's color distribution q to increase reliability and robustness

3.1. Resampling

In our set of particles, we will have undoubtably have samples that move in the wrong direction. Since the Bhattacharyya distance to the target will be big, the weight $\pi^{(n)}$ of those particles will become small. After a few iterations those particles will not contain any information about the target location and the particle set will contain many particles with weight close to 0 and a few particles with high weights. Because those few particles are not enough to detect the movement of the target we will loose track fairly quickly. This phenomena is called particle degeneracy. In order to avoid this effect we will resample our particles for time step t from our particle set at time $t - 1$ with replacement. Our new particle set likely contains multiple copies of particles with high weights whereas particles with low weights are likely to be discarded form the set. This can be done by selecting particles according to their weights. Figure 1 illustrates this nicely.

3.2. Propagate Particles

The particles are described by the parameters (1). Every particle is propagated using the dynamical model defined in (2). Because in the resampling step we eliminate particles that have low weights, the set tends to be grouped around our target (or rather what we believe to be our target). In figure 2 we can see a particle set around a face.

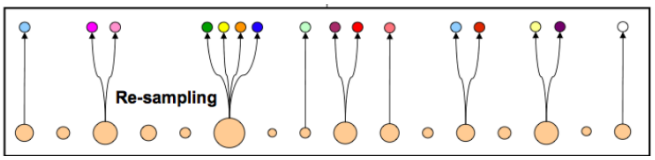

Figure 1. Resampling illustration from [1]. The particles with higher weights (bigger circles) are chosen with a higher probabilty, but the total number of samples stays the same

The deterministic component of the dynamical model A assumes a constant velocity, therefore it is up to the gaussian component w_t to capture small changes in velocity or scale. Choosing proper covariance values is crucial and difficult. In an erratic setting where the target often changes direction and speed - e.g. in case of an ant - we will need big covariance values. But consequently the particles will be spread in a wider area and the danger of loosing the target increases. In case of face tracking, the movement is a little more predictable, the covariance values are smaller. The particle set will be more compact and the chance of loosing track is lower. At the same time this makes the algorithm less robust in case of sudden unexpected movement - e.g. somebody running or somebody tripping.

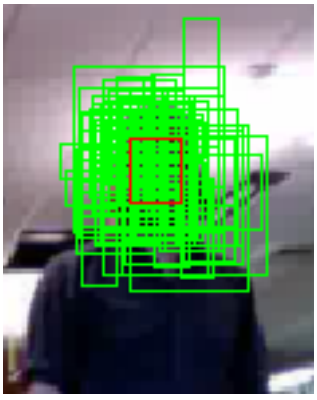

Figure 2. Example of a particle set tracking a face, the green rectangles are the particles and the red rectangle is their posterior (mean state)

In general, a possibility would be to learn covariance values from labeled video sequences. But at this point the covariance values are determined experimentally in this implementation. The authors of [2] don't mention their approach.

Of course the number of particles in the set N is crucial as well. More particles means higher chances of capturing the target but at the same time this directly influences the computation performance. One hundred particles have been found sufficient for this implementation.

3.3. Update Step

Now we have a set of propagated particles where each particle represents a hypothetical state of the target and a new observation - the next frame of the video. The goal is determine the likelihood of each particle given the new frame. First the color distribution is calculated using (5). Next the Bhattacharyya distance (7) is computed, which leaves us with a measure of similarity to the target q for every particle in our set. The distance is used to update the weights

$$\pi^{(n)} = \frac{1}{(\sqrt{2\pi}\sigma)} e^{-\frac{d^2}{2\sigma^2}} = \frac{1}{(\sqrt{2\pi}\sigma)} e^{-\frac{(1-\rho[p_{s^{(n)}},q])}{2\sigma^2}} \quad (10)$$

Finally, to satisfy (8), the weights are normalized by

$$\pi^{(n)} = \frac{\pi^{(n)}}{\sum_{i=1}^{N} \pi^{(i)}} \quad (11)$$

Figure 3. A simple normal gaussian to illustrate the effect σ has on the weight of the particles

In figure 3 you can see that the bigger the Bhattacharyya distance, the lower the weight. The choice of the standard deviation σ of this gaussian has a big influence, in this implementation a value of 0.1 has been found to work well. A big standard deviation (e.g.1) leads to higher weights for particles with big distances. This leads to a particle set that is wider spread which is a little more robust to track the target but at the same time increases the danger of particle degeneracy. On the other hand, a too small standard deviation (e.g. 0.01) leads to a very compact particle set because a few particle will have high weights and will therefore be resampled. As with small covariance values for the dynamical model, this increases the danger of loosing 'sight' of the target. The choice of σ can be seen as a compromise between the danger of loosing the target because of particle degeneracy and the danger of loosing the target because our particle set is too compact. Nummiaro et. al [2] don't discuss this issue.

One of the main advantages of particle filters is their ability to detect multimodal distributions. This is very helpful because it allow the filter to consider multiple hypothesis in case it is not sure where the target is moving. E.g. when tracking a face, if a second face passes by close by, the particles will cover both faces until it has to choose one. Figure

7 illustrates on such example, where the particles 'detected' an arm - which is similarly colored like a face - and the filter stays with both 'hypothesis' for a while, but soon the arm particle filters have been discarded.

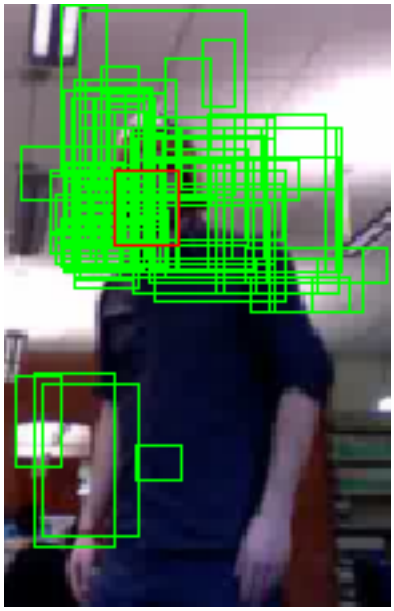

Figure 4. Example of a multimodal particle set tracking a face, but also caught onto an arm

3.4. Estimate

At this point the particles have been propagated and all their weights have been updated. The new posterior $E(S_t)$ is computed as the mean state of all the particles (9). The intuition behind is simple - since the weight of a particle is a similarity measure, the weighted sum should approximate the target state the best. This is easy to do, but as the particle set can maintain multiple hypothesis - represented by a multimodal distribution - the mean state is possibly meaningless. The section 'Target Update' will deal with how to handle this problem. In figure 5 this step is nicely illustrated.

3.5. Target Update

The object that is being tracked will change in appearance over time, therefore it is important that we adapt the target model over time. The target model is represented by the color distribution q. The posterior $E(S_{t+1})$ at time step $t+1$ reflects this change, therefore we can slowly adapt our target color distribution to reflect the possible changes in illumination conditions, rotation and viewing angle. We define a factor α that determines how the color distribution of the posterior $p^{(n)}_{E(S_{t+1})}$ influences the previous target distribution. Over time the target model will have 'forgotten' its original distribution - which of course is a problem at the same time. It is hard to directly measure the impact, but a value of 0.1 seems to return reasonable results. For each bin u we compute

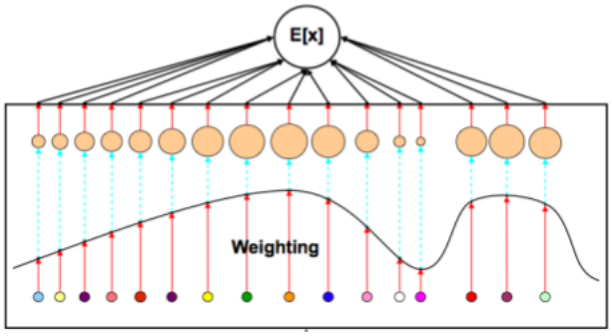

Figure 5. Illustration from [1]. After updating the weights, the mean state is computed

$$q^{(u)}_{t+1} = (1-\alpha)q^{(u)}_t + \alpha p^{(n)}_{E(S_{t+1})} \tag{12}$$

Another problem has already been described in the section 'Estimate'. In case the particle set is distributed over multiple hypothesis, the mean state will be useless. There are two consequences to that. First it is evident that this approach has to choose one of the hypothesis. In the best case all but one eliminate themselves, but if not the algorithm should actively remove outliers to ensure that the filter 'concentrates' on on hypothesis. Secondly, if we expect our target to be not represented properly, the target update should not be executed. To this end we can calculate the likelihood $\pi_{E(S_{t+1})}$ of our posterior $E(S_{t+1})$ state (using (7),(10)). Finally, the target is only updated if $\pi_{E(S_{t+1})} \geq \pi_{thres}$, where π_{thres} is a predefined threshold, otherwise we assume the target is lost or occluded and we hope to 'redetect' it the future.

4. Initialization

The particle filter has been described, but it was always assumed that the original target was known. But how can the target object and the particle set be initialized? In this implementation, the simplest of all was good enough: After a video has been loaded, the user can select its target by drawing a rectangle on the first frame. Four parameters (1) can be extracted from this input, $\{x, y, H_x, H_y\}$. Next, original particles are created by sampling the remaining parameters $\{\widehat{x}, \widehat{y}, \widehat{a}\}$ uniformly from a reasonable interval. This step is repeated N times, and after the first propagation of the particle set they will be nicely spread around the original target.

Nummiaro et. al propose different methods. If one has an initial color distribution - e.g. from a face - but no target state, one could for example place immovable particles around areas where people are expected to enter the scene - e.g. doors and the border of the frame. The weight of

these particles trigger the tracking process once a face is detected. Another, more sophisticated method, would be to employ object detection algorithms to find potential target states.

5. Experiments

The proposed algorithm has been implemented in Matlab. It is not a charm to work with video files in Matlab and a lot of time has been wasted trying to get reasonable video sequences. Every different experiment needs tweaking of the parameters w_t and σ - so it is fair to say that the implementation is not ready to be sold commercially. The code has not been optimized, on a dual-core 2Ghz in OS X manages to process about two frames per second. Note that the performance greatly depends on size of the rectangles of the particles - the bigger the target the slower the particle filter. The number of particles N influences performance as well, where 100 particles proved to be enough in all experiments. In figure 6 you can see how the filter manages to track a face in fairly cluttered environment. Additionally there were plenty of changes in direction. The borders of the frame proved to be tricky since the movement of the particles have to be stopped in order not to disappear, but the filter stays on the target throughout. In the video one can notice a couple of sudden jumps which result from multiple hypothesis which usually push the mean state in the direction of an arm. But overall the tracking was very successful over all 211 frames.

Figure 7 illustrates how the particle filter handles multiple hypothesis. A face is tracked that passes right in front of another face in frame 29. The face is first moving to the right (frame 19), and after passing the face starts moving back to the left. In frame 54 we can see that the particle filter maintains two hypothesis and the posterior is midway between the two faces. Gradually the particle filter is convinced that his target is the left face (frame 57,59). In frame 62 the posterior is fully on target again. This experiment shows that the proposed algorithm is robust even in presence of two very similar objects. When comparing these results to the results in the paper, we find that the performance in the domains tested is very similar.

6. Suggested Extensions

Following a selection of modifications and extensions that could improve the performance of the particle filter and were not mentioned in the original paper. Time constraints unfortunately prevented the transition from theory to practice.

6.1. Likelihood Model

The current likelihood model is based on color histograms. While this seems robust against deformations, rotation and partial occlusion of the target, it completely ignores other features like the shape. It would be interesting to see how the filter would perform if the likelihood would be based on mutual information as defined by Viola and Wells [3] instead. This would probably mean that we would loose much of the robustness, but in a controlled environment where we have more guarantees about the shape and color of the target it could prove useful.

6.2. Focus Factor Θ

Lets assume a target is moving in one direction at a constant speed. The particles with parameters ($\widehat{x}$,$\widehat{y}$) similar to the actual velocity of the target will be assigned higher weights. Those same particles are more likely to be resampled, which leads to a particle set that 'moves' in that direction. When the target changes directions or speed - lets assume that it is moving slower, the particle set will at first continue at the original speed and the propagated particle set will be ahead of the target. In the update step, higher weights will be assigned to particles that moved not too far (be that because of a slower velocity ($\widehat{x}$,$\widehat{y}$) or 'lucky' gaussian noise w_t). This in turn will lead to a posterior (9) which is based on different particles (the slower ones) than at the last time step (the faster ones). In the estimation step we can detect this change of 'focus' between the old weights $\overline{\Pi} = \{\overline{\pi}^{(1)}, ...\overline{\pi}^{(N)}\}$ and the new weights $\Pi = \{\pi^{(1)}, ..., \pi^{(N)}\}$ by calculating a focus factor Θ

$$\Theta = \sum_{n=1}^{N} (\overline{\pi}^{(n)} - \pi^{(n)})^2 \tag{13}$$

A big focus factor Θ indicates a change of the target state parameters (1) that wasn't anticipated by our dynamical model. This changes we are trying to capture with the covariance values of the dynamical model w_t and resampling. But the proposed focus factor Θ gives us a mean to detect such changes. If we are able to find a threshold Θ_{thres} that separates the expected noise in Θ due to the stochastic nature of the particle filter, we can define a new mechanism that triggers as long as

$$\Theta \geq \Theta_{thres} \tag{14}$$

If that is the case there are different methods to react. We could temporarily increase the number of particles in our set to $N_{thres} > N$ to make the algorithm more robust. Another possible measure would be to change the gaussian component of the dynamical model to $w_t^{thres} > w_t$ (2) to increase the spread of the particle set. Similarly one could also change the standard deviation to $\sigma_{thres} > \sigma$ which would lead to higher weights for particles with a high Bhattacharyya distance (7) which in turn would lead to wider variety when resampling the particles. Of course all three

Figure 6. Face tracking with cluttered background and directions and velocity changes - frames 13, 40, 69, 89, 140, 156 (from top left to bottom right)

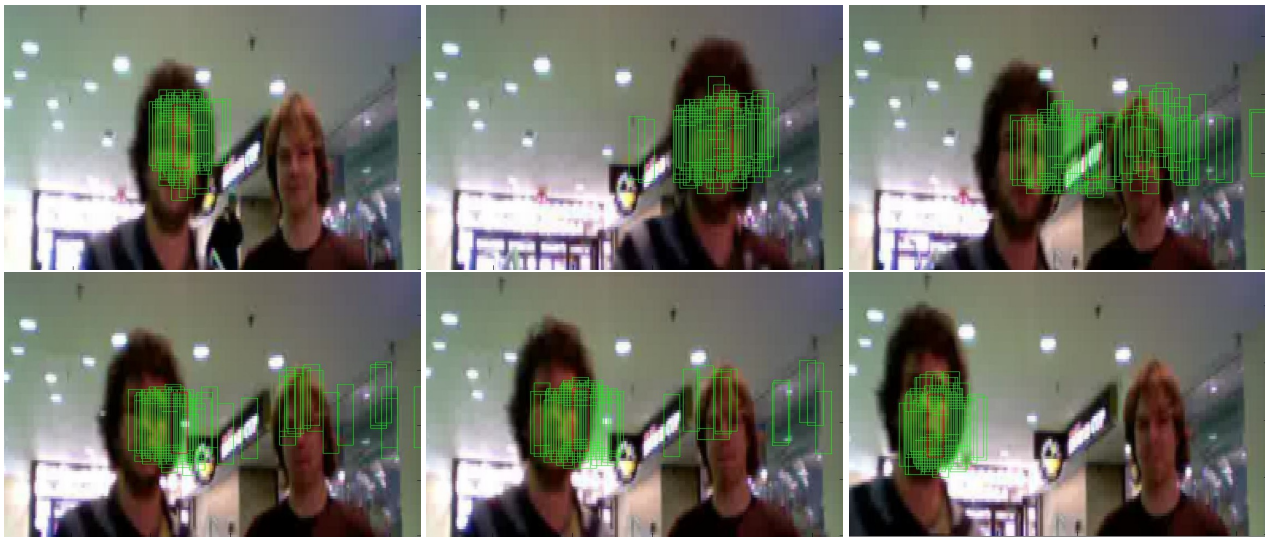

Figure 7. Multiple hypothesis: tracking a face - frames 19, 29, 54, 57, 59, 62 (from top left to bottom right)

measures can be combined. The argument for this extension has been made using the velocity parameters but is of course valid for all parameters.

References

[1] E. Maggio and A. Cavallaro. Hybrid particle filter and mean shift tracker with adaptive transition model. *Acoustics, Speech, and Signal Processing*, 2:221– 224, 2005. 3, 4

[2] K. Nummiaro, E. Koller-Meier, and L. V. Gool. An adaptive color-based particle filter. *Image and Vision Computing*, 21:99–110, 2003. 1, 3

[3] P. V. William and M. W. Iii. Alignment by maximization of mutual information. *International Journal of Computer Vision*, 24(2):137154, 1997. 5