Tracking faces in videos using particle filters

Report author
Jacob Malmberg

January 2020

Abstract

Face tracking is used in a wide array of software such as social media applications and surveillance software. Particle filters have been successfully used to track objects such as faces in the past. In this paper, a color-based particle filter is used to track faces since it is scale and rotation invariant as well as being robust to partial occlusions. The results show that the filter can successfully track faces in a multitude of situations, such as the person moving quickly across the frame and leaving and re-entering the frame. The approach works well even if the color of the face is similar to the background. However, the filter is not efficient enough to be used in real time.

1 Introduction

Face tracking is implemented in a wide array of devices and software, such as smart phones and social media applications [1, 2]. Since smart phones and applications using face tracking are used by millions of people every day it is a problem that deserves attention.

Different algorithms using different models can be used to track faces. These approaches vary from being based on deep neural networks [4] to being based on particle filters [6, 3]. Particle filter based face tracking has been done for a long time, Nummiaro et al. implemented it in 2003 [3] and Wang et al. implemented it in 2019 [6]. It is thus a well entrenched approach which has been improved recently.

The implementations of particle filter based face tracking varies. In [3] only color is taken into account, while [6] uses color as well as edge features. Further, for face tracking to work the face first has to be detected. In [3] this is done using a Support Vector Machine, while in [6] this is done manually.

In this paper, the approaches used in [3] and [6] are combined to implement a color-based particle filter used for face tracking. The results show that the approach works well on a wide range of videos that include fast-moving persons as well as the person leaving and re-entering the video. Fine-tuning of parameters are needed to achieve the best results.

1.1 Contribution

This paper makes the following contributions:

- The results show that color based face tracking using a particle filter works well even if the face and background have similar color distributions.
- The algorithm works well even if the person moves fast.
- The algorithm can manage the person leaving the video and then correctly start tracking the face again as the person re-enters the video.
- An object detection framework is used to initially detect the face so that no manual identification is needed.

2 Related work

Tracking faces using particle filters has been successfully carried out in the past. Nummiaro et al. used a color based color filter to track faces in 2003 [3]. In the article, the authors first created a target histogram of the colors that the face contains. This is done by placing a template around the face and then counting the RGB values of the pixels within this template. Color histograms for each particle are then created similarly by placing a template around each particle and these histograms are then compared to the target histogram in order to create weights, where particles with similar histograms to the target receives higher weights. The estimate of where the face is then acquired by taking the weighted mean of the particles. As the color of the face may change throughout the video due to occlusion or lighting, the authors update the target model over the course of the video. In order to not update the target histogram when the face is out of the frame or severely occluded, the authors use a threshold on the observation probability of the mean of the particles so that the update is only done when the face is clearly identified.

In 2019, Wang et al. proposed an algorithm which uses both color features as well as edge features [6]. By using more than one type of feature, the tracking is more robust.

In [3] the state variable contains the size of the template while in [6] the size of the template is not included in the state variable. Thus, each particle has its own template size in [3] which is propagated in each time step. This allows for multiple hypotheses of the size of the face in the frame. In [6] on the other hand, the template size is the same for all particles. The size of the template in [6] is calculated by comparing the spread of the particles in the current time step to the previous time step, where higher spread in the current time step compared to the previous time step will lead to a larger template and vice versa. The idea is that when the size of the face is small, the particles will be more concentrated than when the face is large, and thus the template should be sized appropriately. The authors refer to this as a self-updating tracking window.

To be able to track a face in a video, a face must first be detected. In [3] this is done using an algorithm based on Support Vector Machines. In [6] this is done manually.

3 Method

The method used in this draws heavily on the method used in [3], but instead of each particle having its own template size the self-updating tracking window from [6] was used. The algorithm is introduced in section 3.1, and the various components are explained in the following sections. In section 3.12 the experiments are explained.

3.1 Algorithm

Algorithm 1: Face tracking

```
Input: template covering the face
initialize particle set S_t, target color histogram q_c;
for all frames do
    if frame > 1 then
        if observation probability of the mean state > threshold then
            update tracking window;
        end
    end
    propagate particles according to motion model;
    for all particles in S_t do
        create color histogram, p_t^n;
        calculate the Bhattacharyya distance from p_t^n to target color histogram q_c, d_t^n;
        calculate the weight using the Bhattacharyya distance, π_t^n;
    end
    normalize the weights π_t;
    estimate the mean state of the set S_t, E[S_t];
    create color histogram for the mean state, p_{E[S_t]};
    calculate the observation probability of the mean state, π_{E[S]};
    if observation probability of the mean state > threshold then
        update target model, q_c;
    end
    resample particles;
end
```

The algorithm used in this paper is displayed above. The various components are explained further in their respective sections.

3.2 Particle filter

The particles had four dimensions:

$$s = \{x, y, \dot{x}, \dot{y}\} \tag{1}$$

where x and y is the center of the template and $\dot{x}$ & $\dot{y}$ are the velocities.

Since the size of the template is not one of the dimensions of the particles, the particles all have the same template size. This approach is identical to the approach in [6] and is further explained in section 3.10.

3.3 Motion model

In the videos, the face will not be stationary but rather move around. The motion model should reflect this. To this end, a first order model is used. The particle set is propagated using the model

$$s_t = As_{t-1} + w_{t-1} \tag{2}$$

where A is the deterministic component and w_{t-1} is a multivariate Gaussian random variable. This approach is the same as in [3]. If a particle was propagated outside the frame, its weight was set to 0.

3.4 Initialization

The input to the face tracking algorithm is a template covering the face. To this end, the face needs to be detected. In [3] and [6] they use a SVM and detect the face manually, respectively. In this paper, the initial detection of the face is done using an object detection framework presented by Viola & Jones in [5]. This approach was chosen since the algorithm is built into MATLAB.

The template returned by the object detection framework contains the face. From this template, the particles are initialized. Thus, the particles are initialized inside the template covering the face in the first frame with uniform weights. Further, from this template the target color histogram is computed. The calculation of the color histograms are explained further in section 3.5.

3.5 Color distribution

To calculate the likelihood of the particles, color histograms need to be created. The approach used in this article is identical to the approach used in [3].

The color histograms are calculated in the RGB

color space using 8x8x8 bins. In the first time step, a target histogram q_c is created from the template created by the object detection framework. In every time step, a color histogram is created for each particle. This is done by first centering a template around each particle. As explained in section 3.2, the template has the same size for each particle. The function $h(\mathbf{x}_i)$ assigns the color at pixel location $\mathbf{x}_i$ to the corresponding bin.

To increase reliability of the method, pixels far away from the respective particles are assigned lower weights using the weighting function

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

where r is the distance from the particle. Thus, pixels far away from the particle receive lower weights to reflect the fact that they may belong to the background. The color distribution for one particle at pixel location $\mathbf{y}$ is defined as $p_{\mathbf{y}} = \{p_{\mathbf{y}}^u\}_{u=1...m}$ where u is a bin and m in this paper is $3 * 8 = 24$. The color distribution is then calculated as

$$p_{\mathbf{y}}^u = f \sum_{i=1}^{I} k(\frac{||\mathbf{y} - \mathbf{x}_i||}{a}) \delta[h(\mathbf{x}_i) - u] \tag{4}$$

where I is the number of pixels within the template, δ is the Kronecker delta function, a is used to normalize the distance from the pixel to the particle using the size of the template, and

$$f = \frac{1}{\sum_{i=1}^{I} k(\frac{||\mathbf{y} - \mathbf{x}_i||}{a})} \tag{5}$$

is a normalization constant to ensure that the color distribution sums to 1 for the particle.

3.6 Bhattacharyya distance

Once the color distribution of the particle has been calculated, we need to compare it to the distribution of the target, the face. This is done using by first calculating the Bhattacharyya coefficient as

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

where p is the color distribution of the particle and q is the color distribution of the target. Larger ρ means that the particle has a similar color distribution as the face. The Bhattacharyya distance, the distance between the distributions is then defined as

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

This is identical to the approach in [3].

3.7 Calculating weights

The likelihoods of the particles are then calculated using the respective Bhattacharyya distance, d. To favor samples with similar color distributions to the target, small distances should correspond to large weights and the distances are therefore fed into a Gaussian function with variance σ_c. These likelihoods are then normalized to acquire the final weights of the particles π. This is identical to the approach in [3].

3.8 Calculating the mean state

Once all the weights of the particles have been calculated, the mean state of the face can be estimated at each time step by

$$E[S] = \sum_{n=1}^{N} \pi^n s^n \tag{8}$$

where N is the total number of particles. This is identical to the approach in [3].

3.9 Updating the target model

As previously stated, the particle filter is initialized by creating a color histogram of the target model, the face. As the video progresses however the lighting may change as well as the scale and rotation of the face. The target model should therefore be updated throughout the video to reflect this. However, the face in any given frame may be occluded or noisy. Therefore, we should only update the target model when the filter has not lost track of the face.

The observation probability $\pi_{E[S]}$ of the mean state can be calculated using by creating the color histogram $p_{E[S_t]}$ belonging to it using equation 4, then calculating the Bhattacharyya coefficient using equation 6 and the Bhattacharyya distance using equation 7, and finally feeding this into the Gaussian described in section 3.7. If $\pi_{E[S]}$ is larger than a pre-defined threshold the update of the target model is done by:

$$q_t = (1 - \alpha) q_{t-1} + \alpha p_{E[S_t]} \tag{9}$$

where q_{t-1} is the target model in the previous timestep. By increasing α, the past model is forgotten faster and the resulting target model q_t consists more of the recent estimate of the target. This is identical to the approach in [3].

3.10 Updating the tracking template

As the size of the face may change throughout the video, for instance by coming closer to the camera, the size of the tracking template needs to be updated. If the particles are further apart in the current time step compared to the previous time step, the tracking template should be larger to reflect this. If a person approaches the camera, the face will take up a larger proportion of the frame and thus the tracking template should be larger.

The tracking template is calculated by

$$\begin{cases} s_t(x) = s_{t-1}(x) * d/d_{t-1} \\ s_t(y) = s_{t-1}(y) * d/d_{t-1} \end{cases} \tag{10}$$

where s_t is the current frame tracking template size and s_{t_1} is the previous frame tracking template size. d is the average distance between the particles and the target state center and is computed by

$$d = \frac{1}{N} \sum_{i=1}^{N} \sqrt{(x - x_i)^2 + (y - y_i)^2} \tag{11}$$

where N is the number of particles, x and y are the coordinates of the current state estimate and x_i, y_i are the coordinates of the respective particles. Every particle thus have the same tracking window size.

The size of the tracking template is only updated during stable conditions, similar to the update of the target model, using a pre-defined threshold.

The approach for updating the tracking template is similar to the approach taken in [6].

3.11 Resampling

To ensure low variance and increase the speed of the algorithm systematic resampling with replacement is applied using the weights calculated in section 3.7.

3.12 Experiments

To test the implementation, a number of videos where recorded on an iPhone 8. The videos have 30 frames per second and the particle filter is performed on every frame. 100 particles were used.

The first video contains a person moving quickly from left to right in front of a blue background. The second video contains the same person approaching the camera and then backing off in front of a blue background. The third video shows a person moving quickly from left to right in front of a red background. The fourth video shows a person moving in and out of the frame.

The noise variables in the motion model as well as the update threshold were set to reflect the situation in the current video. That is, since there were much motion x-wise but not so much y-wise, the noise in the x-direction was set to a larger value than in the y-direction.

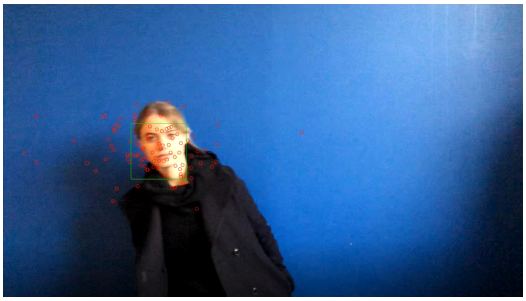
In the video with the person moving in and out of the frame, a threshold on the observation probability of the mean state needed to be set to avoid detecting a face when the person was out of the picture. This threshold was set manually.

4 Results

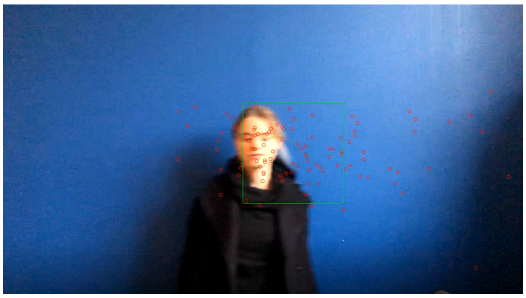
In the images with blue backgrounds the particles are shown as red circles and in the images with red background the particles are shown as blue circles. The estimate of the mean state in conjunction with the tracking template is shown as a green rectangle.

Figure 1 shows a person moving with non-constant speed in front of a blue background. In the beginning of the video, the person is stationary and the filter works well. As the person speeds up in figure 1b, the template lags slightly but still captures the face well. when the person moves very fast, the template becomes too big and captures not only the face but a large part of the background and the clothing of the person.

(a) Stationary.



(b) Fast movement.

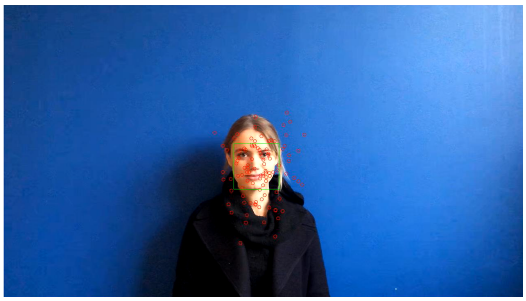


(c) Very fast movement.

Figure 1: As the speed of the person increases, the green template lags the face.

Figure 2 shows the person approaching the camera and then backing off. As the person comes closer to the camera, the tracking template size is increased and when the person moves away from the camera the tracking template size is decreased. Further, the entire face is not captured when the face is very close to the camera.

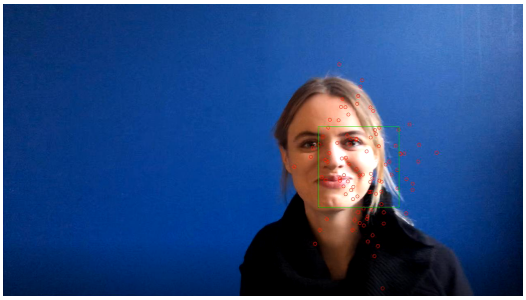
Figure 3 shows another person starting the video being stationary. As the person moves fast, the size of the tracking template is larger than the face of the person and captures a large part of the background and the jacket. As the person slows down, the template catches up.



(a) Start of the video.

(b) Moving toward the camera.

(c) Closest position to the camera.



(d) Moving away from the camera.

(e) End of the video.

Figure 2: As the distance from the face to the camera changes, so does the size of the template.

(a) Stationary

(b) Fast movement.

(c) Slowing down.

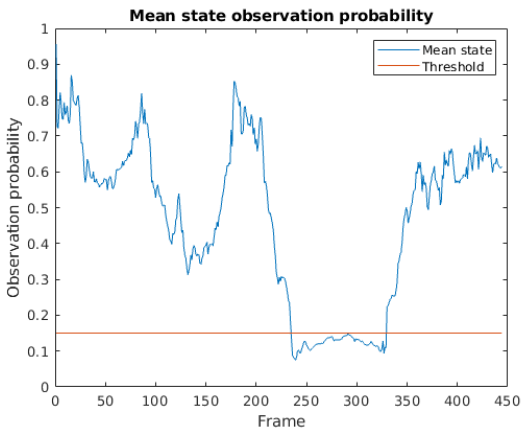
Figure 3: The green template lags the face during motion but catches up when the person slows down.

Figure 4 shows a person moving in and out of the frame. In figure 4a, the person is exiting the frame to the left. As the face of the person is outside the frame in figure 4b, the particles moves to the right of the frame. When the person re-enters the frame in figure 4c the filter correctly starts tracking the face again albeit with a too large tracking template size in the beginning. A graph showing the observation probability of the mean state in displayed in figure 4d. As the person leaves the picture around frame 230 the mean state observation probability plummets and it rises again at about frame 325. While the mean state probability is below the threshold of 0.15, a tracking template is not shown in the video. This is displayed in figure 4b, which lacks a green rectangle.

(a) Person leaving the frame.

(b) While the face is outside the frame, the particles move to the right.

(c) The face is correctly tracked as the face re-enters the frame.



(d) Graph showing the observation probability of the mean state.

Figure 4: The tracking works even if the person leaves and re-enters the frame.

5 Discussion

The face tracking algorithm generally works well, especially during slow movement. As shown in figure 3 and figure 1, it works well regardless of the color of the background. The color of the face and the background are similar in figure 3 but the algorithm still works well despite being based on solely on the color distribution of the face.

When the person moves faster, the tracking template often lags the face. Further, the tracking template may become larger than the face, as shown in figure 1c. To minimize these problems, fine tuning of the parameters of the filter is needed. Lowering σ_c, the variance of the Gaussian that gives the particles their weights, decreases the size of the template. As this gives a more narrow Gaussian function, particles with larger Bhattacharyya distance from the target distribution are assigned lower weights. To combat the lagging of the tracking template, the process noise should be increased. However, this leads to the tracking template jumping around more when the person is stationary. Thus, each video needs some fine-tuning for best performance.

As the tracking template grows larger than the face, the background color distribution is incorporated into the target model during the update of the target model. This leads to worse performance, as the target color distribution becomes similar to the background color distribution. To address this problem, *alpha* was set to 0 or 0.01. This means that during target update using equation 9, the color distribution of the current mean state contributes a minimal amount to the new target model. Essentially, the target model is static over the course of the video. When the lightning changes, this leads to problems since the face will change color but the target model stays the same.

In the video with the person walking out of the frame, displayed in figure 4, a threshold is set to stop displaying the green template when the person is out of the frame. This threshold was manually set and such a threshold needs to be set for every video. The exact threshold depends on the similarity of the color distribution of the face and the color distribution of the background. In this video the threshold was set to 0.15, however if the color distribution of the background and the face were more similar this threshold should be increased. Further, even though systematic resampling was used different seeds fed into the random number generator gave vastly different results. Thus, setting a reliable threshold is arduous.

Figure 2 shows a person approaching and moving away from the camera. As the person comes closer, the template size grows but does not cover the entire face. A higher number of particles may alleviate this problem. A larger template size leads to more computations being made, as all the pixels in the template need to be accounted for when calculating the color distribution using equation 4.

Although the time needed to process one frame using the algorithm never was calculated exactly, the algorithm cannot be used in real time unless the resolution is lowered significantly. To alleviate this, perhaps the algorithm can be executed on every 10th frame instead of every frame. However, this may lead to less accurate results as the face may move to the other side of the picture during the skipped frames.

6 Conclusion

The implementation works well in different settings. It works well even if the face and the background have similar color distributions. If the person in the video moves fast, the particle filter correctly tracks the face. By setting a threshold on the observation probability of the face, the algorithm can stop tracking the face as it disappears from the video and as the person reenters the video the algorithm correctly starts tracking the face again.

The algorithm needs to be tuned to the video for best performance. To avoid the tracking template lagging the face, the process noise should match the velocity of the face.

The algorithm cannot be used in real time with a meaningful video resolution if executed on each frame of the video. Executing the algorithm once every 10th frame and using a low video resolution may alleviate this, with a trade-off in accuracy.

To increase accuracy of the algorithm, edge based features could be used. Since the algorithm currently only takes color into account, this could make the filter work even better if the color of the face and background are particularly similar.

References

[1] Apple. Tracking and visualizing faces. `https://developer.apple.com/documentation/arkit/tracking_and_visualizing_faces`. Accessed: 2019-12-28.

[2] Facebook. What is the face recognition setting on facebook and how does it work? `https://www.facebook.com/help/122175507864081`. Accessed: 2019-12-28.

[3] Katja Nummiaro, Esther Koller-Meier, and Luc

Van Gool. An adaptive color-based particle filter. *Image and vision computing*, 21(1):99–110, 2003.

[4] pyimagesearch. Opencv face recognition. `https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/`. Accessed: 2019-12-28.

[5] Paul Viola, Michael Jones, et al. Robust real-time object detection. *International journal of computer vision*, 4(34-47):4, 2001.

[6] Tao Wang, Wen Wang, Hui Liu, and Tianping Li. Research on a face real-time tracking algorithm based on particle filter multi-feature fusion. *Sensors*, 19(5):1245, 2019.