# Face Tracking using a Particle Filter with Color Features and a Self-Updating Window

**Alexandra Hotti**

## Abstract

In this report, an algorithm for face tracking in consecutive video image sequences was successfully implemented and evaluated. Face tracking within videos has a wide area of applications from being used in surveillance systems to video games.

The tracking in this report was achieved by using a particle filter with color features and a self-updating tracking window. Color-based characteristics are vigorous at handling rotational variations, changes in the size of the object being tracked and when the face being tracked disappears and reapers from the screen. Additionally, the self-updating tracking window used is able to handle significant face scale changes.

## 1. Introduction

In this project, a method for tracking faces between image sequences was implemented.

Face tracking over video frames is used within several different computer vision applications such as video games (Tung & Matsuyama, 2008), real-time surveillance (Segen, 1996) and applications for human-computer interaction to recognize facial expressions(Black & Jepson, 1998).

Currently, there are two popular approaches for solving this problem. One approach using the Mean Shift algorithm (Cheng, 1995) and another particle-based solution (Gordon et al., 1995).

In this project a particle filter approach based on color image features partially based on Nummiaro et al. (2003) was used. However, instead of including the window that surrounds the face as part of the state, a self-updating tracking window was used in accordance with Wang et al. (2019).

### 1.1. Problem Statement

The purpose of this project is to successfully implement a face tracking algorithm based on a color model. This algorithm will have a self-updating tracking window (Wang et al., 2019).

To qualitatively asses that the tracking algorithm worked correctly we have applied it to different video sequences under varying conditions. We used video sequences with varying lighting conditions and different background colors. We also tested whether the tracking algorithm could handle short term occlusion as well as a face moving with considerable speed. Lastly, we experimented to see that our implementation of the self-updating tracking window described by Wang et al. (2019) worked correctly.

### 1.2. Contribution

The following contributions were made by this project:

- The algorithm is able to keep track of the face when the person in the image moves rapidly.
- The tracking algorithm works even when the background is similar in color to the faces being tracked.
- The tracking can continue after occlusion, i.e. when the person disappears and reapers from the screen.
- The algorithm is able to handle scale changes. As the person moves either closer or further away the bounding box around the person's face is resized.
- The initialization of the algorithm is automated. By using the Viola-Jones Algorithm no manual human input is needed to initialize the tracking algorithm.

### 1.3. Outline

In section 2 the previous work that our project is based on is briefly depicted. Next in in sections 3.1 - 3.5 the theory behind our method is described and are then put into a logical order in which the theory was implemented in section 3.6. Then in section 4 the results from four experiments are presented and lastly in section 5 a summary and a conclusion regarding the results are presented.

## 2. Related work

As mentioned in section 1 this project uses a color-based particle filter with a self-updating tracking window to trace

faces in image sequences. This approach is mainly based on three previous studies.

Firstly, to find the initial face location for the particle face tracker the Viola-Jones object detection framework proposed by Viola et al. (2001) was used. This algorithm detects faces based on symmetry properties in human faces using so-called Haar Features.

The face tracking is achieved by using a color-based particle filter described by Nummiaro et al. (2003). The object tracking is achieved by propagating the particles and then measuring the color similarities between the object being tracked and the color in the neighborhoods of the propagated particles. According to Nummiaro et al. (2003), the benefits of using color for tracing objects is resilience against objects disappearing from the screen, i.e. occlusion, objects changing pose relative to the camera, i.e. rotation, scale invariance when the object being tracked changes size by moving closer or further away from the camera and lastly computational efficiency.

However, unlike Nummiaro et al. (2003) the window size and scale of the face being tracked is not included in the state, instead a self-updating tracking window as described by Wang et al. (2019) was implemented. This approach compares the average distance between particles and the tracked object between the current and the previous time step. When the average distance increases compared to the previous time step the window size is increased and when the average distance to the particle center decreases the window is scaled down. According to Wang et al. (2019) this approach improves the tracking algorithms abilities to adjust the face window size when the size of the object substantially changes throughout the video.

# 3. Method

## 3.1. Particle Filter

The tracking algorithm used to track a face in an image sequence is based on particle filtering. An face being tracked is represented by a target state vector $X_t$ at a specific time step $t$. The vector $Z_t$ represents a sequence of observation video frames up to time t: $\{z_1, z_2, .., z_t\}$. The objective of the particle filter is to estimate the posterior density $p(X_t|Z_t)$ (Nummiaro et al., 2003).

The distribution is estimated by a particle set $S = \{(s^m, \pi^m | m = 1, .., M)\}$ were $M$ is the number of particles in the set. Each particle $s^m$ is a hypothetical state of the face being tracked and has an individual sampling probability $\pi^m$ called a weight. Since this is a valid probability distribution all of the weights must be non negative and sum to 1 (Nummiaro et al., 2003). Meaning that:

$$\sum_{m=1}^{M} \pi^m = 1 \tag{1}$$

To estimate a new state, i.e. to estimate the location of the face being tracked in the next video frame, when the particles have been propagated, a weighted mean of the particles x and y coordinates are used to calculate a mean state (Nummiaro et al., 2003).

$$\mathbb{E}[S] = \sum_{m=1}^{M} \pi^m \cdot s^m \tag{2}$$

This will give us a mean center location of all the particles.

A benefit of tracking with a particle filter is that since we have several potential hypotheses at each time step, the algorithm can keep somewhat unlikely face states for a short period. This means that we have a way to deal with occlusion, were the face disappears and reapers from the video (Nummiaro et al., 2003).

## 3.2. Model

### 3.2.1. The state representation

As mentioned above, each particle at a certain time steps represents a hypothetical location of the face being tracked at a specific time step. Every particle has a location in the image denoted as: $x$,$y$ which represents the center of a rectangular box that encompasses the face being tracked. The center location of every particle also has an individual velocity: $\dot{x}$, $\dot{y}$. Depending on the direction of the motion in the image the velocities could be either positive or negative. Thus, to summarise the face being tracked will have 4 components in its state $s$:

$$s = \{x, y, \dot{x}, \dot{y}\} \tag{3}$$

A difference in our state representation compared to the one used by Nummiaro et al. (2003) is that we do not include the bounding box height, width and scale change at each time step. When these parameters are included in the state it means that they can be updated at each time step in the dynamic model see 3.2.2. Instead, we update the window width and height by using a method described by Wang et al. (2019). How this is achieved is described in section 3.5.

### 3.2.2. Dynamic model

To predict the target state at the next time step, i.e. where the face will end up in the next video frame we use a dynamical model. This model predicts the next state location and velocity by propagating all of the particles according to eq. 4.

$$s_t = A \cdot s_{t-1} + G \tag{4}$$

Where G is Gaussian noise and A is a deterministic component of the model that looks like this:

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

All in all, this means that the locations and velocities of the particles will be propagated in the following manner:

$$\begin{aligned} dx_t &= dx_{t-1} + G \\ dy_t &= dy_{t-1} + G \\ x_t &= x_{t-1} + dx_t + G \\ y_t &= y_{t-1} + dy_t + G \end{aligned} \tag{6}$$

### 3.3. Likelihood Model

Given that we have propagated the particles and now have a new predicted state we want to estimate the likelihood of each particle location corresponding to the face location. These likelihoods can then be normalized and used as particle weights $\pi^m$ as described in section 3.1.

Conceptually, the likelihoods are obtained by measuring how similar the color distribution of each particles bounding box is to the mean state color distribution from the previous time step (Nummiaro et al., 2003). How this is done is outlined below.

#### 3.3.1. Color Histograms

In particle filtering, observation likelihoods are based on features. In this project, the observations correspond to the colors within the bounding boxes of each propagated particle. To represent the observations one individual color histogram is created for every single propagated particle at each time step based on the colors within the particles individual bounding boxes (Nummiaro et al., 2003).

The color distributions are created in an RGB-space where each color dimension is discredited into 8 bins. To make the system faster each RGB dimension is treated independently when the histogram is calculated, such that the histogram for 1 particle gets $8 \cdot 3 = 24$ bins in total. The downside of this is that for instance, a purple image consisting of both red and a blue could get the same color histogram as an image where the blue and red are separated into different regions. The upside is that we get 24 computations for each particle instead of $8^3 = 512$ computations.

When a pixel is located near the edges of a bounding box there is a larger risk that the pixel is part of the background or is occluded. Therefore we want to weight the histogram such that the colors of pixels close to the center get large weights and boundary pixels get small weights. This is achieved by the following weighting function (Nummiaro et al., 2003)

$$k(r) = \begin{cases} 1 - r^2 & r < 1 \\ 0 & otherwise \end{cases} \tag{7}$$

here $r$ is the distance from a pixel $x_i$ to that particular particles bounding box center (Nummiaro et al., 2003).

Now we have all the information we need to define the color histogram. The color distribution for a particular particle is built by filling all of the $8x8x8$ bins in the RGB space. To fill one bin $b$ for one particle $m$ with center location $m_{x,y}$ the following formula is used

$$p_m^{(b)} = f \sum_{i=1}^{I} k(\frac{||m_{x,y} - x_i||}{a}) \delta(h(x_i) - b) \tag{8}$$

where $I$ is the number of pixels in the bounding box, $x_i$ represents a specific pixel and $||m_{x,y} - x_i||$ gives us the distance between the bounding box center and the pixel. The function $h(x_i)$ assigns the color at the pixel location $x_i$ to its corresponding color bin. When either of the RGB-components of the pixel matches the current bin $b$ the dirac function will be equal to 1, otherwise it will be 0. Then, to get a valid distribution $f$ normalizes the histogram, thus $f = \frac{1}{\sum_{i=1}^{I} k(\frac{||m_{x,y} - x_i||}{a})}$. Lastly, the $a$ parameter adapts the region size so that we get an equivalent weighting no matter the size of the bounding box. Meaning that we will not give larger weights to pixels close to the borders of a small bounding box, compared to a large bounding box, i.e. when the face is far away vs close to the camera. We used this setting for the $a$ parameter $a = \sqrt{H_x^2 + H_y^2}$.

#### 3.3.2. Distribution similarities - The Bhattacharyya distance

Once we have computed one color histogram per particle we would like to compare each particle's distribution to the mean state color distribution from the previous time step. This will give us a sense for how similar each propagated particles rectangular bounding box is to the mean rectangle of the previous time step where it is assumed the mean state covers the face (Nummiaro et al., 2003).

The comparison is achieved by using the Bhattacharyya distance $d$, which measures the similarity between two probability distributions $p(b)$ and $q(b)$ (Nummiaro et al., 2003). First, we calculate the Bhattacharyya coefficient $\rho$

$$\rho[p,q] = \sum_{b=1}^{\#bins} \sqrt{p^{(b)} \cdot q^{(b)}} \tag{9}$$

$p$ is the particle color distributions and $q$ is the color distribution of the mean state from the previous time step. When $\rho = 1$ is the maximum value of the coefficient and this value indicates that the two distributions are identical (Nummiaro et al., 2003).

Next, we calculate the calculate the actual Bhattacharyya distance between the distributions

$$d = \sqrt{1 - \rho[p,q]} \tag{10}$$

#### 3.3.3. The face color likelihoods

Now that we have a measure for how similar the individual particles colors are to the estimated faces colors we want to update the particles weights based on these observations. For every particle $m$ we compute a weight by using a Gaussian function with a variance $\sigma$.

$$\pi^{(m)} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{d^2}{2\sigma^2}} \tag{11}$$

Note that the weights are normalized such that $\sum_m^M \pi^{(m)} = 1$. Here, $d$ is the Bhattacharyya distance defined in eq. 10. Smaller values for $d$ means that the colors surrounding a particle are more similar to the colors of the estimated state of the face. Thus, a small $d$ value will result in a large weight $\pi^{(m)}$ and vice versa (Nummiaro et al., 2003). A large weight means that the particles will have a larger chance of being sampled in the resampling step of the particle filter, see step 5 in 3.6. Some particles will have several decedents while some will not be resampled at all and will thus have a lineage that dies out.

The $\sigma$ value specifies how much weight we give to particles in relation to their $d$ value. Such that if we use a large $\sigma$ value we are less strict about how similar the face coloring of a specific particles bounding box is to the mean face state coloring. Meaning that we give larger weights to particles with larger distances as we increase $\sigma$.

Thus, using a large $\sigma$ value results in a particle filter that is better at handling for instance changes in lighting conditions. However, this could potentially result in particle degeneracy as we could potentially get high enough likelihoods at other regions then the face. If we use a to small sigma value the risk is that the particles will not spread out enough around the face and as the person moves around we lose the target.

### 3.4. Mean state update

Once we have updated the weights of the propagated particles it is time to estimate the current face location. This is done by using eq. 2. Such that the center of the face is estimated to be the weighted mean of all the particles locations in the image (Nummiaro et al., 2003).

We also want to compute the color distribution of this mean state. This distribution is referred to as $q$ and is used in eq. 9 to compute the Bhattacharyya distance $d$. It is important to update $q$ since lighting conditions and the angle of the persons face relative to the camera could change during the course of the video. $q$ is updated according to (Nummiaro et al., 2003)

$$q_t = (1 - \alpha) q_{t-1} + \alpha p_{mean} \tag{12}$$

here $\alpha$ defines how much we want to incorporate of the new reference state color distribution into $q_t$ and how much we want to keep of historic mean state color histograms. Which means that the influence of old state decreases as the tracking evolves (Nummiaro et al., 2003).

To improve the tracking algorithm we also include a condition for how likely a mean state has to be for us to update the reference distribution $q_t$. Sometimes we will get states that could be considered outliers because of frames containing large amounts of noise or short term occlusion where the person disappears from the frame. This would result in a low probability of the mean state. To mitigate the effects that these improbable states could have on our reference distribution $q_t$, a $p_{mean}$ value is only included in the update of $q_t$ if it is larger than a predetermined threshold (Nummiaro et al., 2003).

### 3.5. Self updating Bounding Box

Finally, we use a method described by Wang et al. (2019) to update the size of the bounding box surrounding the face

$$\begin{cases} w_t = w_{t-1} \cdot \frac{d}{d_l} \\ h_t = h_{t-1} \cdot \frac{d}{d_l} \end{cases} \tag{13}$$

here d is the average distance between the particles and the mean state center. d is defined as

$$d = \frac{1}{M} \sum_{i=1}^{M} \sqrt{(x - x_i)^2 + (y - y_i)^2} \tag{14}$$

While $d_l$ is the average distance from the previous frame to the mean state center. Meaning that we want to scale the window width and height by how much the average distance to the target center has changed from the previous to the current video frame (Wang et al., 2019).

### 3.6. The Tracking Algorithm

Here the steps in the tracking algorithm are presented in the order in which they were implemented in our code.

The algorithm is run for each frame $t$ in a sequence of frames in a video.

**step 0. Initializing Algorithm, If $t = 0$**

0.1. At $t = 0$, the system is initialized. First the parameters of the initial state $x_0, y_0, \dot{x}, \dot{y}$ are set. The initial velocities: $\dot{x}, \dot{y}$ are set to 0 and the center of the initial face bounding box: $x_0, y_0$, is found by using the Viola-Jones algorithm (Viola et al., 2001).

0.2. The Viola-Jones algorithm also returns the initial width and height of the bounding box surrounding the face at the first time step. This bounding box should be able to fit the face being tracked.

0.3. Next, the particle filters' $M$ particles are initialized uniformly within the bounding box found in the previous step. The particles are given a uniform weight of: $1/M$ and zero velocities.

0.4. Equation 13 requires an average distance between the particles and the bounding box center $d$ both for the current $d_t$ and the previous $d_{t-1}$ time step. Therefore, the variable $d_0$ is initialized by calculating the average distance between the initial center $x_0$ and $y_0$ found by the Viola-Jones algorithm above and the initialized particles.

0.5. The color histogram of the image area within the initial face bounding box is calculated using algorithm x. This gives us $q_C$, which is used as a target histogram in equation 8. The target histogram is used to compare the histogram which we get after propagating the particles with the color histogram of the previous time step.

**step 1. Updating the bounding box, If $t \geq 1$**

1.1. The face's current bounding box is predicted using equation 13 and the average distance between the particles and the bounding box center $d_t$ is calculated via eq. 14. This is computed based on the current bounding box, $d_{t-1}$ and the particles.

**step 2. Propagation:** Propagate the particles according to equation 4 and 6.

**step 3. Observing the Particle Color Distributions & Updating the Weights:**

3.1 Calculate a color histogram for each particle based on the area within each particle's bounding box. The particle color histograms are calculated using equation 8.

3.2 **The Bhattacharyya coefficient:** Next, the Bhattacharyya coefficient is calculated for each particle using each particle color histogram and the target state $q$'s color histogram, see eq. 9. Next, the Bhattacharyya distance is calculated based on this coefficient using eq. 10

3.3 **Calculate the Particle Weights:** Now the particle weights are calculated using equation 11 and the Bhattacharyya distances. This equation will give us a face color $y_c$ likelihood for each particle $x$: $p(y_c|x)$. To get a valid distribution for the particle weights the likelihoods are also normalized.

**step 4. Estimate target state:** Now, we calculate the next state. Based on all of the particles and their weights we calculate a weighted mean state, see eq.2. This gives us a mean bounding box center location $x_t$, $y_t$ of all the particles.

**step 5. Systematic Resampling:** Next, new particles are sampled based on the updated weights using systematic resampling. The resampled particles are given uniform weights $1/M$.

**step 6. Update histogram of target state:** The histogram of the bounding box surrounding the new mean state is calculated using eq. 8. This histogram is used as $q$ in the next lap of the algorithm so that we can compare the colors if the new propagated particles in the next iteration with the current mean bounding box.

### 3.7. Implementation

All of the code was implemented in Matlab.

The initial face location which was found with the Viola-Jones algorithm was implemented with a Matlab package called $vision.CascadeObjectDetector$ (Mathworks, 2012).

## 4. Experimental results

As mentioned in the problem statement, to qualitatively asses that the tracking algorithm works correctly we have applied it to different video sequences under varying conditions. We use video sequences with varying lighting conditions and different background colors. We also test if the tracking algorithm can handle short term occlusion as well as a face moving with considerable speed. Lastly, we experiment to see that our implementation of the self-updating tracking window described by Wang et al. (2019) works correctly.

In all of the images the small scattered blue and red circles are the particles at the current time step and the green boxes are the estimated bounding boxes of the current state.

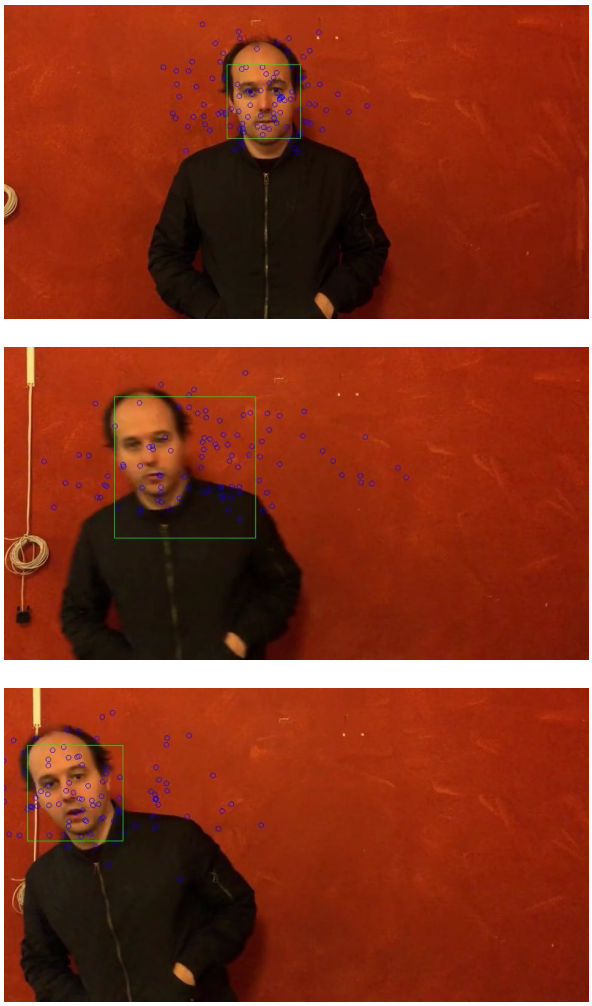

*Figure 1.* Tracking a persons face when he moves quickly to the left. The top image is the first image in a sequence and the bottom image is the last. When he stands still in the first image we get an accurate, tight bounding box. When he starts moving quickly the size of the bounding box increases and as he slows down again the bounding box shrinks in the last image. An extra element of difficulty in this video is the red background which resembles the faces red RGB component.

### 4.1. Red Background with Fast Movement and Even Lighting

First we applied the tracking algorithm to a video where the face is moving quite quickly and the background is red, see Figure 1. Faster movement is more difficult to track compared to slow movements. Also, the red background has more similar RGB values to the beige skin color compared to the blue background which adds to the complexity of tracking this sequence since color histograms of the background becomes more similar to the face.

Something that abbreviate the tracking is that the lighting on the persons face is quite even throughout the video. As a comparison, see the lighting in for instance Figure 2.

When the person stands still the algorithm can easily a bounding box that accurately and tightly fits the person's face. See the first image in Figure 1.

As the person starts to quickly move to the left in the image with a slightly tilted face the bounding box is able to keep track of the face. However, keeping track of the fast face comes at the cost of having a tight bounding box. In the second image in Figure 1 the bounding box has become large as the person moves around quickly. There seems to be a trade off between keeping the face within the bounding box and having a tight bounding box. To model fast motion we need to add large sampled noise to $dx_t$ and $dy_t$ in 6 and if we want to allow for fast motion in one direction we also need to allow for fast motion in the opposite direction. Thus, to keep the face within the bounding box when the person moves quickly to the left we allow particles to quickly move to the right. The result is a large bounding box that keeps the face within the box.

In the third image in Figure 1 the person has again slowed down and the bounding box has shrinked around the face.

### 4.2. Blue Background with Fast Movement and Uneven Lighting

Next, we applied the tracking algorithm to a video sequence where the face is moving very fast at times, the background color is dissimilar to the face and the lighting conditions varies in the room see Figure 2.

In the first frame to the left, we get a tight bounding box around the face. However, as the person moves very quickly in the middle image again we experience the same types of issues described in section 4.1 where the bounding box becomes large. In the last image, the person is not moving quite as rapidly and we get a tighter bounding box.

An added difficulty in this sequence is that the lighting conditions vary across the room. Meaning that the color distribution of the face changes as the person moves around. To get around this issue one could increase the $\alpha$ value in eq. 12 so that we give more weight to newer frames when we update the target reference color histogram. Meaning that we give more weight to local lighting conditions and coloring.

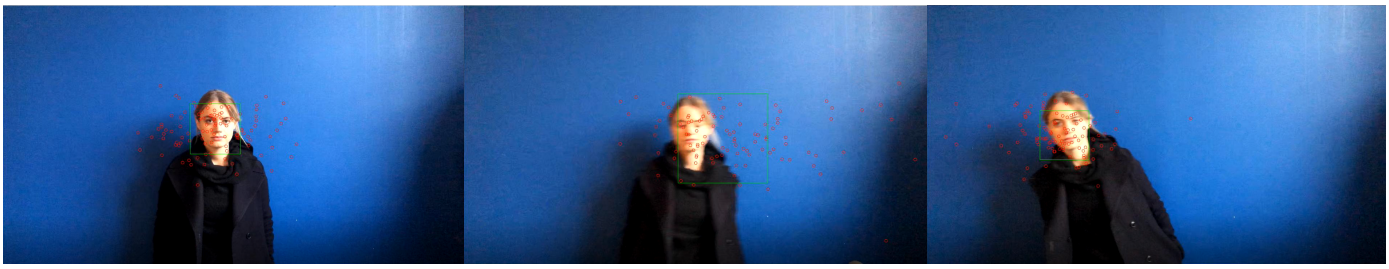

*Figure 2.* Tracking a persons face when she moves quickly to the left. In the first image from the left the person stands still and we get a tight bounding box. In the next image the person moves rapidly to the left and the bounding box has difficulties with covering the face since the lighting varies. Since the face moves quickly the box becomes quite large. In the last image the person is moving but not quite as fast and the algorithm has an easier time tracking the face. An added complexity is that the lighting conditions varies across the screen.

### 4.3. Occlusion

Our next experiment focused on occlusion. We wanted to make sure that the face could disappear and reaper in the image and still be accurately tracked by the algorithm. The results for this section can be found in Figure 3 and 4

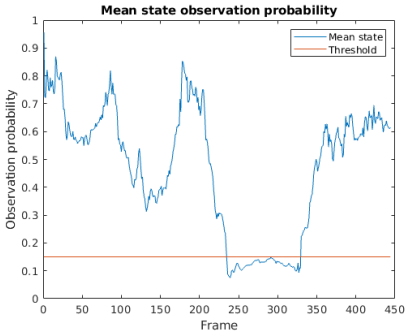


*Figure 3.* The evolution of the observation likelihood of the estimated mean state throughout the video sequence in Figure 4. One can see how occlusion lowers the observation likelihood and when the face reapers it increases again. We can also see the likelihood threshold that we used (the red line) to signify when we did not update the target reference state $q$.

*Figure 4.* Here we make sure that the tracking algorithm can handle short periods of occlusion. In the fast image at the top, the person is moving towards the image boundary. In the next image, his face is completely out of the screen and in the last image he has reappeared.

In Figure 4 we can see in the top image that the person is moving towards the image bounds.

In the second image, the person has moved out of the image and the particles are moving towards the lighter blue regions which mostly resembles the paleness of the face in color. However, some particles still populate regions close to the person. Also, we can see how the observation likelihood of the mean state does a nosedive around the 250:th video frame Figure 3 when the face disappears from the screen.

Here the observation likelihood went below the threshold (the red line) and thus we did not update the target reference color histogram called $q$, see section 12.

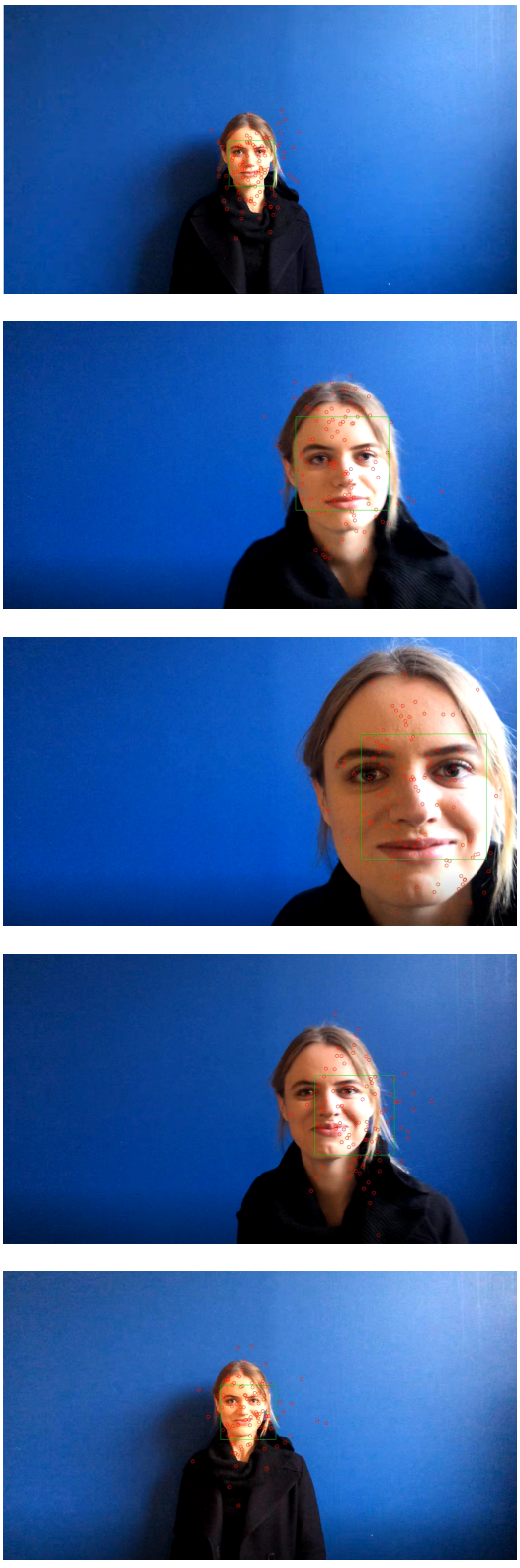

*Figure 5.* Testing the effects of changing the face size by moving towards and away from the camera. The sequence starts from the top image and ends with the bottom image. As the person moves from and towards the camera the self-updating tracking window changes the bounding box scale to fit the face. However, it seems as if the lighting conditions is complicating the process.

Thus, in the bottom frame, when the person's face has reappeared we still have a target reference histogram of the face that resembles the face colors before the person disappeared from the screen. If we had not stopped updating $q$ we would have updated it based on the blue colors in the background. Meaning that we likely would not be able to find the face when the person reappeared. Instead, we would likely start tracking the blue pixels of the background instead.

Now as the person comes into the image we have a reference target that still resembles the face coloring. Therefore, the few particles that are still close to his face likely get very large weights and are resampled several times such that most resampled particles end up around the face. Now we can see how the observation likelihood of the mean state rapidly increases again in the graph.

### 4.4. Self-Updating Tracking Window

Our last experiment focused on testing that the self-updating tracking window described by Wang et al. (2019) worked accurately. These results can be found in Figure 5. The image sequence starts from the top image and ends at the bottom image. Meaning that the person moves towards the camera and then backs away. We can see in the second image that the bounding box increases.

In the second image the bounding box does indeed increase. In the third image the rectangle also increase, however it is no longer able to cover the left side of the face that well. The reason for this could be that the lighting conditions have changed between the frames and there is more shadow on the face in the third image. Again these effects could be mitigated by increasing the $\alpha$ value in eq. 12 as described in section 4.2.

In the fourth and fifth image from the top the person is moving towards the wall and as she moves backwards the size of the bounding box shrinks.

## 5. Summary and Conclusions

Through our experiments, we have put our tracking algorithm through different image conditions. First, the algorithm is able to handle a red background that is more similar in color to the faces being tracked. Furthermore, keeping the bounding box on the target is easier when the lighting conditions are even and the face moves slowly. Our tracking algorithm can handle fast movement. However, when the face moves very quickly we need to add noise to our motion model to keep the face within the bounding box, which comes at the cost of having a tight bounding box.

Moreover, our algorithm can handle occlusion since we stop updating our reference target $q$ when the observation drops below a certain threshold.

Furthermore, the algorithm is able to adapt the size of the bounding box as the face moves closer and further away from the camera. However, the lighting conditions in our experiment seems to have impacted the results negatively.

Lastly, future experiments should focus on trying to mitigate the effects of varying lighting conditions. When the lighting changes in the room we need to give more weight to local coloring when we update our reference target $q$.

## References

Black, M. and Jepson, A. A probabilistic framework for matching temporal trajectories: Condensation-based recognition of gestures and expressions. *ECCV*, 11 1998. doi: 10.1007/BFb0055712.

Cheng, Y. Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8):790–799, 1995.

Gordon, N., Salmond, D., and Ewing, C. Bayesian state estimation for tracking and guidance using the bootstrap filter. *Journal of Guidance, Control, and Dynamics*, 18 (6):1434–1443, 1995.

Mathworks. vision.cascadeobjectdetector. https://se.mathworks.com/help/vision/ref/vision.cascadeobjectdetector-system-object.html, 2012. Accessed: 2019-12-18.

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

Segen, J. A camera-based system for tracking people in real time. In *Proceedings of 13th International Conference on Pattern Recognition*, volume 3, pp. 63–67. IEEE, 1996.

Tung, T. and Matsuyama, T. Human motion tracking using a color-based particle filter driven by optical flow. 2008.

Viola, P., Jones, M., et al. Robust real-time object detection. *International journal of computer vision*, 4(34-47): 4, 2001.

Wang, T., Wang, W., Liu, H., and Li, T. Research on a face real-time tracking algorithm based on particle filter multi-feature fusion. *Sensors*, 19(5):1245, 2019.