Face Tracking using a Particle Filter

Preston Khiev
EE528: State Space Tracking Final Project
Portland State University
Portland, Oregon
pkhiev@pdx.edu

***Abstract*— A face tracker was implemented using a Haar Cascade Classifier for initial face detection and a particle filter algorithm for tracking the movement of the face throughout a video stream. The tracker appears to accurately perform the first state update, however the program crashes shortly after due to bugs.**

I. Introduction

Video face tracking software detects and tracks the presence of a human face across digital video frames, and can be useful in a number of computer and mobile applications. Depending on the specific application, face tracking algorithms can be implemented offline and online.

Face tracking can serve as a basis for more advanced applications such as: following a specific face as it moves within a video stream, count the number of people in a video stream, determine the direction the face is looking, and recognize facial expressions. Face tracking can also give greater accuracy to various online facial recognition technologies.

When working with video data, face tracking offers many advantages compared to relying purely on facial detection for each video frame [1]. Facial detection requires a form of classification and localization to identify a face and pinpoint its location in a frame. This can be computationally expensive if performed in each frame of a video. Depending on the specific application, it may not be strictly necessary to classify and locate a face for each frame. Detection can also fail for a given video frame if a face becomes occluded by an object, or becomes subject to keystone distortion, motion blur, lighting change, etc.

After an initial detection, face tracking can remedy the aforementioned issues with pure face detection by using a state space tracking algorithm to leverage the spatio-temporal properties of the face as it moves across video frames. The objective of this project is to implement a particle filter face tracking algorithm that is more accurate and less computationally expensive than pure face detection on video data.

II. Background

a. Face Detection with Haar Cascade

An image (or video frame) data is represented as a 2D array of pixels where the elements correspond to the x and y coordinates of each pixel. Color for each individual can be represented numerous ways depending on the desired color space representation. For this project, Grayscale, BGR(or RGB), and HSV color spaces were used. Grayscale color space pixels have a single color channel that represent pixel intensities. BGR pixels has 3 channels to represent the intensity of blue, green and red respectively. HSV pixels have 3 channels to represent the hue, saturation, and value respectively. Hue is the color of the pixel, saturation is the strength of the hue, and value is the overall intensity or lightness of the color represented by the pixel.

An image feature is a piece of information about the whether a certain region of the image has certain properties. In computer vision, features provide information for solving a computational task related to a certain application. Haar features are determined by summing the grayscale pixel intensities on adjacent rectangular regions at a specific location in a detection window. By calculating the difference of these sums, Haar features are detected in regions of the image where there is a sudden change in pixel intensities (i.e., where there is a large difference in intensities between two adjacent rectangular regions. Figure 1 shows examples of Haar features in an image.

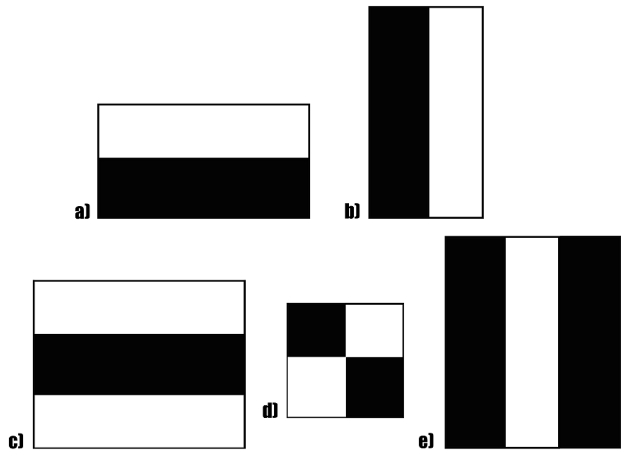

Figure 1. Examples of Haar Features from [2].

To perform face detection, the Haar features are first calculated and then passed to a trained Haar Cascade Classifier. A set of features is classified as face or not face by the Adaboost machine learning algorithm which utilizes a “cascade” of weak classifiers. The final output of the cascade determines if the set of Haar features represents a human face[3].

b. Particle Filter

The state of a stochastic system describes some parameters of the system that we wish to track. The true state of the system at any given sample is not known. Instead, an estimate of the state at each sample is calculated utilizing a noisy process model and measurement model with a set of noisy measurements. In particle filtering, a group of particles are generated to represent the posterior distribution of the state[4]. Each particle represents an estimate of the current state of the system. Particles are weighted according to the posterior distribution of the true state given a set of noisy and/or partial observations.

While tracking, some particles may have a very low weight which reduces the effective number of particles for tracking. To combat this, resampling is implemented. Particles are resampled with respect to the existing distribution so that highly weighted particles have a higher chance of being selected.

With each measurement, the weights of each particle are updated according to likelihood that the state of the particle is the true state of the system. In this project, particles are used during tracking to represent the most likely location of the human face as it moves across frames.

III. Method

This method of tracking was inspired by a particle filter tracking algorithm described in [4]. Face tracking was performed on live video data captured at 30 frames per second from a laptop webcam.

a. State Space Model

The state space model for this project is as follows:
Process Model:

$$p_{n+1} = p_n + u_n \text{(eq. 1)}$$

$$p_{n+1} = \begin{bmatrix} x_n \\ y_n \\ w_n \\ h_n \end{bmatrix} + \begin{bmatrix} u_{x_n} \\ u_{y_n} \\ u_{w_n} \\ u_{h_n} \end{bmatrix} \text{(eq. 2)}$$

Where p is a state vector representation of a rectangular region around a human face in a video frame. x_n and y_n represent the pixel coordinates of the top left corner of the rectangle. w_n and h_n are the width and height of the rectangle respectively. $u_{x_n} \sim \mathcal{N}(0, (0.2w_n)^2)$ and $u_{y_n} \sim \mathcal{N}(0, (0.2h_n)^2)$. For this project, the width and height of the rectangle does not change, so $u_{w_n} = u_{h_n} = 0$

Measurement Model:

$$z_n = p_n + v_n \text{ (eq. 3)}$$

$$z_n = \begin{bmatrix} x_n \\ y_n \\ w_n \\ h_n \end{bmatrix} + \begin{bmatrix} v_{x_n} \\ v_{y_n} \\ v_{w_n} \\ v_{h_n} \end{bmatrix} \text{ (eq. 4)}$$

Where $(v_{x_n}, v_{y_n}) \sim \mathcal{N}(0,1)$. Since the dimensions of the box do not change for this project, $v_{w_n} = v_{h_n} = 0$

b. Initial face detection

The initial state is determined by face detection. Face detection was performed using the cascaded Haar-feature detector and the "haarcascade_frontalface_default.xml" trained classifier from the OpenCV library. This detector detects and localizes the presence of a front facing human face in a frame, returning a pair of coordinates representing a rectangular frame around the face. The initial state was set so that x_0 and y_0 were the coordinates of the upper left corner of the rectangle, and w_0 and h_0 were the width and height of the rectangle respectively. When a face is detected, the pixels from the rectangular region around the face is extracted. Each pixel was classified as either skin or not skin. Skin pixels were defined in the HSV color space as H > 245 or H < 25. These hue values correspond to a "reddish" color. If the rectangular region contains less than 30% skin pixels, the face is eliminated as a false positive.

For the face rectangle, a histogram of pixel intensities was calculated for each of the 3 color channels in the BGR color space using the compareHist function in OpenCv. The histogram has 64 bins and ranges from 0 to 255 (the range of pixel intensities. These 3 color intensity histograms were stored for comparison during tracking.

c. Particle Filter Tracking Algorithm

100 particles were initialized to the initial state as determined by the initial face detection in part b. For each frame after the initial detection, the estimated state was tracked as follows:

The particle states were updated according to the process model. For each particle, a rectangular face region was extracted, and 3 histograms were calculated in the BGR color space similarly to what was done during the initial face detection. For each particle, the correlation was calculated between each of the 3 color histograms and the respective color histograms from the initial detection. These correlations were denoted bCorr, gCorr, and rCorr. A similarity score for the particle was calculated as follows:

$$similarity = 0.3 * bCorr + 0.3 * gCorr + 0.4 * rCorr$$

(eq. 5)

rCorr was given a slightly higher weight because faces should contain mostly redish colors. The weight of the particle was calculated by an exponential decay formula to ensure that particle weights will drop off exponentially as similarity decreases:

$$weight = e^{-\alpha*(1-similarity)} \text{ (eq. 6)}$$

Where $\alpha = 16$. The weights were normalized. Lastly, the estimated face state was updated by calculating the weighted average state of all the particles. Resampling was performed after each state update with samples drawn with probability according to the particle weights.

IV. Results

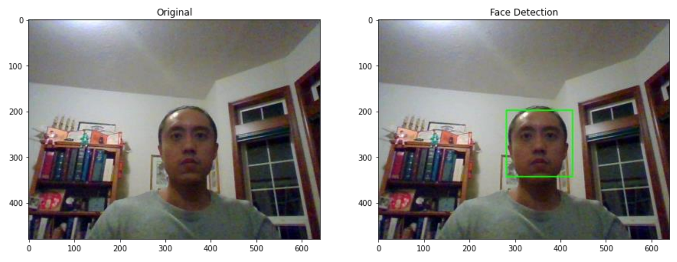

Figure 2. Frame0: Initial face detection using Haar Cascade Classifier. Green rectangle denotes a detected face. This is the initial state.

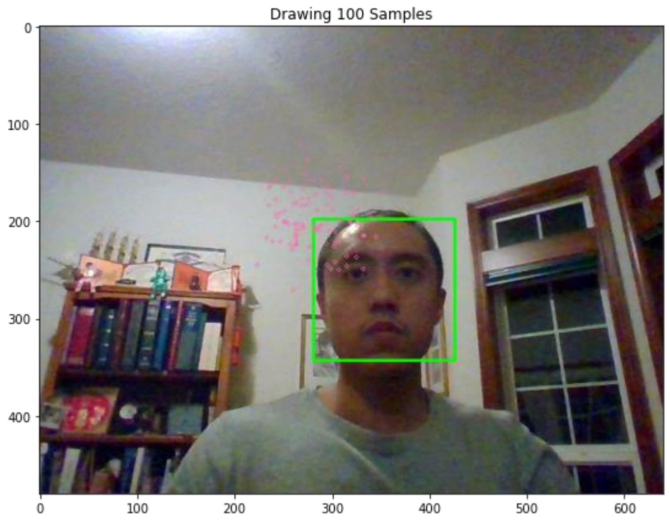

Figure 3. Frame 0: Particle sampling. Initial face detection is the green rectangle, and the x and y coordinates of each particle is represented by a pink circle. Note that the x and y coordinates correspond to the top left corner of the new estimated face rectangle. 100 samples were drawn according to the process model and the initial state.

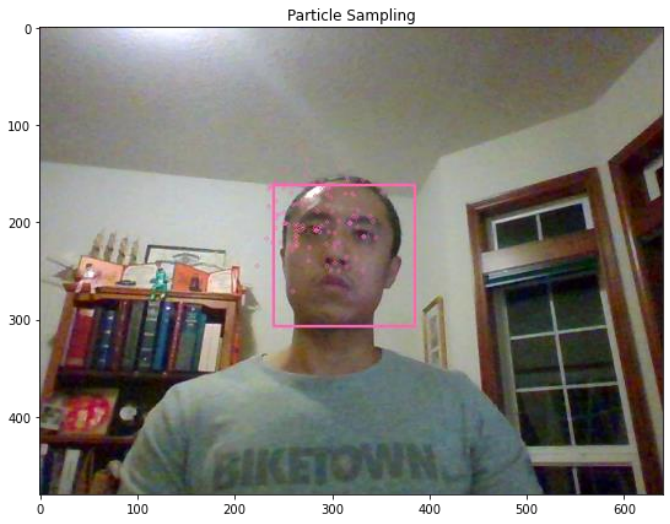

Figure 4. Frame 1: Image of Estimated State Update. The updated x and y coordinates of the new face rectangle is estimated by the weighted mean of the particles. The dimensions of the rectangle remain constant.

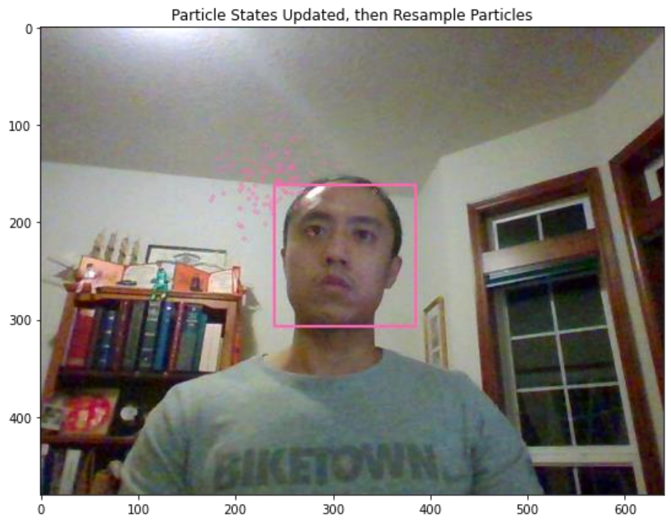

Figure 5. Frame 1: Particles states were updated and then particles were resampled.

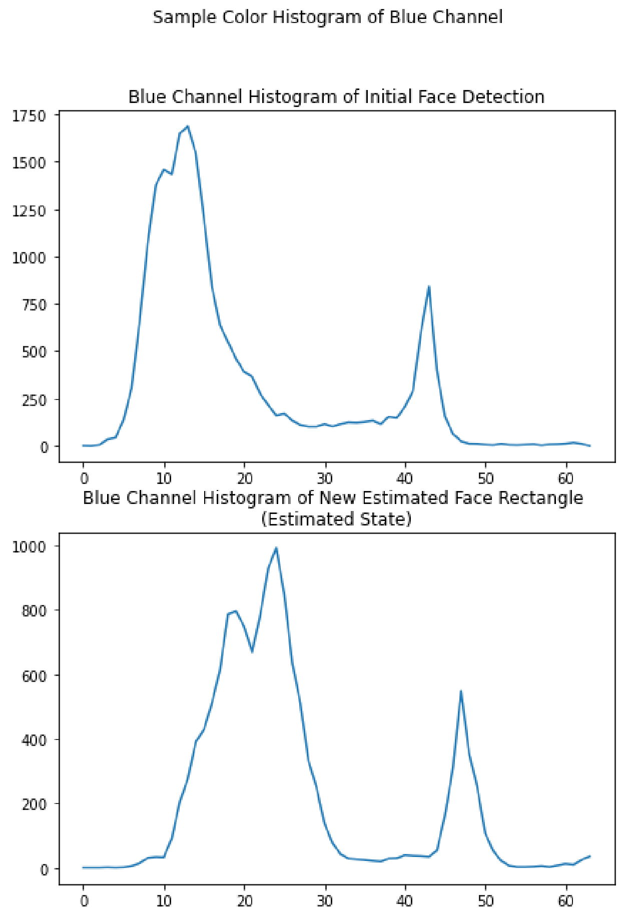

Figure 5. Sample color histograms of rectangular region for the blue channel. Top plot is the histogram for the inital face detection(as shown in figure 2). Bottom plot is the histogram for the updated state estimate (as shown in figure 5)

V. Discussion

I was only able to implement the particle filter face tracker to work for one frame after the initial detection, so I was unable to fully test the tracker. Due to unresolved bugs in the code, the program appears to either crash or freeze after the first state update and resample. It also appears that frames were being read in at a much slower rate than the expected 30 frames per second. However, given the very limited results, the tracker appeared to calculate the first estimated state update accurately when using 100 particles. I was also unable to test the performance of the tracker with occlusion, large head angle changes, lighting changes, etc.

If the tracker was working fully, I would expect that some parameters would need to be tuned to optimize tracking performance. For example, the process noise variance, u_x and u_y, for this tracker was set to be 0.2 times the width and height of the rectangle respectively. This might need to be increased or decreased depending on how face you expect the face to move, and how many particles you are using. This algorithm chose to resample after every estimated state update, but its possible that resampling could be done less frequently without sacrificing accuracy which would improve computational efficiency.

The particle weights for this algorithm was calculated based on color histograms and an exponential decay function. I chose arbitrarily chose 16 for α in the exponential decay weight formula but it is likely that this value would need to be tuned for optimal performance. Its also possible that a different method of calculating particle weights would yield better performance over this method of calculating color histograms.

Lastly, this tracker kept the dimensions of the face rectangle constant after the initial face detection. As such, the tracker may not function as well if the subject moves significantly closer or further to the camera. A solution to this problem can be to either (1) periodically perform face detection to "reset" the initial state of the tracker, or (2) allow the tracking algorithm to update the width and height parameters of the state as well as the x and y coordinates. However, this would require extra computation to calculate and update the rectangle dimensions and will reduce the efficiency of the tracker.

VI. Conclusion

A particle filter tracking algorithm was implemented to track a face as it moves across a video stream. Initial detection was performed with a Haar Cascade Classifier, and subsequent state updates were calculated by the weighted particle mean. Particle weights were calculated by color histogram comparison of the pixels in the estimated face rectangle to the pixels in the original face detection. Tracker appears to perform the first state update estimation accurately, however the program crashes or malfunctions shortly after. Further debugging is needed to fully test the performance of the tracker.

References

(References will be formatted properly in final draft).

[1] Le, D.-D., Wu, X., & Satoh, S. (2008). Face detection, tracking, and recognition for broadcast video. *Encyclopedia of Multimedia*, 228–238. https://doi.org/10.1007/978-0-387-78414-4_79

[2] Viola, P., & Jones, M. (n.d.). Rapid object detection using a boosted cascade of Simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. https://doi.org/10.1109/cvpr.2001.990517

[3] Behera, G. S. (2020, December 29). *Face detection with Haar Cascade*. Medium. Retrieved June 9, 2022, from https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08

[4] Jia, R. R. (2022, January 27). *An easy-to-implement face tracker using particle filtering (part 1)*. Medium. Retrieved June 9, 2022, from https://ricoruotongjia.medium.com/an-easy-to-implement-face-tracker-using-particle-filtering-part-1-8c91971c78db