Probabilistic Face Tracking From Location and Facial Identity Information

Zarni Phyo
Worcester Polytechnic Institute

Probabilistic Face Tracking From Location and Facial Identity Information

Written By:

ZARNI PHYO

Advisor:

PROFESSOR JACOB WHITEHILL

A Major Qualifying Project
WORCESTER POLYTECHNIC INSTITUTE

Submitted to the Faculty of the Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science in Computer Science.

ABSTRACT

In recent times, the advancement in object detection has induced new video processing applications. This paper explores the development of the face tracking system with a probabilistic approach where face similarities and relative positions are considered. The potential applications and further enhancement includes detecting the human faces along with the behavior analysis and video surveillance at the chaotic situation such as objects overlapping. The system includes three stages - face detection and face alignment using MTCNN, face recognition using FaceNet, and face tracking after Gibbs sampling. I evaluate the full-temporal method and three other approaches - baseline, positional, and instantaneous methods.

ACKNOWLEDGEMENTS

I would like to extend my gratitude to my MQP advisor, Professor Jacob Whitehill, for his guidance and support throughout the course of this project.

TABLE OF CONTENTS

LISTS OF TABLES

LISTS OF FIGURES

CHAPTER 1: INTRODUCTION

Face tracking, or face recognition in layman's term, has been a hot topic due to its extensive applications. In parallel with the emergence of the face tracking system, the development and the usage of cameras is accelerating. Cameras can be used to acquire information such as audio and video from the environment. The marriage between cameras and face tracking can yield numerous applications such as tracking students' faces in the classrooms to understand their sentiment so that the conductor can change their teaching style if necessary.

1.1 Motivation

Face detection and alignment are fundamental components of many face related applications such as facial recognition and tracking, facial expression analysis, and emotional inference. In an unconstrained environment, face detection and face alignment are quite challenging problems in modern computer vision due to different poses, lightings, and occlusions.

1.2 Proposed Solution

Face tracking in videos usually involves three stages - face detection and face alignment, face recognition, and face matching across different frames in the videos. I used MTCNN [5], a multi-cascaded convolutional network system published by Zhang K. et al. to detect faces with different illuminations and poses. And then, I used a deep convolutional network-based face recognition approach published by Schroff F., Kalenichenko D., and Philbin J. called FaceNet [6] to compute face embeddings where the Euclidean distance between two face embeddings is directly proportional to the similarity of the faces. My proposed face tracking method uses a probabilistic approach where I compute the probability distribution of every possible matching in each frame using probability distributions based on prior knowledge, face similarities and relative positions of the detected faces. And then, using the Gibbs sampling method with the combined probability distribution, the system matches the detected faces with people.

CHAPTER 2: BACKGROUND AND RELATED WORK

2.1 Face Detection and Alignment

Face detection is a computer vision technology to identify the geometric structure of human faces in digital images and videos. In other words, face detection is one of the individual cases among object-class detection where the position and size of the object are indicated by a rectangular box called bounding box.

Face alignment is a computer vision technology to achieve well-established forms of alignments of human faces based on translation, rotation, or scaling the detected face images. In other words, face alignment can be thought of as data normalization. The goal is to achieve scaled images of faces where the size of the images is relatively the same, and the eyes are approximately on the horizontal line, and the face is centered.

There are various kinds of approaches proposed by researchers from several different kinds of institutes in the last few decades. Most of the time, there is a trade-off between efficiency and effectiveness.

The cascade face detection algorithm from Viola-Jones [1] trains the cascaded classifier with Haar-Like features and AdaBoost to obtain real-time efficiency. But the performance is usually not good enough in real-world applications where variations in head poses, extreme lighting conditions, and occlusions are unavoidable. Other researchers introduce deformable part models (DPM) [2,3,4] where they achieve remarkable performance. But the difficulty in annotating data during the training stage and intensive computation power requirements, this method is not favorable either.

In this project, I used the multi-task framework, which uses the correlation between face detection and face alignment tasks. In the paper "Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks" published by Zhang, et al. [5], they explained:

> "... most of the available face detection and face alignment methods ignore the inherent correlation between these two tasks. Though there exist several works attempt to jointly solve them, there are still limitations in these works."

Due to the effective usage of the inherent correlation between face detection and face alignment, the efficiency and effectiveness of the system are notably improved.

2.2 Face Recognition

According to the book "Handbook of Face Recognition" by Stan Z. Li and Anil K. Jain published in 2011 [7], there are two main methods for facial recognition - face verification and face identification.

Face verification is a classification problem where a given face image is compared with another face image with a known identity to verify whether they are of the same person or not. Thus, face verification systems can be used to authenticate if a person is whom he or she claims to be, answering the question of "Is this the person?"

Face identification is also a classification problem where a given face image is matched against images of faces with a set of known identities. Thus, face identification systems can be used to identify a person among a database of people, answering the question of "Who is this person?"

In this project, I use FaceNet [6] to calculate how similar two faces are. FaceNet is a facial recognition and clustering system published in the paper called "FaceNet: A Unified Embedding for Face Recognition and Clustering" by Schroff F., Kalenichenko, D., and Philbin J. at Google research. In the paper above, they describe FaceNet as:

"... a system, called FaceNet, that directly learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity."

Using the triplet loss function, the FaceNet system maps images of detected faces into the corresponding 128-dimensional vectors called *embeddings*, where the Euclidean distance between one embedding from another represents how similar the corresponding two faces are. I use this metric as a measurement to determine face similarities.

2.3 Face Tracking

Face tracking is a computer vision technology to track the presence of human faces in digital videos. The challenges include many variabilities throughout the videos, such as poses, illuminations, occlusions, and disappearance and reappearance of a person. In recent years, researchers have come up with different solutions to tackle the challenge of tracking faces in videos. A research study, "Face Recognition and Tracking in Videos" published by Tathe S. et al. in Advances in Science, Technology and Engineering Systems Journal, used statistical analysis such as Kalman filter to track the positions of detected faces throughout the video. I use a probabilistic approach where the probability distribution of each configuration of matching between detected faces and actual people are computed based on face similarities and relative positions of the detected faces throughout the video. This approach contains several hyper-parameters that can be optimized using machine learning methods.

CHAPTER 3: METHODOLOGY

3.1 Method

In this model, I use a probabilistic approach where the probability distribution of each *configuration* of matching between detected faces and actual people are computed based on face similarities and relative positions. ***Configuration*** is defined as one possible pair of matching between every detected faces in a frame and the actual persons. There are many possible matchings between the detected faces and the actual persons in a video. Given the detected faces and their relative positions throughout the video, I can calculate the probability of every possible pair of matching between the detected faces and the actual persons.

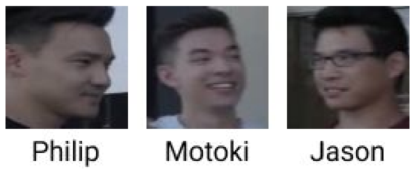

Figure 3.1: Faces from video clip *B* and their correct names

Consider this video with three distinct persons throughout the video - Philip (P), Motoki (M) and Jason (J). Consider a particular frame where the face detector detected three faces as shown in Figure 3.2. Since there are three people in the whole video, it is possible that some or all three of them are missing. Since there are three detections in the frame, it is possible that some or all three of those are false alarms. Therefore, I can draw a bijective bipartite graph to illustrate a configuration - a set of matchings from the set of people and false alarms to the set of detected faces and missings. Figure 3.2 shows the right configuration.

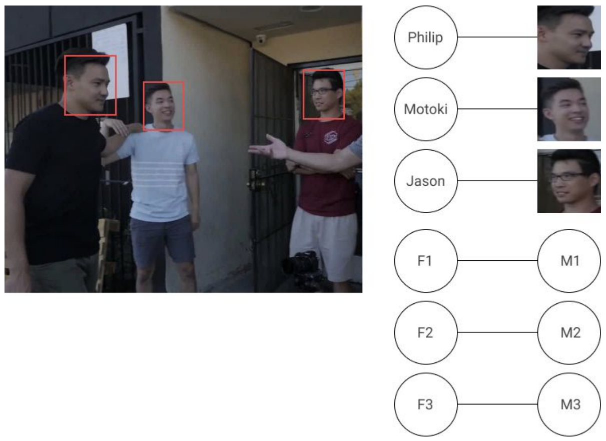

Figure 3.2: Three persons in the frame; Three correctly detected

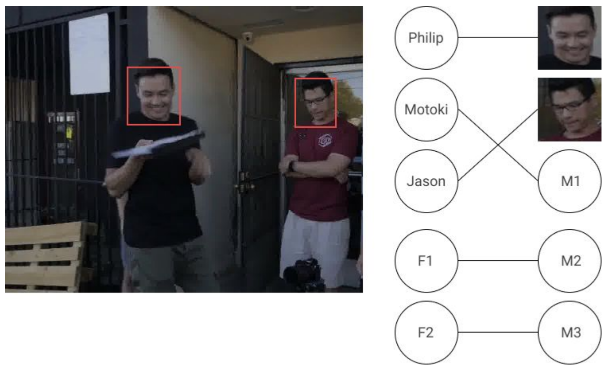

Figure 3.3: Two persons in the frame; Two correctly detected; one missing

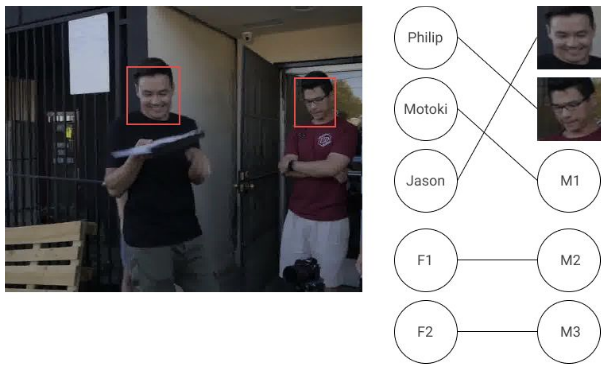

Figure 3.4: Two persons in the frame; two mismatched; one missing

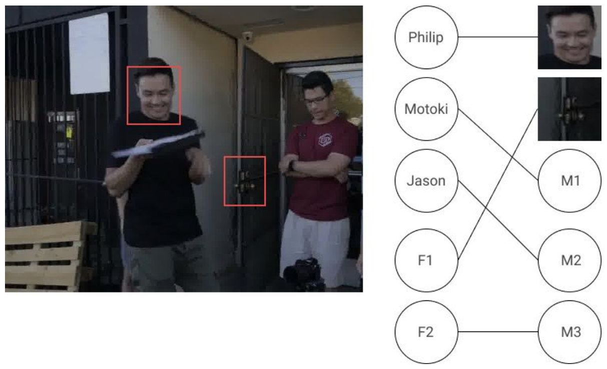

Figure 3.5: Two persons in the frame; one correctly detected; one false alarm; two missing

Consider a particular frame where the face detector detected two faces as shown in figure 3.3. Since there are three people and two detected face, one person must be missing. Figure 3.3 shows the correct configuration. Figure 3.4 shows one possible configuration where Philip and Jason are mismatched.

Consider a particular frame where the face detector detected two detections, where one of them is not a face (false alarm) as shown in figure 3.5. Since there is only one detection that is an actual face, the other detection must be false alarm. Figure 3.5 shows the configuration where one of the detections is false alarm.

Based on the probabilistic model that my MQP advisor derived, I can break down the probability of a particular configuration at frame *t* given the detected images and position of the bounding boxes as follows.

$$\begin{aligned} P(l_t \mid l_{\neg t}, i) &\propto P(l_t \mid l_{t-1}, l_{t+1}, i_t) \\ &= P(l_{t-1}, l_{t+1} \mid l_t, i_t)\ P(l_t \mid i_t) \\ &= P(l_{t+1} \mid l_t)\ P(l_{t-1} \mid l_t)\ P(l_t \mid i_t) \\ &= P(l_{t+1} \mid l_t)\ \tfrac{P(l_{t-1})}{P(l_t)}\ P(l_t \mid l_{t-1})\, P(l_t \mid i_t) \end{aligned}$$

where l_t = a configuration at frame *t*

$P(l_t \mid i_t)$ = probability of the configuration l_t, given the detected images at frame *t*

$P(l_t)$ = prior probability of the configuration l_t

$P(l_t \mid l_{t-1})$ = probability of the configuration l_t, given the configuration of the previous frame l_{t-1}

3.2 Evaluation Approach

To track people throughout the video clip, I compare my proposed method with other 3 approaches - baseline, positional, instantaneous, and full-temporal. I then evaluate the percent accuracy of these 4 approaches.

3.2.1 Baseline Method

The baseline method uses a k-mean clustering algorithm to cluster face embeddings into k clusters. K-mean clustering is the classification of data into k clusters where each data point belongs to the cluster with the nearest centroid. First, all the detected faces are converted into 512-dimensional face embeddings. Assuming that the total number of k distinct people throughout the video is already known, I used a k-mean clustering algorithm to partition the face embeddings into k clusters and compare with the ground truth to obtain the percent accuracy.

3.2.2 Positional Method

Positional method only uses the position of detected bound boxes to track people throughout the video. I can track where a particular person is in a given frame just from the positions of the detected bounding boxes. Given the position of the detected bounding boxes in frame *t-1*, I can compute the probability distribution of every possible configuration for frame *t* using the method which will be mentioned in chapter 4. I use Gibbs sampling to sample from that probability distribution along with the prior probability distribution as mentioned in chapter 4.

$$P(l_t) = P(l_{t+1} \mid l_t) \frac{P(l_{t-1})}{P(l_t)} P(l_t \mid l_{t-1})$$

I iterate the sampling process 10,000 times. In the first iteration, the first sample is initialized using prior probability $P(l_t)$. Then later samples are computed from the positional probability distribution. I then drop the first 10% of the samples and generate the final probability distribution out of those samples. For each frame, the configuration with the highest probability is selected and compared with the ground truth to obtain the percent accuracy.

3.2.3 Instantaneous Method

Instantaneous method only uses the face embeddings of the detected faces to track people throughout the video. The main idea is that if I can figure out how likely a detected face is of a particular person just based on the face similarity, I can track the person throughout the video. For each detected faces, I compute the face embeddings. Then I compute the probability distribution of face similarity as discussed in previous section. For each frame, the configuration with the highest probability is selected and compared with the ground truth to obtain the percent accuracy.

3.2.4 Full-temporal Method

Full-temporal method uses both faces similarity and position of the detected bounding boxes across the frames to track people throughout the video. As I discussed, in some cases, just using the face similarities could result in wrong predictions. In some other cases, just using the position of the detected faces could result in wrong predictions. So if I combine those two methods, my model could potentially become stronger and will theoretically result in better accuracy than the other two.

I iterate the sampling process n times. Similar to the positional method, in the first iteration, the first sample is initialised using prior probability P(l_t). Then later samples are computed from the combined probability distribution.

$$P(l_t \mid l_{\neg t},\, i) \;=\; P(l_{t+1} \mid l_t)\; \tfrac{P(l_{t-1})}{P(l_t)}\, P(l_t \mid l_{t-1})\, P(l_t \mid i_t)$$

Similar to the positional method, I drop the first 10% of the samples and generate the final probability distribution out of those samples. For each frame, the configuration with the highest probability is selected and compared with the ground truth to obtain the percent accuracy.

CHAPTER 4: EXPERIMENT

4.1 Probability Distribution from Face Similarity

Based on prior research [6], given the face images, I can calculate face similarity between two faces from the euclidean distance between the two corresponding face embeddings. I can calculate the probability distribution of configurations based on the face similarity as follows.

In frame *t*, the probability of a particular configuration l_t given all the detected face images *I*, denoted by $P(l_t \mid I)$, depends on four cases. For every pair from a detected face to a prototype, I calculate a cost based on how similar the detected face is to its corresponding prototype by computing the Euclidean distance between the face embedding of the detected face and the corresponding embedding of the matched prototype. For every pair from a detected face to a false alarm, I assign a cost H_{FA} which is a hyperparameter. For every pair from a missed face to a prototype, I assign a cost H_M which is also a hyperparameter. In the ideal case, false alarm will match up with missings. So for every pair from a missed face to a false alarm, I simply assign a cost of 0.

I define the unnormalized probability as inversely proportional to the cost. In order to calculate unnormalized probability $P'(l_t \mid I)$, I invert the sum of the costs over all the pairs in the bipartite graph. Then, I calculate the normalized probability $P(l_t \mid I)$ by dividing each unnormalized probability by the sum of unnormalized probabilities of all possible configurations.

$$P'(l_t \mid I) = \sum_{i=1}^{k} \frac{1}{cost_i}$$

k represents the total number of configurations.

$$P(l_t \mid I) = \frac{P'(l_t \mid I)}{\sum P'(l_t \mid I)}$$

The outcome is a probability distribution across every possible configuration based on face similarity. The implementation of the algorithm can be found in Appendix A with the name `get_prob_from_embeddings`. This function takes the face embeddings of detected faces of every frame and the centroids face embeddings of every person in the video and measure the Euclidean distance between them. And it returns the normalized probability distribution of every possible configuration of each frame based on face similarity.

4.2 Prior Probability Distribution

Based on prior research [5], I believe that my face detector is quite accurate, though not perfect. Hence, it is very unlikely to miss *all* the faces out of a very large number of faces; however it is possible to occasionally miss a face. I can encode this prior believe using the following distribution.

In frame *t*, the probability of a particular configuration l_t, denoted by $P(l_t)$, depends on four cases. For every pair from a detected face to a prototype, I assign a cost H_D which is a hyperparameter. For every pair from a detected face to a false alarm, I assign a cost H_{FA} which is also a hyperparameter. For every pair from a missed face to a prototype, I assign a cost H_M which is also a hyperparameter. For every pair from a missed face to a false alarm, I simply assign a cost of 0.

Then, similar to previous probability distribution, convert all the costs for each configuration to normalized probability and generate the prior probability distribution. The implementation of the algorithm can be found in Appendix A with the name `get_prob_from_nothing`. This function takes the number of distinct people in the video and the number of detected faces in each frame. And it returns the normalized probability distribution of every possible configuration of each frame with no prior knowledge of the position of the detected faces or the faces itself.

4.3 Probability Distribution from Face Position

Given the belief about the correct configuration c_t at frame t, I compute a cost between c_t and c_{t+1} where c_{t+1} can be any configuration at frame *t+1*. First of all, I need to define the cost between any configuration c_t at frame *t* and any config c_{t+1} at frame *t+1*. I define the cost between c_t and c_{t+1} as the sum, over all n people, of the distance between each person *p* according to c_t and person *p* in c_{t+1}. I define this distance as follows.

If a particular person *p* is detected in frame *t* and *t+1*, I calculate a cost based on how far the detected bounding box for person *p* in frame *t* is from the detected bounding box for person *p* in frame *t+1* by computing the Euclidean distance between them. If a particular person is detected in frame *t* and missing in *t+1*, I assign a cost H_M which is a hyperparameter. If a particular person is missing in frame *t* and detected in *t+1*, I assign a cost H_M which is a hyperparameter. If a particular person is missing in frame *t* and missing in *t+1*, I simply assign a cost of 0. Note that there are many possible values for c_t and c_{t+1}; hence I have to compute the distances between all possible pairs of configurations.

Similar to previous probability distributions, I summed the costs over all the people in each configuration and convert it to normalized probabilities. The outcome is a probability distribution of every possible configuration based on the position of the detected faces between two consecutive frames. The implementation of the algorithm can be found in Appendix A with the name `get_prob_from_positions`. This function takes the positions of detected faces of every frame and the total number of people throughout the video, and measure the distance between the position of a detected bounding box in a frame with that of another frame. And it returns the normalized probability distribution of every possible configuration of each frame based on the probability of the position of a person before the current frame.

4.4 Gibbs Sampling from Combined Probability Distribution

From the combined probability distribution, I sampled the configuration for each frame using Gibbs sampling. Gibbs sampling is a sampling technique used when sampling directly from the multivariate probability distribution is difficult. In this case, I use a series of conditional probability distributions.

Before I sample from the conditional probability distribution, I need a way to initialize the sampling process. As shown in Appendix B with the name `initialize_sample_`, the prior probability is used to initialize the configurations for each frame. Then I iterate the following steps 10,000 times. For each frame, generate a random number $0 <= R <= 1$ from a uniform distribution. Given the positional probability distribution table, the configuration of previous frame and the generated random number, the configuration of current frame is computed based on the configuration of previous frame. The computed configuration from previous frame is stored in the samping table. After 10,000 rounds of sampling, the first 10% of the samples are discarded. Then the final distribution of configurations for each frame can be generated from the samples.

CHAPTER 5: EXPERIMENTAL RESULTS

5.1 Data

I use two video clips in my experiment. The figure shows some of the frames from each clip. These videos are available on YouTube.

Clip *A* is a presidential debate between Donald Trump and Hillary Clinton, where both of them are facing the camera most of the time. The length of the clip is 6 minutes and 57 seconds. This video clip was used as an initial evaluation of my tracking system due to ease of labeling the faces, ease of detecting the faces - the subjects facing forward most of the time throughout the video, and ease of developing the minimum viable product. I cut the video starting from 00:18 to 6:57 to remove the camera switches between the podium and the actual debate between the two presidential candidates. After that, I extract the images from the video with an interval of one frame per second, resulting in 399 total frames.

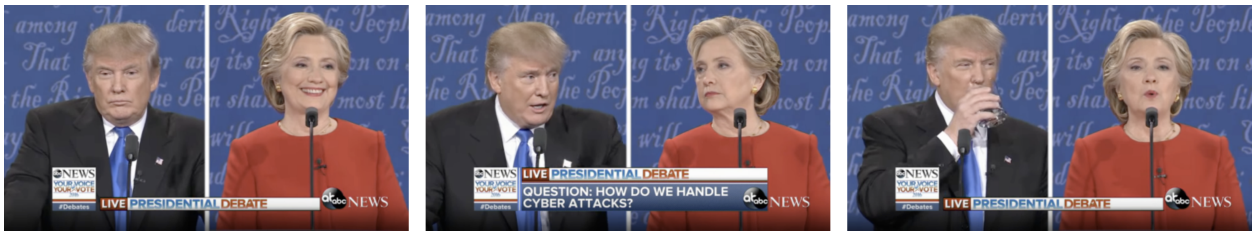

Figure 5.1: A few frames from the video clip *A* demonstrating slightly changing head poses and minor occlusion

Clip *B* is a short behind the scenes video where three people are standing in front of a building and talking, changing their positions, and making funny expressions. The length of the clip is 12 minutes and 7 seconds. This video clip is used to evaluate my tracking system after the clip A further because people switch positions instead of standing still, change poses, act extreme facial expressions, disappear from the video and reappear again after the disappearance. I cut the video starting from 6:18 to 6:25 to remove some of the camera movements and transitions between the scenes. After that, I extract the images from the video with an interval of two frames per second, resulting in 14 frames in total. Originally, there are four people in each frame. To make the labelling faster, each frame is cropped so that there are only three people in each frame.

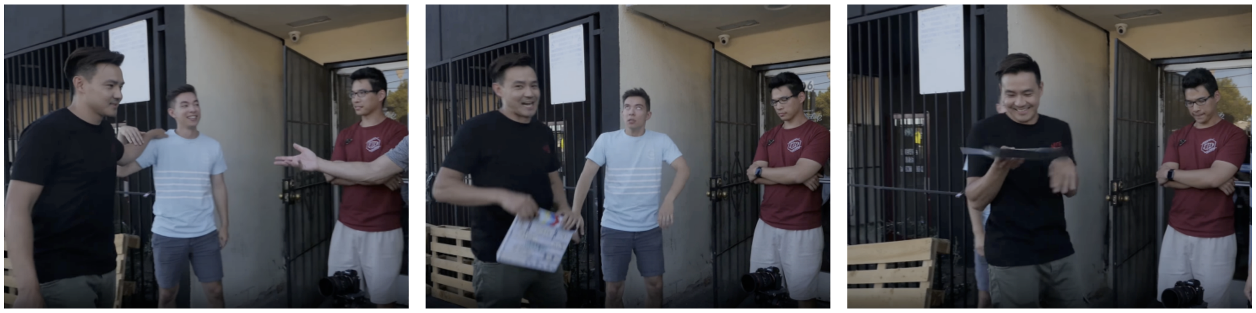

Figure 5.2: A few frames from the video clip *B* demonstrating significantly changing head poses and complete occlusion of the face.

5.2 Results

After conducting the experiment, the results for baseline method, positional method, instantaneous method and full-temporal methods are measured with percent accuracy with each different hyperparameters. Figure 5.3 and 5.4 show the PCA of face embeddings after clustering using K-mean clustering algorithm where the yellow and blue dots represent two different persons in the video clip *A*. Figure 5.5 shows the PCA of face embeddings after clustering using K-mean clustering algorithm where different colors represent different persons in the video clip

B. Figure 5.6 shows the PCA of face embeddings where yellow represents correctly predicted while blue represents errors.

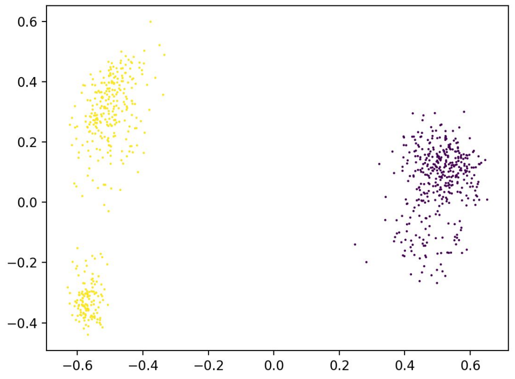

Figure 5.3: Clip *A* - PCA of K-mean clustering

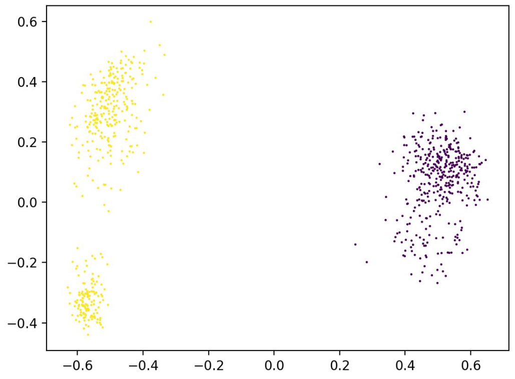

Figure 5.4: Clip *A* - PCA of Baseline Method

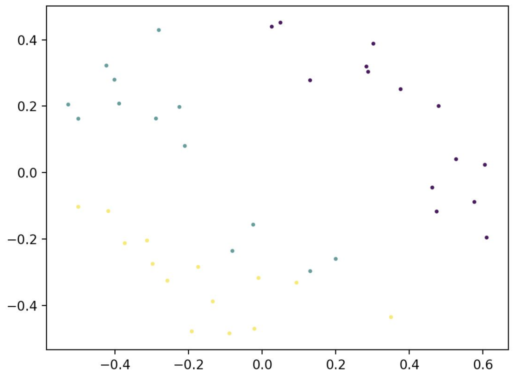

Figure 5.5: Clip *B* - PCA of K-mean clustering

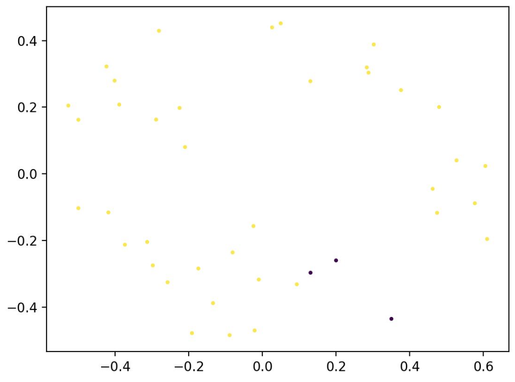

Figure 5.6: Clip *B* - PCA of K-mean clustering indicating errors

		Percent Accuracy			
		Baseline	Positional	Instantaneous	Full-temporal

Clip *A*	1000 Total Samples; Skip first 10	100% (798/798)	100% (798/798)	99.75% (796/798)	95.74% (764/798)
	10000 Total Samples; Skip first 100	100% (798/798)	100% (798/798)	99.75% (796/798)	97.99% (782/798)
	10000 Total Samples; Skip first 1000	100% (798/798)	100% (798/798)	99.75% (796/798)	**98.45%** (786/798)
Clip *B*	1000 Total Samples; Skip first 10	92.68% (38/41)	73.17%7 (30/41)	97.56% (40/41)	82.93% (34/41)
	10000 Total Samples; Skip first 100	92.68% (38/41)	78.05 (32/41)	97.56% (40/41)	87.80% (36/41)
	10000 Total Samples; Skip first 1000	92.68% (38/41)	78.05 (32/41)	97.56% (40/41)	**95.12%** (39/41)

Table 5.1: This table shows the percent accuracy of different methods with different sampling sizes for video clip *A* and *B*.

5.3 Evaluations

5.3.1 Evaluation of the Effectiveness of Baseline Method

As previously mentioned in the table 5.1, the baseline method works very well with the first clip where there are relatively small number of different people in the video and they are facing forward most of the time throughout the video. In the second clip, where there are more people and complex interactions among each other, the accuracy of the baseline method suffers.

Even though the algorithm is extremely fast with linear asymptotic complexity, the accuracy drops drastically as the number of people increases throughout the video.

5.3.2 Evaluation of the Effectiveness of Positional Method

Similar to the baseline method, the positional method works very well for the first clip where there are no occlusions or disappearance of subjects and the positions of the subjects are relatively fixed throughout the video. In the second clip, the accuracy drops drastically due to the complexity of the video. Most of the mismatching occurs near the end of the clip where people switch their positions. While they are switching positions from left to right, one person is standing right behind another person covering his face entirely. If the two detected faces are close together, it can be hard to track the subjects just using their relative positions due to the proximity of the detected faces.

5.3.3 Evaluation of the Effectiveness of Instantaneous Method

As explained in previous section, the instantaneous method based on the face similarities. This method works very well for the first clip in which the two subjects are very distinct from one another - one of the subjects being male and the other female. The only frame that method failed to match correctly is the frame where majority of the face of the subject is covered. Similarly, the instantaneous method works very well for the second clip. When I compared with other methods, this method achieved the highest percent accuracy of 97.56%.

5.3.4 Evaluation of the Effectiveness of Full-temporal Method

Full-temporal method is my proposed method where the tracking system takes into account the face similarities as well as the positions of the detected bounding boxes before and after a given frame. As mentioned in table 5.1, after sampling from the combined probability distributions of each configuration for each frame, the percent accuracy of the full-temporal method on the first clip is 98.45% and the percent accuracy for the second clip is 95.12%. Even though the results should be theoretically better than other methods, the results show that is not the case. The reason might be due to the nature of the dataset or error in sampling method.

However, when I increase the number of subjects in the video, I can see that the percent accuracy is not affected dramatically unlike the baseline method and the positional method where the percent accuracy drops drastically.

CHAPTER 6: CONCLUSION

Based on the results of face tracking with different approaches, I learnt that clustering approach is very efficient and can be effective when the number of people in the video is relatively small. But in an unconstrained environment, the other three probabilistic approaches outperform the clustering approach. The accuracy of the tracking system based on face similarity is remarkably well even in environments where the subjects are making extreme facial expressions. But this approach seems to be falling short when the face is blocked by something else. Face tracking based on face positions works better than other approaches when there are facial occlusions in the video. Therefore the full-temporal approach is more reliable where both probabilities from face similarity and face position are considered before sampling. In the future, the project can be extended to evaluate with more standard datasets.

BIBLIOGRAPHY

[1] Viola, P. & Jones, M. (2004). Robust real-time face detection. *International Journal of Computer Vision*, Vol. 57(Issue. 2), 137-154.

[2] Mathias, M. & Benenson, R. & Pedersoli, M. & Van Gool, L. (2014). Face Detection without Bells and Whistles. *European Conference on Computer Vision*, 720-735.

[3] Yan, J. & Lei, Z. & Wen, L. & Li, S. (2014). The Fastest Deformable Part Model for Object Detection. *IEEE Conference on Computer Vision and Pattern Recognition*, 2497-2504.

[4] Zhu, X. & Ramanan, D. (2012). Face Detection, Pose Estimation, and Landmark Localization in the Wild. *IEEE Conference on Computer Vision and Pattern Recognition*, 2879-2886.

[5] Zhang, K. & Zhang, Z. & Li, Z. & Qiao, Y. (2016). Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. *IEEE Signal Processing Letters*, Vol. 23(Issue. 10), 1499-1503.

[6] Schroff, F. & Kalenichenko, D. & Philbin, J. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 815-823.

[7] Li, S. & Jain, A. (2011). Handbook of Face Recognition. *Springer Publishing Company, Incorporated*.

[8] Tathe, S. & Narote, A. & Narote, S. (2017). Face Recognition and Tracking in Videos. *Advances in Science, Technology and Engineering Systems Journal*. Vol2. 1238-1244.

Appendix

Appendix A

```
# Computing normalized probability distribution of every possible configuration of each frame
# based on how similar two faces are
# by computing the Euclidean distance between their respective embeddings.
# The closer the distance, the more similar the two faces are.
def get_prob_from_embeddings(frames, embeddings, labels, centroids):
    n = centroids.shape[1]
    prob_dict = defaultdict(list)
    # for each frame, ...
    for frame, bbs in frames.items():    # bbs: [index of embedding, ...]
        m = len(bbs)
        # for each configuration out of k configurations: compute the cost,
        # and put it in the 'costs' array
        costs = [0] * factorial(m+n)
        for k, xs in enumerate(permutations(range(m+n))):
            # consider i -> xs[i];
            for i,j in enumerate(xs):
                if i < n:    # person case
                    if j < m:    # person -> detected
                        costs[k] += distance(centroids[:, i], embeddings[:, bbs[j]])
                    else:    # person -> missed
                        costs[k] += MISSING_CONSTANT

                else:    # false alarm case
                    if j < m:    # false alarm -> detected
                        costs[k] += FALSE_ALARM_CONSTANT
                    else:    # false alarm -> missed
                        costs[k] += 0
        prob_dict[frame] = get_prob_from_raw_costs(costs)

    return prob_dict
```

```
# Computing normalized probability distribution of every possible configuration of each frame
# based on the face detection system
# with no prior knowledge of the position of the detected face or the face itself
def get_prob_from_nothing(frames, centroids):
    n = centroids.shape[1]
    prob_dict = defaultdict(list)

    # for each frame, ...
    for frame, bbs in frames.items():      # bbs: [index of embedding, ...]
        m = len(bbs)
        costs = [0] * factorial(m+n)
        for k, xs in enumerate(permutations(range(m+n))):
            for i,j in enumerate(xs):
                if i < n:  # person case
                    if j < m:   # person -> detected
                        costs[k] += DETECTION_CONSTANT
                    else:    # person -> missed
                        costs[k] += MISSING_CONSTANT

                else:   # false alarm case
                    if j < m:   # false alarm -> detected
                        costs[k] += FALSE_ALARM_CONSTANT
                    else:    # false alarm -> missed
                        costs[k] += 0

        prob_dict[frame] = get_prob_from_raw_costs(costs)

    return prob_dict
```

```
# Computing normalized probability distribution of every possible configuration of each frame
# based on the probability of the position of a person before the current frame and
# after the current frame
# by computing the distance between the positions the detected bounding boxes.
# The closer the distance, the more likely that those two people are the same.
def get_prob_from_positions(frames, centroids, positions):
    n = centroids.shape[1]
    frame_keys = list(sorted(frames.keys()))

    probs_dict = defaultdict(list)

    # from previous frame, for each frame ...
    for a in range(1, len(frame_keys)):
        prev_frame_bbs = frames[frame_keys[a-1]]
        curr_frame_bbs = frames[frame_keys[a]]

        for k, xs in enumerate(permutations(range(len(prev_frame_bbs)+n))):
            prev_frame_positions = []
            for i,j in enumerate(xs):
                if i < n:
                    if j < len(prev_frame_bbs):    # person -> detected
                        prev_frame_positions.append(positions[:, prev_frame_bbs[j]])
                    else:    # person -> missing
                        prev_frame_positions.append([inf,inf])
                # things to consider here for FA -> Detection and FA -> Missing ...
                else:
                    prev_frame_positions.append([inf,inf])

            costs = [0] * factorial(len(curr_frame_bbs)+n)
            for l, ys in enumerate(permutations(range(len(curr_frame_bbs)+n))):
                curr_frame_positions = []
                for p,q in enumerate(ys):
                    if p < n:
                        if q < len(curr_frame_bbs):       # person -> detected
                            curr_frame_positions.append(positions[:, curr_frame_bbs[q]])
                        else:       # person -> missing
                            curr_frame_positions.append([inf,inf])
                    # things to consider here for FA -> Detection and FA -> Missing ...
                    else:
                        curr_frame_positions.append([inf,inf])

                # compute the distance between positions
                for prev_pos, curr_pos in zip(prev_frame_positions, curr_frame_positions):
                    dist = pos_distance(prev_pos, curr_pos)
                    costs[l] += dist

            probs_dict[frame_keys[a]].append(get_prob_from_raw_costs(costs))

    return probs_dict

# raw cost_array -> normalized prob_array
def get_prob_from_raw_costs(costs):
    probs = [ exp(1/cost) for cost in costs ]
    total_prob = sum(probs)
    probs = [ prob/total_prob for prob in probs ]
    return probs

def get_cumulative_prob_from_raw_probs(probs):
    total_prob = sum(probs)
    normalized_probs = [ prob/total_prob for prob in probs ]
    for i in range(1, len(normalized_probs)):
        normalized_probs[i] += normalized_probs[i-1]
    return normalized_probs

# prob_dict -> c_prob_dict
def get_cumulative_prob(prob_dict):
    c_prob_dict = defaultdict(list)
    for frame, probs in prob_dict.items():
        c_prob_dict[frame] = probs[:]
        for i in range(1, len(probs)):
            c_prob_dict[frame][i] += c_prob_dict[frame][i-1]
    return c_prob_dict
```

Appendix B

```
# Sample Initialization: used prior probability to initialize the configurations for each
frame

# Combined probability distribution: Generated from prior, face similarity and face positions

# Using the initialized sample with combined probability distribution to execute Gibbs
sampling

def sampling(P_Lt_It, P_Lt, P_Lt_Lt__1):

    S, Sd = initialize_sample_(get_cumulative_prob(P_Lt))

    frames = list(sorted(P_Lt.keys()))


    # for each frame...

    for j in range(TOTAL_ITERATIONS):

        for i in range(len(frames)):

            frame = frames[i]

            # combined(P_Lt_It[frame], P_Lt[frame], P_Lt_Lt__1[frame][S[prev_frame]])

            # probabilities for different configurations

            Pt = [0] * len(P_Lt[frame])


            if i == 0:

                next_frame = frames[i+1]

                next_frame_config = S[next_frame]


                for c in range(len(P_Lt[frame])):

                    Pt[c] = P_Lt_Lt__1[next_frame][c][next_frame_config] * ( 1 /
P_Lt[frame][c] ) * P_Lt_It[frame][c]


                S[frame] = sample(get_cumulative_prob_from_raw_probs(Pt))

                if j >= SKIP_FIRST_ITERATIONS: Sd[frame][S[frame]] += 1

                continue


            elif i == len(frames)-1:

                prev_frame = frames[i]

                prev_frame_config = S[prev_frame]


                for c in range(len(P_Lt[frame])):

                    Pt[c] = ( P_Lt[prev_frame][prev_frame_config] / P_Lt[frame][c] ) *
P_Lt_Lt__1[frame][prev_frame_config][c] * P_Lt_It[frame][c]
```

```
                S[frame] = sample(get_cumulative_prob_from_raw_probs(Pt))
                if j >= SKIP_FIRST_ITERATIONS: Sd[frame][S[frame]] += 1
                continue

            prev_frame, next_frame = frames[i-1], frames[i+1]
            prev_frame_config, next_frame_config = S[prev_frame], S[next_frame]

            # for each possible config... 0..23
            for c in range(len(P_Lt[frame])):
                A = P_Lt_Lt__1[next_frame][c][next_frame_config]
                B = ( P_Lt[prev_frame][prev_frame_config] / P_Lt[frame][c] )
                C = P_Lt_Lt__1[frame][prev_frame_config][c]
                D = P_Lt_It[frame][c]

                Pt[c] = A * B * C * D

            S[frame] = sample(get_cumulative_prob_from_raw_probs(Pt))
            if j >= SKIP_FIRST_ITERATIONS: Sd[frame][S[frame]] += 1

    final_distribution = compute_distribution_from_sampling(Sd)
    print('- done computing final distribution')
    return final_distribution

# initializing the sample with prior probability distribution
def initialize_sample_(C_P_Lt):
    S = defaultdict(int)
    Sd = defaultdict(list)
    for frame, probs in C_P_Lt.items():
        S[frame] = sample(probs)
        Sd[frame] = [0] * len(probs)
    return S, Sd

# probs: Cumulative Probability Distribution (array of probabilities)
def sample(probs):
    rdn = random()
    for i, prob in enumerate(probs):
        if rdn <= prob: return i
    return len(probs) - 1


# After sampling, drops first 10% of samples and generate probability distribution table for
every possible configuration

def compute_distribution_from_sampling(Sd):

    final_distribution = defaultdict(list)

    for frame, freqs in Sd.items():

        total_freq = sum(freqs)

        final_distribution[frame] = [ freq/total_freq for freq in freqs ]

    return final_distribution
```