# Human Face Mosaicing in Videos Using Object Tracking Method Based on Kalman Filter

Qiping Pan, Lyubing Qiang, Wenxin He, Mingyi Tang, and Puchen Xu

University of Michigan, Ann Arbor, MI, USA
{panqp, evelynq, wenxinhe, mycandy, korolxu}@umich.edu

## 1. Introduction

As a result of the rapid proliferation of information technology, privacy becomes a major concern for numerous people. A fair amount of individual privacy leaking occurs through inappropriate usage of personal portraits, which introduces the problem of human face mosaicing.

Human face mosaicing in videos is a relatively specific problem concerning computer vision, which requires the detection and tracking of human faces in videos and certain post-processing methods to mosaic the face patches. With the problem successfully resolved, the protection of personal information could potentially be benefited. For instance, it can serve as the back-end algorithm for video processing applications or social media applications to allow users to mosaic the faces in the videos.

Fortunately, the problem arising from technology can possibly in turn be mitigated by technology. Recent years have witnessed great advancements in computer vision, among which visual object tracking has long been an active topic. Aiming at extracting and predicting the location and trajectory of objects in video streams, visual object tracking has a wide application in various areas such as navigation for autonomous vehicles, visual surveillance, action recognition and industrial robotics. With the growing accuracy in tracking objects along the whole timeline of a video, human face mosaicing can be realized by a combination of face tracking and patch processing.

Previous works in visual object tracking involves classical tracking algorithms like Discriminative Correlation Filter used by openCV built-in CSRT tracker [1] and Kalman Filter proposed by Young Min Kim [2], which we take as references in our work. With the boosting of neural networks in recent years, scholars also integrated deep learning in object tracking. For instance, FairMOT is a benchmark in Multi-Object Tracking implemented by a combination of both classical and deep learning methods [3]. To make the human face mosaicing more powerful and applicable, a possible direction for improvement in the future is the real-time mosaicing processing. From another aspect we might cooperate the idea with face recognition, allowing the algorithm to recognize and track different faces. Thus by applying mosaicing separately the algorithm will have a wider application to suspect investigation, identity authentication, etc. Deep learning is proved also useful in face recognition. Methods like FaceNet have achieved high accuracy in multiple datasets [4].

For simplicity, this project focuses on the mosaicing of a single object (i.e. a single person appearing in the video stream). Our visual object tracking algorithm is building on SIFT features and Kalman Filter for face detection and tracking, while various post-processing methods are experimented such as Gaussian blurring and image replacement. The performance of our algorithm is compared to openCV build-in trackers and optical flow implementation using heterogeneous metrics.

The rest of the report is organized as follows: Section 2 explains the background and detailed implementation of our algorithm. Section 3 presents the validation and evaluation of the algorithm to other benchmarks based on plots and statistics. The difference between our work and the previous one which our project is based on will be indicated in Section 4.

## 2. Approach

Our object tracking algorithm is built on SIFT features and Kalman Filter. Figure 1 is the flow of our algorithm.

### 2.1. SIFT feature generation

SIFT (Scale Invariant Feature Transform) is an algorithm that generates features that can be used for object recognition. In order to recognize and track the human face, SIFT features are generated and saved. In this project, we utilized the sift generation function in openCV.

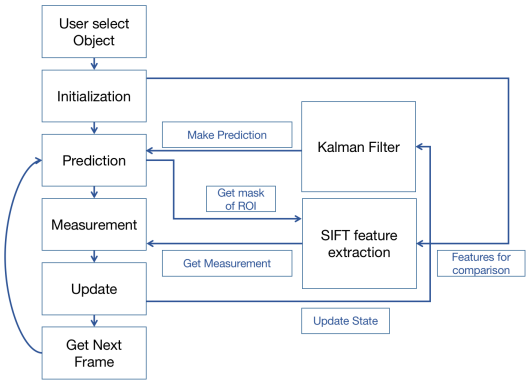


Figure 1. Workflow of our Algorithm

## 2.2. Kalman Filter

The measurements (the position and size of the region of the person's face in this case) are born with noise. To gain a more trusted trajectory of the object, we applied the Kalman filter to our algorithm. The Kalman filter uses a linear model to make predictions about the next move of the object and then updates it using the measurement. In this way, the system can output a more stable and trusted result. As the Kalman filter only does prediction in a linear way, the model can not react well to a sudden acceleration or a change in direction. However, the algorithm provides good performance if the video is sampled in high frequency, in which case the object can be seen as moving in constant velocity across adjacent frames.

## 2.3. Model Setup

We need to recognize the region of the person's face. It requires four parameters: $X$, $Y$, $W$, $H$, where $X$, $Y$ are the position of the object (top-left corner of the rectangle) and $W$, $H$ is the size of the object (width and height of the rectangle).

Since there is no relation between the position and size, we set up two Kalman filters for the two sets of parameters respectively. The state vector of our Kalman filter for position is

$$\vec{s}_{XY} = \begin{bmatrix} X & Y & \dot{X} & \dot{Y} \end{bmatrix}^T,$$

and the state vector of our Kalman filter for size is

$$\vec{s}_{WH} = \begin{bmatrix} W & H & \dot{W} & \dot{H} \end{bmatrix}^T,$$

where $\dot{X}$, $\dot{Y}$, $\dot{W}$, $\dot{H}$ terms denote the changing rate of the corresponding parameters.

The transition equation that predicts the next state is

$$\hat{\vec{s}} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \vec{s} + N(0, Q),$$

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & q^2 & 0 \\ 0 & 0 & 0 & q^2 \end{bmatrix},$$

where $\vec{s}$ is the current state vector ($\vec{s}_{XY}$, $\vec{s}_{WH}$ above), $\hat{\vec{s}}$ is the corresponding predicted next state vector and $Q$ is the process noise covariance matrix.

The measurement vector that we take and feed into the kalman filter is the tentative position and size of the object.

$$\vec{z}_{XY} = \begin{bmatrix} X_{obs} & Y_{obs} \end{bmatrix}^T, \quad \vec{z}_{WH} = \begin{bmatrix} W_{obs} & H_{obs} \end{bmatrix}^T$$

Note that

$$\vec{z} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \vec{s} + N(0, R), \quad R = \begin{bmatrix} r^2 & 0 \\ 0 & r^2 \end{bmatrix},$$

where $\vec{z}$ is the measurement vector ($\vec{z}_{XY}$, $\vec{z}_{WH}$ above) and $R$ is the measurement noise covariance matrix.

The way we correct the predicted state is the same as the method introduced in Section 2.2 of the previous work [2], which is an optimization based on the variance. After we've obtained the Kalman gain $K$, we correct our predicted state $\hat{\vec{s}}$ with

$$\vec{s}' = \hat{\vec{s}} + K(\vec{z} - \mathbf{H}\hat{\vec{s}}), \quad \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Here we $\vec{s}'$ is used as the final state vector for the position and size of the object.

## 2.4. Intialization

At initialization, we setup two Kalman filters for position ($X$, $Y$) and size ($W$, $H$) and read the first frame as the previous frame. We require users to input the position and the size of the object in the first frame and we update the two Kalman filters with these input parameters.

## 2.5. Kalman Filter Predict and Update

After the initialization, we then perform iterations. At the beginning of each iteration, we read the next frame as the current frame. We use KNN to find the matched SIFT features of the object in the previous frame and the current frame. Based on these matched points, we approximate the position and size of the object, *i.e.* $X$, $Y$, $W$, $H$ in the current frame, the method of which will be discussed later. We then feed into the two Kalman filters with these approximated measurements and get the final corrected result from the Kalman filters as the final position and size of the object in the current frame. Note that if there is no matched key points in the current frame, we will just keep $X$, $Y$, $W$, $H$ in the current frame the same as that in the previous frame. Else, if there is only one matched point in the current frame, we will fix $W$, $H$ and change $X$, $Y$ according to the difference of the feature points in the two frames.

## 2.6. Producing Measurement

We first need to obtain the matched key points in the current frame. After that, we need to convert it into the position and size of the object, *i.e.* $X, Y, W, H$, to feed into the Kalman filter. The steps are stated below.

### 2.6.1 KNN for keypoint matching

The descriptor of a feature point is represented by a vector. After extracting the feature points in the previous frame, we used the KNN algorithm to get the matched SIFT feature points in the current frame. We also tuned our parameters for the upper bound of the difference between the descriptors to optimize the performance. Figure 2 is an example of matched SIFT features points between two adjacent frames with the upper bound of the difference between the descriptors being 0.3.

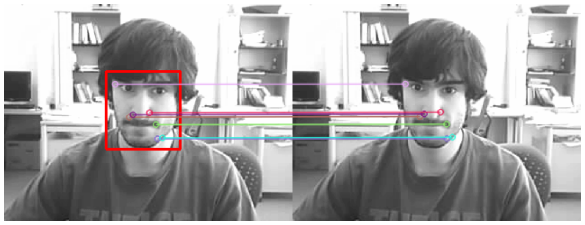

Figure 2. Matched SIFT feature points between two adjacent frames with the upper bound of the difference between the descriptors being 0.3.

### 2.6.2 Position and Size of the Object

With the matched key points in the current frame, we use relative location of the feature points in the region of the previous frame to approximate the position and size of the object in the current frame, which is the same as the method in [2]. This is built on the assumption that the relative position of the matched key points between two adjacency frames remains the same with the motion of the object. If there are multiple correspondences, least square method is performed to generate an optimized result. The method performs well even if the person's face rotates, wears hat or is shielded. The result for each case is shown in Figure 3.

However, the relative position method doesn't perform well if the object's motion is more complex or if there is noise in the matched key points. In this case, we tried other algorithms. First we tried homogeneous transform *i.e.* fitted four corners of the object rather than $w, h$ for each frame. Then we tried a simple mean algorithm. We take the mean position of all feature points as the measured position. These two algorithm are less accurate (more vibrations and inaccuracy of the area) but more robust (can still stick to the object) to the variation of motion. However, due to the goal of masking people, we can not adopt those methods because they are more likely to reveal part of the person's face.

## 2.7. Detailed Parameters

Based on the performance of our trials, we decided our parameters to be the following.

| Parameter Name | Value |
|---|---|
| Process noise covariance ($q^2$) | $1.0 \times 10^{-5}$ |
| Measurement noise covariance ($r^2$) | $1.0 \times 10^{-5}$ |
| Upper bound of the difference between the descriptors | 0.3 |

Table 1. Value of each parameters

# 3. Experiments

## 3.1. Dataset for Testing

In this project, we will use *OccludedFace2* in public Video Object Tracking [5] dataset on Kaggle to test our object tracking model. In this data sequence, the face will move, rotate and be shielded. It can more comprehensively test the performance of a tracker.

## 3.2. Evaluation Methodology

To measure the performance of one object tracking algorithm quantitatively, first we introduce two common evaluation metrics [6].

### 3.2.1 Precision

One of the widely used evaluation metric for object tracking is related to error of center location. In this method, we calculate the Euclidean distance between evaluated center and ground truth. Meanwhile, a huge error in one frame may lead to a great increase of average error distance. Hence, we will plot number of frames whose distance is under different threshold to compare different algorithms, which may reflect the performance in a more comprehensive aspect.

### 3.2.2 Overlap Rate

Another common evaluation method is the overlap score. We make $s_t$ represent the bounding box area and $s_e$ represents evaluated bounding box area. The overlap score is defined as $S = \frac{|s_t \bigcap s_e|}{|s_t \bigcup_e|}$, where $\bigcap$ and $\bigcup$ mean the intersection and union of two block areas. We use the average overlap score to measure performance. However, to test the performance more precisely, we choose different thresholds and plot ratio of frames which have scores higher than them.

To test the performance, we compare our model with tracker with optical-flow method and CSRT tracker.

### 3.3. Qualitative Evaluation Results

The tracker is applied to sequences from our dataset. Here,we just present some representative results in Figure 3. From the results, we can find that the tracker shows good spatial robustness since it can detect faces correctly even after high-speed motion. Meanwhile, when the man shields his face, the tracking area will become smaller and then restored to original state.

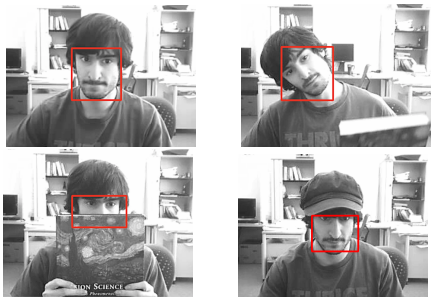

Figure 3. Results of Face Sequences

### 3.4. Quantitative Evaluation Results

In this part, we will compare three trackers with precision scores and overlap range scores. First, we calculate the average performance of them.

| Tracker | Avg. Precision | Avg. Overlap Rate |
|---|---|---|
| Our Tracker | 10.64 | 0.730 |
| Optical Flow | 19.99 | 0.519 |
| CSRT Tracker | 6.67 | 0.788 |

Table 2. Performance of trackers.

From above, our tracker performs much better than Optical Flow method in both ways, our model alleviates the noise caused by other objects blocking the human face with the introduction of kalman filter. For overlap rate, our model performance is close to that of CSRT tracker while It has a gap with CSRT in precision. After analysis, we find that in some frames, our tracker enlarge sizes of tracking areas so that the error distances will become pretty large, which makes precision performance poor. In order to test performance thoroughly, we test frames with different thresholds as in Section 3.2.

From the precision performance figure, the curve for our model is close to that of csrt tracker, which means that in most frames, our tracking results are within acceptable range. And for some large thresholds, our model produces large errors, which is the reason for poor average precision score. For the overlap rate performance, we can also find a comparable performance with CSRT tracker.

In conclusion, our model has the similar performance with CSRT tracker and much better one than optical flow method. However, when the face is under high rate motion,

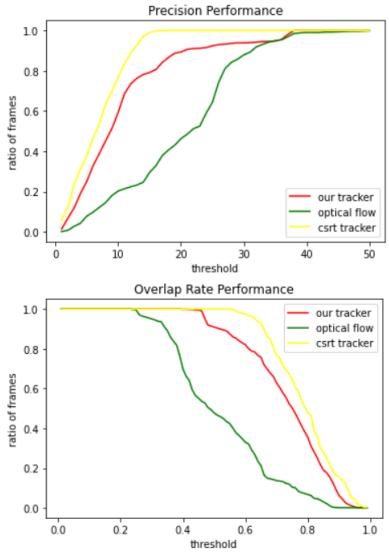


Figure 4. Performance of Trackers

the precision will be influenced in a large scale, but it is able to restore after that.

## 4. Implementation

Our basic structure is built similar as the structure in [2]. Inspired by [2], we implemented the functions for finding and matching features (a wrapping of openCV function for our own needs), converting to relative features and least square optimization for getting the measurement. We also implemented the framework for the algorithm, including image IO and interactions between the functions. To improve the performance and robustness, we tuned parameters and tested on different images sets to fit our own model.

Through implementation, we utilized the underlying kalman filter class, SIFT extraction and matching method in openCV library.

## References


[1] Adrian Rosebrock. Opencv object tracking, Jul 2018.

[2] Young Min Kim. Object tracking in a video sequence. *CS*, 229:366–384, 2007.

[3] Yifu Zhang, Chunyu Wang, Xinggang Wang, Wenjun Zeng, and Wenyu Liu. Fairmot: On the fairness of detection and re-identification in multiple object tracking. *arXiv preprint arXiv:2004.01888*, 2020.

[4] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[5] K Scott Mader. Video object tracking, Jul 2018.

[6] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1834–1848, 2015.