VISUAL FRAMING FEEDBACK FOR DESKTOP VIDEO CONFERENCING

[1]*Chen Wu,* [2]*Ramin Samadani,* [2]*April Mitchell,* [2]*Mary Baker,* [2]*Dan Gelb*

[1]Stanford University, Dept. of Electrical Engineering, [2]HP Labs, Mobile & Immersive Experiences

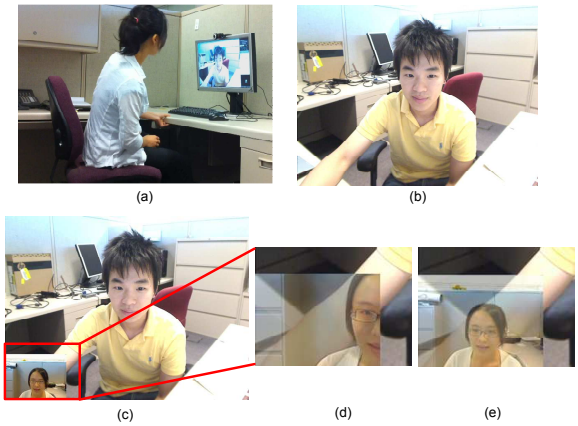

Fig. 1. Out-of-frame detection and feedback.

ABSTRACT

Desktop video conferencing participants are often poorly positioned in their outgoing video – unaware of their appearance on camera. We combine face detection, feature tracking and motion detection for automatic real-time detection of poorly framed participants and subsequently provide *framing feedback* by compositing their incoming and outgoing video streams. Our solution provides participants visual feedback *only* while they have framing problems. Otherwise the display shows *only* the remote participants, allowing users to focus fully on the conference. We analyze the system components and describe a user study that found advantages of our approach over the existing *mirror window* solution.

Index Terms— video conferencing, video analysis.

1. INTRODUCTION

Consider the case of a user having a video conference using a PC desktop or laptop and a web-cam with one or more remote users. Often, the user is poorly positioned or *out-of-frame* so that only a portion of the face is seen in the outgoing video stream. Existing solutions use a mirror window that shows the user's outgoing video. This mirror window 1) occludes a portion of the remote participants video feed, 2) may distract the user - causing neglect of or missed cues from the other conference participants, and 3) fails to alert the user if the user has become too accustomed to the presence of the window and misses the feedback provided.

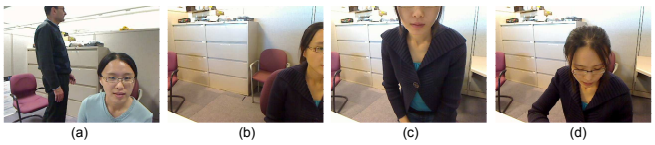

Fig. 2. Failure cases with a single module. (a) With only motion detection, out-of-frame falsely triggers when there is a person walking by. Face detector fails given (b)(c) partially occluded face, and (d) difficult head pose.

We developed feedback methods for one-to-one and one-to-multiple video conferencing layouts. Fig. 1 shows out-of-frame feedback for the one-to-one case. The user is video conferencing with a remote person as in (a). Without any feedback the screen shows only the remote person (b). The window with feedback is shown in (c). With our system, when there is a framing problem a feedback window blends onto the screen (d). Then the feedback fades away when the user goes back to position (e). Our approach frees screen real estate and allows participants to focus their full attention on the conference.

The out-of-frame condition is detected using video analysis. For feedback, the incoming and outgoing video streams are composited [1] *only* when out-of-frame is detected. The detection is real-time, light-weight (preserves computing resources for other tasks), and provides results reliable enough to compete with the simple mirror window solution.

2. SOFTWARE ARCHITECTURE AND ALGORITHM

Our system combines face detection, feature tracking and motion detection for out-of-frame detection because there are failure situations if the individual modules are applied alone. Face or motion activity bounding boxes trigger out-of-frame feedback if they intersect a pre-defined image boundary region. If only motion detection is applied, out-of-frame is falsely triggered by background motion such as a person walking by (Fig. 2(a)). We use a good state-of-the art face detector [2], but by itself it does not reliably detect out-of-frame. In addition, face detection fails for cases important to this application such as the partially out-of-frame or oc-

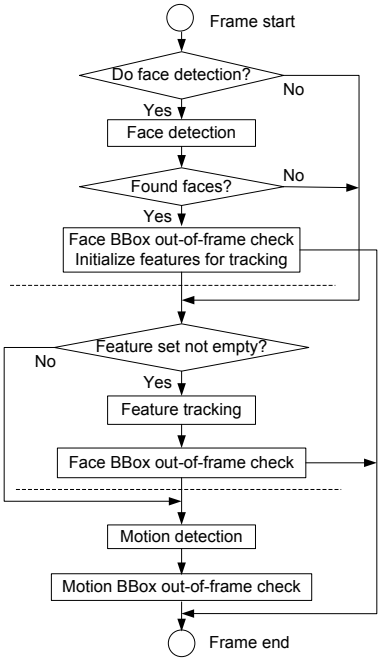

Fig. 3. The flowchart for out-of-frame detection. The system switches between three modules: face detection, feature tracking, and motion detection. BBox: bounding box.

cluded faces shown in Fig. 2(b)(c) or in difficult poses as in Fig. 2(d). By itself, face detection is also unnecessarily expensive. Table 1 describes advantages and disadvantages of the face detection, feature tracking, and motion detection modules when used individually.

2.1. System design and implementation

For overall robustness and lightweight computation, our system switches between the three modules, as shown in Fig. 3.

To overcome its shortcomings, we only perform face detection every few frames and augment it with feature tracking to localize the face in frames when face detection is not performed or has failed. Feature tracking computes much faster than face detection. Harris corner features [3] are initialized within the face bounding box after each face detection. The features are tracked with optical flow using the iterative Lucas-Kanade method with pyramids [4]. Both feature detection and feature tracking use the OpenCV library [5], while our software system is implemented in a video conferencing framework [6] that allows rapid prototyping.

Feature tracking is usually reinitialized by successful face detection. But feature tracking fails if not reinitialized for an extended period due to face detection repeatedly failing, since noise and other distortions cause too few high confidence features that fit the translational motion model to remain valid. When both face detection and feature tracking fail, motion detection provides a fallback mechanism to detect out-of-frame. Thresholded grayscale absolute frame difference ($|g(x, y, t_i) - g(x, y, t_{i-1})| > 30$) is used to detect motion at a pixel. Out-of-frame is triggered when motion is detected within a pre-defined boundary of the video frames. Motion detection is used as a last resort out-of-frame detector since it creates false alarms when there is background motion, such as a person walking behind the user.

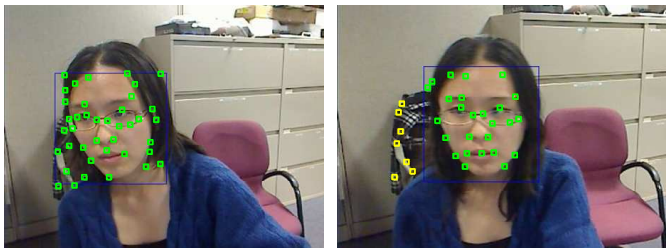

Fig. 4. Face features and background features are detected. All the features are shown in green squares in the left image. Background features should be removed for the face motion calculation (yellow squares in the right image).

Ideally, face detection should run every frame but that generates high computational load. On the other hand, long frame intervals between face detections result in a higher chance of losing track of all features without being able to recover. We trigger face detection every 5th frame, which empirically gives a good balance between computation and tracking robustness. If a face is found, this re-initializes features for tracking, and the face position is reported. If no face is found, tracking continues with the existing set of features. The position of the face is calculated from the features' displacement computed using a RANSAC algorithm (Sec. 2.2). For any frames where both face detection and feature tracking fail, the system switches to motion detection.

2.2. Updating face location from feature tracking

The displacement (motion) of a valid set of tracked features determines the face position. The features initialized inside the rectangular face bounding box may include face features and background features. When the user moves, the features on the face move but the background features stay still (Fig. 4). Therefore, we need to differentiate between face and background features in order to calculate the displacement.

We use a RANSAC (random sample consensus) algorithm to estimate face motion in the presence of outliers (background features). We considered projective and affine camera models and the corresponding motion models (affine, translational) for translating and rotating heads. In the end, we adopted a simple translational motion model that was fast as well as effective:

$$x' = x + d, \tag{1}$$

Module	Advantages	Disadvantages
Face detection	True positive rate is high.	Slow computation. Fails with occlusions and difficult poses.
Feature tracking	Computes fast. Tracks face with occlusions, difficult poses, and fast motion.	Needs to be initialized with face detection.
Motion detection	Computes fast.	Easily triggered by background motion.

Table 1. Advantages and disadvantages of the three modules used individually.

where x and x' are the feature's positions in the first and current frame, correspondingly, obtained from tracking. The objective is to estimate the translation d from the set of valid face features. The modified RANSAC algorithm is shown in Alg. 1. The algorithm outputs the face feature set S_{best_cons} and the best model M_{best}. M_{best} includes the translation d and the error of S_{best_cons} on M_{best} (mean of $||x_i + d - x_i'||$ for the features in S_{best_cons}). S_0 denotes the whole set of features, including the features' original positions and positions from tracking in the current frame. N_0, between 20 and 40 for typical face sizes, is the number of features in S_0. Parameters for the RANSAC algorithm include K, N_1, N_2, where K is the number of iterations, N_1 is the number of features in a random sample, and N_2 is the minimum size of the consensus set. In our system, $K = 10$, $N_1 = 0.7N_0$ and $N_2 = 0.5N_0$. M_1 and $S_{consensus}$ are used to store the model and consensus set for each iteration. Δ is the max tolerance on the error between the observed feature translation and the current model.

Algorithm 1 Estimate face motion model with RANSAC.

for $i = 1$ to K **do**
  $S_1 \leftarrow$ Randomly selected N_1 features from the total N_0 features
  $M_1 \leftarrow$ the mean translation of features in S_1
  Calculate Δ based on M_1
  $S_{consensus}, S_{best\_cons}, M_{best} \leftarrow$ empty
  for each feature i in S_0 **do**
    if $\|$translation of feature $i - M_1 \|\leq \Delta$ **then**
      $S_{consensus} \leftarrow S_{consensus}+$ feature i
    end if
  end for
  if size of $S_{consensus} > N_2$ **then**
    if error of $S_{consensus}$ on M_1 $<$ error of S_{best_cons} on M_{best} **then**
      $S_{best\_cons} \leftarrow S_{consensus}$
      $M_{best} \leftarrow M_1$
    end if
  end if
end for

In our system, features are typically initialized every 5 frames from face detection so that the simple translational model for face motion is fairly accurate for the 5 frames ($\sim$160ms). However, when face detection fails, feature tracking is continued for potentially many frames. In this case, the features may be far from the original initialization positions, and the translation motion model has larger errors. To account for the variable model inaccuracy, we use an adaptive tolerance Δ on the model error, which is proportional to d_f, the distance between the current feature position and the initialization position: $\Delta = \alpha d_f$. In our system $\alpha = 0.4$, but empirically we found the algorithm result is not sensitive to the α value.

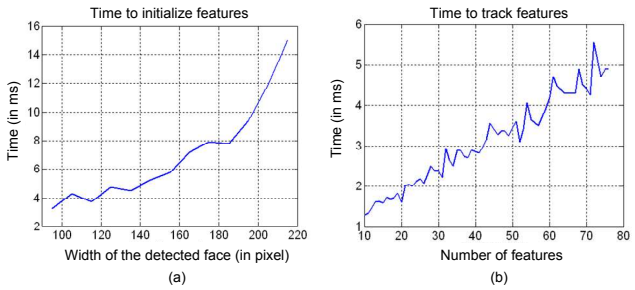

Fig. 5. Execution time for single-frame feature initialization and tracking. (a) Feature initialization time vs. face size. (b) Feature tracking time vs. number of features.

3. EVALUATION

Experiments with our system show that it achieves robust performance in out-of-frame detection. It correctly reports the face location even when the face is partially occluded (Fig. 2(b)(c)) and when the head pose is challenging to a face detector (Fig. 2(d)). When the user is in-frame, the system is not distracted by background motions (Fig. 2(a)). Sec. 3.1 discusses the algorithm efficiency and Sec. 3.2 discusses a user study showing preference for the new feedback method.

3.1. Performance of components

Running time statistics of the system modules were measured using an Intel Xeon 2.93 GHz dual CPU HP-Z800 workstation with 3.48 GB RAM running Microsoft Windows 7. The webcam streamed 640×480 pixel video at 30 fps.

Average face detection execution time, recorded for 260 frames, was 35.9ms with standard deviation 5.1ms. The detection times were similar whether there were 0, 1 or 2 faces

detected since the algorithm always scanned the entire image.

Fig. 5 shows the average time for feature initialization and tracking in single frames. After each face detection, features are detected in the face bounding box. Therefore, feature initialization time should be proportional to the face region area, and polynomial to the width of the detected face (square face region). This is verified by the experimental result in Fig. 5(a). Using 253 frames, for each face width (quantized into 10 pixel bins), feature initialization times are recorded. The mean feature initialization times are plotted in Fig. 5(a). The individual times tightly cluster near the mean (the mean of standard deviations of all bins is 2.2ms). Normally, the face size is between 100 to 180 pixels (the user does not get too close to the camera) so feature initialization time is usually below 10ms. Features are initialized at most every 5th frame when there is a successful face detection.

We recorded feature tracking time of 3071 frames with the corresponding number of features N_f (Fig. 5(b)). For each N_f, the mean tracking time of all frames with that N_f is calculated and plotted in the figure. Since pyramid optical flow is calculated on each feature location, tracking time should be linear to N_f, as confirmed by Fig. 5(b). The mean of standard deviations of all N_f is below our timer quantization of 1ms. Normally there are 20-40 features in the face region. So for most frames, feature tracking takes less than 3ms.

We collected motion detection time for 1018 frames. The mean detection time is 4.6ms with standard deviation 1.1ms.

The above performance evaluation supports our design choices. Feature tracking (initialization 10ms and tracking 3ms) is much faster than face detection (36ms), so it is a good alternative as long as it provides face location. Motion detection is also fast to compute (4.6ms) so it does not hamper the system's overall realtime operation. In practice, the system may be running in two modes: 1) The algorithm alternates between face detection and feature tracking. In our system we chose to do face detection every 5th frame. So on average, each frame's running time is $(36+10+4*3)/5 = 11.6$ms. 2) There is no face detected and the algorithm only does motion detection for each frame. In this case, each frame's average running time is 4.6ms.

3.2. User preferences

The new detection and feedback solution was incorporated into a prototype video conferencing system for users to compare with the industry standard mirror window solution. While details of the user study are available online [7], we present a brief summary here. Fourteen users ranging in age from 20 to 60 and split equally by gender participated. Several methods of feedback with varied sizes and positions of the feedback window were presented in random order during a one-to-one and then one-to-four video conferencing setup. For comparison, the standard always-on mirror window was one of the presented methods in each setup. The users reached for a peripheral object that caused them to go out of frame to trigger the out-of-frame feedback. We found variation in preference about the size and location of the windows, but overall there was roughly a two to one preference in both the one-to-one and one-to-four conferencing layouts for an alternative feedback solution instead of the mirror window.

4. CONCLUSIONS

In this paper, we describe a realtime out-of-frame detection technique. This work enables a new feedback method for video conferencing that allows users to focus on the conference itself by providing visual feedback only when the user is out of frame. The challenges for the algorithm design are the reliability and realtime requirements. For this purpose, face detection, feature tracking, and motion detection are combined. We conducted a user study with the implemented system to identify the preferred out-of-frame feedback types. Even though there were different preferences about window sizes and locations, an alternative feedback was preferred by a two-to-one ratio over the usual mirror window. Since the preferred type of feedback consisted of alpha blending of video in a window, almost all users could be satisfied within the same uniform software architecture by providing control for a alpha blended, movable, resizable window in the user interface for the conferencing system.

5. REFERENCES

[1] T. Porter and T. Duff, "Compositing digital images," *ACM SIGGRAPH Computer Graphics*, 1984.

[2] D. Greig, "Video object detection speedup using staggered sampling," in *Workshop on Appl. of Comp. Vision (WACV)*, 2009.

[3] J. Shi and C. Tomasi, "Good features to track," in *Proc. CVPR Conf.*, 1994, pp. 593–600.

[4] Jean-Yves Bouguet, "Pyramidal implementation of the lucas kanade feature tracker description of the algorithm," 2000.

[5] OpenCV, "http://opencv.willowgarage.com/wiki/," .

[6] D. Tanguay, D. Gelb, and H. Baker, "Nizza: A framework for developing real-time streaming multimedia applications," Tech. Rep. HPL-2004-132, HP labs, 2004.

[7] A. Mitchell, M. Baker, C. Wu, R. Samadani, and D Gelb, "How do I look? An evaluation of visual framing feedback in desktop video conferencing," Tech. Rep. HPL-2010-175, HP labs, 2010, http://www.hpl.hp.com/techreports/2010/HPL-2010-175.pdf.