Tracking Features with Large Motion

Cheng-Hung Ko[2], Yu-Pao Tsai[1,4], Yi-Ping Hung[1,2,3]

[1]Institute of Information Science, Academia Sinica, Taiwan
[2]Department of Computer Science and Information Engineering, National Taiwan University, Taiwan
[3]Institute of Networking and Multimedia, National Taiwan University, Taiwan
[4]Department of Computer and Information Science, National Chiao Tung University, Taiwan

E-mail: chenghung@csie.org, yptsai@iis.sinica.edu.tw, hung@csie.ntu.edu.tw

Abstract

This paper addresses feature tracking when frame-to-frame motion is too large that the popular pyramidal Kanade-Lucas-Tomasi (KLT) feature tracker does not work. To solve this problem, we estimate the motion at the deepest pyramid level by matching the horizontal (and vertical) characteristic curves of the consecutive images. To compute the motion estimates efficiently and effectively, we use dynamic programming to minimize the cost function. These motion estimates will serve as the coarse motion at the deepest pyramid level which makes the residual motion small enough such that the feature tracker can work well. Experiments show that our method can make the feature tracker suitable for features with large motion.

1 Introduction

Feature tracking is an important issue in many computer vision applications. For examples, Sand and Teller [9] aligned a pair of videos by using locally weighted regression to interpolate and extrapolate high-likelihood image correspondences. To solve the emotion recognition problem in live video, Michel and Kaliouby [11] employed an automatic facial feature tracker to perform face localization and feature extraction. Tomasi and Kanade [10] solved the shape and motion recovery problem with a factorization method, which required a set of corresponding feature points from an image sequence. To achieve these applications, a robust and reliable feature tracker is required.

Let I and J be two consecutive images in an image sequence. For an image point $\mathbf{x} = [u \quad v]^T$, $I(\mathbf{x})$ and $J(\mathbf{x})$ are the intensity values of $\mathbf{x}$ on images I and J, respectively. Given a feature point on I, the goal of the tracker is to find the correspondence on J. One of the available approaches is to estimate the displacement vector $\mathbf{d}$ that minimizes the sum of square differences (SSD) residual function

$$\epsilon(\mathbf{d}) = \sum_{\mathbf{x} \in W} (I(\mathbf{x}) - J(\mathbf{x} + \mathbf{d}))^2, \tag{1}$$

where W is a small integration window centered at the feature point. Here, the assumption is that the sampling rate of the camera is sufficiently high that the frame-to-frame motion is not too large, and the shutter speed is also fast enough to capture any movement almost without blur and smear. To solve the feature tracking problem using the SSD method, Tomasi and Kanade [2] developed a feature tracker based on the Newton-Raphson scheme proposed by Lucas and Kanade [1]. Then, Shi and Tomasi [3] proposed an affine model for the feature tracker with a feature selection criterion to automatically select features that can be tracked well. Following this work, Tommasini et al. [4] proposed an automatic approach, called $X84$, to rejecting spurious features. In order to allow the tracker to handle larger motion, people usually use a pyramidal implementation of the KLT feature tracker in practice [6, 7].

The above-mentioned methods provide sufficient accuracy and robustness when the frame-to-frame motion is not too large. However, these methods are not suitable for the image sequences containing heavy vibration, which can hardly be avoided when using a handheld or mobile video camera. In this paper, we propose a method to extend the pyramidal KLT feature tracker to deal with the situation where images I and J are taken from widely different viewpoints.

This paper is organized as follows. In Section 2, we briefly describe the KLT feature tracker. The pyramidal implementation of this feature tracker is also presented. Section 3 introduces the algorithmic details about our method, which estimates the coarse motion at the deepest pyramid level by using dynamic programming to match the characteristic curves. Experiments on realistic images are shown in Section 4, while the conclusions are given in Section 5.

2 KLT feature tracker

When the video camera is perfect and the translational model can completely describe the inter-frame motion, we can assume that a small image patch on I can be shifted to J by using a small displacement vector, i.e., there is a displacement vector which makes the residual function in Equation (1) become zero. However, in practice, the requirements usually can not be satisfied. As a result, the problem is to find the displacement vector $\mathbf{d}$ that minimize the SSD residual function.

The differential approach [12] is applied to minimize the residual function in Equation (1). The first order Taylor expansion of $J(\mathbf{x}+\mathbf{d})$ about the point $\mathbf{x}$ is used to approximate $J(\mathbf{x}+\mathbf{d})$. Then, the first derivative of ϵ with respect to $\mathbf{d}$ is taken to be zero. Let $\mathbf{g} = \left[\frac{\partial I}{\partial u} \quad \frac{\partial I}{\partial v}\right]^T$, we can obtain the linear system

$$G\mathbf{d} = \mathbf{e}, \tag{2}$$

where G is a 2×2 matrix

$$G = \sum_W \mathbf{g}\mathbf{g}^T, \tag{3}$$

and $\mathbf{e}$ is a 2×1 vector

$$\mathbf{e} = -\sum_W (I - J)\mathbf{g}. \tag{4}$$

The displacement vector $\mathbf{d}$ for a feature point $\mathbf{x}$ is computed by using Equation (2) (i.e., $\mathbf{d} = G^{-1}\mathbf{e}$). To get an accurate solution, KLT feature tracker iterates the above procedure based on the Newton-Raphson scheme [1]. Notice that the translational model described in this section is preferable to feature tracking, while the affine model proposed in [3] should only be used to build a reliable rejection scheme [4, 6].

Automatically selecting good features is also an important issue. The feature selection criterion of KLT feature tracker is based on whether Equation (2) can be solved reliably. Let λ_1 and λ_2 be the two eigenvalues of G. KLT feature tracker classifies the image point as a good feature if $\min(\lambda_1, \lambda_2) > \lambda$, where λ is a predefined threshold.

2.1 Pyramidal implementation of the KLT feature tracker

In this subsection, we will describe the algorithmic details of using the image pyramid to enhance the KLT feature tracker to handle relative large motion. Consider two consecutive images I and J of an image sequence. We first construct their pyramid representations in a recursive way. Let I_1 and J_1 (at pyramid level one) be the original images I and J, respectively. Let I_2 be the image obtained by downsampling I_1, I_3 be the image by downsampling I_2, and so on. Denote the height of the image pyramid as L_{max}. We shall continue the construction of the image pyramid of I until we obtain image $I_{L_{max}}$ at the deepest pyramid level L_{max}. Similarly, we can obtain images $J_2, J_3, \ldots, J_{L_{max}}$ for image J.

After constructing the image pyramids of I and J, we apply the pyramidal KLT feature tracker whose details are shown below. Let $\mathbf{d}_{L_{max}}$ be the displacement vector at the deepest pyramid level L_{max}. Then, the first step computes $\mathbf{d}_{L_{max}}$ by minimizing the residual function

$$\epsilon(\mathbf{d}_{L_{max}}) = \sum_{\mathbf{x}\in W} (I_{L_{max}}(\mathbf{x}) - J_{L_{max}}(\mathbf{x}+\mathbf{d}))^2, \tag{5}$$

where W is a small integration window centered at the feature point on the deepest pyramid level L_{max}. Notice that Equation (5) is similar to Equation (1). The second step propagates the displacement vector obtained from the deeper level $k+1$ to level k by minimizing the function

$$\epsilon(\delta\mathbf{d}_k) = \sum_{\mathbf{x}\in W} (I_k(\mathbf{x}) - J_k(\mathbf{x} + s\mathbf{d}_{k+1} + \delta\mathbf{d}_k))^2, \tag{6}$$

where $\mathbf{d}_{k+1}$ is the displacement vector at level $k+1$, and s is the downsampling factor used in constructing the image pyramids. Here, $s\mathbf{d}_{k+1}$ denotes the coarse motion at level k, and $\delta\mathbf{d}_k$ denotes the fine motion at level k. Notice that the displacement vector $\mathbf{d}_k$ at level k is the sum of $s\mathbf{d}_{k+1}$ and $\delta\mathbf{d}_k$. The second step described above will be repeated until the estimate of $\mathbf{d}_1$ at level one is obtained. This pyramidal implementation of the KLT feature tracker has been adopted by Bouguet [6], and also by Birchfield [7].

3 Accommodating very large motion

The assumption of using the approximation of the first order Taylor expansion in the KLT feature tracker [6] is that the L^2-norm of the displacement vector is small enough (for example, less than some threshold η). Consider a feature point whose true displacement vector is $\mathbf{d}^*$. Let $\mathbf{d}_k^* = \frac{\mathbf{d}^*}{s^{k-1}}$, which is the true displacement vector of the feature point at level k. When constructing the image pyramids, the use of a large enough L_{max} will make $\mathbf{d}_{L_{max}}^*$ small enough such that the assumption mentioned above can be satisfied at the deepest pyramid level, i.e., $|\frac{\mathbf{d}^*}{s^{L_{max}-1}}|_2 < \eta$, where $|.|_2$ denotes the L^2-norm.

In practice, L_{max} is chosen to be a small number; otherwise, the image size of $I_{L_{max}}$ and $J_{L_{max}}$ will be too small to carry enough details for each feature. For example, Figure 1 shows four corresponding integration windows, centered at the same feature point, on four different pyramid levels. It can be seen that the feature point dissolves when the height of the image pyramid is too large. When L_{max} is small and $\mathbf{d}^*$ is large, the assumption of $|\frac{\mathbf{d}^*}{s^{L_{max}-1}}|_2 < \eta$ may not hold

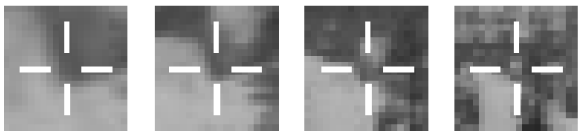


Figure 1: An example used to explain why the height of the image pyramid is usually quite limited. From left to right, we show the integration windows, centered at the same feature point, on pyramid level 1, 2, 3, and 4, respectively. Notice that the sizes of the integration windows are the same at different levels.

anymore. For those cases, our method provides an effective and efficient solution by computing the motion estimates d_x and d_y (one for x-axis and the other for y-axis) at the deepest pyramid level L_{max}. The effect of computing d_x and d_y is to provide a coarse motion at level L_{max} which makes the residual motion small enough to satisfy the assumption. For example, consider a feature point $\mathbf{x} = [u \quad v]^T$. To compute the displacement vector $\mathbf{d}_{L_{max}}$ in the first step of the pyramidal KLT feature tracker, we now use Equation (6), where $[d_x(u) \quad d_y(v)]^T$ will serve as the coarse motion at the deepest pyramid level.

3.1 Characteristic curves

Define φ_x^I as the characteristic curve for x-axis computed from image I. Each value on the curve equals to the average of the pixel values of the corresponding column on image I, i.e.,

$$\varphi_x^I(i) = \frac{1}{M} \sum_{\mathbf{s} \in col_i} I(\mathbf{s}) \quad \forall i = 1, \ldots, N, \tag{7}$$

where col_i represents the i^{th} column, M is the number of rows in the image, and N is the number of columns in the image. Similarly, we can compute the characteristic curve φ_y^I for y-axis from image I by using the function

$$\varphi_y^I(i) = \frac{1}{N} \sum_{\mathbf{s} \in row_i} I(\mathbf{s}) \quad \forall i = 1, \ldots, M, \tag{8}$$

where row_i represents the i^{th} row. One example of the two characteristic curves φ_x^I and φ_x^J for images I and J has been shown on the top of Figure 2(a) and Figure 2(b), respectively.

After the four curves φ_x^I, φ_y^I, φ_x^J, and φ_y^J have been computed for the consecutive images I and J, we compute the motion estimate d_x by matching the two characteristic curves φ_x^I and φ_x^J. In the same manner, the motion estimate d_y can be computed by matching the other two characteristic curves φ_y^I and φ_y^J. Similar idea of using the characteristic curves has been adopted by Piva et al. [5] to solve the video stabilization problem. However, in their work, no feature points are used.

3.2 Motion estimation at the deepest pyramid level

In order to match two one-dimensional curves, many different methods have been proposed in the literature. The differential approach [12] has also been used to match two one-dimensional curves. But this approach is not suitable to match curves when very large moves are allowed. Minimizing the mean square error (MSE) between the two curves [5] might be the simplest solution. However, this method can only give a global displacement between these two curves, and the computational cost of this method is high. Boykov et al. [8] proposed two algorithms (swap move and expansion move) that used graph cuts to solve the energy minimization problem by computing a local minimum. These two algorithms are both suitable to match curves even when very large moves are allowed. However, computing the local minimum via graph cuts needs to perform iterations until the convergence of the energy is occurred. Again, the computational cost is too high for an real-time feature tracker.

In this paper, we adopt the technique of dynamic programming, which makes an exhaustive search for the optimal piecewise match sequences between the two curves φ_x^I and φ_x^J. For brevity, we only show how to compute the motion estimate d_x by matching the two characteristic curves φ_x^I and φ_x^J. The motion estimate d_y can be computed in the same manner. The algorithmic details of the motion estimator are shown below. Let set $\mathcal{P}$ be the domain of the function (or curve) φ_x^I, i.e., $\mathcal{P} = (1, \ldots, N)$, where N is the number of columns in the image. For each element $p \in \mathcal{P}$, we need to assign a label $f(p)$, which represents the displacement, to the element p. Moreover, we allow that the motion estimator can assign the label $\mathcal{O}cc$ (abbreviation of "occlusion") to the element p when the element p is considered to be occluded. The goal of the motion estimator is to find the best labeling f that assigns a label $f(p)$ to each element $p \in \mathcal{P}$.

Define vector $\mathcal{A} = (a_1, a_2, \ldots, a_n)$ be the ordered set of element $p \in \mathcal{P}$ where $f(p) \neq \mathcal{O}cc$. The motion estimator finds the optimal labeling f by minimizing the cost function

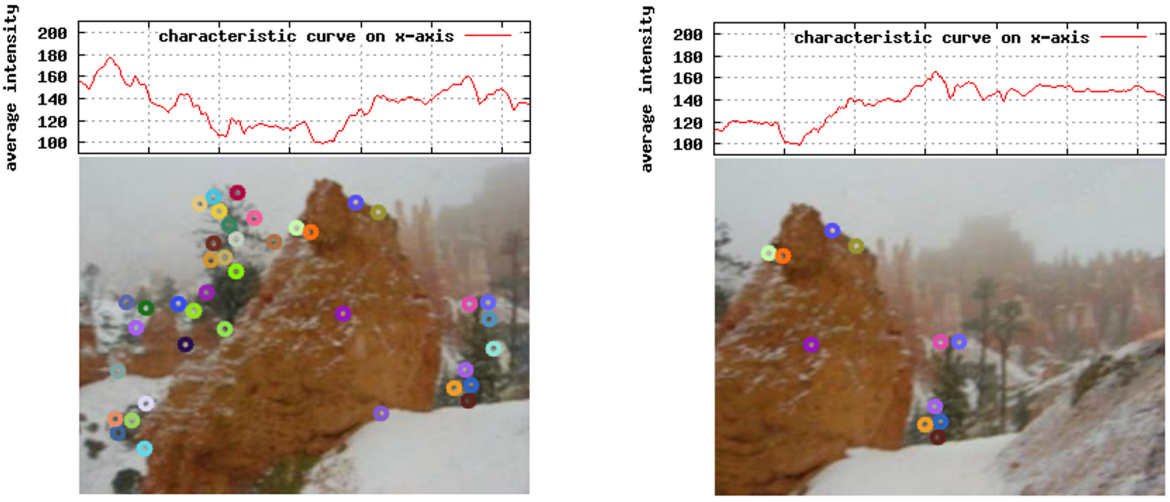
$$C(f) = C_{data}(f) + C_{smooth}(f) + C_{occ}(f), \tag{9}$$

where

$$C_{data}(f) = \sum_{\forall p \in \mathcal{A}} [\varphi_x^I(p) - \varphi_x^J(p + f(p))]^2, \tag{10}$$

$$C_{smooth}(f) = \sum_{\forall a_i, a_{i+1}} \gamma_s |f(a_i) - f(a_{i+1})|, \tag{11}$$

$$C_{occ}(f) = \sum_{\forall p \in \mathcal{P}/\mathcal{A}} \gamma_o. \tag{12}$$



(a) Image I (b) Image J

Figure 2: One example of two consecutive images I and J in an image sequence are shown on the bottom of each image. The characteristic curves for x-axis computed from images I and J are shown on the top of each image. There are 40 features automatically selected to be tracked on iamge I. On image J, we show the tracking results of our feature tracker. Most of the lost features are due to the reason that they are outside the image boundary.

The penalty γ_s in Equation (11) is used to penalize the situations where discontinuity is occurred. On the other hand, the penalty γ_o in Equation (12) serves as a threshold that affects whether the motion estimator should assign a label $\mathcal{O}cc$ to the element $p \in \mathcal{P}$.

The data term $C_{data}(f)$ is inversely proportional to the possibility of assigning every element $p \in \mathcal{A}$ a label $f(p)$. This possibility is measured based on the similarity between the values of the two curves φ_x^I and φ_x^J on p and $p + f(p)$, respectively. The smooth term $C_{smooth}(f)$ is used to ensure that the labeling is piecewise smooth, while $C_{occ}(f)$ makes the motion estimator assign label $\mathcal{O}cc$ to a element p only when p seems to be occluded.

We minimize the cost function via dynamic programming. After finding the optimal labeling, the motion estimator computes the motion estimate d_x by using the function

$$d_x(i) = \frac{1}{s^{L_{max}-1}} f(i) \quad \forall i \in \mathcal{A}, \tag{13}$$

where s is the downsampling factor and L_{max} is the height of the image pyramids. For those elements in $\mathcal{P}/A$, we compute their motion estimates by interpolating the displacements of the elements in the neighborhood.

Once the coarse motion d_x and d_y are determined by our motion estimator, it can be used to compute $\mathbf{d}_{L_{max}}$ at the deepest pyramid level L_{max} by using Equation (6). Figure 2(a) and Figure 2(b) show an example of the tracking results.

3.3 Feature tracker with pre-checking

To improve the efficiency, we introduce a feature rejection scheme by pre-checking the quality of the features based on the motion estimates d_x and d_y. The goal of the pre-checking step is to identify those features which would obviously fall outside of the image boundary in the subsequent image. When a feature is rejected during the pre-checking step, the feature tracker will directly denote that this feature has been lost. Therefore, no computational power is wasted on tracking such features.

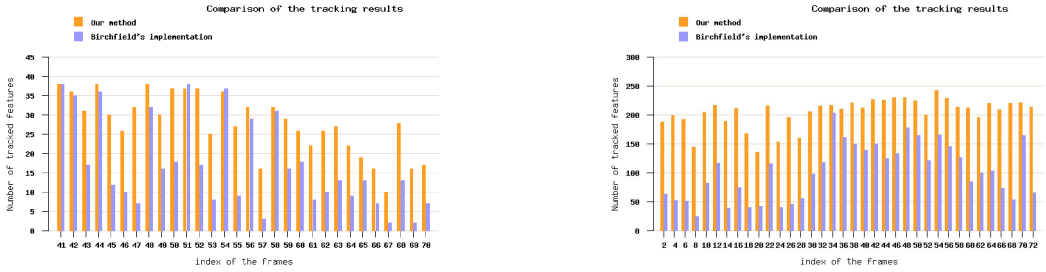
Consider a feature point $\mathbf{x} = [u \quad v]^T$. During the pre-checking step, the feature tracker classifies $\mathbf{x}$ as a lost feature when one of the following conditions is satisfied.

$$\begin{cases} u + d_x(u) > N \\ v + d_y(v) > M. \end{cases} \tag{14}$$

Notice that pre-checking is an intrinsic property when using a motion estimator to provide a coarse motion in the deepest pyramid level. After the feature points are displaced by the coarse motion, the feature tracker can track these features only when they are still inside the image boundary.

4 Results and comparisons

Our method has been widely tested in a series of realistic image sequences. Here, we compare the tracking results



(a) For *Snow* image sequence

(b) For *Building* image sequence

Figure 3: Comparison between the number of the tracked features. Powderblue: the tracking results by using Birchfield's implementation. Orange: the tracking results improved by using our method.

obtained by our feature tracker with those by Birchfield's implementation [7]. We do the experiments on a Pentium M 1.3GHz laptop.

Snow (see Figure 2 and Figure 4). Consider the test images taken from a 71-frame video clip (320×240 pixels for each frame), which can be downloaded at `http://iss.bu.edu/litvin/stabilization/`. Because this video was acquired using a handheld video camera, some heavy vibration occurred, which made it difficult to track features with high accuracy and reliability. In the experiments, 40 features were selected automatically in the first frame. If some features are lost in the subsequent frames, we replace them with the new ones. Figure 3(a) shows the number of tracked features after the 40^{th} frame, where the video contains heavy vibration. It can be seen that our method performs much better. Table 1 compares the running time of tracking the entire image sequence. The running times are evaluated by computing the average running time of five experiments. In this experiment, the computational costs of these two feature trackers are similar, even though we adopt the dynamic programming to estimate the coarse motion. The reason is due to the pre-checking step, which improves the efficiency.

Table 1: Comparison between the running times

Birchfield's	**Ours**
6.347 sec.	6.370 sec.

Building. (see Figure 5) Consider the 73-frame video clip (480×320 pixels for each frame), which contains heavy vibration. In this experiment, 256 features are selected automatically in the first frame. Again, we replace the lost features with the new ones. Figure 3(b) compares the number of tracked features for even frames. Table 2 compares the running time of tracking this image sequence. In this experiment, our feature tracker performs more efficient than Birchfield's implementation by using the pre-checking method to reject the features which are potentially outside the image boundary in the subsequent frame.

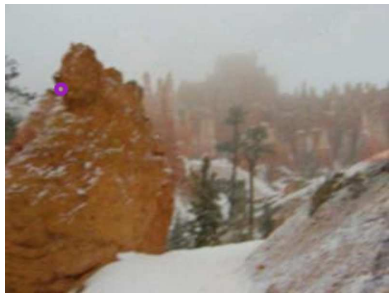


Figure 4: The tracking result of the same image pair in Figure 2 by using Birchfield's implementation. The feature tracker fails due to the large motion between this pair of consecutive images. Notice that the tracked feature is a spurious correspondence.

Table 2: Comparison between the running times

Birchfield's	**Ours**
20.841 sec.	17.370 sec.

5 Conclusions

We employ an effective and efficient motion estimator to give pyramidal KLT feature tracker a coarse motion for each feature at the deepest pyramid level. The motion estimator computes the coarse motion by using dynamic programming to minimize the cost function. At the price of a

(a) Image I (b) Birchfield's implementation (c) Our method

(d) Image I (e) Birchfield's implementation (f) Our method

Figure 5: (a)-(c) and (d)-(f) show two examples of the tracking results. Notice that many of the tracked features by using Birchfield's implementation are spurious correspondences.

little computational cost, our method can track features in image sequences that contain heavy vibration.

Acknowledgement

This work is supported in part by the grants of NSC -93-2213-E-002-037.

References

[1] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," *International Joint Conference on Artificial Intelligence*, pp. 674-679, 1981.

[2] C. Tomasi and T. Kanade, "Detection and tracking of point features," Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.

[3] J. Shi and C. Tomasi, "Good features to track," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593-600, June 1994.

[4] T. Tommasini, A. Fusiello, E. Trucco and V. Roberto, "Making Good Features Track Better," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 178-183, June 1998.

[5] S. Piva, M. Zara, G. Gera, and C. S. Regazzoni, "Color-based Video Stabilization for Real-Time On-Board Object Detection on High-Speed Trains," *IEEE Conference on Advanced Video and Signal Based Surveillance*, pp. 299-304, July 2003.

[6] J. Y. Bouguet, "Pyramidal Implementation of the Lucas Kanade Feature Tracker," The paper is included into Intel Open Source Computer Vision Library (OpenCV) distribution.

[7] S. Birchfield. KLT: An Implementation of the Kanade-Lucas-Tomasi Feature Tracker. Software available at `http://www.ces.clemson.edu/$\sim$stb/klt/`.

[8] Y. Boykov, O. Veksler, and R. Zabih, "Fast Approximate Energy Minimization via Graph Cuts," *IEEE transactions on Pattern Analysis and Machine Intelligence*, Vol. 23, no. 11, pp. 1222-1239, 2001.

[9] P. Sand and S. Teller, "Video Matching," *ACM Transactions on Graphics*, Vol. 23, No. 3, pp. 592-599, July 2004.

[10] C. Tomasi and T. Kanade, "Shape and motion from image streams under orthography: A factorization method," *International Journal of Computer Vision*, Vol. 9, pp. 137-154, November 1992.

[11] P. Michel and R. E. Kaliouby, "Real Time Facial Expression Recognition in Video using Support Vector Machines," *International conference on Multimodal interfaces*, pp. 258-264, November 2003.

[12] R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision*, Vol. 2, Addison-Wesley, 1993.