Interactive Line Drawing Recognition and Vectorization with Commodity Camera

Pradeep Kumar Jayaraman
School of Computer Engineering
Nanyang Technological University, Singapore
pradeepk001@e.ntu.edu.sg

Chi-Wing Fu
School of Computer Engineering
Nanyang Technological University, Singapore
cwfu@ntu.edu.sg

ABSTRACT

This paper presents a novel method that interactively recognizes and vectorizes hand-drawn strokes in front of a commodity webcam. Compared to existing methods, which recognize strokes on a completed drawing, our method captures both spatial and temporal information of the strokes, and faithfully vectorizes them with timestamps. By this, we can avoid various stroke recognition ambiguities, enhance the vectorization quality, and recover the stroke drawing order. This is a challenging problem, requiring robust tracking of pencil tip, accurate modeling of pen-paper contact, handling pen-paper and hand-paper occlusion, while achieving interactive performance. To address these issues, we develop the following novel techniques. First, we perform robust spatio-temporal tracking of pencil tip by extracting discriminable features, which can be classified with a fast cascade of classifiers. Second, we model the pen-paper contact by analyzing the edge-profile of the acquired trajectory and extracting the portions related to individual strokes. Lastly, we propose a spatio-temporal method to reconstruct meaningful strokes, which are coherent to the stroke drawing continuity and drawing order. By integrating these techniques, our method can support interactive recognition and vectorization of drawn strokes that are faithful to the actual strokes drawn by the user, and facilitate the development of various multimedia applications such as video scribing, cartoon production, and pen input interface.

Categories and Subject Descriptors

I.4 [**Image Processing and Computer Vision**]: Applications

Keywords

line drawing; vectorization; online computation; pen interface; webcam

1. INTRODUCTION

The acquisition and vectorization of hand-drawn strokes on paper is beneficial to a wide range of multimedia applications: from handwriting recognition to cartoon production, video scribing and sketch-based modeling. This bridges the gap between paper and digital representations, and facilitates the development of computational methods for editing and interaction.

In general, there are four major approaches for recognizing and vectorizing paper-based line drawings according to the kind of inputs being processed:

- *Manual approach* involves manual effort of tracing individual strokes in a scanned input image. While being accurate and flexible, this approach can be extremely tedious subject to the complexity of the drawing.
- *Image-based approach* [2, 20] takes a scanned image of a completed drawing as an input, and attempts to automatically analyze the drawing contours to infer the stroke geometry and produce the vector representation. Major challenges of this approach are the image noise and the ambiguities in reconstructing the strokes due to image resolution and line junctions.
- *Sensor-based approach* [25, 30] employs hardware devices to track the pen trajectory on paper to detect the pen strokes. This approach, however, requires specially-designed devices that are not common, e.g., sensors in the ballpoint tip of a pen, etc.
- *Camera-based approach* [19, 23] captures the video of the drawing process, and recognizes strokes on paper by tracking and analyzing the pen-tip trajectory by computer-vision methods. The two main advantages of this approach are 1) the use of original writing instruments, i.e., pens/pencils, so involving no learning overhead in practice, and 2) the availability of temporal information from the captured video.

Our goal in this work is to develop an interactive camera-based method to capture and vectorize hand-drawn strokes on paper through a commodity low-cost webcam that captures at 480p/720p resolution and 30 frames/second. By this, we can faithfully vectorize drawn strokes with improved stroke continuity, and also recover the temporal information and drawing order of the strokes. With the emergence of recent ubiquitous devices like the Google Glass [9], camera-based approach could be particularly useful to help recognize hand-drawn inputs in front of the embedded camera, and to facilitate the development of pen input interface.

Previous methods for vectorizing hand-drawn strokes either exist as online/offline approaches, which are constrained by the stroke length, size, and direction, or strictly offline methods, which process the scanned images of the final drawings. To the best of our knowledge, this is the first attempt of developing an online camera-based method to acquire and

MM'14, November 3–7, 2014, Orlando, Florida, USA.

http://dx.doi.org/10.1145/2647868.2654939.

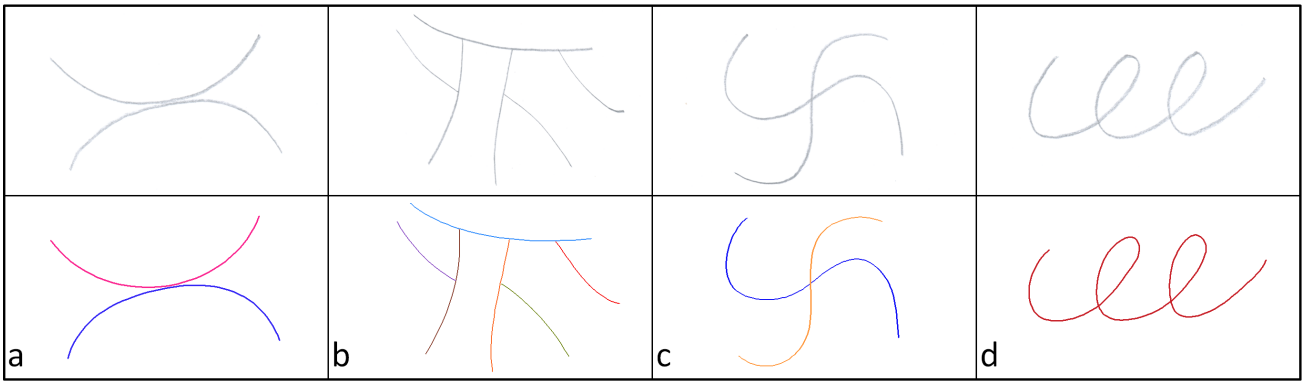

Figure 1: *Top row: four line drawings on paper, revealing four common stroke recognition ambiguities: (a) spatial adjacency, (b) junction ambiguities, (c) multiple-stroke intersection, and (d) self-intersection. Existing vectorization methods may misrecognize the original strokes as strokes of some other drawing continuity or break the original strokes into several shorter strokes at the junctions and intersections. Bottom row: our method can faithfully recognize and vectorize these strokes even in the presence of the ambiguities; color-coding is used to indicate individual recognized strokes.*

vectorize freehand line drawings.

However, such problem is very challenging since we need to robustly track the tip of the pen in the video, accurately model the pen-paper contacts, handle the pen-paper as well as hand-paper occlusion, and complete the entire computation with interactive performance. To address these issues, we propose the following novel techniques, which are the major technical contributions of this work:

- First, we identify three distinctive features, i.e., variance, color, and shape, and integrate them to robustly locate the tip of pen/pencil while quickly rejecting negative samples. We also exploit a domain-specific idea to generalize our shape feature to be invariant to rotation and partial perspective distortion.
- Second, we model the pen-paper contact implicitly by analyzing the zero-crossings of the Laplacian of Gaussian filter response along the pen-tip trajectory, and extracting an appropriate subset of it to estimate the potential region of the drawing strokes.
- Lastly, we combine the spatio-temporal information to define a cost function to reconstruct a stroke while preserving the stroke continuity and drawing order even in the presence of recognition ambiguities.

Our method has several advantages over existing offline vectorization methods, which work only with completed drawings. First, it is *online*, so it can capture and produce vectorized strokes with real-time feedback. Second, it involves only everyday pens/pencils and a simple camera rather than specialized pen/paper hardware. Third, our online method enables the capture of strokes with temporal information and drawing order, thus facilitating the development of various multimedia applications, and could potentially be integrated with camera-based interaction device such as the Google Glass. Lastly, by using temporal information, we can effectively avoid various stroke recognition ambiguities that are common in existing vectorization methods, see Figure 1: spatial adjacency, junction ambiguities, multiple-stroke intersection, and self-intersection, see also [20]. From the results shown in Figure 1, we can see our method can avoid these recognition ambiguities and faithfully recover each stroke natural to the actual stroke drawn by the user while preserving the stroke continuity in the vectorization.

2. RELATED WORK

Line Drawing Vectorization. Line drawing is a common form of raster images for vectorization. For example, manga artists traditionally create their cartoon drawings on paper with pencils, and then scan and vectorize their drawings for further digital processing and editing.

Early methods for line drawing vectorization can only deal with rather simple drawings such as technical layout plans and maps, where the methods could approximate the drawings with straight line segments [11] or employ morphological operations such as thinning [14] and skeletonization [31, 34] to extract the pixel-wide contours. Hilaire and Tombre [10] also attempted to fit line segments and arcs in a least-squares sense to images of technical drawings. As such, these methods are generally incapable of handling freehand drawings.

Several research works have been proposed to handle the vectorization of higher-order primitives. Among them, Chang et al. [5] studied piecewise fitting of Bezier curves for vectorizing cartoon images. Bao et al. [2] proposed to trace near-constant-width lines in line drawings by estimating a dense orientation field that guides the tracing direction. Noris et al. [20] reconstructed robust centerlines in topologically-complex line drawings in the presence of certain junction ambiguities. Some works that estimate temporal information and tracing directions of strokes [8, 13] and extract curves to trace contours with connectivity and orientation [26, 6] may also be used in vectorizing line-drawings.

While recent methods mentioned above and commercial software like Adobe Illustrator [28] and WinTopo [24] could produce reasonable vectorized lines given in line drawing images, they often cause fragmentation and/or stroke distortion problems especially near line junctions due to various stroke recognition issues, see Figure 1. Moreover, none of them could faithfully reconstruct strokes with spatio-temporal coherence following how the artist creates a drawing.

Pen Interface. Another relevant research area is the development of pen interfaces to capture and reconstruct pen strokes in an environment that mimics how people use pens, see [27] for a related survey. Generally, there are two major approaches for pen interface:

i) Sensor-based approach develops pen interfaces [25, 7] by using a combination of specially-designed hardware, e.g.,

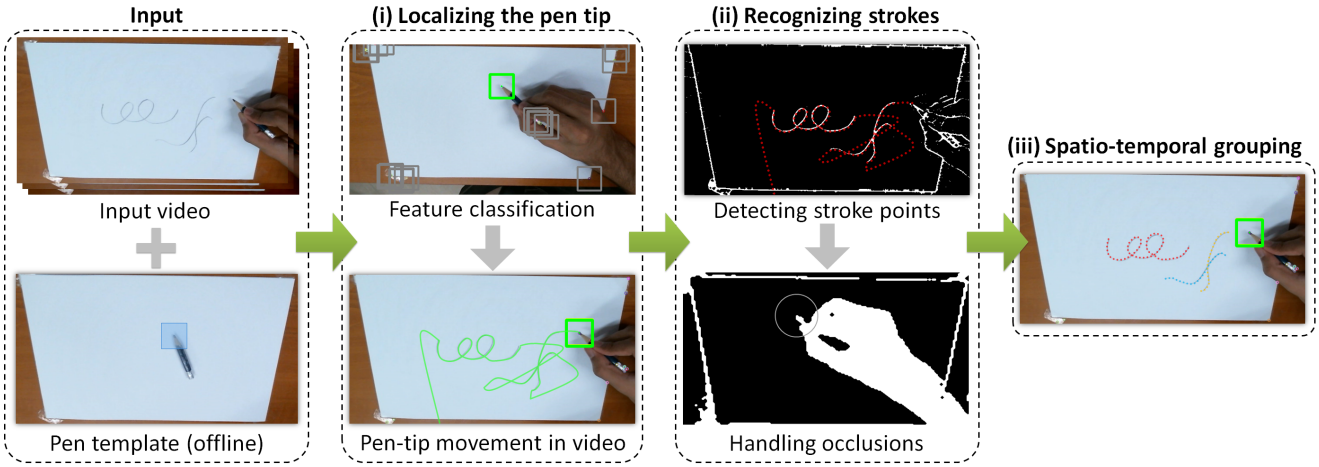

Figure 2: *Overview of our stroke recognition and vectorization method. From left to right: our input is a video stream and a simple pen-tip template; (i) potential patches are searched in video frames to localize the pen tip and track its trajectory; (ii) Laplacian of Gaussian filter response helps extract stroke points to estimate the actual drawn stroke; hand and pen occlusions are also tracked; and (iii) lastly, spatio-temporal clustering is performed to identify individual strokes (color coded).*

special paper with invisible dot pattern, camera-augmented pen, capacitive touch sensor, infrared camera, etc. Though there are some related commercial products, e.g., Anoto Digital pens [1] and the Wacom Inkling [30], this approach requires special hardware devices that are not as common as everyday pens and paper.

ii) Camera-based approach captures video frames of the pen writing/drawing process through a camera, and attempts to either localize or track the pen tip in each frame to obtain the pen-tip trajectory and stroke. Lin et al. [15] and Tang et al. [29] proposed various video-analysis schemes to recover the spatio-temporal grouping of strokes in Chinese characters. They first extract strokes by thinning/skeletonizing the final video frame of the completed drawing, and then infer the stroke continuity by analyzing the evolution of strokes over time. However, their method does not track the pen tip and attempts to infer stroke connectivity by analyzing the relevant pixels in the stroke skeleton throughout the video. Hence, the method is highly sensitive to shadow and noise, and requires tedious fine-tuning before the operation. Since the occlusion problem is not handled, the analysis is performed offline, and there is no interactive feedback.

Munich and Perona [18, 19] captured handwriting strokes by analyzing visual input from a conventional camera. The method was later extended to support handwriting recognition by Wienecke et al. [32]. In detail, they used a correlation-based template matching method to detect the pen tip. Hence, the method cannot handle planar rotations and perspective distortions. Pen-paper contact is estimated based on a simple local analysis of intensity, which can be easily affected by shadow or neighboring strokes. Since it assumes a small image region that confines the handwriting strokes, occlusions are handled with a fixed-size mask. More recently, Seok et al. [23] assumed a specific conical-shaped colored pen, and tracked the pen tip by simple color detection and geometric cues. While robust to orientation and illumination changes, this method restricts the color and shape of the pen tip. Though the method is online, it cannot handle the occlusion problem because it assumes that the handwriting strokes are unidirectional, and the hand and pen body would move away from the video view constantly. This assumption does not hold in our case for freehand drawing since user's hand may move around and occlude previous strokes. Moreover, [23] does not consider temporal information to disambiguate strokes (see again Figure 1) to support meaningful stroke-based editing, and assumes specific color and shape constraints to detect the pen tip.

In sharp contrast, our proposed method is more general and can overcome a number of problems and assumptions taken by existing state-of-the-art methods: it considers hand and pen occlusions over the paper; it can retain stroke continuity and drawing order; it is stable for local cluttering caused by neighboring strokes; it is an online method capable of delivering interactive feedback; and it has the potential for meaningful stroke-based editing.

3. OVERVIEW

Figure 2 overviews our method with a running example. Our system setup includes a drawing space with a sheet of paper fixed on a desk, an everyday pen/pencil for drawing, a commodity webcam that oversees the drawing space from the top, and a desktop computer that processes the webcam's video stream and performs our method to recognize and vectorize the drawing strokes. In summary, our method consists of the following three major steps:

i) **Localizing the pen tip.**
Extracting discriminable features around the pen tip for efficient, rotation-invariant, and illumination-invariant per-frame detection of pen tip (Section 4.2); *Pruning* the search space by corner detection to accelerate the performance (Section 4.3.1); and *Classifying* the extracted feature vectors using a cascaded approach to quickly identify and track the pen tip over time to obtain a meaningful pen-tip trajectory (Section 4.3.2).

ii) **Reconstructing Strokes.**
Extracting a subset of points along the pen-tip trajectory to estimate the footprint of the drawn strokes by analyzing the edge profile along the trajectory (Section 5.1); and *Resolving* occlusion problem caused by user's hand and pen over the paper (Section 5.2).

iii) **Spatio-temporal Grouping.**
Grouping the extracted points into a stroke by formulat-

ing a cost function that considers the stroke connectivity, and spatial and temporal distance information (Section 6); and *Vectorizing* the clustered stroke by fitting a cubic spline over the grouped points, and outputting the final result in SVG vector format.

In our implementation, the above computation takes only ~0.072 sec. per video frame, showing that our method can support online stroke recognition and vectorization at interactive speed. However, since some drawn strokes may be occluded fully or partially by user's hand or pen, we can only reconstruct their unoccluded parts during the online computation. Our method keeps tracking the hand and pen locations (see Section 5.2) to determine if the occlusion has been removed, and then reconstructs and renders the previously-occluded strokes once they become visible and can be detected by our method. Hence, in a single video frame, our method may recover more than one stroke at a time.

4. LOCALIZING THE PEN TIP

4.1 Initialization

Our input consists of a video stream of the pen drawing process and a pen-tip template. To prepare a new pen-tip template, which is just an offline one-time step, it only takes a few seconds to mark a 60×60 image region on a frame of the video stream to enclose the pen tip. Here we only assume the pen is placed over the paper, on which the drawing would be done, and place no restrictions on the pen orientation as our method can later automatically normalize the template input to a canonical form that is rotation-invariant.

In detail, when extracting the pen-tip template, we also need a foreground-background segmentation [33] on the video frame to obtain a background-subtracted binary image, assuming that the pen is not part of the initial scene. Here we set a high adaptation time period in the background model to avoid too-early merging of foreground into the background.

4.2 Feature Extraction

Given a patch, which could be a pen-tip template or a 60×60 region in a video frame, we next need to extract distinctive features in the patch for supporting the pen-tip localization. The set of features we carefully selected in this work are *background confidence*, *intensity variance*, *color*, and *shape*. These features are simple-to-compute and yet discriminative to help identify the pen tip under rotation, illumination variation, and mild perspective distortions.

i) *Background Confidence ($\mathcal{B}_i$).* By applying foreground-background segmentation [33] on every video frame (including the one for the pen-tip template and also the video frames of the drawing process), we can obtain a background-subtracted binary image M whose background pixels take zero values. Hence, given a patch in M, we can count the number of background pixels in it, and compute the patch's background confidence ($\mathcal{B}_i$), i.e., the ratio of the number of background pixels to the total number of pixels in the patch.

ii) *Intensity Variance (σ^2).* The second feature we employed is the statistical variance of pixel grayscale values. This feature helps to measure the uniformity of pixel intensities in a patch. Moreover, after we obtain the pen-tip template, we pre-compute its intensity variance, say σ^2, and define low and high intensity variance thresholds $0.5\sigma^2$ and $1.5\sigma^2$, respectively, to support the fast cascaded classifier later on.

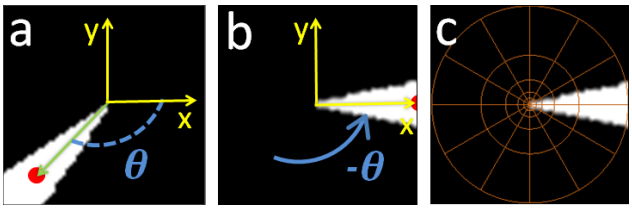

Figure 3: *Rotation-invariant shape feature. Given a binary patch extracted around a detected corner point in a video frame, we compute (a) the center of mass (red) of the largest foreground blob, rotate and normalize the patch into (b) a canonical shape representation, and then use (c) a log-polar histogram to extract the shape feature vector.*

iii) *Color ($\mathcal{C}_i$).* Third, we use the CIELab color-space to compute the color feature of a patch. Since the L channel represents the color luminance, we discard it for the sake of illumination invariance. Then, we quantize the chrominance by computing a 2D histogram of a and b channels with 16 bins in each dimension, and flattening the histogram as a 256-dimensional color feature vector. By this, we can compute the color feature vector of the pen tip template, say $v_t^{(c)}$, as well as the color feature vector of any patch in a video frame, say $v_i^{(c)}$. Hence, we can further compute the color similarity measure between them by Euclidean distance: $\mathcal{C}_i = \|v_t^{(c)} - v_i^{(c)}\|$.

iv) *Shape ($\mathcal{S}_i$).* Lastly, we compute the shape feature of a patch by devising an efficient rotation-invariant method based on the shape context [3]. Note that from the video frames, since we create candidate patches to be centered at detected corner points (Section 4.3.1), we can be certain that the pen tip (if any) would always appear at the center of a patch. Hence, for each patch, we can compute the shape feature as follows:

(a) Find the largest connected foreground blob in the given patch, and compute its center of mass, see the red dot in Figure 3(a);

(b) Compute its offset angle, say θ, from the positive x-axis in the image space, see again Figure 3(a);

(c) Rotate the patch image about its center (pen tip, if any) by $-\theta$ in order to normalize the patch to a canonical representation, see Figure 3(b);

(d) Calculate a 60-dimensional shape feature vector by centering a log-polar histogram (5 radial bins and 12 angular bins) at the patch center and computing its shape context, see Figure 3(c); and

(e) Lastly, compute shape similarity measure $\mathcal{S}_i$ for a given patch against the pen-tip template similar to the mechanism for the color similarity measure.

Note that the advantages of using the log-polar histogram here are: 1) it is simple and efficient to compute; 2) its coarse binning can sufficiently describe the pen-tip shape; and 3) its uneven binning relates and helps alleviate the effect of mild perspective distortion.

4.3 Feature Classification

4.3.1 Pruning the search space

In typical object localization, a sliding window is often used to go over a given video frame and extract all possible patches (potentially hundreds of thousands) for feature matching.

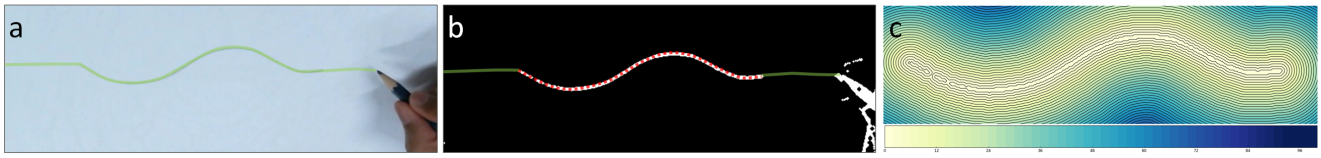

Figure 4: *Detecting drawn strokes: (a) captured pen-tip trajectory in green; (b) thresholded $\boldsymbol{L} \circ \boldsymbol{G}$ filter response reveals the drawn stroke; stroke points (red) are further extracted by using (c) level-sets defined around the detected stroke.*

Clearly, this approach is infeasible to deal with a large search space for interactive applications, so we prune the search space by two key observations in our problem:

- First, we assume fixed camera and paper. This is a reasonable assumption in most camera-based methods. Given that, we assume that the pen-tip relatively retains it's scale while drawing, so we only consider patches of the same size as the pen-tip template.
- Second, since the pen tip is always a *sharp corner*, we perform corner detection [22], which is a very fast process, to extract fixed-size patches around detected corners to accelerate the search for the pen tip.

By these, we can reduce the cardinality of the search space to the number of corners detected in each video frame and greatly accelerate the pen-tip localization.

4.3.2 Cascaded Classifier

Given the set of patches $\{s_i\}$ extracted from the pruning method above, we exploit the four features described in the previous subsection, and devise a fast cascaded approach to further classify patches in $\{s_i\}$. Note that this is a lazy approach in applying the features, so that we do not need to compute all the four features for every patch in $\{s_i\}$.

- In the first step, we compute the background confidence score $\mathcal{B}_i$ of each candidate patch in $\{s_i\}$, and retain only those with $\mathcal{B}_i$ less than a threshold T_b, where T_b is empirically set to be 0.95. This feature is highly discriminative, and it helps to quickly filter out most of the patches that belong to the background.
- In the second step, we compute the intensity variance of the remaining patches, and filter out those with intensity variance outside the prescribed threshold range $[0.5\sigma^2, 1.5\sigma^2]$. This step enables us to quickly reject almost all irrelevant patches that are not similar to the pen-tip template.
- In the third step, we compute the color similarity measure $\mathcal{C}_i$ of each remaining patch using the color feature vectors of the patch and the pen-tip template. By this, we can filter and retain only the patches whose $\mathcal{C}_i$ is less than a threshold T_c, which is set to be 0.15 in all our experiments.
- The last step is similar to the third step, but now we compute the shape feature vector of the remaining patches, and employ the shape similarity measure $\mathcal{S}_i$ to retain only the patches with $\mathcal{S}_i$ less than a threshold T_s, which is set to be 0.3 in all our experiments.

Since the fast cascaded filtering process above may still retain more than one patch in the end, we may need to further compute the average value of color and shape similarity measures for each remaining patch, and select the one with the largest average to locate the pen-tip patch.

4.4 Tracking

There may also be scenarios where the pen tip is not detected, i.e., no pen-tip patches are identified by the cascaded classifier above, e.g., missing the pen tip in the corner detection, and severe visual interference such as noise, shadow, and clutter. Hence, in addition to per-frame detection, we also employ the detected pen-tip location in the previous video frame to perform short-term tracking of the pen tip using the Lucas-Kanade optical flow method [16]. By this, we can improve the pen-tip localization, and avoid breaking the pen-tip trajectory due to the above mentioned reasons. In detail, we initialize the tracker by each detected pen-tip location from the cascade classifier, and continue the short-term tracking until the cascade classifier finds a subsequent detection, at which time the tracker is reinitialized. By combining detection and tracking, we can obtain a highly smooth pen-tip's trajectory, say $\mathbf{T}$, over the drawing space as a function of time (e.g., see Figure 4(a)):

$$\mathbf{T} = \left\{(p_1, t_1), (p_2, t_2), \ldots, (p_n, t_n) \mid p_i \in \mathbb{R}^2, t \in \mathbb{R}\right\},$$

where $p_i = (x_i, y_i)$ denotes a spatial coordinate, and t_i a monotonically-increasing timestamp (frame index).

5. RECONSTRUCTING STROKES

5.1 Detecting Stroke Points

Once we capture the pen-tip trajectory across the video frames, our next step is to determine the set of stroke points on the paper, say $\mathbf{S}$ (see Figure 4(a)). In detail, we determine $\mathbf{S}$ by analyzing the response of the Laplacian of Gaussian ($\mathbf{L} \circ \mathbf{G}$) filter over the video frames. The $\mathbf{L} \circ \mathbf{G}$ filter corresponds to how the human visual system observes sudden changes in contrast and downplays subtle differences [17]. Mathematically, it is represented by the curvature, or the second derivative of the image intensity, see Figure 4(c). The intensity profile of strokes in clean line drawings are generally in the form of a parabolic profile (see Figure 5(top)), and the maxima of the filter response corresponds to the cen-

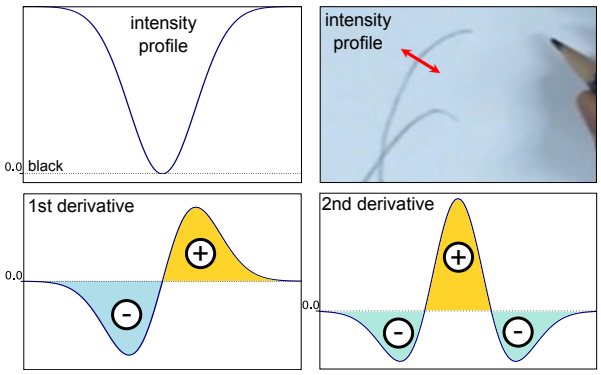

Figure 5: *Stroke intensity profile. Top: intensity profile across a stroke (perpendicular to stroke direction). Bottom: first and second derivatives of the intensity profile, respectively. The maxima of the second derivative corresponds to the centerline of the stroke.*

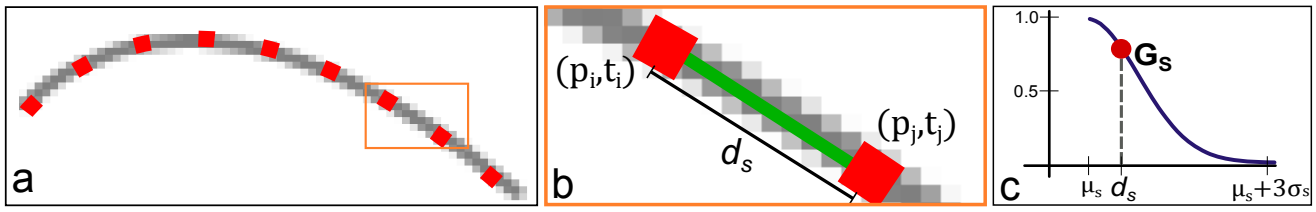

Figure 7: *Spatio-temporal grouping of stroke points: (a) a subset of stroke points in red; (b) edge connectivity estimated between two temporally-consecutive stroke points; and (c) the Gaussian function f for computing the spatio-temporal grouping criterion* $\mathbf{G_S}$ *(and similarly also for* $\mathbf{G_T}$*).*

terline of edges, or in our case, the centerline of the drawn strokes (see Figure 5(bottom)). From each frame, say F, we derive a new image $\mathcal{L} = \max(0, \mathbf{L} \circ \mathbf{G}(F))$ by applying the $\mathbf{L}\circ\mathbf{G}$ filter on F and clamping negative filter response to zero. After that, we further threshold $\mathcal{L}$ at a value of 15 to remove noise and weak responses, and obtain a set of edge pixels over the video frame, say $\{\mathbf{q}\}$. Next, we can define a level-sets around the edge pixels in $\mathcal{L}$ using a distance transform with the Euclidean distance metric: $D(\mathbf{x}) = \min_q \|\mathbf{x} - \mathbf{q}\|$, where $\mathbf{x}$ is each pixel in the video frame. By this, we can extract a set of stroke points $\mathbf{S}$ as a subset of the trajectory $\mathbf{T}$ lying close to the drawn strokes within a threshold:

$$\mathbf{S} = \{(p_i, t_i) \mid D(p_i) \leq \tau; \forall (p_i, t_i) \in \mathbf{T}\} \ ,$$

where τ defines the maximum level set below which points are on the drawn strokes; in practice, it is set to be 2.0.

5.2 Handling Occlusions

In the previous subsection, we presented an analysis method to obtain stroke points on the paper based on the $\mathbf{L}\circ\mathbf{G}$ filter response. However, it is important to note that not all edges in the filtered image $\mathcal{L}$ actually correspond to the drawn strokes. Typically, the user's hand and pen inside the video frames may occlude some of the drawn strokes, and affect the edge profile and the stroke point analysis. Hence, to obtain meaningful strokes points, we continuously track the positions of pen and hand in the video stream, and perform the stroke point analysis (Section 5.1) only on the non-occluded image region. In other words, we postpone the stroke point analysis until the occlusion is clear. In detail, we estimate the pen and hand occluding regions as follows:

- By the pen-tip localization method in Section 4, we can always locate the pen tip over the video frames. Hence, we can approximate the pen occluding region by a small circular region around the pen tip (radius equal to the size of the pen-tip template), see the gray circle in Figure 6(c).
- To estimate the hand occluding regions, we first convert the colors in each frame to the HSV color space. Then, we perform a pixel-level classification based on [21] to determine the skin color, and generate an initial mask of the hand. Note that the V channel is ignored in the computation to account for illumination invariance. By further applying morphological post-processing operations such as erosion and closing, we can obtain a clean mask of the user's hand as shown in Figure 6(c). Note also that we experimented our method with three users of different skin colors, and obtained the following threshold ranges in the classification model for hand tracking: $0 \leq \textit{hue} \leq 10^\circ$ and $0.08 \leq \textit{saturation} \leq 0.59$, see Figure 6(a).

Figure 6: *Occlusion detection: (a) thresholding skin color in the hue-saturation space; (b) a sample frame from the video; and (c) masks obtained for the pen (gray circle) and for the hand (white).*

6. SPATIO-TEMPORAL GROUPING

Given a set of strokes points $\mathbf{S}$ obtained from the previous stage, our next task is to group them and form meaningful drawing strokes that are spatio-temporally coherent. To do so, we first need to remove redundant points with exactly the same spatial coordinates. Note that redundant points could be produced if the pen tip stays on the same image location for more than one video frame.

After that, we define a cost function to measure whether two temporally-consecutive stroke points, say (p_i, t_i) and (p_j, t_j), should be grouped together in forming a drawing stroke. The cost function has the following three components:

i) **Edge Connectivity ($\mathbf{G_E}$).** This criterion explores the presence/absence of a stroke between (p_i, t_i) and (p_j, t_j). In detail, we sum the $\mathbf{L}\circ\mathbf{G}$ response along the line from (p_i, t_i) to (p_j, t_j) (e.g., the green line in Figure 7(b)), and normalize the sum by the length of the line; if the sum is above a threshold value of 0.4, we put $\mathbf{G_E}$ as 1, and 0 otherwise.

ii) **Spatial Cost ($\mathbf{G_S}$).** The second criterion explores the spatial proximity between (p_i, t_i) and (p_j, t_j) since we should not group stroke points that are too far away from each other. To compute this criterion, we measure the spatial distance, say d_s, between the two stroke points using the L^2 norm, i.e., $d_s = \|p_i - p_j\|$, and compute $\mathbf{G_S}$ by mapping the distance value to range $[0, 1]$ (see Figure 7(b&c)): $\mathbf{G_S} = f(\ d_s,\ \mu_s,\ \sigma_s\ )$, where μ_s and σ_s are set to be 1.0 and 3.0, respectively, and f is a Gaussian function:

$$f(x, \mu, \sigma) = \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) .$$

iii) **Temporal Cost ($\mathbf{G_T}$).** The third criterion is similar to the second one, but it explores the temporal proximity between (p_i, t_i) and (p_j, t_j). Here it computes the temporal distance, say d_t, between the two points using the L^1 norm, and again use f to map the distance value to range $[0, 1]$: $\mathbf{G_T} = f(\ d_t,\ \mu_t,\ \sigma_t\ )$, where μ_t and σ_t are set to be 1.0 and 5.0, respectively.

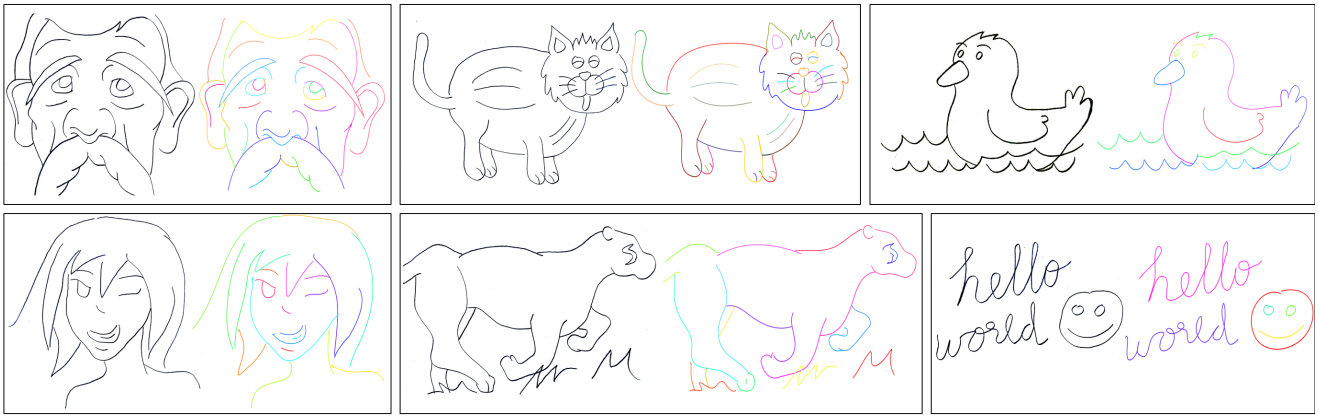

Figure 8: *Vectorization results of six different line drawings generated by our method. In each box, we show the vectorization result with randomly-colored strokes on the right and a contrast-enhanced scanned image of the original drawing on the left. From left to right: the drawings are* OLD MAN, CAT, DUCK, GIRL, CHEETAH, *and* HELLO-WORLD*; our method recognizes 42, 44, 17, 24, 25, and 6 strokes from them, respectively.*

The combined cost $\mathbf{G}$ of grouping points (p_i, t_i) and (p_j, t_j) to a continuous stroke is then defined as:

$$\mathbf{G}(i,j) = \mathbf{G_E}\left[\ (1-w)\cdot \mathbf{G_S}(i,j) + w \cdot \mathbf{G_T}(i,j)\ \right],$$

where w is set to be 0.6 for putting more emphasis on the temporal information. If $\mathbf{G}$ is greater than 0.5, we consider the two points to go along a common continuous stroke. Once we examine and group each pair of temporally-consecutive points, we can form longer strokes, and then vectorize each of them by cubic spline fitting.

7. RESULTS AND EVALUATION

7.1 Implementation

We implement and run our method on a desktop computer with a 3.20 GHz Intel i7 CPU, 12 GB memory, and 64-bit Windows 7. We employ the Logitech C615 webcam to capture video streams at 480p/720p and 30 frames/sec. See Figure 9: the webcam is fixed at ∼1.5ft above the drawing space opposite to the user at an evaluation angle of ∼60 degree and a field-of-view of 74 degree.

Estimating the homography transformation between the webcam view and paper is a one-time process: During the system initialization, we first identify the largest white blob in the image, and then use Hough line transform to extract straight lines around the blob's boundary edges. After that, we compute line intersections to obtain the paper corners, show them to the user for confirmation, and then estimate the homography matrix by assuming the paper dimension. We implement our software in Python 2.7, and use the following library APIs: OpenCV [4], NumPy, and SciPy [12] for vision-based methods and numerical computation.

7.2 Evaluation: Performance

First, we perform an experiment to evaluate the performance of our method in delivering interactive online computation. To do so, we measure the time taken (in sec.) to process each video frame (resolution 720p) for each stage in the pipeline (see again Figure 2). Two drawing cases, CHEETAH and HELLO-WORLD (see Figure 8), with totally 7680 and 3150 frames, respectively, are used here. The average processing time taken by each stage is reported in Table 1.

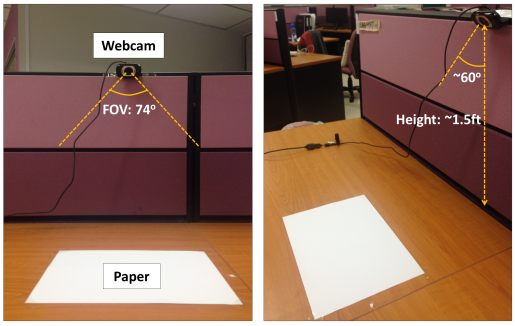

Figure 9: *System setup from two different views.*

From the performance results, we see that the entire computation consistently takes around 0.07-0.08 sec. per frame, showing that it can attain interactive performance.

stage	method	avg. time (sec)	
		CHEETAH	HELLO-WORLD
1	pen-tip localization	0.035	0.035
2a	detecting stroke points	0.012	0.012
2b	handling occlusions	0.009	0.008
3	spatio-temporal grouping	0.012	0.021

Table 1: *Performance evaluation result: average time taken to complete the various stages in our method.*

7.3 Evaluation: Accuracy

Second, we evaluate the accuracy of our results by comparing them with the actual strokes drawn on paper, as well as with the vectorization results produced from the commercial tool WinTopo Professional [24] (with the default option *One-Touch Vectorization* in our experiments). The six line drawings shown in Figure 8 are used in our evaluation and the following four criteria are used:

i) Number of Strokes. First, we compare the total number of actual strokes drawn on paper against the total number of strokes vectorized by our method, and also by WinTopo, see Table 2. From the results, we can see that our online method is able to accurately recognize all the user-drawn strokes without missing any stroke. In sharp contrast, WinTopo produces a lot more strokes than the actual num-

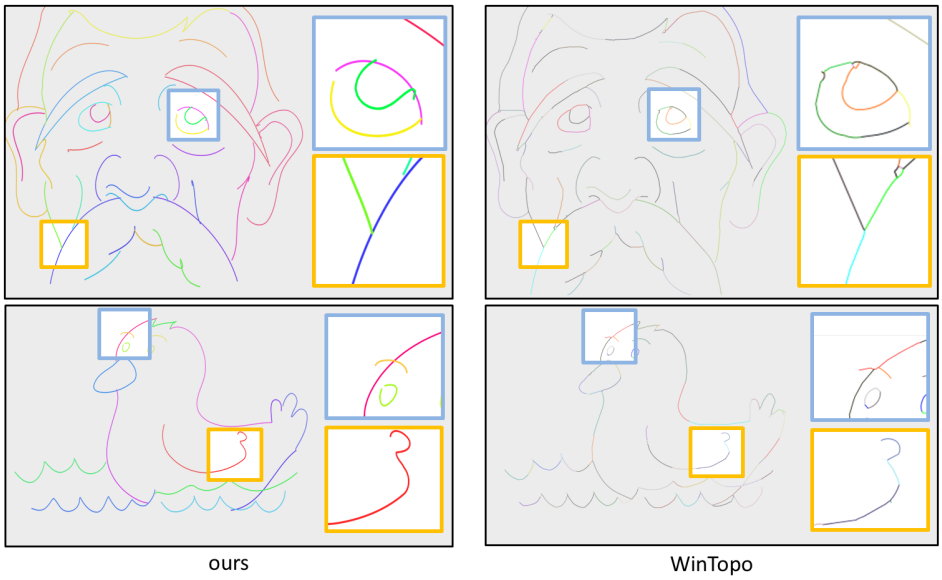

Figure 10: *Visual comparison of stroke reconstruction results: our method (left) and WinTopo (right).*

drawing	# actual strokes	# reconstructed		overlap %	avg. deviation
		ours	WinTopo		
CAT	44	44	74	88.77	1.568
GIRL	24	24	43	88.50	1.520
HELLO-WORLD	6	6	30	94.89	1.215
OLD-MAN	42	42	50	86.29	2.957
DUCK	17	17	35	87.34	2.135
CHEETAH	25	25	42	83.66	1.994

Table 2: *Evaluation of accuracy: the number of reconstructed strokes, overlap percentage, and average deviation.*

ber of strokes drawn by user due to its deficiency in resolving various stroke recognition ambiguities, see again Figure 1.

ii) Overlap Percentage. Second, we evaluate how close our vectorization results are to the actual drawing. Here we first scan the paper drawing and use a simple binary segmentation to extract the actual strokes. Then, we transform and overlay our vectorized strokes onto the scanned image (through the recovered homography) and compute the overlap percentage: $|A \cap B|/|B|$, where A and B are sets of pixels covered by the scanned strokes and the vectorized strokes, respectively. Note that this measure aims to see how the vectorized strokes (actually one-pixel-wide lines) are contained within the actual strokes in the scanned image space.
From Table 2, we can see that our method can attain very high overlap percentage, but not yet perfect, because: 1) the homography recovered for computing the overlap percentage is not perfect, and 2) we use a discrete set of stroke points (captured in each frame) to reconstruct a stroke by spline-fitting. Since WinTopo vectorizes strokes by binary-thinning the scanned images, its vectorized results can always perfectly overlap with the original strokes. However, from Figure 8, we can still see how similar our vectorization results are with respect to the scanned drawings.

iii) Average Deviation. Third, we quantitatively measure the deviation of our reconstructed strokes from the scanned drawings in pixel units. Like overlap measure, this is done in the image space of the scanned drawings.
Let P and Q be the set of pixels belonging to the scanned strokes and our reconstructed strokes (1-pixel wide), respectively. We compute the average deviation between them as $dev(P,Q) = \sum_{q_i \in Q} \min \|q_i - p_j\| / |Q|$, where p_j is the pixel in P that is the nearest to q_i in the image space. From the results shown on the rightmost column of Table 2, we can see that the average deviation is consistently around 1 to 2 pixels, indicating that our results are sufficiently close to the original strokes. Note that the resolution of the scanned images used in this experiment is 877×637 pixels.

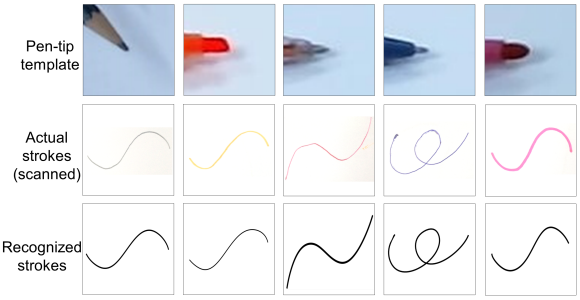

Figure 11: *Our method supports a variety of pens and pencils. Top row: different pen-tip templates; middle row: actual strokes drawn on paper; and bottom row: corresponding strokes recognized by our method with the pen/pencil.*

iv) Visual Comparison. Lastly, we randomly colorize the recognized strokes for both our results and WinTopo, and perform a visual comparison. Two comparison examples are shown in Figure 10. From the zoomed views shown in the figure, we can clearly see that WinTopo suffers from stroke recognition ambiguities, and tends to break strokes at junctions, thereby producing excessive strokes. On the contrary, our method, which employs temporal information, can accurately reconstruct the long strokes while retaining the original stroke continuity and avoiding various stroke recognition ambiguities.

7.4 Evaluation: Variety of Pens and Pencils

Our method supports a variety of pens and pencils, see Figure 11 (first row). To include a new pen/pencil to stroke recognition, we only need a few seconds' time to take a picture of the pen tip in front of the camera, and then apply our

Figure 13: *Video scribing. Our method enables us to easily re-render the vectorized strokes, which have temporal information, and produce a video scribing effect. The drawing video is spatio-temporal coherent with the original drawing by the user but goes without the pen and the user's hand.*

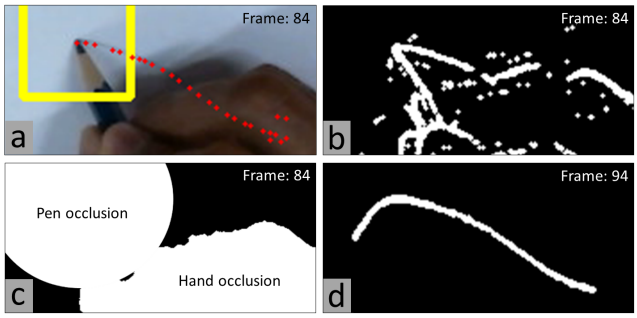

Figure 12: *Example of how occlusion could interfere the analysis of stroke points: (a) notice a number of stroke are points occluded by the hand and the pen-tip at the moment; (b) $\boldsymbol{L} \circ \boldsymbol{G}$ response affected by the occlusion; attempting to recognize stroke points in the presence of occlusion is error-prone; (c) occlusion mask computed by our method; and (d) valid $\boldsymbol{L} \circ \boldsymbol{G}$ response of the stroke after the occlusion is clear.*

interface to crop the picture and create a pen-tip template. After that, our method can employ the pen-tip template to automatically locate the pen-tip trajectory.

Figure 11 presents the related results with five different kinds of pens and pencils. The top, middle and bottom rows in the figure show the pen-tip templates, scanned image of a drawn stroke, the recognition results respectively.

7.5 Evaluation: Occlusion Detection

Next, we explore and illustrate how occlusion detection helps to improve the stroke recognition. A typical case is presented in Figure 12. During the drawing process, hand and pen occlusions could seriously interfere the stroke point analysis process described in Section 5.1. See the drawing in Figure 12(a) and the resulting $\mathbf{L} \circ \mathbf{G}$ response in Figure 12(b). If we analyze this response and extract the stroke points around the pen-tip trajectory at this moment (video frame #84), the analysis result would be error-prone.

Hence, we detect and track the hand and pen occlusion regions over the video frames, see Figure 12(c), and postpone the stroke point analysis and extraction till the occlusion is clear, see Figure 12(d). By this mechanism, we can improve the extraction quality while attaining full automation.

7.6 Evaluation: Vectorization Results

Figure 8 presents the vectorization results generated by our method and compares them side by side with scanned images of the original drawings. Notice the individual reconstructed strokes (each randomly-colored). They are free of the various stroke recognition ambiguities we mentioned in Section 1 (see also Figure 1); in addition, they can correspond nicely to the actual strokes drawn by the user. This demonstrates one key advantage of our online method in improving the stroke recognition quality.

Furthermore, we develop a video-scribing application, which renders the vectorized strokes on a virtual paper according to their spatio-temporal orders but without the pen and user's hand, see Figure 13 for image snapshots. By this, we can simulate and visualize the actual drawing process, and create visually-pleasing animation effects.

8. CONCLUSION

This paper presents a novel interactive camera-based method to capture and vectorize freehand line drawings on paper. There are three key contributions in this work. First, we propose a robust spatio-temporal tracking technique that can efficiently localize and track the pen tip in video frames. This technique combines the strength of a fast-cascade classifier with discriminable features and optical-flow tracking (which is less accurate) to achieve high performance and accuracy. Second, we extract stroke points in the video frames by considering not only the edge profiles around the estimated pen-tip trajectory but also the hand and pen occlusion to improve the stroke recognition accuracy. Lastly, we develop an efficient spatio-temporal clustering method to produce meaningful strokes that are coherent with the actual strokes drawn by the users. As for the evaluation, we perform a number of experiments to examine different aspects of our method: computation performance, accuracy against actual drawings and an offline commercial tool for vectorization, as well as pen and hand occlusion. In the end, we use the method to produce assorted vectorization results, as well as apply it to produce the video-scribing animation effect with some of our results.

Future Work. First, we would look for ways to enable a moving camera, so that we could extend our interface for smart-glasses. Second, we would also like to improve the

system performance, e.g., by multi-core processing on commodity CPUs. Third, we would explore the use of a parameter calibration module for better adaptation to varying lighting conditions. Lastly, we also plan to explore the use of crowdsourcing to study and evaluate our interface.

Acknowledgments. We thank reviewers for various constructive comments that helped to improve this paper. This work is funded in part by MOE Tier-2 grant (MOE2011-T2-2-041 (ARC 5/12)) and MOE Tier-1 grant (RG 29/11).

9. REFERENCES

[1] Anoto. Anoto Digital Pens. `http://www.anoto.com`.

[2] B. Bao and H. Fu. Vectorizing line drawings with near-constant line width. In *IEEE International Conference on Image Processing*, pages 805–808, 2012.

[3] S. Belongie and J. Malik. Matching with shape contexts. In *IEEE Workshop on Content-based Access of Image and Video Libraries*, pages 20–26, 2000.

[4] G. Bradski. OpenCV. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] H.-H. Chang and H. Yan. Vectorization of hand-drawn image using piecewise cubic bézier curves fitting. *Pattern Recognition*, 31(11):1747 – 1755, 1998.

[6] M.-M. Cheng. Curve structure extraction for cartoon images. In *Harmonious Human Machine Env.*, pages 13–25, 2009.

[7] M. Chikano, K. Kise, M. Iwamura, S. Uchida, and S. Omachi. Recovery and localization of handwritings by a camera-pen based on tracking and document image retrieval. *Pattern Recognition Letters*, 35:214–224, 2014.

[8] D. Doermann and A. Rosenfeld. Recovery of temporal information from static images of handwriting. *International Journal of Computer Vision*, 15(1-2):143–164, 1995.

[9] Google. Google Glass. `http://www.google.com/glass/start/`, 2013.

[10] X. Hilaire and K. Tombre. Robust and Accurate Vectorization of Line Drawings. *IEEE Trans. on Pattern Analysis Machine Intell.*, 28(6):890–904, 2006.

[11] R. D. Janssen and A. M. Vossepoel. Adaptive vectorization of line drawing images. *Computer Vision and Image Understanding*, 65(1):38 – 56, 1997.

[12] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001.

[13] Y. Kato and M. Yasuhara. Recovery of drawing order from scanned images of multi-stroke handwriting. In *Proceedings of International Conference on Document Analysis and Recognition.*, pages 261–264, 1999.

[14] L. Lam, S.-W. Lee, and C. Suen. Thinning methodologies - A comprehensive survey. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 14(9):869–885, 1992.

[15] F. Lin and X. Tang. Dynamic stroke information analysis for video-based handwritten chinese character recognition. In *IEEE International Conference on Computer Vision*, volume 1, pages 695–700, 2003.

[16] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proc. of International Joint Conference on Artificial Intelligence*, volume 2, pages 674–679, 1981.

[17] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London Series B*, 207(1167):187–217, 1980.

[18] M. Munich and P. Perona. Visual input for pen-based computers. In *Proceedings of the 13th International Conference on Pattern Recognition, 1996.*, volume 3, pages 33–37, 1996.

[19] M. Munich and P. Perona. Visual Input for Pen-based Computers. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 24(3):313–328, 2002.

[20] G. Noris, A. Hornung, R. W. Sumner, M. Simmons, and M. Gross. Topology-driven Vectorization of Clean Line Drawings. *ACM Transactions on Graphics*, 32(1):4:1–4:11, 2013.

[21] S. Phung, A. Bouzerdoum, and S. Chai, D. Skin segmentation using color pixel classification: analysis and comparison. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 27(1):148–154, 2005.

[22] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, 2006.

[23] J.-H. Seok, S. Levasseur, K.-E. Kim, and J. H. Kim. Tracing handwriting on paper document under video camera. In *Proceedings of the International Conference on Frontiers in Handwriting Recognition*, pages 109–110, 2008.

[24] SoftSoft. WinTopo. `http://wintopo.com/`, 2012.

[25] M. Sperber, M. Klinkigt, K. Kise, M. Iwamura, B. Adrian, and A. Dengel. Handwriting reconstruction for a camera pen using random dot patterns. In *International Conference on Frontiers in Handwriting Recognition*, pages 160–165, 2010.

[26] C. Steger. Extracting curvilinear structures: A differential geometric approach. In *Computer Vision - ECCV '96*, volume 1064 of *Lecture Notes in Computer Science*, pages 630–641. 1996.

[27] J. Steimle. Survey of pen-and-paper computing. In *Pen-and-Paper User Interfaces*, Human-Computer Interaction Series, pages 19–65. 2012.

[28] A. Systems. Adobe Illustrator. `http://www.adobe.com/products/illustrator.html`.

[29] X. Tang, F. Lin, and J. Liu. Video-based handwritten Chinese character recognition. *IEEE Trans. on Circuits and Sys. for Video Tech.*, 15(1):167–174, 2005.

[30] Wacom. Inkling. `http://inkling.wacom.com/`.

[31] L. Wenyin and D. Dori. A survey of non-thinning based vectorization methods. In *Advances in Pattern Recognition*, volume 1451 of *Lecture Notes in Computer Science*, pages 230–241. 1998.

[32] M. Wienecke, G. A. Fink, and G. Sagerer. A handwriting recognition system based on visual input. 2nd International Workshop on Computer Vision Systems, pages 63–72, 2001.

[33] Z. Zivkovic and F. van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recognition Letters*, 27(7):773 – 780, 2006.

[34] J. J. Zou and H. Yan. Cartoon Image Vectorization Based on Shape Subdivision. In *Proceedings of the International Conference on Computer Graphics*, CGI '01, pages 225–231, 2001.