Scale Space Based Grammar for Hand Detection

Jan Prokaj[1] and Niels da Vitoria Lobo[2]

[1] University of Central Florida, Orlando, FL 32816, USA `j.prokaj@computer.org`
[2] University of Central Florida, Orlando, FL 32816, USA `niels@cs.ucf.edu`

Abstract. For detecting difficult objects, like hands, we present an algorithm that uses tokens and a grammar. Tokens are found by employing a new scale space edge detector that finds scale invariant features at object boundaries. We begin by constructing the scale space. Then we find edges at each scale and flatten the scale space to one edge image. To detect a hand we define a hand pattern grammar using curve tokens for finger tips and wedges, and line tokens. We identify a hand pattern by parsing these tokens using a graph based algorithm. We show and discuss the results of this algorithm on a database of hand images.

1 Introduction

Object detection is one of the fundamental problems of computer vision. The most successful technique for detecting objects is to find invariant features, and then match them to a previously defined set. Finding good features, however, is difficult. Lowe [1] has shown that scale space invariance is especially important. SIFT features produce excellent results when detecting rigid objects that are affinely trackable. But these features are less useful for objects that have important information encoded within the shapes of their boundaries. Hands, and other non-rigid objects are a good example of this. Thus, there is a need for different features, which are still scale space invariant, but do not suffer from the lack of affine trackability at boundaries.

Much work has been done on hand tracking [2–4], and hand pose estimation [5, 6]. But these do not work for the difficult task of hand detection in a single cluttered image.

We cast the problem of finding these features as doing a "lexical analysis" on the input image, producing tokens. In the next section we present a method for finding these scale invariant tokens at boundaries. Then in the following section, we define a grammar using this token alphabet that produces all strings identifying a hand. Subsequently, we present an algorithm that implements the grammar and "parses" these tokens, giving a scale space hand detection algorithm.

2 Finding Tokens

To find tokens at boundaries, it is natural to work with an edge image. A single edge image, however, is computed at only one scale. A common technique is to smooth the image with a particular σ, calculate its derivative, and use gradients to find local maxima [7]. This can be a problem because if the scale is too large, needed detail may be lost; if the scale is too small, the edges are likely to be disconnected. Motivated by this problem, we offer a scale space edge detector, which avoids these problems. The detected edges then serve as a good source of scale invariant tokens.

For open hand detection we use two kinds of tokens, curves and lines. Curve tokens are used to find finger tip curves, and wedge curves for the region where fingers join. Line tokens are used to find the border of extended fingers. The algorithms are presented in Sect. 2.2 and 2.3.

2.1 Scale Space Edge Detection

A scale invariant, high-fidelity edge image must have at least two properties. It has to have much detail in order to have the best chance of finding a particular shape, as well as have edges that are smooth and connected. These two properties occur at opposite ends of the scale space [8].

By constructing the scale space of an image, we gain access to the result of all possible smoothing operations. This information can be used to calculate one edge image that has a *lot of detail* using data from the bottom of the scale space, and at the same time has *continuous* edges, using data from the top of the scale space.

There are two steps to calculate a scale space edge image. In the first step, a Canny scale space is contructed. In the second step, this scale space is "flattened" to a final edge image.

Step 1: Canny Scale Space. The Canny scale space is derived from the Gaussian scale space. The Gaussian scale space is constructed as a pyramid according to [1], but with a maximum of three octaves. From this scale space, the Canny scale space is derived by applying the Canny edge detection algorithm on every image in the Gaussian scale space. However, the algorithm is modified to compute the gradient without additional blurring in the convolution.

Step 2: Flattening the Space. To convert the Canny scale space into one edge image, it must be flattened without losing too much information. In the first step, the edge images within each octave are combined into one octave edge image; in step two these octave images are then combined into a final edge image.

Combining the images within an octave is straightforward. It begins by unioning the images in each octave. As a result of this operation, an "on" pixel in the union image says that it was "on" in at least one scale in this octave. This union image is then smoothed by setting on any "off" pixels that are surrounded by at least six "on" pixels. To produce thin edges, the smoothed image is skeletonized. See Fig. 1.

Combining the octave edge images is more involved, because the images are different dimensions. The combinations proceed down the scale. Therefore, the

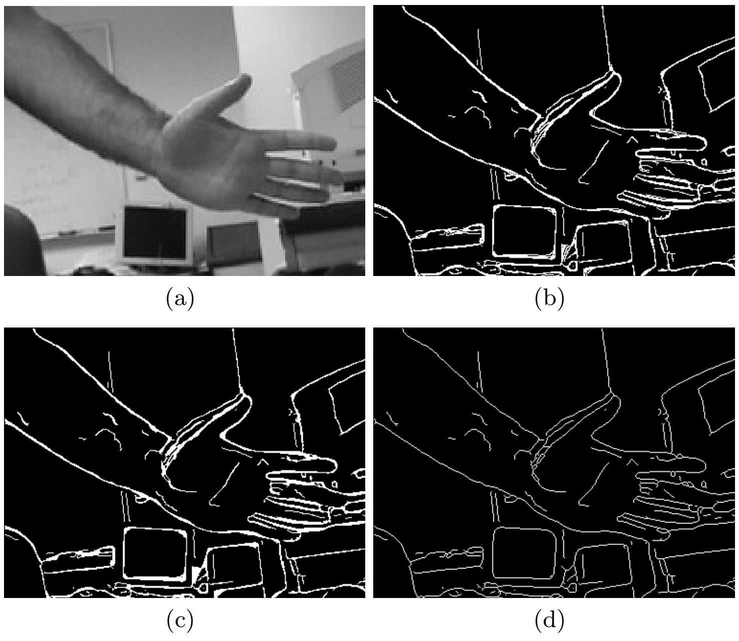
Fig. 1. Combining the images within an octave. (a) The original image. (b) Unioned octave. (c) Smoothed octave. (d) Skeletonized octave.

top two octaves are combined first, followed by a combination with the lowest octave image. The combination algorithm overlaps the lower scale image (larger dimension) with the higher scale image (smaller dimension), thus filling in any gaps in the lower scale image that are connected in the higher scale image. The overlap is done segment by segment, rather than all at once. For each connected segment in the higher scale image, the corresponding points in the lower scale image are traversed until reaching an end point. At this point, the traversal continues on the higher scale segment, marking the edge on the lower scale image, until it is possible to resume again on the lower scale image. See Fig. 2.

After all octave edge images are combined in this manner, the result will be one scale invariant edge image, that is suitable for finding the needed tokens.

2.2 Curve Tokens

A finger tip and a wedge between fingers can both be characterized as curves. They only differ in proportion, one is wider than the other. To find these curves in the edge image, a model-based approach is used. The algorithm is based on [9]. The models used in experiments are shown in Fig. 3.

Curves are found by superimposing a model's edge image to different locations in the scale space edge image, and calculating a similarity score. These "candidate" locations of curves are taken to be the end points and mid points of

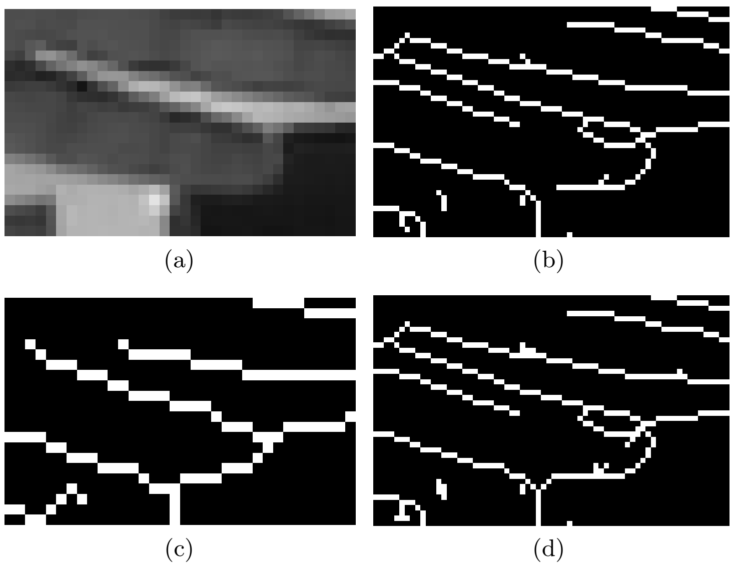

Fig. 2. Combining octave images (detail). (a) The original image. (b) Lower scale image. (c) Higher scale image. (d) Final edge image.

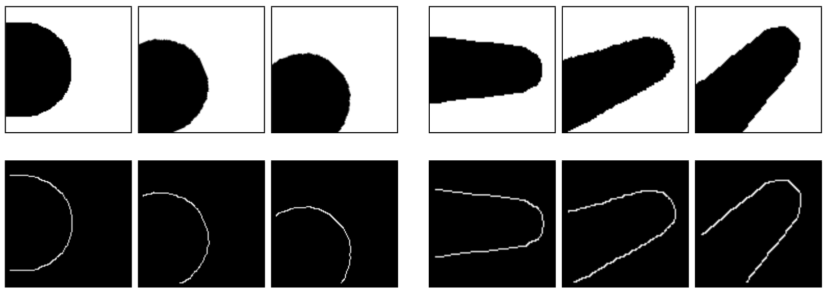

Fig. 3. This figure shows a subset of the finger tip (left) and wedge (right) models, and their edges (bottom), used in the scoring method.

lines found by a line finder, which is described in Sect. 2.3. To eliminate locations that are not likely to contain curves, mid points of lines are not taken for the the top 25% of longest lines.

To handle multiple orientations of curves, the base model is rotated to give 16 rotated versions of the model. Some of these orientations are shown in Fig. 3. Different curve sizes are handled by precomputing different sized models for each rotated version, keeping the same proportion.

The score for a given candidate location is the maximum of the scores of all sizes of the model with the same orientation as the location. The orientation for any point in the edge image is determined from the gradient of the image at the smallest scale in the Gaussian scale space, because this image has the most accurate orientation data. For each size, actually two opposite orientations are tried, because the gradient can be negative or positive and have the same orientation.

To superimpose a model correctly, the center point of a model's edge curve, which is the top of a curve, is aligned with the candidate location. The score of a model, having a particular size and orientation, is calculated using

$$\frac{\sum_k e^{-0.25 \min(D_k)} \left|\boldsymbol{m}_k \cdot \boldsymbol{v}_k\right|}{k} \quad , \tag{1}$$

where k is each model edge point, $\min(D_k)$ is the minimum of the five distances to the closest image edge point that would be obtained by shifting the model 0 and ± 1 units in the parallel and perpendicular directions to the model's orientation, $\boldsymbol{m}_k$ is the unit normal vector at the k-th model edge point, and $\boldsymbol{v}_k$ is the unit normal vector at the closest image edge point, given by $\min(D_k)$. In other words, for each model edge point the score is determined by the distance to the closest image edge point and the orientation difference between the two points. The model is shifted one unit in four directions, because this increases its flexibility. The score for a candidate location is compared to a threshold, set to 0.78 in our experiments, and the location is eliminated if the score is less than this.

To eliminate curves that are not likely to be finger tips, we make sure that each curve found is supported by a line token, as found by a modified line finder described in the next section. The line must be parallel, and close to either one of the ends of the curve. This is accomplished by searching for lines in a rectangle 7 pixels wide and $2 * \text{curvesize}$ pixels tall and rotated to be parallel with the curve's orientation. The rectangle is searched twice, the first time when its side is centered on one curve end, and the second time when its side is centered on the other end. The lines must have an orientation within 0.55 rad of the curve's orientation, and their length must be at least $1.1 * \text{curvesize}$.

Since the number of candidate locations is large, especially on curved edges, it is possible that this process produces overlapping curves. These duplicate curves are eliminated by calculating the intersection of the bounding rectangles of two curves. If the intersection is greater than 15% of the area of the smaller curve, the curve with the lower score is eliminated.

This curve detection algorithm is run with finger tip models, as well as with finger wedge models.

2.3 Line Tokens

To find lines on a scale space edge image, we use a modifed Burns line finder [10]. The lines are directly extracted from the edge image, instead of using the raw image as in the original algorithm. As mentioned above, the orientation data for the scale space edge image comes from the gradient of an image with the smallest scale. The Burns algorithm uses two systems of lines to avoid boundary effects of orientation partitioning. These two systems of lines are resolved differently to determine the final set of lines.

The lines from either system are processed in order from the longest to the shortest. For each line, both systems are analyzed for possible merges with other lines. First, a line merges with the corresponding (on the same pixels) lines in the other system. If this causes the line to partially extend over neighboring lines in its own system, then it merges with them as well. The lines in the other system on the same pixels as this new extension are adjusted (shortened), but not merged with. The line lengths are updated as the merging goes on, so that the longest line can be found in the next iteration. This method of resolving the two systems favors long, unbroken lines, which is what is needed to find finger lines.

3 Parsing Tokens

To parse curve and line tokens into strings that identify a hand, a grammar is defined to enforce the detection of the following properties: 1) A finger tip curve has an opposite orientation of a wedge curve. 2) The endpoints of the curves must be closer than the centers of the curves. 3) There is a line token connecting a finger tip curve with a wedge curve. These properties are formally written below, where t is a finger tip curve, w is a wedge curve, and l is a line token.

$$\begin{aligned}
&H \rightarrow tlwltlwltlwltlwlt\\
&H \rightarrow tlwltlwltlwltlw \mid wltlwltlwltlwlt\\
&H \rightarrow tlwltlwltlwlt\\
&H \rightarrow tlwltlwltlw \mid wltlwltlwlt\\
&H \rightarrow tlwltlwlt\\
&H \rightarrow tlwltlw \mid wltlwlt
\end{aligned}$$

This grammar eliminates strings with three or less curves. The -t-w-t- sequence produces a “zigzag” pattern across the hand’s fingers. See Fig. 4 for an illustration. Any token can be missing, but as soon as this happens, it is no longer allowed.

3.1 Open Hand Detection (pattern parser)

To find the desired pattern, we first construct a complete graph with nodes being all curve tokens, finger tips and wedges. The edges in the graph, termed g-edges,

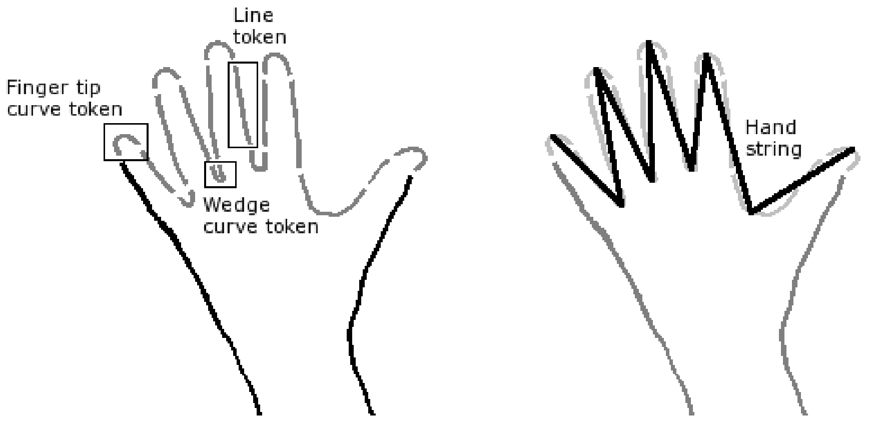

Fig. 4. This figure illustrates the zigzag pattern we are looking for.

correspond to all possible curve combinations. The weight of each g-edge is a pattern distance [11] between two curves. The pattern distance is defined as a Euclidian pattern distance,

$$\sqrt{\left(\frac{x_1 - x_2}{xRange}\right)^2 + \left(\frac{y_1 - y_2}{yRange}\right)^2 + \left(\frac{\theta_1 - (\theta_2 + \pi)}{\pi}\right)^2}\,, \qquad (2)$$

where $xRange$ is the maximum distance in the x-dimension and $yRange$ is the maximum distance in the y-dimension. The pattern distance between two curves is the shortest when they are close to each other and have opposite orientations from each other. This ensures a -tip-wedge-tip- sequence.

The next step is to remove those g-edges that cannot possibly make a valid combination. The criteria are determined from the properties above. Using the first property that adjacent curves have to have opposite orientations, g-edges with orientation difference less than $\frac{\pi}{2}$ are removed. The second property says that a g-edge must connect curves back to back, where the back is the two endpoints, not the center of the curve. To enforce the third property that there is a line token connecting the two curves, a rectangular region centered on the g-edge and 7 pixels wide is searched for a line parallel to the g-edge. The orientation difference between the g-edge and line must be less than 0.18 rad, and the length must be between 0.675 and 1.6 times the g-edge length. If no line satisfying this criteria is found, the g-edge is removed.

Once impossible g-edges are removed, the next step is to find a zigzag pattern in this graph. This is done using a back-tracking algorithm. Starting at one node, all possible g-edges connecting it are sorted by the pattern distance above. The g-edge is picked if it satisfies these criteria: the supporting line token has not been used previously in this pattern, the line token is less than 1.6 times the length of the previous line token in the pattern, the angle between this g-edge and the previous g-edge is between 0.12 rad and 1.05 rad, and the g-edge preserves the zigzag – does not cross the previous g-edges. Once the g-edge is picked, the algorithm recursively jumps to that connected node, and looks at all connecting g-edges again. If it is impossible to continue from a node, the algorithm backtracks to the previous node, and takes the next option. The search stops when we have at least 4 curves, but not more than 9. The pattern parser is run on each node of the graph.

To allow one missing curve token when a particular node has run out of options, we create a virtual curve that is the same size and opposite orientation as the node, and set its location 3$*$curvesize in the parallel and opposite direction of the node and 0.5 times the distance to the next curve token in the perpendicular direction. To allow one missing line token, we skip checking for a supporting line when looking at possible curves.

4 Results

On a database of 216 images of open hands against cluttered backgrounds, containing 1087 fingers, the overall rate of finger tip curve detection is 75%. This means that on average, close to 4 finger tips are detected. The rate of wedge curve detection is 65%. The number of false positives, as expected, is high. Figure 5 shows some examples.

To minimize the number of false positives, hand detection was run with the option to allow missing tokens turned off. The hand pattern was successfuly found 73% of the time on a database of 244 images of open hands with cluttered backgrounds. On average, about 4 false positives are found per image, but this largely depends on the image's contents. Anything that has a zigzag pattern will be detected, such as stacked binders. Figure 6 shows some examples.

5 Discussion, Future Work, and Conclusions

False negatives in curve token detection are caused by the lack of contrast in the input data, which is necessary for edges to be detected in at least one scale. Given the difficulty of the region where fingers are joined, it is encouraging that wedge curves were found. With closed fingers in particular, detection of wedges is difficult. In the future, more curve models with variable widths can be used to increase detection.

The number of false positive curve tokens found is proportional to the number of edge pixels in the edge image. This is because edges are often curve shaped, and thus detectable by the curve finding algorithm. Also, the curve models are not unique in a sense that a finger tip curve model may detect a wedge curve feature, and a wedge curve model may detect a finger tip curve feature. The number of false positives is not critical, however, because the grammar has really helped to eliminate most of them.

False negatives in hand detection are caused by missing curve tokens. Hand detection performed the best when the fingers were spread, but it worked with closed fingers as well. This is because a hand with fingers spread has a higher

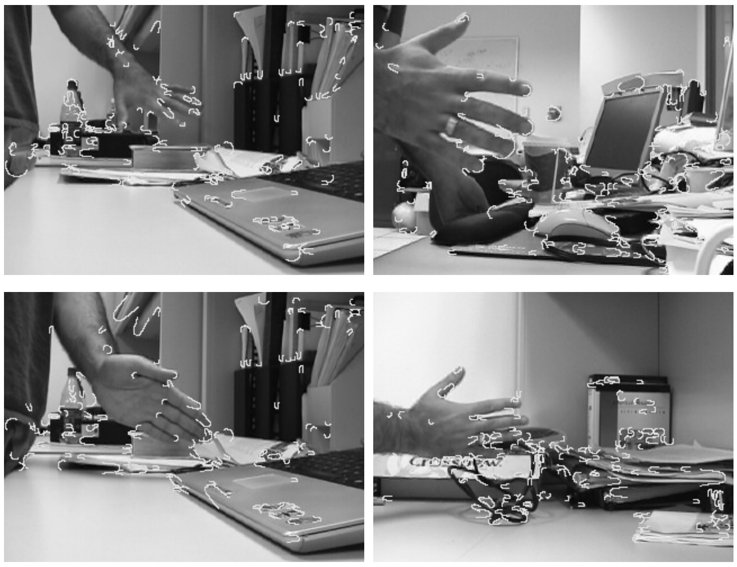

Fig. 5. Examples of detected curve tokens.

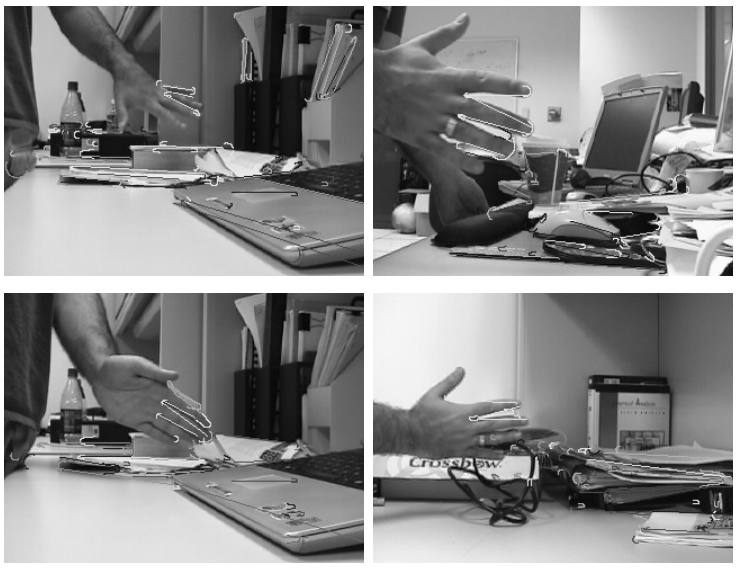

Fig. 6. Examples of detected hand patterns in Fig. 5.

chance of detecting a wedge curve token, as explained above. The already low false positive rate can be decreased further by considering more features, such as a hand's palm, or an image intensity at the location of the finger tip curve tokens.

In this paper, we presented a new scale space edge detector that enabled us to find scale invariant tokens at object boundaries. This token finding should be useful for any object, where its boundary features are important. We showed its promise for open hand detection.

References

1. Lowe, D.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision **60** (2004) 92–110
2. Pavlovic, V.I., Sharma, R., Huang, T.S.: Visual interpretation of hand gestures for human-computer interaction: A review. IEEE Transactions on Pattern Analysis and Machine Intelligence **19** (1997) 677–695
3. Stenger, B., Mendonca, P., Cipolla, R.: Model-based 3d tracking of an articulated hand. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (2001) 310
4. Yuan, Q., Sclaroff, S., Athitsos, V.: Automatic 2d hand tracking in video sequences. In: Seventh IEEE Workshop on Application of Computer Vision. (2005) 250–256
5. Wu, Y., Huang, T.: View-independent recognition of hand postures. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (2000) 2088–2094
6. Athitsos, V., Sclaroff, S.: Estimating 3d hand pose from a cluttered image. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (2003) 432–439
7. Canny, J.: A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence **8** (1986) 679–698
8. Lindeberg, T.: Scale-space theory in computer vision. Kluwer, Boston, MA (1994)
9. Garcia, J., da Vitoria Lobo, N., Shah, M., Feinstein, J.: Automatic detection of heads in colored images. In: Second Canadian Conference on Computer and Robot Vision. (2005) 276–281
10. Burns, J.B., Hanson, A., Riseman, E.M.: Extracting straight lines. IEEE Transactions on Pattern Analysis and Machine Intelligence **8** (1986) 425–455
11. Jain, A., Dubes, R.: Algorithms for Clustering Data. Prentice Hall, Englewood Cliffs, NJ (1988)