Scale Space Based Grammar for Hand Detection

by

Jan Prokaj

A thesis submitted in partial fulfillment of the requirements
for the Honors in the Major Program in Computer Science
in the College of Engineering and Computer Science
and in The Burnett Honors College
at the University of Central Florida
Orlando, Florida

Spring Term
2006

Thesis Chair: Dr. Niels da Vitoria Lobo

Abstract

For detecting difficult objects, such as hands, an algorithm is presented 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. First, the scale space is constructed. Then edges at each scale are found and the scale space is flattened into a single edge image.

To detect a hand pattern, a grammar is defined using curve tokens for finger tips and wedges, and line tokens for finger sides. Curve tokens are found by superimposing a curve model on the scale space edge image and scoring its fit. Line tokens are found by using a modified Burns line finder.

A hand pattern is identified by parsing these tokens using a graph based algorithm. On a database of 200 images of finger tips and wedges, finger tip curves are detected 85% of the time, and wedge curves are detected 70% of the time. On a database of 287 images of open hands against cluttered backgrounds, hands are correctly identified 70% of the time.

Acknowledgments

This project would not have been possible without my research advisor, Dr. Niels Lobo, who has guided me through this difficult field. I would also like to thank the REU program at University of Central Florida, and the College of Engineering and Computer Science for their support of the project.

Contents

List of Figures

List of Tables

1. Introduction

Object detection is one of the fundamental problems of computer vision. The problems of pattern recognition and artificial intelligence have been worked on for a long time, and as a result much progress has been made in detecting rigid objects in images. Detecting hands, however, has appeared to be difficult. One reason for this is that a hand has 21 degrees of freedom (15 for joints, 3 for position, 3 for rotation), allowing it to be in a very large number of configurations. Yet, a solution to this problem has many applications, including human computer interaction, and video surveillance to name a few.

There has been much effort on hand tracking [2], [3], [4], [5], but these do not give answers on detection in a single cluttered image. 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. Without scale invariance, the features will not be stable, and detection will suffer. Indeed, this property plays an important role in the presented scale space hand detection algorithm.

Interestingly, the feature finding approach can be intuitively viewed using formal language concepts. The problem of finding features is cast as a "lexical analysis" on the input image. Features, or tokens, found this way can be "parsed" according to a particular grammar to give a higher meaning. In this case the grammar defines the language of the object to be detected, and the parser becomes an object detection algorithm. In the following chapters, this new approach is described in detail from token finding to token parsing.

2. Methods

The algorithm is divided into two stages: token finding (lexical analysis), and token parsing. These two stages correspond to the feature finding and feature processing stages in standard object detection.

In token finding, the raw input, an array of pixels in this case, is virtually categorized into a set of tokens. Each token type gives new information about the input, therefore it is good to have many different tokens. For example, compare using two tokens with using four tokens. In an input of ten units, and assuming one unit represents one token, the former method will allow one to distinguish $2^{10} = 1024$ different inputs, while the latter will distinguish $4^{10} = 2^{20} = 1,048,576$ different inputs. It is possible that a state represented using two tokens is actually $1,024$ different states! The precision of object detection grows exponentially with the number of tokens.

In token parsing, the tokens are analyzed for certain combinations of their positions in input. These combinations are defined by the grammar of the language we want to "understand." Simple grammars are preferred, because they are easy to implement, but they are not always possible to derive considering the object at hand. Non-rigid objects, such as hands, are especially difficult to reduce to simple grammar rules. Notably, the grammar determines the accuracy of object detection.

The overview of the algorithm is shown in Figure 2.1. Each step is explained in detail in the following sections.

1. Find tokens
    - Calculate scale space edge image
    - Find curve tokens
        - Use finger tip models
        - Use finger wedge models
    - Find line tokens
2. Parse tokens
    - Create a weighted graph using pattern distance
    - Back-tracking search for strings of the grammar

Figure 2.1: Algorithm overview.

2.1 Finding tokens

The kinds of tokens one wants to find depends on the object to be detected. There is a large number of choices: corners, curves, lines, or SIFT features. To detect a hand curve tokens, and line tokens will be used. These naturally describe the shape of fingers, which are the most discriminating feature of a hand. A finger top is curved, as is the wedge between fingers. These are connected by a finger side, easily modeled as a line. This is illustrated in Figure 2.2.

While the tokens are generally unrestricted, Lowe [1] has shown that they should be scale invariant. An image's scale is the level of blur. A small blur provides a very detailed view of an image. This corresponds to "zooming in." A large blur only shows the major characteristics of an image. This corresponds to "zooming out." Without scale invariance, the tokens are not stable and the detection rate is lower. Lowe's SIFT tokens ensure this property, and they produce excellent results when detecting rigid objects that are affinely trackable. But they are less useful for objects that have important information encoded within the shapes of their boundaries. Hand with its 15 joints, and other non-rigid objects

Figure 2.2: Major hand features and joints.

are a good example of this. The curved finger top and a wedge both occur on the boundary of a hand. See Figure 2.2.

Thus, the curve and line tokens that were selected for hand detection will need to be scale invariant. To achieve this goal, the scale space of an image is used. The scale space is defined as the convolution of a variable-scale Gaussian, $G(x,y,\sigma)$, with an input image, $I(x,y)$:

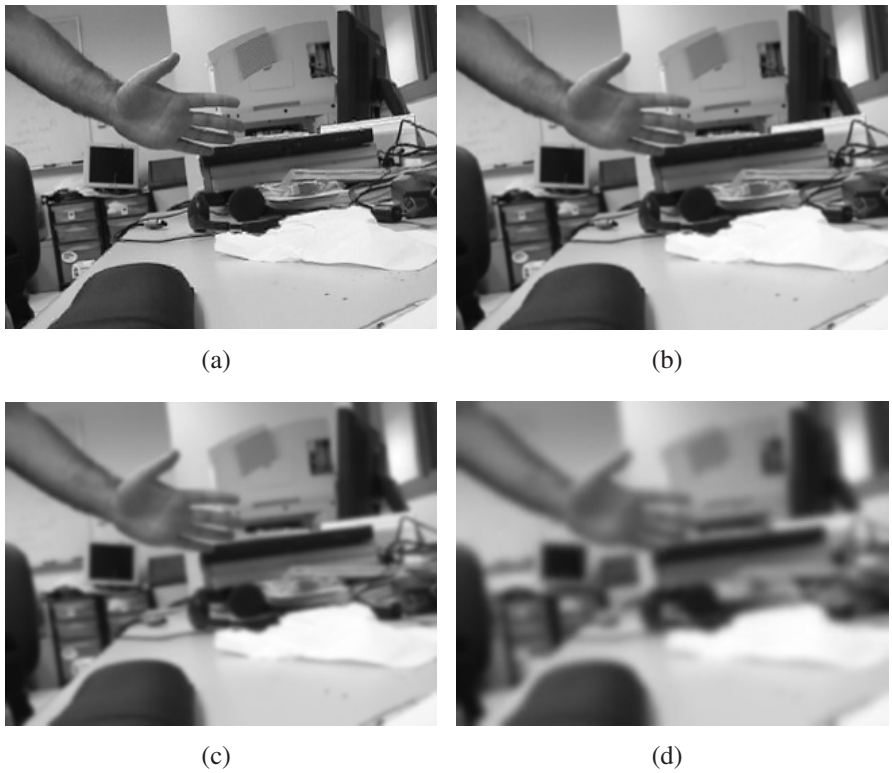
$$G(x,y,\sigma) * I(x,y),$$

where $*$ is the convolution operation in x and y, and

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}}.$$

Figure 2.3 illustrates a portion of the scale space for one image with three different values of σ.

However, to find tokens at boundaries, edge information is needed as well. A single edge image, however, is computed at only one scale. A common technique is to blur the



(a) (b) (c) (d)

Figure 2.3: A portion of the scale space of an image. (a) The original image. (b), (c), (d) Blurred with increasing σ.

image with a particular σ, calculate its gradient, and use it to find local maxima [6]. 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. A scale space of edges is a natural solution. The edges detected using this structure then serve as a good source of scale invariant tokens.

2.1.1 Scale space edge detection

By constructing the scale space of an image, we gain access to the result of all possible blurring 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 constructed. In the second step, this scale space is "flattened" into a single 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. Calculating more octaves does not produce significantly better results.

The Canny scale space is computed 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 any blurring in the convolution, because the blurring is already done by the scale space. A standard 1×3 edge mask, $[-1\ 0\ 1]$, is used for this purpose. The high threshold in the double thresholding step of the algorithm is set so that 25% of the image is edges and the low threshold is set to 35% of the high threshold. This step is illustrated in Figure 2.4.

Step 2: Flattening the space

To convert the Canny scale space into a single edge image, it must be flattened without losing too much information. First, images within each octave are combined into one octave edge image and then the octave edge images are combined into one 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 means that it was "on" in at least one image (or scale) in this octave. Since increasing blurring causes edges to slightly move, the edges from each image in an octave do not overlap each other, and the union image does not have thin edges anymore. Also, the thick edges may have an "off" pixel in the middle, producing a "double" edge. To solve this problem, the

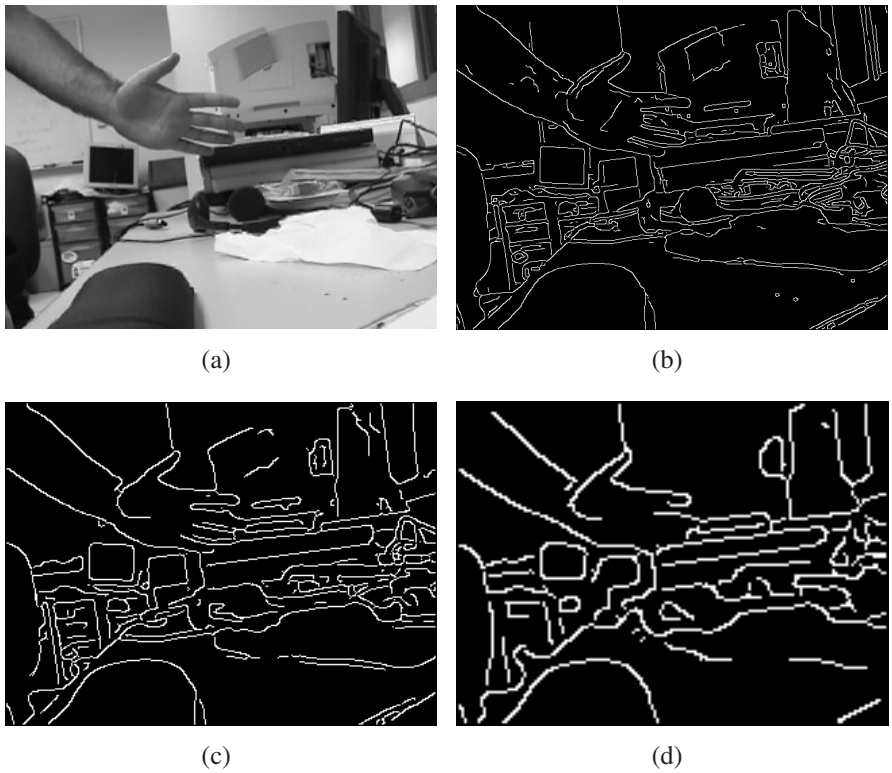


Figure 2.4: A portion of the Canny scale space of an image. (a) The original image. (b), (c), (d) Blurred with increasing σ.

union image is first smoothed by setting on any "off" pixels that are surrounded by at least six "on" pixels, and then it is skeletonized. A skeletonizing operation makes thick edges one pixel thin. See Figure 2.5.

Combining the octave edge images is more involved, because the images are different dimensions. The combinations proceed down the scale. Therefore, the top two octaves are combined first, followed by a combination with the lowest octave image. If more than three octave were used, the combinations would continue this pattern. The combination algorithm overlaps the lower scale image (larger dimension) with the higher scale image (smaller dimension), 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. In other words, 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,

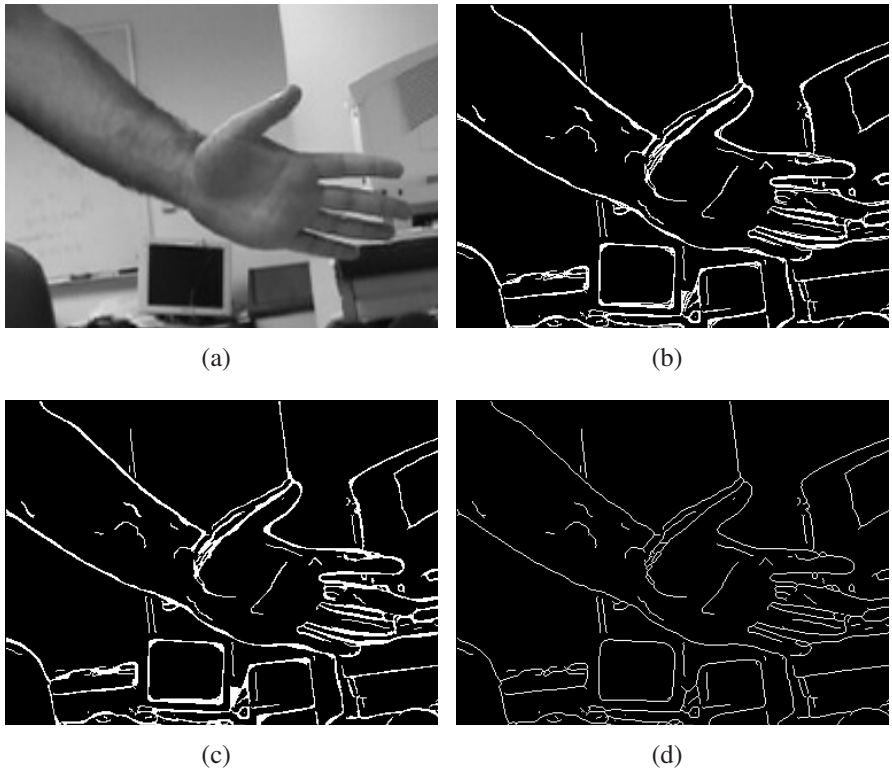


Figure 2.5: Combining the images within an octave. (a) The original image. (b) Unioned octave. (c) Smoothed octave. (d) Skeletonized octave.

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. After all connected segments in the higher scale image are processed, the lower scale image will have as many gaps filled as possible. See Figure 2.6.

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.

2.1.2 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. This is illustrated in Figure 2.2. To find these curves in the scale invariant edge image, a model-based approach is used. The algorithm is based on [8].

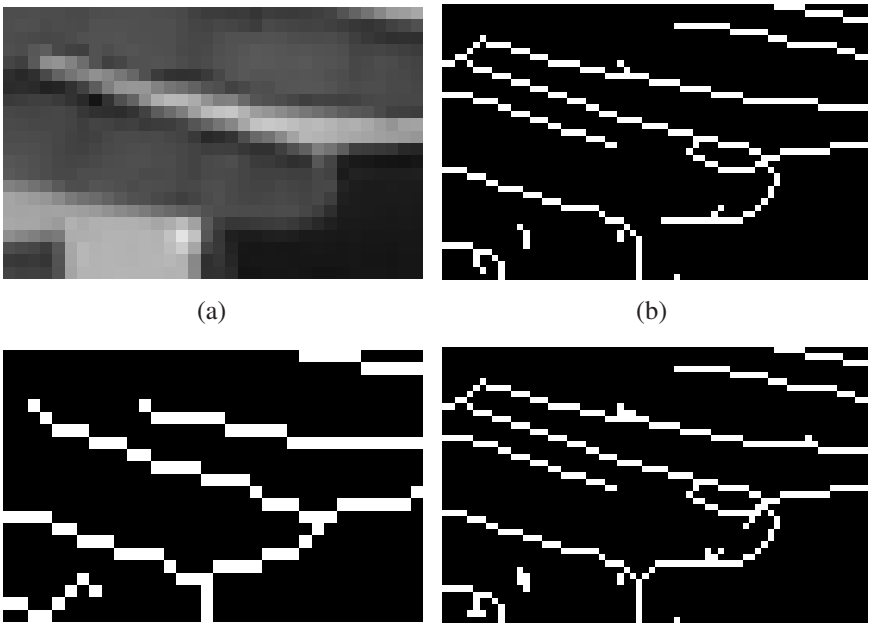


Figure 2.6: Combining octave images (detail). (a) The original image. (b) Lower scale image. (c) Higher scale image. (d) Final edge image.

In a model-based approach to detection, the model is naturally a crucial part of the method. To determine the best model to use for detecting finger tip curves and wedge curves, seven different models are tested. These are illustrated in Figure 2.7. Ideally, only two models are used, where one detects only finger tips, and the other one only wedges between fingers.

Once a model is selected, curves are found by superimposing a model's edge image to different locations in the scale space edge image, and calculating a similarity score. To determine these "candidate" locations, several methods can be used. The best method gives a low number of locations, and gives accurate locations. A simple method would be to divide the edges into different segments, and let the centroid of each segment be a candidate location. This method is fast, but the problem is that it does not guarantee that the locations are on edges. If a location is not on an edge pixel, it is much more difficult to align the model to the image.

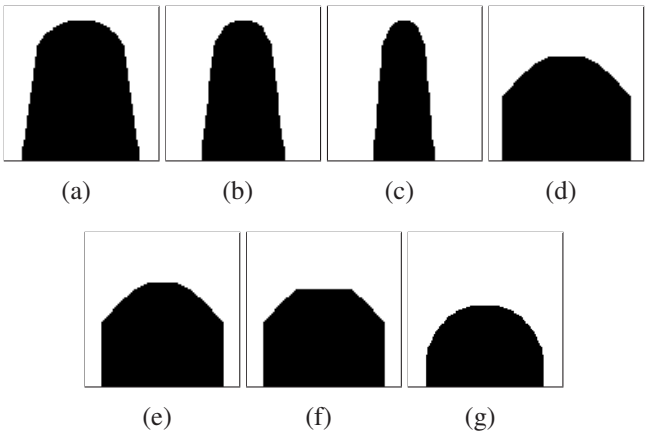


Figure 2.7: Potential models for finger tips and wedges.

A better, but slower, method is to calculate lines in the edge image, and for each line let a candidate location be the endpoints and the midpoint of a line. This method produces many candidate locations at curved edges, which increases the likelihood of finding the best location for a model, but the increased number causes more computation. This is the method that is used in this curve detection algorithm. The line finder used is described in the next section, but it can be any other line finder as long as it calculates lines from edges to guarantee that the candidate locations are on edge pixels.

The algorithm must also deal with different sizes, and orientations of curves. To handle multiple orientations of curves, the selected model is rotated to give 16 rotated versions of the model. More orientations can be handled by creating additional rotated versions. Different curve sizes are handled by precomputing different sized models for each rotated version, keeping the same proportion. There are 41 different sizes used, all with odd dimensions to produce consistent scores across rotation.

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, 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.

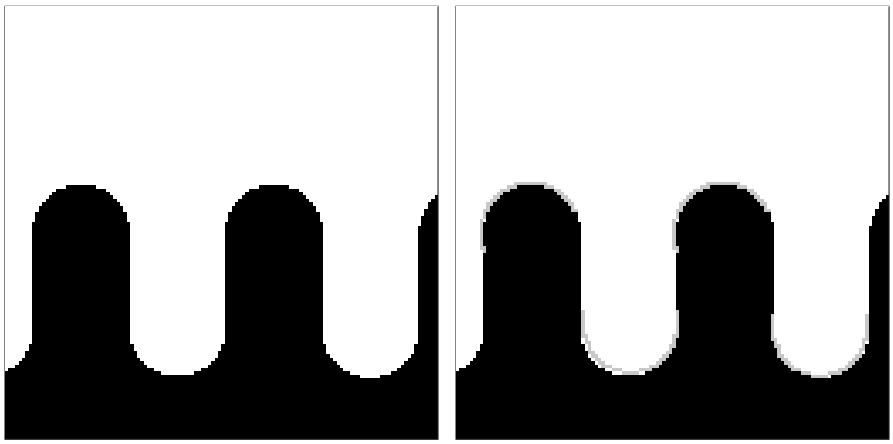
To superimpose a model correctly, the center point of a model edge curve, which is the top of a curve, is aligned with the candidate location. The score of a model, having a particular size and orientation, is calculated using

$$\frac{\sum_k e^{-0.25\min(D_k)} |\vec{m}_k \cdot \vec{v}_k|}{k}$$

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, $\vec{m}_k$ is the unit normal vector at the k-th model edge point, and $\vec{v}_k$ 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 score is the highest if the two points are overlapping each other (distance is zero) and have parallel orientations. The model is "shifted" one unit in four directions during the scoring (for a total of five positions), because this increases its flexibility and increases the score for curves that do not have the exact shape as the model. Orientation data for the model edge points, which is needed to compute the magnitude of the dot product, is determined directly from the model gradient. To quickly determine the closest edge points in the edge image, a distance transform algorithm is used.

The score for each candidate location is compared to a threshold, set to 0.78 in the algorithm, and the location is eliminated for not fitting the model if the score is less than this. This score, however, only tells how well *any* curve fits the model. It does not distinguish between curves on fingers or any other curves.

To eliminate curves that are not likely to be finger tips or wedges, it is required that each curve found is supported by a line token, as found by a line finder described in the next section. The line must be parallel, and close to either one of the endpoints of the curve. This is accomplished by searching for lines in a rectangle 7 pixels wide and $2 * curvesize$



(a) (b)

Figure 2.8: Curves detected on the sample image (a) are shown in gray.

pixels long 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 $0.55rad$ of the curve's orientation, and their length must be at least $1.1 * curvesize$.

Since the number of candidate locations is large, especially on curved edges, it is possible that this process produces overlapping curves for neighboring candidate locations. These duplicate curves are eliminated by calculating the intersection of the bounding rectangles for all pairs of 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 model(s) for finger tips as well as for wedges. To determine these models from the seven tested, a combination with the highest detection rate and a reasonable false positive rate is used. This combination is determined in the results and discussion sections. Figure 2.8 shows a sample output of this algorithm.

2.1.3 Line tokens

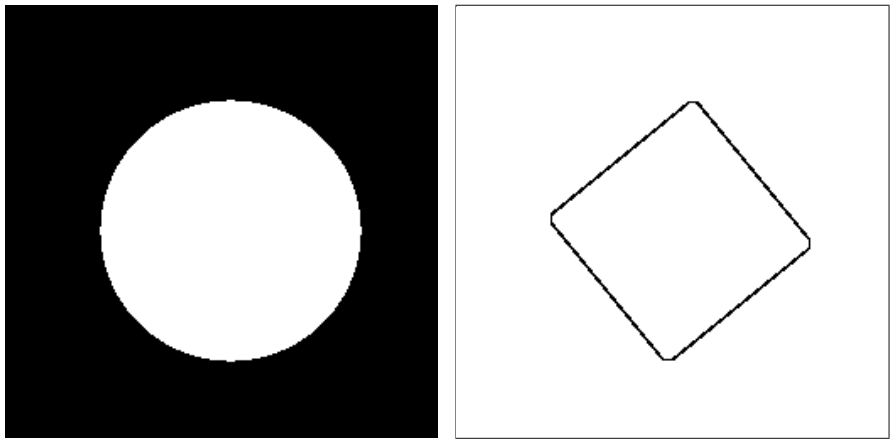
To find lines on a scale space edge image, a modified Burns line finder [9] is used. The original Burns algorithm calculates the lines from the raw intensity image. This is not ideal,

because the curve tokens are based on edge information. If the lines are not determined from the same source, they would not necessarily be aligned with the curves, potentially causing problems in the token parsing stage. Therefore, the algorithm is modified to extract the lines directly from the edge image. 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 finds lines by looking for a group of pixels with the same orientation. Lines are discretized into four different types, so that the orientation of pixels on one line can vary by 45°. To avoid boundary effects, two such systems are determined, the second system having orientations offset by 22.5°. These two systems are then resolved into a final set of lines. As a result of the first modification to use edge information, the resolving operation also has to be modified.

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, the line merges with line(s) on the same pixels 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 line(s) 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 lines on finger sides. Figure 2.9 shows the output of this algorithm on a synthetic sample image.

Even with this controlled merging, few inconsistencies may arise in the final set of lines. For example, a line may have some of its pixels separated from the main segment. These cases can not be prevented easily, as they arise with the formation of two line systems. This is not a concern, however, as this does not happen frequently, or in critical places in the image.



(a) (b)

Figure 2.9: Lines detected on a circle (a) using the modified Burns line finder.

2.2 Parsing tokens

In the second stage of object detection, the tokens from the first stage are analyzed for certain patterns. These patterns are called strings, because they are defined by the grammar of the object's "language." The set of tokens is then the alphabet of this language.

To detect a hand using this alphabet of curves and lines, the grammar is defined to enforce 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 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.

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

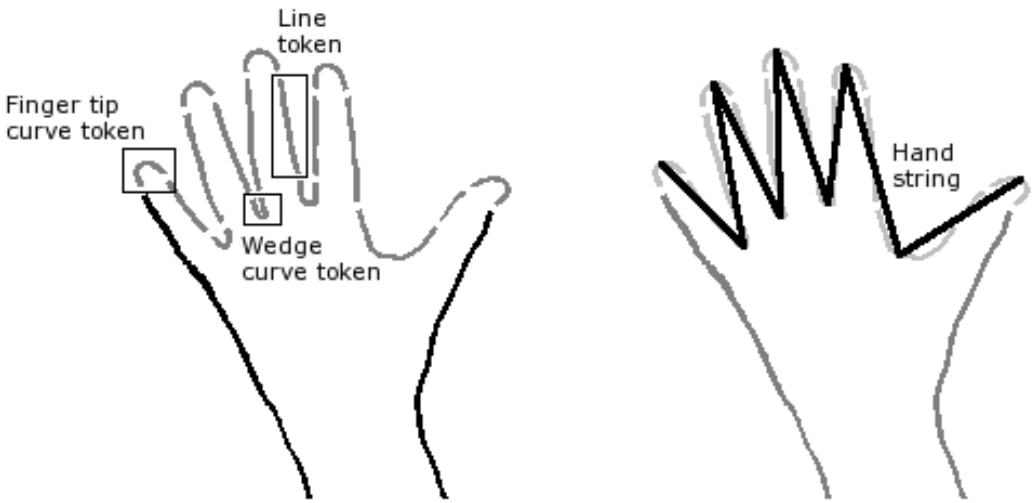


Figure 2.10: This figure illustrates the zigzag pattern we are looking for.

This grammar eliminates strings with three or less curves. This means that only hands with two or more fingers will be detected. Three curves or less do not give enough confidence that a hand is found, so this requirement eliminates many false positives. The -t-w-t- sequence produces a "zigzag" pattern across the hand's fingers. See Figure 2.10 for an illustration. It is allowed for one token to be missing, but this is not easily expressible in the grammar.

2.2.1 Parsing algorithm

As discussed above, the input to the parsing algorithm, the parser, is a set of tokens found in the first stage. The parser analyzes these tokens for strings of the "hand" language, as defined by the grammar. Since the grammar is simple, just a repeating pattern, it tremendously simplifies parsing.

The strings can be viewed as particular combinations of tokens. A natural approach for searching for combinations is a graph. First, a graph is constructed containing all possible pairs of tokens. In practice, however, the graph contains only possible curve pairs, and such pairs not separated by a line token are removed in the second step. The edges in the graph,

termed g-edges and not to be confused with image edges, correspond to the possible curve pairs. The weight of each g-edge is the pattern distance [10] between two curves (nodes). The pattern distance is defined as a Euclidian pattern distance,

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

where *xRange* is the maximum distance in the x-dimension and *yRange* is the maximum distance in the y-dimension. Using this formula, the 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 using the first property of the grammar.

In the second step those g-edges are removed that cannot possibly make a valid combination. The criteria are determined from the properties defined 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 essentially says that a g-edge must connect curves back to back, where the back is the two endpoints, not the center of the curve. This is determined by comparing two pixel distances, the plain distance between two curves and a distance from the point one step above the top (center) of one curve to the second curve. If the second distance is less than the first, the curves are not oriented back to back and the g-edge is removed. 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 the line must be $< 0.18 rad$, 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 recursive 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 and the algorithm goes forward if it satisfies several criteria. If it is impossible to

continue from a node, the algorithm back-tracks to the previous node, and takes the next option. The search stops when at least four curves form the pattern, but not more than nine. The pattern parser is run on each node of the graph.

The criteria are as follows. The supporting line token, found in the previous step, must not have been used previously in this pattern, and it must be less than 1.6 times the length of the previous line token in the pattern. This comes from the fact that fingers on a hand are more or less the same length. The angle between this g-edge and the previous g-edge is between $0.12rad$ and $1.05rad$. This is a range for the natural spread of the fingers. Also, the g-edge must preserve the "zigzag." It is possible that a g-edge satisfies all the previous criteria, but crosses a previous g-edge(s), breaking the pattern. This only needs to be checked for the third and later curves in the pattern. To enforce this, a g-edge is removed if it intersects, or is on the wrong side of a triangle formed from the previous two g-edges. Once the g-edge is picked, the algorithm recursively jumps to that connected node, and looks at all connecting g-edges again.

To allow one missing curve token when a particular node has run out of options, a virtual curve can be created 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, checking for a supporting line is skipped when looking at possible curve pairs. This case is not implemented however.

Therefore, the output of this algorithm is groups of curves and lines (strings) that outline the hand's fingers. It is possible to get more than one group of curves for the same hand, since the parser is run on each node of the graph. However, the groups would be subsets of one large group covering all curves on the hand, so this is not a cause for concern. Nevertheless, a small addition to the parser would be to remove these subsets.

3. Results

There are two sections of results. The first one shows the results of determining the best models to use for finger tip and wedge curve detection. To determine the best models for finger tip curve detection, the seven models were tested on a database of 200 images of finger tips against cluttered backgrounds. Similarly, to determine the best models for wedge curve detection, the models were tested on a database of 200 images of wedges against cluttered backgrounds. For each image in each database it was recorded what models detected the curve. The total number of curves found was recorded as well to calculate the false positive rate.

The second section shows the results of hand detection using the curve and line tokens, and the parsing algorithm. To decrease the number of false positives, the parser was run with the option to allow missing tokens turned off. The algorithm was run on a database of 287 images of open hands against cluttered backgrounds. For each image in the database, it was recorded whether a hand was detected or not. A hand was considered detected if at least one hand string was found across the hand's fingers.

3.1 The best models for curve detection

The detection rate and false positive rate for models run on the finger tip database is shown beginning with Table 3.1. To see the models, refer back to Figure 2.7. The false positive rate in the tables is a ratio of total curves found to correct curves found. Therefore, a ratio of 1.5 means that for every correct curve found, there are 0.5 false positive curves found.

There were 3 types of combinations investigated: single models, pairs of models, and groups of three models. Larger groups were avoided for high computation cost with presumably only a slight improvement in detection rate.

Following the tables, there are three example finger tip images shown. On the first one, models (d), (e), (f), and (g) detect the tip, while on the second example, models (a), (b), (c), detect the tip. The first image also shows some false positives. The third image shows the case when a finger tip is detected by all models except (c).

The detection rate and false positive rate for models run on the wedge database is shown beginning with Table 3.4. Again, to see the models, refer back to Figure 2.7. As with the finger tip curves, there are three example wedge images shown following the tables. On the first one, models (a), (b), (c), and (f) detect the wedge, while on the second example, models (d), (e), (f), and (g) detect the wedge. The second image also shows some false positives. The third image shows the case when a wedge is detected by all models.

3.2 Hand detection

The models (c), (e), and (f) were chosen for hand detection. This is explained in the next section. The hand detection rate using models (c), (e), and (f) is 70%. The number of false positives was not explicitly counted, but the false positive rate by inspection is between 4.0 and 5.0.

There are 8 example images shown at different parts of the algorithm. The first part, Figure 3.7, shows the curve tokens in these images. Each color corresponds to a different model: yellow is model (c), blue is model (e), and green is model (f). The second part, Figure 3.8, shows the line tokens. The colors are only used for distinguishing between adjacent lines. The third part, Figure 3.9, shows the detected hand strings as defined by the grammar. Each string is assigned one of six colors.

Table 3.1: Results of testing single models on a finger tip database.

	(a)	(b)	(c)	(d)	(e)	(f)	(g)
Detection rate (%)	74.0	49.0	39.5	74.0	78.5	75.5	79.5
False positive rate	1.49	1.90	1.86	1.51	1.57	1.48	1.67

Table 3.2: Results of testing pairs of models on a finger tip database.

	(a) + (b)	(a) + (c)	(a) + (d)	(a) + (e)	(a) + (f)	(a) + (g)
Detection rate (%)	80.5	81.0	82.5	83.5	82.0	84.0

	(b) + (c)	(b) + (d)	(b) + (e)	(b) + (f)	(b) + (g)	(c) + (d)
Detection rate (%)	53.0	86.0	87.0	85.0	87.0	85.5

	(c) + (e)	(c) + (f)	(c) + (g)	(d) + (e)	(d) + (f)	(d) + (g)
Detection rate (%)	86.5	85.5	87.5	79.5	79.0	81.0

	(e) + (f)	(e) + (g)	(f) + (g)
Detection rate (%)	80.5	82.5	81.0

Table 3.3: Results of testing groups of three models on a finger tip database.

	(a) + (b) + (c)	(a) + (b) + (d)	(a) + (b) + (e)	(a) + (b) + (f)
Detection rate (%)	82.5	87.0	88.5	87.0

	(a) + (b) + (g)	(a) + (c) + (d)	(a) + (c) + (e)	(a) + (c) + (f)
Detection rate (%)	88.5	87.0	88.5	87.0

	(a) + (c) + (g)	(a) + (d) + (e)	(a) + (d) + (f)	(a) + (d) + (g)
Detection rate (%)	88.5	84.0	83.5	84.5

	(a) + (e) + (f)	(a) + (e) + (g)	(a) + (f) + (g)	(b) + (c) + (d)
Detection rate (%)	83.5	85.0	84.5	87.5

	(b) + (c) + (e)	(b) + (c) + (f)	(b) + (c) + (g)	(b) + (d) + (e)
Detection rate (%)	88.5	87.0	89.0	87.5

	(b) + (d) + (f)	(b) + (d) + (g)	(b) + (e) + (f)	(b) + (e) + (g)
Detection rate (%)	87.5	88.5	87.5	89.0

	(b) + (f) + (g)	(c) + (d) + (e)	(c) + (d) + (f)	(c) + (d) + (g)
Detection rate (%)	87.5	87.0	87.5	88.5

	(c) + (e) + (f)	(c) + (e) + (g)	(c) + (f) + (g)	(d) + (e) + (f)
Detection rate (%)	87.5	89.0	88.0	81.5

	(d) + (e) + (g)	(d) + (f) + (g)	(e) + (f) + (g)
Detection rate (%)	82.5	82.0	83.0

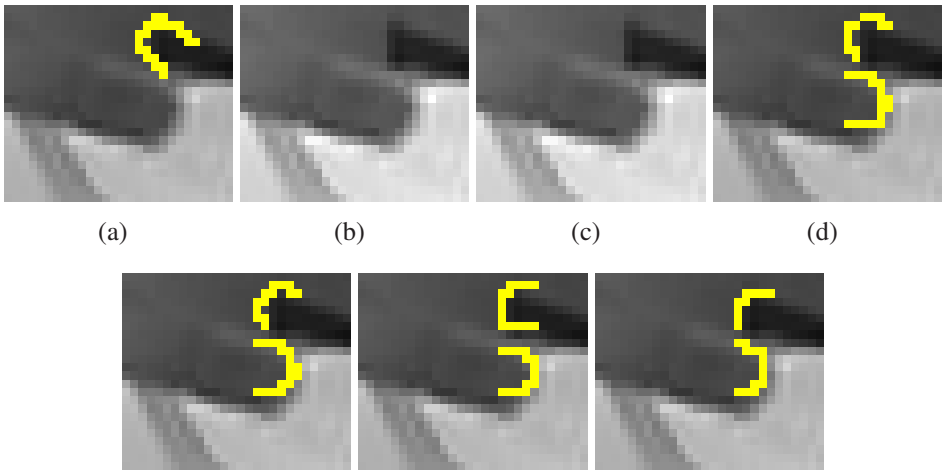


Figure 3.1: Tips detected on an example image 1 using different models.

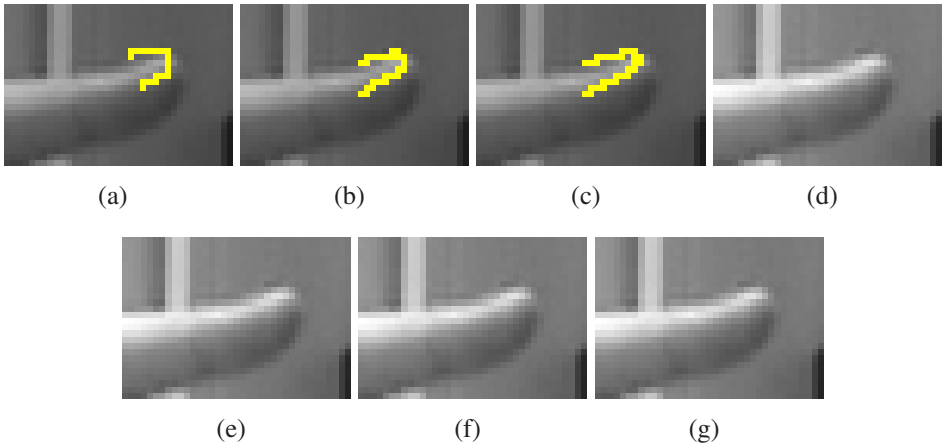


Figure 3.2: Tips detected on an example image 2 using different models.

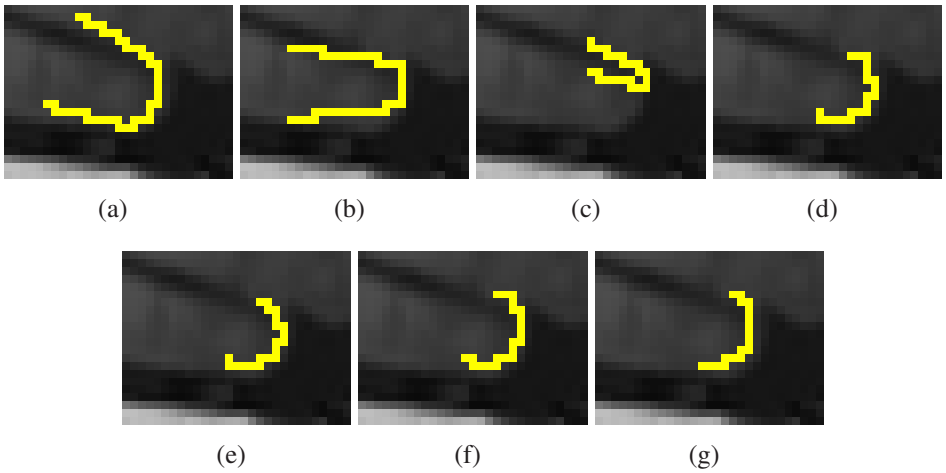


Figure 3.3: Tips detected on an example image 3 using different models.

Table 3.4: Results of testing single models on a wedge database.

	(a)	(b)	(c)	(d)	(e)	(f)	(g)
Detection rate (%)	55.5	55.5	54.0	47.5	59.0	56.5	59.0
False positive rate	3.31	3.05	3.01	3.08	2.88	2.83	3.06

Table 3.5: Results of testing pairs of models on a wedge database.

	(a) + (b)	(a) + (c)	(a) + (d)	(a) + (e)	(a) + (f)	(a) + (g)
Detection rate (%)	60.5	62.5	64.0	66.0	66.5	67.5

	(b) + (c)	(b) + (d)	(b) + (e)	(b) + (f)	(b) + (g)	(c) + (d)
Detection rate (%)	59.5	67.0	69.0	70.5	69.5	67.5

	(c) + (e)	(c) + (f)	(c) + (g)	(d) + (e)	(d) + (f)	(d) + (g)
Detection rate (%)	69.5	71.0	70.0	59.5	59.0	61.0

	(e) + (f)	(e) + (g)	(f) + (g)
Detection rate (%)	64.0	63.0	64.5

Table 3.6: Results of testing groups of three models on a wedge database.

	(a) + (b) + (c)	(a) + (b) + (d)	(a) + (b) + (e)	(a) + (b) + (f)
Detection rate (%)	63.0	68.0	69.5	70.5

	(a) + (b) + (g)	(a) + (c) + (d)	(a) + (c) + (e)	(a) + (c) + (f)
Detection rate (%)	70.0	69.0	70.5	71.0

	(a) + (c) + (g)	(a) + (d) + (e)	(a) + (d) + (f)	(a) + (d) + (g)
Detection rate (%)	71.0	66.5	67.5	68.5

	(a) + (e) + (f)	(a) + (e) + (g)	(a) + (f) + (g)	(b) + (c) + (d)
Detection rate (%)	68.0	68.0	69.5	68.5

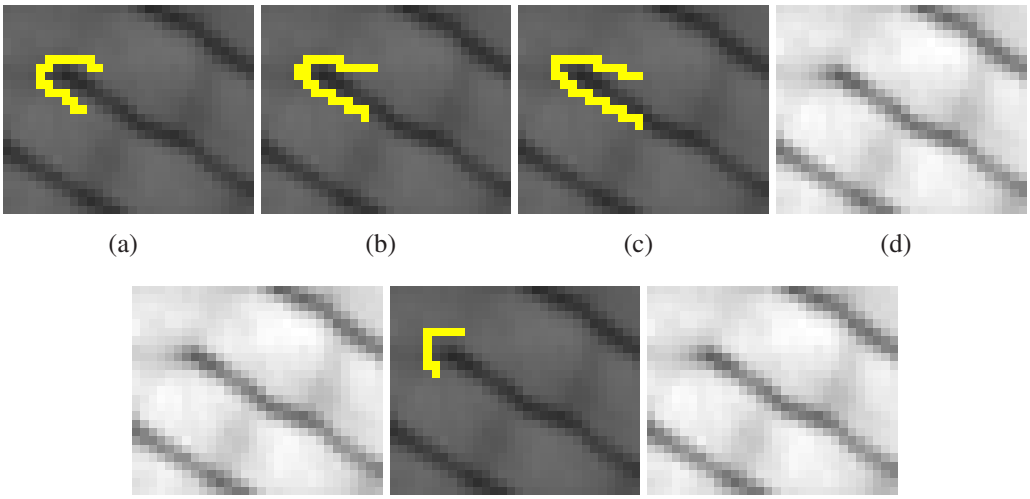
	(b) + (c) + (e)	(b) + (c) + (f)	(b) + (c) + (g)	(b) + (d) + (e)
Detection rate (%)	70.0	71.5	70.5	69.5

	(b) + (d) + (f)	(b) + (d) + (g)	(b) + (e) + (f)	(b) + (e) + (g)
Detection rate (%)	71.5	70.5	71.5	70.5

	(b) + (f) + (g)	(c) + (d) + (e)	(c) + (d) + (f)	(c) + (d) + (g)
Detection rate (%)	72.0	70.0	71.5	71.0

	(c) + (e) + (f)	(c) + (e) + (g)	(c) + (f) + (g)	(d) + (e) + (f)
Detection rate (%)	72.0	71.0	72.5	64.0

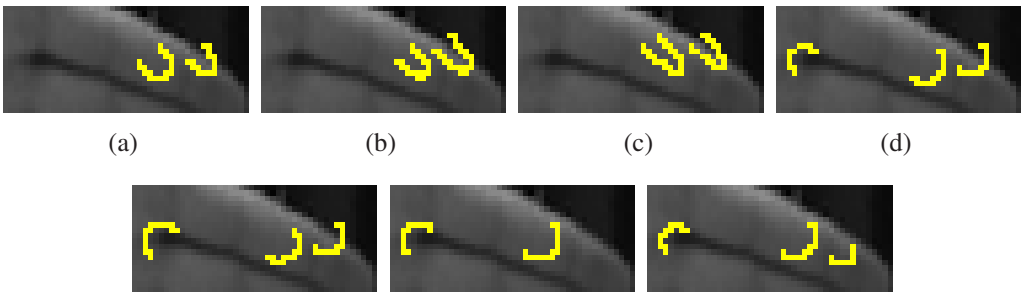
	(d) + (e) + (g)	(d) + (f) + (g)	(e) + (f) + (g)
Detection rate (%)	63.5	65.5	67.0



(a) (b) (c) (d)

(e) (f) (g)

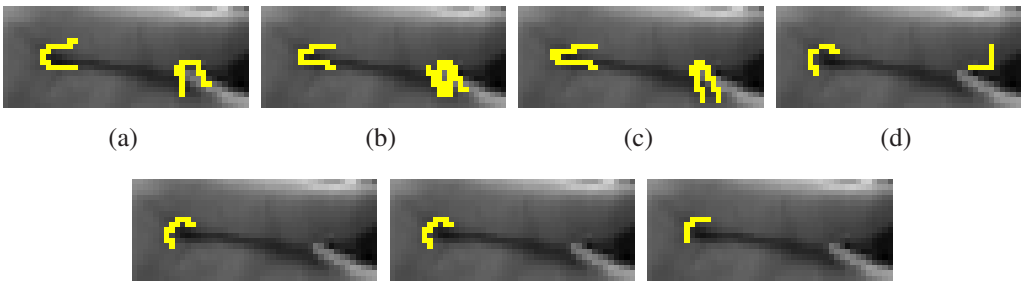
Figure 3.4: Wedges detected on an example image 1 using different models.



(a) (b) (c) (d)

(e) (f) (g)

Figure 3.5: Wedges detected on an example image 2 using different models.



(a) (b) (c) (d)

(e) (f) (g)

Figure 3.6: Wedges detected on an example image 3 using different models.

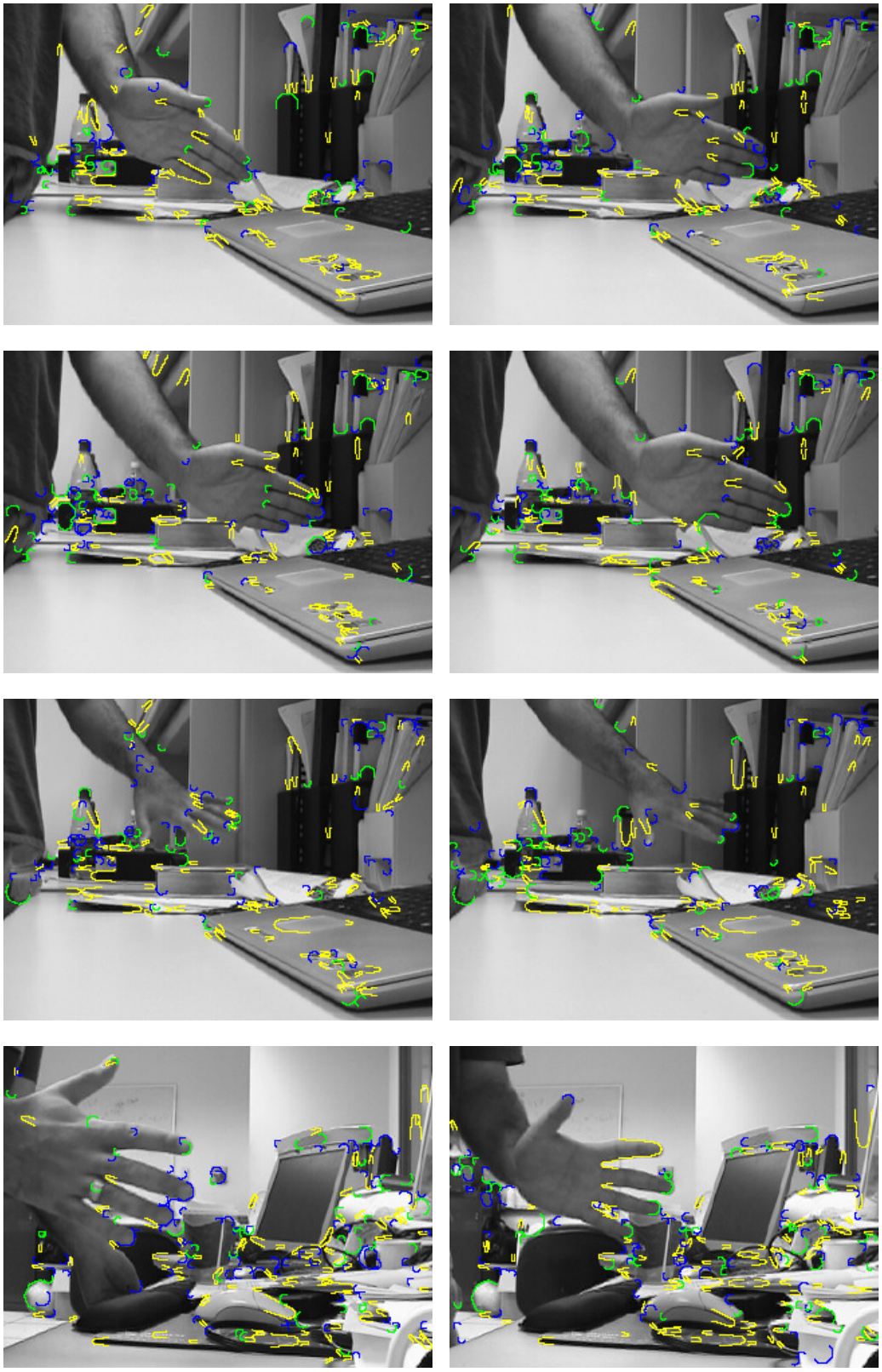


Figure 3.7: Examples of detected curve tokens.

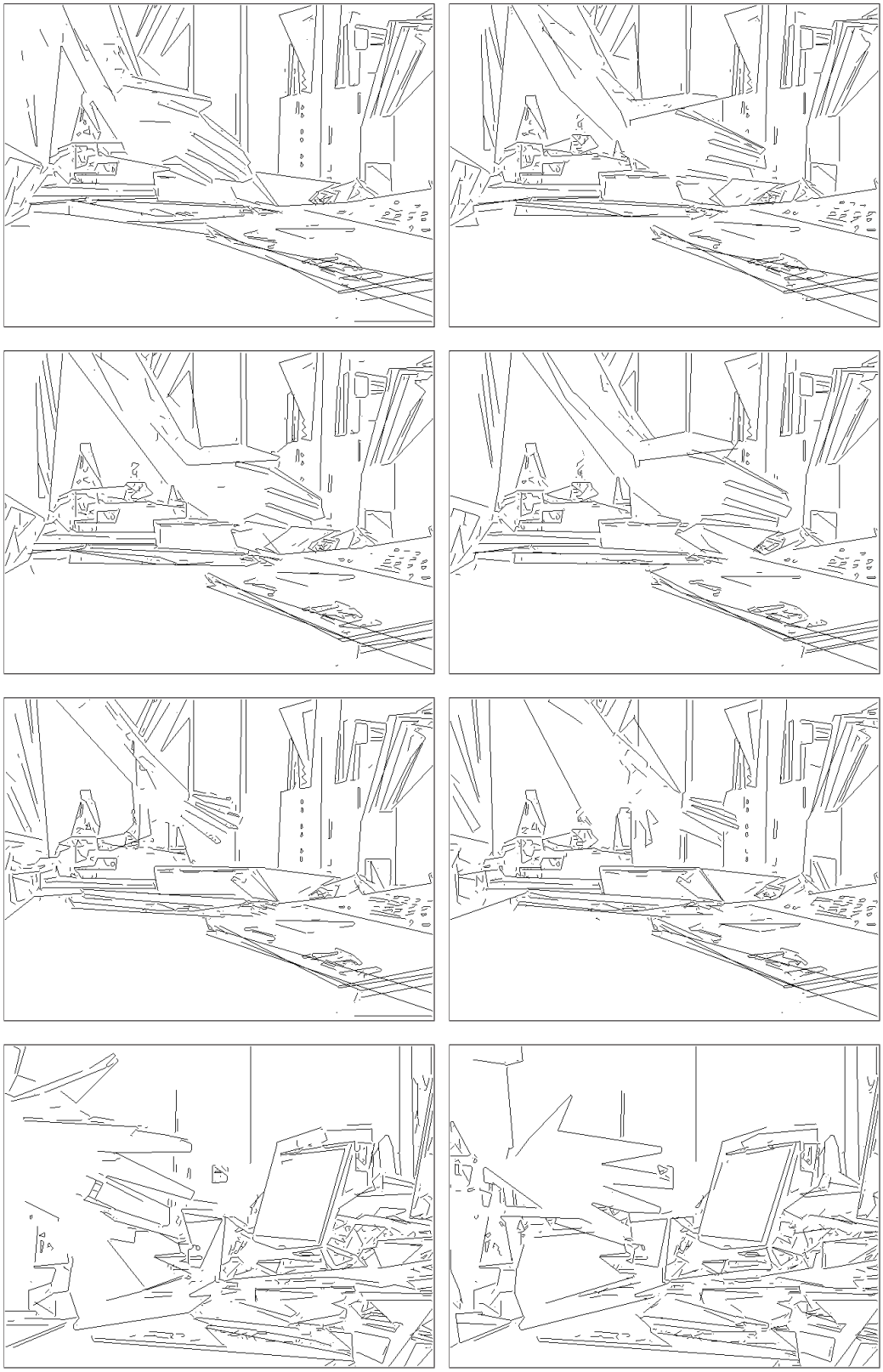


Figure 3.8: Examples of detected line tokens for images in Figure 3.7.

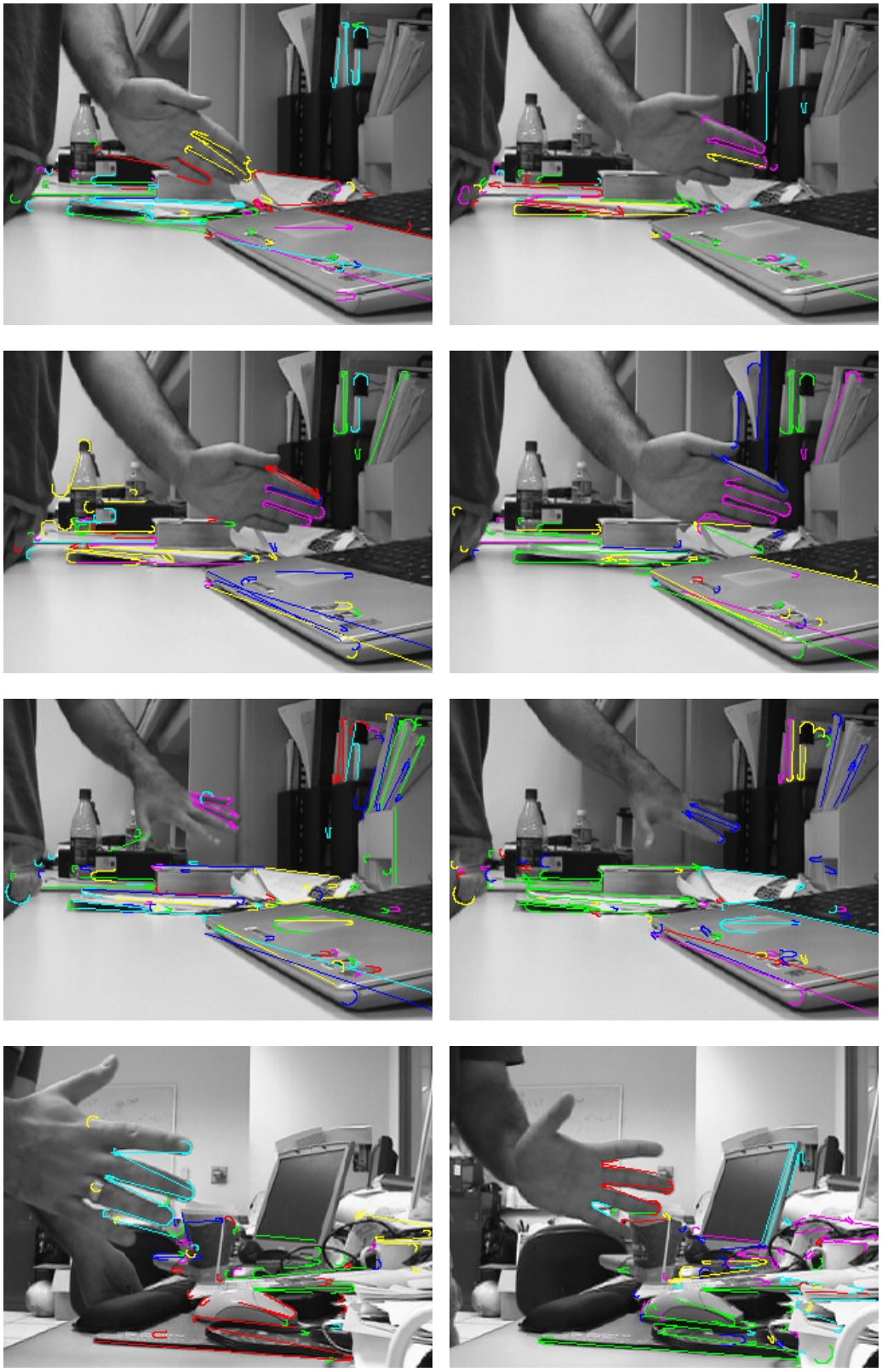


Figure 3.9: Examples of detected hand patterns for images in Figure 3.7.

4. Discussion

The highest detection rate of finger tips using a single model is 79.5% for model (g). Using two models, this figure jumps to 87.5% for a pair of models (c) and (g). Going to a group of three models is not as dramatic, the highest detection rate rises to 89% for models (b), (c), and (g). The false positive rate, however, rises with each model addition as well, because the number of new detections is lower than the new total of curves found. To determine the best model(s) for curve detection, both the detection rate and the potential false positive rate must be considered.

With these criteria, using two models is the best option. It is better than using a single model, because the detection rate rises significantly; the maximum goes up 8%, and the average goes up over 14%. It is also better than using three models, because the detection rate further rises only 1.5% for the maximum, and 4.5% for the average. This change is likely to be offset by the higher false positive rate.The highest detection rate of two models is 87.5% given by (c) and (g), with the false positive rate being 1.86 and 1.67 respectively. A better alternative is to use models (c) and (e), which gives a slightly smaller detection rate of 86.5%, but a lower false positive rate of 1.86 and 1.57 respectively.

The best models for wedge detection are determined similarly. The highest detection rate of wedges using a single model is 59% for model (g). Using two models, this figure jumps to 71%, and using three models it is 72.5%. Again, using two models is the best option. It is better than using a single model, because the detection rate is significantly better; the maximum goes up 12%, and the average goes up 10%. It is also better than using three models, because the detection rate further rises only 1.5% for the maximum,

and 4% for the average. This change is likely to be offset by the higher false positive rate.The highest detection rate of two models is given by models (c) and (f), with the false positive rate being 3.01 and 2.83 respectively. This combination is the best, because it gives a relatively low false positive rate, while achieving a high detection rate.

It is important to realize that when the selected models are used in the curve token detection algorithm, they are used together, and no distinction is made which models are detecting finger tips, or wedges. Therefore, the models used for detecting tips and wedges are (c), (e), and (f). The corresponding detection rate is 87.5%/72% for the two databases. The 87.5% detection rate on finger tip database means that on average, 4 out of 5 finger tips are detected. The 72% detection rate on wedge database means that on average, 2 out of 4 wedges are detected.

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 at such a rate. With closed fingers in particular, detection of wedges is difficult.

The number of false positive curve tokens is high, because it is proportional to the number of edge pixels in the edge image. Edges are often curve shaped, and thus detectable by the curve finding algorithm. The high number of false positives is not critical, however, because the grammar really helps to eliminate most of them.

The hand detection rate is 70%. A major factor in this rate is the detection of wedges. The parser needs a continuous sequence of curves, which means that when all finger tips are found, and there are no wedges found, a hand will not be detected. The same applies when all wedges are detected and no finger tips are, but this case is rare, since wedges have a detection rate almost 20% lower than finger tips do. This can be solved by allowing missing curve tokens. The parser has an option of allowing one missing curve token, but it creates too many false positives and needs to be investigated further before it can be useful.

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 chance of detecting a wedge curve token, as explained above. The false positive rate largely depends on the contents of the image. Anything that has a "zigzag" pattern will be detected, such as stacked binders. The false positive rate can be decreased by considering more features, such as a hand's palm, or an image texture at the location of the finger tip curve tokens for greater accuracy.

5. Conclusions

A new approach to object detection was presented. Using tokens and then parsing them according to the grammar of the object's language is an intuitive approach that shows the promise of using formal language theory in computer vision.

Additionally, a new scale space edge detector was presented that enables to find scale invariant features at object boundaries. This edge detector has many applications; it can be used anywhere standard edge detectors are. It should be especially useful for those objects where boundary features are important.

An algorithm for hand detection was presented as well. It gives a promising direction for detecting hands in cluttered images. Extending its applicability to more hand poses should be seriously considered.

Bibliography

[1] Lowe, D. *Distinctive image features from scale-invariant keypoints*. IJCV. Vol. 60 (2004). Issue 2. 92-110.

[2] Kolsch, M. and Turk, M. *Robust hand detection*. Proc. Intl. Conf. Face and Gesture Recognition. 614-619, 2004.

[3] Pavlovic, V. I. et al. *Visual interpretation of hand gestures for human-computer interaction: A review*. IEEE Trans. on PAMI. Vol. 19 (1997). Issue 7. 677-695.

[4] Wu, Y. and Huang, T. *View-independent recognition of hand postures*. Proc. Intl. Conf Computer Vision and Pattern Recognition. 2088-2094, 2000.

[5] Yuan, Q. et al. *Automatic 2D hand tracking in video sequences*. Proc. IEEE work, Motion and Video Computing. 250-256, 2005.

[6] Canny, J. *A Computation Approach to Edge Detection*. IEEE Trans. on PAMI. Vol. 8 (1986). Issue 6. 679-698.

[7] Lindeberg, T. Scale-space theory in computer vision. Boston: Kluwer, 1994.

[8] Garcia, J. et al. *Automatic Detection of Heads in Colored Images*. CRV 2005. 276-281.

[9] Burns, J. et al. *Extracting straight lines*. IEEE Trans. on PAMI. Vol. 8 (1986). Issue 4. 425-455.

[10] Jain, A. and Dubes, R. Algorithms for Clustering Data. Englewood Cliffs, NJ: Prentice Hall, 1988.