MEMO #93–11
August 1993

Finding Lines under Bounded Error

Thomas M. Breuel
IDIAP, C.P. 609, 1920 Martigny, Switzerland
tmb@idiap.ch

ABSTRACT

A new algorithm for finding lines in images under a bounded error noise model is described. The algorithm is based on a hierarchical and adaptive subdivision of the space of line parameters, but, unlike previous adaptive or hierarchical line finders based on the Hough transform, measures errors in image space and thereby guarantees that no solution satisfying the given error bounds will be lost. In addition, the algorithm can find interpretations of all the lines in the image that satisfy the constraint that each image feature supports at most one line hypothesis–a constraint that is often useful to impose in practice. The algorithm can be extended to compute the probabilistic Hough transform and the generalized Hough transform a variety of statistical error models efficiently.

1 Introduction

Finding straight lines in images is one of the most fundamental problems in computer vision. This paper describes a new algorithm for finding lines in images. The algorithm differs in a number of important ways from existing methods. It can also be extended to detect other analytic shapes or even arbitrary geometric models; in this paper, we will focus our attention on a version of the algorithm that finds lines in an image subject to given bounds on the deviations of the location and orientation of image features from a hypothesized line.

This paper has been submitted for publication to "Pattern Recognition".

The algorithm can be used to find the maximal or "optimal" line in an image, in the sense of finding the line for which the greatest total length is supported by edge pixels in the image under the given error bounds.

An extension of the algorithm can be used to find quickly a global interpretation of the edge pixels in an image as lines in decreasing order of length of support, imposing the constraint that no edge pixel can be part of two different lines.

There are a number fundamental concerns for any method for finding lines. These can be roughly grouped into correctness and efficiency. Let us illustrate these concerns using the simple or standard Hough transform–probably the most popular method for finding lines in images–as an example.[1,2]

The standard Hough transform is implemented in the following way. Lines are considered to be parameterized by ρ, their distance from the origin, and θ, their angle with the x-axis (other parameterizations are possible[3] and used commonly). In the computation of the Hough transform, the parameter space is quantized and represented as a discrete data structure, often an array or a hash table. Each "cell" or "bin" in this data structure corresponds to a small range of parameters ρ and θ.

For each edge pixel (optionally associated with a local orientation), there is an associated set of lines that the edge pixel could be a part of. This set of lines corresponds, in turn, to a set of bins in the quantized parameter space.

The standard Hough transform now considers each edge pixel in the input image and increments a counter in each of the corresponding bins in the quantized parameter space ("vote for that bin" in the language of the Hough transform). More sophisticated versions of the Hough transform do not increment each counter by a fixed amount, but instead compute a "degree of membership", for example related to a probability distribution, for each edge pixel to a bin in the quantized parameter space.

In a final stage, the quantized parameter space is searched for bins that contain a number of votes that is larger than a given threshold and/or forms a local maximum.

There are a number of theoretical and practical problems with such a simple approach, and a large number of solutions and modifications to the simple Hough transform have been proposed and studied in the literature.

First, the quantization of parameter space can easily result in the splitting of the votes belonging to a single hypothesized line among a number of bins. To overcome this problem, parameter space is sometimes subjected to filtering operations, or, similarly, neighboring bins are considered together in the final evaluation of the quantized parameter space.[4,5] Quantization has been found to be a problem, for example, with the Adaptive Hough Transformation,[1] and methods for "anti-aliasing" the Hough transformation have been proposed.[6] The method described in this paper explicitly avoids quantization errors, and no separate post-processing step to counteract the effects of quantization is necessary.

Closely related is the issue that errors in the localization of edge pixels are modeled in parameter space rather than in image space. Since properties of the processing stages preceding line finding (edge detection, pixel chaining, etc.) are most often naturally expressed in image space, this complicates the problem of obtaining solutions with well-defined geometric or statistical properties.[5,7,8,6,9]

Furthermore, the votes in a single bin may represent multiple, different lines (e.g., nearby parallel lines or widely separated colinear segments that should be treated separately). This problem has been addressed partially in the literature by backmapping the edge pixels from a bin in the quantized parameter space into the image and applying some kind of verification procedure to the set of edge pixels obtained in that way.[10,11] However, such an approach is not entirely satisfactory, because the constraints used for verification might also profitably be employed during the construction of the quantized parameter space, not just in a final verification step. The method described in this papers allows a wide range of constraints on the solution of the line finding problem to be incorporated at all stages of the computation.

Finally, Hough space can be big. Even just generating and searching a 512×512 bin Hough space is non-trivial once more sophisticated peak-detection and backmapping algorithms are being used. For analytic shapes or objects (rigid or non-rigid), parameter space might even be much higher dimensional than just two-dimensional. Methods like the Adaptive Hough Transform[12] (AHT) and the Fast Hough Transform[13] (FHT) have tried to address this problem using recursive subdivisions of parameter space. The algorithm presented in this paper is also based on a recursive subdivision of parameter space, but avoids the quantization errors common to those algorithms,[1] and extends the method in several ways, such as the simultaneous accumulation of multiple solutions.

2 Algorithm

2.1 Line Finding

The basic line finding algorithm is given in Figure 1. The algorithm essentially implements a depth-first search of a spatial decomposition of the parameter space by a binary tree (in practice, a best-first algorithm is actually used).

That is, at each step, there is a box (or rectangle) (variable: `box`) in parameter space that is under consideration. Initially, that box consists of the set of all possible parameters. During the execution of the algorithm, the box will be subdivided. Regions or boxes that provably cannot contain a solution under the given error bounds will be eliminated from further consideration.

A representative subdivision of transformation space as it is explored during an actual line finding problem is shown in Figure 7.

Associated with each box is a set of image features that are consistent under the given error bound with any of the lines corresponding to the parameters contained in `box`. It is crucial to realize that consistency with the box is defined here as consistency under the given error bounds as measured in the image. In this, the algorithm differs from other line finding algorithms based on the Hough transform or based on recursive subdivisions of parameter space.

The next ingredient of the algorithm is a function `bound_quality` that, given `box` and the set of image features consistent with `box`, estimates an *upper bound* on the quality of any solution to the line finding problem for all the combinations of line parameters contained in `box` under the given error bound; by "quality" we mean for the purposes of this paper the the total length of segments of the line accounted for by edge pixels in the image under the given error bound.

However, other notions of quality are desirable in some applications. For example, we might want to penalize hypothesized lines that are supported by a large number of fragmented, short stretches of edge pixels, compared with hypothesized line that are composed of a small number of long, connected stretches of edge pixels.

Another notion of quality might weight features differently depending on the amount of their deviation from the hypothesized line. For example, if we assume that pixel deviations from the line are given by some distribution Φ, we might weight the additional support that a feature gives to a hypothesized line by some function $F(d)$, where d is the distance of the feature from the line; such methods are described in the literature.[5,8,6,9]

For using any kind of quality measure with the line finding algorithm described in this paper, all that is necessary is that we can quickly bound the largest possible quality for any hypothesized line described by line parameters contained in `box`.

We can now sketch the operation of the function `search`, the heart of the algorithm. Initially, it is given a rectangular region in parameter space, `box`, and a set of features, `features`. The subset of `features` consistent with `box` under the given error bound is found (`consistent_features`). Now, there are two major cases.

First, the current box does not yet represent an accurate solution to the line finding problem for the set of `consistent_features`. This is determined by the function `is_done` (we will discuss how this is determined in more detail in Section 2.3). In that case, the current box is split into two halves, and the search is repeated for each half of `box` in parameter space.

Of course, a conceptually trivial and quite useful modification of this step is to explore that half of `box` first that has the larger upper bound on the potential solution, a kind of best-first algorithm. To keep the presentation simple, this is not shown in Figure 1.

The second case is that the current `box` and the current set of `consistent_features` represent a possible solution. In that case, the algorithm compares this possible solution against the best solution found so far. If it is better, it is recorded in the variables `best_quality`, `best_box`, and `best_features`. In either case, the algorithm returns in order to allow the exploration of other parts of parameter space.

At the end of this process, the best solution, in the sense of the quality measure, is left in the variables `best_quality`, `best_box`, and `best_features`. Of course, often we are interested in identifying multiple lines in an image, not just in finding a single "optimal" or "maximal" line; how we can go about doing this is discussed in Section 2.4.

2.2 Testing for Consistency

One of the key components of the algorithm is the test of whether an individual feature is "consistent" with a given box of line parameters under the given `error_bounds`.

The exact nature that this test takes depends on the primitive features that we extract from the image. The two kinds of features we are considering in this paper are *point features* and *line segment features*.

Point features correspond to individual edge pixels in the image. Each point feature has a location and an associated orientation.

Line segment features can result, for example, from a polygonal approximation to the output of an edge detector. Line segment features have two end points and an orientation (the orientation may either be the orientation of the line passing through the endpoints of the segment, or it may be measured from the gradient associated with the edge pixels making up the line segment).

To test for consistency of either kind of feature with a hypothesized line under bounded error, we use two primitive tests: a test of whether a point is within a given error bound of some line described by the line parameters contained in box and a spatial error bound, (`point_consistent`), and a test whether the orientation is within a given angular error bound from within the orientation of a range of orientations, (`angle_consistent`).

The implementation of `angle_consistent` is relatively straightforward and will not be discussed here. The implementation of `point_consistent`, however, involves some subtleties.

We parameterize lines by their angle θ with the x-axis and their distance ρ from the origin. A box in transformation space consists of a range of angles $[\theta_0, \theta_1]$ and a range of possible angles $[\rho_0, \rho_1]$.

Figure 2 shows the four lines parameterized by (θ_0, ρ_0), (θ_1, ρ_0), (θ_0, ρ_1), and (θ_1, ρ_1), corresponding to the four corners of the box. These four lines enclose a bow-tie shaped region in image space. Since we want to determine whether a given image point p lies within a distance of ϵ of any one of the lines determined by any pair (θ, ρ) of parameters contained in box, we might at first sight conclude that all we need to determine is whether p is either directly contained in that bow tie shaped region, or whether it is at least located within a distance of ϵ of that region. This test is neither very difficult nor very expensive: we need two dot products to determine whether a point is above or at most a distance ϵ below either line (θ_0, ρ_0) or (θ_1, ρ_0). Likewise, we need two dot products to determine whether a point is below or at most a distance ϵ above either line (θ_0, ρ_1) or line (θ_1, ρ_1).

However, this is not quite accurate. In fact, some points that lie on lines parameterized by parameters (θ, ρ) contained in the box are actually outside this bow-tie shaped region. We therefore need to modify the above procedure slightly. This is illustrated in more detail in Figure 3.

Here, two lines, corresponding to the lower two lines delimiting the bow tie, are shown (marked (θ_0, ρ_0) and (θ_1, ρ_0)). But consider now the line $(\frac{\theta_1+\theta_0}{2}, \rho_0)$. Its parameters are certainly contained within the box of parameters $[\theta_0, \theta_1] \times [\rho_0, \rho_1]$. However, there is a non-negligible segment of that line between points A' and B' that falls outside the bow tie region. Therefore, if we simply used the bow tie region to test for consistency of a feature with the set of lines corresponding to the parameters inside the box, we would run the risk of falsly classifying a point as "inconsistent", even though it actually can be found within the given error bound of some line contained in the box.

There two possible solutions to this problem. First, we could simply carry out the exact test. This would require testing whether a given point is within a distance ϵ of the curved triangle ABC. Such a test is not too difficult to carry out, but the runtime cost is non-negligible. Given that the consistency test is in the inner loop of the algorithm, and, as it turns out, actually dominates the running time of the algorithm as determined by an execution profile, we would prefer a method that affects the running time of the algorithm less.

The second solution is to overestimate the region in which points are accepted as consistent slightly. This does not present a problem as long as the overestimation is sufficiently small in absolute terms and goes to zero quickly as the dimensions of `box` shrink during the execution of the algorithm. This was the approach adopted in the algorithm actually implemented.

There are two simple methods that offer themselves for modifying the constraints in order to ensure that no consistent image point is falsely rejected. We can either translate both lines (θ_0, ρ_0) and (θ_1, ρ_0) closer to the origin by a small amount τ, or we can introduce a third line passing through

AB with line parameters of $(\frac{\theta_1+\theta_0}{2}, \rho - \tau)$ and say that a point is consistent if it is consistent under bound ϵ with either (but not necessarily both) of the two linear constraints for ρ_0 or the line AB, and if it is consistent under bound ϵ with either of the two linear constraints for ρ_1.

Using elementary geometry, the distance τ, which is the same in both cases, can be read from Figure 3 as

$$\tau = \rho\,(1 - \cos\frac{\Delta\theta}{2})$$

where $\Delta\theta$ is the difference between θ_1 and θ_0. It should be noted that τ approaches 0 as the square of $\Delta\theta$.

Now, we can return to the original problem of determining when a point feature (edge pixel) or a line segment is consistent with a given box. We say that a point feature or edge pixel is consistent with a given box if its location satisfies the `point_consistent` test and its orientation satisfies the `angle_consistent` test with the parameters contained in the box.

For a line segment feature, we would like to say that it is consistent with a given box if all the edge pixels that make up the line segment feature satisfy the `point_consistency` test. However, such a test is, again, relatively costly. A simpler approach is to test whether each of the two endpoints satisfies the `point_consistent` test. While this not imply that all the individual edge pixels that make up the line segment feature are contained in the image region swept out by the lines described by parameters in the box under the given error bound, the approximation to the precise test becomes nearly perfect as the current box shrinks. In the limit of $\Delta\theta = 0$, the test is easily seen to be exact. In addition to testing the endpoints, we also test whether the orientation of the line segment feature satisfies the `angle_consistent` test, since very short line segment features would otherwise be nearly unconstrained in their orientation.

2.3 Termination

Before going on, we should discuss the issue of termination, that is, when the function `is_done` in Figure 1 returns `true`. There are, in fact, a number of different criteria we might want to use for stopping.

Ideally, we would like to determine exactly whether the set of edge segments represented by the variable `consistent_features` represents a solution to the line finding problem under the given error bounds. If yes, then we can simply accept this set as a solution and return. While such a computation is possible in principle, in practice it is far too expensive, given that the test for termination is one of the most frequently executed operations in the algorithm.

A much simpler termination condition is to check whether the current box has become "sufficiently" small. The notion of "sufficiently" here requires some explanation.

As we saw above, the test for consistency of a feature with the current box allows for two kinds of uncertainty: the first results from the given `error_bounds`, while the second results from the finite dimensions of the box itself. That is, the current set of `consistent_features` is not necessarily an exact match under the given `error_bounds`, but instead a match under slightly larger error bounds that are determined, for a given image, by the dimensions of the box.

If we terminate the search when the box has become sufficiently small, rather than by verifying consistency exactly of the result with the given error bounds, the line finding algorithm is transformed into a *weak* geometric algorithm.[14] That is, the error bounds satisfied by the maximal solution are uncertain by at most a small bounded amount determined by the maximal dimensions of the input image and the chosen dimensions for the terminal box.

The Hough transform, of course, also suffers from the same problem. The weakness of the Hough transform is related to the size of the individual bins in the quantized parameter space. However, in contrast to the Hough transform, with the present methods, we can easily choose error bounds and the weakness of the solution completely independently. While for the Hough transform, the dimension of the individual bins is related to the size of the accumulator array as the inverse square, which means that making the individual bins significantly smaller increases both the running time and the amount of space required by the Hough transform greatly, the running time and amount of space required by the present algorithm only varies proportionally to the logarithm of the weakness parameter (see below).

2.4 Global Interpretation

In the algorithm shown in Figure 1, only a single maximal solution is found, where "maximal" refers to the line that corresponds to the greatest total length of edge segments in the image compatible with that line under the given error bounds.

Often, we are not interested in just finding a single maximal line, but instead in finding all "reasonable" lines in the image. For the Hough transform, probably the most common approach is to report all those bins in Hough space that form local maxima and that are above some threshold.

Such an approach is somewhat unsatisfactory because it usually results in the reporting of multiple lines that are really only slightly different interpretations of nearly identical sets of edge segments. To alleviate this problem, it is possible to permit the reporting of only a single local maximum within a bounded region in Hough space.

We will use a similar idea below. Before proceeding, however, it is a good idea to reflect upon the real-world constraints that give us the intuition that multiple nearby line hypotheses are unlikely and undesirable in the first place. There are essentially two basic reasons.

First, image acquisition is a band-limited process, and edge detection itself usually involve some kind of convolution operation. This, however, limits the density of parallel lines that can be resolved, and postulating that two lines that are closer to one another than this limit are present simultaneously in the image is not sensible, given that there is no way we could support such a conclusion from the input data to the line finding algorithm. This suggests that if there are two very similar line hypotheses, we should choose only one.

Furthermore, in many applications we can use the assumption of a "general viewpoint", that is, that the image was taken with very high probability from a position such that different lines do not coincide. This means that we should not allow two different line hypotheses to share any edge pixels.

Incorporating these additional constraint then suggests the following approach to finding a global interpretation of the lines in the image. We start by running the line finding algorithm to find the maximal solution given all edge pixels in the image. We then remember this solution and remove the corresponding edge pixels from the image (they would not be allowed to participate in the match of any other line). We then re-apply the line finding algorithm to the remaining edge pixels and repeat this process until we have explained all the edge pixels in the image. Restarting the algorithm multiple times seems somewhat costly, however (and that suspicion is born out in practice, being nearly 10 times slower than the alternative approaches described below).

We might reduce this cost if we do the we somehow run it in a way to find a representation of the set all possible solutions, and then enforce the constraint of unique correspondences in a second step.

If we discretize parameter space sufficiently coarsely (similar to a Hough transform) and set a lower threshold on the total length of support for a line that we are interested in, this turns out to be a feasible approach (the FHT algorithm,[13] for example, also returns such a complete representation of Hough space). While the list of all solutions contains many redundancies and duplications, the simple greedy postprocessing algorithm shown in Figure 4 then quickly finds the desired interpretation.

This greedy algorithm works similarly to the sequential interpretation process we described above. That is, from the list of all hypotheses, it picks the best hypothesis. Then, it removes all the features associated with this best hypothesis from the support for all the remaining hypotheses and recomputes the quality for each remaining hypothesis. The process is then repeated until either no hypotheses remain, or until the quality of the remaining best hypothesis falls below some threshold.

But ideally, we would like to avoid generating a complete list of hypotheses. In particular, if we choose as our termination criterion simply the dimensions of the box in parameter space, the number of hypotheses generated in this way can be seen to grow as the square of the dimensions of the box at a leaf. Clearly, this is not very desirable, and we would like to be able to choose the termination criterion, which determines the accuracy or weakness of the solution, without paying such a high cost.

A solution to this dilemma is to accumulate solutions for small regions of transformation space. That is, we replace the variables `best_quality`, `best_box`, and `best_features` themselves with arrays corresponding to quantized versions of parameter space. We can choose the quantization of those arrays to be significantly coarser than the dimensions of the terminal box in the search algorithm. The effect of this is that locally sub-maximal solutions near (in parameter space) a locally maximal solution tend to be suppressed. By choosing the quantization of the arrays holding the locally optimal solutions suitably, we can make certain that sub-maximal solutions only are suppressed if they share a significant number of features with the nearby maximal solution.

Because of the quantization of the arrays holding the locally maximal solutions, this approach does not guarantee, however, that solutions in different bins do not share features. Therefore, even in this approach, we still need to run the greedy algorithm shown in Figure 4 to make sure that all the line hypotheses found by the line finder are supported by disjoint sets of features in the image.

3 Results

The algorithm as described above has been implemented in CMU CommonLisp[15] on a SparcStation 2. The input to the algorithm was obtained by using a Canny-Deriche edge detector[16,17] implemented in ANSI C.

The algorithm is currently being used in the development of a vision system for an industrial inspection task. However, for the following discussion, we will use the example image in Figure 5, a 566 by 544 pixel image of 5 BIC razors. For all the experiments described below, the error bounds were set to two pixels.

As we mentioned above, the line finder described in this paper can cope with both point features and with line segment features.

Using point features is perhaps the most straightforward comparison with the Hough transform. The image shown in Figure 5 yields 4533 point features (edge pixels with associated orientation). If we apply the line finder described above directly to these features, finding the solution (essentially the same solution as shown in Figure 6) takes 388 seconds (a little less than 7 minutes). While this is quite slow compared with a simple Hough transform, it should be kept in mind that the algorithm finds solutions under well-defined error bounds, that it ensures a unique interpretation of each edge pixel, and that it is not subject to the aliasing problems of the standard Hough transform.

Fortunately, we have means at our disposal for speeding up the operation of the algorithm significantly. In particular, instead of using point features as input to the algorithm, we can use line segment features.

For the standard Hough transform, there is no significant advantage to grouping edge pixels into line segments before carrying out the Hough transform–each edge pixel is only considered once by the algorithm, and all the "intelligence" for the Hough transform is put into post-processing the accumulator space.

The line finder described in this paper, however, carries out repeated geometric operations involving the input features. It pays therefore to preprocess the input features in such a way as to represent them more compactly and better suited for carrying out these geometric operations.

In order to do this, each connected chain of pixels in the edge image output by the Canny-Deriche edge detector is identified and approximated to within an error bound of one pixel by a polygonal chain using a splitting algorithm.

This step greatly reduces the number of features that need to be considered by the line finding algorithm. Instead of 4533 point features, it can now operate on 111 line segment features. The time required for the execution of the line finding algorithm is reduced from 388 seconds to 9 seconds. The subdivision of transformation space explored during this line finding problem is shown in Figure 7.

In using this grouping step, we have to ask ourselves, however, whether it affects the accuracy or robustness of the line finding algorithm significantly.

With regards to accuracy, a point on the line segment is at most one pixel away from the location of the corresponding edge pixel, and this amount could be reduced as much as desired using sub-pixel accuracy edge detection and approximation.

With regards to robustness, we have to ask ourselves whether the grouping of edge pixels into line segments prior to the line finding algorithm perhaps precludes some important line hypotheses from being found. But line segments are only extracted for connected chains of pixels and are (by necessity) broken at points of high curvature. Therefore only pixels that naturally form part of the same line hypothesis are grouped together, and the subset structure imposed on the set of all edge pixels by the grouping step is still completely compatible with all "reasonable" line hypotheses.

The next question that is important to ask is how the running time of the line finding algorithm is related to the number of input features. From benchmarks and the analysis of a related algorithm,[18] we expect a nearly linear dependence of the running time on the number of input features. To see whether this is true of the line finding algorithm as well, the line finding algorithm was applied to randomly generated test images.

Each of the test images consisted of between 20 and 380 randomly placed line segments that were each 30 pixels long. In addition, each image contained 5 randomly placed long lines. Each of those lines was visible as 5 line segments in the image with a total length of 170 pixels. The line finding algorithm was required to find any line that was supported by at least 150 pixels in the image. An example of one of these images is shown in Figure 8.

The results of these simulations are shown in Figure 9. Each "$\times$" symbol represents the average running time of 100 trials. We find a nearly linear relationship between the number of edge pixels (or, equivalently, line segments) in the image and the running time of the algorithm.

Another interesting question to ask is how the running time of the algorithm depends on the termination condition. For the experiments above, we chose to terminate the exploration of a solution as soon as the box in transformation space had dimensions smaller than 1 pixel in the ρ dimension and $0.29°$ in the θ dimension. For the present example, this corresponds to a Hough space of about 800 by 630 pixels.

As we saw above, this adds some additional uncertainty ("weakness") to the error bounds, and for certain applications, we may prefer more exact solutions. Figure 10 shows the dependence of the running time of the algorithm for different choices of the dimensions of terminal box when applied to the image in Figure 5. The horizontal axis (on a logarithmic scale) shows the size of the terminal box, with a scale of 1 corresponding to a terminal box of dimensions 1 pixel by $0.29°$. (To compensate for variability due to garbage collection times and operating system overhead, each data point is the average of five runs on the same data.)

As we can see, the running time of the algorithm is approximately logarithmic in the inverse of the dimensions of the terminal box (a similar relationship holds for the amount of space required). This is similar to the adaptive or multiresolution Hough transforms, but is in significant contrast to the standard Hough transform, for which the running time and space requirements are quadratic in the inverse of the dimensions of each Hough bin.

4 Discussion

This paper has described an efficient algorithm for finding lines with well-defined geometric and combinatorial properties. In particular, lines found by the algorithm satisfy a bounded error criterion, and it is guaranteed that each feature is counted towards only a single line hypothesis.

In applications, we have found that picking parameters for the algorithm is simple and intuitive. The only parameters that are critical are the error bounds on location and orientation of features relative to a hypothesized line and the minimum total length of edge pixels by which a hypothesized line must be supported in the image; these parameters depend, of course, on the application.

The only other parameters that need to be picked are the weakness parameters (the dimensions of the box or rectangle in parameter space at which we terminate the search), and the size of the bins for the local maximization of results. Because the running time of the algorithm depends only linearly on the magnitude of the logarithm of the weakness parameters, we can pick them conservatively in applications that require that the specified error bounds be satisfied accurately. The choice of the size of the bins for the local maximization of line hypotheses depends on properties of the edge detector and on the particular applications, but in most applications, we do not require the detection of very closely spaced parallel lines, and bins that are of the order of magnitude of 8 pixels in the ρ dimension and $11°$ in the θ dimension have proven sufficient for several applications.

Given the vast amount of research on the subject, it is not surprising that the line finder described in this paper has close relations to a number of other approaches.

Foremost, the algorithm is similar to an adaptive or dynamically quantized version of the Hough transform.[13,12,19–22] It is also somewhat reminiscent of an exploration of Hough transform space using the converging squares algorithm.[23] Like those methods, it begins with a coarse subdivision of parameter space and refines it in regions that look "promising", in the sense of possibly containing good line hypotheses. However, the present method differs from those other methods in its error model.

Other methods compute the set of all possible transformations that would be compatible with a given edge pixel in image space without explicitly taking into consideration the amount of error that may be present on the location of that edge pixel. Some robustness against errors is then achieved by integrating votes over local regions in parameter space, usually collections of small, non-overlapping rectangles.

From our foregoing geometric analysis in Section 2.2, the problems with such an approach should be clear: a rectangle in parameter space corresponds to a bow-tie region (plus a curved triangle) in image space, something that hardly constitutes a good implementation of any interesting noise model of lines in images. Furthermore, the fact that the accumulator rectangles in parameter space are non-overlapping in many Hough transform based line detection methods means that votes may be split among several rectangles.

The line finder presented in this paper interprets the subdivision of parameter space more carefully. For each rectangle in parameter space, it asks which edge pixels are compatible with any of the lines described by parameters in that rectangle under the given error bounds. In the standard Hough transform view, this would mean that each rectangle in the subdivision is dilated before testing it against the line parameters corresponding to a particular edge pixel.

More importantly, other error models, such as those based on influence functions or probabilistic considerations[8,9,6,5] can easily be used with the current algorithm, in place of the uniform error bounds used in the description and derivation above. Even the direct incorporation of constraints such as connectivity requirements in image space[11] into the current algorithm is easy.

It is also interesting to relate the current algorithm to the optimization view of the Hough transform.[5] Stephens views the problem of line detection as the problem of maximizing the logarithm of the likelihood function of the line parameters given the edge pixels in the image, and he proposes the use of *local optimization algorithms* like gradient ascent to find optimal solutions for line parameters. The scheme described in this paper can be viewed as a simple yet powerful *global optimization algorithm* applied to the maximization of a likelihood function.

Abstractly, in order to maximize a function f on a region D, it divides D into two subregions D_1 and D_2, and computes an upper bound b_i on $\max_{x \in D_i} f(x)$. It then explores D_i further only if b_i is greater than the best maximal solution found so far; if it is not, then D_i can safely be excluded from further exploration.

An approach for finding lines that is completely different from the Hough transform is based on search.[24–26] In that approach, pairs of lines are grouped together if they satisfy certain tests of colinearity and/or proximity. Such an approach can also be regarded as related to the line finder described in this paper. Like the line finder described here, such methods are often based on more complex features than individual edge pixels–for example, line segments. However, the search strategy itself is entirely different, since search based methods are organized around extending collections of features, rather than around regions in parameter space. This dichotomy is similar to the dichotomy between search-based approaches to object recognition and parameter space based approaches for object recognition such as the Hough transform; the main disadvantage of search based approaches is that they tend to have exponential time complexity unless they incorporate heuristic pruning methods.[27]

In summary, this paper has presented an efficient algorithm for finding lines under bounded error, useful for many practical applications. In addition, the key idea of organizing the search around adaptive subdivisions of parameter space while at the same time measuring errors in image space should prove fruitful for a much larger class of problems, including efficient computation of the probabilistic Hough transform and general object recognition. Some tentative steps in that direction have already been undertaken.[18]

References

[1] J. Illingworth and J. Kittler. A Survey of the Hough Transform. *Computer Vision, Graphics and Image Processing*, 44:87–116, 1988.

[2] V. F. Leavers. Which Hough Transform? In *IEE Colloquium on Hough Transforms, London*, Savoy Place, London WC2R 0BL, U.K., May 1993. IEE.

[3] T. Risse. Hough Transform for Line Recognition: Complexity of Evidence Accumulation and Cluster Detection. *Computer Vision, Graphics and Image Processing*, 46:327–345, 1989.

[4] V. F. Leavers and J. F. Boyce. The Radon Transform and its Application to Shape Parameterization in Machine Vision. *Image and Vision Computing*, 5(2):161–166, 1987.

[5] R. S. Stephens. Probabilistic Approach to the Hough Transform. *Image and Vision Computing*, 9(1):66–71, February 1991.

[6] N. K. Kiryati and A. M. Bruckstein. Antialiasing the Hough Transform. *Computer Vision, Graphics and Image Processing*, 53(3):213–222, May 1991.

[7] Imants D. Svalbe. Natural Representations for Straight Lines and the Hough Transform on Discrete Arrays. *IEEE T-PAMI*, 11(9):941–950, 1989.

[8] Isaac Weiss. Line Fitting in a Noisy Image. *IEEE T-PAMI*, 11(3):325–329, 1989.

[9] J. Princen, J. Illingworth, and J. Kittler. Hypothesis Testing: A Framework for Analyzing and Optimizing the Hough Transform Performance. In *Proc. 3rd Int. Conf On Computer Vision, Osaka, Japan*, pages 427–434, 1990.

[10] G. Gerig. Linking image-space and accumulator-space: a new approach for object recognition. In *First International Conference on Computer Vision, (London, England, June 8–11, 1987)*, pages 112–117, 1987.

[11] S. Y. K. Yuen. Connective Hough Transform. In *British Machine Vision Conference, Glasgow*, pages 127–135, 1991.

[12] J. Illingworth and J. Kittler. The Adaptive Hough Transform. *IEEE PAMI*, 9(5):690–698, 1987.

[13] H. Li, M. A. Lavin, and R. J LeMaster. Fast hough transform–a hierarchical approach. *Computer Vision, Graphics, and Image Processing*, 36(2–3):139–161, 1986.

[14] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer Verlag, Heidelberg, 1988.

[15] Robert A. MacLachlan (editor). CMU Common Lisp User's Manual (Version 16f). FTP: `lisp-rt1.slisp.cs.cmu.edu`, `/afs/cs.cmu.edu-/project/clisp/release`.

[16] John F. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[17] Rachid Deriche. Using Canny's Criteria to Derive a Recursively Implemented Optimal Edge Detector. *International Journal of Computer Vision*, 1:167–187, 1987.

[18] Thomas M. Breuel. Fast Recognition using Adaptive Subdivisions of Transformation Space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 445–451, 1992.

[19] Kenneth R. Sloan. Dynamically Quantized Pyramids. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 734–736, 1981.

[20] Joseph O'Rourke. Dynamically Quantized Spaces for Focusing the Hough Transform. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 737–739, 1981.

[21] S. C. Jeng and W. H. Tsai. Fast generalized Hough transform. *Pattern Recognition Letters*, 11(11):725–733, 1990.

[22] M. Atiquzzaman. Multiresolution Hough Transform–An Efficient Method of Detecting Patterns in Images. *IEEE PAMI*, 14(11):1090–1095, November 1992.

[23] L. O'Gorman and A. C. Sanderson. The converging squares algorithm: An efficient method for locating peaks in multidimensions. *IEEE T-PAMI*, 6:280–288, 1984.

[24] R. Weiss and M. Boldt. Geometric Grouping applied to Straight Lines. In *Proc. IEEE Conf. Computer Vision And Pattern Recognition*, pages 489–495, 1981.

[25] David G. Lowe. *Perceptual Organization and Visual Recognition*. Kluwer Academic Publishers, Boston, 1986.

[26] A. Etemadi, J-P. Schmidt, G. Matas, J. Illingworth, and J. Kittler. Log-level Grouping of Straight Line Segments. FEX and LPEG Software Distribution (`a.etemadi@ee.surrey.ac.uk`), 1991.

[27] Eric Grimson. *Object Recognition by Computer*. MIT Press, Cambridge, MA, 1990.

```
global functions: bound_quality, evaluate_quality global
parameters: rmax, error_bounds global variables:
best_quality, best_box, best_features

procedure rast_lines(features)
begin
   best_quality := 0
   best_box := none
   best_features := none
   box := ([0,2π],[0,rmax])
   search(box,features)
end procedure

procedure search(box,features)
begin
   consistent_features := select features consistent with box
                             under the given error_bounds
   if not is_done(box,consistent_features) then
      bound_on_quality := bound_quality(consistent_features)
      if bound_on_quality ¿ best_quality then
         split box into box1 and box2
         search(box1,consistent_features)
         search(box2,consistent_features)
      end if
   else
      quality := evaluate_quality(consistent_features)
      if quality ¿ best_quality then
         best_quality := quality
         best_box := box
         best_features := consistent_features
      end if
      return from search
   end if
end procedure
```

Figure 1: The basic line finding algorithm. In practice, the algorithm is implemented as a best-first search, and there are a number of small modifications described in the text.

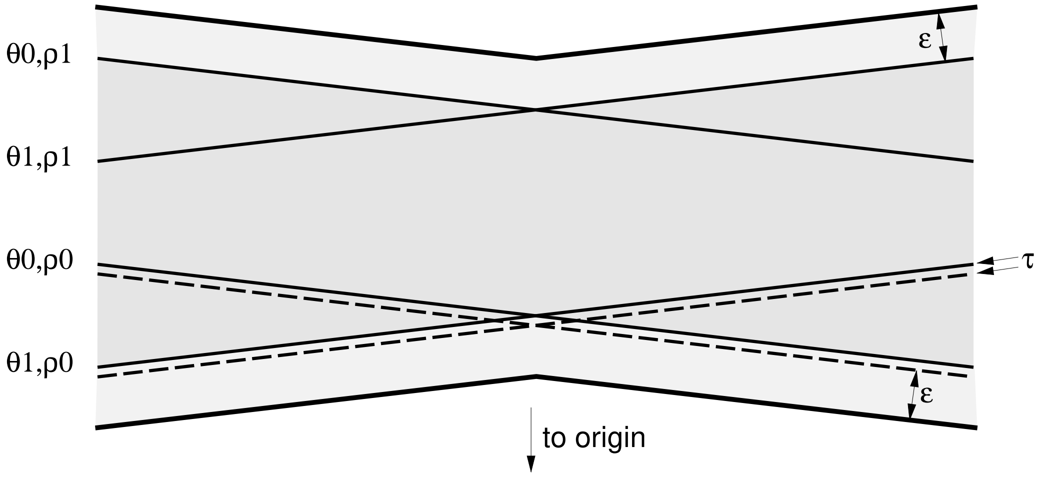

Figure 2: The geometry of consistency between a point and a `box` of line parameters $[\theta_0, \theta_1]$, $[\rho_0, \rho_1]$ in parameter space under an error bound of ϵ. See the text for a more detailed explanation.

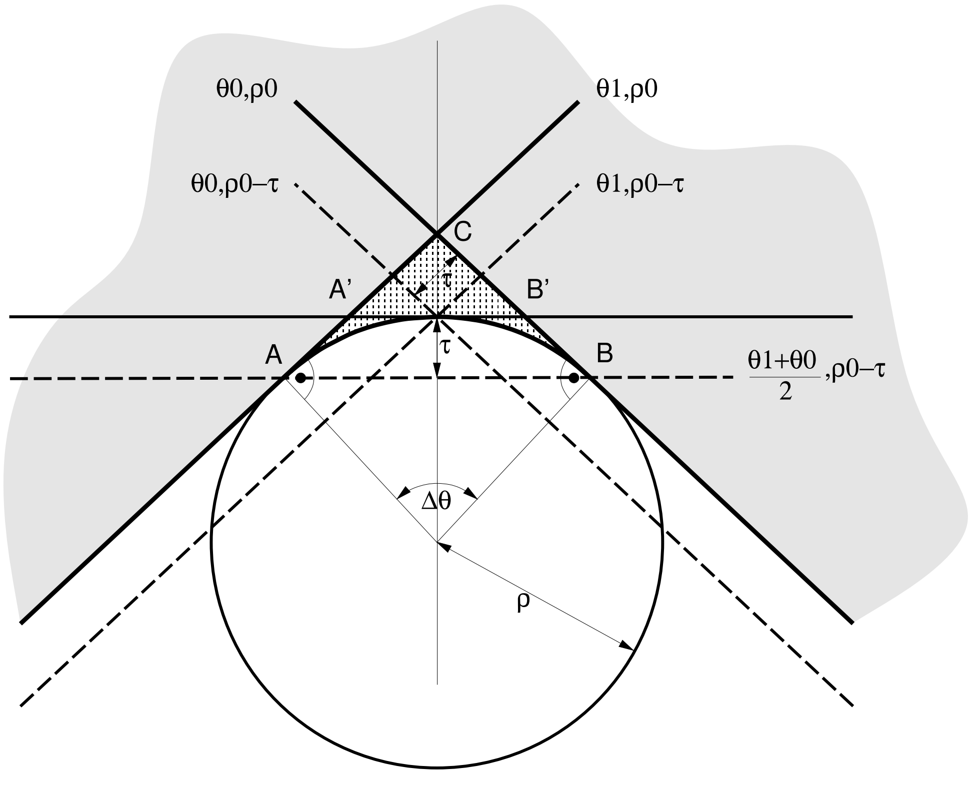

Figure 3: The derivation of the distance τ in the previous diagram.

```
global parameter: minimum_quality

function greedy_post_process(set_of_hypotheses)
Note: each hypothesis in set_of_hypotheses consists of a pair
of line parameters (θ, ρ) and a set of edge segments that
are matched by that hypothesis; the set of features associated with
each hypothesis is destructively modified during the execution of
the algorithm.
begin
   if set_of_hypotheses = the empty set then
      return the empty set
   end if
   best := the hypothesis in set_of_hypotheses whose features
       have the best total quality
   if quality(best) ¡ minimum_quality then
      return the empty set
   else
      remaining := set_of_hypotheses − {best}
      remove the features matched by the hypothesis best from
         each hypothesis in remaining
      processed := greedy_post_process(remaining_hypotheses)
      return {best} ∪ processed
   end if
end function
```

Figure 4: The greedy algorithm used for post-processing a list of hypotheses to ensure that the set of features matched by any two hypothesized lines are disjoint from one another.

Figure 5: The input image (a collection of 5 BIC razors) used for Figure 6. Applying the Canny-Deriche edge detector to this image yields 4553 edge pixels that can be grouped into 111 line segment features within an error bound of one pixel.

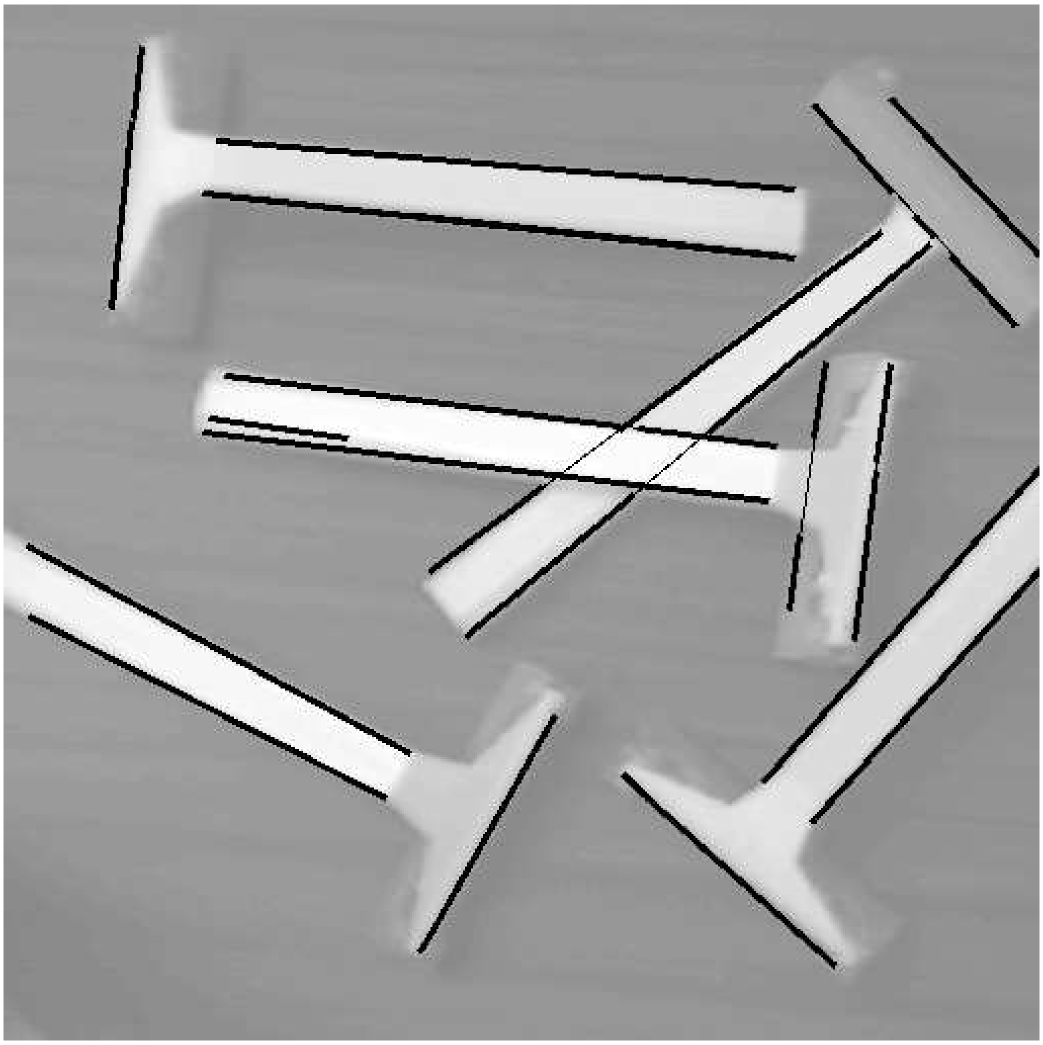

Figure 6: The features found by the method described in the text. Detection of candidate lines took 9 seconds using the algorithm described in the text, and post-processing to obtain a unique interpretation for each edge segment took 0.1 seconds on a SparcStation 2 in CommonLisp. Error bounds were set to 2 pixels, and the minimum required total length for the edge segments corresponding to a line was set to be 60 pixels, a choice which selected specifically the handles and heads of the razors.

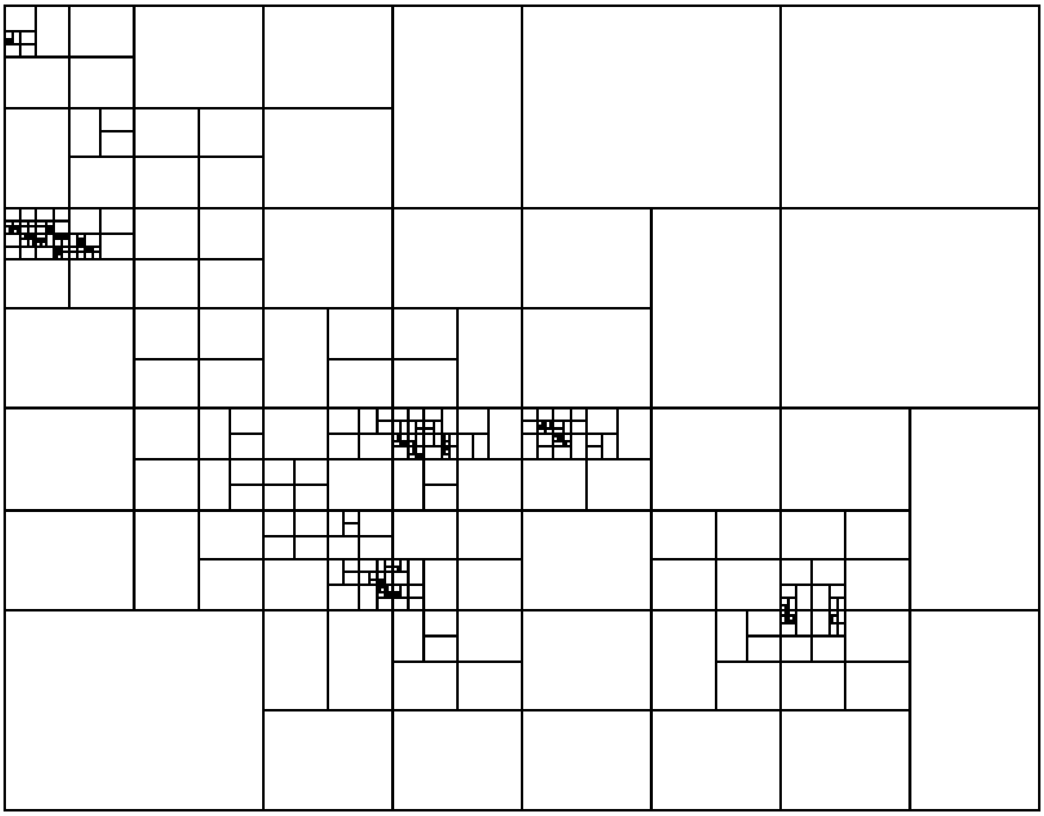

Figure 7: The subdivision of parameter space explored during the detection of the features shown in Figure 6.

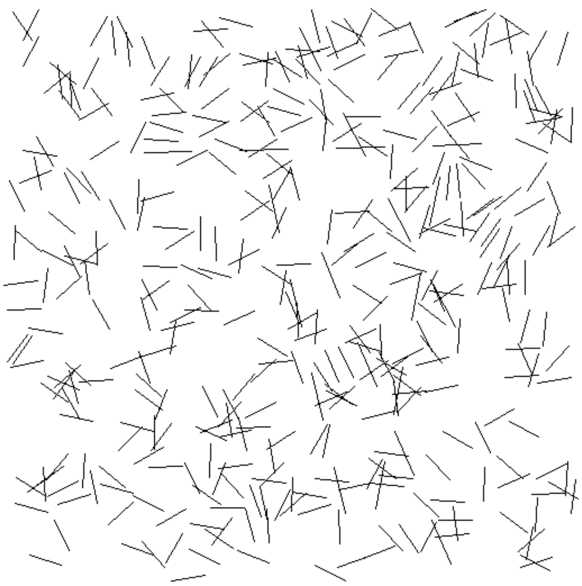

Figure 8: A representative example of a simulated image used for the benchmarks for 380 randomly placed background segments.

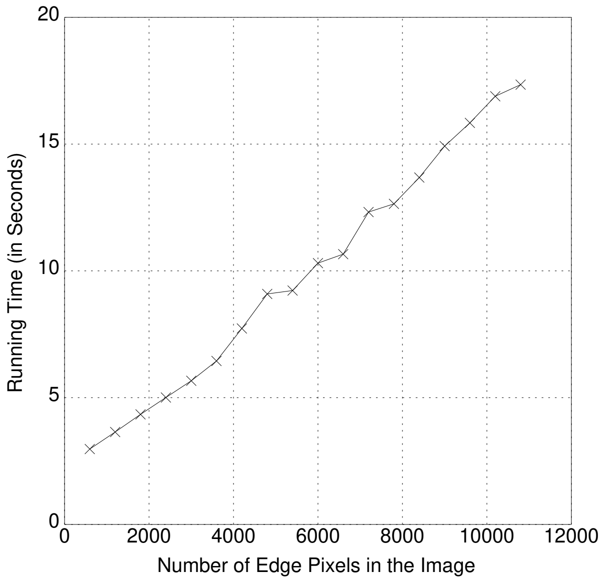

Figure 9: The running time of the algorithm on simulated images. Each image contained 5 different groups of 4 colinear edge segments of a total length of 170 pixels plus between 20 and 380 randomly placed segments of 30 pixels each.

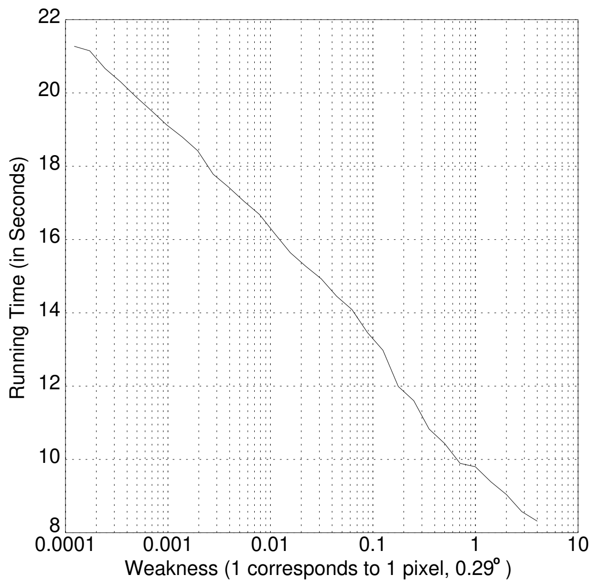

Figure 10: The running time of the algorithm for different choices of the size of the terminal box ("weakness"). The image used in this benchmark was the same as in Figure 5.