A line linker based on Frequentist Statistics tools: Splitting lines with Chi-square tests and merging lines with the Akaike Information Criterion

Fabio Cozman Eric Krotkov
Robotics Institute, Carnegie Mellon University, Pittsburgh

CMU-RI-TR 96-23

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

December 10, 1996

Abstract

This report describes a line linker based on tools of Frequentist Statistics: Chi-square significance testing and the Akaike Information Criterion. The line linker has been implemented as an AVS^{TM} module and is available as freeware. This report describes the basic facts about the distributed linker and its use.

This research is supported in part by NASA under Grant NAGW-1175. Fabio Cozman was supported under a scholarship from CNPq, Brazil.

1 Introduction

This report describes a line linker based on tools of Frequentist Statistics: Chi-square significance testing and the Akaike Information Criterion. *Line linking* starts with an image where pixels belong to two categories (background/edgels), and produces a description of the image in terms of lines.

The line linker has been implemented as an AVS^{TM} module and is available as freeware. The distribution also contains a Canny operator and thinning algorithms that are necessary for effective use of the linker. This report contains the basic facts about the linker and its use. The line linking activity is performed in two steps:

- The first step groups chains of edgels and fits lines to these chains. Lines are generated by least-squares regression. Residuals are monitored through Chi-square testing; lines are broken based on the result of such tests.
- The second step refines and completes the first step by merging (if possible) adjacent lines and by investigating whether lines continue through small gaps. The line merging step is handled through the Akaike Information Criterion, an estimator for the Kullback-Leibler information of the estimated lines.

2 A Brief Review

Line linking is a well-known problem in computer vision. A classic tool for line linking is the Hough Transform, which returns a list of line segments without connectivity information [2]. A different approach uses algorithms that search of lines through a maze of edgels, recovering connectivity information in the process [6, 8]. A complete line linker must also make decisions at intersections and at gaps: should neighboring lines be merged? For example, Pavlidis and Horowitz's algorithm [7] contains splitting and merging steps in line linking that address all these problems.

Line linking can also be approached as a problem of statistical estimation. For example, the Markov field image representation of Geman and Geman leads to an iterative solution for line linking using Gibbs sampling [5]. The computational cost of Markov field models may be prohibitively high. A different idea is championed by Cox, Rehg and Hingorani [4]. In their model, all lines in the image can be modeled by linear dynamical systems subject to random disturbances. They look for best lines by applying Bayes' rule to a number of line candidates. Since the number of candidates may grow exponentially, they use heuristics to terminate some candidates. The lines are then tracked by Kalman filters.

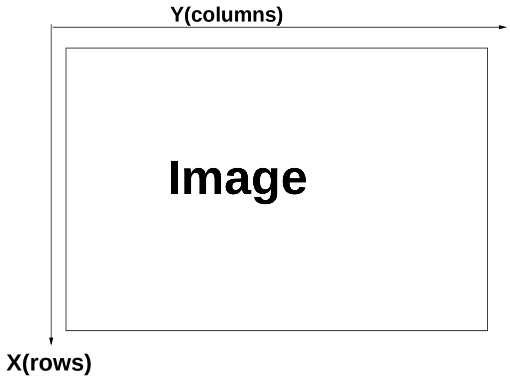

Figure 1: Coordinate system

We use a statistical framework, but we avoid the global distributions used previously. The new idea behind our algorithm is to divide the line linking problem into its smaller components (linking, mergins, splitting) and provide a statistical solution for each of them. Tools from Frequentist Statistics are used: linear regression is used to create lines, Chi-square tests are used to split lines, and the Akaike Information Criterion is used to merge lines. The procedure emphasizes local search for efficiency and uses these statistical techniques to enforce more global constraints.

3 A Simple Model for Line Linking

The lines have a parametric representation: for each line we use a dummy variable t and we define the line as the set of points (x, y) such that[1]:

$$x = a + bt \qquad\qquad y = c + dt$$

We adopt a coordinate system where the top left pixel is the origin, the x axis is vertical from top to bottom and the y axis is horizontal from left to right (Figure 1).

We assume a simple statistical description of the image, similar to Cox, Rehg and Hingorani's model: edgels are normally distributed around the "true" lines. In this model, each line is a pair of Gaussian linear processes with variance σ^2.

With this model, we can employ a least-square technique for estimating lines. Conveniently, this estimator is recursive (similar to the Kalman filter) and we present equations for recursive

[1] The algorithm could be extended to other polynomial curves; circles could be approximated by $x = a + bt$, $y = c + dt + et^2$.

calculation of residuals. The residuals can be analyzed through statistical tests to check whether a line should be continued or broken. The Gaussian assumption leads to Chi-square significance tests.

We can now analyze the possibility of merging adjacent lines and the possibility of jumping through gaps between close lines. We approach these questions using a statistical measure of fitness for models, the Akaike Information Criterion. The implementation of the algorithm divides the least-squares estimation step and the merging step, so that the user can simply perform the former or can perform both.

4 Description of the Line Linker

The line linker assumes that a thinned image containing only edgels is given. For example, the image may be the result of a Canny operator followed by a thinning operation. The implementation expects the "background" of the image to correspond to zero-valued pixels, and the "edgels" to correspond to non-zero-valued pixels. We assume the image may contain gaps between edgels, but no gap is larger than a fixed number β. Two parameters define our model: β and σ^2.

4.1 First Pass: Connecting Neighbor Edgels

This step takes an image and segments it in chains, fitting straight lines on-the-fly. By definition, an edgel is part of a chain if it is connected only to two neighbors. Edgels that have more than two neighbors are taken to be intersections of chains (that is why the image is assumed to be thinned). A region in the image composed of pixels that have more than two neighbors is called a *connection*. Every time an edgel is found to be in a connection, the connection is grown in order to encompass all edgels that belong to it. The chief reason for the distinction between chains and connections is that a pixel in a chain can be unambiguously followed, while a pixel in a connection cannot. The basic algorithm is depicted in figure 2. As a chain is followed, two activities take place: lines are fit to the edgels, and lines are broken based on Chi-square tests.

4.1.1 Fitting a straight line to a chain

In order to fit a line to a chain, we need to estimate four parameters: a, b, c and d. The least-squares estimator for the four parameters minimizes [3]:

$$J = \sum_{i=1}^{n} \frac{(x_i - a - bt_i)^2}{\sigma^2} + \frac{(y_i - c - ct_i)^2}{\sigma^2}, \qquad (1)$$

The algorithm visits every pixel once, in raster order. When it finds a pixel that is unlabelled (not yet classified as a chain-pixel or as a connection-pixel):

- If the pixel has more than two neighbors that are unlabelled or connections, the pixel itself is a connection (a connection must be created and grown).
- If the pixel has no unlabelled neighbor, then:
  - If the pixel has no neighbor connection, it is a chain of one pixel.
  - If the pixel has one or two neighbor connections, it is a chain attached to the connections.
- If the pixel has one unlabelled neighbor, then:
  - If the pixel has no neighbor connection, it starts a chain (a chain must be created and followed).
  - If the pixel has one neighbor connection, it starts a chain attached to that connection (a chain must be created and followed).
- If the chain has two unlabelled neighbors, then it is in the middle of a chain (a chain must be created and followed in two directions).

Figure 2: Algorithm for First Pass

Since x and y are independent of each other, the minimization of J is equivalent to the minimization of J_x and J_y as below:

$$J_x = \sum_{i=1}^{n} \frac{(x_i - a - bt_i)^2}{\sigma^2} \qquad\qquad J_y = \sum_{i=1}^{n} \frac{(y_i - c - dt_i)^2}{\sigma^2}$$

This minimization can be solved recursively for t larger than 2, so we can build the estimates for the lines as the chains are followed.

For completeness, we reproduce the least-squares recursive equations, with the obvious modifications required by our notation [3]. We have, at pixel t_{k+1}:

$$\begin{array}{ll} \tilde{a}_{k+1} = \tilde{a}_k + A_{k+1} & \qquad \tilde{b}_{k+1} = \tilde{b}_k + B_{k+1} \\ \tilde{c}_{k+1} = \tilde{c}_k + C_{k+1} & \qquad \tilde{d}_{k+1} = \tilde{d}_k + D_{k+1} \end{array}$$

where ($s_i = \sum_{j=1}^{k} t_j^i$):

$$\begin{aligned} A_{k+1} &= \frac{s_2 - s_1 t_{k+1}}{s_o s_2 - s_1^2}(x_{k+1} - \tilde{a}_k - \tilde{b}_k t_{k+1}) \\ B_{k+1} &= \frac{-s_1 + s_o t_{k+1}}{s_o s_2 - s_1^2}(x_{k+1} - \tilde{a}_k - \tilde{b}_k t_{k+1}) \\ C_{k+1} &= \frac{s_2 - s_1 t_{k+1}}{s_o s_2 - s_1^2}(y_{k+1} - \tilde{c}_k - \tilde{d}_k t_{k+1}) \\ D_{k+1} &= \frac{-s_1 + s_o t_{k+1}}{s_o s_2 - s_1^2}(y_{k+1} - \tilde{c}_k - \tilde{d}_k t_{k+1}) \end{aligned}$$

4.1.2 Breaking a line

As each edgel is incorporated in the estimates $\tilde{a}$, $\tilde{b}$, $\tilde{c}$ and $\tilde{d}$ of a line, the fit between the estimated line and the data is checked. A significance test is applied to verify the goodness of fit. The test is based on the following fact [3]:

- If the chain under consideration is actually a straight line, the quantities $\tilde{J}_{x,k}$ and $\tilde{J}_{y,k}$ ($k > 2$) are distributed as χ^2_{k-2} random variables.

$$\begin{aligned} \tilde{J}_{x,k} &= \sum_{i=1}^{k} \frac{(x_i - \tilde{a} - \tilde{b}t_i)^2}{\sigma^2} \\ \tilde{J}_{y,k} &= \sum_{i=1}^{k} \frac{(y_i - \tilde{c} - \tilde{d}t_i)^2}{\sigma^2} \end{aligned}$$

When an edgel is incorporated in a line estimate, $\tilde{J}_{x,k}$ and $\tilde{J}_{y,k}$ are evaluated and compared with the critical value given by $\chi^2_{k,\alpha}$. Values of $\tilde{J}_{x,k}$ or $\tilde{J}_{x,k}$ larger than the critical value cause the line to be broken and a new line starts to be fitted. Notice: since α and critical values are inversely related, it follows *the larger α, the "tighter" the fit is, because the easier it is to break lines.*

The operations described above rely on the availability of the residuals as each edgel is processed. It is possible to obtain the value of $\tilde{J}_x$ recursively:

$$\begin{aligned}
\tilde{J}_{x,k+1} &= \sum_{i=1}^{k+1} \frac{(x_i - \tilde{a}_{k+1} - \tilde{b}_{k+1} t_i)^2}{\sigma^2} \\
&= \sum_{i=1}^{k+1} \frac{(x_i - \tilde{a}_k - \tilde{b}_k t_i)^2}{\sigma^2} - 2\sum_{i=1}^{k+1} \frac{(x_i - \tilde{a}_k - \tilde{b}_k t_i)(A_{k+1} + B_{k+1} t_i)}{\sigma^2} + \sum_{i=1}^{k+1} \frac{(A_{k+1} + B_{k+1} t_i)^2}{\sigma^2} \\
&= \tilde{J}_{x,k} + (\sigma^2)^{-1} \Big((x_{k+1} - \tilde{a}_k - \tilde{b}_k t_{k+1})^2 + (A^2_{k+1} + 2\tilde{a}_k A_{k+1})(k+1) + \\
&\qquad 2(A_{k+1} B_{k+1} + \tilde{b}_k A_{k+1} + \tilde{a}_k B_{k+1}) \sum_{i=1}^{k+1} t_i + 2(-A_{k+1}) \sum_{i=1}^{k+1} x_i + \\
&\qquad 2(-B_{k+1}) \sum_{i=1}^{k+1} t_i x_i + 2(\tilde{b}_k B_{k+1} + B^2_{k+1}) \sum_{i=1}^{k+1} x_i \Big)
\end{aligned}$$

$\tilde{J}_{y,k+1}$ obeys a similar relationship.

After following all chains, two structures are produced:

- A vector containing the description of all chains, which we call **chains**.
- A vector containing the description of all connections, which we call **connections**.

4.2 Second Pass: Analyzing Gaps and Intersections

This step operates on the image resulting from the first pass and on the vectors **chains** and **connections**. The algorithm is depicted in figure 3. The algorithm uses the concepts *associate* and *try to associate* defined below.

Two lines are associated if they must be considered parts of a single line even though there is an intersection or a gap between them (Figure 4). The algorithm *tries to associate* two lines l_1 and l_2 by comparing the Akaike Information Criterion [1] of the lines as separate entities and as a single entity. The AIC is an information-theoretic figure of merit for comparison of goodness of fit when models have different numbers of parameters, just as we have here.

Each chain created in the first pass (and contained in **chains**) is analyzed. For each chain c with length larger than one (see the meaning of *associate* in the text):

- Pick the extremities C_1 and C_2 of the chain c.
- For C_1 and C_2:
  - If the extremity is already *associated* to some other chain, skip.
  - If the extremity is connected to a connection, try to *associate* the chain with all the other chains in connection.
  - If the extremity is not connected, search a region around the extremity (a square region of size 2β), trying to *associate* the chain with a chain lying in this region.

When a chain is *associated* to another chain, the new chain is now analyzed by the same process. The algorithm follows a sequence of *associated* chains as far as it can.

Figure 3: Algorithm for Second Pass

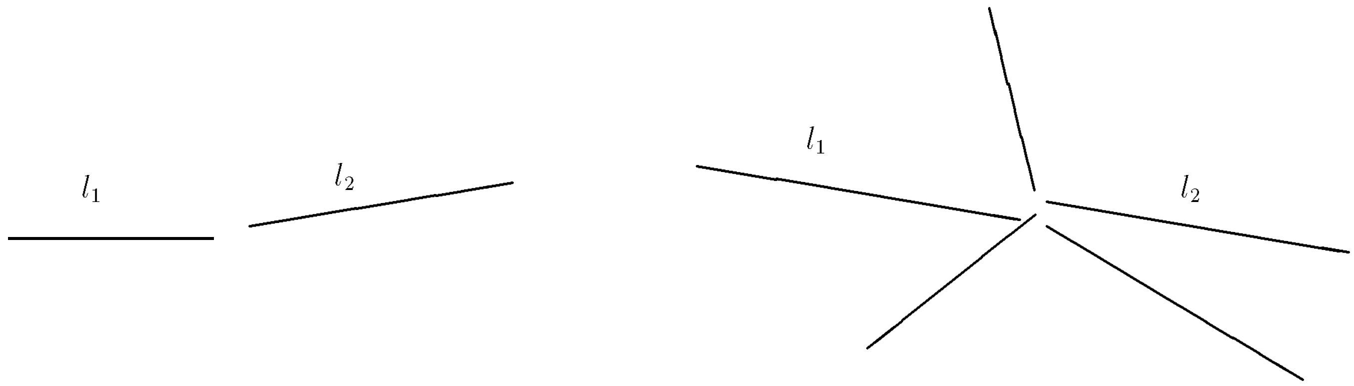

Figure 4: Two lines l_1 and l_2 are associated

Call n_1 the number of edgels fitted by l_1 and n_2 the number of edgels fitted by l_2. A new line is fitted, l_3, using the same least-squares estimator as before, to all points of l_1 and l_2.

This process gives us six quantities:

- $\tilde{J}_{x,n_1}, \tilde{J}_{y,n_1}$: Residuals for the first line.
- $\tilde{J}_{x,n_2}, \tilde{J}_{y,n_2}$: Residuals for the second line.
- $\tilde{J}_{x,n_3}, \tilde{J}_{y,n_3}$: Residuals for the line fitted to the points of the first and second lines.

The Akaike Information Criterion is defined as:

$$AIC = -2(\text{likelihood of data under model}) + 2(\text{number of parameters estimated in the model})$$

In our case we have:

- Two lines: $AIC_{12} = \tilde{J}_{x,n_1} + \tilde{J}_{y,n_1} + \tilde{J}_{x,n_2} + \tilde{J}_{y,n_2} + (n_1 + n_2)\ln(2\pi\sigma^2)^2 + 4$.
- One global line: $AIC_3 = \tilde{J}_{x,n_3} + \tilde{J}_{y,n_3} + (n_1 + n_2)\ln(2\pi\sigma^2)^2 + 2$.

If $AIC_3 < AIC_{12}$, then the lines are *associated*.

5 Description of the Implementation

The linker described previously has been distributed in two forms, as an AVS^{TM} module and as a standalone executable. The latter distribution was based on the GIL library, which has been abandoned. The package is still available, but its interaction with GIL calls and functions is no longer supported; this distribution is briefly described in Appendix A.

A possibly more useful and functional distribution is available as AVS^{TM} modules. AVS^{TM} is a system for visualization and rendering of geometric data and images; for AVS^{TM} users, this distribution is convenient because it conceals the implementation details of the linker.

The distribution contains source code, basic documentation and executable modules for Sun and Silicon Graphics workstations. To obtain the distribution:

```
ftp ftp.cs.cmu.edu
cd /afs/cs/project/lri-3/ftp/outgoing
get avs-linker.tar.Z
```

Then use uncompress avs-linker.tar.Z and tar -xf avs-linker.tar to extract all files. The distribution contains two modules, avs-canny and avs-linking, that perform the basic image operations:

- avs-canny: Canny operator for edge detection;
- avs-linking: thinning and line linking operations;

and a module that can be used to display lines from avs-linking:

- draw-lines: draw lines superimposed in an image.

The README file contains minimal basic information and the /Doc/ directory contains AVS^{TM} networks and data to guide you in using the modules.

The only inconvenience of the AVS^{TM} distribution is that there is no way to make the linker module output the produced lines in some AVS^{TM} data format (AVS^{TM} places strict restrictions on the kinds of data that can be transported from one module to another). Currently the module avs-linking writes the lines and connections to a file, which can then be read by draw-lines. The user can modify this behavior by editing the function translate_chains_connections_to_file in avs-linking.c.

6 Examples

We used images produced by the Canny Operator followed by a thinning operation with:

$$\begin{aligned} \sigma^2 &= 0.3 \text{ pixel}^2 \\ \beta &= 2 \text{ pixels} \\ \alpha &= 0.35 \end{aligned}$$

We show two examples:

- The first example displays fibers in muscular tissue. Figure 5 depicts the original image and the lines found in the image after running Canny operator, thinning and the linker here described. Only lines larger than 5 pixels long are selected for display.
- The second example shows a cube over carpet. The image of the cube has been processed with Canny and thinning, and show some clear lines together with surrounding texture. In the linked result, only lines larger than 10 pixels long are selected for display.

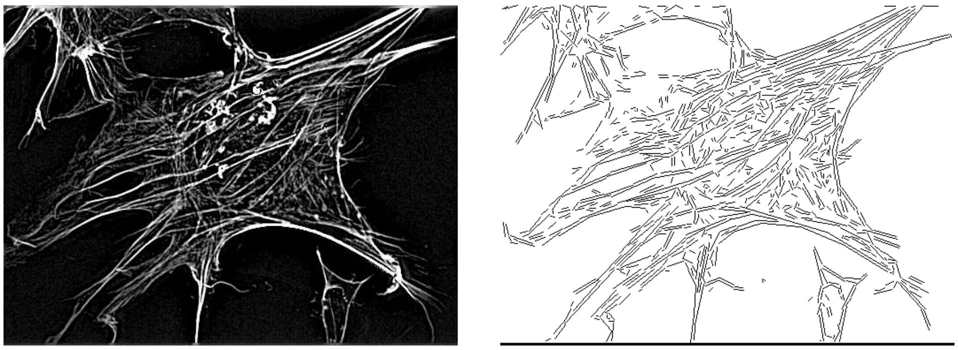

Figure 5: Fiber image

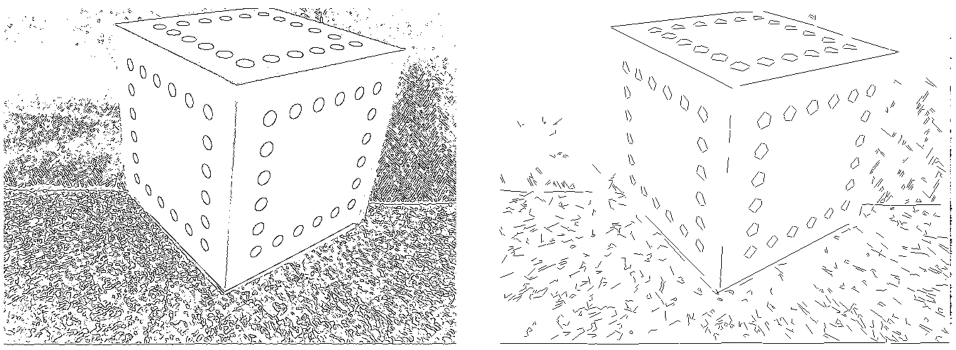

Figure 6: Cube image

Both examples were run on a Silicon Graphics Impact workstation, as a separate module in an AVS^{TM} network; the linker takes approximately 1 second to run, including the time to move images back and forth in the AVS^{TM}.

7 Conclusions

We presented a line linker that uses some basic tools of Frequentist Statistics:

- Least-squares estimation for the best fit in a sequence of edgels.
- Chi-square tests for discovering the best points at which to break chains of edgels.
- Akaike Information Criterion for determining when to merge nearby chains of edgels.

The algorithm requires only three parameters:

- The variance of the noise in the image σ^2.
- The size of the search for the continuation of a chain, β.
- The level of significance for testing the break of a chain, α.

The parameters σ^2 and β can be determined by statistical analysis of series of images. α, on the other hand, is a threshold fixed by the user; intuitively, it fixes the probability that a particular straight line could, by chance, cause a Chi-square test to fail.

The linker is distributed as an AVS^{TM} module. The distribution also contains a Canny operator module, and the linker module contains a thinning algorithm that can be optionally used.

A The gil2lines Distribution

The standalone gil2lines is an older implementation of the linker described in this report. The executable is still available in the VASC computing environment, but its interaction with GIL calls and functions is no longer supported. The source code tries to hide the GIL calls by using typedefs, but some pieces of it have to be adapted for other image libraries. For GIL, you must have access to the GIL run-time libraries. The program gil2lines uses an old implementation of Canny operator that was standard within VASC but that does not seem to compile properly in gcc or recent compilers. To obtain a very simple but possibly useful Canny operator, try this:

```
ftp ftp.cs.cmu.edu
cd /afs/cs/project/lri-3/ftp/outgoing
get general.h
get canny.c
```

In order to use canny.c, take a look at the documentation on top of the file. Again, you need to take a look at general.h: this file defines the operations on images that the rest of the code uses. You must provide functions and types for your preferred image library.

You may find it interesting to take a look at gil2lines or its documentation. In the VASC environment, man gil2lines provides you with a manual page and the World-Wide-Web page http://www.ius.cs.cmu.edu/help/Software/Line/gil2lines.html gives a detailed account of the program.

Here is a simple example of how you can use the program:

gil2lines dummy.gif -only-canny 1 -threshold-canny 10 -dump-edgels 1

(It opens dummy.gif, runs Canny operator, thresholds result at 10 and writes results to default file edgels.gif.)

Here is a more involved example:

gil2lines dummy.gif -only-thin 1 -dump-edgels 1 -verbose 0 -xwin 2 -display-input 1

(It opens dummy.gif, displays dummy.gif in xwin2 and performs thinning. The resulting thinned image is written in edgels.gif. Verbose is switched off.)

The image of the fiber presented in section 6 was created with the command:

gil2lines fiber.gif -xwin 0 -display-lines -significance 6 -threshold-canny 30 -check

and the image of the cube was created with the command:

gil2lines quad.gif -canny 0 -thin -linker -glue 0 -xwin 0 -display-lines -significance 6 -check

The program has some 40 options, which may be a bit daunting for the beginner. But most operations will require very few options; as you try to display images to a xdisptool device or write results to a file, you will need to use some more involved options.

Feel free to get a copy of the C code for Canny operator, thinning and line linking. They may help you at least get some ideas. Try this:

```
ftp ftp.cs.cmu.edu
cd /afs/cs/project/lri-3/ftp/outgoing
get thin-link.tar
```

Then use tar -xf thin-link.tar to obtain a working directory. You need to take a look at general.h: this file defines the operations on images that the rest of the code uses. You must provide the functions fg_i_rstart, fg_i_rend, etc, using your preferred image library. You also need to provide implementations for the types FG_Image, FG_Subimage, etc. The file gil-general.c shows what was done to translate the GIL library calls into the format expected by general.h.

The thinning algorithm is in thin.c and thin.h. The thinning algorithm is very simple and gives good results when the lines in an image are already reasonable thin, for example, what is produced by a Canny operator. An image with thick lines produces bad results, because it is hard to define what the "thinned" image should look like. In general the thinning code works well, but the software is not well documented.

The linker code is introduced in the README file. There is a lot of information at the top of the file linker5.c (which actually contains most of the linker). Take a look at Appendix B for some minimal information about the code.

B Source Code Fundamentals

The linker algorithm is implemented in ANSI C. The first and second pass (fitting lines is the first and merging lines is the second) have been divided in two functions. This makes it easy to use only the first pass if desired. In general. there is no need to merge lines in the second pass if the lines will be used in some search to construct representations of objects or some similar scheme.

- Function linking executes the first pass. Linking receives an input image and two parameters: σ^2 and α. The level of significance α is passed as an integer in the set $\{0, 1, 2, 3, 4, 5, 6\}$; 0 means level 0.01, 1 means level 0.05, 2 means level 0.1, 3 means level 0.15, 4 means level 0.2, 5 means level 0.25, 6 means level 0.35. Linking produces two vectors of data, one containing the description of chains (chains) and another containing the description of connections (connections). Linking also produces two images:
  - An image containing the labels for all pixels: 0 if pixel is background; if not, the number of its chain or connection (chain numbers are positive, connection numbers are negative).
  - An image containing chain codes for all edgels. The chain code for an edgel is a positive integer of 8 bits, where each bit is 0 or 1 depending on the state of one of the eight neighbors of the edgel. If the neighbor is background, the corresponding bit is 0; if the neighbor is an edgel, the corresponding bit is 1. The relationship between neighbor and bit is:

    First bit North neighbor.

    Second bit South neighbor.

    Third bit West neighbor.

    Fourth bit East neighbor.

    Fifth bit North-West neighbor.

    Sixth bit North-East neighbor.

    Seventh bit South-West neighbor.

    Eighty bit South-East neighbor.

- Function glue_chains executes the second pass. glue_chains receives two parameters: σ^2 and β. It also receives the three images processed by linking and the vectors chains and connections.

The function examines all chains in chains and makes modifications in specific flags in the description of chains and connections; the images are not modified. Notice that many lines are already connected by linking; glue_chains attempts to additionally connect lines that are close and similar. In many situations the extra power of glue_chains will not be necessary.

There are also functions for manipulating the chain codes and dealing properly with the data structures (printing, freeing, etc). To obtain a description of the various aspects of the implementation, take a look at the gil2lines distribution described in Appendix A.

References

[1] H. Akaike. A New Look at the Statistical Model Identification, *IEEE Trans. on Aut. Control*, AC-19, n. 6, pp. 716-124, Dec. 1974.

[2] D. H. Ballard and C. M. Brown. *Computer Vision*, Prentice-Hall, Englewood Cliffs N.J., 1982.

[3] Y. Bar-Shalom and T. E. Fortmann. *Tracking and Data Association*, Academic Press, Boston, 1988.

[4] I. J. Cox and J. M. Rehg and S. Hingorani. A Bayesian Multiple-Hypothesis Approach to Edge Grouping and Contour Segmentation, *International Journal of Computer Vision*, 11, n.1, pp. 5-24, 1993.

[5] S. Geman and D. Geman. Stochastic Relaxation, Gibbs Distributions and the Bayesian Restoration of Images, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-6, n. 4, pp. 721-741, Nov. 1984.

[6] D. G. Lowe, Three-dimensional Object Recognition from Single Two-dimensional Images. *Artificial Intelligence*, 31, pp. 355-395, 1987.

[7] T. Pavlidis and S. L. Horowitz. Segmentation of Plane Curves, *IEEE Trans. Comput.*, C-23, pp. 860-870, Aug. 1974.

[8] J. Sklansky and V. Gonzalez. Fast Polygonal Approximation of Digitized Curves, *Pattern Recognition*, 12, pp. 327-331, 1980.