Automated Digital Conversion of Hand-Drawn Plots

Ruo Yu Gu
Department of Electrical Engineering
Stanford University
Palo Alto, U.S.A.
ruoyugu@stanford.edu

Abstract — An algorithm has been developed using Matlab that takes an image of a sketch of one or more functions, plotted on the x- and y- axes, and converts it into an electronic plot, along with the set of equations and parameters that recreate the same plot. This is accomplished by extraction of the plot data, followed by data grouping based on three different heuristic methods (colour, slope and process of elimination), followed by least-squares fitting.

Keywords-graph; plot; sketch; digitization; conversion

I. INTRODUCTION

Quick sketches of plots (i.e. an x-y axis system with one or more functions) often accompany handwritten descriptions and equations, being useful to explicate concepts that are best represented in graphical form.

Digitization of these documents is an oft-occurring problem, and there are currently widely-available algorithms and implementations [1,2] that are able to convert handwritten text and mathematical equations into a digital format. Unlike equations or descriptions, however, the plots cannot be easily digitized; to do so, the plot must either be left as an image or the user must be forced to manually re-create the same plot.

The main disadvantages of leaving the plot as an image are the typical lack of accuracy in hand-drawn plots, and the higher difficulty of modifying digital images. The main disadvantage of leaving re-creation to the user is the time required, not just to input the function but also to set the correct bounds and colors. The user also may not know the exact function and/or its parameters.

This paper describes a method to automatically read a hand-drawn plot and generate a digital version. Two sample images of the types of drawn graphs that this algorithm can act on are included:

![Fig. 1. Left: color plot in second quadrant on white paper. Right: black and white plot in first quadrant on lined paper.](image)

II. PRIOR WORK

A. Academic Work

Academically, most image processing techniques used in the sub-steps of this algorithm (e.g. Hough transform, morphological thinning) have been established in prior papers. No prior work has been found detailing the synthesis of these techniques for conversion of a hand-drawn plot into a digital one, likely because of its narrow scope.

B. Commercial Work

There are a few software packages available whose stated purpose is ‘graph digitization,’ among them [3] and [4]. However, the intended and stated use of these packages is for obtaining data, usually from printed versions of originally computer-generated graphs. For example, one possible input image cited by both [3] and [4] is a photograph of a page in a scientific paper that contains some plotted data. These commercial products can thus be described as ‘data-acquisition’ algorithms, with a number of differences between the ‘graph digitization’ algorithm as described to in this paper, to be described as follows.

Accuracy is often a top concern in data acquisition, whereas for graph digitization accuracy is desired but not key. Correspondingly, data-acquisition is usually demanding of user-attention and does not handle well images with different lighting or graphs of the same color. The output is usually the data itself, and any fitting and plotting must again require user input. In sum, for the original goal cited (to digitize a graph that the user himself has drawn), he would save more time simply by generating the digital plot manually.

Overall, while there is overlap between data acquisition and graph digitization, this is not enough for the former to be particularly useful for the latter. No products have been found to focus on digitization of hand-drawn graphs as described here.

III. DESCRIPTION OF ALGORITHM

The algorithm consists of 5 main steps: (1) pre-processing, (2) plot area identification, (3) graph cleaning and splitting, (4) data labeling, and (5) fitting and plot generation.

The input image is assumed to be either a single-color (pen or pencil) drawing on lined or blank paper, or a colored drawing on blank paper or a whiteboard.
Other inputs to the algorithm are the mathematical functions to be fitted, four numerical bounds of the axes and whether the graph is on lined paper, out of which only the functions are mandatory inputs. Empty bounds will result in equations fitted to normalized data. The number of functions input provides the total number of graphs expected, N.

Throughout section III, a test image (displayed in Fig. 2) shall be shown at each step of the algorithm for demonstrative purposes.

![Fig. 2. Reference image 4, original image.](image)

### A. Pre-processing

If the plot was drawn on lined paper (blue horizontal lines, red margin line), the image is converted into the YCbCr domain and the Cb and Cr domains are used to remove the red and blue lines from the Y domain. This step is extremely fast, but the algorithm suffers when used on color graphs drawn on lined paper.

An alternative method using the Hough transform to detect the lines on lined paper is possible, but was found to have the disadvantages of lower reliability and increased processing time (more than doubling the total for the entire algorithm; see experimental results, section IV). Colored graphs drawn on lined paper was deemed to be a sufficiently unlikely case that the disadvantages outweighed the advantages and this alternative was not used.

In the next step, a black and white (BW) image is obtained using adaptive thresholding (32 by 32 pixel squares with 50% overlap) and small region removal is performed.

![Fig. 3: Image after adaptive thresholding](image)

### B. Plot Area Identification

A Hough transform is performed on the black and white image. With this, a pair of points close to 90 degrees from each other and of sufficient intensity in position/angle space is obtained; these correspond to the axes. Unfortunately, this step can be fooled by the presence of two other straight lines 90 degrees apart that are not the axes.

A nearest-neighbor projective transform orients the axes properly, and the bounds of the axes are found and cropped to. The reason for the nearest neighbor transform is that conversion to BW followed by a nearest-neighbor projective transformation is much faster than a bilinear projective transformation on the color image followed by a conversion to BW; the former also only leads to a slight loss in data quality.

Connected components of a particular size and shape near the axes are identified as corresponding to characters, and these are removed from the black and white image.

The characters themselves are easily located, oriented and resized. Attempts were made to apply character recognition to these; however, even limiting possible characters to digits and x/y, these were unsuccessful. This was either due to lack of accuracy in the pre-written algorithms found, such as in [5], or lack of time to implement theoretical results such as in [6]. This is why the user must enter the bounds of the graph manually if she wishes to have a non-normalized graph.
C. Graph Cleaning and Splitting

Small holes in the lines are removed, a skeletonizing morphological operation is applied and small branches are pruned (hole removal prevents loops from appearing in the thinned image), leaving the lines one pixel wide.

These lines are then cleaned so that each pixel (aside from branch- or end- points) has exactly two 8-connected neighbors, allowing easy identification of branch points. These branch points are identified and deleted to separate every segment of the graph. The axes are then separated and removed from the image.

D. Data Labeling

To demonstrate these steps, it shall be expedient to focus on a small area of the image, namely the right intersection between the line and the quadratic in Fig. 5.

With all line segments separate and no intersections, each line segment corresponds to one set of data. These line segments are then grouped together using three methods:

1) With the input to the program, the maximum number of separate line segments (N) in one column of the image is known. In a particular area, if only one line breaks with all other lines present (which would occur e.g. if the drawn line is too thin and thresholding failed), then the broken line segments can be safely spliced. This step should theoretically never result in false splices and is thus performed first.

2) If the line segments are of sufficiently different color, then likely the lines are color coded. Here the line segments are grouped into N groups based on their locations in CbCr space (ignoring Y due to different possible lighting conditions), and if they are sufficiently far from each other in CbCr space the plot is likely in colour. All graphs sufficiently
close to their CbCr group centroid are then grouped together. False splices are very unlikely to occur during this step, but may happen.

(3) The last grouping step examines the endpoint angles and positions, using a point up to 5 pixels (less if the liner segment is shorter) away from the endpoint to give every endpoint an angle between 0 and 360 degrees. All areas with two or more endpoints in which a pair of endpoints are unmatched and have an angle difference close to 180 degrees are identified, with the endpoints then being matched together. This step is most likely to cause false splices, and is performed last.

![Figure 8](image.png)

Fig. 8: Area of figure 6, after successful matching using endpoint angles and positions. Note two endpoints remain unmatched.

All three previous methods in this step are conservative, weighted to be unlikely to cause false splices at the cost of perhaps failing to splice segments that should be grouped together. This is because the final step (fitting) can function with greater than N line segments at only the cost of increased processing time, which is usually insignificant for lower numbers of graphs (see experimental results, section IV). In contrast, false splices cannot be handled and may cause large loss of accuracy in fitting.

This is also the reason that each step is performed only once; repeating the steps sequentially until no new matches are performed would result in more matches, but possible errors would be propagated in each iteration.

E. Plot Generation

When \( n \geq N \) labels (i.e. total number of unmatched line segments) are present, a binary \( n \times n \) matrix governing which labels can share a group is generated. If two labels share a column, the matrix records that they cannot be grouped together.

In the fairly rare case where a line segment breaks while mostly vertical without having its endpoints matched in the previous step (strictly impossible for functions, but hand drawings may be messy) this step will never group them together, causing sub-optimal fitting. Also, if \( n < N \), the algorithm will fail and exit. This is very unlikely due to the algorithms being weighted such that false splices are infrequent.

Finally, a set of \( N \) groups together containing all labels is randomly generated using the matrix above. The x- and y-values of all points in a group are gathered together, and the user-input equations are fitted to these groups. This results in a total root-mean squared error (RMSE) for all the graphs, which is recorded. This process is repeated a number of times (proportional to \( N \)) and the iteration with minimum RMSE is used to generate a final graph.

The final graph generated is a single plot containing \( N \) graphs of the proper color, with axes either normalized or with bounds input by the user, and the best-fit parameters in each equation are provided.

![Figure 9](image.png)

Fig. 9: Final plot generated.

A preview is also generated with this graph super-imposed on the original image (projective transformed, colors inverted and cropped to the plot area only).
V. CONCLUSIONS

The original goal was an algorithm that can create an accurate representation of a sketch of a graph with multiple plotted functions, in an amount of time significantly shorter than it would take the user to create the sketch manually. This can, broadly, be considered to have been accomplished. However, many areas of the algorithm still require improvement.

The long running time of the total algorithm is due to the long running time of the Hough transform, which itself is due to the transform being both fine and performed over a large range of angles and line positions. A smarter method other than an exhaustive search, such as described in [7], would greatly cut down on processing time.

With regard to the objective of as little required user input as possible, a feature this algorithm should but does not currently implement is automatic detection of the bounds and insertion of labels onto the axes to correspond with the sketch. Especially considering the proven successes of handwriting recognition for a limited subset of characters, future work could easily improve this area.

Robustness to errors when grouping labels is an important objective. Since scalability (to higher N, i.e. more functions) is limited by processing time and errors, and processing time itself depends on the number of labels going into the fitting step, more accurate grouping of labels would improve almost all aspects of the algorithm. Thus, implementation of a training set to teach the program how best to decide whether to splice, in order to minimize both incorrect splices and missed splices, would be an effective improvement. Currently the algorithm tries too heavily to prevent incorrect splices.

IV. EXPERIMENTAL RESULTS

Total processing time usually runs to approximately 16 seconds on a commercial desktop. Of this time, approximately 10 seconds are spent on the Hough transform and nearest neighbor projective transformation. Thus, running time is mostly independent of N for the usual conceivable N that would be present in a sketch (taken to be 5 or less).

<table>
<thead>
<tr>
<th>N</th>
<th>Notes</th>
<th>Running Time (s)</th>
<th>RMSE (normalized bounds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Computer-generated, Linear, quadratic [C, Bl]</td>
<td>6.052</td>
<td>0.0065</td>
</tr>
<tr>
<td>1</td>
<td>Linear [BW, Bl]</td>
<td>13.870</td>
<td>0.0082</td>
</tr>
<tr>
<td>1</td>
<td>Linear [BW, Li]</td>
<td>16.216</td>
<td>0.04</td>
</tr>
<tr>
<td>1</td>
<td>Linear [BW, Li] Rotated version of previous</td>
<td>15.738</td>
<td>0.0406</td>
</tr>
<tr>
<td>2</td>
<td>Linear, quadratic [BW, Bl]</td>
<td>18.511</td>
<td>0.8512</td>
</tr>
<tr>
<td>2</td>
<td>Linear, sine [Co, Bl]</td>
<td>17.357</td>
<td>1.0378</td>
</tr>
</tbody>
</table>

* TABLE I.

Test results on 6 reference images used. Code in notes is: [ (Color vs BW) , (Lined vs Blank paper) ]. All applications of algorithm on the reference image set (used to write and optimize the algorithm) resulted in success.

REFERENCES