Auto-Digitizer for Fast Graph-to-Data Conversion

EE 368 Final Project Report, Winter 2018

Deepti Sanjay Mahajan dmahaj@stanford.edu
Sarah Pao Radzihovsky sradzi13@stanford.edu
Ching-Hua (Fiona) Wang chwang9@stanford.edu

Abstract

A digitizer for converting graphs to data is frequently used in many research fields. Although digitizers currently exist, they require a great deal of user input, including manual clicking to define axis coordinates and select each data point. This manual selection is very time-consuming and not suitable for extracting data from massive sets of graphs. In this project, we provide a solution to this issue by focusing on common graph formats to develop an auto-digitizer that can help researchers process many graphs faster and more effortlessly.

1. Introduction

In most research fields, it is important to compare results with data from other research papers. Therefore, convenient graphical image-to-data conversion is desirable in order to more efficiently use previous work and facilitate new research. Catering to this need, digitizers[1,2,3] have been created and made available to serve this purpose. However, because the existing digitizers are designed to be very general, they require inconvenient amounts of manual clicking for defining axis coordinate and data points. This can be very time-consuming and tiring for users, especially when there are large amounts of data in the image to convert and/or many graphical images to process. For this reason, it would be very beneficial to develop a faster digitizer that can process certain format of graphs.

In this project, we developed an auto-digitizer that requires minimized user inputs. This auto-digitizer is implemented in Matlab, which is already a popular tool for general data processing. The input to our system is a cropped image of a graph and the output is the extracted data points.

We focus on a common format of graph as illustrated in Fig. 1. We generated graph templates with different label formats/ranges, different shape of symbols, y-axis in linear or log scale, and axes with major and minor ticks. Some basic assumptions of graphs are that the x-axis is at the bottom of the graph and y-axis is at the left side of the graph; also, the x-axis is in linear scale.

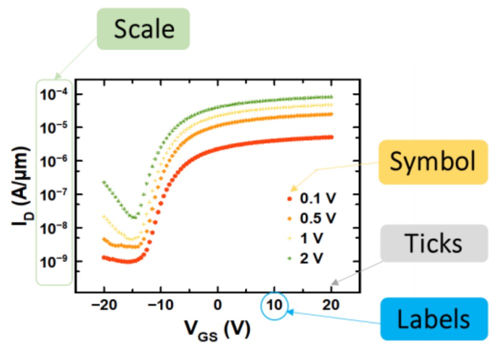


Fig. 1 The format example for the auto-digitizer

To achieve our goal of extracting data, we divide the implementation into three parts: (1) identify axes and the data range, (2) detect data points, and (3) convert pixel location to data values. In the last section, we will discuss the experimental result and future work for improving robustness and accuracy of the auto-digitizer.

2. Preprocessing

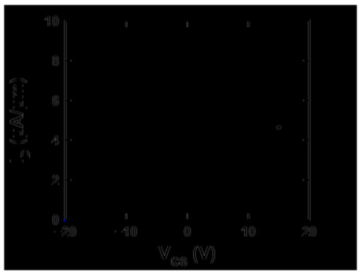
Since there is a possibility that the user may input a graph image whose axes are not perfectly vertical and horizontal, we preprocess the image to straighten any skew in the image. We first use a Canny edge detector to isolate the edges within the image. Then, we apply a Hough transform to find the straight lines in the image. The lines corresponding to the highest four peaks of the Hough transform are then found, and out of these the most prominent two lines (with different angles) are chosen. These should correspond to the axes. The smaller angle of the two is used to rotate the image, and the rotated image is then used for data extraction.

3. Define Data Range on Axis

3.1 Axis Isolation

Firstly, for isolating the x-axis, we apply the horizontal Sobel filter to extract all lines in horizontal direction. In this filtered Sobel image (Fig. 2a), by detecting the first line (from the bottom) that has more detected values than half of the image width, we can find the x-axis. Then, we find the beginning and the end of the x-axis to get an exact axis location. We repeat the same methodology on the image with vertical Sobel filter (Fig. 2b), and extract the left-most vertical line on the graph, which is y-axis. With the information of x and y axis locations, we can identify the coordinates of the origin in the bottom left corner.

(a)



(b)

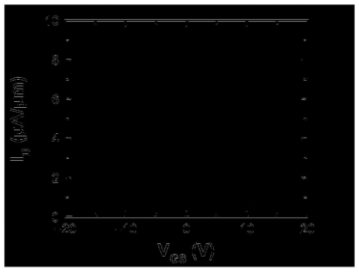


Fig. 2 (a) Image after vertical Sobel filter (b) Image after horizontal Sobel filter

3.2 Label recognition

To minimize user input, we used Optical Character Recognition (OCR) in Matlab to recognize the labeled values along the axes automatically. The purpose of OCR is to classify optical patterns (often contained in a digital image) corresponding to alphanumeric or other characters. In order to make the OCR work most reliably with our application, we fine-tuned the OCR function by providing information regarding how the character format and the type of characters it should look for (numeric-related characters only). The process of OCR involves several steps including segmentation, feature extraction, and classification. After the processing, the function generates a confidence value to each digit, as shown in Fig. 3. The function successfully identifies the number when the confidence value is larger than a threshold of 0.9. However, the OCR function does not work for exponential numbers in the current version. In the case that the OCR is not able to identify the labels of a particular graph, the auto-digitizer asks the user to input the needed information.

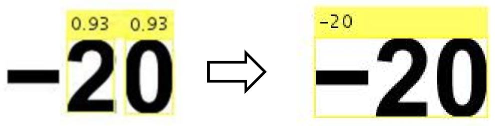


Fig. 3 The OCR function generates confidence value with for each digital number, and identify label value successfully when the confidence value is larger than 0.9

3.3 Axis value mapping

In order to compute the data value for each data point, the values at the endpoints of each axis are needed. Firstly, we use the number detected by the OCR function and assign this number to the corresponding points on the axis to get a data range. The corresponding points of label values on the axis are major tick locations. However, we found it is difficult to differentiate between major and minor ticks. Therefore, we assume each label bounding box is centered at its corresponding tick mark (as illustrated in Fig. 4). With this assumption, we can find the lowest and highest labeled value on x and y axis, then extrapolate the data values corresponding to the beginning and the end points of each axis. This minimum and maximum x/y value is used, along with the data point locations, to generate the data's x and y values.

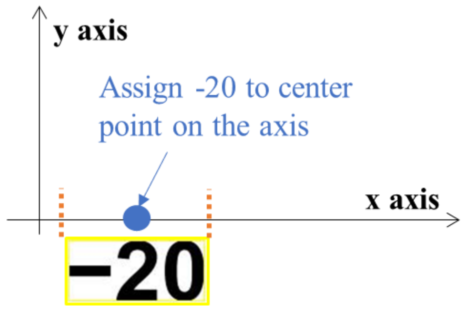


Fig. 4 Assigning data value to the corresponding point on the axis by locating the center of bounding box.

4. Data Extraction

Before mapping a data point to an actual value, we extract the pixel location for the data points (plotted as lines or symbols). Firstly, the auto-digitizer will classify if a graph is a line graph or scatter plot. If a graph contains a line connected to axis, the total regions surrounded by axis would be only 2. Therefore, the auto-digitizer counts the number of regions and classify the graphs as line graph if region number is 2, and as a scatter plot otherwise. Here we crop out the axes and potential tick marks along the edges of the graph so that the tick marks are not detected as data points. However, this results in some of the extreme data points being ignored.

This classification methodology may not work for graphs with a different format or multiple lines. To improve this classification in the future, we can either ask for some additional user input or use a more complex approach that considers some of the region properties.

4.1 Data point location detection for lines

When a graph is classified as a line graph, we use a preset resolution of output data (how many output points). After cropping the binarized image in order to remove tick marks, we find linearly spaced columns at the desired resolution. We then invert the image, so the background is black, and search for nonzero values along each column. The positions of these nonzero values indicate the location of each data point along the line.

4.2 Data point detection for scatter plots

4.2.1 Extracted symbol shape with user input

In the case of a scatter plot, as the symbol shapes in graphs are very diverse, we require users to identify the symbol of interest by circling an isolated symbol, as illustrated in Fig. 5.

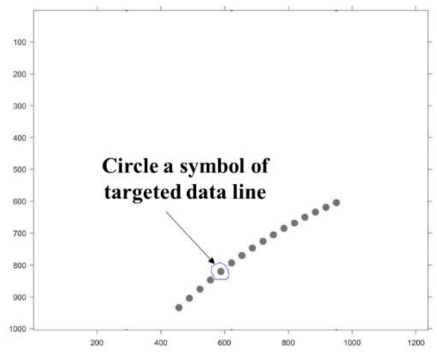


Fig. 5 Asking user to identify the symbol of interest

The selected region by user is then used as a mask over the binarized image, as shown in Fig. 6. As a result, we can isolate the symbol shape to use as a template for the following process.

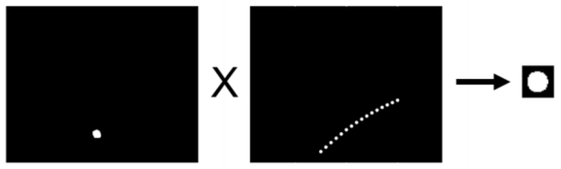


Fig. 6 Isolating symbol from circled region

4.2.2 Erosion with extracted symbol shape

As the input image may have some noise (depending on its resolution), we perform a closing operation on the full image to reduce the noise. Then, we erode the symbol template with a 3x3 square to ensure that it works robustly as a structuring element. Furthermore, to avoid detecting tick marks along the edge of the graph as data points, we crop the image, ignoring a margin of 1/30th of the plot width. Thus, any data points that are very close to the edges may not be detected. To obtain symbol positions, the image is eroded with the extracted symbol as a structuring element. The coordinates of the resulting nonzero values, as shown in Fig. 7, are the data point locations within the graph image.

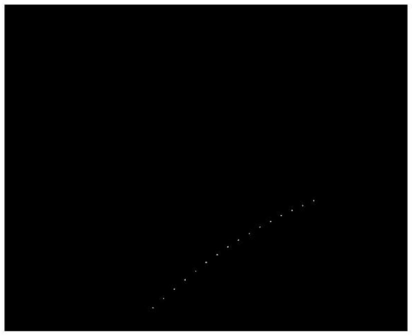


Fig. 7 Image of graph after erosion with template

5. Data Point Location to Data Conversion

To map each data point to a data value, the auto-digitizer asks users if the graph has log or linear scale. For now, we assume the x-axis is linear, so we ask the user to indicate whether the y axis is in log or linear scale. Given the minimum and maximum labeled x and y values found previously, Fig. 8 illustrates the interpolation to derive the value associated with the endpoint of the y-axis (in log scale). We can use the same idea of interpolation to extract the x and y raw data values corresponding to each detected data point.

$$\alpha = \frac{f}{h}\left(\log(y_{min}) - \log(y_{max})\right)$$

$$y_{corner} = 10^{\log(y_{min})-\alpha}$$

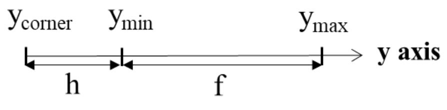
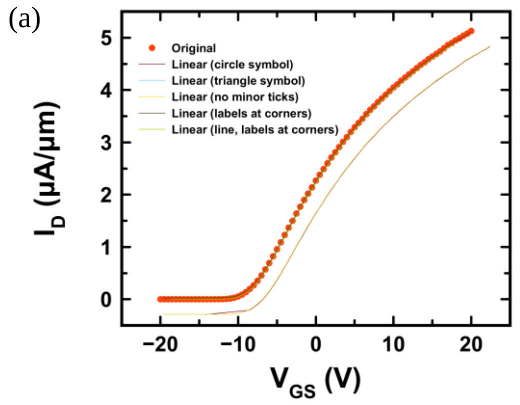


Fig. 8 Illustration of derivation of corner value in log scale

6. Experiments and Performance

In this section, we apply the auto-digitizer to different graph formats to evaluate current performance. From Fig. 9a, by comparing graphs in linear scale, we can see that our auto-digitizer is not sensitive to symbol shape and ticks format. However, there is a small offset, especially when the axis data label numbers are not at the corners. In log scale, we can see that some data near the edges is missing due to image cropping for noise reduction.



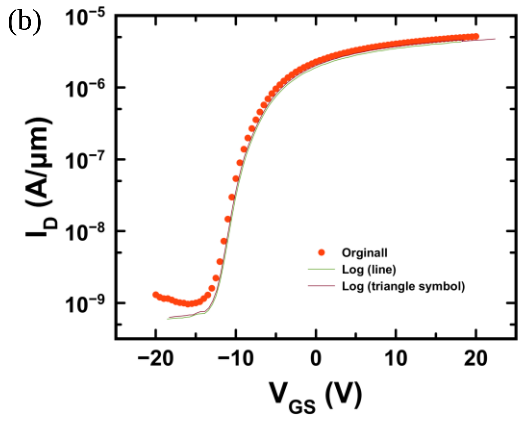


Fig. 9 Comparison of output data of different formats of graph to original data (a) in linear and (b) in log scale

Figure 10 illustrates again the performance of our current auto-digitizer. We averaged the offset in Fig. 9 at different x values (when V_{GS} = -5, 0, and 5 V). The offset is larger when labels are not at the corners. For log scale graphs, even with label numbers at the corner, the absolute error value is larger than in linear scale as offsets are magnified in log scale.

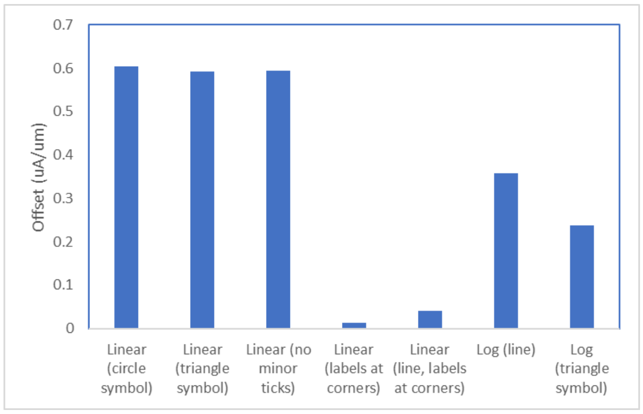


Fig. 10 Comparison of extracted data error at the same voltage (x value) in differently formatted graphs

We tested two graphs from papers, as shown in Fig. 11. The left sides are the images of extracted unsorted data as they are scattered plots. Although the resolutions of the graphs were too poor for OCR to work correctly, the data was able to be extracted from user input of the minimum and maximum axis values.

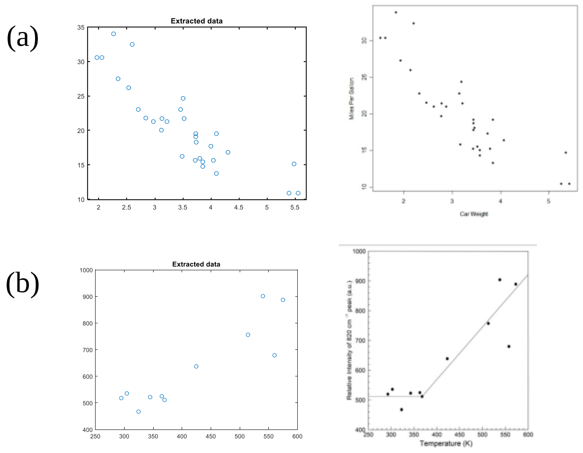


Fig. 11 Left images are extracted data with sorted; right images are cropped from papers.

7. Conclusion and Future Work

We compare our auto-digitizer to the current available "plot digitizer". In the plot digitizer tool, users must to click on each data point they want to extract. Increasing with the number of data points there are in the graph, this is tedious and time-consuming work. In our auto-digitizer, if the OCR can detect the axis label numbers successfully, the total process time is a constant of only 20s and requires less effort.

Our current version of the auto-digitizer is still sensitive to the format of input graphs and exhibits a slight offset in extracted data values to the original graphs. Moving forward, we will focus improving the robustness to graph format and background noise, and develop a methodology for data mapping that yields more accurate results.

References:

[1] http://plotdigitizer.sourceforge.net/

[2] https://engauge-digitizer.soft112.com/

[3]https://www.originlab.com/doc/Origin-Help/Tool-Digitizer

[4] https://github.com/ankitrohatgi/WebPlotDigitizer

Appendix: Author Contributions

Deepti Mahajan: Developed code for conversion of data point location to raw data and data point detection for lines; Contributed to report write up

Sarah Radzihovsky: Developed code for axis and origin detection, OCR for axis labels, and extrapolating the data value range of axes; Contributed to report write up

Ching-Hua (Fiona) Wang: Developed code for extracting symbol shape template and data point detection for scatter plots; Proposal, poster, and report write up