Automated data extraction from 3D bar chart images

Jinho Lee Jinhwan Sul Myeonghyeon Song Tackmin Kim
Seoul National University
ljh0728@snu.ac.kr jhsul51@snu.ac.kr sniaw21@snu.ac.kr taxis853@snu.ac.kr

Abstract

We propose an automatic way to extract data from 3D bar charts. There were some studies which tried to extract data from the chart, but most of them relied on human interaction and the dataset was limited to 2D charts. In contrast to the recent work, we have focused on the automation and the 3D bar chart dataset. We made several assumptions based on the characteristics of Matlab 3D bar chart, since it is one of the most popular data visualization tools. Our method consists of 6 pipelines as follows: 1) Finding axes, 2) Detecting label values, 3) Separating series with colors, 4) Finding the top face of each bars, 5) Finding the bottom line of series, 6) Data extraction.

With 35 randomly generated ground truth data, we measured our performance by calculating the mean error rate. We could achieve high accuracy, which means the mean error rate below 10%, in 9 test images.

1. Introduction

Charts are a basic data visualization method widely used in digital documents. From daily newspapers to academic papers, we can easily see charts in many types such as 2D bar chart, 2D line chart, 3D bar chart, 3D pie chart, and so on. People would need underlying data for further analysis, but usually, the data are not provided.

It is difficult to extract data with bare hands and eyes from any chart image and it may also lead to an accuracy issue. There exist several studies on chart data extraction systems, but 3D charts were not included in their dataset. Since 3D charts are commonly used as much as 2D charts, there are needs for the 3D chart data extraction system, too.

The main goal of our project is to extract underlying data from the 3D bar charts. However, because 3D bar charts are projected to 2D images from 3D space, there are some challenging parts in this problem. First, they need extra care for measuring axes and values. Second, some data could be occluded and partly invisible.

In this project, we automatically extracted data from 3D bar chart images by our novel idea. We approached the problem by conceptualizing human behavior of how we read the 3D bar charts. Humans first find the top face of each bar and move it parallel to the axes to read its underlying data. Therefore, we implemented this method in our algorithms. Our code is available at `https://github.com/sniaw21/data-extraction-from-3d-chart`.

2. Related work

Several related papers have focused on data extraction from charts. However, all of them only considered 2D charts.

ReVision [5] extracted underlying data from bar charts and pie charts by exploiting connected components analysis. The performance has reached state-of-the-art on the data extraction at that time(2011). The study did not handle 3D effects.

Chartsense [2] tried to solve the data extraction problem by using user interactions. It first determined the chart type using CNN. By helping users to easily interact with the program, it successfully extracted data from the chart. However, it concentrated on a slightly different subject which is how to make user interaction effective. It also excluded the 3D effect.

WebPlotDigitizer [4] is the most popular data extraction tool on the web. However, it also requires many user interactions which makes them uncomfortable.

Reverse-Engineering Visualizations [3] suggested recognizing text regions that cannot be handled at ChartSense [2] and WebPlotDigitizer [4]. It first found the bounding box of the text and created features by using the position of the detected bounding box and aspect ratio. Then it used multiclass SVM to classify the texts and applied additional post-processing. It had a high F1 score on recognizing and classifying texts but it suggested further study for improving robustness and interpreting legends.

Text detection plays an important role in data extraction. We introduce 2 works used by our project.

CRAFT [1] is a scene text detection method based on neural networks, which was developed by NAVER CLOVA.

To effectively detect text area, it produces two scores, region score and affinity score. The region score is used to localize individual characters and the affinity score is used to group each character into a single instance. Here, we used CRAFT to detect labels in the chart.

Tesseract OCR [6] is an open source OCR (Optical Character Recognition) engine, which was originally developed at HP lab. Since 2006, it has been developed by Google. The latest major version is 4, which is based on the deep learning based method LSTM. We used tesseract to read the labels detected from CRAFT.

OpenCV is an open source image processing library developed by Intel first. It has many features regarding computer vision, we could use them as a part of our main algorithms. erode, dilate, threshold, calcHist, matchTemplate are examples. We also used it to visualize our intermediate progress.

3. Dataset and assumptions

MATLAB is one of the most popular softwares for numerical computing and it has various functions and options to visualize the data. In this project, our focus was on MATLAB 3D bar charts. First, we randomly generated 35 ground truth data in MATLAB. In order to maintain the generality, bar colors and bar width was randomly chosen as well. Figure 1 represents the example chart images from the dataset.

There are some characteristics in MATLAB 3D bar charts. Based on those characteristics, we made seven assumptions.

1. Axes and labels are well defined and easily noticeable.
   This means that labels are not overlapped and easy to read, axes are fully visible on the image, and ticks and labels are easily matched with each other by human vision.
2. The direction of axis is same.
   We assumed that the x, y, z axis remained the same viewing direction, which means they are in the same pose. Here, we defined the z axis as vertical axis. We call it y axis which the color of the bars changes along. The last axis is defined as the x axis.
3. Orthographic projection is applied.
   Height ratio among bars are maintained regardless of the location of bars in the image.
4. Bars stand vertically, not horizontally.
5. Each series of bars have distinct colors.
   In our program, we distinguished series with colors. Therefore, all the bars must have distinct colors, even with the background including axes and labels.
6. Auxiliary lines are parallel to the axes.

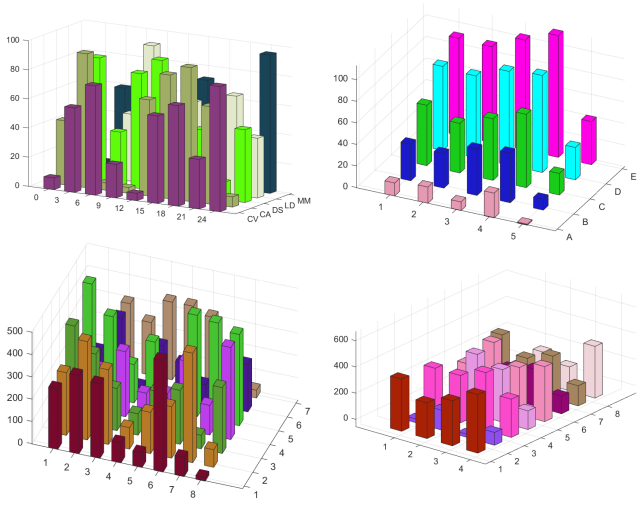


Figure 1: Example chart images from dataset.

   As shown in Figure 1, there are auxiliary grid lines on the image. We assumed that they are parallel to the axes.
7. Top face of each bar is visible.
   The viewpoint is above the height of charts, so we can see the top face of each bar.

4. Method

We solved the problem with the following pipeline.

1. Finding axes
2. Detecting label values
3. Separating series with colors
4. Finding the top face of each bar
5. Finding the bottom line of series
6. Data extraction

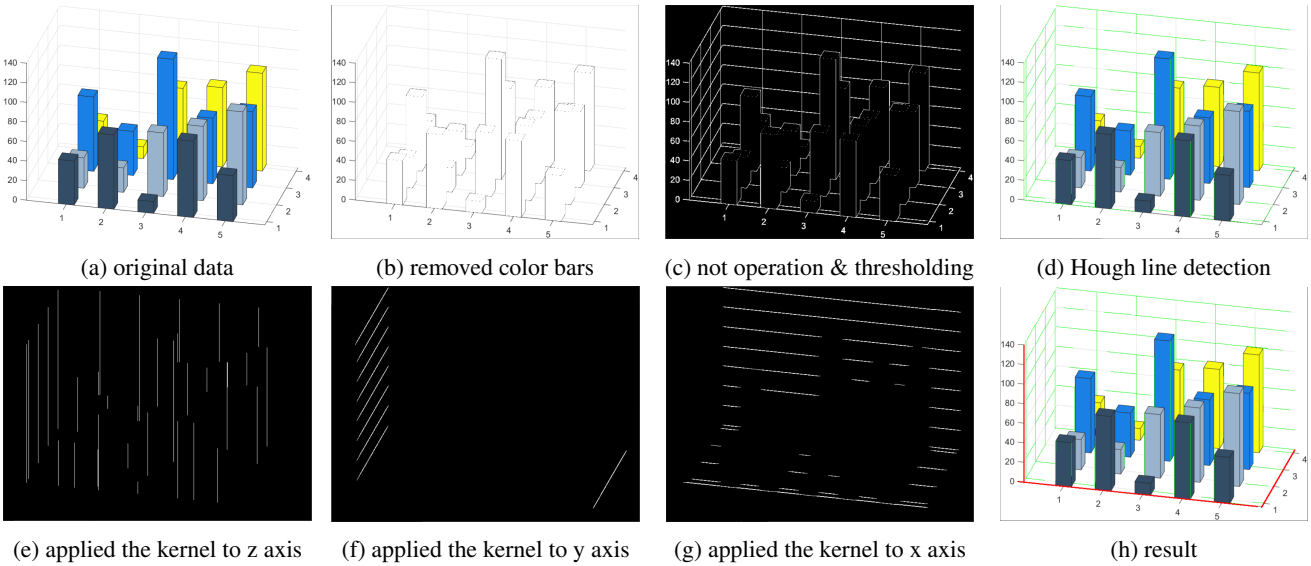
4.1. Finding axes

Finding axes is the first step of the algorithm. Because the locations of axes are used in the following pipelines, it is very important to detect them correctly. The goal of this step is finding coordinates of start and end point of each axis and detecting the degree of each axis.

We first considered applying OpenCV's Hough line transform to directly detect axes. But it showed the terrible performance. It only detected the segment of axes and there were a lot of false positives. So we decided to use the erode and dilate method. The details are as below.

Preprocessing

First, colored bars were removed using HSV information. We simply left the pixels whose S value is 0. After



(a) original data (b) removed color bars (c) not operation & thresholding (d) Hough line detection

(e) applied the kernel to z axis (f) applied the kernel to y axis (g) applied the kernel to x axis (h) result

Figure 2: Intermediate results of finding axes step.

that, bitwise not operation and adaptive threshold were applied. Figure 2 (b), (c) represent the results.

Finding the degrees

Because of the characteristic of MATLAB 3D bar charts, we could assume that there are auxiliary lines which are parallel to the axes. We used this assumption to infer the degree of each axis. OpenCV's Hough line transform was applied to the preprocessed input image. While doing this, we set the parameters large enough to detect only axes and auxiliary lines, not texts or labels. The green lines in Figure 2 represent detected lines. We computed the degrees of detected lines and counted the rounded up values. After applying non-maximum suppression to the degree histogram, we found top 3 frequent degrees.

Making kernels

To use erode and dilation functions, we had to make kernels first. We used the degrees of axes here. If the detected degree is 90 or 0 (vertical or horizontal case), we simply made a kernel as (25, 1) or (1, 25) matrix whose elements are all one. If it is not, we made a kernel whose width is $25 \cdot cos(degree)$ and height is $25 \cdot sin(degree)$. Then, we set its diagonal elements as one and the rest as zero. We chose 25 as diagonal length because it empirically removed the labels and left only axes and auxiliary lines well.

Applying the kernels

We applied OpenCV's erode and dilation functions with kernels made in the last step. By doing this, we could leave only axes and auxiliary lines parallel to the detected degree as shown in the Figure 2 (e), (f), (g).

Finding the start and end point of each axis

When finding the start and end point of z-axis, since the degree is always 90 by assumption, we simply found the leftmost column of where the lines appeared and found the uppermost and bottommost points of that column. When the detected degree is between 0 and 90, which means it is y axis, we found the bottommost point and the rightmost point of where lines appeared. If the detected degree is smaller than 0, which means it is x axis, we found the bottommost point and the leftmost point of where lines appeared.

4.2. Detecting label values

In this section, we're going to discuss how we recognized the ratio between pixel and label value. There are 4 steps, text box detection, missing label detection, ocr, picking the most frequent value difference.

Text box detection

First, we ran text detection using CRAFT. From acquired text bounding boxes, we filtered them by distances from each axis. The distance threshold was 40px.

Missing label detection

To increase accuracy, detecting missing labels is also required. We assumed that there are at least 2 sequential labels detected, so we could calculate the minimum label position difference for each axis. We found candidate text boxes by repeatedly adding the minimum position difference to the start position of the axis line. You can see the result with an example at Figure 4.

OCR

We ran tesseract OCR for each candidate text box. 5 different images were tried to enhance accuracy as below,

(a) Original image (b) Binarized image by otsu's thresholding (c) 5x upscaled original image (d) 5x upscaled binary image (e) Binarized image from the image (d)

Figure 3: Preprocessed images used in OCR.

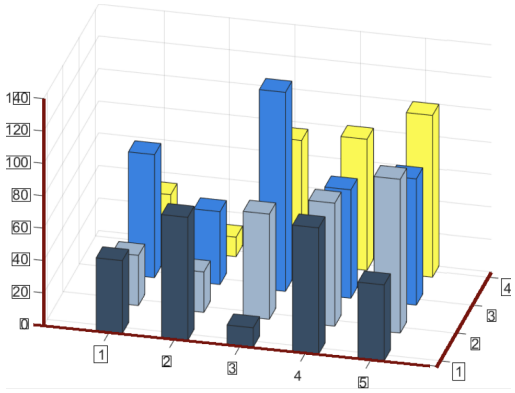


Figure 4: Red lines indicate axes and boxes around numbers are the result of the pipeline 2.

you can see an example at Figure 3.

1. Original image
2. Binarized image by otsu's thresholding
3. 5x upscaled original image
4. 5x upscaled binary image
5. Binarized image from the image 4

It sometimes failed to detect them correctly. For example, number 0 is frequently detected as alphabet o.

Picking the most frequent value difference

Finally, from the list of label values, we calculated the value differences between ticks. Then we voted them to pick the most frequent value difference. That was the final result which is a ratio between pixel and label value. We will refer to it as the pixel-label ratio. Additionally we also acquired labels of x and y axis, even though they were not used for accuracy evaluation.

4.3. Separating series with color

In order to make the problem simple, we separated multi-series chart images into single series chart images by colors. Humans distinguish series of 3D bar chart with colors, which means the color is one of the most significant features in a chart image. Therefore, we introduced HSV color space to distinguish series. After that, the number of series was detected. Also, distance between series was calculated.

Finding the most frequent (H, S) value

For the first step of separating the chart image, we converted the image into the HSV color space. We acquired a HS histogram using OpenCV's histogram calculation function. From the HS histogram we excluded pixels which had zero S value because it represents achromatic color, which are mostly background, axes and auxiliary lines.

Find the number of series

We set the threshold by heuristic, and found the colors that appeared more frequently than the threshold in the histogram. The number of colors found in the histogram was the number of series. The threshold was set as 8% of the maximum number of pixels in the HS histogram. After the colors were obtained, we had to eliminate colors counted as duplicates using another heuristic. Colors closer to each other with a certain amount of (H,S) value were considered as the same color and merged. Through this step, the original chart image was separated into single series chart images shown in Figure 5.

Finding the distance between series

We set a strategy of measuring distance between the series. First, we applied a median filter in the separated images to remove noises. Second, the distance between series was obtained by calculating the difference in the lowest location of each series.

4.4. Head detection

Definition of head

The way humans read the 3d bar charts is to recognize the top part of the chart and move it parallel to the axis to match it. Considering the human behavior, head detection is necessary. Therefore, we defined the term "head" in this project as the top face of each bar.

3D bar charts generated by MATLAB have some features. Their heads are easy to distinguish from the bars be-

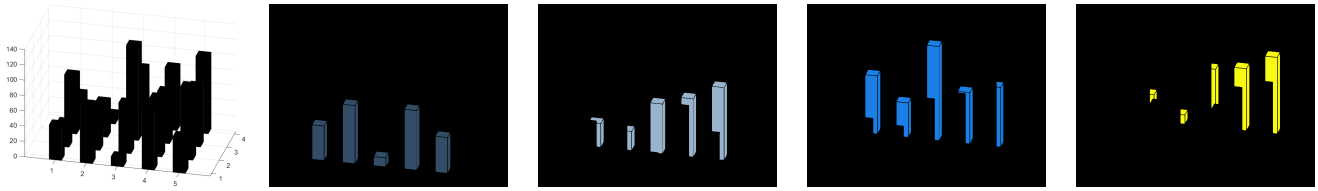
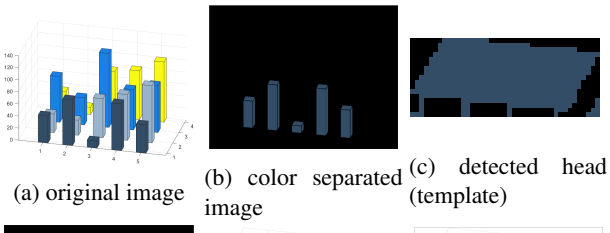
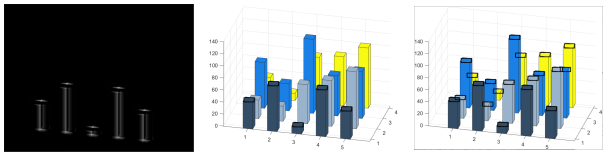


Figure 5: Leftmost image is background of chart image and other images were separated single series images.



(a) original image (b) color separated image (c) detected head (template)



(d) template matching similarity (e) matched parts for one color (f) matched parts for all colors

Figure 6: Intermediate results of head detection

cause they are separated by black lines. On the other hand, some of the heads are partly occluded or entirely occluded.

Head detection

Head detection is a process of finding heads and saving them as template images. In the previous pipelines, we obtained the images of each separated color. We used these images to find the uppermost head of each series.

Before executing head detection, preprocessing is needed. It is to make the edges thicker to help the distinction between the head and the bar clear. We used OpenCV's erode function to do this.

After converting the edge-enhanced images as grayscale images, we could find the head of each series by finding the upper most non-black area and saving it as a head template image.

However, we might have a problem from occlusions. We solved this problem by taking the template which has the maximum size among the previously detected templates as a function argument. If the detected head is smaller than the maximum size head, then replace it with the maximum size head and change the color into its series. In this way, we could obtain the full head even if the uppermost head is partly occluded.

As a result, the function returns the template image of the detected head, and an information of size of each head. These will be used in the following step, head matching.

Head matching

Head Matching is a process of matching head templates to the color separated images. Then we save the coordinates of the matched heads which denote the information where the heads are.

The first step is matching each template to the color separated image. Template matching method was done repetitively from high threshold (0.9) by lowering the threshold until it finds at least one part to avoid the worst case that no parts are matched. In addition, we might have matched parts adjacent within a few pixels. Therefore, we applied non-maximum suppression among the adjacent matched parts.

The horizontal interval between the heads could be calculated with the assumption that the bars are spaced equally in horizontal direction. Then, we could solve a possible problem of skipping partly occluded heads as follows.

Partially occluded heads might not be matched to the template previously because of the high threshold. Hence, by finding the most matched part within the interval, we could even detect the partly occluded heads even if the matching similarity was lower than the threshold we set before.

As a result, the function returns coordinates of the matched parts, and minimum horizontal interval of heads which would be used in finding the occluded parts of the next series. The whole process of head detection with intermediate results is shown in Figure 6.

4.5. Finding the bottom line of series

From the axis location, the degree of axis, and distance between series, we found the bottom line location of each series. From the distance between series and axis information, we evaluated the jump vector, which moves the start and end point of the axis with one series interval. Therefore, by iterating through this process, we located the bottom line in each series as shown in the Figure 7.

4.6. Data extraction

From the bottom line result, we measured the vertical distance from the bottom line to the head of each bar. We needed one preprocessing for the result of head detection.

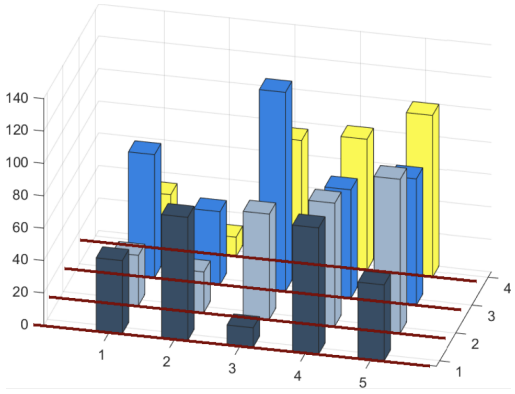


Figure 7: Result of finding bottom line of series.

Since we might skip some heads because of occlusion, we needed to match each head position to each bar. So we projected each head to the bottomline. Then we sorted heads and inferred the corresponding bar index. Therefore, we filled 'None' value to head-missing bars, and finally we predicted the underlying data based on the pixel-label ratio.

5. Experiments and results

In order to verify the program, we evaluated a mean error rate for all 3D chart images in the dataset. The mean error rate was calculated from the average errors for each bar in the image. Because there are completely occluded data, we excluded undetected values when calculating mean error rate.

$$mean\ error\ rate = \frac{1}{N}\sum_{i}\delta_i\frac{|GT - extracted\ value|}{GT}$$

N = the number of detected data

GT = ground truth value.

$$\delta_i = \begin{cases} 1, & \text{if predicted value is not None,} \\ 0, & \text{if predicted value is None.} \end{cases} \quad (1)$$

Table 1 represents the mean error rate result from the dataset. The program showed 0% $\sim$ 10% of mean error rate for 9 images, and those were the cases when the program performed well as shown in Figure 8 (c). The program showed 10% $\sim$ 40% of mean error rate for 9 images, and those were the cases when head detection showed few outliers as shown in Figure 8 (b). Error rate increased when more outliers were shown in head detection. For the 7 images, the program showed more than 40% of mean error rate, and those errors originated from data extraction. The biggest issue was mismatching of the information between the number of bars in one series calculated from head detection and that from the ground truth. It might occur because the heads were over-detected or the heads were under-detected(skipping issue). Hence, the mean error rate was evaluated with unmatched bars and showed huge error as shown in Figure 9. Lastly, the program failed to extract data for 10 images. Those were the cases when both head detection and text detection had failed to find the exact number of bars in one series. Figure 8 (a) shows the case when head detection had failed in huge error.

Mean Error Rate	Number of images
0% $\sim$ 10%	9
10% $\sim$ 40%	9
40% $\sim$	7
Failed	10
Total	35

Table 1: Mean error rate results from dataset.

6. Discussion and future work

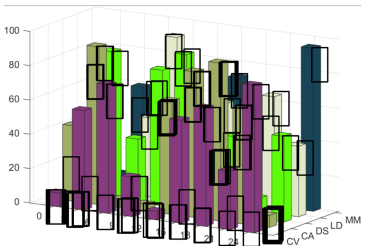
6.1. Data extraction accuracy

Although we have set several assumptions and applied heuristics, we succeeded to create accurate results for some of the charts. There were two cases of failure: large errors and abortion during processes. First, large errors occurred because some occluded bars were detected as wrong positions during the head detection. Second, abortion during processes was caused by failure of text recognition or failure of detecting all the heads in a whole series. Hence, we may be able to increase the accuracy by improving the algorithm of head detection and text recognition. Finding a better algorithm for detecting partly occluded heads and achieving robustness of text recognition would result in the better accuracy.

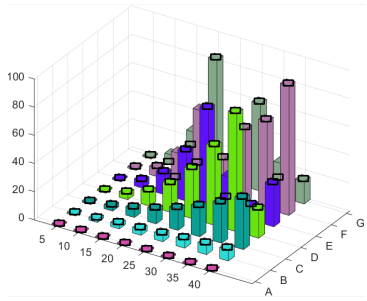
6.2. Generalizability

Dataset was limited to 3D bar charts generated by MATLAB. We applied some assumptions such as no perspective applied, clearly visible axes with auxiliary lines, series distinguished by different colors and heads separated by black lines. They were only valid in the chart images from MATLAB.

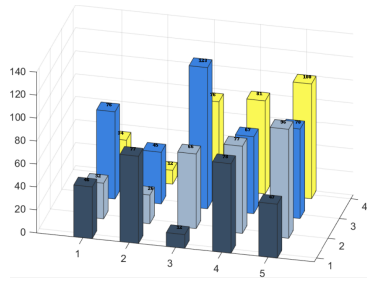
However, in the real world, various kinds of 3D bar charts are in use. To include those in the dataset, we should revise our algorithm. Axes finding algorithms should find axes without auxiliary lines. Separating color algorithms should separate each series of bars even if they have similar or same colors. Head detection algorithms should detect heads even if they are not separated from the bars they belong to. Data extraction algorithms should deal with the images with perspective by considering the vanishing point while moving the coordinates of heads to the axes.



(a) Head detection result of Matlab21.png.

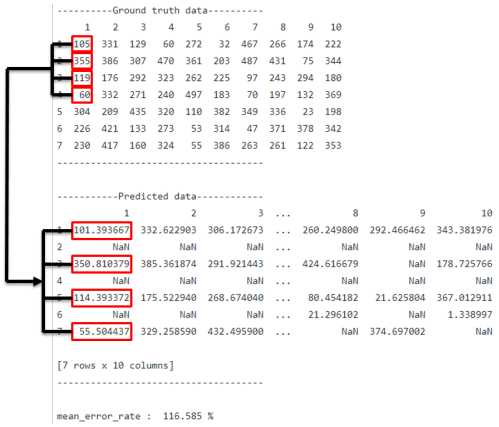


(b) Head detection result of Matlab24.png.



(c) Final result of Matlab27.png

Figure 8: (a) Head detection had failed and eventually failed to extract data. (b) Some of the head locations were detected wrong. (c) Data were extracted successfully.



```
----------Ground truth data----------
   1   2   3   4   5   6   7   8   9  10
1 105 331 129  60 272  32 467 266 174 222
2 355 386 307 470 361 203 487 431  75 344
3 119 176 292 323 262 225  97 243 294 180
4  60 332 271 240 497 183  70 197 132 369
5 304 209 435 320 110 382 349 336  23 198
6 226 421 133 273  53 314  47 371 378 342
7 230 417 160 324  55 386 263 261 122 353
-----------------------------------


----------Predicted data-----------
          1          2          3  ...          8          9          10
1 101.393667 332.622903 306.172673  ... 260.249800 292.466462 343.381976
2       NaN        NaN        NaN  ...        NaN        NaN        NaN
3 350.810379 385.361874 291.921443  ... 424.616679        NaN 178.725766
4       NaN        NaN        NaN  ...        NaN        NaN        NaN
5 114.393372 175.522940 268.674040  ...  80.454182  21.625804 367.012911
6       NaN        NaN        NaN  ...  21.296102        NaN   1.338997
7  55.504437 329.258590 432.495900  ...        NaN 374.697002        NaN
[7 rows x 10 columns]
-----------------------------------

mean_error_rate :  116.585 %
```

Figure 9: Data extraction result of Matlab36.png. The mean error rate was evaluated with unmatched bars.

7. Conclusion

We presented a novel method of automated data extraction from 3D bar charts: a system that extracts underlying data from 3D bar chart images generated by MATLAB automatically. Our quantitative evaluation showed that our algorithm achieved enoughly high accuracy for some chart images. However, it showed clear limits in for extracting data from partly occluded bars and low robustness of text recognition. Furthermore, for detecting other kinds of 3D bar charts, the algorithm needs partial modification and improvements.

References

[1] Youngmin Baek, Bado Lee, Dongyoon Han, Sangdoo Yun, and Hwalsuk Lee. Character region awareness for text detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9365–9374, 2019.

[2] Daekyoung Jung, Wonjae Kim, Hyunjoo Song, Jeong-in Hwang, Bongshin Lee, Bohyoung Kim, and Jinwook Seo. Chartsense: Interactive data extraction from chart images. In *Proceedings of the 2017 chi conference on human factors in computing systems*, pages 6706–6717, 2017.

[3] Jorge Poco and Jeffrey Heer. Reverse-engineering visualizations: Recovering visual encodings from chart images. In *Computer Graphics Forum*, volume 36, pages 353–363. Wiley Online Library, 2017.

[4] Ankit Rohatgi. Webplotdigitizer, 2017.

[5] Manolis Savva, Nicholas Kong, Arti Chhajta, Li Fei-Fei, Maneesh Agrawala, and Jeffrey Heer. Revision: Automated classification, analysis and redesign of chart images. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 393–402, 2011.

[6] Ray Smith. An overview of the tesseract ocr engine. In *Ninth international conference on document analysis and recognition (ICDAR 2007)*, volume 2, pages 629–633. IEEE, 2007.