Generation of Slides from Hand-Drawn Sketches

Muneeb Ahmed
Electrical Engineering
Stanford University
Stanford, California 94305
Email:ahmedm@stanford.edu

Jeff Wheeler
Electrical Engineering
Stanford University
Stanford, California 94305
Email: jeffsw@stanford.edu

***Abstract*—As engineers, we often find ourselves spending a non-trivial amount of time converting sketched diagrams and figures into digital format. A lot of research has been done in conversion of hand-drawn sketches to computer drawings. However, most of it is domain-specific with training templates which must first demonstrate the shapes the user first intends to recognize as suggested by Kara and Stahovich [1] as well as Puzicha et al. [2]. Other algorithms are optimized for specific domains (e.g. for military drawings [3]). Due to the variability in drawing trends and variation in interpretation, a robust domain-independent algorithm is lacking.**

In this paper we describe a domain-independent algorithm for basic shape detection. The shape detection algorithm works by detecting corners using the Harris corner detector. The corners are then removed which allows for line detection. The lines are then combined into shapes. Since the images are captured using a hand-held device, perspective distortion is taken into account by applying homography. Slightly slanted lines are then converted into perfect horizontal/vertical lines due to unavoidable imperfections in human hand-drawn shapes.

I. Introduction

We aimed to convert hand-drawn shapes to slides. To simplify the conversion process, we only attempt to characterize polygons and ellipsoidal shapes. Once characterized, we do some manipulation to account for perspective distortion and human error before generating the slide. This process is shown in Figure 1. We also created an Android application that would capture the image and send it to the server. Upon receipt of the captured image, the server will process the image and the application will show a preview of the shape recognition. Upon approval, the server will forward the slides with digitalized shapes to the email address that the user provided. The process is shown in Figure 3.

II. Algorithm

The algorithm is implemented in python using openCV and NumPy. The algorithm can be broken down into discrete steps which are explained as follows:

A. Image Acquisition (Figure 1a)

A set of images is fed into the algorithm. The images are captured using a hand-held device. This means that the images are generally from 2 MP - 10 MP range. The algorithm is set up such that it assumes an input of high-resolution images. It then rescales the image by 0.2-0.3x to account before further processing to improve the turn-around time.

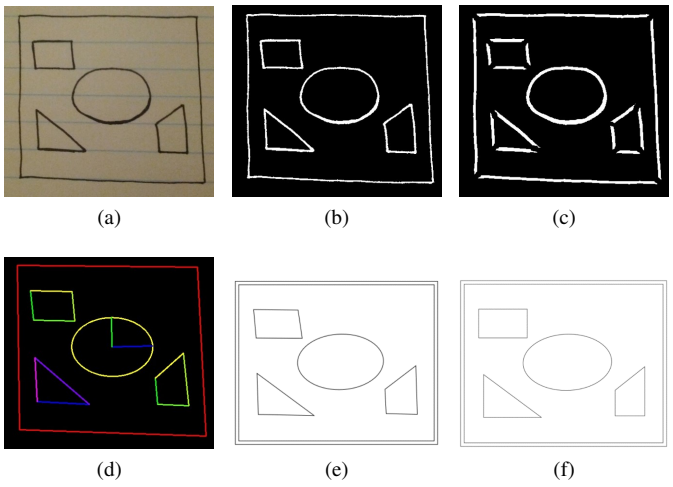

Fig. 1. Algorithm Process Flow: (a) Image captured with a hand-held device (b) Binarized image after closing operation (c) Image with corners removed (d) Detected shapes shown with different colors. Slight color variation within the same shape signify the detection of individual edges. (e) Display of detected shapes after homography (f) Display of detected shapes are angle straightening

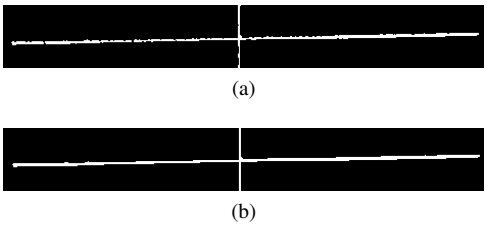

Fig. 2.
(a) Binarized image before closing (b) Binarized image after closing

B. Binarization and Closing (Figure 1b)

To begin with shape detection algorithm, we first binarize the image to ease manipulation with the shapes. This process is extremely important as all the other steps rely on high-quality binarization to function properly. We found that adaptive thresholding (using weighted sum with a Gaussian window) gave the best results since it took into account changing lighting conditions within the same image. Our algorithm works well even for shapes drawn on a lined paper. This is because we set the parameters of adaptive thresholding in a way that minimizes the retention of background lines.

We also notice that even after optimizing the adaptive thresholding process, some long edges had breaks in them.

To fix this, we performed a closing operation by first dilating and then eroding the image. We found out that for our purposes, using a 9x9 box structuring element gave the best results. Figure 2 shows the comparison of binarization with and without closing.

C. Corner Detection and Removal in Polygons (Figure 1c)

There are different ways to detect straight lines such as Hough transform and line detection algorithms such as one suggested by Lagunovsky and Ablameyko [4]. However, we applied a different approach to detect lines (to account for unintentional curves in human drawings). To detect polygons, we first attempt to detect the corners using Harris corner detector. We then attempt to separate the edges in the linear shapes. To do so, we enlarge the detected corners and subtract it from the binarized image. The enlargement of corners ensures complete separation of all the edges in the shapes. We use these isolated sides in the shape detection algorithm as described in the Shape Detection section that follows.

D. Shape Detection (Figure 1d)

Several domain-independent algorithm already exist for detecting different shapes [5].

We detect line segments in the image after corner removal by calculating the eccentricity of the regions in our image. A large eccentricity indicates that the image has a much larger major axis than minor axis, and therefore is very elongated, indicating a line segment. A very small eccentricity indicates an ellipsoidal region, and is stored independently in a list of ellipsoidal shapes.

We implement linear shape detection by iterating over all pairs of sides detected in the image. For every vertex in the pair of sides, we determine whether the corresponding sides likely intersect by finding the intersection point and verifying it's near to two vertices in the pair of sides. We then combine the pair of segments into a partial shape, and as we iterate through all pairs of segments in the image, these partial shapes are eventually completed.

At the end of this step, we then have a list of all the linear and ellipsoidal shapes. The coordinates of these shapes still correspond to the positions in the original captured image.

E. Homography (Figure 1e)

Since our initial image acquisition is by means of a hand-held device, perspective distortion is inevitable. This is due to the angle at which the image is captured. To account for such distortion, we require our users to draw a bounding box surrounding their shapes. The bounding box serves to provide anchoring points in our perspective distortion algorithm. It allows us to use it as a reference frame and rectify other shapes according to it. To detect the bounding box we calculate the area of all the quadrilaterals and assign the quadrilateral with the highest area to be the bounding box.

We use the homography process as described by Tong and Zhang [6]. To begin the homography process, we first rectify the vertices of the bounding box to a rectangle of width of 10 pts and height of 8 pts. The algorithm defined by Tong and Zhang takes as inputs the original and rectified vertices of the bounding box, which we statically define, and provides an equation for transforming the rest of the vertices accordingly to move into the new 10x8 bounding box.

F. Angle Straightening (Figure 1f)

Due to the inherent imperfections in human drawings, we implement an angle correction function. This function takes in the coordinates of all the linear shapes and returns the rectified coordinates. The rectification is done by calculating the angle between two points with respect to x-axis. For this algorithm, we have set the rectification angle to be $15°$. Thus, any edge having an angle between $75°$ and $105°$ is corrected to $90°$ and any edge having an angle between $-15°$ and $15°$ is corrected to $0°$. This process allows for the shapes that were intended to be aligned to the axis to maintain their alignment.

III. Android Application

The Android application was written both as a demonstration of the image processing algorithm and as an example of the potential UI that could wrap the algorithm we developed. We kept both of these goals in mind while designing the application, which led to a user interface that was both as simple as possible (to keep the focus on the image processing), while also supporting the most common user flow (saving several slides and emailing them to oneself).

A. User Interface Flow

The overall flow of the user interface is shown in Figure 3. The flow was very much designed with a specific application flow in mind, and helps the user easily follow its interface patterns:

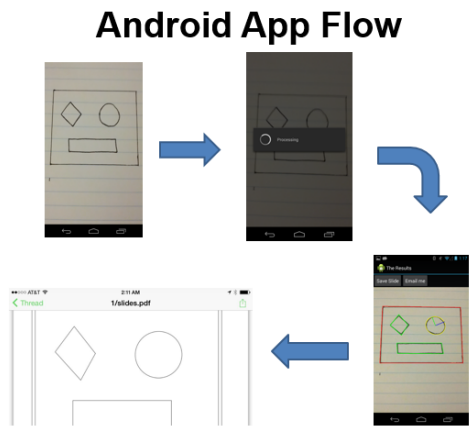

Fig. 3. Various stages of Android application flow.

Users go through the following flow to receive a complete set of slides in their inbox:

1) Launch the Android application.
2) Take a focused photo (the application automatically focuses when the camera is steady).
3) The application then uploads the captured photo and then shows the user the recognized shapes overlaid on their original photo, as seen in the third screen capture. If the system recognized the shapes in the

image, as it should, then the user can tap "Save Slide" to continue capturing more photos, or use Android's back button to return to the capture mode and capture the slide again.

4) After capturing the last slide in the presentation, the user can tap "Email Me."
5) The application prompts the user for their email address, and if they submit theirs, the system will automatically send the collected slides in a PDF to the user's address.

There are two `Activity` instances in the application. The first corresponds to the view where the user is capturing the photo, and the second corresponds to the view where the user accepts or rejects a preview of the shapes recognized by the server.

B. Server Communication

The application communicates with the server in most steps of the application flow, so that the server stays synchronized with the application's state.

For each image captured by the phone, the application sends a `POST` request to `/sample_image/` with the full resolution image data included. The server directly responds with the image of the matched shapes, which can cause a slow delay for this request sometimes. The server remembers the shapes recognized in this image, so if the user next requests the server to save or email their PDF, it is available.

When the user taps "Save Slide," a `GET` request is made to `/save/`. The response of this request is not important, but the action the server takes is: it moves the cached shape data from the "last captured" data into the cache for that user's PDF to be generated later. That is, it saves the shapes from the last captured photo into a list to generate the PDF.

Finally, when the user taps "Email Me," a `GET` request is made to `/email/`. When a request is received at this address, the server takes the same action as for `/save/` (i.e. saving the last captured slide), but then also generates the PDF using TikZ and sends it using Amazon's *SES* (Send Email Service) SMTP server.

IV. Results

Our algorithm was able to detect the shapes that we had planned for (which includes polygons and ellipsoidal shapes). To demonstrate, the variety of images the algorithm is able to process a set of six different images fed into the algorithm. The inputs and outputs are shown in Figure 4.

In our implemention, we only consider closed shapes. Therefore, for any shape for which all the edges are not connected as drawn by the user, the algorithm rejects the shape. Furthermore, if the user has drawn a closed shape but our algorithm fails to detect one of the sides, then similarly the shape is again rejected. This is the cause of the majority of our failures. Also, sometimes the corner removal fails to isolate the edges completely and therefore the eccentricity does not indicate a line segment for that region.

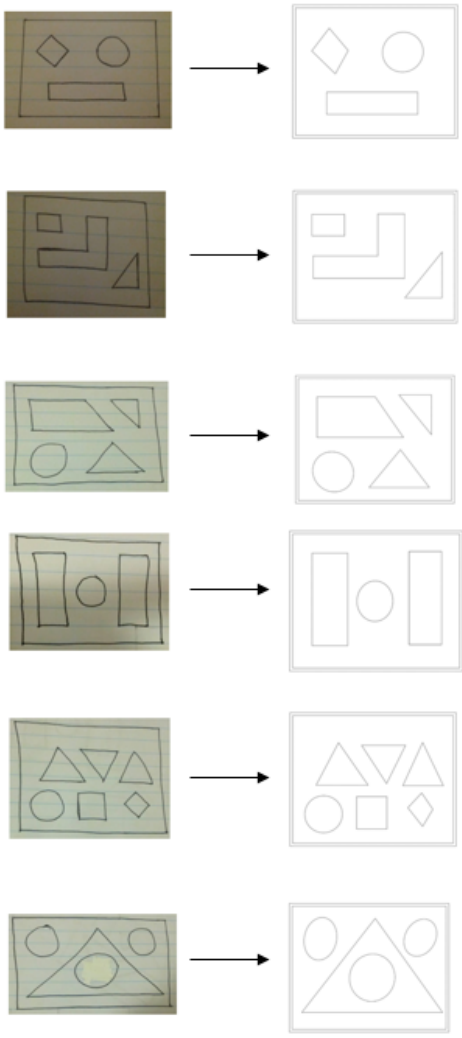

Fig. 4. Output of 6 different images fed into the algorithm. We can see that the algorithm is able to successfully digitilize and rectify the shapes.

V. Conclusion and Further Work

We were able to implement the algorithm to detect polygons and ellipsoidal shapes. However, our algorithm only works when the shapes are not close together due to the way we group the vertices together into shapes. Further work can be done to allow for more shapes to be detected and perhaps altering the shape detection approach to allow for intersecting shapes. With greater confidence in our binarization and corner removal, we could allow for a larger variety of shapes to be detected. This is because our algorithm would reject fewer shapes including line segments. Such algorithms would especially be useful in digitalization of flow-charts which often have arrows connecting rectangles. Not rejecting intersecting shapes would also allow for more complex shape diagrams including digitalizition of hand-drawn circuit diagrams.

VI. Appendix: Contributions

The contributions were made as follows:

- Binarization and closing optimization: Muneeb Ahmed

- Corner detection and removal: Jeff Wheeler
- Shape detection: Jeff Wheeler
- Homography: Muneeb Ahmed
- Angle straightening: Muneeb Ahmed
- Android implementation: Jeff Wheeler

ACKNOWLEDGMENT

The authors would like to thank Sam Tsai for his valuable suggestions on improving the algorithm.

REFERENCES

[1] L. B. Kara and T. F. Stahovich, "An image-based, trainable symbol recognizer for hand-drawn sketches," *Comput. Graph.*, vol. 29, no. 4, pp. 501–517, Aug. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.cag.2005.05.004

[2] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 4, pp. 509–522, Apr 2002.

[3] T. Hammond, D. Logsdon, J. Peschel, J. Johnston, P. Taele, A. Wolin, and B. Paulson, "A sketch recognition interface that recognizes hundreds of shapes in course-of-action diagrams," in *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '10. New York, NY, USA: ACM, 2010, pp. 4213–4218. [Online]. Available: http://doi.acm.org/10.1145/1753846.1754128

[4] D. Lagunovsky and S. Ablameyko, "Straight-line-based primitive extraction in grey-scale object recognition," *Pattern Recognition Letters*, vol. 20, no. 10, pp. 1005 – 1014, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865599000677

[5] K. Refaat, W. Helmy, A. Ali, M. AbdelGhany, and A. Atiya, "A new approach for context-independent handwritten offline diagram recognition using support vector machines," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, June 2008, pp. 177–182.

[6] L. Tong and Y. Zhang, "Correction of perspective text image based on gradient method," in *Information Networking and Automation (ICINA), 2010 International Conference on*, vol. 2, Oct 2010, pp. V2–312–V2–316.