Digitalization of Offline Hand-drawn Flow Diagrams

***Abstract*—The goal of this work is to apply the concepts of perceptual grouping to digitize hand-drawn flow diagrams and render an aestheticized image made of polygons and connectors. We address typical imperfections in human drawing such as missed intersections of lines, overshooting of lines beyond the intersection point and curved line segments. Our method works on camera acquired images of flow diagrams and is able to handle illumination variations, shadows and speckle noise in the input.**

I. Introduction

We often face many situations where we need to present our ideas in the form of flow diagrams, for they are the best representations of dataflow concepts. The easiest way to make a flow diagram is hand drawing on paper using a pen. Therefore, our aim is to process an image of a hand-drawn flow diagram of polygons and will convert it into an aestheticized digital flow diagram. We make use of the Gestalt principles of proximity, continuity, connectedness and closure to recognize polygons connected through lines. Our aim is to convert a hand-drawn image consisting of polygons connected through lines into a digitalised image, while removing the human errors and other kinds of noise. The main feature point of our algorithm is that it makes use of heuristics which reduces time complexity. Human errors such as broken edges, incomplete polygons (missed intersections of line segments), crossing edges (line segments overshooting past the intersection points) and somewhat curved lines which are intended to be drawn as straight lines.

It also completes an incomplete polygon. Many times, while drawing a polygon in a hurry, we do not complete the figure, i.e. edges are open. Since our algorithm first breaks a polygon into set of edges, therefore this human error is automatically rectified. We achieved a high accuracy of 93% correct rendering of the intended convex polygonal shapes on a dataset of more than 500 hand-drawn polygons.

II. Related Work

Previous research in hand-drawn shapes recognition is done by either online or offline approach. Very few research has been done in the field of offline approach. In this paper, we discuss about the offline approach.

For off-line sketch interpretation Notowidigdo and Miller [1] developed UDSI (User Directed Sketch Information) which used heuristic approach to recognize 3 shapes (Circle, Rectangle and Diamond) and arrows. Shapes were recognized using corner detection, then a heuristic filter is applied to recognize unrecognized shapes (i.e. broken edges etc) and finally a greedy elimination algorithm is run which provides an effective filter among the false and true positives. But this work was limited to recognition of only 3 shapes.

For recognition of hand-drawn flowcharts Wioleta Szwoch and Micha Mucha [2], [3] created a Flow Chart Analyzer (FCA) system for recognizing, understanding and aestheticization of freehand drawing flow charts. The recognition algorithm counts similarity measure of a recognized figure and ideal patterns. If a recognized figure does not match with any ideal pattern, then it is considered as line. They used flowgram programming approach to create the flowchart. One of the limitation of this research is that the recognition is limited only to the given ideal polygons. Unlike [1] and [2] our algorithm can recognise every polygon.

Research on offline hand-drawn sketch using images of pen-and-paper diagrams is less common. Valveny and Marti [4] discussed a method for recognizing handdrawn architectural symbols using deformable template matching. They achieve recognition rates around 85%. For digitization of hand-drawn diagrams Regina Altmann [5] implemented generalised Hough Transform to detect the shapes in the flowchart. However the approach worked only for diagrams created with a digital image editor and did not give correct results for camera captured images of hand-drawn diagrams. Processing time of this method was very large and memory requirement was high.

Perceptual organization has been widely used in computer vision to extract 2D and 3D structures and generate descriptions [6], [7]. Cohen and Deschamps [8] used perceptual grouping to find a set of contour curves in 2D and 3D images. Their method could find new complete curves from a set of edge points. We adopt the same philosophy of perceptual organization and develop a computationally frugal method to identify hand-drawn polygons and connectors. The input diagram is broken down into line segments which are then grouped to form polygons. The detailed steps are described in Section III. Experimental results are given in Section IV and the paper is concluded in Section V.

III. Methodology

A. Preprocessing

The image is captured using a smart-phone camera whose camera specification ranges from 8 mega pixels to 21 mega pixels. However prior to processing the input image is rescaled down to width of 850 to improve the time of algorithm. The camera should be focussed onto the page. Sample images can be seen in Fig 1 and Fig 2. We expect our input images to have spatial variations in illumination and thus eliminate the background noises and shadows.

1) The input image is binarized with adaptive thresholding in order to handle spatial variations in illumination. Adaptive thresholding often leads to breaking of long edges and some irregularity in the figure. Therefore, to make the curves smooth, morphological closing is done.
2) Morphological closing operation is applied using a 9×9 structuring element. It may be noted that before

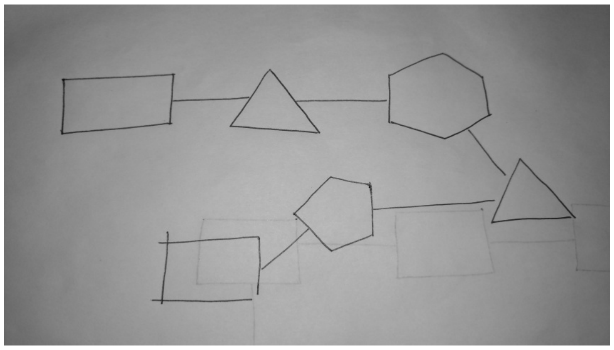


Fig. 1: Sample Input Image. The image consist of many polygonal shapes connected using straight lines. It also contains incomplete polygons and crossing edges with shadows and noise in the background.

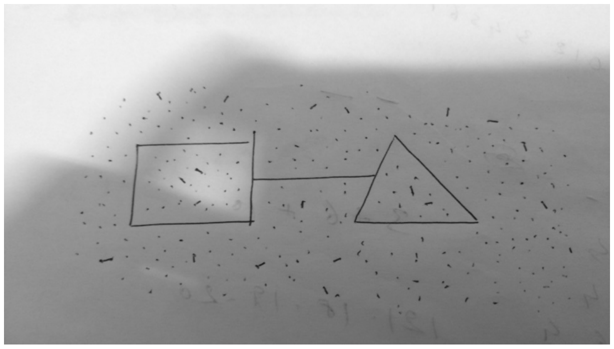


Fig. 2: An input image with shadows and speckle noise

applying the morphological closing step the image needs to be inverted so that the foreground pixels are white.

Fig. 3 shows the result of applying the preprocessing steps on the input image shown in Fig. 2.

B. Extracting straight line primitives

Since hand-drawn flow diagrams can depict wide variations such as missed intersections, extended intersections, etc. we first break down the hand-drawn diagram to get the straight line segments which will be grouped using the subsequent stages. The steps are as follows:

1) We use the Harris corner detector to detect all corners where the angle between the edges is less than 150^o degrees.
2) Each corner is blackened with a 5×5 mask so as to convert all foreground pixels within this mask to background. This has the effect of disconnecting the edges that are incident on the corner.

The diagram is now a set of broken edges as shown in Fig 4. The next step is to model these edge segments as straight line segments. Since the hand-drawn lines can be slightly curved the problem is to get the best approximation of the line segment. This involves i) getting the best fitting line, (ii) getting the best locations for the end points. It may be noted that since we are doing line fitting to every edge separately we do not need the Hough Transform to get the line parameters. Instead we do a simple line fitting to the contour points of the edge. The line fitting procedure minimizes the sum of distances of every contour point to the straight line. To get the best locations for the end points:

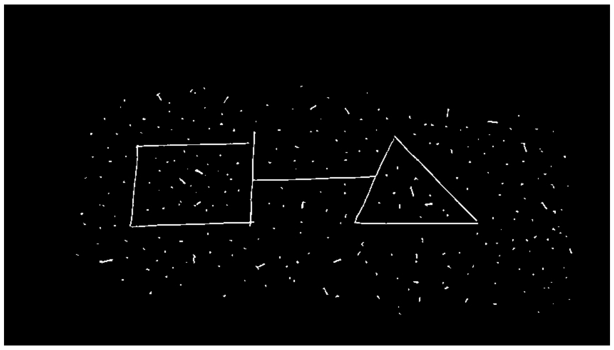


Fig. 3: Inverted image with adaptive thresholding and morphological closing

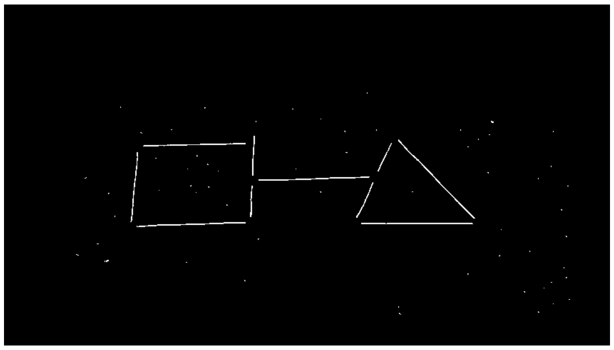


Fig. 4: Final image with minimum noise and corner removed

1) We find the minimum enclosing circle for the contour points. This is the smallest circle which could enclose this contour completely. Following geometrical deductions we can adopt the diameter of the circle to be upper bound on the length of the approximated line segment.
2) The intersection of the smallest enclosing circle and the approximated straight line gives the end points of the approximated line segment.

We observed that the minimum enclosing circle based method of finding the end points of straight line segments gave a better similarity with the area of the hand-drawn polygon compared to the other option of simply projecting the end points of the hand-drawn segment onto the best fitting straight line. The procedure is illustrated in Fig 5. The top image has been obtained after corner detection and removal so it is left with only line segments. The second image shows how our algorithm works, by drawing a circle and an approximate contour with lines and getting it's intersection to draw the straight lines.

We now have a list of all the line segments represented in the form of the coordinates of its two end points.

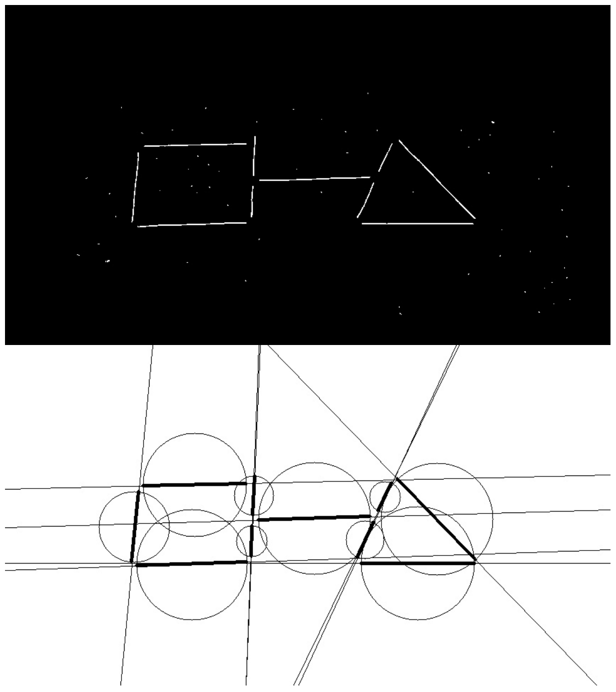


Fig. 5: Line detection

C. Merging broken lines: Grouping based on proximity of end-points

The next step is to generate grouping hypotheses for the line segments and use the best hypotheses to connect the neighbouring line segments. Every pair of line segments generates 4 grouping hypotheses since 4 combinations of end points can be used to generate distance values. A hypothesis is considered valid if the angle formed by the two line segments is less than a chosen threshold and/or the distance between the end points is less than a chosen threshold. The valid hypotheses are examined in the order of increasing distance values and the line segments are merged at the end points corresponding to that hypothesis. Fig 6 shows the output of this step where the broken left side of the triangle is shown to get merged.

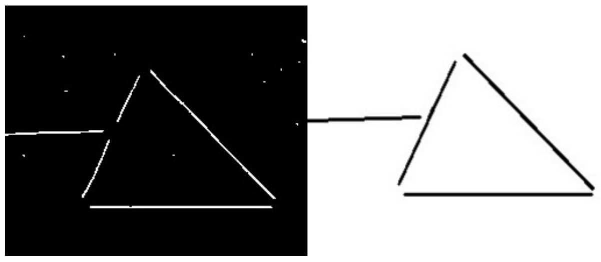


Fig. 6: One of the sides of the triangle is broken. It gets merged after the grouping step.

D. Finding 3 incident lines: Grouping based on proximity of end points

The list of line segments is iterated and this time every triplet of 3 different line segments is considered. A total of 3 line segments means there would be 8 combinations of distances between their end points. Similar to the previous sub section, if the distance between any one of the eight combinations is less than a decided threshold value, then those 3 end points are merged together and this point of intersection of the three respective lines is shifted to the centroid of those 3 points.

E. Finding line intersections: Grouping based on proximity of end points

The point of intersection is calculated for every pair of line segments. Then the distance from the point of intersection to the 4 possible pairs of points is calculated. If any of the 4 combinations gives fruitful value, then it can be inferred that these two line segments were actually intersecting in the original image. After that, as shown in Fig 7, the two end points near the point of intersection are changed into the point of intersection.

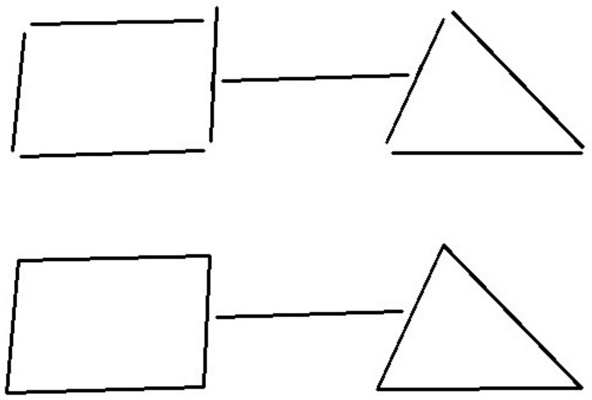


Fig. 7: The algorithm iterates through every line segment and tries to make a closed polygon out of it.

F. Tracing lines to form Polygons: Grouping to find closed regions

We now have a list of all the line segments such that all the discontinuous and intersecting lines are joined together. The line segments are now grouped such that a polygon can be traced with the segments in order. The algorithmic steps are listed in Algorithm 1. The algorithm iterates through every line segment and continues searching for the next line segment adjoining it so as to complete a polygon. If for a given line segment, no such next line segment is found, it would mean that the shape is not a closed polygon. The set of vertices forming a polygon are stored in a list called *vertices*. If this list has n points it means that it is a polygon with $n-1$ sides. The procedure returns a list of all polygons identified.

G. Rendering the Polygons

Each polygon hypothesis is examined to identify the polygon as a triangle, quadrilateral or other regular polygon. We assume that when a user draws a quadrilateral he would like to draw a rhombus, rectangle or a parallelogram. When the polygon has 4 vertices it is recognized to be a quadrilateral and the following steps are followed:

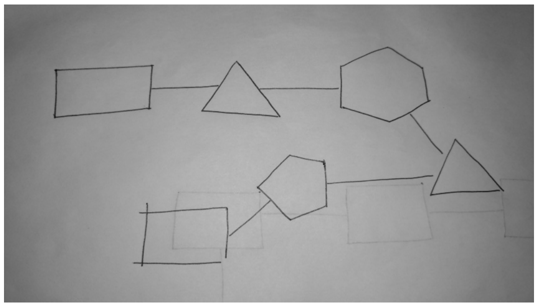
Algorithm 1 Grouping line segments to form closed polygons

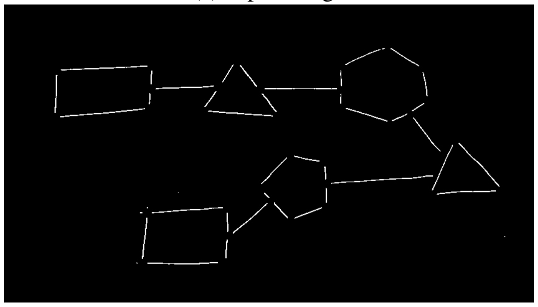
```
procedure GETCLOSEDPOLYGON(line_list)  ▷ lines is an array of line segments
    polygon_list← [ ]                  ▷ List of polygons identified
    while line_list is not empty do    ▷ Consider each line
                                       ▷ to see if it is a part of a polygon
        first← line_list[0]            ▷ first line in line_list
        vertices← {first.x1,first.y1},{first.x2,first.y2}
                                       ▷ Initialize a list of vertices for tracing a polygon
        next_vertex← {first.x2,first.y2}
        line_list.pop()                ▷ First line in line_list is removed
        while next_vertex ≠ {first.x1,first.y1} do   ▷
            flag ← 0                   ▷ flag is set when another line with a
                                       ▷ matching vertex is found
            for each line in line_list do
                v1←{line.x1, line.y1}
                v2←{line.x2, line.y2}
                if (next_vertex == v1) then
                    vertices.append(v1)
                    next_vertex = v2
                    line_list.remove(line)
                    flag = 1
                    break
                if (next_vertex == v2) then
                    vertices.append(v2)
                    next_vertex = v1
                    line_list.remove(line)
                    flag = 1
                    break
            if flag==0 then
                break
            polygon_list.append(vertices)   ▷ Set of vertices
                                            ▷ form a closed polygon
    return polygon_list                ▷
```

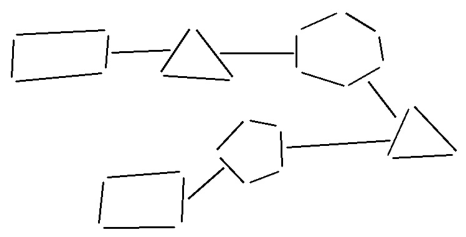
1) The diagonals are constructed and their lengths and the slopes are calculated. If the lengths are nearly the same and the diagonals are approximately perpendicular, then the quadrilateral is a rhombus.
2) If the length of one diagonal is shorter than the length of the other diagonal then the quadrilateral is a parallelogram.
3) Otherwise the quadrilateral is considered to be a rectangle.

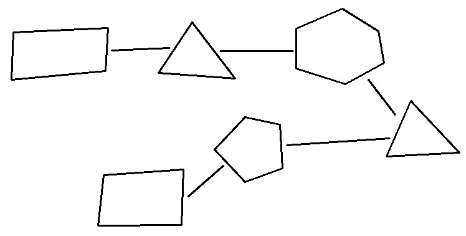
When rendering a parallelogram we have the option of aligning a pair of its sides parallel to the horizontal axis. Polygons with 5 or more vertices are rendered as a regular convex polygon. The centroid of all the vertices is calculated and the average distance of all the current vertices from the centroid is taken as the radius. Taking the vertex just above the centroid as the reference the coordinates of the remaining vertices are re-calculated in clockwise order and a regular polygon is rendered.

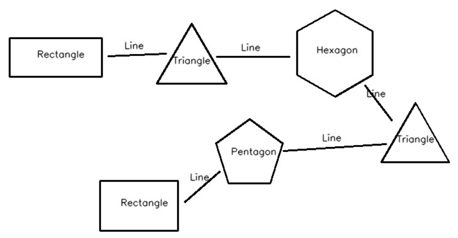
As a final step all the remaining line segments (connectors) are rendered.


(a) Input image


(b) Noise removal using Adaptive thresholding and Morphological closing, Edge detection and removal using Harris corner detector


(c) Line detection and merging of two discontinuous lines to make a single one


(d) Joining the line segments to complete a polygon



(e) Shape detection, recognition and drawing of perfect shape

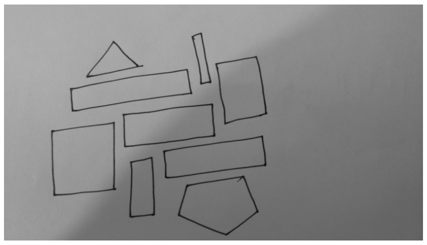
Fig. 8: Steps of the algorithm

IV. RESULTS

In order to evaluate the recognition algorithm, we asked more than 100 subjects to draw 5 polygons of each type. The polygons were, namely, triangle, square, rhombus, rectangle, parallelogram, pentagon, hexagon and few shapes with a line connecting them. Each subject drew the figures on a blank white paper with blue dot pen. No prior information was given to the subjects about the size or shape of the polygon for which our algorithm can work best. Subjects were told that their flow diagram will be used for shape recognition, therefore they did not draw any other non polygonal or open shape.

Our algorithm is able to detect the closed polygonal shapes connected using a line, thus making an aestheticized figure of the hand-drawn flowchart. It is also able to neglect some human errors and rectify them. Some of the salient features of the algorithm are as follows:

Our algorithm is able to recognize polygons from clutter of shapes (see Fig 9b). Shapes close to another shapes are easily separated.


(a) Input test image with clutter

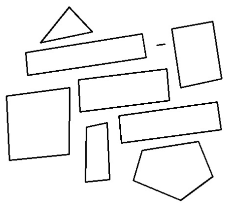
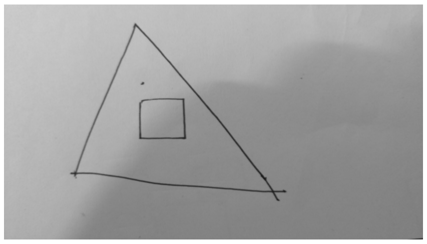

(b) Recognition of shapes in clutter

Fig. 9: Clutter of shapes

As shown in Fig 10b, our algorithm is also able to distinguish shapes nested into other shapes. For example, if a square lies inside a triangle, it is detected separately.

Our generalised algorithm can draw a polygon with any number of sides. It uses a formula to find centroid and takes mean distance of all the vertices from the centroid as radius to draw a regular convex polygon. As shown in Fig 11, it is able to distinguish between a rectangle, square, parallelogram and a rhombus from a given 4 sided polygon.

Of all the test cases given as inputs to our application, we could generate desired outputs for 93 percent of them. When considered specifically for each polygon, we could detect a triangle correctly 97% of the times, and differentiate between different quadrilaterals 93% of the times. Fig 12 and Fig 13 show sample results of digitalizing flow diagrams.


(a) Input test image with square inside triangle

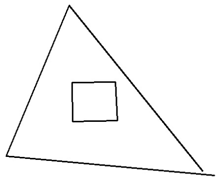

(b) Recognition of both shapes

Fig. 10: Nested Shapes

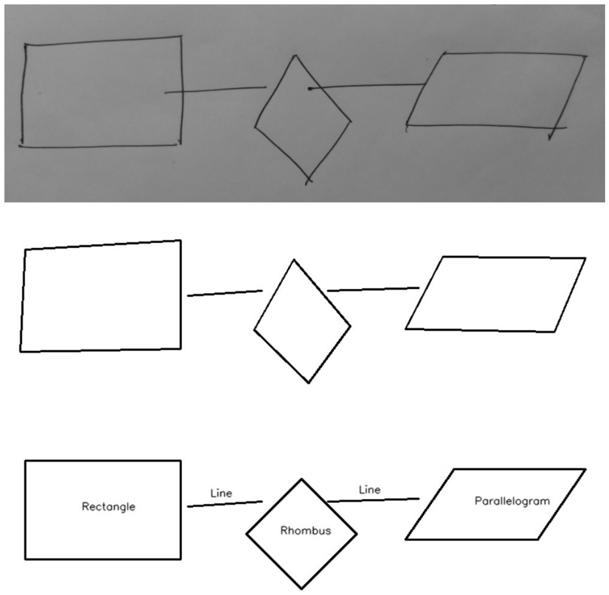


Fig. 11: Different quadrilaterals

V. CONCLUSION AND FURTHER SCOPE

We have proposed a perceptual grouping based method to detect polygons in hand-drawn flow diagrams. Our future work is to incorporate methods to digitize ellipsoid shapes and recognize handwritten text written in the flow charts. Our method has application for digitizing architectural drawings.

REFERENCES

[1] M. Notowidigdo and R. C. Miller, "Offline-sketch interpretation," in *Making Pen-Based Interation Intelligent and Natural. AAAI Fall Symposium*, Menlo Park, California, 2004, pp. 120 – 126.

[2] W. Szwoch and M. Mucha, *Recognition of Hand Drawn Flowcharts*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 65–72.

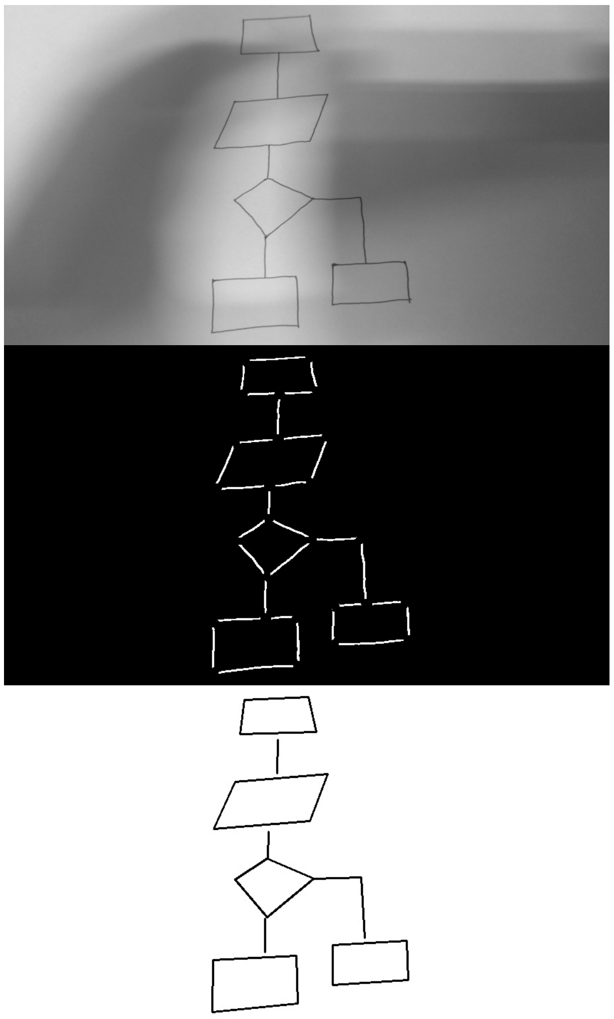


Fig. 12: Sample Result

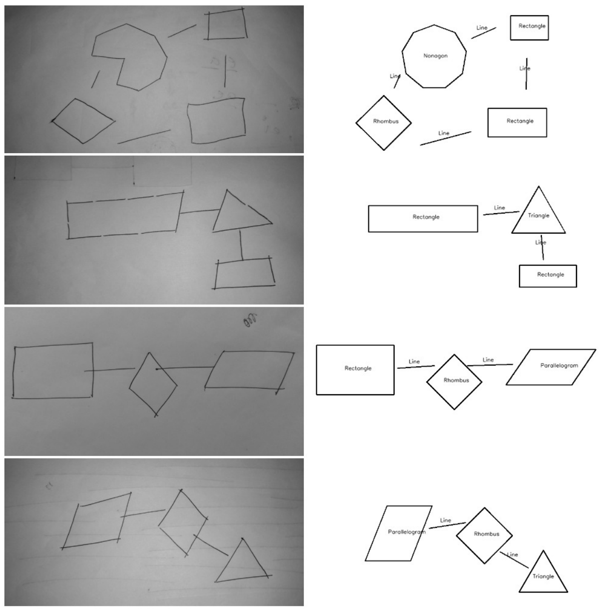


Fig. 13: Sample Results

[3] W. Szwoch, *Aestheticization of Flowcharts*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 423–426.

[4] E. Valveny and E. Marti, "Application of deformable template matching to symbol recognition in hand-drawn architectural drawings," in *International Conference on Document Analysis and Recognition*, 1999.

[5] R. Altmann, "Digitization of hand drawn diagrams," Thesis, Aalto University School of Sicence and Technology, Helsinki, Helsinki, 2015.

[6] G. N. Khan and D. F. Gillies, "Extracting Contours by Perceptual Grouping," *Image and Vision Computing*, vol. 10, no. 2, pp. 77 – 88, 1992.

[7] R. Mohan and R. Nevatia, "Using perceptual organization to extract 3-D structures," vol. 11, pp. 1121 – 1139, Nov 1989.

[8] L. D. Cohen and T. Deschamps, "Grouping connected components using minimal path techniques. Applications to reconstruction of vessels in 2D and 3D images," in *Proc. Computer Vision and Pattern Recognition*, 2001.