Fine-Tuned Line Connection Accompanied Boolean Expression Generation for Hand-Drawn Logic Circuits

Thisaru Dhanushika
Department of Information Technology
University of Moratuwa
Katubedda, Sri Lanka
thisarugalagedara@gmail.com

Lochandaka Ranathunga
Department of Information Technology
University of Moratuwa
Katubedda, Sri Lanka
lochandaka@uom.lk

***Abstract* - Nowadays, electronic circuit designers, students and many who interest in the field of circuit designing are practicing and experimenting about implementing the most accurate and reliable electronic circuits. Therefore, they are fond of doing that task in a short time with less effort. Actually, before going to the development phase of a circuit, they have to understand the problem and need to model the problem. For that, they have to come up with a Boolean Expression to get an idea of the output of the circuit. Currently, most of the people are using online or open-source tools to generate Boolean expression with computer-designed logic gates diagrams. But this is not good at all the time and sometimes the user has to put an extra effort into drawing the circuit in the computer environment. Therefore, as a solution this research paper discussing a way of generating the Boolean expression from a hand-drawn logic circuit diagram using image processing. It mainly helpful for the user to make the input data anywhere at any time without using any of the above tools and can get the final Boolean expression of the hand-drawn logic circuit diagram.**

Keywords- Canny edge, Morphological Transformation, Hough Line Transform, Probabilistic Hough Line Transform, Logic Gates

I. INTRODUCTION

People who are interested in circuit designing are always modelling real-world problems and find easy solutions to address each of them using digital electronic circuits. When identifying the problem with all the possibilities that can happen, it is good to model the problem through a logic gates diagram. Using the logic gates diagram, it is easy to check whether all the cases are addressed with the different combinations of input data. After confirming the logic gates diagram is correct, the user can get the final Boolean expression for the entire logic gates diagram. Then only the user can start modelling the digital electronic circuit. Nowadays, there are much open-source software which useful in circuit designing. But all of the above software use logic gates diagrams which are drawn by using computer graphics. Or at least need a breadboard format of the circuit with all the standard symbols of the components used in circuit designing. This is not a natural process and consumes a lot of time than hand drawing with the need of memorized all the standard symbols that used in drawing logic gates diagram. Additionally, the user has to re-draw the diagram, if the given output is not what is he expected.

This paper proposed a system that can be used to get the final Boolean expression from the hand-drawn input image anywhere at any time without using any of the existing software to generate Boolean expression. The system needs only a camera image and it will detect all the components in a hand drawn logic gate diagram. The hand-drawn logic gate diagram includes seven different types of logic gates. They are AND, NAND, OR, NOR, XOR, XNOR, NOT. And also, it includes different English letters that are used to indicate inputs and outputs. And also, all the logic gates and letters are interconnected with lines. Based on the connectivity among lines can identify T -junctions and L junctions between different line segments.

This paper is organized with related work done in similar research areas, and explains the proposed method of the system. Experimental results of algorithms are also presented. Finally, it provides a conclusion and further works on the study.

II. LITERATURE REVIEW

There are few similar works done related to logic gates detection in a hand-drawn image. Detection of logic gates done in two approaches. They are, feature extraction approach and classification related approach. In the feature extraction approach, shape is the most important feature to identify objects in an image. Some studies use the boundary-based feature extraction method to identify features other than the region-based feature extraction [1]. Under the boundary-based technique, it uses one popular technique called "Fourier descriptor". This helps in detecting the scale, rotation and translation of objects [4]. And also feature extraction can be done with the Scale Invariant Feature Transform algorithm. SIFT technique is used to pattern matching and it is not scaled dependent [2]. Therefore, the expected image can be on one scale and the testing image can be on another scale. Hence, this algorithm further refines the pattern matching accuracy. In the classification related approach, the Support Vector Machine is used to classify the circuit components [3] and [5]. SVM is given training input data and then it analyses those data. Learn from trained data set and prepare a model to classify test data. Here it outputs possible classes in which the input object is belongs. Template matching technique is used to detect letters in a circuit diagram [6]. And to identify lines in an image it used Bresenham's algorithm [8]. This algorithm either calculates X or Y coordinate and identify lines, curves and circles only using incremental integer calculations. Most of the work on this filed is to identify gates, flip flops and other logic symbols. There is no similar work on building the relationships among detected lines and other components in a circuit diagram.

III. PROPOSED METHOD

A. Input Image of Hand Drawn Logic Gates Diagram

The input hand-drawn image needs to be fixed with some specific features that make it easier to process with the proposed methodology. The user has to maintain a considerable gap between logic gates and lines and even with other components like English letters, T-junctions, Nodes. Lines should not cross each other and can use a bridge connection only along y-axis to prevent lines crossing. To indicate a bridge at the end of lines can use a black Node. When drawing the diagram, it can find T-junctions among the

978-1-7281-3706-3/19/$31.00 ©2019 IEEE

interconnected line segments. Those T-junctions can be only along y – axis. English letters can be used to indicate inputs and outputs. Input image is a camera photo and need to maintain high-resolution (about 400PPI) with fixed distance (29cm) to the hand-drawn image and 1x magnification.

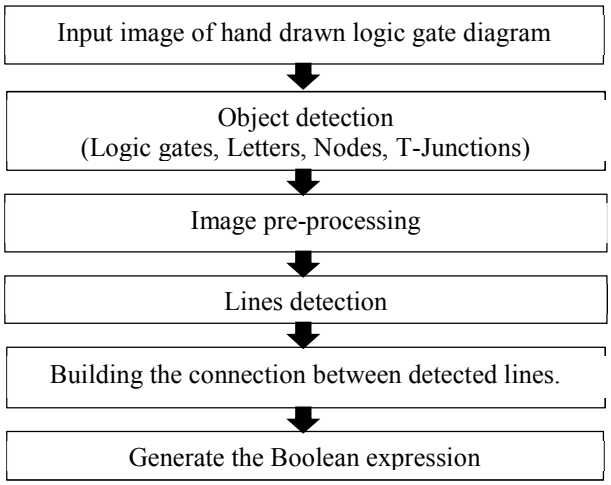

Fig. 1 High level diagram of the system

B. Object Detection (Logic gates, English Letters, Nodes, T-junctions)

Two different approaches used for object detection. They are You Look Once Only (YOLO) system and the open-source framework TensorFlow. A result obtained from the TensorFlow environment can be seen in Fig. 2.

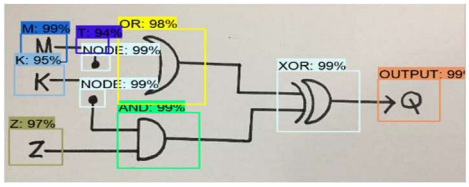

Fig. 2 Objects detection of input hand drawn logic gates diagram

C. Image Pre-processing

First input the raw hand-drawn logic gates diagram, then converts it to a grayscale image. By binarizing this grayscale image finally can get the binary inverse image of the input image.

D. Lines Detection

1) Morphological Operation: This approach always works with binary images and needs two inputs as the original image and the kernel (Structuring element). This Kernel will decide the nature of the operation. Two basic operations include in this approach. They are Erosion and Dilation. Erosion is removing unwanted areas of the image. It may cause to thin the areas where the kernel is running over. Here, if all the pixels are under the kernel then all those pixels become 1, else 0. Dilation is thick the area where the kernel is running over. All the pixels under the kernel will become 1, else 0. This approach can be used to identify Horizontal lines as in Fig.3. (a) and Vertical lines as in Fig.3. (b).

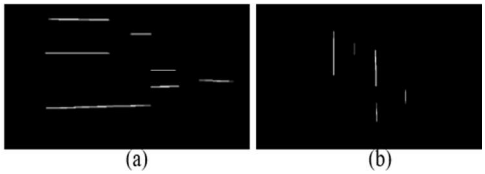

Fig.3 (a) Horizontal lines (b)Vertical lines

2) Hough Line Transform: The second approach for the line's detection was Hough Line Transform. Before applying this operation, it is a must to apply the canny edge detection and the threshold value. The threshold value is the minimum vote, which needs to consider it as a line. The number of votes always depends on the number of points on the line. This method always considers all the points in the line.

3)Probabilistic Hough Line Transform: The third approach for the line's detection was Probabilistic Hough Line Transform. In this approach, it only considers some sets of points in the line other than the Hough Line Transform. These set of points are randomly taken and those numbers of points are much sufficient to detect lines as in Fig.4. Generally, Probabilistic Hough Line Transform applies on the canny edge detected image.

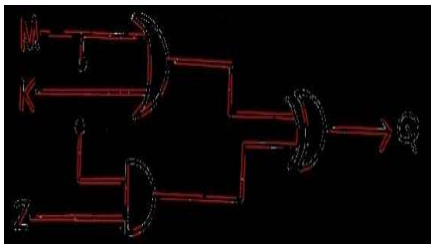

Fig. 4 Lines detection with Probabilistic Hough Line Transform

This approach can be applied without using canny edge detection and can get more accuracy on it. Because of that, coordinates of starting point and ending point in each line segment can be taken most accurately.

E. Building the Relationships among Detected Lines

Before building the relationships between detected line segments, it needs to remove the line segments which are completely inside the bounding boxes since they are making noise to build the relationship among perfectly drawn line segments.

Algorithm I: Removing line segments which are completely inside the bounding boxes

function: RemovingLinesBB(*linesSet, boundingboxesSet*):
Input: ***linesSet*** –all the lines segments detected
boundingboxesSet - all the bounding boxes detected
Output: ***linesSet***– lines which do not have both the end points inside bounding boxes
boundingboxesSet- all the bounding boxes detected

```
1:  for each bounding box in boundingboxesSet do
2:      x1 = top_left_x
3:      y1 = top_left_y
4:      x2 = right_bottom_x
5:      y2 = right_bottom_y
6:      for each line in linesSet do
7:          point_x1 = start_point_x
8:          point_y1 = start_point_y
9:          point_x2 = end_point_x
10:         point_y2 = end_point_y
11:         if the two end points of each line
            completely inside considering bounding box
            then Remove that line from linesSet
12:         endif
```

Return: ***linesSet, boundingboxesSet***

And also, it needs to remove the line segments which have the same starting point or ending point of another line. Because, in this case, it causes duplicate lines for one drawn line segment. So, it is a barrier to create the final Boolean expression.

Algorithm II: Removing duplicate line segments which has same starting point or ending point of another line.

function: RemovingDuplicateLines(*linesSet):*
Input: ***linesSet*** –all the lines segments detected
Output: ***linesSet***

1: **for** *each line in linesSet* **do**
2: x1 = start_point_x
3: y1 = start_point_y
4: x2 = end_point_x
5: y2 = end_point_y
6: **if** the (x1, y1) or (x2, y2) is finds in another line segment **then** ***Remove*** *that line* **from** *linesSet*
7: **endif**

Return: ***linesSet***

Algorithm III: Grouping Nodes which are closed to each other

function: GroupingNodes (*nodesList*):
Input: ***nodesList*** –all the bounding boxes of Nodes
Output:***newNode***–list of pair of nodes

1: Initialize *newNode* **List –** empty list
2: **for** *each Node* in *nodesList* **do**
3: x1 = top_left_x
4: y1 = top_left_y
5: x2 = right_bottom_x
6: y2 = right_bottom_y
7: **if** one of the node in the nodesList is inside the rectangle box (x1-100,y1-200,x2+100,y2+200) **then** ***Add*** *relevant two nodes to newNode*
8: **endif**

Return: ***newNode***

When creating virtual rectangular boxes in any of the following algorithms, defined pixels values are taken as an average considering with the space that the user needs to maintain between every component. In each scenario test the accuracy of the algorithms that use virtual boxes by changing those pixels values until they get the maximum accuracy and finally get the most accurate pixel range for creating virtual rectangular boxes.

According to the above algorithm, all the Nodes that are closed to each other are making pairs and grouped them as lists of pairs. This can be done by making a virtual rectangle box based on one Node of a bridge connection as in Fig.5.

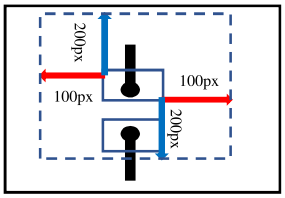

Fig. 5 Explanation of grouping Nodes closed to each other

Algorithm IV: Grouping T-Junctions with its horizontal line

function: TjuncWithHorizontal (*linesList, tnodesList*):
Input: ***linesList*** –all the line segments detected
tnodes*List* –all the bounding boxes of T- junctions
Output: ***juncBaseLine*** –list of pair of T-junctions with its horizontal line

1: Initialize *juncBaseLine* **List –** empty list
2: **for** *each extracted line in linesList* **do**
3: x1 = start_point_x
4: y1 = start_point_y
5: x2 = end_point_x
6: y2 = right_bottom_y
7: new_y1 = y1-100
8: new_y2 = y2+100
9: Draw a rectangle box (x1, new_y1,x2,new_y2)
10: **for** each bounding box of tnodesList **do** find the Bounding box of T-junction which is completely inside the above rectangle and **Add** both the bounding box of T-junction and particular line to *juncBaseLine*

Return: ***juncBaseLine***

According to the above algorithm, all the bounding boxes of T-junctions and the Horizontal line segments of that T-junctions will be grouped. To group, those two elements create a virtual rectangular box from the coordinates of two end points of the horizontal line of the T-junction. Then that particular T-junction with its relevant Horizontal line segment will be grouped as in Fig.6.

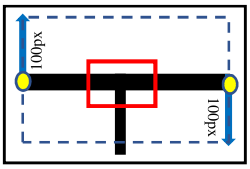

Fig. 6 Explanation of grouping T-Junctions with its horizontal line

Algorithm V: Grouping starting point of Horizontal line with ending point of vertical line in T-junction.

function: addBasePoint (*linesList, juncBaseLine*):
Input: ***linesList*** –all the line segments detected
juncBaseLine –list of pair of T-junctions with its horizontal line
Output: **addBasePoint** –lists of groups of starting point of horizontal line with ending point of vertical line in T-junction.

1: Initialize *addBasePoints* **List –** empty list
2: **for** *each items in juncBaseLine* **do**
3: x11 = start_point_x_of_horizontal-line
4: y11 = start_point_y_ of_horizontal-line
5: x1 = top_left_x_of_T-junction
6: y1 = top_left_y_of_T-junction
7: x2 = right_bottom_x_T-junction
8: y2 = right_bottom_y_T-junction
9: **for** *each lines in linesList* **do**
10: Find the line segment from linesList which is act as the vertical line segment of above T-junction. **then Add** the (x11,y11) with the ending point of vertical line segment which is not inside the T-junction

Return: **addBasePoints**

After that can remove the Vertical line segment of T-junctions from the main sets of lines detected.

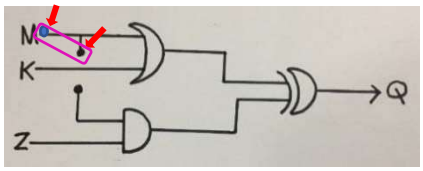

Fig. 7: Explanation of grouping starting point of Horizontal line with ending point of vertical line in T-junction

Algorithm VI: Grouping closest line segments

function: Group lines(*linesList*):
Input: ***linesList*** –all the lines segments detected
Output: ***finalPoints*** – sets of lines which are grouped

1: Initialize *finalPoints* **List –** empty list
2: **for** *each lines in linesList* **do**
3: point_x1 = start_point_x
4: point_y1 = start_point_y
5: point_x2 = end_point_x
6: point_y2 = end_point_y
7: Draw a rectangle box (point_x1-23,point_y1-23, point_x1+23,point_y1+23)
8: Draw a rectangle box (point_x2-23,point_y2-23, point_x2+23,point_y2+23)
9: **If** one of the end point of any line segment in linesList is inside the one of above rectangle **then**
group those line segments and
Add to finalpoints
Remove from linesList
10: **endif**

Return: ***finalPoints***

According to the above algorithm, we need to consider endpoints of each line segment. If one endpoint of a line segment is within a small range of both the x-axis and y-axis of the one of the endpoints of another line segment as in Fig.8, then it can be grouped as those two-line segments are closed to each other. This case will be true to all the L-junctions between the line segments. And finally, we can add, single lines that do not make a link with another line also to the final results of the above algorithm.

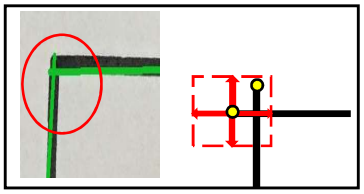

Fig. 8 Explanation of grouping line segments which make L - Junctions

With the limitations of the input image, the user can go through three drawing patterns. Following algorithms explain the handling of connectivity among line segments according to three types of drawing patterns.

Algorithm VII: Pattern1, vertical line of T junction is connecting with a normal line segment

function: pattern1 (*add Basepoints, finalPoints*):
Input: ***addBasepoints*** –lists of groups of starting point of Horizontal line with ending point of vertical line in T-junction
finalPoints –all the sets of lines grouped
Output: ***removeBasePoints*** –lists of lines that need to remove from addBasePoints

1: Initialize *removeBasePoints* **List –** empty list
2: **for** *each points* in *addBasePoints* **do**
3: x1 = start_point_x
4: y1 = start_point_y
5: x2 = end_point_x
6: y2 = emd_point_y
7: **for** *each lines in finalPoints* **do**
8: Draw a rectangle box for each end point of all the lines in sets having 30 pixels range for both the x axis and y axis
9: **if** any of the rectangle box includes the(x2,y2)
then
Add the line segment (x1,y1,x2,y2) to relevant list in **finalPoints**
Remove line(x1,y1,x2,y2) **from addBasePoints**
endif
10: **Return:** ***removeBasePoints***

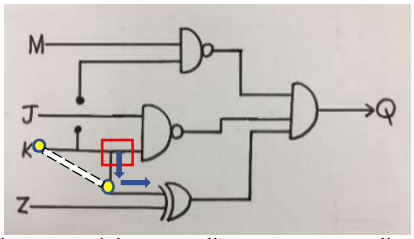

Fig. 9 Handling connectivity among line segments according to pattern 1

Algorithm VIII: Pattern2, vertical line of T junction is connecting with a node of a bridge connection (upward y-axis or downward y-axis)

function: pattern2 (*add Basepoints, newNode*):
Input: ***addBasepoints*** –lists of groups of starting point of Horizontal line with ending point of vertical line in T-junction
newNode –all the sets of grouped Nodes
Output: **new-item** – lists of pair Nodes and single Nodes
newNode –all the sets of grouped Nodes

1: Initialize *new-item* **List –** empty list
2: **for** *each line* in *addBasePoints* ***do***
3: x1 = start_point_x
4: y1 = start_point_y
5: x2 = end_point_x
6: y2 = emd_point_y
7: **for** *each pairNodes* in newNode **do**
8: **if** any of the pairNodes contains the (x2,y2) ending point **then**
Remove the ***Node from*** *pairNode*
Add starting point x1,y1 & remaining Node of the pair **to new-item**

Return: ***new-item***

// new-item has pairs which has 1 or 2 Nodes in one list with starting point of horizontal line in T-junction//

Input: finalPoints –all the sets of lines grouped
9: **for** each *items* in *new-item* **do**
10: x11 = start_point_x
11: y11= start_point_y
12: x1 = top_left_x
13: y1 = top_left_y
14: x2 = right_bottom_x
15: y2 = right_bottom_y
16: **for** *each lines* in finalPoints **do**
17: **if** one of the end points of a line inside the rectangle (x1,y1,x2,y2) **then** **Add** (x11,y11) with that matching point **to** finalPoints **endif**

Return: ***finalPoints***

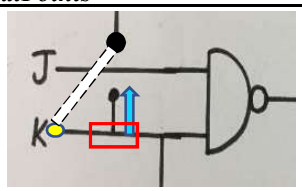

Fig. 10 Handling connectivity among line segments according to pattern 2 (T-junction is downward of y-axis)

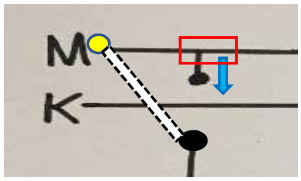

Fig. 11 Handling connectivity among line segments according to pattern 2 (T-junction is upward of y-axis)

Algorithm IX: Pattern3-only couple of Nodes and no T-junctions

function: pattern3 (*add Basepoints, newNode*):
Input: newNode –all the sets of grouped Nodes (Remaining pairs with 2 Nodes)
Output: **finalPoint**s –all the sets of lines grouped
1: **for** *each pairNodes* in newNode **do**
2: **for** *each sets of lines in finalPoints* **do**
3: **Find** two lines sets in finalPoints which has ending points within each Node in the pair **Add** two-line segments in to one set in finalPoints and remove the above two separate line segments from the **finalPoints**

Return: ***finalList***

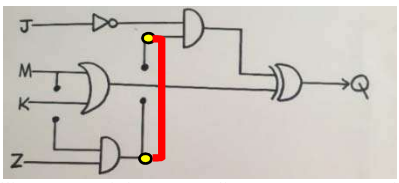

Fig. 12 Handling connectivity among line segments according to pattern 3

Algorithm X: Generating the Boolean Expression

function: booleanExpression (*linesList,gatesNodes*):
Input: ***finalPoints List*** – all the sets of lines grouped gateNodes **List**–all the bounding boxes of logic gates
Output: ***String output*** – Boolean Expression
1: Initialize *sorted_lines* **List –** empty list
2: **for** *i* in *linesList*
3: connected = i[0]
4: not_connected = i[1:]
5: **while not_connected**
6: Sort the line segments in i using Euclidian Distances and
7: **Add** to sorted_lines
8: Return sorted_lines
9: Calculate the centre points of each bounding box of logic gates
10: For b in linesList
11: Startpoint = b [0]
12: Endpoint = b[1]
13: **for** *k* in gatesNodes
gate1 = for node in gatesNodes if strat_point in node
gate2 = for node in gatesNodes if strat_point in node
14: **if center for** startpoint < endpoint
15: Sourcegate = gate1
16: Destgate = gate2
17: **else**
18: sourcegate = gate2
19: destgate = gate1
20: destgate.append (sourcegate)

Return: ***output***

According to the above algorithm, identified lines for each input and output of the logic gates from the left side to the right side of the image need to be sorted and need to find the gates connected to each end. Select the leftmost as the source gate and the rightmost as the destination gate. Update the inputs of the destination gate and the outputs of the source gate. At last print the output of the rightmost gate as the final Boolean Expression.

IV. EXPERIMENTAL RESULTS

In this section, the complete set of results of carried out experiments are presented to exhibit the ability and performance of the proposed algorithms with different user drawn logic gates diagrams. Total of 300 datasets of different hand-drawn logic gates diagrams of different users collected and among them 240 images taken as the training dataset and 60 images taken as the test dataset. Detection is done with YOLO and TensorFlow environments. The accuracy of detecting objects using YOLO is 40% and TensorFlow is 90%.

Since all of the algorithms are based on coordinates of the Cartesian plane, can evaluate each algorithm with manual dataset or by using coordinates that are taken from images with perfectly detected objects and lines. All of the algorithms are rule-based and, all of them straightforwardly give 100% accuracy. When it comes to evaluating the system accuracy, presented algorithms are based on the lines which

are detected by using Probabilistic Hough Line Transform. So, the accuracy of the algorithms is entirely based on the lines that are detected by that approach. Altogether, object detection and line detection steps need to be more accurate for the final result. Because if one of the objects is not detected or when the one-line segment is not detected perfectly, then it affects the final Boolean expression and can generates a wrong Boolean expression. Therefore, it is needs to analyse the error rate of object detection and line detection by making use of 40 hand-drawn logic circuits diagrams from different users. And also need to analyse the error rate of joints detection based on the lines detected by the Probabilistic Hough Line Transform as in Table. I.

TABLE I
PRECISION, RECALL AND ACCURACY ANALYSIS OF OBJECTS, LINES AND JOINTS DETECTION

	478 Objects	394 lines	219 L-junctions
True Positive	474	374	196
False Positive	47	48	0
False Negative	4	28	23
Precision	0.909788	0.88625	1
Recall	0.991631	0.93034	0.8949
Accuracy	90.28571%	83.11111%	89.4977%
Error Rate	9.71428%	16.88888%	10.5022%

Objects need to be detected with their belonged classes and lines need to be drawn on the user drawn lines. Based on line detection, need to evaluate the grouping of lines with identified joints between line segments. According to the above results, the total error rate is 37.12%. Therefore, system accuracy is highly depending on the accuracy of objects and lines detection. It needs to minimize the error rate in detecting objects and lines. Finally, it can be evaluating the deviation of generated Boolean expression from the expected Boolean expression using 40 images as in Fig.13.

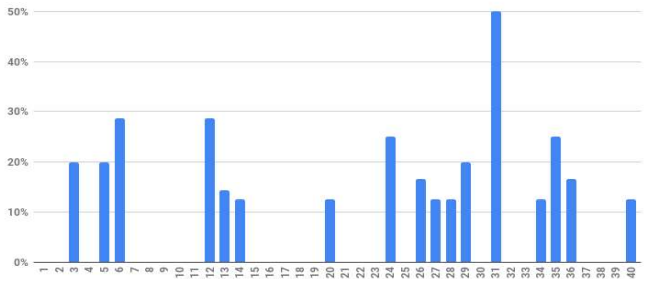

Fig. 13 Deviation of generated Boolean expression from expected result

According to the above graph its analysis the deviation of expected Boolean Expression due to the above-discussed error rate. The Deviation is the number of terms in the generated Boolean Expression to the number of terms in the actual Boolean Expression as a percentage. Therefore, it is clear that every single component of the image needs to be detected in a perfectly to get an accurate result of this proposed methodology.

V. CONCLUSION AND FURTHER WORKS

In this paper, a methodology is proposed to generate the Boolean expression from a hand-drawn logic gates diagram. The proposed method is mainly based on object detection and lines detection. Therefore, to get an accurate output it is a must to detect all the objects and lines in the input image perfectly. Then only, relationships among lines can be taken in a perfectly. As a result, it needs to fine-tune the above two main stages. When making the connection between lines, need to grouped lines that are closed to each other. In this case, used the neighbourhood analysis of two end points of each line segment. But this proposed algorithm may not work for all the cases since the considering gap between two endpoints may not be enough or sometimes it is too much. Therefore, it needs to improve this algorithm to match with any of the given input data. And also, when grouping Nodes used the neighbourhood analysis and grouped the closest Nodes together. But in any case, it can be failed due to exceeding the range which is defined to group them as one pair.

When handling T junctions, it always separates one path into two or many paths. But here consider the T-junctions along the Y-axis only. Therefore, it can be improved to have T-junctions in both the X-axis and Y-axis. And also, when considering the algorithm of grouping Nodes, it is done for the Nodes which are along with Y-axis only. Here also it can be improved to have Nodes along the X-axis and Y-axis. Not only that but also it can be improved to deals with complicated bridges. Therefore, two-line segments that are more far away to each other can be linked with more than one bridge connection.

As further development, it can be used to analysis lines that are not straight and drawn in a free hand. Therefore, in future, it is better to develop a model that can generate the Boolean Expression from the free hand-drawn images. Because of that users can achieve the maximum benefits regarding circuits designing.

REFERENCES

[1] "Detection and identification of logic gates from document images using mathematical morphology - IEEE Conference Publication", *Ieeexplore.ieee.org*, 2018. [Online]. Available: http://ieeexplore.ieee.org/document/7490040/. [Accessed: 12- Sep- 2018].

[2] [*Stacks.stanford.edu*, 2018. [Online]. Available: https://stacks.stanford.edu/file/druid:yt916dh6570/Jagasivamani_Recognition_of_Digital_Logic_Circuits.pdf. [Accessed: 12- Sep- 2018].

[3] *Ijcaonline.org*,2018.[Online]Available: https://www.ijcaonline.org/archives/volume143/number3/patare-2016-ijca-910058.pdf. [Accessed: 14- Sep- 2018].

[4] Pdfs.semanticscholar.org.(2019).[online]Available at: https://pdfs.semanticscholar.org/5d22/b9113739b3b6217aa603f48e5cd3caf6fb64.pdf [Accessed 2 Mar. 2019].

[5] Anon, (2019). [ebook] Available at: http://eudl.eu/pdf/10.4108/eai.13-4-2018.154478 [Accessed 2 Mar. 2019].

[6] "Optical Character Recognition By Using Template Matching (Alphabet)", *Academia.edu*, 2018. [Online]. Available: http://www.academia.edu/714194/Optical_Character_Recognition_By_Using_Template_Matching_Alphabet_. [Accessed: 19- Sep- 2018].

[7] Anon,(2019).[ebook]Available at: https://brage.bibsys.no/xmlui/bitstream/handle/11250/250824/347986_FULLTEXT01.pdf?sequence=2&isAllowed=y [Accessed 2 Mar. 2019].

[8] Anon,(2019).[ebook]Available at: http://www.ijrst.com/images/short_pdf/1426575098_kislay_anand.pdf [Accessed 2 Mar. 2019].