

Symbol/Object Detection and Recognition in Electrical Circuit Sketches

Master Thesis report

Submitted in partial fulfilment of the requirements for the
degree of

Master of Technology

by

Saurabh Jain

Roll no. 143050048

under the guidance of

Prof. Abhiram Ranade

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai



Acknowledgement

I would like to express my greatest gratitude to **Prof. Abhiram Ranade**, for giving me aspiring guidance and friendly advice, for this Project work throughout the numerous consultation. I am also thankful to him, for his timely suggestions and constant motivation which helped me a lot in the completion of this phase of Project.

Saurabh Jain

Abstract

Sketches are used to express and represent ideas in various domains such as machine drawing in mechanical engineering, circuit drawing in an electrical drawing, architectural and Software design. Increased use of Touchscreen devices such as smart phones and tablets have made sketches, a very popular form of human-computer Interaction. In this report, we propose an improvement on symbol recognition in electrical circuits sketches drawn in DrawCAD.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 8 |
| 1.1 | Motivation | 8 |
| 1.2 | DrawCAD | 8 |
| 1.2.1 | Primitive segment recognition | 9 |
| 1.2.2 | Constraints identification and its satisfaction | 9 |
| 1.2.3 | Sketch-symbol/Sketch-object recognition | 10 |
| 1.3 | Related Work | 11 |
| 1.4 | Work Done | 12 |
| 1.5 | Outline of report | 12 |
| 2 | Object/Symbol Detection | 13 |
| 2.1 | Detection of the Candidate Symbol | 13 |
| 2.2 | Desired Improvements | 15 |
| 2.3 | Modified Detection algorithm | 17 |
| 2.3.1 | Definition of Ink density | 17 |
| 2.3.2 | Bounding box definition. | 18 |
| 3 | Object/Symbol Recognition | 20 |
| 3.1 | Classification of Detected Symbol | 20 |
| 3.1.1 | Feature-Set | 20 |
| 3.1.2 | Normal Probability Distribution Matching | 21 |
| 3.2 | Least-Square Matching | 22 |
| 3.3 | Desired Improvements | 23 |
| 3.4 | Modified Recognition process | 24 |
| 3.4.1 | Changing feature set | 24 |
| 3.4.2 | Training modification | 24 |
| 3.4.3 | Matching value modification | 25 |
| 3.4.4 | Changing Bounding Box threshold | 25 |
| 4 | Segmentation | 26 |
| 4.1 | Segmentation | 26 |
| 4.2 | Segment Point Detection Scheme | 27 |
| 4.2.1 | Direction | 27 |
| 4.2.2 | Curvature | 27 |
| 4.2.3 | Speed | 28 |
| 4.2.4 | Existing Method: Segmentation | 28 |
| 4.2.5 | Drawback in Previous Segmentation Implementation | 29 |

| | | |
|----------|--|-----------|
| 5 | Testing | 31 |
| 5.1 | Test-case Analysis for Symbol recognition | 31 |
| 5.1.1 | Previous System Performance Analysis | 32 |
| 5.1.2 | Least-square Approach Performance Analysis | 32 |
| 5.1.3 | Current System Performance Analysis | 33 |
| 6 | Additional Features | 35 |
| 6.1 | Functionalities for an Object/Symbol | 35 |
| 6.2 | Segmentation Mode | 35 |
| 6.2.1 | One stroke per segment (OSS) | 35 |
| 6.2.2 | Multiple stroke per segment(MSS) | 36 |
| 6.3 | Immediate Recognition | 37 |
| 7 | Conclusion and Future Work | 38 |
| 7.1 | Conclusion | 38 |
| 7.2 | Future work | 38 |
| 7.2.1 | Use Multivariate Normal Probability Matching | 38 |
| 7.2.2 | Fixing bugs and Adding functionalities | 39 |
| 7.2.3 | Global Beautification | 39 |
| 7.2.4 | Symbol predictor | 40 |
| 7.2.5 | Fuzzy classification of primitives | 40 |
| 7.2.6 | Domain Independence | 41 |
| 7.2.7 | Solver to the circuit Diagram | 41 |
| A | Description of Architecture of DRAWCAD | 42 |

List of Figures

| | | |
|------|--|----|
| 1.1 | Primitive segment recognition (a) as line and (b) as circular arc. . . | 9 |
| 1.2 | Hard constraints | 10 |
| 1.3 | Symbol recognition in the electric circuit | 11 |
| 2.1 | Symbol detection using temporal ordering in resistor [5] | 14 |
| 2.2 | Forward algorithm to detect end point of candidate symbol[5] . . . | 14 |
| 2.3 | Backward algorithm to detect end point of candidate symbol[5] . . | 15 |
| 2.4 | Noise while drawing a resistor | 15 |
| 2.5 | Noise tolerance in the identification of NPN(a), capacitor(b) and battery symbol(c) | 16 |
| 2.6 | Resistor can have any number of crest and troughs. | 16 |
| 2.7 | Inductance drawn in different order. | 17 |
| 2.8 | Hidden ink in while drawing capacitor. | 18 |
| 2.9 | Bounding box alignment on capacitor symbol | 18 |
| 2.10 | Oriented bounding box | 19 |
| 3.1 | Incorrect recognition of two different object to same class | 23 |
| 3.2 | Incorrectly recognized NAND gate and Inductor | 23 |
| 3.3 | Incorrect recognition of capacitor | 25 |
| 4.1 | Segmentation example | 26 |
| 4.2 | Challenge in the segmentation [5] | 27 |
| 4.3 | Merging of two consecutive segment [5] | 29 |
| 4.4 | Failed TestCases in Segmentation | 30 |
| 5.1 | Symbol used for testing | 31 |
| 5.2 | Arbitrary ordering of segments | 33 |
| 5.3 | A small line segment between the two curve strokes while drawing inductor | 34 |
| 5.4 | Ambiguity in identifying symbol as NAND gate or NOT gate . . . | 34 |
| 5.5 | Result of drawing various strokes in MSS mode | 34 |
| 6.1 | Result of Drawing various strokes in OSS mode | 36 |
| 6.2 | Result of drawing strokes in MSS mode | 36 |
| 7.1 | Global beautification. | 39 |
| 7.2 | Symbol prediction | 40 |
| A.1 | Screen shot of DrawCAD | 42 |

List of Tables

| | | |
|-----|--|----|
| 5.1 | Performance analysis of previous approach | 32 |
| 5.2 | Performance analysis of least-square approach | 32 |
| 5.3 | Performance after applying modification in recognition process . . . | 33 |

Chapter 1

Introduction

1.1 Motivation

Sketches play an important role in human visual perception and are the primary way of communicating and representing information for humans. With the increasing availability of pen-based computers, sketch recognition has gained attention as an enabling technology for natural pen-based interfaces.

Sketches form a basis for a variety of computer vision problems such as Sketch-based image retrieval, Sketch-based shape retrieval, Sketch detection, and recognition etc. Sketch recognition has a variety of applications in domains such as electric circuit diagrams, math sketches, molecular diagrams etc. Recognizing shapes in these sketches is a challenging task, due to the imprecision with which they are drawn. A sketch based electrical circuit diagram recognition system has been developed in IIT Bombay. Our goal is to improve it and extend it.

Section 1.2 gives a brief description of DrawCAD project and some of the previous contributions to this project. Also, section 1.2.3 gives a brief overview of the problem definition we are trying to solve in the first phase of the project. Section 1.3 illustrates related work done towards solving this problem. Section 1.4 gives the outline of the report.

1.2 DrawCAD

Our recognition system is build using DrawCAD(Version 5 of Vivek) as the platform which is an IIT Bombay project for engineering drawing. It is different from a typical menu-based tool. It has a primitive identification module to recognize raw strokes drawn by a user as a point, line or circular arc. The basic problem description of the project is to develop a powerful design tool that can draw a 2-Dimensional diagram by free-hand drawing, which is more intuitive and natural way of drawing. Other Drawing tools require typical menu-driven, toolbar-driven or command-driven interface which is non-intuitive and time-consuming. Following are the three major aspects of DrawCAD:

- Primitive segment recognition
- Constraint identification and its satisfaction
- Sketch-symbol/Sketch-object recognition

1.2.1 Primitive segment recognition

This problem is formally stated in [6] as:

Given a sequence of mouse drawn strokes, where each stroke is a sequence of coordinate points with timing information (x,y,t) , recognizing the constraints and the primitive segment intended by the stroke and replacing the stroke by a precise form of recognized primitive segment. This problem also involves editing of strokes.

In other words, a raw stroke is beautified to a corresponding primitive segment. We now define the basic inputs and outputs of the system. A *Stroke* is the array of sampled points between a mouse press (pen-down) and mouse release (pen-up) event. Thus, the input is sequence of sample points $\langle x, y, t \rangle$, where t is the time stamp of the sample point $\langle x, y \rangle$. And we have three primitive segments as output, which are points, lines are circular arcs.



Figure 1.1: Primitive segment recognition (a) as line and (b) as circular arc.

When a stroke is drawn we need to find the closest fit of the stroke to these primitives. For each type of fit, a confidence value is returned. Currently, this confidence value is just the negation of the error value, where error value is the sum of least square of the pixel position of the raw segment. Suppose a stroke is identified as a line, we need to decide what are its end points and middle point with respect to the stroke drawn. Similarly, if a stroke is identified as a circular arc, we need to decide its end points, the center of the circle of which arc is a part and radius of the arc. Other higher degree polylines such as spline can also be fitted, but presently, the system is restricted to only the above-defined primitives. This preprocessing also involves identification of the Markers that explicitly defines the hard constraints, which user might want impose. The type of markers such as equal length constraint marker or parallel line constraint marker is recognized during this process. This is discussed in more detail in next subsection.

1.2.2 Constraints identification and its satisfaction

A wide range of constraints is supported in DrawCAD, are commonly useful while drawing sketches. Two basic categories of constraints are:

- *Soft constraints:* These are the constraints that are implicitly derived from the user drawing
- *Hard constraints:* These are the constraints that are explicitly specified by the notation we used in the high school geometry and required to detect in a manner similar to the regular strokes in the drawing.

Constraints can also be classified as independent constraints and relative constraints. Independent constraints such orientation of a line, either horizontal or

vertical. Another type of constraints that are relative constraints describes the relation between different segments. Currently, following are the relative constraints in our program:

1. Relative length
2. Parallel lines
3. Perpendicular lines
4. Intersection
5. Tangency
6. Cocentricity
7. Points collinearity

To explicitly enforce the hard constraints, user need to add some kind of marker so that our system can identify the relation between the segments. Based on these relations, feasible constraints are enforced in the diagram. Currently, supported markers are equal angle, equal length, parallel and perpendicular line markers.

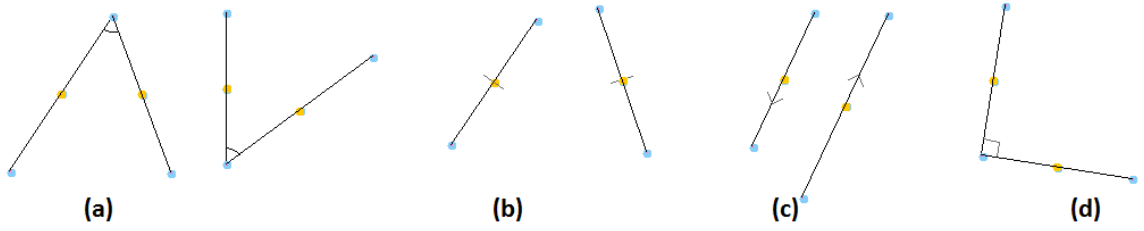


Figure 1.2: Hard constraints

Fig 1.2 shows the various hard constraints, (a) shows equal angle constraint, (b) shows equal length constraint, (c) shows parallel line constraint and (d) shows perpendicular constraints

After recognizing various constraints, constraint solving is done using a technique defined in [Shi05], which uses *Newton's method in multiple dimensions*.

1.2.3 Sketch-symbol/Sketch-object recognition

Similar to primitive recognition in the sketches, a higher level primitive, i.e. an object or symbol recognition in the sketches can be useful in various domains such as flow chart diagrams, electrical circuit domains, mechanical engineering drawing etc. Currently, this object or symbol identification is applied in the domain of electrical circuits.

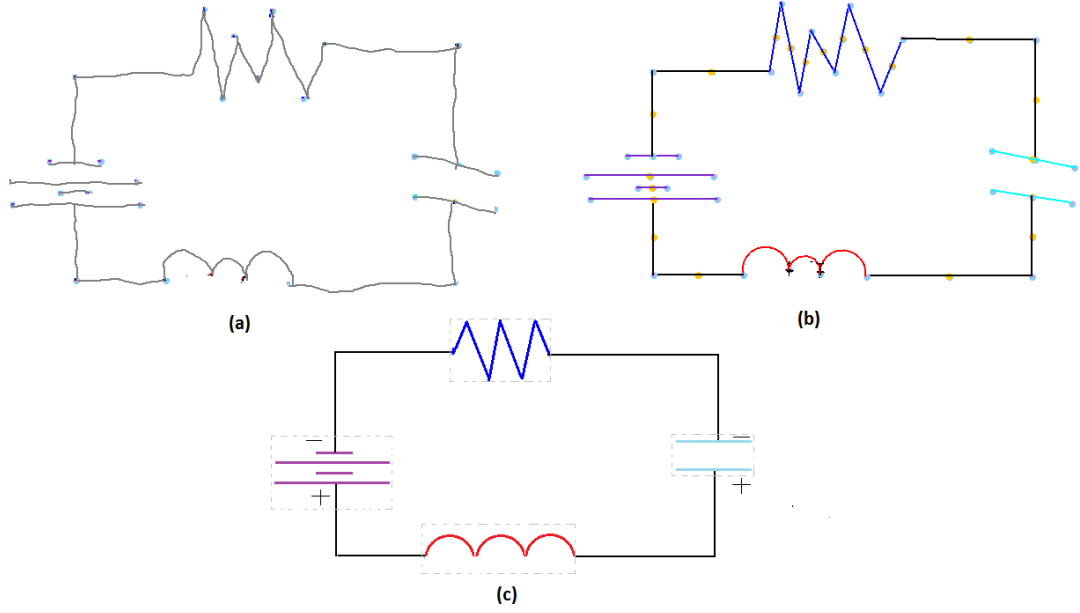


Figure 1.3: Symbol recognition in the electric circuit

Sketch-symbol recognition is the major goal of my thesis project. The idea is to develop an efficient Sketch recognition system which can identify the symbols or components in electrical circuits. The approach adopted in this thesis is from [4, Kara et al. 2005] work. The natural order of the strokes with which they are drawn is used to detect the symbol and using the basic features (number of strokes, connection information etc.) to recognize the symbol in the electric diagram.

In figure 1.3(a), after we have a drawing of the electrical circuit, we want a computer to recognize the various symbols in the circuit drawing such as the resistors, batteries and capacitors as shown in 1.3(b). Similar to primitive recognition, where a set of points of a raw stroke is fitted to primitives such as a line or a circular arc, here also, after the recognition process of a symbol (a set of segments), it can be fitted to its corresponding standard set of high-level primitives. This high-level primitive can be a well-defined set of segments or can be an image that represent the a true object or a symbol. Furthermore, we can associate the various properties to this high-level object such as its values, its connected components, current flow, voltage drop, etc. This is the main motivation to improve recognition of symbols in sketches. The identified symbols from Fig 1.3(b), can be replaced by standard symbols or images in Fig 1.3(c) (shown in the dotted box).

1.3 Related Work

Symbols are more complex, and typically require multiple strokes to be drawn, one of the main challenges of multiple stroke recognizer is that it should be rotational and scale invariant. In the past, a domain based sketch recognition system had been proposed that uses hierarchical shape description language to describe symbol in the domain [1, Christine Alvarado and Randall Davis, 2004]. And, for finding the objects it creates the Bayesian network to recognize the most probable group of strokes as an object. This requires the domain knowledge to be given in their language.

Lee et Al. [8, 2007] represents the symbol as graphs where basic elements (line and arcs) form the nodes and the edges represent the pairwise relationship between the nodes. Invariance is achieved by casting the classification as a graph matching problem. In [2, Deng et al. 2013] a structural method for identification of objects. It matches the entire sketch with reference object and calculates the error score according to miss-match. The system is rotational and scale invariant but cannot recognize the objects in the entire sketch.

In [11, Sezgin et Al. 2005], a simple, innovative approach is proposed. In this authors show that it is possible to view a sketch as an interactive process in order to use HMM(Hidden Markov model) for sketch recognition modeling.

A statistical framework based on dynamic Bayesian network in [10, Sezgin, T. M. and Davis, R. 2008] has been proposed that explicitly models the fact, that objects can be drawn interspersed. [4, Kara et al. 2005] is a sketched recognition system for electrical circuits. It considers the Temporal ordering of the strokes to detect the object and then uses the geometric features to recognize the object to calculate the heuristic score. For each matching feature, it assigns a different kind of score. From the final summation of a score, it identifies an object.

1.4 Work Done

The main goal of this project is to create an effective symbol recognition system for circuit sketches. In this thesis work, I have studied and analyzed already existing approaches and solutions for Sketch-Symbol detection and recognition. I have researched on various aspects of DrawCAD [9, 7, 6] and understood its architecture. I have also analyzed the contribution by Vivek Kantaria [5, 2015] in symbol recognition of electrical circuits and in the segmentation process. Further, I have created a new dataset to test the efficiency of the existing system, identified various flaws and bugs present in the existing implementation and corrected them. In addition to all this, I have implemented new approaches to improve the efficiency of the symbol recognition process. I have integrated these changes with various panels and modules of DrawCAD.

1.5 Outline of report

In chapter 2, existing candidate symbol detection algorithm by Vivek in [5] is discussed and later on the improvement and modification in the algorithm is discussed. In chapter 3, symbol recognition phase is discussed in details. We will talk about the feature set and matching techniques by Vivek [5], followed by a discussion on modification proposed in the recognition algorithm in this phase of the project. In chapter 4, segmentation process by Vivek [5] is discussed. In this chapter, segmentation point detection scheme and merging algorithm are discussed. In the later part, drawbacks in the implementation are discussed.

In chapter 5, Performance analysis of system on various techniques of chapter 2 and chapter 3, on both existing and new methods is performed. Also, some test case failure analysis is done. Chapter 6, discuss the features added to DrawCAD. Chapter 7 gives a brief note on conclusion of the work done in this semester and future work to be done in the next phase of the project.

Chapter 2

Object/Symbol Detection

The chapter gives the description about the symbol detection algorithm used in the existing system by Vivek [5] describing the concept of ink density and bounding box used. In section 2.1, Detection process is discussed. In section 2.2, the necessity of improvement in detection is discussed. In section 2.3, modification performed in detection process and training approach is discussed. The new modified approach describes the variation in the Ink density and bounding box definition.

2.1 Detection of the Candidate Symbol

A typical sketch consists of some symbols connected together by connectors. These symbols and connectors are domain specific. For example, in a directed graph, symbols represent graph nodes and connectors represent graph edges. In the case of the electrical circuit diagram, symbols can be resistance, capacitance, battery or any other electrical circuit element and connectors are wires connecting them.

Terminologies

Before starting the process of recognition, we need to identify the Candidate Symbols in the diagram. A *Candidate Symbol* is a set of primitives segment that can represent an actual object from our domain. In the detection step, all the candidate symbol are identified, which further will be used in next phase of the recognition process. This section describes the candidate detection process used by Vivek [5].

An *Object/Symbol* is a Candidate Symbol which definitely represent some element from our domain. In our domain context, it can be a resistor, capacitor, gates etc. In other words, it is a set of segments with semantics.

The sketches we are dealing with has all the symbols connected to each other by *Connectors*. These sketches are different from sketches where objectmbol has no connection between them, and the symbol/object can be detected by grouping the segments that are directly in contact or within a range. The notion of connectors in these sketches makes symbol detection process a difficult task.

Detection phase takes a series of primitive segments as input and gives the list of candidate symbols as output. Each candidate is a set of consecutively drawn primitive segment. The important thing to notice is that we are considering that user will always draw a complete object at a time i.e. once user start drawing an object, (s)he should finish it and then goes to next symbol in the diagram.

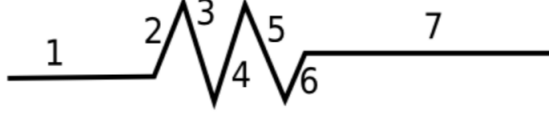


Figure 2.1: Symbol detection using temporal ordering in resistor [5]

In figure 2.1, the input is set of segments $\{1, 2, 3, 4, 5, 6, 7\}$. Then suppose, identified candidate symbols are $\{2, 3, 4\}$, $\{3, 4, 5, 6, 7\}$ and $\{2, 3, 4, 5, 6\}$. but $\{2, 3, 4, 5, 6\}$ is our actual symbol for resistor. So, this set is our object which is identified in the next phase of recognition process.

There will be a large number of points gathered in a place where there exists a symbol. Using this basic idea, we can observe that primitives that are close, constitute a real object. For quantification of the concentration of points, a notion of *Ink Density* is defined in [5] as:

$$InkDensity = \frac{InkLength}{BoundingBoxArea} \quad (2.1)$$

where Ink Length represents the summation of the length of all segments. Bounding box is defined as a rectangle aligned to principle axes, which covers all segments. To find the candidate symbol, two recursive procedures are used.

First procedure tries to find the end point of the candidate symbol. Initially, a bound box is created for the first two segments and ink density is calculated. Then at each step, next segment is added to it and again the ink density is calculated. If the ink density drops below 5%, it will not be included in the current candidate symbol and the previously added segment will be marked as the possible end point of the symbol. Algorithm by [5, Vivek] is given in fig 2.2.

```

input : Series of primitive segments  $segs[0 : n]$ 
output: List of candidate symbols

 $RESULT = \{\}$ 
Initialize Bounding Box  $B$  with  $segs[0]$  and  $segs[1]$ 
for  $i \leftarrow 2$  to  $n$  do
    if  $i \geq 6$  then
        break
    end
    Add  $segs[i]$  to  $B$ 
    if Density drops more than 5% then
         $L1 = \text{Find\_Front}(segs[0 : i - 1])$ 
         $L2 = \text{Find\_End}(segs[i : n])$ 
         $RESULT = RESULT \cup L1 \cup L2$ 
    end
end
 $L1 = \text{Find\_Front}(segs[0 : i - 1])$ 
 $L2 = \text{Find\_End}(segs[i : n])$ 
 $RESULT = RESULT \cup L1 \cup L2$ 
return  $RESULT$ 

```

Figure 2.2: Forward algorithm to detect end point of candidate symbol[5]

In the second procedure, the start point for the candidate symbol is calculated from the identified endpoint from the first procedure. In this procedure, last two drawn segments are considered, and then bounding box after that ink density is calculated. Then it will move backwards and adds previously drawn segment to the box and new ink density is calculated. Whenever ink density drops below 5%, it will be marked as the starting point. The procedure is given in Fig 2.3 :

```

input : Series of primitive segments  $segs[0 : n]$ 
output: List of candidate symbols

 $RESULT = \{\}$ 
if  $n \leq 1$  then
    | return  $RESULT$ 
end
Initialize Bounding Box  $B$  with  $segs[n]$  and  $segs[n - 1]$ 
for  $i \leftarrow n - 2$  to 0 do
    | if  $n - i + 1 > 6$  then
    | | break
    | end
    | Add  $segs[i]$  to  $B$ 
    | if Density drops more than 5% then
    | |  $RESULT = RESULT \cup \{ segs[i + 1 : n] \}$ 
    | | end
    | end
end
 $RESULT = RESULT \cup \{ segs[i + 1 : n] \}$ 
return  $RESULT$ 

```

Figure 2.3: Backward algorithm to detect end point of candidate symbol[5]

Each pair of starting and ending point that represent a candidate object is then added to the candidate list.

2.2 Desired Improvements

In Sketch-symbol recognition process, there is required to tolerate some amount of noise. The motivation here is that the user is likely to make mistakes. If a user makes a small mistake while drawing a symbol, and if this mistake does not change the overall semantics of the symbol, then it can be tolerated.

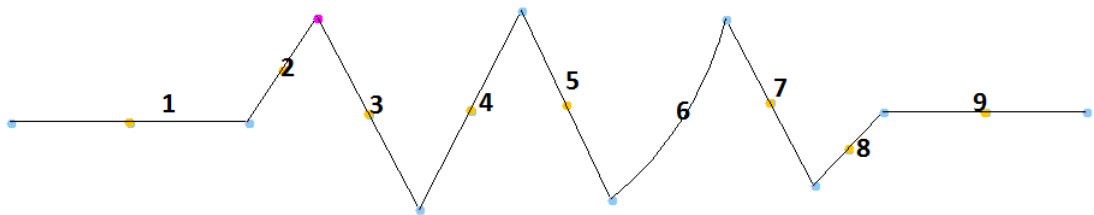


Figure 2.4: Noise while drawing a resistor

For example in fig 2.4, we can see, the user made a mistake in drawing the 6th segment as a curve, instead of drawing a line segment. This case should be captured and our recognition algorithm should recognize it as a resistor.

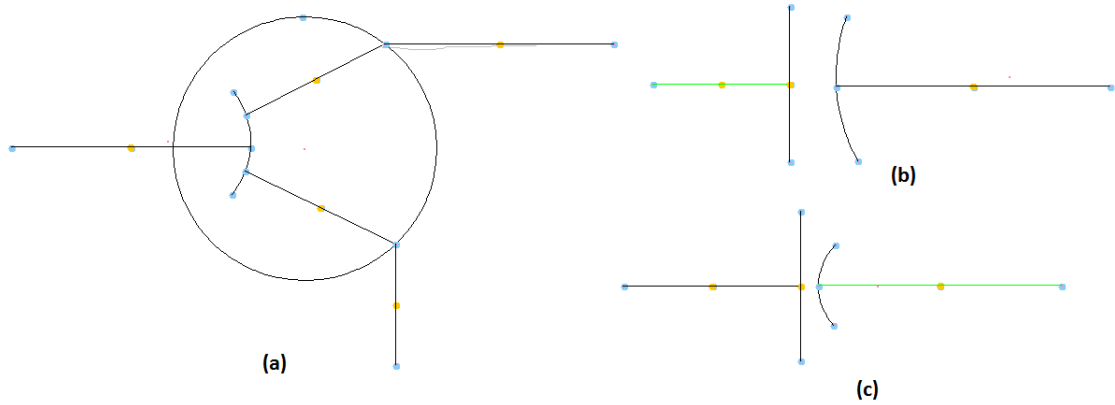


Figure 2.5: Noise tolerance in the identification of NPN(a), capacitor(b) and battery symbol(c)

Also in fig 2.5, some amount of noise can be tolerated. In (a), the small curve connecting base emitter and collector of the npn transistor can be tolerated, as the semantics of the symbol does not change much. Similarly, in (b) and (c), the curves can be tolerated.

To achieve this, we have trained our system with reference symbols with some noisy segments. However, this will violate the assumption, that the features are independent.

Also, in Fig 2.6, some amount of variation in length of the particular symbol is allowed. For example, a resistance can have any number of crests and troughs. Similarly, an inductor can have any number of circular arcs. This problem can be solved by considering the identified symbol as a prefix or suffix of the drawn symbol and thus accepting the whole symbol.

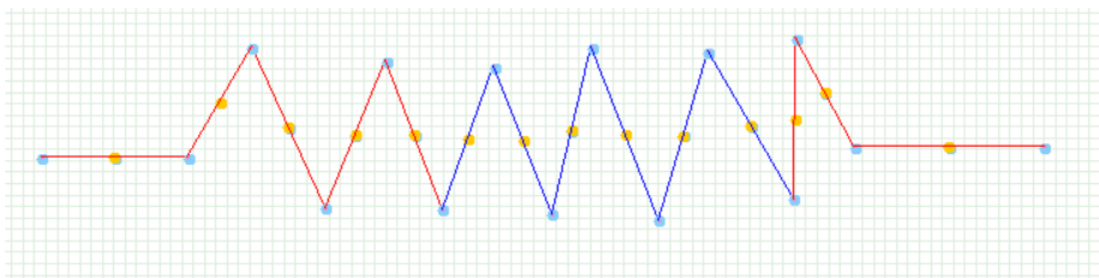


Figure 2.6: Resistor can have any number of crest and troughs.

2.3 Modified Detection algorithm

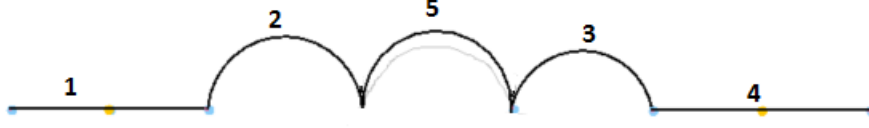


Figure 2.7: Inductance drawn in different order.

The Symbol detection algorithm defined above is order dependent and uses the natural drawing order i.e. temporal order to detect symbol in the circuit. As shown in figure 2.7, suppose user draws an inductor in stroke order, then it must be recognized as an inductor. To recognize the strokes in any order, we should take all its subsets and then find the candidate symbol whose ink density (in the bounding box of that subset) is more than the threshold. Although, it is not feasible since it is of exponential order.

Also, we want to capture an object that has a large number of segments, for example in case of resistor or inductor, a whole candidate object should be detected as candidate object. To catch these properties we have performed experiments by changing the detection algorithm as follows:

- Definition of Ink density.
- Bounding box definition.

These modification are described in subsequent subsections:

2.3.1 Definition of Ink density

I have performed experiments by changing the ink definition in many ways:

- Using the square of Ink length as used in [4] while calculating the Ink Density as:

$$InkDensity = \frac{InkLength^2}{BoundingBoxArea} \quad (2.2)$$

This requires a change in density threshold, value of which was chosen 20 % .

- Instead of using the actual ink length, the concept of hidden length is used as explained in [4]. The hidden ink is defined as ink that would occur if user did not release mouse while drawing. As shown in Fig 2.8, gray line represents the hidden ink which could have occurred, if the user did not release the mouse click.

So, the new Ink Length was defined as:

$$Inklength = ActualInklength + 0.2 * HiddenInk \quad (2.3)$$

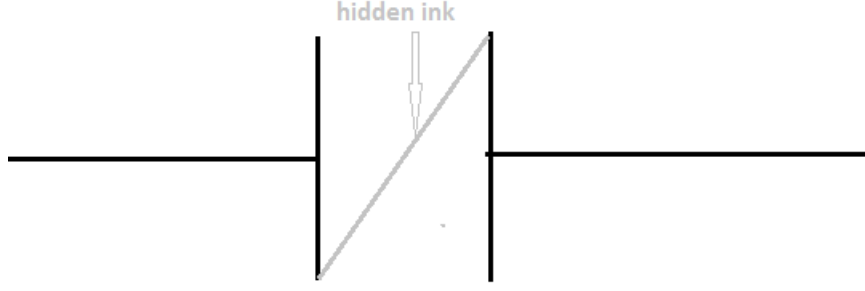


Figure 2.8: Hidden ink in while drawing capacitor.

Here, we are adding partial hidden ink length to Actual length, not entire ink length as used in [4]. Here 0.2 factor is taken empirically. This concept allows a candidate object to be detected that has been missed out due to density threshold.

2.3.2 Bounding box definition.

Previously, we were considering that, bounding box would be aligned with the principle axes. But, if we consider bounding box to be aligned on different angles or the bounding box as a smallest rectangle that contains segments, we can capture the cases where the symbols were not aligned with the axes.

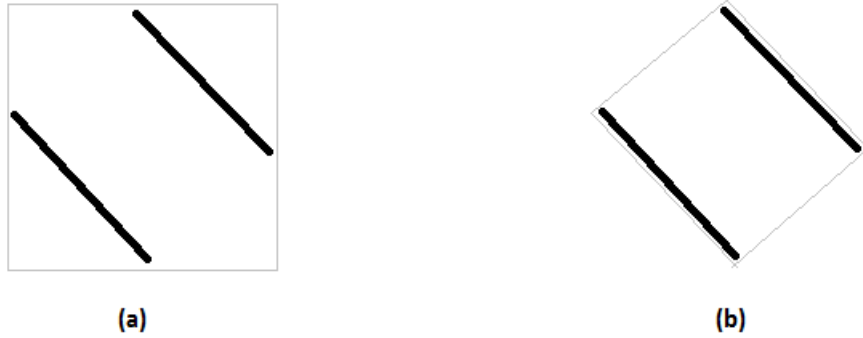


Figure 2.9: Bounding box alignment on capacitor symbol

As shown in Fig 2.9 (a), the bounding box has large area when considered aligned to the principle axes. However, (b) has lowest area bounding box, which is the smallest rectangle covering the object. For density calculation, we need only the area of the best-oriented bounding box. To calculate the best-oriented bounding box, the simple solution is, to do all the calculation in rotated coordinate plane. For simplicity, the orientation of the plane is done in 6 directions (15° apart in the range 0° - 90°) as shown in fig 2.10. To calculate the coordinate points (x, y) of segment lying in the plane rotated by an angle θ , following equation is used:

$$x_{new} = \sin(\theta) * y + \cos(\theta) * x \quad (2.4)$$

$$y_{new} = \cos(\theta) * x - \sin(\theta) * y \quad (2.5)$$

Where (x_n, y_{new}) are new coordinate of point (x, y) . Bounding box coordinates (x_{min}, y_{min}) and (x_{max}, y_{max}) is calculated by taking the minimum and maximum of all x_{new} and y_{new} .

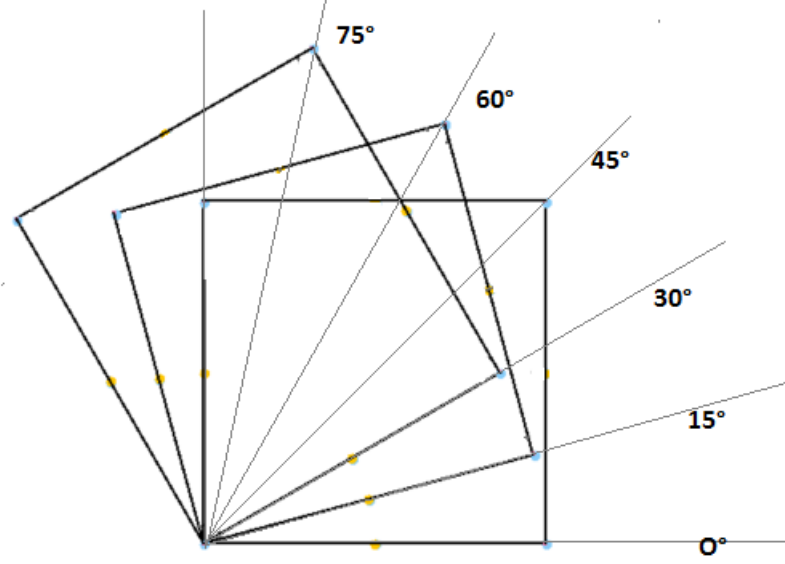


Figure 2.10: Oriented bounding box

The bounding box that has minimum area $((x_{max} - x_{min}) * (y_{max} - y_{min}))$ among all 6 planes, is said to be the best oriented bounding box.

Chapter 3

Object/Symbol Recognition

The chapter gives a description of approach used in identifying the class of a candidate symbol detected. The first section describes the feature set and matching approaches used in the previous work by Vivek. Also, algorithm used for the recognition process is explained. In the second section, modified recognition approach used in this phase of the project is discussed.

3.1 Classification of Detected Symbol

Terminologies

- **Class or Symbol Type** : A class represents a set of any one type of electrical symbol. So, a candidate Object can belong to any of the defined classes such as a resistor, capacitor etc in our Domain context.
- **Connector**: Connectors are segments which are used in a sketch to connect the other object or symbols.

3.1.1 Feature-Set

After the detection of Candidate symbols, we need to classify these symbols as various Symbol classes or connectors. Given a list of candidate symbols as input, our output will be a list of actual symbols in the sketch.

To classify objects in various classes, we need a set of features of an object which uniquely defines its class. Common intuitive geometry features of the object are used. The features used are as follows.

1. Total number of segments
2. Number of line segment
3. Number of arc segments
4. Number of parallel lines
5. Number of perpendicular lines
6. Number of an end point to end point intersection segments.
7. Number of an end point to mid point intersection segments.

8. Number of midpoint to midpoints intersection segments.

9. Ratio of average length of the segment to largest length of the segments.

Theses feature values of the training symbols along with its label are stored in a file as our reference. When an unknown candidate symbol comes for classification, the above mentioned features are calculated, and the most matched reference symbol label is returned by the matching algorithm.

3.1.2 Normal Probability Distribution Matching

As per [5], the existing system was trained using file having reference symbols' feature values and labels. For each class, the classifier is trained by calculating the mean and variance of each feature, for all its reference symbol. Suppose with the label of class C we have n reference symbols, then for each feature mean and variance are calculated for those n references. And for a class, mean and variance of features is stored in a vector.

To classify the symbol, a Naive Bayesian Classifier [3, 1996] is used. Here the assumption is that each features in vector $\mathbf{F} = (f_1, f_2, \dots, f_n)$ is independent random variable having a normal distribution. Using Bayes' Theorem, the probability of a candidate symbol C having the class label k and feature f is calculated as:

$$p(C_k, f) = \frac{1}{\sqrt{2\pi\sigma_{f,k}^2}} \exp\left(-\frac{(x_f - \mu_{f,k})^2}{2\sigma_{f,k}^2}\right) \quad (3.1)$$

where $\mu_{f,k}$ is the mean of feature f of class label k
and, $\sigma_{f,k}^2$ is the variance of feature f of class k
and, x_f is the value of feature f of candidate object C

After calculating the probability of all features of candidate symbol C for a class k , the *Matching Value* MV is calculated as the likelihood of candidate symbol's features, which is defined as:

$$MV(C, k) = \prod_{f \in \mathbf{F}} p(C_k, f) \quad (3.2)$$

where $MV(C, k)$ is the matching value of candidate object C for class label k .

The candidate object is classified label as C^* , that maximizes the Matching value:

$$C^* = \arg \max_k MV(C_k, k) \quad (3.3)$$

where $k \in \mathbf{K}$ which is set of all the symbol classes.

In the classification algorithm, first candidate having highest matching value is marked as real symbol. Then, all other candidate symbol that share common segments (i.e. they overlaps), are dropped from the list. The procedure is repeated until there are no more candidates in the list. The remaining segments which are not identified as symbols are classified as connectors. The classification algorithms is as follows :

Algorithm . Classification of Candidate Symbol

Input List of candidate symbols LST

Output List of objects with class label

```
1: for each entry in  $LST$  do
2:   calculate the matching value against all possible elements in the domain ;
3:   assign label to the element for which highest matching value ;
4:   set highest matching value as  $MV$  for current entry ;
5:   if  $MV < threshold$  then
6:     remove current entry from  $LST$  ;
7:   end
8: end
9: sort  $LST$  in descending order of  $MV$  ;
10: for each entry in  $LST$  from start to end do
11:   Mark current entry as actual object ;
12:   remove all candidates from  $LST$  which has at least one common segment
13:   with current object ;
14:   Find segments connecting to this object, call them connectors
15:   remove all candidates from  $LST$  which contains connectors
16: end
```

The assumption that, for a class, every feature is independent of other, is not correct. Because, the above defined features can be dependent. For example, number of end to end connections depends upon number of segments in case of resistor. Because of its simplicity, this approach is used and also, only two or three reference symbols are enough to train the system.

3.2 Least-Square Matching

In Least-Square matching, for each candidate symbol, we will find a reference symbol that has least *Square_Error* with respect to candidate symbol. The squared error between candidate symbol C and a reference symbol R is defined as:

$$Squared_Error(C, R) = \sum_{f \in F} (C_f - R_f)^2 \quad (3.4)$$

Where C_f is the value of feature f of symbol C and F is the feature set (in this case 9 features as explained above). Now, the label of the candidate symbol will be same as the label of Reference Symbol, that has least *Square_Error* which is defined as:

$$Label(C) = Label(\underset{r \in R}{\operatorname{argmin}}(Squared_Error(C, r))) \quad (3.5)$$

where, R is set of all reference Symbols and $Label(C)$ is the label of symbol C . This trivial approach can be one possibility for matching the symbol to a reference symbol. I have implemented to compare it with other matching approaches, as it was not implemented previously.

3.3 Desired Improvements

As shown in 3.1, (a) & (b) can represent different symbols, and (a) & (b) both will be recognized as same object. Because we have considered features that has number of lines and curves, and end to end connection (end to end or middle to middle) and both (a) and (b) have similar feature vector. Here we can make use another information about the symbol i.e. the number of intersection points occurring in a reference symbol and add it to our feature-set.

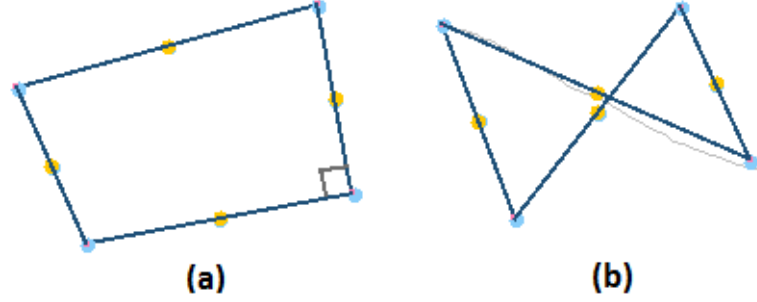


Figure 3.1: Incorrect recognition of two different object to same class

Also, suppose in fig 3.2, we want to recognize the sketch (a) as a NAND gate, and above discussed feature set is used, then sketch (b) will also get identified as NAND gate. Also, (c) was recognized as inductor by our program which does not look like an inductor, thus the information of the one circle being complete can be helpful in defining symbols.

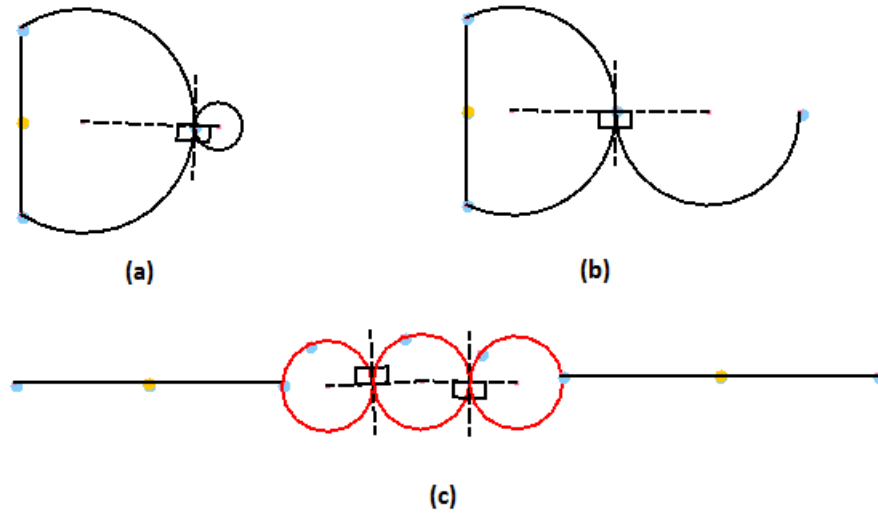


Figure 3.2: Incorrectly recognized NAND gate and Inductor

3.4 Modified Recognition process

Some modification was done in the recognition process are:

3.4.1 Changing feature set

To capture cases discussed in the previous section, the feature-set is redefined. The complete feature-set used here consists of:

1. Total number of segments
2. Number of line segments
3. Number of arc segments
4. number of almost parallel segments
5. number of almost perpendicular (segments not necessarily touching).
6. Number of an end point to end point intersection segments.
7. Number of an end point to mid point intersection segments.
8. Number of midpoint to midpoints intersection segments.
9. Ratio of average length of the segment to largest length of the segments.
10. number line segment or arc segment less than or greater the average length,
11. number of concentric arc segments
12. number of arcs which are approximately semi-arc, full circles or quarter arcs.
13. number of intersection points in the symbol.

3.4.2 Training modification

A symbol is set of segments (line or circular arc). At the time of training, when a new reference symbol is added, following steps are taken:

- Create multiple copies (5 in this case) of the newly added reference symbol.
- Out of these copies, we select a few copies (3 in this case) and introduce noise in them. **Noise** is defined as changing a line segment to an arc segment or vice-versa. For the experiment, 1 segment is changed.
- Now for each copy, its corresponding feature-set values are calculated and label are stored.

All these steps were carried out manually in this phase. We can automate this task in future.

3.4.3 Matching value modification

In Fig 2.6, one reason why the entire resistor was not recognized can be the high matching value of its sub-parts (as a candidate symbol) than the entire resistor. So, I have applied some heuristics in the Matching Value calculation. The Matching value is now multiplied by a factor C^n , where C is some constant (empirically 10) and n is number of strokes in a symbol. This makes the matching value high for the complete symbol than its subset. Thus, this will increase the probability of largest matching symbol.

3.4.4 Changing Bounding Box threshold

In fig 3.3, we can see that, as the bounding box is small, these two segment in the gray box will be recognized as a capacitor as they have a high matching value corresponding to the Capacitor. The question here is, whether to accept this situation or not? As it does not look like capacitance. Also, it may happen to these two segments are part of different symbols, i.e. end of one symbol and starting of another. This is taken care of by setting a bounding box condition. When the ratio of the smallest to largest side of the bounding box is greater than 8 and the ratio of largest segment in the box to the largest side > 5 , then we will drop this candidate symbol.

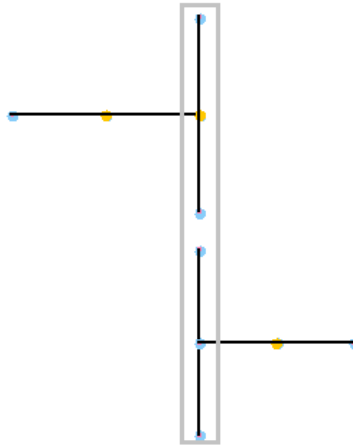


Figure 3.3: Incorrect recognition of capacitor

Chapter 4

Segmentation

4.1 Segmentation

It is the process of breaking raw strokes into the simplest building blocks or primitive segments. These primitive segments are points, line, and the circular arc.

Let us consider an example, we want to draw a resistor as shown in Fig.4.1(c). Suppose, instead of drawing 8 primitive lines of a resistor by one stroke at a time, the user draws a continuous stroke as shown in fig 4.1(a). After drawing, this stroke can be broken down into 8 primitive strokes to form a resistor. We need a mechanism to identify these primitive segments where the large continuous stroke could be broken. The point at which stroke is broken are called segmentation points, these points are represented by red points in fig 4.1(b). The goal of the segmentation process is to identify these points.

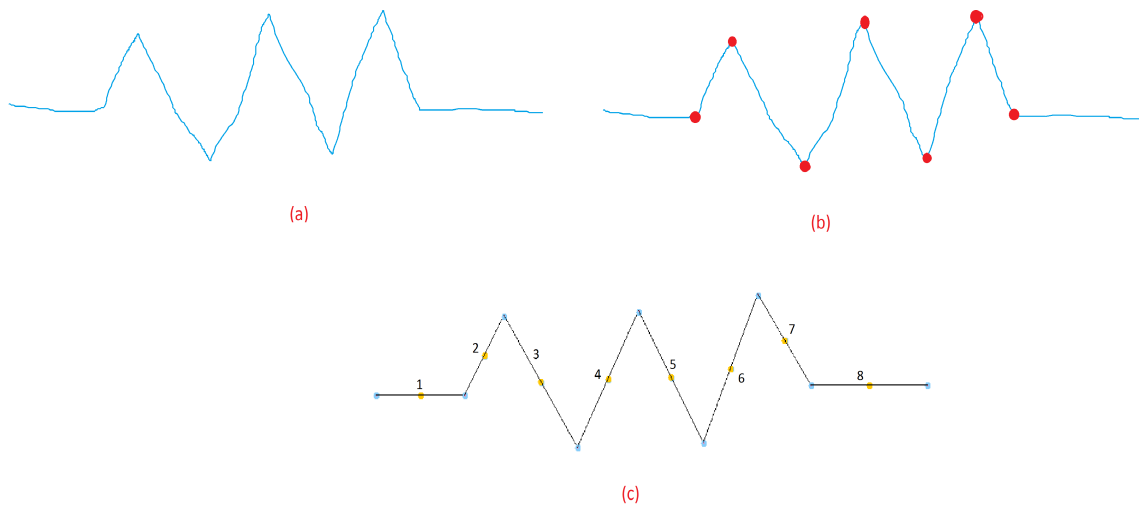


Figure 4.1: Segmentation example

4.2 Segment Point Detection Scheme

The segmentation process must have following features;

- It should ignore unwanted pen fluctuations and useful information should be kept intact. As in Fig 4.2 (a) bottom line can be divided into parts because of slight curvature, this situation should be avoided.
- Is able to ignore noise at the ending and starting of the stroke. Generally, while drawing there is some noise at the beginning or ending of the stroke as shown in Fig 4.2 (b), and this type of noise must be avoided.
- Is a real-time solution.
- Is able to adjust to the speed of drawing of different users.

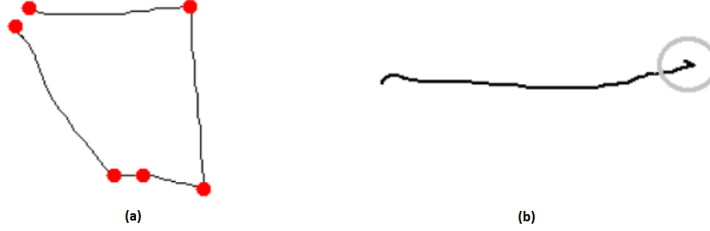


Figure 4.2: Challenge in the segmentation [5]

Different properties used to find segment points are described below:

4.2.1 Direction

The value of direction [13] at any point (x_n, y_n) is given by:

$$d_n = \arctan \frac{y_{n+1} - y_n}{x_{n+1} - x_n} \quad (4.1)$$

The user changes directions frequently while drawing the strokes. This information is noise sensitive because only 2 neighbouring points are considered. We can change this method by finding the direction between end points of some neighbourhood of fixed window. However, the sharp change in the direction will not be captured by this approach.

4.2.2 Curvature

Curvature[13] can be defined as an extended feature of direction. It is given by:

$$c_n = \frac{\sum_{i=n-k}^{n+k-1} |(d_{i+1} - d_i)|}{D(n-k, n+k)} \quad (4.2)$$

where, k = constant neighbourhood window,

$D(p, q)$ = length of path between the points p and q , which is summation of distance of all the points between them ,

d_i = direction value at point i,

Curvature measures the change in the neighbourhood. Hence, curvature value is useful in identification of points having sharp direction change.

4.2.3 Speed

The speed[12] value at any point in stroke is defined as:

$$s_n = \frac{D(n-1, n+1)}{t_{n+1} - t_n} \quad (4.3)$$

where t_i = time stamp of i th point, $D(p, q)$ = path length between p and q. The motivation here is that, most of the users generally slow down at segmentation points. So, local minima of speed can be used to locate the segment points.

4.2.4 Existing Method: Segmentation

In the Segmentation work done by Vivek in [5] following two step process for segmentation is used :

1. Finding initial segmentation points using on the basis of Direction, Speed, and Curvature.
2. Then merge the segments.

Finding initial Segments

This step finds the set of segment points found by both using speed, and curvature method, which is actually a superset of actual segment points. As some more points can be identified as segment points because of noise. At a certain point, if speed is less than $speed_mean * 0.4$ or curvature is greater than $mean * 1.5$, then it is classified as an initial segmentation point. Also, the points close to the end point of the stroke is ignored because there can be a possibility of noise at the end points.

Merging Segments

In the first step, many other points were also identified as segments points. In this process, the decision of whether a point to be taken as segment point or not, is taken by merging the two segments having that common segment point. In this process, two consecutive identified segment are taken and depending upon the type of segments, we divide merging process in three ways:

1. **Line-Line Merge :** Two line segments are merged when the angle between them is less than the threshold (20°).
2. **Curve-Curve Merge:** Two curve segments are merged when the distance between their center is less than the threshold (25), and the difference of the radius is less than the threshold (15).
3. **Line-Curve Merge:** Line and curve segment will be merged to curve when the radius and center of the curve segment and the radius and center of resulting merged segments are less than the threshold.

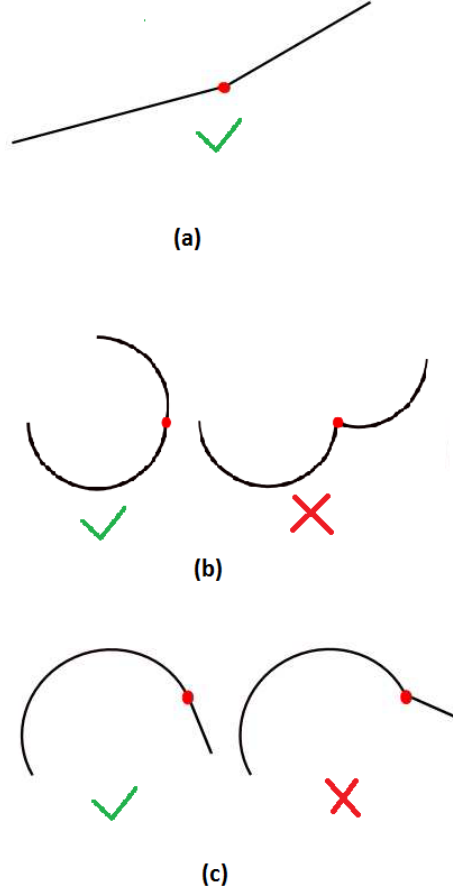


Figure 4.3: Merging of two consecutive segment [5]

4.2.5 Drawback in Previous Segmentation Implementation

The current implementation of segmentation by Vivek [5], does not correctly recognize the segments corresponding to complex strokes. This may occur due to variation in speed while drawing the strokes or due to not pausing for sometimes at corner points or user was trying to draw a stroke that is very small. Generally, the mistake is in miss recognizing the line segment as a curve segment. In fig 4.4 we show the test cases where segmentation does not work properly. Clearly, we can see that segmentation process does not work properly. Improvement is needed in this regard, to correctly detect the segment from the stroke. I will try to improve this in future.

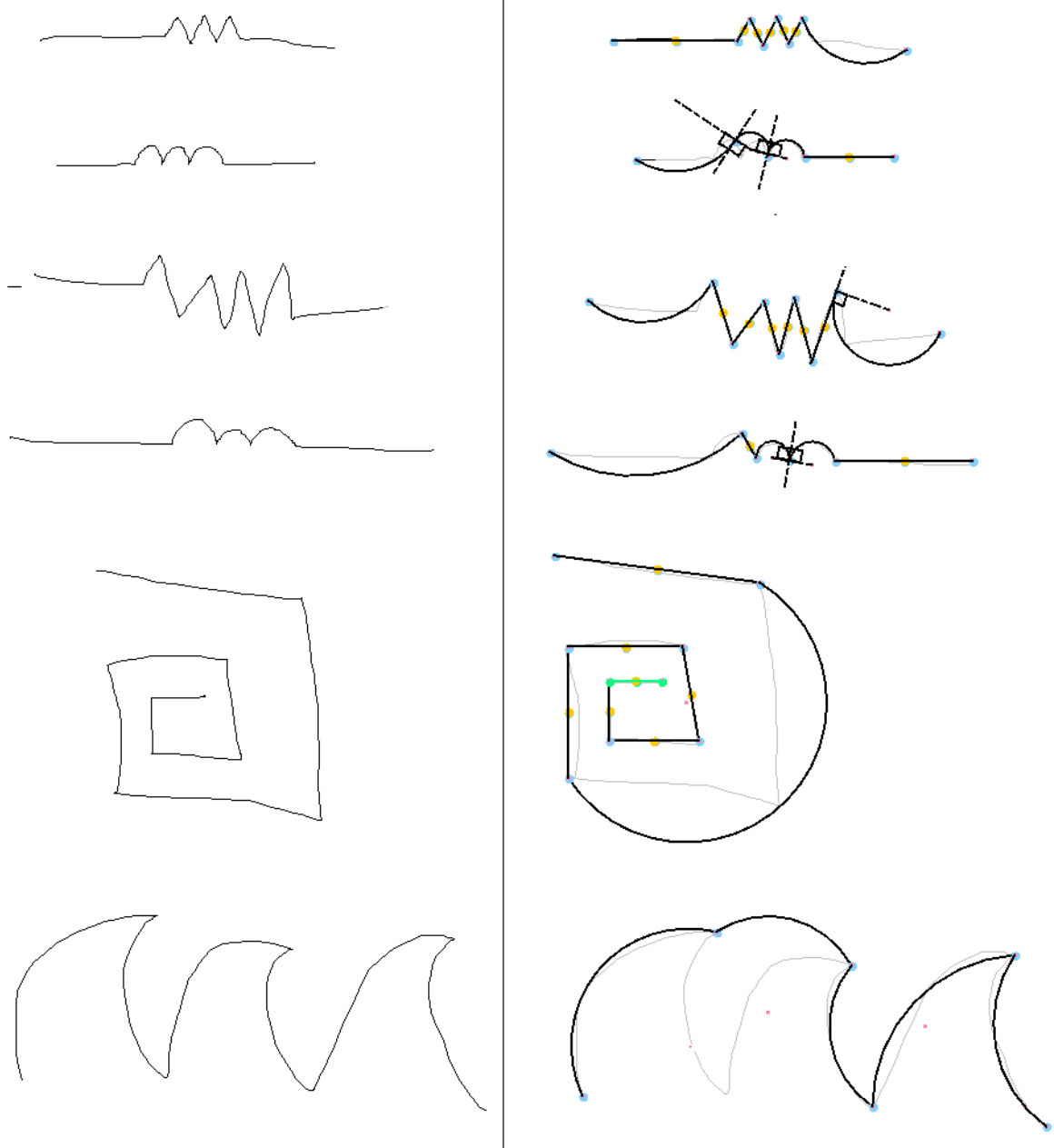


Figure 4.4: Failed TestCases in Segmentation

Chapter 5

Testing

5.1 Test-case Analysis for Symbol recognition

The symbol recognition must be rotationally and scale invariant. Also, some amount noise can be tolerated in a symbol recognition, as discussed in section 2.2. Previously there was no defined dataset for our problem that is compatible with Draw-CAD. So in this phase I have created around simple 140 test cases and 15-20 complex test cases to generate a dataset.

Simple test cases contain only single type of symbol with no connection between them. *Complex* test cases will be like complex electrical circuits having average 6-8 symbols.

To test the recognition efficiency of the system, in some of the test cases, a few noise segments (line segment instead of arc or vice-versa) are introduced. Also, in some of the test-cases, order of drawing of stroke is changed. These test cases will fail for our implementation, as we had taken the order of drawing into account during the detection process.

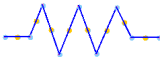

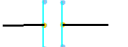




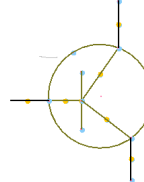
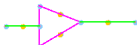

| | | | |
|---|--------------------|--|------------|
|  | Resistance |  | NOT |
|  | capacitance |  | OR |
|  | Inductance |  | AND |
|  | Battery |  | NPN |
|  | Buffer | | |
|  | Box | | |

Figure 5.1: Symbol used for testing

In some of test case symbols are aligned at different angles i.e. horizontal,

vertical, diagonal to test whether our system is orientation invariants. We are currently using the 10 classes of electrical symbols. These symbols are shown in 5.1:

5.1.1 Previous System Performance Analysis

In the Previous system, only the convention algorithm described for object detection in section 2.1 was used and for object recognition Normal Probability Distribution Matching of section 3.1.2 was used.

Analysis: The following Table 5.1 shows the results of previous system. It shows precision, recall, accuracy by calculating TP (True Positive), TN (True Negative), FP (False Positive) and, FN (False Negative) for the system. The previous system has very high precision, but a low recall and accuracy it is due to the fact, that our test dataset has test cases that are order-dependent and contains a lot of noises. If we ignore these cases, then the system has a good accuracy of about 80%.

| Symbol | TOTAL | TP | TN | FP | FN | Precision | Recall | Accuracy |
|--------------------|------------|-----------|-----------|----------|-----------|-------------|-------------|-------------|
| <i>Resistance</i> | 36 | 9 | 10 | 2 | 15 | 0.82 | 0.38 | 0.53 |
| <i>Inductor</i> | 10 | 7 | 1 | 1 | 1 | 0.88 | 0.88 | 0.8 |
| <i>Not</i> | 6 | 3 | 0 | 0 | 3 | 1 | 0.5 | 0.5 |
| <i>Box</i> | 7 | 4 | 2 | 0 | 1 | 1 | 0.8 | 0.86 |
| <i>Npn</i> | 5 | 1 | 0 | 0 | 4 | 1 | 0.2 | 0.2 |
| <i>And</i> | 6 | 5 | 0 | 0 | 1 | 1 | 0.83 | 0.83 |
| <i>Or</i> | 7 | 4 | 0 | 0 | 3 | 1 | 0.57 | 0.57 |
| <i>Battery</i> | 11 | 7 | 2 | 0 | 2 | 1 | 0.78 | 0.82 |
| <i>Buffer</i> | 16 | 9 | 2 | 0 | 5 | 1 | 0.64 | 0.69 |
| <i>Capacitance</i> | 37 | 12 | 5 | 3 | 17 | 0.8 | 0.41 | 0.46 |
| Overall | 141 | 61 | 22 | 6 | 52 | 0.91 | 0.54 | 0.59 |

Table 5.1: Performance analysis of previous approach

5.1.2 Least-square Approach Performance Analysis

| Symbol | TOTAL | TP | TN | FP | FN | Precision | Recall | Accuracy |
|--------------------|------------|-----------|-----------|----------|-----------|-------------|-------------|-------------|
| <i>Resistance</i> | 36 | 15 | 12 | 0 | 9 | 1 | 0.63 | 0.75 |
| <i>Inductor</i> | 10 | 6 | 2 | 0 | 2 | 1 | 0.75 | 0.8 |
| <i>NOT</i> | 6 | 2 | 1 | 0 | 3 | 1 | 0.4 | 0.5 |
| <i>Box</i> | 7 | 3 | 1 | 1 | 2 | 0.75 | 0.6 | 0.57 |
| <i>Npn</i> | 5 | 5 | 0 | 0 | 0 | 1 | 1 | 1 |
| <i>And</i> | 6 | 5 | 0 | 0 | 1 | 1 | 0.83 | 0.83 |
| <i>Or</i> | 7 | 4 | 0 | 0 | 3 | 1 | 0.57 | 0.57 |
| <i>Battery</i> | 11 | 6 | 1 | 1 | 3 | 0.86 | 0.67 | 0.64 |
| <i>buffer</i> | 16 | 6 | 2 | 0 | 8 | 1 | 0.43 | 0.5 |
| <i>capacitance</i> | 37 | 17 | 5 | 3 | 12 | 0.85 | 0.59 | 0.59 |
| overall | 141 | 69 | 24 | 5 | 43 | 0.93 | 0.62 | 0.66 |

Table 5.2: Performance analysis of least-square approach

We carried out the experiment with a least square approach as described in 3.2. Following were the results:

The least Square approach shows slightly good result as compared to the previous approach. The Overall precision is increased to 0.91 and had good recall to 0.62. It also had good accuracy than the previous approach.

5.1.3 Current System Performance Analysis

In the current system, for detection of candidate symbol, same algorithm as in section 2.1 with modification of described in section 2.3 (hidden ink length and Modifying ink-density). Also in training examples some noisy references are used and the bounding box taken as the smallest box.

For recognition of symbol, the modification are done as described in section 3.4. The current system performance is quite satisfactory with respect to the previous system as well as least-square approach, as accuracy is improved from 0.59 to 0.79, which is quite good. Even some of the order cases were also captured, because of addition hidden length concept. The system also performs quite well on recognizing symbols that contain some noisy segments. If we ignore the ordering and noisy cases, our system has recognition accuracy of around 90 %.

| Symbol | TOTAL | TP | TN | FP | FN | Precision | Recall | Accuracy |
|--------------------|------------|-----------|-----------|----------|-----------|-------------|-------------|-------------|
| <i>Resistance</i> | 36 | 20 | 10 | 2 | 4 | 0.91 | 0.83 | 0.83 |
| <i>Inductor</i> | 10 | 7 | 1 | 1 | 1 | 0.88 | 0.88 | 0.8 |
| <i>NOT</i> | 6 | 3 | 1 | 0 | 2 | 1 | 0.6 | 0.67 |
| <i>Box</i> | 7 | 2 | 2 | 0 | 3 | 1 | 0.4 | 0.57 |
| <i>Npn</i> | 5 | 4 | 0 | 0 | 1 | 1 | 0.8 | 0.8 |
| <i>And</i> | 6 | 6 | 0 | 0 | 0 | 1 | 1 | 1 |
| <i>Or</i> | 7 | 6 | 0 | 0 | 1 | 1 | 0.86 | 0.86 |
| <i>Battery</i> | 11 | 9 | 2 | 0 | 0 | 1 | 1 | 1 |
| <i>buffer</i> | 16 | 9 | 2 | 0 | 5 | 1 | 0.64 | 0.69 |
| <i>capacitance</i> | 37 | 23 | 5 | 3 | 6 | 0.88 | 0.79 | 0.76 |
| overall | 141 | 89 | 23 | 6 | 23 | 0.94 | 0.79 | 0.79 |

Table 5.3: Performance after applying modification in recognition process

Now, we will Discuss some of the Test Cases where our algorithm failed:

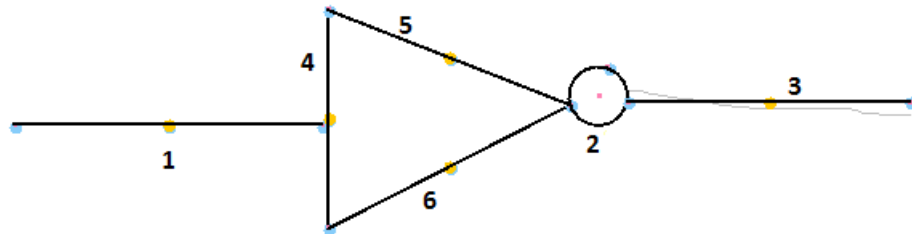


Figure 5.2: Arbitrary ordering of segments

- In Fig 5.2 The Symbol (4,5,6) will be recognized as a buffer instead of Group of strokes (2,4,5,6) as a NOT gate.

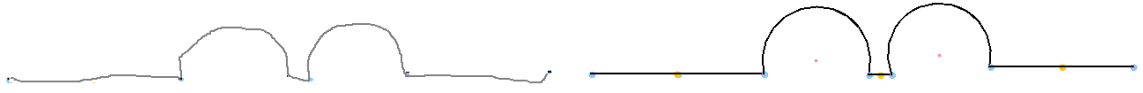


Figure 5.3: A small line segment between the two curve strokes while drawing inductor

- In Fig 5.3, the recognition of inductor is missed because of a small (noisy) line segment between the curve segments. The question arises here is whether, our system should ignore the line segment and recognize symbol as an inductor or does not recognize this symbol?



Figure 5.4: Ambiguity in identifying symbol as NAND gate or NOT gate

- The Fig 5.4 is an ambiguous case, because if we allow some amount of noise to be tolerated in the recognition process, then it is not clear whether the object should be classified as NOT gate or NAND gate .

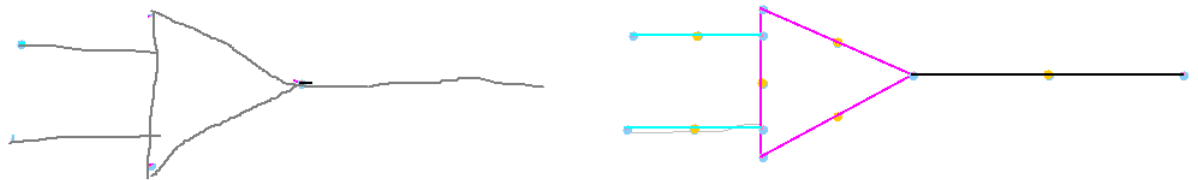


Figure 5.5: Result of drawing various strokes in MSS mode

- In Fig 5.5, the symbol is a combination of two symbols, buffer, and capacitor. In this case, what should be done, whether we should keep both the symbols or discard the capacitor symbol? If we are discarding the capacitor symbol, how to resolve one symbol over other? A symbol is more likely to be connected with wires rather than touching each other. We can discard the symbol that has low matching value.

Chapter 6

Additional Features

6.1 Functionalities for an Object/Symbol

Following are the functionalities that have been added to the DrawCAD in this version.

- **Colour Codes** for different Detected Symbols: Currently in the system, the color code for different symbols has been defined. In future, we allow the user to choose the color code for the new reference symbol.
- **Select** a whole symbol/object; We are allowing a user to select a recognized object as a whole. The description of the symbol is displayed in the constraint view, currently only label information is displayed. This selection of the symbol can be useful in an application, like electric circuit solver, where a user may want to add a property corresponding to the symbol. For example, a resistance value(say, 40 ohms) can be given to resistor symbol, also user can replace the whole symbol with a standard symbol or an image of the resistor for the more beautified sketch. In future, other information can be added to a symbol, for example, what are the symbols that have direct connections with this symbol, what is its value etc.
- **Delete** a selected symbol.

6.2 Segmentation Mode

In this phase of the project, two modes of segmentation in which user can draw are:

1. One stroke per segment (OSS)
2. Multiple stroke per segment (MSS)

Previously (Vivek version), only MSS mode was allowed, now both modes (OSS/MSS) are integrated.

6.2.1 One stroke per segment (OSS)

In this mode, in order to draw a symbol, the primitive segments of the symbols such as line, arcs, and circle must be drawn as a single stroke at a time, i.e. single stroke

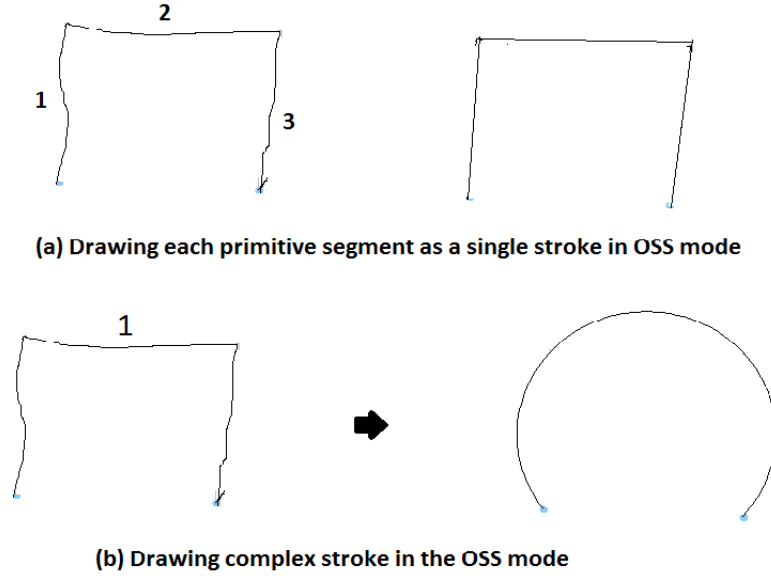


Figure 6.1: Result of Drawing various strokes in OSS mode

must not be a complex stroke (combination of two or more segments). If the user tries to draw a complex stroke in this mode, then the primitive segments fitting the stroke will be a single segment having an average fit to the whole stroke.

As shown in 6.1(a) when a user tries to draw a sketch by drawing stroke 1,2,3 one at a time in OSS mode, then the quadrilateral is correctly fitted. In fig 6.1(b) when a user tries to draw the figure using the complex stroke i.e. whole figure as single complex stroke (stroke 1 without lifting the pen), then an arc is fitted as a primitive segment corresponding to stroke 1.

6.2.2 Multiple stroke per segment(MSS)

In this mode, the complex figure can be drawn using a single stroke only. In other words, two or more simple strokes are drawn as a single stroke without lifting the pen. In this mode, strokes drawn undergo a process of segmentation as described in chapter 5, where the complex stroke is broken down into a set of simple primitive segments. This improves the user experience of drawing, as a user can draw many segments in one stroke only.

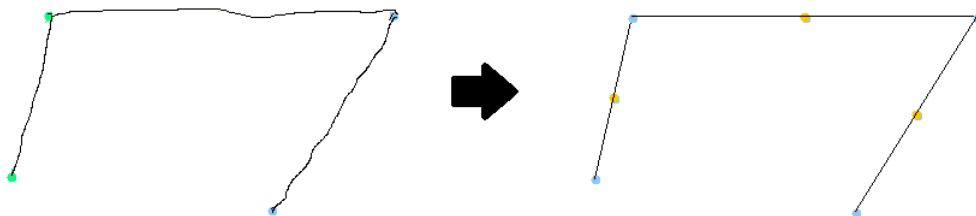


Figure 6.2: Result of drawing strokes in MSS mode

In figure 6.2, when a complex stroke (left) is drawn in the MSS mode, then this

stroke is segmented into the primitive segments as shown at right in the figure.

These modes are indicated in the toolbar panel by default, the OSS mode is ON (indicated in cyan color). And the MSS mode can be selected by clicking on OSS mode, which toggles the mode to MSS (indicated in green color).

6.3 Immediate Recognition

This is a user mode for the recognition process. In this mode, there is continuous recognition of symbols. The recognition process is carried out, when user adds a new segment or makes some changes like editing, translating, deleting a segment. Previously, if user had made a mistake while drawing a sketch, (s)he would realize it only after calling the recognition process explicitly. Immediate recognition is helpful in such cases. Because with immediate recognition, user can correct the mistakes on the fly, i.e. when a segment is added to the sketch and if the symbol was not recognized as user wanted, then (s)he can correct the mistake by modifying it.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, algorithms for symbol detection and recognition for circuit sketches drawn in DrawCAD were studied and analyzed. Results of previous approaches, least square approach, and modified approach were compared. It is observed that modified system has a higher recognition rate as compared to the previous system, and it also tolerates noisy input. Various bugs found in the existing system were fixed. A test Dataset is prepared to check the efficiency of the system. New functionalities were added; different segmentation modes (OSS/MSS), color coding of symbols, immediate recognition to increase the user experience, selection and deletion of a symbol. All the changes made are integrated properly with various panels of DrawCAD.

7.2 Future work

7.2.1 Use Multivariate Normal Probability Matching

In recognition process, instead of assuming the independence of the feature-set, let us consider the features are dependent on each other. So, for a candidate object O the Matching Value $MV(O, C)$ for a particular symbol class C is defined as:

$$MV_{\mathbf{x}}(O, C) = \frac{1}{\sqrt{(2\pi)^k |\sum_C|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu_c)^T \sum_C^{-1} (\mathbf{x} - \mu_C)\right) \quad (7.1)$$

where \mathbf{x} is the k dimensional feature vector

and \sum_C is the $k \times k$ covariance matrix for the k features of class C .

$|\sum_C|$ is the determinant of the covariance matrix.

μ_C is the k dimensional vectors of means of k features.

Thus, the candidate symbol will be recognized as a class label c whose matching value is highest for all classes. I will implement this in the next phase of the project.

7.2.2 Fixing bugs and Adding functionalities

In next phase following following bugs will be fixed and new functionalities will be added:

Bug Fixes

- Hard constraints by adding marker is not recognized in the MSS mode.
- Fixing the failures in segmentation implementation as described in section 4.2.5.

Adding functionalities

- Currently, the testing procedure is not automated. I would be automating the process of testing.
- Automate the training task as described in section 3.4.2.
- **Drag or Nudge** a whole object/symbol to a new location. This type of feature requires a careful implementation because moving, rescaling of a symbol can affect the other connected symbols. As this is very different from primitive nudging, which tries to solve the global constraint problem. Here, relative position or size of symbol segments with respect to the symbol must not change. However, the symbol as a whole can change its position and size.

7.2.3 Global Beautification

The future work can be aimed at a global beautification of the whole drawing. For example, all the wires in the circuit are axes aligned, all the components are of same size orders and aligned.

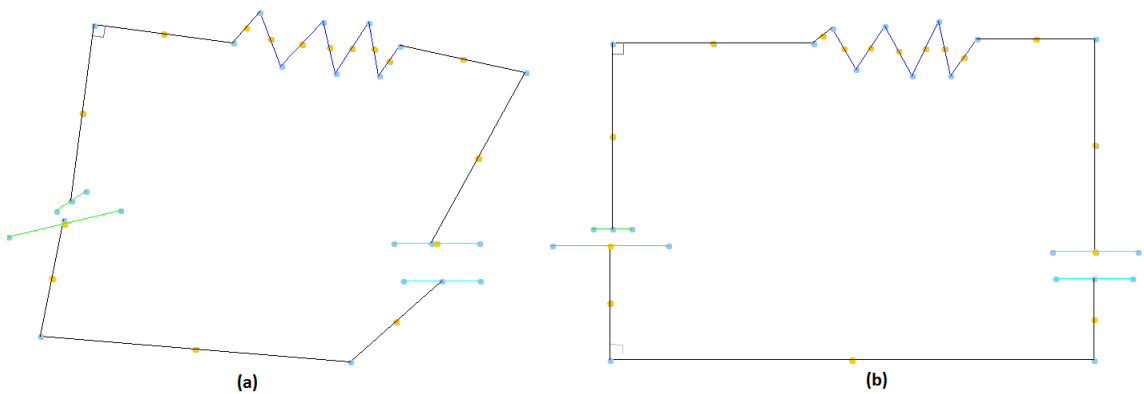


Figure 7.1: Global beautification.

Suppose user draws a figure 7.1(a) and want the figure to be globally beautified as in Fig 7.1(b), where the connectors are aligned to axes and even the symbol is also aligned. One More layer of beautification is added, after recognition of symbol properly user drawn symbols can be replaced by the a standard defined symbol as

described in section 1.2.3. Hence, all the symbol of the same class will look same to a standard symbol. I will work on this in next phase of the project.

7.2.4 Symbol predictor

Suppose immediate recognition mode is ON while drawing a symbol in the diagram and if our system identifies or predicts what our user is trying to draw, we can give an auto complete option to the user which can be more interactive for the user.

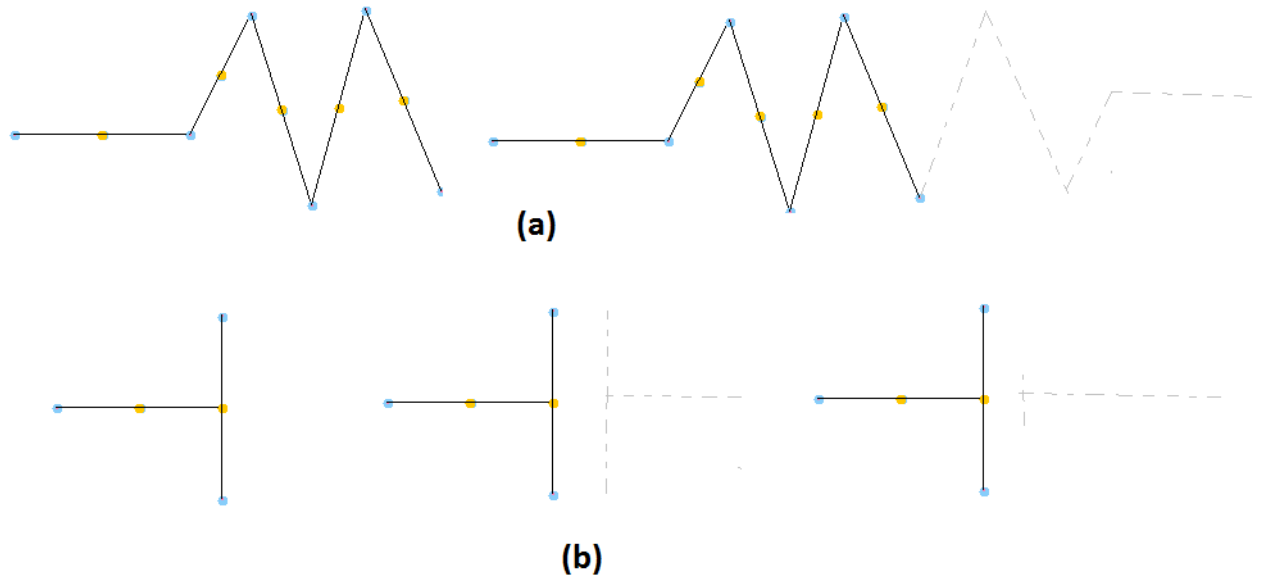


Figure 7.2: Symbol prediction

Suppose a user is trying to draw a resistor as in Fig 7.2, after few strokes, the system detects it as a resistor and gives an auto complete option to the user. Or can give suggestion another symbol as autocomplete, for example, capacitor or battery in (b). For this, we can add a separate panel that shows the predicted symbol with the current knowledge of the sketch. In the next phase of the project, I will try the above stated.

7.2.5 Fuzzy classification of primitives

In our system while beautifying a raw stroke as a line or curve, we find the closest fit to the raw stroke and assign it to corresponding primitive. However, the changing of miss-recognized stroke can be done by selecting the segment and changing it to other primitive from the edit panel or from the CONVERT option in the toolbar. We can have a fuzzy classification, instead of 0-1(0-Line,1-curve) classification for line and curve primitives. A segment with value 0.6 can be classified as a curve, but can be misjudged by our system. This fuzzy classification can be helpful in handling noisy test cases in the symbol classification process. For this, we can have an option in the toolbar or menubar panel which can be chosen when we want the classification of symbols to be done by handling the fuzzy stroke into account. In the next phase of project, I will try to integrate it in DrawCAD.

7.2.6 Domain Independence

The aim is to build a domain independent sketch Recognition system, That is a generalized multi-domain system to draw class diagrams, data flow diagrams, flow charts, graphs, network configuration, stickman figures, UML diagrams, state machines etc. The challenge here will be here how to detect the candidate symbol? what should be our feature set in the recognition process? Does the order of drawing matters?

7.2.7 Solver to the circuit Diagram

Suppose we have a circuit diagram sketch where the symbols are identified and their values (for example 40 ohms for resistor) are given. The aim is to attach a solver that can process all the information of the circuit diagram and is able to calculate the current flow in the various connector or symbols of the circuits or potential drop across the resistor or capacitor. The factors to be considered here is, how to represent the model of the circuit diagram? how to model the symbols present in the sketch? This can be a very effective tool in understanding and explaining of an electric circuit diagram.

Appendix A

Description of Architecture of DRAWCAD

DrawCAD [9, 7, 6] is an IIT Bombay project for engineering diagram. This is different from the traditional drag drop drawing tools. It leverages sketching to draw the 2-D technical diagrams. This tool is not a commercial tool, it is more of a proof of concept.

It allows a user to draw/edit freehand sketches, the system recognizes it and show the beautified drawing as output by converting strokes to primitive segments. It also allows the user to enforce geometric constraints on the drawing. The aim is to use natural way drawing to draw sketches with the powerful features of existing CAD tools. In this Chapter, we will Discuss the basic architecture of DrawCAD.

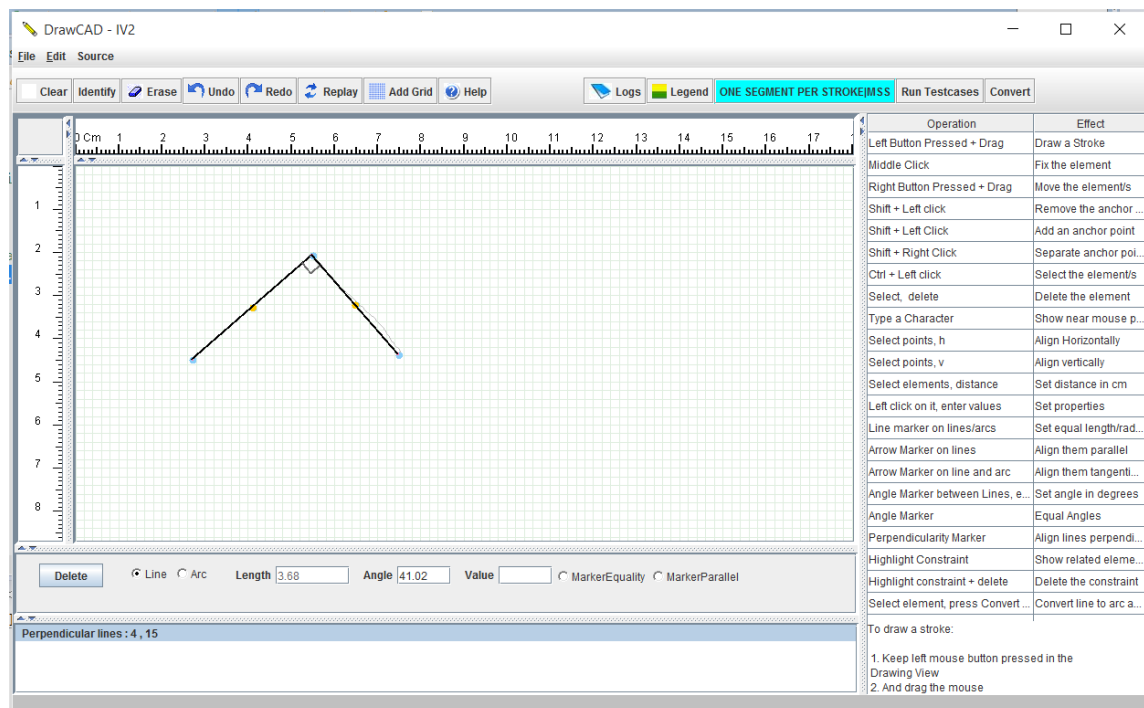


Figure A.1: Screen shot of DrawCAD

Figure A.1 shows a screen shot of the DrawCAD interface. The main window consists of six important panels.

- **Drawing View:** It is the area where the sketch is drawn. In this area user can draw the strokes by using the mouse left button and dragging it. Right click can also be used to nudge an element. Also, Hard constraints are drawn on the sketch itself. The user can also select an object by ALT+ Left Click.
- **Constraint Window:** It is bottom left window as shown in the figure. Which shows the list of constraints associated with the selected segment such as parallel line constraints. equal line constraints. Now, the object information can be seen in the constraint window, if user hovers the mouse on the object.
- **Edit Window:** It is the window above the constraint window and below the Drawing area. Mainly used for manual editing of the strokes such as changing the length of the strokes, radius of arc, changing a line segment to arc segment etc.
- **Help Window:** It is the right side panel, which guides the user while using the system.
- **menu bar:** A menu bar is just like a typical menu in any Windows based application. The File menu has options like save the current drawing, save an object as a reference, save an object as the test case, generate netlist file etc. Edit menu has options like clear the drawing area, erase the selected segment, undo, redo and replay option etc.
- **Toolbar** DrawCAD toolbar has options for Segmentation modes(OSS/MSS), Identify button to start identifying the objects in the sketch, legend to see various color codes, logs show the user moves or steps in the drawing process etc.

Bibliography

- [1] Christine Alvarado and Randall Davis. Sketchread: a multi-domain sketch recognition engine. In *In UIST 04 ACM symposium on User interface software and technology (2004)*, pages 23–32, 2004.
- [2] Wei Deng, Lingda Wu, Chao Yang, and Zhongwen Zhao. A structural method for online sketched symbol recognition. *JSW*, 8(9):2213–2217, 2013.
- [3] Pedro Domingos and Michael Pazzani. Beyond independence: Conditions for the optimality of the simple bayesian classifier. In *Machine Learning*, pages 105–112. Morgan Kaufmann, 1996.
- [4] Leslie Gennari, Levent Burak Kara, Thomas F. Stahovich, and Kenji Shimada. Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Comput. Graph.*, 29(4):547–562, August 2005.
- [5] Vivek kantariya. Front-end for electrical circuit simulators. *Master Thesis Report, IIT Bombay*, 2015.
- [6] VISHAL KHANDELWAL. *Master Thesis Report, IIT Bombay*, 2007.
- [7] SUNIL KUMAR. Drawcad-iii. *Master Thesis Report, IIT Bombay*, 2010.
- [8] WeeSan Lee, Levent Burak Kara, and Thomas F. Stahovich. An efficient graph-based symbol recognizer. In *Proceedings of the Third Eurographics Conference on Sketch-Based Interfaces and Modeling*, SBM’06, pages 11–18, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.
- [9] Abhiram Ranade and Shripad Sarade. Drawcad: mouse-sketch-based engineering drawing. In *Proceedings of the 11th Asia Pacific Conference on Computer Human Interaction*, pages 344–353. ACM, 2013.
- [10] T. M. Sezgin and R. Davis. Sketch recognition in interspersed drawings using time-based graphical models. *Comput. Graph.*, 32(5):500–510, October 2008.
- [11] Tevfik Metin Sezgin and Randall Davis. Hmm-based efficient sketch recognition. In *Proceedings of the 10th International Conference on Intelligent User Interfaces*, IUI ’05, pages 281–283, New York, NY, USA, 2005. ACM.
- [12] Tevfik Metin Sezgin, Thomas Stahovich, and Randall Davis. Sketch based interfaces: Early processing for sketch understanding. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH ’06, New York, NY, USA, 2006. ACM.

- [13] Bo Yu and Shijie Cai. A domain-independent system for sketch recognition. In *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, GRAPHITE '03, pages 141–146, New York, NY, USA, 2003. ACM.