AMNA MASOOD
01-134132-025

Automata Studio (FSA/TG)

Bachelor of Science in Computer Science

Supervisor: Ghulam Ali Mirza

Department of Computer Science
Bahria University, Islamabad

May 2017

Certificate

We accept the work contained in the report titled “AUTOMATA STUDIO”, written by Ms. Amna Masood as a confirmation to the required standard for the partial fulfillment of the degree of Bachelor of Science in Computer Science.

Approved by . . . :

Supervisor: Ghulam Ali Mirza (Assistant Professor)

Internal Examiner: Dr. Sabina Akhtar (Assistant Professor)

External Examiner: Bilal Pervez Khokhar (Senior Lecturer)

Project Coordinator: Dr. Arif Ur Rehman (Assistant Professor)

Head of the Department: Dr. Faisal Bashir (Associate Professor)

May 15th, 2017

Abstract

As we know students feel difficulties in doing their assignment specially when it requires some hand written work. Finite state machine is one of the thing that requires hand written diagrams and students feel very frustrated in drawing them over and over again. Because of these reasons I have developed this application which accepts hand drawn images as an input, and generates the graphical design view of output same as input. User can then save the output in form of image, or can edit it. User can create FSA's newly from the scratch, and can save it in form of image. Automata Studio is a system developed for drawing finite state machine with the help of graphics and image processing techniques. This application has been designed for the ease of students and teachers who has to draw the diagrams again and again. Clean and visible diagrams can be drawn by the help of this application. This application is for now being limited to the drawing of the finite state machines. But in future, (INSHALLAH) this project can be extended for generation of Regular Expressions, Transition Graphs, Push-down Automata.

Acknowledgments

In the name of ALLAH the most Gracious, the most Merciful. First of all I would like to thank Allah for his countless blessings. I would like to give a special thanks to my family and friends who helped me a lot in finalizing this project within the limited time frame. They have always supported me and have always been so encouraging and motivated in every step on life. Without their prayers and support, I would have not been able to achieve this.

Last, but not the least, I would like to express my very great appreciation to my supervisor Sir Ali Mirza for his valuable and constructive suggestions at every step of planning and development of this project. His willingness to give his time and being so patient all the time is appreciated.

AMNA MASOOD
Bahria University Islamabad, Pakistan

May 2017

"We think someone else, someone smarter than us,
someone more capable, someone with more resources will solve that problem.
But there isn't anyone else."

Regina Dugan

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

FSM	Finite State Machine
FSA	Finite State Automata
GDI	Graphics Device Interface
GLCM	Gray Level Co-occurrence Matrix
PDA	Push Down Automata
SVM	Support Vector Machine

Chapter 1

Introduction

1.1 Project Background/Overview

Finite-State Machines (FSM), also called Finite State Automata (singular: automaton) or just finite automata are much more restrictive in their capabilities than Turing machines. Building a FSM is a very different form of process. These machines requires each state to be connected with an explicit transition to the next state. No programming language requires this, everything is done implicitly based on the semantics of the language itself (e.g. a C++ compiler builds sequences from statements). The process of editing the logic of a FSM is very low-level and quite mechanical. You often find yourself rebuilding the similar behaviors over and over again from scratch, which takes a lot of time. All you can do is edit transitions from one state to another. This application will be designed for desktop computers. This system generates Finite State Machine also known as Transition Graph. This application uses image processing techniques to read image from external source (like camera). Then loads it in the editor and offers basic tools for working with Finite Automata, such as basic algorithms for Finite Automata. Image recognitions, shapes detection, text extraction and all techniques will be used in this software [3]. This software will be an open source software which will be available for everyone. One most important benefit of this software is that user can load an image into the editor, which no other editor can do.

1.2 Problem Description

In daily life people finds it very difficult to draw a finite state machine. With just one little mistake they have to draw it over and over again. It takes a lot of time too. Teachers find it tough to check, or to draw FSM for so many students, as well as students also faces difficulties in doing assignments. Benefit of this software is that teachers and students will be able to draw and read FSM without any difficulty and waste of time.

1.3 Project Objectives

The objective of this project is to detect FSA through a captured image (or may be scanning) to create a finite state machine in the Automata Studio. Basically developed for the ease of teachers and students.

1.4 Project Scope

Automata Studio will be an open source application, easily available to everyone. This will help the user to load an image into the software, which will read the state machine diagram, and editing can be done over it. System will provide following functionality:

- Image loading from camera and creating FSA of that image
- Creating newly FSA and applying all editing functionality
- Insertion of state in existing FSA
- Deleting any state
- Modification in FSA

This software is expected to create regular expressions as well, on the basis of finite state machines. This system can be modified to generate ’Turing Machine’, as well as ’Push Down Automata (PDA)’. But as I am doing this project alone, so main focus and concern of this project is to read finite state machine using image processing techniques, and providing editing options to the user. If these are requirements are timely fulfilled, I might extend my work to generate regular expressions.

Chapter 2

Literature Review

2.1 Introduction

In this Literature review, our aim is to provide a detail study about the work that has been done by the researchers on finite state machine (automata) through which this software has been made i.e. Automata Studio. Many different approaches and algorithms are used by the researchers to implement such systems which can be more efficient and optimal.

2.2 Circle and Lines Detection

In an image processing application detection of lines and circle is the fundamental step.

2.2.1 Hough Transformation

In image processing Hough transform is one of the famous feature extracting technique. It's used to identify lines, circles, and curve structured objects in an image. Kerbyson and Atherton conducted a research on Hough transformation to detect circles of different radii using single parameter space. They expressed a modernistic circle Hough transformation technique using oriented and distant information for the circle position accuracy. [1]

2.2.2 Inverted Gradient Hash Maps

Hough transform is not always the efficient algorithm in field of image processing. While there are many other algorithms present for the detection of line and circle but they are all slow. However R.Gonzalez has presented an algorithms which is fast enough to detect line and circles using inverted gradient hash maps. This algorithm computational time by reducing the processing pixels of an image. [2]

2.2.3 Gradient Threshold

Detection of line is a classical problem in computer vision and image processing field. Most of the common algorithms uses gradient magnitude and direction technique. Some researchers proposed a flexible and robust method using gradient threshold with the weight mean shift procedure. According to their experiments there method is most accurate when compared with the Hough transformation, line detector, and other algorithms. [3]

2.3 Text Detection

Text detection and recognition from an image is one of the most challenging task in image processing applications.

Ye et al made a research on text detection and recognition in an image and they discuss the problems and challenges, proposed schemes and performance of text detection and recognition in an image. They have highlighted some special issues associated the processing of text. Fundamental comparisons and analysis are also discussed in there research. [4]

In another research, a new algorithm is proposed for the detection of text in still images. This algorithm is based on the truncation error generated by high contrast edges of text boundaries when image is compressed to .JPEG. After many experiments on a set of hundreds of image they have concluded that the new algorithm is an effective and gives good accuracy in detecting overlay text. [5]

Xu-Cheng Yin and members have made research in which they proposes a framework learning and multi-oriented scene text detection system. There proposed system is so effective that it is graded on several public scene text databases. [6]

2.4 Critical Evaluation

A lot of research has been made in the detection of circle and line. Many new fast and efficient algorithms have been proposed. Different techniques are being implemented to produce good results.

Chapter 3

Requirement Specifications

3.1 Existing System

There are many existing online web application based FSM generator available online which allows you to create finite state machines. There are also many websites which allow you to generate finite state machines based on regular expressions.

3.1.1 Limitations

This current system is limited to the detection and drawing of FSA. But can be extended to generate regular expression based on the FSA image drawn.

3.2 Proposed System

The proposed system is an open source software that will allow the user to create finite state machines. Along with the editing options. User can load images and software will create the FSA just like made on image, as well as user can create FSA from the start which can be saved as an image. This proposed system is a complete studio which creates a finite state machine (transition graph) using image processing techniques. This system can also a FSM editor.

3.3 Requirement Specification

3.3.1 Functional Requirements

The functional requirement from the user point of view are as following:

1. **User Requirements (FSA drawn via image)**

    - Start the application

- Load an image
- FSA is drawn on the screen based upon the FSA in image
- Save the image
- Print the image
- Close the image

2. **User Requirements (FSA drawn from scratch)**

   - Start the application
   - Load an image
   - FSA is drawn on the screen using toolbox by drag and drop
   - Save the image
   - Print the image
   - Close the image

3. **Benefits**
   Benefits of this software are that this is specially made for ease of teachers and students who needs to draw finite state machines over and over again. This software will reduce their time and will help them in their work.

4. **Technical issues**
   This software might have some technical issues:

   - if image is not scanned properly.
   - image might take some time in loading and processing.
   - result may vary from the image loaded if not scanned properly.

5. **Risks**
   This system involves few risks, which are following:

   - if in an image the objects are not drawn with some space, then it might produce failed results.
   - shape detection and text recognition may fail some time.

3.3.2 Non-Functional Requirements

The non-functional requirement of the system are following:

1. **Performance**
   The system should be efficient and performing the FSA design without any noticeable delay by the user.

2. **Reliability**
   The system should reliably segment and recognize the shapes, lines and the text. The position of shapes, line with respect to the image as well as with respect to one another should also be reliably drawn so that the drawn FSA exactly like the one made on image.
3. **Portability**
   The software is a windows based application and can be easily installed on any operating system and can be used effectively.
4. **Security**
   The system does not have any specific security requirements.
5. **Availability**
   The system does not requires any Internet connectivity and will be available once the system is installed on device.
6. **Maintainability**
   The maintenance of the application will be carried out by the developers, if required.

3.4 Use Cases

Following is the full system Use-Case diagram.

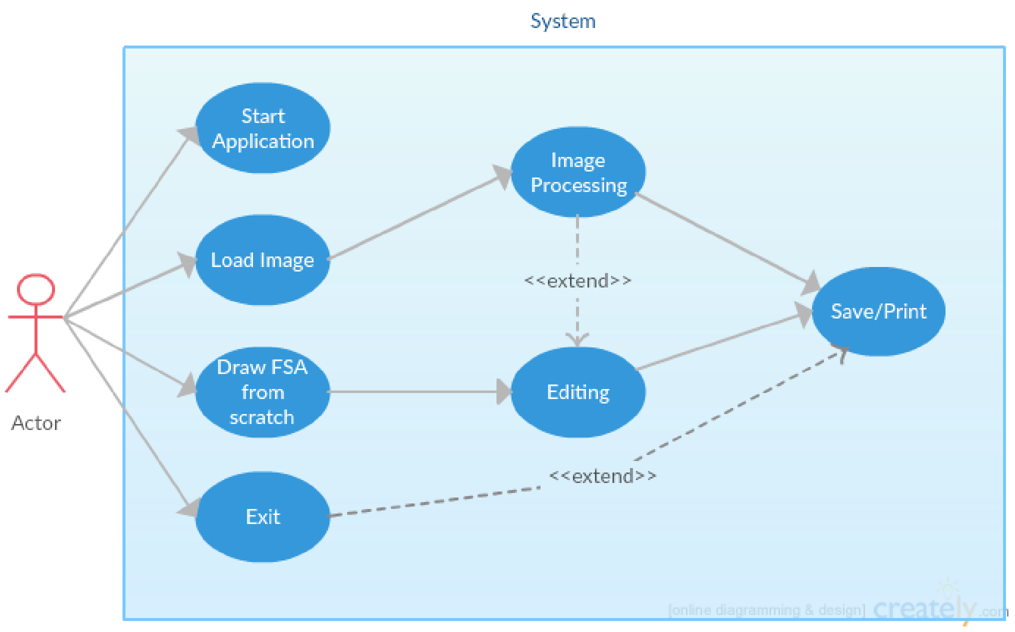


Figure 3.1: System Use-Case Diagram

Following are the detailed tables of each use case.

3.4.1 Use Case: 1 Start Application

Use Case Number	UC-1
Use Case Name	Start Application
Description	User has to start the application by clicking on the logo button.
Actors	User
Pre-Conditions	None
Basic Flow	1. Click on the application icon 2. Application will be launched
Post-Conditions	Application will launch

Table 3.1: Use Case: Start Application

3.4.2 Use Case: 2 Loads Image

Use Case Number	UC-2
Use Case Name	Loads Image
Description	User loads image into the system.
Actors	User
Pre-Conditions	Application should be launched
Basic Flow	1. Application launches 2. User loads an image in to the system
Post-Conditions	Image processing techniques will be applied

Table 3.2: Use Case: Loads Image

3.4.3 Use Case: 3 Draw FSA from scratch

Use Case Number	UC-3
Use Case Name	Draw FSA from scratch
Description	User can,also draw FSA from the start without loading image. User can drag and drop,the objects from the toolbox.
Actors	User
Pre-Conditions	Application should be launched
Basic Flow	1. Launch the application 2. From the toolbox, drag the circles 3. Drag the lines 4. Insert text
Post-Conditions	Editing can be applied

Table 3.3: Use Case: Draw FSA from scratch

3.4.4 Use Case: 4 Image Processing

Use Case Number	UC-4
Use Case Name	Image Processing
Description	When user loads an image in system, system processes some functions over it to extract the required data
Actors	User
Pre-Conditions	Application should be launched and Image should be loaded
Basic Flow	1. User loads an image 2. System converts bitmap image to un-managed image 3. System converts image into gray-scale image 4. Sauvola threshold is applied 5. System uses canny edge detector to detect edges 6. Image passes through morphological steps 7. Through connected components objects are separated 8. Blobs are detected and are separated through neural network testing 9. Objects are then classified into groups and shapes and are drawn over the screen as shown in image.
Post-Conditions	Objects,are drawn and image can be saved or print as respect to user choice.

Table 3.4: Use Case: Image Processing

3.4.5 Use Case: 5 Editing

Use Case Number	UC-5
Use Case Name	Editing
Description	When user,loads image or draw it by itself, further more options are provided to user, to modify the image which has been drawn by the system.
Actors	User
Pre-Conditions	Image should be loaded or drawn by the user
Basic Flow	1. If user loads the image 2. After image processing techniques the system draw the FSA as shown in the image 3. Editing options are given to the user 4. Deleting any state(s) 5. Adding more states
Post-Conditions	Saving or printing the image.

Table 3.5: Use Case: Editing

3.4.6 Use Case: 6 Save and Print

Use Case Number	UC-6
Use Case Name	Save/Print
Description	After all the processing user is given the option to save its work
	or directly print it.
Actors	User
Pre-Conditions	Image,should be loaded, Image processing should be applied or editing should be,done by user.
Basic Flow	1. Click on to save option 2. Image will be saved 3. Image can be directly be printed too.
Post-Conditions	Exit the application or load more image to continue the work.

Table 3.6: Use Case: Save/Print

3.4.7 Use Case: 7 Exit Application

Use Case Number	UC-7
Use Case Name	Exit application
Description	User can exit the application by choosing exit option.
Actors	User
Pre-Conditions	Application should be launched
Basic Flow	1. Select exit option from the menu bar 2. Application will be closed 3. Saving or printing the work done option will be given before exiting.
Post-Conditions	None.

Table 3.7: Use Case: Exit application

3.5 Sequence Diagrams

The sequence diagram of the system are as following:

3.5.1 System Sequence Diagram

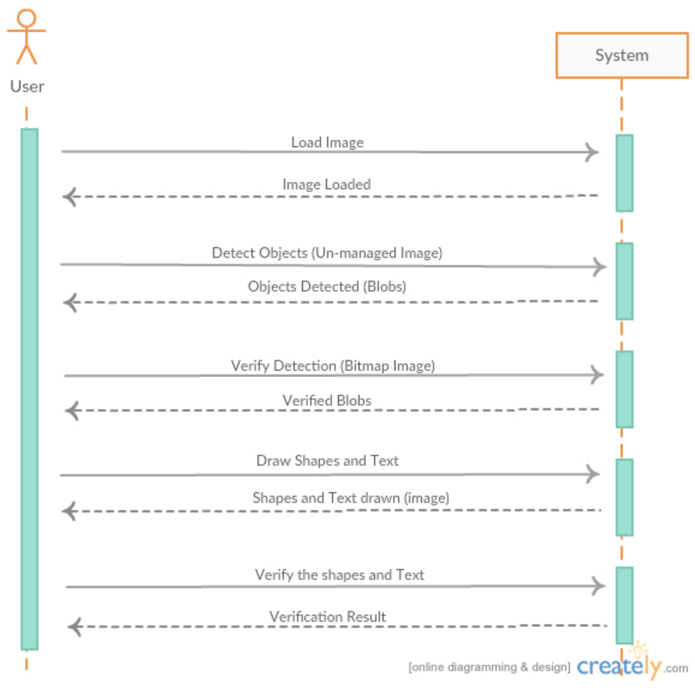

User
System
Load Image
Image Loaded
Detect Objects (Un-managed Image)
Objects Detected (Blobs)
Verify Detection (Bitmap Image)
Verified Blobs
Draw Shapes and Text
Shapes and Text drawn (image)
Verify the shapes and Text
Verification Result
[online diagramming & design] creately.com

Figure 3.2: System Sequence Diagram

3.5.2 Sequence Diagram of Loading Image

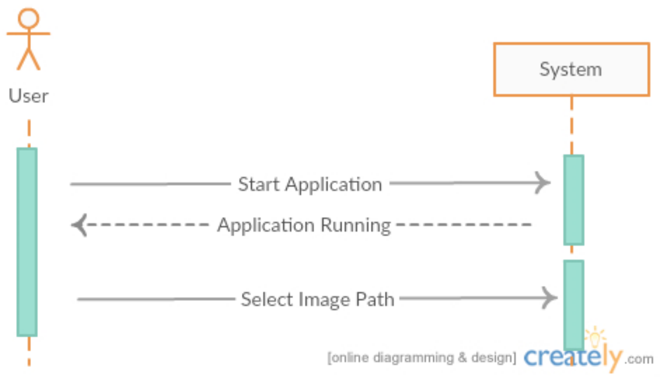


Figure 3.3: Sequence Diagram of Loading Image

3.5.3 Sequence Diagram of Objects Detection

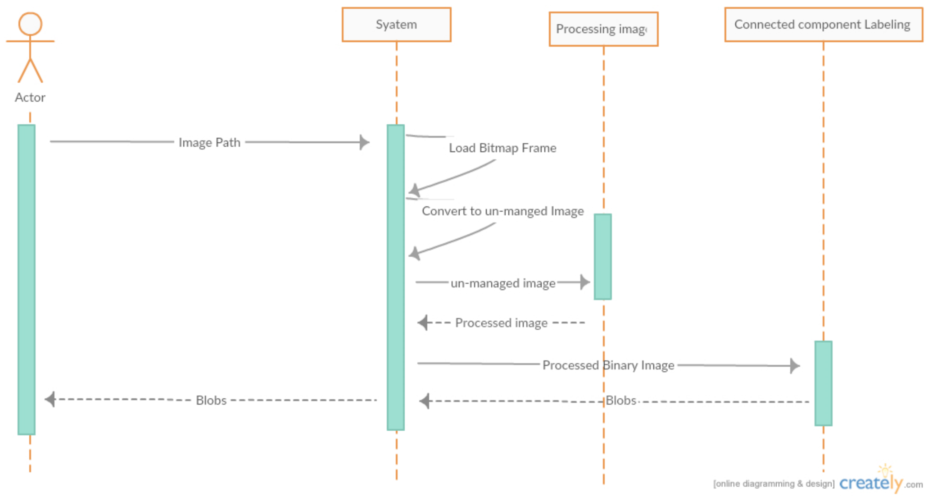


Figure 3.4: Sequence Diagram of Objects Detection

3.5.4 Sequence Diagram of Processing Image

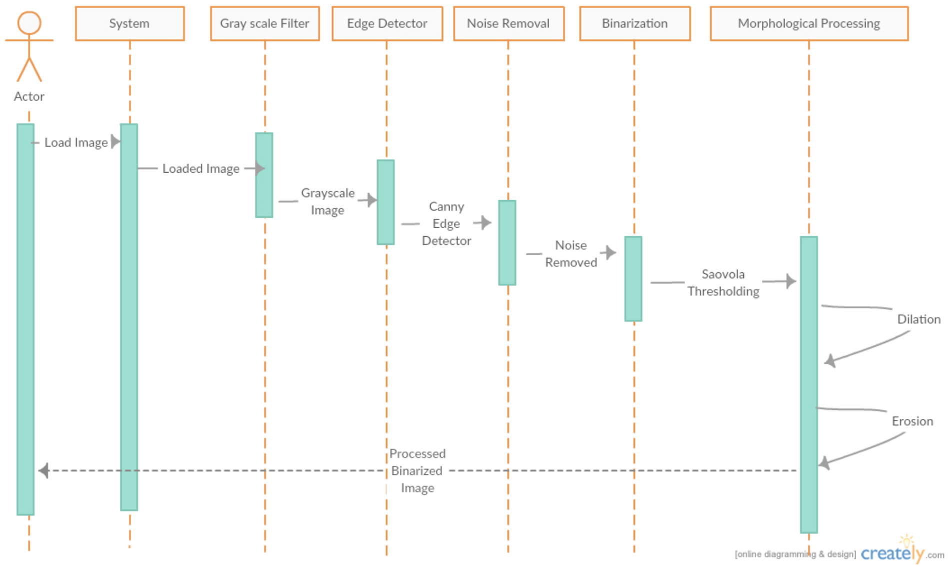


Figure 3.5: Sequence Diagram of Processing Image

Chapter 4

Design

In this chapter the design and architecture of the system are discussed.

4.1 System Architecture

The system being developed comprises of user interface where user loads image, or drags the objects from toolbox to draw finite state machine. System architecture design is shown in the figure 4.1

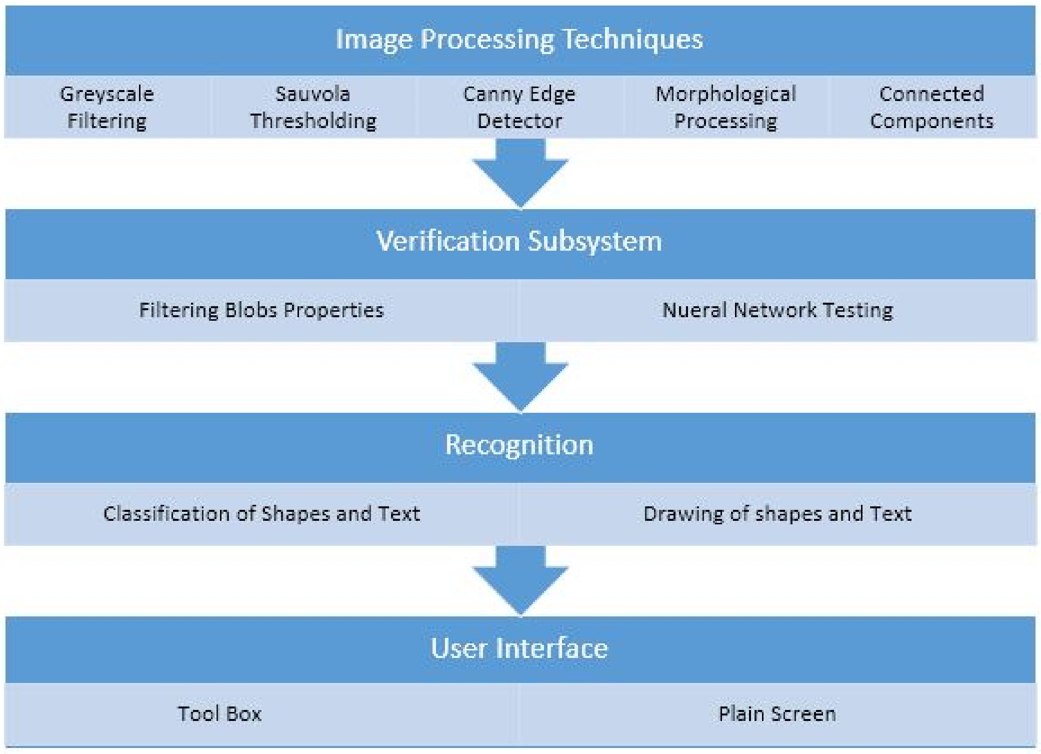


Figure 4.1: System Architecture Design

4.2 Design Methodology

Following figure shows the system design methodology.

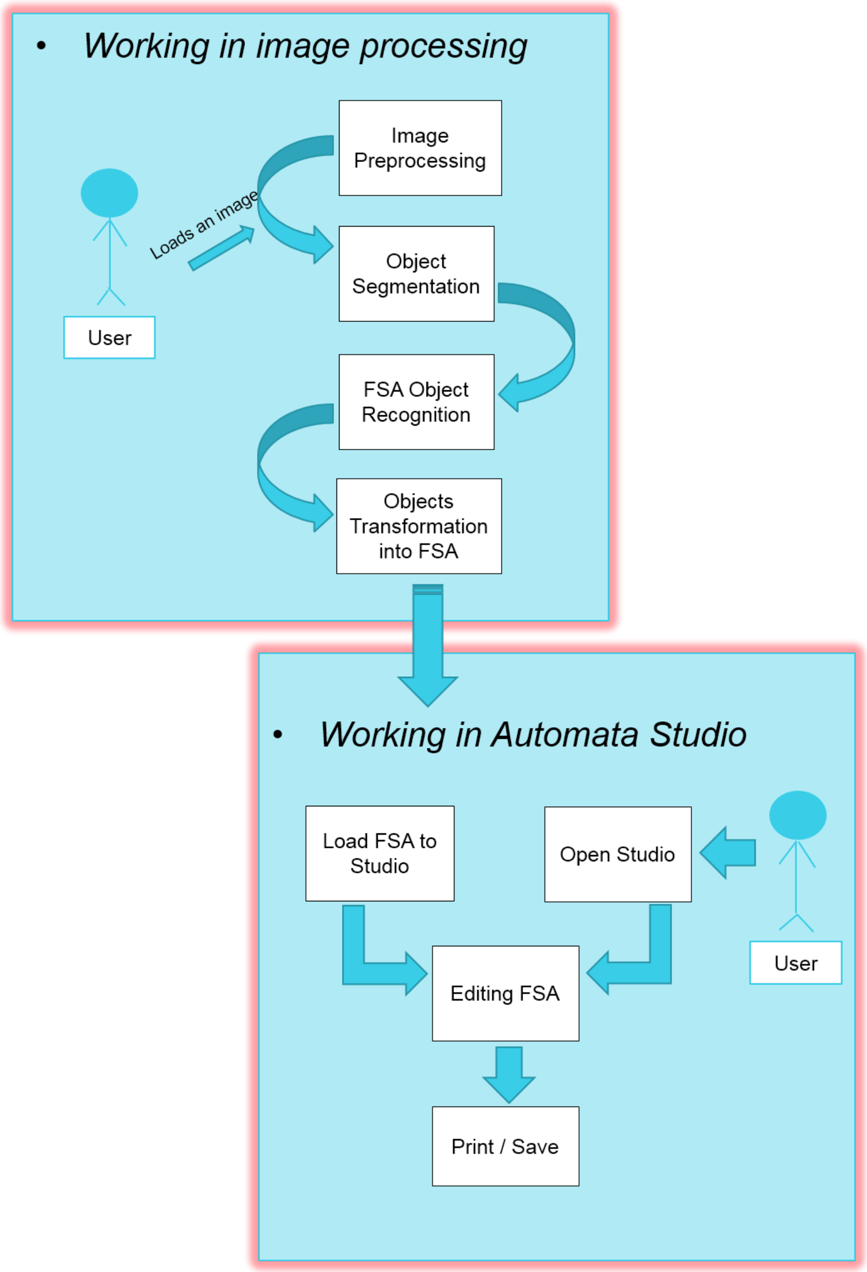


Figure 4.2: System Methodology

4.3 High Level Design

Following is the system high level design.

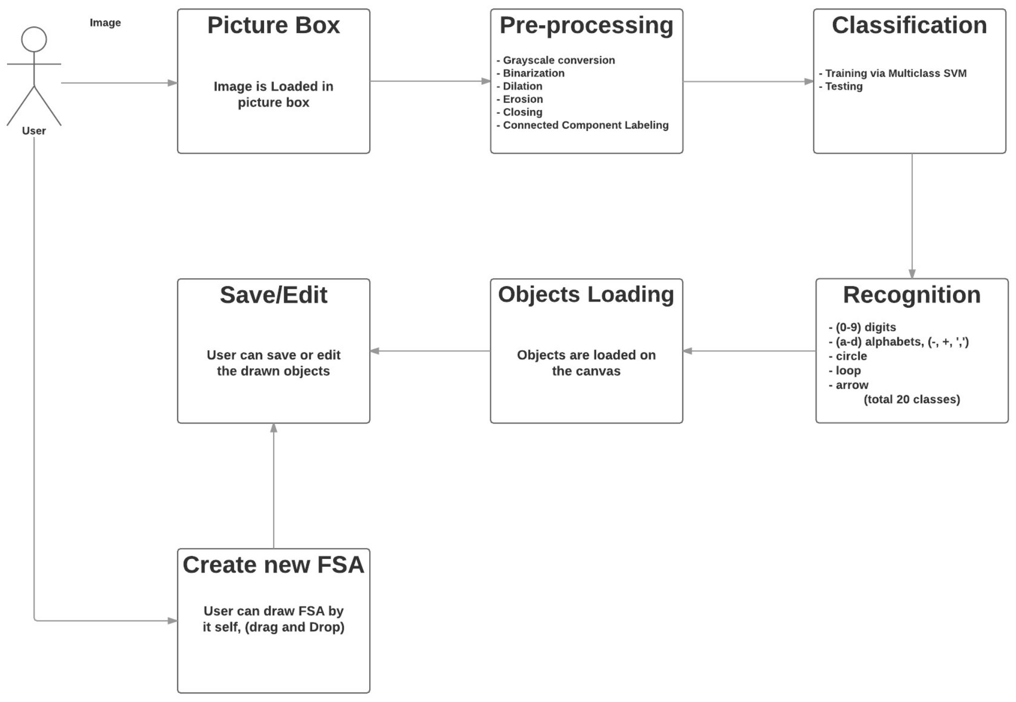


Image
User
Picture Box
Image is Loaded in picture box
Pre-processing
- Grayscale conversion
- Binarization
- Dilation
- Erosion
- Closing
- Connected Component Labeling
Classification
- Training via Multiclass SVM
- Testing
Recognition
- (0-9) digits
- (a-d) alphabets, (-, +, ',')
- circle
- loop
- arrow
(total 20 classes)
Objects Loading
Objects are loaded on the canvas
Save/Edit
User can save or edit the drawn objects
Create new FSA
User can draw FSA by it self, (drag and Drop)

Figure 4.3: System High Level Design

4.3.1 System Flow Diagram

Following diagram shows the flow of the system.

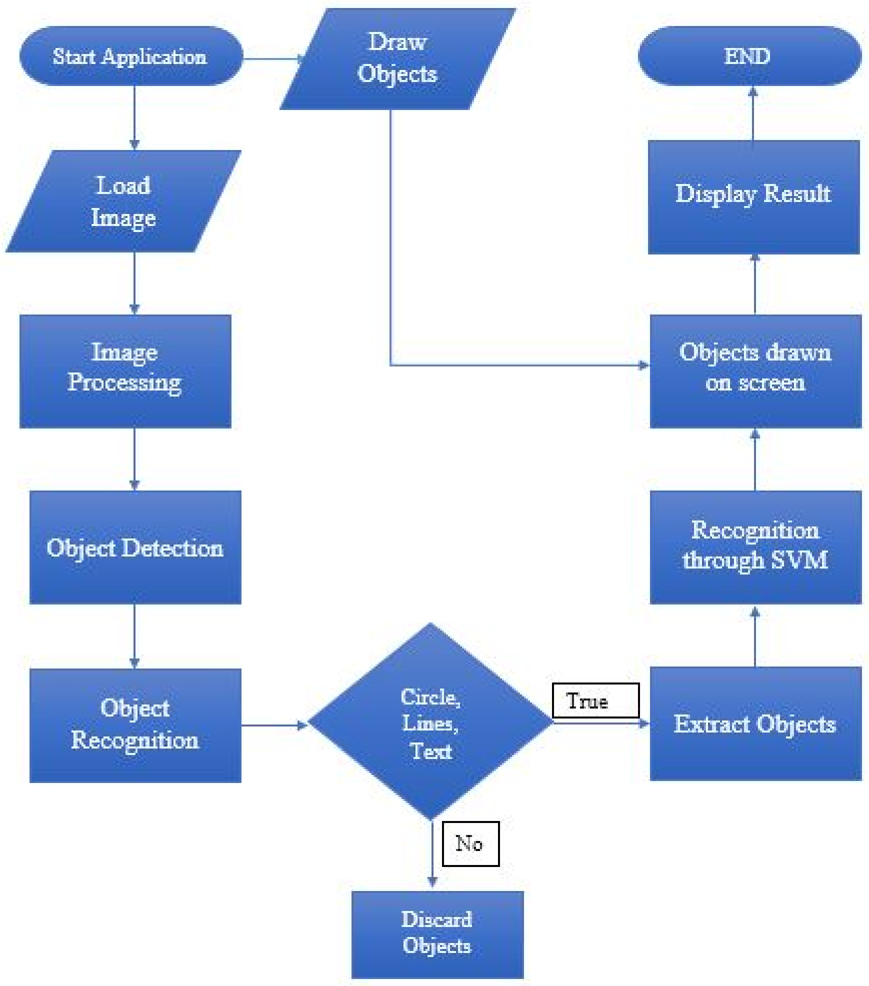


Figure 4.4: System Flow Diagram

4.4 Low Level Design

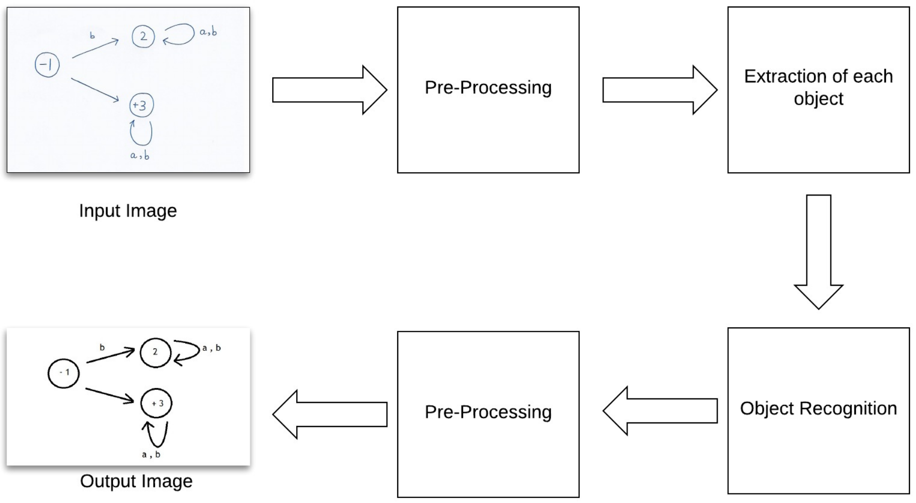


Figure 4.5: System Low Level Design

4.5 Class Diagram

Following figure is the class diagram of the developed system.

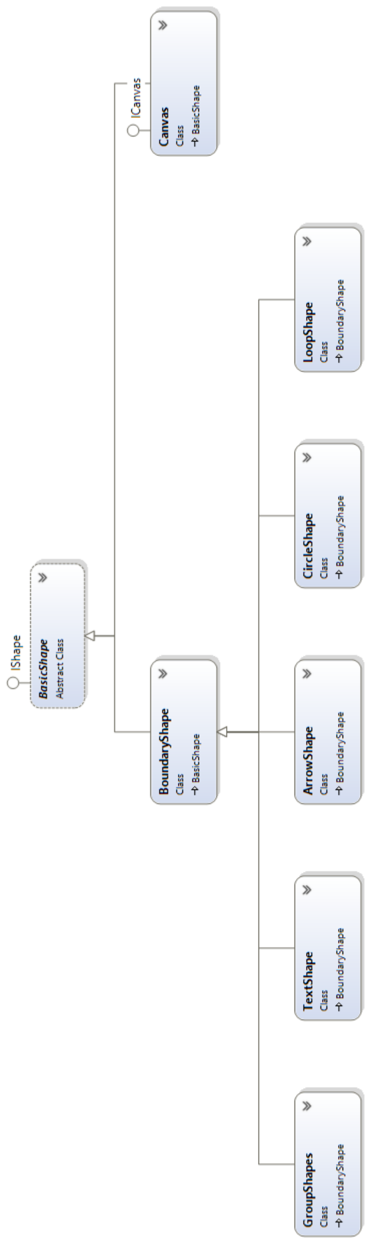


Figure 4.6: System Class Diagram

Chapter 5

System Implementation

This chapter consists of system implementation, detailed description of tools and technologies used in this project along with the algorithms.

5.1 Tools and Technology

This system uses following tools for the recognition and drawing of the objects.

5.1.1 Visual Studio 2013

Microsoft visual studio 2013 uses a platform where integrated environment systems are developed. This software application provides different programming languages code editor. This software can also be used for applications such as GUI, database application, web designing etc. This platforms supports C++, C#, Visual Basic, Web developer and many more. Whereas this proposed system is developed using C-Sharp (C#) language.

5.1.2 AForge.NET

AForge.Net is a framework which supports C# language, consisting several libraries for computer vision and artificial intelligence. This framework provides a scientific computing framework. Its different libraries also available in executable programs giving a wide range to scientific computing applications, such as feature extracting, machine learning, and pattern recognition, including computer vision and computer audition.

5.1.3 Accord.NET

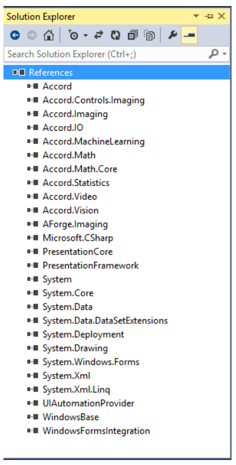
Accord.net is an extended framework of AForge.NET in field of computer vision and artificial intelligence. It is a machine learning platform consisting of audio and image processing libraries. These libraries are written in C-sharp language and are available in form of source code and nugget packages. This framework is very good for commercial use applications. Following are the few libraries which are used in this project.

- **Accord.Imaging** - This library contains interested point detectors (such as Surf, Fast, Harris and Freak), image filters, image matching and image stitching methods, as well as feature extractors such as Histograms of Oriented Gradients and Haralick's textural feature descriptors.
- **Accord.Control** - Histograms, scatter plots and tabular data viewers for scientific applications.
- **Accord.Controls.Imaging** - Windows Forms controls to show and handle images.
- **Accord.Statistics** - Contains probability distributions, hypothesis testing, statistical models and methods such as Linear and Logistic regression, Hidden Markov Models, (Hidden) Conditional Random Fields, Principal Component Analysis, Partial Least Squares, Discriminant Analysis, Kernel methods and many other related techniques.
- **Accord.MachineLearning**- Provides machine learning classes, such as Support Vector Machines (SVM), Decision Trees, Naive Bayesian models, K-means, Gaussian Mixture models.

5.1.4 Graphical Device Interface (GDI+)

GDI+ is a library that provides programming interface to create Microsoft Windows applications and Web Graphics applications that interacts with graphical devices. GDI is very useful for drawing 2D graphics on multiple devices. GDI consists of sets of different programming languages which are used in both managed and un-managed code. Code written in C# language is the managed code. In .Net Framework, provided managed GDI+ classes are defined in *System.Drawing* namespace. This namespace has further five more namespaces which are used in this project. They are:

- System.Drawing.Design
- System.Drawing2D
- System.Drawing.Imaging
- System.Drawing.Printing
- System.Drawing.Text



References
- Accord
- Accord.Controls.Imaging
- Accord.Imaging
- Accord.IO
- Accord.MachineLearning
- Accord.Math
- Accord.Math.Core
- Accord.Statistics
- Accord.Video
- Accord.Vision
- AForge.Imaging
- Microsoft.CSharp
- PresentationCore
- PresentationFramework
- System
- System.Core
- System.Data
- System.Data.DataSetExtensions
- System.Deployment
- System.Drawing
- System.Windows.Forms
- System.Xml
- System.Xml.Linq
- UIAutomationProvider
- WindowsBase
- WindowsFormsIntegration

Figure 5.1: References used in System

5.2 Methodology and Algorithm Development

5.2.1 Image Loading

Samples images are already present in the system, they can be loaded to the picture box and tested even. User can load other pictures as well.

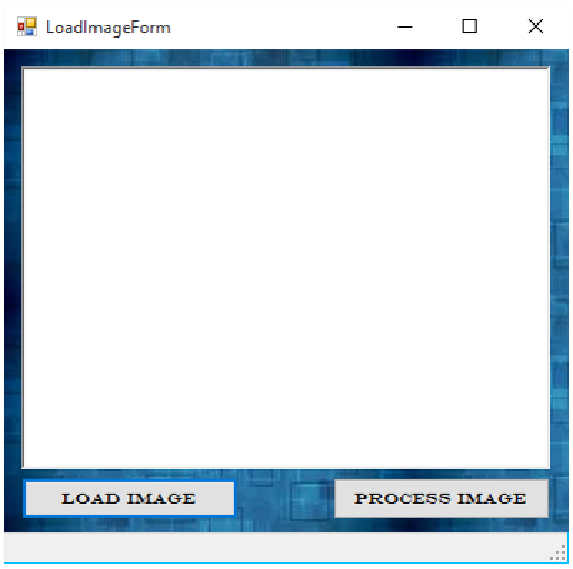
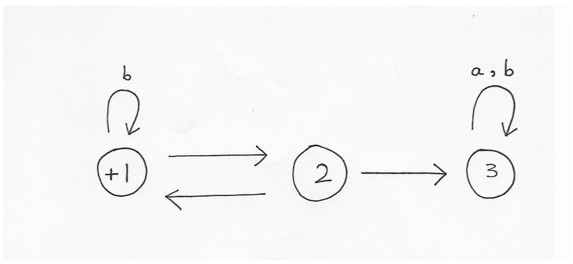


Figure 5.2: Loadin Image

5.2.2 Segmentation

Segmentation is the first step of pre-processing image after loading it in system. It uses Accord libraries for processing.

1. **Gray-scaling**
   First step is to convert 24-bit image (3-channel) into single channel i.e 8-bit gray-scale image. Gray-scale image has 256 colors ranging form 0 (black) to 255 (white). The image is converted into gray-scale using Accord gray-scale filter.



$$\begin{tikzcd}
+1 \arrow[loop above, "b"] \arrow[r, shift left] & 2 \arrow[l, shift left] \arrow[r] & 3 \arrow[loop above, "{a, b}"]
\end{tikzcd}$$

Figure 5.3: Gray-scale Image

2. **Binarization**
   Binarization is the conversion of gray-scale (8-bit) image to binary image. This is done by using Sauvola Thresholding method. Now the image is converted into black and white.

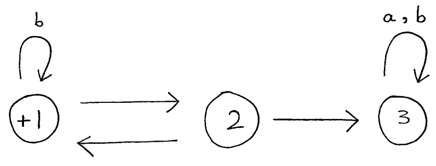


Figure 5.4: Binarized Image

3. **Dilation** The next step is to dilate the image, which fills any small gaps present in the between object. It helps in detecting components.

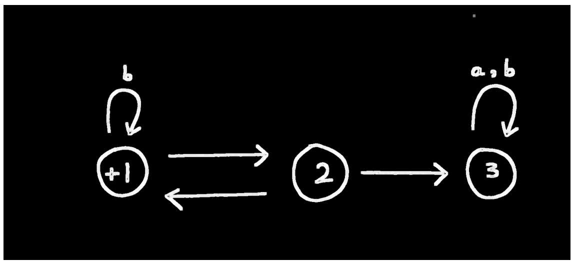


Figure 5.5: Dilation

4. **Erosion** The next step is erosion step, which helps in separation of any two or more components closely made.

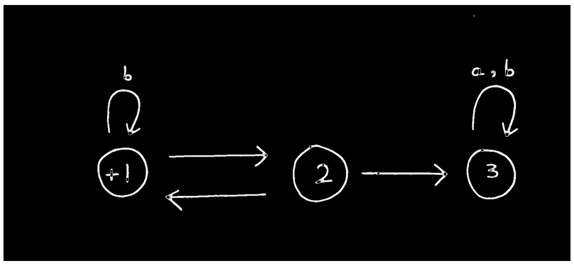


Figure 5.6: Erosion

5. **Connected Component Labeling** the components are extracted using Blob class, which counts and extracts each component in image using connected component labeling algorithm.

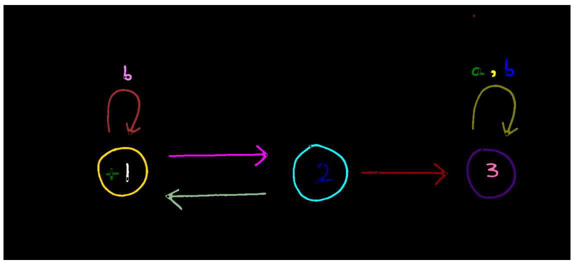


Figure 5.7: Connected component labeling through colors

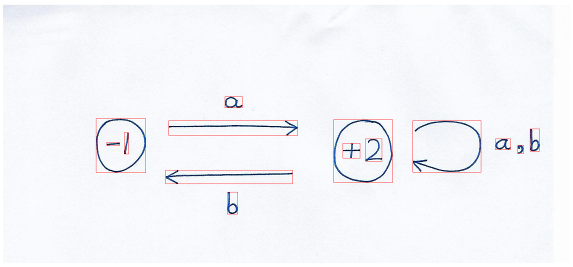


Figure 5.8: Connected component labeling through Rectangles

5.2.3 Recognition

For objects recognition, this problem involves 20 classes, consisting of all objects (circle, arrow, loop, digits (0-9), alphabets(a-z, +,-,',').

1. **Feature Extraction**

   After processing images, features are extracted and saved in excel file for classification stage.

   Gray Level Concurrence Matrix (GLCM)

   Features are extracted using GLCM class which present in Accord.NET. This is a matrix of frequencies at which two pixels, separated by a certain vector, occur in the image called co-occurrence matrix. During feature extraction of each object, class number is assigned to it which helps in classification. GLCM represents the distance and angular spatial relationship over an image sub-region of specific region of specific size and distance. The GLCM calculates, how often a pixel with gray-level value *i* occurs either horizontally, vertically or diagonally to adjacent pixels with the value. GLCM of an image is computed using a displacement vector d, defined by its radius and orientation. Features computed from GLCM are textual information about the image proposed by Haralick. These are Energy, Entropy, Contrast, Correlation, Homogeneity, Angular Second Movement, Contrast, Correlation, Variance, Inverse Difference Moment, Sum Average, Sum Variance, Sum Entropy, Difference Variance, Difference Entropy, Mean of Correlation.

2. **Classification**
   The Support Vector Machine is a supervised learning method for classification and regression. One way to extend the simple Support Vector Machine to multiple classes is to create a one-Vs-one scheme where multiple Support Vector Machines are specialized to recognize each class. For this proposed system Multi-class Kernel Support Vector Machine Classifier is used, which uses an array of *inputs[]*, and array of *outputs[]* and a kernel (Gaussian). Kernel class is also present in Accord.NET. It decides the hyper plane dimensions. For this proposed system, we trained a separate SVM for this system. Inputs and outputs are extracted from Excel file, in which features were saved.

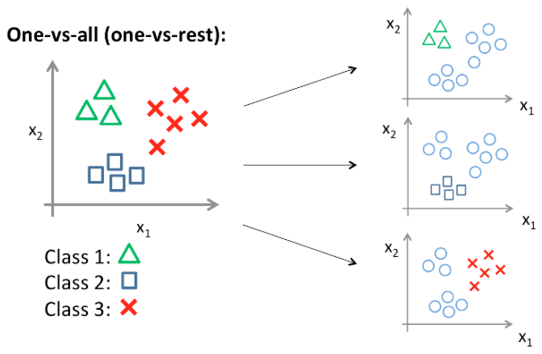


Figure 5.9: Multiclass SVM example

5.2.4 Graphics Loading

After pre-processing the image, required objects are created on the screen (as shown in Figure 5.12) with the help of 2D-graphics. For creating objects on screen GDI+ library is used. User can now Edit the image, or save it directly in form of an image.

5.2.5 Creating FSA from scratch

By the help of using GDI+ libraries, Finite state machine can be created easily by drag and drop method. After creating FSA, user can save the work in form of image and print it later on.

5.3 System Design GUI

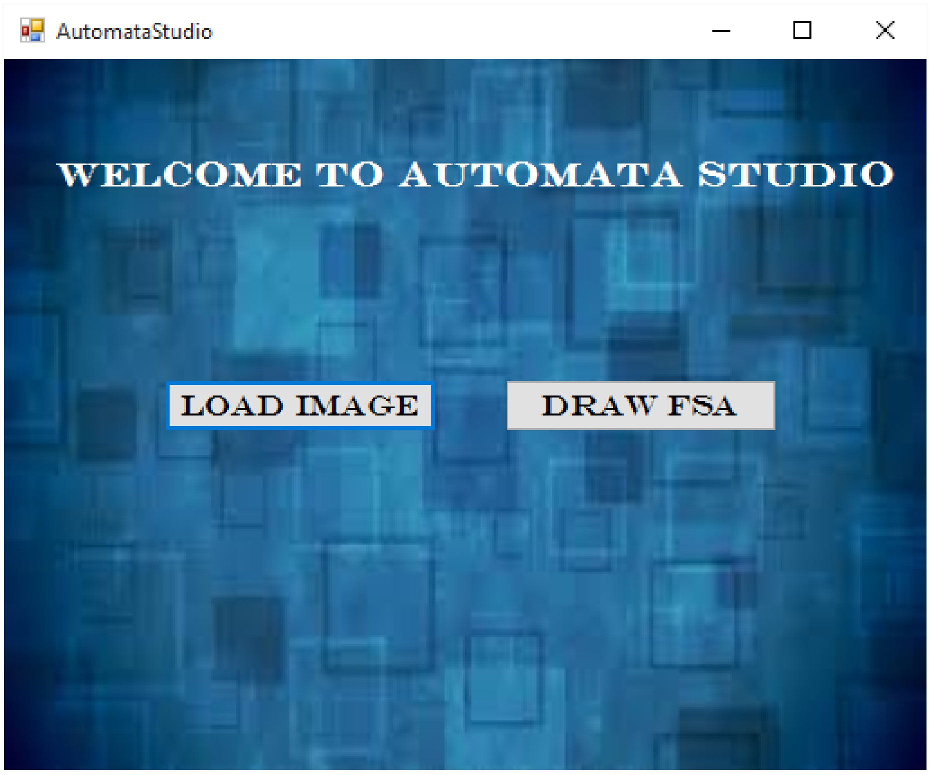


Figure 5.10: Main Form of the System

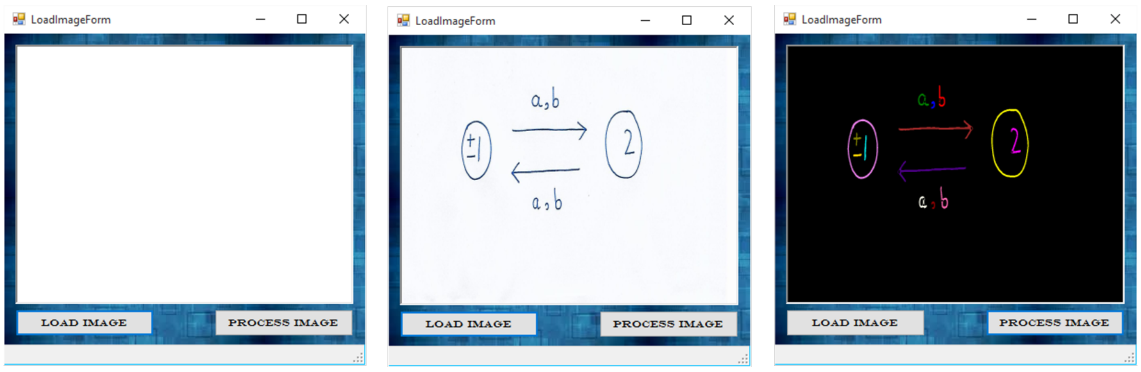


Figure 5.11: (LEFT) Blank Loading Form, (CENTER) Loaded Image, (RIGHT) Processed Image

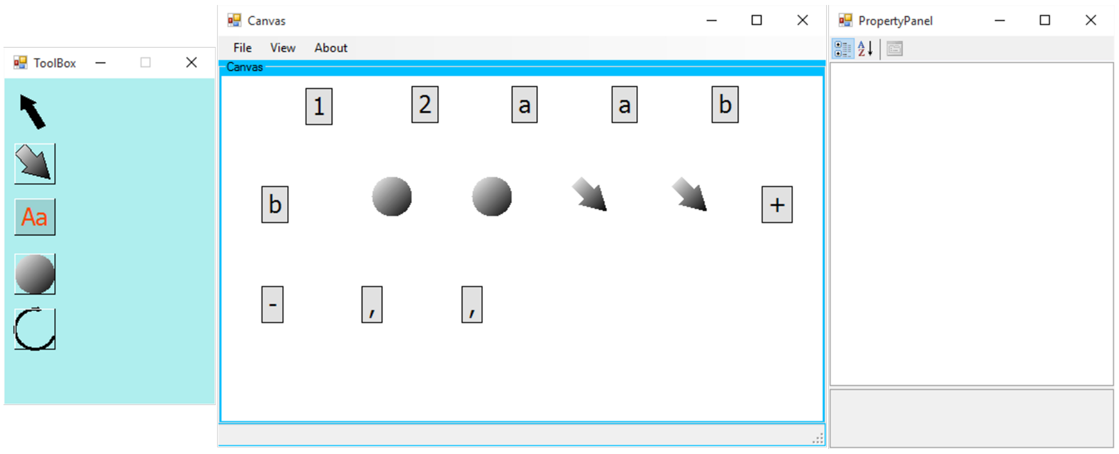


Figure 5.12: Loaded objects on screen

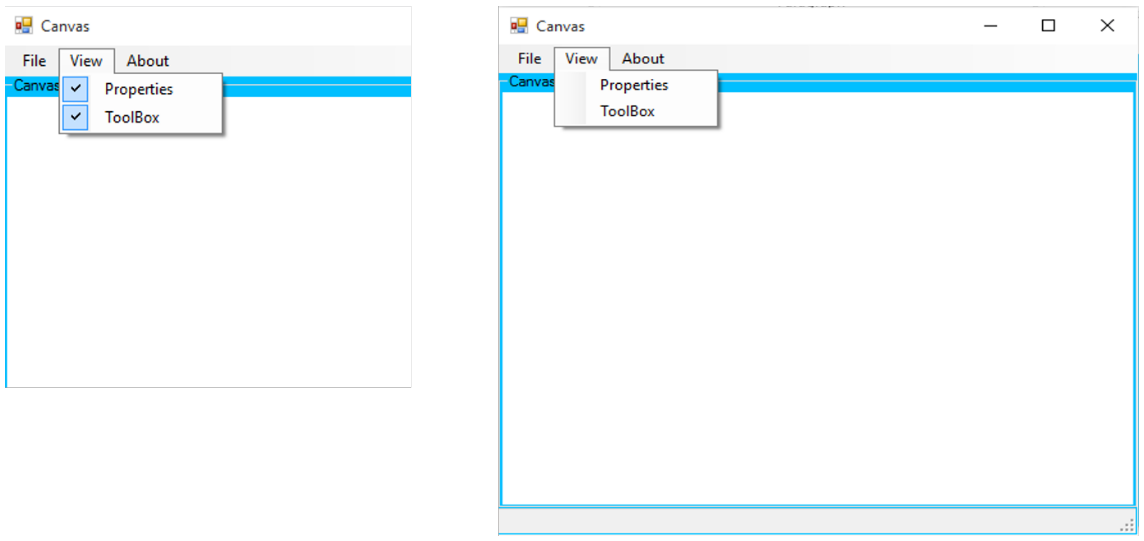


Figure 5.13: Viewing options of ToolBox and PropertyPanel

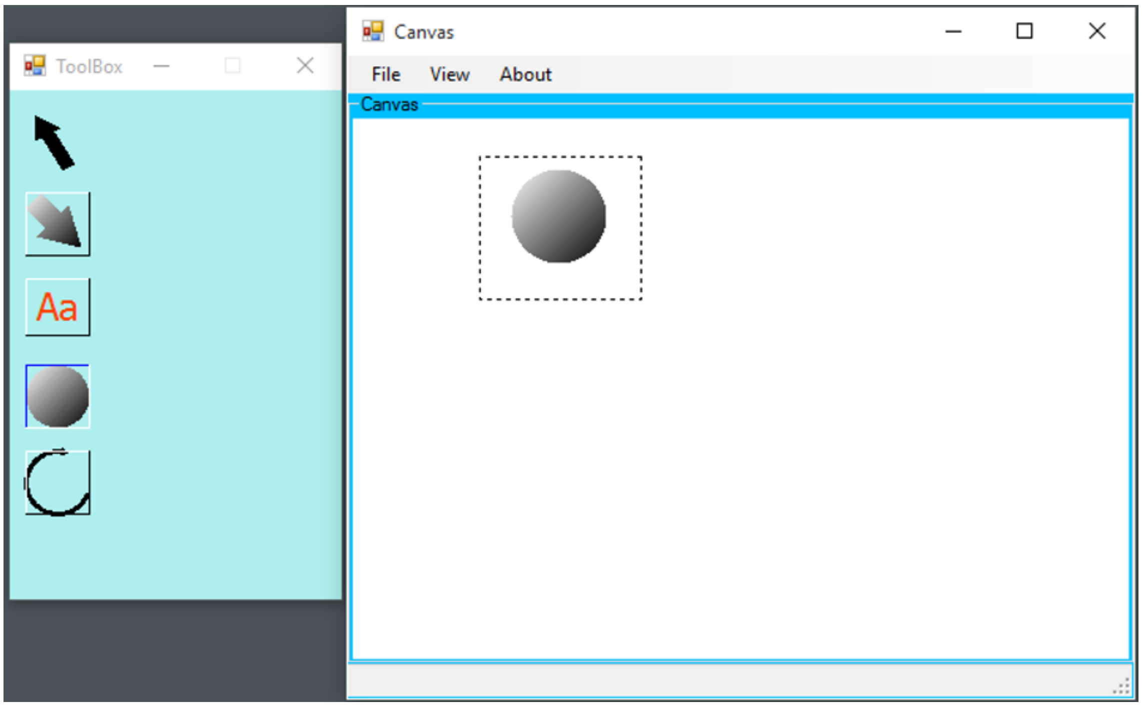
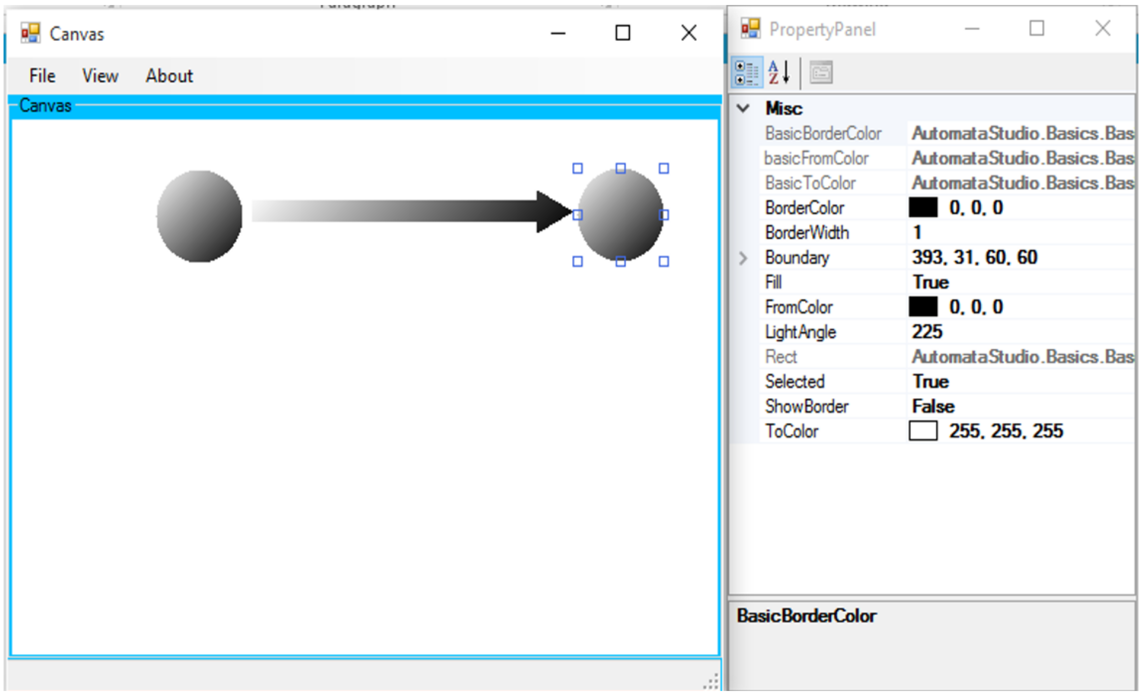


Figure 5.14: Select Tool and Draw on Cavas



Misc	
BasicBorderColor	AutomataStudio.Basics.Bas
basicFromColor	AutomataStudio.Basics.Bas
BasicToColor	AutomataStudio.Basics.Bas
BorderColor	■ 0, 0, 0
BorderWidth	1
Boundary	393, 31, 60, 60
Fill	True
FromColor	■ 0, 0, 0
LightAngle	225
Rect	AutomataStudio.Basics.Bas
Selected	True
ShowBorder	False
ToColor	□ 255, 255, 255

BasicBorderColor

Figure 5.15: Changing Properties of Selected Object

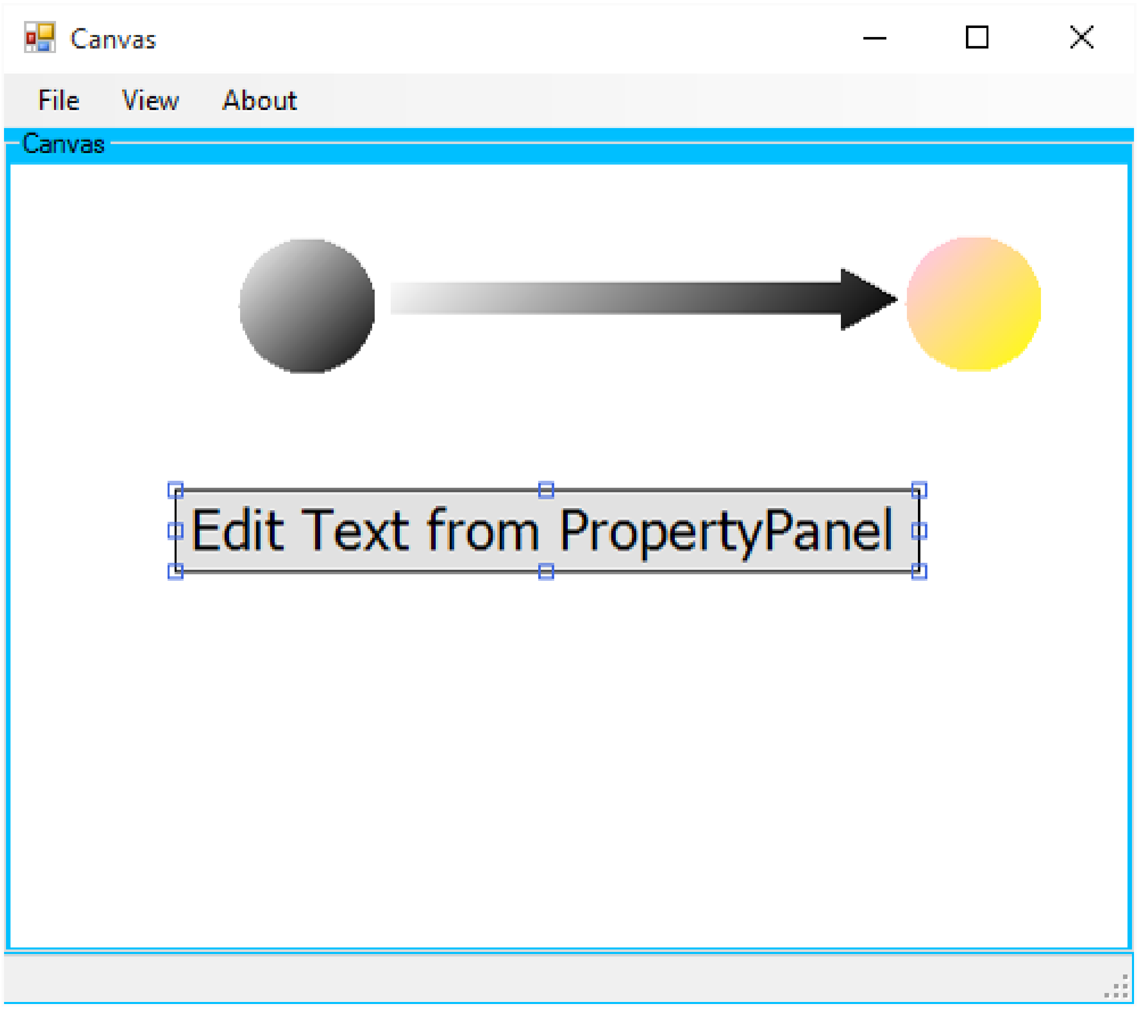
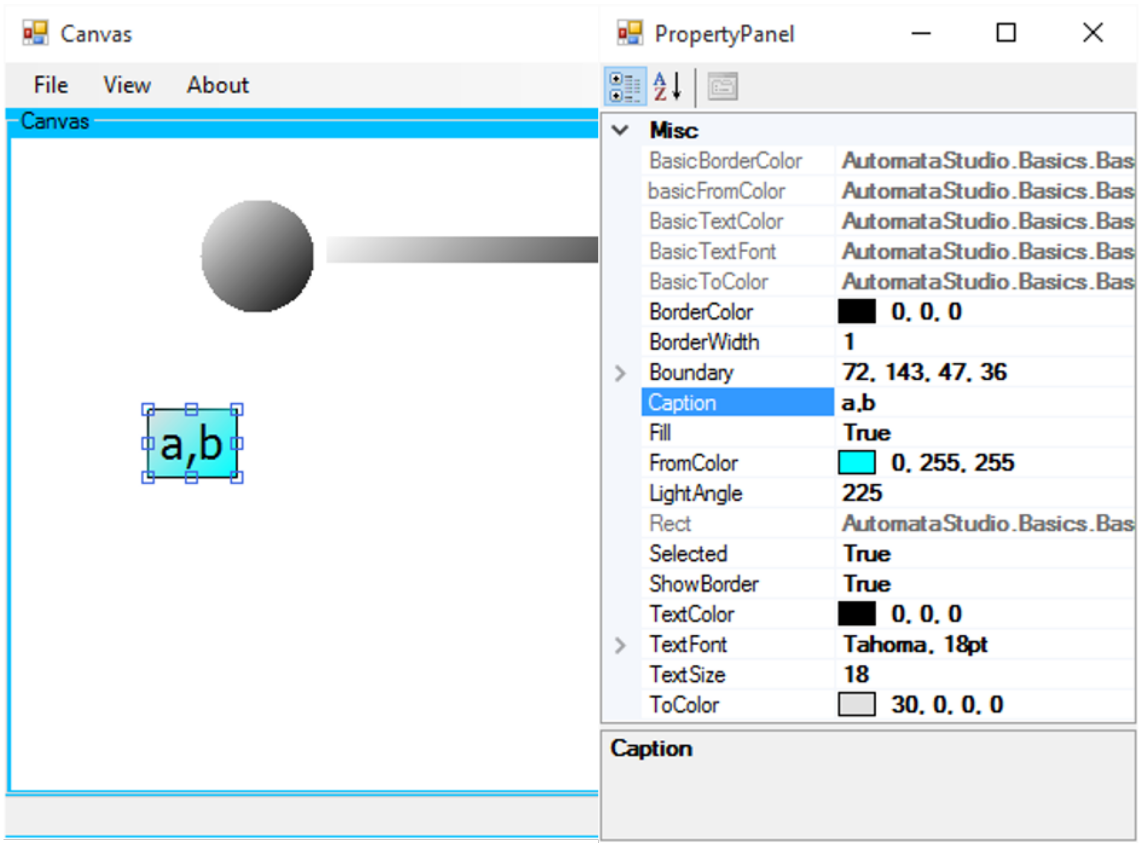


Figure 5.16: (a) Select ‘Text’ tool from the ToolBox and Edit its properties from ‘PropertyPanel’ Form (before)



BasicBorderColor	AutomataStudio.Basics.Bas
basicFromColor	AutomataStudio.Basics.Bas
BasicTextColor	AutomataStudio.Basics.Bas
BasicTextFont	AutomataStudio.Basics.Bas
BasicToColor	AutomataStudio.Basics.Bas
BorderColor	■ 0, 0, 0
BorderWidth	1
Boundary	72, 143, 47, 36
Caption	a,b
Fill	True
FromColor	■ 0, 255, 255
LightAngle	225
Rect	AutomataStudio.Basics.Bas
Selected	True
ShowBorder	True
TextColor	■ 0, 0, 0
TextFont	Tahoma, 18pt
TextSize	18
ToColor	□ 30, 0, 0, 0

Figure 5.17: (b) Select ‘Text’ tool from the ToolBox and Edit its properties from ‘PropertyPanel’ Form (after)

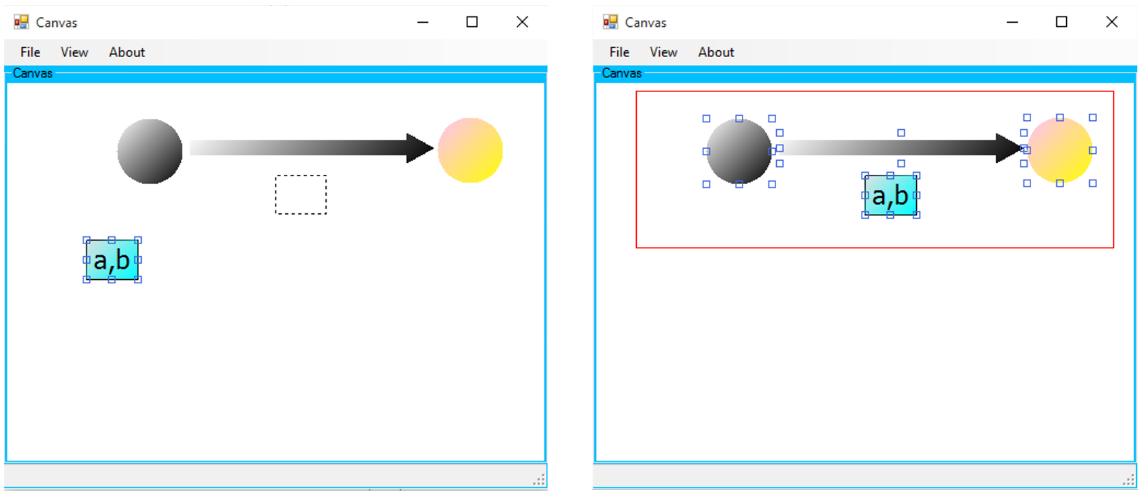


Figure 5.18: (LEFT) Select an object and drag it to the desired location, (RIGHT) Group selection of the objects.

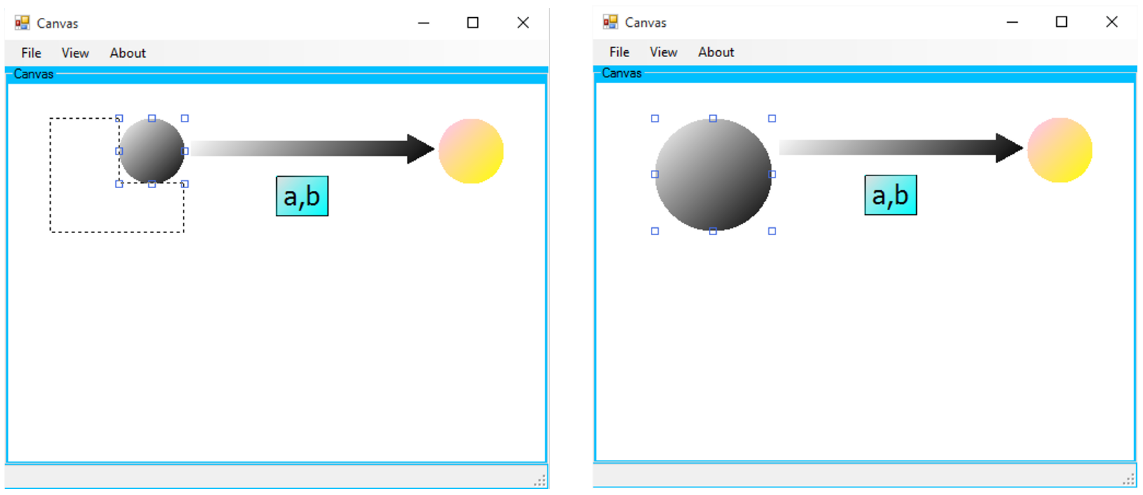


Figure 5.19: (LEFT) Resizing object, Select desired object and resize by the help of mouse (before), (RIGHT) Resizing object (after)

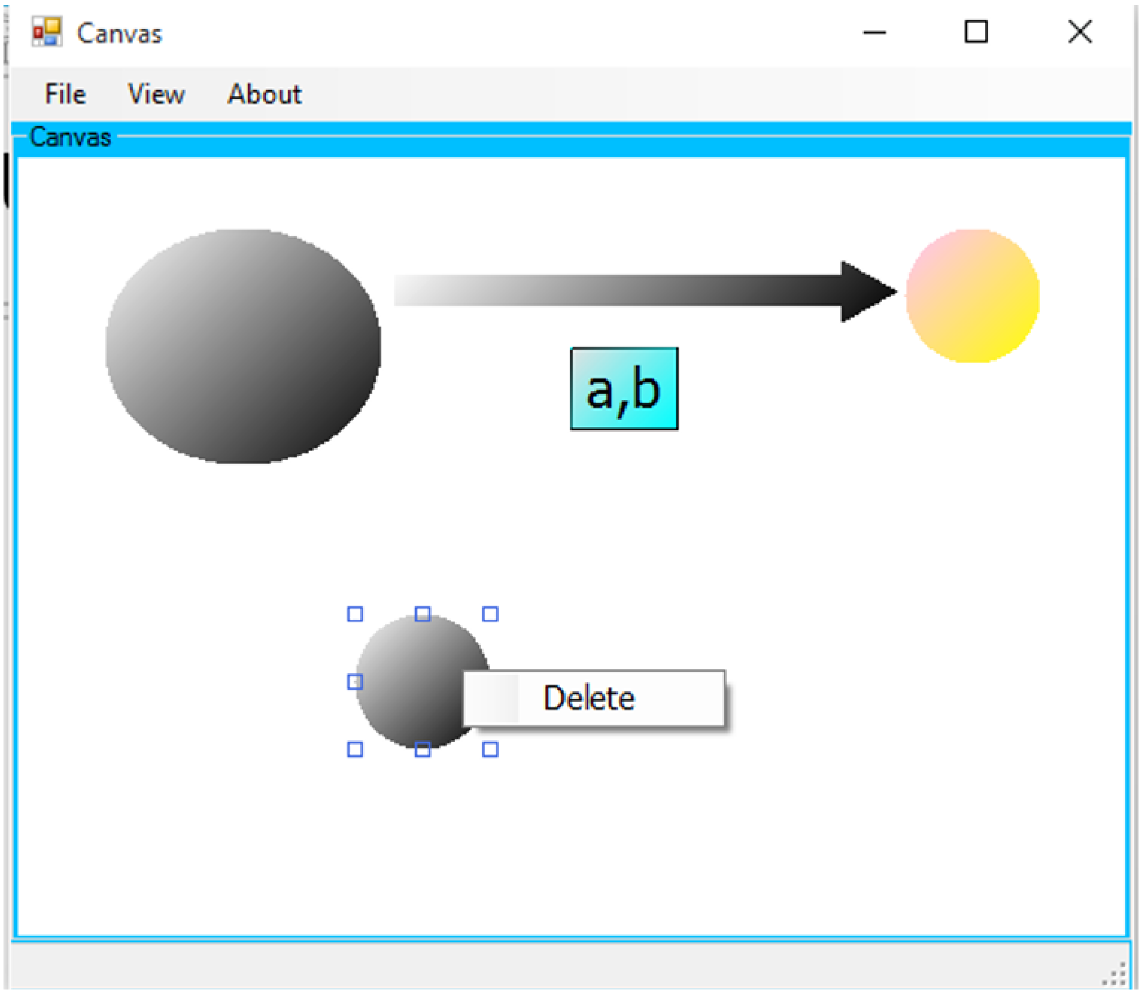


Figure 5.20: Objects can be deleted, On Right Button press of mouse.

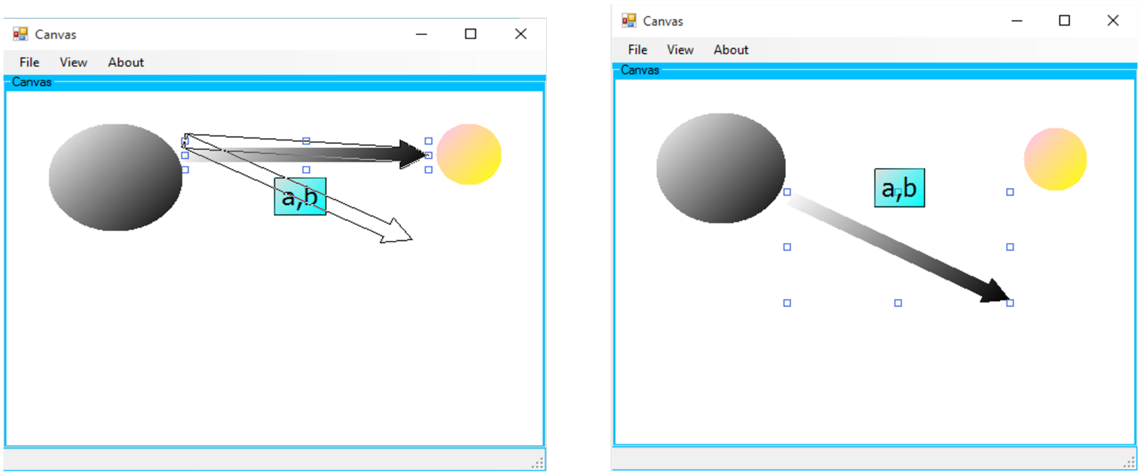
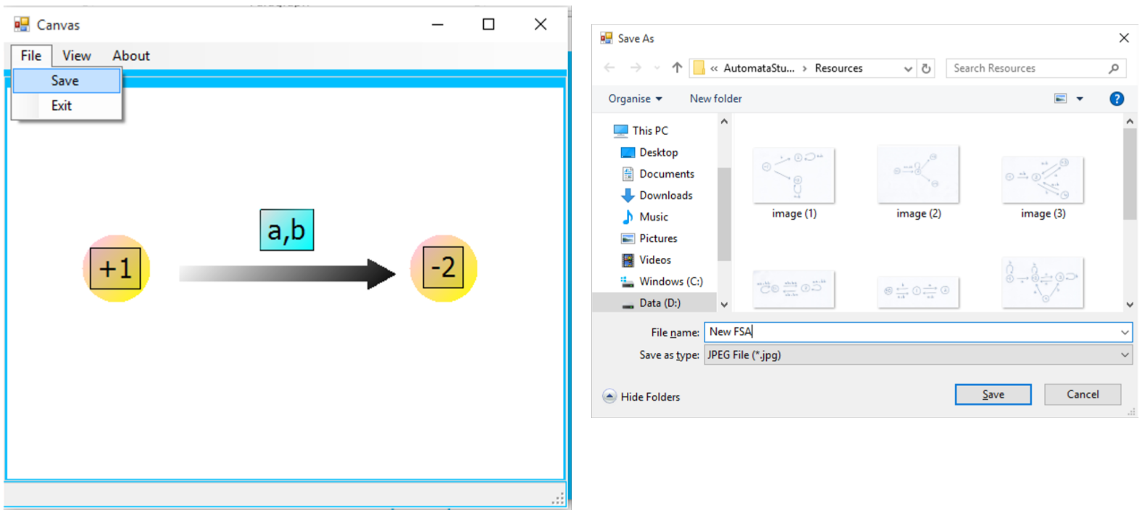


Figure 5.21: (LEFT) Arrow can be rotated (before), (RIGHT) Arrow can be rotated (after)



File name: New FSA
Save as type: JPEG File (*.jpg)

Figure 5.22: (LEFT) Save Canvas in form of Image, (RIGHT) Save Dialog Box

Chapter 6

System Testing and Evaluation

Testing is a process of executing a program with aim of discovering errors and trying to make sure that system fulfills the required needs as in accordance with software requirement documentation. Software requirement testing detects any if malfunctioning in the system occurs, and corrected before delivering the product. The primary goal is to guarantee that the developed system is up to required standards. There are many types of testing which are as following:

6.1 Graphical User Interface (GUI) Testing

User interface plays an important role in interaction of user with the system. If the interface is simple, easy and user friendly, user can easily learn the use of system. This system provides a clean simple interface with a canvas and toolboxes. User can create Finite state machine very easily with the help of drag and drop of objects. User can also load image into the system and graphical finite state machine is created by the system and displayed to the user on screen.

6.2 Usability Testing

The system is developed under standard and with easy to use features. System has been tested by the subjects and gave feedback on system efficiency, usability, and on different aspects of interface. The user did not report any major issue in the system usability and were able to create Finite State machines easily.

6.3 Software Performance Testing

Software performance testing is carried out to ensure that system developed is delivering the desired functionality, efficiently and reliably. Objects recognition is based on correct segmentation, detection and labeling of features and recognition. Classification rate of this system is displayed in the following table:

Class	Training	Testing
0	109	40
1	123	30
2	79	20
3	132	30
4	95	20
5	138	30
6	137	20
7	131	20
8	145	10
9	144	20
a	36	10
b	31	5
c	83	10
d	54	5
-	63	10
+	51	10
,	45	10
circle	104	20
loop	120	10
arrow	69	15

Table 6.1: Classification Rate of system

6.4 Compatibility Testing

Compatibility testing is the process to ensure that system is compatible with different versions of Windows operating system. Currently system has been developed on Visual Studio 2013 which is compatible with different version of windows.

6.5 Exception Handling

The system may generate an exception if the Accord.NET .dlls are not placed in the resource file of the project. The .dlls must therefore be added correctly in order to avoid this exception.

6.6 Load Testing

Load testing is done to check the system behavior at normal conditions as well as heavy load conditions. The system should operate in both conditions, however it might take few minutes if high resolution images is loaded for processing. This system does not requires high resolution images, features can be easily segmented and recognized in the normal resolution images.

6.7 Test Cases Testing

6.7.1 Test Case 1: Load Image

Test Case Id	TC-1
Unit of Test	Test to verify that image is loaded successfully.
Steps to be executed	1. Click on Load Image from File Menu. 2. Open File Dialog open, select image.
Expected Result	Image is loaded successfully.
Actual Result	Image is loaded successfully.
Status	Pass.

Table 6.2: Test Case: Start Application

6.7.2 Test Case 2: Pre-processing Image

Test Case Id	TC-2
Unit of Test	Test to verify that image is successfully pre-processed.
Steps to be executed	Press Create FSA button.
Expected Result	Image is pre-processed successfully.
Actual Result	Image is pre-processed successfully.
Status	Pass.

Table 6.3: Test Case: Pre-processing Image

6.7.3 Test Case 3: Objects Loading on Canvas

Test Case Id	TC-3
Unit of Test	Test to verify that objects are successfully loaded on screen after pre-processing.
Steps to be executed	Check objects drawn on canvas.
Expected Result	Objects should be drawn to the image.
Actual Result	Objects are drawn on screen.
Status	Pass.

Table 6.4: Test Case: Objects Loading on Canvas

6.7.4 Test Case 4: Saving Image

Test Case Id	TC-4
Unit of Test	Test to verify that image is saved successfully in desired folder.
Steps to be executed	Press ’Save’ button, open file dialog appears.
Expected Result	Image should be saved successfully in desired folder
Actual Result	Image is saved successfully.
Status	Pass.

Table 6.5: Test Case: Objects Loading on Canvas

Chapter 7

Conclusions

7.1 Conclusion

Automata Studio is a system developed for the ease of students and teachers. Although there are many online editors present out there which can draw finite state machines, but this system uses image processing techniques and can load hand drawn sketch image into system and can draw state machine according to the image. The finite state machines drawn on a paper are binarized, and objects are segmented from the image. The extracted objects are then recognized using Multi-class Support Vector Machine classifier. Once recognized, the objects are placed on a new canvas using graphics libraries. The resultant drawn image can be edited and saved on the run time. User can create Finite state machine from scratch also.

7.2 Perspective

The present version of the system relies on limited hand drawn objects. These can be further enhanced by using more data sets, and classified using multi-layer artificial neural network. The image processing techniques used for the pre-processing can be improved by using other advanced algorithms. This system can be further implemented for generating regular expressions on basis of drawn finite state machine. It is expected that this study will contribute towards the development of a complete tool used for finite state machine, regular expression, transition graphs.

Chapter 8

Sample Images

Following are some Sample images used in the system.

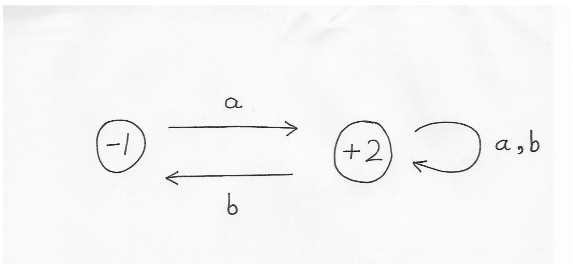


Figure 8.1: Sample Image (1)

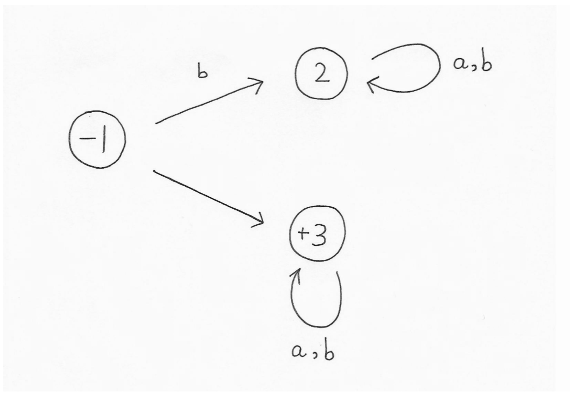


Figure 8.2: Sample Image (2)

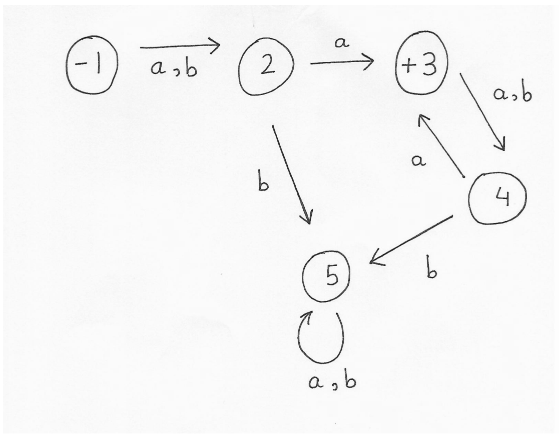


Figure 8.3: Sample Image (3)

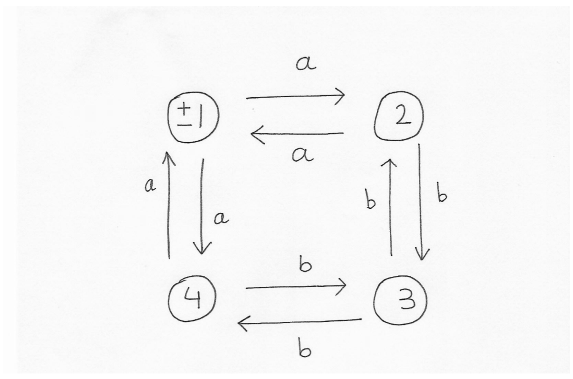


Figure 8.4: Sample Image (4)

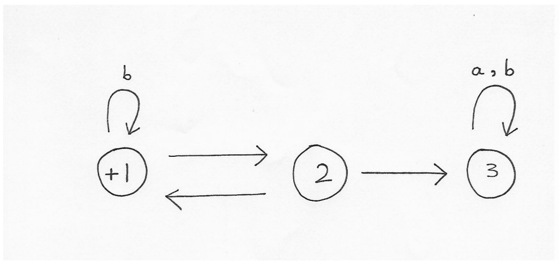


Figure 8.5: Sample Image (5)

References

[1] T. J. Atherton. Kerbyson, D. J. Circle detection using hough transform filters. *Image Processing and its Applications 1995., Fifth International Conference on.*, 1995. Cited on p. 3.

[2] Ruben. Gonzalez. Fast line and circle detection using inverted gradient hash maps. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015. Cited on p. 3.

[3] et al. Wang, Yi. Line detection algorithm based on adaptive gradient threshold and weighted mean shift. *Multimedia Tools and Applications*, 2016. Cited on p. 4.

[4] Qixiang Ye and David Doermann. Text detection and recognition in imagery: A survey. *IEEE transactions on pattern analysis and machine intelligence.*, 37:1480–1500, 2015. Cited on p. 4.

[5] Dinesh Bhardwaj and Vinod Pankajakshan. Image overlay text detection based on jpeg truncation error analysis. *IEEE Signal Process. Lett.*, 23:1027–1031, 2016. Cited on p. 4.

[6] et al. Yin, Xu-Cheng. Multi-orientation scene text detection with adaptive clustering. *IEEE transactions on pattern analysis and machine intelligence.*, 37:1930–1937, 2015. Cited on p. 4.