CHAPTER THREE

METHODOLOGY

3.1 Overview:

As mentioned in chapter 2, we will use the OpenALPR engine and modify it so that it will be able to recognize the Arabic numbers used in the Sudan plate. In order to make the Sudan plates recognizable we will go throw a few steps and use some utilities to train the OCR.

3.2 OCR training

Since the classifier is able to recognize English characters easily, it should be trained to recognize Arabic Numbers as well. We trained the classifier on 20 samples of 38 characters using 2 fonts in 10 pt size and 4 styles (normal, bold, italic, bold italic), making a total of 6080 training samples. Here are the steps we gone throw in order to train the OCR.

3.2.1 Creating a Microsoft word Document

We created a Microsoft office word doc and typed the Arabic numbers using “Arabic Typesetting” font and 1.5-line spacing, 2 pt character spacing and font size 10 pt. In this doc all the numbers were repeated 20 times; in order to be well recognized in Tesseract [22].

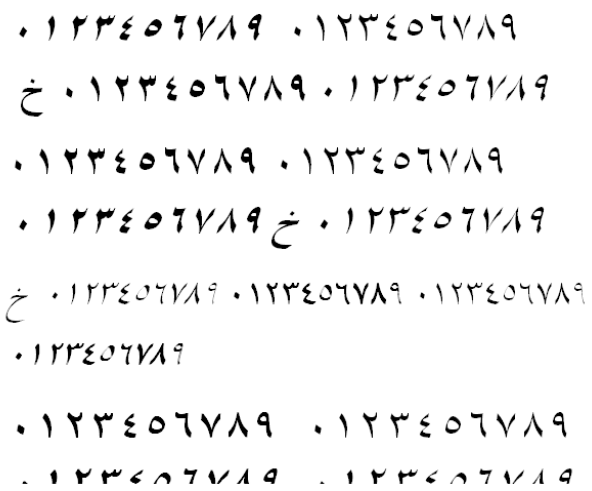

Figure 3-1 Arabic number character tiles

3.2.2 Converting the word document to pdf

We Saved the word document as:

lsd.ArabicTypeSetting.exp0.pdf (Format:<lang_code>.<font_name>.exp0.pdf 4)

3.2.3 Converting the PDF file to .tif file

We used the GhostScript utility to convert the pdf file to a tiff image file with the command:

```
gswin64c.exe -sOutputFile= lsd.ArabicTypeSetting.exp0..tif -
sDEVICE=tiffg4 -r300x300 -g2550x3300
lsd.ArabicTypeSetting.exp0.pdf
```

The output was a file named:

lsd.ArabicTypeSetting.tif

3.2.4 Making the box file

Using tesseract.exe with the following command to get the box file:

```
tesseract.exe lsd.ArabicTypeSetting.exp0.tif
lsd.ArabicTypeSetting.exp0 batch.nochopmakebox
```

The output was a file named:lsd.ArabicTypeSetting.box

3.2.5 Correcting the misrecognized characters

We usedjTessBoxEditor utility with the tif and the box file for editing. Gone through each number character to verify if they match the text-coded character, if not then the character must be corrected to match the text-coded character. More corrections increase the OCR accuracy.

Then we collected character tiles for each character and each font. The character tiles are slightly different in shape, this is necessary for the OCR training to understand how to detect characters.

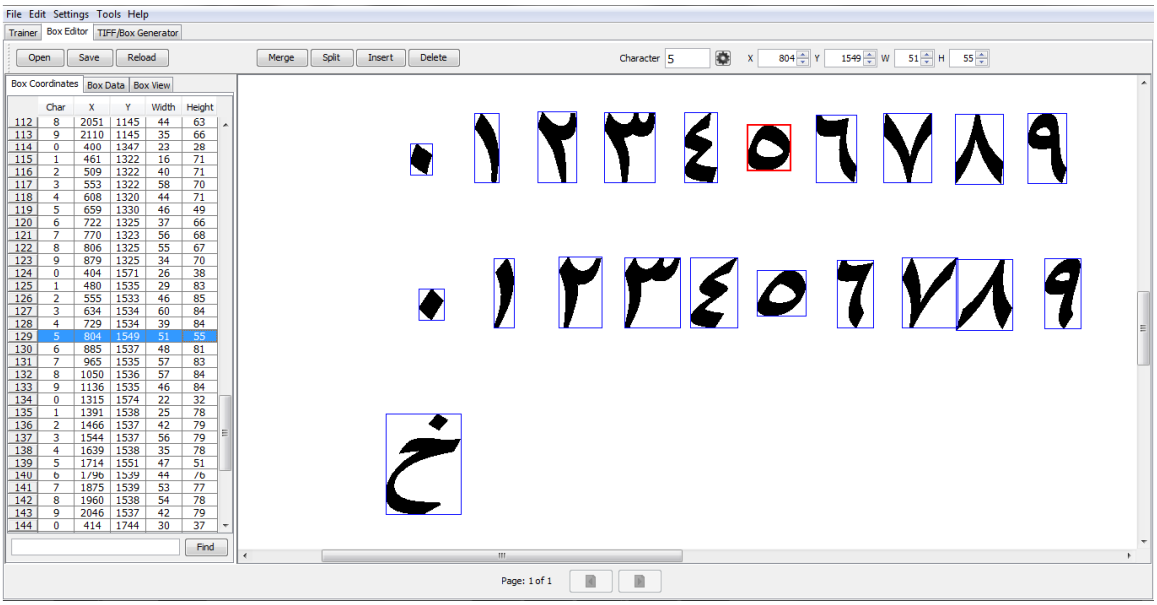

Figure 3-2 jTessBoxEditor interface

Once we assigned all the characters to its corresponding machine-coded character, we scanned through the list of characters to make sure that the assignments match the images. Once we've done this, it was the time to create the training sheet (the .traineddate file).

3.2.6 Combining the TessData

In order to get the final .traineddata file we used OpenSerakTrainer to combine the TessData which are the tif and box file.

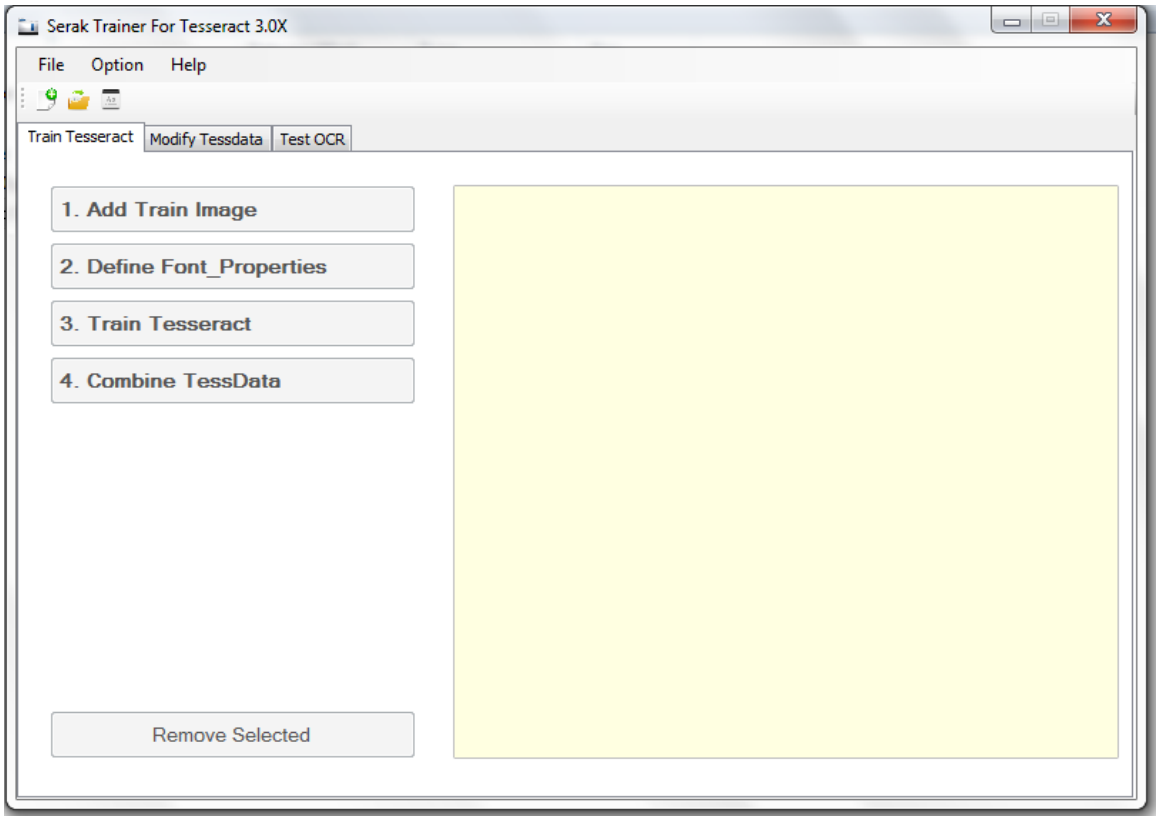

Figure 3.3 SerakTrainorTesseract interface

Going throw the steps of this program subsequently as shown in figure 3.3 will result the trained OCR Arabic numbers file.

Finally we have a new file named sd.traineddata. We copied this file into theruntime_directory (runtime_data/ocr/tessdata/) and it is now ready for the OpenALPR to use.

3.3Modifying the OpenALPR configuration files

3.3.1 Modifying OpenALPR.conf

In the previous section we trained the OCR for the Sudan country plates, Now we have to configure the dimensions of the plate and characters in the configuration file (openalpr.conf) which is located at the main directory of the OpenALPR.

We edited the following values:
[width of full plate in mm]
[height of full plate in mm]
[width of a single character in mm]
[height of a single character in mm]
[whitespace between the character and the top of the plate in mm]
[whitespace between the character and the bottom of the plate in mm]
[Minimum size of a plate region to consider it valid.]
[Minimum size of a plate region to consider it valid.]
[name of the OCR language (lsd) -- typically just the letter l followed by your country code]

3.3.2 Matching the Pattern

As mentioned in chapter 2 (2.5.8) the OpenALPR is doing a regular expression matching (Pattern matching) to the plate, then chooses the highest one matching

the pattern form the top N results (regardless of its confidence) and marks it as True.

We created the pattern/sd.conf file to match the Sudan license plate format, which can be as follows: The license plate containing 5 numbers, The license plate containing 4 numbers, The license plate containing 3 numbers.

The equivalent of the previous 3 lines using the OpenALPR regular expression:

```
sd #####
sd ####
sd ###
```

Since the # symbol represents a number, and the @ symbol represents a letter.

3.4 SQL Database

Using Microsoft Visual Studio 2015 we created a database to link the extracted license plate with the owner information.

First we created a new SQL project, added a new table to the Database named "Table" which was a 6 column Table, filled the table with experimental values under the names of (Owner Photo, Name, Gender, Owner Address, Phone Number, License Number).

Finally, we linked the Database to the OpenALPR project using Visual Studio.

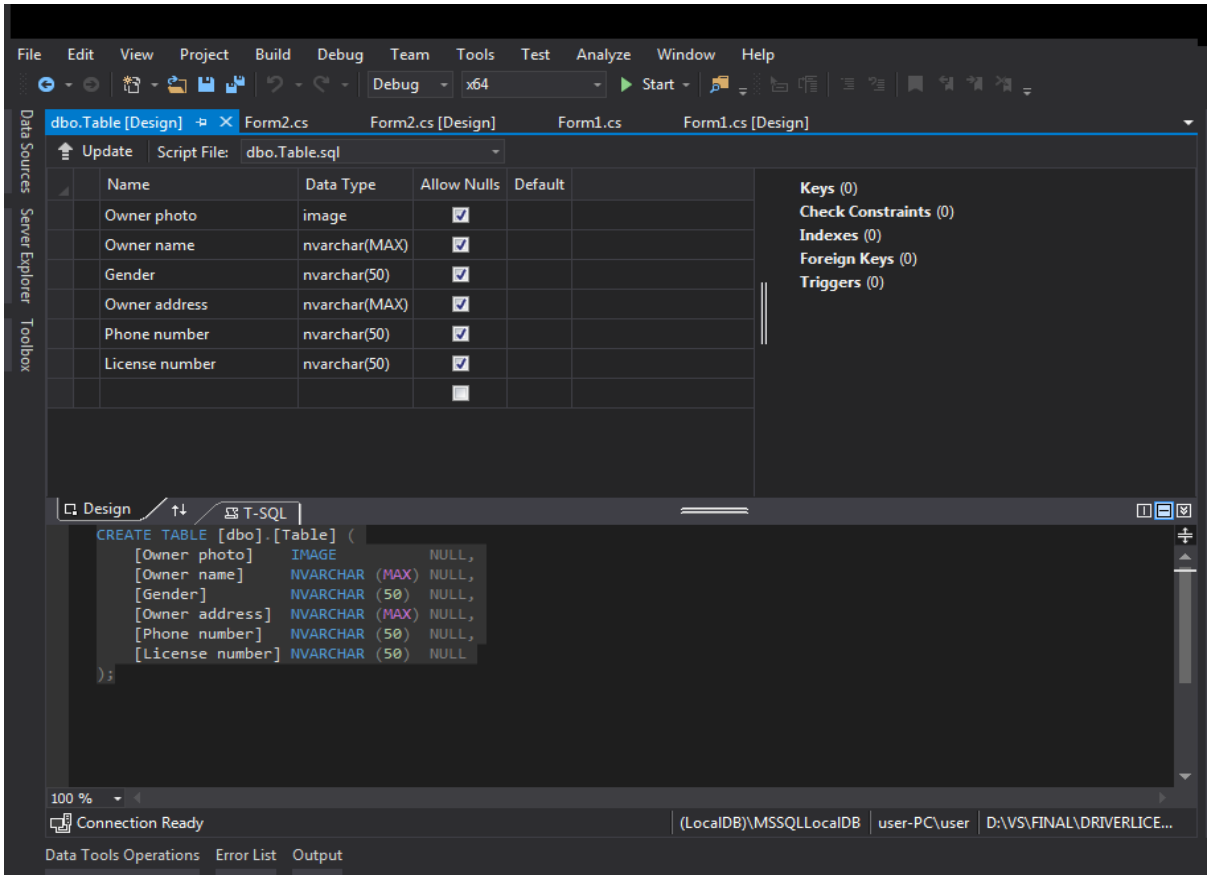

Figure 3-4 SQL Database

3.5 GUI

Visual studio 2015 has a useful utility called “Windows forms application” which we used to create a Graphical user interface.

From the “New” menu we clicked on “New Project” then “Windows forms application”, and began dragging and dropping the basic GUI elements (Button, input fields, etc..) and linking the elements to the code using C# programming language which was easier to deal with it than using regular C++.

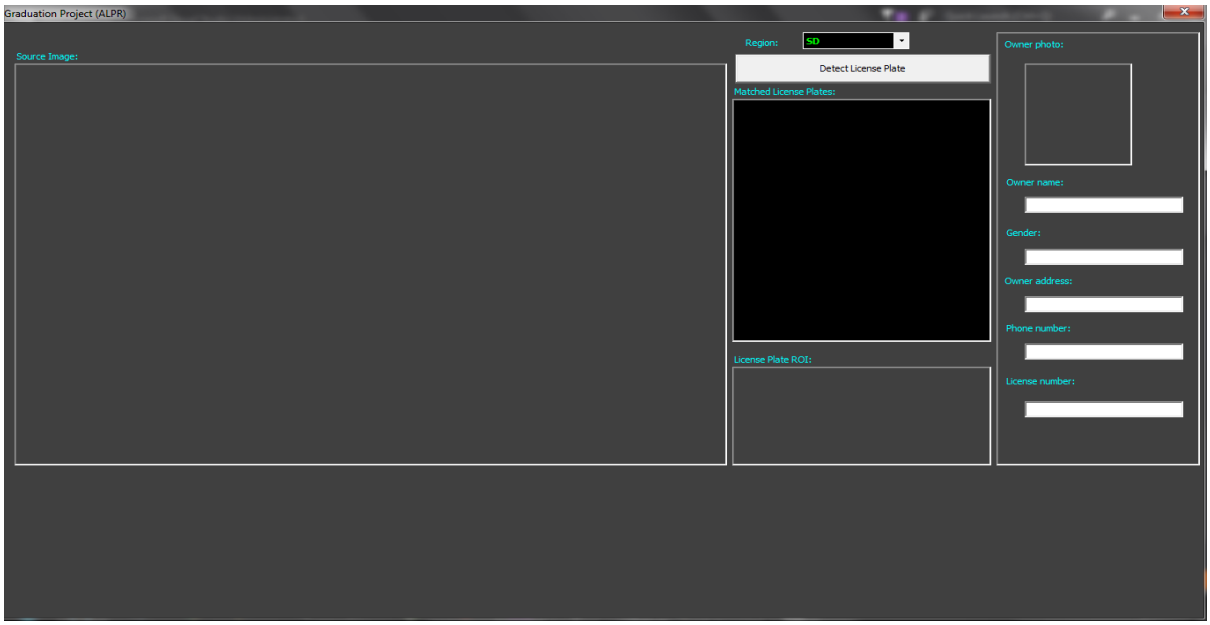

Figure 3-5 The final interface

And now the program is ready to run.