Tolls VES Auto Region of Interest Selection: Developed by Jamal D. Scott

This write up will demonstrate how the Tolls VES auto ROI selection process works for C# Windows OS. For this process, you will need several tools that you can find online for free.

* Microsoft Visual Studio 2013 or higher.
* Emgu\_CV 3.2.0
* Notepad++
* A Haar Cascade Classifier

For a copy of Microsoft’s Visual Studio 2013 (or higher) you can find it on Microsoft’s product download page at: <https://www.visualstudio.com/>

Emgu\_CV is a C# wrapper class for OpenCV libraries which were originally written in C++; Emgu\_CV allows OpenCV to operate within a C# environment. You can install the latest version through Visual Studio’s NuGet package installer. We will later discuss how to accomplish this, but for now you can review Emgu\_CV at: <http://www.emgu.com/wiki/index.php/Main_Page>

Notepad++ will be used to edit the properties of a program, you may already have it installed on your system, if you do not it can download it at this location: <https://notepad-plus-plus.org/>

The final piece is the Haar Cascade Classifier. The primary function of the cascade classifier is to note specific features of a specified object and save the average of those properties into an XML file. This file will then be referenced when called by the Cascade Classifier class. These properties are noted by comparing the features of many “positive” images to those of many “negative” images. Positive images are images solely of the object that you wish to detect where negative images are images that do not contain the object whatsoever or items similar to the object.

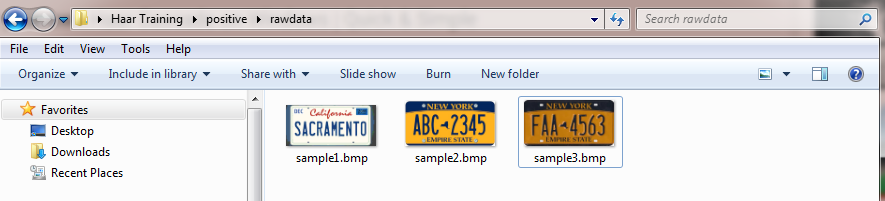
A Cascade Classifier can be trained in various ways, for the Tolls VES application, a prebuilt trainer was used; these files are packaged with this documentation. This application is a 3rd party application and was not developed with Tolls VES.

Training a classifier:

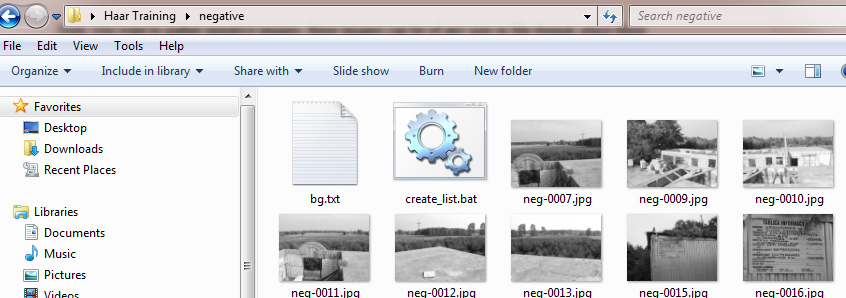
First, we need to generate the Haar Cascade Classifier XML file. This is achieved by running a training program on positive and negative images. For positive images, you can either capture your own, or do a simple Google search to find the desired images. This part is the most crucial as the types of images chosen to train the Cascade Classifier will greatly determine the success or failure of your application’s abilities to detect objects when using the XML file. For example, not all of the images below will be detected as a license plate. 

The quantity of the images is very important as well, the cascade classifier simply will not work on a small image count. With some luck, others have been able to achieve minor and perhaps inconsistent object detection with around 40 positive images and about 200 negative images. More accurate classifiers have used anywhere from 1000 to 5000 images. Ideally you will want more negative images than positive. Defining an exact amount for a success threshold is unclear but rather it is worth while noting that the more images tested against, the more accurate the classifier will be. 100% object detection is not guaranteed.

Also, if possible, it might be best to shrink the images down to a size of no more than 500KB and convert them to a bitmap file as large images may cause the application to slow down and the application was originally designed to work with the bitmap file type. Place these images into the Haar Training/positive/rawdata folder.

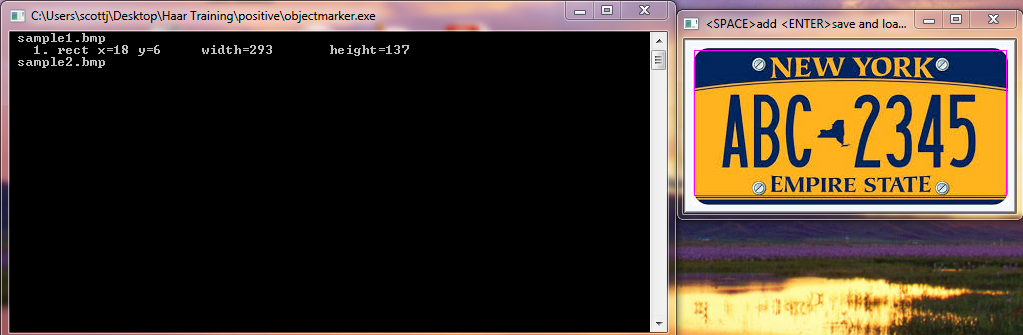


Next, you want to gather negative images, these images can be of any size or file format, place these images into the Haar Training/negative folder:



After doing so, click on the create\_list.bat file to generate the bg.txt file, this will list all of the negative images that the classifier needs to refer to.

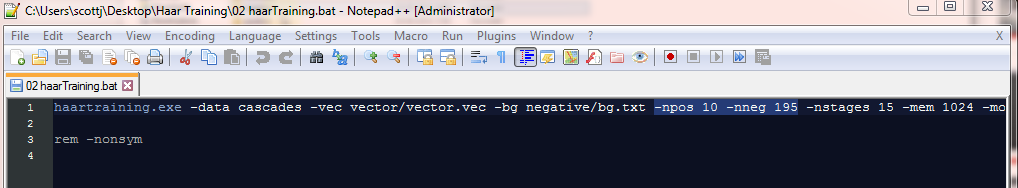
Next we can begin marking our objects. Navigate to Training/positive and run objectmarker.exe. This is the pre-built tool that will allow you to mark the area of the positive regions for detection. If the image files are not in bitmap format, the application will most likely close immediately upon startup. When the application starts, the images located in the positive training data folder will then appear in a window along with a second console application. Use the mouse to select the region of interest around the object that you wish to detect. You can always reselect and ROI (region of interest.) The space bar must be hit to save the region’s location and you can press the enter key to save that data and move on to the next image. The program will close upon completion of all images.



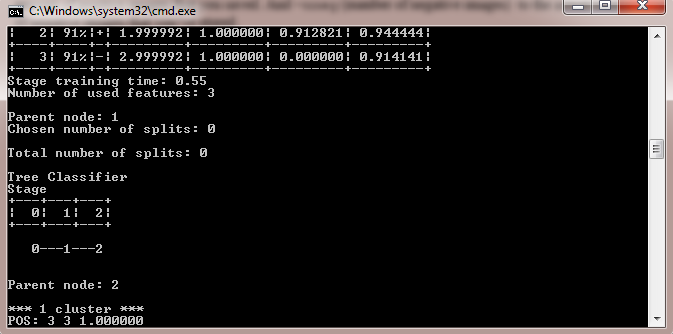
If the image you chose has multiple objects in it that you wish to detect, you can select each of those objects then hit the space bar to save each individual ROI’s location.

After marking all of the objects, we need to create vector file regarding the positive images. To do this, navigate back to the Haar Training root folder, and run samples\_creation.bat. This will generate our vector file which is located in the same directory under the “vector” folder.

Next, navigate to the “Cascades” folder and delete any pre-existing data, this is important to avoid potential writing errors when the program wants to write out the cascade data. After all the old cascade data has been removed, navigate back to the root directory and open up haarTraining.bat in Notepad++ . This next step is very important as well as it tells your program what it’s looking at and what it should be outputting. Change the values of –npos (number of positive images) to the actual amount of positive images that you saved. And –nneg (number of negative images) to the actual number of negative images that you’ve stored.

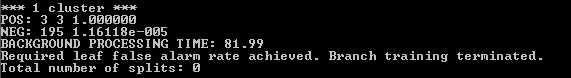


Finally, we can begin training our Haar Cascade Classifier. Navigate back to the root directory and run haarTraining.bat. This will begin processing all of the images and generating the cascade classifier xml. **Note**: training can be a very slow process, with a decent amount of images, training may take several hours or even days. The exact time is difficult to put an estimation on. Again, a successful completion of training may not be guaranteed, errors may always occur.

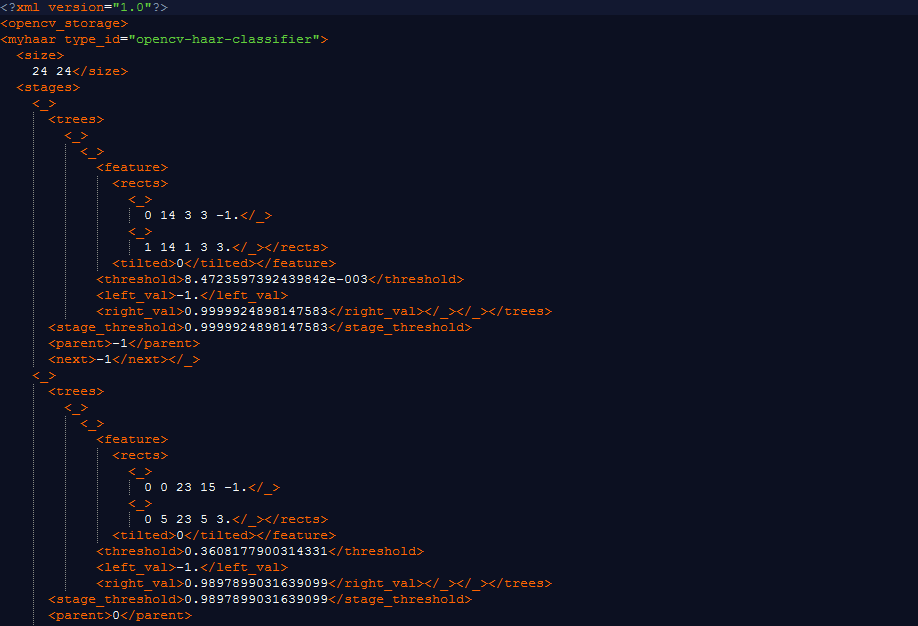


The following line of text will notify you when the training has been completed:

“Branch training terminated.”

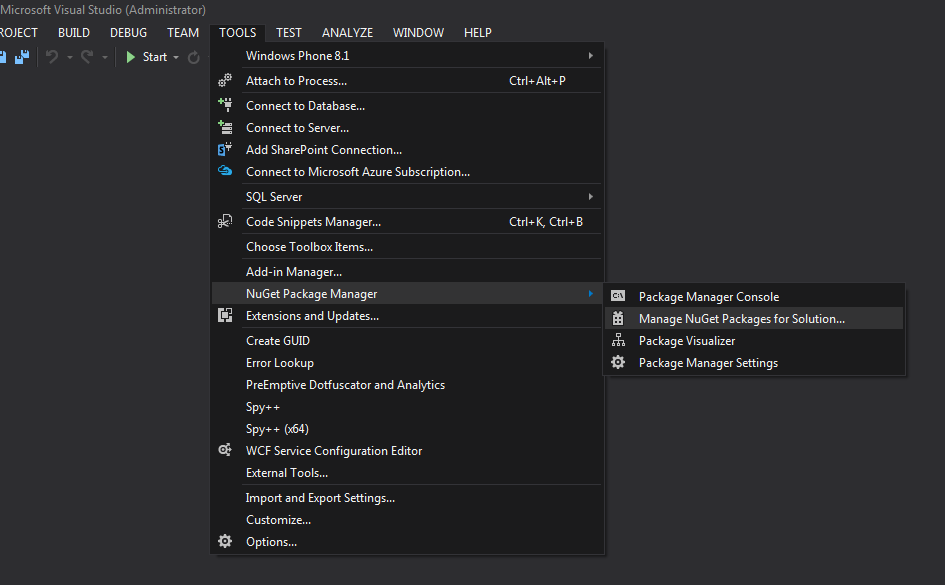


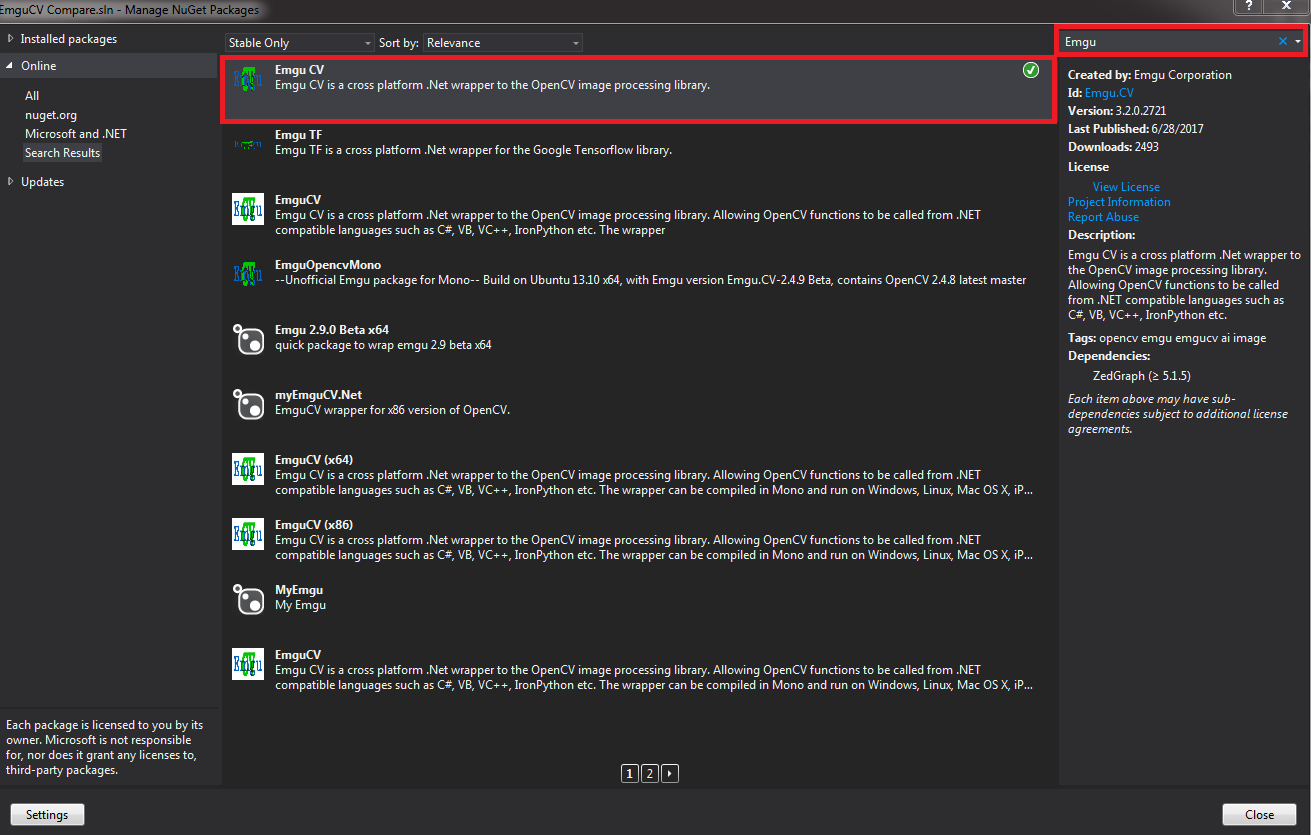
Finally, we can then run convert.bat, which is located in the root directory. This will convert all training data into a usable XML file that we can reference in our program.



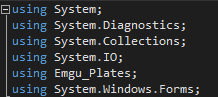
Installing Emgu\_CV and using the XML:

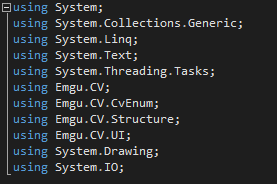
Now that we’ve completed our XML generation, we’re ready to test it in a program. Open up Visual Studio 2013, create a new project, and go to Tools -> NuGet Package Manager -> Manage NuGet Packages for Solution



This will bring up the NuGet manager screen. In the upper right hand corner, search for “Emgu” and install the first option. 

This will install all of the tools and libraries that you need to run Emgu\_CV for C#. You may also want to use the following libraries in your program:



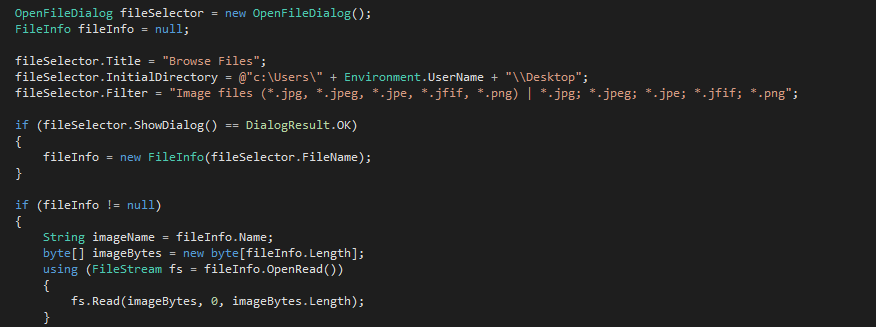


Finally, we can begin the code. There are several major parts to the code to get object detection. We only focus on these:

* Reading in an image.
* Storing the image in a Mat() object
* Referencing the cascade XML
* Converting the image to grayscale
* Detecting the object(s)
* Drawing a rectangle around the detected object
* Outputting the image

First, we want to allow the user to select any image they want manually, below is the following code that will open up a window browser so that the user can browse their file system to select an image.

OpenFileDialog allows the navigation and retrieval of the images on the system, and FileStream will read in the image to the program.



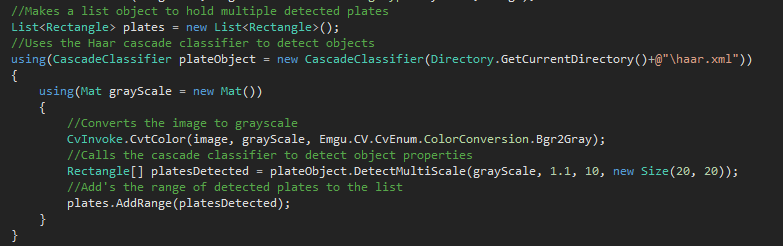
Be sure to use the following attribute to whatever method you are using the OpenFileDialog in or else an exception will occur and your program will not run: (in this case, we are calling OpenFileDialog from the main method.)



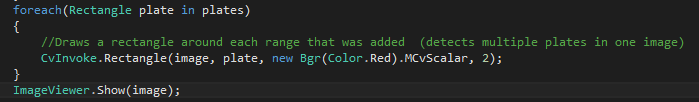
Now that we read in our image and stored it into a byte array via the FileStream reader, we can then covert the image to a Mat() object and detect it. Create a new, empty Mat() object and use the following line of code to read the image from a bye array into a mat object:



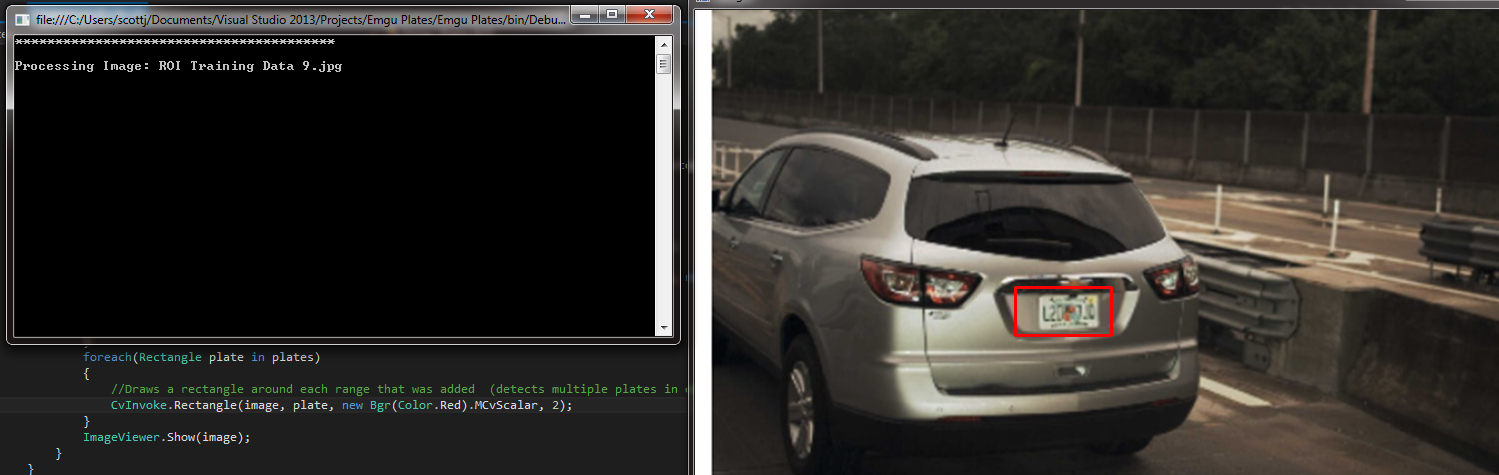
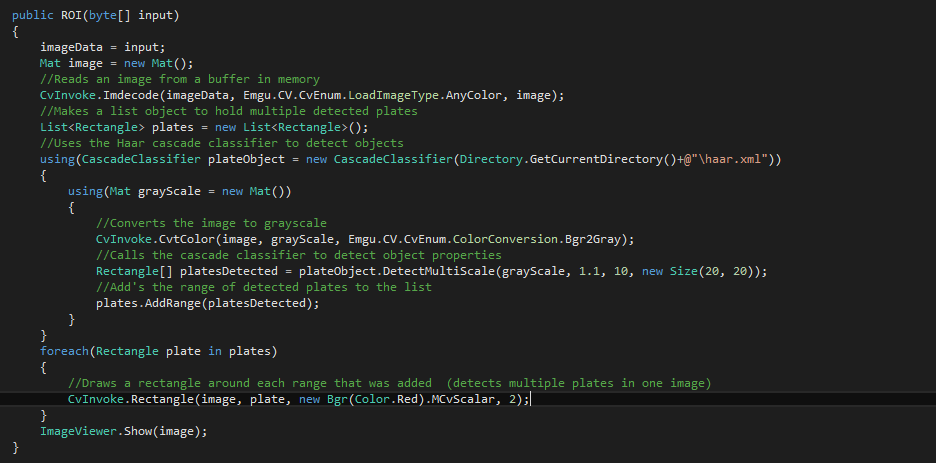
imageData is a Byte[] array that we used to initially read in our image. Next we will need to make a list data structure of type Rectangle to hold a range of different rectangles. This will allow us to detect multiple objects in one image. After creating our data structure, we can then call the cascade classifier class and load our XML file. Note that you would want to place the XML file in the same directory as your program. If not, you must specify the exact file path to the file. Directory.GetCurrentDirectory gets the current directory of the executing program.

 Next we will use a new Mat() object to hold the grayscale of our image, detect the object using the CascadeClassifier class and add the ROI to the list data structure:

Finally, now that we have our detected regions, we can draw the rectangle around each detected object and output the image:



Collectively, the algorithm follows:

Then you can run the program. If no object is detected, there are several scenarios, how you process your image in the code may be incorrect, make sure the grayscale Mat() is being used. If not, ensure that the image you wish to detect an object in is not difficult for the program to recognize the object. Finally, the cascade trainer’s XML file simply may not have successfully documented properties well. If so, you may need to repeat the training process.

These are the barebones of simple object detection and may not always be the most efficient. Tweaks or complete revamps of code may need to be made depending on what you wish to accomplish.

For further research about motion and object detection, OpenCV or different utilities, refer to OpenCV’s website: <http://opencv.org/>