# Sign Finder – High Level Description of Detection API

The core of Sign Finder is C++ computer vision detection code that is called either by the Android app or on the Windows command line. For debugging and performance assessment purposes, we provide a driver that runs the detection algorithm on a desktop machine. The source code for the Sign Finder algorithms is on Bitbucket ([link](https://bitbucket.org/skerisignfinder/wicab_signfinder/)).

## YAML Configuration File

The detector parameters are set up using the provided YAML files (one for each sign at this stage), which currently include exit\_sign\_config.yaml and restroom\_sign\_config.yaml.

For more details about the content of the configuration file, please refer to the Training document.

## Classifier Model Files

The detection algorithm uses two stages of classification. The first stage uses an AdaBoost cascade classifier trained on LBP features. The output of the first stage (i.e. candidate image patches) is fed to the second stage, which is an SVM classifier that assigns a confidence to each candidate patch from 0 to 1 (0 means very unlikely to be the target sign, 1 means very likely to be the target sign). In the specific case of the restroom signs, patches that pass the SVM tests are then classified by a second SVM classifier to distinguish between men and women restroom signs.

A model file is associated with each classifier and sign. For instance, in the case of the exit signs the AdaBoost classifier is defined by the file exit\_sign\_cascade.xml while the SVM is defined by the file exit\_sign\_model.svm . The XML files that describe the AdaBoost cascade are in a standard format used by the OpenCV library. The SVM model files are plain text files generated by libSVM during the training.

If you’d like to generate your own classifier model files, please refer to the Training document.

## Invoking the Sign Detection Code

The detection algorithm and the driver are written in C++ 11 and it relies on OpenCV 2.4.11 for image data structures and image processing and on libSVM 3.20 for the SVM classifier.

The file [Wicab\_SignFinder/src/main.cpp](https://bitbucket.org/skerisignfinder/wicab_signfinder/src/adb439ad1767?at=master) contains the driver and it shows how to invoke the detection algorithm and how to handle its output.

On line 175,

ObjDetector **detector**(options.configFile);

the object detector is initialized passing as input the name of the yaml configuration file to use. This will load the classifiers and initialize the detector itself.

On line 251,

**auto** result **=** detector.detect(frame, fps, options.doTrack);

the detection function takes as input an image in the OpenCV format (cv::Mat), the variable fps that will contain the frame rate on exit and a boolean flag that specifies whether to use the tracking (if true) algorithm or not.

The output variable result is a list of detections represented using the following structure:

**struct** DetectionInfo

{

cv**::**Rect roi; *///< detection location*

**double** confidence; *///< detection confidence as estimated by the SVM*

**int** iLabel; *///< label (-1, 1 if 3 stages, 0 otherwise) associated to the ROI*

std**::**string sLabel; *///< string associated to the label specified by iLabel*

};

where roi represents a bounding box (cv::Rect) specified by the top left corner and its height and width, confidence is the confidence value associated to the detection, iLabel represents the classification label (-1 or 1 whether the classified patch is classified as belonging to the set of negative or positive images, respectively), while sLabel will contain a string to identify the type of element detected (specified in the configuration file). Both iLabel and sLabel are used when two SVM classifiers are used, as in the case of the restroom sign.

## Running the SignFinder console application

SignFinder (the driver) is a console application and it takes the following parameters as input:

USAGE**:** SignFinder **-**c configfile **-**i input **[-**p prefix**]** **[-**m maxdim**]** **[-**s**]** **[-**d**]** **[-**f**]** **[-**t**]** **[-**n**] [**-rroisFilename**]** **[**-l label] **[-**o output**]**

**-**i**,** **--**input input**.** Either a file name**,** or a digit indicating webcam id (0

for integrated webcam, 1 for a USB connected camera)

**-**c**,** **--**configFile location of config file

**-**d**,** **--**debug**=[false]** whether to show intermediate detection stage results

**-**f**,** **--**flip**=[false]** whether to flip the input image

**-**h**,** **--**help**=[true]** print **this** message

**-**m**,** **--**maxdim**=[**640**]** maximum dimension of the image to use **while** processing

**-**n**,** **--**notrack**=[false]** whether to turn off tracking

**-**o**,** **--**output **if** a name **is** specified**,** the detection results are saved to a

video file given here

**-**p**,** **--**patchPrefix prefix **for** dumping detected patches to disk **if** one **is** provided

**-**s**,** **--**saveFrames**=[false]** whether to save frames to disk

**-**t**,** **--**transpose**=[false]** whether to transpose the input image

**-**v**,** **--**version**=[false]** version info

-r, --roisFile save detected ROIs to the text file specified here

-l, --label specifies the label to associate to the ROIs when writing to

file

While the software is running, as the video plays, a window will show the detection results over each single frame in the video sequence in the form of one or more rectangles that indicate patches of the image in which the target sign has been detected.

Press Esc to halt the application.