# **LPR Software, Calibration procedure – Appendix**

[Overview 1](#_Toc501573527)

[Step 1: ROI selection 2](#_Toc501573528)

[Step 2: Plate Detection 3](#_Toc501573529)

[Step 3: Characters Recognition 5](#_Toc501573530)

[Step 4: Miscellaneous parameters 7](#_Toc501573531)

[Step 5: Sweep Space 8](#_Toc501573532)

## **Overview**

The purpose of this appendix it to describe the calibration procedure, which is mandatory for a new installation. The outcome of that procedure is a stable working-point for the LPR algorithm, which is optimal for the installation scene.

The calibration procedure comprises multiple steps which shall be taken sequentially, one by one. Skipping one of the steps is not recommended and might fail the calibration flow.

Figure-1 provides an example input frame, that applied as case-study for demonstrating the calibration procedure:

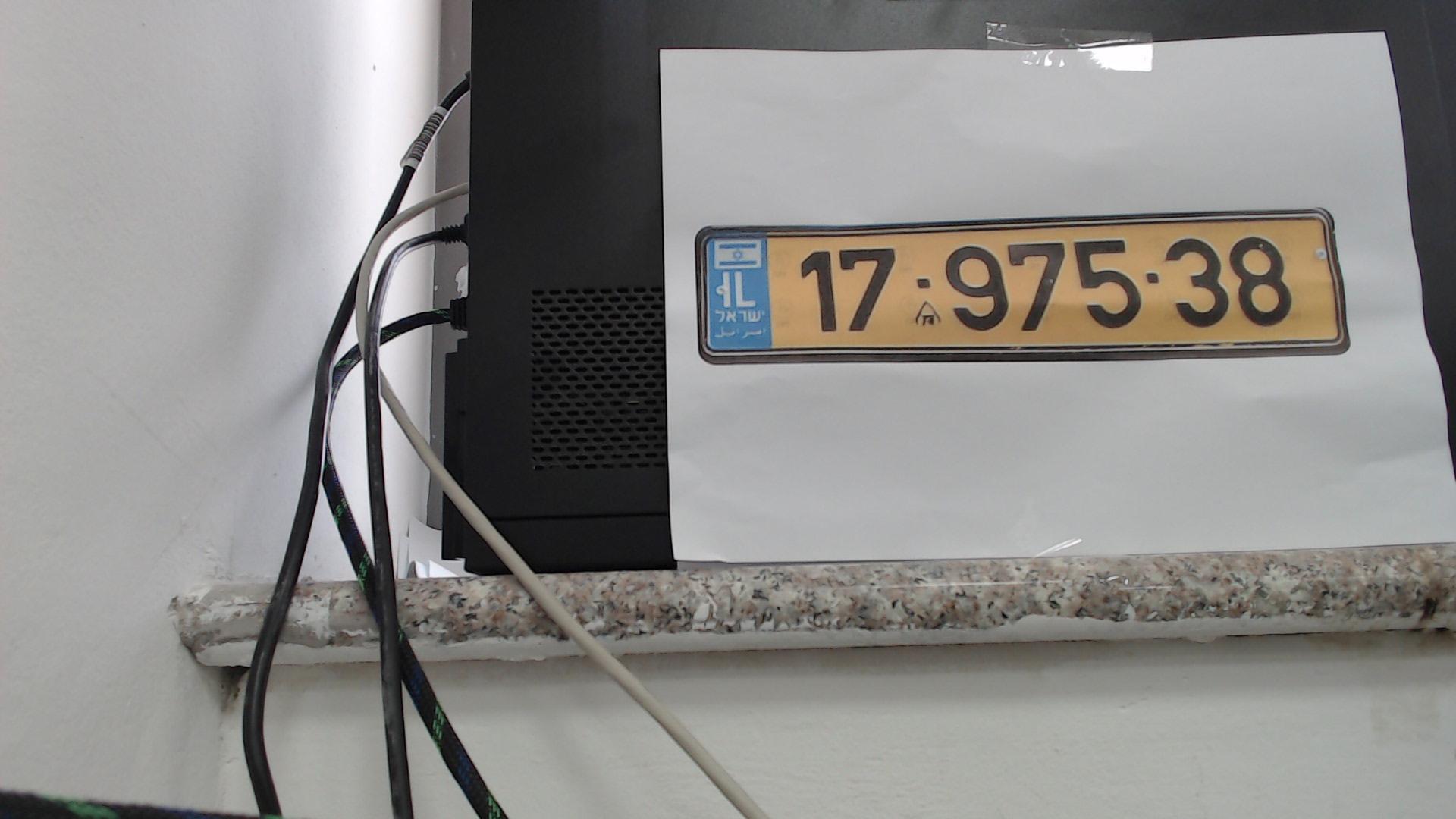


Figure - case-study, input-frame

Note: all of the following steps will apply the *--batch --debug* flags, which are useful for the calibration procedure. The *--debug* flag can be omitted once calibration is completed.

## **Step 1: ROI selection**

**Goal**: Find an appropriate ROI, in which the license-plate is fully captured.

Run the LPR executable with the input image, Using the -i flag:

% *./build/lpr -i $cwd/CalibrationDemo\_1797538.jpg --batch –debug*

Figure-2 describes the cropped image, as generated by ***image\_original.jpg***.

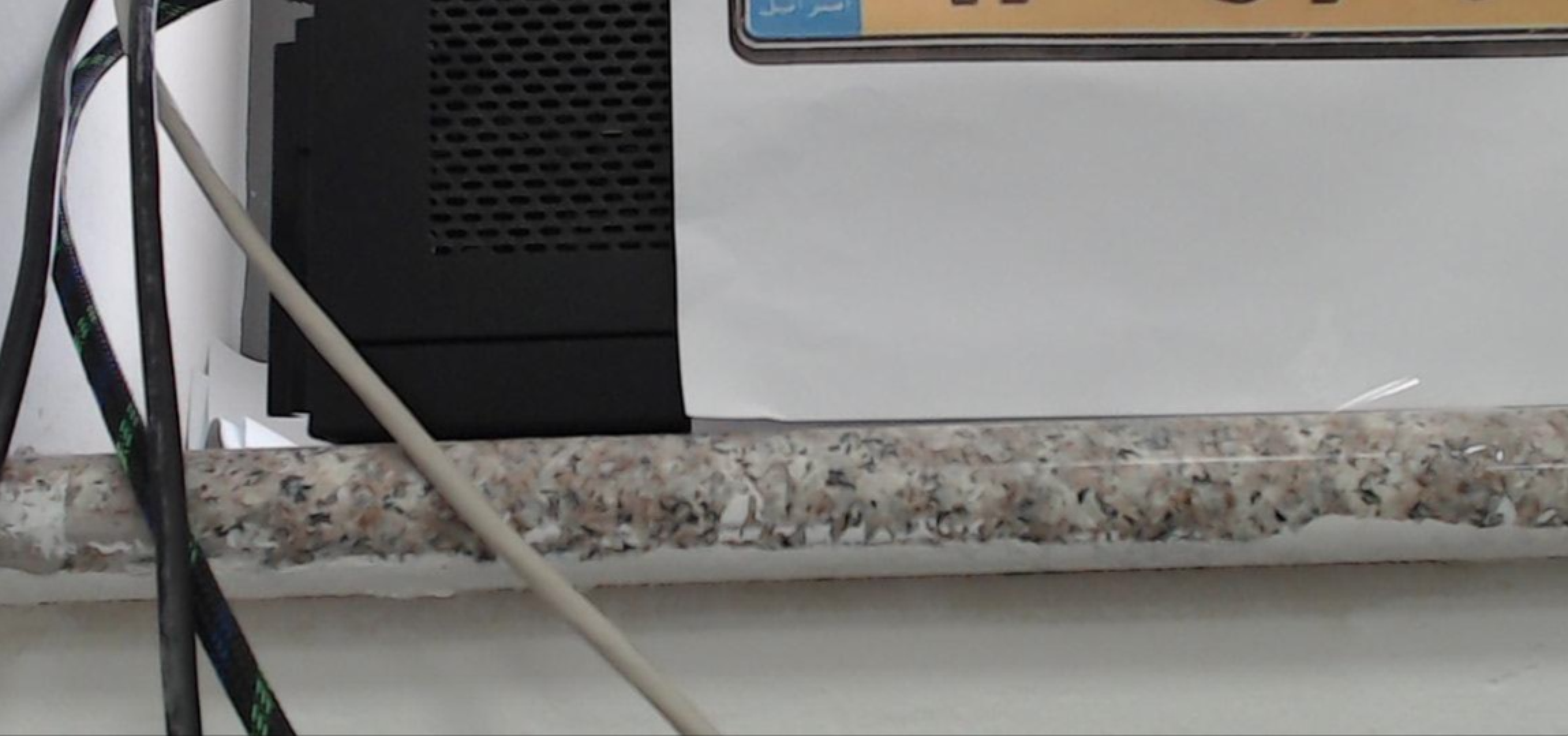
**

Figure - ROI selection, before using the ROI flag

The original input-image size is 1920x1080, and it’s obvious that the default ROI (mid-bottom of the image) is not suitable for that input-image.

The user shall find the appropriate ROI that is suitable for the given scene, as demonstrated hereby below with upper-left point set to (800,100) and (W,H) set to (1000,500):

% *./build/lpr -i $cwd/CalibrationDemo\_1797538.jpg --ROI="(800,100,1000,500)" --batch –debug*

Figure-3 describes the cropped image, as generated by ***image\_original.jpg***.



Figure - ROI selection, after using the ROI flag

The license-plate is fully contained within the cropped image, which is the desired accomplishment for this step.

## **Step 2: Plate Detection**

**Goal**: Get a clear notion of the license-plate characters, for the plate-detection phase

Figure-4 describes the contour-analysis images that corresponds with Figure-3, as generated by ***image\_contours.jpg*** (all contours), ***image\_contours\_possible\_chars.jpg*** (possible characters, i.e. contours that might be representing characters), and ***image\_contours\_matching\_chars.jpg*** (groups of characters, each in a different color).

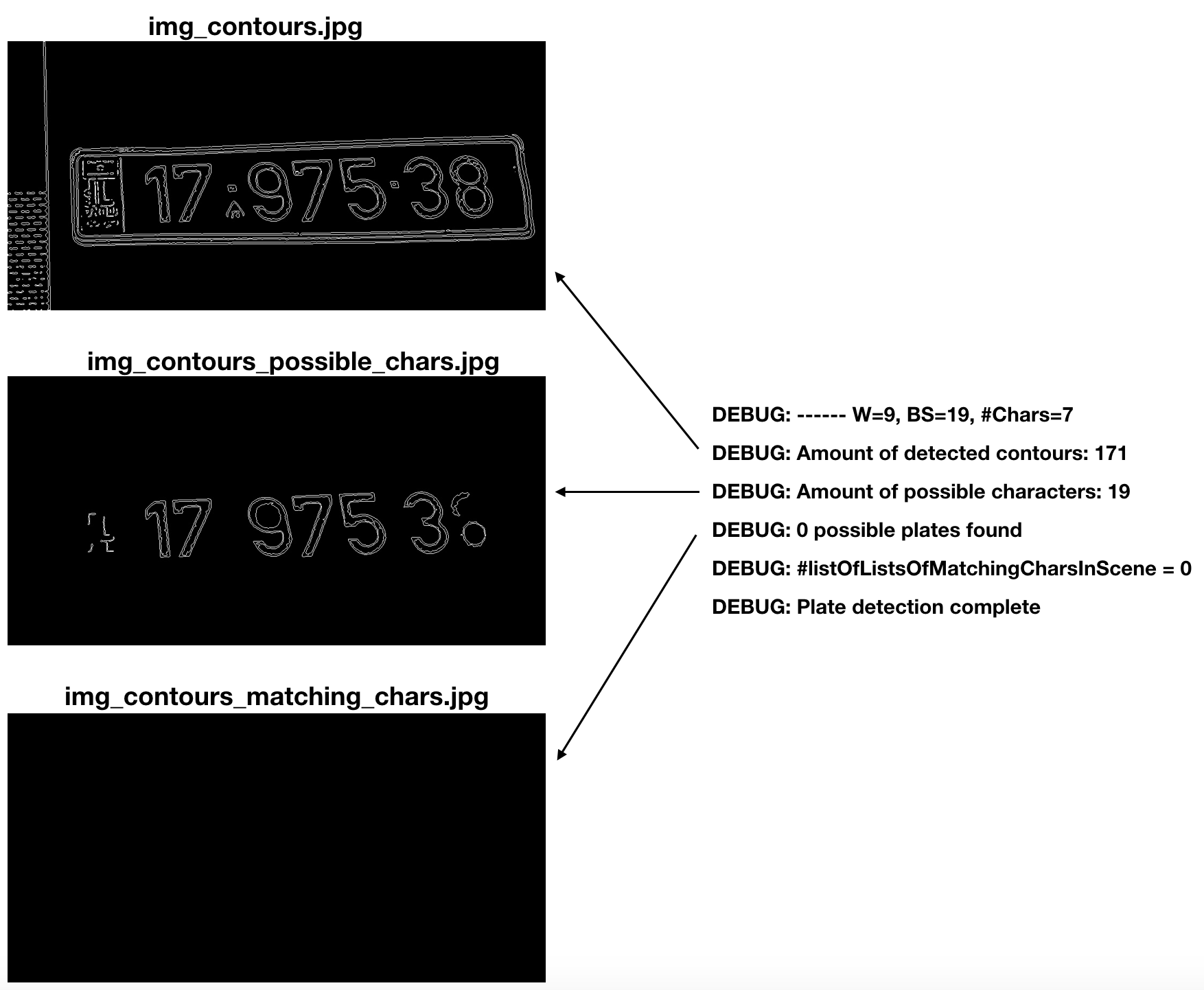


Figure - Contour analysis, prior to calibration step 2

The algorithm detected overall 171 contours, where 19 of them were classified as ‘possible characters’ but none could be grouped into any groups of matching characters (‘possible plates’). That result implies on an incompatible working-point, i.e. algorithm is not well calibrated.

This calibration step consists of tweaking two parameters that affects the preprocessing phase of the LPR algorithm, termed *PreprocessThreshBlockSize* and *PreprocessThreshweight*.

Both parameters directly affect the adaptive-threshold operation, in which threshold value is the weighted sum of neighborhood values where weights are a gaussian window.

The first parameter controls the block-size (Size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, etc.) and the second parameter controls a constant that is subtracted from the mean or weighted mean.

In this case-study example, the calibration step has been ended with values 59 and 9 respectively for these two parameters:

% *./build/lpr -i $cwd/CalibrationDemo\_1797538.jpg --ROI="(800,100,1000,500)" --PreprocessThreshBlockSize=59 --PreprocessThreshweight=9 --batch --debug*

Figure-5 describes the contour-analysis images that corresponds with the above command line, as generated by ***image\_contours.jpg***, ***image\_contours\_possible\_chars.jpg***, ***image\_contours\_matching\_chars.jpg*** and **img\_contours\_possible\_plates\_0.jpg**.

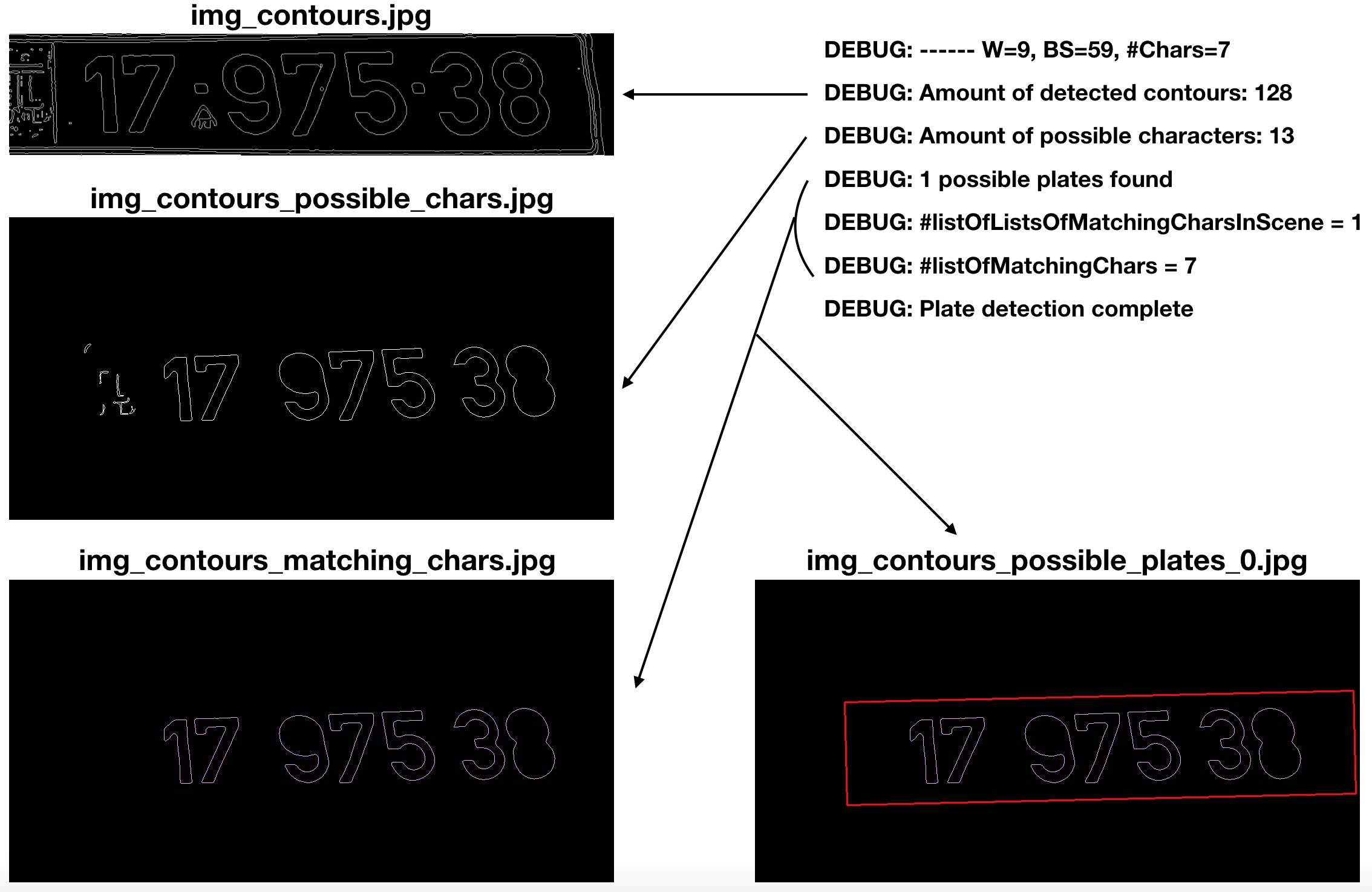


Figure - Contour analysis, at the end of calibration step 2

It can be obtained that the 7 characters were detected and grouped correctly, and thereby lead to a valid plate detection. That is the desired accomplishment for this step.

Note that few of the ‘possible-characters’ (left side) are not representing real characters, but they were filtered out during the groups classification (‘matching characters’).

If possible, it is recommended to try and ‘assist’ the algorithm with filtering out those contours explicitly from the ‘possible characters’ phase. In this case-study, it’s indeed possible with setting the *MinPixelArea* parameter to 4500, and the following steps will utilize this setup.

## **Step 3: Characters Recognition**

**Goal**: Get a clear notion of the license-plate characters, for the character-recognition phase

This calibration step consists of tweaking the parameters that affects the characters recognition phase of the LPR algorithm, termed as following:

* *MinNumberOfMatchingChars*
* *MaxNumberOfMatchingChars*
* *MinAngleBetweenChars*
* *MaxAngleBetweenChars*
* *MinChangeInArea*
* *MaxChangeInArea*
* *MinChangeInWidth*
* *MaxChangeInWidth*
* *MinChangeInHeight*
* *MaxChangeInHeight*
* *MaxDiagSizeMultipleAway*

Each of the above parameters has a specific role and a clear intuition, implied by its name (documented inline within the code, see the args\_t struct definition in LPR\_wrapper.hpp).

It is recommended to start with loose boundaries (low Min values and High Max values) and then tightening them interactively, until obtaining a valid characters recognition result.

Figure-6 describes the contour-analysis and OCR images that corresponds with Figure-5, as generated by ***img\_possible\_plate[1-4]\_0.jpg*** (intermediate phases of for plate #0), and ***img\_ocr\_result\_0.jpg*** (OCR result).

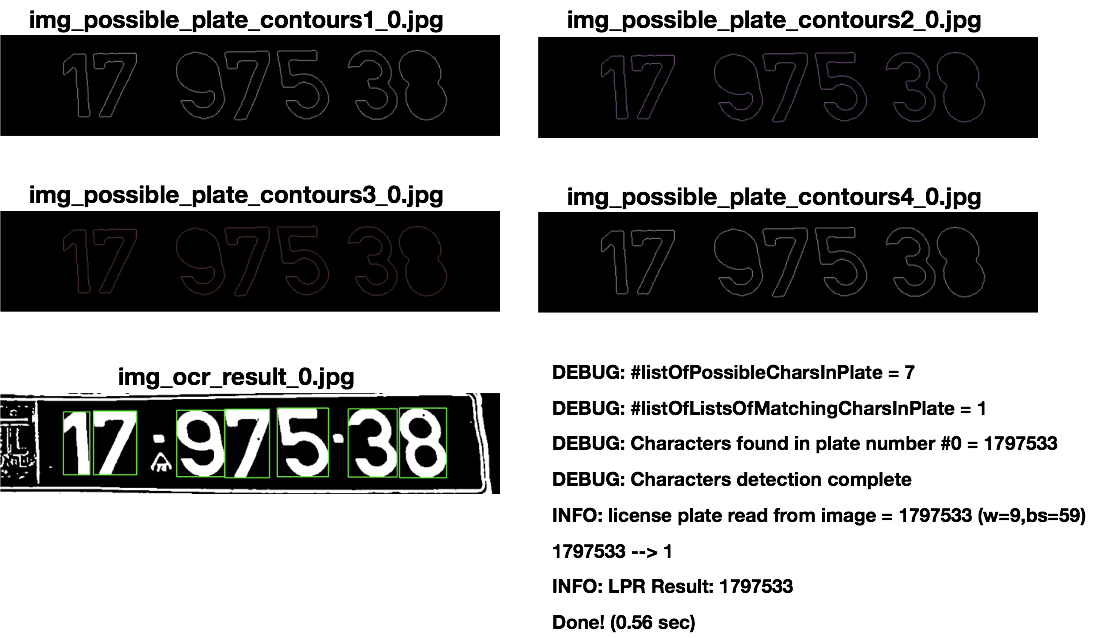


Figure - Characters Recognition, at the end of calibration step 3

It can be obtained that the 7 characters were detected and grouped correctly in plate #0. The OCR is valid as well, and each character is wrapped correctly by the green rectangle. That is the desired accomplishment for this step. Hence, no need to tweak the above parameters further.

Note: The rightmost character in that case study was detected as ‘3’ instead of ‘8’ due to a non-rigid distortion of the input image (wrinkled paper, left side is lower than the right side, see Figure-1 and Figure-3). That distortion has confused the KNN engine which is behind the OCR phase. That’s not a real case, so that the attention in that appendix shall remain focused on the calibration procedure.

## **Step 4: Miscellaneous parameters**

In case that the above steps (mainly 2-3) didn’t worked out well, it might be needed to tweak some other miscellaneous parameters, termed:

* *blueMaxThrH*
* *blueMinThrS*
* *imgEnhancementEn*

The first two parameters filters out characters that has “too blue” (targets the Israeli blue symbol on the license-plates), and might be calibrated to meet the given scene properties. Filtering out such plates is accompanied with a corresponding notification printout to stdout, so it might be needed to teak this parameters if such notifications pops up.

The third parameter (*imgEnhancementEn*) enables an additional preprocessing stage that applies a Warming effect (+CLAHE) and a Saturation effect (+Gamma). Enabling this stage is recommended for relatively dark scenes with a narrow dynamic range.

Finally, a designated autonomous errors-correction (ECC) mechanism wraps up the KNN engine, and validates its results. The main purpose of this mechanism is to detect wrong KNN decisions and fixed them automatically. The mechanism has hardcoded thresholds values, which might be fine-tuned. The mechanism is implemented by *OcrKnnCorrections()* and *OcrTextualCorrections()* functions in DetectChars.cpp, which can be disabled by the corresponding parameters *NoOcrKnnFixes* and *NoOcrTextualFixes*.

## **Step 5: Sweep Space**

**Goal**: Find a broad and stable working-space

This calibration step consists of tweaking two parameters that affects the preprocessing phase of the LPR algorithm, termed *PreprocessThreshBlockSize* and *PreprocessThreshweight* (aka ‘sweep-variables’). Those are the very same parameters that were set in Step-2, but here to goal is to broad them into a stable working-space.

The sweep-variables are given as arguments for *frame\_decoder()* function, each argument is passes with the annotation of start value, end value and increment step.

The previous calibration steps for the case-study ended up with *PreprocessThreshBlockSize*=59 and *PreprocessThreshweight*=9.

Setting the sweep-variables in main.cpp to:

sweep.push\_back(sweep\_st(7, 13, 2)); //args.PreprocessThreshweight

sweep.push\_back(sweep\_st(55, 65, 2)); //args.PreprocessThreshBlockSize

ended up with the following histogram:

1797533 --> 90

1797538 --> 150

INFO: LPR Result: 1797538

It can be obtained that even the right-most character has been detected correctly, due to the sweep support.

The goal in that calibration step is to reach as distinct histogram as possible, for the correct result. It’s therefore possible to continue and tweak the sweep-variables values until reaching a desired result.