Intelligent Sensors lab 1

Number plate recognision program

Remco Geuze and Casper Le Gras

This document is made in order to help read the code of the numberplaterecognision program. The reason why such a document might be helpful is because the code for this program was written over the course of 11 months and has seen many revisions. Large portions of the code are obsolete however a deadline has made sure that we did not have time to remove these parts of the code.This documentation takes a look at the features, the performance, the history and the general working of the program.

Table of Contents

[Performance 3](#_Toc437379180)

[History 4](#_Toc437379181)

[Interface 5](#_Toc437379182)

[Button panel 5](#_Toc437379183)

[Text output panel 5](#_Toc437379184)

[Image panels 5](#_Toc437379185)

[Overview of the code 7](#_Toc437379186)

[Contact 9](#_Toc437379187)

# Performance

Before going into how the program works it might be worth discussing the programs performance.

The program is built for quadcore or higher core count cpu’s. It can take advantage of up to 5 cores, how that happens will be discussed.

Due to this property the program runs far worse on lower end desktops and on previous and older generation laptops since it cannot take advantage of what it was built for: high core count programs.

On Remco his pc which runs an 8 core 4ghz processor the program takes around 17 seconds to process all 26 given images. This is around ¾ of a second per image. And only when real-time is off. (real-time is only available for the first processing steps and currently takes a large amount of cpu power and slows the program down to the speed of a snail). This speed is quite good taking into account how inefficient parts of the code are.

The program uses around 1.3 gb of ram if working at full speed, most of this is the different instances of images that are loaded in memory. Considering all the old inefficient parts in the code this is actually better than we expected.

The program can recognize all but 2 images, image number 1 and 13 are not recognized at this moment. If the schedule would allow it we would have optimized different filters in a way to get these 2 images recognized as well.

The program is designed for a 1080p resolution monitor and is currently not nicely scalable. You can scale the program down however you then cannot in real-time see what is going on with each number plate. We therefore recommend running the program on a 1080p monitor. The program has multi monitor support and will start on the third monitor if possible. In the case that this is not possible it will start on the default monitor. Of course the program can be moved around at will (multi monitor support is implemented without forcing fullscreen which is often done in java applications).

# History

As stated in the abstract the program has been in development for 11 months, however most of this is not active development. During the winter of 2014/2015 development was very active. Development has not or barely happened for months in a row as well.

The program after realizing the deadline had passed in late 2014 became a hobby project. A project to try new things in. This is clearly recognizable in the fact that there was a linear (single threaded) program which had around 50-60% of the code working still in the program.

Over these months we have learned a lot about programming in general and java in specific. Artifacts of this learning process are also clearly visible. For example in the blobdetection method, which changes the way it detects blobs at multiple points due to trying out new methods which were actually more efficient and/or easier to program. Active development started again in September 2015 and running into October of that year. Development than halted until the last couple of days before the deadline. This is due to both team members being occupied with their minor projects and simply not having the time to do this as well.

Due to this we haven’t had time to continue working on this program in order to resolve the bugs (2 boards are not detected) and optimize the code (and remove the outdated code).

Most of the coding is done by Remco (especially the more recent parts). However most of the algorithms and detection methods have come to be by discussion between Remco and Casper.

Due to this Casper will do more programming in the second lab where again both team members will discuss how to solve the problem at hand and which methods to use.

During the third and final lab both group members will be doing equal parts of programming as well as devising methods and algorithms in order to solve the problems.

# Interface

The program has 2 interfaces, one for single threaded version and one for the multi-threaded version. However since the multi-threaded version is the finished version this is the one we will discuss.

The interface consists of 3 parts:

A button panel

A text output panel

Realtime and Semi-realtime images that are being processed.

Due to the 16:9 resolution it made the most sense to have the button and the text panel underneath each other and use the rest of the space for showing the number plates.

## Button panel

The button panel has a total of 7 buttons.

The top 2x2 area allows the user to continue one of the processes by 1 step and show the result and then halt again.

The ‘Next step all’ button has the same effect as the individual buttons however it will tell all 4 images to continue processing.

The ‘Keep Going All’ button allows the user to stop the halting in between steps and allows for a quick run though of all the images if kept one. This is a toggle button.

The ‘Disable Progress updates’ button allows the user to disable updates on what each process is doing. The program will still output the outcome of each final analysis.

## Text output panel

This panel is used to communicate to the user.

The System output stream has been redirected here, allowing for a simple System.out.println() to get information to the user. Depending on the setting of the last button as discussed above it will either only output the final analysis or it will also keep the user up to date on different processes that are going on. This program has also and can also be used for debugging by printing values or statements to the user. In the case that you want to use that I do suggest only letting one thread continue at the time since that will make it clearer what is happening.

The text output panel will wrap long lines in order to display them and will have a scrollbar if the amount of text does not fit on the size of the textfield.

## Image panels

There are 4 panels to the right. These panels have a label with an imageIcon on them. By updating this imageIcon the programmer can show the user what the current state of the analysis of a numberplate is.

This is also the greatest bottleneck for realtime visualization. This method of showing an image relies on the image being the correct size, since the image being processed is by definition not the correct size this image needs to be rescaled and saved in a new bufferedimage. This makes showing progress in realtime hard on cpu power since it will need to constantly compute the new images.

Each image panel has a tooltip that shows the numberplate number that is being shown. This updates if a new numberplate is starting processing on the place of an old already finished numberplate.

# Overview of the code

Following is an overview of the methods used, without going into great detail, in the analysis of the numberplates.

At the start of the program the interface is made, this is done in the AltUI class. The runner simply calls this class. If you set the AltUI variable to false it will instead load the single threaded program which as stated before is not finished.

The buttons on the panel sometimes notify a certain thread to continue or notify all the thread to continue, the last 2 buttons set variables which change the behavior of the program.

After this the program starts 4 threads, one on each section of the screen and with the first 4 numberplates.

If a numberplate is done it tells the UI and it starts a new thread on the same location with the next numberplate that needs to be processed.

This is done in the startProcessing method of the AltUI class.

After this the actual processing starts in the run method of the ThreadedProcessing class.

The thread starts with reading in the image (all connections with files are handled in static methods in the FileAccess class) and store it in a buffer and show it on the screen. (at this point the tooltip is also configured/updated)

The image is then send through a histogram equalisation in order to get all the images on similar colour ranges (makes colour filtering easier) (From heron referred to as HGE)

After each step the image is saved into a folder for detailed analysis if the user wants to do that.

After histogram equalisation the image is run through a yellow filter. The resulting image is a black and white image with if the filter is correct the largest blob of white being the numberplate.

A blobdetection algorithm is run to determine the largest blob. This algorithm actually finds the 9 largest blobs which allows it to be used later on as well.

After this the coordinates of the largest blob are retrieved and a new images is cut out of the original image (after histogram equalisation) in order to retrieve a smaller image with only the place on it.

Here the first check comes in. A numberplate is far wider than it is high normally. This is used to determine if the colorfilter worked correctly for the specific numberplate. If not the program will send the original image (after HGE) through a different filter and do the same steps as above.

The end result of this can be found in the OnlyPlate folder. This is a small image with the complete numberplate on it (for all but the first image).

This image is then send through a black filter in order to find the individual characters. This is then send through the blobdetection algorithm. Here is where the second adaptive check comes into place. This checks if one of the blobs is wider than it should/could be if the image was filtered correctly. This allows the program to see character that are on an angle, leaving darker areas in the corner which can be detected as letters.

The image is then segmented in the 6 blobs which are most likely the characters in order from left to right.

The program will use a different black filter if the numberplate was not directly found and had to use the second colour filter. This because the colour ranges in the images are different if the image was not directly found and this requires a different black filter.

If the numberplate was directly found then the output of blob detection algorithm for the segments is send through a smoothing filter in order to connect areas for detecting.

After all this each individual segment is cut from this output. It is first set to black and white (instead of the different colours which show different blobs).

Then the image is compared with all the templates, both black and white pixels are compared in order to reach a certain percentage certainty that the given segment is a character.

In here the program has a third adaptive check. It checks if the given segment is large enough to be a character.

If not then the program will rerun using again a different black filter, a larger area smoothing filter followed by another blob detection algorithm and another segmentation test.

If the segment is still not of the correct minimum size then the program will run the previous step again using slightly tweaked filters. The filters are all setup in such a way that they work very well for most of the inputs and have really good detectable outcomes if they will not work for this image, this allows the program to per numberplate go through the necessary steps until it finds the best filter and can recognise the numberplate.

There is another check in the segmentation method. Since we are comparing templates which are of a different size then our segmented images we resize either (or both) images until both are the same size. However this led to some problems with small segments consisting mostly of black areas (black as in on the numberplate black, not as in after the filtering black) and which where rotated or at a slight angle. These images would give a high likelihood to being the character ‘i’ since when this is scaled it is just a white square. In order to combat this we came up with the following solution:

If the non-scaled segment is far taller than it is wide then it is possible to be an ‘i’, however if not then it is more likely to be a small different number that does not scale nicely or is at an angle.

Therefore if we check for the character ‘i’ we also check for the ratio between width and height. If this is off then we multiply the found percentage for this character by 0.5. The idea is that if we have a small ‘i’ we can still detect it however we won’t get false readings with the character ‘i’ as often anymore.

At this point all that needs to be done is writing this output to the screen for the user to check and check it with the build in database of numberplates which can be found in the AltUI class.

# Contact

For a more detailed explanation of what happens certain parts of the code please refer to the attached javadoc documents, these are mostly up to date with the latest changes. If this is not sufficient please contact Remco Geuze for a demonstration and walkthrough of the code.

The complete code can be found on https://github.com/Ylvakiller/Intelligent\_Sensors/ by the end of this week (due to lack of internet at the computer where the latest changes are programmed on it is not possible to synchronise the commits until the end of this week. The repository should be up to date by 14 december 2015).