Haute école ICHEC-ECAM-ISFSC

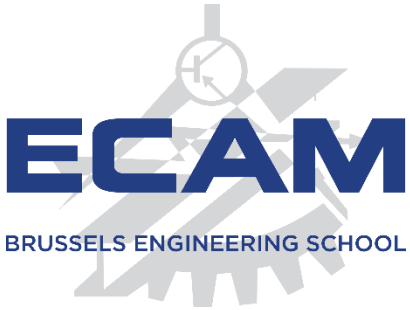


Image Processing II: Assessment report

Ishihara test cheat software

Hugo de HEPCÉE

Academic year 2025-2026

1. Intro

This project takes place in the Image Processing II course given at the ECAM, as the final assessment of the year. Its goal was to implement the skills learned through the course in the development a complex computer vision software.

This report focuses on the implementation of a software designed to cheat the Ishihara test (Fig.1), aimed to detect colour blindness.

Test plates are shown to the computer's camera, and the number should be converted into text (i.e., the number should be decoded from a picture and turned into characters). A black and white version of the plate should also be superimposed on the camera view window.

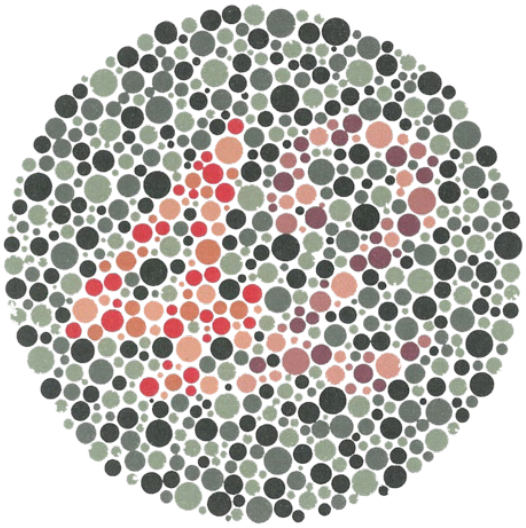


Fig. 1: Example of an Ishihara test that shows 42.

This project was implemented in Python, mainly using the opencv library, along with scikit-learn (used for clustering, refer to point 3b) and tesseract (for OCR purposes, refer to point 3d).

2. Working principle

When launched, the program shows a video stream from the camera, allowing the user to position the plate and check it before conversion. Note that the conversion isn't happening continuously, as it is a fairly high CPU-intensive process.

When the user presses the space bar, a snapshot of the camera frame is taken and fed into the detection and conversion process, which consists of the following steps:

1. Circle detection.
2. Colour sorting.
3. Smoothing.
4. Image to character conversion.

Additionally, the black and white image from step 2 will be superimposed onto the camera video stream display.

3. Key challenges & solutions

Let's go through every single step of the process and identify the main challenges and the solutions implemented:

1) Circle detection

This step finds and isolates the Ishihara test plate into the pictures taken from the camera. It first looks for one circle in the frame (using the Hough Gradient method) and uses it to mask out the zones that aren't the plate. This step wasn't the most complex to implement. The main concern was to tune the circle parameters.

2) Colour sorting

This step sorts the dots by colour to determine whether they belong to a digit or the background, to convert it as a black and white image. This image will be used to extract the digits out of it through OCR and be superimposed onto the camera view.

The first strategy implemented to answer this challenge was to create a mask on the image based on the most present colour. To fetch that very colour, the code iterates through every single pixel in the frame, counting for each found value. This gives a dictionary with, as keys, every colour value present in the image and with the values representing the number of pixels being that colour. That code was tested using RGB, HSV and BGR colour spaces.

The major flaw with this technique is that the mask's colour bounds must be tuned very precisely and required extensive testing. Another technique was therefore needed to be found.

The solution was to use clustering. It works by grouping objects that are similar to each other in groups called *clusters*. Here, the sorting criterion is the colour of each pixel, in a YCbCr colour space (Y being the luma, Cb the blue difference and Cr the red difference). The pixels of the image are then sorted in two different clusters, allowing the creation of a black and white image (Fig. 2) showing in what cluster is every single pixel (black for one cluster, white for the other). This is the processed image that will be used to extract a number out of it.

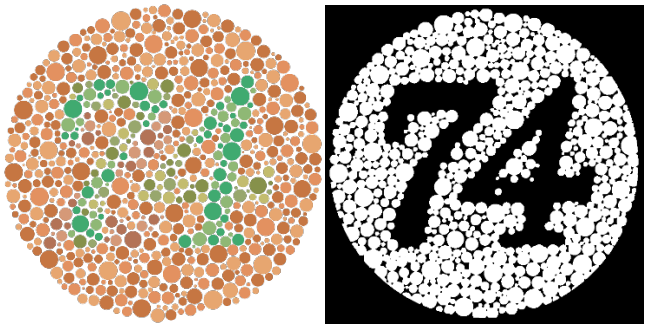


Fig. 2 : Ishihara plate before and after clustering.

3) Smoothing

The goal of this step is to smooth the black and white picture extracted previously, trying to form a continuous shape ready to be decoded with OCR.

As in step 1, the main challenge here was to find out how much to smooth the image.

4) Image to character conversion

Finally, the last step is to recognize the numbers in the plate and translate them into characters by means of OCR (Optical Character Recognition).

Google's tesseract library was used through the pytesseract library. It looks for numbers in the image and outputs them as a text string, which is then shown on the live camera view.

4. Results and discussion of their quality

A video presenting the results is available here, alongside with a demo:

https://www.youtube.com/watch?v=TmbpMhuRm3M.

When using a virtual camera, the program performs as expected (cf. Addendum 1). When the plate is the right size, it manages to pick it out of the frame, detect and convert the numbers. However, if slightly too zoomed in or out, the program doesn't detect the plate's circle properly and thus fails to convert it.

It is not performing as well when using a real camera. Due to the uneven light conditions and the shadows that may appear on the plate, the program struggles to pick out the circle. It is often off-centre. Even though, when the circle is picked up as it should, the shadows are interfering with the clustering process and often prevent the number from being read by the OCR.

5. Improvement perspectives

As previously stated, capturing plates with a real camera doesn't work reliably. One possible improvement perspective would be to implement a light correction algorithm to preprocess the frame, correcting for shadows or uneven lighting. This would increase the reliability of the program by feeding it pictures with somewhat similar conditions.

Another improvement would be to make it convert continuously the input, removing the need to press the spacebar to start reading a plate. This would require reducing the code run time, maybe by switching to another programming language (such as C++ for instance).

Finally, for awareness-rising purposes, it could be interesting to implement a way to show what a colourblind person would see. It could also be done the other way, i.e., converting the image in a range a colourblind person could look at and not be affected by their handicap.

AI Usage section

Apart from GitHub Copilot auto-completion extension on VSCode, no GenAI tool was used during the development of this project. ChatGPT was used to read this PDF and to point out the spelling/grammar mistakes.

About the libraries used in the code, opencv was used mostly for image manipulation (circle detection, masking, blurring, adding text, etc). scikit-learn was used for the clustering, as its documentation is way more thoroughly written and organized. Regarding tesseract, it was used as it has been used in another project of Ishihara decoding available online[1].

[1] (TheReal Monkey, 2021)

References

Delhaye, Q. (2024). *Image Processing - Session [1-3].* Brussels: ECAM Brussels Engineering School.

OpenCV. (2025, December 11). *OpenCV-Python Tutorials*. Retrieved from https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

pypi. (2024, August 16). *pytesseract 0.3.13*. Retrieved from pypi: https://pypi.org/project/pytesseract/

Scikit. (s.d.). *Clustering*. Retrieved from Sckikit Learn: https://scikit-learn.org/stable/modules/clustering.html

Syed, N. R. (2018, March 25). *Image segmentation via K-means clustering to decipher color blindness tests*. Retrieved from Najam R. Syed: https://nrsyed.com/2018/03/25/image-segmentation-via-k-means-clustering-to-decipher-color-blindness-tests/

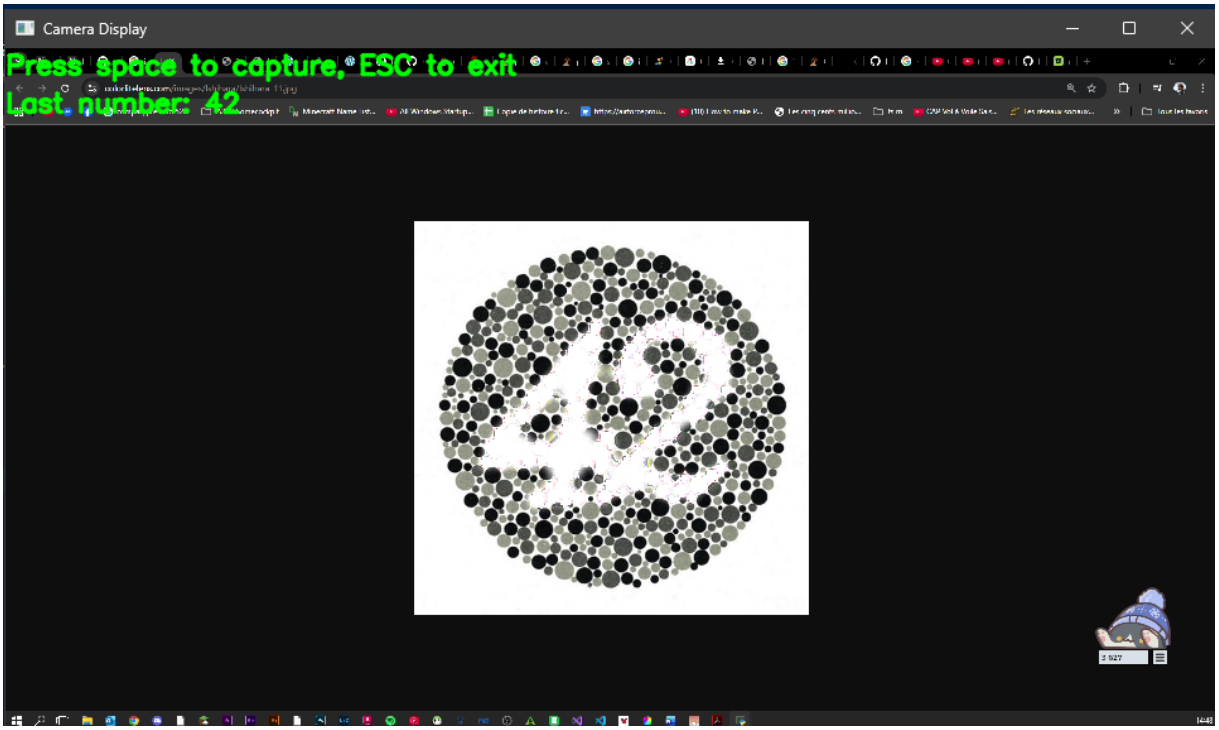
TheReal Monkey. (2021, August 10). *number_recognizer.py*. Retrieved from Bitbucket: https://bitbucket.org/itsmonks/colorblind_test/src/master/number_recognizer.py

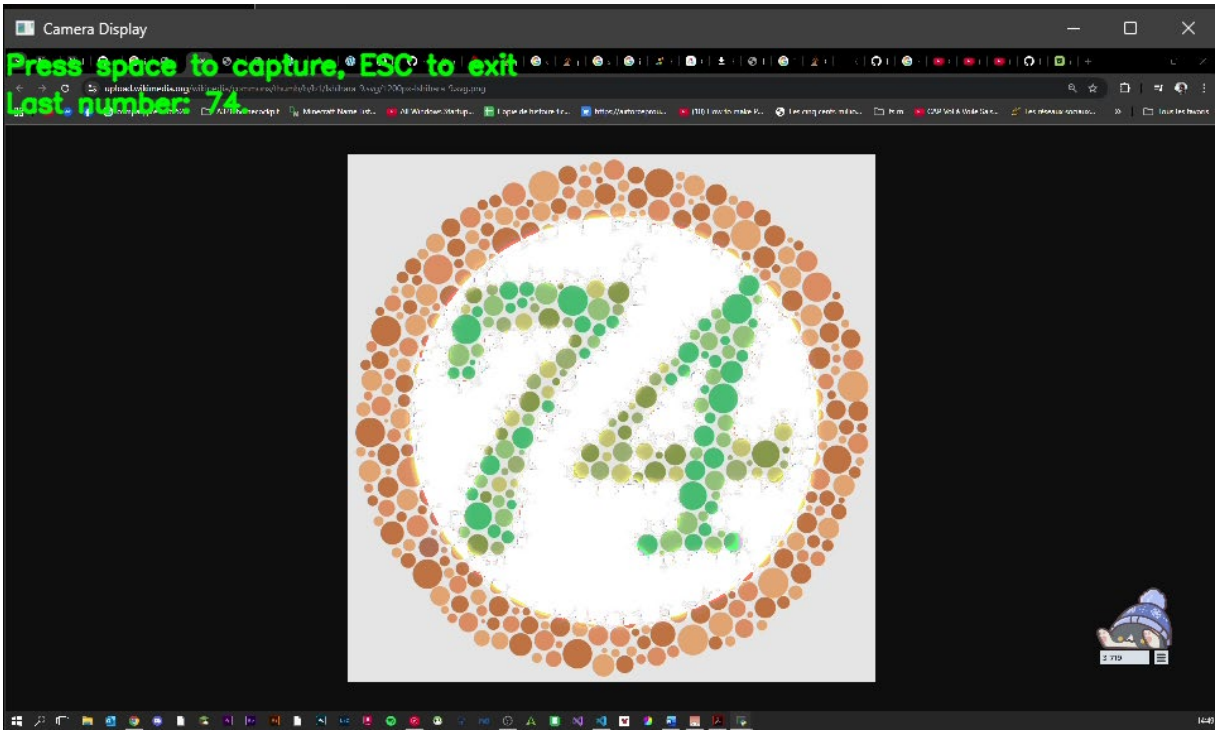
Wikipedia Contributors. (2025, December 11). *Cluster Analysis*. Retrieved from Wikipedia: https://fr.wikipedia.org/wiki/Clustering

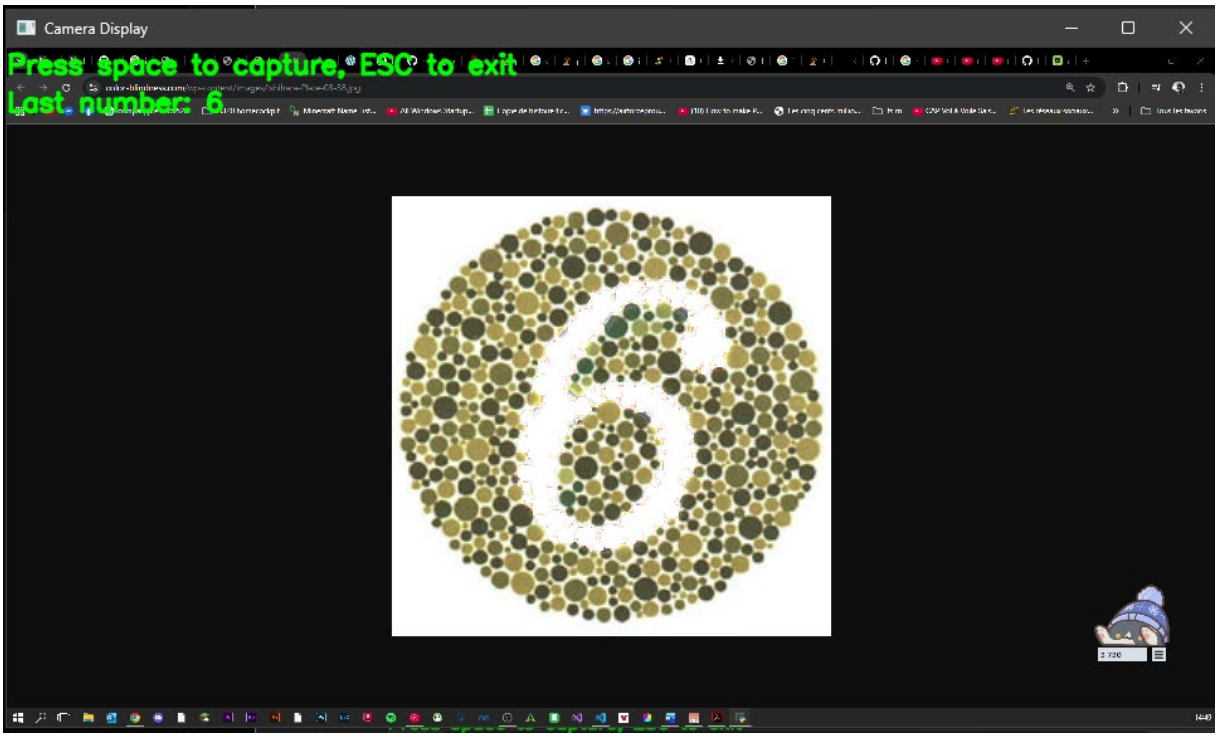
Addendums

1. Results with a virtual camera

The white number/circle around the number is superimposed by the program. The captured image is actually a regular Ishihara test. The decoded number is shown in the top left corner.






Camera Display
Press space to capture, ESC to exit
Last number: 6