Rubik's Cube

FACE DETECTION

By Ceron Andrea, Donà Ivan

Introduction

1. Detect a Rubik’s cube colors
   To help colorblind people play this game

2. Work with three faces in unison
   Increased difficulty compared to having only one face

3. No usage of Neural Networks
   Helps to better learn the Open CV2 environment

4. Adaptability to different cube designs
   Since a Neural Network is not used, a total redesign is not necessary

Image preprocessing

1/10

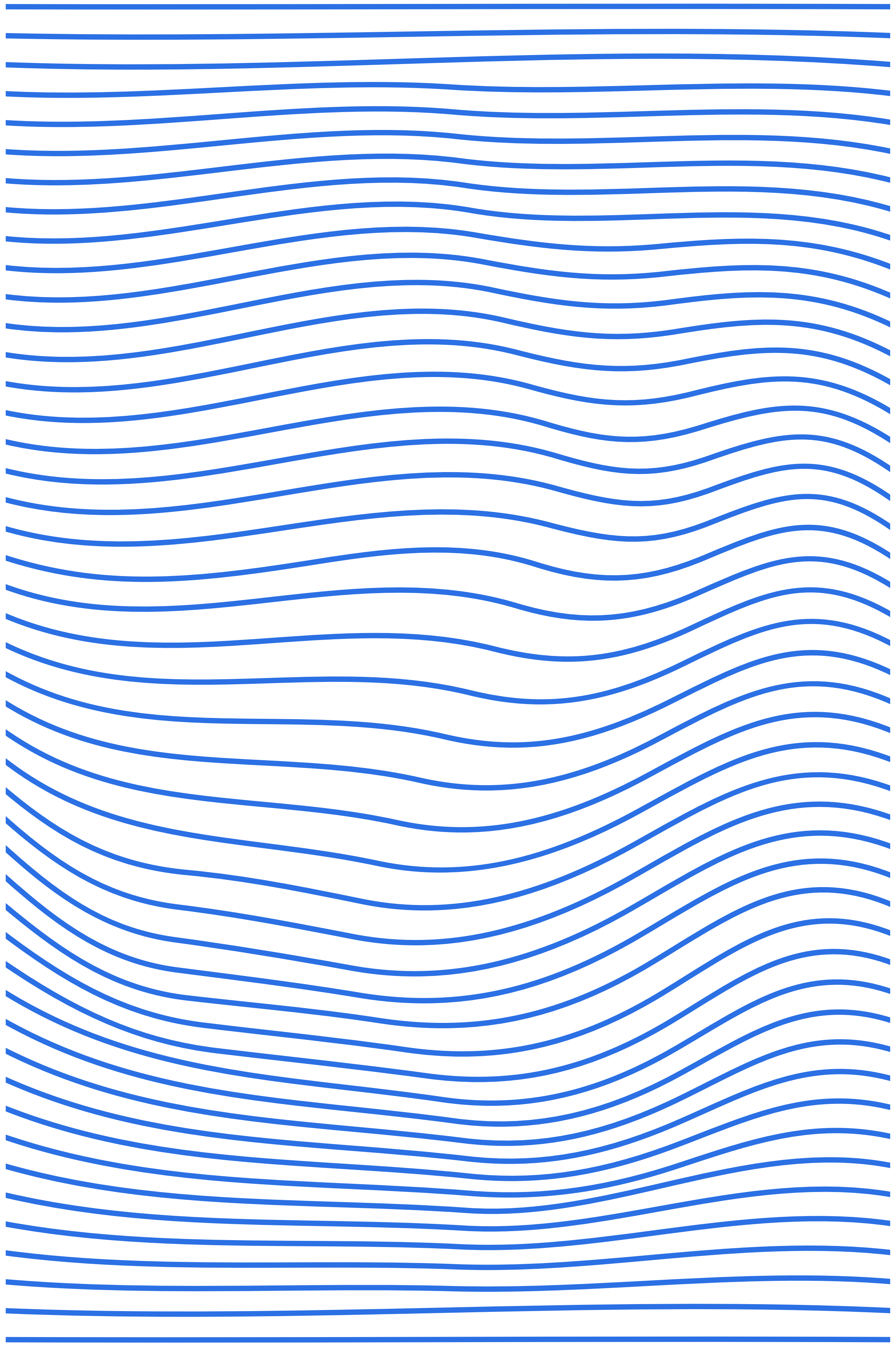
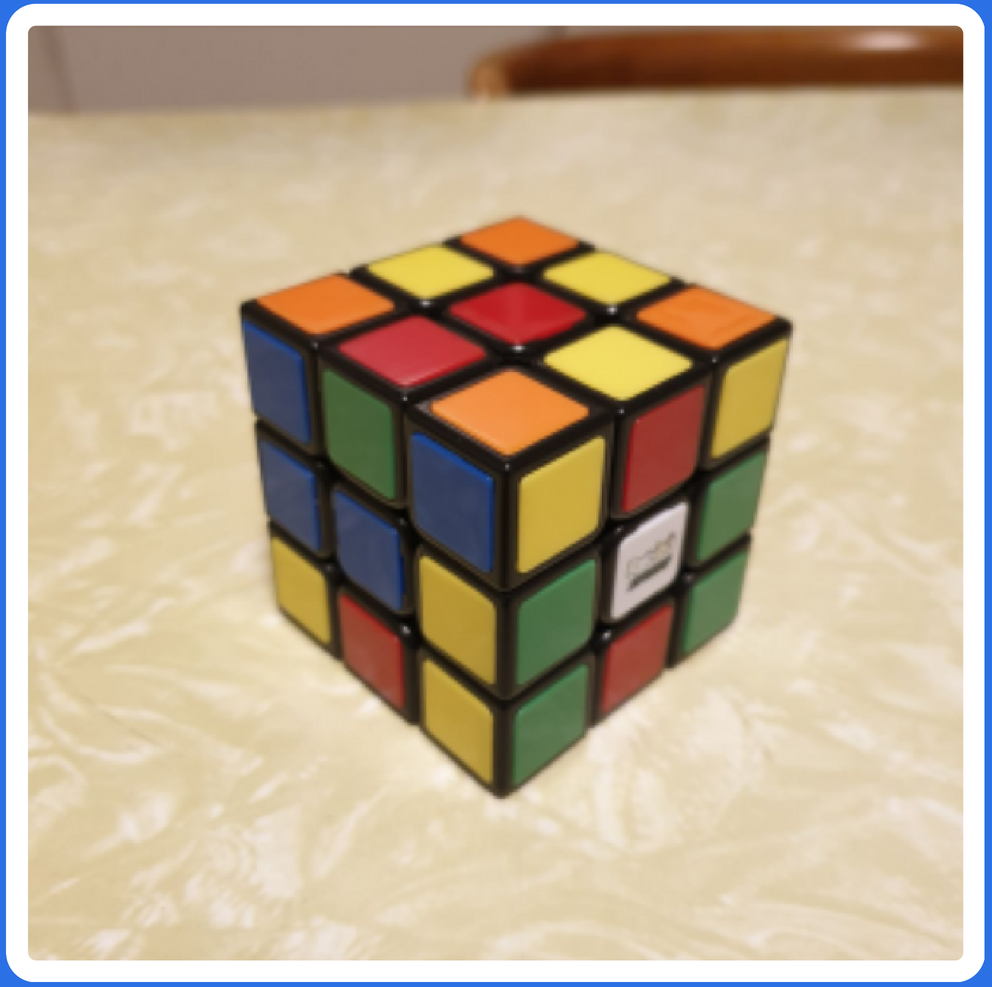


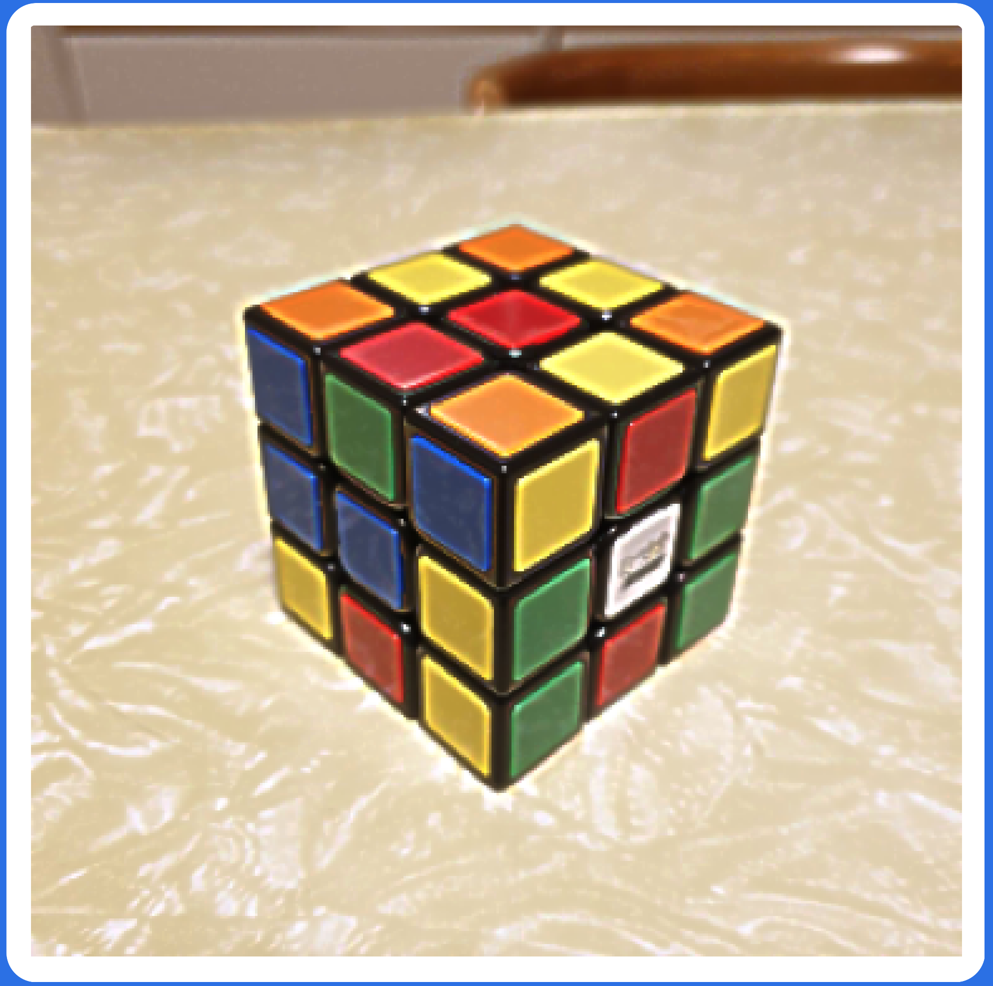
Image resized to 300x300

Extending the border pixels if not already square
Then resizing to desired resolution
This effectively removes unnecessary information



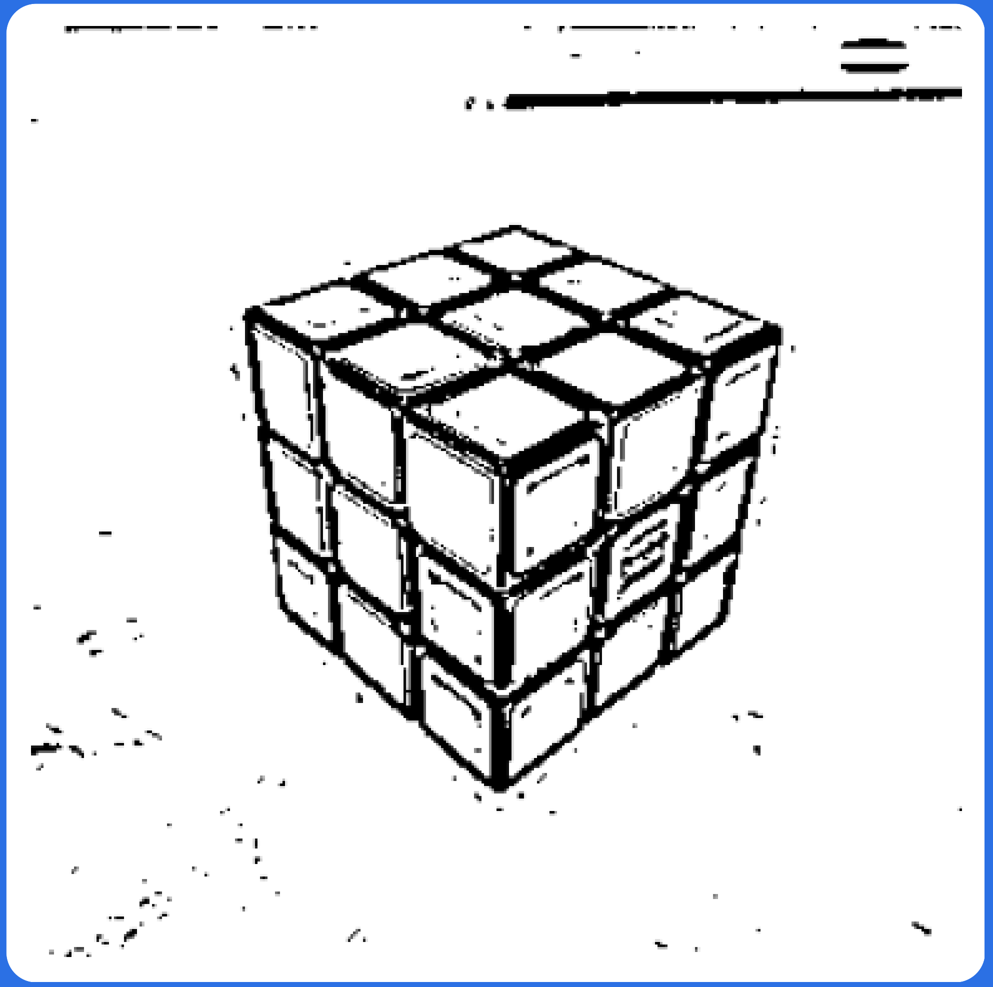
Edges enhanced

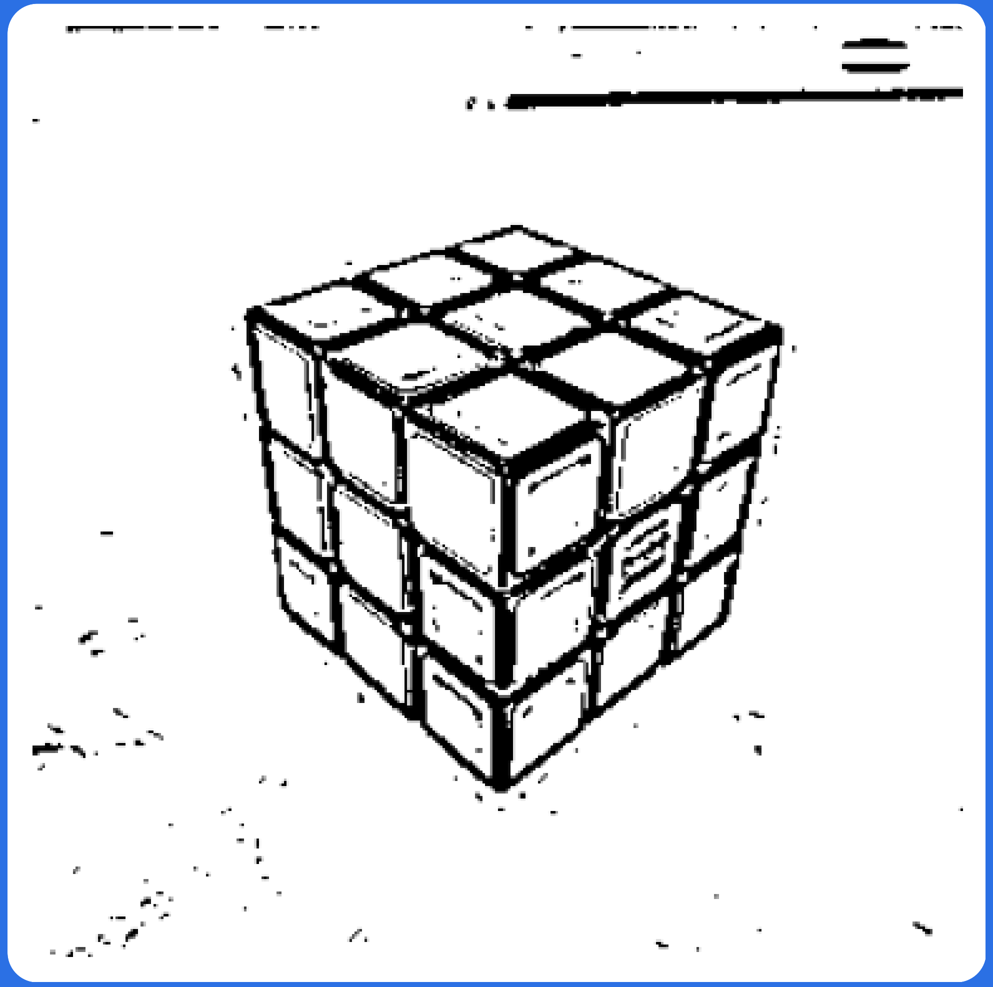
By utilizing a high-pass filter
Applied to each one of the three color channels
Results combined in to a single BGR image



Adaptive thresholding

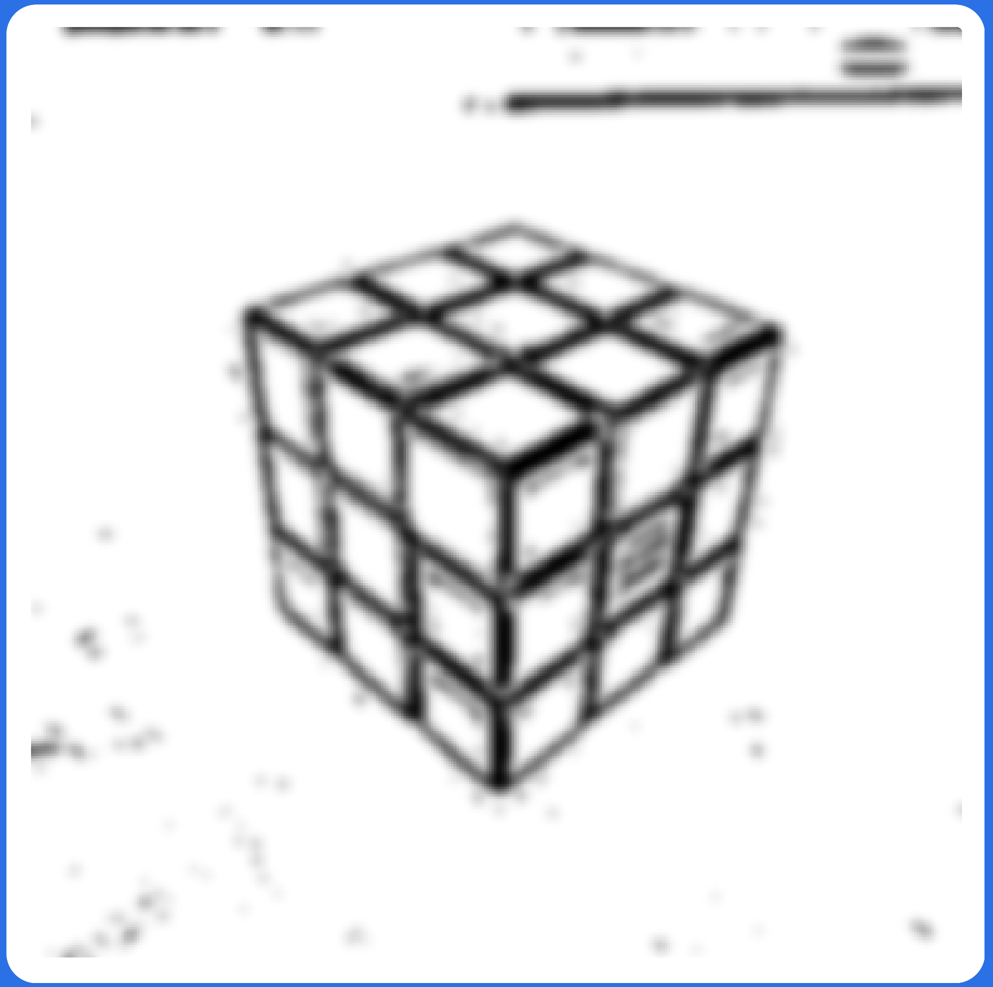
The first step it to turn the image monochromatic
The following thresholding emphasizes the edges
The threshold utilized is based on a local region



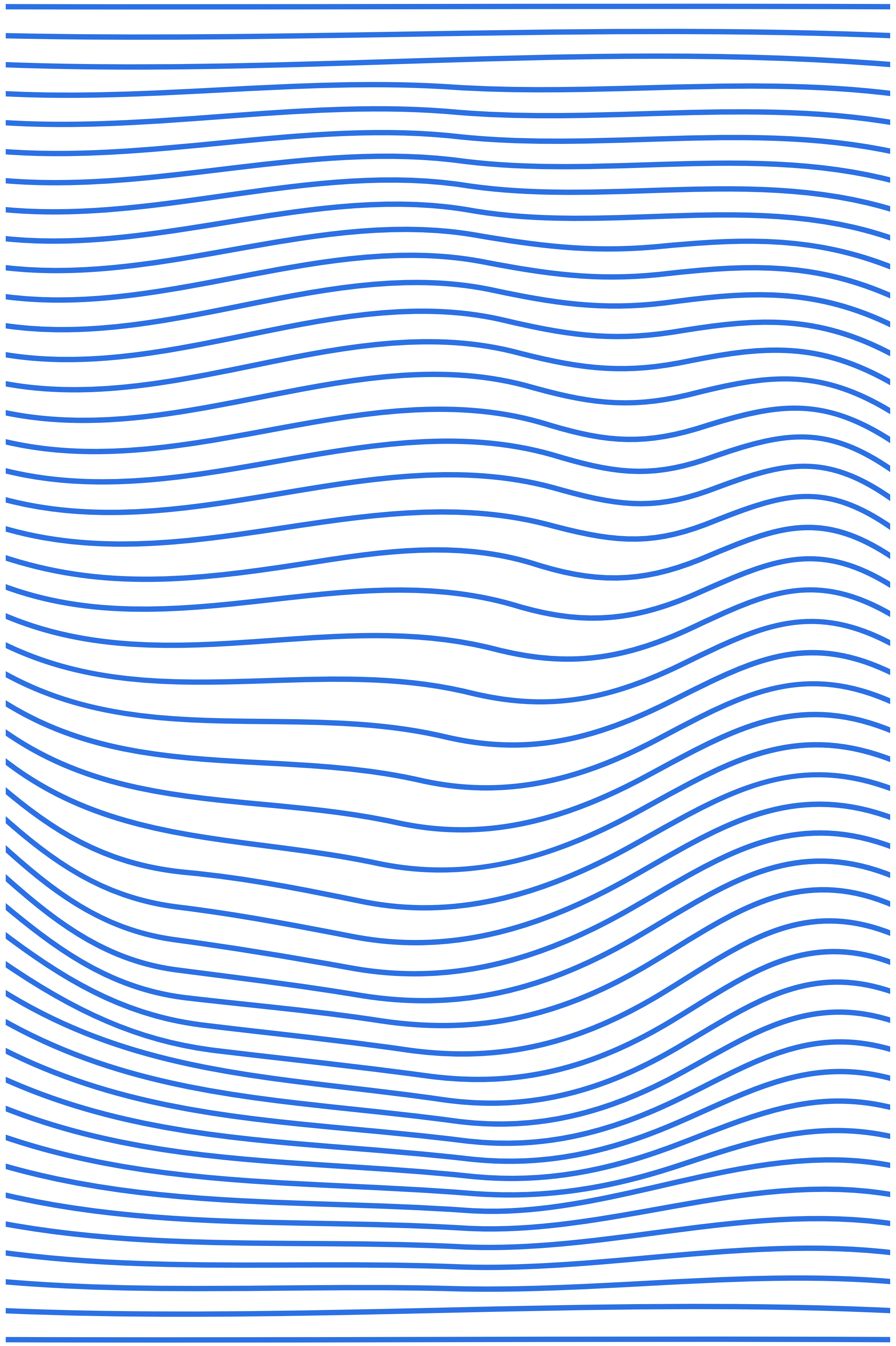


Gaussian filtering

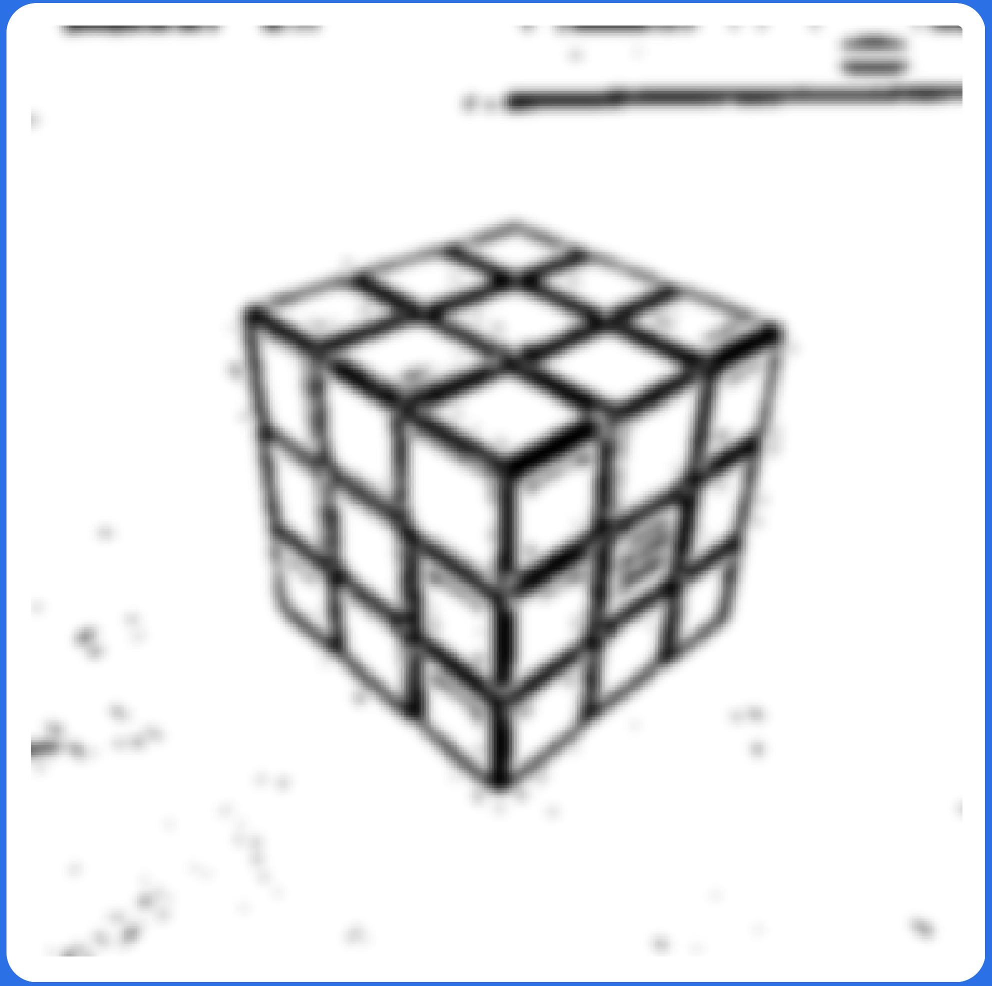
Lowers the impact of the noise, making it brighter
The borders of the objects maintain their darkness
On the result is applied a brightness normalization



Separation from fine noise

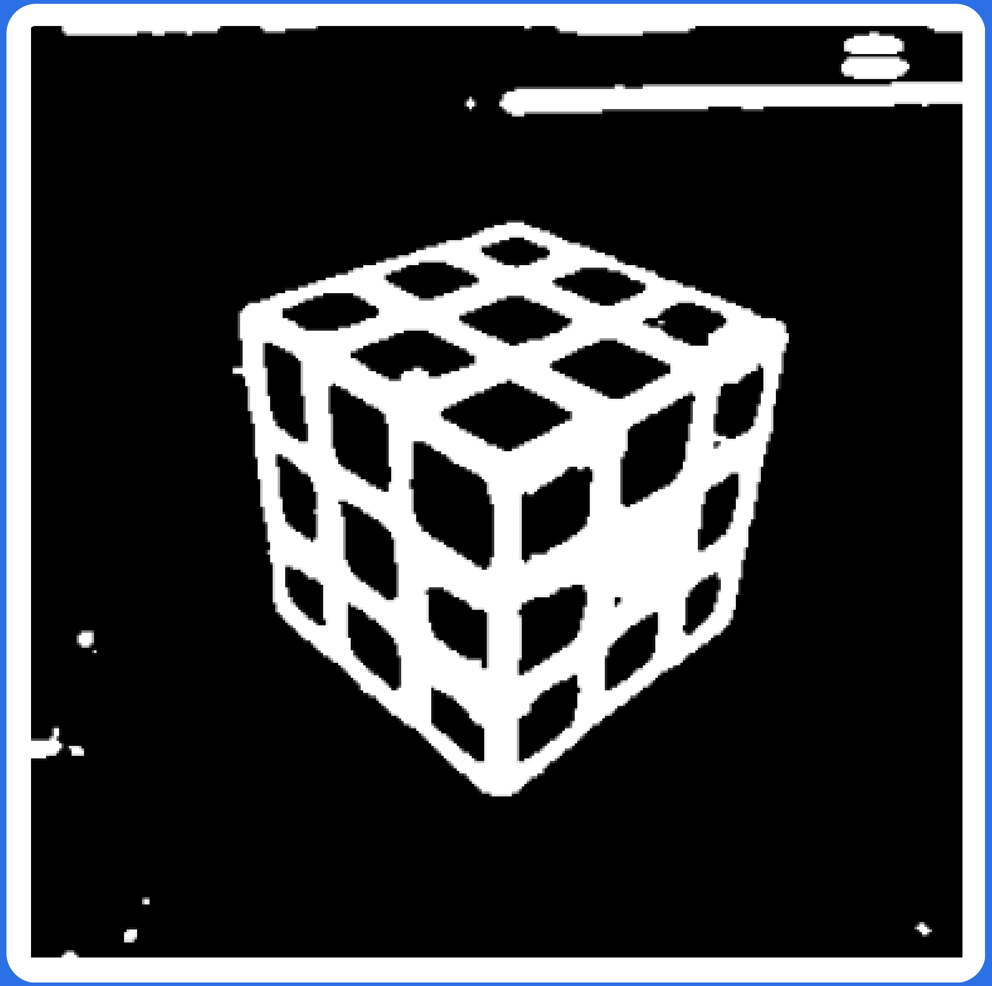


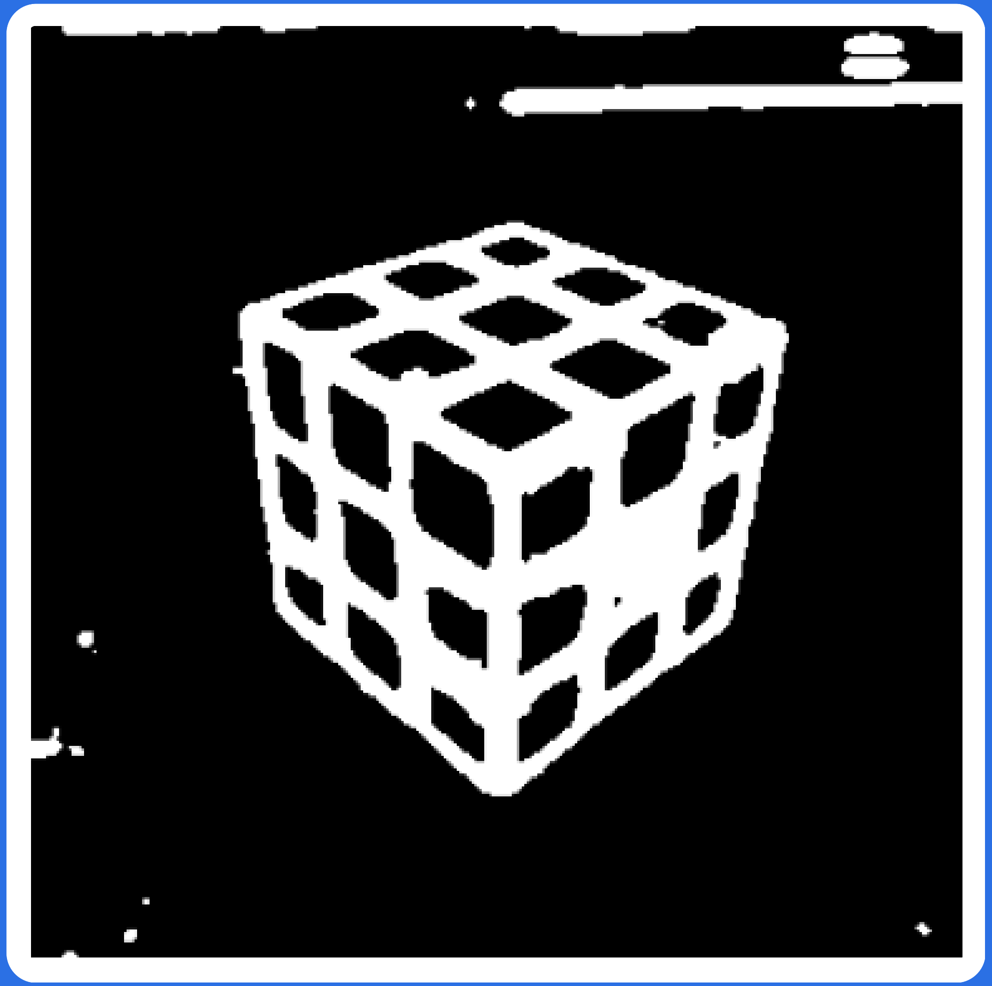
2/10



Binary thresholding

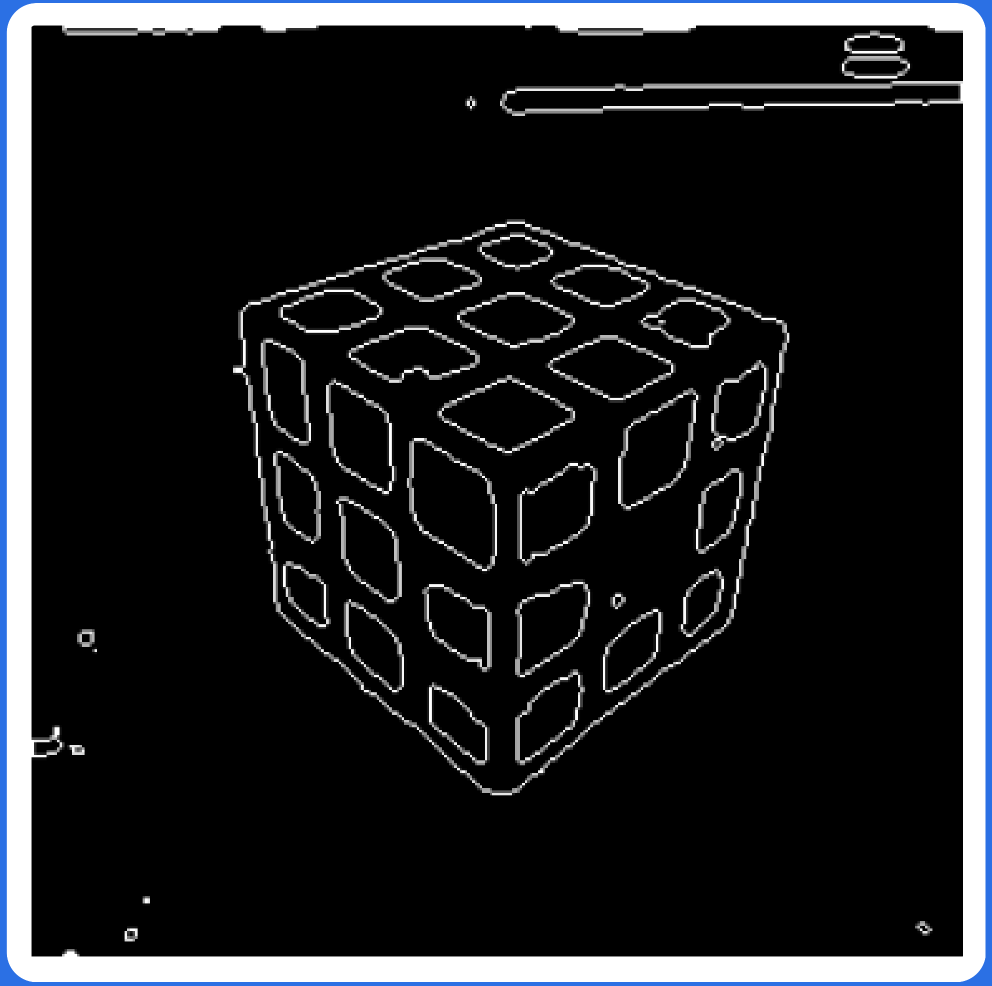
This threshold, instead of the previous, is based globally
The process removes the fine noise
The output is inverted, as white is considered foreground



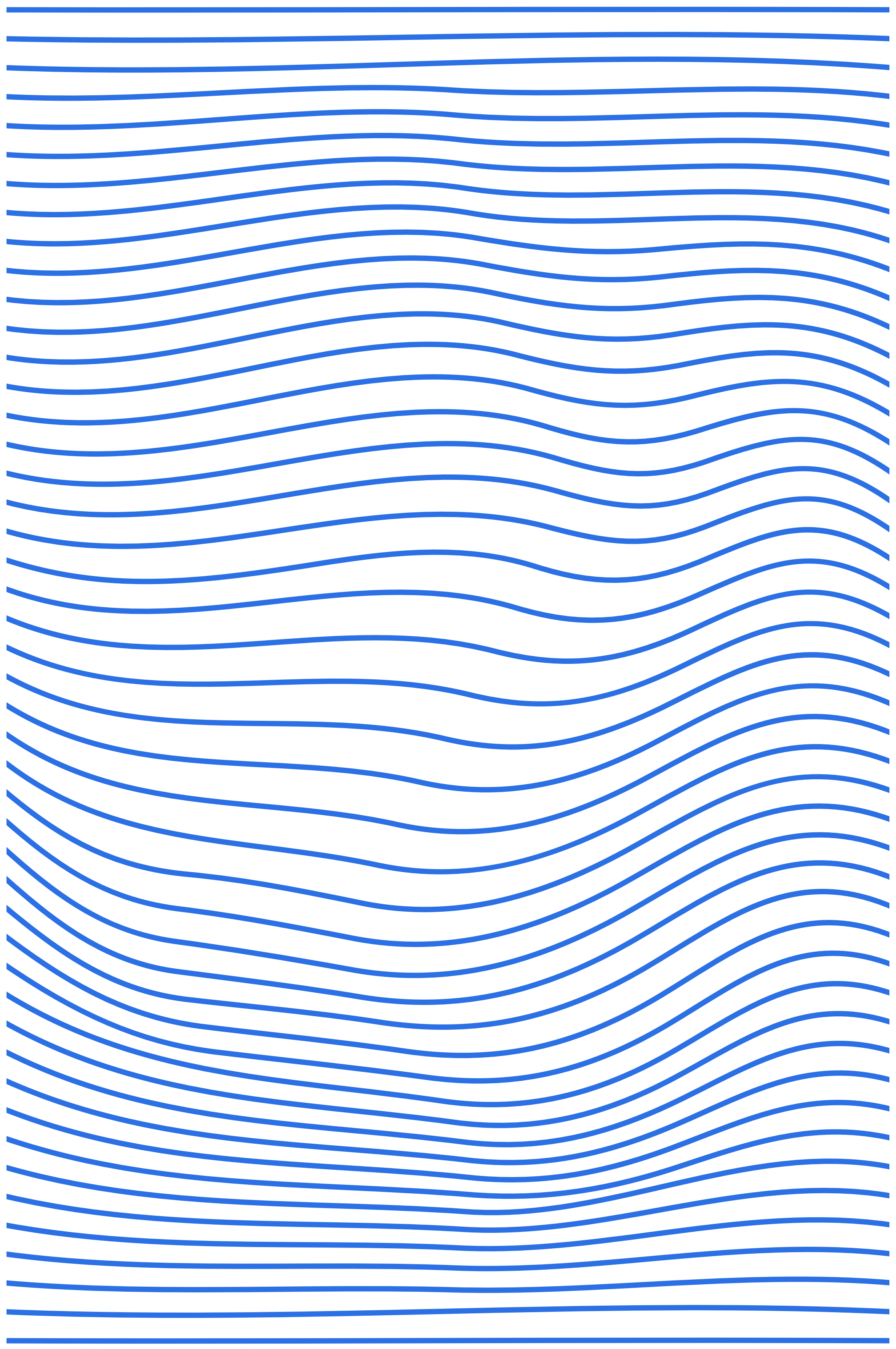


Contours detection

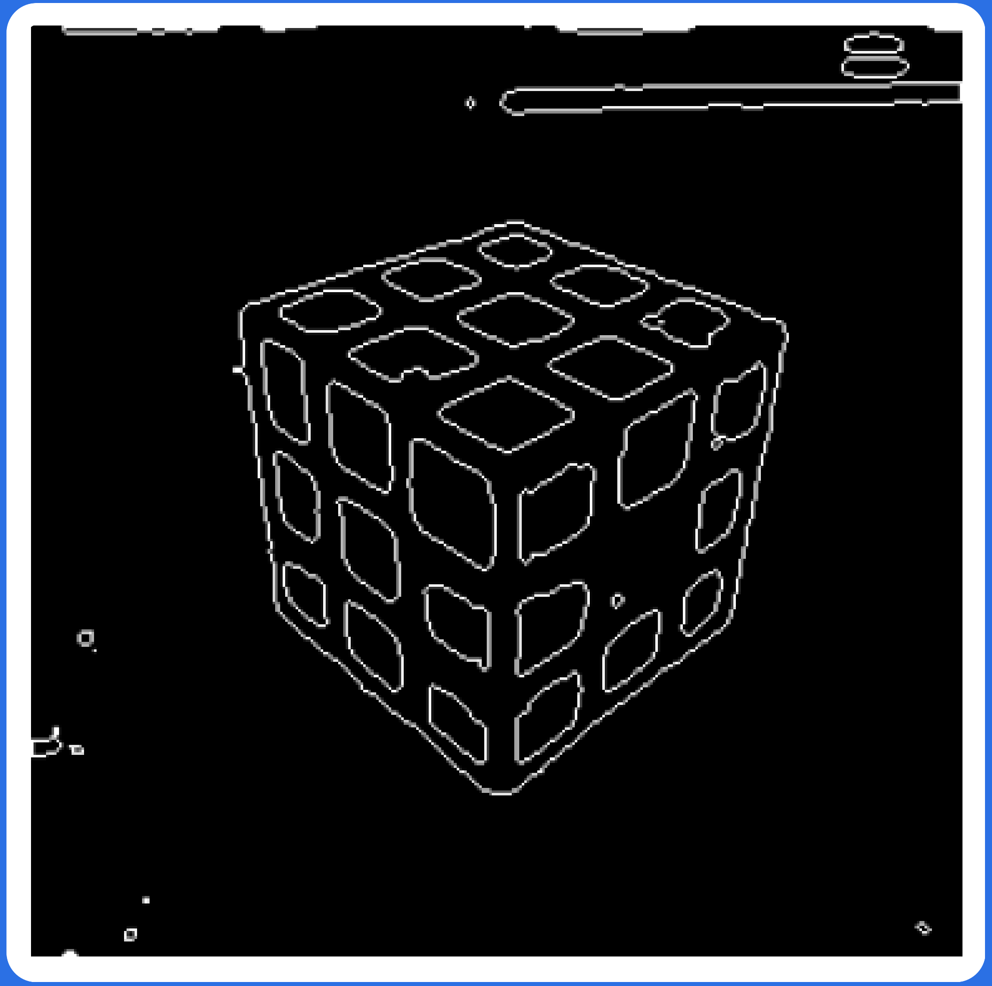
The process identifies the rough noise
But most importantly, identifies the objects
The output leaves to white only the detected borders



Separation from rough noise

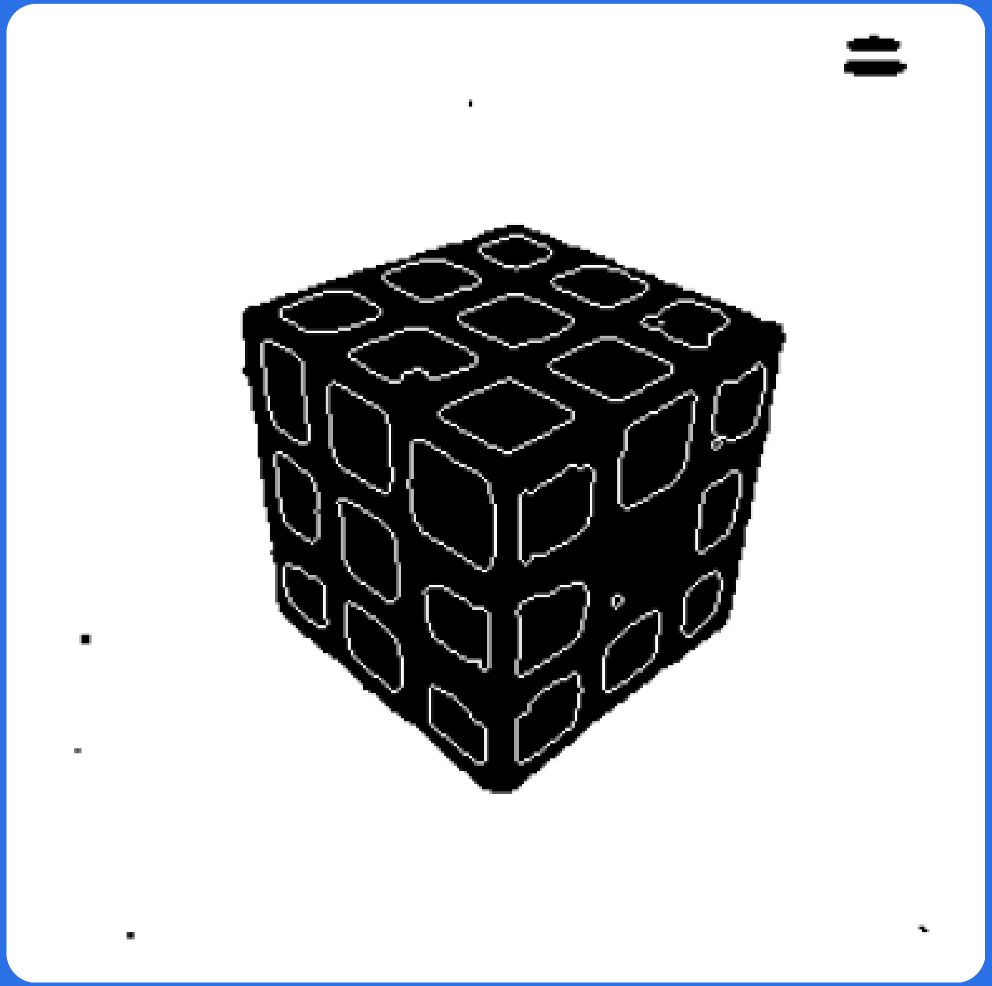


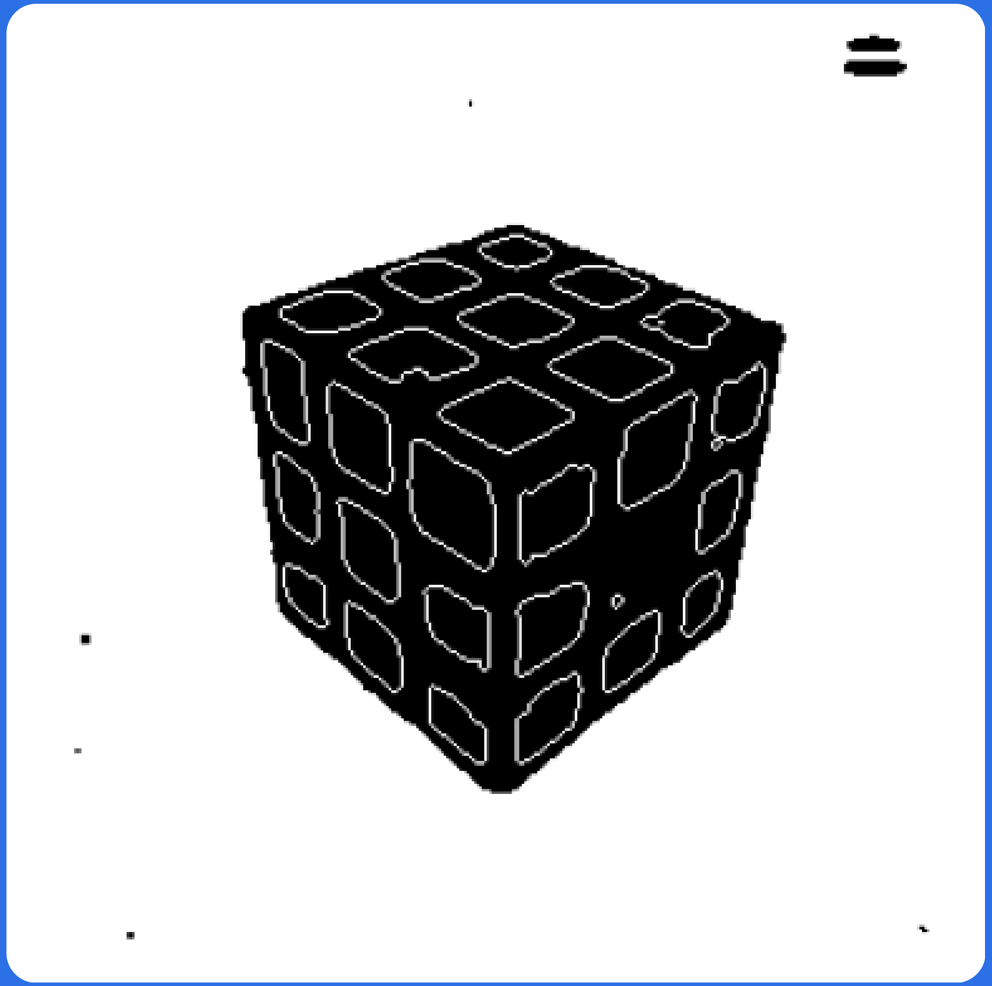
3/10



Filling with the color white

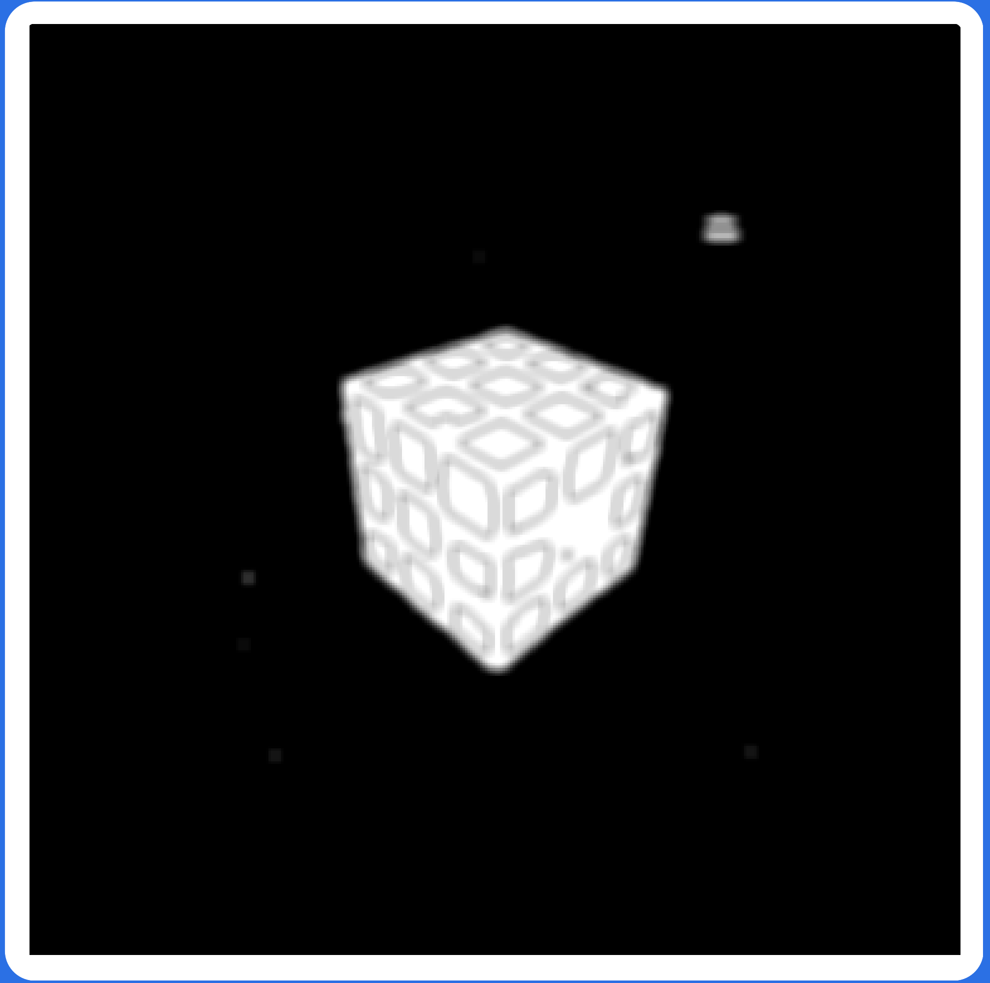
Process started from black lines at the edges of the image
The filling does not penetrate the contours
The remaining noise and objects are mostly hollow

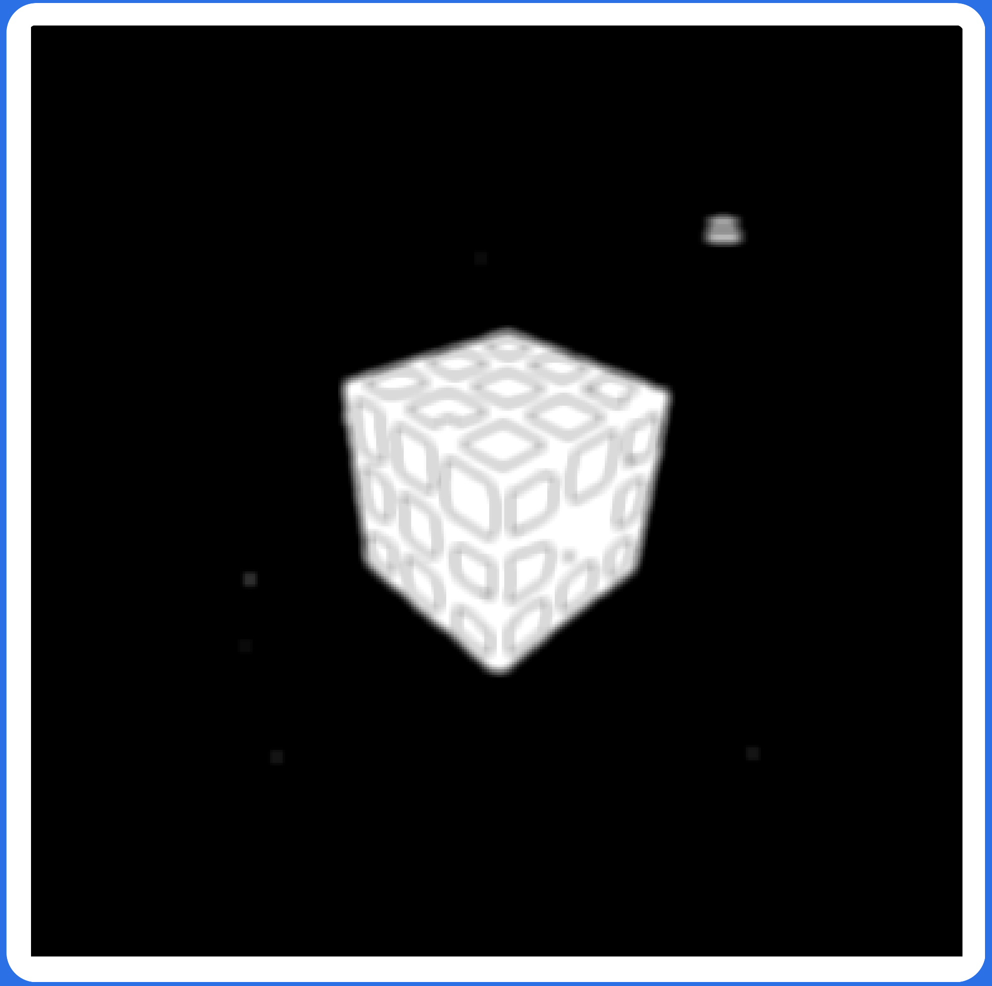




Smoothing on enlarged image

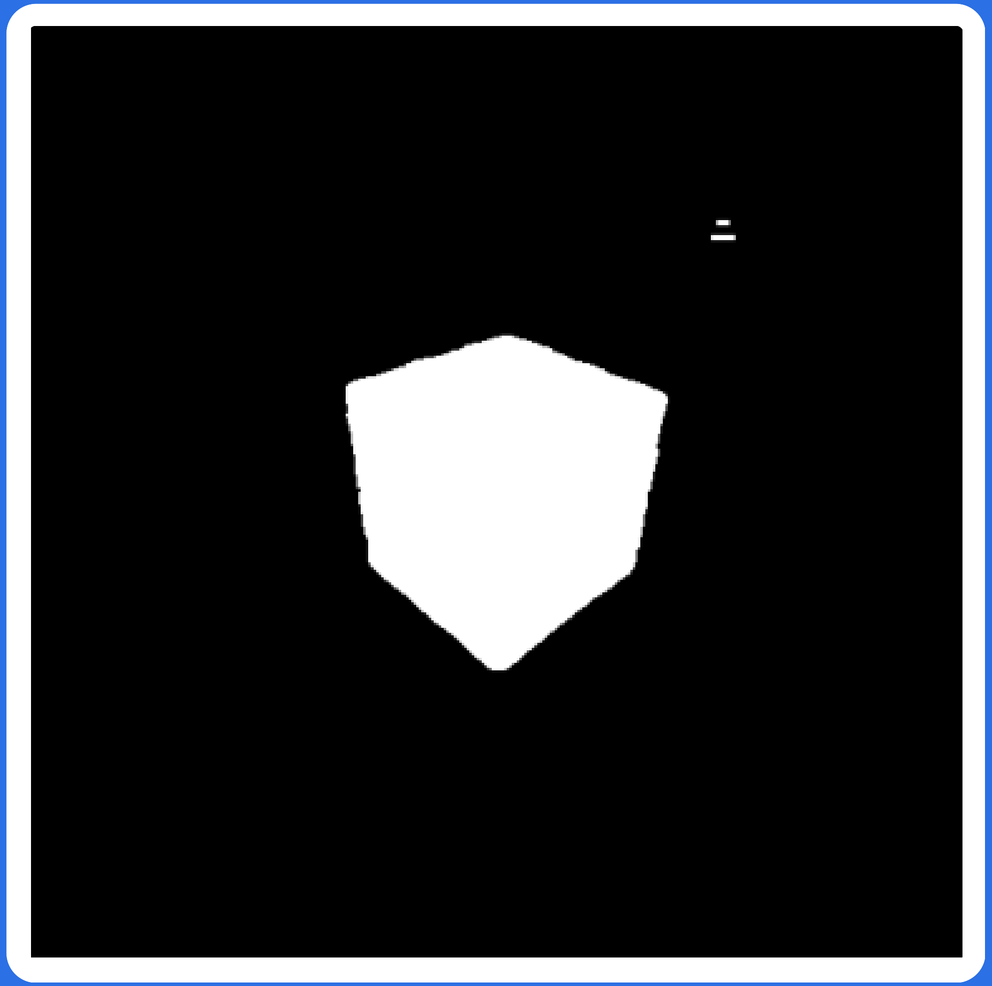
Resizing of the picture to 500x500 pixels
Applied a mean kernel to the output
The output is inverted, as white is considered foreground



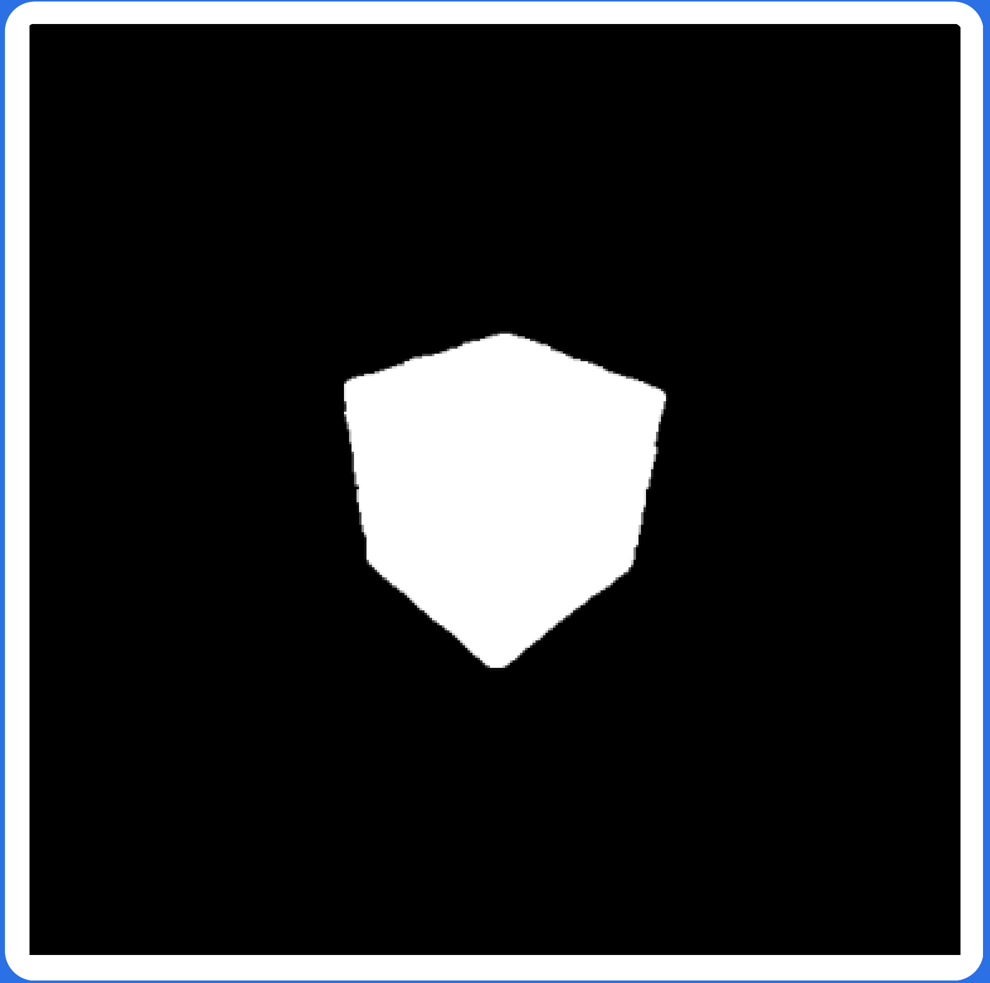


Binary thresholding

Essentially fills the internal area of the Rubik's cube
Removes the majority of the rough noise
In the output are only left the detected objects

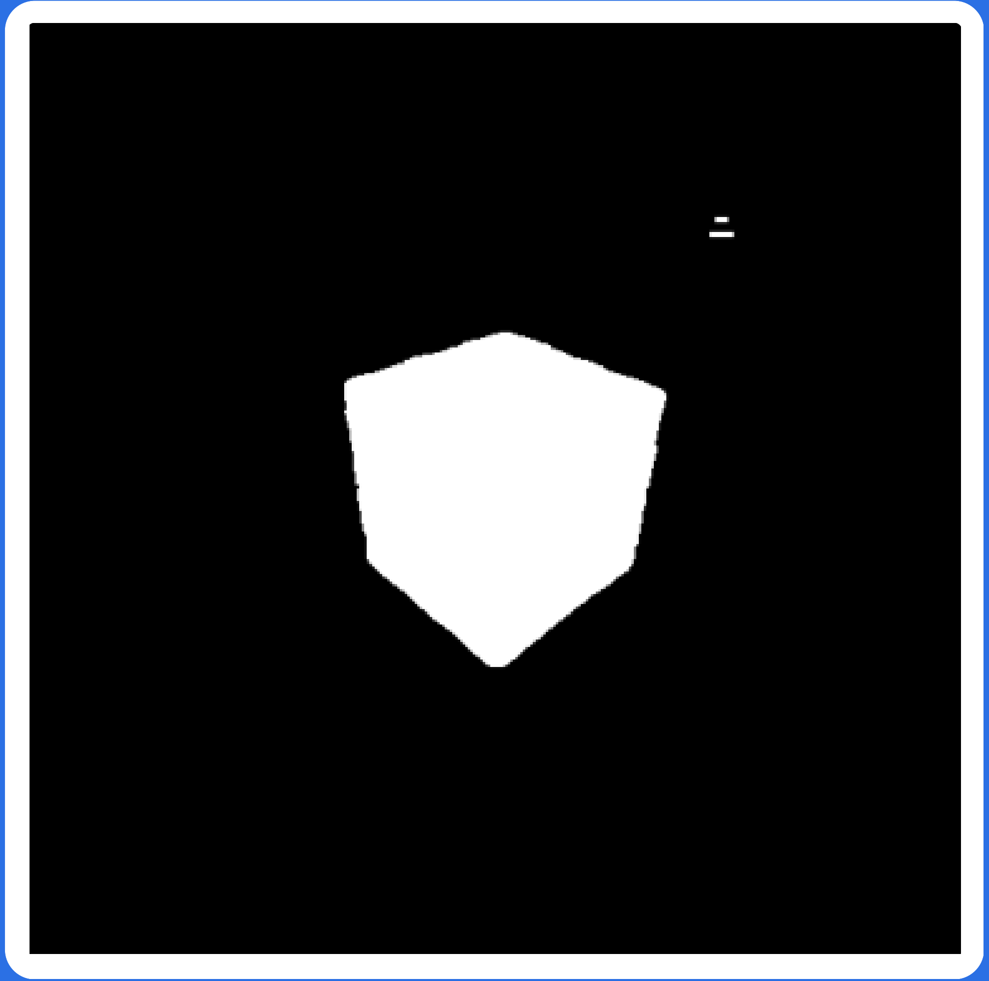


Rubik’s cube Region Of Interest



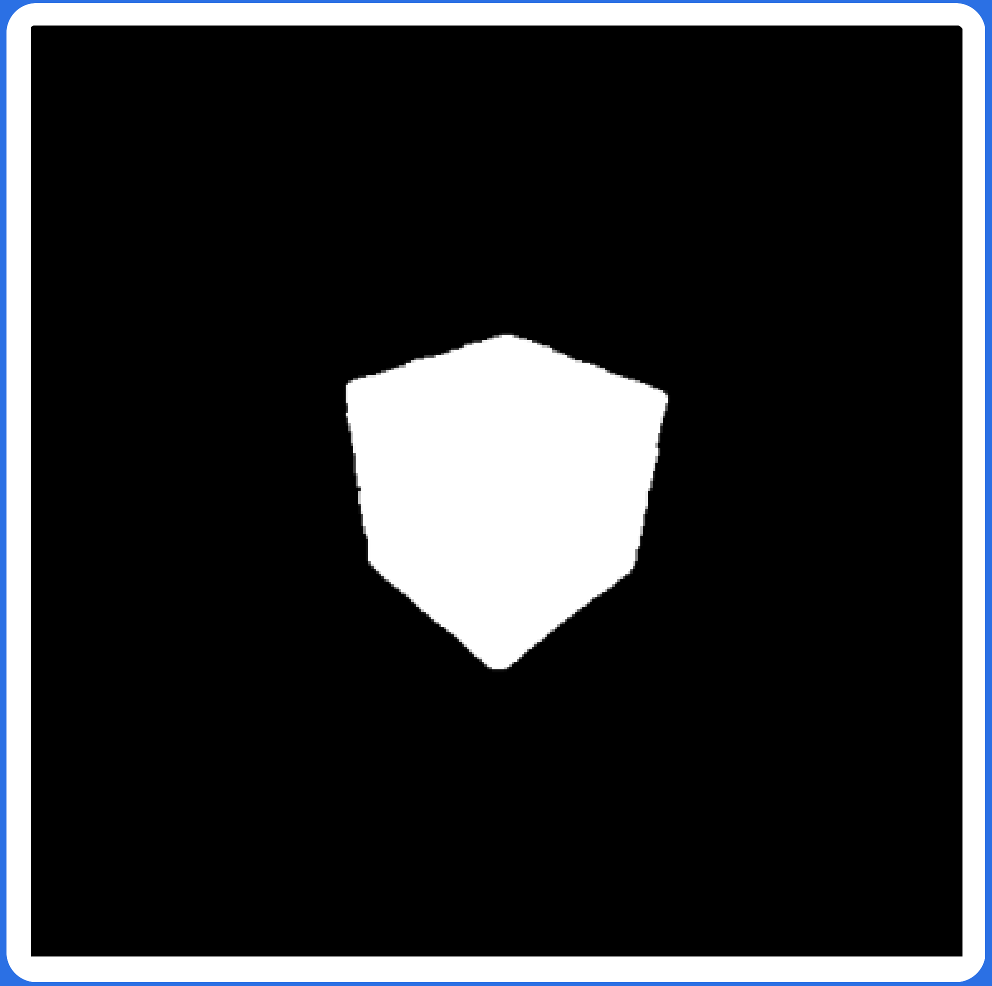
Isolating the Rubik’s cube

By considering the contours approximable to a circle
By then choosing the contour with the largest area
Result are then dilated to remove crevices



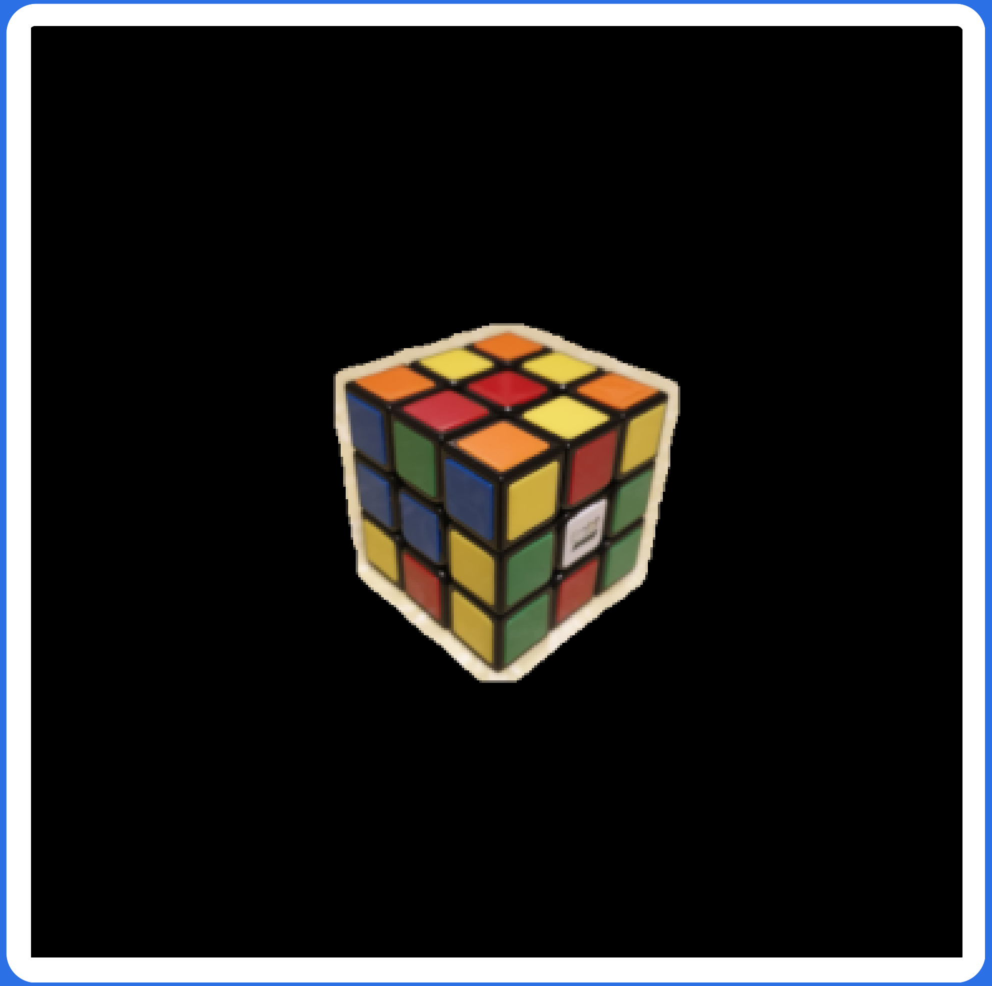
Rubik’s cube isolation from image

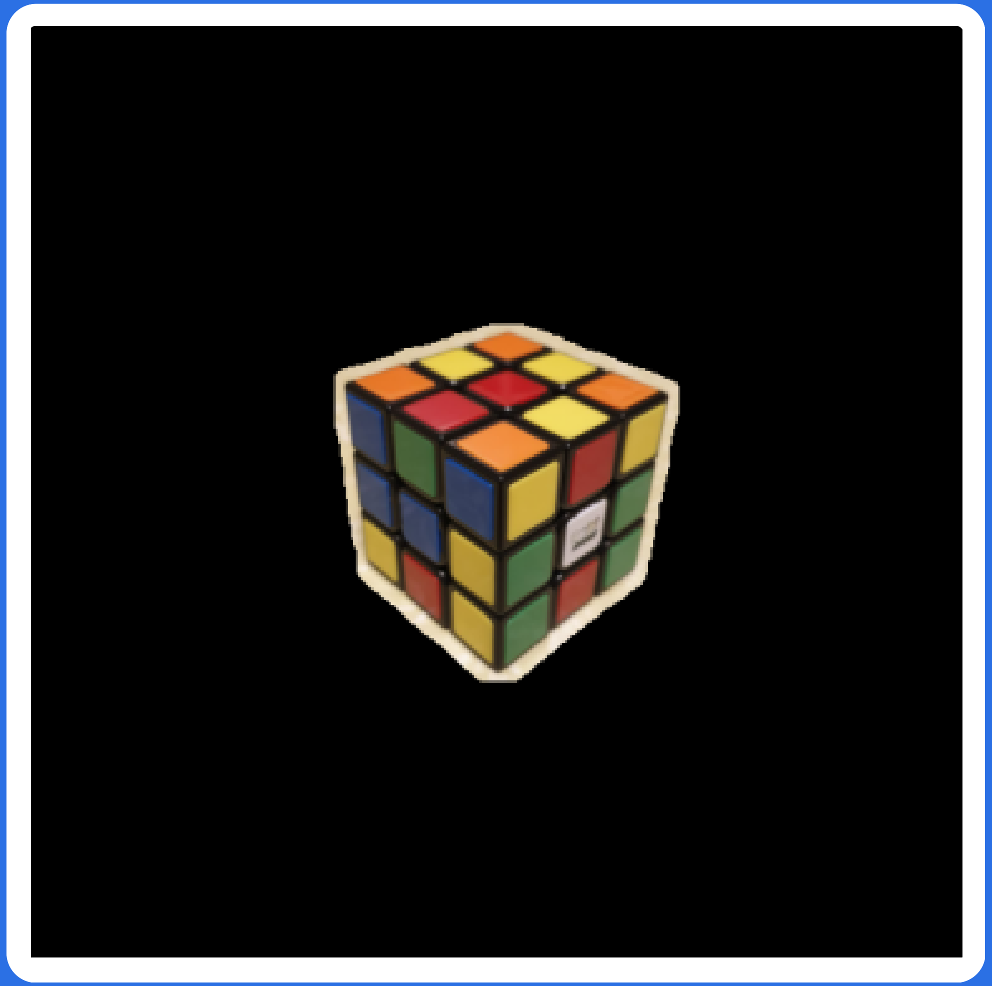
5/10



Projection on to starting image

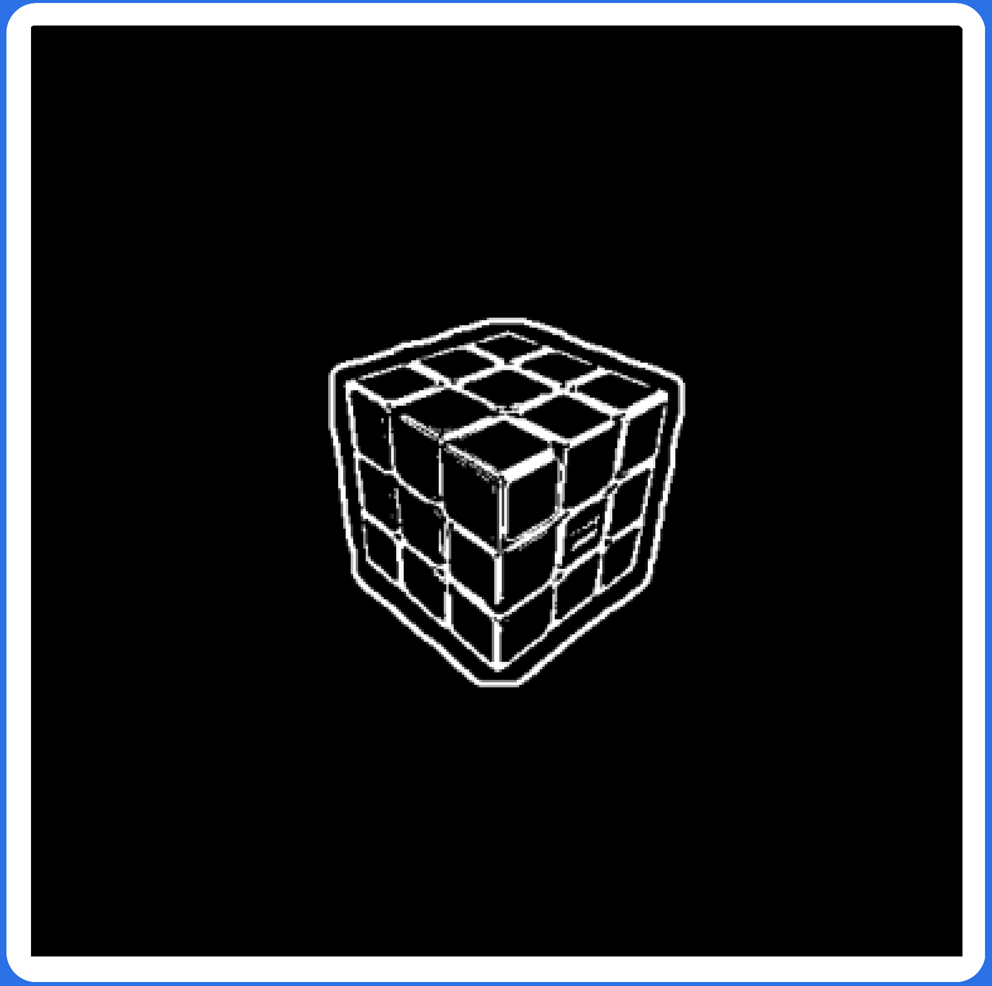
With the application of a bitwise operation
The white section is considered transparent
Meanwhile, the black section is considered opaque

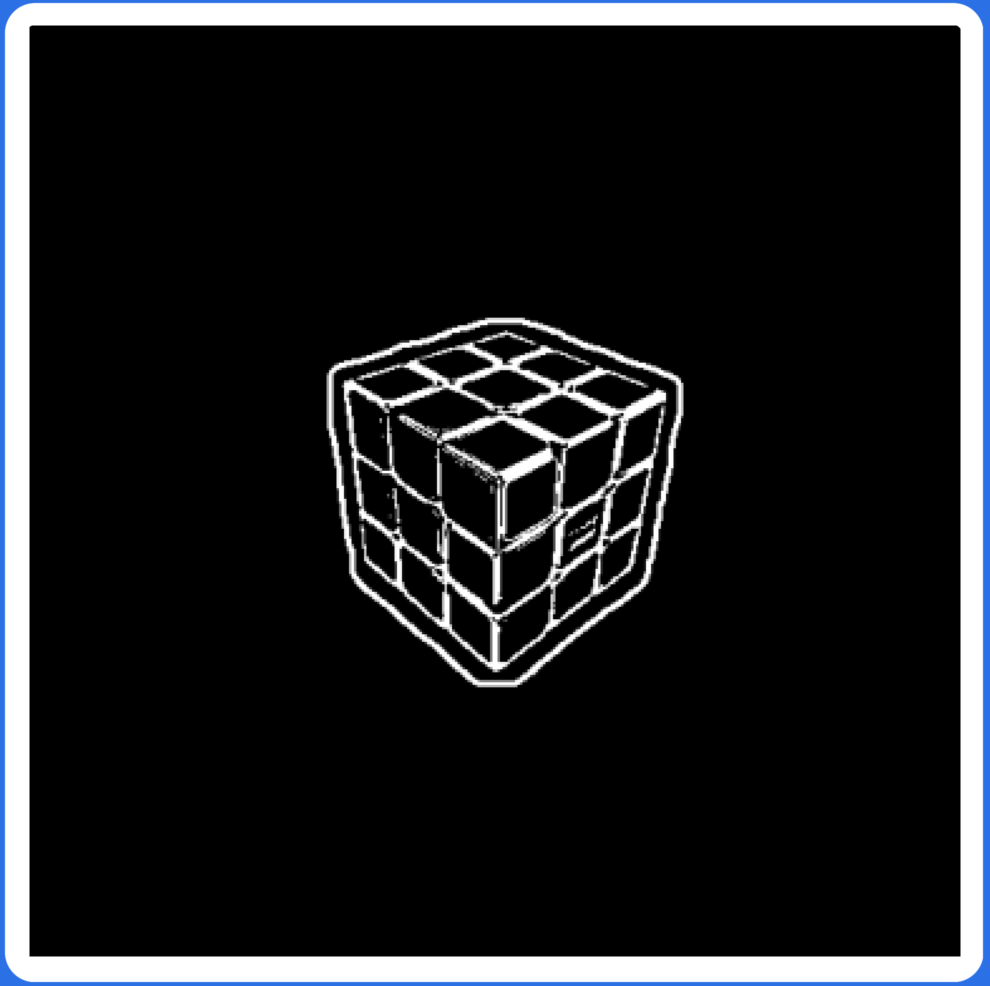




Adaptive thresholding

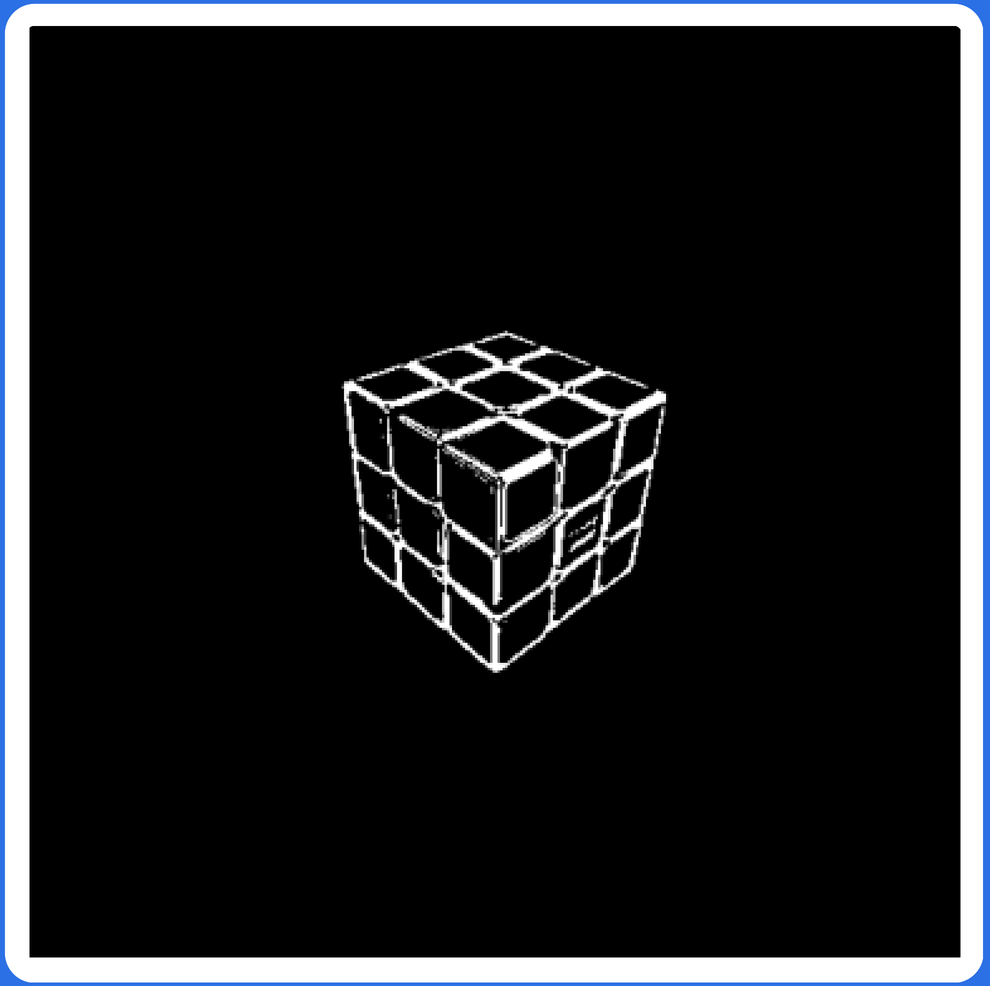
This step excels in distinguishing the cube from noise
Generates a high-quality representation of the cube
This is possible thanks to the small gap around the cube

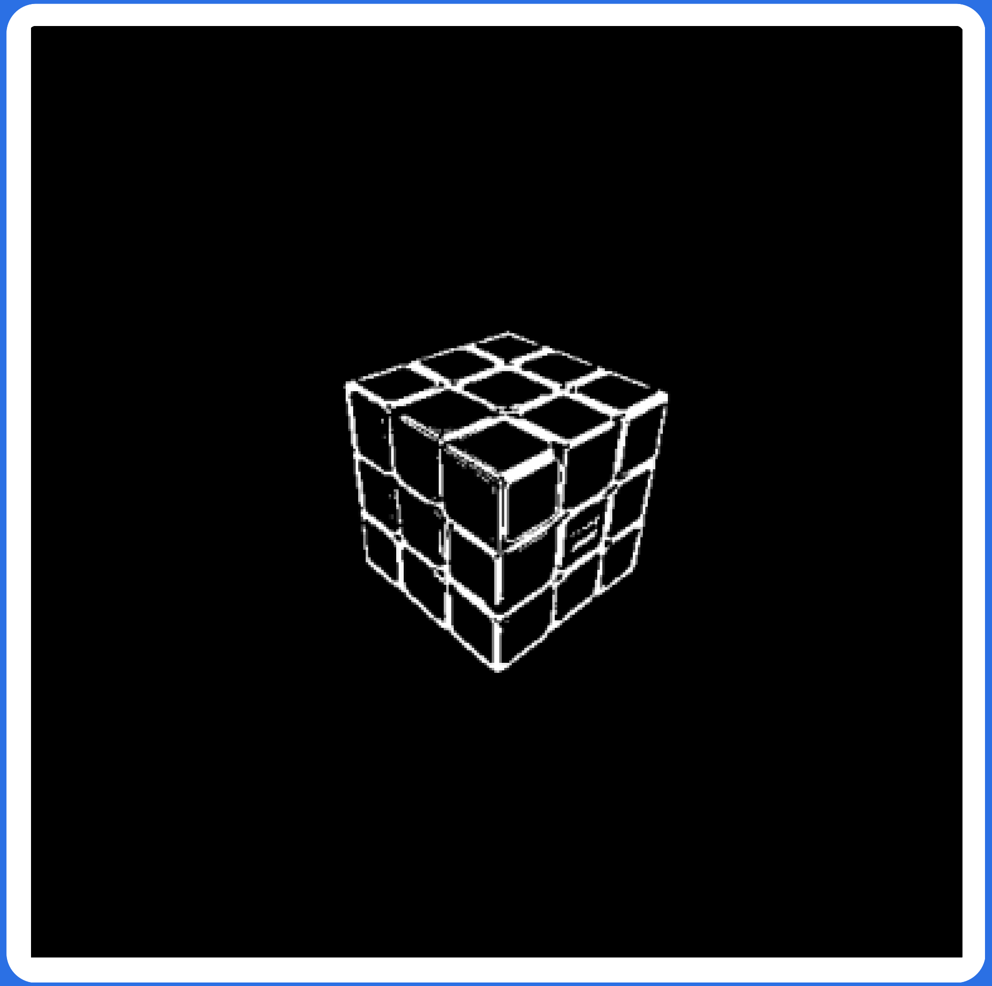




ROI border removal

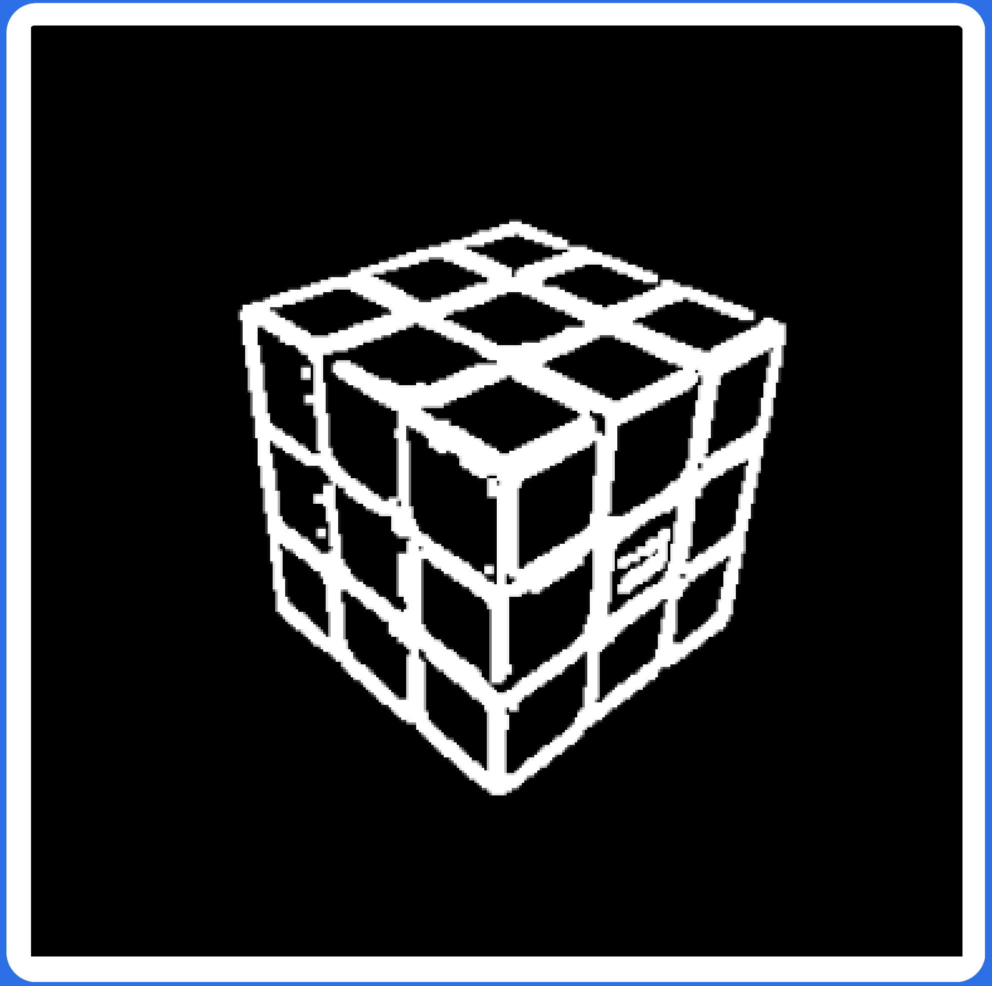
Firstly is selected the outermost contour
The black filling is started on a contour's point
In the image remains only the high-quality representation





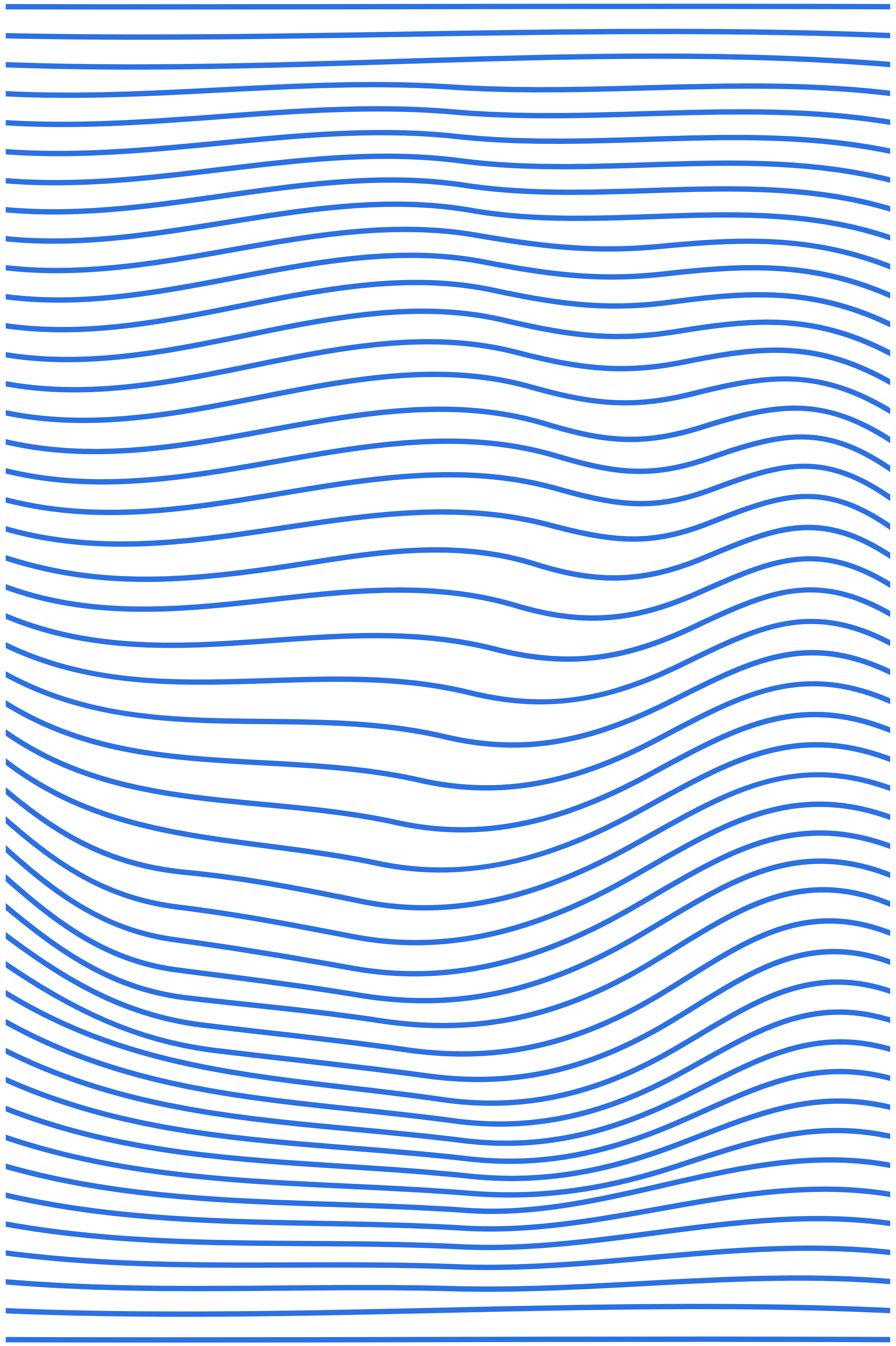
Unification and cropping

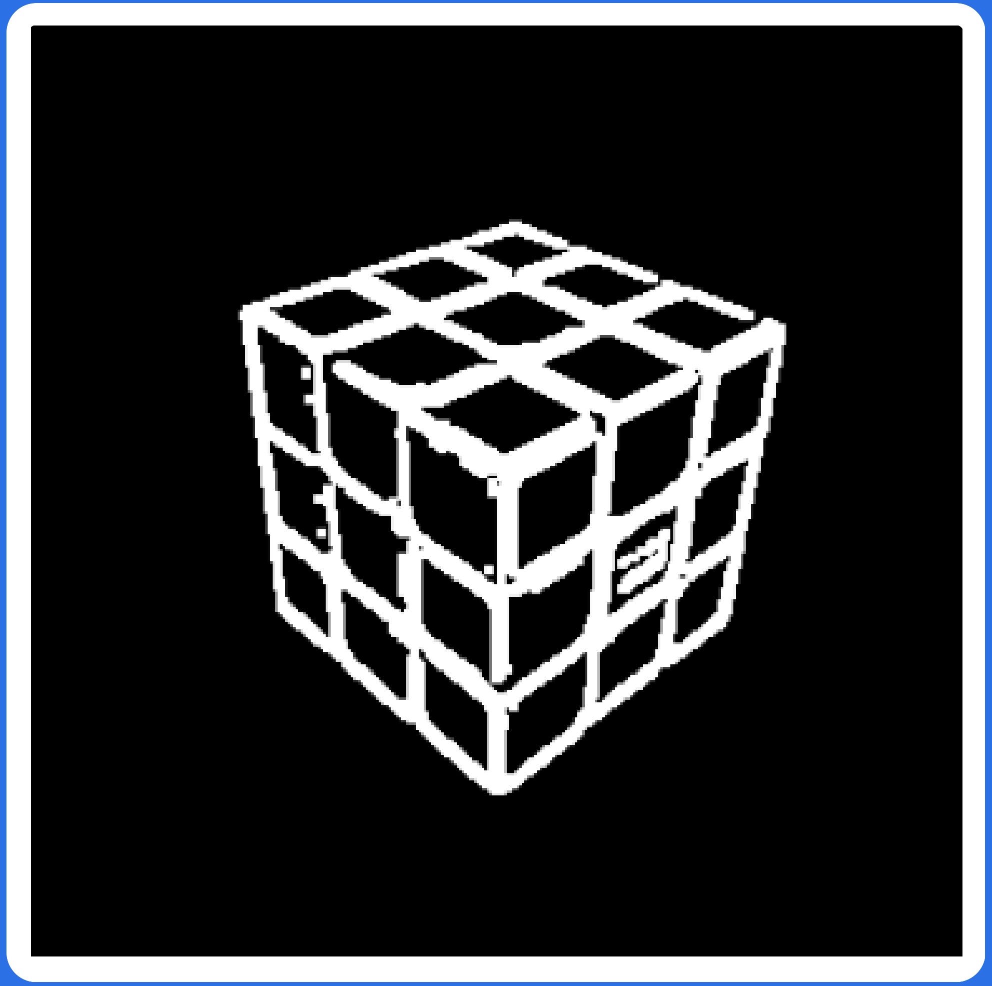
On the image is applied a dilatation to unify the object
The output is cropped to the riginal 300x300 resolution



Rubik’s cube edges retrieval

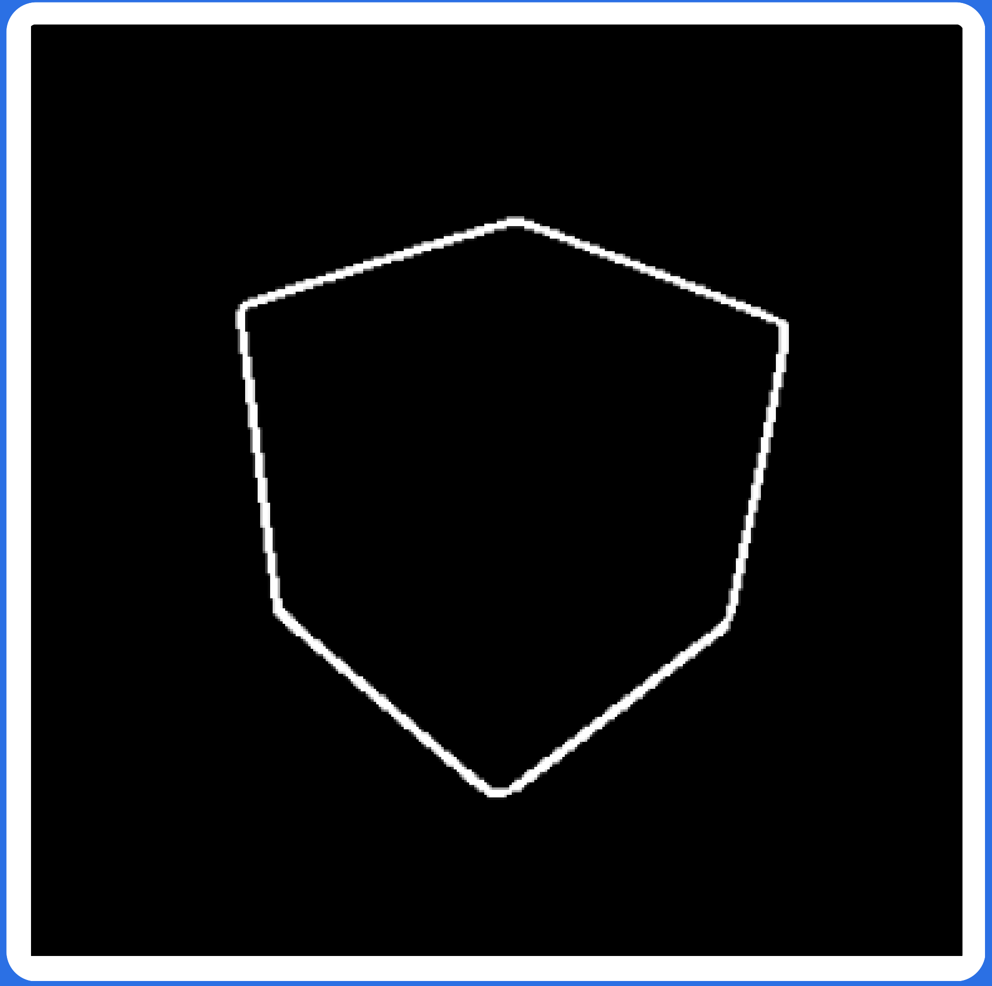
6/10

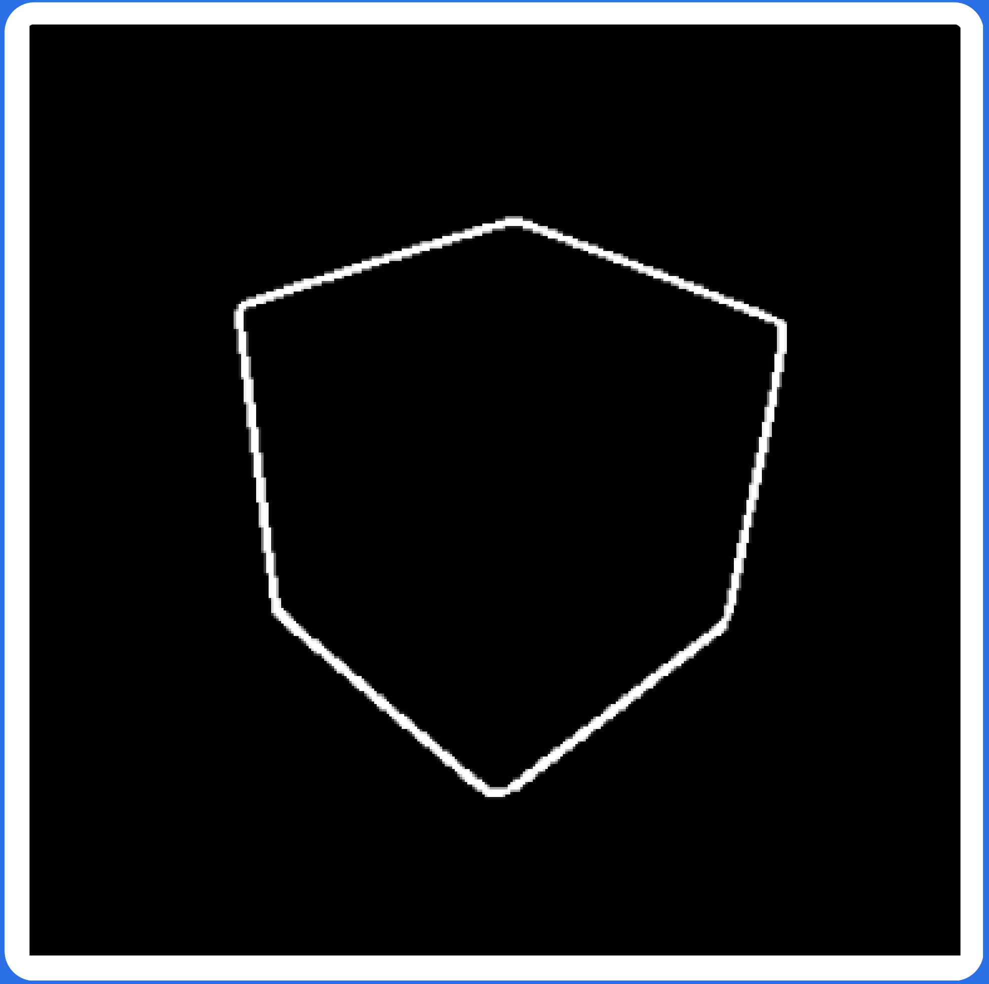




Convex hull creation

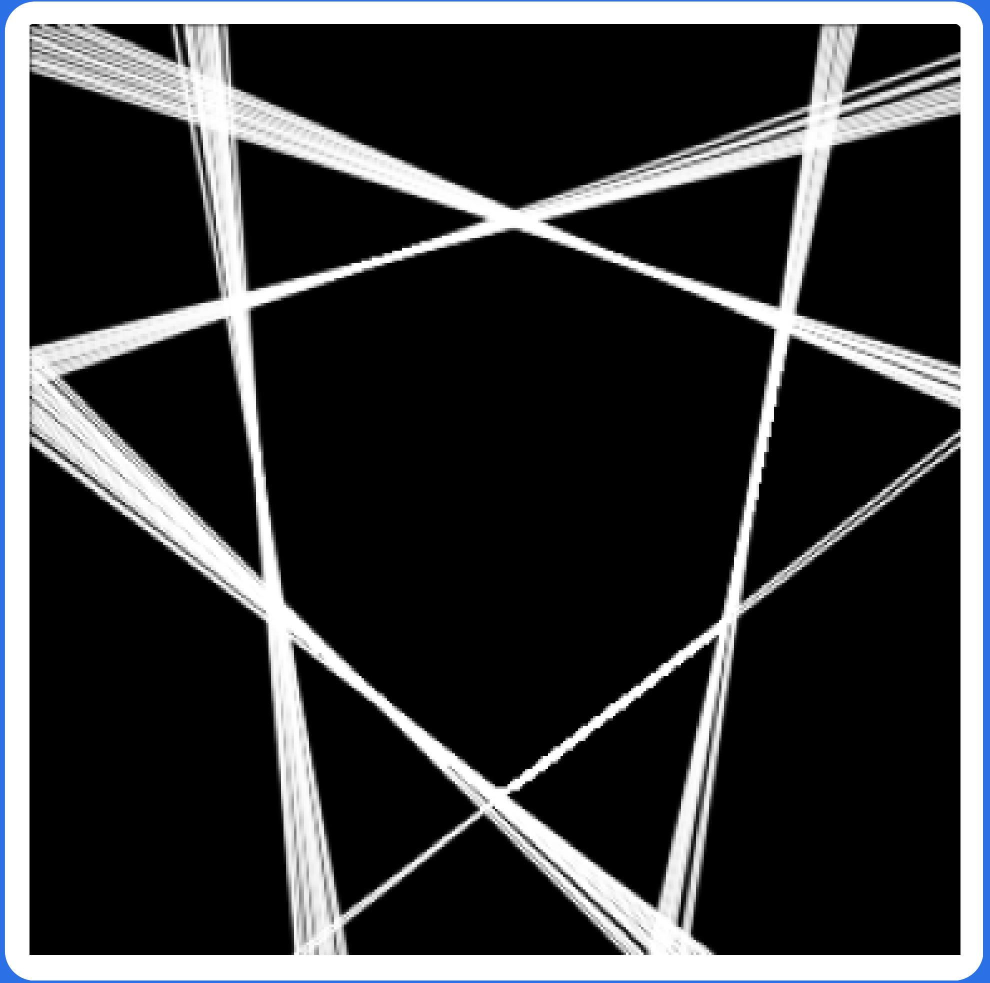
This is applied to the contour with the highest area
The convexity ignores gaps on the cube's borders
Everything else is discarded



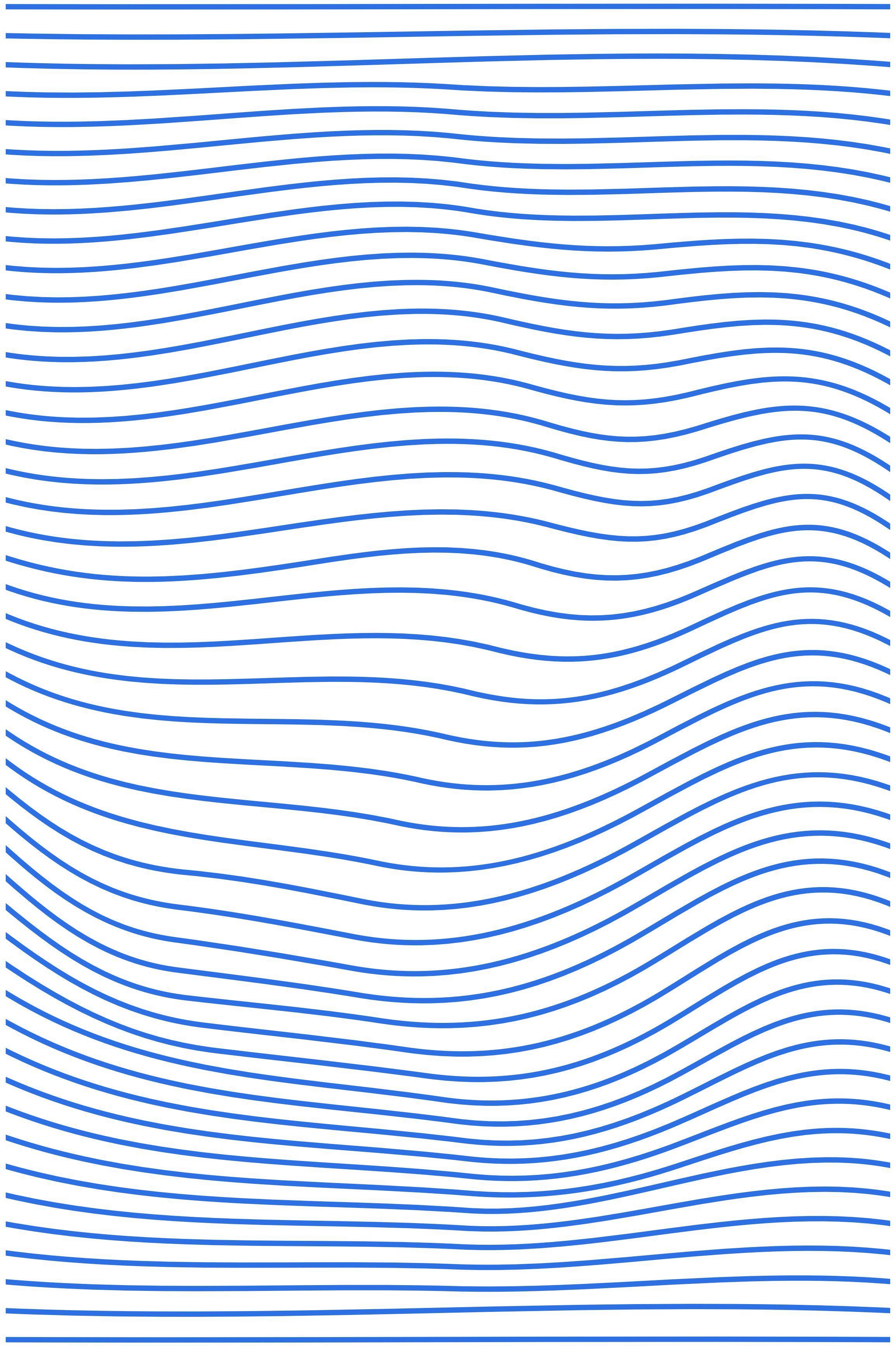


Assigning lines to edges

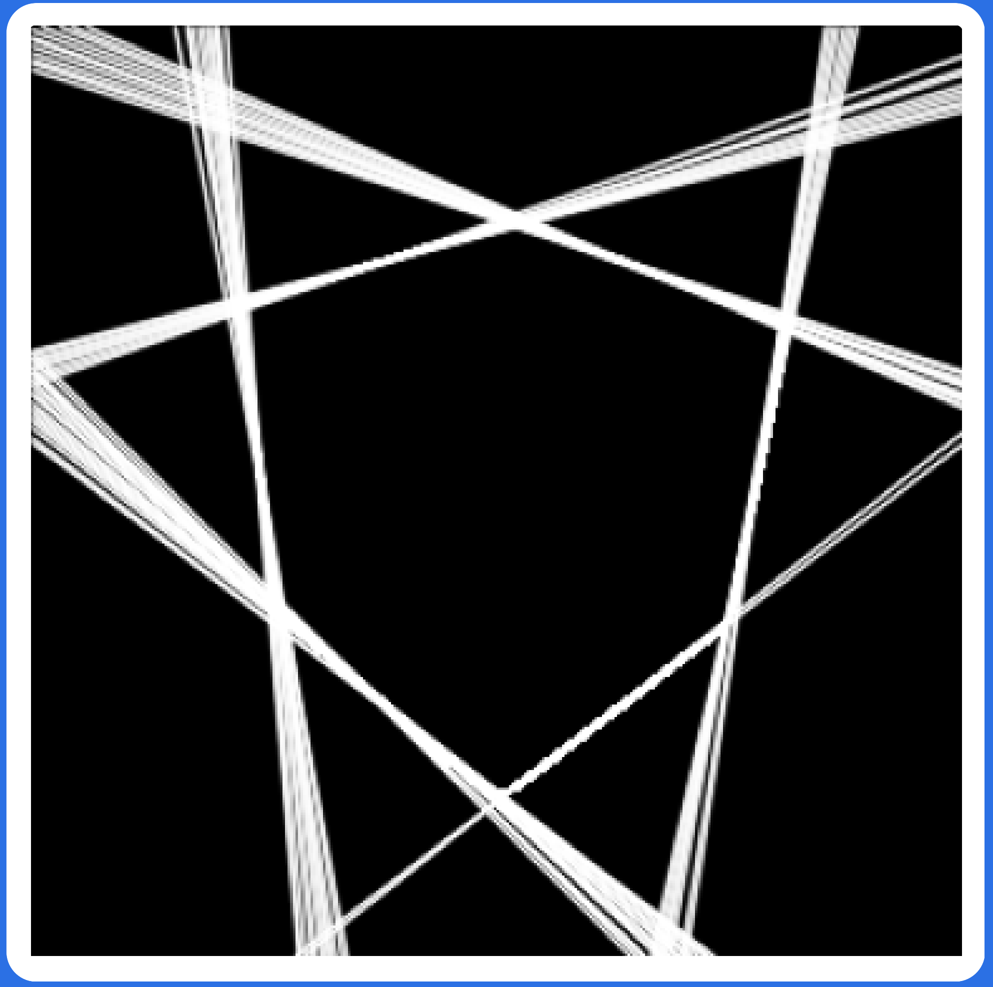
Each edge is assigned to least a new line
Overlaps are present to avoid an edge not being assigned
Useful when are present round edges on the cube



Rubik’s cube corner retrieval

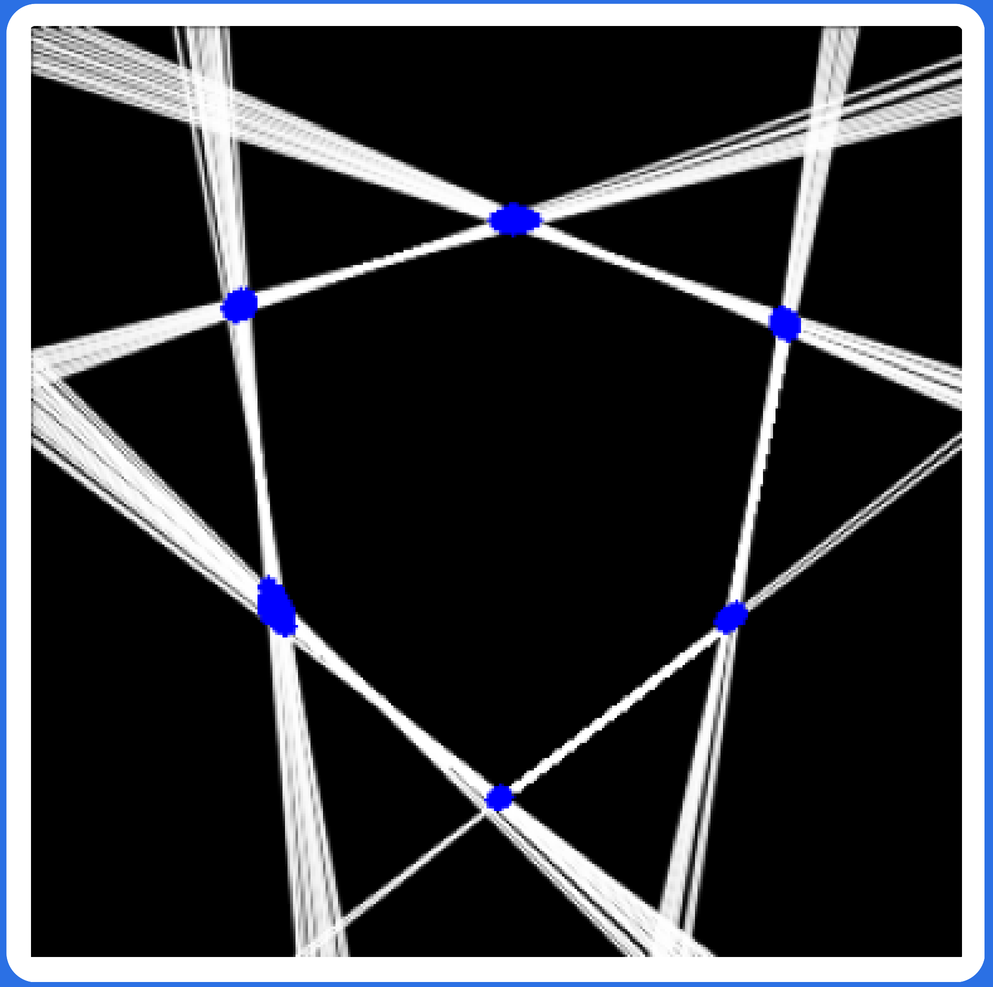


7/10



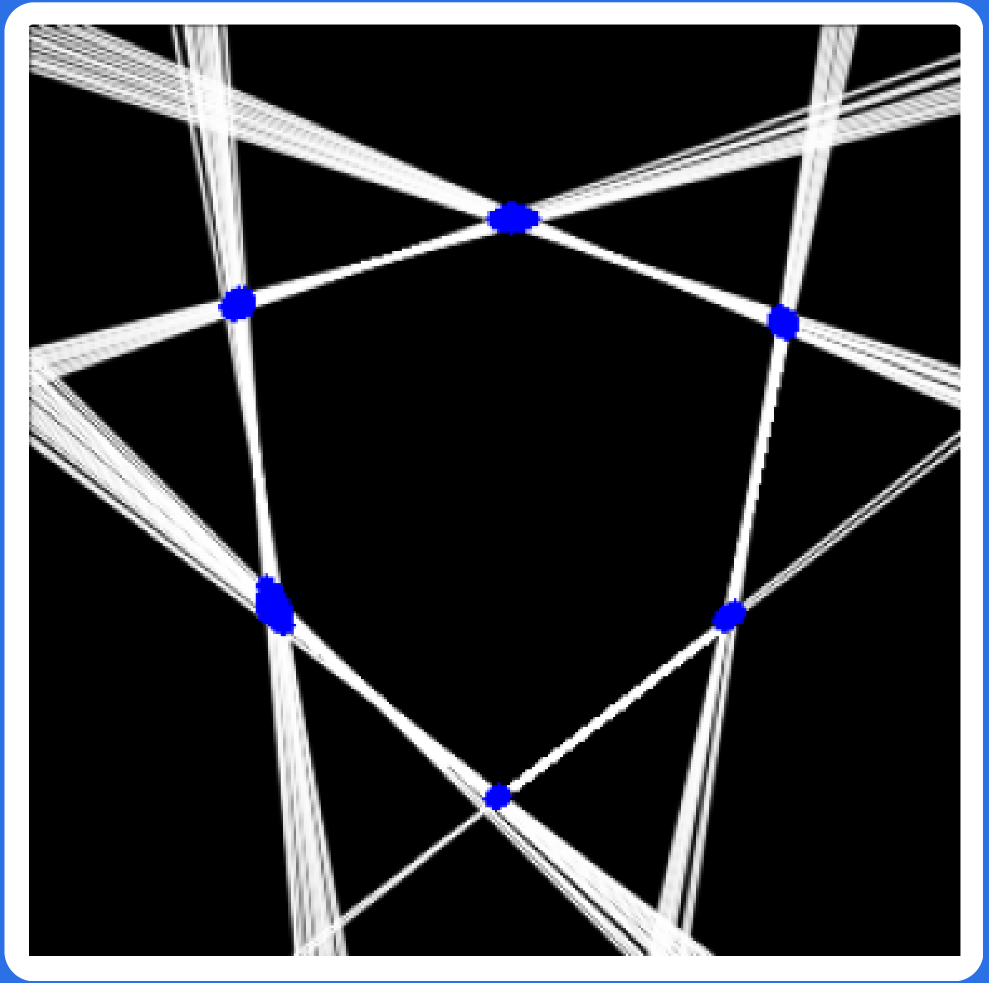
Generation of the corners

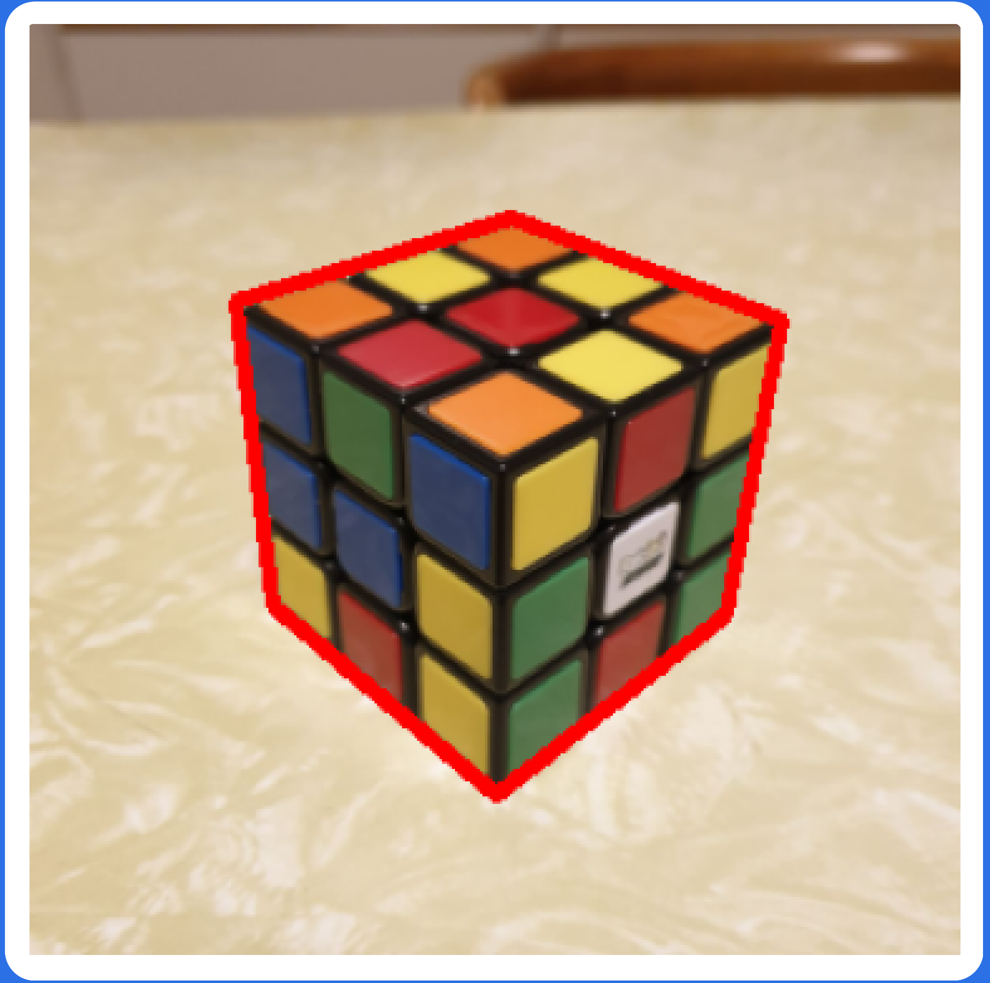
Firstly, by a filtering by angle and distance of intersections
Secondly, by a clustering on the group of remaining points
Thirdly, by considering the centroids as the cube's corners



Sorting the corners

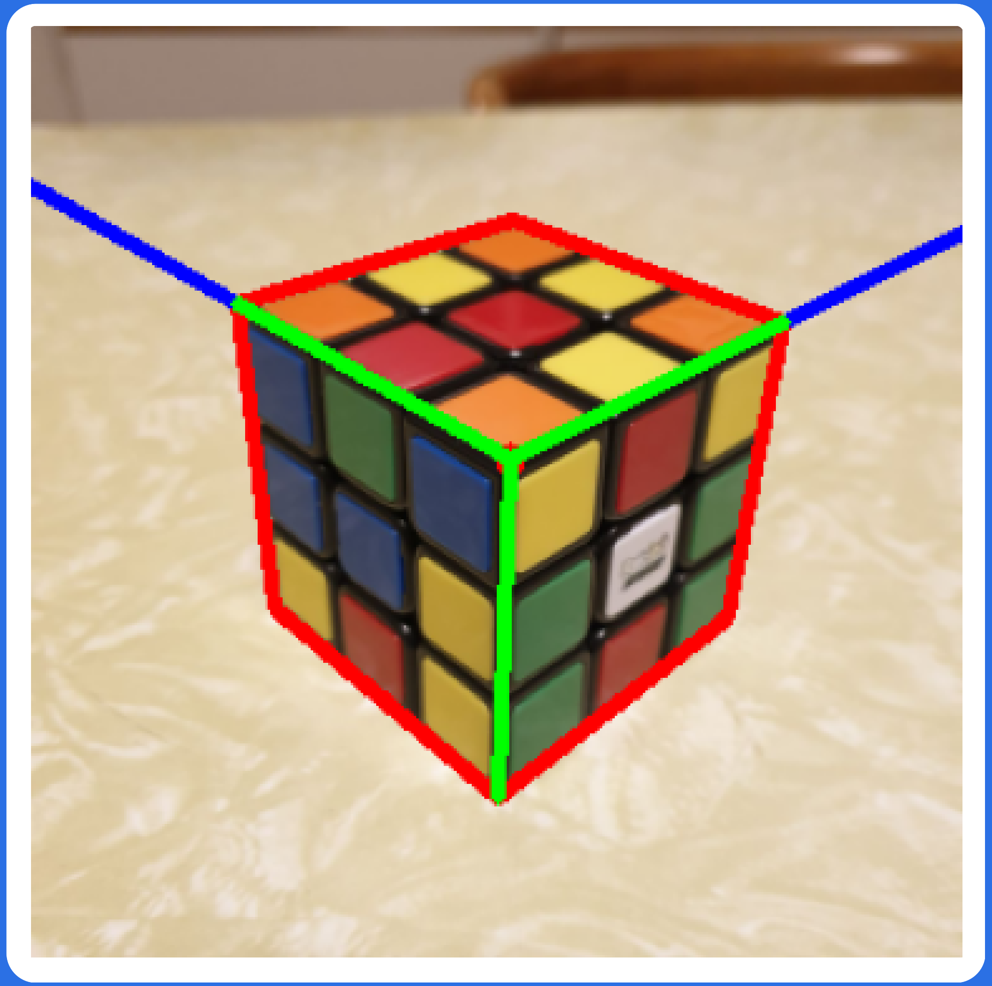
First they are sorted from top to bottom
Then they are sorted from left to right
As a result, an accurate hexagon can be generated





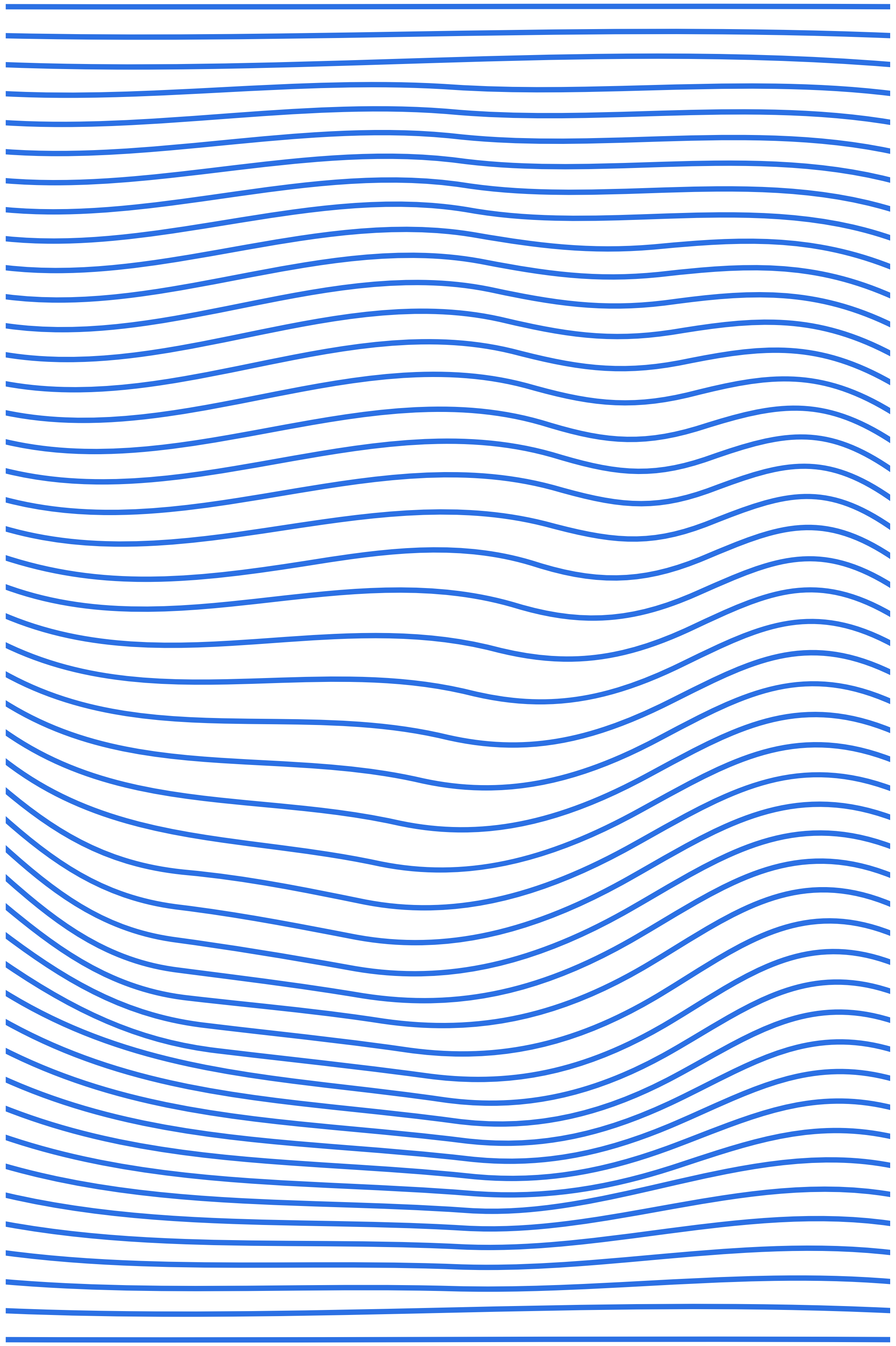
Calculating the central point

This is possible through a geometric perspective operation
Automatically determines the perspective height
Also determines the lateral shift inclination



Face projection and color detection

8/10

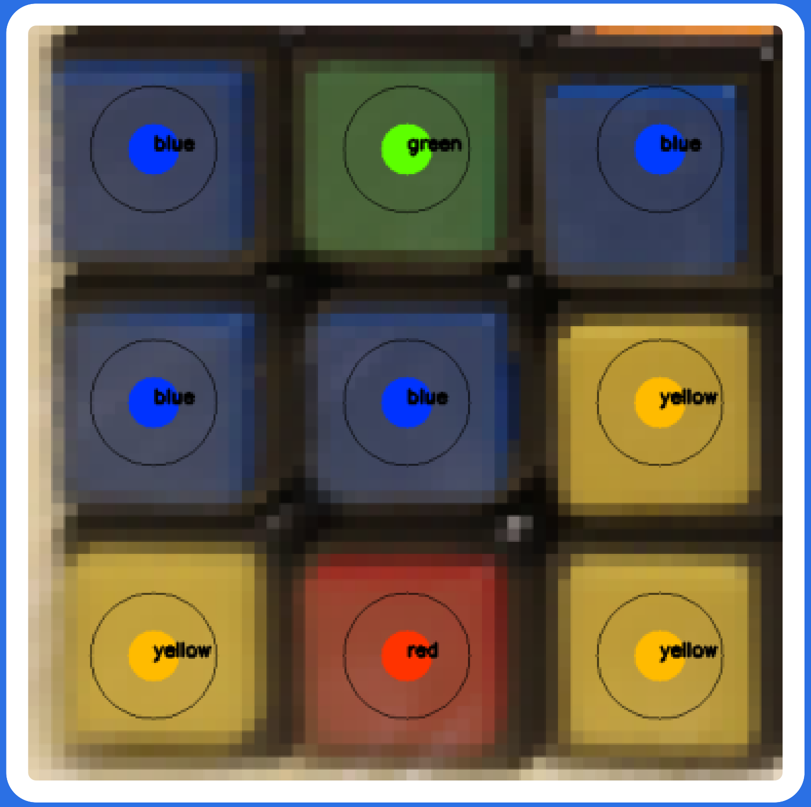


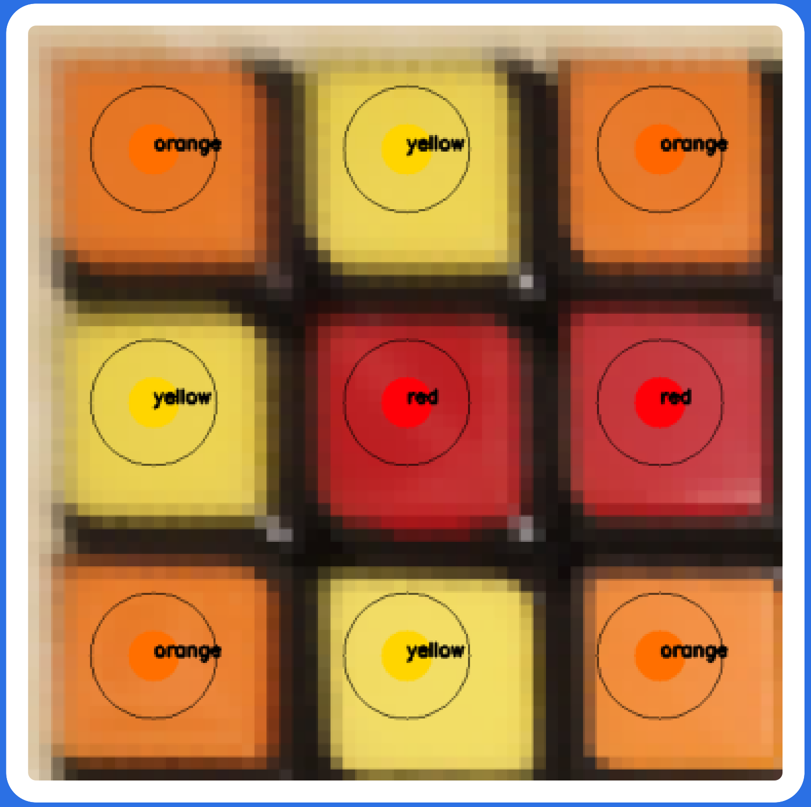
Separating the three faces

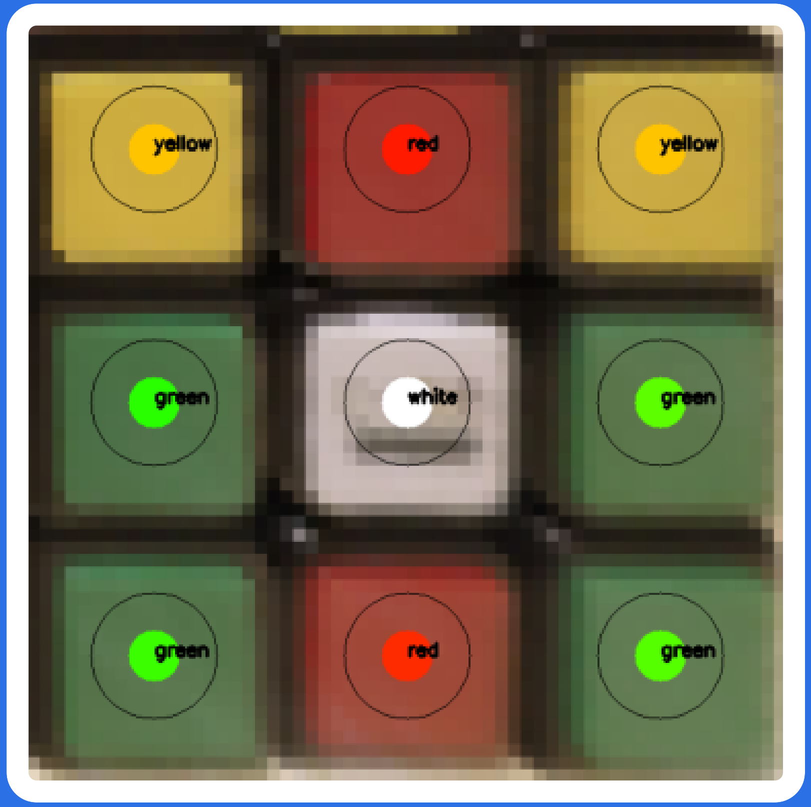
The three faces are each projected in to a new flat image
Then they are converted in HSV format for a better sampling
Finally, the detection of hue is done via a circular ROI

Displaying the results

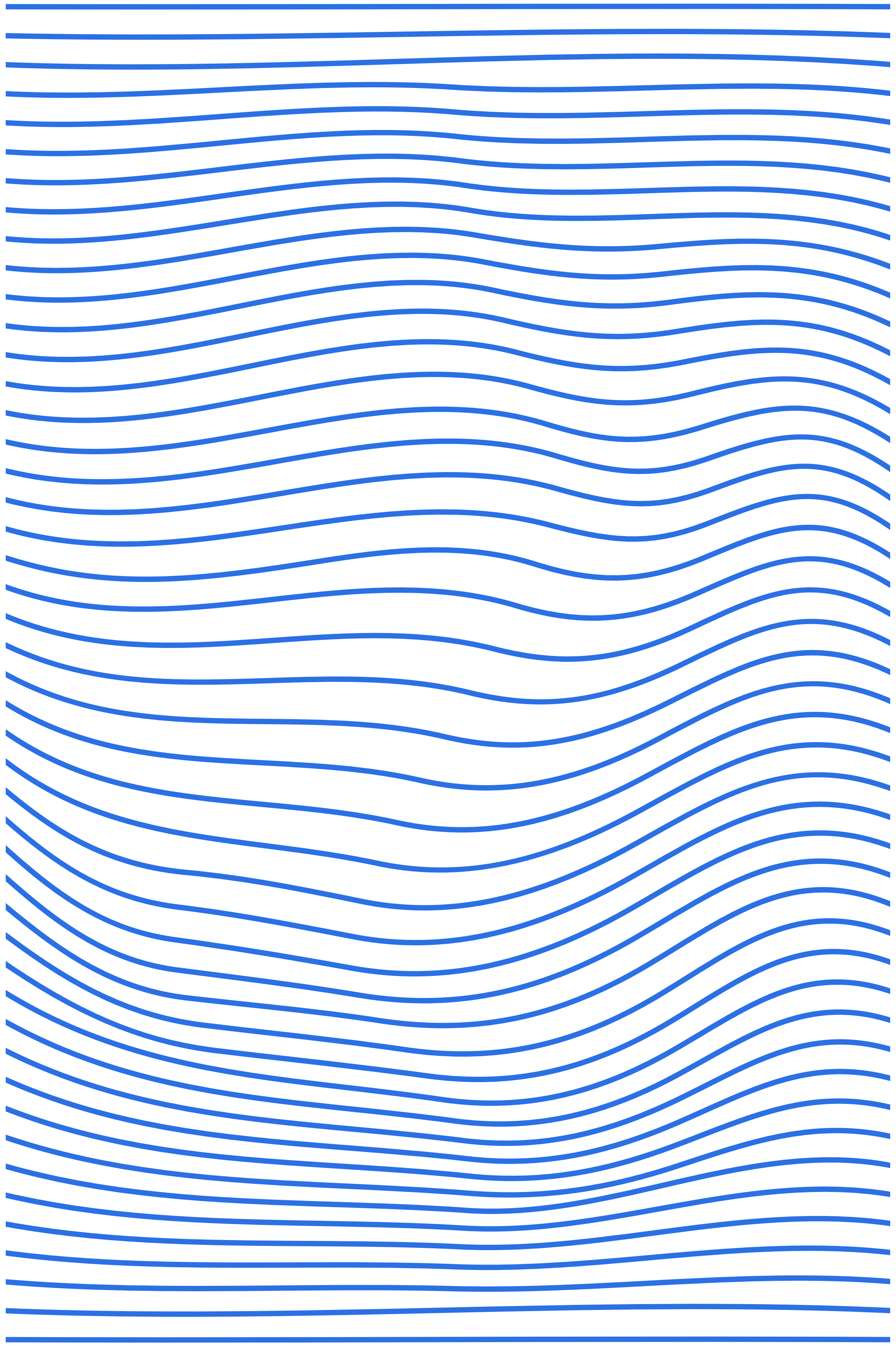
The three faces have their size increased
Then is displayed the utilized ROIs during detection
Also displayed the values and labels of the detected colors







Positive aspects VS limitations and challenges



Positive aspects

1. Good separation from clutter
   Due to the contour filtering by area and roundness
2. Good separation from noise
   Due to the thresholding and ROI combination
3. Implementation technique
   Due to this approach mostly using standard CV2 functions
4. Adaptability to different cube designs
   Due to not using a Neural Network to re-train
5. Precision in color detection
   Due to working in the HVS domain instead of the BGR
6. Precision in central corner detection
   Due to utilizing a geometrical property

Limitations and Challenges

1. Edge detection failures
   Due to the reliance on the filling method

2. Perspective issues
   Due to the corner ordering

3. Corner issues
   Due to the HoughLines detection approach

4. Background color
   Due to relying on the cube’s black border

THANKS FOR YOUR TIME