Computer Vision Project Documentation

December 2, 2025

1 Board Extraction

For board extraction I converted the image to grayscale, applied different filters to remove noise, applied different thresholds, applied two erodes for clean-up, used canny for edge detection and then extracted the contour with the highest area. For canny I also used softer thresholds to create connections more easily. Here is the full pipeline.

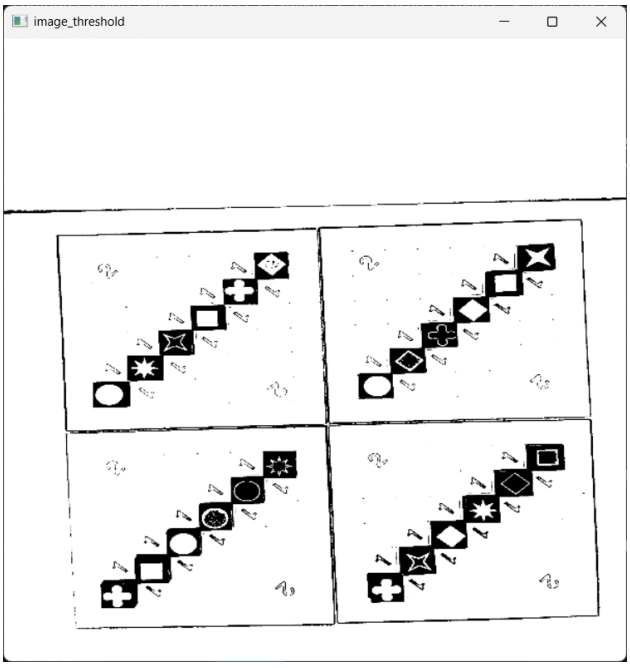

Figure 1: Filters + Threshold

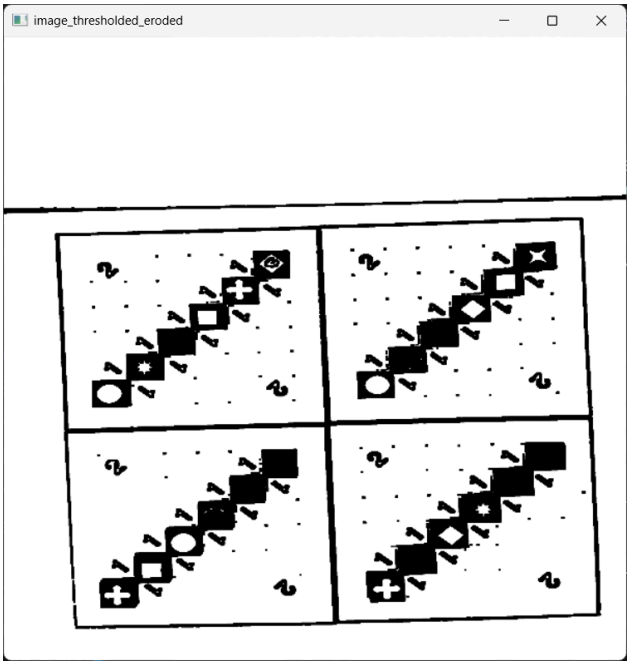

Figure 2: 2 erodes with 7x7 kernel

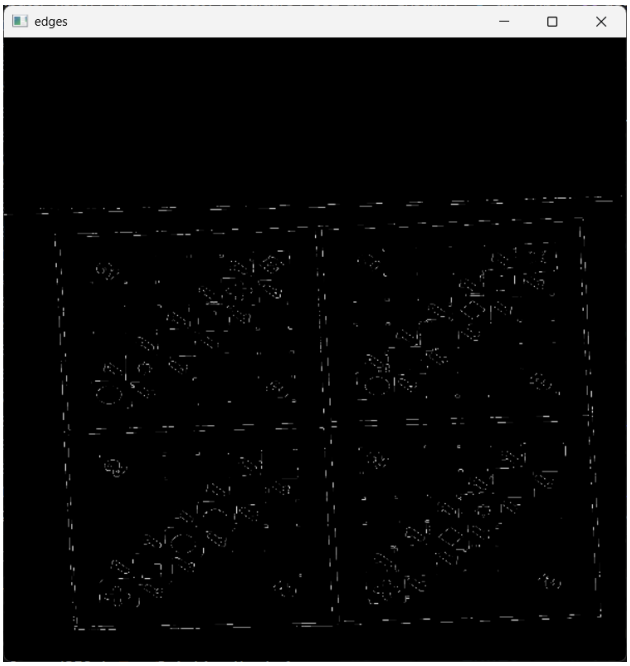

Figure 3: Normal canny

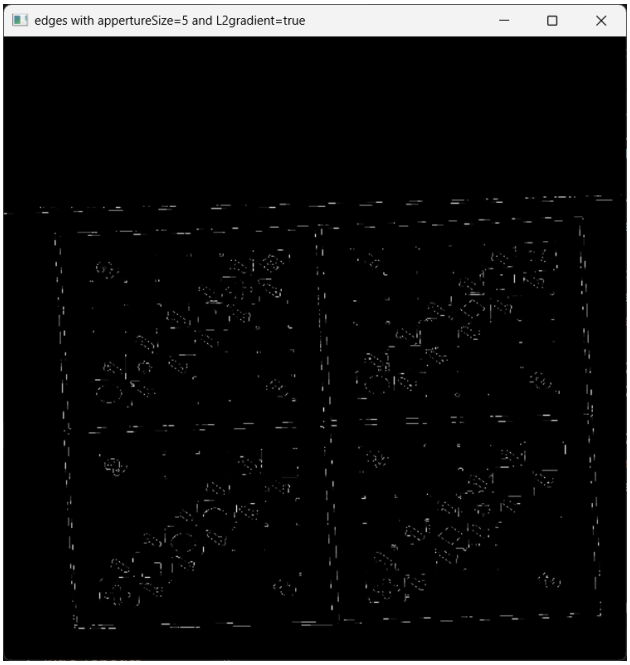

Figure 4: Canny with fancy parameters

The biggest improvements came from the erodes as they helped connect the board lines better and from Canny's optional parameters, aperture_size, L2Gradient for higher precision.

Here is a table extracted without the parameters:

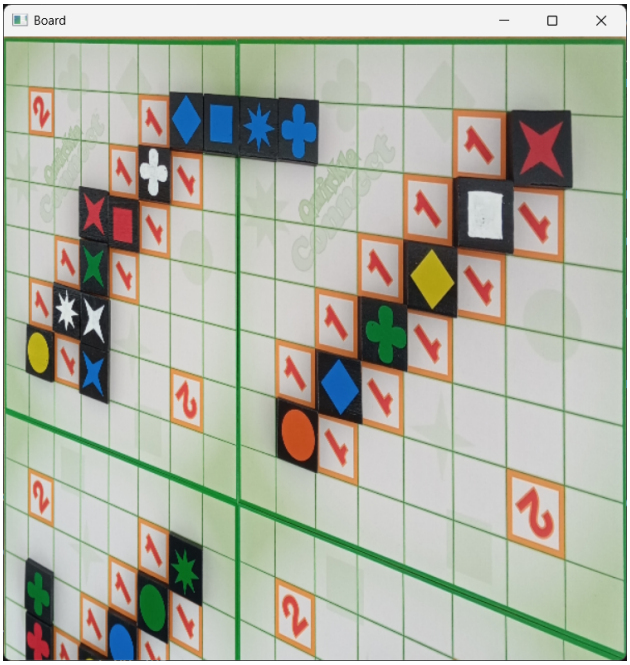

Figure 5: Table extraction without special canny params

As you can see it fails to capture the bottom points. This was the main problem I

encountered with the table extraction as it always seemed that the especially the bottom left part was not well connected, probably because it was darker in the picture.

After that I just extract the cells in the table (because it's size is known, 16x16) and go to classification. One important note is that, to improve classification accuracy later, I calculate the centroid of the strongest connected component in the thresholded image (the same threshold described below for empty/non-empty classification) and regenerate the cell patch based on the centroid.

Figure 6: Before centralization

Figure 7: After centralization

2 Cell Classification

I split my classification in multiple parts starting with the simplest.

2.1 Empty/Non-empty Classification

Here I found out HSV exists and that it gives better precision then converting to gray. I started by analyzing the value (brightness) in order to apply a threshold on V to make the empty cells as close to being fully white as possible and for cells with pieces to actually capture the piece being there (I was trying to maximize the value of empty cells and minimize for full cells to separate them more easily). Then I computed the mean value of the threshold and by testing on different cells I found another threshold to actually correctly separate empty cells from cells with pieces.

2.2 Bonus points Classification

Now there are 2 types of empty cells: Empty cells that give bonus points and empty cells that don't. For that I tried to keep only the numbers for the bonus cells to differentiate more easily. I chose saturation as it seemed like it separated them the best. By testing, I found and applied a binary threshold, then eliminated noise by cutting from all directions (like 10% of the image) and by keeping the strongest connected component. After that I separated by looking at the mean value of different cells.

Then I needed to separate between 1's and 2's. Using template matching seemed like a hassle because the pieces could be rotated and so I would've had to use a lot of templates.

So instead I calculated the circularity of the digit using this formula: $circularity = 4 * \pi * area/(perim * perim)$ where area and perim are the digit's. I was expecting 2 to have a higher degree of circularity but it seems like 1 did. Testing I found a solid threshold that separated 1 from 2.

2.3 Shape Classification

Now for the interesting stuff. For this I'm using the same thresholded image by value (from HSV) I used for the empty/non-empty classification combined with a erode + dilate to eliminate some of the noise. I initially tried template matching (before realizing I can centrailize the image with centroids) but it seemed like it performed very poorly so doing a bit of research I stumbled upon cv.matchShapes which does matching using Hu moments which are "invariant" to scaling, translation and to some degree to rotation.

This rotation invariance created a problem when it came to differentiating between squares and rhombs (you can probably guess why). To overcome it, I approximated the shape into points and I computed the absolute distance between two consecutive points along each of the axis. For squares, the difference should be small along at least one of the axis (they either create a vertical/horizontal line, difference is small on the x axis for the first and on y axis for the second). If it's not a square it's rhomb.

I also found misclassifications between circles, clovers and 8-edge stars. By testing, I found that they have kinda the same number of points when approximating so that's probably why matchShapes failed.

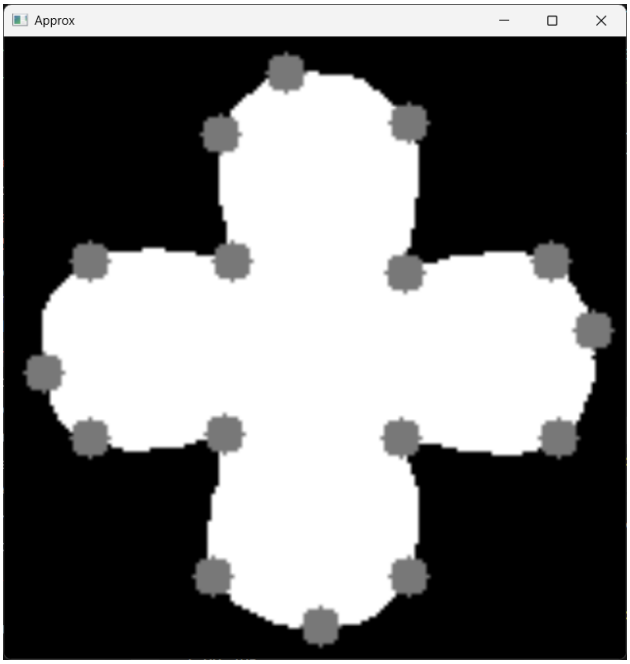

Figure 8: Clover with its approximation points

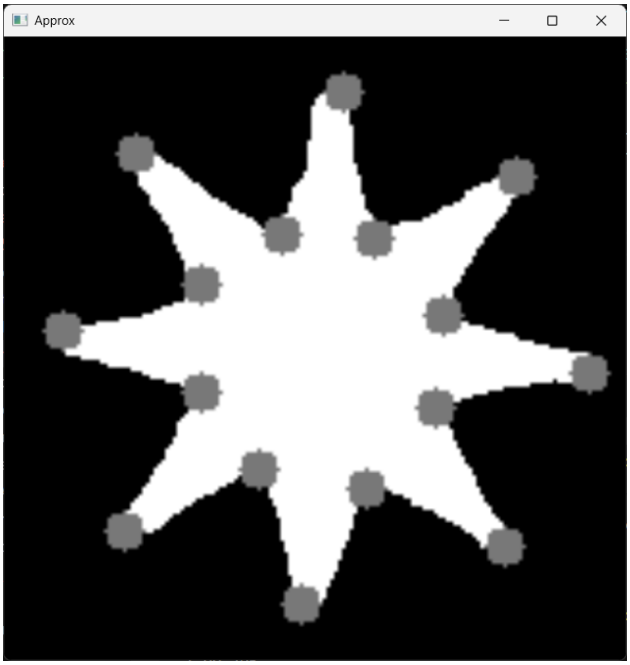

Figure 9: 8-star with its approximation points

But they surely had quite different circularity (as it can also be seen from the pictures). Again, testing, looking at the min/max circularity of each shape I created thresholds to separate them.

Very important note: The templates received the same treatment the cells do.

2.4 Color Classification

For color classification, I used mean hue and saturation after extracting the shape from the image. Saturation was used to differentiate between white and not white cells. Hue was used to differentiate between the different colors.

Here are the observed maximums and minimums for each color:

Channel	**Stat**	**B**	**R**	**W**	**Y**	**G**	**O**
V	min	154	147	198	159	92	161
V	max	187	190	225	194	121	192
S	min	213	151	5	186	201	161
S	max	229	216	25	224	228	213
H	min	104	163	63	25	64	5
H	max	106	177	99	27	74	7

Table 1: HSV bounds (integer-truncated) for each color.

3 Closing notes/Improvements/Interesting Tests

I haven't tried template matching since image centralization and I wonder if it outperforms simple shape matching (without my added edge cases).

I also think the shape classification can be done without using cv.matchShape, but by using point approximation + circularity + maybe HU moments.