Milestone #3

Effective Detection of the board

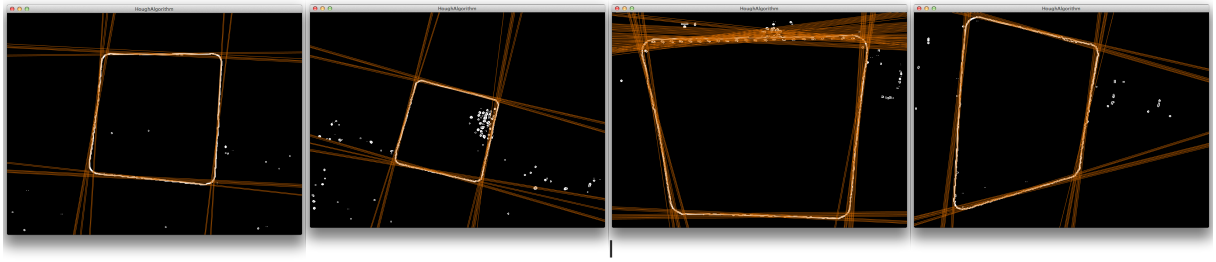
The method *boardEdgesDetection()* returns a greyscale image with the edges of the board. This method is composed of four parts:

1) Color thresholding
2) Convolution with gaussian filter
3) Intensity thresholding
4) Sobel Algorithm

[Optional] To be as precise as possible, the color thresholding is done using the three axes of the HSV color space, i.e the hue, the saturation and the brightness. (See *calibration for color thresholding* paragraph). We tested sobel algorithm with and without a gaussian filter in the pipeline. We did obtain better edges detection when first blurring the images. The more we convolute the image with a gaussian filter, the sharper are the edges after sobel algorithm. However, to avoid too much computation, we will later apply the gaussian filter only once, since it gives us a satisfactory result to pursue with the lines detection.

Hough transform

On week 9, we applied the hough transform on the four images to do a rough lines detection as it is shown on [Illustration 1].



Ilustration 1: Lines detection with minVote = 200 (week 9)

On week 10, we used a comparator to sort the best lines and then select the best candidates around a neighboorhood to keep only the "local maxima lines". Once we had our best selected lines, we computed their intersections to find the corners of the board, as it is shown on the next page [Illustration 2]

We tuned our parameters to be as precise as possible. Here is a table with the results after the hough algorithm after week 10.

	board1.jpg	board2.jpg	board3.jpg	board4.jpg
MinVote = 400	7 lines detected	6 lines detected	14 lines detected	8 lines detected
After Local Maxima Selection	4 lines selected	4 lines selected	6 lines selected	4 lines selected

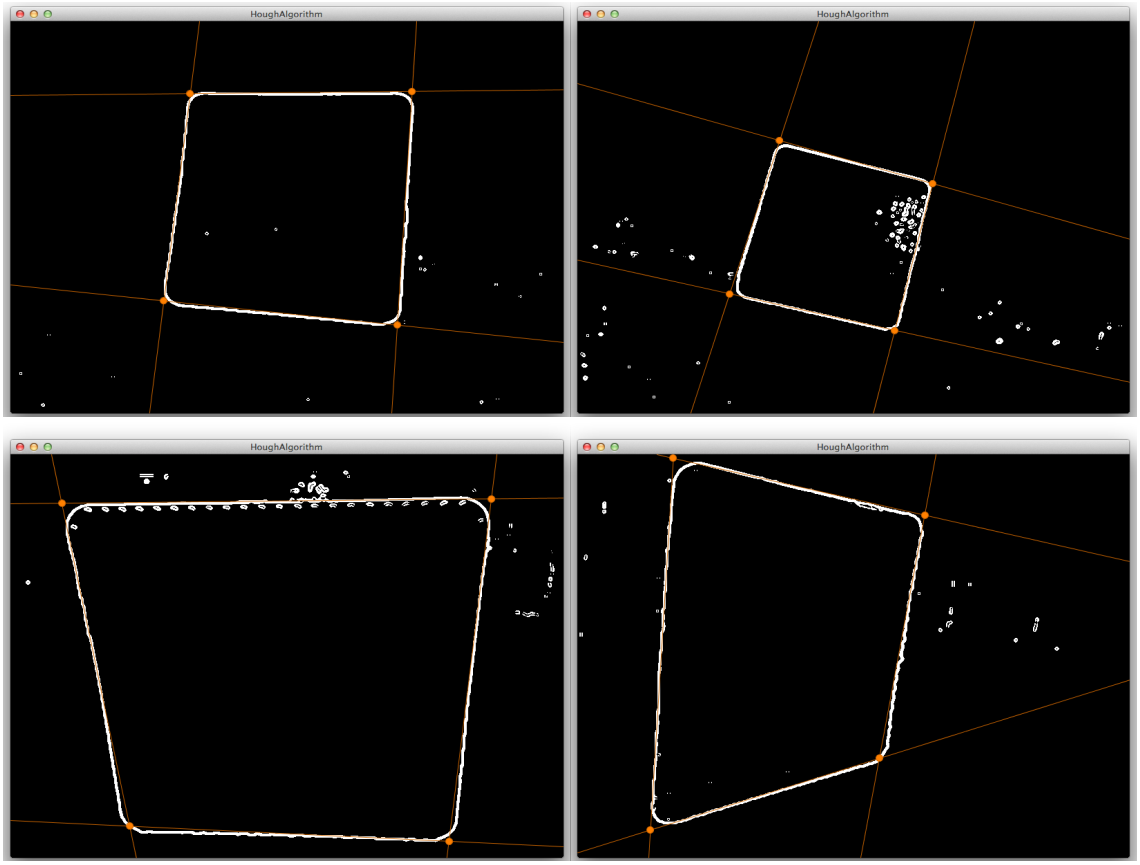


Illustration 2: Lines selection and corners detection (week 10)

[Optional] We did the optimization of saving the cos and sin values in two arrays for each angle phi of our discretized problem. We did store the *Math.cos(ang)* and *Math.sin(ang)* without the inverse, since we use the cos and sin value at some other places.

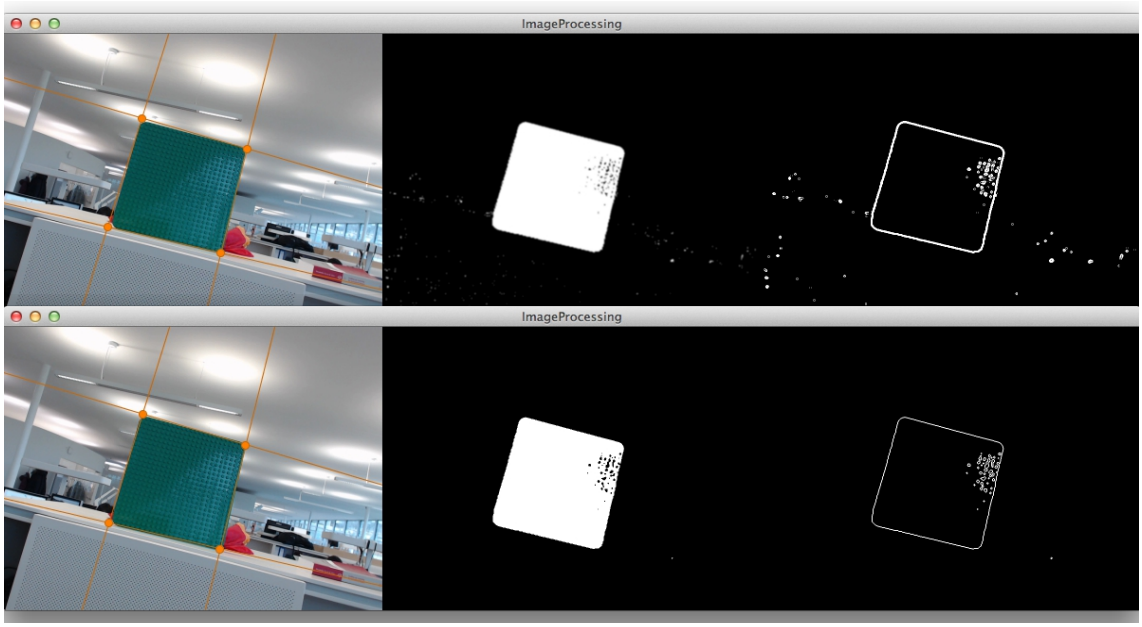


Illustration 3: Improvement of the corners detection after intensity thresholding (week 11)

On week 11, we added an intensity thresholding in the pipeline, between the gaussian filter and the sobel algorithm. We can see on [Illustration 3] that it removes some noise and do a more accurate corners detection. With this new filtering introduced, there are much less lines detected, so we had to decrease the number of minVote to 150, such that the hough algorithm still recognizes at least 4 lines.

Quads Selection

Finally, we did the quads detection. For the purpose of the assignment, we set *nLines* to 6. With *board2.jpg*, we obtained a graph that contains eleven cycles of length 4 for instance. After removing the non-convex, the too small and the almost flat quads, we ended up with only four correct quads in this example case [see Illustration 4].

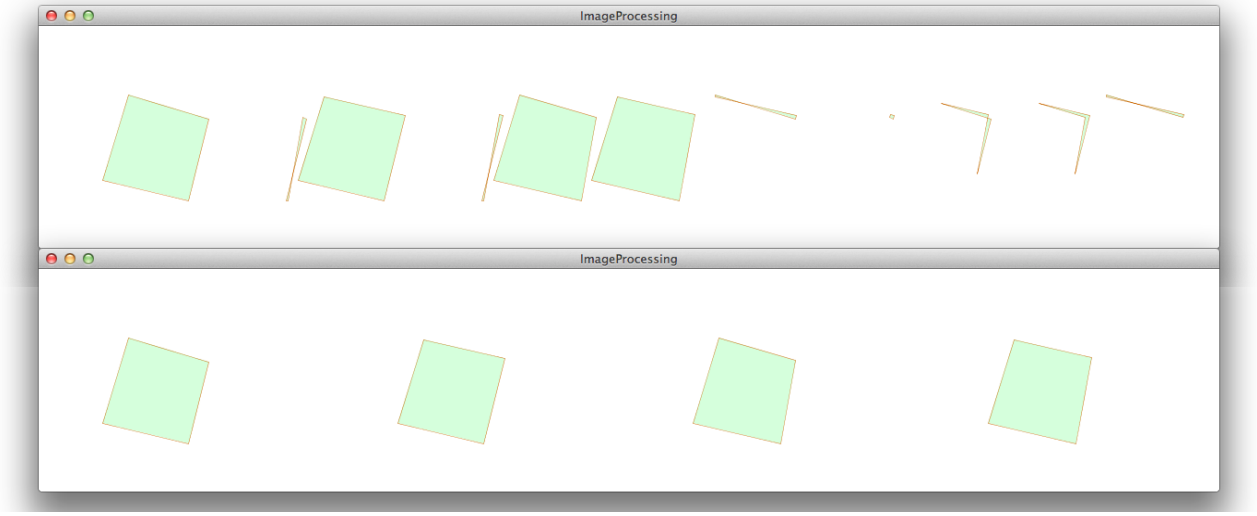


Illustration 4: eleven cycles of length 4 in the graph when we set nLines to 6 (Top)- four quads remain after removing the incorrect quads (Bottom)

Note: The quad detection and selection is commented in the *draw()* fonction. You can uncomment the four lines to display the valid quads on the first left image of the randering window. The variable *nLines* is by default set to 4 in our code. As a side note, *board4.jpg* returns no valid quad because of the angle of the board (non-flat quad).

Calibration for color thresholding using the webcam

Since the detection is very sensitive to the environment in which we are, (i.e depending on the light intensity, shadows, specular highlights, etc), we decided to do a calibration before starting the game. By doing so, we ensure that the user interactivity will work well whereever he wants to play.

[Optional] We implemented a little programm that enables us to calibrate the different thresholds. First, the webcam stream starts and takes four pictures with 10 seconds interval. To do a good calibration, we change the angle and the positions of the board between each capture. These four pictures are then used to calibrate the color thresholds.

On [Illustration 5], each column displays the hue map, resp. the saturation map and

the brightness map. The last column is the result when combining the three thresholdings. We use scrollbars such that we can interactively change the ranges for each criterion. The idea is to obtain a good board detection on the four images, using the same thresholds for all of them. This leads to an average among all the pictures and an accurate detection in general. Once we are satisfied, we simply press the UP key to display the threshold values in the console. These values will be the ones used during the game.

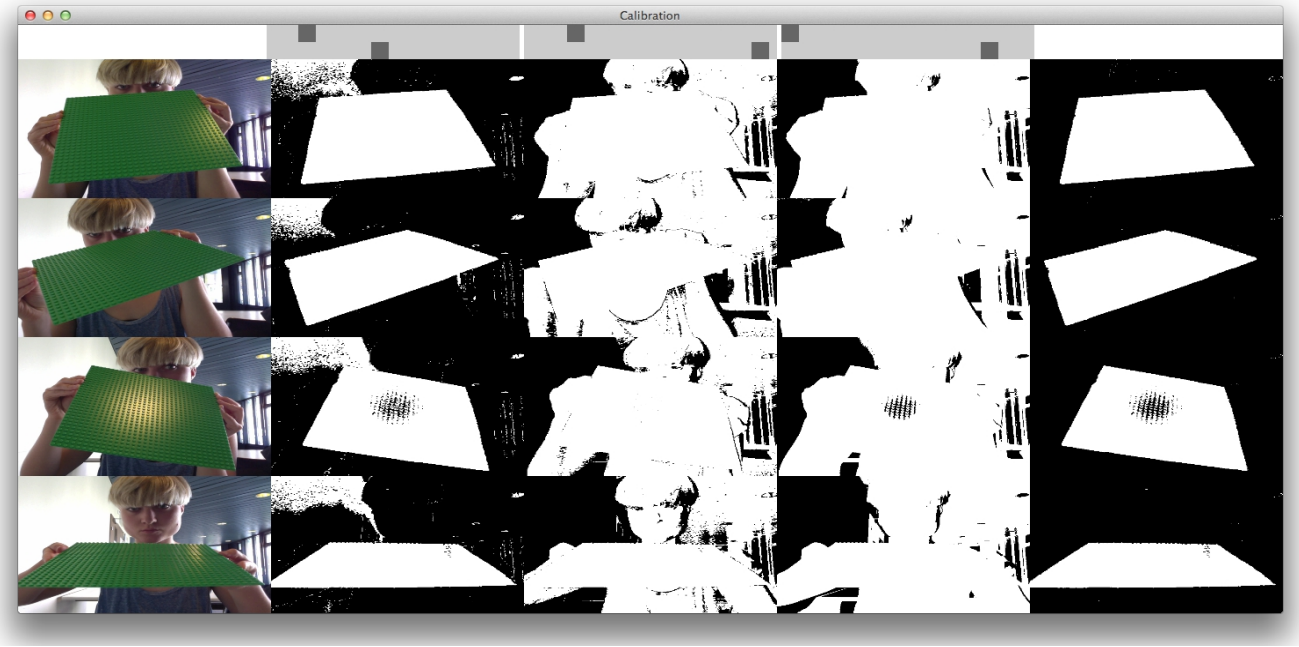


Illustration 5: Calibration

Note

You can download our source files .java on the github link given on Moodle, under the tag Milestone3. To compile and run the code with Eclipse, you should first import the Processing libraries into the project and add them to the build path (at least the *core.jar*).