Introduction to Visual Computing

Assignment# 9

Blob-based detection of the board

April 24, 2018

Description

This assignment aims at improving the detection of the board by looking for the biggest blob in the picture. This stage is performed **after the color threshold** and **before the edge detection** in order to exclude the small areas that do not belong to the game board.

Objectives

Detect the biggest area using a blob-detection algorithm.

Specific Challenges

Implement blob detection using a connected component algorithm.

Preliminary steps

This assignment assumes you already have a good threshold method.

Part I

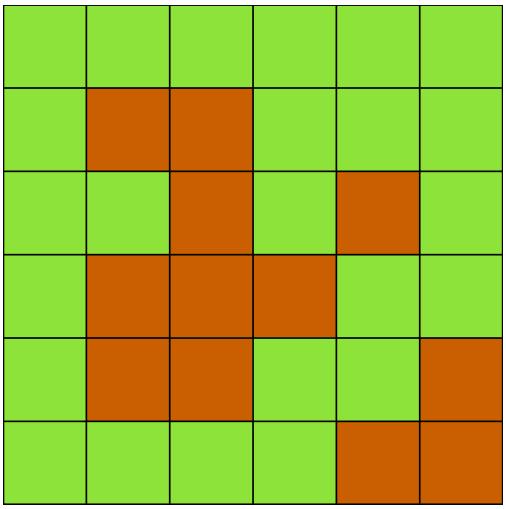
Blob detection

Step 1 – The connected components algorithm

The *connected component labeling* algorithm is a two-pass algorithm that finds and labels similar pixels (in our case, pixels with similar hue) that form **blobs** (groups of similar pixels that touch each others).

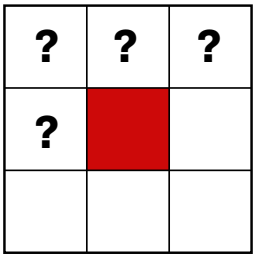
The algorithm works as following:

On a pixel array like this one



we want to detect the orange blobs. We first *label* (by attributing indices from 1 to n) each of the pixels using the following rule:

For each orange pixel,

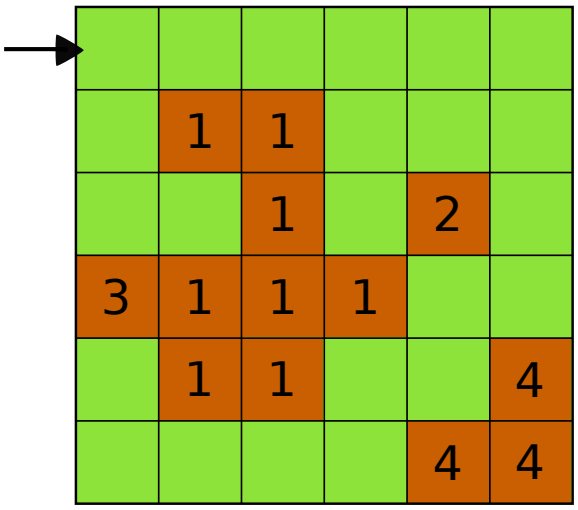


- if some of the four neighbour pixels marked with a "?" already have a label:
  - if they all have the same index, use it to label the pixel ;

  - if they have different indices, take a smallest one to label the pixel, and mark the other indices as being *equivalent* to the smallest index.

- otherwise, create a new label

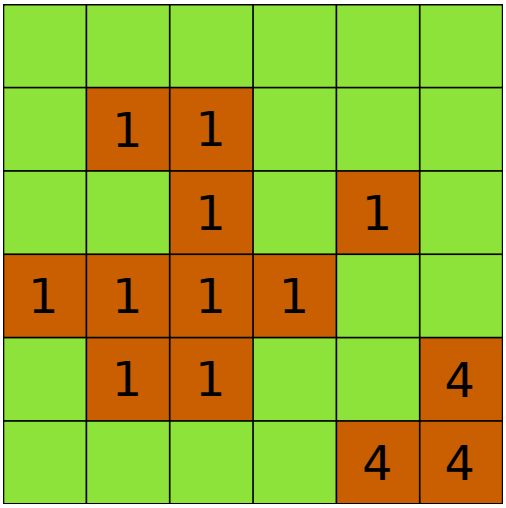
At the end of this first pass, you should obtain this labeling:



And these *equivalence classes*:

- **Eq. 1**: 1, 2, 3
- **Eq. 2**: 1, 2, 3
- **Eq. 3**: 1, 2, 3
- **Eq. 4**: 4

During the second pass, we *re-label* the pixels, using the smallest index in their equivalence class:



(you can read an extended explanation of the algorithm on Wikipedia, along with pseudo-code: `http://en.wikipedia.org/wiki/Connected-component_labeling`)

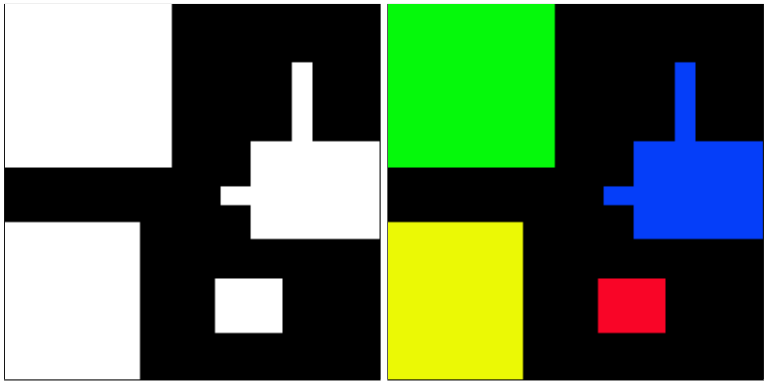
Step 2 – Implementation

Implement the algorithm to find white blobs in a black and white image, using the following skeleton. `findConnectedComponents` should return a new image. If the flag `onlyBiggest` is true, the image contains the biggest blob colored in white and the others in black; otherwise, each blob is colored in a random uniform color.

For the implementation we suggest to use the class `TreeSet` from java.util. TreeSet implements a set in which the elements are sorted in natural order (`https://docs.oracle.com/javase/9/docs/api/java/util/TreeSet.html`). The idea is to store the equivalence classes in the list `labelsEquivalences`: the first element of *labelsEquivalences* contains the set of labels equivalent to the label "1"; the second element of *labelsEquivalences* contains the set of labels equivalent to the label "2", and so on. The smallest index in a equivalence class can be retrieve calling the function `first` of TreeSet, ex. `labelsEquivalences[0].first`.

```
import java.util.ArrayList;
import java.util.List;
import java.util.TreeSet;

class BlobDetection {

    PImage findConnectedComponents(PImage input, boolean onlyBiggest){

        // First pass: label the pixels and store labels' equivalences

        int [] labels= new int [input.width*input.height];
        List<TreeSet<Integer>> labelsEquivalences= new ArrayList<TreeSet<Integer>>();

        int currentLabel=1;

        // TODO!

        // Second pass: re-label the pixels by their equivalent class
        // if onlyBiggest==true, count the number of pixels for each label

        // TODO!

        // Finally,
        // if onlyBiggest==false, output an image with each blob colored in one uniform color
        // if onlyBiggest==true, output an image with the biggest blob colored in white and the others in black

        // TODO!

    }
}
```

Test you implementation on the test image available on Moodle (`BlobDetection_Test.png`).



Step 3 – Integration

Use the blob detection functions after the color threshold to have a better segmentation of the green board from the background. Sparse small blobs will disappear, hence the edge detection will be more accurate.

Next week you will implement the Hough transform technique to extract the board lines.

Bonus

You will be able to use a similar method to detect your tangible object on the Lego Board in the coming weeks. Already, you can implement the detection of two blobs of different color and test it with the *nao.jpg* image. Change your code and pipeline to detect the orange blob of the object as well as the board. (You might need to implement a `add(...)` function that adds up two images)

Below is an example of what you should obtain after the double thresholding (center image) and the blob detection (right image).