Computer vision: project

Thomas Aelbrecht[1], Andreas De Witte[1], Jochen Laroy[1], Pieter-Jan Philips[1], and Gillis Werrebrouck[1]

Ghent University, Valentin Vaerwyckweg 1, 9000 Ghent, Belgium

Abstract. The abstract should briefly summarize the contents of the paper in 150–250 words.

Keywords: First keyword · Second keyword · Another keyword.

1 Introduction

Write introduction

2 Semi-supervised painting detection

For the unsupervised painting detection, a series of actions are executed. The way this works is as follow. All the contours are being detected by a naive method; however, these contours are not being shown yet. A graphical user interface has been made to make the unsupervised image detection easy and user friendly by providing a variaty of actions. The available actions are the following:

1. Add
2. Remove
3. Draw
4. Drag
5. Convert
6. Clear canvas
7. Save to database
8. Next image

Add action The "Add" action allows the user to click onto the image. The idea is that the user clicks on a painting to be "detect". For this action, the contours that were found at the beginning are being used. When clicking on a painting in an image, the code checks if that click event has been triggered in the bounding box of a detected painting, if so then a rectangle is being drawn on the image. All contours in which the user clicked will be made visible.

Remove action The "Remove" action allows the user to click into visible contours to remove them. This is a necessary action because the algorithm sometimes detects more contours than it should, so these can be removed.

Draw action The “Draw” action has been provided in case the algorithm can’t recognize a painting. The user can drag a new contour on the image in case a painting hasn’t been detected by the algorithm.

Drag action The “Drag” action makes it possible to drag the individual corners. This is the most useful part of the unsupervised painting detection because the user can adjust individual corners if necessary. The contours need to be converted before the “Drag” action can work. The corners will have little squares on them once the contours are converted.

Convert action The “Convert” action will convert the contours from rectangles into draggable quadrilaterals. This action needs to be executed to be able to drag individual corners.

Clear canvas action The “Clear canvas” action will remove all the visible contours. This is perfect to reset the image to the original state.

Save to database action The “Save to database” action will save all the quadrilaterals that are visible to the database. This action will also display the next image.

Next image action The “Next image” action will display the next image on which all above actions can be performed.

3 The naive painting detection algorithm

The contour detection is the actual logic part. This is the part that decides what contours are in the image and it works as described below.

Before any contour detection can be done, as many unnecessary details as possible have to be removed. OpenCV has several options to solve this problem, for example erode, dilate, blurring (median, Gaussian. . .) or downscaling. The image needs to get rid of as many details as possible to prevent detection of contours inside the paintings or on the wall. Although this won’t be perfect, the removal of details will decrease the number of incorrectly detected contours.

So the first step of the algorithm is to remove details by resizing the image. Our implementation scales the image down and back up with a factor 5. To scale the image back up, pixel values are calculated using the pixel area. The scale up was mostly done to be able to show the original image, but it’s also a possibility to use the algorithm with the downscaled image. This will result in a slight increase in performance because the image it’s working on would be smaller.

The next step is to convert the image to grayscale. This is done to make it easier to differentiate certain parts. This grayscaled image is then dilated and eroded to remove even more noise at the borders of the painting or on the walls. The last step of removing noise is to do a median blur over it. The biggest advantage of using median blur is that it will preserve edges while removing noise. The idea behind these steps is to smeare as many details as possible, in other words to make bigger blots with the same color. This makes it easier to detect the borders and remove noise in the background.

The next step is to detect edges with Canny. The result of the Canny function will then be dilated once again to make the found edges stronger. After this the contours can be detected using OpenCV's algorithm. This algorithm will make sure only the most outer contours are returned when a hierarchical structure of contours is found. A small final detail to prevent unlogical solutions is the following: any contour with a ratio smaller than 1:10 will be removed. This prevents very small contours to appear around noisy parts of the image.

3.1 Strengths and weaknesses

This biggest strength of this algorithm is that it will give an output in almost every image. An example of an image where no painting will be detected is an image where the framework of the painting is not visible, meaning the image only contains the painting itself, no wall and no framework. With this kind of paintings, the desired sulution is a contour containing the entire painting, while this is not possible because the painting has no border at all. Another advantage is that the algorithm is very fast in detecting contours due to the fact that all the parts of the algorithm are standard functions that have a good performance.

However, the biggest weakness of this algorithm is that almost every found contour is not precise enough, so almost every solution needs a slight adjustment. For some solutions, the algorithm detects contours that are slightly larger than the actual painting, while for other paintings, it doesn't even include the border of the painting.

Another weakness of the algorithm is that paintings sometimes have overexposure or shadows, which makes it harder to detect it's contours. Some paintings even have a small tag with a description next to the painting, sometimes this small tag is detected as being a painting. This is a logical decision, because there's a big contrast in colors and the tag has a clear contour, but this is not a desired effect.

This algorithm is a first version to quickle be able to fill the database with the groundtruth. The actual algorithm in its final version is completely different and doesn't have these issues anymore.

4 Unsupervised painting detection

The algorithm in assignment 2 drastically differs from the algorithm in assignment 1 (see Section 3), it was designed from scratch with the weaknesses of the previous algorithm in mind.

4.1 Strengths and weaknesses

Aanvullen

5 Quantitative comparison

In order to make a quantitative comparison, two things are needed. First of all, the quadrilaterals have to be found autonomous (as described in Section 4). The second thing is the solution created in assignment 1 (see Chapter **??**), which are the presumably perfect paintings. The first part of finding these presumably perfect painting, is retrieving these from the database.

The next part is already building the solution. For this solution, it's required to measure the amount of false negatives (= paintings that are not found at all), the amount of false positives (= detected paintings that aren't paintings) and the bounding box accuracy (= average intersection divided by union).

With the creation of the solution for this problem, a new problem occurs: how to find the intersection of these two shapes? The solution to this problem is made by using "Shapely". To do so, the quadrilaterals have to be transformed into a polygon. The moment this is done, "Shapely" can find the intersection immediately. It has been tested whether this gives correct intersections if they are not intersecting, or when they are sharing only a line. Also some more general intersections have been tested. Once the intersection is made, it's easy to find the area of the polygon (= solution), using "Shapely" once again.

For each of the presumably perfect paintings, all of the found quadrilaterals are checked. The intersection is made and when the "intersection divided by union" is bigger than the previous maximum, than a new best match is found. In the end, when all found quadrilaterals are tried, a check is done whether the area of the intersection is existing. If this is the case, this value is added to the average intersection divided by union parameter and the amount of found paintings is incremented. If this was not the case, than a painting that needs to be found was not found and the amount of false negatives is incremented. After all of the paintings from the database are checked, the average intersection divided by union parameter is divided by the amount of paintings found. Also, the amount of false positives is calculated by subtracting the amount of paintings found from the amount of quadrilaterals.

5.1 Problems

There are two problems with this solution, and both of these problems have the same cause. When quadrilateral is matched with one of the presumably perfect paintings, it's not removed from the list. This means that the quantitative comparison presumes that the algorithm in Section 4 is working correctly. The problem is that when a found quadrilateral from Section 4 is overlapping with more than one painting or when a painting is found twice, this gives more "found paintings" than there actually are. Sometimes resulting in a hidden falsely solution, but sometimes in non hidden falsely solutions as well, meaning the amount of false positives is sometimes incorrect. This can even mean a negative amount of false positives.

6 Qualitative evaluation

aanvullen als dit gedaan is

References