GHENT UNIVERSITY

Faculty of Engineering and Architecture

Computer vision: project assignment

Group 10

by

Thomas Aelbrecht
Andreas De Witte
Jochen Laroy
Pieter-Jan Philips
Gillis Werrebrouck

Academic year 2019–2020

Table of Contents

Todo list

1

Introduction

Write introduction

2
Assignment 1

2.1 General

for the execution of unsupervised painting detection, a series of actions are executed. The way this works is as following. When the image gets loaded in, all the contours are being tracked. however, these contours are not being shown yet. Next we made a graphical user interface, where it's possible to chose which action you want to execute. Each of these actions work in it's own way. The 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 actually click in the painting he wants to “detect”. For this action, the contours that were found at the beginning are being used. These contours are used to get an approach but creating rectangles out of them. The code will use the coordinates of the point in the contour and use the width en height of the contour to check if the click was inside of it. Like this, all contours that are clicked inside will be made visible.

Remove action

The “Remove” action allows the user to click into visible contours and remove these again. This is needed because the algorithm sometimes finds more contours than it should, so these can be removed.

Draw action

The “Draw” action is here in case the program can not recognize a painting. It needs to be possible to draw a contour by ourselves. This works by selecting the top left and the bottom right corners.

Drag action

The “Drag” action makes it possible to drag the individual corners. This makes it possible to perfectly position the individual corners. The contours need to be converted before the “Drag” action can work.

Convert action

The “Convert” action will convert the contours from rectangles into draggable quadrilaterals. This action needs to be done before the dragging can begin.

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 datebase. This action will also go to the next image.

Next image action

The “Next image” action will make the program go to the next image.

2.2 Contour detection

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 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 faster algorithm because the image it's working with is smaller.

For the next steps, the painting needs to be grayscaled. 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 idea of 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 width or height 10 times bigger than the other dimension will be removed. This prevents very small contours around noisy parts of the image given to the Canny algorithm.

2.2.1 Performance

Measure performance

2.2.2 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.

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 puts the contour further than just the painting (so it includes a part of the background), 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 paper with a description next to the painting, even this small paper is detected as being a painting. This is a logical decision, since there's such a big contrast in colors and since the paper has a clear contour, but this is not a desired solution.

misschien minimale afmeting (of minimaal in %) voor correcte contouren opstellen om dit resultaat tegen te gaan en dan deze zin verwijderen.

3

Assignment 2

3.1 Unsupervised painting detection

The algorithm in assignment 2 drastically differs from the algorithm in assignment 1 (see Section 2.2), it was designed from scratch with the weaknesses of the previous algorithm in mind. However the basic idea of the algorithm is still the same: removing as many details as possible before detecting edges or straight lines.

Auteur van de referentie uitvissen

This algorithm starts with grayscaling the image, for the same reason as in assignment 1 (see Section 2.2).

The next step of the algorithm is a bilateral filter. This filter is used to remove noise from an image with preservation of the edges. It uses a small neighborhood of a pixel to calculate the new value. [1]

Thereafter the Canny algorithm is used to detect edges in the image. The resulting images is morphologically transformed using a rectangular kernel, this makes sure rectangular shapes are being closed when there is a lot of noise at their border (e.g. missing pieces). This operation basicly means that the image is first dilated and then eroded.

Then the contours are being detected using OpenCV's algorithm, this is again the same as in assignment 1. Then every contour is being transformed into a polygon so the algorithm can throw away contours that do not meet the following requirements:

- it has four corners;
- it is a valid polygon (e.g. it may not be self-intersecting);
- it has an area bigger than 10% of the image size.

3.1.1 Strengths and weaknesses

3.2 Quantitative comparison

In order to make a quantitative comparison, two things are needed. First of all, the quadrilaterals have to be found fully automatical (as described in Section 3.1). The second thing is the solution created in assignment 1 (see Chapter 2), which is the presumably perfect paintings. The first part for 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 (= program thinks he finds a painting, but it is not a painting) and the bounding box accuracy (= average intersection divided by union).

For the creation of the solution for this problem, a new problem occurs: how to find the intersection of these two things? 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.

3.2.1 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 3.1 is working correctly. The problem is that when a found quadrilateral from Section 3.1 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 aswell, meaning the amount of false positives is sometimes incorrect. This can even mean a negative amount of false positives.

3.3 Qualitative evaluation

aanvullen als dit gedaan is

A
Appendices

Add appendices here

Bibliography

[1] TODO. Bilateral filtering for gray and color images. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html

List of Figures

List of Tables