Computer vision: project

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

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

Kies één van deze twee openingszinnen (of pas wat aan)!

Are you tired of exploring a museum on a map and figuring out where you are?

Have you ever wondered what the final trip, made during an excursion, was?

This paper focusses on describing the different steps to reconstruct the path of an excursion through a museum. The first section describes how the paintings are cut out of high quality pictures. The following section explains an algorithm that can independently find paintings in a picture. This section also describes how accuracy metrics can be collected, which is useful to search for possible improvements. The third section is about how painting are being matched. This means that this section describes performant solutions to figure out which painting is visible in a picture. The fourth section is about how to localize where a person is. This uses an implementation based on the Hidden-Markov model to decide which location is a logical result. The fifth and final section is about the visualization of the result. This section describes a modern way of showing the visited rooms in the museum.

2 Semi-supervised painting detection

2.1 The naive painting detection algorithm

The contour detection is the actual logic part of the semi-supervised painting detection. 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.

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. 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 quickly 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.

3 Unsupervised painting detection

3.1 The detection algorithm

The algorithm in assignment 2 drastically differs from the algorithm in assignment 1 (see Section 2.1), it was designed from scratch with the weaknesses of the previous algorithm in mind.

The first step in the algorithm is a mean shift segmentation (see Figure 1)). This is a method to remove noise by taking the mean of the pixels within a certain range. A big advantage is that it partialy removes color gradients and fine-grain textures which would cause issues in the further steps of the algorithm.

Fig. 1. Mean shift segmentation.

A common technique in object detection is by creating a mask of the object. The problem with this is that we aren't certain about the size and ratio of the paiting as well as where the painting is located in the image and how many

there are in the image. This can be solved by thinking the other way around. The one certainty that is consistent throughout all images is that all paintings hang on a wall. A mask for the wall will be made instead of making a mask for the paintings. The mask of the painting(s) can then be obtained by simply inverting the mask.

Another issue is that this mask can't be statically programmed in code for each image since we wan't to be able to use the algorithm on any image or video frame. This can be solved by using a technique called flooding. Flooding will create a mask from a starting position in the image and will fill all neighbouring pixels if they have a color close to the color of the starting position. This is done recursively until no pixels are withing the color range of the starting position anymore.

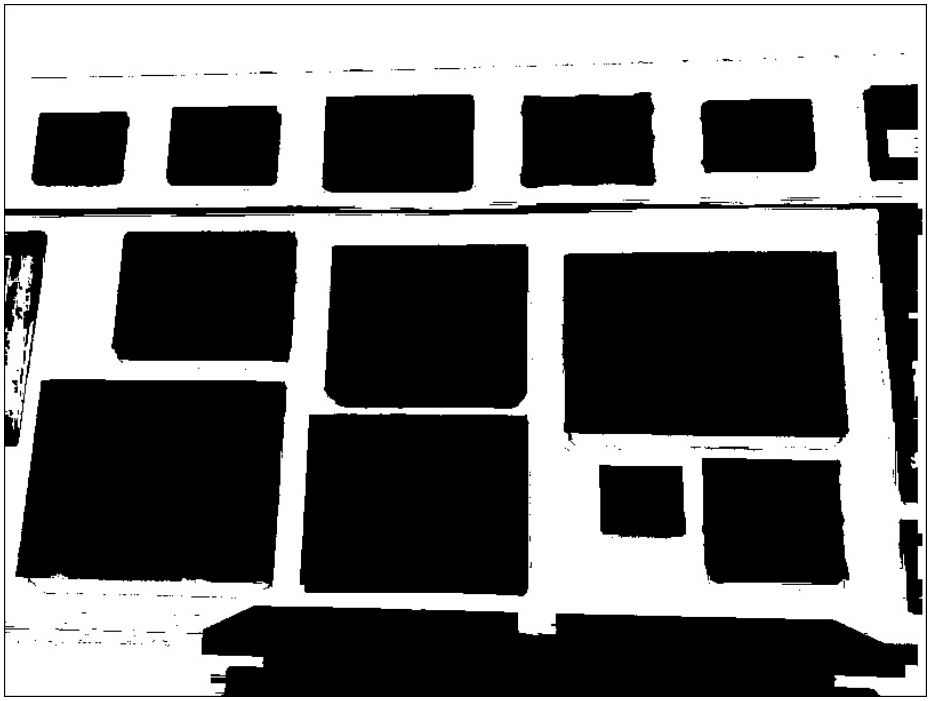

Fig. 2. Wall mask.

The next problem is that there needs to be a good starting position to have a good and correct mask for the wall. To find this, the unsupervised algorithm will iterate over all pixels of the image with a certain step size and perform floodfill with that pixel as starting position. Once a mask is found that has the same height and width as the image, then this is the mask that will be used. If no such mask is found after iterating over all pixels (with a certain step size),

then the mask with the bigest size is used. This is because sometimes part of the floor or other elements in the image will obstruct the floodfill algorithm to find a mask that has the same size of the image. See Figure 2) for an example of such mask.

Once the mask of the wall has been obtained, it is inverted to have the mask of the paintings (see Figure 3). The mask is then eroded to remove small imperfections in the mask and a median blur is used to smooth the edges of the mask.

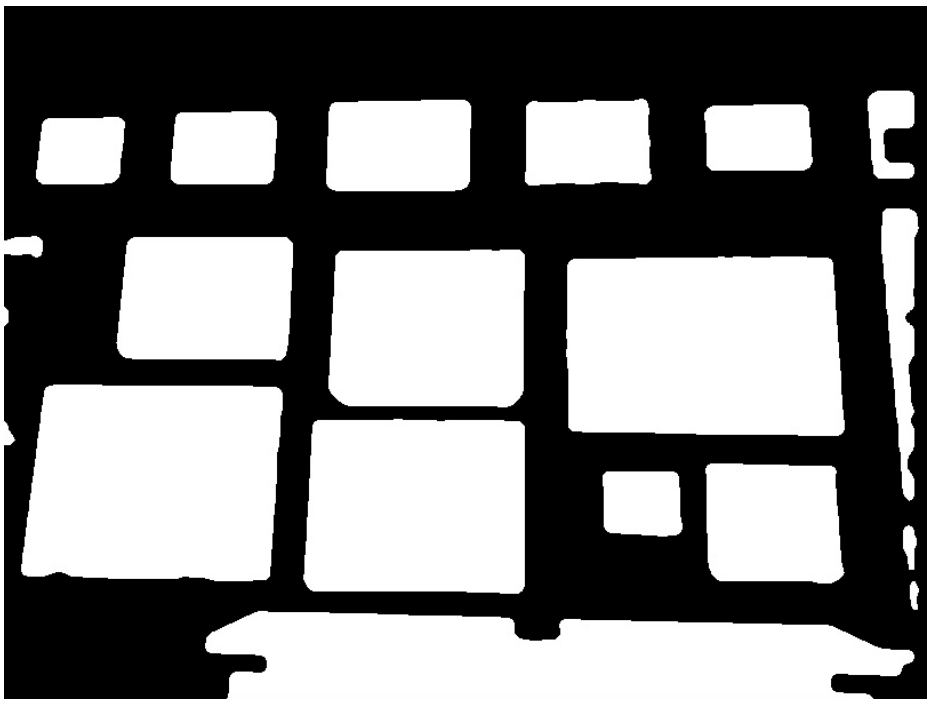

Fig. 3. Paintings mask.

The next step is to use Canny, the difference with the naive approach is that the two threshold values are determined by using the Otsu algorithm to choose the optimal threshold value.

Next, a morphological transformations is performed on the Canny edges (see Figure 4). This will close edges that are close to each other as this will improve the detection of closed contours. This morphological transformations is in essence a dilation followed by an erosion.

The next steps are exactly the same as the naive painting detection algorithm as described before. The contours are detected in the Canny edges and only fully enclosed polygons with 4 sides are returned as quadrilaterals.

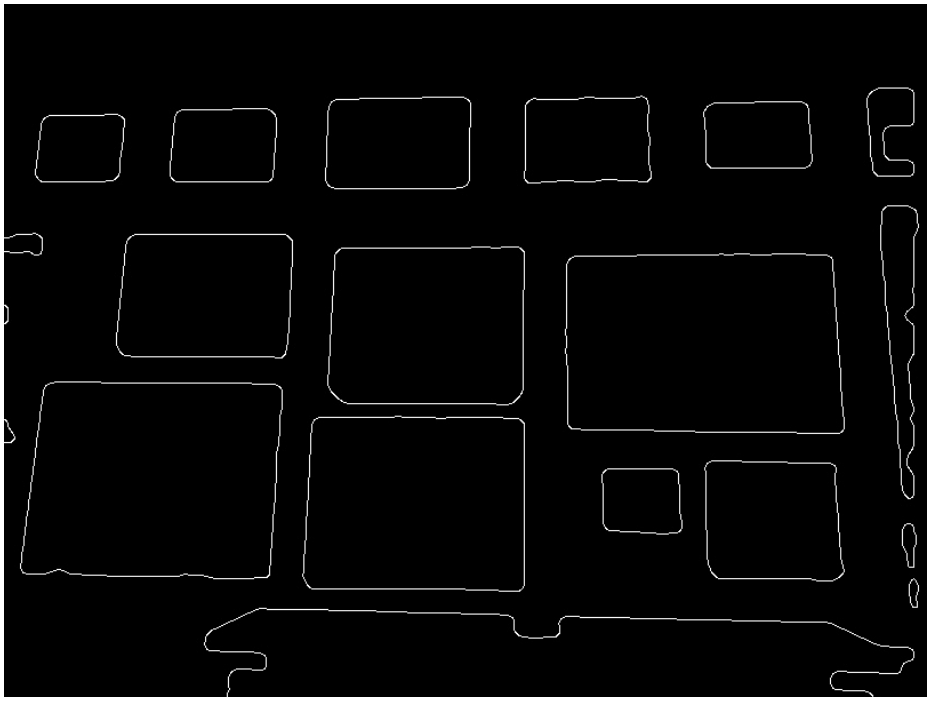

Fig. 4. Paintings edges.

3.2 Strengths and weaknesses

The most important improvement of the detection algorithm is that it isn't based on smearing out colors and finding contours in it anymore. The key to success with this algorithm is the use of automatic mask creation by using floodfill as main technique. Most of the paintings are detected in most of the images. The detected paintings also have a better fit compared to the naive algorithm.

Most paintings in the dataset are correctly detected as well those in the test set. Only on some occasions there is a miss detection. One flaw that has been found is that some doorways are detected as a painting. This is logical because of how the mask creation works with the floodfilling. In the case of such a detection flaw, the doorways obstructs the floodfilling algorithm to correctly create a mask of the non-painting area. This is something that doesn't happen too often.

A minor problem that the algorithm has, which is also the case in the naive algorithm, is that it sometimes detects dark shadows as part of the painting. Although, this is less of an issue in this algorithm then it was in the naive algorithm. It doesn't affect the accuracy too much because if there is a shadow, it only appears on one side of the painting and it doesn't reach far.

A drawback of this algorithm is that it is less performant than the naive algorithm. This is mainly because of the mask creation, specifivally the floodfilling. There has been a slight performace gain by using threading for this but it still

remains slower. A possiblity to resolve this and speed up the algorithm would be to use GPU processing power to perform the mask creation. NVidia for example has Cuda cores on which algorithms can be preformed many times faster than on CPUs. This is not possible for this project because OpenCV (developed by Intel) in Python doesn't have the possiblity to execute on GPU's.

3.3 Quantitative comparison

In order to make a quantitative comparison, two things are needed. First of all, the quadrilaterals have to be found autonomously (as described in Section 2.1). The second thing needed is the groundtruth of the dataset which has been generated by using the solution created in assignment 1 (see Chapter 2).

To measure the accuracy of the painting detection algorithm, 3 things are required; 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. Once this is done, "Shapely" can compute the intersection. It has been tested whether this gives correct intersections if two polygons 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, using "Shapely" once again.

hier wordt misschien teveel beschreven over hoe de code echt werkt -> inkorten kan indien plaats te kort

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, a new best match is found. In the end, when all found quadrilaterals have been tested, 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.

The accuracy results for the detection algorithm were fairly good. An example of the detection can be seen in Figure 5. Red is the detected contours and blue is the groundtruth as determined with the use of assignment 1. The dataset consists of 553 images which all together contain 836 paintings. The detection algorithm had 68 false negatives (paintings) and 35 false positives (paintings). The average bounding box accuracy is 82.80%.

Fig. 5. Paintings detection.

3.4 Qualitative evaluation

For the qualitative evaluation, the unsupervised painting detection algorithm is used to check how accurate the paintings are found on a more difficult set. Hereby the test set and the video files are used. The painting detection is working well, most paintings are being found. However, some difficulties occur. The algorithm sometimes defines a window as being a painting and it almost always defines a doorway as painting.

Another problem is images shot at a sharp angle. For example when it seems like the borders of to adjacent paintings are overlapping. These images are sometimes not recognized. Images that are too blurred are sometimes not found either.

4 Matching

4.1 Features

Before diving deeper into the matching algorithm, it's important to explain the features that used to do the matching. The algorithm only depends on two types histograms of the detected paintings. The first type is a histogram of the full

painting, the second type is a collection of histograms gathered from different blocks of the painting. The block sized used in this algorithm divides the painting in 4 rows by 4 columns, independent of the painting's size. Both features are saved to the database for each of the labeled paintings and are both used in the matching algorithm explained in Section 4.2.

4.2 The matching algorithm

The first step in matching a given painting with the entire dataset of paintings is a light intensity equalization. This way different light intensities have no influence on the histograms gathered from the detected painting.

The second step consists of fetching the histograms as described in Section 4.1. Thereafter these histograms are compared against all histograms in the dataset. Per known painting the distance between the histograms is calculated using a correlation. This way a chance between 0 and 1 is obtained from this calculation. The histogram of the whole painting gives one chance, the block histogram gives a total of 16 chances which are combined by taking the average. These two chances are combined using Formula 1 which calculates a weighted average. In this formula $P(X = P_i)$ stands for the chance that the detected painting X is painting P_i from the dataset, B stands for the average of the block histogram distances and F stands for the distance of the normal histograms. The block histogram gets a much higher weight because the predictions of these histograms are much more reliable than these of the normal histograms.

$$P(X = P_i) = \frac{(8 * B) + (1 * F)}{9} \tag{1}$$

The matching algorithm gives per detected painting a list of possible rooms with the according chances, even if the room is not possible. How impossible rooms are treated, will be explained in Section 5. It's also possible to ignore chances that are below a given threshold.

5 Localization

5.1 Hidden Markov model

In order to establish an intelligent localization of the user by eliminating teleportations a model, inspired by the Hidden Markov model [1], was designed. The reason why no pure Hidden Markov can be used is because the input and output of the model is identical. Another reason is that there's no way to determine the initial room of a given video without have a number of observations, which are rooms in this case. Therefor an alternative Hidden Markov model was designed.

5.2 Alternative Hidden Markov model

The alternative model basicly keeps a history of n observations and predicts the most common observation as the current room. Initially the model doesn't

know the current room and therefor it doesn't emit any prediction until enough observations are obtained.

The model takes a list of probabilities per painting as an input which could possibly contain different probabilities for the same room. However the model needs one probability per room. So Formula 2 [2] is used to combine all probabilities for a given room into one probability. In this formula, $P(X = R)$ stands for the chance that the user is in room R, $P_{test_i}(X = R)$ stands for the chance given by $test_i$ that the user is in room R and $P_{test_i}(X \neq R)$ is therefor the chance that the user is not in room R.

$$P(X = R) = \frac{\prod_i P_{test_i}(X = R)}{\prod_i P_{test_i}(X = R) + \prod_i P_{test_i}(X \neq R)} \tag{2}$$

Although this formula can combine probabilities, it doesn't take the current room into account. Therefor a weight is added to each of the chances. This weight is equal to 1 when the room is accessible from the current room and can be choosen for rooms that aren't accessible, the default is 0.5. This results in Formula 3.

$$P(X = R) = \frac{\prod_i w * P_{test_i}(X = R)}{\prod_i w * P_{test_i}(X = R) + \prod_i w * P_{test_i}(X \neq R)} \tag{3}$$

By calculating the result of this formula per room, a list of probabilities per room is obtained. This list is then used to find the room with the highest probability which will be the observation for this input.

If the current observation is accessible from the current room, it is added to the history of observations. If this is not the case, the current room is added the history. Now, the most common room in the history is taken as the prediction for this input.

6 Visualization

In order to create a visualization of the algorithms described in the previous sections, a 3D model of the floor plan of the museum was created. First, this floor plan is converted to an SVG image so it can easily be altered in Python. Working this way makes it possible to dynamicly create an HTML page showing the SVG floor plan and the currently analyzed frame along with the detected paintings. Because some operations need a lot of processing power, two seperate processes are spawned. One process handles the painting detection and room prediction, the other one handles the visualization of the HTML page in a webview. One of the major points of attention during the creation of the visualization is performance, hence the HTML file is never writen to disk in order to bypass the IO operations. Therefor interprocess communication is used to send the HTML page to the second process. When this process receives an HTML page, it can be visualized in the webview. An example of this webview is shown in Figure **??**. Besides showing the floor plan and the image, other information is shown as well. This includes information about the predicted room, which is also visualized

on the floor plan, a probability for this prediction and the currently processed video's name.

thomas pliz send help -> screenshot van de webview met 3D grondplan

References

1. Eddy, S.R.: Hidden markov models. Current opinion in structural biology **6**(3), 361–365 (1996)
2. Genest, C., Zidek, J.V., et al.: Combining probability distributions: A critique and an annotated bibliography. Statistical Science **1**(1), 114–135 (1986)