Indoor Location Tracking in Museums by Painting Detection and Matching

Raymond Cheung, Robrecht Meersman, Ralph Moens, Niels Verleysen

{Raymond.Cheung, Robrecht.Meersman, Ralph.Moens, Niels.Verleysen}@ugent.be

Abstract

We propose an application for accurately tracking a users location inside a museum by detecting paintings from a video feed and recognizing them by comparing them against a database. We combine the location of these paintings with knowledge of the layout of the museum to determine what path the user has followed. This combination is very important, as painting detection and recognition can be very hard because of doorways, statues, other paintings and other visitors. The problem is separated in three phases: (1) the extraction of paintings; (2) the description and matching; (3) the most likely path estimation given a match. We experiment with different state-of-the-art mechanisms in the different subproblems and supplement them with custom algorithms. Where recent work uses a small database or tends to be slow, our system is used on a large set of paintings and is speed-wise still able to be used in a real life application. The accuracy still has room for improvement, which we justify by the difficulty of the used video data. When putting restrictions on the quality of video footage, the accuracy achieves usable results.

Index terms — Painting recognition, SURF, ORB

1 Introduction

Museums and art galleries often consist of multiple rooms connected together in a maze-like structure. Furthermore, they often rely on signs and paper maps to show the way. In order to find your way around, a user must first find a sign to determine in which room he is, then he has to search for that room on the map to finally determine his route. An alternative approach to this problem would be a mobile application whereby users in an art gallery can obtain their indoor location by taking out their smartphone and simply pointing the camera at the room.

The goal of this paper is to develop a similar kind of end-to-end application, which tracks the location of a user inside a museum given a video of his point of view. The location (i.e. the current room inside the museum) is determined by detecting and recognizing the paintings visible in the video against a database of known paintings. We will experiment with multiple possible approaches and compare the results in terms of accuracy and speed.

1.1 Related Work

A great amount of work has already been done in the detection of artworks in a museum, including some relatively popular mobile applications. The Smartify[1] app is a free mobile application that allows you to scan and identify artworks and access their description and interpretation. The Smartify app recognizes paintings in 44 museums and galleries in just a matter of seconds. Similar apps are Magnus[2], which recognizes over 8 million art pieces with more than 70% accuracy, and Mereasy[3] which is based on Google's image recognition algorithm and images taken from Google Arts, Wikipedia and specialized websites.

In the thesis of Broderick [3], paintings are cropped using a Canny edge detector followed by dilation and contour detection. A SIFT descriptor is used on the cropped version of the image to extract interesting points. The descriptors are matched using k-NN search. Out of a dataset of 92 paintings, 90% are extracted successfully and 58.24% of the matchings are correct. The total process takes 3.34s on average. Bay et al. [1] tackled the recognition of not only paintings, but all museum objects with SIFT and SURF descriptors. Using a dataset of 205 images from objects in the Swiss National Museum in Zurich, 91.1% are matched successfully, without prior cropping of the image. The process takes 16s for feature detection and description, and an additional 59s for matching. In order to reduce the search space, Bluetooth emitters are suggested. Temmermans et al. [16] use a combination of SURF and eigenvectors to achieve a matching accuracy of 88% on a validation set of 216 images. Ruf et al. [14] use a public collection 1,002 works from the Louvre Museum and experiments with both SIFT and SURF. The most accurate results use SIFT and achieve almost 90% accuracy, but take 1440.36s for matching one artwork. In

[1] www.smartify.org

[2] www.magnus.net

[3] www.mereasy.com

[4], a two-layer neural network is used in combination with Bluetooth emitters and trained directly on the mobile phone in an app called PhoneGuide. An article from 2014 [8] only focusses on the cropping phase and achieves 85% success rate when using a chain of Canny edge detection, dilation, contour detection and throwing out all contours that do not fulfill some predefined constraints.

In most previous work, the time needed to match a painting is too long for actual usage. Ideally, a painting is recognized within a second. More than 10 seconds waiting will lead to a bad user experience and most users leaving the application, thinking it is defect [12]. The overview of previous work also hints that dataset size has a large influence on matching time.

Where recent work uses a small database or tends to be slow, our system is used on a large set of paintings and is speed-wise still able to be used in a real life application. Furthermore, by combining the detected paintings with the known layout of the museum, we can extend the use case of the application to indoor location tracking. Our application can accurately track the users followed path, even if a lot of false detections are made.

1.2 Overview

The remaining sections of the paper will discuss the implementation of the end-to-end application using multiple approaches along with a comparison and analysis.

In the next section we will thoroughly discuss the method used for building a database, extracting feature descriptors of interesting points, matching two descriptors and determining the followed path by the user. Section 3 will provide a detailed description of the experimental setup used for comparing different algorithms, along with the results of those experiments and an evaluation of the results. Finally in Section 4, we discus and summarize the use of this application and provide suggestions for improvement.

2 Method

In most applications, there is no access to a normalized database of the paintings present in the museum. It is however practically possible to manually photograph each painting, label them by room, and normalize them using a software implementation. The normalization (of lighting and perspective) is important, since the paintings detected in the video will be noisy and distorted and therefore harder to match with unnormalized images. Furthermore we want to remove all surrounding bias by cropping the image around the painting borders. This bias, such as the painting information panels hanging next to them, or parts of the floor and ceiling are similar in most images and will result in incorrect matches.

In a first stage, the **database building stage**, the painting contours will be detected and extracted from the images labeled by room. The paintings are corrected for perspective transformations and are all normalized to a 512×512 pixels image. In the normalized images, interesting points can be detected and stored along with them. The database building stage only happens once and can be executed at any time before the matching stage, since the interesting points and normalized images will be stored on disk.

In the second stage, the **database matching stage**, the video footage of the user is analyzed. The detected paintings are normalized to the same 512×512 dimensions and their interesting points are compared against the database in order to find the best match. Based on the label of the match and the match confidence, the most likely path of the user throughout the video is inferred.

Because the two stages contain similar concepts, we will approach the methods in a more logical sense. In §2.1 we discuss the preprocessing strategies on both video and images. In §2.2 the detection algorithm of the paintings contours is presented, which is also applicable to video and image. The matching process and location estimation are described in §2.3 and §2.4.

2.1 Preprocessing

Before an image or a video frame will be used for painting detection and extraction we first have to make sure that the image is usable and yields optimal results. For those reasons the image will first go through several steps of preprocessing.

Blur Detection

The first step in preprocessing will be blur detection, which is only performed in the case of video. Before an image frame from a video can be used for line, corner and eventually painting detection, it is very useful to check the blurriness of the given image. Video footage is subject to motion blur due to moving the camera too quickly in combination with the low shutter speed of smartphone cameras. If the image is too blurry, it will lead to sub-optimal detection results due to the application's inability to detect crucial keypoints or lines. We have used the variance on the Laplacian to compute a single metric that represents how sharp the given image is. The Laplacian or second derivative output is computed by convolving the image with a 3×3 kernel:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

This single metric value is calculated by convolving the image with the above mentioned kernel and taking the variance (i.e. standard deviation squared) of that response. In our implementation a threshold value of $t = 70$ is used to compare it with the variance, in case this variance value is higher than the threshold t, the frame is considered to be not blurry. If this metric value is below the chosen threshold, the frame is considered blurry and will be ignored.

Median Blur

After it is determined that an image is sharp enough to be used for detection, the image will first undergo blurring by using a 5×5 median filter. As a result of median blurring the image will be smoothened and noise is removed. This method is very effective for removing impulsive types of noise like salt and pepper noise.

Horizontal Erode for Motion Blur Removal

At this stage of preprocessing the frames considered too blurry have been removed using the Laplacian Blur detection and the noise has been removed using a median filter. Even

Figure 1: Horizontal and vertical DoG filters. Each filter is 64×64 pixels.

though all blurry images have been filtered away, the remaining images might still contain a small amount of motion blur. In the video footage used in this paper, the motion blur was always dominant in the horizontal direction, due to the person pointing the camera at different paintings (by making a horizontal rotation with the camera). By eroding all horizontal structural elements present in the image, the amount of horizontal motion blur can be minimized to some extend.

2.2 Painting Detection & Extraction

The goal of painting detection and extraction is finding the 4 points that form a bounding tetragon around the painting in the original image. These will be used later to crop and transform the images to fixed dimensions. The painting detection and extraction happens in three sequential phases. Each phase receives the output values of the previous one.

Edge Detection

After image preprocessing, an edge detection filter is applied. We experiment with two mechanisms of edge detection. In the first experiment, a Canny edge detector [6] is used. Canny edge detectors are very robust and detect many edges, including texture in the frame of the painting and edges in the painting itself. In a second experiment, the preprocessed images are filtered with two Differential of Gaussian (DoG) filters. The DoG filters are presented in Figure 1 and are each suited for detecting strong, long edges in respectively horizontal and vertical orientation. This prevents edges inside paintings (which are mostly non-horizontal/vertical) and shorter edges from being picked up. The absolute values of the two filter results are added up, and a binary threshold followed by an erosion are applied. Figure 2 shows the results of both the Canny edge detector and the use of DoG filters. Note that the DoG edge detector is based on several heuristics. When the user holds the camera in a 45° angle, when the painting itself has a diamond-shaped frame, or when the user is looking at a painting from a too sharp angle, the DoG edge detector will perform very poorly.

Line Detection

On the binary edge detected image, lines are found using Probabilistic Hough Transform [9]. This optimization of Hough Transform does not take all the points into consideration when voting for possible lines, but only a random subset of points. We experiment with the vote threshold, the resolution and the minimum line length.

Most Likely Corners Estimation

Now that we have detected all possible painting borders in an image, the 4 most likely have to be chosen in order to extract the painting from its surroundings. With the intent of avoiding an $O(n^4)$ method, several heuristics are applied:

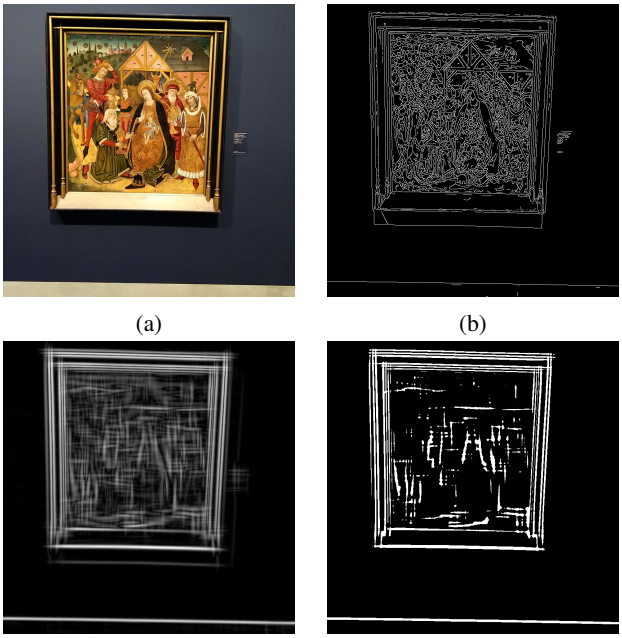


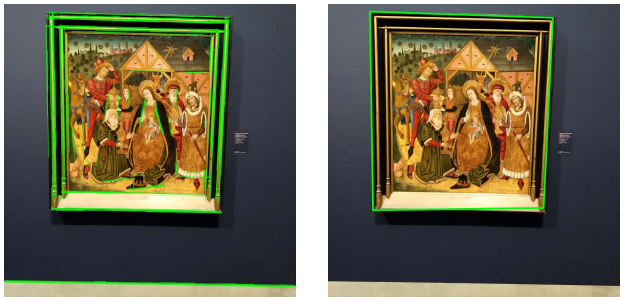
Figure 2: (a) Original image; (b) Canny edge detector result; (c) DoG filter result; (d) Theshold of 50 and erosion with kernel size 3 on DoG filter result.

- All paintings have a rectangular shape and have been subjected to a relatively small homography transformation.
- The frames of the paintings have a lot of parallel lines (as in Figure 2)
- All paintings have an aspect ratio between 3:1 and 1:3.

We propose two algorithms. The first algorithm is $O(n^2)$, the second is $O(n)$.

The first algorithm consists of three steps. In a first step, we look for all lines parallel and perpendicular to each line i. The lines are represented in polar (ρ, θ) coordinates and we tolerate a deviation on θ of $\pm 5.7^\circ$ because we are looking for trapezoids. While doing this, the distance of the parallel line to the current line i is calculated and the total amount of parallel/perpendicular lines is counted. In the second step, four candidate borders are picked. Out of all lines, the line with the largest amount of parallel/perpendicular lines is chosen as a first candidate l_1. The other three lines are chosen based on their difference in ρ so that we pick the outermost lines. In the last step, the aspect ratio of the candidate polygon is evaluated. If the aspect ratio is not within bounds, the polygon is narrowed by picking another candidate from the lines parallel or perpendicular to l_1, depending on the orientation. If no polygon with a correct aspect ratio is found or if we don't find parallel lines at all, no prediction is made and the frame is skipped. Figure 3 presents the detected edges and the 4 chosen estimates on the paintings border.

The second algorithm is quite similar to the first, but uses 18 buckets for $\theta \in [0, \pi[$ to divide all lines by orientation. The two buckets with the highest amount of lines are chosen



(a) (b)

Figure 3: (a) Edge detection using Probabilistic Hough Transform; (b) Corner prediction using Algorithm 1.

as the horizontal and vertical values for θ. From each bucket, two lines with the smallest and highest ρ are picked as candidates and the aspect ratio of the candidate polygon is evaluated. Note that while being the faster implementation, this comes with the drawback of the real painting borders being separated into > 2 different buckets due to unlucky bucket boundaries. Increasing the amount of buckets leads to less tolerance for perspective transformations, and decreasing the amount of buckets leads to incorrect lines being chosen as candidates.

2.3 Feature Detection & Matching

When matching a detected painting in the video against a database of normalized paintings, a combination of global and local feature detection and matching is used. Each matching mechanism outputs a series of logits for each image, which are converted to probabilities with a softmax function. Those probability distributions are combined using a weighted sum and the final prediction is made:

$$\mathbf{p} = \text{softmax}(\mathbf{y}) = \frac{\exp(\mathbf{y})}{\sum_{j=1}^{V} \exp(y_j)}$$

$$i = argmax(w_1 \cdot \mathbf{p}_{global} + w_2 \cdot \mathbf{p}_{local})$$

Global: Histogram Comparison

One way to match images with each other is using the histograms extracted from those images. By comparing the histograms, we can determine how well the images match with each other using a scoring metric. The higher the matching score, the more likely the images contain similar or common elements. Before the actual matching of the histogram of the (frame) image and the histograms of the images in our database we first need to calculate and extract those histograms. In our case we create a histogram for every image present with 256 bins, each representing a single color in the color range. The color for every pixel will be extracted from the image and sorted in the correct bin according to their color value, these histograms will be stored in a pickle file so it can be used later. There are different methods for comparing histograms. We use the method of calculating the correlation between the two histograms to compute the matching score. This score has a range from -1 to 1 indicating how similar the compared images are with -1 completely different to 1 indicating the compared images are perfectly identical. Out of every match, the database image with the highest score will be chosen.

Local: Feature Detection and Matching

The best approach to matching a scene against a database of images is by using feature descriptors. These look for interesting patches in the image, called *keypoints* and encode information about these keypoints into a vector of numbers. Such points usually lie on high-contrast regions of the image, such as corners or textures. This *feature* vector acts as a numerical ID that can be used to differentiate one keypoint from another. Ideally this information would be invariant under image transformation, so we can find the feature again even if the image is transformed in some way. These transformations may include scaling, homography or even affine transformations and noise, depending on the feature descriptor algorithm used. Common examples of feature descriptors are SIFT, SURF and ORB:

SIFT [10] – The Scale Invariant Feature Transform encodes information about the image gradients in a 16×16 pixel window around the keypoint. SIFT can robustly identify objects even under partial occlusion, because the feature descriptor is invariant to uniform scaling, orientation, illumination changes, and partially invariant to affine distortion. Despite being the slowest of the three descriptors mentioned, SIFT leads to the most accurate results in most scenarios.

SURF [2] – The Speeded Up Robust Features algorithm is a faster variant of SIFT, which uses fast approximations for the derivatives and integrals used in SIFT. SURF is approximately 3 times faster than SIFT [7].

ORB [13] – The Oriented FAST and rotated BRIEF is a combination of the FAST keypoint detector and the BRIEF descriptor [5]. ORB performs slightly faster than the SURF algorithm, but the features are mostly concentrated in objects at the center of the image while in SURF, SIFT the keypoints are more distributed over the image. [7]

Once we have computed a set of descriptors in the detected artwork, the descriptors can be queried against the database. Although there are better data structures such as k-d trees and hash tables [15], we choose to match the features against those of every image in the database. The feature descriptors of the images in the database can be precomputed. Still, every image contains lots of features, which need to be matched efficiently. There are two approaches for matching the descriptors of two images:

Brute-Force matching – In a greedy approach, each pair of descriptors is compared against each other. The metric for comparison depends on the type of feature descriptors used. For SIFT and SURF, the Euclidian distance in the feature space is computed. ORB uses binary feature vectors instead of floating points, so the Hamming distance can be used.

FLANN-based matching [11] – Fast Linear Approximate Nearest Neighbors builds a k-d tree, an efficient data structure that will be used to search for an approximate

neighbor. FLANN is a trade-off between speed and accuracy. This means that it will find a good matching, but not necessarily the best possible one. When using large (> 1K) sets of descriptors, FLANN finds a match quicker than a brute force approach.

2.4 Indoor Location Estimation

When combining the described methods, we can already detect and classify paintings. Thus, giving the location of the user in the museum if the camera is fixed at a painting. However, most of the time the camera will not be facing a painting directly, causing the previous described feature matching to predict the wrong room. There can also be other things than paintings in the camera view, which can confuse the painting detector. Those cases are not exceptions as the objects are very common in a museum, e.g.: doorways, windows, statues and other visitors. In a majority of frames the described process will detect the wrong painting and therefore wrong room.

The layout of the museum is known beforehand, thus enabling the use of logic to determine exactly where the user is in the museum, even though a lot of frames result in a detection of the wrong room. To do this we defined an algorithm which uses the known layout and a score-mechanism to determine the most likely path being followed by the user. A schematic view of this algorithm is given in figure 4.

The algorithm keeps track of the current most likely path and every detected room between each room on that path. For every room in this list we also keep track of how many times this room has been visited, we call this the score.

After detecting a room, the score of this room is updated. If this room is the last room of the current path, the algorithm can return the current path unchanged. If not, we try to connect the room to the path and if successful the path is updated and returned. If the room cannot be appended to the path, we have to check if a segment on the end of the path has to be replaced by this room. To do this the score of the room and the segment are calculated. For the segment this is the sum of all rooms from the segment of the path. For the detected room this is the sum of every occurrence of that room from the last element of the current path that is not in the segment to the latest added room. If the score of the segment is larger than the score of the detected room, nothing happens and the current path is returned. However, if the score of the detected room is higher than the score of the segment, we should check if this segment could be replaced. If the detected room can be appended to the current path without the segment, the segment is replaced and the new path is returned. If the segment cannot be replaced by the detected room, the size of the segment is increased and the scores are recalculated. These scores are then again compared. The segment is increased in size until the score of the segment is higher than the score of the room, or the segment is replaced by the room.

This algorithm increases the accuracy of determining in which room the user is by being more strict in estimating the current room. This strictness can also have a negative effect in certain cases. When no good or not enough detections are made in a room and the user goes to the next room, the algorithm will need to replace the whole path by this next room in the worst case. Before this can be done however, the score of the next room should become higher than the score of the path. As the path grows longer, this becomes a harder thing to do. This could also happen when a wrong room is detected many times in a row (e.g. by looking at a wall, door or window for some time). In an ideal use-case, the user should not rush through rooms and should make sure to focus on at least one painting in a room. If a room is skipped the algorithm can again detect the real room if the user focusses the camera on a painting in the room. The path until then will however be lost in that case.

3 Results

In this part of the paper we go over the performance of the different parts of the application and explain where the difficulties lie. First the performance of the rectangle detection algorithm is reviewed. This algorithm has a decent performance when a single painting is visible in the image, but has difficulties with multiple paintings at the same time. Second, we look at the results of the painting matching, comparing the performances of the different kinds of descriptors. SIFT and SURF prove to be more accurate, but also slower than ORB. Finally the complete end-to-end application is reviewed on multiple videos, corresponding to general use-cases. From these results we can determine that the application already has a very good accuracy in certain use-cases, but has its limitations. When these limitations are reached, the accuracy drops drastically.

3.1 Painting Detection & Extraction

In the painting detection phase, we experiment with the types of edge detectors used and the algorithm for estimating the paintings corners. All experiments were evaluated on a validation set of 40 randomly selected samples from different rooms. As a metric of evaluation we use the intersection over union (IoU). The IoU is calculated as the ratio of two areas:

- Area of overlap (intersection) between the ground truth tetragon and the prediction
- Area of union between the ground truth tetragon and the prediction

The IoU can thus be interpreted as an accuracy percentage.

Table 1 shows the impact of the different edge detectors on the accuracy. Both use Algorithm 1 for corner estimation and use a median blur with kernel size 5 before edge detection. We experiment with different thresholds. In the case of the Canny detector, the threshold refers to the upper threshold. The lower threshold is calculated as $2/3$ of the upper threshold. For the DoG, we filter the image with a 64×64 DoG kernel and normalize the output. Pixel intensity values above the threshold value will be detected as edges. After thresholding, an erosion of 3 pixels is applied.

Table 2 presents the accuracy of both corner estimation algorithms described in §2.2. We see that the first algorithm results in a better intersection over union accuracy, but the second algorithm is 45% faster.

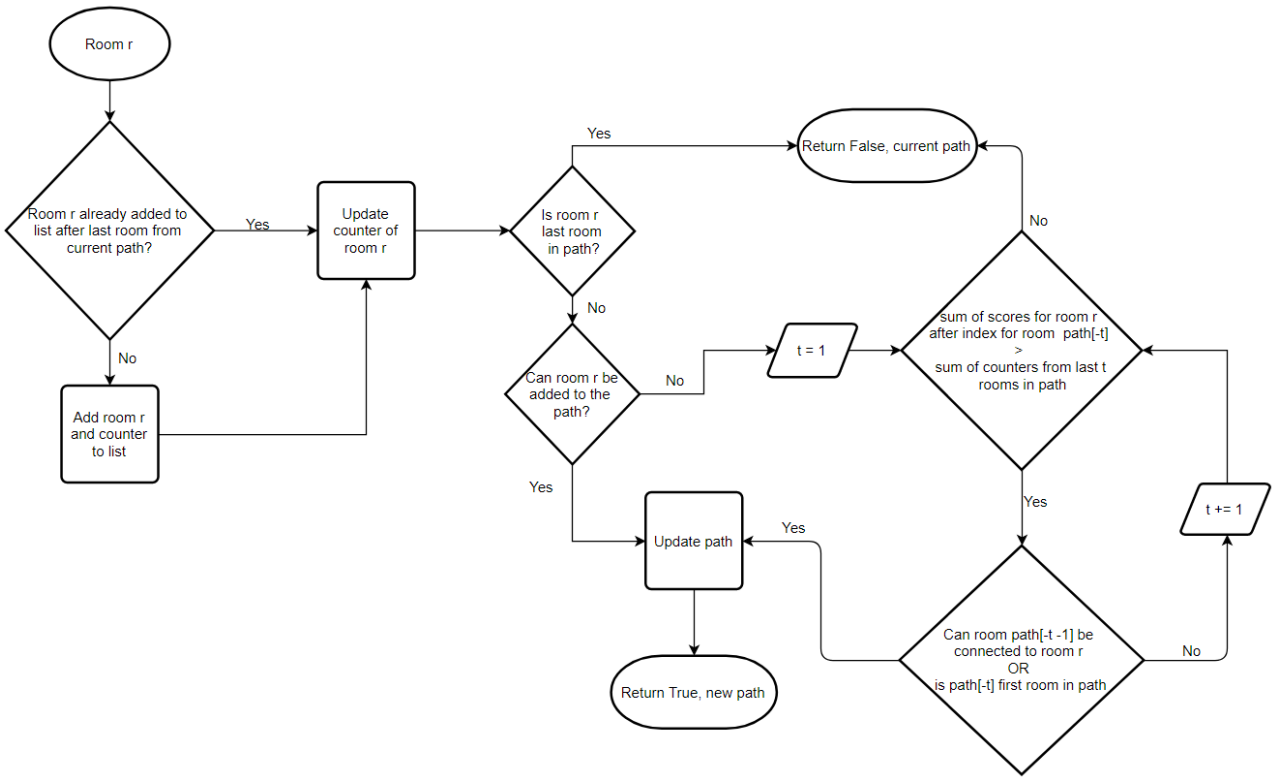


Figure 4: Most likely path algorithm.

Table 1: Impact of different Edge Detectors and thresholds on the intersection over union (IoU) for a validation set of 40 samples

Method	**Threshold**	**IoU** (%)
Canny	20	42.98 ± 35.50
	40	$\mathbf{69.80 \pm 28.68}$
	60	62.78 ± 34.22
	80	63.88 ± 32.94
DoG	40	66.52 ± 28.78
	50	65.61 ± 26.68
	60	$\mathbf{70.30 \pm 29.13}$
	70	62.30 ± 34.61
	80	61.02 ± 37.76

Table 2: Comparison in speed and accuracy of different corner estimation algorithms

Method	**IoU** (%)	**Speed** (s per image)
Algorithm 1	70.30 ± 29.13	1.29
Algorithm 2	62.88 ± 36.12	0.89

When we analyze the mistakes, we see some patterns occurring. In theory, the outer edge of the frame of the painting should be detected. Because both algorithms select the outermost polygon, the detected borders are typically a little larger than the actual frame. Other parallel lines, such as baseboards or the frames of other paintings, may be mistaken for the edge of the painting. While this means that some redundant information is passed to the classifier, the perspective transformation will in most cases still be correct. When the detected frame is smaller, it is typically the upper edge of the frame which is omitted.

The detection of the frames heavily relies on the frames being rectangular. Other polygonal frames (e.g. an octagonal frame) are still detected, but will contain large parts of the background. Ellipsoid frames naturally cannot be detected in this way.

When there are multiple paintings in view at the same time, they are often grouped together inside one detected "painting". An example of this phenomenon can be seen in Figure 5.

When paintings are only partially in view, e.g. one corner is out of view, the algorithm can still detect the painting by extending the visible lines. When for example a full side of the painting is not in view, the algorithm will fail to find 4 candidate lines.

3.2 Painting Matching

For the feature descriptors, we experiment with the different descriptor strategies discussed in §2.3. From a collection of

Figure 5: Detection when multiple paintings are in view (detected frame in blue)

Table 3: Accuracy of different feature descriptors on a validation set of 10 manually selected frames.

Descriptor	**Accuracy**	**Speed** (s per frame)
SIFT	9/10	17.92
SURF	9/10	8.54
ORB	8/10	1.01

video files, 10 frames are manually selected on the basis that they do not contain motion blur. The frames contain 10 paintings from 6 different halls. These paintings are then manually extracted in order to remove any bias due to the inaccuracies from the corner estimation algorithm. On the perspective corrected paintings, feature description and matching is executed. The weights of the sum of histogram comparison and descriptor matching are respectively set to 0 and 1, so that we solely rely on descriptor matching. In the case of SIFT and ORB, the amount of detected keypoints is set to a fixed value of 400. When using SURF, a Hessian threshold of 600 is used. The results of the different feature descriptors are presented in Table 3. The differences in speed behave as predicted. Each descriptor claims to be faster than its predecessor, which also is the case in our results. In terms of matching accuracy, the three descriptors all perform acceptably, with a small decrease for the ORB descriptor. However, the validation set is too small to state that the ORB descriptor performs worse. Note that these results are obtained with perfectly cropped paintings. When a painting is cropped using the algorithm discussed in §2.2, more mistakes will be made.

3.3 Indoor Location Estimation

When looking at the performance of the end-to-end application, we need to look at two different important metrics. The accuracy of the application in determining the location of the user is the end goal and thus the most important metric. The second metric is the speed of this process, as an application that is incredibly slow is not really useable. The application was tested on 8 different videos of about 5 minutes each, showing the point of view of a user walking through the MSK Museum of Fine Arts in Ghent, Belgium. The videos have been manually labeled, stating the room in function of the frame number. As a database source, we make use of 688 input images. Each image contains a unique painting, labeled with one of the 38 different rooms. The images are normalized using DoG edge detection and Algorithm 1 and the videos are matched using SURF. The measurements are displayed in Table 4.

Table 4: Accuracy rate and speed of the end-to-end application on some example videos

Video	**Accuracy** (%)	**Speed** (s per frame)
Video 1	12.3	5.13
Video 2	38.9	6.43
Video 3	38.9	10.30
Video 4	42.3	9.50
Video 5	27.9	6.93
Video 6	79.0	6.82
Video 7	46.8	10.40
Video 8	63.8	13.70
Average	43.78 ± 19.12	8.34 ± 2.12

As can be seen from the table, the application processes a single frame on average in 8.34 ± 2.12 seconds. When using an asynchronous approach, the user would wait for 8 seconds after pointing its smartphone at a painting, which is an acceptable delay. Note that the speed is highly dependent on the used hardware. In a real-world mobile application, communication with a server should be used to increase the matching speed. The average time to process each frame also depends on the amount of lines detected by the probabilistic Hough transformation and the amount of interesting keypoints found in the video. This can be an explanation for the difference of processing time between each video.

On average, the correct room is returned in $43.78 \pm 19.12\%$ of the frames in the videos, which does not look remarkably high. Notice that the accuracy between videos varies a lot, translating in a standard deviation of 19.12. However, this low and varying accuracy can be explained by looking at the qualitative performance of the application on the videos.

In all videos the location estimation starts of really strong. The first room is found quickly most of the time and when the user moves to the next room this is detected as well. For some time the application gets an accuracy of about 70% or better. In most videos however, something happens that causes the algorithm to lose track of the real path. In some videos the user moves too quickly through a room, causing the algorithm to miss this room. After skipping a room, the application still detects the correct room, but because this room cannot be connected to the current path, the algorithm is stuck in the first room. This continues until the end of the video, causing the accuracy to keep dropping. In other videos the application has wrongly detected a painting, which happens to be in a connected room. The path is then wrongly extended to

Figure 6: Database image of room 12, which the classifier predicts when looking at doors or windows.

that room. The user then proceeds to another room that cannot be connected to this path. If the application cannot detect enough paintings in that new room, the real path is again lost. A final case that happened in some videos was that the camera remained focused on some point where the application consistently detected a wrong room. This wrong room then got a higher score than the real path that had been found, causing the algorithm to replace the real path with this wrong room. This is especially problematic with doorways, where the application consistently mispredicted a rack from room 12, see Figure 6.

If these circumstances do not happen, the algorithm can keep track of the followed path. This can be seen from the result on video 6, where the global accuracy of the application is 79%. This result is very good, as getting an accuracy of 100% is almost impossible. The camera does not always have a painting in view when transiting from one room to another, making the application unable to immediately detect this transition and lowering the maximum possible accuracy.

These limitations of the algorithm were already predicted in §2.4. This application would get a higher global accuracy if the user would use it in a certain way. By moving more slowly through certain rooms and focusing the camera more on paintings and less on other things, it would be a lot easier for the algorithm to keep track of the followed path. After losing track of the real path, the algorithm is given the ability to recover. To do this however, the user has to fix the camera on a painting until the application has found the current room again. As this is not done in the videos, the algorithm remains stuck in the wrong room for a big part of these videos. This requested behavior of the user is not user friendly. In general use-cases the described limits of the current application will be crossed, causing bad performance.

However, the application does a very good job of tracking the location of the user when the described limits of the application are not crossed, even if the wrong room is detected in a majority of the frames.

4 Conclusion

In this paper we introduced an end-to-end application to track the location of the user in a museum, based on video material. This application exists out of three major parts. First, after the frames of the video footage have been preprocessed, the application tries to detect and extract a single painting from the image. Second, the application determines which painting this is by comparing the extracted painting to a database of labeled paintings. Finally, these detections are combined with the known layout of the museum to determine the most likely followed path by the user.

The initial goal of the project was met. An application was developed that can track the user in a museum through a video of their surroundings with some fidelity. While paintings are matched fairly reliably, incorrect matches can lead to an incorrect determination of the location. This can in turn disturb the detection of consecutive rooms, because the transition between those rooms may be impossible. If wrong detections do not pile up, the application can be up to 79% accurate in its determination of the users location. In comparison to previous work, the application is reasonably fast. When using it for real-time applications, a delay of 8.34 ± 2.12 seconds can be expected.

4.1 Future Work

In future work this application should be made more robust and user friendly. This can be done by improving three different components: the detection of the rectangles, the speed of the painting matching and the algorithm that determines the most likely path.

First the rectangle detector could be improved. When only one painting is in view, the used algorithm is very good in detecting it. However, when a multitude of paintings or other objects are in view, this algorithm finds weird rectangles. The rectangle detection algorithm thus has a lot of room for improvement, especially when multiple paintings, doorways, other people, etc. are in view.

In future work, the speed of this application should also be increased to allow real-time processing of the video. The slowest component of the application is the feature matching, which takes multiple seconds for each frame. To improve this, another way of comparing the detected features with those in the database should be used.

Finally, the algorithm to determine the followed path as we described in this paper, is not robust enough for general usage. To improve this algorithm some modifications could be made. Some of these modifications can be very simple, like asking the user to fix the camera on a painting in the current room when a large segment is replaced. Other modifications are more complex, like taking paths formed by previous detected rooms into account. When a new room is detected the algorithm could try to append it to the other paths as well. These other paths would make it easier for the algorithm to find the followed path after it has been lost. A final proposed modification would be to add the ability to detect when a room has been skipped and to add it to the most likely path.

With these proposed modifications, this application could become more robust and user friendly, allowing it to be used

in real life and in real time. However, even though these modifications can seem easy to add, doing this efficiently might not be. When adding these modifications, it should also be made sure that the performance of the algorithm is not weakened. The current algorithm increases the accuracy of the application by being strict. Easy implementations of these modifications however, could reduce this strictness and thus the performance as well.

References

[1] Bay, H., Fasel, B., and Van Gool, L. (2006a). Interactive museum guide: Fast and robust recognition of museum objects. In *Proceedings of the first international workshop on mobile vision*.

[2] Bay, H., Tuytelaars, T., and Van Gool, L. (2006b). Surf: Speeded up robust features. In *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg. Springer Berlin Heidelberg.

[3] Broderick, C. (2016). Robust recognition and identification of paintings using computer vision techniques. Master's thesis, University of Dublin.

[4] Bruns, E., Brombach, B., Zeidler, T., and Bimber, O. (2007). Enabling mobile phones to support large-scale museum guidance. *IEEE multimedia*, 14(2):16–25.

[5] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). Brief: Binary robust independent elementary features. In Daniilidis, K., Maragos, P., and Paragios, N., editors, *Computer Vision – ECCV 2010*, pages 778–792, Berlin, Heidelberg. Springer Berlin Heidelberg.

[6] Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698.

[7] Karami, E., Prasad, S., and Shehata, M. S. (2017). Image matching using sift, surf, BRIEF and ORB: performance comparison for distorted images. *CoRR*, abs/1710.02726.

[8] Kavalerov, I. (2014). Using pattern recognition to automatically crop framed art. https://artsy.github.io/blog/2014/09/24/using-pattern-recognition-to-automatically-crop-framed-art/. Accessed: 2019-05-29.

[9] Kiryati, N., Eldar, Y., and Bruckstein, A. (1991). A probabilistic hough transform. *Pattern Recognition*, 24(4):303 – 316.

[10] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2.

[11] Muja, M. and Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press.

[12] Nielsen, J. (2009). Powers of 10: Time scales in user experience. https://www.nngroup.com/articles/powers-of-10-time-scales-in-ux/. Accessed: 2019-05-29.

[13] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571.

[14] Ruf, B., Kokiopoulou, E., and Detyniecki, M. (2008). Mobile museum guide based on fast sift recognition. In *Adaptive Multimedia Retrieval*.

[15] Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition.

[16] Temmermans, F., Jansen, B., Deklerck, R., Schelkens, P., and Cornelis, J. (2011). The mobile museum guide: artwork recognition with eigenpaintings and surf. In *Proceedings of the 12th International Workshop on Image Analysis for Multimedia Interactive Services*.