Recognition based indoor positioning at MSK Ghent

Computervision project group 8, 2021-2022

Robbe Decorte
robbe.decorte@ugent.be

Bram De Bleecker
bram.debleecker@ugent.be

Lance Dewaele
lance.dewaele@ugent.be

Benoît D'Haene
benoit.dhaene@ugent.be

Lennert Steyaert
lennert.steyaert@ugent.be

***Abstract*—An indoor positioning program for the MSK Ghent was developed in this article. Videos made by a visitor of the MSK Ghent are used to determine the visitor's position and the taken path. From those videos, paintings are extracted and compared with a database. The unsupervised painting detection algorithm achieved an F1-score of 85% and an average intersection of union score of 89% (for the detected paintings) on a test dataset with 801 samples. To match the extracted paintings with the database one of three different matcher combinations can be used (keypoint-based matching (ORB), feature vector-based matching, or a combination of the previous two). The keypoint-based feature and vector-based matchers were extensively tested and benchmarked to select either an optimal amount of keypoints (in the case of ORB) or the kind of distance metric for the feature vector case. Based on the results we've chosen 100 keypoints and as distance metric cityblock and euclidean. The vector generation was executed using the VGG16 deep learning architecture. For the localization, a hidden Markov model was used to predict the changes of the user being in a certain room. Finally, these chances are visualized with colors on a floor plan of the MSK Ghent. The most likely used path is also shown on the floor plan.**

***Index Terms*—Computer vision, painting detection, unsupervised learning, ORB, hidden Markov model, image similarity, image preprocessing**

I. Introduction

Many techniques exist to determine the position of a user or a device relative to its environment. One possible example is visual positioning, which doesn't rely on communication-based techniques such as satellite navigation or ultra-wideband (UWB) localization. Visual positioning prevents the necessity of a direct line of sight for signal transmissions or the scattering of beacons inside the building, but it does require that the environment contains enough distinctive landmarks unique to its location. Live camera images are compared to a database of previously recorded images that include positional annotations of said landmarks. If the live image matches one of the database images with high certainty, the user is known to be at the location associated with this database image.

In this paper, we propose a visual positioning solution for indoor localization at the Museum voor Schone Kunsten (MSK) in Ghent (floor plan shown in figure 1). This proposal helps visitors by providing the current room number, but more importantly the museum curators themselves. It can give insights on frequently used routes in the museum and show which rooms are more popular based on the number of visitors and the average amount of time spent in that room. Each room contains a large number of recognizable landmarks, the paintings themselves (see figure 2).

Multiple videos of visitors walking around the museum are available. Besides that, we have access to a dataset that contains the paintings belonging to each room and a testing dataset that contains images with positional annotations of the paintings. Section II presents two image preprocessing steps to improve the overall accuracy and performance of the algorithm. The first technique is a way to sample frames based on how blurry the content is. It prevents the frames from entering other stages of the localization pipeline if needed. This section ends with the calibration procedure of the GoPro camera that was used to record parts of the dataset. The unsupervised painting detector is discussed in section III and includes benchmark performance and common causes of failure. Section IV explains the matching procedure against the database. All previous components come together in section V. By combining all parts and extending it with a hidden Markov model, it is possible to accurately determine the location and return the path taken to its current location.

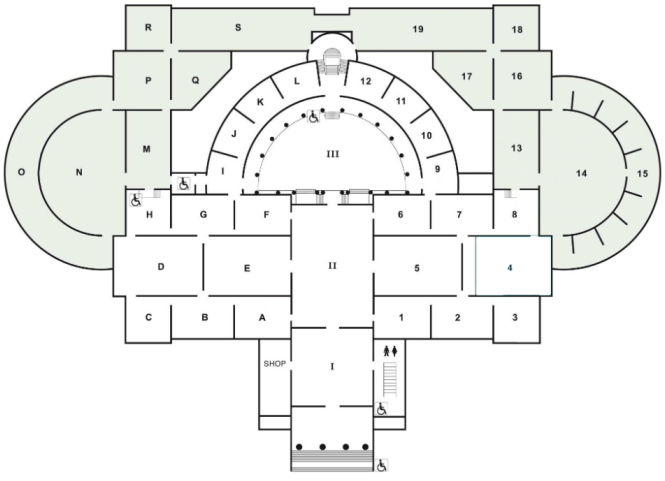


Fig. 1. Floor plan of Museum van Schone Kunsten Gent

Fig. 2. Sample image of the museum

II. Frame processing

A. Frame selection using a sharpness metric

Running the detection and matching algorithm on every frame in the video is unfeasible when realtime positioning is required because the time needed to complete every stage takes longer than the available time[1]. If this is ignored, the video stream would be playing in an extreme form of slow motion. Note that it is also possible that some frames may be blurry and thus are a complete waste of computational resources. Motion blur is caused by moving the camera around and severely hampers the matching of details (edges are less visible, strong keypoints become undetectable, etc.). The frames will most likely not be matched correctly and should be avoided. Image sharpness is closely related to acutance. It describes how quickly image information transitions at an edge, and so high acutance results in sharp transitions and detail with clearly defined borders [1]. In practice, only images with high acutance are wanted to feed the detection pipeline. That is why a procedure is needed to measure sharpness in an image.

One possible solution to handle this problem is to implement the algorithm proposed by Tong et al. [2]. The proposed scheme makes use of the Haar wavelet transform to decompose a given image into an approximation image and oriented detail coefficients (horizontal, vertical, and diagonal direction). Edges are generally classified based on how sudden the transition is. Immediate transitions can be seen as a Dirac-like structure while slower transitions are more like a skewed step function [2] [3]. Images normally contain all kinds of edge structures while the images affected by motion blur generally lack strong edges. The general technique to detect a blurry image is to classify and count different kinds of edges on consecutively decomposed images. For each decomposition level, an edge map is created based on the detail coefficients of that level. This results in a pyramid like structure (image dimensions of the approximation reduces every iteration) in which different edge types are counted. The image is classified as blurry if the ratio between Dirac edges and skewed step functions (with a given slope) exceeds a given threshold.

[1]Available time is based on the FPS of the video feed, in perfect circumstances we would match the original FPS with all calculation included.

B. Handling camera distortion

Parts of the videos included in the dataset are filmed with a GoPro camera. This camera contains a so-called fish eye lens. It causes an extreme wide-angle perspective that can be great for an immersive action look but not so much when applying detection techniques to the captured frames. The distortion is especially noticeable for straight lines in the frame. If the line is centered in the frame, it can remain quite straight. But because of the way the lens distortion works, the more the line is moved to the edges of the frame (top, bottom, left, right), the more curved the line will be.

This makes the detection of straight lines more difficult and two versions of the algorithm would be required. One for a *normal* camera and one for cameras with a fish eye lens. Therefore, we need to calibrate the GoPro camera so the distortion effects are reduced. This functionality is implemented in the following OpenCV functions:

1) Point correspondences between multiple frames are found by moving around a rectangular chessboard. The inner corners of the board are automatically detected using **cv2.findChessboardCorners**. Locations of found corners are refined with the **cv2.cornerSubPix** (integer to floating-point coordinates). These correspondences act as the input of the calibrateCamera function in the next step.
2) **cv2.calibrateCamera** returns the camera matrix, distortion coefficients, rotation and translation vectors (intrinsic and extrinsic camera parameters). This function implements the algorithm proposed by Zhengyou Zhang [4]. Results are persisted for each camera type.
3) **cv2.undistort** reduces the distortion effects. Note that the resulting image should be cropped using the valid pixel ROI metrics provided by **cv2.getOptimalNewCameraMatrix**.

III. Painting detection

The core of this work is the detection of paintings. For this purpose, an unsupervised segmentation algorithm was designed to extract all visible paintings in any arbitrary image. The essence of the algorithm is to associate straight lines that represent the enclosed region of a painting frame. For convenience, the detector is designed to find rectangular shapes when viewed head-on or quadrilateral shapes when viewed at a different angle.

A. Proposed algorithm

The first step in the detection pipeline is to construct an edge map of a given single-channel image using the Canny operator. Since edge detection algorithms are heavily impacted by noise, the first step should always be to reduce the effects of noise by applying a filter. The original paper by John Canny [5] suggested using a Wiener filter for optimally estimating the noise component of an image-noise two-component signal.

Since this requires knowledge of the noise spectrum it is much easier to apply a Gaussian smoothing filter. All incoming images are resized to a width of 500 pixels and filtered using a 9x9 Gaussian kernel with $\sigma = 1$. The image size is fixed to ensure consistent results throughout the dataset and to speed up the calculations. Images with higher resolution may cause suboptimal results of the smoothing kernel and would require a variable size of the kernel. The high and low threshold parameters of the Canny operator are calculated using the Otsu method [6] [7]. Its basic principle is to assign every pixel of the image to either foreground or background bucket and to calculate an optimal threshold value based on the intra-class variance. The resulting threshold value minimizes the sum of foreground and background spreads. The resulting edge map of figure 2 is shown in figure 3. Note that after applying the Canny operator, some detected edges are disconnected. The results can be improved by dilating the edge map using a 3x3 kernel.

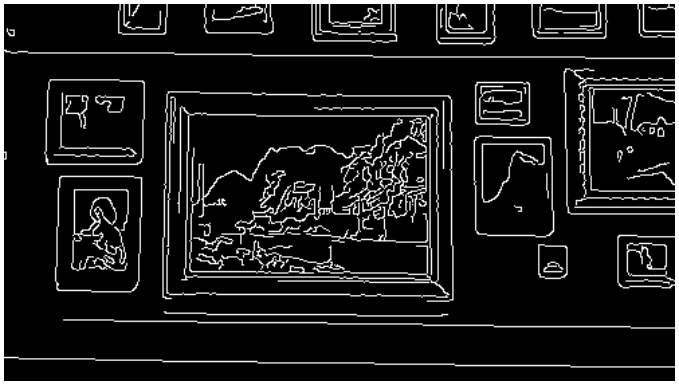


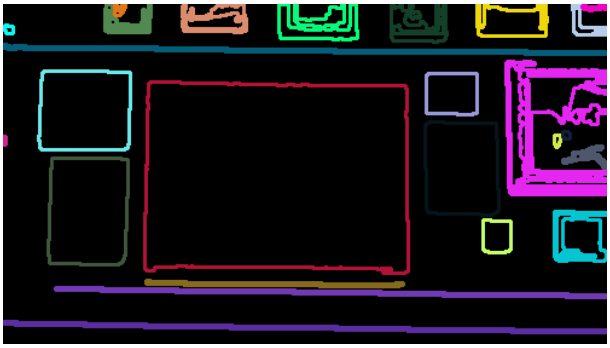
Fig. 3. Edgemap of figure 2

Cv2.findContours extracts contours using the binary edge map from the previous step. A contour is defined as a curve joining all the continuous points (along the boundary), having the same color or intensity [8]. Thus the result of this function is a list of groups. Each group consists of points (that are connected to each other) in the binary image. To exclude contours that lie fully within other contours the retrieval mode can be set to **cv2.RETR_EXTERNAL**. Contours are sorted by area and limited to the 25 largest to prevent small detected patches that are unusable by the matching algorithm anyway. The next step tries to select contours that are closely related to the shape of a painting frame (squares, rectangles, quadrilaterals, etc.). This implies that we first need to simplify each group of points to a general geometric shape so the corners can be assigned. First of all, the convex hull of the contour is generated and simplified using the **cv2.approxPolyDP** function to correct small errors in straight lines. Afterward, every contour may be considered a candidate painting if the simplified convex hull can be represented using four points (the corners of the quadrilateral) and its solidity is greater than 60% (see equation 1). The solidity check prevents the acceptance of contours that are an extreme mismatch compared to their respective convex hull.

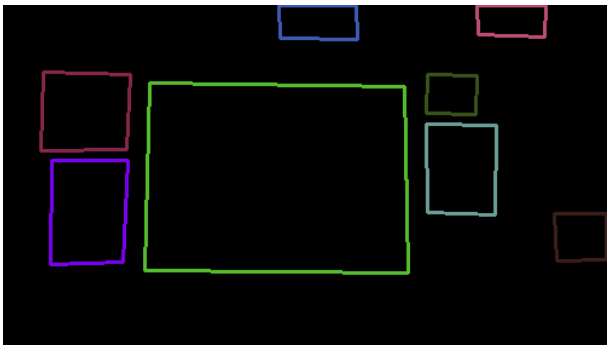
$$Solidity = \frac{Contour\ Area}{Convex\ Hull\ Area} \tag{1}$$

The final step in the algorithm is to rectify every contour that satisfies the previous criteria. This is easily done as the corners of the quadrilaterals are known. The matching algorithm and subsequent steps require the inputs to be rectangular to ensure optimal results.

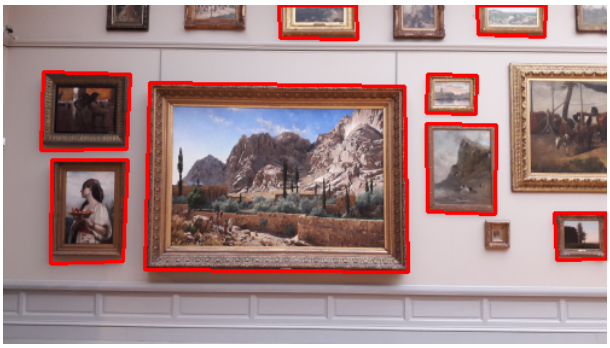
The rectified image crops are passed through the blur detection scheme from section II-A. Blurry or vague crops are far more likely to cause a miss classification in the matcher. This mainly occurs when paintings are detected with an acute viewing angle.



(a) All contours found on figure 2



(b) Filtered contours, amount of corners and solidity checks applied



(c) Contours drawn on the original image

Fig. 4. Visualization of the contour filtering procedure (every color represents a different contour)

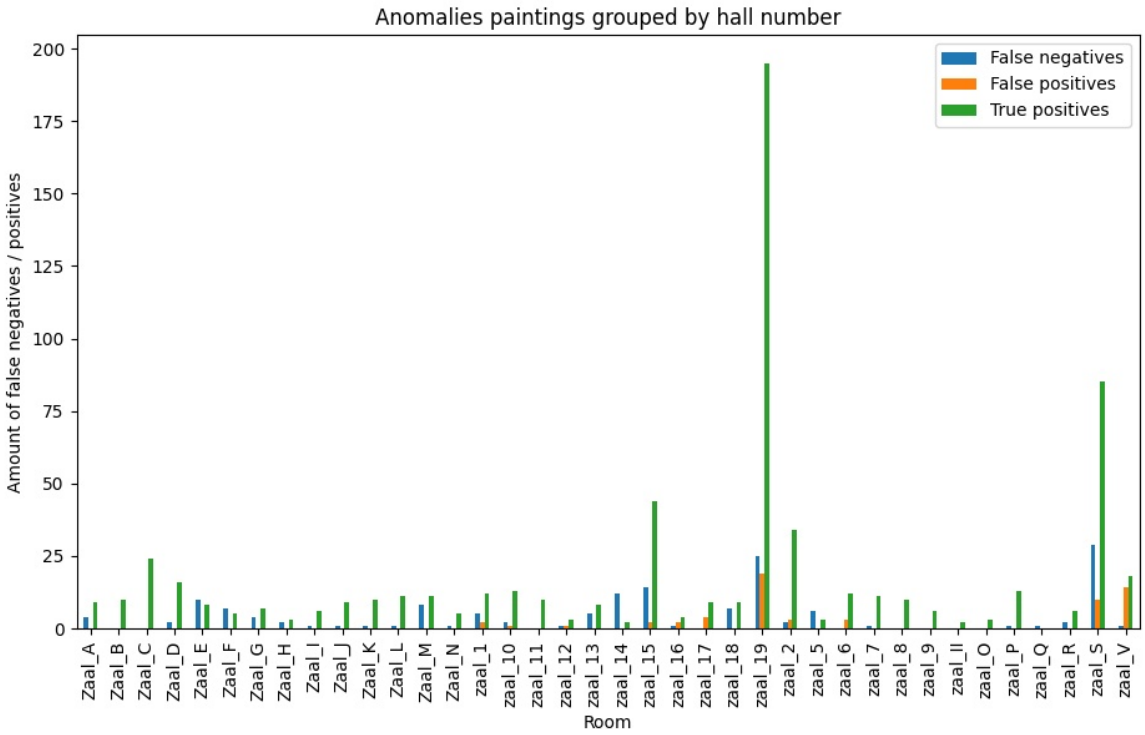


Fig. 5. Benchmark results displaying a histogram of true positives, true negatives and false negatives

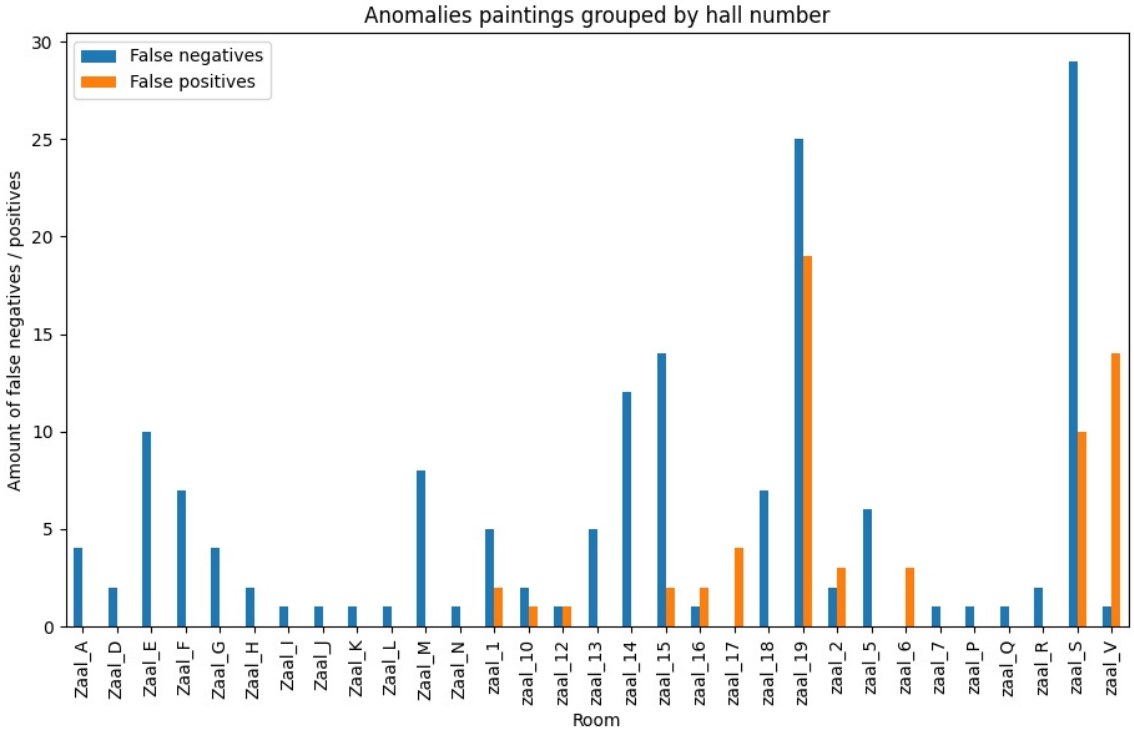


Fig. 6. Benchmark results with true Positives omitted

B. Benchmark results

The quality of the painting detector was assessed using a validation dataset and a CSV file that contains the corners of all paintings in the images in the validation set. The results are shown as a confusion matrix in table I and correspond to a $recall = 0.8065$, $precision = 0.9137$ and $F1 = 0.8568$. For the detected paintings (646 detected paintings for 801 paintings in the dataset), we achieved an average intersection over union score (IOU) of 0.8934. The distribution of IOU scores is displayed in figure 7 and is further elaborated on in section III-C.

TABLE I
CONFUSION MATRIX OF THE BENCHMARK DATASET

		Ground truth: Positive	Ground truth: Negative	Total
Predicted	Positive	646	61	707
	Negative	155		
	Total	801		801

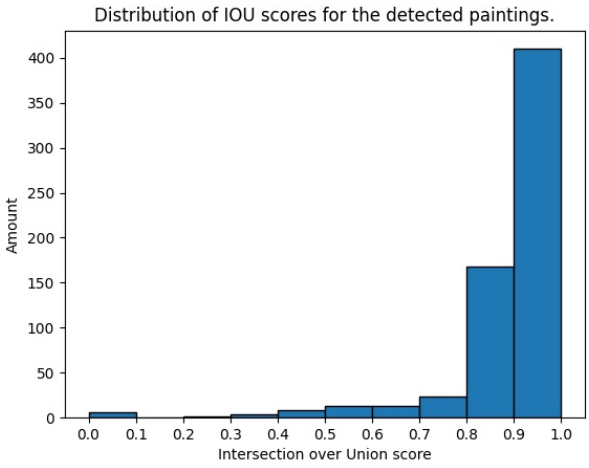


Fig. 7. Intersection over union distribution of the detected paintings in the benchmark set

C. Common causes of failure

Figure 8 shows one of the most common detection errors, the exclusion of the painting frame. These are detections that correspond with an IOU score between 0.75 and 0.95 (see figure 7). This doesn't pose too much of a problem since the actual painting information is preserved. However, all the frame information which may contain strong keypoints to match is lost. Those keypoints may be included in the database representation of the ground truth images.

Other common circumstances when a painting is not or less accurately detected are:

1) Painting frame is not completely visible in the image.
2) The difference between the painting background and the wall is not strong enough to be detected as a line.
3) Shadows caused by the painting frame commonly form a strong edge and are included in the painting contour. This causes a lower IOU score for the detection.
4) Next to most of the paintings hangs a small plaque (containing information) that is falsely classified as a painting when its color is different from the wall color.
5) Hall 14 is plastered with a wallpaper that contains quadrilateral shapes which interfere with the contours of the paintings and causes a very large amount of false negatives. This behavior can be verified using figure 6.

Fig. 8. One of the most common errors (the picture frame is not included in the detection), green shows the ground truth bounding box and red is the predicted one

IV. PAINTING MATCHING

The elaboration of the painting matching consists of two approaches which are discussed in IV-A and IV-B.

Realtime painting matching must have preprocessing, meaning extraction of keypoints, features and storing it into a database beforehand. The project code contains code to generate such a database. This database will be used to compare a particular painting frame with all it's contents.

A. Keypoint matching

The keypoint matching approach relies on Oriented Fast and Rotated BRIEF (ORB) feature detection. ORB is an alternative to SIFT and SURF. These algorithms are patented while ORB is not. [9]

According to reference [10] ORB is faster than SIFT or SURF. However, in general ORB has lower correct matching results than SIFT or SURF. [11] Considering the application, time is the more important factor.

Practically, two drawbacks have to be taken into account when using ORB. At first, ORB is not scale-invariant. Thus the detected images should be scaled back to the original size of the image painting stored in the database. For this application,

all images are scaled with a fixed width of 800. The second drawback is the influence due to shearing. Matching sheared images implies lower matching scores. [11] This particular drawback is countered by the rectification step.

Another practical issue due to blurry painting detection was established. Blurry images can cause lots of wrong matching results. This issue is mainly prevented by the wavelet filters as described in II-A.

In addition to the choice for ORB, the correct matching algorithm had to be determined. OpenCV provides two types of matching algorithms: Fast Library for Approximate Nearest Neighbors (FLANN)-based and brute-force (BF)-based. FLANN matching works faster than BF matcher in case of a large dataset (high amount of key points). Considering the acceptable results (discussed below) based on BF matching the FLANN version wasn't used. [12]

ORB relies on one important variable, the number of key points (features called in OpenCV). Changing this variable should result in a difference in the matching score, distances between matches, and execution time.

Table II shows us the positive and negative matches for each amount of keypoints. In total 437 images were evaluated, they were retrieved from the given dataset. As can be seen in table II only a slightly better score is reached when increasing the number of keypoints above 100. Based on this result the final solution is restricted to 100 keypoints. Only the 20 largest distances (keypoint distance) for a particular image are summed to obtain a distance value for each image in the database.

TABLE II
KEYPOINT/FEATURE VARIATION CORRECTNESS RESULTS

No. keypoints	Positive matches	Negative matches	Mean score (%)
50	402	35	91.99
100	423	14	96.80
200	424	13	97.03
300	425	12	97.24

Distance distributions for matches are also interesting to determine useful information about the matching. Figure 9 shows the distance distribution for the correct and incorrect matches. The correct distance matching distribution shows many outliers in the case of the matcher with only 50 keypoints. The amount of outliers decreases as the number of keypoints/features increases. The median for each scenario also decreases as the amount of features increases. It implies fewer high matching distances. The same findings are reflected in the case of incorrect matches. In addition, it can be determined that the distance range for the correct and incorrect matches shrinks as the number of keypoints grows. Table III confirms this fact based on the means.

Most of the wrong matched images have a strong intensity deviation or contain only dark colors. (figure 10 and figure 11) The effect of varying intensity is reviewed in [11]. SURF

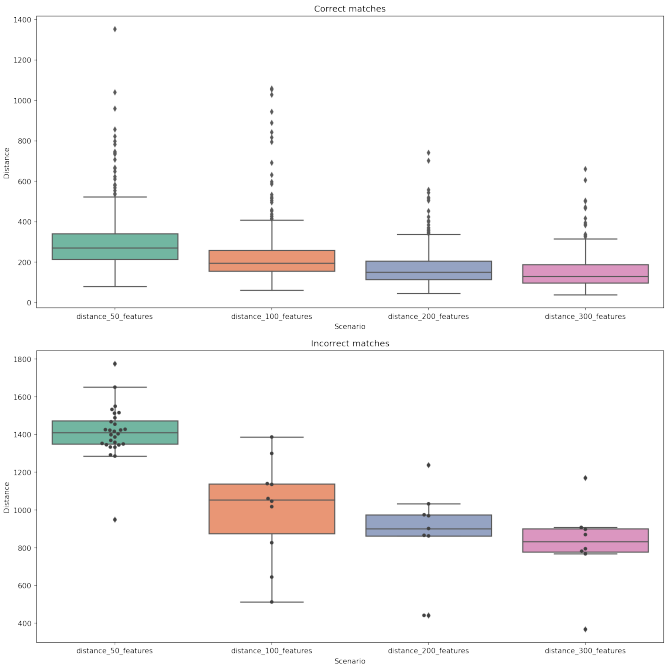


Fig. 9. Distance distribution matches (variation of keypoints)

TABLE III
DISTANCE DISTRIBUTION AVERAGE VALUES

No. keypoints	Mean	Mean match	Mean no match
50	374.98	302.66	1413.36
100	247.87	229.92	1007.40
200	187.71	173.47	858.89
300	164.64	152.30	819.88

and SIFT are less sensitive to this and will perform better. Nonetheless, this issue does not cause much trouble when using the videos.

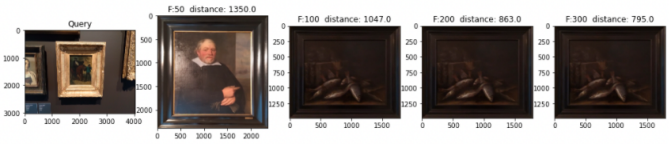


Fig. 10. Incorrect matching results due to darkness

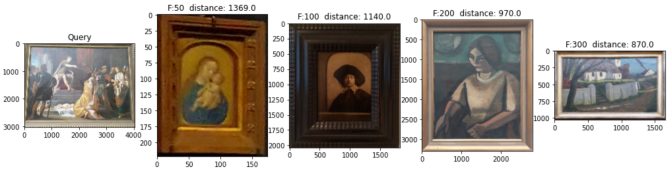


Fig. 11. Incorrect matching results due too much light

Distances between first and second matches may reflect information about the quality of a particular match. Figure

12 confirms this assumption, the distances between the first and second matches are remarkably lower in case of incorrect matches.

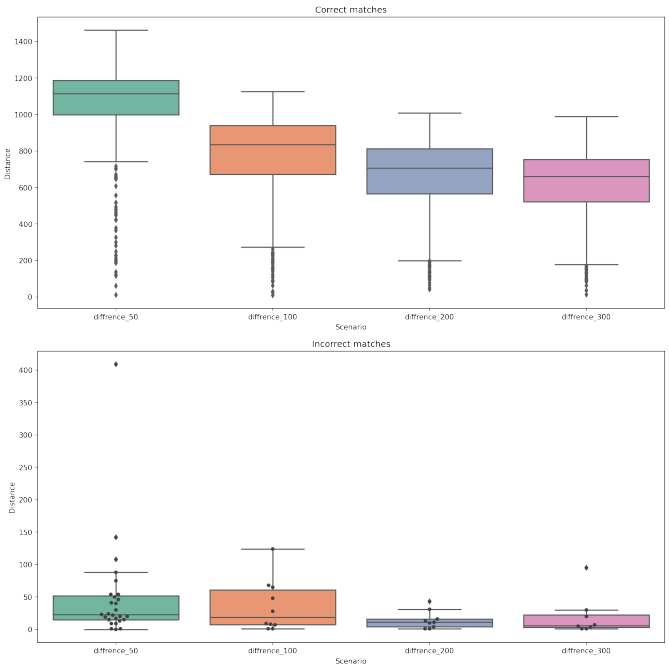


Fig. 12. Difference in distance between the first and second matching result

Earlier the chosen amount of keypoints/features was made on the matching score. In addition, the time to match one image is also a decisive factor. Table IV illustrates the average time to match an image to the database. The matcher with 200 keypoints/features needs a double amount of time compared to the matcher with only 100 keypoints/features. Because of the realtime requirements, the choice for 100 keypoints is obvious.

TABLE IV
KEYPOINT/FEATURE VARIATION COMPUTE TIME

No. of keypoints	Compute time (ms)
50	60
100	107
200	238
300	353

B. Feature vector matching

Our approach for feature vector matching relies on feature extracting with a pre-trained model. The chosen model is VGG16 with weights from imagenet. Such a model provides interesting feature generation which is convenient for applying distance measurements [13].

Figure 13 illustrates all layers. The red square around the fully connected layer represents the prediction layer. This (dense) layer was cut off to obtain the feature vectors.

Various distance metrics are applicable to obtain a distance result between the feature vectors. The following common metrics were evaluated: euclidean, cityblock, Minkowski, Chebyshev, and cosine. Table V shows the matching results on the same evaluation dataset in IV-A. Euclidean seems to have the highest positive ratio. The cityblock and cosine metrics also have acceptable scores. Only Chebyshev scores substandard.

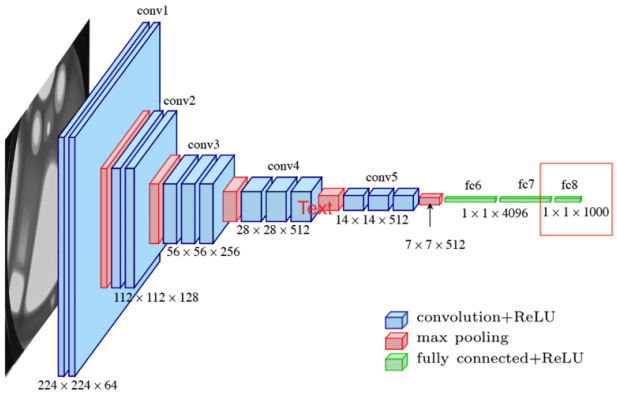


Fig. 13. VGG16 layer visualization [14]

TABLE V
DISTANCE METRIC VARIATION CORRECTNESS RESULTS

Metric	Positive matches	Negative matches	Mean score (%)
euclidean	363	74	83.07
cityblock	358	79	81.92
chebyshev	315	122	72.08
cosine	360	77	82.38

Each distance metric results in another range of distance values. (Table VI) Comparing the correct matches distribution with the incorrect matches, an overlapping region is established for each distance metric.

TABLE VI
DISTANCE DISTRIBUTION AVERAGE VALUES

Metric	Mean	Mean match	Mean no match
euclidean	49.70	302.66	1413.36
cityblock	1420.49	1371.98	1640.33
chebyshev	5.22	5.16	5.38
cosine	0.16	0.14	0.24

Despite this property, differences between the first and second matches are still useful. The mean values for the differences between the first and second matches are shown in table VII.

Next to these distance and matching scores, time should also be taken into account when choosing a suitable distance metric. Table VIII illustrates the average time to match an image upon the database. The matcher using the cityblock metric seems to have the fastest execution, followed by euclidean and cosine. The chebyshev version scores substandard again.

TABLE VII
DIFFERENCE IN DISTANCE DISTRIBUTION AVERAGE VALUES

Metric	Mean match	Mean no match
euclidean	11.64	1.78
cityblock	356.85	54.80
chebyshev	1.02	0.22
cosine	0.08	0.02

Based on the collected results euclidean and cityblock are the better choices. Both distance metrics are used to demonstrate the effects on the videos.

TABLE VIII
COMPUTE TIME DISTANCE METRIC DEPENDENT

Distance metric	Compute time (ms)
euclidean	69.60
cityblock	54.87
chebyshev	221.30
cosine	77.15

C. Matching evaluation

All collected metrics for both matching approaches result in three final matchers which can be used in the realtime application: the keypoint-based matcher with 100 keypoints/features, the feature vector-based matching with euclidean distance metric, and the feature vector-based matching with cityblock distance metric.

The feature vector-based matchers have a faster compute time than the keypoint-based matcher while the keypoint-based matcher is more reliable in terms of correct matching. Because of this reason next to the implementations where only one kind of matcher is used, we also added a combination of a feature vector and a keypoint-based matcher.

This implementation takes advantage of the speed of the feature vector-based matcher. The matcher returns the first 50 images that may be a match. Next, the keypoint-based matcher only uses the 50 images which were returned by the feature vector-based matcher. The matching time for the keypoint-matcher is strongly reduced because only 50 images are considered instead of the whole database.

In general, one can conclude that the matching works well. It scores high when evaluating the dataset. In practice, as quoted before, blurry images and variant intensity caused by video usage still have an influence that cannot be ignored.

V. LOCALISATION

Now that we can extract a matching score from the paintings, we can use this to deduce the room we're currently in. If we only use the matching score our room prediction would be very heavily influenced by possible mismatches. This is why we opted for the use of a hidden Markov model (HMM).

A. Hidden Markov Model

A hidden Markov model is a statistical model that can combine the matching score with spatio-temporal features to become a room prediction. This is perfect for our assignment because, as stated previously, a prediction that's only based on the matching score would be very dependent on the quality of the matching algorithm which would make it very susceptible to mismatches.

1) Architecture: The underlying architecture of a HMM consists of 2 layers, a hidden layer and a layer with observed variables. The hidden layer consists of the transition probabilities between different states. The observed variables are observations that belong to a certain state with a certain probability. In this project, these "states" represent the different rooms, while the transition probability stands for the probability that a person stays in the current room or goes from the current room to another room. The processed video frames are the observed variables, where the probability of this frame belonging to a certain state/room is calculated by using the matching score of multiple matches. An example of the hidden Markov model architecture is shown in figure 14.

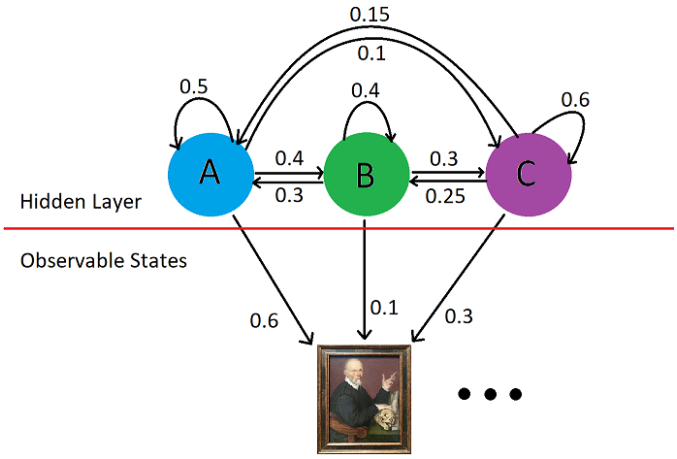


Fig. 14. Hidden Markov Model architecture

2) Calculating the transition probabilities: The amount of rooms between the start and destination room is used as a metric to calculate the transition probabilities between the rooms. If there are more rooms between the two rooms, the probability of the next frame being in the destination room would be lower than when the start and destination room would be adjacent. The reasoning behind this is pretty straightforward. It's simply very unlikely that a person goes from a certain room to the other side of the museum in a matter of seconds. To apply this in our program, we first need to know the number of rooms that are between the start and destination room. Note that we only have the floor plan of the museum to start with. Starting from this floor plan, an undirected graph was created where the connected nodes represent the adjacent rooms. After creating this graph, we extract the adjacency matrix to become a matrix structure

that shows the adjacent rooms. From this adjacency matrix, we can calculate the distance matrix by using the Floyd Warshall algorithm [15], which is an algorithm that's used to calculate the optimal path between all the different nodes in a graph. Now that the distance matrix is calculated, a distribution has to be applied to the rows which makes sure that the shortest distances have the highest probability, while the longest distances have the lowest probability. For this project, a linear and a Gaussian distribution were created and after extensive testing, the Gaussian distribution with $\mu = 0$ and $\sigma = 1$ was found to be the best performing distribution.

The process to create the transition probabilities is visualized below for the example in fig. 14 where we assume that the rooms are in a straight line (A - B - C). We start from the connectivity matrix, to become the distance matrix and end up with the probability matrix after applying the Gaussian distribution.

$$\begin{array}{c} \\ A \\ B \\ C \end{array}\!\!\begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \end{array} \Rightarrow \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0,58 & 0.35 & 0.07 \\ 0.27 & 0.46 & 0.27 \\ 0.07 & 0.35 & 0.58 \end{bmatrix}$$

3) Calculating the probabilities for the observable variables: Next up, the probabilities for the observable variables have to be calculated. The observable variables in the HMM are the frames that are effectively processed and matched against the database. The probabilities are calculated based on the matching scores (distance with ORB) from the extracted contours. The algorithm goes as follows: For every extracted contour, the best matching score (lowest = best) is taken for each possible room based on the list of soft matches from that contour. The room scores for every room are then added together. Based on this list of scores for every room, the probability is calculated so the lowest score gets the highest probability and similar scores get similar probabilities. To reduce the effect of mismatches, the probability of the previous room is set to 2% if there are no soft matches for that room.

4) Making the room prediction: To make the room prediction, the transition probabilities are combined with the probabilities for the observable variables. For every room, the probability is calculated by using the forward algorithm [16] [17], which is a dynamic algorithm that uses the following formula to calculate the probability of the frame being in a certain room $\alpha_t(x_t)$.

$$\alpha_t(x_t) = p(y_t|x_t) \sum_{x_{t-1}} p(x_t|x_{t-1})\alpha_{t-1}(x_{t-1})$$

This formula simply calculates the chance that the current frame is in a certain room, based on the probabilities for each room in the previous frame $p(x_t|x_{t-1})\alpha_{t-1}(x_{t-1})$. The room probabilities based on the matching score alone are represented as $p(y_t|x_t)$. When processing the first frame no previous prediction exists. Therefore, the stationary distribution π is used for the first frame. The stationary distribution represents the row vector such that $\pi = \pi P$, where P represents the hidden layers of the model. By using this, we become starting probabilities where the rooms with more neighbors get a higher probability.

After applying the forward algorithm, the location prediction is obtained by simply selecting the room with the highest probability.

B. Hidden Markov Model visualization

The room predictions from the hidden Markov model are extracted and used to color-code a floor plan of MSK Ghent (see figure 15). Every room is assigned to a color between red and green where the colors correspond to a low, respectively high percentage of currently being in that room. The top three predictions of the most likely rooms are also displayed in the bottom-left part of the image with the prediction percentage. The rooms with the highest percentages in time are also connected with lines to show the traveled path on the floor plan of MSK Ghent.

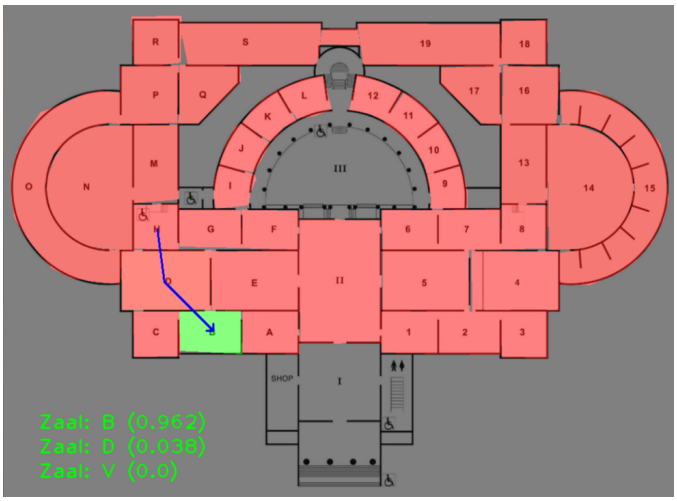


Fig. 15. Hidden Markov Model visualization

CONCLUSION

The final process is visualized in figure 16. The sequence of all steps ensures a working implementation.

First, the sharpness of the frame is calculated. When the sharpness is too low and doesn't satisfy the defined threshold, the frame is discarded. When the sharpness does satisfy the threshold, the frame is used to detect the painting contours. The detected painting contours are also checked. When the contours don't meet all the conditions, the frame is discarded. When the conditions are met, the painting is extracted, cropped, and rectified to get the frontal view.

After these steps, the matching itself starts. This is done with a combination of keypoint matching and feature vector matching. The best 50 matches from the database are sought. From those matches, the room probabilities (probabilities of

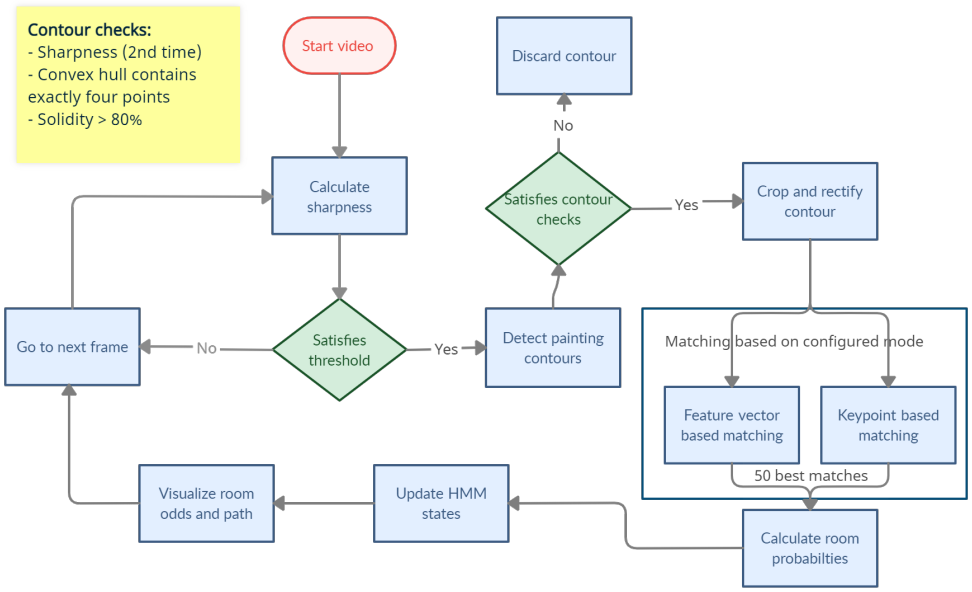


Fig. 16. Block diagram of the complete pipeline

the painting in a certain room) are calculated. The hidden Markov model then calculates the final room probabilities holding into account the location of the paintings in the previous frame.

The predictions are then visualized with color on the floor plan of the MSK Ghent. Finally, the path most likely used by the user is also drawn on the floor plan.

When using the techniques on the videos, room for improvement can be determined (wrong matches, incorrect detections, etc.). Although remarkable good results are obtained. Next to improvements related to detection and wrong matching an optimization of the time each frame needs to compute may also be an improvement. This may be solved with multithreading. Nevertheless, the final implementation provides a good proof of concept considering various algorithms and techniques.

REFERENCES

[1] "Tutorials: Sharpness." [Online]. Available: https://www.cambridgeincolour.com/tutorials/sharpness.htm

[2] H. Tong, M. Li, H. Zhang, and C. Zhang, "Blur detection for digital images using wavelet transform," in *2004 IEEE International Conference on Multimedia and Expo (ICME) (IEEE Cat. No.04TH8763)*, vol. 1, 2004, pp. 17–20 Vol.1.

[3] Y. Tang, L. Yang, and L. Feng, "Characterization and detection of edges by lipschitz exponents and masw wavelet transform," in *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No.98EX170)*, vol. 2, 1998, pp. 1572–1574 vol.2.

[4] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[5] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[6] M. Fang, G. Yue, and Q. Yu, "The study on an application of otsu method in canny operator," 01 2009.

[7] A. Greensted, "The lab book pages." [Online]. Available: http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html

[8] "Contour approximation method." [Online]. Available: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html

[9] "Orb (oriented fast and rotated brief)." [Online]. Available: https://docs.opencv.org/3.4/d1/d89/tutorial_py _orb.html

[10] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571.

[11] E. Karami, S. Prasad, and M. S. Shehata, "Image matching using sift, surf, BRIEF and ORB: performance comparison for distorted images," *CoRR*, vol. abs/1710.02726, 2017. [Online]. Available: http://arxiv.org/abs/1710.02726

[12] "Feature matching." [Online]. Available: https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html

[13] "Feature extraction and reverse image search." [Online]. Available: https://github.com/ml4a/ml4a/blob/master/examples/info_retrieval/image-search.ipynb

[14] "Automatic localization of casting defects with convolutional neural networks." [Online]. Available: https://www.researchgate.net/figure/Fig-A1-The-standard-VGG-16-network-architecture-as-proposed-in-32-Note-that-only_fig3_322512435

[15] I. H. Toroslu, "Improving the floyd-warshall all pairs shortest paths algorithm," 2021. [Online]. Available: https://arxiv.org/abs/2109.01872

[16] P. Blunsom, "Hidden markov models," *Lecture notes, August*, vol. 15, no. 18-19, p. 48, 2004.

[17] A. Ahmadian Ramaki, A. Rasoolzadegan, and A. Javan Jafari, "A systematic review on intrusion detection based on the hidden markov model," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 11, no. 3, pp. 111–134, 2018.