(Sm)Art Path: indoor localization through painting recognition

Authors: Mateo Van Damme, Jonas Neirynck, Ellen Mullie, Ahmad Alkaddor
Mentors: David Van Hamme, Sanne Roegiers

***Abstract*—In order to provide added value for visitors of the Museum of Fine Arts (MSK) in Ghent, a localization estimation algorithm could be useful. This could make it possible to provide additional information about paintings that are currently in front of the visitor. Visual positioning is a potential approach to tackle this problem. It doesn't need any external infrastructure or transmitters to make a location estimation, unlike other alternatives, and is therefore a good candidate. This paper will describe the implementation and evaluation of such a system. Using frames of a video stream, the location of the user will be estimated. First, paintings will have to be detected on an image. Then, these paintings have to be compared and matched with the ones available in a database. In a final step, the information about these matches is used to estimate the current location of the user. The results of this system indicate that the painting detection achieves an accuracy of 49%. The painting matching obtains an accuracy of 69%. While the system is capable of establishing the general route of a visitor, more research is needed to improve its robustness.**

***Index Terms*—indoor localization, image recognition, image matching**

I. Introduction

Various techniques exist to determine the indoor position of a person. WiFi, Bluetooth and Ultra-wideband (UWB) could be used to estimate the current location. However, these are not the most practical solutions, as these systems require active transmitters to be installed. Visual positioning is an alternative approach to achieve this goal. In this technique, live images are compared to database images with known locations, resulting in an estimation of the current location. Museums are ideal test grounds for this technique, as they have lots of unique and well documented landmarks.

The main objective of this paper is the implementation of an indoor positioning algorithm, based on painting recognition in the MSK in Ghent. Starting from a video, the idea is to extract paintings from consecutive video frames, compare these extracted paintings with the ones available in a database, and subsequently make an estimation of the rooms that are being visited, based on matches found in the database.

The first part of the algorithm covers the extraction of paintings from images. This involves the use of a Canny edge detector [1], which is used as input for a contour method. To find these contours, Suzuki's method is applied [2]. The result of this procedure returns polygons, of which only those that have four corners - considering the rectangle shape of a painting - qualify. As a final step, a transformation is applied to the detected polygons in order to obtain rectangular paintings.

The second part of the system includes a matching step to compare images with those in a database. The extracted paintings are first resized, after which they are compared to the paintings available in the database. This comparison is done by a Brute-Force (BF) matching, based on the features extracted by a Scale-Invariant Feature Transform (SIFT) detector [3]. This detector is used because it is considered to be resilient to common image transformations, such as translation, rotation and scaling [4–6]. Previous work shows that SIFT is amongst the top performers within the category of interest region descriptors, because of its robustness and distinctive character [7].

The last part of the algorithm focuses on the localization and positioning aspect. The matched paintings from the database in the previous step contain information about the room they are in. However, it must be taken into account that incorrectly matched paintings may occur. This component must therefore handle false matches and ensure that transitions between different rooms are somewhat realistic.

II. Recognition based localization

This section describes the three major components of the system: painting detection, painting matching and localization. Figure 1 gives an overview of how this system operates.

Detection is used to retrieve paintings from images. These paintings can then be compared with paintings from the database, based on extracted features. The localization step then establishes a path of rooms that have been traversed.

A. Painting detection

The first step towards localization is the detection of paintings. Broderick and Dawson-Howe [8] proposes a painting detection process in five steps:

1) A pre-processing step involving Gaussian blur
2) Canny edge detection
3) Dilation
4) Finding contours
5) Filtering contours

The painting detection process proposed in this paper has a large overlap with this method, it uses the same last four steps and adds one new step. This last step is inherent to the application of this paper, as the objective is to analyze video frames. This is somewhat different compared to using static images of specific paintings. By using video frames, paintings are shot from an angle, meaning a homography has to be applied to get the original shape of the painting. The process proposed in this paper has the following five steps:

1) Canny edge detection

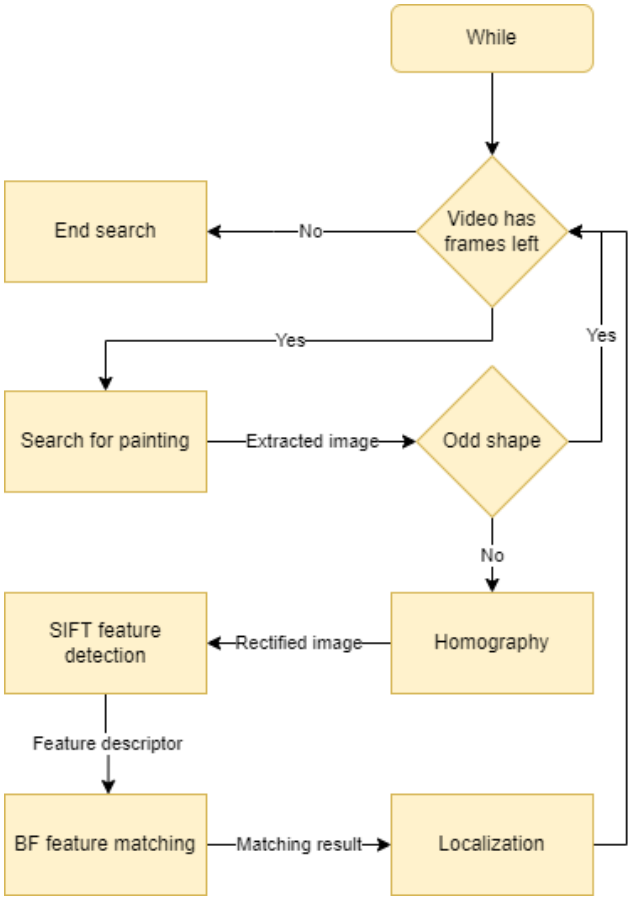
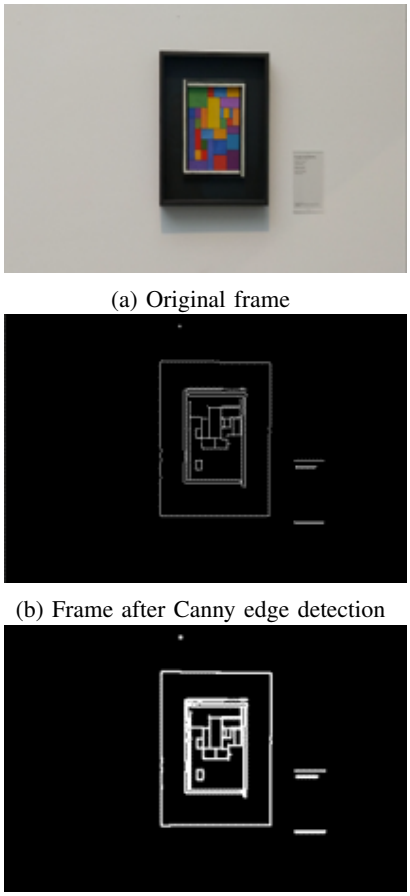


Fig. 1: Flowchart of the system



(a) Original frame

(b) Frame after Canny edge detection

(c) Frame after dilation

Fig. 2: Example of Canny edge detection and dilation

2) Dilation
3) Finding contours
4) Filtering contours
5) Homography

All steps are explained in detail in the rest of this section.

1) Canny edge detection: The different frames of a video fragment are checked one by one for the presence of paintings. The characteristics of those painting can be helpful in this process. For example, a painting will often have a frame. It can be assumed that this frame will provide a sharp transition with the surroundings. Therefore, a Canny edge detector is used to find these sharp edges. The Canny edge detector takes a gray scale image as input and classifies pixels as edges when their intensity changes sharply in relation to their neighboring pixels. The output is a black image with the edge pixels colored white [8]. An example of this output is shown in Figure 2(b)

2) Dilation: When looking at results of the Canny edge detection, it is noticeable that some edges are not consistent straight lines, resulting in less rectangular shaped contours. Therefore, dilation is used to fill in the gaps between pieces of lines and make one straight line possible [8]. The result of dilation on detected edges is shown in Figure 2(c).

3) Finding contours: Once these dilated edges are found, they can be used to search for contours. Contours connect continuous points with the same intensity or color [8] and are found with Suzuki's method [2].

4) Filtering contours: It is possible that several contours are found in each image. Therefore, the results are sorted by area, with paintings expected to represent the largest areas. Also, since paintings are expected to have a rectangular shape, only contours with 4 points will be considered. The algorithm eventually proceeds with the largest contour containing 4 points.

At this point the contour can have any shape as long as it had 4 corners, consequently weird shapes can be extracted when they are the contour with the largest area and 4 corners, two examples of this phenomenon are shown in Figure 3. The result of rectifying such shape is a very warped image, example shown in Figure 4, and can be confusing to matching algorithms. Therefore, two checks are implemented to ensure the multitude of weird shaped contours is ignored: checking if the diagonals of the shape are longer than the sides and checking if the two points not included in the diagonal lie on a different side of that diagonal. To define the diagonals in a random 4 corner shape, those 4 corners are first defined and given a label: top left (TL), top right (TR), bottom left (BL) and bottom right (BR). These labels are given intuitively, the most left point of the upper two points is the top left point, the other the top right point and so on. The diagonals are the two lines between corners TL and BR and between TR and BL, it

is possible that these are part of the lines of the contour, see the middle painting in Figure 3. When applying these checks on the three examples of Figure 3 only the right painting is classified as a possible rectangle. The left painting has one very short diagonal and points BL and TR lie on the same side of a diagonal, thus it fails on both checks. The middle painting has two sides that are longer than the BL-TR diagonal and fails only this test.

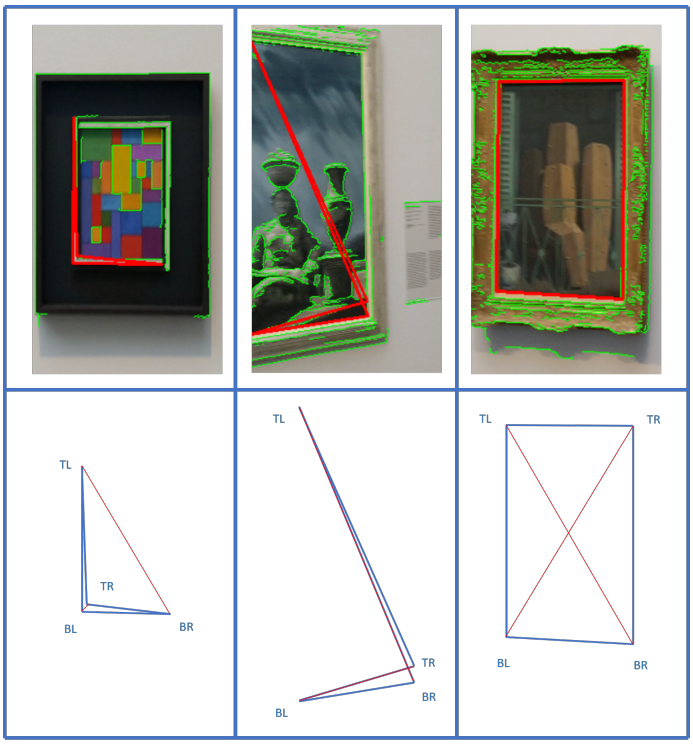


Fig. 3: Examples of the contours found with 4 points: top of the figure shows the contours found on three paintings, the lower part shows those contours in blue with labels for their corners and red lines for their diagonals

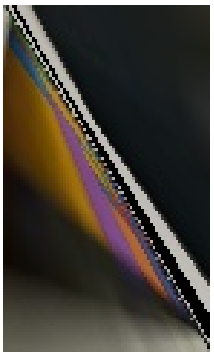


Fig. 4: Example of a warped painting due to the rectifying of a contour with a weird shape

5) Homography: After filtering the contours for unwanted shapes, a homography is then applied to this contour, in order to obtain a rectified painting. This rectified version can be used as input for the matching stage.

Extra: wide-angle lenses: Images captured with a wide-angle or fish-eye lenses capture a larger field of view than *normal* lenses. However, with the larger view comes the introduction of significant distortion, resulting in a pixels shifting away from the optical center. This wide-angle lens distortion is a type of radial distortion called barrel distortion. This distortion could make the detection of paintings harder, as it is possible that they no longer have straight borders in the distorted image, as shown in Figure 5(a). To transform the distorted image to a more normal looking one, the intrinsic parameters of the fish-eye lens need to be determined with a calibration method [9]. After finding the intrinsic parameters, the images can be transformed to less distorted images, as shown in Figure 5(b). When dealing with a video made with a fish-eye lens, like the GoPro camera in this experiment, there is an extra step in the painting detection that transforms the frame to an undistorted image. This will then become the first step in the painting detection.

(a) A video capture from a gopro camera (b) The same video capture un-distored

Fig. 5: Undistorting images from a GoPro camera

B. Feature matching

To enable feature matching, keypoints and descriptors of all images in the database are calculated using SIFT. This is done once during an initial run of the application. This way, these calculations do not have to be repeated every time. After all, the images in the database won't change.

When a new painting is detected by the previous step, the keypoints and descriptors are also calculated for this image. These features can then be used to search for a match in the database. This matching is performed by a BF matcher, which uses a distance metric to determine how related descriptors of two images are [10].

As successive frames of the video often contain the same painting, the extracted painting is not immediately compared against the entire database. The painting that is considered to be the best match of the previous frame is the first painting to be considered a potential match. If there are too few similarities between these two paintings, all paintings in the database will be taken into consideration to perform a match.

C. Localization

Figure 6 shows a visual representation of the localization algorithm.

Step 1

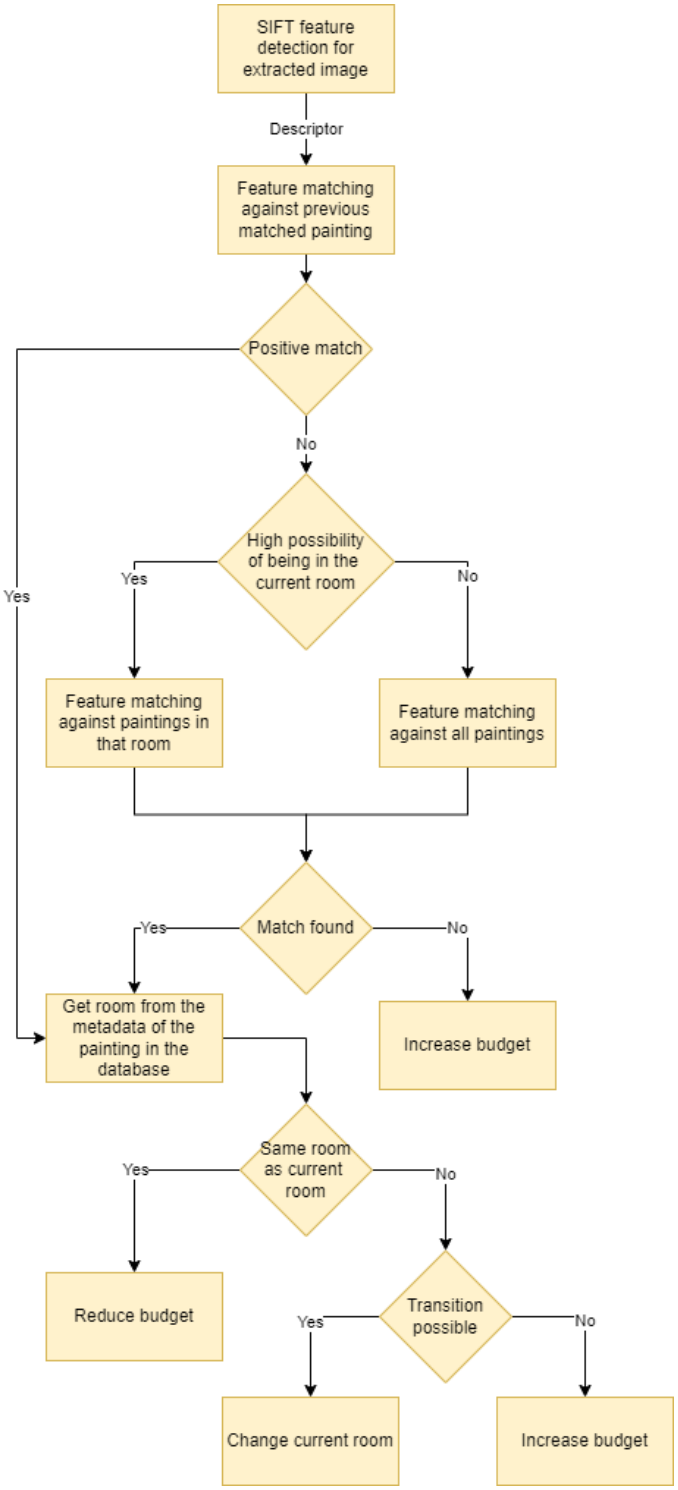


Fig. 6: Localization algorithm

The starting point is an extracted image from a video frame. The descriptor of this image is matched against the descriptor of the previous matched painting. If the matching score of the BF matcher gives an indication that these could be the same, the algorithm proceeds to step 3. Otherwise, the painting is matched against other paintings in the database. If there is a high probability that the user is still in the current room, based on the budget, only those paintings in that room are taken into consideration. If not, the matching is performed against all available paintings.

Step 2

If a match is found with a reasonable probability, the algorithm advances to step 3. If no match is found, the budget is increased.

Step 3

Based on the found match, the room can be extracted from the metadata of the image in the database. This room is compared against the current room. If the room of the found painting is the same as the current one, the budget is reduced. When these rooms are not the same, a transition to the new room is considered. If the budget allows it, a transition is made to that new room, which will replace the current room. The cost is deducted from the budget to reduce the possibility of a new transition after one just occurred. In the other case, when there is not enough budget to make the transition, the budget is increased.

The parameters for the different costs were determined using a combination of trial and error and evaluating the algorithm with all provided videos to avoid overfitting. See Table I.

TABLE I: Budget parameters

Name	Value	Description
costLow	1.0	Cost to go to an adjacent room
costHigh	30.0	Cost to go to a room that isn't adjacent
minBudget	0.5	Minimum budget allowed
maxBudget	31	Maximum budget allowed

The current values lead to the following behavior: When the budget is at its lowest, four matches with paintings from another room are needed to make the budget rise above the costLow threshold. If another incorrect match is registered after this and the matched painting is in an adjacent room, the locator will transition to that room. The same logic can be applied to switching to a room that isn't adjacent, but in this case, 23 consecutive matches in another room are needed when starting from the minimum budget.

Budget

A match with a painting in a room that differs from the current one can indicate two things: (1) the person shooting the video has changed rooms, or (2) the match is wrong. In case the match is wrong, the algorithm should not change to that room. To prevent this, a budget is used. Transitions between rooms have a cost, so in order to change rooms, the budget has to be high enough. If more paintings are found in the same room, the budget will decrease. It is expected that the next painting is also in that room, making a transition less likely. When a wrong match is made with a painting from another room, the budget will be too low to make that transition. The

localization component won't notice the wrong match and will stay in the same room, but at the same time it will raise the budget.

However, when the match is valid, it should be possible to change rooms. For that reason, when a match is made with a painting from a room that differs from the current one, the budget is increased. Once enough matches are made with paintings from another room, the budget will be high enough. Since adjacent rooms have a lower cost in comparison to rooms that are further away, transitions to neighbouring rooms are made faster than to more distant rooms.

III. Evaluation set-up

This sections describes the tests that are carried out to evaluate the system. When the program ends it prints out the traversed path. By comparing this to the correct solution the algorithm can be evaluated on its accuracy.

In Table II we show the traversed path and correct path for the first 5 videos.

TABLE II: Route evaluation

Video	Found Path	Solution
MSK_01	1, 2, 5, 7, 6, 8, 19, 13, 17, 15, 14, 15	1, 2, 5, 7, 6, 7, 8, 13, 16, 17, 15, 14, 15
MSK_02	V, S, R, Q, P, M, P, N, M, H	V, S, R, Q, P, N, M, H
MSK_03	H, D, B, A, B, D, G, F	H, D, B, A, B, E, D, G, F
MSK_04	9, 10, 11, F, K, J, I, F	9, 10, 11, 12, L, K, J, I, F
MSK_05	1, 2, 5, 7, 6, 7, 8, 13, 19, 17, 13, 17, 18	1, 2, 5, 7, 6, 7, 8, 13, 17, 18
MSK_06	A, B, D, E, G, F	A, B, C, D, E, G, F

A. DataSet

Before starting the evaluation, it would be useful to analyze the data used for matching. The dataset provided consists of 801 images of all art items in the museum. To speed up the working of the program, the images are pre-processed. Pre-processing the images involves four stages:

1) Reading the images in grayscale. Grayscale images rather than color images are generally used to get SIFT descriptors in order to reduce the complexity. The regions which have a similar grayscale level but different hues tend to produce wrong matching results in this case. Therefore, the loss of color information may result in decreasing of matching ratio.
2) Since the images extracted while filming in the MSK are blurry (moving camera), the images from the provided dataset are also blurred while pre-processing, this will help to find the matches. Blurring the images also helps to reduce the noise, thus only the relevant information like the shape and edges remain.
3) In the third step, the images are resized: width = 400 and height = 400 $*$ $\frac{height}{width}$.
4) The last step of pre-processing is calculating the key-points and the descriptors of the image.

The result of pre-processing is shown in Figure 7.

The information retrieved from each image (image name, keypoints, and descriptors) is kept in a dictionary using the image name as a key. The Pickle package from Python is used to serialize and store this dictionary in a database-like file.

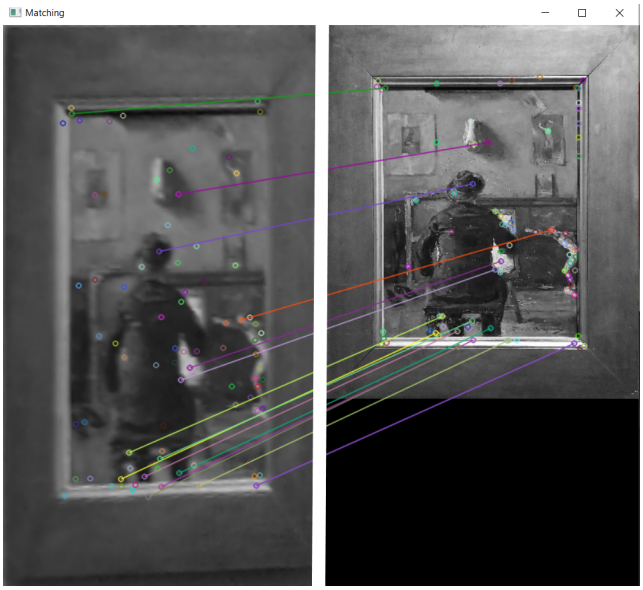


Fig. 7: Image extracted from video (left) vs. pre-processed image (right)

There is no need to compute the keypoints and descriptors of each image in the given database and match them with the extracted image while the program is running. Instead, we compare the extracted image's descriptors and keypoints to the locally preserved dictionary (which is calculated in the beginning). This saves a significant amount of processing resources. It should be noted that the first time the application is launched on a computer, it will take a few minutes for all images to be analyzed and saved in a local file. If the pickle file is detected in the working directory, this pre-processing stage will be skipped.

IV. Results

A. Painting detection

For each image in the dataset, the four corner points, which form the enclosing polygon of the painting, are known. These polygons are shown in green in Figures 8 and 10. Figure 8 gives an example of such an image, where the actual bounding box of the painting is highlighted in green. The algorithm tries to identify this bounding box by using a Canny edge detector. An attempt of the algorithm to determine this border frame is also shown on Figure 8, indicated by the red polygon.

Table III summarizes some of the metrics that evaluate the quality of the painting recognition. The percentage of false positives indicates which polygons were found by the algorithm but appeared not to be paintings. The false negatives are paintings that remained undetected by the algorithm. The true positives are paintings that were correctly identified by the algorithm.

The detection is performed twice. First, the painting recognition is executed as it is implemented in the current system. In this case, only the biggest area that is detected on an image is compared against the ground truth of that image. This implies that when two or more paintings are present in an image, the algorithm will find at most one painting, which will be the

Fig. 8: Example of the ground truth (green polygon) compared to the detected area (red polygon)

TABLE III: False positive, false negative and true positive rate of the painting detection

		One painting per image	Multiple paintings per image
False positive	Absolute	272	101
	Percentage	0,34	0,13
False negative	Absolute	107	111
	Percentage	0,13	0,14
True positive	Absolute	422	589
	Percentage	0,53	0,73

one with the largest area. All other paintings on that image will not be found.

The second run of the painting recognition is slightly adapted. The algorithm now tries to find multiple polygons in an image instead of just one with the largest area. For each ground truth of a painting in an image, the polygon with the largest area that intersects that ground truth is now selected from the polygons found. This polygon is then used throughout the further steps of the algorithm.

It can be noticed that the first run produces more false positives. This is because an image is examined as many times as there are paintings on it. During the first run, the same rectangle is returned each time, and it will correspond to at most one painting. All the other times the image is examined, the same rectangle is returned, and will therefore result in a false positive. The second run will possibly return a different rectangle each time the image is examined, allowing more paintings to be found. This causes more true positives.

Table IV shows the average accuracy of the detected bounding boxes over all images. The accuracy is higher, because more paintings were found in the second run of the algorithm.

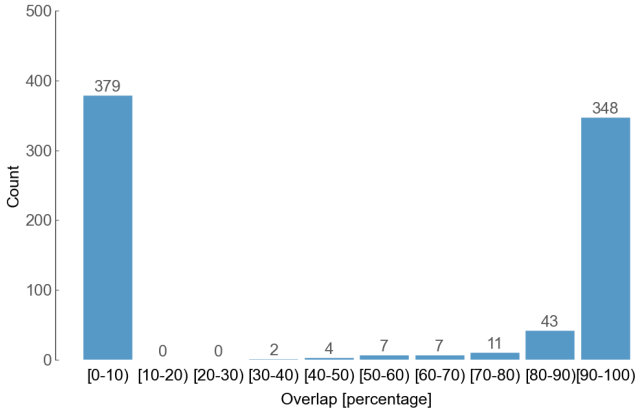
To determine the bounding box accuracy, the intersection over union is calculated. This value indicates the overlap between two areas, in this case between the bounding box of the ground truth and the bounding box found by the algorithm. To determine this value, two areas are introduced: A_d is the area of the detected painting, A_g is the ground truth area of that

TABLE IV: Average bounding box accuracy

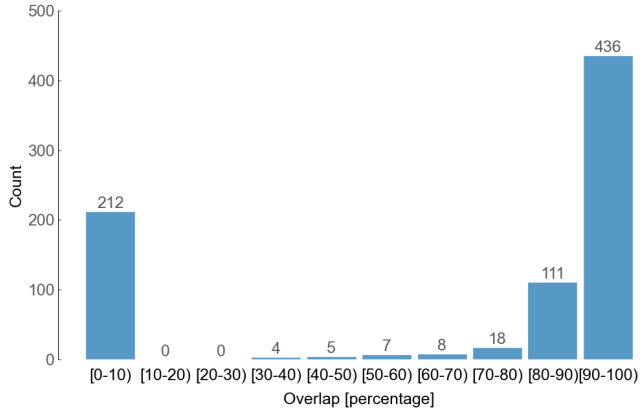
	One painting per image	Multiple paintings per image
Boundig box accuracy	0,49	0,67

painting. The ratio of A_d to A_g describes how much overlap there is between the two. If this ratio becomes 100%, there is a perfect overlap. If it is 0%, there is no overlap. In case the area A_d of the detected painting would be larger than A_g, the ratio of A_g to A_d is taken.

Figure 9 shows how many paintings are detected, together with the amount of overlap that is found between the areas of the ground truth and the detected area. This is the intersection over union as described above.



(a) Histogram of overlaps when only one painting per image is detected



(b) Histogram of overlaps when multiple paintings per image are detected

Fig. 9: Amount of paintings with a certain amount of overlap with the ground truth

There are more paintings with 0 percent overlap on Figure 9a. This is because at most one painting can be detected per image, whereas the implementation, of which the results are shown on Figure 9b, is able to detect multiple paintings per image.

Problems

Some problems of the algorithm can be observed by looking at the bounding boxes themselves. When Figure 8 is considered again, it can be noticed that the edge detector is not always able to find the correct edge. For example, when there is a white frame around the painting, it can happen that the inner edge is detected, and not the outer edge. This is due to the fact that the edge between the colorful painting and the white frame stands out more than the edge between the white frame and the white wall.

Figure 10 shows another problem. If a shadow is present around the frame, it may be detected as an extension of that frame. The shadow has a different intensity compared to the wall and will therefore be detected by the edge detector. Since the algorithm always detects the largest polygon, this shadow may be included in the area.

Fig. 10: Example of the ground truth (green polygon) compared to the detected painting (red polygon). The detected area contains part of the shadow underneath the painting

B. Painting matching

The painting matching algorithm is also evaluated. This is done using the test set. Images in this test set are passed through the matcher one by one, where a visual check is performed to see if the matcher can find the corresponding painting in the database. A count is kept of how many times the matcher finds a correct painting, and how many times a wrong painting is returned.

Of the 236 images in the test set, 162 were correctly identified. There were 74 paintings which the matcher could not link to the correct painting from the database. This gives a matching score of 69%.

Problems

Certain paintings are more difficult to match than others. For example, some are fairly dark and have little contrast. In this case, SIFT has trouble to find distinctive keypoints that really identify that painting. Figure 11 shows such an example. The painting on the left hardly contains any contrast.

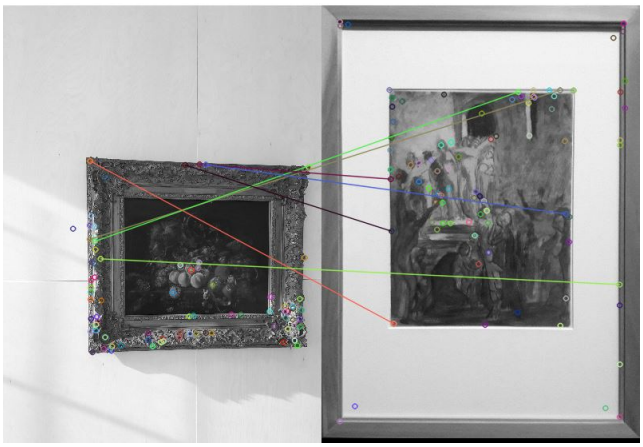


Fig. 11: Example of an incorrect match of the algorithm: keypoints are difficult to find when the painting is dark and has little contrast

V. Discussion

A. Final result discussion

The most common error is that one or more rooms in the route are skipped. By going back and watching the program process the video in real time, it becomes clear that the matching algorithm is responsible for this problem. No matches are provided to the localization algorithm in this room, so it can't detect that the room was visited. In some instances, there are also wrong rooms in the traversed path. For example, in MSK_05 the algorithm thought room 19 was passed instead of room 8. By looking at the console log at the moment the error occurs, it can be determined that the matching algorithm is again the culprit. Tens of incorrect matches are sent to the localization algorithm in subsequent frames.

By visually observing the found matches, a couple of patterns can be observed. The matching algorithm does well on paintings with a lot of contrast and sharp edges. When there is little contrast, the likelihood of incorrect matches increases. Blur also has a negative effect on matching quality, but this is less pronounced.

There are a couple noteworthy rooms that are especially problematic for the algorithm:

1) Room 16 has green walls, but the bottom and top are painted white. Incidentally, the top and bottom of most paintings in this room correspond with the transition from green to white. Therefore no rectangles are detected, subsequently no matches are made, resulting in the localization algorithm always skipping this room.
2) Room E has a lot of sketches on the walls protected by a glossy film. On the other side of the room there are multiple windows. Because of the low contrast in the

sketches combined with the pronounced reflections, no correct matches are made in the first 6 videos in this room.

B. Bottleneck

The biggest bottleneck in the current implementation is the keypoint matcher. Too many false positives are generated. For a human observer it is very easy to pick out the false positives. Often the painting on the left barely resembles the painting on the right. In incorrect matches, the keypoints in the left image are matched to keypoints in the right image that are in completely different locations, as seen in Figure 11

The keypoint matcher from the cv2 library doesn't take keypoint location into account by default because it strives to be invariant to different viewing angles. Consequently, matching keypoints should be found in both pictures of the same object, even if the perspective has changed between the two images. (A keypoint on the bottom of the image should be able to match a keypoint on the bottom of the other image because one of the images could be rotated 180° degrees)

In this algorithm, the detected shape is already transformed into a rectangle with similar dimensions to the painting we're comparing against before it is handed over to the matcher. Therefore the relative position of the keypoints in the left and right image should be roughly similar as well. The invariance to different viewing angles is usually an advantage in keypoint matching, but is disadvantageous in our application.

C. Future work

One possible improvement is to incorporate the keypoint location in the matching process. A first solution is to write a custom implementation of the OpenCV Brute-force descriptor matcher that takes position into account. Another solution is to take the output of the current matcher and remove matches in keypoints that aren't in roughly the same relative location on the painting. A last proposed option requires more changes, but tackles the problem at the root. A custom keypoint descriptor could be developed where the keypoint position plays a central role. For example, the image could be divided into 16 rectangles, where each rectangle is a keypoint. A couple metrics such as the gradient and relative brightness to the other keypoints are calculated for each keypoint. Finally, all the keypoint matcher has to do is calculate the distance between each keypoint in the left image to its counterpart in the same location in the right image. If the average distance is below a threshold, the two painting are regarded a match.

Another possible improvement could be found in the application of neural networks [11, 12]. Some results indicate that a Convolutional Neural Network (CNN) could be used for the retrieval of visual links of paintings. They could possibly perform better than SIFT, because a CNN has fewer difficulties with the variability in patterns, styles and colors [12].

VI. Conclusion

This paper described the implementation and evaluation of an indoor positioning algorithm, based on painting recognition in the MSK in Ghent. The algorithm consists of three components: painting detection, painting matching and localization estimation.

Results of the detection component indicate a bounding box accuracy of 49%. This average performance is mainly due to almost half of the paintings in the database that could not be detected at all. This is mostly the case when multiple paintings are present on a single image.

The matching component achieves an accuracy of 69%, based on the images and paintings available in the test set. Most of the mismatches are due to dark paintings that have little or no contrast. SIFT seems to have more trouble in finding distinctive and useful keypoints in those cases.

Because of the average results of these two components, the localization estimation does not have the ability to establish the correct route in every video. Sometimes incorrect rooms are visited. However, the attempts made are certainly not disastrous, since the algorithm is capable of establishing the general direction of the routes.

Bibliography

[1] 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.

[2] S. Suzuki and K. A. Be, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, apr 1985.

[3] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision 2004 60:2*, no. 2, pp. 91–110, nov.

[4] R. Choudhury, "Recognizing pictures at an exhibition using sift," *Retrieved on September*, 2008.

[5] T. Dalens, J. Sivic, and I. Laptev, "Painting recognition from wearable cameras," vol. 13, 2014.

[6] Y. Hong and J. Kim, "Art Painting Identification using Convolutional Neural Network," *International Journal of Applied Engineering Research*, vol. 12, no. 4, pp. 532–539, 2017.

[7] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1615–1630, oct 2005.

[8] C. Broderick and K. Dawson-Howe, "Robust recognition and identification of paintings using computer vision techniques," Master's thesis, University of Dublin, Trinity College, 2016.

[9] S. Shah and J. Aggarwal, "Intrinsic parameter calibration procedure for a (high-distortion) fish-eye lens camera with distortion model and accuracy estimation," *Pattern Recognition*, vol. 29, no. 11, pp. 1775–1788, 1996.

[10] A. Jakubović and J. Velagić, "Image feature matching and object detection using brute-force matchers," *Proceedings Elmar - International Symposium Electronics in Marine*, vol. 2018-September, pp. 83–86, nov 2018.

[11] I. Santos, L. Castro, N. Rodriguez-Fernandez, Á. Torrente-Patiño, and A. Carballal, "Artificial Neural

Networks and Deep Learning in the Visual Arts: a review," *Neural Computing and Applications*, vol. 33, no. 1, pp. 121–157, jan 2021. [Online]. Available: https://link.springer.com/article/10.1007/s00521-020-05565-4

[12] B. Seguin, C. Striolo, I. DiLenardo, and F. Kaplan, "Visual link retrieval in a database of paintings," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9913 LNCS, pp. 753–767, 2016. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-46604-0_52