Sematic Segmentation based Building Façade Recognition

Project Report

Group 13

Noor Ul Ain,
Tapani Honkanen,
Topias Harjunpää,
Samu Syrjänen,
Zihao Li

May 4, 2024

UNIVERSITY OF HELSINKI

Table of Contents

1. Introduction

Augmented Reality (AR) is a rapidly expanding field, recognizing and interacting with real-world structure with ease is critical in AR applications. This project sets out to transform urban interaction through AR with primary focus on building façade recognition and user viewpoint localization. The typical solution for these challenges is to recognize the features from the images, match recognized feature positions and wrap the image accordingly. But the drawback of utilizing visual features is that they are not robust to variations in weather, time of day and seasons, additionally it requires sending the entire image to the backend to extract features. To avoid these problems, we have used semantic segmentation-based matching in which we utilized the semantic features (windows, balconies, doors) of building façade and template matching for building recognition. This technique can reduce the reliance of the application on backend services as the data base of any visited area can be easily stored on the front end device as the amount of data is minimal (only semantic information is required). Additionally, this solution is robust to seasonal and weather changes because the visual features change depending on these factors, but the semantic features are not affected by these factors.

The goal of this project is to design an algorithm for recognizing building façade using hierarchical filtering and user viewpoint localization, with the primary focus on accuracy and secondary consideration for speed. In the first iteration of the project, we applied individual filters to recognize the building façade and evaluated those filters. After that we used hierarchical filtering to reduce the search space as fast as possible by applying these filters in a particular order. These filters utilize sematic features such as number of stories, number of balconies, number of windows/doors, wall/door color, roof shape, optical characters and building texture. After reducing the search space our aim was to develop a user viewpoint localization algorithm for recognizing the correct building façade.

2. Technical Approach

This section provides the details of the tools and methods we have used in our project.

2.1 Tools and Infrastructure

- Programming Language: Python
- Libraries: OpenCV, Matplotlib, Numpy, PaddleOCR
- Data Visualization and Analysis: Jupyter Notebooks
- Version Control System: Git, with GitHub as the remote repository

2.2 Data Processing

The dataset consists of two publicly available datasets of 139 image pairs consisting of original image and sematic segmented image. The first data set is eTRIMS and the second dataset is ParisArtDecoFacadesDataset. Semantic segmentation categorizes each pixel into a class such as window, door, balcony, wall, road, or sky. To standardize the dataset, we have extracted each feature into a separate mask. We also cropped the images into quarters and distorted them by flattening to evaluate our algorithm for its robustness. Figure 1 shows the image pair in the data set, image quarter and the mask holding the label “window”.

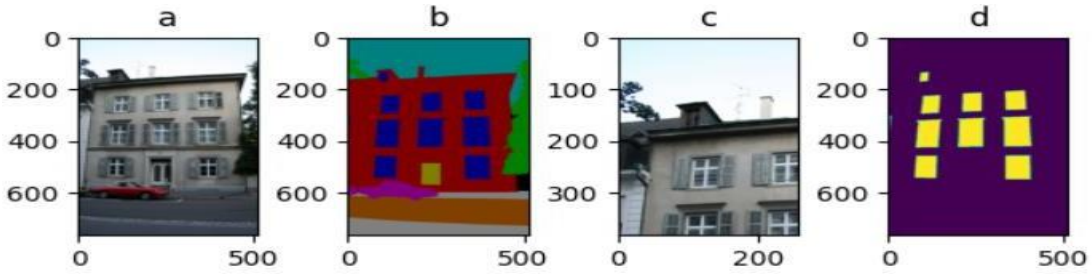

Figure 1 a. Original Image b. Annotated Image c. Image Quarter d. Mask holding window label

2.3 Hierarchical filtering

In hierarchical filtering, we apply a series of filters on the query image to narrow down the search space. The filters are arranged based on their individual performance in terms of both accuracy and speed. Beginning with the first filter, we provide the query image along with the entire dataset as input. The filter then produces a reduced dataset, retaining only the most promising images. This refined dataset is subsequently passed to the next filter along with the original query image. Each filter aims to include the best-matched image in the reduced dataset. In this way our algorithm reduces the search space by passing query images through all the filters. The reduced data set is then used for localizing user viewpoint in the query image to find the best match image in the reduced dataset. Applying hierarchical filtering before localizing user viewpoint reduces the dataset for finding the correct image using user localization algorithm. User viewpoint localization algorithm is time consuming as compared to the individual filters. Figure 2 shows the overview of our algorithm. The algorithm and evaluation of each filter is outlined in this section.

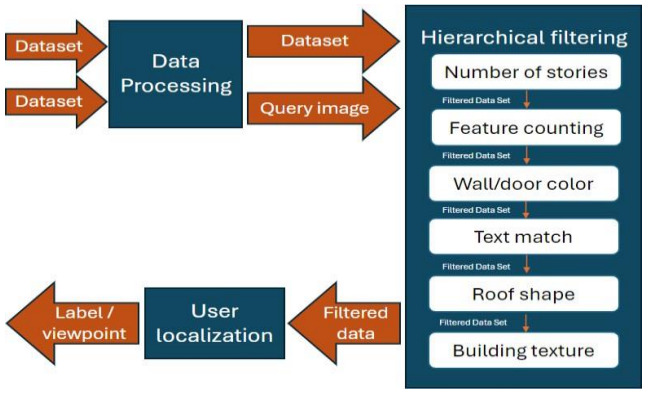

Figure 2 Overview of Facade Recognition Algorithm

2.3.1 Number of Stories

The filter works by counting the number of stories in a building and operates as follows: First, the number of stories is calculated by finding the maximum number of windows bounding boxes that simultaneously intersect with a vertical line as it moves horizontally across the image. Candidates with fewer stories than those in the query image are then removed from consideration.

The method is very efficient because the same window bounding boxes can be reused by the feature-counting filter, and collisions with axis-aligned bounding boxes can be detected by simple comparisons. It filters many of the same candidates as feature counting, so the improvement in filtering efficiency is minimal. The overall performance gain is around 0.2 %. However, the method is worth including because

it achieves perfect accuracy (100%) on cropped images and has a minimal impact on the execution time (approximately one millisecond). As with some other methods, the filter relies on accurate semantic segmentation, which could affect its real-life performance. As future work, the filter could be made stricter if it is possible to confirm that the image contains a complete vertical slice of the building.

2.3.2 Feature Counting

The primary objective of feature count filtering is to efficiently eliminate any building facades that do not match the number of characteristics of a given query image. For example, if a query image contains 10 windows and 2 balconies, only facades with at least these numbers of features should be considered; those with fewer can be confidently excluded.

This filtering method utilizes counts of stable features, which are less likely to change over time, including windows, doors, balconies, shops, and vegetation. For each feature, feature counting involves the following steps: The algorithm receives a feature mask as input, which is extracted from the semantically segmented annotations during preprocessing. Features are then quantified by identifying the bounding boxes within the feature mask and calculating their counts. This process is facilitated by using the OpenCV Contour Features and is illustrated at left side image in Figure 3.

During the initial visualization, it was identified that some features, such as windows, were often fully or partially obscured by obstacles like trees. This fragmentation led the feature count operation to report artificially high counts. To address this issue, a morphological operation was applied to the feature mask using a sufficiently large kernel. This process helped consolidate the fragmented features back into a single entity. Although the solution is not perfect, its primary goal is to prevent the computation of additional feature counts, reducing the risk of incorrectly filtering out the correct facade. The second issue was related to counting the number of vegetation instances. Due to the irregular shapes of vegetation, such as trees, there was a risk that a single piece of vegetation could be split into multiple distinct annotated areas in the query image. This could inadvertently lead to the exclusion of the correct facade. To resolve this, we decided to implement a binary check to determine the presence of vegetation, rather than counting the individual instances.

Figure 3. The left side image illustrates the process of window counting. Right side plot shows distribution of remaining candidates after feature count filtering.

As an outcome, feature-based filtering operates with good efficiency, taking approximately 1 millisecond per run and achieving 100% accuracy in our testing settings, meaning it never incorrectly filters out the correct label from the list of remaining candidates. On average, this method reduces the list of candidates by about 50%. However, its effectiveness heavily depends on the features present in the query image. The

left-side plot in the Figure 3 illustrates the distribution of remaining candidates for each of the 556 query images tested.

2.3.3 Feature Color

Filtering based on feature color involved focused on filtering the images based on the similarity of the feature color of test image and the images in the data set. We have used wall color and door color of the buildings in this filter. Filtering based on feature color included following steps:

1. Extracted the average feature color (wall and door) using the mask holding the label "building" and "door" and stored these colors.
2. Passed the query image and extracted the average feature color (wall or door) of all the images to the filter.
3. Extracted the feature color (door or window) of the query image.
4. Calculated the Euclidean distance between the average color of query image and all the images in the data set.
5. Normalized and sorted the Euclidean distances. In the first iteration of the algorithm, the filter gave the sorted Euclidean distances as the output. The best match image had the minimum Euclidean distance. At this point we evaluated this filter first using the original images in the data set as query image. Table 1 shows the evaluation of this filter.

Metric	Wall Color	Door Color
Accuracy	100%	90%
Average Execution Time	~13 msec	~12 msec

Table 1. Evaluation metric of Filtering based on feature color using original image as query image

After that we evaluated this algorithm for cropped images as query image to check the robustness of the algorithm. The accuracy for the image quarter dropped significantly both for wall color and door color because the average wall color of cropped image was notably different from average wall color of full image Additionally, there were cropped images having no door at all or have half door. But this algorithm was still useful for reducing the data set.

6. In the second iteration we adjusted our algorithm for hierarchical filtering. In this updated filter a threshold was applied to normalized Euclidean distances. This process involved filtering out images from the dataset whose Euclidean distances exceeded the predefined threshold. The resulting dataset, consisting of only those images that met the threshold criteria, was then provided as the output of the filter. We adjusted the threshold for door color and wall color in such a way that the correct image is always present in the reduced dataset. Figure 4 shows the effect of threshold on accuracy, we used threshold of **0.63** and **0.88** for wall color and door color, respectively.

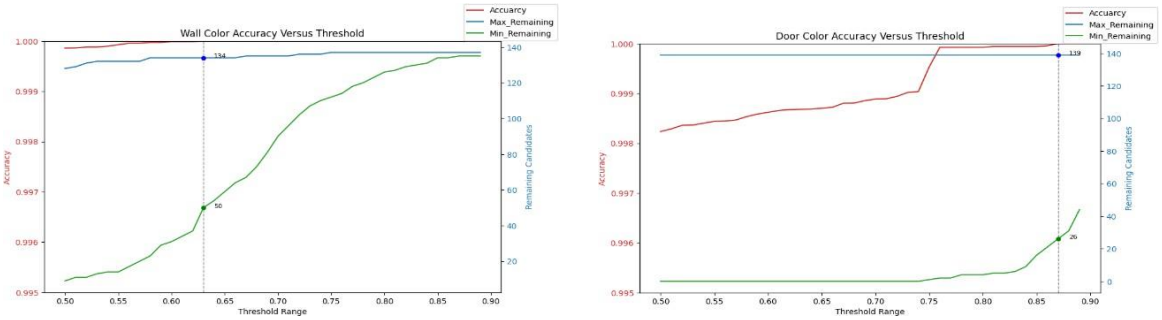

Figure 4. The plot of Accuracy, number of remaining candidates after filtering (Maximum and Minimum) for wall color on Left side and door color on Right side.

2.3.4 Text Match

Text matching in images involves identifying and comparing text extracted from various images to determine similarities or exact matches. This process is crucial for applications where textual content within images needs to be cross-referenced or verified against a database or other images. Following are the details of text matching algorithm.

1. **Text Detection and Recognition:** Utilize PaddleOCR to perform Optical Character Recognition (OCR) on all images in the dataset. Each image's detected text is then stored in a JSON file. This file acts as a reference database for future text comparisons.
2. **Extracting Text from New Images:** When a new image is input, run OCR to extract the text. This OCR process converts image-based text into a searchable text format.
3. **Loading the Database:** Load the JSON file containing the OCR results from previously processed images. This JSON structure is crucial as it allows quick access to pre-analyzed text data, facilitating efficient searches.
4. **Searching for Matches**: Compare the text extracted from the new image against the texts stored in the JSON file. Implement a text comparison algorithm that can recognize similar or identical text strings. Techniques such as substring searches, Levenshtein distance, or more sophisticated natural language processing (NLP) methods can be used to determine textual similarities.
5. **Output Results:** List images whose text content matches or closely resembles the text in the new image. This step involves sorting or ranking images based on the degree of text match to prioritize more relevant results.

We evaluated the text matching system's performance based on metrics such as accuracy and execution time details are given in Table 2.

Metric	Original	10 degrees rotation	15 degrees rotation
Average Execution Time	0.1178 s	0.1817 s	0.1545 s
Accuracy	85.71%	75.00%	71.43%

Table 2. Evaluation of text matching for original and rotated images

To refine our matching technique and focus on the most relevant aspects of urban imagery, we explored the possibility of image matching that specifically targets the building portions of images. This approach was aimed at minimizing the influence of extraneous elements, such as pavement signage, which do not contribute meaningfully to building recognition.

The technique involved segmenting the images to isolate building facades before applying our matching algorithm. However, a significant limitation was observed with this approach. Our dataset primarily consists of images where the buildings have minimal textual content. Despite the limitations within our

dataset, this method holds promise for real-world applications. In a practical setting, where buildings may have more distinct and text-rich facades, this targeted approach could effectively enhance the relevance and accuracy of image matching systems.

2.3.5 Roof Shape

We have tried a couple of ways to use the building shape to filter out candidates. The first approach was to try the simplest possible solution. The OpenCV library includes a shape-matching functionality based on Hu Moments, which are a set of seven numbers that capture characteristics of a shape. The first six Hu Moments are known to be invariant to translation, rotation, and scale. We tested the shape-matching algorithm on various targets and identified the contour between the sky and the building masks as the most promising option. This contour is less likely to be obscured by other objects. However, the shape-matching algorithm struggled with partial matches, resulting in inferior performance on cropped images.

Our final approach is to extract the contour between the sky and the building and then simplify it using Ramer-Douglas-Peucker algorithm with an epsilon value of 0.01. We filter candidates based on the number of points in the simplified contour. All candidates with a point count within a certain tolerance are included in the filtered dataset. The basic idea is to differentiate flat roofs from more complex roof shapes. As shown in the histogram in Figure 5(Left), the point count differences between original and cropped images suggests that a threshold of 2 is sufficient to achieve high accuracy (> 99%) with our dataset. This threshold is also strict enough to effectively filter out candidates as we can deduct from the vertex count distribution (see Figure5 Right). The average execution time for the filter is approximately 40 milliseconds.

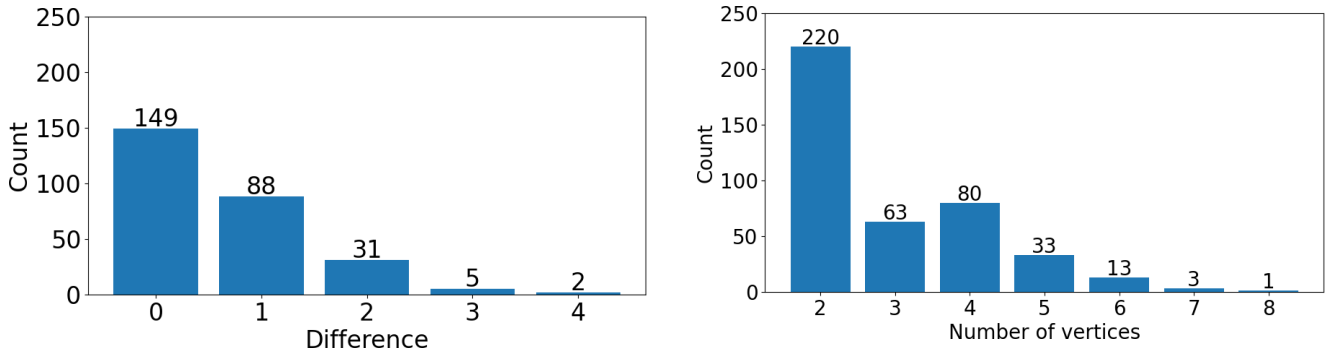

Figure 5: Left is the histogram of differences in the number of vertices between the original image and cropped images, Right side is the distribution of number of vertices in the simplified roof contours

An additional observation is that the annotations around the edges of different classifications are messy, leading to unreliable edge extraction. To address this, the edges needed cleaning. We tried various approaches, but the most effective one involves iteratively copying the classification of the top-left pixel to any unclassified pixels. This method is both performant and reasonably accurate in practice.

2.3.6 Building Texture

Building texture can be analyzed and compared with other images to select the best matches. We decided to use the Local Binary Pattern (LBP) algorithm for the texture analysis. LBP is considered a lightweight texture analysis algorithm, which could be used in real time.

The general implementation of our texture analysis algorithm is described in the following steps:

1. First a binary pattern is assigned to every pixel in the image. The appendix Figure 1 and appendix Figure 2 show this process. The center pixel's value is compared with each surrounding pixel to generate a binary number, or a binary pattern. The number of surrounding pixels considered for each center pixel, and the radius of the surrounding pixels is determined by the algorithm parameters. From the generated binary patterns, we can get information about the positions and frequencies of edge, flat, and corner areas, as shown in the appendix figure 2.
2. The normalized frequency (or sum) of each possible binary pattern in the image is then converted into a histogram, which shows the normalized frequency of each unique binary pattern on y-axis, and all the unique binary patterns on x-axis. An example is shown in appendix Figure 3 about a brick wall.
3. After calculating the LBP images and histograms, we can compare the similarity of each histogram from different images to find the best matches. This is done with Kullback–Leibler divergence, which gives a similarity score for two different histograms.

Our implementation of LBP algorithm uses “uniform” method to calculate the unique binary patterns. It means that the starting point of the calculated binary number does not matter when considering its 'uniqueness’ only the pattern’s order of content matters. This minimizes the impact of image rotation in matching. The functionality of the “uniform” is described in detail in Scikit-Image's documentation of the local binary pattern.

An example LBP image generated from our dataset can be seen in appendix Figure 4. Unlike in the example appendix Figure 3, we include the entire facade area which consists of building, window, and door masks. Other masks such as road or foliage are not considered in the texture analysis (shown as black areas in the example). Including the entire facade area increased the accuracy of the algorithm, but simultaneously increased the running time.

The main parameters for Scikit-Image's local binary pattern algorithm include the radius of the surrounding pixels (say R), and the number of surrounding pixels considered for each center pixel (say P). For P, we carried out an optimization process which maximized the accuracy of the algorithm as its first criterion and minimized the running time as its second criterion. For R, the optimization process was more difficult since the resolution of the dataset images had some variability. The solution was to anchor the R value to the width of the windows found in the images. Example is shown on appendix figure 5. This way the R value would always be standard to the real-world scale in the images. The R parameter was optimized similarly as P, but because of the scale standardization, used window width as its unit (for example: 0.5 = half a window). More information about the LBP algorithm can be found in the links shared in Appendix 6.2.

2.4 User localization

The user localization task serves two functions in the final algorithm. Firstly, it determines the user's point of view using template matching. Secondly, it employs the similarity score output from this process to identify the correct label among all remaining candidates from the hierarchical filtering. The primary challenge in the user localization was enhancing its efficiency to enable real-time operation on local

devices. Additionally, the template matching process was designed to utilize only annotations, thus eliminating the need to transmit original images to the frontend.

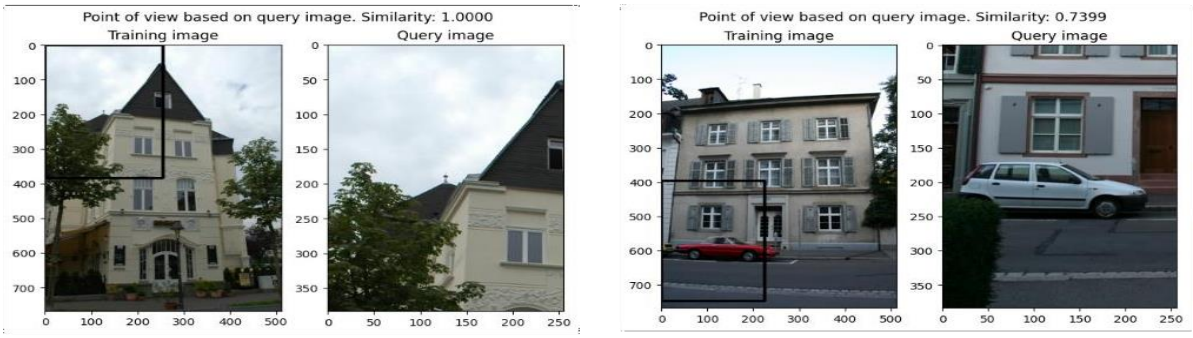

Figure 6: Two distinct outputs from the user localization process. The point of view from the rescaled query image is overlaid on top of the corresponding training image, and a similarity score is computed to assess the match.

Our client had implemented an initial solution for window-based template matching, and our task was to enhance its computational efficiency. This approach involved individually rescaling the query image to match the size of each window in the training image. The rescaled query image is then overlaid on top of the training image, and a similarity score is computed for the overlaid area between the training and query images. Figure 6 illustrates two different outcomes of the user localization process. The left-side image pair represents a correct match, successfully identifying the correct facade and accurately localizing the user's point of view. The right-side image pair depicts a scenario where the correct facade is not included in the set of training images. To enhance template matching efficiency, we implemented two modifications to the existing approach. Firstly, we leveraged feature counting by computing similarity scores between overlaid images only when their window counts matched, reducing the number of iterations required. Secondly, we initially computed the resized query region to determine the overlaying area for the training image, but we refrained from resizing the query image in advance. This strategy allowed us to first compare window counts and subsequently resize only when necessary for similarity score calculations.

The window-based template matching technique, along with the described performance enhancements, proved to be an efficient and accurate method for user localization. However, it encountered limitations when query images lacked windows, which occurred in approximately 10% of cases in our testing settings. Additionally, it failed to achieve real-time performance with larger pools of remaining candidates. To address these challenges, we first applied the matchTemplate function from OpenCV when query images did not contain windows. Although this method did not perform as robustly with image distortions, it was sufficient for the small number of affected cases, improving overall accuracy from 90% to 97% in scenarios with the worst image distortion. To enhance real-time performance, we employed ThreadPoolExecutor to execute template matching asynchronously, enabling us to identify and localize the correct facade in sufficient time with larger pools of remaining candidates. Table 3 displays the computation times for the user localization using different sizes of training image sets. For each training set size, user localization was computed individually for all 556 query images. Table 3 shows the slowest, fastest, and average execution times.

Training image set size	Average execution time	Fastest execution time	Slowest execution time
139	741 ms	365 ms	1329 ms
60	328 ms	149 ms	581 ms
40	213 ms	115 ms	368 ms
20	114 ms	62 ms	216 ms

Table 3: Execution times for user localization using different sizes of training image sets.

3. Results

3.1 Cropped Images

We tested the algorithm on cropped images. The evaluation metrics we measured for the hierarchical filtering are given in Table 4. The "total filtered column" represents the portion of the full dataset that has been excluded by this filter. The value depends on the order in which filters are applied, as the same image may be excluded by multiple filters. We can observe that hierarchical filtering is effective at reducing the number of candidates for user localization. The accuracy is measured as the proportion of trials in which the correct label was among the set of remaining candidates.

Filter	Total filtered	Avg. execution time (of total)	Accuracy
Number of stories	12.05%	1.4ms (0.23%)	100%
Feature counts	37,00%	0.8ms (0.13%)	100%
Wall color	5.15%	3.2ms (0.51%)	99.82%
Door color	2.45%	1.5ms (0.43%)	96.58%
Text match	1.89%	35ms (5.54%)	97.48%
Roof shape	7.7%	13ms (1.94%)	96.76%
Texture	8.33%	414ms (63.64%)	94.06%
Total	74.57%	480ms (72.42%)	84.71%

Table 4. Evaluation of hierarchical filtering for cropped image as query image.

The user localization accuracy was 100%, which was expected as it is based on template matching. This indicates that errors were caused by the correct label being filtered out during the hierarchical filtering process. Consequently, the accuracy of the full algorithm is 84.71%. The average execution time for user localization was 182 milliseconds (27.58% of total runtime). This suggests that running most of the filters can improve performance. The execution time for the full algorithm was also quite consistent as shown in Figure 7.

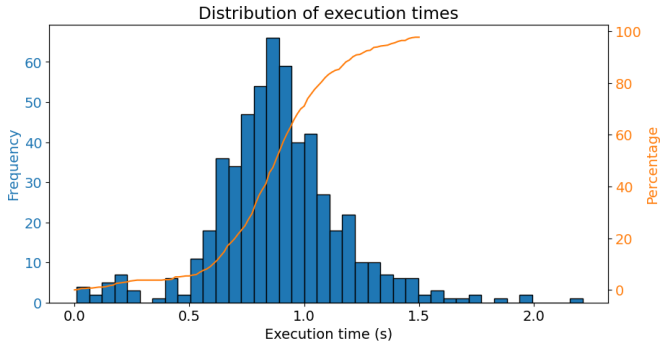

Figure 7: Distribution of Execution time of full algorithm

3.2 Distorted Images

To test our system's robustness, we test the hierarchical filtering on distorted images. The detail results of individual filters and overall algorithm are given in Table 5 and Table 6 respectively.

Filter Name	Filter Method	Total Filtered	Total Execution Time (s)	Correctly Filtered	Average Execution Time (ms)	Accuracy (%)
Stories	Original	9315	0.9425	9315	1.6952	100.00
	Image Rotation	9315	1.0428	9315	1.8755	100.00
	Image Stretching	9315	0.8473	9315	1.5239	100.00
	Image Flattening	422	0.0172	422	0.6158	100.00
Feature Count	Original	28592	0.6059	28592	1.0898	100.00
	Image Rotation	28592	0.5731	28592	1.0307	100.00
	Image Stretching	28592	0.5332	28592	0.9590	100.00
	Image Flattening	1466	0.0193	1466	0.6897	100.00
Wall Color	Original	3978	2.9484	3977	5.3029	99.82
	Image Rotation	3823	1.8957	3820	3.4096	99.46
	Image Stretching	3955	1.6990	3954	3.0558	99.82
	Image Flattening	285	0.0616	285	2.2015	100.00
Door Color	Original	1896	2.5452	1877	4.6025	96.58
	Image Rotation	1914	1.6161	1894	2.9224	96.40
	Image Stretching	1889	1.5733	1870	2.8449	96.58
	Image Flattening	83	0.0476	83	1.6988	100.00
Text Match	Original	1457	32.8261	1443	59.5754	97.48
	Image Rotation	1380	27.9669	1368	50.7566	97.84
	Image Stretching	1466	25.7384	1452	46.7971	97.48
	Image Flattening	0	1.0193	0	36.4028	100.00
Roof Shape	Original	5950	10.6079	5932	19.7909	96.76
	Image Rotation	7332	11.0885	7252	20.6489	85.61
	Image Stretching	5024	8.4395	4971	15.7748	90.47
	Image Flattening	218	0.4785	216	17.0901	92.86
Building Texture	Original	6445	259.8596	6412	485.7190	94.06
	Image Rotation	8324	237.9636	8272	455.8689	90.65
	Image Stretching	8414	221.9311	8361	421.9222	90.47
	Image Flattening	446	8.6464	443	308.7997	89.29

Table 5: Filter Performance Details

Filter Method	Total Filtered	Total Filtering Time (s)	Average Filtering Time (ms)	Total Localization Time (s)	Average Localization Time (ms)	Average Execution Time (ms)	Correct Label in Filtered Data (%)	Correct Count (%)
Original	57633	310.3613	592.2926	182.7388	348.7383	886.8708	84.71	84.71
Image Rotation	60680	282.1711	568.8934	144.3395	291.0070	767.1054	69.96	69.78
Image Stretching	58655	260.7857	516.4074	135.1115	267.5476	712.0454	74.82	72.66
Image Flattening	2920	10.2908	367.5292	7.1095	253.9110	621.4402	82.14	82.14

Table 6: Summary Performance

4. Conclusion

We developed an algorithm for building facade recognition using sematic features. The algorithm was developed with a primary focus on accuracy, with speed being a secondary consideration. Through hierarchical filtering we efficiently reduced the search space by applying series of filters based on sematic features such as the number of stories, features counting, feature color, roof shape, text matching, and building texture. This approach optimized the speed of our algorithm by reducing the data set for user viewpoint localizing that was used to find the best match image from the reduced data set.

Our algorithm was evaluated for different scenarios. The results of our algorithm are satisfactory, achieving high accuracy rates across different scenarios. However, there is still room for improvement and future work. Improvements can focus on enhancing the text matching procedure, particularly in situations when building facades have little to no text. Additionally, the algorithm could also be enhanced to cater to weather and seasonal changes using customized dataset (using Images of same building in different times of the day and different seasons). Furthermore, real-time performance might be improved by further refining the user localization algorithm, especially in situations involving larger datasets. In terms of follow-up work, broadening the dataset to include more varied urban environments and architectural types might improve the robustness of the algorithm. Incorporating machine learning methods, like deep learning models, may also make the algorithm more adaptive to changing architectural elements and environmental variables. Overall, our project lays solid foundations for advances in AR applications in Urban Environments.

5. Grading

Group's Deliverables Grade: 5

The topic of the project was challenging for us because most of the group members did not have any previous work experience of computer vision and image processing. The project scope is clearly stated in the project plan, presentations, and the final report. In the presentations we gave an overview of our technical plan due to time limitation, the technical details and evaluation of different approaches used in the projects are discussed thoroughly in the project report. We have not only elaborated the implementation of our algorithm, but we have also highlighted the technical issues we faced and how we modified our approach to cater for those issues. In our project report, we have also referenced multiple online sources, aiding readers in understanding our algorithm. In addition to collaborating with other group members each group member worked independently on assigned tasks within the scheduled timeline. All the deliverables are prepared according to the instructions provided. The conclusion of the project report and the presentations was clear and concise. All the project deliverables are refined and well formatted.

Group Work Grade: 5

Our group meeting was scheduled every Monday. The meeting was well organized, have a proper agenda. Each group member shared the details of his/her work of the previous week, the challenges they faced and set the goal for the coming week. We as a group had analytical discussions and there was no irrelevant discussion. In our biweekly meeting with the client all of us presented our work. We had a clear goal and each of us worked on individual tasks that contributed to achieving that goal.

The group environment was quite interactive; group members took responsibility for their individual tasks and responsibilities were fairly assigned. The tasks that needed input from all group members such as project plan, presentations, and project report were timely completed due to effective collaboration. The group work enhanced our learning, especially the details of each algorithm provided by the respective members in our GIT Repository helped in understanding the algorithm.

6. Appendix

6.1 Building texture figures

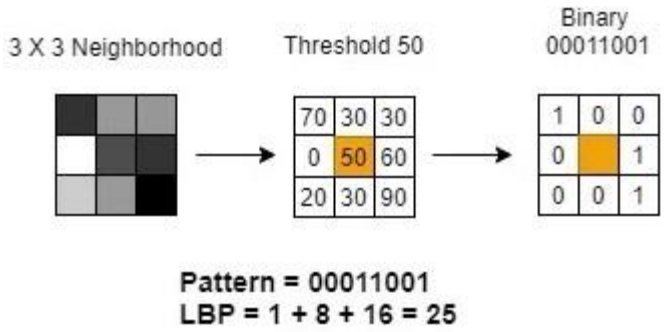

Figure 1 https://aihalapathirana.medium.com

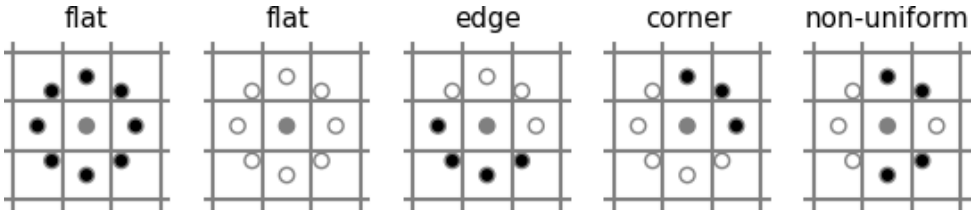

Figure 2 https://scikit-image.org

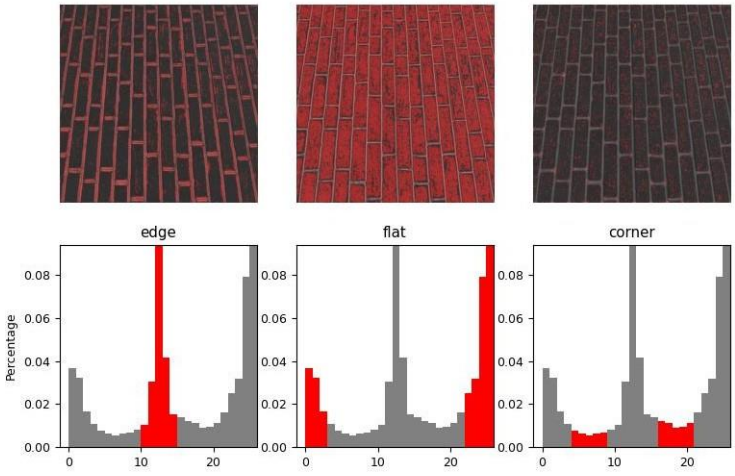

Figure 3 https://scikit-image.org

Figure 4 Example LBP image from our dataset.

Figure 5. LBP radius parameter real-world scaling.

6.2 Local Binary Pattern Algorithm

https://scikit-image.org/docs/stable/auto_examples/features_detection/plot_local_binary_pattern.html

https://en.wikipedia.org/wiki/Local_binary_patterns

https://aihalapathirana.medium.com/understanding-the-local-binary-pattern-lbp-a-powerful-method-for-texture-analysis-in-computer-4fb55b3ed8b8

https://becominghuman.ai/local-binary-pattern-features-for-texture-classification-d0dfd86ebf29

https://towardsdatascience.com/the-power-of-local-binary-patterns-3134178af1c7