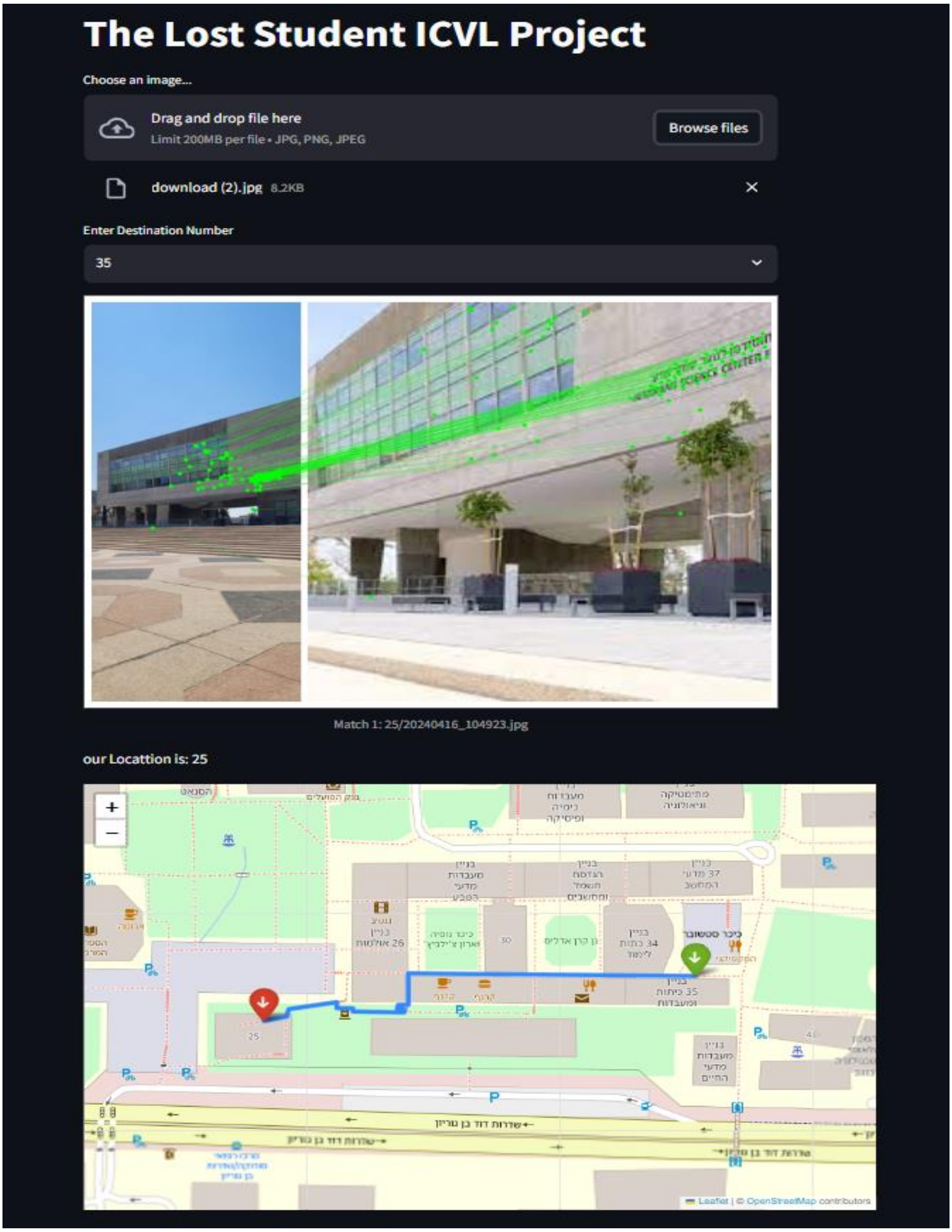
THE LOST STUDENT APP

By Roy Amoyal, Ron Haikin and Omri Hirsh



The Lost Student ICVL Project

Choose an image...

Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files

download (2).jpg 8.2KB

Enter Destination Number

35

Match 1: 25/20240416_104923.jpg

our Locattion is: 25

+
−
Leaflet | © OpenStreetMap contributors

Introduction

Imagine being a freshman at Ben-Gurion University, stepping onto campus with excitement and anticipation, only to find yourself lost in a maze of buildings, unsure of where the building of your next class is located.

It's a common scenario – asking strangers for directions, rushing through unfamiliar corridors, risking being late for class. In addressing this challenge, we embarked on a mission driven by empathy and innovation.

What if we could harness the power of Computer Vision to guide these students seamlessly through their campus journey?

Thus, our project was born, with a vision to empower students to effortlessly navigate their academic environment using nothing but their smartphone's camera.

Our goal was clear: to eliminate the confusion of building numbers and convoluted navigation instructions.

Instead, we sought to provide a solution that intuitively understands the student's location within Ben-Gurion University, allowing for swift and precise guidance to their destination.

Central to our endeavor was the formidable task of localization – the ability to pinpoint the student's exact whereabouts with a single snapshot from their camera.

This few-shot images became the cornerstone of our approach, unlocking the potential to revolutionize campus navigation for students.

Related Work

In the realm of computer vision and image processing, numerous methodologies have been explored to tackle the challenge of Image Based Localization and Image Retrieval, particularly in the context of autonomous navigation. Among these methodologies, feature matching stands out as a pivotal technique for accurately determining the user's location within a given environment (or doing SLAM - Simultaneous Localization and Mapping).

One of the fundamental approaches to feature matching involves corner detection algorithms. These algorithms identify distinctive points or corners within an image that exhibit significant changes in intensity or gradient, thereby serving as robust features for matching and localization. Classical methods such as the Harris corner detector and the Shi-Tomasi corner detector have been widely employed in various applications due to their simplicity and effectiveness in identifying salient image features.

Additionally, feature-based techniques like Scale-Invariant Feature Transform (SIFT) and Oriented FAST and Rotated BRIEF (ORB) have emerged as powerful tools for feature extraction and matching. SIFT offers scale and rotation invariance, making it robust to changes in viewpoint and illumination. ORB, combines speed and accuracy by utilizing binary descriptors, making it well-suited for real-time applications.

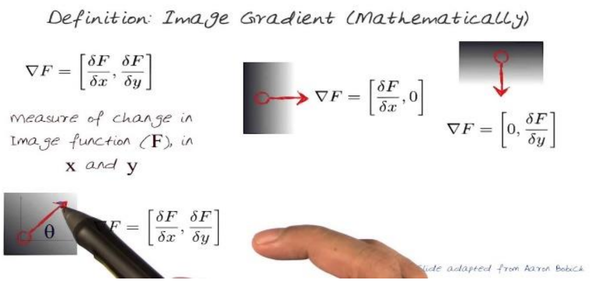
Moreover, recent advancements in feature matching have introduced sophisticated algorithms such as SuperPoint and its derivative, SuperGlue. SuperPoint excels in feature extraction, generating high-quality key points even in challenging scenarios. Its successor, SuperGlue, enhances feature matching accuracy by employing a learned matching model, thereby improving robustness and reliability in localization tasks.

Mathematical Background

Feature detection and extraction are pivotal components of computer vision algorithms, enabling the identification of distinctive points or regions within an image. These techniques are fundamental for tasks such as object recognition, image registration, and visual localization. Here, we provide an overview of key mathematical concepts and methodologies involved in feature detection and extraction.

Gradients and Edge Detection

Gradients represent the rate of change of intensity in an image and are essential for edge detection, a fundamental step in feature extraction. Techniques like the Sobel and Prewitt operators compute gradient magnitudes and orientations, highlighting regions with significant intensity changes, which often correspond to edges or boundaries between objects.



Definition: Image Gradient (Mathematically)

$$\nabla F = \left[\frac{\delta F}{\delta x}, \frac{\delta F}{\delta y}\right]$$

measure of change in Image function (**F**), in **x** and **y**

$$\nabla F = \left[\frac{\delta F}{\delta x}, 0\right]$$

$$\nabla F = \left[0, \frac{\delta F}{\delta y}\right]$$

$$\nabla F = \left[\frac{\delta F}{\delta x}, \frac{\delta F}{\delta y}\right]$$

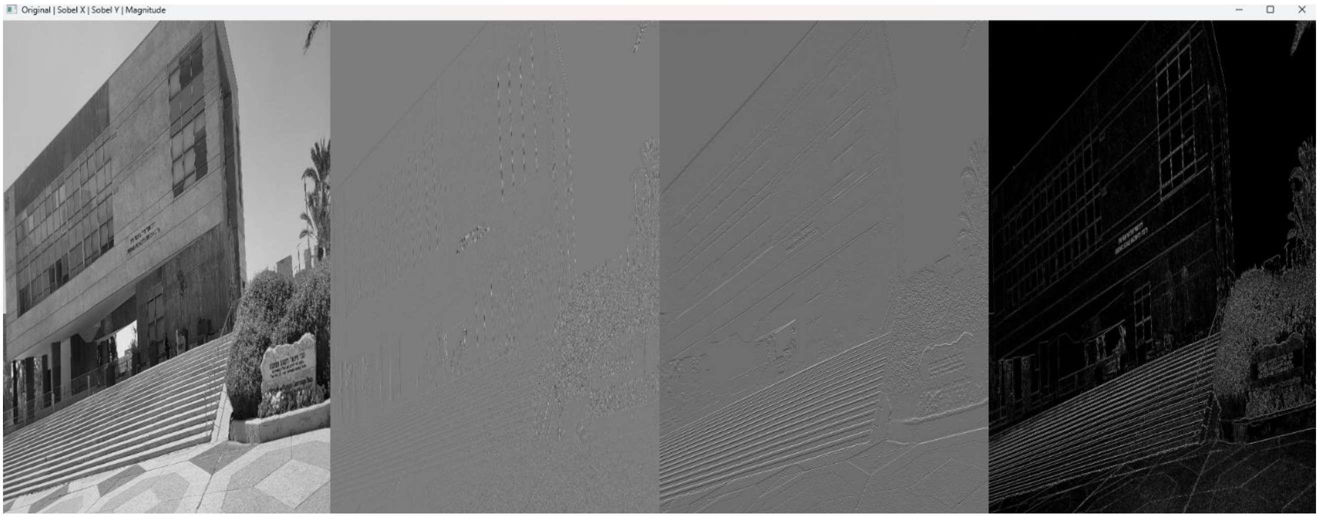
Slide adapted from Aaron Bobick

1. How to find Gradients?

For example - Sobel Filter: (we will use it for the other mathematical background sections)

The Sobel is one of the most commonly used edge detectors. It is based on convolving the image with a small, separable, and integer valued filter in horizontal and vertical direction and is therefore relatively inexpensive in terms of computations. The Sobel edge enhancement filter has the advantage of providing differentiating (which gives the edge response) and smoothing (which reduces noise) concurrently.

Original Sobel-X Sobel-Y Magnitude



$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2} \qquad \mathbf{\Theta} = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

Magnitude Direction

2. Finding Local Maxima Algorithm:

1. Define a neighborhood size within which you want to search for local maximum points.

2. Iterate over each pixel in the **magnitude** image.

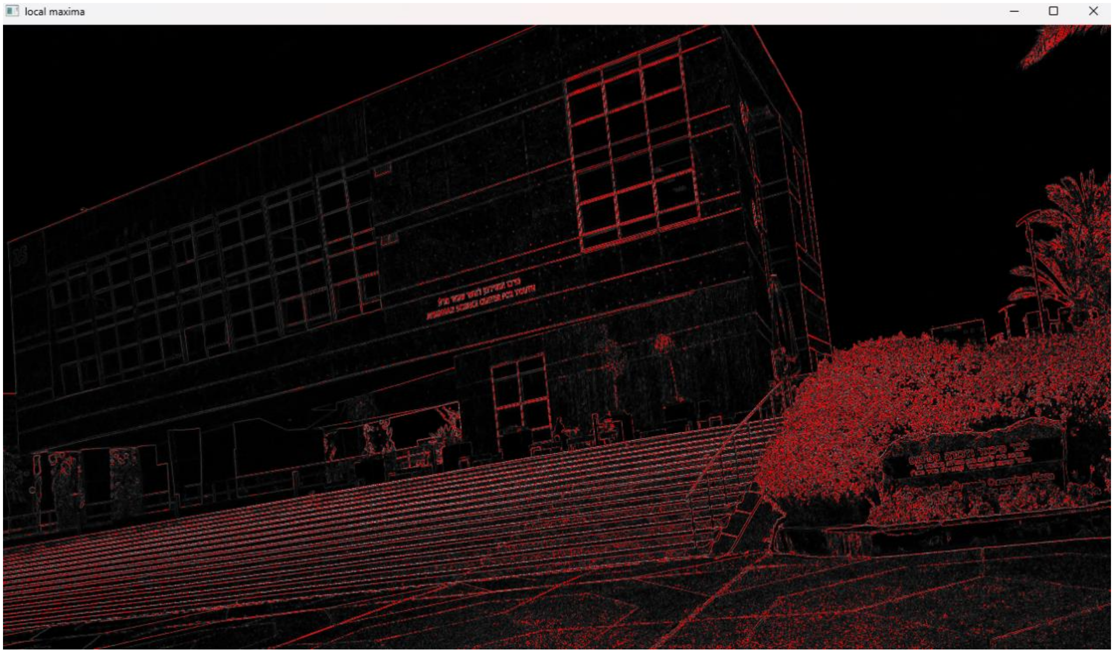
3. For each "local suspect" pixel (pixels with values greater than 125), define a neighborhood around it.

4. Check if the magnitude of the current pixel is greater than the magnitudes of all the neighboring pixels within the defined neighborhood size.

5. If it is greater than all neighbors by at least threshold value (maximum neighborhood's value multiply by the threshold), mark it as a local maximum point.

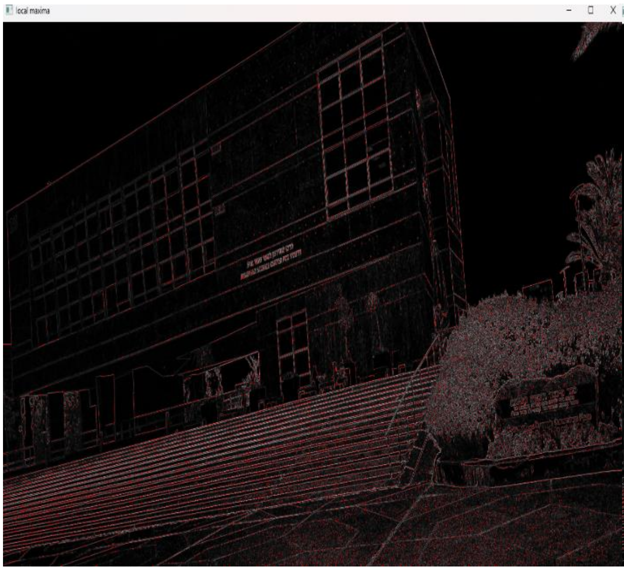
8	4	0
0	237	1
32	64	128

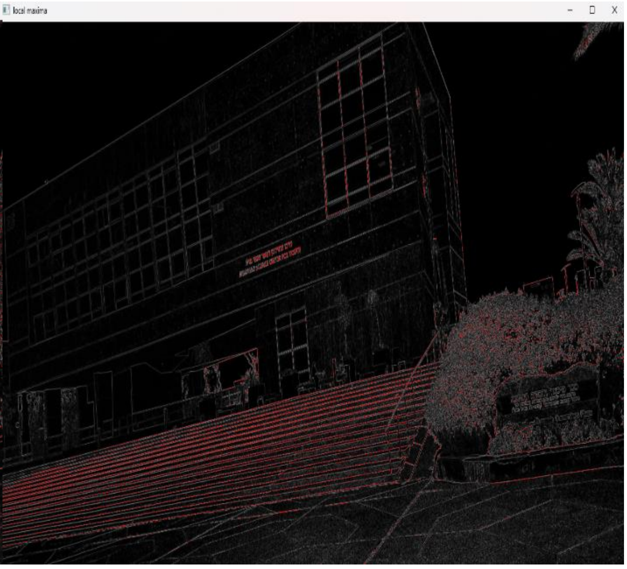
Local Maxima for example in 3x3 neighborhood



Neighorhood 3x3, Suspect Local Maxima pixel Value Threshold = 65, Bigger than neighorhood value Threshold = 0.99

Different Configuration settings:





3. *Image Blurring:*

Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content (eg: noise, edges) from the image. So, edges are blurred a little bit in this operation (there are also blurring techniques which don't blur the edges)

1. Averaging blurring:
   Convolution with average kernel and the image. (for example kernel size 5)

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

2. Median Blurring:

   Instead of replacing the central pixel with the average of the neighborhood, we replace the central pixel with the median of the neighborhood.

3. "Gaussian Blurring", Mathematically is referred to as the convolution of the Gaussian operator and the image. Gaussian blur has a particular expression or “operator” that is applied to each pixel.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

Blurred image

Where G(x,y,sigma) is the Gaussian Blur operator and I(x,y) is an image. x,y are the location coordinates and σ is the “scale” parameter, kind of the amount of blur, greater the value, greater the blur.

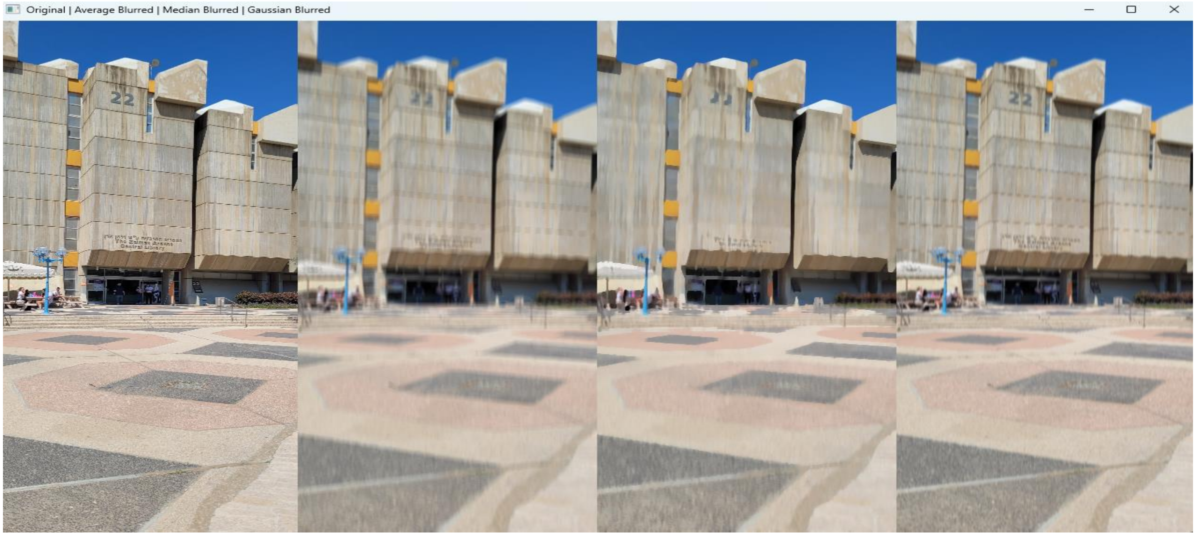
$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

Gaussian Blur operator

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

Original Average Blurring Median Blurring Gaussian Blurring


Original | Average Blurred | Median Blurred | Gaussian Blurred

We assume you already familiar with gradients, Local Maxima and Blurring from ICVL course.

4. *Histogram Of **Oriented** Gradients:*

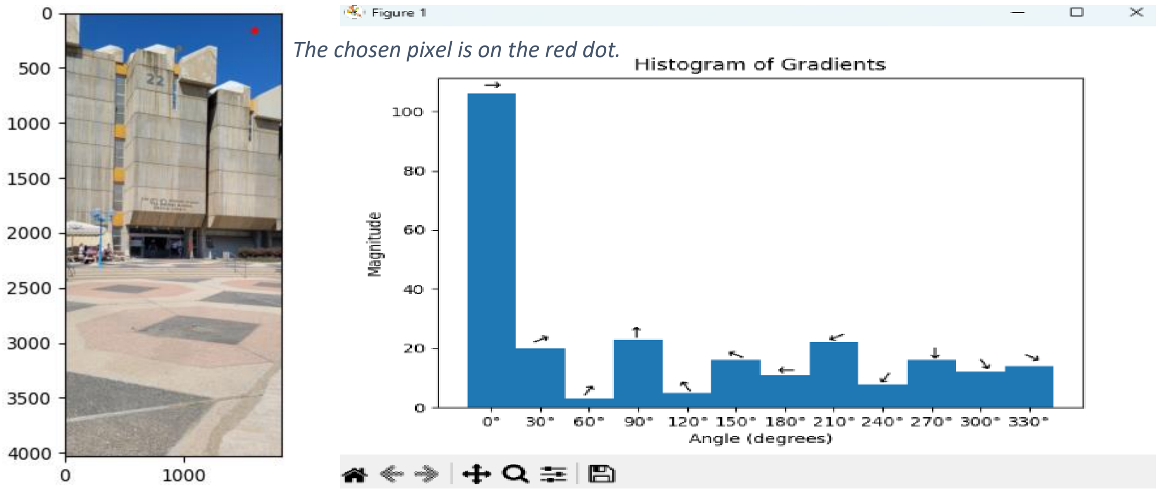
Histograms of gradient orientations are computed based on the gradient magnitudes and orientations of the pixels. These histograms capture the distribution of the gradient orientations within the local region (neighborhood around pixel for example), providing a robust representation of the texture and shape information around the keypoint.

1. Calculate the gradients and their orientations for the image pixels.
2. Choose region (or multiple regions) to calculate the histogram of the gradients.
3. Discretize the angles degrees (0°-360°) to bins.
   <u>For example</u>: 9 bins will lead to possible angles values: [0°,40°,80°,120°…,320°]
   If a pixel has gradient orientation of 45° it will round to 40°.
4. Create a histogram of discretize orientations for the regions.

Important Note:

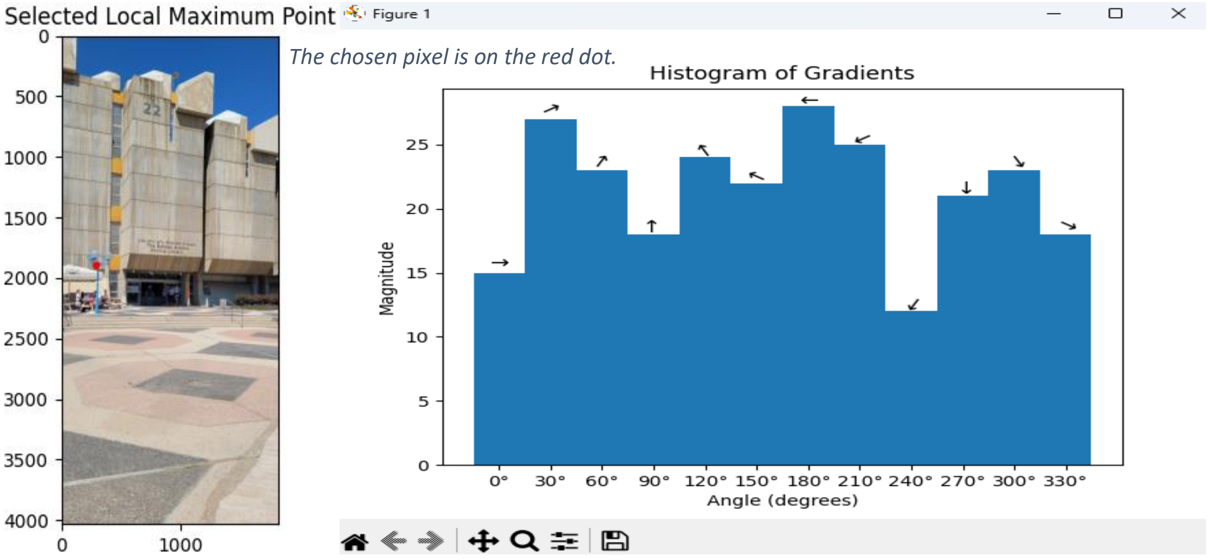
Local maxima, in particular "corners" pixels, tend to have more informative histograms of oriented gradients in their neighborhood than edges pixels and especially more than pixels in homogenous areas.

Example for pixel in homogenous area - the sky (16x16 Neighborhood, 12 bins)



The chosen pixel is on the red dot.

Example for **<u>local maxima</u>** pixel (16x16 Neighborhood, 12 bins)



The chosen pixel is on the red dot.

Our method in brief:

1. We create images database key locations in our lovely Ben Gurion University.

2. We utilize the famous Computer Vision algorithm SIFT - Scale-Invariant Feature Transform, to extract features and their descriptors for each image in our database.

3. Given a query image, we use SIFT again for that image, and then utilize different features matching techniques, including Deep Learning and postprocessing, to find the correct corresponds keypoints matches between the query image and the database images.

4. We use K-Nearest-Neighbor algorithm in order to choose the best location in our database that match the query image location.

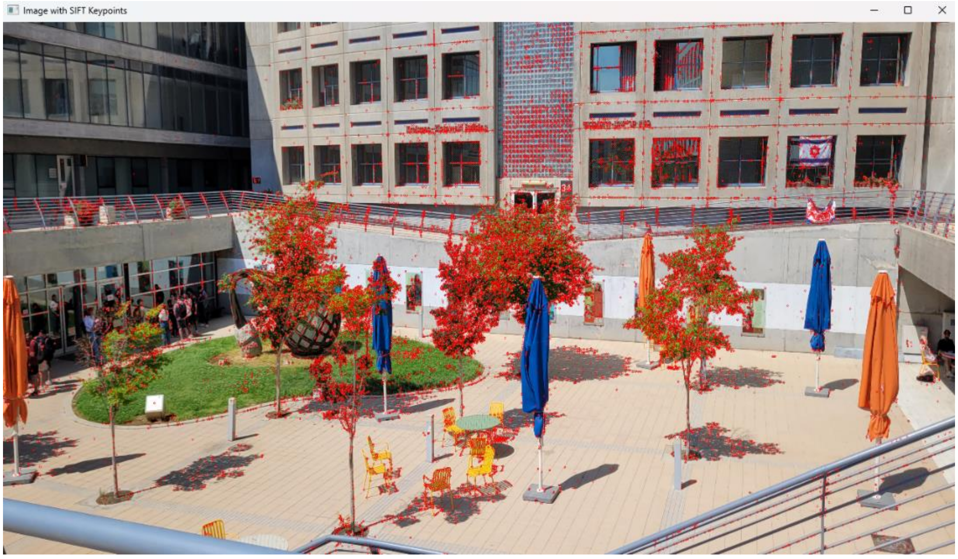
5. Finally, given a destination location, we provide a path in the university map, to get from his location to the destination.

1. Creating the dataset:

a. We took pictures from various locations in our University such as the library, Buildings 37,35,97,25, Student House etc.

b. We took 5-10 pictures from each location with different viewpoints.

2. Use SIFT Algorithm to extract features:



<u>How SIFT works?</u>

- **Scale-space peak selection:** Potential location for finding features.

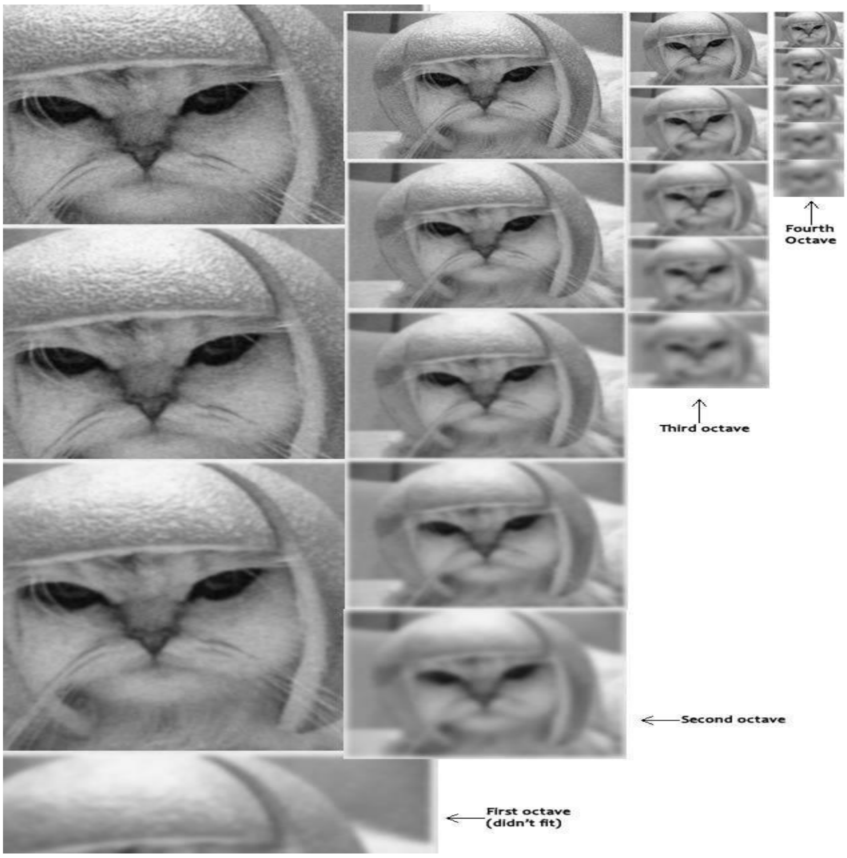
- **Keypoint Localization:** Accurately locating the feature keypoints.

- **Orientation Assignment:** Assigning orientation to keypoints.
- **Keypoint descriptor:** Describing the keypoints as a high dimensional vector.
- **Keypoint Matching**

Scale-space peak Selection

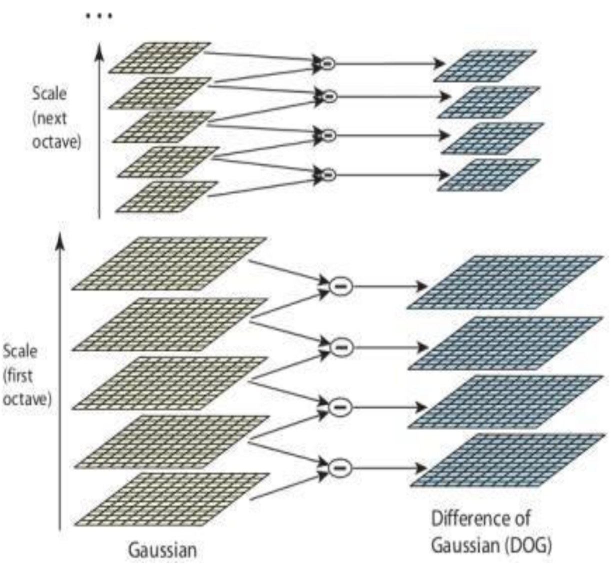
Scale-space (Please refer to mathematical background section 3)

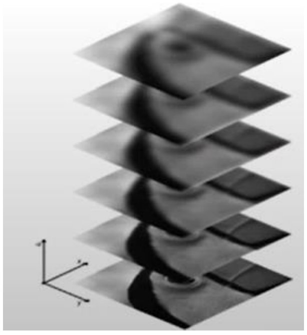
The scale space of an image is a function L(x,y,σ) that is produced from the convolution of a **Gaussian kernel(Blurring)** at different scales with the input image. Scale-space is separated into octaves and the number of octaves and scale depends on the size of the original image. So we generate several octaves of the original image. Each octave's image size is half the previous one.

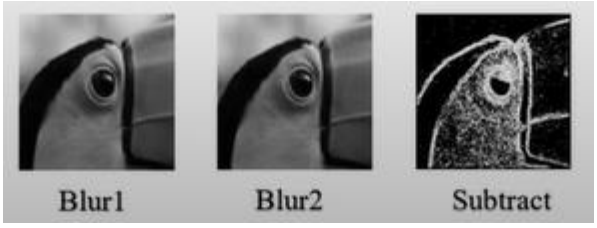


DOG(Difference of Gaussian kernel) (mathematical background section 3)

Now we use those blurred images to generate another set of images, the Difference of Gaussians (DoG). These DoG images are great for finding out interesting keypoints in the image. The difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different σ, let it be σ and *kσ*. This process is done for different octaves of the image in the Gaussian Pyramid. It is represented in below image:

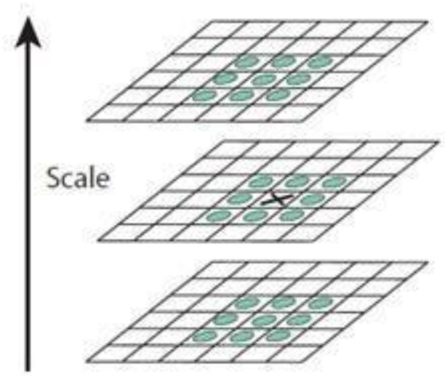






Finding potential keypoints (mathematical background section 2)

One pixel in an image is compared with its 8 neighbors as well as 9 pixels in the next scale and 9 pixels in previous scales. This way, a total of 26 checks are made. If it is a local extrema, it is a potential keypoint. It basically means that keypoint is best represented in that scale.



Keypoint Localization (mathematical background section 1)

Keypoints generated in the previous step produce a lot of keypoints. Some of them lie along an edge, or they don't have enough contrast. In both cases, they are not as useful as features. So we get rid of them. For low contrast features, we simply check their intensities.

Using Taylor series expansion of scale space gets more accurate location of extrema, and if the intensity at this extrema is less than a threshold value (0.03 as per the paper), it is rejected. DoG has a higher response for edges, so edges also need to be removed. Using a 2x2 Hessian matrix (H) to compute the principal curvature, can help to remove edges keypoints. For example we can use 2 sobel filters (one after another) to get the Hessian!

- Reject flats:
  - $|D(\hat{\mathbf{x}})| < 0.03$
- Reject edges:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Let α be the eigenvalue with larger magnitude and β the smaller.

$$\mathrm{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$

$$\mathrm{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

Let r = α/β.
So α = rβ

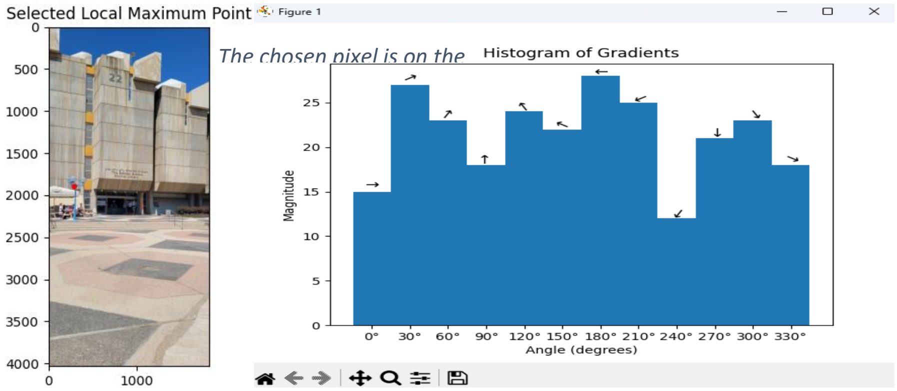
$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

(r+1)²/r is at a min when the 2 eigenvalues are equal.

  - $r < 10$

Orientation Assignment: (mathematical background section 4)

The next thing is to assign an orientation to each keypoint to make it rotation invariance. We compute the Histogram of gradients of the keypoint, and the highest pick (the highest magnitude), it's the "main orientation" of that keypoint. We can use that later in the matching process between another keypoint to eliminate the rotations between the keypoints using their main orientations difference.



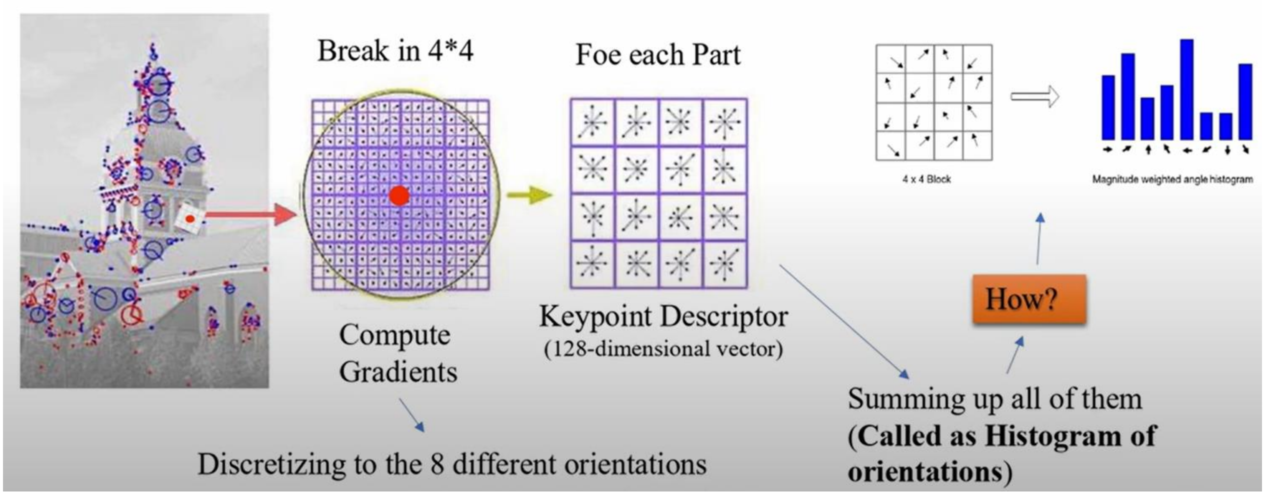
The chosen pixel is on the

The main orientation for that keypoint, is 180 degrees.

Keypoint descriptor (mathematical background section 4)

Next is to compute a descriptor for the local image region about each keypoint that is highly distinctive and invariant as possible to variations such as changes in viewpoint and illumination.

1. To do this, a 16x16 window around the keypoint is taken. It is divided into 16 sub-blocks of 4x4 size.
2. For each sub-block, 8 bin orientation histogram is created.
3. So 4 X 4 descriptors over 16 X 16 sample array were used in practice. 4 X 4 X 8 directions give 128 bin values. It is represented as a feature vector to form keypoint descriptor.


Break in 4*4

Foe each Part

Compute Gradients

Keypoint Descriptor (128-dimensional vector)

How?

Summing up all of them **(Called as Histogram of orientations)**

Discretizing to the 8 different orientations

This feature vector introduces a few complications. We need to get rid of them before finalizing the fingerprint.

- **Rotation dependence** The feature vector uses gradient orientations. Clearly, if you rotate the image, everything changes. All gradient orientations also change. To achieve rotation independence, the keypoint’s rotation is subtracted from each orientation. Thus each gradient orientation is relative to the keypoint’s orientation.

- **Illumination dependence** If we threshold numbers that are big, we can achieve illumination independence. So, any number (of the 128) greater than 0.2 is changed to 0.2. This resultant feature vector is normalized again. And now you have an illumination independent feature vector!

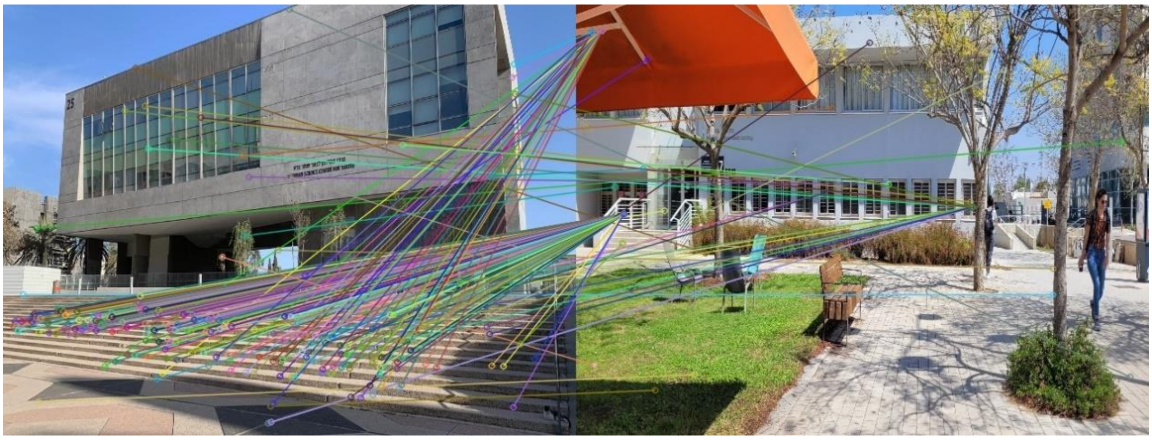
Keypoint Matching:

Keypoints between two images are matched by identifying their nearest neighbors. But in some cases, the second closest-match may be very near to the first. It may happen due to noise or some other reasons. In that case, the ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected. It eliminates around 90% of false matches while discards only 5% correct matches, as per the paper.

To measure the distance between the descriptors we can use:

$$L2\_Norm(\mathbf{descriptor}_1, \mathbf{descriptor}_2)$$

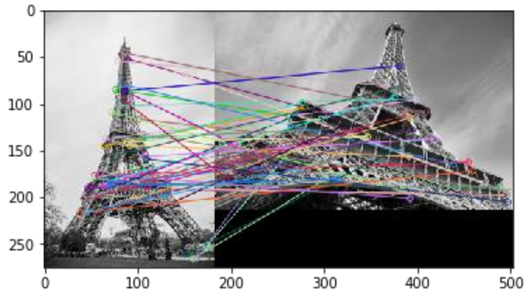
Using SIFT out of the box just **DOESN'T WORK!!!**



Results for image taken from building 97. As you can see, its not the same building (SIFT found wrongly building 25 as the best match)

The tradeoff between more matches and False positives:

Lowering SIFT's thresholds enhances its robustness in detecting keypoints across different perspectives, aiding to recognize the same place from varying viewpoints and perspectives.



However, this adjustment increases the **occurrence of false positive matches**, where keypoints are incorrectly associated between wrong images due to noise or misleading patterns.

Unfortunately, the number of false positive are too high for the image retrieval task.

Why it doesn't work?

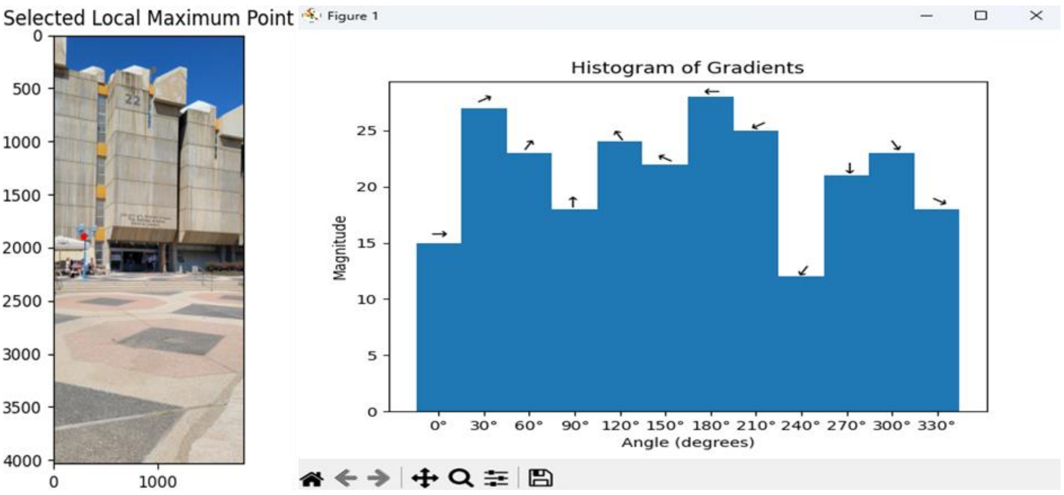
Even trying different techniques for feature detection as **ORB,SURF,AKAZE** and different image enhancement like **Histogram Equalization, White Balance**, we still got bad results.

After digging and thinking a little bit further we revised the idea of SIFT descriptors again. SIFT descriptors are gradient oriented based, which can lead to even more complications!

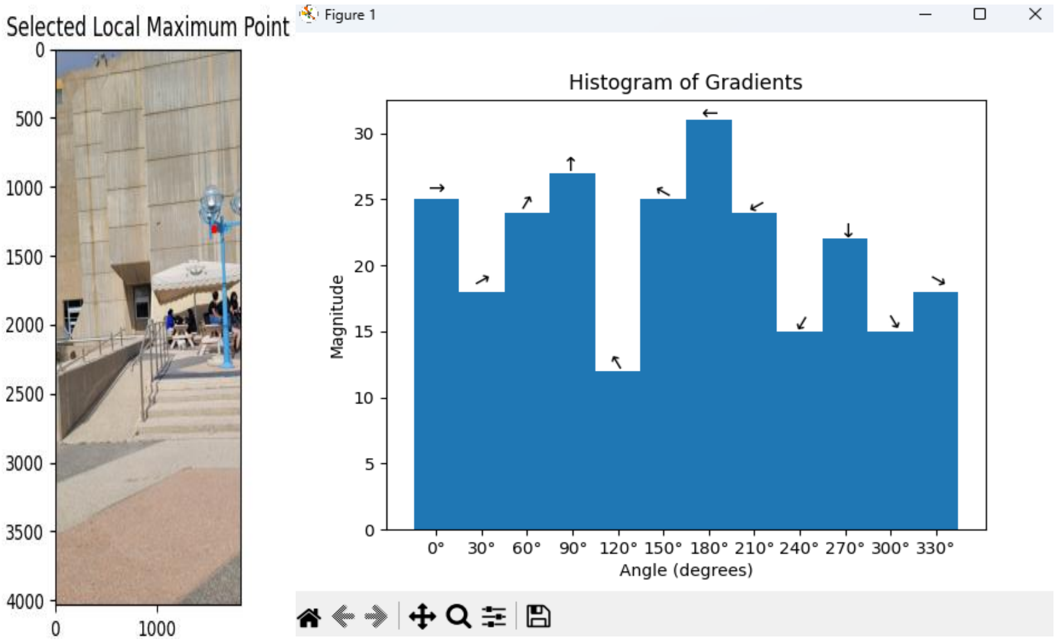
SIFT may be invariant to yaw rotation but here the task is more complicated. We take pictures from **different perspectives and different cameras which leads to totally different histograms of gradients for the same keypoint**. **(The pitch and the roll angles are different)**

Example: The red keypoint on the blue lamp.

Perspective 1:



Perspective 2: (Histogram after SIFT rotation invariant approach using the main orientation)



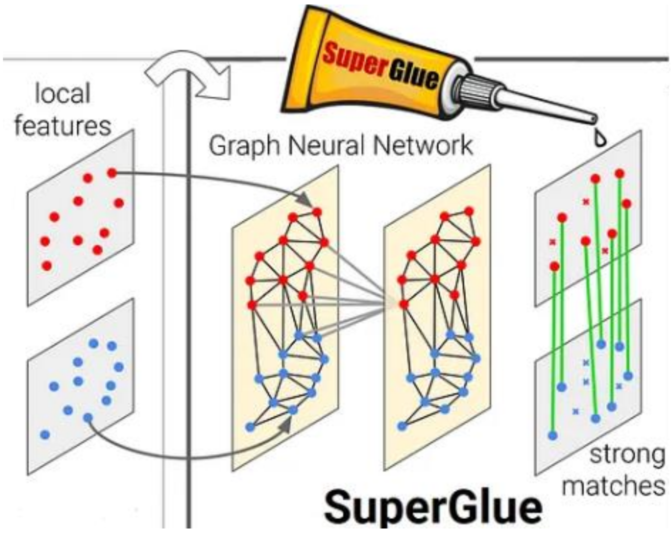
As you can see, the histograms for the same keypoint are different and the L2-Norm (or Hamming distance) matching will lead to wrong results.

3. SIFT Keypoints Matching

We replaced the matching process of the SIFT descriptors by Deep Learning approach. We used pretrained model for keypoints matching called LightGlue!

- Deep neural network for matching keypoints in images
  - Input: Descriptors Vector 128 Dim - Image1,
    Descriptors Vector 128 Dim - Image2
  - Output: Matching Between the keypoints

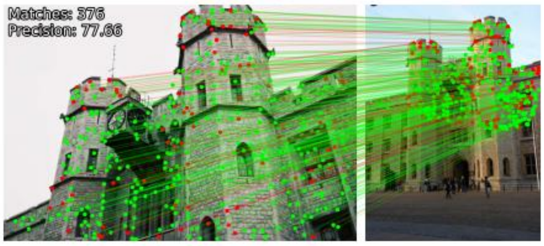
- An improvement to “SuperGlue”, the previous state-of-the-art method
- Achieves high accuracy while being much faster
- Adaptively reduces complexity for easy image pairs



LightGlue is based on graph neural network and cross-attention and transformers mechanism, making it stronger and more robust matcher.

Lightglue was chosen because it the fastest, SOTA algorithm in our current time. (ICCV 2023)

SIFT+LightGlue



Note : In our first try, we replace the whole process, using DISK for feature extraction and LightGlue for matching, but we wanted to tackle the challenge to use mostly classical computer vision algorithms for our project for educational purpose and ICVL course related material.

You can read further in LightGlue paper.

Improvements: (Deep Learning matcher doesn't solve it alone)

Although we can finally find logical corresponds keypoints, there are many outliers that causes our system to fail in some common cases.

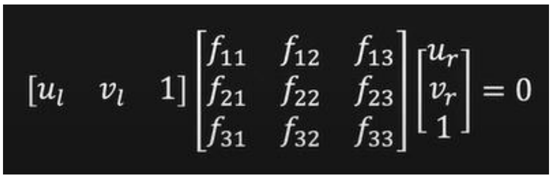
In order to filer outliers matches, we used our corresponds keypoints to solve another problem that can filter that outliers.

Fundamental matrix (Epipolar Geometry)

We used Fundamental matrix that represents a geometric relationship between two images of the same scene taken from different viewpoints and cameras.

You can read further about it here.

The fundamental matrix constraint:



$$[u_l \quad v_l \quad 1]\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}\begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = 0$$

Where Fij are unknown and u_l,v_l is a feature (keypoint location) in the left image, and the u_r,v_r is the correspond feature (keypoint location) in the right image.

Using Least Square to solve the linear system

We can write the constraint equations in Least Square problem:

$$\begin{bmatrix} u_1u_1{'} & u_1v_1{'} & u_1 & v_1u_1{'} & v_1v_1{'} & v_1 & u_1{'} & v_1{'} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_nu_v' & u_nv_n{'} & u_n & v_nu_n{'} & v_nv_n{'} & v_n & u_n{'} & v_n{'} & 1 \end{bmatrix}\begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ \vdots \\ f_{33} \end{bmatrix} = \mathbf{0}$$

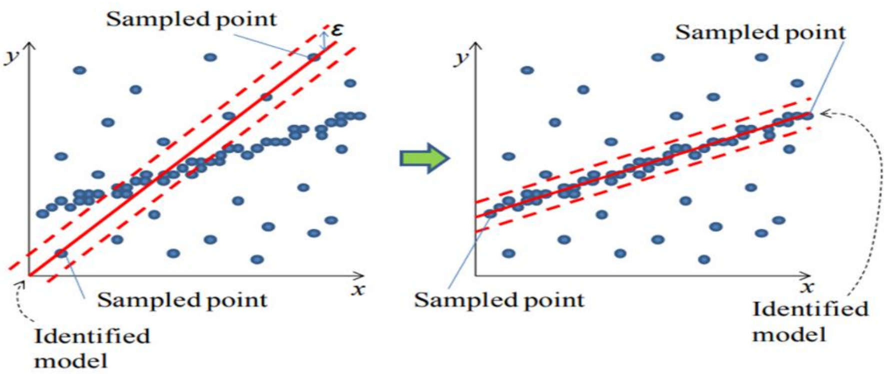
RANSAC (Robust Maximum Likelihood Estimation) algorithm

In order to solve linear least squares robustly, we can use RANSAC.

How the algorithm works?

- Sample the minimum required number of datapoints to solve the system.
- Open an epsilon distance space around that solution.
- Count how many datapoints inside that space
- Save the minimum required number of datapoints, that lead to max count in step 3.
- Go again to step 1 (until you reach maximum sampling) , remember the solution from step 4.

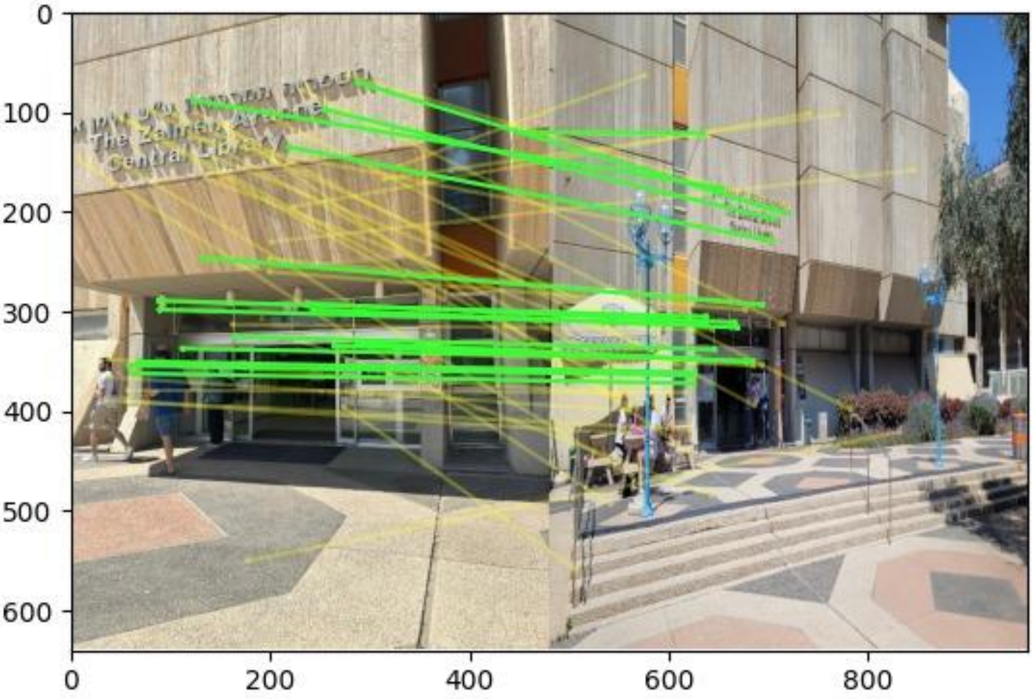
Return the solution from step 4.



Results:

We considered the outliers from the Robust Least Squares (datapoints that far away from the solution of the system) as outliers keypoints and matches.

As you can see, the yellow lines are the rejected matches (outliers) found by our method.



The green lines represent the correct final keypoints matches of our method.

4. Choose the best location for the query image.

Naive approach:

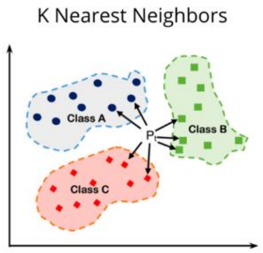
The naive idea will return the location of the image in our database that have the most of the matches with the query image.

Due noise in the images and the keypoints extracting and matching process, sometimes the best match is actually a wrong image location while the second best match is the right one.

Better approach:

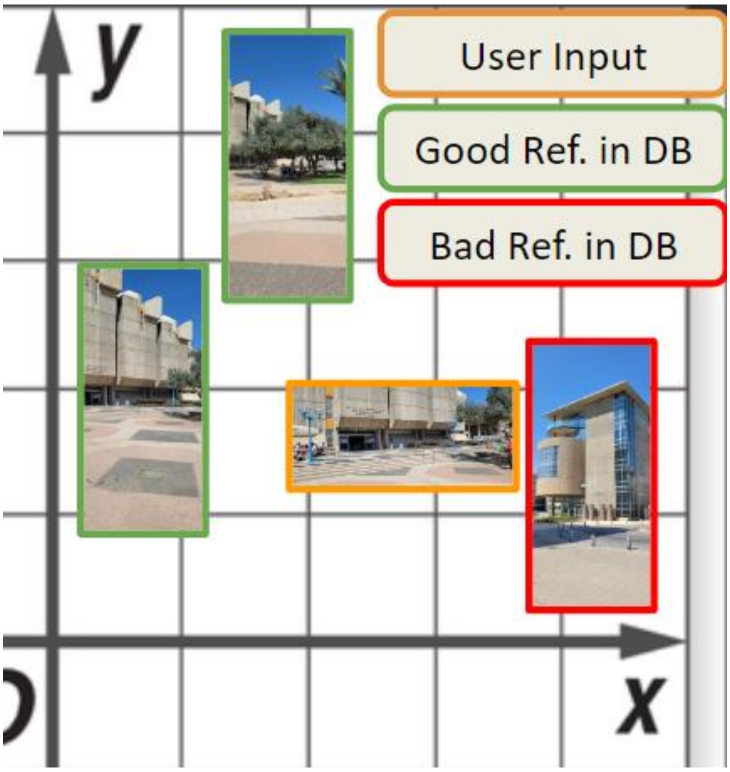
We can use the idea of K-Nearest-Neighbors ("Nearest" is the images with more corresponds matches) to find the right solution.

The KNN algorithm is a simple yet powerful method for feature matching and classification.



We will return the location of the majority top 5 (Nearest) database images with our query image.

A real example from our algorithm with K=3.

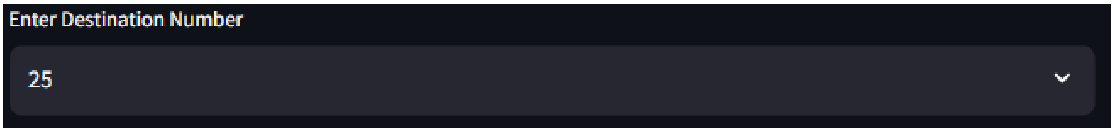


Our method will return the correct location while the naive approach will fail.
We found that K=5 is best for our approach and database.

5. Returning a path from the location we found to the query destination.

A picture is worth a thousand words.

A real example from our app:

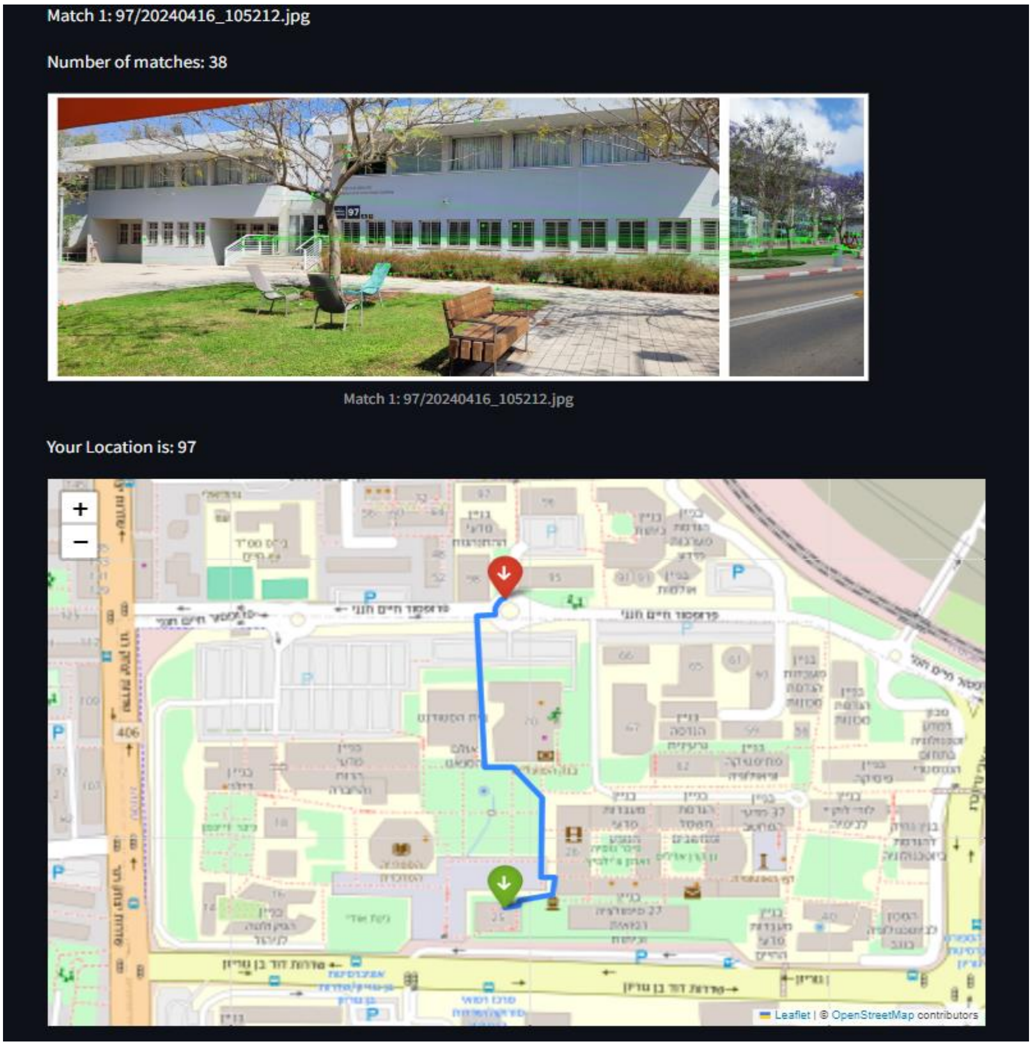


Enter Destination Number

25

The query image is on the right.

Even for image without rich information, from extreme different perspective from our database images and lower resolution image (comparing to our database), We created a strong and robust solution.



Match 1: 97/20240416_105212.jpg

Number of matches: 38

Match 1: 97/20240416_105212.jpg

Your Location is: 97

Our application:

We used Streamlit framework to create our cross-platform website, and used Streamlit-Cloud to host online our application, making it available for any user to use our app, without the need to run it locally.

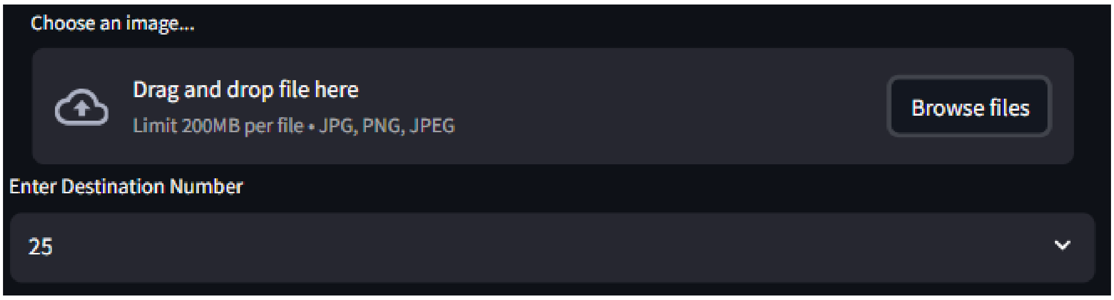
Step 1: Connect to https://the-lost-student-app.streamlit.app/ to use our app online.

If it's not available, you can use it locally in your computer using
1. git clone -b sift-lightglue https://github.com/RoyAmoyal/The_Lost_Student_App

2. (Create virtual env/conda env python=3.11) pip install -r requirements.txt

3. In terminal/CommandLine run:
streamlit run streamlit_main_app.py

Step 2: Choose an image and destination



Choose an image...

Drag and drop file here

Limit 200MB per file • JPG, PNG, JPEG

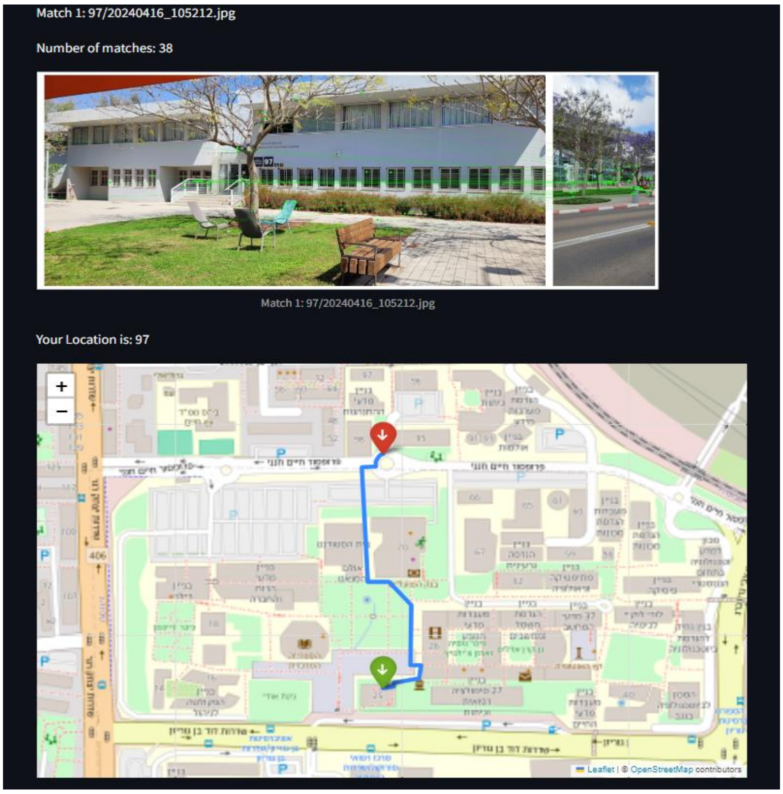
Enter Destination Number

25

Step 3: Wait for the process to finish, it make take few seconds.

Note: in Streamlit Cloud link the calculations will take more time because it's a free server with really low computations capabilities)

Step 4: You can see the best image match to your location and you get a map with a path for your destination!



Match 1: 97/20240416_105212.jpg

Number of matches: 38

Match 1: 97/20240416_105212.jpg

Your Location is: 97

Conclusion:

We had a fantastic time working on our final project, which aimed to help students navigate the university campus using images. It was a challenging but incredibly rewarding experience as we gained valuable knowledge. The course materials served as our reliable guide, leading us through complex algorithms like SIFT. Exploring concepts such as gradients, orientations, and blurring techniques was both difficult and fascinating.

However, the real highlight was the project itself. We had the opportunity to experiment with various feature extraction and matching techniques, collaborating and brainstorming together. It wasn't just about completing the task; it was about exploring new ideas, innovating, and growing as a team. By the end, we not only enhanced our technical skills but also developed a stronger sense of teamwork and appreciation for algorithmic design.

Thanks!

References:

https://medium.com/jun94-devpblog/cv-10-local-feature-descriptors-harris-and-hessian-corner-detector-7d524888abfd

https://medium.com/@deepanshut041/introduction-to-sift-scale-invariant-feature-transform-65d7f3a72d40

https://www.geeksforgeeks.org/hamming-distance-two-strings/

https://github.com/cvg/LightGlue

https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html

https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/

https://www.youtube.com/watch?v=6kpBqfgSPRc&ab_channel=FirstPrinciplesofComputerVision

https://www.youtube.com/watch?v=D_Lm7NitBoA&t=2407s&ab_channel=TomaszMalisiewicz

https://streamlit.io/