Distinctive Image Features from Scale-Invariant Keypoints

Project ID - 3 | Team name - Team Agile

Samruddhi Shastri (2019111039)
Pranjali Pathre (2019112002)
Anandhini Rajendran (2019101055)
Ayush Goyal (2019111026)

Github link

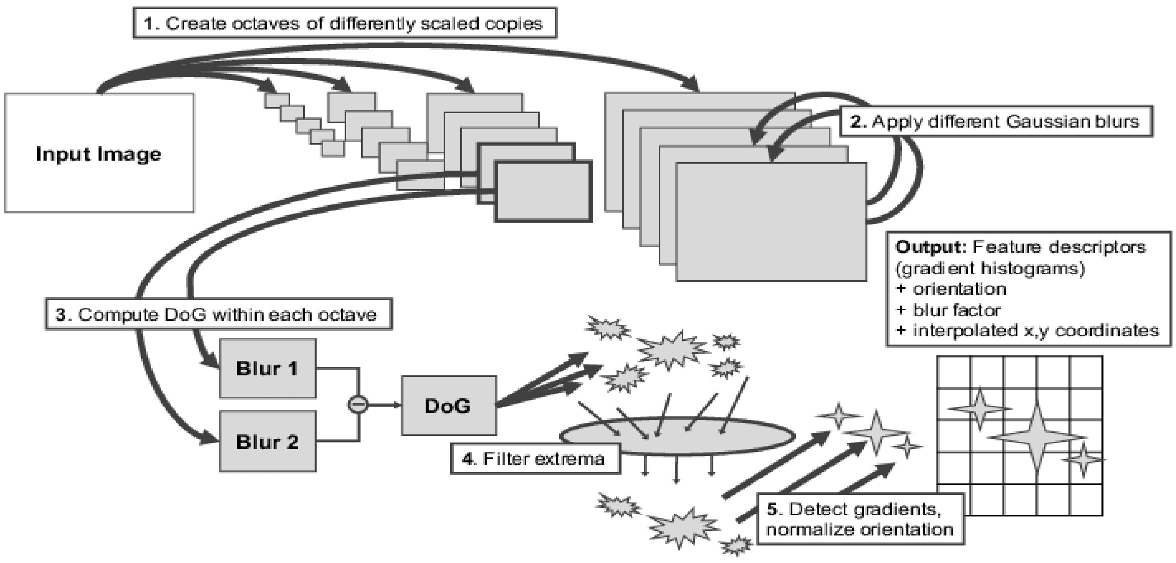
https://github.com/pranjali-pathre/SMAI-Project

Problem Statement

Image matching is a fundamental aspect of many problems in computer vision, including object or scene recognition, solving for 3D structure from multiple images, stereo correspondence, and motion tracking.

We present a method for extracting distinctive invariant features from images that can be used to perform reliable matching between different views of an object or scene. The obtained features are invariant to image scaling and rotation, partially invariant to illumination, well localized in spatial and frequency domains, and highly distinct which allows us to match features with high accuracy. They are well localized in both the spatial and frequency domains, reducing the probability of disruption by occlusion, clutter, or noise. The cost of extracting the features is reduced by using a cascade filtering approach. This approach to recognition can robustly identify objects among clutter and occlusion while achieving near real-time performance.

TEASER IMAGE


1. Create octaves of differently scaled copies
Input Image
2. Apply different Gaussian blurs
Output: Feature descriptors (gradient histograms)
+ orientation
+ blur factor
+ interpolated x,y coordinates
3. Compute DoG within each octave
Blur 1
Blur 2
DoG
4. Filter extrema
5. Detect gradients, normalize orientation

Goals and Approach

SIFT Feature Extraction is implemented in the following steps:

1. Scale-space extrema detection
2. Keypoint localization
3. Orientation assignment
4. Keypoint descriptor

These sections are briefly described below.

The sample image used is:

Implementing Scale Space and Image Pyramids

Theory:

The concept of Scale Space deals with the application of a continuous range of Gaussian Filters to the target image such that the chosen Gaussian has differing values of the sigma parameter.

1. We use theImplementing Scale Space and Image Pyramids Gaussian function as a scale-space kernel. So the scale space of an image is defined as a result of convolution of a variable-scale Gaussian, *G(x, y, σ)* with an input image.
$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$$
2. We use scale-space extrema, i.e., the convolved Gaussian difference to detect the stable points.
$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$$
The maxima and the minima $\sigma^2\nabla^2 G$ produce the most stable image features.
$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2\nabla^2 G$$

The above process is implemented in the code as below.

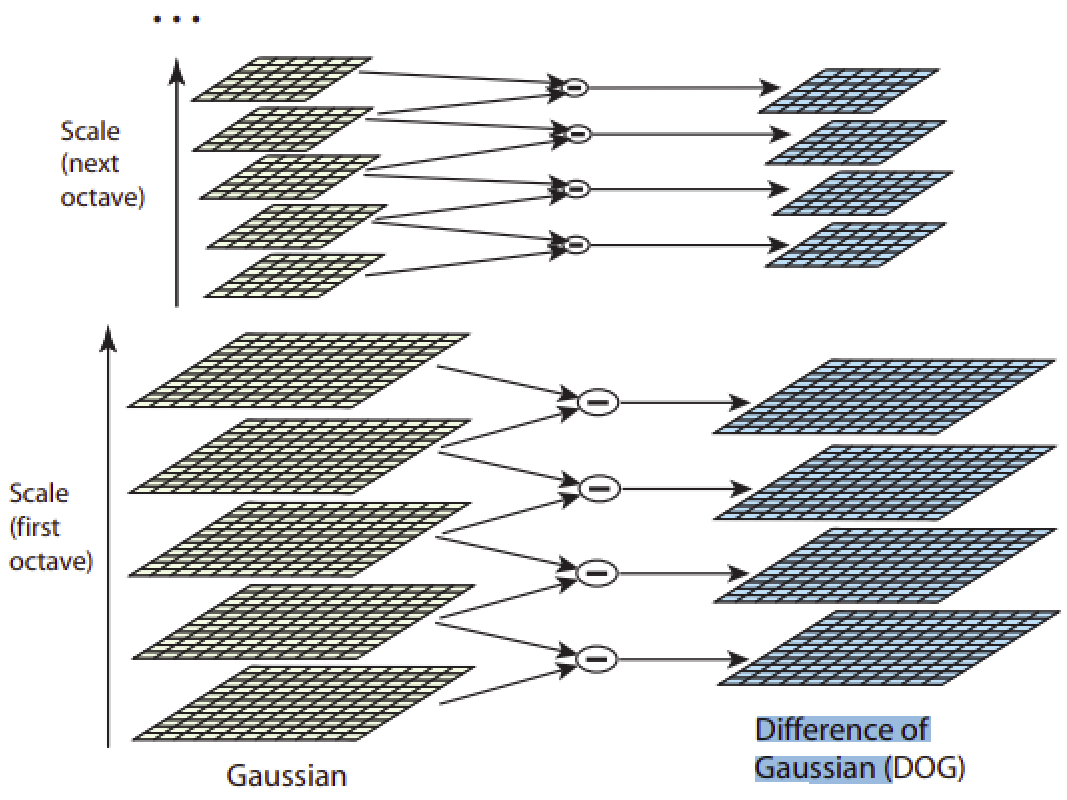
Procedure:

This section primarily included the implementation of setting the number of octaves, generating the Gaussian kernels and images, and the Difference of Gaussian.

1. Setting the number of octaves:
The number of octaves in an image pyramid is a function of the shape of the base image. It represents the number of times we can repeatedly halve an image until it becomes too small. If *l* is the shorter side length of the image, then we have the following relation between the number of octaves and the length of the shortest side.
$$l/2^x = 1$$
Here *x* is the number of times we can halve the base image. We can take the logarithm of both sides and solve for *x* to obtain the following:
$$x = log(y)/log(2).$$
We round *x* down to the nearest integer (floor(x)) to have an integer number of layers in our image pyramid.

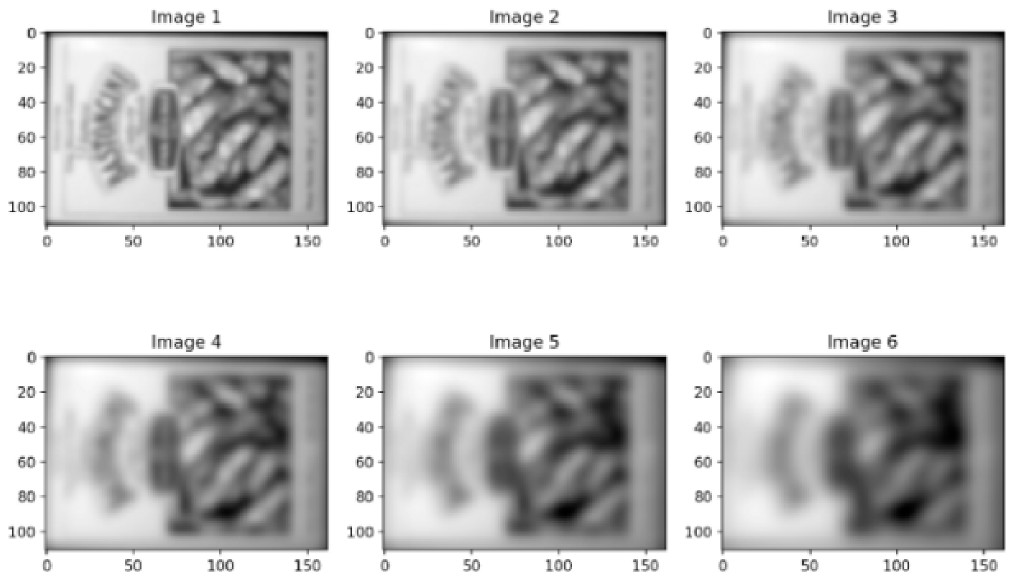
2. Generating the Gaussian kernels and images :
The initial image is incrementally convolved with Gaussians to produce images separated by a constant factor k in scale space, shown stacked in the left column. We choose to divide each octave of scale-space (i.e., doubling of σ) into an

integer number, s, of intervals, so $k = 2^{1/s}$. We must produce s + 3 images in the stack of blurred images for each octave so that final extrema detection covers a complete octave.



For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeated.

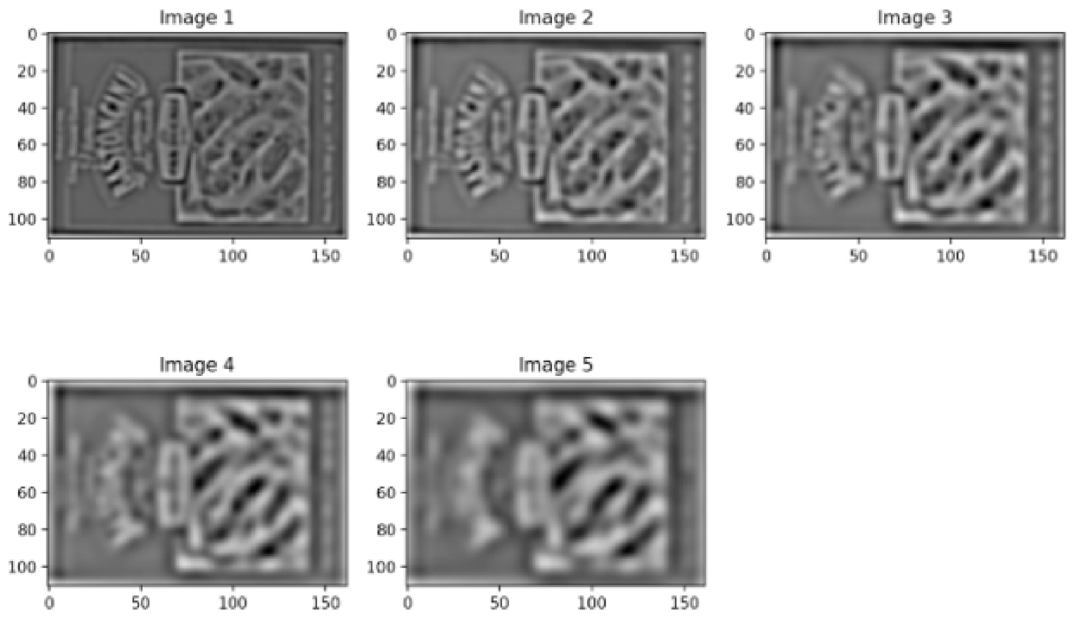
Generating the Gaussian kernels sets the amount of blur for each image belonging to an octave. Each octave has a (number of intervals + 3) image. All the images in an octave have the same dimensions but varying amounts of blur added to each image. We add two extra images at the beginning and the end to have a total number of intervals + 3 when the (number of intervals +1) images are covered. We start with the initial value set for sigma to blur the image.



Initial image obtained with varying blur belonging to one particular octave

3. <u>Generating the Gaussian images difference of Gaussian:</u>
   Next for generating Gaussian images, we blur each image according to the value set in the Gaussian kernel.
   Now that we have Gaussian images, for generating the difference of Gaussian images, we subtract two successive pairs of blurred images.

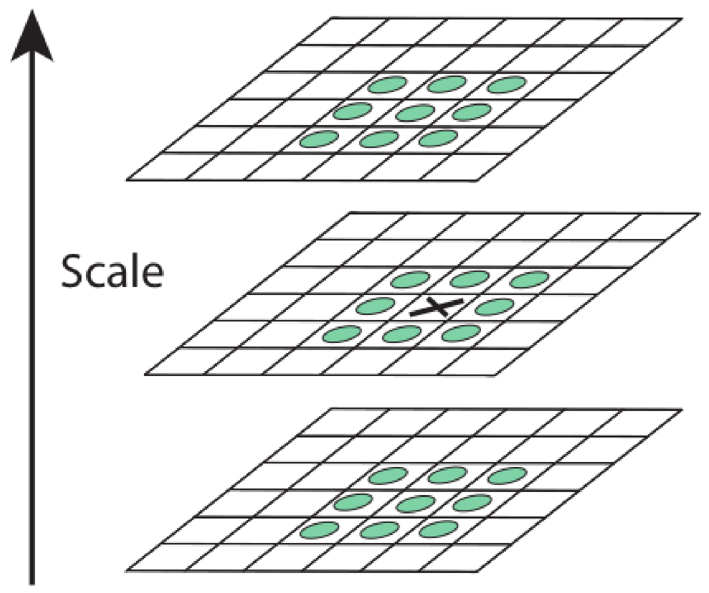
   The output of the sample image used for a particular octave is displayed below.



Scale-space extrema detection

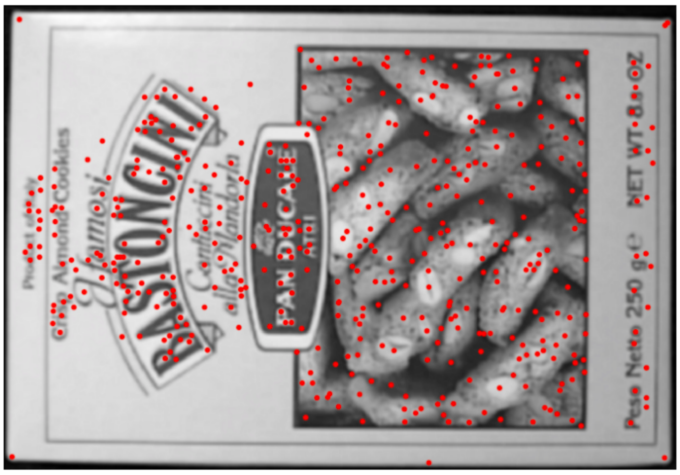
Here we aim to detect the locations that are invariant to scale change of the image. The scale-space difference-of-Gaussian function has a large number of extrema and that it would be very expensive to detect them all. Fortunately, we can detect the most stable and useful subset even with a coarse sampling of scales.

In order to detect the local maxima and minima of D(x, y, σ), each sample point is compared to its eight neighbors in the current image and nine neighbors in the scale above and below as shown in the figure below.



Maxima and minima of the difference-of-Gaussian images are detected by comparing a pixel (marked with X) to its 26 neighbors in 3x3 regions at the current and adjacent scales (marked with circles).

In each triplet of images, we look for pixels in the middle image that are greater than or less than all of their 26 neighbors: 8 neighbors in the middle image, 9 neighbors in the image below, and 9 neighbors in the image above. These are our maxima and minima. When we've found an extremum, we localize its position at the subpixel level along all three dimensions (width, height, and scale) using localizeExtremumViaQuadraticFit(), explained in more detail below.



Our initial key points plotted over the base image.

Accurate Keypoint Localization

Theory

Once a keypoint candidate has been found by comparing a pixel to its neighbors, the next step is to perform a detailed fit to the nearby data for location, scale, and ratio of principal curvatures.

1. We use the Taylor expansion (up to the quadratic terms) of the scale-space function, D(x, y, σ), shifted so that the origin is at the sample point

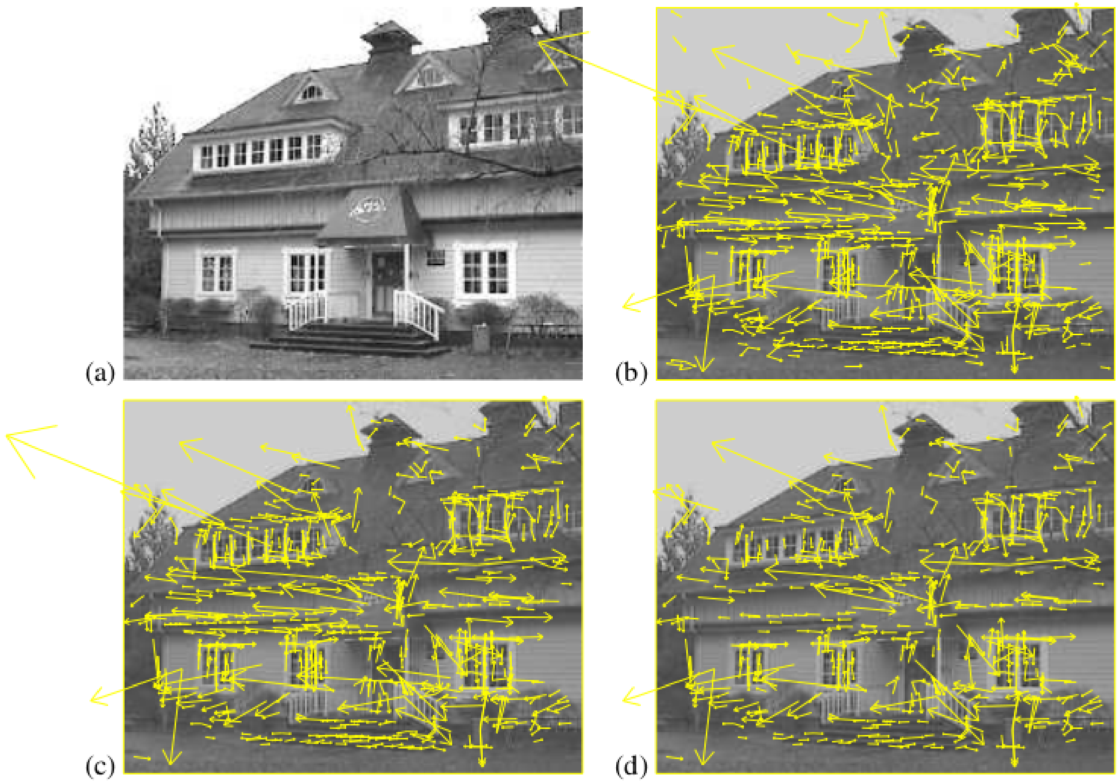
$$D(\mathbf{x}) = D + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^\mathbf{T} \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

   D and its derivatives are evaluated at the sample point and x = (x, y, σ)^T is the offset from this point

2. The location of the extremum, x̂, is determined by taking the derivative of this function with respect to x and setting it to zero, giving:

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} \frac{\partial D}{\partial \mathbf{x}}$$

3. If the offset x̂ is larger than 0.5 in any dimension, then it means that the extremum lies closer to a different sample point. In this case, the sample point is changed and the interpolation is performed instead. The final offset x̂ is added to the location of its sample point to get the interpolated estimate for the location of the extremum.



This figure shows the stages of keypoint selection.
(a) The 233x189 pixel original image.
(b) The initial 832 keypoints locations at maxima and minima of the difference-of-Gaussian function. Keypoints are displayed as vectors indicating scale, orientation, and location.
(c) After applying a threshold on minimum contrast, 729 key points remain.
(d) The final 536 key points that remain following an additional threshold on the ratio of principal curvatures.

Eliminating edge responses:

- For stability, it is not just sufficient to reject keypoints with low contrast. The difference-of-Gaussian function will have a strong response along edges, even if the location along the edge is poorly determined and therefore unstable to small amounts of noise.
- A poorly defined peak in the difference-of-Gaussian function will have a large principal curvature across the edge but a small one in the perpendicular direction.
- Therefore we select those key points that have at least two large eigenvalues along the edges.

Procedure

A quadratic model is fitted to the input keypoint pixel and all 26 of its neighboring pixels. We update the keypoint's position with the sub-pixel-accurate extremum estimated from this model. We iterate at most 5 times until the next update moves the keypoint less than 0.5 in any of the three directions. This means the quadratic model has converged to a one-pixel location. The two helper functions centerPixelGradientComputation() and centerPixelHessianComputation() implement second-order central finite difference approximations of the gradients and Hessians.

Generating Descriptors

Theory

Descriptors: They encode information about the neighborhood of key points. It is used to compare key points.

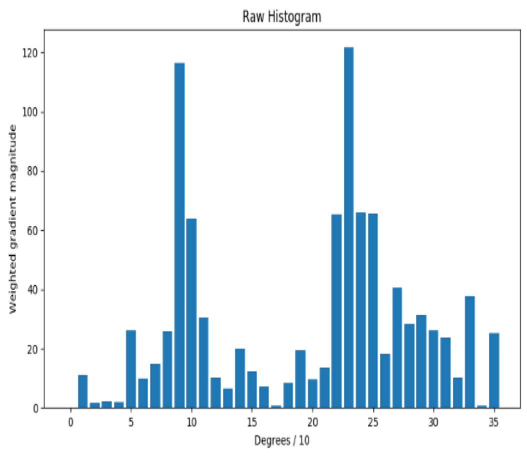
Orientation Assignment:

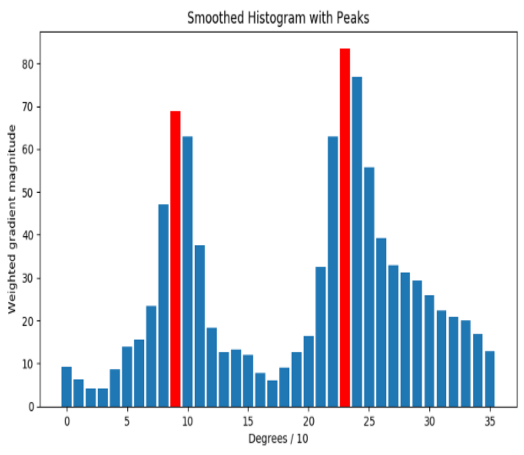
- By assigning a consistent orientation to each keypoint based on local image properties, the keypoint descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation.
- For each image sample, L(x, y), at this scale, the gradient magnitude, m(x, y), and orientation, θ(x, y), is precomputed using pixel differences:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = \tan^{-1}((L(x, y+1) - L(x, y-1))/(L(x+1, y) - L(x-1, y)))$$

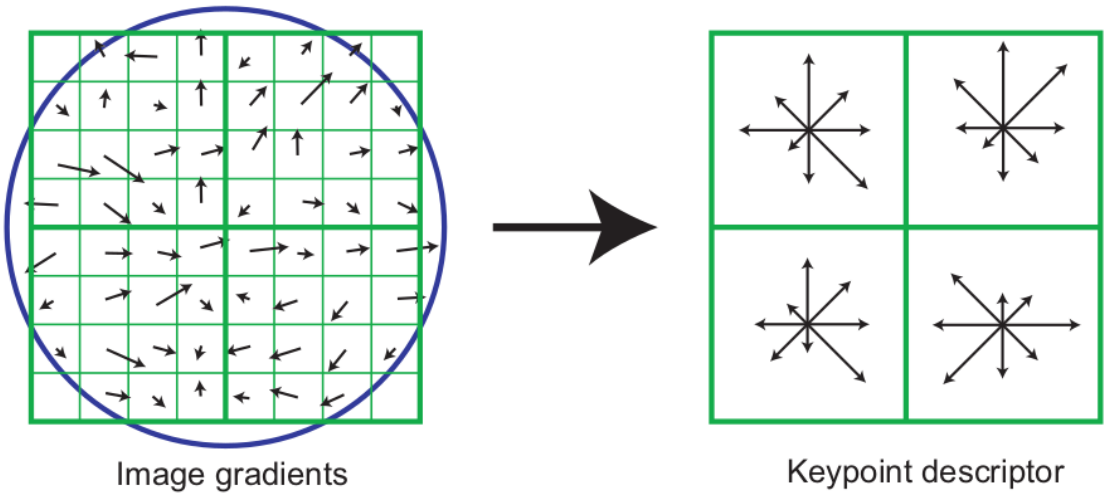
- An orientation histogram is formed from the gradient orientations of sample points within a region around the keypoint.
- The orientation histogram has 36 bins covering the 360 degree range of orientations.
- Peaks in the orientation histogram correspond to dominant directions of local gradients.
- The highest peak in the histogram is detected, and then any other local peak that is within 80% of the highest peak is used to also create a keypoint with that orientation.





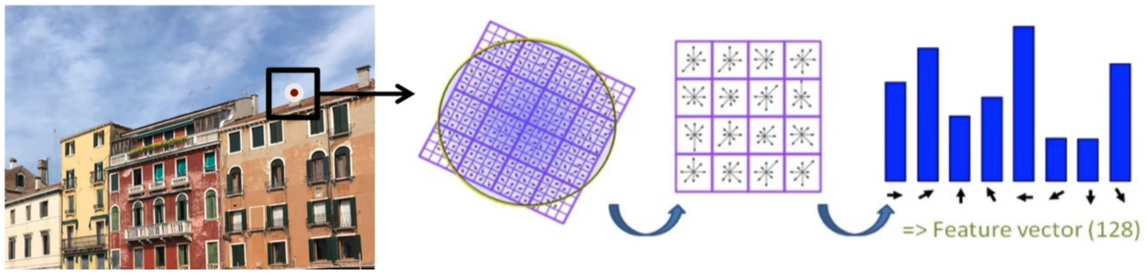
The local image descriptor:

- Here we compute a descriptor for the local image region that is highly distinctive yet is as invariant as possible to remaining variations, such as change in illumination or 3D viewpoint.
- First, the image gradient magnitudes and orientations are sampled around the keypoint location, using the scale of the keypoint.
- Next, to achieve orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the keypoint orientation.
- We create orientation histograms over 4x4 sample regions. The figure shows eight directions for each orientation histogram, with the length of each arrow corresponding to the magnitude of that histogram entry.



- Trilinear interpolation is used to distribute the value of each gradient sample into adjacent histogram bins to avoid boundary effects.
- The descriptor is formed from a vector containing the values of all the orientation histogram entries, corresponding to the lengths of the arrows on the right side of the above figure.
- Finally, the feature vector is modified to reduce the effects of illumination change by first normalizing it to unit length.

Procedure

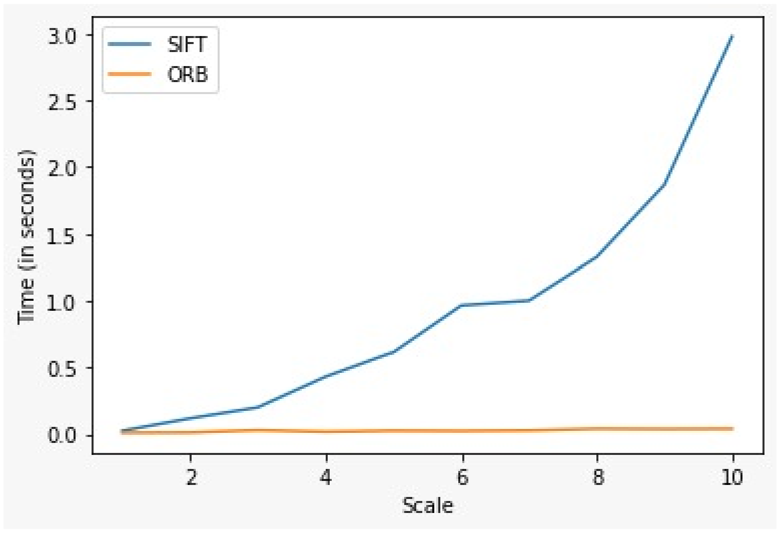


We need to compute orientations for each of our keypoints. We might produce keypoints with identical positions but different orientations. Then we'll sort our finished keypoints and remove duplicates. Finally, we’ll generate descriptor vectors for each of our keypoints, which allow key points to be identified and compared.

1. Create a histogram of gradient orientations for key points using `computeKeypointsWithOrientations()`.
    a. Consider a square neighborhood.
    b. Compute the magnitude and orientation of the 2D gradient at each pixel in this neighborhood
    c. Rotate it by keypoint's angle (This makes SIFT invariant to rotation)
    d. Compute row and column bins which are indices local to the neighborhood that denote where each pixel is.
    e. Calculate each pixel's gradient and magnitude.
    f. The actual value we place in that bin is that pixel’s gradient magnitude with a Gaussian weighting
    g. Store histogram bin index and bin values for each pixel. (Total 8 bins to cover 360 degrees)
2. Smoothing
    a. Smooth the weight gradient magnitude for each neighborhood pixel.
    b. Trilinear interpolation: is a method of multivariate interpolation on a 3-dimensional regular grid. It approximates the value of a function at an intermediate point within the local axial rectangular prism linearly, using function data on the lattice points. We will use this function's inverse to distribute the histogram's value to eight neighbor bins and each of these parts add up to the original. (NOTE: We don't distribute the histogram values uniformly as 1/8th of its neighbors since fractional indices are possible and points closer must have more values. This issue is solved by trilinear interpolation inverse where each point is split into 2 points, then into 4 points, then into 8 i.e. take the center value of the cube and distribute to its 8 neighbors.
3. Flatten the smoothed image into a descriptor of length 128.
4. Apply threshold on the descriptor.
5. Normalize and scale such that the pixels lie between 0 to 255.

DISADVANTAGES OF SIFT

- Still quite slow (SURF provides similar performance while running faster).
- Generally doesn't work well with lighting changes and blur
- It is mathematically complicated and computationally heavy.
- SIFT is based on the Histogram of Gradients. That is, the gradients of each Pixel in the patch need to be computed and these computations cost time. Therefore, it is not effective for low powered devices.
- Does not give good results with highly uniform textured images.



Comparison of time taken by SIFT and ORB for increasingly scaled images

IMPLEMENTED APPLICATIONS:

1. Odometry Calculation

Potential application of the approach is to place recognition, in which a mobile device or vehicle could identify its location by recognizing familiar locations.

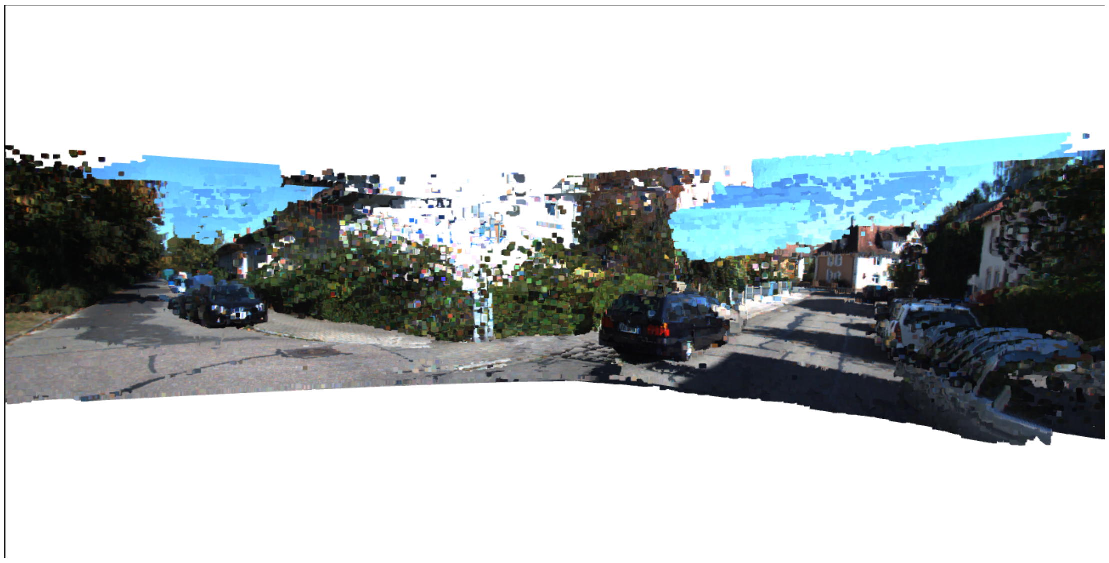
Aim:
To estimate the pose using the RGB image data alone.

Procedure:

I. Initially, we start with computing features across the two images and matching them. Since we already calculated the depth map , we now have correspondences between the depth maps of two images as well from the RGB feature matches.
II. Next, we can now convert this depth map to a point cloud.

III. Since we have correspondences between image points in the depth map, we have 3D correspondences here as well. Perform ICP here to get a good pose estimate.
IV. Next, we Feed these initial pose estimates into the PnP pipeline and optimise further.

Output:



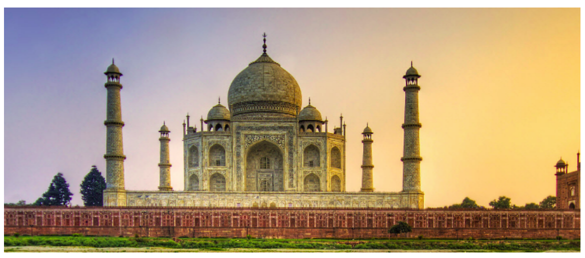
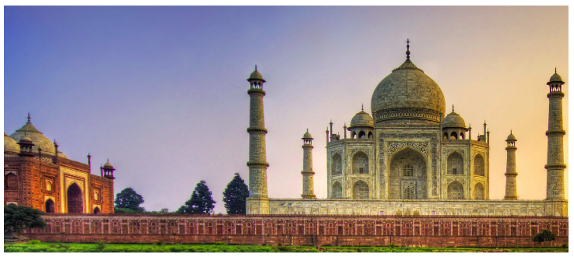
3D reconstruction obtained through SIFT

2. Image Stitching

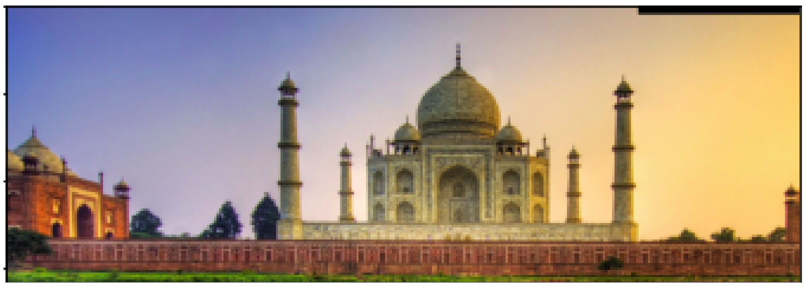
Aim:
Combine multiple images of a scene to create a panoramic view

Procedure:

I. Initially, we start with computing features across the two images and matching them.
II. Next we compute the homography matrix using the matches.
III. We apply transitions to the first image using the matrix.
IV. Lastly, we merge those images.

Left and right images to be stitched



Stitched image

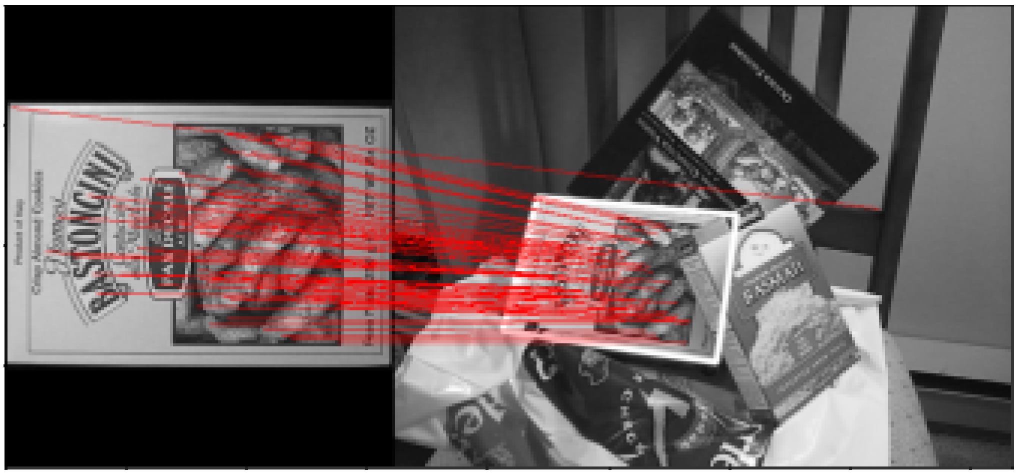
3. Template Matching

Aim:

Given a template image, the goal is to detect it in a scene image and compute the homography.

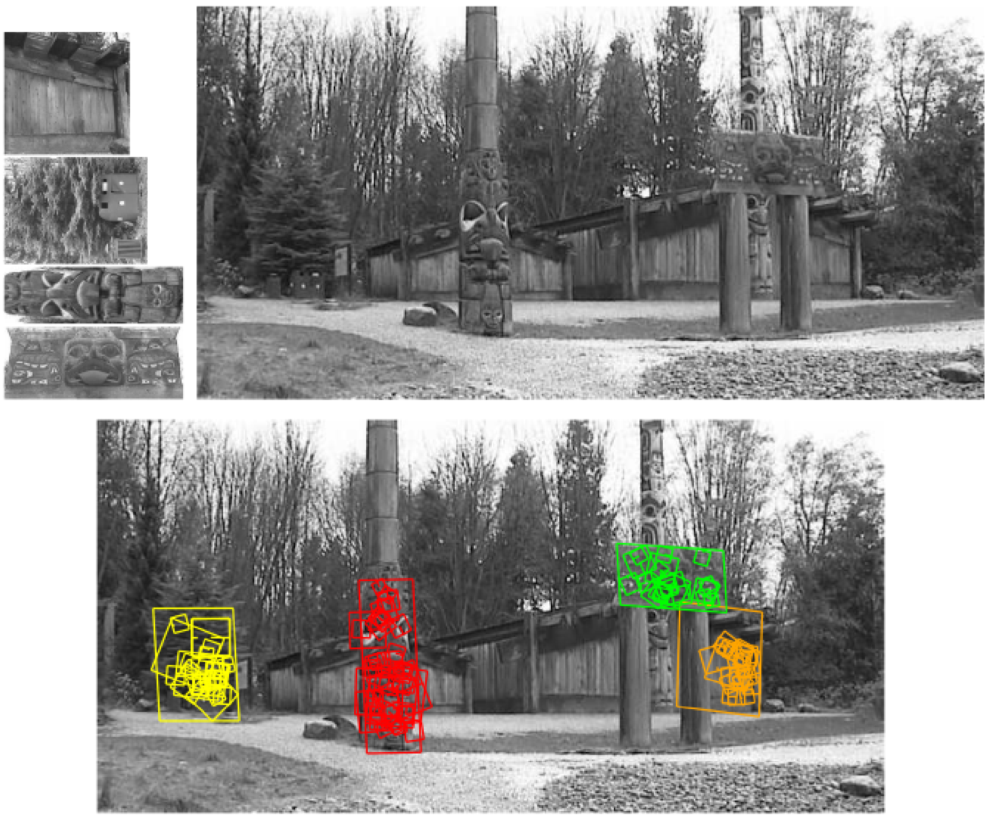
Procedure:

- We compute SIFT keypoints and descriptors on both the template image and the scene image.
- Perform approximate nearest neighbors search on the two sets of descriptors to find similar keypoints.
- Keypoint pairs closer than a threshold are considered good matches.
- Finally, we perform RANSAC on the keypoint matches to compute the best fit homography.



OTHER APPLICATIONS:

OBJECT RECOGNITION:



An example of object recognition for a cluttered and occluded image containing 3D objects is explained below. The training images of a toy train and a frog are shown on the left. The middle image (of size 600x480 pixels) contains instances of these objects hidden behind others and with extensive background clutter so that detection of the objects may not be immediate even for human vision. The image on the right shows the final correct identification superimposed on a reduced contrast version of the image.

The key points used for recognition are shown as squares with an extra line to indicate orientation. The sizes of the squares correspond to the image regions used to construct the descriptor. An outer parallelogram is also drawn around each instance of recognition, with its sides corresponding to the boundaries of the training images projected under the final affine transformation determined during recognition.