Scale Invariant Feature Transform

Aishwarya Gujrathi
Aditya Chondke
Kunal Goyal
Moni Shankar Dey
Punit Galav

SIFT - Scale Invariant Feature Transform

- Proposed by D. Lowe in 1999
- Detects salient, stable feature points in an image
- Provides set of “features” that describes small image region around the point
- Features are invariant to rotation and scale

SIFT Algorithm

1. Determine approximate location and scale of salient feature points (keypoints)
2. Refine their location and scale
3. Determine orientation for each keypoint
4. Determine descriptors for each keypoint

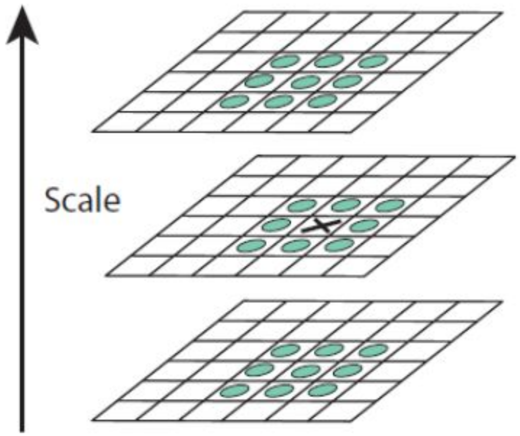
Step 1 : Keypoint location

- Look for intensity changes using the Difference of Gaussians (DoG) at two nearby scales

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

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma). \end{aligned}$$

- Maxima or Minima of DoG detected by comparing pixel to its nearest neighbour at current & adjacent scale
- Keypoints are maxima or minima in the “scale-space-pyramid”
- We get both the location as well as the scale of the keypoint



Step 2: Refine keypoint location

- Keypoint location and scale is discrete
- Can interpolate location for greater accuracy
- Express the DoG function in a small 3D neighborhood around a keypoint 2nd order Taylor-series

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

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

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

- Remove those keypoints with a value of $| D(\hat{\mathbf{x}}) |$ less than 0.03

Eliminate edge keypoints

- Discard the keypoints lying on edges even if they have high response in DoG filter
- 2nd derivative in DoG is an Hessian matrix
- Eigenvalues of **H** give the maximal and minimal principal curvature of the surface
- Keypoint that is not an edge has high maximal and minimal curvature

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

Eliminate the keypoints that have a ratio between the principal curvatures greater than r

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

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

or

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}$$

Step 3 - Orientation Assignment

- Compute gradient magnitudes and orientations in a small window around the keypoint at the appropriate scale

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

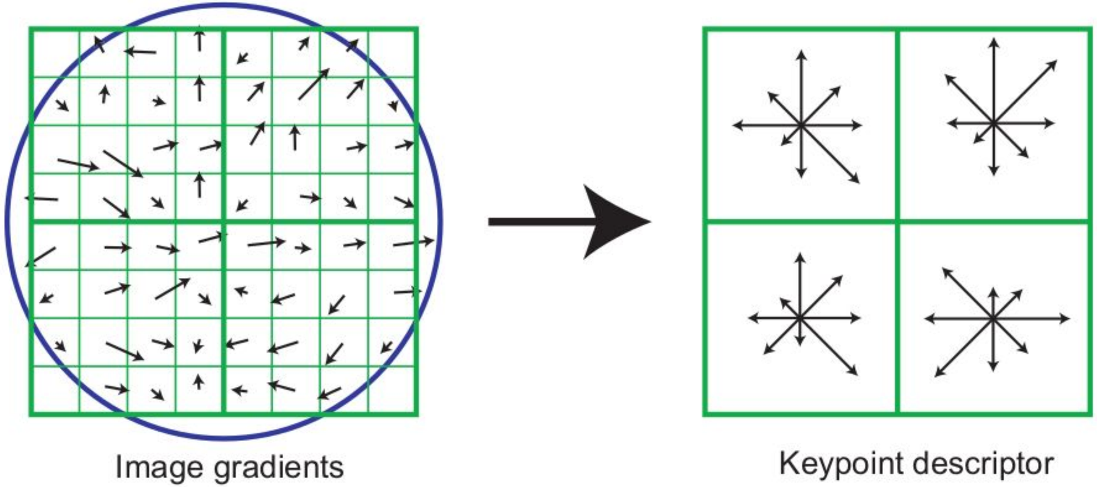
$$\theta(x,y) = tan^{-1}(L(x+1,y) - L(x-1,y) / (L(x,y+1) - L(x,y-1))$$

- Assign dominant orientation as keypoint orientation
- For multiple peaks, create seperate descriptor for each orientation

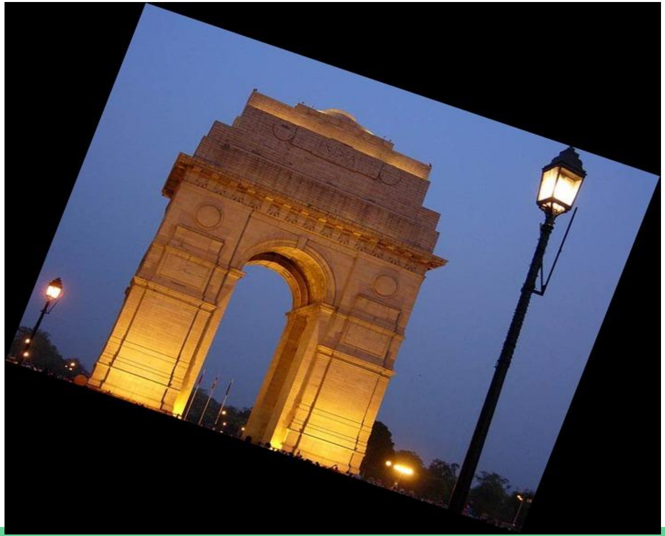
Step 4 - Keypoint Descriptor

- Divide small region around keypoint into nxn cells with cell size 4x4
- Build a gradient orientation histogram in each cell
- Histogram entry weighted by the gradient magnitude and a Gaussian weighting function
- For orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the keypoint orientation.
- We get descriptor size of r x n x n size (r = bins in histogram)

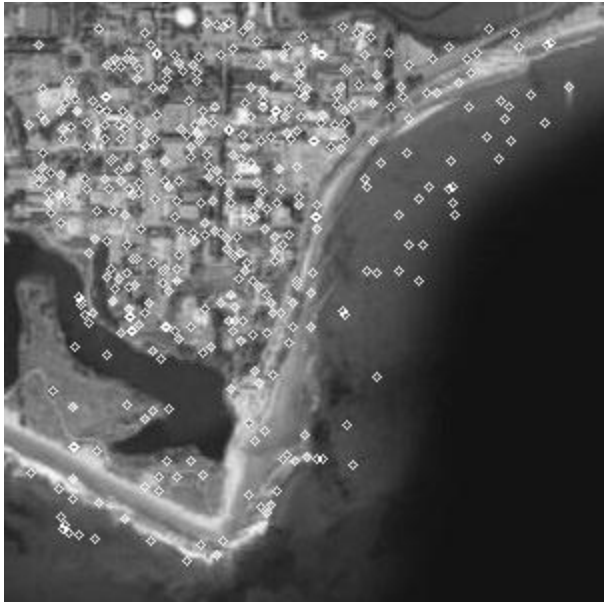
- Histogram entries are weighted by gradient magnitude
- Descriptor vector is normalized to unit magnitude to make it intensity invariant








Results



Keypoint Detection

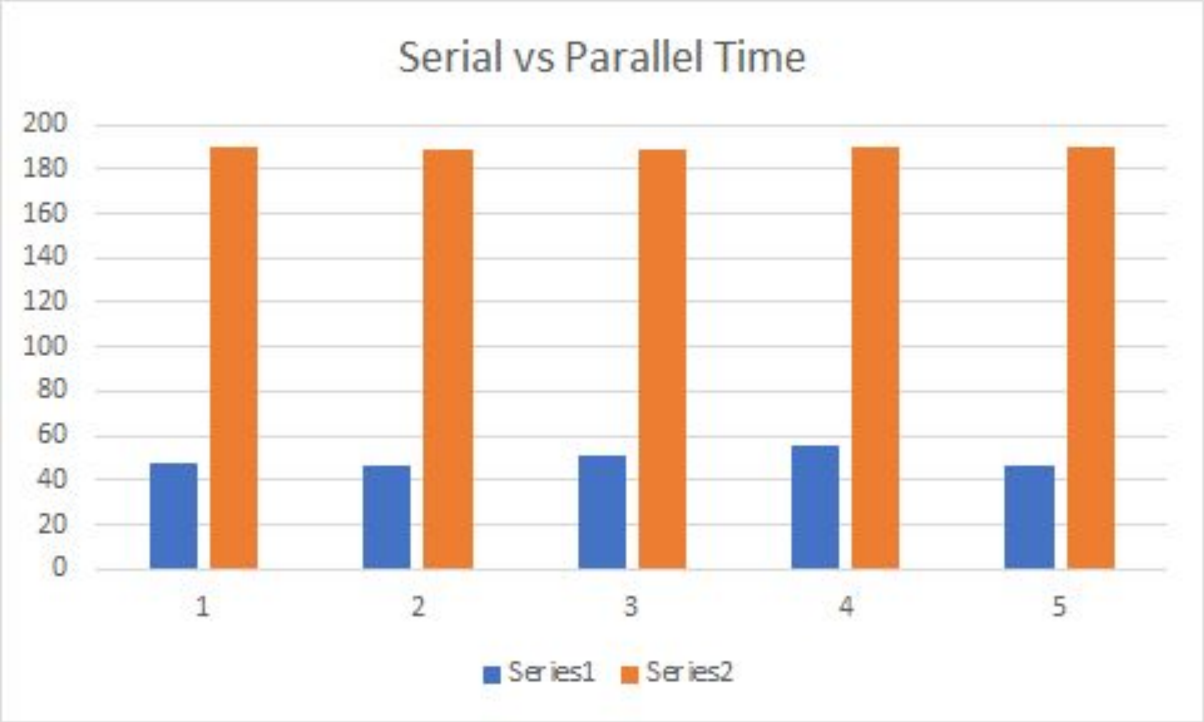
Keypoint matching between the images

Parallelization of Code

- The code have some part which does not have any dependence and was parallelizable
- We first tried parallelizing using pycuda but the results were not that good so shifted to numba
- We achieved about three times faster execution for the code

Githhub link: https://github.com/AdityaChondke/SIFT-Algorithm-

Performance Checking



Thank You !