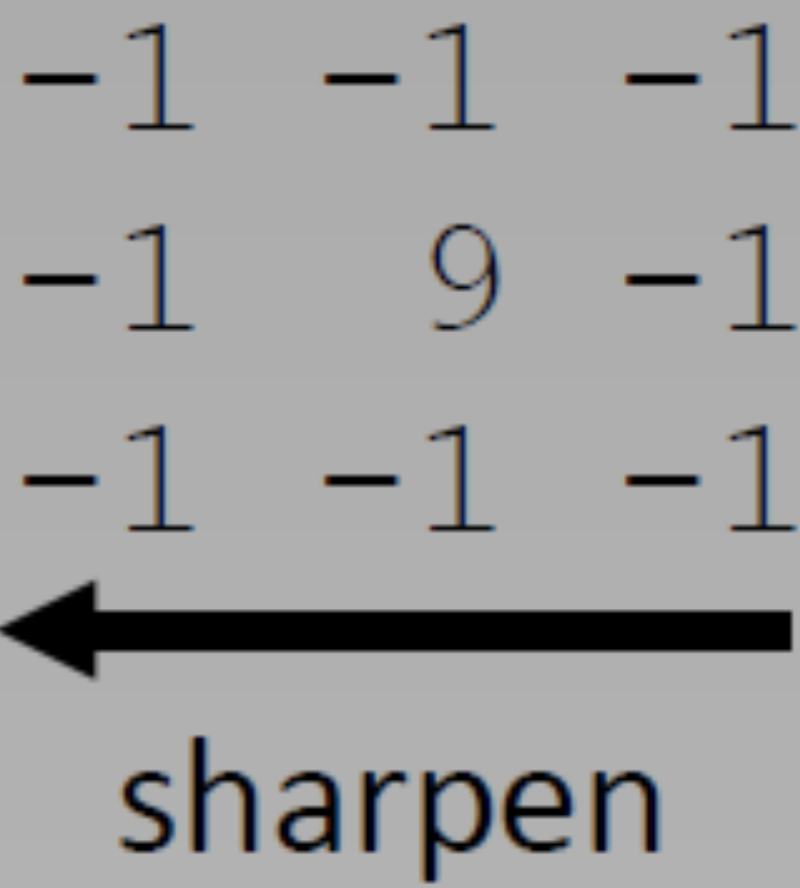
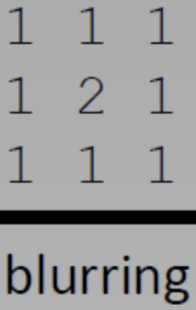
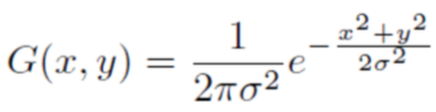
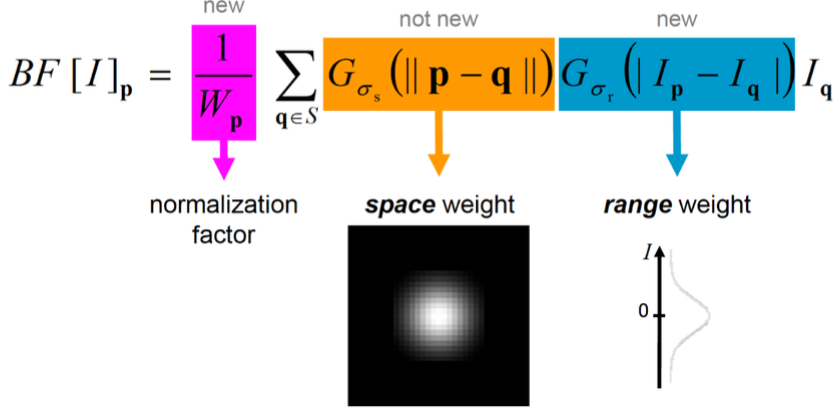
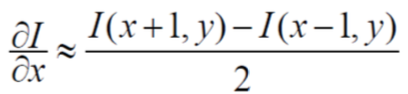
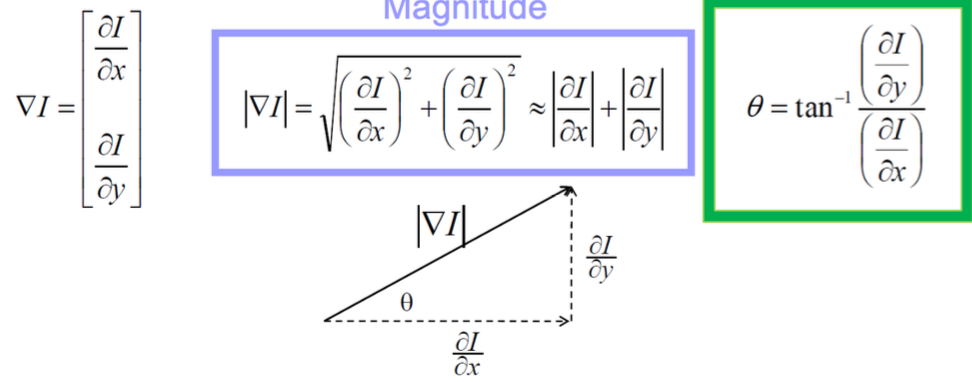
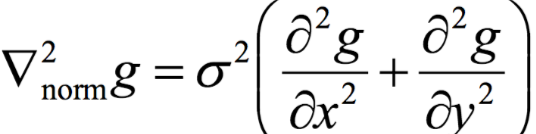
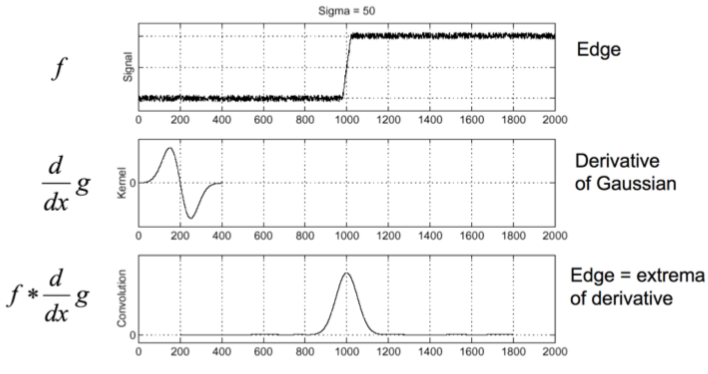
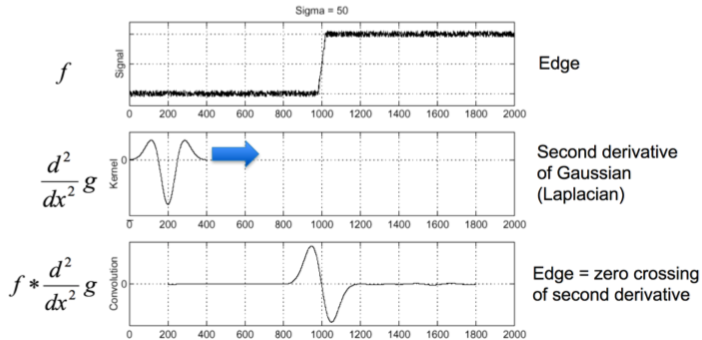
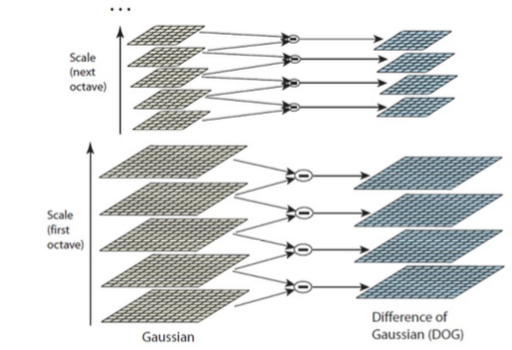
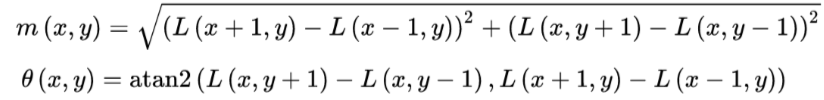
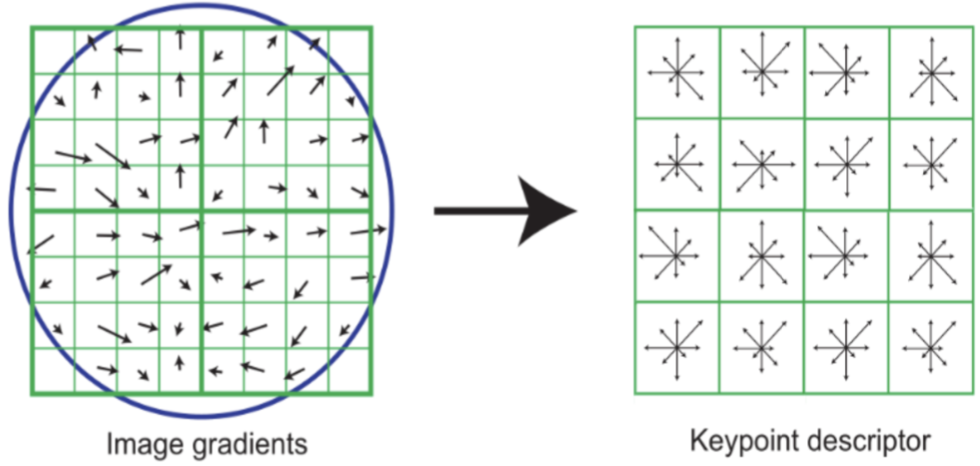
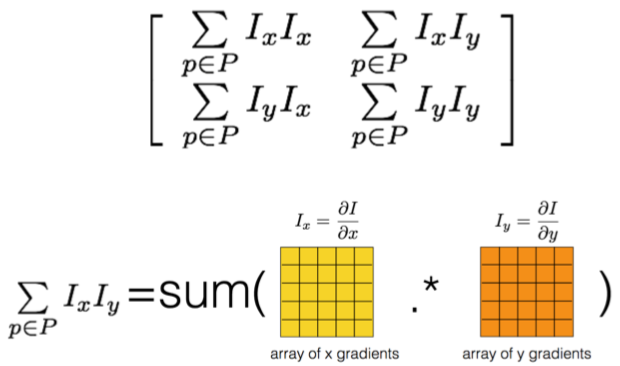
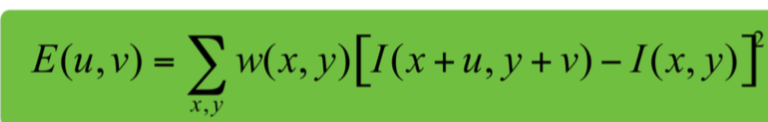
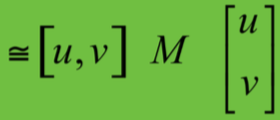
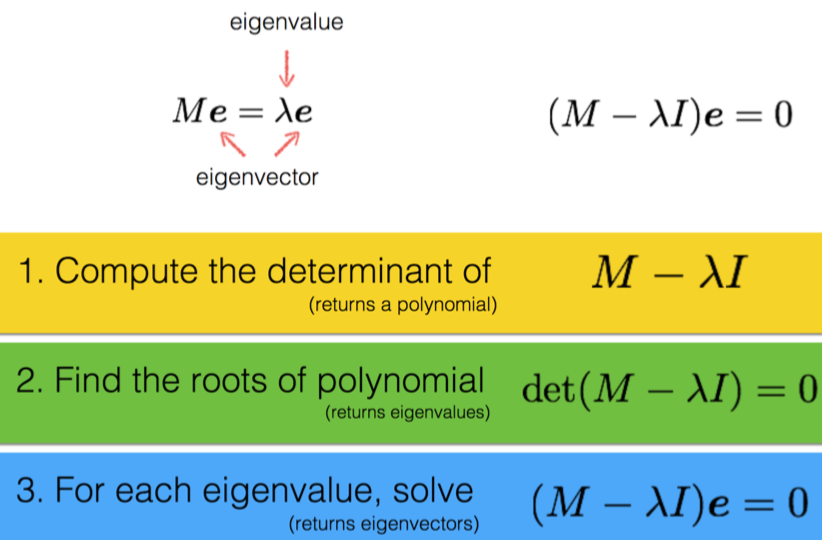
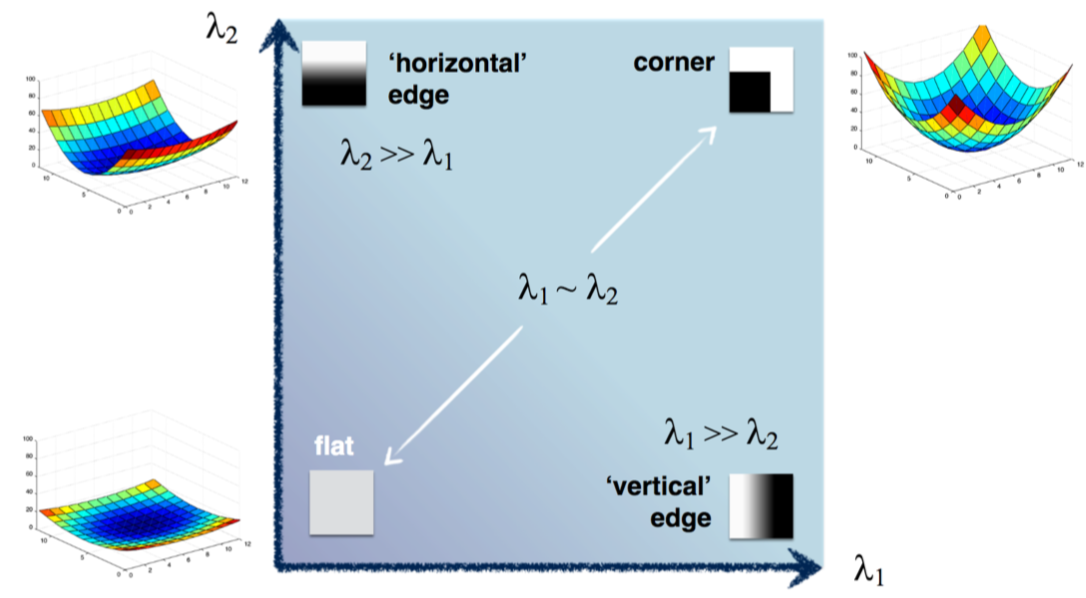
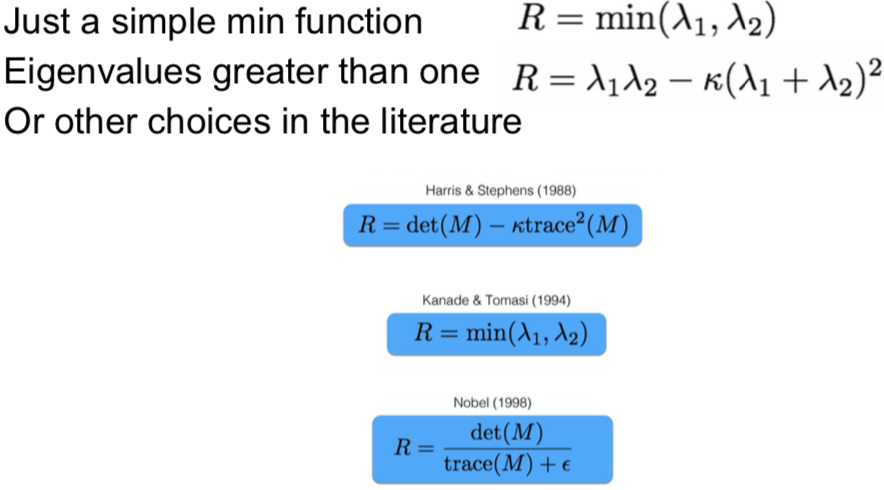
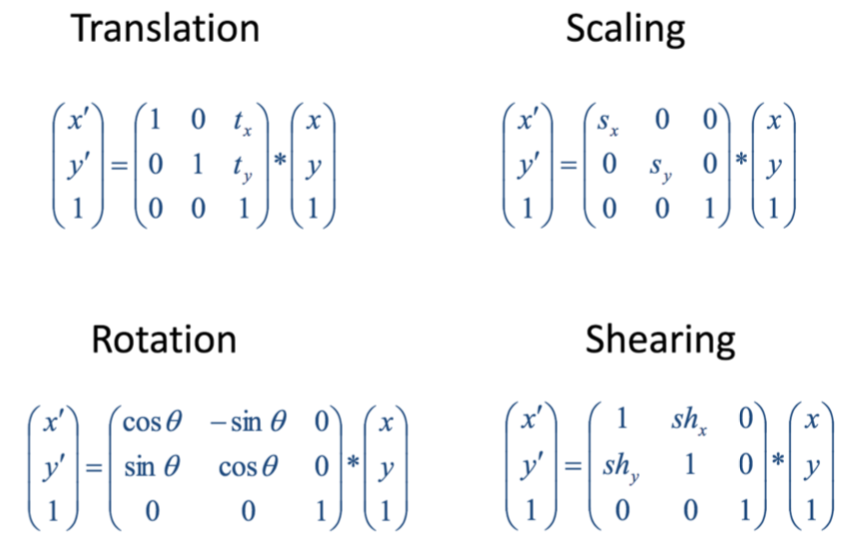
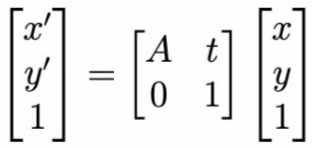
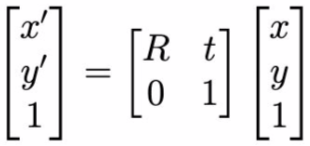
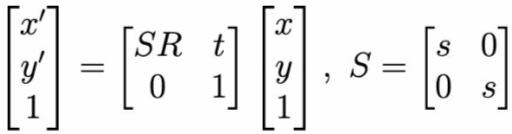
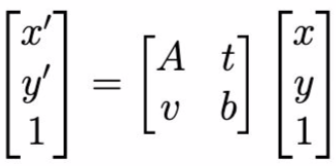
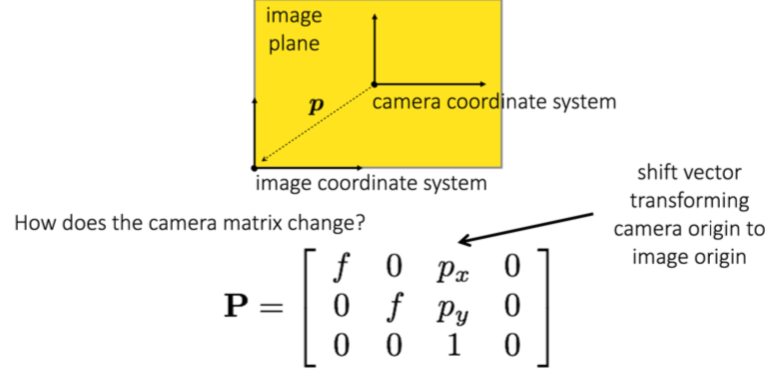
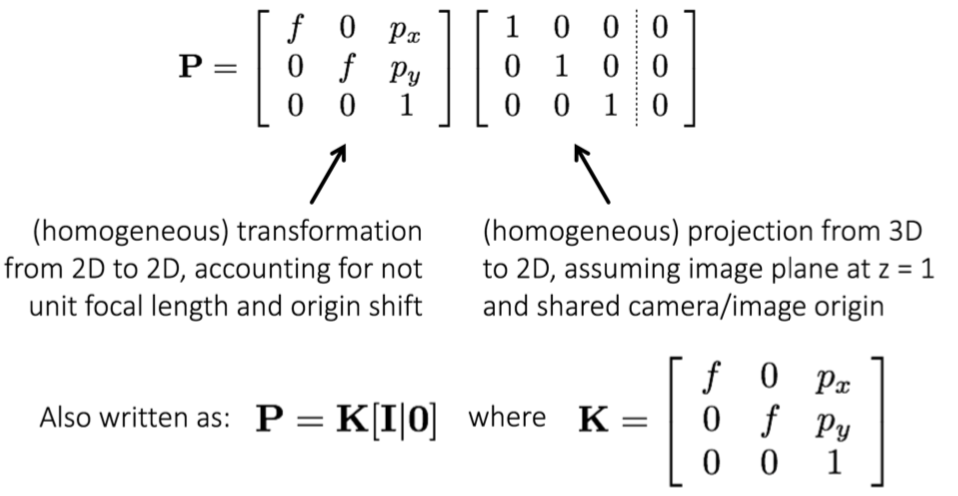
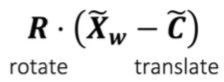
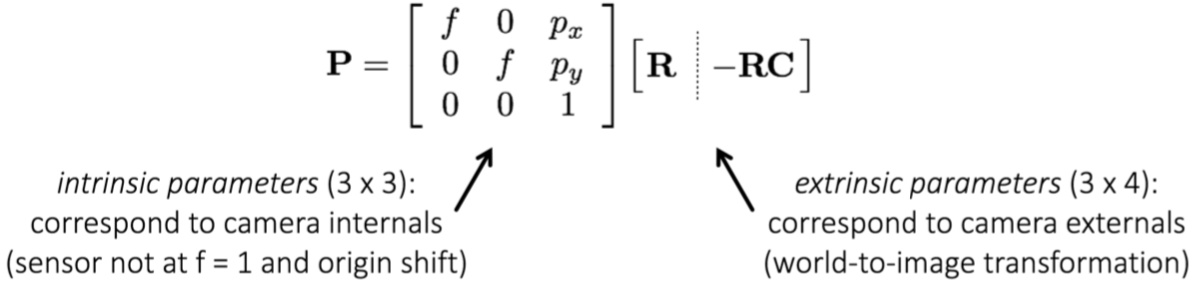
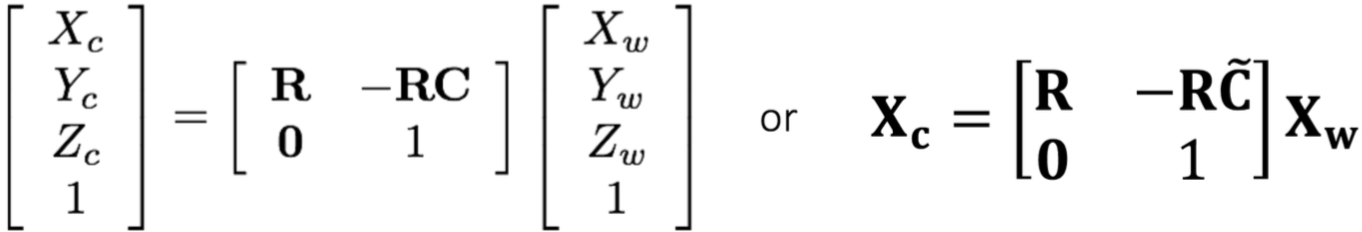
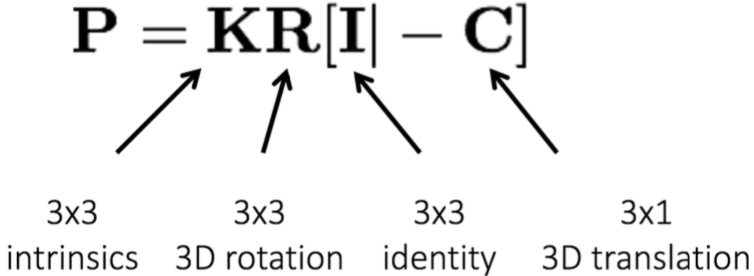
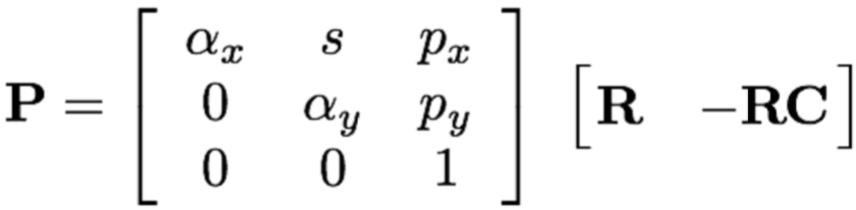
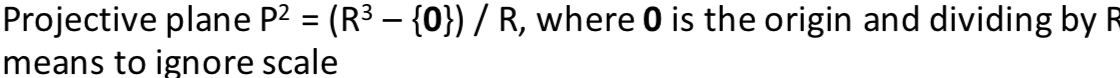
1. Image filtering
   1. **Convolution**
      1. Adds contribution of each element in the image to its local neighbors and weights it by a kernel.
      2. 3 channels for RGB: 5x5x3 filter --> activation maps, number of convolutions = number of maps
   2. **High-pass filter**: keep frequencies higher than a threshold. Sharpen
      1. 
      2. Image gradient: edge detection, feature extraction(SIFT)
   3. **Low-pass filter**: keep frequencies lower than a threshold. Blur --> **Gaussian filter**
      1. 
      2. To decrease the size of an input image with minimal content loss, we should **low pass filter and down-sample** the image
   4. Image = High-pass + low-pass
2. **Gaussian filter**
   1. 
   2. Edges are blurred, too --> Bilateral filtering. Gaussian filter smooths image but also smooths edges, so we add gaussian factors that use difference in intensities
      1. st. dev. for each gaussian factor, sigma\_s: size of the considered neighborhood, sigma\_r: minimum amplitude of an edge
      2. If intensities in small patch are all the same (*not the edge*) --> high gaussian factor, *smoothing*
      3. If intensities change --> low gaussian factor, non-edge is ignored in smoothing --> *edges not ignored*
3. **Image Gradients**
   1. 
   2. 
4. Gaussian derivative and Laplacian
   1. Compute derivative: apply gaussian blur and take derivative
   2. Laplacian: apply gaussian and take second derivative
      1. **Second derivative is used for edge detection and key point detection**
5. Image Features for classification
   1. Detection: where is it? Description: What does it look like? We want image features to be invariant to translations, rotations, scale, illumination (**downsampled image do not have these invariances**)
   2. Small shifts, rotations, scale changes, and intensity changes do not change the overall meaning of an image
   3. Sense of spatial arrangement
   4. From covariant region to invariant features: extract affine regions -> normalize regions (from eclipse to circle) -> eliminate rotational ambiguity -->
      1. <option 1> raw pixels
      2. <option 2> derivatives: 1st gradient, 2nd laplacian
      3. <option 3> sift: local distributions of derivatives
   5. **Blob detection**
      1. to detect a blob of radius r, we use Laplacian filter with scale sigma = r/sqrt(2) --> so that negative part of Laplacian aligned with blob to produce largest magnitude response from convolution
      2. normalize Laplacian: 
      3. blob size: convolve normalized Laplacian --> produce largest magnitude response
   6. **Edge detection**
      1. **Not invariant to scale or rotation**
      2. second derivative cross 0, derivate has max magnitude [1, 0, -1]
      3. 
      4. 
   7. **Example of bad image features**
      1. Down-sampled image have no invariances
      2. Edges not invariant to scale or rotation
      3. Histogram of colors in image patch loses spatial arrangement
      4. Combine histograms over various small patches in a big image patch (spatial histogram) not invariant to rotation --> correct for orientation to get rotation invariance
6. **SIFT (scale-invariant feature transform)**
   1. Invariance to translation, scale, rotation, illuminance and spatial arrangement (viewpoint)
      1. Scale: key points are found at original scale due to DoG pyramid; gradient orientation due to Gaussian-smoothed image at key point scale
      2. Translation: kp original location
      3. rotation: independent of orientation, descriptors from normalized dominant orientation
      4. Illumination/constant intensity shift: normalized gradient orientation historgram
      5. Spatial arrangement: info preserved about smaller patches
   2. **Extract sift features steps**
      1. **Difference of gaussians** at neighboring scales and find **local maxima** along with corresponding scale --> candidate key point: (x,y,sigma), sigma = scale, the patches we are considering are now **scale-invariant**
         1.  --> find local maxima (scale space extrema robust to scale change)
            1. DoG is an approximation to Laplacian, useful for finding edges
      2. **Coordinate refinement**: Taylor series approximation of DoG, take derivative and set to 0, solve to refine key points; Remove noisy key points and key points along the edges because they are invariant to translation along that edge, can’t identify location.
         1. Key points are usually *corners*.
      3. **Orientation assignment**: Find dominant orientation of each key point. Compute gradients in the key point’s neighborhood and check their overall direction.
         1. Gaussian L(x, y, sigma) for magnitude and orientation. Compute histogram of gradient orientations weighted by gradient magnitude --> (x,y,sigma, theta) 
         2. Now we can align them given their orientation and be **invariant to rotations**.
      4. Compute **Sift descriptors**, spatial *histogram* of patch around each key point, invariant to shifts in pixel value and **illumination change**. Size of descriptor = large\_patch / small\_patch \* #bins\_orientation\_hist (weight by gradient magnitude and normalize histograms, normalize orientation using theta)
         1. Also invariant to translation because stored relative to key point position
         2. 
         3. All histograms are normalized
   3. Bag of words
      1. detect interest point features
      2. find closest visual word to region around detected points
      3. record number of occurrences, but not position
7. **Harris Corner Detector** --> Multi-scale orientated patches features (MSOP)
   1. Why detect corners?
      1. 2 edges in different orientations and 2 strong gradients in different directions
   2. Properties
      1. Invariant to rotation
      2. invariant to affine intensity shifts
      3. **Not invariant** to scale
   3. **Steps**
      1. Compute Image gradients in small window around every point, for corner we should see strong gradients in multiple directions
      2. Subtract mean of gradients found from all the gradients (remove DC offset)
      3. Compute covariance matrix of gradients
         1. 
         2. Harris Error Function for intensity change with shift direction
            1.  
      4. Compute eigenvalues and eigenvectors for covariance matrix
         1. 
         2. 
      5. Compute function of eigenvalues (Harris response function) and threshold to detect corners, strong eigenvalues indicate strong corners 
   4. Multi-scale + Harris detector
      1. downsample image with several different sizes and apply harris detector
      2. apply harris detector and smooth with gaussians of different scales --> find location and scale that produces greatest response
8. MOPS
   1. Steps
      1. Given feature (x,y,s,theta), get 40x40 image patch and subsample every fifth pixel to get 8x8 patch (s for scale invariance, theta for rotation invariance), subsample to reduce localization errors --> context around key point
      2. Normalize image patch by subtracting mean and dividing by standard deviation
      3. Compute harr wavelet transform of normalized image patch (get low frequency info)
9. **Classification pipe**
   1. Extract SIFT/SURF/ORB descriptors from all images
   2. Cluster Descriptors using kmeans or agglomerative clustering and obtain vocabulary (cluster centers)
      1. K-Means: random assign centroids, repeatedly assign points to nearest centroids and update centroids
   3. Get BoW representation for all images in training set by finding # of descriptors that are closest to each cluster center, normalize the histogram
   4. Train classifiers to classify images in training set
      1. KNN: majority vote
      2. SVM
   5. Extract BoW for test images and run classifiers
10. Image transformations
    1. Linear transformation Q = T(P) = MP
    2. 
    3. T R S p = T(R(S p))) – scaled, rotated, translated
    4. **Affine transformation**
       1. Linear transformation + translation 
       2. preserve points, straight lines, and parallelism
       3. viewpoint change, eliminate rotation ambiguity by assigning unique orientation to image patch using **gradient orientation**
    5. **Isometric transformation** (or Euclidean isometrics)
       1. Preserve distance
       2. Rotation + translation 
    6. **Similarity transformation**
       1. Preserve shapes (ratio of lengths and angles)
       2. isometric transformation + scaling 
    7. **Projective transformation** of homographies
       1. any transformation that maps lines to lines (doesn’t need to preserve parallelism) 
       2. Generalizes affine transformations
    8. Issues with applying transformations
       1. Holes in image -> interpolate to find pixel values of output image at integer pixel locations, better way is to apply **inverse transformation** on output image, and then **interpolate** in input image to get pixel values for output image.
       2. **Bilinear interpolation**
11. Pinhole **camera model: mapping 3D world to 2D image**
    1. Upside down image depending on how far image plane is from camera, the larger than pinhole size, the blurrier the image.
    2. 2x pinhole diameter --> 4x light
    3. 2x focal length --> 1/4x light
    4. Lens camera: use only central rays + in focus
    5. Rearranged: put image plane on same side of pinhole as scene
    6. **Camera to Image** frame
       1. Camera matrix 3x4 transforms points in scene (4x1) to points in image (3x1) [**x = PX**]focal length + O translation
       2. Degrees of freedom for essential matrix: 3 for rotation, 3 for translation, 1 removed up to a scale
    7. **World to Camera** Transform
       1. Translate world to camera, rotate world to camera
       2.  Heterogeneous coordinates +tildes
       3. Rewrite world to camera transform in homogeneous coordinates  
       4. Intrinsic parameters of a camera model is affected by focal length, offset of optical center, and image resolution (not by exposure)
       5. Since camera pixels may not be square and as camera sensor may be skewed, we allow intrinsic matrix to have different factors for x and y coordinates and add a skew term
    8. **Projective geometry**
       1. two parallel lines intersect in exactly **one** point, generalize affine transformation
       2. ****
       3. Not all light rays passing through the image center actually hit the image plane, the light rays that do not move in Z direction at all do not (x,y,0) --> points at infinity where parallel lines intersect. Since ignore scale , set z = 1, (x,y,1) --> homogeneous coordinates
       4. **Homogeneous coordinates** used to represent image coordinates represent elements in the projective plane, projection from 3D to 2D
          1. Vector: [v1,v2,v3,0], point: [p1,p2,p3,1]
       5. Not a vector space (no origin). Transformations on homogeneous coordinates/in projective plane are generally not linear transformation.
12. **Structure from Motion(SfM)**
    1. given images of the same scene from different angles --> reconstruct 3D model
    2. Simultaneous localization and mapping (SLAM)
    3. How the scene 3D is projected onto the image plane? geometry: points --> pixels; photometry: light --> intensities
    4. How moving around affects the image? Deformed surfaces by camera motion, preserved properties of the image
    5. Deduce from the transformation in my images: pose estimation + depth estimation + 3D reconstruction
    6. Determining a pixel color
    7. The problem of infinity
    8. Ponzo effect
13. Image classification
    1. KNN
    2. Linear classifier
    3. RBF classifier = Kernel projection + linear classifier

Exercise

1. You are using k-means clustering in color space to segment an image. How-ever, you notice that although pixels of similar color are indeed clustered together into the same clusters, there are many discontiguous regions because these pixels are often not directly next to each other. Describe a method to overcome this problem in the k- means framework.

*Concatenate the coordinates (x, y) with the color features as input to the k-means algorithm.*

2. How does SIFT achieve rotation invariance?

*Assigning orientations to the key points and then rotating the patch to a canonical orientation --> done by constructing* ***histograms of gradients*** *in a neighborhood around the feature point and assigning the largest bin as the* ***corresponding direction of the key point****. Descriptors are obtained from normalized dominant orientation in the histogram of orientation gradients. All detected features are rotated so that the corresponding orientations are vertically aligned.*

3. Determine the pixel value of the point

*Exposure time, illumination, aperture, focus*

4. Produce 3D point from homogeneous coordinates:

*step1: change all to (x,y,z,1)*

*step2: add two coordinates*

*step3: remove 1*

5. Pixel values greater than a threshold are thought to reflect moving objects. What are other factors besides object movement that can cause nonzero pixel values?

*Change in lighting, noise in the images varying over time. Motion of the camera.*

6. Compute sum of pixel values in a patch; filter Bj applied.

