Feature Matching: A Basic Summary

CompRobo Fall 2018, Lydia Zuehsow

Feature Detection

Two important concepts in feature detection are scale-invariance and rotation-invariance.

In the case of scale-invariance, images appear differently when enlarged or reduced. Enlarging an image may introduce new, previously unseen features, while reducing the size or quality of an image may cause previously seen features to disappear. This effect can be resolved by artificially reducing the sharpness of images. Scale space was less important for StarFinder, as the ceiling remained at a constant height and skew from the robot.

In the case of rotation-invariance, objects may reveal more or less features when rotated, or keypoints may be in a different orientation. Rotation invariance can be resolved by choosing feature detectors that are rotationally invariant. For example, in StarFinder, MJ chose to use keypoints' nearest neighbors as a feature to compare, at least in part because it was rotationally invariant.

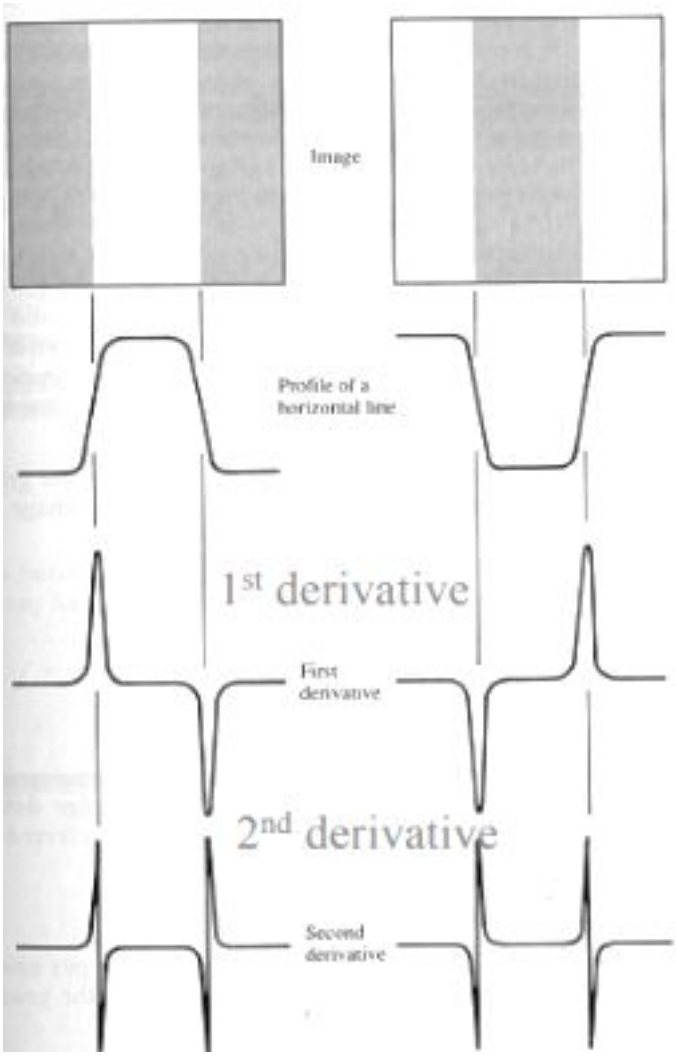

***Figure 1 (right)**: A visualization of an edge in the first and second derivative of an image.*

Scale-Invariant Feature Detection

One commonly used equation for blob or edge detection is the Laplacian of Gaussian, or LoG. (Fig. 4) This is a derivative filter (Fig. 1), so it detects sudden changes in the image. As a result, it is very sensitive to noisy data, because most noise is sharply differentiable from the surrounding image. It is important to reduce the detail and sharpness in the image prior to running LoG to avoid spurious results from noise. However, it is equally important to retain all prior existing key features as-is for better feature recognition.

Because convolving a Gaussian blur kernel is able to reduce sharpness without introducing any previously non-existent features or changing the size of the actual image resolution, it is the primarily used method for preprocessing images prior to feature detection.

The Gaussian **scale-space**, then, is the series of images obtained by repeatedly performing this convolution, where the original image is at the bottom of the stack (t=0) and images higher up the stack are of consecutively lower resolution. (t = 1,4,16) All images remain the same size and shape; only resolution changes. (Fig. 2)

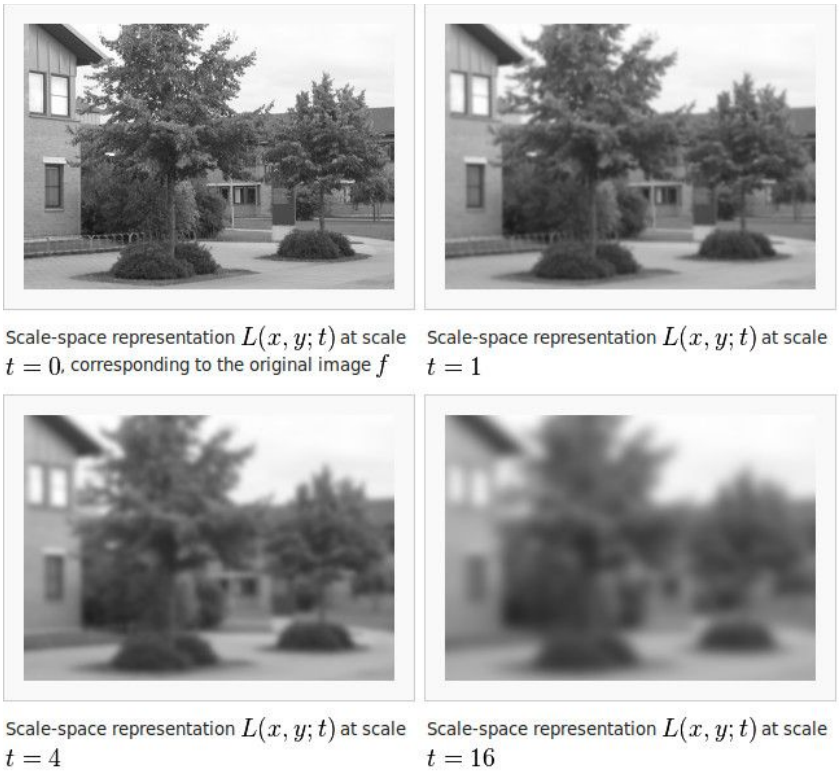

Figure 2: *Gaussian scale space showing images at scale 0, 1, 4, and 16.*

The Gaussian scale-space can be calculated either through iterative convolution with the Gaussian kernel, or by finding the product of the derivative of the Gaussian kernel and the original image. (Eqn. 1)

$$f_\sigma(x, y) = g_\sigma(x, y) * f(x, y). \qquad \frac{\partial f_{\sigma_1}}{\partial x} = \frac{\partial g_{\sigma_1}}{\partial x} * f.$$

Equation 1: Left: *The value of f at scale σ and location (x,y) is equal to the Gaussian kernel at the same scale and location, multiplied by the original image.* ***Right:*** *The derivative of f at scale σ with respect to the signal (brightness, 0-255) is equal to the derivative of the Gaussian kernel at scale σ with respect to the signal, convolved with the original image.*

The Laplacian of Gaussian works by adding the second derivatives of the intensity values, with respect to the x and the y direction. (Eqn. 2) The second derivative with respect to a direction essentially determines the rate of change of changing intensity values: extreme changes like sharply delineated edges create higher signal spikes than gradual gradients. The LoG is typically applied to an image after Gaussian smoothing via one of two kernels, which are pictured in Figure 3. The sign of the LoG does not matter, only the magnitude.

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

Equation 2: Left: *The equation for the Laplacian L(x,y) of an image with pixel intensity values I(x,y).*
Figure 3: Right: *Two common kernels used to apply a Laplacian filter to an image.*

Figure 4: *Comparison of original image (left) and LoG image (right).*

The feature detection algorithm Scale-Invariant-Feature Transform (SIFT) uses an approximation of LoG called DoG, or Difference in Gaussian, which is less costly. Once the DoG is found, low contrast keypoints, edges, and corners are removed with various methods to refine points of interest to match on.

Rotation-Invariance

In feature matching, a feature is defined as a keypoint and a square matrix of the image intensity gradients and/or magnitudes surrounding that keypoint. Rotation invariance is achieved through finding the dominant orientation of the keypoint in question, then subtracting gradient orientation from keypoint orientation. Dominant orientation of a keypoint is defined as the principal component vector of the gradient magnitudes of the neighboring pixels, where the scale assigned to the keypoint determines the number of neighboring pixels included in the principal component analysis. To find the principal component vector, SIFT breaks the 360 degrees around the keypoint into 36 bins, calculates each bin's gradient magnitude and orientation (Fig. 5), and thresholds the bins to determine the orientations with peak values.

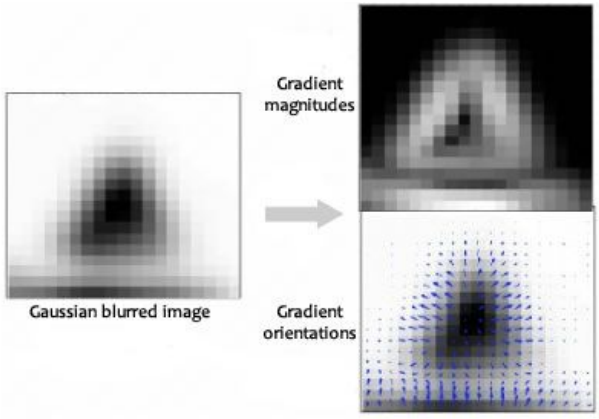

Figure 5: *Gradient magnitudes and orientations for an image.*

Feature Matching

The OpenCV implementation of SIFT contains two feature matchers: A Brute Force matcher, and a FLANN-based matcher. I'll primarily be focusing on the FLANN-based matcher. FLANN, or Fast Library for Approximate Nearest Neighbors, is a library that contains several methods for nearest neighbor classification.

In Nearest Neighbor problems, a series of datapoints must be clustered such that closely neighboring points are placed in the same cluster and distant datapoints are placed in different clusters. This becomes a difficult problem when the number of clusters is not previously known. In StarFinder, we want to group our star keypoints into two groups- matching keypoints, and non-matching keypoints. Because we know the number of clusters, we can use K-Nearest-Neighbors, or KNN, as our feature matcher.

KNN works by comparing points using a distance metric, then grouping points that have similar distances together. Two popular distance metrics are Euclidean distance and cosine similarity. Euclidean distance is simply the straightline distance between two points. (Eqn. 3) Cosine similarity is a little more interesting, in that it calculates the difference in direction between two vectors rather than the direct magnitude. (Eqn. 4) We use both distance measures in our current implementation.

$$E(x, y) = \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2}$$

Equation 3: *The Euclidean distance between two points x & y.*

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

Equation 4: *The cosine distance between two points A & B.*

In SIFT, the k-NN matcher checks the K nearest neighbors' cosine similarity, then adds the point is added to whichever group has the greater weights (matched or unmatched points). In StarFinder, the returned group of matches is then screened for only the good matches. If enough good matches exist (>4) then the homography, or conversion matrix, between the two images' matches is calculated and used to transform the two input matrices so that the matched sections are aligned.

A few transforms are available for this purpose: namely, full affine transformation, partial affine transformation, and projective transformation. Projective transformation, or homography is the generalized form of transform, and the most powerful as it is able to encode 3D transforms and has 8 degrees of freedom. Affine transforms are a subset of homography that can encode only 2D transforms. The primary differentiation between full affine and partial affine is the degrees of freedom; full affine has 6 degrees of freedom, and can handle rotation, scaling, shearing, translation and reflection, while partial affine has only 4 degrees of freedom and can only handle rotation, uniform scaling and translation.

In StarFinder's case, MJ decided to use a partial affine transformation. Because our features are all essentially on a 2D plane directly above our Neato, the additional degrees of freedom for a projective or full affine transformation are superfluous.

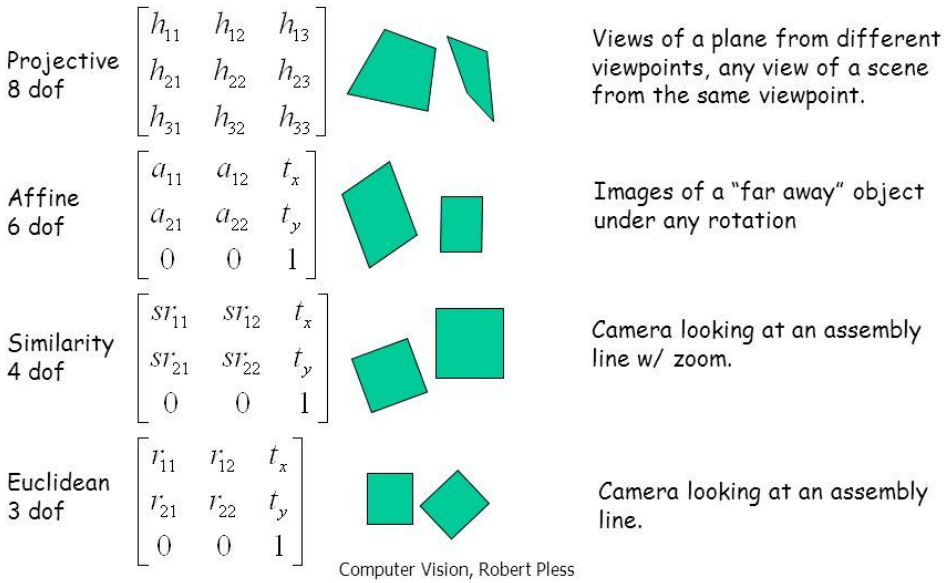

Figure 6: Chart of different transform types and use cases.

Rotational Coordinate Transforms

The current set of coordinates can be expressed as a 2x1 matrix $(x, y, z, 1)^T$. In a projective transformation, this vector is multiplied with a matrix representing a projective transform. The output is a vector, and represents the effects of some combination of scaling, shear, rotation, and translation on the original data.

$$S_v p = \begin{bmatrix} v_x & 0 & 0 & 0 \\ 0 & v_y & 0 & 0 \\ 0 & 0 & v_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} v_x p_x \\ v_y p_y \\ v_z p_z \\ 1 \end{bmatrix}.$$

If one of the v_k is not equal to the other two, the image will scale proportionally to that component, along that axis. If all 3 are larger than 0, the image will scale uniformly on all axes.

StarFinder Implementation

For StarFinder, MJ doesn't have to perform feature detection, as brightness can be used effectively to isolate all keypoints. Thus, she can skip this step and proceed directly to feature matching. Her current implementation is a two step combination of approximate Nearest Neighbors (FLANN) and Random Sample Consensus (RANSAC), where FLANN is used to quickly filter out the most likely matches based on the clustering of points, and RANSAC is then run on those matches to determine the match with the least error.

Sources:

Edge Detection:
http://www.me.umn.edu/courses/me5286/vision/VisionNotes/2017/ME5286-Lecture7-2017-EdgeDetection2.pdf

Features v. Keypoints:
https://stackoverflow.com/questions/42116583/what-is-difference-between-features-and-keypoints-in-computer-vision?rq=1

KNN:
https://towardsdatascience.com/introduction-to-k-nearest-neighbors-3b534bb11d26

Laplacian:
https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm
https://academic.mu.edu/phys/matthysd/web226/Lab02.htm

Scale space:
http://www.cs.uu.nl/docs/vakken/ibv/reader/chapter9.pdf

SIFT:
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
http://aishack.in/tutorials/sift-scale-invariant-feature-transform-keypoint-orientation/

Transforms:
http://nghiaho.com/?p=2208
Slide 47, https://slideplayer.com/slide/3249143/