Feature matching

Once we have extracted features and their descriptors from two or more images, the next step is to establish some preliminary feature matches between these images.

Given a Euclidean distance metric, the simplest matching strategy is to set a threshold and to return all matches from other images within this threshold.

The problem with using a fixed threshold is that it is difficult to set:
- useful range of thresholds can vary a lot.

Feature matching

A better strategy in such cases is to simply match the nearest neighbor in feature space.

Another strategy is to compare the nearest neighbor distance to that of the second nearest neighbor.

We can define this nearest neighbor distance ratio as $\mathrm{NNDR} = \frac{d_1}{d_2}$

where d1 and d2 are the nearest and second nearest neighbor distances.

If NNDR is smaller than threshold, the first nearest neighbor is matched.

LSH

We should first find the nearest neighbor.

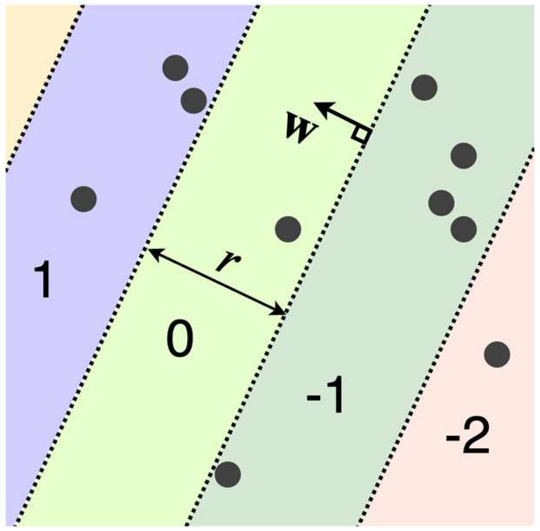
One of the algorithms is locality-sensitive hashing, which maps similar descriptors into same buckets based on some function applied to each descriptor vector.

if $d(p,q) \leq R$ then $h(p) = h(q)$ with probability at least P1
if $d(p,q) \geq cR$ then $h(p) = h(q)$ with probability at least P2

if P1 > P2, family of $h(x)$ is called locality sensitive

LSH

The LSH functions for the Euclidean distances are based on random projections.



$$h(\mathbf{x}) = \left\lfloor \frac{\mathbf{w}^\top \mathbf{x} + b}{r} \right\rfloor$$

Quantizing projection values into a set of hash buckets:

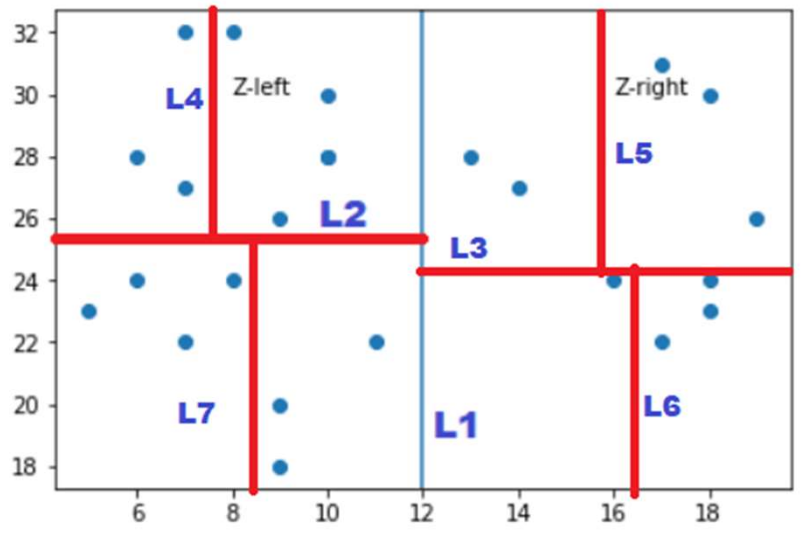
- nearby items will fall into the same bin
- find nearest neighbor by selecting minimum distance vector in chosen bucket

k-d Tree

Another widely used algorithm is k-d(dimensional) trees.

k-d tree is a space-partitioning data structure for organizing points in a k-D space.

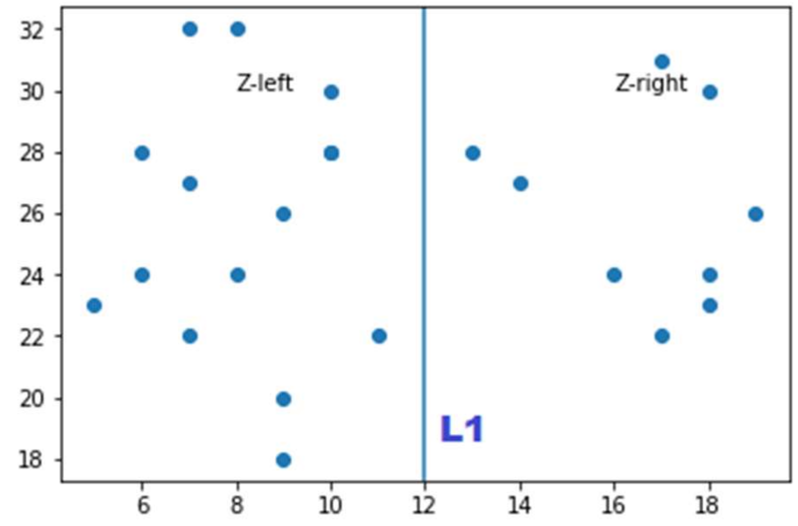
The k-d tree recursively splits plane along axis-aligned (horizontal or vertical) planes.

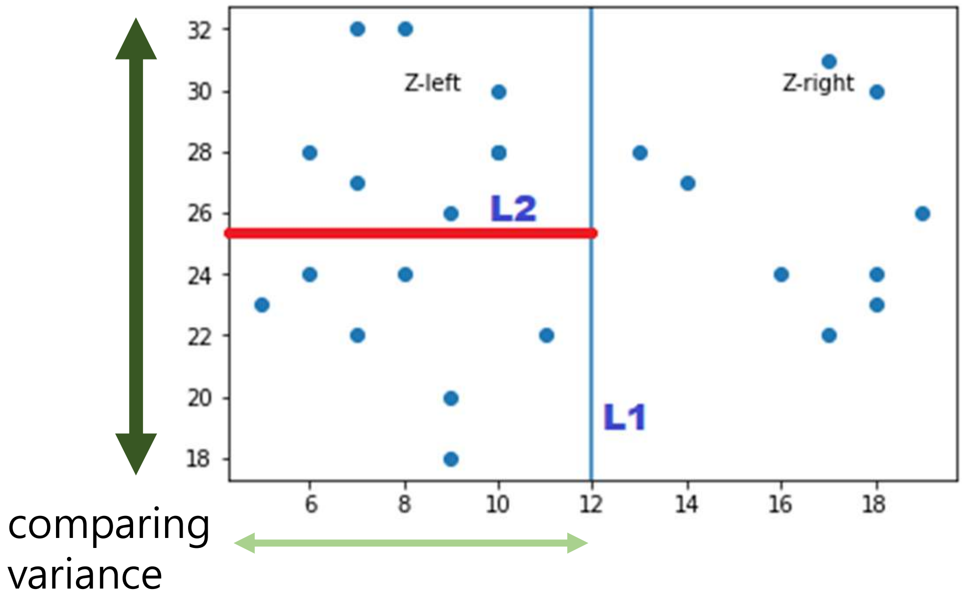


k-d Tree

At each iteration, the variance of each feature is computed and the data is split into two parts on the feature with maximum variance.

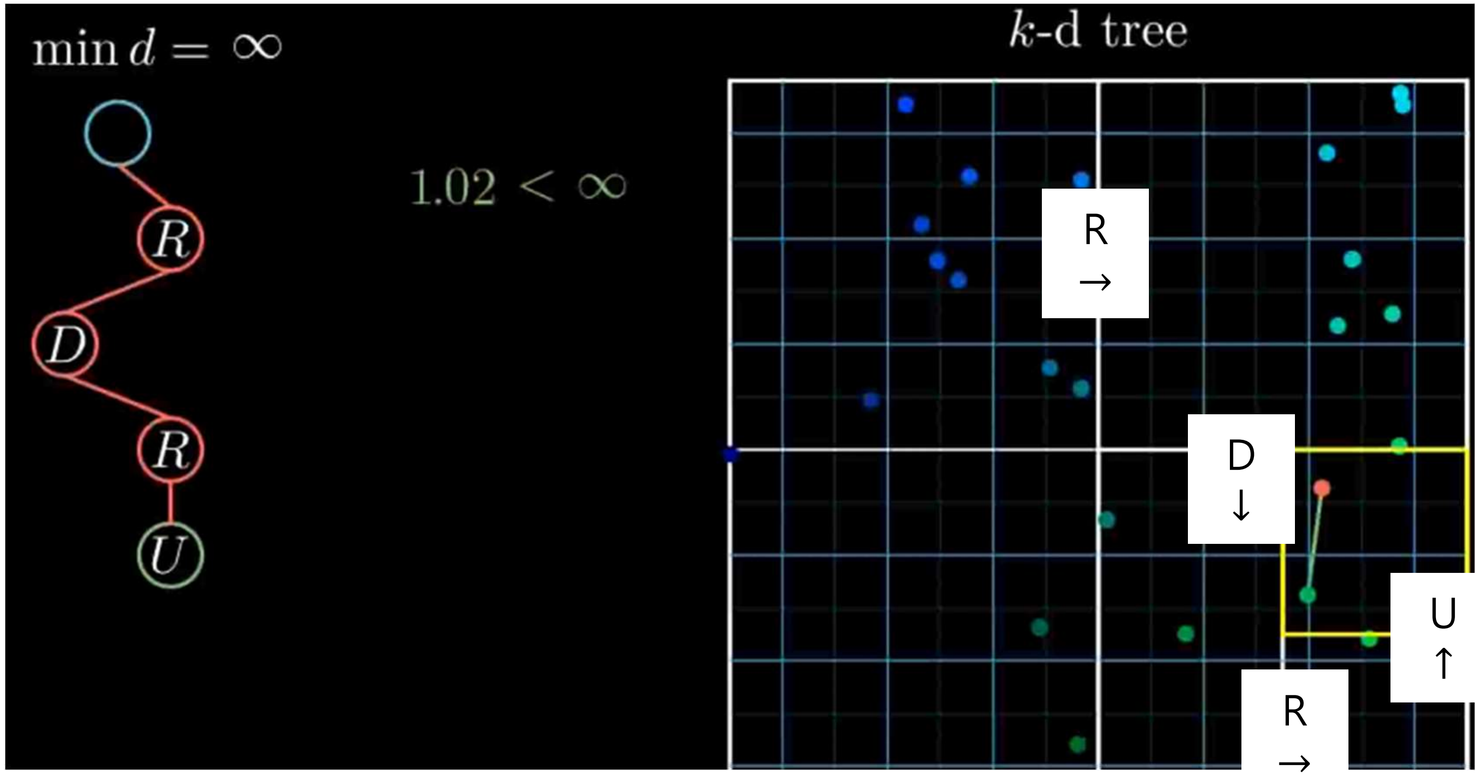
The splitting threshold can be selected to be the mean or the median.





k-d Tree

Starting with the root node, the algorithm moves down the tree recursively, find the point in the tree that has the nearest distance to a given input point.



Estimating transformation

Feature-based matching is invariant to the strength of the geometric deformation.

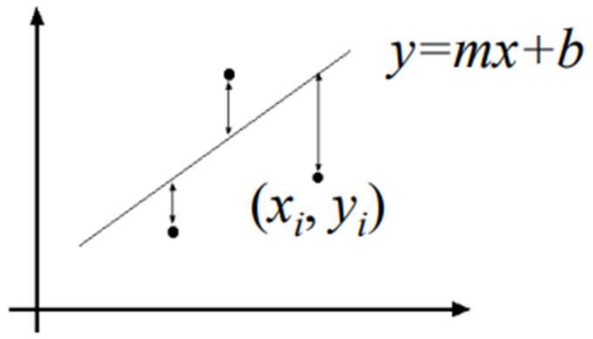
We can estimate transformation to verify which matches are inliers and which ones are outliers.

Least squares

Suppose the data is 2-D.

Data: $(x_1,\ y_1),\ \ldots\ ,(x_n,\ y_n)$

Transformation equation: $\mathbf{p} = y_i = x_i + b$



Find $m,\ b$ to minimize $E = \sum_{i=i}^{n}(\,y_i - x_i - b)^2$

$$E = \sum_{i=1}^{n}\left(\begin{bmatrix} x_i & 1 \end{bmatrix}\begin{bmatrix} m \\ b \end{bmatrix} - y_i\right)^2 = \left\|\begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix}\begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}\right\|^2 = \|\mathbf{Ap} - \mathbf{y}\|^2$$

$$= \mathbf{y}^T\mathbf{y} - 2(\mathbf{Ap})^T\mathbf{y} + (\mathbf{Ap})^T(\mathbf{Ap})$$

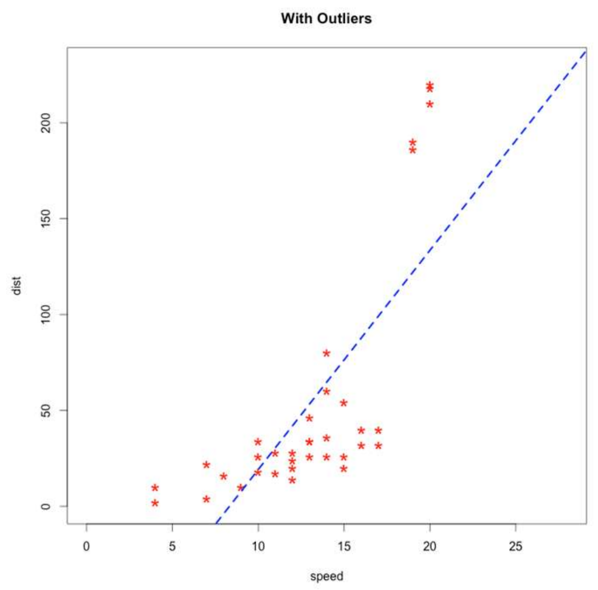
$$\frac{dE}{dp} = 2\mathbf{A}^T\mathbf{Ap} - 2\mathbf{A}^T\mathbf{y} = 0$$

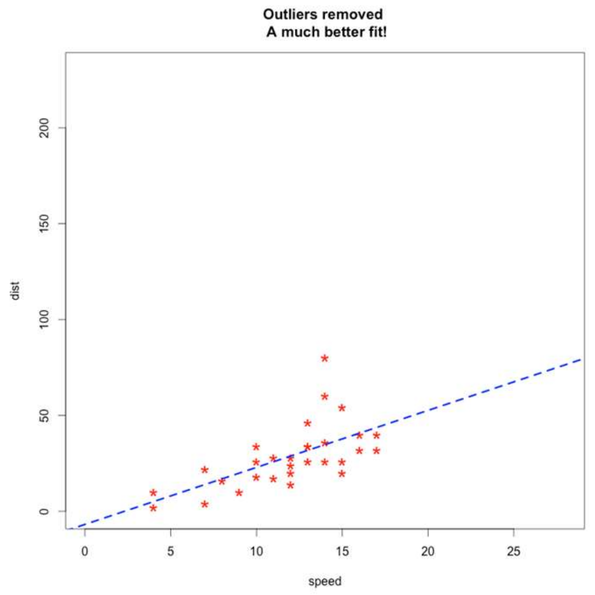
$$\Rightarrow \mathbf{p} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{y}$$

Least squares

Least squares method is easy to optimize.

But it is sensitive to outliers.



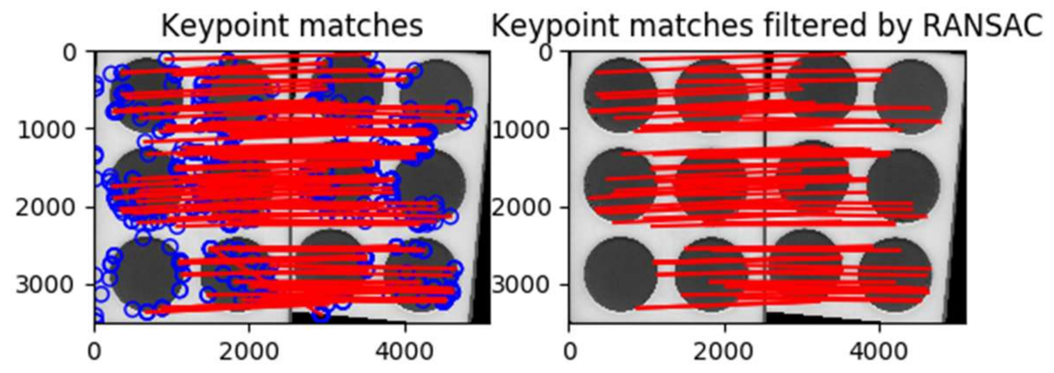


RANSAC

1. Sample a set of matching points and solve for transformation parameters.

2. Score parameters with number of inliers and repeat.

3. Filtering out some outliers.



Feature Matching

1. Find a set of distinctive descriptors. (x, y, σ, θ_i, $\mathbf{x}_i$)
   - SIFT, SURF keypoints -> descriptors

2. Matching Local Descriptors
   - compute the nearest neighbors(k-d tree, Hashing)

3. Estimating Transformation Matrix
   - Least squares, RANSAC