CS 131 Computer Vision: Foundations and Applications

Junbin Huang

October 8, 2017

Abstract

The lecture notes are based on Prof. Feifei Li' lecture on computer vision. This lecture note is based on the second part Regions of Images, and Segmentation.

- Basic Concepts of Segmentation
- K-means Clustering
- Feature tracking
- *Linear Classifiers & Classification
- *Visual Tracking

1 Clustering and Segmentation

1.1 Introduction to segmentation and clustering

The **goal** of segmentation is to separate the input image into coherent "objects" or group together similar-looking pixels (as a kind of "superpixels") for efficiency of further processing.

One way to think about "segmentation" is clustering. The **goal** of clustering is to group together similar data points and represent them with a single token.

So, why do we clustering? The reasons are following:

- Summarizing data:
  - Browsing: Look at large amounts of data.
  - Compression: Represent a high-dimensional vector with cluster index.
- Counting:
  - Histograms of texture, color, SIFT vectors.
- Segmentation
  - Separate the image into different regions.
- Prediction
  - Images in the same cluster may have the same labels.

The methods of clustering:

- Agglomerative clustering
  - Start with each points at its own cluster and iteratively merge the closest clusters.
- K-means
  - Iteratively re-assign points to the nearest cluster center.
- Mean-shift clustering
  - Estimate modes of pdf.
- Spectral clustering
  - Split the nodes in graph based on assigned links with similarity weights.

General ideas of clustering:

- Tokens
  - Whatever we need to group (pixels, points, surface elements, etc., etc.,)
- Bottom-up clustering
  - Tokens belong together because they are locally coherent.
- Top-down clustering
  - Tokens belong together because they lie on the same visual entity (object, scene...)
- **Note**: These two above ways are not mutually exclusive.

1.2 Gestalt theory for perceptual grouping

Gestal means "shape, form" in German. Now we use it to represent whole or group. Whole is other than sum of its parts. The relationships among parts can yield new properties/features.

1.3 Agglomerative clustering

Clustering is an unsupervised learning method. Given items $x_1, x_2, \ldots, x_n \in \mathbb{R}$, the goal is to group them into clusters.we need a pairwise distance/similarity function between items,and sometimes the desired number of clusters.

When data (e.g. images, objects, documents) are represented by feature vectors, a commonly used similarity measure is the cosine similarity. Let x, y be two data vectors. There is an angle θ between the two vectors x, y. The cosine similarity is defined as:

$$\begin{aligned} sim(x, y) &= cos(\theta) \\ &= \frac{x^T y}{\|x\| \cdot \|y\|} \\ &= \frac{x^T y}{\sqrt{x^T x}\sqrt{y^T y}} \end{aligned} \tag{1}$$

In contrast, Euclidean distance measure would be:

$$sim(x, y) = x^T y \tag{2}$$

Agglomerative clustering, or bottom-up hierarchical clustering is a simple algorithm to realize the clustering method. The algorithm is:

- Initialization: Every point is its own cluster.
- Repeat:
  - Find "most similar" pair of clusters.
  - Merge them into a parent cluster.
- Until:
  - The desired number of clusters has been reached.
  - Or there is only on cluster.

Then, there are still problems with this algorithm. How to define the cluster similarity? And how many clusters?

How to define cluster similarity?

- Minimum distance (single-linkage – equivalent to MST)
- Maximum distance (complete-linkage)
- Average distance between points
- Medoids ++

How many clusters?

- Clustering creates a dendrogram (a tree)
- Can cut the tree at any level
- Threshold based on max number of clusters or based on distance between merges.

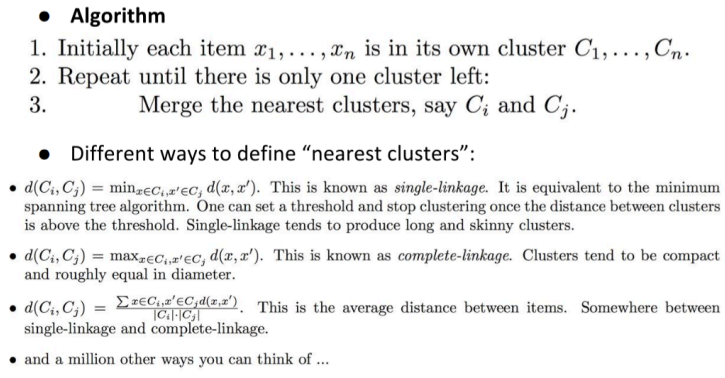

Figure 1: Agglomerative Hierarchical Clustering

Conclusions: Agglomerative Clustering
Pros:

- Simple to implement, widespread application

- Clusters have adaptive shapes
- Provides a hierarchy of clusters not just a single partitioning
- Can incorporate connectivity constrains

Cons:

- May have imbalanced clusters
- Still have to choose number of clusters or threshold
- Need to use an "ultrametric" to get a meaningful hierarchy
- $O(n^2)$ time complexity (versus O(kN) for k-means - stay tuned)

2 K-means and Mean-shift Clustering

2.1 K-means clustering

Goal choose three "centers" as the representative intensties, and label every pixel according to which of these centers it is nearest to.

The best cluster are those that minimize Sum of Square Distance (SSD) between all points and their nearest cluster center c_j:

$$SSD = \sum_{i}^{k} \sum_{x \in c_i} (x - c_i)^2 \tag{3}$$

The goal is to minimize the distortion in data given clusters. Then we can propose the objective function as below:

- Preserve information

$$c^*, \sigma^* = \arg\min_{c,\sigma} \frac{1}{N} \sum_{j}^{N} \sum_{i}^{k} \sigma_{ij} (c_i - x_j)^2 \tag{4}$$

Then we can now give the algorithm of K-means Clustering:

- Initialization: Choose k cluster centers.
- Repeat:
  - Assignment Step: For every point find its closest center.
  - Update Step: Update every center as the mean of its points.
- Until:
  - The maximum number of iterations is reached, or
  - No changes during the assignment step, or
  - The average distortion per point drops very little.

- **Input:** N examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ ($\mathbf{x}_n \in \mathbb{R}^D$); the number of partitions K
- **Initialize:** K cluster centers $\mu_1, \ldots, \mu_K$. Several initialization options:
  - Randomly initialized anywhere in $\mathbb{R}^D$
  - Choose any K examples as the cluster centers
- **Iterate:**
  - Assign each of example $\mathbf{x}_n$ to its closest cluster center

$$\mathcal{C}_k = \{n : \quad k = \arg\min_k ||\mathbf{x}_n - \mu_k||^2\}$$

($\mathcal{C}_k$ is the set of examples closest to μ_k)
  - Recompute the new cluster centers μ_k (mean/centroid of the set $\mathcal{C}_k$)

$$\mu_k = \frac{1}{|\mathcal{C}_k|} \sum_{n \in \mathcal{C}_k} \mathbf{x}_n$$

  - Repeat while not converged

The K-means objective function

- Let $\mu_1, \ldots, \mu_K$ be the K cluster centroids (means)
- Let $r_{nk} \in \{0, 1\}$ be indicator denoting whether point $\mathbf{x}_n$ belongs to cluster k
- K-means objective minimizes the total distortion (sum of distances of points from their cluster centers)

$$J(\mu, r) = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||\mathbf{x}_n - \mu_k||^2$$

- Note: Exact optimization of the K-means objective is NP-hard
- The K-means algorithm is a heuristic that converges to a local optimum [1]

Figure 2: K-means Clustering

2.1.1 K-means: Initialization

- K-means is extremely sensitive to initialization
- Bad initialization can lead to:
  - Poor convergence speed
  - Bad overall clustering
- How to initialize?
  - Randomly from data
  - Try to find K "spread-out" points (k-means++)
- Safeguarding measure:
  - Try multiple initializations and choose the best.

Because of the exist of local minima, our optimizing algorithm would trap in them without looking for the global minima. In order to prevent arbitrarily bad

local minima, the K-means++ is proposed.

The algorithm of K-means++:

1. Randomly choose first center.
2. Pick new center with prob.proportional to $(x - c_i)^2$.
   - (Contribution of x to total error).
3. Repeat until K centers.
4. Expected error O(logK) (optimal).

2.1.2 K-means: Choosing K

- One way to select K for the K-means algorithm is to try different values of K, pot the K-means objective versus K, and look at the "elbow-point" in the plot.

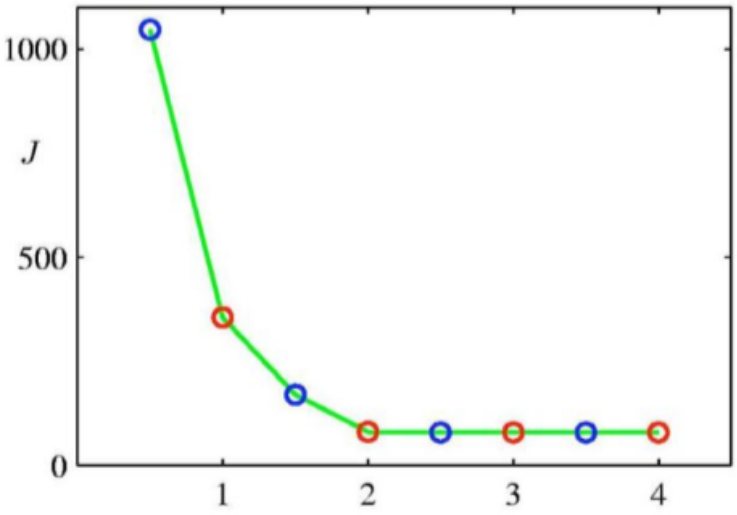

Figure 3: K-means bojective versus K with the elbow point K=2

- Validation set:
  - Try different numbers of clusters and look at performance. When building dictionaries, more clusters typically work better.

2.1.3 How to Evaluate Clusters?

- Generative:
  - How well are points reconstructed from the clusters? → "Distortion".
- Discriminative:
  - How well do the clusters correspond to labels? → Purity
  - Note: unsupervised clustering does not aim to be discriminative.

Algorithm 1 SLIC superpixel segmentation

/ Initialization */*
Initialize cluster centers $C_k = [l_k, a_k, b_k, x_k, y_k]^T$ by sampling pixels at regular grid steps S.
Move cluster centers to the lowest gradient position in a 3×3 neighborhood.
Set label $l(i) = -1$ for each pixel i.
Set distance $d(i) = \infty$ for each pixel i.

repeat
  / Assignment */*
  for each cluster center C_k **do**
    for each pixel i in a $2S \times 2S$ region around C_k **do**
      Compute the distance D between C_k and i.
      if $D < d(i)$ **then**
        set $d(i) = D$
        set $l(i) = k$
      end if
    end for
  end for
  / Update */*
  Compute new cluster centers.
  Compute residual error E.
until $E \leq$ threshold

Figure 4: K-means clustering for superpixels

2.1.4 k-means Clustering: Limitations

- Makes hard assignments of points to clusters
  1. A point either completely belongs to a cluster or not belongs at all.
  2. No notions of a soft assignment (i.e. probability of being assigned to each cluster: say K = 3 and for some point $x_n, p_1 = 0.7, p_2 = 0.2, p_3 = 0.1$)
  3. Gaussian mixture models and Fuzzy K-means allow soft assignments.
- Sensitive to outlier exmaples (such examples can affect the mean by a lot)
  - K-medians algorithm is a more robust alternative algorithm for data with outliers.
  - Reason: Median is more robust than mean in presence of outliers.
- Works well only for round shaped, and of roughtly equal sizes/density/clusters
- Does badly if the clusters have non-convex shapes. Spectral clustering or kernel-sized K-means can be an alternative.

2.1.5 K-means Pros and Cons

- Pros

1. Finds cluster centers that minimize conditional variance (good representation of data)
2. Simple and fast, easy to implement.

- Cons

1. Need to choose K
2. Sensitive to outliers
3. Prone to local minima
4. All clusters have the same parameters (e.g. distance measure is non-adaptive)
5. *Can be slow: each iteration is O(KNd) for N-dimensional points.

2.2 Mean-Shift

Mean-shift is an advanced and versatile technique for clustering-based segmentation.

Algorithm:

1. Initialize random seed, and window W.
2. Calculate center of gravity (the "mean") of $W = \sum_{x \in W} xH(x)$.
3. Shift the search window to the mean.
4. Repeat step 2 and 3 until converge.

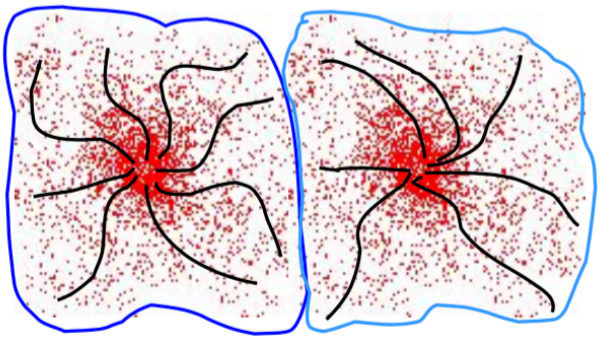

Figure 5: The Trajectories in Mean-Shift Clustering

Cluster: all data points in the attraction basin of a mode.
Attraction basin: the region for which all trajectories lead to the same mode.

So we can apply Mean-shift Clustering / Segmentation in images with:

- Find features (color, gradients, texture, etc)
- Initialize windows at individual pixel locations.
- Perform mean shift for each window until convergence.
- Merge windows that end up near the same "peak" or mode.

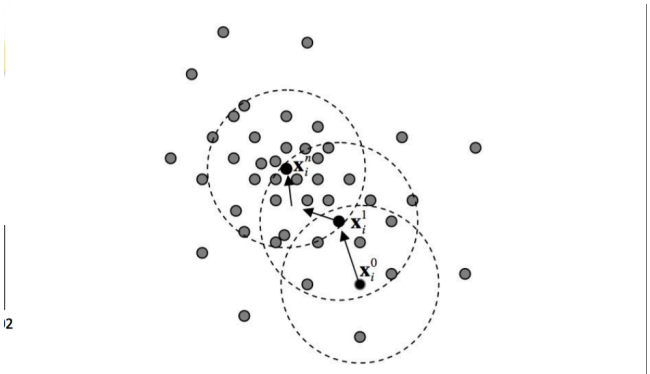

Figure 1: Mean shift procedure. Starting at data point $\mathbf{x}_i$, run the mean shift procedure to find the stationary points of the density function. Superscripts denote the mean shift iteration, the shaded and black dots denote the input data points and successive window centres, respectively, and the dotted circles denote the density estimation windows.

Figure 6: Mean-Shift Clustering

Summary of Mean-shift:

- Pros:
  1. General, application-independent tool
  2. Model-free, does not assume any prior shape (spherical, elliptical, etc.) on data clusters
  3. Just a single parameter (window size h). h has a physical meaning (unlike K-means)
  4. Finds variable number of nodes
  5. Robust to outliers
- Cons:
  1. Output depends on window size
  2. Window size (bandwidth) selection is not trivial
  3. Computationally (relatively) expensive (2s/image)
  4. Does not scale well with dimension of feature space

In order to overcome these cons, there are several modified version of Mean-shift:

3 Tracking motion features - optical flow

3.1 Introduction

Optical flow: Optical flow is the apparent motion of brightness patterns in the image. (note: apparent motion can be caused by lighting changes without any actual motion)

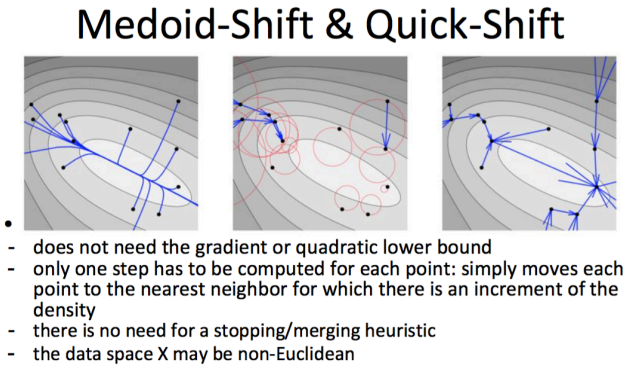

Figure 7: Advanced Mean-Shift Clustering

Feature-tracking: Extract visual features (corners, textured areas) and "track" them over multiple frames. **Goal**: Recover image motion at each pixel from spatio-temporal image brightness variatations (optical flow).

3.2 Estimating optical flow

- Given two subsequent frames, estimate the apparent motion field u(x,y), v(x,y) between them.

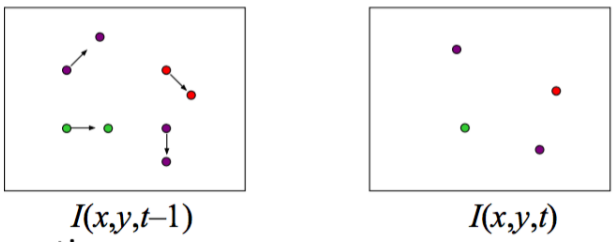

Figure 8: Subsequent frames

- **Key assumptions**:
  - Brightness constancy: projection of the same point looks the same in every frame.
  - Small motion: points do not move very far.
  - Spatial coherence: points move like their neighbors.
- The brightness constancy constraint.
  - Brightness Constancy Equation:

$$I(x, y, t-1) = I(x + u(x, y), y + v(x, y), t) \tag{5}$$

Linearizing the right side using Taylor expansion:

$$\begin{aligned} I(x+u, y+v, t) &\approx I(x, y, t-1) + I_x \cdot u(x, y) + I_y \cdot v(x, y) + I_t \\ I(x+u, v, t) - I(x, y, t-1) &= I_x \cdot u(x, y) + I_y \cdot v(x, y) + I_t \\ Hence,\ I_x \cdot u + I_y \cdot v + I_t &\approx 0 \rightarrow \nabla I \cdot [u, v]^T + I_t = 0 \end{aligned} \quad (6)$$

The component of the flow perpendicular to the gradient (i.e., parallel to the edge) cannot be measured. If (u,v) satisfies the equation, so

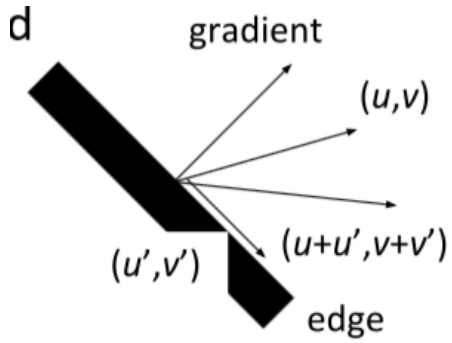

Figure 9: Component of flow

does (u+u', v+v') if

$$\nabla I \cdot [u'\ v'] = 0 \quad (7)$$

– The aperture problem

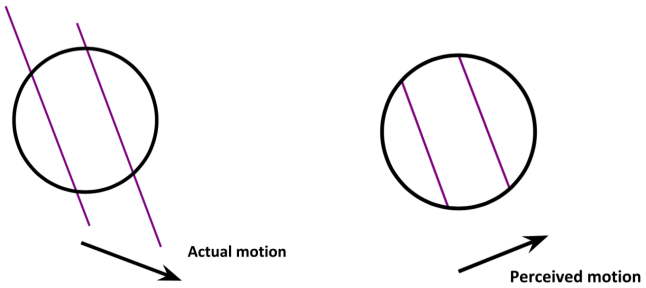

Figure 10: The aperture problem

– Lucas-Kanade flow:
Then we can use formula (6), $\nabla I \cdot [u, v] + I_t = 0$ to recover image motion (u,v) at each pixel. But we now only get one equation with two unknowns.
According to the Spatial coherence constraint - assume the pixel's neighbors have the same (u,v), if we have a 5×5 window, then it gives us 25 equations per pixel.

$$O = I_t(P_1) + \nabla I(P_i) \cdot [u\ v] \quad (8)$$

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \cdots & \cdots \\ I_x(p_25) & I_y(p_25) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \cdots \\ I_t(p_25) \end{bmatrix} \underset{25\times2}{A} \underset{25\times1}{d} = \underset{25\times1}{b} \quad (9)$$

This is called Lucas-Kanade flow. Solve it with least squares solution for d given by $(A^TA)d = A^Tb$, we can get:

$$\begin{bmatrix} \sum I_xI_x & \sum I_xI_y \\ \sum I_xI_y & \sum I_yI_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_xI_t \\ \sum I_yI_t \end{bmatrix} \quad (10)$$

$$A^TA \qquad\qquad A^Tb \quad (11)$$

The summations are over all pixels in the $K \times K$ window.

- When is the Lucas-Kanade equation solvable?
  1. A^TA should be invertible; 2. A^TA should not be too small due to noise which means the eigenvalues λ_1 *and* λ_2 of A^TA should not be too small; 3. A^TA should be well-conditioned which means λ_1/λ_2 should not be too large (λ_1 = larger eigenvalue)
- THEN! The $M = A^TA$ is the **second moment matrix**! The **Harris corner detector**!

$$A^TA = \begin{bmatrix} \sum I_xI_x & I_xI_y \\ \sum I_xI_x & I_xI_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} = \sum \nabla I (\nabla I)^2 \quad (12)$$

- The eigenvectors and eigenvalues of A^TA relate to edge direction and magnitude. The eigenvector associated with the larger eigenvalue points in the direction of fastest intensity change. The other eigenvector is orthogonal to it.

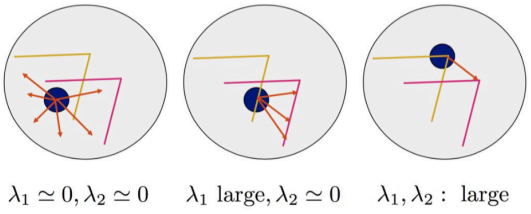

Figure 11: The relationship between eigenvalues and edge

- Errors in Lucas-Kanade equation:
  - Is the small motion assumption always right? - Probably not. Solution: Reduce the resolution!

3.3 Feature tracking

The features with "quality" which can be measured from just a single image are good to track. Hence, tracking Harris corners (or equivalent) guarantees small error sensitivity.

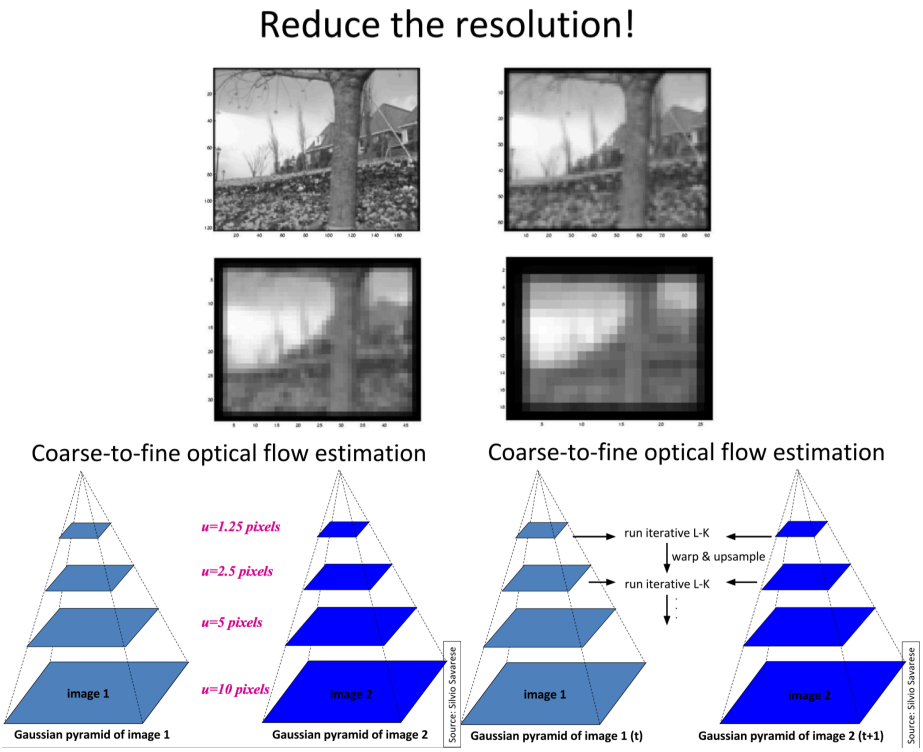

Figure 12: Coarse-to-fine optical flow estimation

- Find a good point to track (Harris Corner)
- Track small patches (5×5) to 31×31 (e.g. using Lucas-Kanade)
- For rigid objects with affine motion: solve motion model parameters by robust estimation (RANSAC)

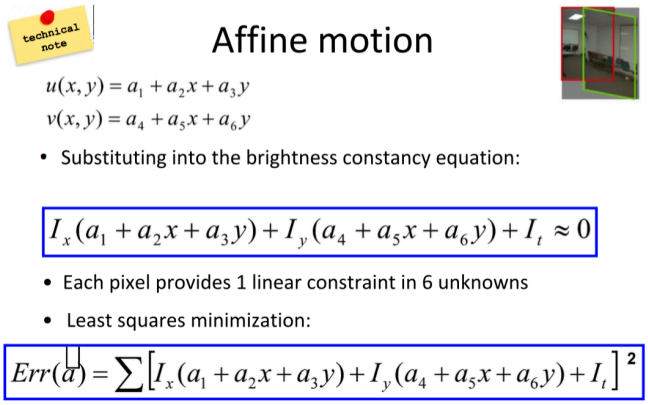

Figure 13: Modification of affine motion

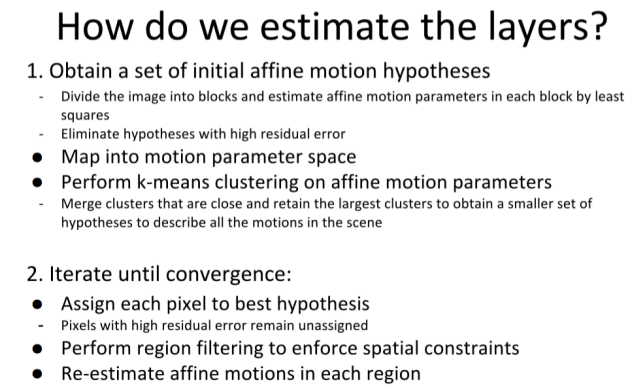

Figure 14: Layers estimation