# CS 131 Computer Vision: Foundations and Applications

Junbin Huang

September 22, 2017

**Abstract**

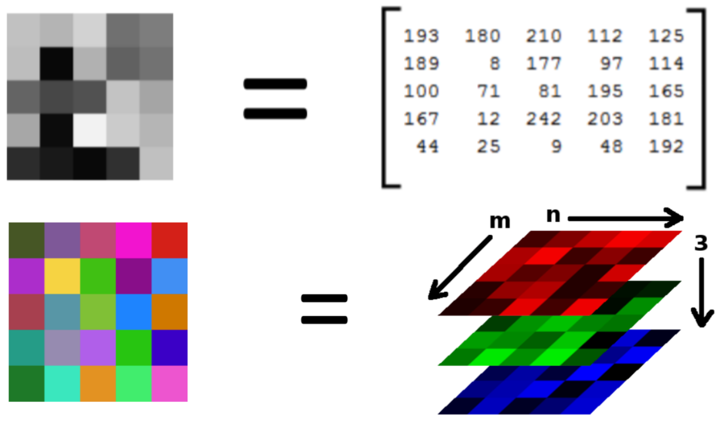
The lecture notes are based on Prof. Feifei Li' lecture on computer vision. This lecture note is based on the first part Pixels, Features, and Cameras.

- Linear Algebra Primer
- Pixels and Filters
- Feature detection: Edge detection, RANSAC, Harris, Difference of Gaussians, SIFT
- Camera

## 1 Linear Algebra Primer

### 1.1 Matrix of image

MATLAB represents an image as a matrix of pixel brightness. The dimension is (n,m,3) for color images and (n,m,1) for grayscale images. Like the picture shown below. Note that the matrix coordinates are **not** Cartesian coordinates which means the upper left corner is [y,x] = (1,1).



$$= \begin{bmatrix} 193 & 180 & 210 & 112 & 125 \\ 189 & 8 & 177 & 97 & 114 \\ 100 & 71 & 81 & 195 & 165 \\ 167 & 12 & 242 & 203 & 181 \\ 44 & 25 & 9 & 48 & 192 \end{bmatrix}$$

Figure 1: matrix of images

## 1.2 Transformation

Matrix can be used to transform the vector in useful way through multiplication $x' = Ax$

- Rotation in coordinate system: New x coordinate is [original vector] dot [the new x axis] and New y coordinate is [original vector] dot [the new y axis].
- Rotation by vector (Counter-clockwise rotation by an angle $\theta$):

$$x' = cos\theta x - sin\theta y$$
$$y' = cos\theta y + sin\theta x$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta x & -sin\theta y \\ cos\theta y & sin\theta x \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{1}$$

  The rotation matrix has the properties of:

$$R \cdot R^T = R^T \cdot R = I$$
$$det(R) = 1$$

- Scaling + Rotation + Translation:: Sometimes we can stick a 1 at the end of each vector, and it is called "homogeneous coordinates". This makes the calculation of changes easier.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix} \tag{2}$$

  Scaling + Rotation + Translation: P' = (T R S) P

$$P' = T \cdot R \cdot S \cdot P$$

$$= \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta x & -sin\theta & 0 \\ sin\theta x & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} RS & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3}$$

## 1.3 Matrix Inverse and Rank

- Inverse: Inverse does not always exist. If $A^-1$ exists, A is invertible or non-singular. Otherwise, it's singular. Inverse does not exist for non-square matrices.
- Inverse in MATLAB: MATLAB will try several appropriate numerial methods (including the pseudoinverse if the inverse does not exist). And it will return the value of X which solves the equation: 1. if there is no exact solution, it will return the closet one. 2. if there are many solution, it will return the smallest one.

- Matrix Rank: 1. Rank means the maximum number of independent column/row vectors of the matrix. 2. The rank of column and row are always equal. 3. For transformation matrices, the rank tells you the dimensions of the output. 4. If an $m \times m$ matrix is rank m, we say it is **full rank**. An inverse matrix can be found. If rank < m, then it is **singular**. Its inverse does not exist.

## 1.4 Singular Value Decomposition (SVD)

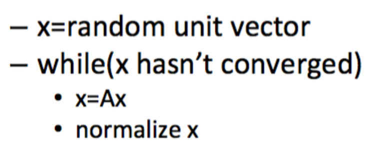
$$U\Sigma V^T = A \tag{4}$$

Where **U** and **V** are rotation matrices and $\Sigma$ is a scaling (diagonal) matrix.

- Principal Component Analysis (PCA): According to the raw value of the $\Sigma$, we can know the importance of the column vectors in the matrix U. We can call the most weighted column vectors of U as the Principal Components of the data - the major patterns that can be added to produce the columns of the original matrix.

- Eigenvector: Suppose we have a square matrix **A**. We can solve for vector x and scalar $\lambda$ such that $Ax = \lambda x$. That means we can transform vector x with A so that the only effect is to scale them with no change in direction. So the vectors are called eigenvectors and the factors $\lambda$ are called eigenvalues.

- Algorithm of eigenvectors:

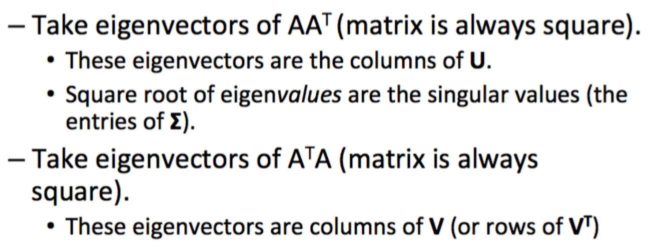


```
– x=random unit vector
– while(x hasn't converged)
    • x=Ax
    • normalize x
```

Figure 2: Algorithm of calculating eigenvectors

- Compute the SVD in MATLAB: The MATLAB command: [U, S, V] = svd(A).

- Algorithm:



– Take eigenvectors of $AA^T$ (matrix is always square).
  - These eigenvectors are the columns of **U**.
  - Square root of eigen*values* are the singular values (the entries of **Σ**).

– Take eigenvectors of $A^TA$ (matrix is always square).
  - These eigenvectors are columns of **V** (or rows of $\mathbf{V}^T$)

Figure 3: Algorithm of calculating SVD

## 2 Pixels and Filters

### 2.1 Linear Systems (Filters)

- Linear filtering: Form a new image whose pixels are a weighted sum of original pixel values. Use the set of weights at each point.

- S is a linear system (function) iff S satisfies:

$$S[\alpha f_1 + \beta f_2] = \alpha S[f_1] + \beta S[f_2] \tag{5}$$

### 2.2 Convolution and (Cross) Correlation

A convolution is a integral that express the amount of overlap of one function as it is shifted over another function.
—-Convolution is a filtering operation
Correlation compares the similarity of two sets of data. Correlation computes a measure of similarity of two input signal as they are shifted by one another. The correlation result reaches a maximum at the time when the two signals math best.
—-Correlation is a measure of relatedness of two signals.

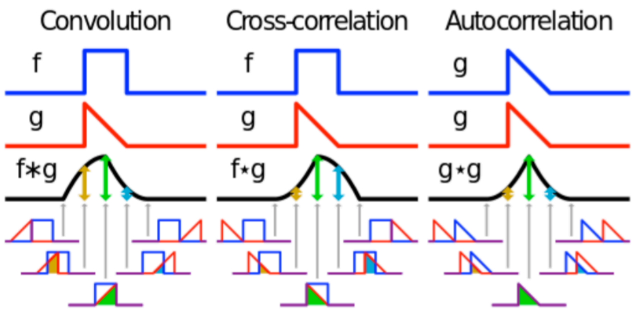


Figure 4: Convolution vs. (Cross) Correlation

**Properties**

- Commutative property:

$$f**h = h**f \tag{6}$$

- Associative property:

$$(f**h_1)**h_2 = f**(h_1**h_2) \tag{7}$$

- Distributive property:

$$f**(h_1 + h_2) = (f**h_1) + (f**h_2) \tag{8}$$

- Shift property:

$$f[n,m]**\sigma_2[n-n_0, m-m_0] = f[n-n_0, m-m_0] \tag{9}$$

- Shift-invariance:

$$\begin{aligned} g[n,m] =& f[n,m] * *h[n,m] \\ \Rightarrow& f[n-l_1, m-l_1] * *h[n-l_2, m-l_2] \\ =& \, g[n-l_1-l_2, m-l_1-l_2] \end{aligned} \tag{10}$$

## 3 Edge Detection

### 3.1 A simple edge detector

An edge in an image is a place of rapid change in the image intensity function.

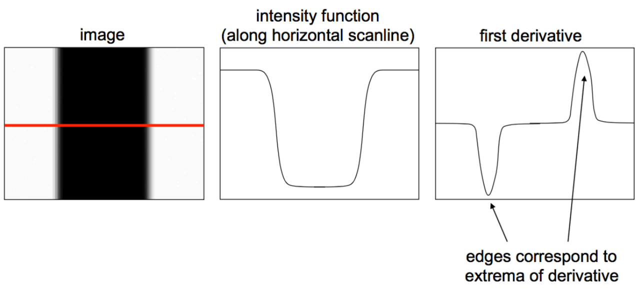


Figure 5: Characterizing edges

So, as a function, we can describe the edge with a mathematic way - the gradient of an image:

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}] \tag{11}$$

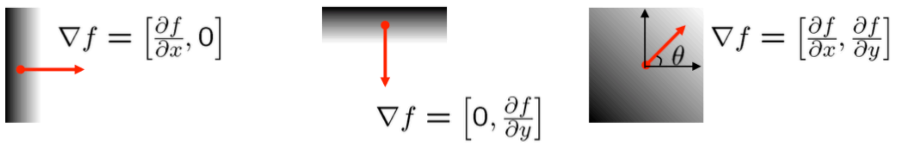

$\nabla f = [\frac{\partial f}{\partial x}, 0]$
$\nabla f = [0, \frac{\partial f}{\partial y}]$
$\theta$
$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$

Figure 6: Gradient

The gradient vector points in the direction of most rapid increase in intensity. The gradient direction is given by

$$\theta = tan^{-1}(\frac{\partial f}{\partial y}/\frac{\partial f}{\partial x}) \tag{12}$$

The edge strength is given by the gradient magnitude:

$$\|\nabla f\| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2} \tag{13}$$

However, finite difference filters respond strongly to noise. The image noise results in pixels that look very different from their neighbors. And generally, the larger the noise, the stronger the response. In order to solve this problem, we can smooth the image by forcing the pixels different to their neighbors to look more like neighbors.

2D Gaussian function is:

$$f(x,y) = e^{\frac{-(x-x_0)^2-(y-y_0)^2}{2\sigma^2}} \tag{14}$$

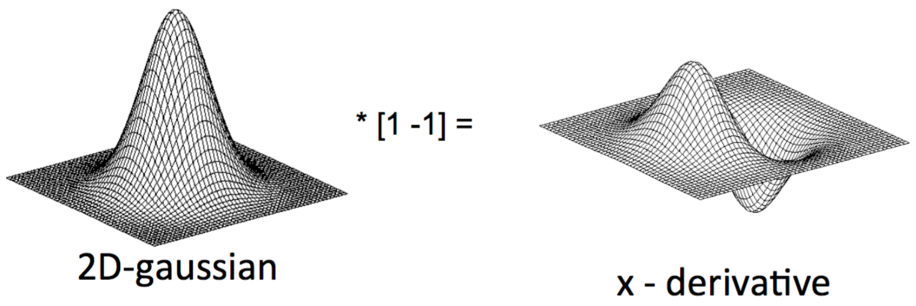


Figure 7: Derivative of Gaussian filter

The following is the derivative theorem of convolution:

$$\frac{d}{dx}(f * g) = f * \frac{d}{dx}g \tag{15}$$

## 3.2 Canny Edge Detector

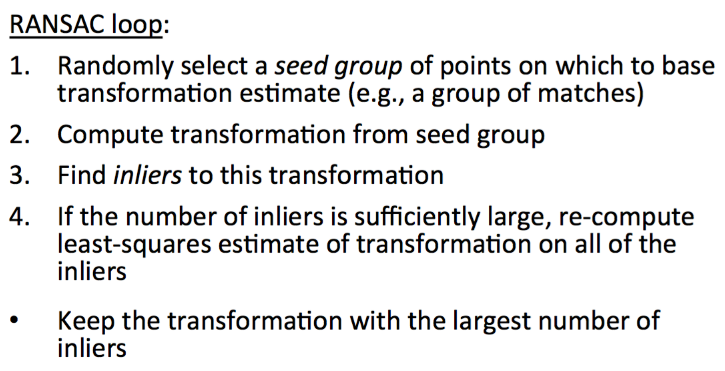
This is probably the most widely used edge detector in computer vision.

- Theoretical model: step-edges corrupted by additive Gaussian noise.
- Canny has shown that the first derivative of the Gaussian closely approximates the operator that optimizes the product of signal-to-noise ratio and localization.
- After apply the DoG to the image, we can get the gradient. And by calculating $\theta = atan2(g_y, g_x)$ (in MATLAB), we can get the orientation at each pixel.
- We find the maximum or minimun value in its surrounding area to get the edge. And then we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points.
- Hysteresis thresholding: use the thresholding to check whether the maximun/minimun value of gradient value is sufficiently large/small. Use a high threshold to start edge curves and a low threshold to continue them.
- MATLAB: edge(image, ’canny’)

## 3.3 RANSAC - a model fitting moethod for edge detection

The full name of RANSAC is Random Sample Consensus. This is an approach to avoid the impact of outliers and look for the inliers.

- Voting: It is not feasible to check all the combinations of features by fitting a model to each possible subst. To handle with this situation, voting is a general technique where we can let the feature vote for all models that are compatible with it. Cycle throuogh features, cast votes for model parameters. Then look for model parameters taht receive a lot of votes. Noise and clutter features will cast votes too, but typically their votes should be inconsistent with the majority of good features.
- Intuition in RANSAC: if an outlier is chosen to compute the current fit, then the resulting line will not have much support from rest of the points.
- The RANSAC loop:



RANSAC loop:

1. Randomly select a *seed group* of points on which to base transformation estimate (e.g., a group of matches)
2. Compute transformation from seed group
3. Find *inliers* to this transformation
4. If the number of inliers is sufficiently large, re-compute least-squares estimate of transformation on all of the inliers

- Keep the transformation with the largest number of inliers

Figure 8: RANSAC loop

- Pros and Cons of RANSAC:
  Pros: 1. General method suited for a wide range of model fitting problems. 2. Easy to implement and easy to calculate its failure rate.
  Cons: Many real problems have high rate of outliers (but sometimes selective choise of random subsets can help).
- A voting strategy, The Hough transform, can handle high percentage of outliers.

# 4 Finding Feature

## 4.1 Local invariant features

The general approach to find the math points between image:

1. Find a set of distinctive keypoints.
2. Define a region around each keypoint.

3. Extract and normalize the region content.
4. Compute a local descriptor from the normalized region.
5. Match local descriptors.

As for the keypoint, we usually choose the **corner points** as keypoint because the corners are repeatable and distinctive. We use the change of intensity for the shift of window to identify the corner: By shifting the window in any

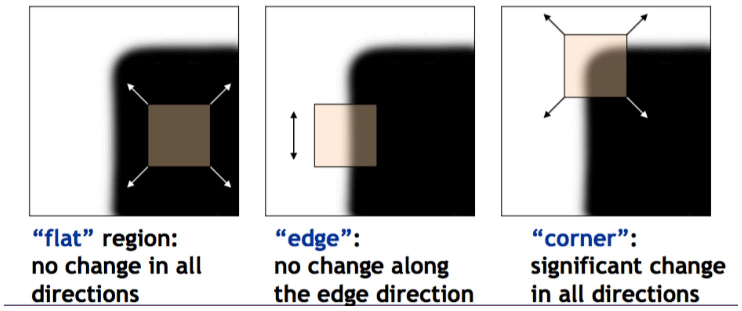


Figure 9: shifting window for feature detecting

direction, it should give a large change in intensity when it is localized in the point of a corner. We can use Harris Detector Formulation to calculate this change. The change of intensity for the shift [u,v] can be represented as:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2 \tag{16}$$

The w(x,y) here is the window function, the I is the intensity function of image.This measure of change can be approximated by

$$E(u,v) \approx [u\ v]M[u\ v]^T$$

Where M is a $2 \times 2$ matrix computed from image derivatives:

$$M = \sum_{\substack{x,\ y \\ (the\ checking\ area)}} w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \tag{17}$$

$$M = \begin{bmatrix} \sum I_xI_x & \sum I_xI_y \\ \sum I_xI_y & \sum I_yI_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} = \tag{18}$$

Let's consider an axis-aligned corner: Then the M becomes:

$$M = \begin{bmatrix} \sum I_x^2 & \sum I_xI_y \\ \sum I_xI_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \tag{19}$$

For general case, since M is symmetric, according to the Eigenvalue Decomposition, we have

$$M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R \tag{20}$$

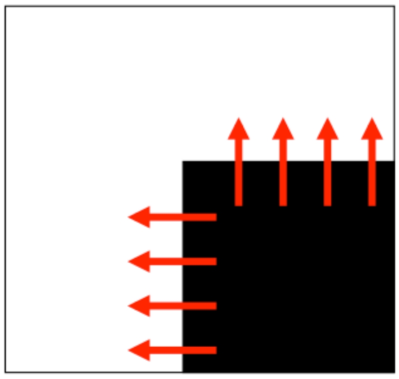


Figure 10: Axis-aligned corner

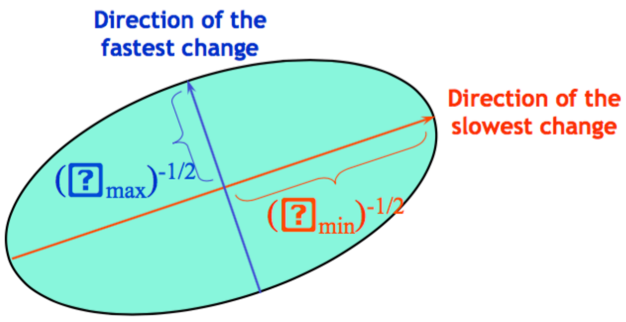

Direction of the fastest change
Direction of the slowest change
(?max)-1/2
(?min)-1/2

Figure 11: Visualization of M

Then we can visualize M as an ellipse with axis lengths determind by the eigenvalues and orientation determind by R: Now, we can classify the image point according to the uigenvalues of M: And, we can use a so-called "Corner Response Function" to do fast approximation:

$$\theta = det(M) - \alpha trace(M)^2 = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2 \tag{21}$$

**Choosing window function:**

- Option 1: uniform window - sum over square window. This function has the problem of being not rotation invariant.
- Option 2 Smooth with Gaussian function. Gaussian already perform weighted sum and its result is rotation invariant.

**Procedure of Harris Detector**

- Compute second moment matrix (autocrorrelation matrix)

$$M(\sigma_1, \sigma_D) = g(\sigma_1) * \begin{bmatrix} I_x^2(\sigma_D) & I_xI_y(\sigma_D) \\ I_xI_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

  where g is the Gaussian filter.

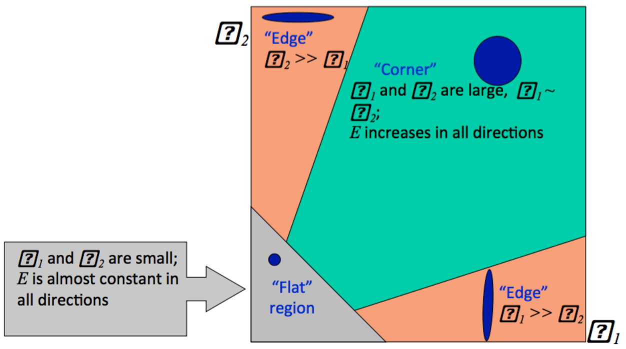

“Edge”
“Corner”
E increases in all directions
“Flat” region
“Edge”
E is almost constant in all directions

Figure 12: Value distribution of the eigenvalues of M

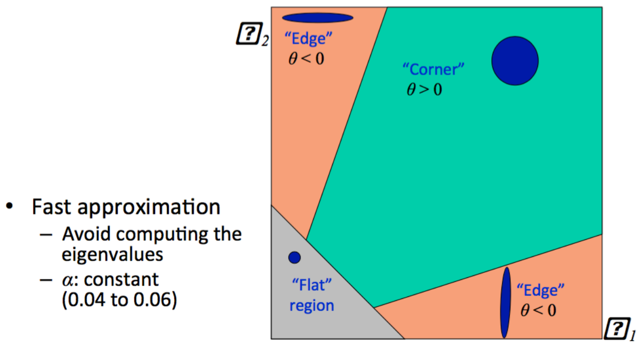


Figure 13: Value distribution of $\theta$

- Calculate the Cornerness funtion and its two eignvalues:

$$\begin{aligned}\theta &= det[M(\sigma_I, \sigma_D)] - \alpha[trace(M(\sigma_I, \sigma_D))]^2 \\ &= g(I_x^2)g(I_y^2) - [g(I_x I_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2\end{aligned}$$

- Perform non-maximum suppression.

**Harris Detector: Properties**

- Translation invariance
- Rotation invariance
- Not invariant to image scale (when the image get magnified, the corner will be classified as edges)

## 4.2 Scale invariant region selection

Solution to find a scale invariant detection:
1. Design a function on the region (circle), which is “scale invariant” (the same for corresponding regions, even if they are at different scales).
2. For a point in one image, we can consider it as a function of region size (circle

radius).
The common approach: Take a local maximum of this function.
**Important**: this scale invariant region size is found in each image independently!

- Function for determining scale:

  $$f = kernel \times image$$

  The Laplacian kernel is:

  $$L = \sigma^2(G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma)) \tag{22}$$

  The Different of Gaussians kernel is:

  $$DoG = G(x,y,k\sigma) - G(x,y,\sigma) \tag{23}$$

  where the Gaussian is $G(x,y,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$ . **Note**: both kernels are invariant to scale and rotation.

- Scale invariant detectors:
  1. Harris-Laplacian [Mikolajczyk, Schmid]: maximize Laplacian over scale, Harris measure of corner response over the image.
  2. SIFT [Lowe]: maximize Difference of Gaussians over scale and space.

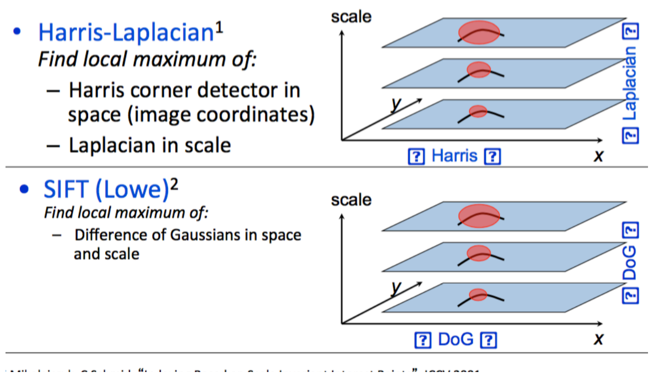


Figure 14: Scale invariant detector

## 4.3 SIFT: an image region descriptor

In order to find a rotation invariant descriptor, for a specific given keypoint and its scale from DoG, we will select a characteristic orientation for the keypoint based on the most prominent gradient there). Then all the feature will be described relative to this orientation and causes features to be rotation invariant (if the keypoint appears rotated in another image, the features will be the same, because they're relative to the characteristic orientation).
**Local Descriptor Formation Procedures:**

- Choosing characteristic orientation:
- Use the blurred image associated with the keypoint's scale. Look at pixels in a square around (say, 16×16)
- Compute gradient direction at each pixel (using vertical and horizontal edge filters)
- Create a histogram of these local gradient directions
- Keypoint orientation = the peak of that histogram (the histogram contribution of each pixel is weighted by the magnitude of its gradient and how close it is to the keypoint)
- Get a stable 2D coordinate (x,y,scale,orientatoin).

**SIFT Descriptor Formation**

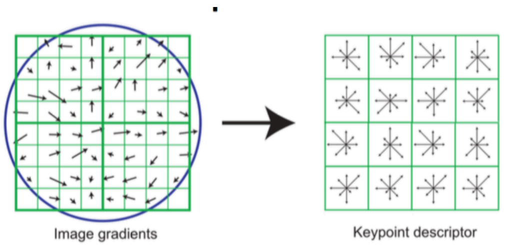


Figure 15: SIFT descriptor

- Using precise gradient location is fragile. We'd like to allow some "slope" in the image, and still produce a very similar descriptor.
- Create array of orientation histograms (a 4×4 array is shown)
- Put the rotated gradients into their local orientation histograms:
  - A gradient's contribution is dividen amonge the nearby histograms based on distance. If it's halfway between two histogram locations, it gives a half contribution to both.
  - Also, scale down gradient contributions for gradients far from the center.
- The SIFT authors found that best results were with 8 orientation bins per histogram, and a 4×4 histogram array.
- 8 orientation bins per histogram, and a 4×4 histogram array, yields $8 \times 4 \times 4 = 128$ numbers.
- So a SIFT descriptor is a length 128 vector, which is invariant to rotation (because we rotated the descriptor) and scale (becausewe we worked with the scaled image from DoG)
- The descriptor is made of gradients (differences between pixels), so it is already invariant to changes in brightness.

- A higher-contrast photo will increase the magnitude of gradients linearly. So, to correct for contrast changes, normalize the vector (scale to length 1.0) is necessary.
- Very large image gradients are usually from unreliable 3D illumination effects (glare, etc). So, to reduce their effect, clamp all values in the vector to be $\leq 0.2$ (an experimentally tuned value). Then normalize the vector again.
- Then we can compare each vector from image A to each vector from image B to find matching keypoints (Euclidean "distance" between descriptor vectors gives a good measure of keypoint similarity).

## 5 Camera Models

### 5.1 Pinhole Camera

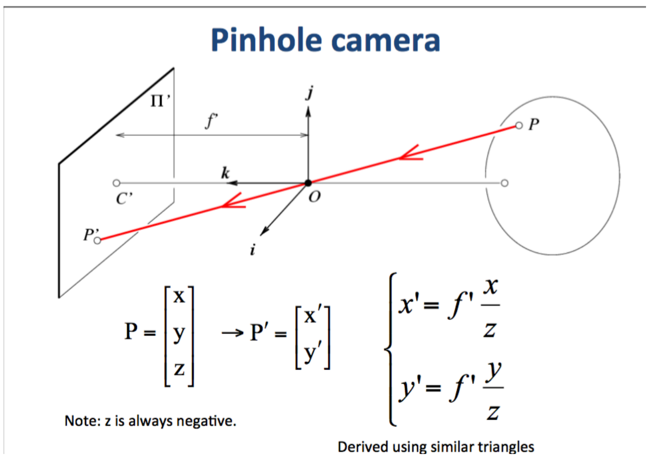

Pinhole camera
Π'
f'
j
P
k
C'
O
P'
i
$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow P' = \begin{bmatrix} x' \\ y' \end{bmatrix}$
$\begin{cases} x' = f' \frac{x}{z} \\ y' = f' \frac{y}{z} \end{cases}$
Note: z is always negative.
Derived using similar triangles

Figure 16: Pinhle camera

1. In Cartesian coordiantes:

$$P = (x, y, z) \rightarrow P' = (f\frac{x}{z}, f\frac{y}{z}) \tag{24}$$

2. In homogeneous coordinates:

$$P' = \begin{bmatrix} fx \\ fy \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{25}$$

Then we can write down the "Projection matrix":

$$P' = MP \tag{26}$$

This equation relate the real world to the image on the camera. Let's use a new matrix K to describe the first 3 columns of M. Then,

$$P' = K[I \quad 0]P \tag{27}$$

This equation has a prerequisite that the optical center sets at (0,0). If the optical center has the coordinate of $(u_0, v_0)$, then

$$P' = K[I \quad 0]P \Rightarrow w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & u_0 & 0 \\ 0 & f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{28}$$

And, what if the pixel in the camera is rectangular but not square? Let's see:

$$P' = K[I \quad 0]P \Rightarrow w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & u_0 & 0 \\ 0 & \beta & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{29}$$

Then...how about an alien camera which has a skew pixels?

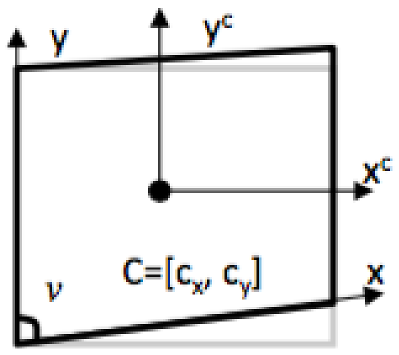


Figure 17: Skewed window

$$P' = K[I \quad 0]P \Rightarrow w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & s & u_0 & 0 \\ 0 & \beta & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{30}$$

Then, the transformed K can be called "intrinsic parameters".

Now, let's talk about an un-usual situation of camera. The final generic

$$P' = K[R \quad \vec{t}]P \Rightarrow w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Figure 18: Extrinsic parameters

projection matrix is shown below:

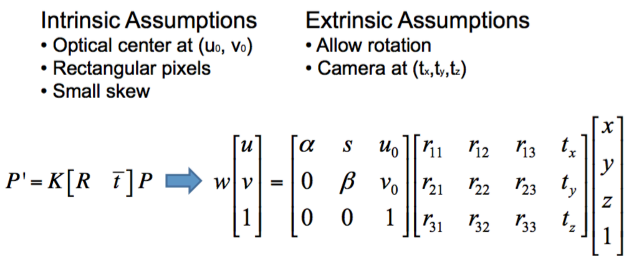

Intrinsic Assumptions
• Optical center at ($u_0$, $v_0$)
• Rectangular pixels
• Small skew
Extrinsic Assumptions
• Allow rotation
• Camera at ($t_x$,$t_y$,$t_z$)
$P' = K\begin{bmatrix} R & \bar{t} \end{bmatrix} P \Rightarrow w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$

Figure 19: Generic projection matrix

# 6 Stereo Vision

## 6.1 Epipolar geometry

The figure below depicts two pinhole cameras looking at point P. In real cameras, the image plane is actually behind the focal center, and produces an image that is the symmetry about the focal center of the lens. Here, however, the problem is simplified by placing a virtual image plane in front of the focal center i.e optical center of each camera lens to produce an image not transformed by the symmetry. O and O' represent the centers of symmetry of the two cameras lenses. P represents the point of interest in both cameras. Points p and p' are the projections of point P onto the image planes. Since the optical centers of

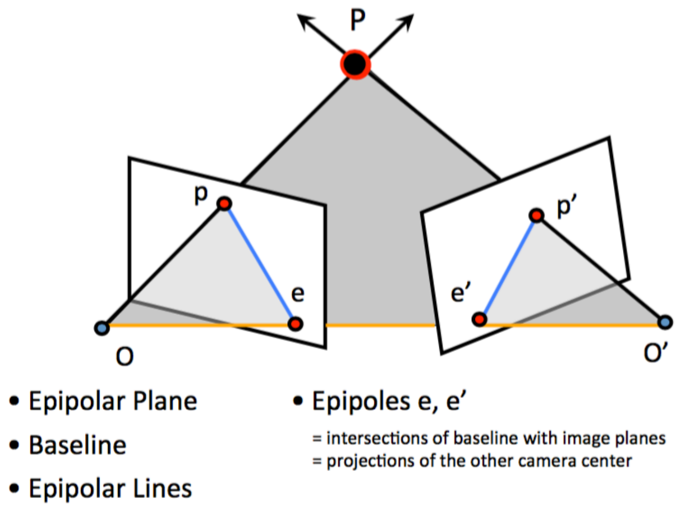

P
p
e
p'
e'
O
O'
• Epipolar Plane
• Baseline
• Epipolar Lines
• Epipoles e, e'
= intersections of baseline with image planes
= projections of the other camera center

Figure 20: Epipolar geometry

the cameras lenses are distinct, each center projects onto a distinct point into the other camera's image plane. These two image points are denoted by e and e' are called epipoles or epipolar points. Both epipoles e and e' in their respective image planes and both optical centers O and O' lie on a single 3D line.

The line O–P is seen by the left camera as a point because it is directly in line with that camera's lens optical center. However, the right camera sees this line as a line in its image plane. That line (e'–p') in the right camera is called an epipolar line. Symmetrically, the line O–P seen by the right camera as a point is seen as epipolar line e–p by the left camera.

An epipolar line is a function of the position of point P in the 3D space, i.e. as P varies a set of epipolar lines is generated in both images. Since the 3D line O–P passes through the optical center of the lens O, the corresponding epipolar line in the right image must pass through the epipole e' (and correspondingly for epipolar lines in the left image). All epipolar lines in one image contain the epipolar point of that image. In fact, any line which contains the epipolar point is an epipolar line since it can be derived from some 3D point P.

As an alternative visualization, consider the points P, O and O' that form a plane called the epipolar plane. The epipolar plane intersects each camera's image plane where it forms lines—the epipolar lines. All epipolar planes and epipolar lines intersect the epipole regardless of where P is located. If the relative

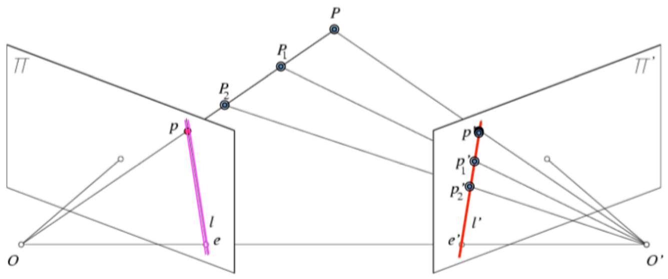


Figure 21: Epipolar constraint

position of the two cameras is known, this leads to two important observations:

- If the projection point xL is known, then the epipolar line e'–p' is known and the point P projects into the right image, on a point x' which must lie on this particular epipolar line. This means that for each point observed in one image the same point must be observed in the other image on a known epipolar line. This provides an epipolar constraint: the projection of P on the right camera plane x' must be contained in the e'–x' epipolar line. Note also that all points P e.g P1, P2, P3 on the O–p line will verify that constraint. It means that it is possible to test if two points correspond to the same 3D point. Epipolar constraints can also be described by the essential matrix or the fundamental matrix between the two cameras.

- If the points p and p' are known, their projection lines are also known. If the two image points correspond to the same 3D point P the projection lines must intersect precisely at P. This means that P can be calculated from the coordinates of the two image points, a process called triangulation.

- Summary: Potential matches for p have to lie on the corresponding epipolar line l'; potential matches for p' have to lie on the corresponding epipolar line l.

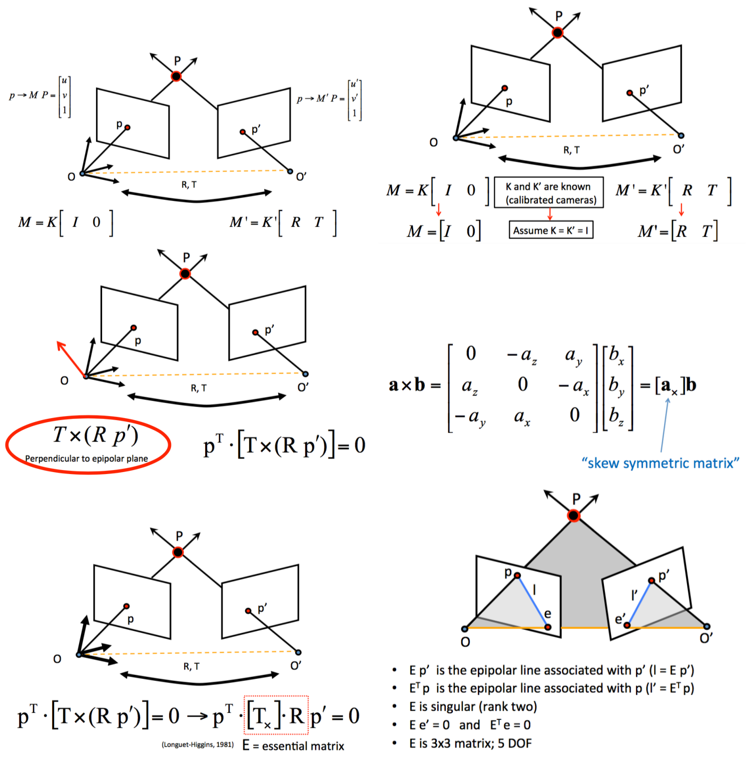


Figure 22: Epipolar transform

Let's talk about a simplest case: parallel images.

Here are the assumptions:

- Image planes of cameras are parallel to each other and to the baseline.
- Camera centers are at same height
- Focal lengths are the same
- Then we can know that: epipolar lines fall along the horizontal scan lines of the images.

According to the epipolar constaint:

$$R = I, \quad t = (T, 0, 0)$$

$$p^T E p' = 0, \quad E = [t_x]T$$

$$E = [t_x]R = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -T \\ 0 & T & 0 \end{bmatrix}$$

The skew symmetric matrix is:

$$[a_x] = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}$$

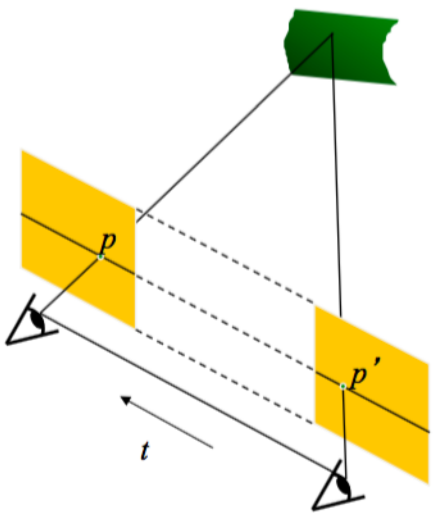


Figure 23: Parallel images

Then,

$$(u\ v\ 1)\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & -T\\ 0 & T & 0\end{bmatrix}\begin{pmatrix}u'\\ v'\\ 1\end{pmatrix} = 0$$

$$\Rightarrow (u\ v\ 1)\begin{bmatrix}0\\ -T\\ Tv'\end{bmatrix} = 0$$

$$\Rightarrow Tv = Tv'$$

Now we can do the triangulation to calculate the depth from disparity. The

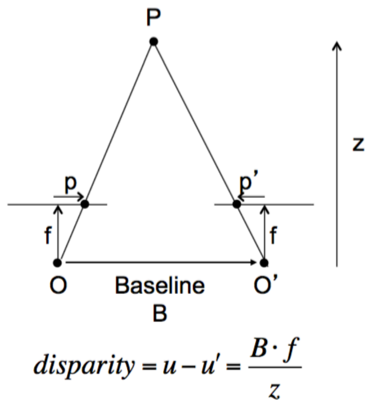


Figure 24: Triangulation

disparity is inversely proportional to depth!

According to this simplest case, we can get into understand the useful stereo image rectification algorithm:

- Re-project image planes onto a common plane parallel to the line between optical centers
- Pixel motion is horizontal after this transformation
- Two transformation matrices, one for each input image reprojection.

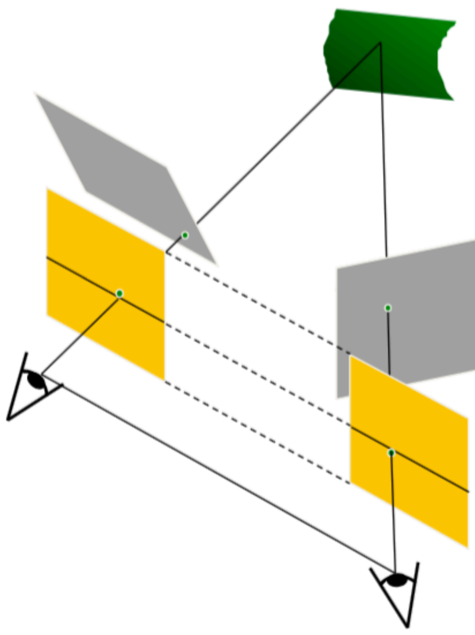


Figure 25: Stereo image rectification

## 6.2 Basic stereo matching algorithm

Assumptions to simplify the problem:

- The baseline is relatively small (compared to the depth of scene points)
- Then most scene points are visible in both views
- Also, matching regions are similar in appearance
- the optical center are at same height

Then we can slide a window along the corresponding scanline and compare contents of that window with the reference window in the original image. The matching cost we use to evaluate the similarity can be SSD Sum of squared differences or normalized correlation.

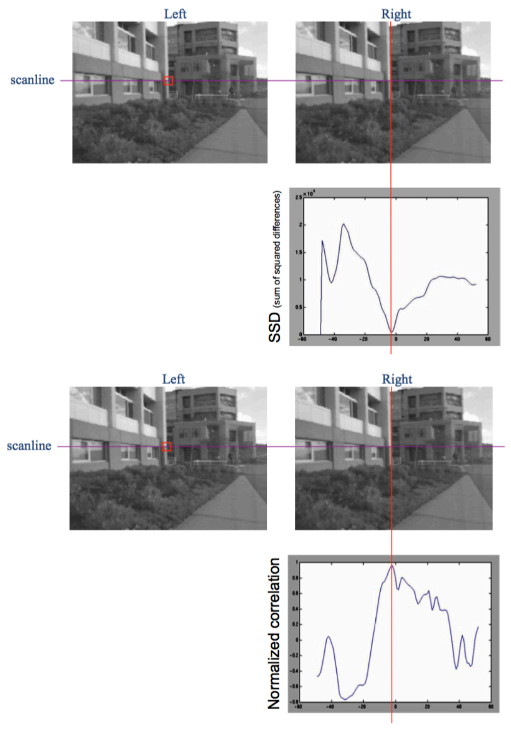


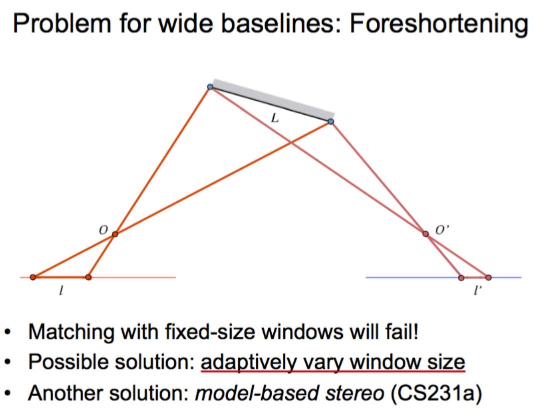
Figure 26: Correspondence search with similarity constraint

Table 1: Effect of window size

| Smaller window | Larger window |
|---|---|
| 1. More detail;<br>2. More noise. | 1. Smoother disparity maps;<br>2. Less detail. |

Table 2: The role of the baseline

| Smaller baseline | Larger baseline |
|---|---|
| Large depth error<br>difficult search problem | |



## Problem for wide baselines: Foreshortening

- Matching with fixed-size windows will fail!
- Possible solution: adaptively vary window size
- Another solution: *model-based stereo* (CS231a)

Figure 27: Correspondence search with similarity constraint