IMAGE UNDERSTANDING

UTKU SAVAŞ

6-BITS

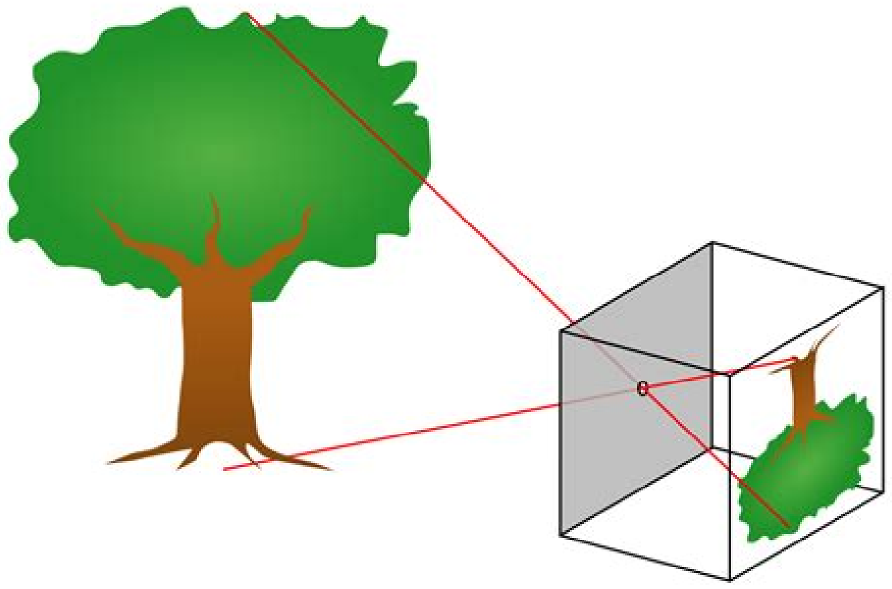
Contents

1 Image Formation and Representation

1.1 Image Formation

1.1.1 Pinhole Cameras

- A pinhole camera is a simple camera without a lens but with a tiny aperture, a pinhole – effectively a light-proof box with a small hole in one side.
- Light from a scene passes through the aperture and projects an inverted image on the opposite side of the box, which is known as the camera obscura effect.
(Camera Obscura // Pinhole Image: Inverted image of the original.)



1.1.2 Modern Cameras

• Lens • Aperture • Shutter • Sensor • Image Processor

1.1.3 Lens

- A camera lens (also known as photographic lens or photographic objective) is an optical lens or assembly of lenses used in conjunction with a camera body and mechanism to make images of objects either on photographic film or on other media capable of storing an image chemically or electronically.
- The two fundamental parameters of an optical lens are the focal length and the maximum aperture.

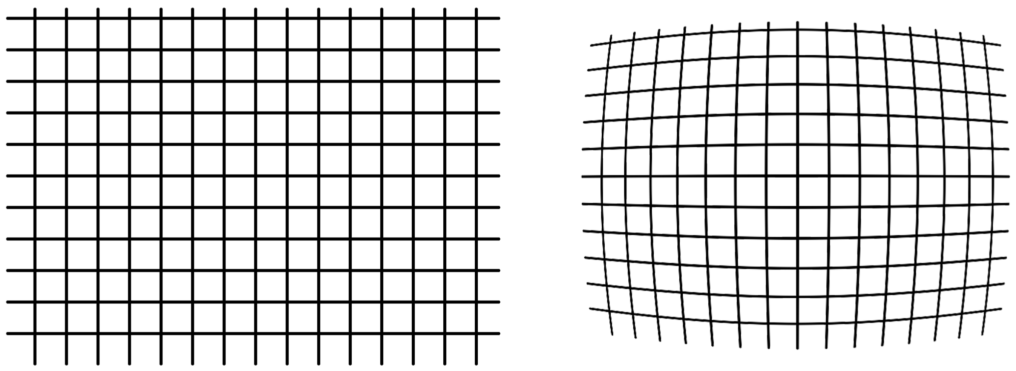
- The lens' focal length determines the magnification of the image projected onto the image plane, and the aperture the light intensity of that image.
- For a given photographic system the focal length determines the angle of view, short focal lengths giving a wider field of view than longer focal length lenses.
- A wider aperture, identified by a smaller f-number, allows using a faster shutter speed for the same exposure.

Focal Length : The distance between lens and the sensor when the subject is in focus.

1.1.4 Aperture

- Aperture is a hole within a lens, through which light travels into the camera body.
- With a wide aperture (f/1.8), it gives a shallow depth of field-sometimes less than a millimetre with a macro lens. Because a lot of light is reaching the sensor, this allows for fast shutter speeds.
- With a narrow aperture (f/22), the depth of field is much greater which is useful for things like landscape photography-it will limit the amount of light reaching your sensor, so you will get slower shutter speeds.
→ With a bigger aperture, background of the image gets blurred.

Radial Distortion:



- Higher distortion for wide angle lenses.
- It is possible to calibrate the camera to estimate the distortion parameters.
- Usually distortion is removed as a preprocessing stage.

1.1.5 Shutter

Shutter is the device that allows light to pass for a determined period.

Shutter Speed:

- Shutter speed or exposure time is the length of time when the film or digital sensor inside the camera is exposed to light, also when a camera's shutter is open when taking a photograph. The amount of light that reaches the film or sensor is proportional to the exposure time.
- Small exposure time is better at taking photographs of moving objects.

1.1.6 Sensor

- An image sensor or imager is a sensor that detects and conveys information used to make an image.

Sensor – Photographic Film:

- Plastic film coated with a gelatin emulsion.
- Contains tiny silver halide crystals that are light sensitive.
- Exposure to light converts the silver halide to metallic silver that blocks light.
- Early forms used in early 1850s, achieved modern form in late 1880s.

Sensor-Digital Era:

- First developed during the 1980s and gradually replaced photographic film.
- Two major types: **CCD** and **CMOS**.
- CCD sensors are generally produce better images.
- CMOS is less expensive and consumes less power.

Sensor Size and Aspect Ratio

- Sensor size describes the physical dimensions of a sensor.
- Sensor aspect ratio merely defines the ratio between width and height of the sensor.
- ”35 mm” is the size of a single image on photographic film.

Sensor-Global/Rolling Shutter

Rolling Shutter: Cheaper cameras read sensor pixels one after another one at slightly different times.
Global Shutter: High-end cameras read all sensor pixels at the same time.

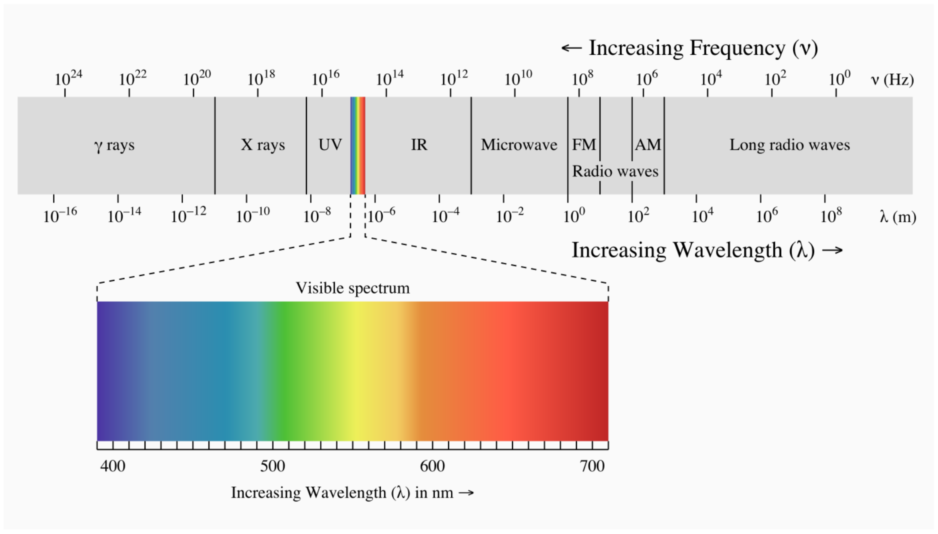
Analog to Digital(A/D) Converter

- Sensors capture analog data.
- The data need to be quantized to be processed by a digital circuit.

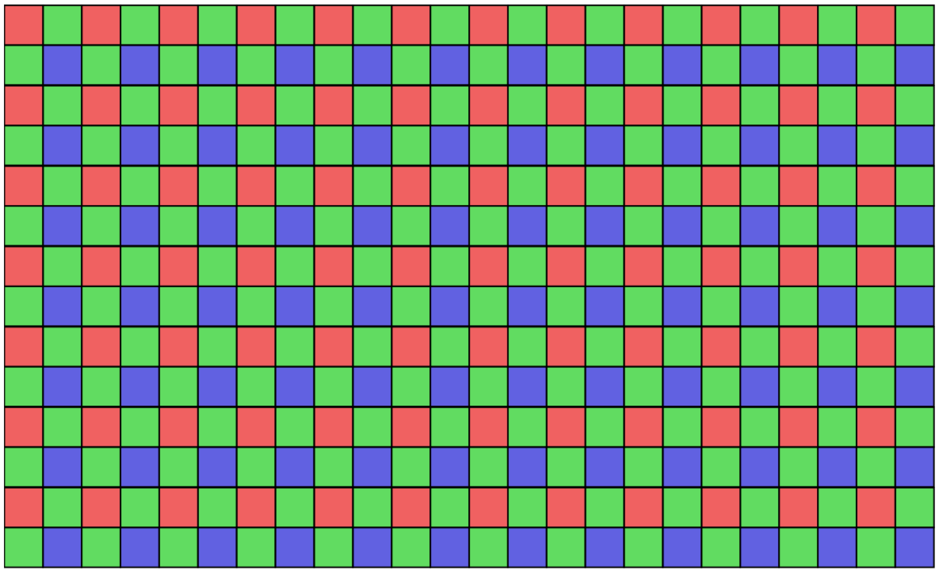
- An Analog-to-Digital Converter(ADC) quantizes the analog signal usually into a 12-14 bits digital signal(RAW).
- Most current image representations like JPEG use an 8-bits representation of this data.

1.2 Color and Hyper-spectral Imaging

1.2.1 Light Spectrum


← Increasing Frequency (ν)
10^{24} 10^{22} 10^{20} 10^{18} 10^{16} 10^{14} 10^{12} 10^{10} 10^{8} 10^{6} 10^{4} 10^{2} 10^{0} ν (Hz)
γ rays
X rays
UV
IR
Microwave
FM
AM
Radio waves
Long radio waves
10^{-16} 10^{-14} 10^{-12} 10^{-10} 10^{-8} 10^{-6} 10^{-4} 10^{-2} 10^{0} 10^{2} 10^{4} 10^{6} 10^{8} λ (m)
Increasing Wavelength (λ) →
Visible spectrum
400 500 600 700
Increasing Wavelength (λ) in nm →

1.2.2 Color Filter Arrays

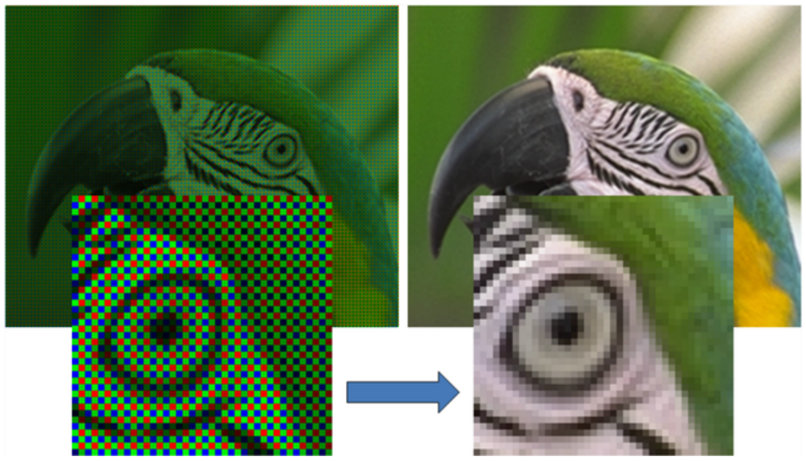


Bayer Pattern

- A bayer pattern is consist of %25 Red, %50 Green, and %25 Blue pixels.

1.2.3 Demosaicking

- A demosaicing (also de-mosaicing, demosaicking or debayering) algorithm is a digital image process used to reconstruct a full color image from the incomplete color samples output from an image sensor overlaid with a color filter array (CFA)
- Demosaicking algorithm is used to interpolate color information to create a full array RGB image data.



1.2.4 Infrared Imaging

- Infrared imaging is a technique of capturing invisible infrared images and converting them into visible images.

1.2.5 Hyper-spectral Imaging

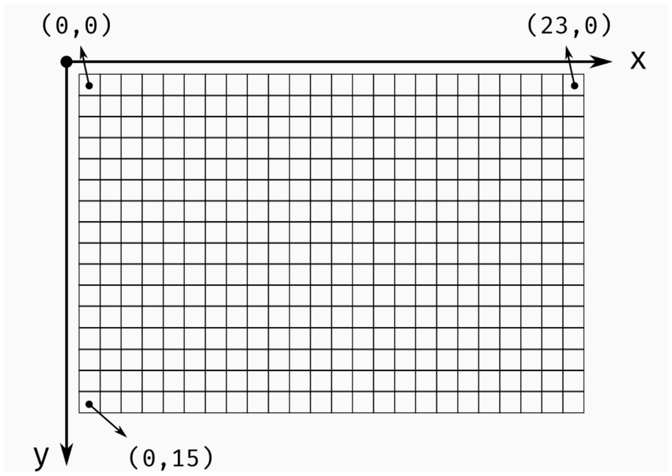
- Hyper-spectral imaging,collects and processes information from across the electromagnetic spectrum.
- Hyper-spectral imaging produces a spectrum(represented by several hundred numbers) at each pixel in an image. While grayscale or colour images can discriminate between-rocks and vegetarian- hyperspectral images can discriminate between different types of rock and vegetarian. Spectral signature images revealing objects chemical composition.

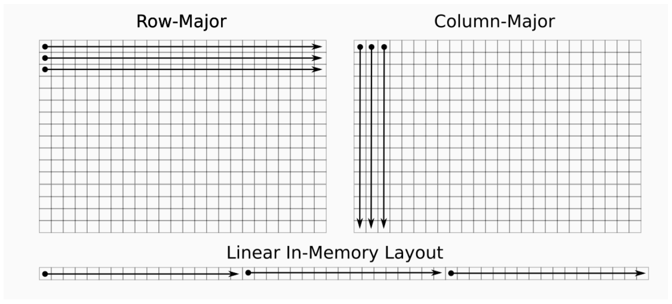
Spectroscopy
- Spectroscopy is a commonly used technique for analyzing the composition of materials.
- Accurate spectral analysis of one spatial pixel only.

Color Imaging
- Seeing RGB colors of one image only.

1.3 In-memory Representations

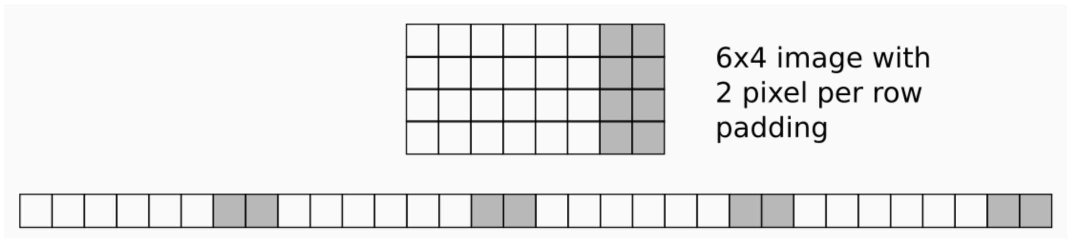
1.3.1 Images as Matrices





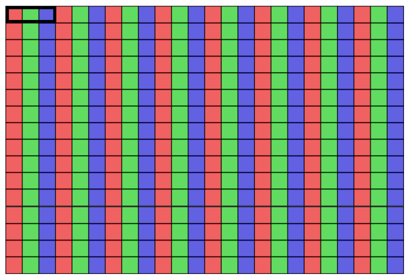
1.3.2 Padding

- Sometimes image data is padded to make it a multiple of 4/8 bytes per-row.
- The amount of bytes to skip between rows/columns is usually called step/rowStep/leadingDimension/stride.



1.3.3 Multichannel Images

Multichannel Images – RGB Interleaved



- Single pixel consists of 3 channels{R=Red, G=Green, B=Blue}.

Multichannel Images – Transparency
- Single pixel consists of 4 channels{R=Red, G=Green, B=Blue, A=Alpha}
- Sometimes there is need to specify some parts of the image as transparent.
- Transparency is stored in the alpha channel.
- Usually, alpha is 255 for fully opaque parts and it is 0 for fully transparent parts.
- Other channel orderings are possible such as BGRA, ARGB, etc.

Multichannel Images – Endianness
Endianness: Refers to the sequential order in which bytes are arranged into larger numerical values when stored in memory or when transmitted over digital links.
- Be careful how you access pixel data programmatically.
- Using **uint8_t*** or **uint32_t*** to access pixels of an image might yield different results due to endianness issues.

1.4 Non-volatile Representations

1.4.1 Raster/Vector

- In computer graphics , a raster graphic is a dot matrix data structure that represents a generally rectangular grid of pixels.
- A bitmap,single bit raster, corresponds bit-for-bit with an image displayed on a screen, generally in the same foramt used for storage in the display's video memory, or maybe as a device independent bitmap.
- A raster is technically characterized by the width and height of the image in pixels and by the

# of bits per pixel(or color depth which determines the # of the colors it can represent).
- Vector graphics are computer graphics images that we defined in terms of 2D points which are connected by lines and curves to form polygons and other shapes. Each of these points has a definite position on the X and Y axis of the work plane and determines the direction of the path; further, each path may have various properties including values for stroke color, shape, curve, thickness, and fill.

1.4.2 Lossy/Lossless Compression

Lossy Compression: - In information technology, **lossy compression** or **irreversible compression** is the class of data encoding methods that uses inexact approximations and partial data discarding to represent the content.
- These techniques are used to reduce data size for storing, handling, and transmitting content.

Lossless Compression:
- Lossless compression is a class of data compression algorithms that allows the original data to be perfectly reconstructed from the compressed data.
By contrast, lossy compression permits reconstruction only of an approximation of the original data, though usually with improved compression rates (and therefore reduced file sizes).

1.4.3 Raster Image Formats

PNM
- A family of formats PPM/PGM/PBM
- Easy to read/write
- No compression
- Good for compatibility

PNG
- Short for Portable Network Graphics
- Created as a patent-free version of GIF
- Supports lossless compression and transparency
- Good for computer graphics especially when you have sharp edges

JPEG
- Short for Joint Photographic Experts Group
- Good for natural images, most cameras record JPEG images
- Does not support transparency
- Possibility to save extra data in Exif format

1.4.4 Vector Image Formats

SVG/PDF/PS

• Vector formats do not store pixels but vector data for generating the image.
• This means they can be scaled without artifacts
• For images the most common format is SVG, scalable vector graphics.
• Good for text, plots, etc.
• PDF and PS are also vector formats

2 Basic Image Processing

2.1 Per-pixel Operations

2.1.1 Brightness and Contrast

- We can transform image pixel values linearly to change the brightness of the image:

$$\begin{gathered} \bullet I(x,y) \leftarrow max(min(\alpha * I(x,y) + c, 255), 0) \\ \bullet I(x,y) \leftarrow clamp(\alpha * I(x,y) + c) \\ \alpha \rightarrow Contrast \\ c \rightarrow Brightness \end{gathered}$$

$255 + 1 \neq 256 (0 \leq I(x,y) \leq 255)$
$255 + 1 = 0 \rightarrow$ Operation is a modular type operation

2.1.2 Grayscale Conversion

- We can convert a color image to grayscale by taking a weighted average of color channels per pixel:
- The coefficients usually heavily weights the green channel but depend on the application:

$$\bullet I_G(x,y) \leftarrow 0.299 \times R(I_C(x,y)) + 0.587 \times G(I_C(x,y)) + 0.114 \times B(I_C(x,y))$$

2.1.3 Thresholding

- We can create a black and white image by thresholding the image:
- This is useful for segmentation if foreground and background objects are different enough:

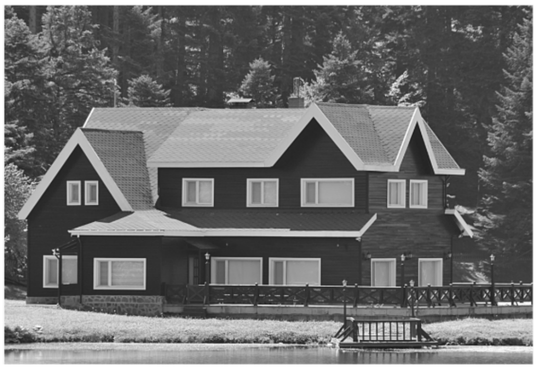
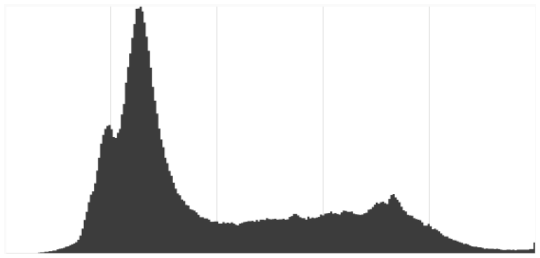
$$\bullet I(x,y) \leftarrow \begin{cases} 0 \quad if \quad I(x,y) < t, \\ 255 \quad if \quad I(x,y) \geq t \end{cases}$$

2.1.4 Histograms

- An intensity histogram is an array of counts of pixel values that fall into specific range of values:

$$H(i) = \{\text{\# of pixels between i} * l \text{ and } (i+1) * l\},$$

$$l = \frac{256}{\text{\# of histogram bins}}$$

2.2 Operations on Local Regions

2.2.1 Averaging

- Things get more interesting if the transformations we perform can operate on the values if the neighboring pixels around (x, y), not just the value of I(x, y)
- For example consider averaging the pixel values in a neighborhood of 3 × 3:

$$\bullet I(x,y) \leftarrow \sum_{i=-1}^{+1} \sum_{j=-1}^{+1} \frac{1}{9} \times I(x+i, y+j)$$

$$I(x,y) \leftarrow I(x,y) * \begin{array}{|c|c|c|} \hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \end{array} \rightarrow$$

This operation is not a mathematical multiplication.
It is a convolution operation.
We put the center of the kernel on the selected pixel in matrix(image).
After the convolution operation, we get a new value for the selected pixel.

2.2.2 Handling Borders

- Close to the borders, the neighborhood of a pixel is undefined, i.e. I(−1, 0) does not exist. So we need a policy to handle the pixels that are outside the borders:
Assume a constant value: We can assume $I(x, y) = c$ outside the borders.
Assume periodic: We can assume $I(x + k * w, y + k * h) = I(x, y)$ outside the borders.
Assume reflection/mirroring: We can assume $I(x + i, y + j) = I(x + k * w - i, y + k * h - j)$ outside the borders.
Assume replicated border: We can assume the border pixels I(x, 0), I(0, y), I(w − 1, y), I(x, h − 1) and the corners repeat indefinitely.

2.2.3 Separability

- For averaging, instead of summing up the neighborhood pixels individually,

$$I(x,y) \leftarrow \frac{I(x-1,y-1)}{9}+\frac{I(x,y-1)}{9}+\frac{I(x+1,y-1)}{9}+\frac{I(x-1,y)}{9}+\frac{I(x,y)}{9}+\frac{I(x+1,y)}{9}+\frac{I(x-1,y+1)}{9}+\frac{I(x,y+1)}{9}+\frac{I(x+1,y+1)}{9}$$

We can do the following instead:

$$\begin{aligned} I(x,y) \leftarrow & \tfrac{1}{3}\times\left(\tfrac{I(x-1,y-1)}{3}+\tfrac{I(x,y-1)}{3}+\tfrac{I(x+1,y-1)}{3}\right) \\ & +\tfrac{1}{3}\times\left(\tfrac{I(x-1,y)}{3}+\tfrac{I(x,y)}{3}+\tfrac{I(x+1,y)}{3}\right) \\ & +\tfrac{1}{3}\times\left(\tfrac{I(x-1,y+1)}{3}+\tfrac{I(x,y+1)}{3}+\tfrac{I(x+1,y+1)}{3}\right) \end{aligned}$$

- This is nice because we can share computations for $I(x,y)$ and $I(x^{'},y^{'})$ if they are close enough!

$$\text{I(x,y)} \leftarrow I(x,y) * \begin{array}{|c|c|c|}\hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \end{array} \iff I(x,y) \leftarrow I(x,y) * \begin{array}{|c|c|c|}\hline \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \hline \end{array} * \begin{array}{|c|}\hline \frac{1}{3} \\ \hline \frac{1}{3} \\ \hline \frac{1}{3} \\ \hline \end{array}$$

- The ability to split a 2D kernel into a set of 1D kernels is called <u>separability</u>.
- Every rank one matrix A can be written as $U^T V$.

2.2.4 Gaussian Smoothing

- Averaging with the same weight is not so nice we want to give the center area more weight.
- A mathematically nice way of doing that is to pick the weights to resemble the Gaussian curve:

For 1D $\rightarrow G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(x-\mu)^2}{2\sigma^2}}$ For 2D $\rightarrow G(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} \exp^{-\left[\frac{(x-\mu_x)^2}{2\sigma_x^2}+\frac{(y-\mu_y)^2}{2\sigma_y^2}\right]}$

μ : mean & σ^2 : variance & σ : standart deviation of the distribution

- The Gaussian smoothing operator is a 2D convolution operator that is used to "blur" images and remove details and noise. In this sense, it is similar to the mean filter , but it uses a different kernel that represents the shape of a Gaussian('bell shaped') bump.

<u>Convolution</u>
- Convolution is a simple mathematical operation which is fundamental to many common image processing operators. Convolution provides a way of 'multiplying together' two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values.

- The convolution is performed by sliding the kernel over the image, generally starting at the top left corner, so as to move the kernel through all the positions where the kernel fits entirely within the boundaries of the image. (Note that implementations differ in what they do at the edges of images, as explained below.) Each kernel position corresponds to a single output pixel, the value of which is calculated by multiplying together the kernel value and the underlying image pixel value for each of the cells in the kernel, and then adding all these numbers together.

Kernel :
- A kernel is a (usually) smallish matrix of numbers that is used in image convolutions. Differently sized kernels containing different patterns of numbers give rise to different results under convolution. For instance, Figure 1 shows a 3×3 kernel that implements a mean filter.

Mean Filter
- The idea of mean filtering is simply to replace each pixel value in an image with the mean ('average') value of its neighbors, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings.

$$\begin{array}{|c|c|c|}\hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \end{array} \Rightarrow \text{ 3x3 averaging kernel often used in mean filtering}$$

2.2.5 Image Derivatives

- Smoothing makes the image edges slowly disappear. We might want to make make them stronger instead. If so we can use the derivative filters:
- It is usually nicer to smooth a little bit before you take the derivative

$$I_x(x,y) \leftarrow I(x,y) * \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|}\hline +1 \\ \hline 0 \\ \hline -1 \\ \hline \end{array} \qquad I_y(x,y) \leftarrow I(x,y) * \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|}\hline -1 \\ \hline 0 \\ \hline +1 \\ \hline \end{array}$$

2.2.6 Simple Edge Detection

- Once you compute $I_x(x,y)$ and $I_y(x,y)$, you can compute the edge strength as:

$$I_E(x,y) \leftarrow I_x(x,y)^2 + I_y(x,y)^2$$

- You can threshold this value to get an edge map.
- A more sophisticated method is to use the Canny edge detector(Uses a more intelligent thresholding algorithm).

2.3 Geometric Transforms

2.3.1 Scaling an Image: Nearest Neighbor Interpolation

- Consider $x^{'}$and $y^{'}$ are locations from the scaled image.

If [integer part]$\frac{x'}{\text{scale factor}} = x$ & [integer part]$\frac{y'}{\text{scale factor}} = y \Rightarrow I'(x', y') = I(x, y)$

- x and y are locations from the original image and I(x,y) is the pixel value of (x,y) coordinate.

$$\begin{array}{|c|c|}\hline 10 & 4 \\ \hline 2 & 8 \\ \hline \end{array} \xrightarrow[\text{by 2}]{\text{Scale}} \begin{array}{|c|c|c|c|}\hline 10 & 10 & 4 & 4 \\ \hline 10 & 10 & 4 & 4 \\ \hline 2 & 2 & 8 & 8 \\ \hline 2 & 2 & 8 & 8 \\ \hline \end{array}$$

2.3.2 Scaling an Image: Bilinear Interpolation

- If $x' = x + \alpha$ with integer x and $0 \leq \alpha \leq 1$, then we can linearly interpolate $f'(x')$ from $f(x)$ and $f(x+1)$ as

$$f'(x') = (1-\alpha)f(x) + \alpha f(x+1)$$

- In 2D, if $x' = x + \alpha$ and $y' = y + \beta$ with integer $x, y, 0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$, then we can bilinearly interpolate $I'(x', y')$ from $I(x, y), I(x+1, y), I(x, y+1)$ and $I(x+1, y+1)$ as

$$I'(x', y') = (1-\alpha)(1-\beta)I(x, y) + \alpha(1-\beta)I(x+1, y) + (1-\alpha)\beta I(x, y+1) + \alpha\beta I(x+1, y+1)$$

2.3.3 2D Rotations

- In rotation, we only change the locations, not the intensity value of pixel.

$$I_\theta(x', y') = I(x, y) \Rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \left.\vphantom{\begin{bmatrix} x \\ y \end{bmatrix}}\right\} \begin{array}{c} \text{Rotate from (0,0)} \\ \text{Clockwise rotation} \end{array}$$

- If we want to rotate not from (0,0) but a specific point (w', h'), we do:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \left(\begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} w' \\ h' \end{bmatrix} \right) \right) + \begin{bmatrix} w' \\ h' \end{bmatrix}$$

- If we want to show the rotated image with all parts(fit to image) - First we rotate corners
- We calculate the corners' new locations
- Find the distance between opposed corners using Euclidean distance formula
- Scale the image with $\frac{\text{new width distance}}{\text{width}}$ and $\frac{\text{new height distance}}{\text{height}}$

2.3.4 Similarity Transformations

- We can apply scaling to rotation in a different manner

$$I_S(x', y') = I(x, y) \Rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = s \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \left.\vphantom{\begin{bmatrix} x \\ y \end{bmatrix}}\right\} \text{ s} \rightarrow \text{scale factor}$$

2.3.5 Affine Transformations

- A is an any invertible matrix.

$$I_A(x', y') = I(x, y) \Rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} A_1 & A_3 \\ A_2 & A_4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\text{A can be written as } R_{-\theta} \begin{bmatrix} \lambda_1 & \\ & \lambda_2 \end{bmatrix} R_\theta$$

3 Image Pyramids

3.1 Multi-scale Image Representation

3.1.1 The Need for Multiple Scales

- The world is inherently multi-scale, everything is composed of things at a smaller scale. This is also true for image content:

- Fine v.s. coarse texture on objects

- Same object at different depths

3.1.2 The Scale Space

- We can build a multi-scale representation using the scale space approach. In the scale space representation an image becomes a function of three parameters: $I(x, y, \sigma)$
- The σ parameter represents the scale, image features smaller than σ does not appear in the image.
- We can convolve with a Gaussian filter to increase the scale parameter. If the current scale is σ_c and the target sigma is σ_t, then we should use a filter with σ_f that satisfies

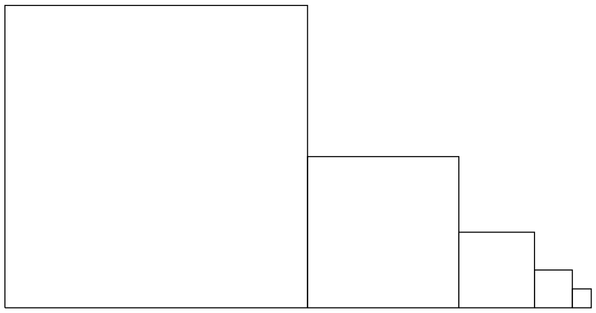
$$\sigma_t^2 = \sigma_c^2 + \sigma_f^2$$

- So for example to double the scale σ_0 , we need a filter with $\sqrt{3}\sigma_0$
- With bigger σ parameter, image gets more blurred.
- Images have depth value. Depth of image gives information about "depth" of an object or "z" information of the object in real world coordinate system.

3.2 Pyramid Types

3.2.1 Gaussian Pyramids with Coarse Scale

- Every doubling of the scale parameter is equivalent to a change of one octave (analogy from musical notes).
- Once the scale parameter is doubled we can subsample the image by two in each direction.
- This results in an image pyramid, with each pyramid level seperated by an octave of scale change:



- Because of subsampling σ value doesn't change between original and subsampled images.

3.2.2 Gaussian Pyramids with Fine Scale

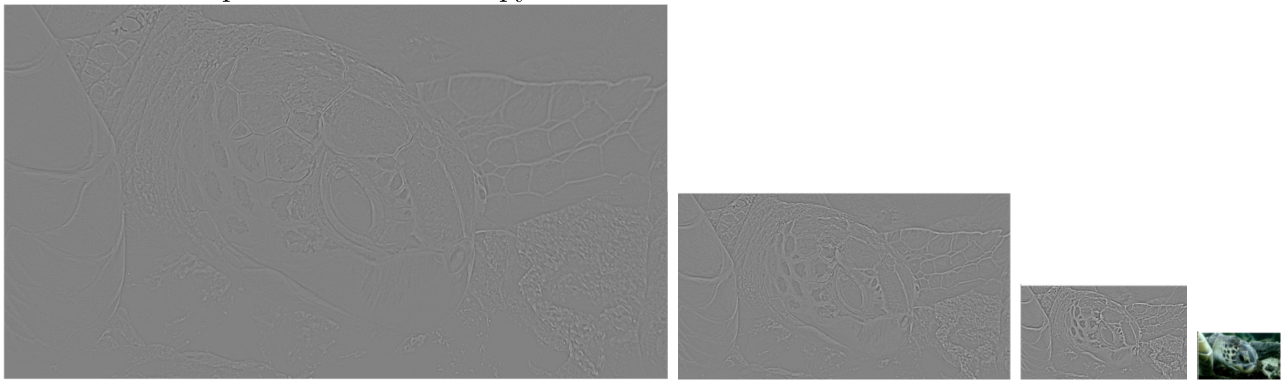
- Sometimes we want a denser sampling of the scale space.
- In this case we can generate N images per octave and subsample only after reaching the end of an octave.
- How should we pick scales within an octave?

$$\begin{array}{ccccccccc}
\sigma_0 & & & ... & & & 2\sigma_0 & ... & 4\sigma_0 \\
2^0\sigma_0 & & & ... & & & 2^1\sigma_0 & ... & 2^2\sigma_0 \\
2^{\frac{0}{N}}\sigma_0 & & & ... & & & 2^{\frac{N}{N}}\sigma_0 & ... & 2^{\frac{2N}{N}}\sigma_0 \\
2^{\frac{0}{N}}\sigma_0 & 2^{\frac{1}{N}}\sigma_0 & ... & 2^{\frac{j}{N}}\sigma_0 & ... & 2^{\frac{N-1}{N}}\sigma_0 & 2^{\frac{N}{N}}\sigma_0 & ... & 2^{\frac{2N}{N}}\sigma_0
\end{array}$$

3.2.3 Laplacian of Gaussian Pyramids

- Instead of storing the Gaussian levels, we can store the difference between the original version of image and the blurred version of image(difference between each layer), the last layer is kept as is(only the last gaussian level image stored).

This forms a Laplacian of Gaussian pyramid:



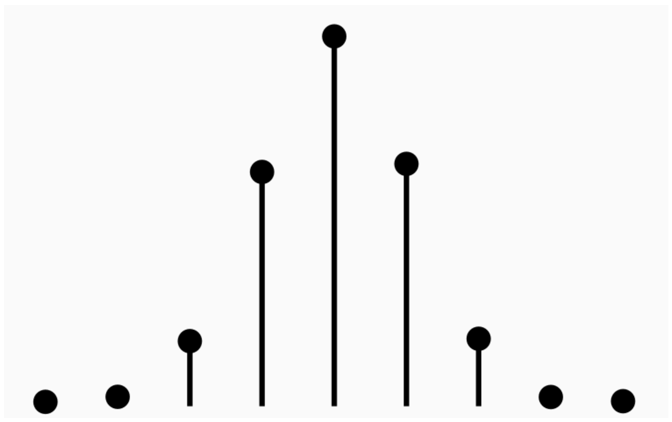
3.3 Pyramid Implementation

3.3.1 The Sampled Gaussian

- Remember the Gaussian filter:

$$G_{\mu,\sigma}(x) = \tfrac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- To make a discrete filter of 2∗K+1 values, we can sample the Gaussian.



For K = 4

3.3.2 Alternatives

- Derive a discrete scale space and convolve with

$$T(n, \sigma^2) = e^{-\sigma^2} I_n(\sigma^2)$$

where I_n denotes the modified Bessel functions of integer order n.
- Use an infinite impulse response (IIR) recursive filter.

4 Keypoint Detection

4.1 Keypoints

- Keypoints are points in the image that satisfy the following

•They are mathematically well-defined
•They have a well-defined position, sometimes also an orientation and scale
•They are detected on the images of the same scene point from different viewpoints (repeatability)
•They are expected to have rich texture around them (more on this later)

- Keypoints are special because no matter how the image changes, whether image rotates, shrinks/expands etc, we should be able to find the same keypoints in the modified image too when comparing with the original.

- Keypoints are scale invariant. Scale invariant means that even if we change the size of the image(scaled up/down), we should able to find those points.

- Keypoint detector algorithm is an algorithm that finds points in the image which has "good features".

- Keypoint descriptor algorithm is an algorithm that encodes properties of that feature like contrast with neighbours etc., so it can be compared with the other keypoints from different images(have different scale/orientation(or both)).

- Keypoint matching algorithm is an algorithm that determines correspondence between descriptors in two views.

4.2 Keypoint Detector Algorithm

4.2.1 Harris Corner Detection

- Harris-Corner Detector is an algorithm to extract corners and inner features of the image.
- A corner is a point whose local neighborhood stands in two dominant and different edge directions. In other words, a corner can be interpreted as the junction of two edges where an edge is a sudden change in intensity values.
- Corners is not the same thing as in the real world. (There is some similarities)

$$E_{x,y} = \sum_{u,v} W_{u,v}[I_{x+u,y+v} - I_{u,v}]^2 = \sum_{u,v} W_{u,v}[I^2_{x+u,y+v} - 2I_{x+u,y+v}I_{u,v} + I^2_{u,v}] \quad \left| \begin{array}{c} u,v \rightarrow \text{Pixel Coordinates} \\ x,y \rightarrow \text{Shifting Amount} \\ W \rightarrow \text{Weight} \end{array} \right.$$

- We usually use weight for rotated images.
- Even if it rotated, with higher weight in center, matching won't be affected by rotating.

$E_{x,y} = \sum\limits_{u,v} W_{u,v}[xX + yY + O(x^2+y^2)]^2$ with $X = \frac{\partial I}{\partial x}$ & $Y = \frac{\partial I}{\partial y}$

$I(u+\Delta x, v+\Delta y) \approx I(u,v) + \Delta x\frac{\partial I}{\partial x} + \Delta y\frac{\partial I}{\partial y} + \text{H.O.T} \Rightarrow [I(u,v) + \Delta x\frac{\partial I}{\partial x} + \Delta y\frac{\partial I}{\partial y} - I(u,v)]^2$

- For small x and y:

$$E(x,y) = \sum_{u,v}[xI_x + yI_y]^2 = \sum_{u,v} W_{u,v}(x^2I_x^2 + 2xyI_xI_y + y^2I_y^2)$$

$$E(x,y) = x^2\underbrace{\sum_{u,v} W_{u,v}I_x^2}_{\overline{I}_x^2} + 2xy\underbrace{\sum_{u,v} W_{u,v}I_xI_y}_{\overline{I}_{xy}} + y^2\underbrace{\sum_{u,v} W_{u,v}I_y^2}_{\overline{I}_y^2} = x^2\overline{I}_x^2 + 2xy\overline{I}_{xy} + y^2\overline{I}_y^2$$

$$Tr(M) = \alpha + \beta = \overline{I}_x^2 + \overline{I}_y^2$$
$$Det(M) = \alpha\beta = \overline{I}_x^2\overline{I}_y^2 - \overline{I}_{xy}^2$$

$$E(x,y) = \begin{bmatrix} x & y \end{bmatrix} \underbrace{\begin{bmatrix} \overline{I}_x^2 & \overline{I}_{xy} \\ \overline{I}_{xy} & \overline{I}_y^2 \end{bmatrix}}_{M} \begin{bmatrix} x \\ y \end{bmatrix}$$

$R = Det(M) - kTr(M)^2$ ($R \Rightarrow$ cornerness response) ($k \approx 0.04$)

$$\Rightarrow \overline{d} = \begin{bmatrix} x \\ y \end{bmatrix} \qquad E(\overline{d}) = \overline{d}^T \underbrace{\begin{bmatrix} \overline{I}_x^2 & \overline{I}_{xy} \\ \overline{I}_{xy} & \overline{I}_y^2 \end{bmatrix}}_{M} \overline{d} = \overline{d}^T M \overline{d}$$

$$M \rightrightarrows \left.\begin{matrix} \lambda_1, \overline{V}_1 \\ \lambda_2, \overline{V}_2 \end{matrix}\right\} \begin{matrix} M\overline{V}_1 = \lambda_1\overline{V}_1, & \overline{V}_1^T\overline{V}_1 = 1 \\ M\overline{V}_2 = \lambda_2\overline{V}_2, & \overline{V}_2^T\overline{V}_2 = 1 \end{matrix} \quad \overline{V}_1^T\overline{V}_2 = 0$$

$$\overline{d} = \begin{bmatrix} c_1 & c_2 \end{bmatrix} \begin{bmatrix} \overline{v}_1 \\ \overline{v}_2 \end{bmatrix} = c_1\overline{v}_1 + c_2\overline{v}_2 \qquad E(\overline{d}) = (c_1\overline{v}_1 + c_2\overline{v}_2)^T M (c_1\overline{v}_1 + c_2\overline{v}_2)$$

$$E(\overline{d}) = (c_1\overline{v}_1^T + c_2\overline{v}_2^T)(c_1M\overline{v}_1 + c_2M\overline{v}_2) = (c_1\overline{v}_1^T + c_2\overline{v}_2^T)(c_1\lambda_1\overline{v}_1 + c_2\lambda_2\overline{v}_2)$$

$$E(\overline{d}) = c_1^2\lambda_1\underbrace{\overline{v}_1^T\overline{v}_1}_{1} + c_1c_2\lambda_2\underbrace{\overline{v}_1^T\overline{v}_2}_{0} + c_1c_2\lambda_1\underbrace{\overline{v}_1\overline{v}_2^T}_{0} + c_2^2\lambda_2\underbrace{\overline{v}_2^T\overline{v}_2}_{1} \quad \{\text{Small displacement}\}$$

- Harris Corner Detection is based on the local auto correlation function of a signal which measures the local changes of the signal with patches shifted by a small amount in different directions.

4.2.2 Moravec Corner Detection

- Moravec algorithm defines interest points as points where there is a large intensity variation in every direction, which is the case at corners, the Moravec algorithm is considered a corner detector.

Algorithm:

1) For each pixel location (x,y) in the image, calculate the intensity variation from a shift u,v as:

$$V_{u,v}(x,y) = \sum_x \sum_y [I(x,y) - I(x+u, y+v)]^2 \quad (u,v) \rightarrow \text{Shift amount}$$
$$(u,v) \text{ is considered as } \{(-1,-1),(-1,0),(-1,1),(0,-1),(0,1),(1,-1),(1,0),(1,1)\}$$

2) Construct the cornerness map by calculating the cornerness measure $C(x,y)$ for each pixel location (x,y)

$$C(x,y) = min(V_{u,v}(x,y))$$

3) Threshold the interest map by setting all $C(x,y)$ below a threshold T to zero.
- All non-zero points remaining in the cornerness map are **corners**.

4.2.3 Features from Accellerated Segment Test – FAST

1) Select a pixel **p** in the image which is to be identified as an interest point or not. Let its intensity be $\mathbf{I_p}$.
2) Select appropriate threshold value **t**.
3) Consider a circle of 16 pixels around the pixel under test.
4) Now the pixel **p** is a corner if there exists a set of **n** contiguous pixels in the circle (of 16 pixels) which are all brighter than $\mathbf{I_p + t}$, or all darker than $\mathbf{I_p - t}$. **n** was chosen to be 12
5) A **high-speed test** was proposed to exclude a large number of non-corners. This test examines only the four pixels at 1, 9, 5 and 13 (First 1 and 9 are tested if they are too brighter or darker. If so, then checks 5 and 13).

→ If **p** is a corner, then at least three of these must all be brighter than $\underline{I_p + t}$ or darker than $\underline{I_p - t}$. If neither of these is the case, then **p** cannot be a corner.

4.2.4 Scale Invariant Feature Transform – SIFT

- Matching features across different images in a common problem in computer vision. When all images are similar in nature (same scale, orientation, etc) simple corner detectors can work. But when you have images of different scales and rotations, you need to use the SIFT algorithm.

→ **Scale-Space Construction**
- Scale space is constructed by the convolution of image using Gaussian kernel.

- In order to detect to blob area, Lowe constructs differences of Gaussian to approximate the Laplace detector.
- Then find the local extreme point among 8 points around in the same scale space and 18 points in the adjacent scale image.
- When we get the extreme point, we also get the location and scale.

→ <u>**Extreme Points Location**</u>
...
→ <u>**Main Direction Computation**</u>
...
→ <u>**Feature Descriptors Construction**</u>
...

4.3 Region Detectors

4.3.1 Maximally Stable Extremal Regions – MSER

- This method is used as a method of <u>blob detection</u> in images.
- Blob is a region of an image in which some properties are constant or approximately constant.
    - All points in a blob can be considered in some sense to be similar to each other.
    - The most common method for blob detection is <u>convolution</u>

5 Keypoint Description and Matching

5.1 Comparing Image Patches

- Image patch is a <u>container</u> of pixels in larger form. For example, let's say we have an image of 100px by 100px. If we divide this image into 10px × 10px patches, then we will gain an image with 100 patches(100px for each patch).
- In image processing <u>patch</u> and <u>window</u> is used interchangeably most of time, but patch is usually used in a context when our algorithm mainly focused on a fact that bunch of pixels share same similar property.

For instance, patch is used in context of sparse representation or image compression, while window is used in edge detection or image enchament.

5.1.1 Sum of Squared Differences (SSD)

- To compare two image patches, they must have same dimensions.
- If we want to compare an image patch R in I to another image patch $R^{'}$ of the same size in $I^{'}$, we can go over the image regions and sum the squared difference of corresponding pixel intensities.

$$\Rightarrow SSD_{RR'} = \sum_{(x,y)\in R,(x^{'},y^{'})\in R^{'}} (I(x,y) - I^{'}(x^{'},y^{'}))^2$$

$\rightarrow SDD_{RR'}$ considered as a metric. A metric should be ≥ 0.
Even we change the positions, the result won't change.

5.1.2 Zero–Mean SSD

- SDD is sensitive to changes in image brightness, so we might want to remove the average intensity from each region before comparing.
- If

$$\Rightarrow \mu_R = \frac{1}{N} \sum_{(x,y)\in R} I(x,y) \quad \Rightarrow \mu_R = \frac{1}{N} \sum_{(x,y)\in R} I(x,y)$$

where N is the number of pixels in R. Then the zero-mean SSD is computed as

$$ZMSSD_{RR'} = \sum_{(x,y)\in R,(x',y')\in R'} [(I(x,y) - \mu_R) - (I'(x',y') - \mu'_R)]^2$$

5.1.3 Normalized Cross–Correlation

- Another way of comparing two regions is the normalized cross–correlation, which is more robust to some more complex lighting changes. If

$$\mu_R = \frac{1}{N} \sum_{(x,y)\in R} (I(x,y))$$

is the mean intensity and

$$\sigma_R^2 = \frac{1}{N-1} \sum_{(x,y)\in R} (I(x,y) - \mu_R)^2$$

is the variance of intensity, then

$$NCC_{RR'} = \sum_{(x,y)\in R,(x',y')\in R'} \frac{(I(x,y)-\mu_R)(I'(x',y')-\mu'_R)}{\sigma_R \sigma'_R}$$

5.1.4 Descriptor Computation

- Both ZM–SSD and NCC are good measures of image similarity if R and R $'$ are related by a small deformation. If the deformation is large, as is the case for general perspective transformations, then they both fail to represent patch similarity.
- For general keypoint matching, we compute keypoint descriptors and compare these descriptors to match keypoints.

5.2 Keypoint Description with Gradients

5.2.1 Scale Invariant Feature Transform (SIFT)

- SIFT is an algorithm for both the computation and description of keypoints.

→ **Compute Keypoints:**
- Compute a Gaussian pyramid
- Compute a DoG pyramid by taking the difference of levels
- Find maxima points both spatially and between scales at each level

→ **Compute Descriptors: For each keypoint,**
- Compute an orientation by averaging gradient orientations
- Compute several histograms by binning gradient orientations
- Compute the descriptor by concataneting the histograms
- Normalize the descriptor to unit length

- In SIFT algorithm, we first create the scale space. To generate scale space, we take the original image generate progressively blurred out images. Then, you resize the original image to half size. And you generate blurred out images again. And you keep repeating.
- Images of the same size (vertical) form an octave

→ Octaves and Scales
- The number of octaves and scale depends on the size of the original image. We will have to decide for yourself how many octaves and scales we want.

→ Blurring
- Mathematically, "blurring" is referred to as the convolution of the gaussian operator and the image.
- When scale paramater doubled, it means we reached the end of the octave and we should down-sample the blurred image.

- In first step of SIFT, we generate several octaves of the original image. Each octave's image is half of the previous one. Within an octave, images are progressively blurred using the Gaussian blur operator.
- To create Difference of Gaussian(DoG) pyramid, we took two consecutive images in an octave and subtracted one from the other. Then the next consecutive pair is taken and the process repeats. This is done for all octaves. The resulting images are an approximation of scale invariant laplace of gaussian.
- To find maxima/minima points, we look for all neighbours of selected pixel. If selected pixel is a "keypoint", then it means it is the greatest or smallest of all neighbours.
- Comparing neighbour pixels includes currents scale space, the above scale space and the below scale space. So 26 pixels are considered as a neigbour.

- Keypoints are not detected in the lowermost and uppermost scale. There simply aren't enough neighbours to do the computation.

5.2.2 Computing Keypoint Orientations

- Compute orientation and magnitude of the image gradients around each keypoint.
- If the gradient **g** is

$$g = \begin{bmatrix} I_x \\ I_y \end{bmatrix}$$

then

$$m = \sqrt{\frac{\partial I}{\partial x} * \frac{\partial I}{\partial x} + \frac{\partial I}{\partial y} * \frac{\partial I}{\partial y}} = \sqrt{I_x * I_x + I_y * I_y}$$
$$\theta = \arctan \frac{I_y}{I_x}$$

- The keypoint orientations are picked as the local maxima of the histogram of orientations.

Orientation Binning

- To compute a descriptor, the region around the keypoint is split into several subregions and for each one an orientation histogram is computed. The descriptor is the concatanetation of these histograms.

Normalization

- The descriptor is first normalized to unit length

$$d \leftarrow \frac{d}{|d|}$$

then any dimension that is larger than 0.2 is set to 0.2 and the descriptor is normalized to unit length once again.
- The last step is to increase robustness to non–linear intensity changes.
- If angle of gradient is between $0° - 45°$ or smt. like that, we use interpolation.

5.3 Keypoint Matching

5.3.1 Nearest Neighbor from Euclidean Distance

- Given a set of keypoints $k_1, ..., k_N$ in image I and another set of keypoints $k_1^{'}, ..., k_M^{'}$ in image $I^{'}$, we can compute the corresponding sets of descriptors $\mathbf{d_1}, ..., \mathbf{d_N}$ and $\mathbf{d_1^{'}}, ..., \mathbf{d_M^{'}}$. To match k_i to the keypoints in $I^{'}$, we first find the keypoint whose descriptor is closest to $\mathbf{d_i}$ as measured by the Euclidean distance:

$$k_i \longleftrightarrow k_{j^*}^{'} \text{ if and only if } j^* = \underset{j}{argmin} |\mathbf{d}_i - \mathbf{d_j^{'}}|_2$$

- Minimum doesn't have to be same.

5.3.2 Distance Ratio Test

- If the nearest and the second nearest neighbors are too close to each other, we can label the correspondence as ambigous and discard it. If the nearest neighbor distance is d_1 and the second nearest neighbor distance is d_2 , we can threshold their ratio

$$\text{Accept when } r_{12} = \frac{d_1}{d_2} > \tau$$

5.3.3 Cosine Similarity

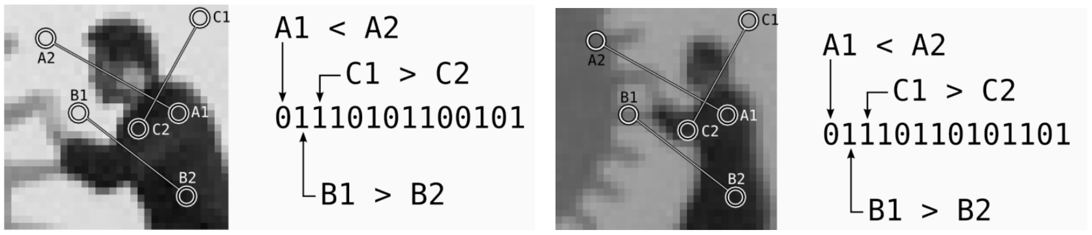
- We can also measure distances by the angle between two descriptor vectors. Since descriptors are usually normalized to unit length

$$\theta = \arccos \mathbf{d}_i^T \mathbf{d}'_{\mathbf{j}}$$

5.4 Binary Keypoint Description and Matching

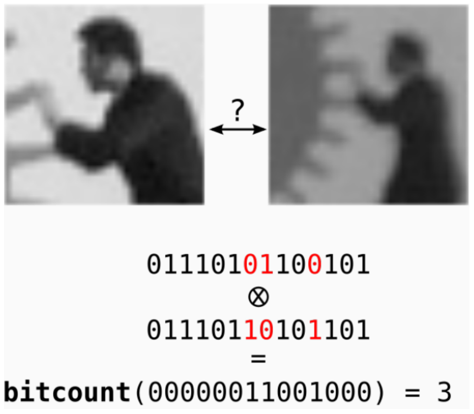
5.4.1 BRIEF

- Instead of using gradients we can simply compare average intensity around different pixel locations. Each comparison yields a binary result stored in a single bit. This leads to a binary descriptor. Surprisingly we can pick the locations to compare completely at random



5.4.2 Hamming Distance

- We can not use the Euclidean distance or the cosine similarity to compare binary descriptors. Instead we use the Hamming distance, which is simply the number of different bits.



5.5 Oriented BRIEF - ORB

- When the orientation involves, we use ORB instead of BRIEF.
- BRIEF is not robust to orientation changes (rotations on the image plane). OpenCV team has developed another descriptor ORB that uses the orientation from the keypoint detector to rotate the tests. This is much more robust if you care about orientation changes, such as matching to upside down images.

6 Planar Projective Geometry

6.1 Cameras Acting on a Plane

- Cameras preserve **connectivity** and **collinearity**.

Connectivity
- Pixels in an image relate to their neighbours

Collinearity
- In geometry, collinearity of a set of points is the property of their lying on a single line.
- If points on a single line in reality, they are on a single line in image too.

- Collinearity and connectivity is preserved until there are some issues due to the lens(es).

6.1.1 Transformations and Invariants

- The action of a camera on the points of a plane is not arbitrary, it preserves some qualities such as connectivity and collinearity.
- These are called the invariants of the transformation and we can understand a transformation by studying its invariants.

6.2 Euclidean and Similarity Transformations of 2D

6.2.1 Translation

- The simplest we can do is to translate the camera while keeping it perpendicular to the plane normal.

$$T(\overline{x}) = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad T^{-1} = \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Preserve	Don't Preserve
- Connectivity	- Position
- Collinearity	
- Distance	
- Angle between lines	
- Angle between the lines and the coordinate axes	
- Area	

6.2.2 Rotation

- We can rotate the camera while keeping it perpendicular to the plane normal.
- Counter–clockwise rotation by θ

$$R(x,\theta) = R_\theta x = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$R^{-1}(x,\theta) = R_{-\theta} x = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Preserve	Don't Preserve
- Distance	- Position
- Angle between lines	- Angle between the lines and coordinate axes.
- Area	

$R_\theta^{-1} = R_\theta^T$

6.2.3 Euclidean/Rigid–Body transformation

- We can rotate and translate at the same time

$$\varepsilon(x,\theta) = R_\theta + t = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

- First rotation and after translation.
- It is not the same thing as first translate and after rotate.

- $R_\theta x + t \neq R_\theta(x+t) \Rightarrow R_\theta(x+t) = R_\theta x + R_\theta t$

6.2.4 Uniform Scaling

- We can $\underline{\text{approach to}}$ or $\underline{\text{move away from}}$ the plane.
- Uniform scaling by λ on points

$$S(x,\lambda) = S_\lambda x = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad S^{-1}(x,\lambda) = S_{\frac{1}{\lambda}} = \begin{bmatrix} \frac{1}{\lambda} & 0 \\ 0 & \frac{1}{\lambda} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- If height scaled by 2, then width scaled by 2 too.

6.2.5 Similarity Transformations

- We can combine Euclidean transformations with uniform scaling

$$S(x) = \lambda R_\theta x + t = \begin{bmatrix} \lambda cos\theta & -\lambda sin\theta \\ \lambda sin\theta & \lambda cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Preserve	Don't Preserve
- Distance Ratio	- Distance
- Angle between lines	- Area
- Area ratio	- Position
	- Angle between lines and coordinate axes

6.3 Homogeneous Coordinates

6.3.1 Homogeneous Representation for Points

- Transformations of 2D points cannot be represented by a single matrix in the Cartesian Coordinates(Euclidean Space).
- In order to represent transformations by a single matrix, we add a third dimension as a free scaling parameter to the point representation.

$$x = \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{array}{ccc} \underline{\text{Homogeneous Coordinate}} & & \underline{\text{Cartesian Coordinate}} \\ x \equiv \begin{bmatrix} x & y & w \end{bmatrix}^T & \underset{w \neq 0}{\Longleftrightarrow} & x \equiv \begin{bmatrix} \frac{x}{w} & \frac{y}{w} \end{bmatrix}^T \end{array}$$

6.3.2 Homogeneous Representation of Similarity Transformations

$$S(x) = \begin{bmatrix} \lambda R_\theta & \bar{t} \\ 0_2^T & 1 \end{bmatrix}_{3\times3} \bar{x} \equiv \begin{bmatrix} \lambda cos\theta & -\lambda sin\theta & t_x \\ \lambda sin\theta & \lambda cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \equiv \begin{bmatrix} (\lambda R_\theta)_{2\times2} & \bar{t}_{2\times1} \\ (\bar{0}_2^T)_{1\times2} & 1 \end{bmatrix}_{3\times3} \begin{bmatrix} \tilde{x} \\ w \end{bmatrix}_{3\times1} \equiv \begin{bmatrix} \lambda R\theta\tilde{x} + \bar{t}w \\ w \end{bmatrix}_{3\times1}$$

- $\left[\begin{array}{c|c} \text{A} & \text{B} \\ \hline \text{C} & \text{D} \end{array}\right] \begin{bmatrix} \bar{a}_1 \\ \bar{a}_2 \end{bmatrix} = \begin{bmatrix} A\bar{a}_1 + B\bar{a}_2 \\ C\bar{a}_1 + D\bar{a}_2 \end{bmatrix}$

6.3.3 Homogeneous Representation for Lines

- The line equation in 2D

$$\left. ax + by + c \equiv 0 \equiv \begin{bmatrix} a \\ b \\ c \end{bmatrix}^T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv \mathbf{l}^T\mathbf{x} \right\} \text{Homogeneous equation}$$

- Homogeneous means multiplying it with a scale factor s does not change the equation
- If point x is on the line l, then $l^T x \equiv 0$

6.3.4 Point Line Duality

- The line passes through two points $\mathbf{x}_1 \text{and} \mathbf{x}_2$

$$l_{12} \equiv x_1 \underset{\textbf{(cross product)}}{\times} x_2 \equiv \begin{bmatrix} x_1 \\ y_1 \\ w_1 \end{bmatrix} \times \begin{bmatrix} x_2 \\ y_2 \\ w_2 \end{bmatrix} \equiv \begin{bmatrix} y_1 * w_2 - y_2 * w_1 \\ w_1 * x_2 - w_2 * x_1 \\ x_1 * y_2 - y_1 * x_2 \end{bmatrix}$$

- Exchanging the roles of points and lines, we can find the intersection point of two lines $\mathbf{l}_1$ and $\mathbf{l}_2$ as

$$\left. x_{12} = l_1 \underset{\textbf{(cross product)}}{\times} l_2 \right\} \text{Same operation as above}$$

Duality: Points and lines in 2D have the similar representations in homogeneous coordinates and can be exchanged with each other in formulas.

6.3.5 Vectors and Points at Infinity

- If the free scaling parameter is zero, then it means that the point is at infinity.
- Intersection point of the lines $[1, -1, 1]^T$ and $[1, -1, -1]^T$ is $[2, 2, 0]^T$

6.3.6 Effect of Translation on Vectors

- Vectors are unaffected by translations

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \equiv \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

6.3.7 Effect of Uniform Scaling on Vectors

- Vector directions are unaffected by uniform scaling

$$\begin{bmatrix} \lambda & 0 & t_x \\ 0 & \lambda & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \equiv \begin{bmatrix} \lambda x \\ \lambda y \\ 0 \end{bmatrix} \equiv \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

6.3.8 Effect of Rotations on Vectors

- Vectors are rotated by rotations

$$\begin{bmatrix} R_\theta & 0_2 \\ 0_2^T & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \equiv \begin{bmatrix} R_\theta \hat{x} \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \equiv \begin{bmatrix} xcos\theta - ysin\theta \\ xsin\theta + ycos\theta \\ 0 \end{bmatrix}$$

6.4 Affine and Projective Transformations of 2D

6.4.1 Nonuniform Scaling

- We can scale image using two different scale factors for x and y points

$$\Lambda(\mathbf{x}, \lambda_1, \lambda_2) \equiv S_{\lambda_1,\lambda_2}\mathbf{x} = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \equiv \begin{bmatrix} \lambda_1 x \\ \lambda_2 y \\ w \end{bmatrix}$$

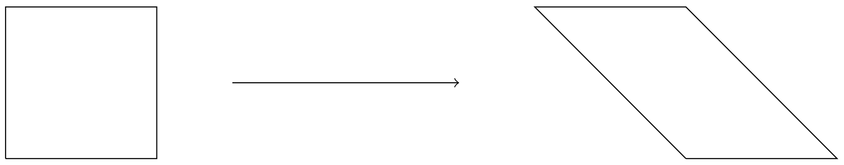
<u>**NOTE**</u>: If $w = 0$, then direction of vector will change.

6.4.2 Shearing

- Shearing is a modification that the scale of x or y is affected by other one.

$$Q_x(\mathbf{x}, a) \equiv \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \equiv \begin{bmatrix} x + ay \\ y \\ w \end{bmatrix}$$

$$Q_x(\mathbf{x}, a) \equiv \begin{bmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \equiv \begin{bmatrix} x \\ y + ax \\ w \end{bmatrix}$$



6.4.3 Affine Transformations of 2D

- Affine transformation is a transformation that combines all transformation into $2x2$ invertible matrix A.

$$A(x) \equiv \begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}_2^T & 1 \end{bmatrix} \mathbf{x} \equiv \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

<u>Preserve</u>	<u>Don't Preserve</u>
- Distance ratio between points on a line	- Distance
- Parallelism	- Area
- Area Ratio	- Position
	- Angle between lines

6.4.4 The Line at Infinity

- Consider a point $\overline{x}$ is a point at infinity, then it means that $l_\infty^T \overline{x} \equiv 0$

$$\overline{x} \equiv \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \quad l_\infty \equiv \begin{bmatrix} a \\ b \\ c \end{bmatrix} \Rightarrow [a, b, c] \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \equiv \underbrace{a}_{0} x + \underbrace{b}_{0} y + \underbrace{c}_{1} 0 \equiv 0 \} \text{Value of c is not important}$$

- All points at infinity line on a line $\mathbf{l}_\infty = [0, 0, 1]^T$
- If points are transformed by $\mathbf{A}$, then lines are transformed by $\mathbf{A^{-T}}$

$$l^T x \equiv 0 \rightarrow (Kl)^T Ax \equiv 0 \rightarrow l^T \underbrace{K^T A}_{I} x \equiv 0$$

$$K^T A = I \rightarrow K^T = A^{-1} \rightarrow (K^T)^T = (A^{-1})^T \rightarrow K = A^{-T}$$

-Affine transformations change $\mathbf{l}_\infty$ to

$$\mathbf{A}^{-T} \mathbf{l}_\infty \equiv \begin{bmatrix} A^{-T} & \mathbf{0}_2 \\ -\mathbf{t}^T A^{-T} & 1 \end{bmatrix} \mathbf{l}_\infty \equiv \mathbf{l}_\infty$$

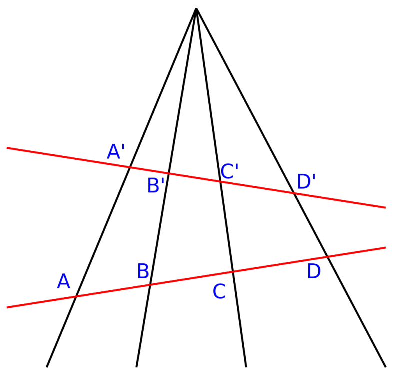
6.4.5 Projective Transformations of 2D

- If we change the zero entries in the last row of affinity matrix, points at infinity may move to finite points, and parallel lines might intersect in the image plane.
- We can write

$$\mathbf{P}(\mathbf{x}) \equiv H\mathbf{x} \equiv s \begin{bmatrix} h_1 & h_4 & h_7 \\ h_2 & h_5 & h_8 \\ h_3 & h_6 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \equiv$$

$$\begin{bmatrix} sh_1 & sh_4 & sh_7 \\ sh_2 & sh_5 & sh_8 \\ sh_3 & sh_6 & s \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \Bigg\}$$ This is called a **perspective transformation** of 2D, the transformation matrix H is called a **homography**.

- Perspective transformations preserve the cross-ratio
- Perspective do not preserve angles, parallelism, areas or their ratios



- Given four points A,B,C,and D on a line, their cross-ratio is defined as; $(A, B; C, D) = \frac{AC.BD}{BC.AD}$
- Points A,B,C,D and A′, B′, C′, D′ are related by a projective transformation so their cross ratios, (A, B; C, D) and (A′, B′; C′, D′) are equal

6.5 The Direct Linear Transformation (DLT) Algorithm

6.5.1 Point Correspondences

- Points on two images of a planar object are related to each other by: $x^{'} \equiv Hx$
- Since the vectors on both sides are parallel to each other, we can write: $x^{'} \times Hx = 0$
- If two vectors are parallel, then angle between vectors is 0°or 180° : $\overline{v}_1 \times \overline{v}_2 = \|\overline{v}_1\|\|\overline{v}_2\| sin\theta = 0$

$$x^{'} \times Hx = 0$$

$$\begin{bmatrix} x^{'} \\ y^{'} \\ 1 \end{bmatrix} \times \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

- This gives us two independent equations in the entries of H

$$\begin{bmatrix} 0 & 0 & 0 & -x & -y & -1 & y^{'}x & y^{'}y & y^{'} \\ x & y & 1 & 0 & 0 & 0 & -x^{'}x & -x^{'}y & -x^{'} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = \overline{0}$$

- The equation above is for one point. We can concatenate equations from 4 different points.
- We have a linear system of equations

$$\mathrm{A}\mathbf{h} = 0$$

- Assuming that A's columns are independent, it has a one dimensional null-space and we can find a unique solution by fixing $\|\mathbf{h}\| = 1$.
- The solution is given by the right eigenvector of A corresponding to the smallest singular value of A
- It is computed by the Singular Value Decomposition (SVD) of A
- In SVD, A is written as USV^T

6.5.2 Normalization

- DLT algorithm is used to compute the homography, but numerically it is sensitive to the scaling of the components $\underline{x_i}$.
- Because of that, we first normalize point coordinates by:

$$\left.\begin{array}{c}\text{- Moving their center to (0,0)}\\ \text{- Uniformly scale coordinates, so that}\\ \text{average distance to (0,0) is } \sqrt{2}\end{array}\right\}\ \begin{array}{l}\text{The purpose of setting average distance to}\\ \sqrt{2} \text{ is to obtain points like or near to (1,1)}\end{array}$$

- Once we compute H, we need to transform it back, so that it transforms points in the original space.
- If

$$\left.\begin{array}{c} x^{'} \equiv HX \\ \hat{x} = T_1 x \text{ and } \hat{x}^{'} = T_2 x^{'} \\ \hat{x}^{'} = \hat{H}\hat{x} \end{array}\right\}\ \begin{array}{c} 1)\hat{x}^{'} = \hat{H}\hat{x} \rightarrow T_2 x^{'} = \hat{H} T_1 x \rightarrow x^{'} = T_2^{-}\hat{H}T_1 x \\ 2) x^{'} \equiv Hx \\ \text{- If we combine these equations, we obtain } \underline{H = T_2^{-}\hat{H}T_1} \end{array}$$

- Then

$$H = T_2^{-1}\hat{H}T_1$$

- The relation between points on two images of same planar object can be written as;

$$\left. x^{'} \equiv Hx \right\}\ \begin{array}{c} x^{'} \rightarrow \text{Points on image 2} \\ H \rightarrow \text{Homography matrix} \\ x \rightarrow \text{Points on image 1} \end{array}$$

$$\left. x^{'} \equiv Hx \right\} \text{ We use } \equiv \text{ symbol because there is a } \lambda(\text{scale}) \text{ factor.}$$

$$x^{'} = \lambda H x$$

- We can write $x^{'} \equiv Hx$ as

$$\begin{bmatrix} x^{'} \\ y^{'} \\ 1 \end{bmatrix} \equiv \lambda \begin{bmatrix} h_1x + h_2y + h_3 \\ h_4x + h_5y + h_6 \\ h_7x + h_8y + h_9 \end{bmatrix} \xrightarrow[\text{Cartesian}]{\text{Convert to}} \begin{bmatrix} \frac{x^{'}}{1} \\ \\ \frac{y^{'}}{1} \end{bmatrix}_{2\times1} = \begin{bmatrix} \frac{\lambda(h_1x+h_2y+h_3)}{\lambda(h_7x+h_8y+h_9)} \\ \\ \frac{\lambda(h_4x+h_5y+h_6)}{\lambda(h_7x+h_8y+h_9)} \end{bmatrix}_{2\times1} \Rightarrow \begin{array}{l} x^{'} = \frac{h_1x+h_2y+h_3}{h_7x+h_8y+h_9} \\ \\ y^{'} = \frac{h_4x+h_5y+h_6}{h_7x+h_8y+h_9} \end{array}$$

$$\bullet x^{'}xh_7 + x^{'}yh_8 + x^{'}h_9 - xh_1 - yh_2 - h_3 = 0$$
$$\bullet y^{'}xh_7 + y^{'}yh_8 + y^{'}h_9 - xh_4 - yh_5 - h_6 = 0$$

$$\begin{bmatrix} -x & -y & -1 & 0 & 0 & 0 & x^{'}x & x^{'}y & x^{'} \\ 0 & 0 & 0 & -x & -y & -1 & y^{'}x & y^{'}y & y^{'} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

7 Optimization Techniques

7.1 Parameter Estimation

7.1.1 Parametric Modelling

- In computer vision, we usually have a model of the process that generated the image, i.e, a 2D projective transformation of a planar object.

$$\mathbf{x}^{'} \equiv H\mathbf{x}$$

- This model is usually parametrized by a number of unknowns. In the case of a homography, we have nine parameters that are defined up to a scale factor.

$$H \equiv \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

- Most of the cases, we cannot find a model that fit points with 0 error. Because of that, we are trying to minimize the error.
- Consider three points $(x_1, y_1), (x_2, y_2), (x_3, y_3)$

$$\left.\begin{array}{l} y_1 = mx_1 + b \\ y_2 = mx_2 + b \\ y_3 = mx_3 + b \end{array}\right\} \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \Rightarrow A\overline{p} = y \ \ \& \ \ \overline{p}^T A^T = \overline{y}^T$$

$$\begin{aligned} \|\overline{e}\|^2 = \|\overline{y} - A\overline{p}\|^2 &= (\overline{y} - A\overline{p})^T(\overline{y} - A\overline{p}) \\ = (\overline{y}^T - \overline{p}^T A^T)(\overline{y} - A\overline{p}) &= \overline{y}^T\overline{y} - \overline{y}^T \underbrace{A\overline{p}}_{y} - \underbrace{\overline{p}^T A^T}_{\overline{y}^T} \overline{y} + \overline{p}^T A^T A\overline{p} \\ &= \overline{y}^T\overline{y} - 2\overline{p}^T A^T\overline{y} + \overline{p}^T A^T A\overline{p} \end{aligned}$$

- To minimize error, we calculate $\frac{\partial\|\overline{e}\|^2}{\partial p} = 0$
$\rightarrow \frac{\partial\|\overline{e}\|^2}{\partial p} = 0 = 2A^TA\overline{p} - 2A^T\overline{y} \Rightarrow A^TA\overline{p} = A^T\overline{y}$

- If we solve the equation above, we will find the model that cause minimum error.
- y=mx+b is not a good way to modeling lines. It cannot be used for perpendicular lines(slope may be infinite)
- ax + by + c = 0 is a good way to modeling lines.

$$\left.\begin{array}{l} ax_1 + by_1 + c = 0 \\ ax_2 + by_2 + c = 0 \\ ax_3 + by_3 + c = 0 \end{array}\right\} \begin{bmatrix} x_1 & y_1 & c \\ x_2 & y_2 & c \\ x_2 & y_3 & c \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Rightarrow A\overline{p} = 0$$

$$-min\|A\overline{p}\|^2 \text{ such that } \overline{p}^T\overline{p} = 1 \Rightarrow \overline{p}^T\overline{p} - 1 = 0$$
$$-\ min\|A\overline{p}\|^2 - \lambda(p^Tp - 1)$$

$$\frac{\partial(\overline{p}^T A^T A\overline{p} - \lambda(\overline{p}^T\overline{p} - 1))}{\partial\overline{p}} = 0 \left.\right\} \begin{array}{c} 2A^TA\overline{p} - 2\lambda\overline{p} = 0 \\ A^TA\overline{p} = \lambda\overline{p} \\ \overline{p} \text{ is eigenvector of } A^TA \end{array}$$

7.1.2 Fitting to the Data

- In practice, we usually obtain some data points that is generated by the parametric model that we have.
- For the homography example, we obtain point correspondences $X_i = (\mathbf{x}_i, \mathbf{x}'_i)$, for $i = 1, ..., N$. Then we estimate the model parameters that best explain these data. To estimate the model, we

•Pick a mathematical criterion that defines a measure of goodness (best in what sense?).
•Find a method to estimate parameter values that is the best for the selected measure of goodness (an optimization algorithm).
•Implement the algorithm to satisfy given constraints on numerical precision, speed, memory requirements, and similar.

7.2 Minimizing Algebraic Distance

7.2.1 Algebraic Minimization

- Often it is possible to take the model equations and turn them into an algebraic criterion that needs to be minimized.
- For the case of the homography, this algebraic criterion can be given by

$$\mathbf{x}'_i \times H\mathbf{x}_i = 0$$

7.2.2 Normalization

$$\mathbf{x}'_i \times H\mathbf{x}_i = 0$$

- The trivial solution is to set all h_i to zero.
- Since that is not what we want and there is a free scale parameter, we need to normalize the nine parameters in some way.
We will consider two normalization schemes:
• Set $h_9 = 1$
• Set $\sum_{j=1}^{9} h_j^2 = 1$

7.2.3 Linear Least Squares

$$\mathbf{x}'_i \times H\mathbf{x}_i = 0$$

We can set $h_9 = 1$ and this leaves us eight parameters to stack in a vector $\mathbf{h} = [h_1, ..., h_8]^T$
- For each data point i this gives two equations

$$\begin{bmatrix} 0 & 0 & 0 & -x_i & -y_i & -1 & x_i y_i' & y_i y_i' \\ x_i & y_i & 1 & 0 & 0 & 0 & -x_i x_i' & -y_i x_i' \end{bmatrix} \mathbf{h} = \begin{bmatrix} -y_i' \\ x_i' \end{bmatrix}$$

- Stacking all the equations for all i into a matrix A gives

$$A\mathbf{h} = \mathbf{b}$$

Usually the data points are noisy, so in general it is not possible to find a unique $\mathbf{h}$ that simultaneously satisfies $A\mathbf{h} = \mathbf{b}$ for all i when $N > 4$.
- Instead we can minimize the sum of the squared error terms

$$\mathbf{h}^* = \underset{\mathbf{h}}{argmin} \|\mathbf{e}\|^2 = \|A\mathbf{h} - \mathbf{b}\|^2 = (A\mathbf{h} - \mathbf{b})^T (A\mathbf{h} - \mathbf{b})$$

- Solution is found by taking the derivative w.r.t. $\mathbf{h}$ and setting it to zero as the solution of

$$(A^T A)\mathbf{h}^* = A^T \mathbf{b}$$

7.2.4 Weighted Linear Least Squares

- In case we know the characteristics of the noise in the data, we can weigh the errors for data points differently by minimizing

$$\mathbf{h}^* = \underset{\mathbf{h}}{argmin} \|W\mathbf{e}\|^2$$

where W is a matrix of weights. In this case, $\mathbf{h}^*$ is the solution to

$$(A^T W^T W A)\mathbf{h}^* = A^T W^T W \mathbf{b}$$

- In case of zero mean Gaussian noise with covariance matrix V, we should take $W^T W = V^{-1}$ and solve

$$(A^T V^{-1} A)\mathbf{h}^* = A^T V^{-1} \mathbf{b}$$

7.2.5 SVD Solution

$$\mathbf{x}'_i \times H\mathbf{x}_i = 0$$

- We can instead set $\sum_{j=1}^{9} h_j^2 = 1$ and this gives us nine parameters to stack in a vector $\mathbf{h} = [h_1, ..., h_9]^T$ and a constraint $\mathbf{h}^T\mathbf{h} = 1$
- This gives the Direct Linear Transform algorithm that we have covered before with

$$A\mathbf{h} = \mathbf{0} \text{ and } \mathbf{h}^T\mathbf{h} = 1$$

- The solution to this is given by the minimization problem

$$\mathbf{h}^* = \underset{\mathbf{h}}{argmin} \|A\mathbf{h}\|^2 \text{ subject to } \mathbf{h}^T\mathbf{h} = 1$$

- The best $\mathbf{h}^*$ is the eigenvector of A^TA corresponding to the smallest eigenvalue which can be computed by the Singular Value Decomposition of A.

7.3 Minimizing Geometric Distance

7.3.1 Geometric Minimization

- Usually algebraic formulation has a closed form solution that is somewhat easier to compute.
- Unfortunately, it does not always yield the best possible results since algebraic error is usually <u>not invariant</u>(affected) under Euclidean transformations.
- If we move and rotate the data points the algebraic error does not stay constant.
- Instead we can formulate the minimization problem in terms of geometric distances which are constant under Euclidean transformations

$$\mathbf{h}^* = \underset{\mathbf{h}, \hat{\mathbf{x}}_i, \hat{\mathbf{x}}'_i}{argmin} \sum_i d(\mathbf{x}_i, \hat{\mathbf{x}}_i)^2 + d(\mathbf{x}'_i, \hat{\mathbf{x}}'_i)^2 \text{ subject to } \hat{\mathbf{x}}'_i \equiv H\hat{\mathbf{x}}_i$$

where d($\mathbf{x}$, $\mathbf{y}$) is the Euclidean distance between the points $\mathbf{x}$ and $\mathbf{y}$
- The bottom image pair corresponds to the optimization problem

$$\mathbf{h}^* = \underset{\mathbf{h}, \hat{\mathbf{x}}_i, \hat{\mathbf{x}}'_i}{argmin} \sum_i d(\mathbf{x}_i, \hat{\mathbf{x}}_i)^2 + d(\mathbf{x}'_i, \hat{\mathbf{x}}'_i)^2 \text{ subject to } \hat{\mathbf{x}}'_i \equiv H\hat{\mathbf{x}}_i$$

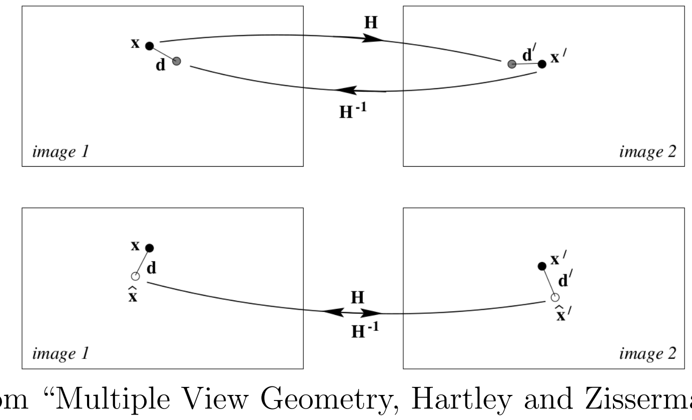


Figure from “Multiple View Geometry, Hartley and Zisserman, 1st Ed.

- A comparison between symmetric transfer error (upper) and reprojection error (lower) when estimating a homography.
- The points $\mathbf{x}$ and $\mathbf{x}^{'}$ are the measured (noisy) points.
- Under the estimated homography the points $\mathbf{x}$ and $H\mathbf{x}^{'}$ do not correspond perfectly.(and neither do the points x and $H^{-1}\mathbf{x}^{'}$)
- However, the estimated points, $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}^{'}$ do correspond perfectly by the homography $\hat{\mathbf{x}}^{'} = H\hat{\mathbf{x}}$
- Using the notation $d(\mathbf{x}, \mathbf{y})$ for the Euclidean image distance between $\mathbf{x}$ and $\mathbf{y}$, the symmetric transfer error is $d(\mathbf{x}, H^{-1}\mathbf{x}^{'})^2 + d(\mathbf{x}^{'}, H\mathbf{x})^2$; the reprojection error is $d(\mathbf{x}, \hat{\mathbf{x}})^2 + d(\mathbf{x}^{'}, \hat{\mathbf{x}}^{'})^2$

7.3.2 Nonlinear Minimization

- Unfortunately, the geometric minimization problem is non–linear and it does not have a closed form solution.
- The solution is found by iterative minimization. There are multiple algorithms for doing that, each with its own advantages and disadvantages.
- The general framework is the following minimization problem

$$\theta^* = \underset{\theta}{argmin} E(\theta)$$

- A solution is found by iteration $\theta_k = \theta_{k-1} - \Delta\theta$ until $E(\theta) < \tau_E$ or $k > \tau_k$. Different algorithms have different approaches for picking $\Delta\theta . \theta$ is usually obtained by closed form algebraic minimization.

7.3.3 Steepest Descent

- An algorithm for finding the nearest local minimum of a function which presupposes that the gradient of the function can be computed.
- Also called the gradient descent method.
- Starts at a point P_0 and, as many times as needed, moves from P_i to P_{i+1}, by minimizing along the line extending from P_i
- Steepest descent is a simple approach that picks $\Delta\theta$ in the same direction as the negative of the gradient of $E(\theta)$ and then moves a certain amount in that direction at each iteration

$$\theta_k = \theta_{k-1} - \lambda_k \frac{\partial E}{\partial \theta}$$

- Steepest descent is easy to understand and implement but in certain problems it is very slow to converge towards a local minimum of E.

7.3.4 Gauss-Newton

- If $E(\theta)$ is in the form of sum of squared error terms, we can use a more advanced technique called Gauss-Newton that converges faster

$$E(\theta) = \sum_i e_i(\theta)^2 \text{ and } \theta_k = \theta_{k-1} - \Delta\theta$$

- In this case the negative step size $\Delta\theta_k$ is the solution to

$$J^T J \Delta\theta_k = J^T e(\theta_{k-1})$$

where $e(\theta_{k-1})$ is the vector of errors from using θ_{k-1}, and $J = \frac{\partial e}{\partial \theta}$ is the Jacobian matrix of error with respect to θ evaluated at time $k-1$ with $J_{ij} = \frac{de_i}{d\theta_j}$

7.3.5 Levenberg-Marquardt

- The minimization of

$$E(\theta) = \sum_i e_i(\theta)^2$$

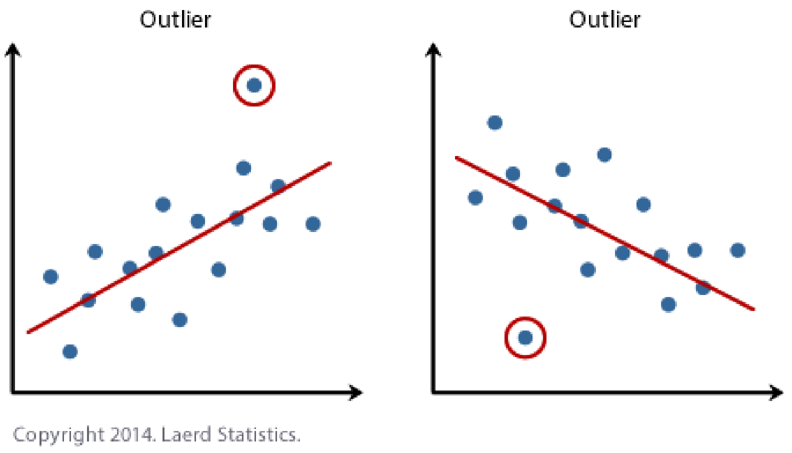
can also be performed with a more stable and faster algorithm using an adaptive step size called the Levenberg-Marquardt algorithm. The negative step size is found by solving

$$\Big(J^T J + \lambda_k \mathbf{diag}(J^T J)\Big)\Delta\theta_k = J^T e(\theta_{k-1})$$

where at each iteration λ is decreased when the reduction in E is large (getting closer to Gauss-Newton) and λ is increased when the reduction in E is small (getting closer to steepest-descent).

7.4 Line Fitting Problem

$$\overline{y} = mx + b + \epsilon, \epsilon \sim N(0, \sigma^2)$$
$$y_i = mx_i + b$$



- Outlier: An observation that deviates too much from other observations. A value that is far from the general distribution of the observed values.
- Inlier: An observation that is explained by underlying probability density function. A value that lies within te general population of the observed value closed to the mean.
- Gross Error: These errors are mistakes that make the measurement very far off the known/accepted value.

7.5 Random Sample Consensus (RANSAC)

- **Random sample consensus (RANSAC)** is an <u>iterative method</u> to estimate parameters of a mathematical model from a set of observed data that contains outliers.
- The RANSAC algorithm works by identifying the outliers in a data set and estimating the desired model using data that <u>does not contain outliers</u>.
- RANSAC is accomplished with the following steps
    1) Randomly selecting a subset of the data set
    2) Fitting a model to the selected subset
    3) Determining the number of outliers
    4) Repeating steps 1-3 for a prescribed number of iterations

- We pick a <u>minimal subset</u> at random and check support from the other data points.
- In line equation: $\overline{y} = mx + b + \epsilon \rightarrow$ 2 points are enough to fit a line $(x_1, y_1), (x_2, y_2)$
- We select a line equation (choosing some coefficients) and assume it is best model. This is hypothesis phase.

p+1	**p+2**	
inlier	inlier	
inlier	outlier	$\longrightarrow$ Does not matter which one is outlier
outlier	outlier	

- For homography, we use 4 points correspondences.

$$\begin{array}{ccc|ccc} x_0 & \leftrightarrow & x_0' & \tilde{\overline{x}}_i' & = & H\overline{x}_i \Rightarrow \|\overline{x}_i' - \tilde{\overline{x}}_i'\| < t \\ x_1 & \leftrightarrow & x_1' & & & \updownarrow \\ \vdots & & \vdots & & & \overline{x}_i' \\ x_{n-1} & \leftrightarrow & x_{n-1}' & & & \end{array}$$

$\tau \rightarrow$ threshold value(application dependent)

$\rightarrow$ Assume there are 1000 data points. What is the minimum number of iterations to find best model?

- There is no right answer. Because answer is dependent to application.
- If all of the data points are inlier, then 1 iteration is enough.
- If only two of them are inlier, then RANSAC may never find the correct solution.

- # of inliers and # of outliers are important for RANSAC
- We don't know how many outliers/inliers exist.

$$\begin{array}{ccc} H^1 & \rightarrow & 2 \\ H^2 & \rightarrow & 5 \\ H^3 & \rightarrow & 2 \\ \vdots & & \\ H^* & \rightarrow & 50 \\ \vdots & & \qquad \Rightarrow \quad H^* = H^k \\ H^{max} & & \end{array}$$

- Supporting inliers' number is determined by the threshold value.
- Fitting to a line is easier then fitting to a homography
- In order to fit points to a line, we shouldn't choose two points which are near. Due to noises it may left to many points as an outlier.
- We should choose two points that far from each other. With that we can scan more area, and left less outlier.

The line may not be the best estimation. If this is the case, we should include all inliers to estimate a better line.
- We should contiune this process until finding the best estimation.
- When we find an estimation for homography H^*, refine it with all inliers(Least Squares(DLT))
- Guided Matching -With More Correspondences-

– $H^{**} \rightarrow$ Better than H^*

7.6 Progressive RANSAC (PROSAC)

- In progressive sampling consensus, input data is assumed to have a quality on it.
- Quality of input data can be exposed in our sampling strategy by smartly sampling the high quality data points first, then progressively sampling the rest of the data set.
- If the distance in descriptors is small, then they are inlier more input data can be exposed in our sampling strategy by smartly sampling the high quality data points first, then progressively sampling the rest of the data set.
- If the distance in descriptors is small, then they are inlier more likely.

Formula

•N: # of correspondences
•P: Probability of success(picking one minimal subset of inliers)
• $\underset{(ratio of inlier)}{r} : \frac{\# of inliers}{N}$
•K: # of iterations
•M: Size of minimal subset

$$\bullet \text{P} = 1 - (1 - r^M)^K$$
$$\bullet \text{K} = [\frac{log(1-p)}{log_1 - r^M}]_{\text{rounds closest integer}}$$

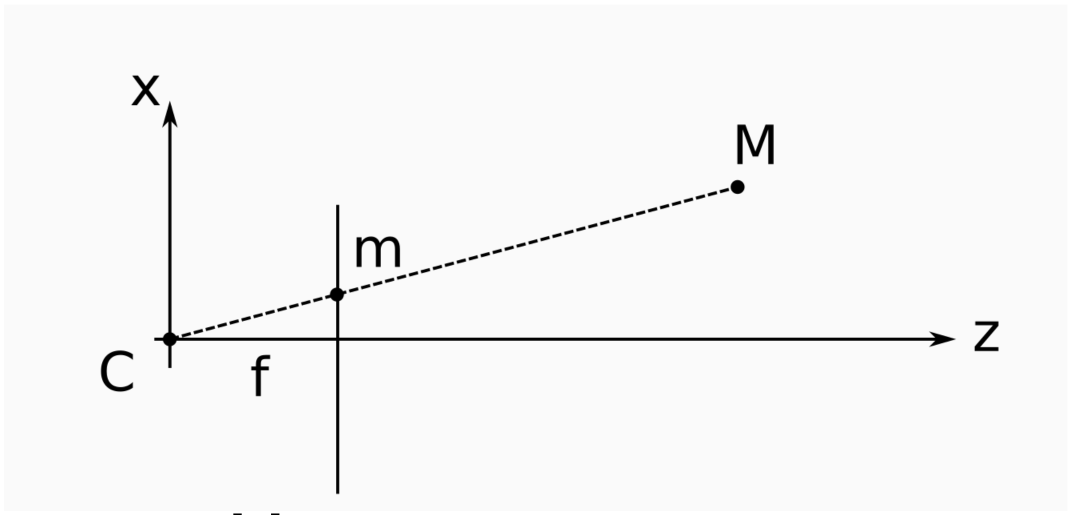
8 CAMERA GEOMETRY

- A camera is a mapping between the 3D world(object space) and a 2D image.

8.1 Projective Camera

8.1.1 Projection in 2D

- Pinhole camera model based(No lens effect)

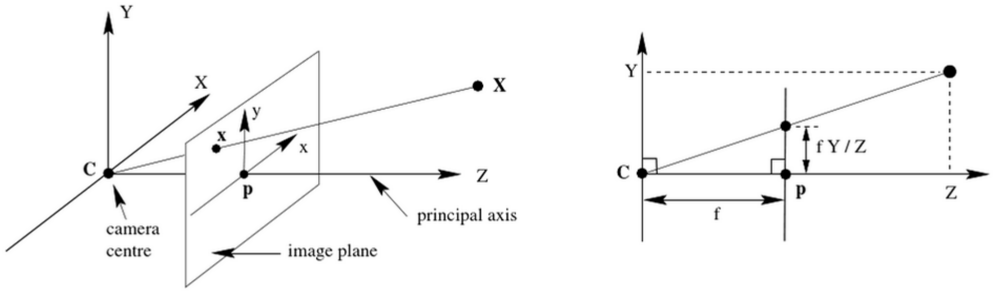


$\overline{M} = \begin{bmatrix} X \\ Z \end{bmatrix}$ **f**: focal length of camera (principle distance)

•If we use <u>triangular similarity</u>, then

$$\tfrac{k}{X} = \tfrac{f}{Z} \Rightarrow k = \tfrac{fX}{Z}$$

- Consider the projection of a 2D point onto a line segment. If the Cartesian coordinates of **M** are (X,Z), then the height of **m** is given by $\frac{fX}{Z}$.



p is the principle point

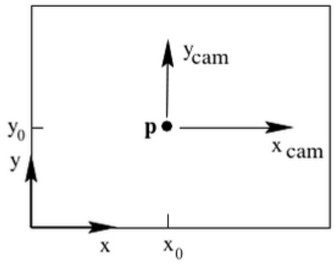
$$\overline{X} = \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ W_C \end{bmatrix}_{\text{Projective Geometry with scale parameter}} \xrightarrow{Euclidean} \begin{bmatrix} \frac{X_C}{W_C} \\ \frac{Y_C}{W_C} \\ \frac{Z_C}{W_C} \end{bmatrix}_{\text{Euclidean coordinate}}$$

- Assume $\overline{X} = \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix}$
- To calculate the coordinates, we first draw the coordinate system on sensor(image) plane.
- The process that we are doing is mapping 3D points into 2D.

$$\overline{m} = \begin{bmatrix} m_x \\ m_y \\ 1 \end{bmatrix} \quad m_x = \frac{fX_c}{Z_c} \; m_y = \frac{fY_c}{Z_c}$$

8.1.2 Projection in 3D

$$\overline{M} = \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} = \begin{bmatrix} M_x \\ M_y \\ M_z \\ M_w \end{bmatrix} \Rightarrow \begin{matrix} X_C = \frac{M_x}{M_w} \\ Y_C = \frac{M_y}{M_w} \\ Z_C = \frac{M_z}{M_w} \end{matrix}$$

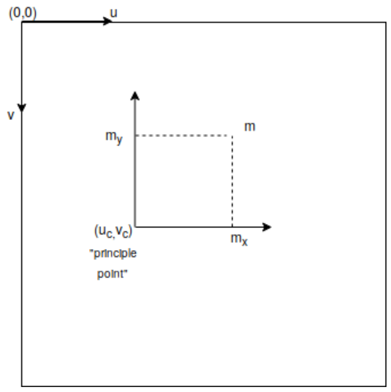


$$\overline{M} = \begin{bmatrix} M_x \\ M_y \\ M_z \\ M_w \end{bmatrix} \quad \overline{m} = s\begin{bmatrix} m_x \\ m_y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} M_x \\ M_y \\ M_z \\ M_w \end{bmatrix} = \begin{bmatrix} fM_x \\ fM_y \\ M_z \end{bmatrix}$$

$$\begin{bmatrix} fM_x \\ fM_y \\ M_z \end{bmatrix} = \begin{bmatrix} f\frac{M_x}{M_z} \\ f\frac{M_y}{M_z} \\ 1 \end{bmatrix}$$

$\rightarrow$ Unit of m_x/m_y is the same as unit of focal length(f)

$\rightarrow$ (0,0,f) is the principle point on plane



$$m_u = \alpha_x m_x + u_c$$
$$m_y = -\alpha_y m_y + v_c$$

$\bullet$Unit of α is $\left(\frac{\text{pixel}}{\text{milimeter}}\right)$

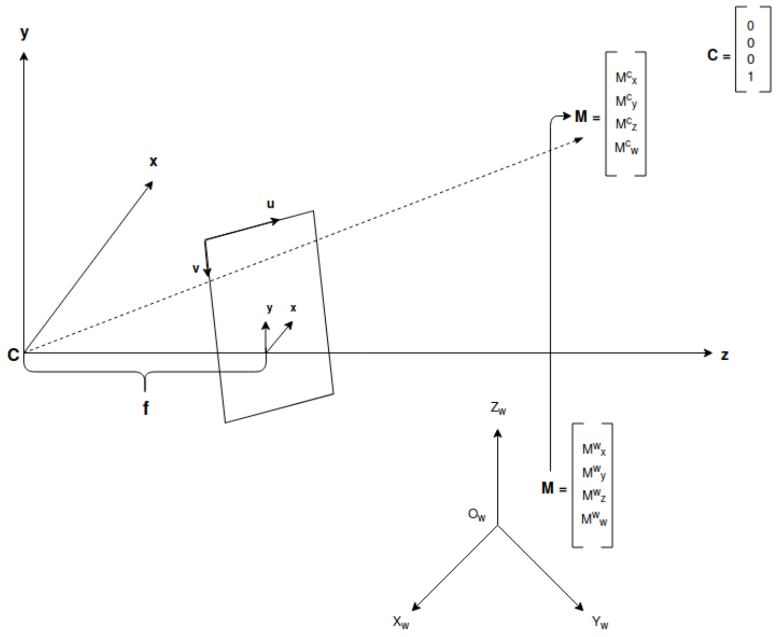
8.1.3 Image Plane Transformation

$$\begin{bmatrix} \alpha_x & 0 & u_c \\ 0 & -\alpha_y & v_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} \alpha_x f & 0 & u_c & 0 \\ 0 & -\alpha_y f & v_c & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

•Unit of focal length(f) is milimeter
•Unif of $\frac{f_x}{f_y}$ is pixels
•$f_x = \alpha_x f = \frac{\text{pixel}}{\text{milimeter}}$.milimeter = pixel

$$\lambda \begin{bmatrix} m_u \\ m_v \\ 1 \end{bmatrix} = \begin{bmatrix} f_X & 0 & u_c & 0 \\ 0 & f_y & v_c & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} M_x \\ M_y \\ M_z \\ M_w \end{bmatrix} = \begin{bmatrix} f_x M_x + u_c M_z \\ f_y M_y + v_c M_z \\ M_z \end{bmatrix} = \begin{bmatrix} \frac{f_x M_x}{M_z} + u_c \\ \frac{f_y M_y}{M_z} + v_c \\ 1 \end{bmatrix}$$

8.1.4 World to Camera Transformation



- To convert or more precisely finding corresponding coordinates of plane from one coordinate system to another coordinate system, we use rotation and translation.

$$\begin{bmatrix} M^c_x \\ M^c_y \\ M^c_z \\ 1 \end{bmatrix} = \begin{bmatrix} & [R]_{3\times3} & & \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}_{3\times1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} M^w_x \\ M^w_y \\ M^w_z \\ M^w_w \end{bmatrix} \Rightarrow M^c = \begin{bmatrix} R & \bar{t} \\ \bar{0}^T_3 & 1 \end{bmatrix} M^w$$

$$\begin{bmatrix} M_x^c \\ M_y^c \\ M_z^c \\ 1 \end{bmatrix} = \begin{bmatrix} R_{3\times3} & \bar{t}_{3\times1} \\ [0 \; 0 \; 0] & 1 \end{bmatrix} \begin{bmatrix} M_x^w \\ M_y^w \\ M_z^w \\ M_w^w \end{bmatrix} \Rightarrow \begin{bmatrix} M_x^c \\ M_y^c \\ M_z^c \end{bmatrix} = R \begin{bmatrix} M_x^w \\ M_y^w \\ M_z^w \end{bmatrix} + \bar{t}$$

$$\lambda \begin{bmatrix} m_u \\ m_v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_c \\ 0 & f_y & v_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} M^c \rightarrow K = \begin{bmatrix} f_x & 0 & u_c \\ 0 & f_y & v_c \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{K} \rightarrow \underset{\text{Dependent on camera features}}{\text{Internal Calibration Matrix}}$$

$$\lambda \begin{bmatrix} m_u \\ m_v \\ 1 \end{bmatrix} = K \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right] \begin{bmatrix} R & \bar{t} \\ \bar{0}_3^T & 1 \end{bmatrix} M^w = \underbrace{K[R|\bar{t}]}_{P} M^w$$

•$M^w \rightarrow$ A vector in world coordinate system
•$P_{3\times4} \rightarrow$ Camera projection matrix

8.1.5 Estimation of P

$$R = R_x^\phi R_y^\theta R_z^\alpha = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{bmatrix} \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \begin{bmatrix} cos\alpha & -sin\alpha & 0 \\ sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}_{C^c} = \begin{bmatrix} R & \bar{t} \\ \bar{0}_3^T & 1 \end{bmatrix} C^w$$

$$C^w = \begin{bmatrix} R & \bar{t} \\ \bar{0}_3^T & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} R^T & -R^T t \\ \bar{0}_3^T & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = -R^T t$$

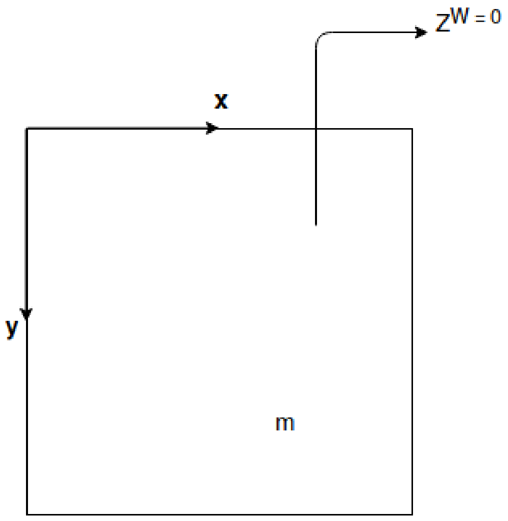
$\rightarrow$ Rotation matrix is orthogonal matrix ($R^T = R^{-1}$)

- $X^c = RX^w + t \Rightarrow X^w = R^{-1}(X^c - t) = R^T X^c - R^T t$
- $m_o = PM_o \rightarrow m_o \times PM_o = \bar{0}_{3\times1}$
- P has $\underset{(3\times4)}{12}$ variant and degree of freedom is 11.

- Each "$m_o \leftrightarrow M_o$" relation has 2 independent equations.
- For 11 degrees of freedom(dof), we should use 6 point of correspondences. For one correspondence, we gain 2 independent equation.

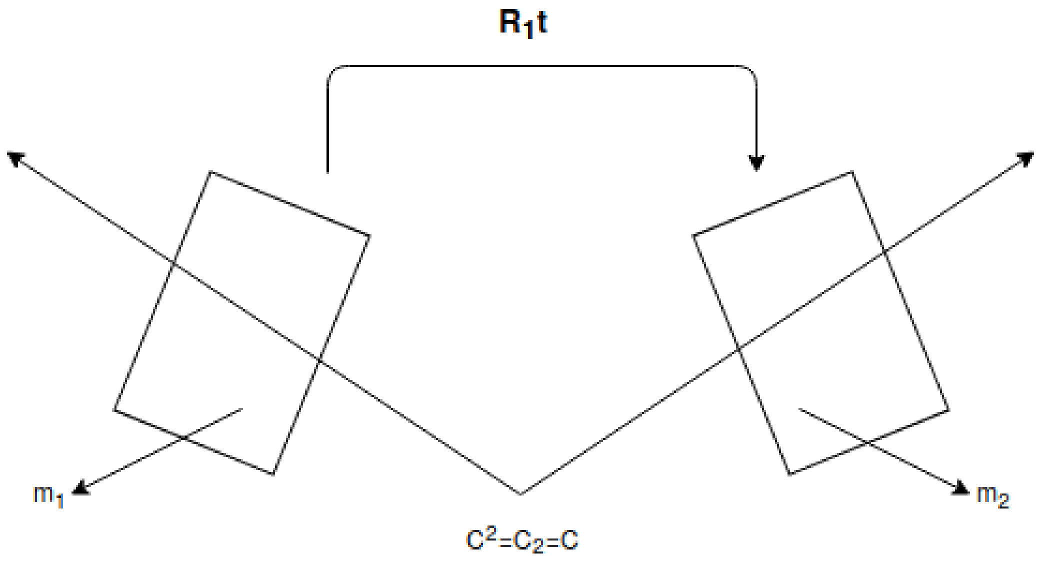
$$\rightarrow \boxed{\overline{\mathbf{m}} = \mathbf{K}[\mathbf{R}|\overline{\mathbf{t}}]M^w} \Big\} \text{ From 3D world into 2D image plane}$$

- A plane in real world has three coordinates, but one of them is 0



$$\overline{m} = K[\overline{r}_1\overline{r}_2\overline{r}_3|\overline{t}]\begin{bmatrix} M_x^w \\ M_y^w \\ M_z^w \\ M_w^w \end{bmatrix} \Rightarrow \overline{m} = \underbrace{K[\overline{r}_1\overline{r}_2\overline{t}]}_{H}\begin{bmatrix} M_x^w \\ M_y^w \\ M_w^w \end{bmatrix}$$

- If we keep the position of camera same, and only rotate a bit, there will be occur a homography between two images.



$$\overline{m}_1 = K[I|0]M$$
$$\overline{m}_1 = K[I|0]\begin{bmatrix}(KR)^{-1}\overline{m}_2\\ M_w^w\end{bmatrix}$$
$$\overline{m}_1 = K(KR)^{-1}\overline{m}_2$$
$$\overline{m}_1 = \underbrace{KR^T K^{-1}}_{H}\overline{m}_2$$

$$\overline{m}_2 = K[R|O]\begin{bmatrix}M_x^w\\ M_y^w\\ M_z^w\\ M_w^w\end{bmatrix}$$
$$\overline{m}_2 = KR\begin{bmatrix}M_x^w\\ M_y^w\\ M_z^w\end{bmatrix}$$
$$\begin{bmatrix}M_x^w\\ M_y^w\\ M_z^w\end{bmatrix} = \left(KR\right)^{-1}\overline{m}_2$$

In order to prepare this document, İYTE CENG-391 course slides were used.
Introduction to Image Understanding