

Geometric and 3D Computer Vision

0. Introduction

Computer Vision

Set of computational techniques aiming at estimating or making explicit the geometric and dynamic properties of the 3D world from digital images.

Computer vision vs. computer graphics

Computer graphics → 3D to 2D → information loss

Computer graphics is creation, manipulation and storage of geometric objects (modeling) and their images (rendering)

Computer vision → 2D to 3D → need for models

Computer vision is a discipline that studies how to reconstruct, interpret and understand a 3D scene from its 2D images in terms of the properties of the structure present in scene.

Levels of vision

1. Low (image processing)
 - a. image restoration | contrast enhancement | noise reduction
 - b. image → image
2. Medium
 - a. segmentation | shape recognition
 - b. image → attributes, features, geometry
3. High
 - a. scene understanding
 - b. image → concepts

1. The Image Formation process

Light quantities

- Radiance (W): total amount of energy that flows from the light source
- Luminance (lm): amount of energy an observer perceives from a light source
- Brightness: subjective descriptor of light "intensity". One of the key factors in describing color sensation.
 - *perceived* brightness for a region with the same luminance across different images can be different

Image sensing

Scene must be illuminated, because to produce an image the camera captures the light reflected by the objects in the scene.

Incoming light radiation transformed into a voltage by an imaging sensor which is sensible to specific wavelengths. Sensors usually arranged in linear or 2D arrays.

CCD vs CMOS

CCD: A/D conversion from voltage to digital signal at the end of each row/column

CMOS: A/D conversion from voltage to digital signal at each sensing cell

CMOS are usually better (average light conditions); CCD faster and better in low light (less noise; more "precise").

Imaging function

Image can be modelled as function $I : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}$

Domain is a (usually rectangular) subset of the real image plane. $I(x,y)$ proportional to amount of light energy (*intensity*) collected at the image plane at coordinates (x,y) .

Converting real image into digital one

- **Sampling:** reduce image domain to finite set of spatial coordinates $\mathbb{R}^2 \rightarrow M \times N$
 - each sample is called *pixel*
- **Quantization:** reduce sensor response (function codomain) to finite set of values $\mathbb{R} \rightarrow [0, 1, \dots, 2^b]$

The *digitization* process requires M, N and L (usually power of 2) to be chosen.

- M, N: spatial resolution (matrix size)
 - measure of the smallest discernible detail in an image, usually defined as pixels per unit distance.
- L: intensity resolution (quantized intensity levels)
 - smallest discernible change in intensity level

Dynamic range and contrast

Beware of the difference!

Dynamic range:

- characteristic of an *imaging system*
- ratio of the maximum measurable intensity to the minimum detectable intensity level

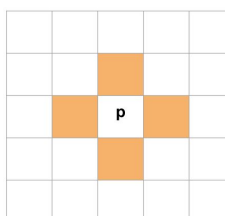
Contrast:

- characteristic of an *image*
- difference between the highest and the lowest intensity level of an image

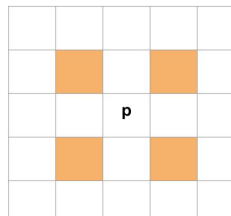
Tone mapping creates images using HDR (non-linear mapping)

Relationships between pixels

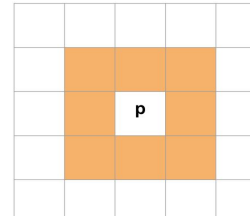
4-neighbors (N_4)



diagonal neighbors (N_d)



8-neighbors (N_8)



Adjacency

Let V be the set of intensity values used to define adjacency.

p, q with values from V are:

- 4-adjacent if $q \in N_4(p)$
- 8-adjacent if $q \in N_8(p)$
- M-adjacent if either
 - $q \in N_4(p)$
 - $q \in N_d(p)$ and $N_4(p) \cap N_4(q)$ has no pixels with values from V
 - note: it is a "special kind" of 8-adjacency

Connected components

Two pixels p and q are connected if there exists a path between them. Given a pixel p , the set of all pixels connected to p is called a connected component.

Let R be a subset of pixels in an image. R is called a region if it contains only one connected component.

2. Intensity Transformations

Introduction

Techniques that modify the intensity of pixels implemented in the *spatial domain* (image plane containing pixels of the image).

A spatial domain process can be described by the expression $g(x,y)=T[f(x,y)]$

where g is the output image, T is an operator on f defined over a neighborhood of (x,y) , and $f(x,y)$ is the input image.

Neighborhood of (x,y) is usually rectangular, centered on (x,y) , much smaller than the image.

When the neighborhood has size 1×1 , $g(x,y)$ depends only on the value of f at (x,y)

T is an *intensity transformation function* $s=T(r)$ where s and r are the intensity of g and f at a generic point (x,y) . Note that this must not be confused with spatial transformations

$g(x,y)=f(s(x,y))$

Negative

The negative of an image with intensity levels in the range $0 \dots L-1$ is obtained by the following expression: $s=L-1-r$

This processing enhances white or grey details embedded in dark regions.

Gain/Bias

Multiplication and addition with a constant: $s=\alpha r + \beta$

The two parameters $\alpha > 0$ and β are often called *gain* and *bias* and control contrast and brightness respectively.

Log transformations

Compress the dynamic range for images with large variation in pixel values

$s = c \cdot \log(1+r)$

With c being an arbitrary constant, r being the input intensity level and s being the output intensity level.

Log transformations can both be used to map a narrow range of low intensity values in input to a wider range of output levels, and vice versa.

Gamma transformations

$s = c \cdot r^\gamma$ with c and γ positive constants.

We compress values in a similar fashion to the log transformation with more flexibility.

Curves generated with $\gamma > 1$ have the opposite effect of those with $\gamma < 1$

Identity transformation when $c = \gamma = 1$

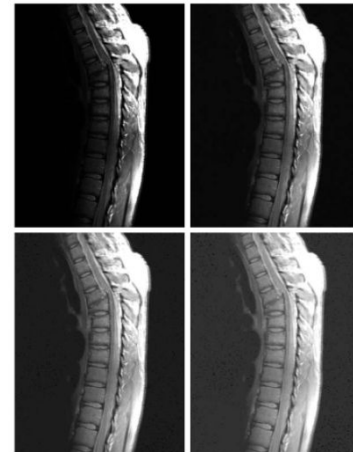
Gamma transformations are useful because many devices have a power-law (non-linear) response

Image appears too dark

Corrected with $\gamma = 0.4$

Corrected with $\gamma = 0.6$

Corrected with $\gamma = 0.3$



Contrast enhancement

1. Piecewise-linear function with a sigmoid shape
2. Thresholding: $s=0$ if $r \leq t$, $s=1$ otherwise

Image histogram

We want to determine the best parameters for the aforementioned functions.

The *image histogram* allows us to analyze problems in the intensity (or color) distribution of an image. In particular, it is the **empirical distribution of image intensities**.

Let $0 \dots L-1$ be the intensity levels of an image. The image histogram is a discrete function

$$h(r_k) = n_k$$

where r_k is the k^{th} intensity value and n_k is the number of pixels in the image with intensity r_k

Usually the histogram is normalized by dividing each component by the total number of pixels. This way each histogram component is an estimate of the probability of the occurrence of the intensity r_k

Histogram equalization

$s = T(r)$ mapping function such that the resulting histogram of s is flat (uniform distribution).

Formally, intensity levels can be viewed as random variables in interval $0 \dots L-1$. The image histogram of s is an estimate of the PDF of s , $p_s(s)$ and the histogram of r is an estimate of $p_r(r)$. When we apply a function $s = T(r)$ to a random variable r we obtain $p_s(s) = p_r(r) |dr/ds|$. Notice that T must be continuous, differentiable and strictly monotonic.

As a matter of fact, if we use $T(r) = (L-1) \int_0^r p_r(w) dw$ (integral being the CDF of f)

we obtain $ds/dr = 1/(L-1)$, i.e. a uniform distribution.

Since we are working in a discrete domain (pixels), we can apply the function

$$s_k = T(r_k) = (L-1) \sum_{j=0}^k p_r(r_j) = \frac{L-1}{MN} \sum_{j=0}^k h(r_j) \quad \text{where the last summation is the sum of the first } k$$

components of the input image histogram. By applying this transformation, we obtain an almost flat histogram of the output image.

Histogram matching

We want to specify the shape of the histogram we want to have in the output image.

Input image has intensities described by r with PDF $p_r(r)$

Output image has intensities described by z with given (by us) PDF $p_z(z)$

Let s be a random variable (described the same way as before) with the following property:

$$s = T(r) = (L-1) \int_0^r p_r(w) dw \quad s \text{ is the equalized version of } r$$

We define a function $G(z)$ as follows: $G(z) = (L-1) \int_0^z p_z(t) dt$.

We can clearly see that $G(z)=T(r)$ and both are monotonically increasing, thus $z=G^{-1}[T(r)]=G^{-1}(s)$

We can now formalize the histogram matching algorithm:

1. Compute the PDF (normalized histogram) of $p_r(r)$
2. Use the specified PDF $p_z(z)$ to obtain $G(z)$
3. Obtain the inverse transformation $z=G^{-1}(s)$
4. Equalize the input image to obtain s . Apply $G^{-1}(s)$ to s , hence obtaining the corresponding output image.

When all pixels are processed, the PDF of the output image will be (almost) equal to the specified one. "Almost" because in the discrete case $G(z)$ is implemented as a lookup table with L entries, thus:

- in case of duplicate entries while inverting, the smallest z is chosen by convention
- in case of missing values, choose the nearest one (or interpolate)

Otsu Thresholding

The histogram can also give useful clues on the threshold level to use in an image.

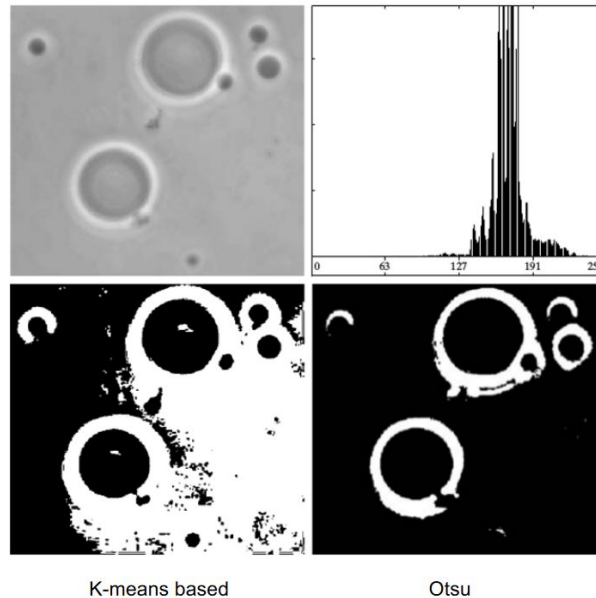
Thresholding is basically a clustering problems in which two clusters (black and white pixels) are sought. To find these two clusters, we minimize the *within-class* variance. This is proved to be the same problem as maximizing the *between-class* variance, which is an easier problem to solve!

Algorithm:

1. Compute the normalized histogram of the input image. Denote each component of the histogram as p_i , $i=0,1,...,L-1$
2. Compute the cumulative sums $P_1(T) \quad \forall T=0,...,L-1$
3. Compute the cumulative means $m(T) \quad \forall T=0,...,L-1$
4. Compute the global intensity mean m_G
5. Compute the between class variance $\sigma_B^2(T) \quad \forall T=0,...,L-1$

6. Apply threshold with a value of T for which $\sigma_B^2(T)$ is maximum

Otsu thresholding iterates on image histogram, instead of iterating on image pixels such as other global methods (e.g. K-means).



3. Spatial filtering

A spatial filter consists of:

- a **neighborhood** (usually a small rectangle)
- a predefined **operation** that is performed on the image pixels encompassed by the neighborhood

A spatial filter creates a **new pixel** with coordinates equal to the **centre** of the neighborhood and whose **value** is the **result** of the filtering operation

Linear filters

A filter is defined in terms of a **coefficient matrix** W

Operation: sum of products of the filter coefficients and the image pixels encompassed by the filter.

$$g(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x + s, y + t)$$

Observations:

- $w(0,0)$ is aligned with the pixel at location (x,y)
- for a neighborhood of $M \times N$ we assume that $M=2a+1$ and $N=2b+1$
 - we assume filters of odd size, centered at (x,y) , at least 3×3 in size

Correlation and Convolution

Correlation: process of moving a filter mask over a signal and computing the sum of products at each location

Convolution: correlation but the filter mask is first rotated by 180° .
(often the filter is symmetric \rightarrow they are usually the same)

1D correlation/convolution

When needed, some padding (0) is usually added to the original signal. Once the correlation/convolution has been applied, we remove the padding so that the result has the same size as the input. Note that:

- Correlation is a function of displacement of the filter
- Correlating a filter w with a function (signal) that contains all 0s and a single 1 yields a result that is a copy of w rotated by 180°

A fundamental property of convolution is that convolving a function with a unit impulse yields a copy of the mask at the location of the impulse.

2D correlation/convolution

In case of 2D functions such as images, the correlation works in a similar manner.

For a filter of size $M \times N$ we first pad the image with a minimum of $M-1$ rows at top/bottom and $N-1$ columns at top/bottom, all filled with 0s. We shift the filter at each vertical and horizontal shift to perform the correlation/convolution operation.

$$(w * f)(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x + s, y + t)$$

$$(w \star f)(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x - s, y - t)$$

Properties of convolution

Commutative: $F \star G = G \star F$

Associative: $(F \star G) \star H = F \star (G \star H)$

Linearity: $(aF + bG) \star H = aF \star H + bG \star H$

Template matching

Find a specific pattern inside an image by using convolution/correlation.

Output makes no sense, but the bright values in the output image are where the region is coherent with the mask used while applying convolution. Take the (first n) maxima to get the specific points in the image. In other words: if the filter is a portion of the image itself, we will see a bright spot where the filter overlaps with itself in the original image.

Smoothing spatial filters

Useful for blurring and noise reduction. The output of a smoothing, linear spatial filter is simply the average of the pixels contained in the neighborhood of the filter mask

$\frac{1}{9} \times$	1	1	1
	1	1	1
	1	1	1
Average filter			
$\frac{1}{16} \times$	1	2	1
	2	4	2
	1	2	1
Weighted average filter			

Special type of weighted average filter: Gaussian filter. Each element has the form $e^{-\frac{1}{2} \frac{x^2+y^2}{\sigma^2}}$

Order-statistic filters

Non-linear spatial filters whose response is based on:

1. ordering the pixels contained in the image area encompassed by the filters
2. replacing the value of the central pixel with the value determined by the ranking result

These filters cannot be performed as convolution or correlation as they may require sorting the pixels etc. As a matter of fact, the most common are:

- Median-filter: replaces the value of a pixel with the median of the intensity values in the neighborhood of that pixel (including the old value of the pixel in the computation)
- Max-filter: replaces the value of a pixel with the brightest intensity value in the neighborhood
- Min-filter: replaces the value of a pixel with the darkest intensity value in the neighborhood

Filter and noise

Smoothing and median filters are particularly useful for image denoising.

Different noise models:

- Additive noise: $\hat{I}(x, y) = I(x, y) + \omega$
- Salt and pepper (impulse) noise

α -trimmed mean filter

To eliminate both additive and impulse noise use a robust estimate of the mean.

- Eliminate top and bottom $\alpha/2$ values
- Take the average of the remaining pixels

Sharpening filters

Goal: highlight transitions in intensity.

Unsharp masking (high-boost filtering):

1. blur original image
2. subtract blurred image from the original to obtain a mask
3. add the mask to the original (multiplied by a constant for high-boosting)

In general, sharpening filters are based on the concept of differentiation.

Blurring: Averaging pixels in a neighborhood; integration; remove details

Sharpening: Differentiation (1st or 2nd order derivatives); enhance details

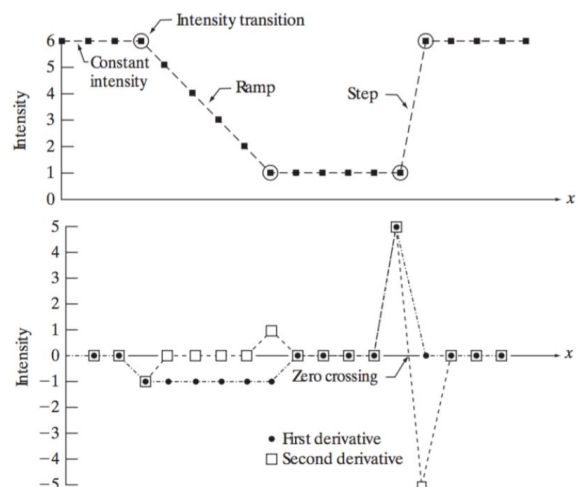
Derivatives (approximations)

First order: $f(x+1) - f(x)$

- zero in constant-intensity areas
- non-zero on intensity steps or ramps

Second-order: $f(x+1) + f(x-1) - 2f(x)$

- zero in constant-intensity areas
- non-zero on onsets, end of ramps and steps
- zero along ramps



Sharpening and derivatives

We want to highlight intensity transitions (edges)

First-order derivative: would result in thick edges because the derivative is nonzero along a ramp

Second-order derivative: would produce a double edge one pixel thick, separated by zeros. It enhances fine details much better than the first derivative, and is also easier to implement

Laplacian filtering

Laplacian: simplest isotropic second-order derivative operator.

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

In discrete form, it can be expressed in terms of finite differences. Since derivatives are linear operators, the Laplacian is a linear operator and can thus be implemented as a convolution. The diagonal directions can be incorporated in the definition of the discrete Laplacian by adding two more terms, one for each of the two diagonal directions.

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

0	1	0
1	-4	1
0	1	0

This filter gives an isotropic result for increments of 90°

1	1	1
1	-8	1
1	1	1

This filter gives an isotropic result for increments of 45°

The Laplacian operator highlights intensity discontinuities in an image and de-emphasizes regions with slowly varying intensity levels.

Adding the effect of the Laplacian operator $g(x, y) = f(x, y) + c(\nabla^2 f(x, y))$ to the original image sharpens the details.

4. Morphological Image Processing

Morphology: unified and powerful approach to numerous image processing problems; exploits *set theory*.

We consider *thresholded* B/W images. The *set of white pixels* ($\subset Z^2$) contains the complete morphological description of the image. Each vector expresses the coordinates of white/black pixels.

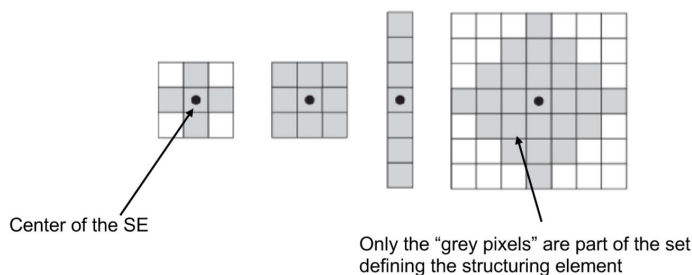
Set translation by a point $z=(z_1, z_2)$ $(B)_z = \{c \mid c = b + z \quad \forall b \in B\}$

Set reflection $\hat{B} = \{w \mid w = -b \quad \forall b \in B\}$

Structuring elements

Morphological operations are based on structuring elements (SEs).

SEs: small sets or subimages used to probe an image under study for properties of interest. Conceptually similar to the convolution kernel in convolution.



Erosion

With A and B sets in Z^2 , the erosion of A by B is defined as $A \ominus B = \{z \mid (B)_z \cap \bar{A} = \emptyset\}$

I.e. the erosion of A by B (B erodes A) is the set of all points z such that B , translated by z , is contained in A .

Dilation

With A and B sets in Z^2 , the dilation of A by B is defined as $A \oplus B = \{z \mid (B)_z \cap A \neq \emptyset\}$

I.e. the dilation of A by B (B dilates A) is the set of all displacements such that the reflection of B and A overlaps by at least 1 element. The basic process of flipping B around its origin and then sliding it over the image A is analogous to the spatial convolution (remember convolution = correlation with 180° rotation, but in most cases the filter is symmetric).

One of the simplest applications of dilation is for bridging gaps, e.g. in OCR.

Opening given an image A and a SE B , the opening is defined as $A \circ B = (A \ominus B) \oplus B$

I.e. the opening is an erosion followed by a dilation

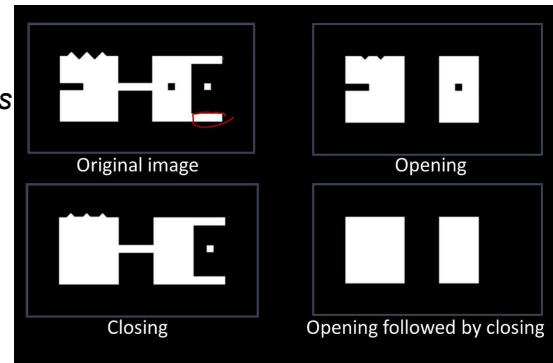
The opening smooths the contour of objects, breaks narrow isthmuses and eliminates thin protrusions. Opening is pretty useful for removing noise in an image.

Closing given an image A and a SE B , the closing is defined as $A \bullet B = (A \oplus B) \ominus B$

I.e. the closing is a dilation followed by an erosion

The closing smoothes sections of contours, fuses narrow breaks and long thin gulfs, eliminates small holes and fills gaps in the contour. Closing is useful to "close the holes" in an image.

A sequence of opening and closing both eliminates noise and closes all the erroneous breaks due to an imperfect thresholding.



Hole filling

A hole may be defined as a background region surrounded by a connected border of foreground pixels. Given an initial point inside an hole, the goal is to fill the hole with 1s.

Let X_0 be the initial array with the same size of A , filled with 0s except for the location of a given point in each hole. The following iterative procedure fills all holes with 1s:

$$X_k = (X_{k-1} \oplus B) \cap \bar{A}$$



Connected components

Let X_0 be the initial array with the same size of A , filled with 0s except for the location of a known point in each region. The following iterative procedure fills the connected component with 1s:

$$X_k = (X_{k-1} \oplus B) \cap A$$



Boundary extraction

The boundary of a set A can be obtained by first eroding A by B and then performing the set difference between A and its erosion: $\beta(A) = A - (A \ominus B)$

Moore boundary tracking

Extract an ordered sequence of foreground boundary points from a region. Work with binary thresholded images (0 is background, 1 is foreground) padded with a border of 0s (no foreground region touches the image border).

Algorithm:

1. Let the starting point b_0 be the uppermost and leftmost point in the image that is labeled 1. Let c_0 be the west neighbor of b_0 [i.e. if b_0 is at (x,y) , c_0 is at $(x-1, y)$]
2. Examine the 8-neighbors of b_0 , starting at c_0 and proceeding in a clockwise direction
 - a. let b_1 be the first neighbor encountered whose value is 1
 - b. let c_1 be the (background) point immediately preceding b_1 in the clockwise sequence
3. Let $b=b_1$, $c=c_1$
4. Let the 8-neighbors of b , starting at c and proceeding in a clockwise direction, be denoted by n_1, n_2, \dots, n_8 . Find the first n_k labeled 1
5. Let $b=n_k$ and $c=n_{k-1}$
6. Repeat (4) and (5) until $b=b_0$ and the next boundary point found is b_1 . The sequence of b points found when the algorithm stops is the set of ordered boundary points.

Grayscale morphology

Erosion, dilation, opening and closing can also be defined for grayscale images.

Given:

- $f(x,y): Z^2 \rightarrow R$
- $b(x,y): Z^2 \rightarrow R$

the structuring element b is used as a probe to examine a given image for specific properties. It can be flat or non-flat (more common).

Grayscale erosion

The erosion of f by a flat structuring element b at any location (x,y) is defined as the minimum value of the image in the region defined by b when the origin of b is at (x,y) .

$$[f \ominus b](x, y) = \min_{(s, t) \in b} \{f(x + s, y + t)\}$$

This operation is conceptually similar to a convolution that takes the minimum.

Grayscale dilation

The dilation of f by a flat structuring element b at any location (x,y) is defined as the maximum value of the image in the region defined by the reflection of b when the origin of it is at (x,y)

$$[f \oplus b](x, y) = \max_{(s, t) \in b} \{f(x - s, y - t)\}$$

We use the minus sign instead of the + because b is reflected, hence $\hat{b}=b(-x,-y)$

Opening and closing

Same as their binary counterparts. Geometrically:

- opening is like pushing the structuring element up from below against the under-surface of f while being translated. Opening attenuates bright features
- closing is like pushing down the structuring element on top of the curve while being translated. Closing attenuates dark features

Morphological smoothing

Since both opening and closing usually suppress details (bright and dark), they are often used in combination for image smoothing and noise removal.

Morphological smoothing

Dilation thickens regions, erosion shrinks them. Their difference emphasizes boundaries between regions. $g = (f \oplus b) - (f \ominus b)$

The net result is an image in which the edges are enhanced and the contributions of the homogeneous areas are suppressed, thus producing a "gradient" effect.

Top-hat and Bottom-hat

Top-hat: f minus its opening $T_{\text{hat}}(f) = f - (f \ominus b)$

Bottom-hat: closing of f minus f $B_{\text{hat}}(f) = (f \oplus b) - f$

Goal: select objects from an image by using a structuring element in the opening or closing operation that does not fit the objects to be removed. The difference then removes just the

selected objects. For instance, top-hat can be used to correct illumination before applying some thresholding.

5. Edge Features

Feature: local, meaningful, detectable part of an image.

In computer vision, a feature is a **location of sudden change**. We look for features because their information content is high, they are mostly invariant to changes in viewpoint or illumination, and using them reduces the computational burden.

A good feature is invariant to viewpoint, lighting conditions, object deformations and partial occlusions, and should be unique, easy to be found and easy to extract.

Edges

Edge pixels are pixels at which the intensity of an image function changes abruptly. They represent a simple and meaningful type of feature:

- edges in an image have many interesting causes
- looking at edges reduces the information required (few pixels in binary image instead of all pixels in grayscale image)
- biological plausibility

An edge may be found because of depth discontinuity, surface orientation discontinuity, reflectance discontinuity, surface color discontinuity, ...

Edge detection

Steps: detect short linear edge segments (edgels), aggregate edgels into extended edges, possibly combine the edges.

Edgels detection: find pixels that present abrupt local changes in intensity

Edge models

- *step* edge: transition between two intensity levels occurring ideally over a 1 pixel distance
- *ramp* edge: more common. Blurring introduced by limitations in the focusing mechanism. Slope of the ramp inversely proportional to degree of blurring in the edge
- *roof* edge: may arise in range imaging (thin objects closer to the sensor than their equidistant background)

Problem: noise results in deviations from the ideal shapes. The sharper and less noisy the images, the better!

Derivatives

First derivative: magnitude can be used to detect presence of an edge at a point.

Second derivative: produces 2 responses for each edge, we can locate the center using the zero-crossing

Image gradient

Useful to find edge strength and direction. Points in the direction of the greatest rate of change of f at point (x,y) and its magnitude is the amount of change in that direction.

$$M(x,y) = \text{mag}(\nabla f) = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \quad \text{Magnitude: } \alpha(x,y) = \tan^{-1} \left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right)$$

Direction:

Computing image gradient

Approximation of derivatives in image: along x axis $f(x+1,y) - f(x,y)$, along y axis

$f(x,y+1) - f(x,y)$

Can be implemented using Roberts gradient operators. Problem: even sized windows don't have a central pixel.

Prewitt operators use 3x3 masks to approximate partial derivatives

Sobel operators: variants of Prewitt, use weight 2 instead of 1 in central pixels.

-1	-1	-1	-1	0	1
0	0	0	-1	0	1
1	1	1	-1	0	1

Derivatives and noise

Noise can have a significant impact on derivatives (especially second order).

Image smoothing is usually a mandatory step prior to the use of derivatives.

Marr-Hildreth edge detector

Key observations: intensity changes depend on image scale; sudden intensity change will generate a peak or a trough in the 1st derivative; image noise and nature of edges should be taken into account. Usage of **Laplacian of Gaussian** (LoG) operator instead of Sobel masks.

Gaussian part blurs the image, reducing the intensity of structures (noise included) at scales much smaller than σ ; the Laplacian is isotropic (uniform in all orientations), thus it responds equally to changes in intensity in any direction, whereas 1st derivative requires multiple masks (being directional). The Laplacian also resembles the characteristic of the human visual system.

Algorithm:

1. Convolve input image with the LoG filter
 - a. this step is linear, therefore it can be split into:
 - i. blur the image with a Gaussian filter
 - ii. compute Laplacian of the blurred image (with Laplacian mask)
2. Find the zero-crossing of the convolution to determine the locations of edges in the input image
 - a. using a 3x3 neighborhood centered at a point p , a zero crossing at p implies that the signs of at least two of its opposing neighboring pixels are different
 - i. 4 cases: up/down, left/right, diagonals
 - ii. usually only checking signs is very sensitive to noise, therefore we also impose a threshold on the absolute value of the difference

Canny Edge Detector

Optimal: low error rate, edge points well localized, single edge point response. If we express these properties mathematically, we can find optimal solutions.

Given a filter f , define two objective functions:

$\Delta(f)$ large if f produces good localization $\Sigma(f)$ large if f produces good detection

Problem: find the best filter f that maximizes $\Sigma(f) \Delta(f)$ with the additional constraint that a single peak should be generated at a step-edge. The derivative of a Gaussian is a very close approximation! It is the optimal filter in the 1D case of a step-edge corrupted by noise. This result can be generalized to the 2D case by recognizing that the 1D approach still holds in the direction of the edge normal, i.e. the derivative of Gaussian should be applied in all possible directions. Since the derivative is also a linear operator, the 2D detector can be implemented by 1) smoothing the image with a 2D Gaussian filter and 2) computing the gradient of the result.

Non-maxima suppression

Computing the gradient magnitude with the derivative of a Gaussian yields good results that typically contain wide ridges around local maxima. Non-maxima suppression is used to thin the ridges: we select the single maximum point across the width of an edge.

Hysteresis thresholding

Final operation: threshold non-maxima suppressed gradient image to reduce false edge points. Tradeoff between low and high threshold is tackled with hysteresis thresholding.

Use two thresholds: T_L and T_H . Edges in T_H are strong, edges in T_L are weak.

Idea: follow strong edges until they fall below T_L (i.e. they can be connected using weak edges).

Algorithm:

1. Locate next unvisited strong edge p
2. Mark as valid all weak edges connected to p (8-neighbors)
3. If there still exist unvisited edges, go back to step 1
4. Mark all remaining unvisited pixels as invalid and remove them from final edges

Complete algorithm for Canny:

1. smooth input image with Gaussian filter
2. compute gradient magnitude and angle
3. apply non-maxima suppression to gradient magnitude image
4. use hysteresis thresholding to detect and link edges

De-facto standard because of strong mathematical background and step 4, which is an important heuristics.



6. Finding curves

Often we have unstructured environments. All we have is an edge image. We do not know where the objects of interest may be.

Fitting: process of decomposing an image or set of tokens (pixels, isolated points, sets of edge points, ...) into components that belong to one or another simple family (circles, lines, ...). Useful in segmentation processes to make compact representations that emphasize relevant image structures or to analyze and *measure* man-made or geometrical objects. General problem: given some points, determine which curve could have generated that set of points.

Subproblems:

1. parameter estimation: we already know the association tokens-curves. We just need to recover the parameters of each curve
2. token-curve association: we only know how many curves are present, but not which token came from which curve. This must be solved in addition to parameter estimation
3. counting: no prior knowledge on the data. We need to estimate how many curves there are, the association tokens-curves and the curves' parameters

Parameter estimation

We assume to have observed a set of points generated by a certain curve model with unknown parameters. Goal: find the best set of parameters that justify the observations.

Approach 1: minimize loss function (sum of distances points-curve)

Approach 2: describe curve as generative model and find parameters that maximize probability of generating the observed data

In some cases such as 2D lines we obtain the same result.

Fitting lines

Simple line model

$Y = ax + b + \varepsilon$ $\varepsilon \sim N(0, \sigma^2)$ and independent from x_i and across observations.

Y depends on x and an additive Gaussian noise. Only the y coordinate is affected by noise!

$$\text{PDF: } p(y_i | x_i; a, b, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - (ax_i + b))^2}{2\sigma^2}}$$

Using maximum likelihood estimation (and applying the log) we obtain:

If we set

$$\beta = \begin{pmatrix} a \\ b \end{pmatrix} \quad f(x_i, \beta) = \beta_1 x_i + \beta_2 \quad \mathbf{X} = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

then the solution of $\operatorname{argmin}_{a,b} \sum_{i=1}^N (y_i - f(x_i, \beta))^2$ is $\operatorname{argmin}_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|^2$

$$\beta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Limitations:

- only vertical offset: error is dependent on reference frame
- works badly for almost vertical lines

Better model

Locus of points $ax + by + c = 0$. We impose $a^2 + b^2 = 1$ without loss of generality, since we can multiply both sides of the equation by any scalar and it will still hold.

Distance point-line = $|ax_i + by_i + c|$

The Gaussian noise is orthogonal to the line!

$$\operatorname{argmin}_{a,b,c} \sum_{i=1}^N (ax_i + by_i + c)^2 \text{ using Lagrange multipliers } \begin{pmatrix} \overline{x^2} & \overline{xy} & \overline{x} \\ \overline{xy} & \overline{y^2} & \overline{y} \\ \overline{x} & \overline{y} & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \lambda \begin{pmatrix} 2a \\ 2b \\ 0 \end{pmatrix}$$

subject to $a^2 + b^2 = 1$

Solved numerically as eigenvalue problem by setting $c = -a\bar{x} - b\bar{y}$

Fitting curves

To solve a problem with a polynomial of degree k , up to k^2 solutions may have to be considered. To solve the problem we can either consider specific properties of the curve or use an approximation of the distance function (as it is hard to compute a linear point-curve distance).

- **Algebraic distance:** measure point-curve distance by evaluating the polynomial equation at that point. $\text{dist}(d_x, d_y, \phi) = \phi(d_x, d_y)$
 - only accurate if point is very close to the curve
 - polynomial coefficients should be normalized before evaluating distance
 - algebraic and geometric distance don't usually coincide
 - choosing the normalization factor may remove possible good solutions

- Problem similar to lines, but in many cases it is hard to give an exact
$$\text{dist}(d_x, d_y, \phi) = \frac{\phi(d_x, d_y)}{\|\nabla \phi(d_x, d_y)\|}$$

formulation of the point-curve distance, thus we often approximate it.

Note: not all curves are guaranteed to have real points in them!

- Curve model: $\phi(x,y)=0$. Given data point (d_x, d_y) find closest point (u,v) on a curve such that $\phi(u,v)=0$ [the point belongs to the curve] and $\vec{s} = (d_x, d_y) - (u,v)$ is normal to the curve.
- This is a difficult problem to be solved: for a polynomial of grade k , up to k^2 solutions must be
- *Normalized distance (using the gradient):*
 - pros: -no need to choose normalization factor
-more accurate (normalized by length of normal)
 - can still be inaccurate for points far away from normal

Token-curve association

We either adopt:

- consensus-based approaches such as RANSAC
 - use minimal subset of points to define a curve; remaining points give consensus on validity of such curve
- voting schemes such as Hough Transforms
 - every point votes for all the curves that could have generated it. We choose the curve that gets most votes

RANSAC

Random sampling of observed data. Given a dataset with inliers and outliers, RANSAC uses a consensus scheme to find the optimal fitting result.

Assumption: given a (usually small) set of inliers, there exists a procedure which can estimate the parameters of a model that optimally explains or fits this data, e.g. given 2 points we can compute a line model that optimally explains the set.

For $i = 1 \dots N$:

1. Select a random subset of the original data. Call it *hypothetical inliers*
2. Fit the model to the hypothetical inliers
3. Test all other points against the model. Mark points either as inliers or outliers according to a loss function. Inliers = consensus set

return model that produced largest consensus set

$$1 - (1 - (1 - e)^s)^N = p \quad \rightarrow \quad N = \frac{\log(1 - p)}{\log(1 - (1 - e)^s)}$$

e = probability that a point is an outlier

s = number of points needed to fit a model

p = desired probability to find a good model given the samples

Hough transform

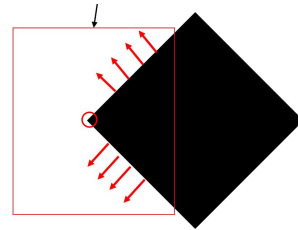
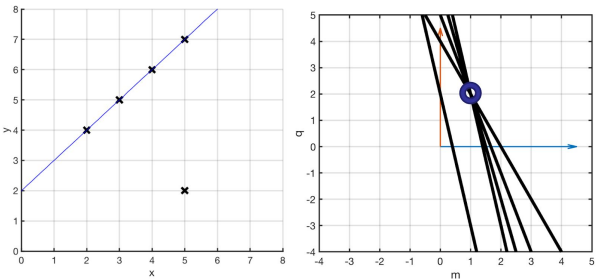
Idea: map points from the image space to the parameter space of the model.

Every point votes for all the curves that could have generated it by filling an accumulator in the parameter space. Extract curve with more votes.

Classical $y=mx+q$ parametrization is not convenient since slope approaches infinity when the line approaches vertical direction. Normal representation of the line: $r = x \cdot \cos \theta + y \cdot \sin \theta$

Algorithm:

1. Initialize $H[r, \theta] = 0$
2. for each edge point $p=(x,y)$ in the image
 - a. for $\theta=0$ to π
 - i. $r = x \cdot \cos \theta + y \cdot \sin \theta$
 - ii. $H[r, \theta] += 1$
3. find (r, θ) for which $H[r, \theta]$ is maximum
4. the detected line is given by $r = x \cdot \cos \theta + y \cdot \sin \theta$



7. Point Features

Edge detectors perform poorly at corners, but corners are important to detect because they provide *repeatable points for matching*.

Finding a corner: gradient is ill defined at the corner, while in the region around it the gradient has two or more different well-defined vectors.

Corners and gradient

Corner point exhibits strong rapid changes in image intensities. For a small region around point x_0 we can consider Taylor expansion of function $I(x,y)$ and express change of intensity as function of the image gradient and a displacement vector h .

We are not interested in the sign of the variation, only in the magnitude. So we can compute the square of it.

$$I(x_0 + h) \approx I(x_0) + \nabla I(x_0)^T h$$

$$I(x_0 + h) - I(x_0) \approx \nabla I(x_0)^T h \quad (I(x_0 + h) - I(x_0))^2 \approx h^T \nabla I(x_0) \nabla I(x_0)^T h$$

To be more resilient to noise we can compute intensity difference by averaging over a region centered at x_0 . Then we can apply the Taylor expansion again and call the function $E(x_0)$

$$E(x_0) = h^T \left(\sum_{x \in \Omega_{x_0}} w(x - x_0) \nabla I(x) \nabla I(x)^T \right) h \quad C = \sum_{x \in \Omega_{x_0}} w(x - x_0) \begin{bmatrix} I_u^2(x) & I_u(x)I_v(x) \\ I_v(x)I_u(x) & I_v^2(x) \end{bmatrix}$$

$E(x_0)$ can thus be written as $E(x_0) = h^T C h$. The summation can be moved inside the matrix, producing the second moment matrix (we discard weights for clarity)

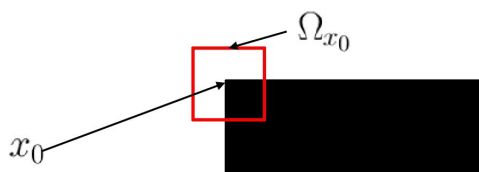
Properties:

- depends on first order derivatives
- is symmetric
- is positive semi-definite (both eigenvalues ≥ 0)
- each element is obtained as sum over a small region around x_0

$$C = \begin{bmatrix} \sum I_u^2 & \sum I_u I_v \\ \sum I_v I_u & \sum I_v^2 \end{bmatrix}$$

Simple case

Image intensity changes either in x or y direction, but not both



$$C = \begin{bmatrix} \sum I_u^2 & \sum I_u I_v \\ \sum I_v I_u & \sum I_v^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

When x_0 is at a flat region, we expect $\lambda_1 = \lambda_2 = 0$

When x_0 is at an horizontal edge, we expect $\lambda_1 = 0, \lambda_2 \gg 0$

When x_0 is at an vertical edge, we expect $\lambda_1 \gg 0, \lambda_2 = 0$

When x_0 is at a corner, we expect $\lambda_1 \gg 0, \lambda_2 \gg 0$

General case

In the general case, $\sum I_u I_v$ is nonzero. Since C is symmetric, we can

use SVD and obtain
 $C = R \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} R^T$ where R is a rotation matrix and λ_1, λ_2 are the singular values of C ,
i.e. the square root of the eigenvalues of $C^T C$

Since the rotations do not change the magnitude of h , the singular values of C can tell us if x_0 is in a flat region, edge or corner.

Harris corner detector

Analyzing singular values of C requires computation of SVD at each pixel \rightarrow expensive

Harris proposes following function as a corner response: $R(x_0) = \det(C_{x_0}) - k \text{trace}^2(C_{x_0})$

Also, we can prove that $\text{trace}(C_{x_0}) = \lambda_1 + \lambda_2$ and $\det(C_{x_0}) = \lambda_1 \lambda_2$

Therefore $R(x_0) \gg 0$ on a corner, $R(x_0) \ll 0$ on an edge.

The structure tensor is positive semi-definite.

Algorithm:

- Compute image gradient $I_u(x,y), I_v(x,y)$
- Compute matrix C for each pixel
 - 3 convolutions needed: $K \star I_u^2, K \star I_u I_v, K \star I_v^2$
 - convolution kernel K is usually gaussian and determines the scale of the corner
- Compute Harris response for each pixel
- Threshold the result and (optionally) perform non-maxima suppression

Harris corner detector works well in practice but is not invariant to scale. Convolution window size affects scale of the corner detected. To solve complex high-level CV problems we need more invariances and a way to distinguish and identify features.

Matching elements in two different images

Which elements of an image correspond to which elements in another image?

Common use cases:

- recognize and locate objects (may be occluded/deformed)
- track position and movement of object across a video sequence (easier if high framerate)

Tracking - SIFT algorithm

Provides **invariance to scale, rotation and illumination** while providing good localization!

Idea: image content is transformed into *local feature coordinates* that are invariant to translation, rotation, scale, and other imaging parameters.

Advantages:

- locality: features are local \rightarrow robust to occlusion and clutter
- distinctiveness: individual features can be matched to a large database of objects
- quantity: many features can be generated, even for small objects
- efficiency: close to real-time performance

- extensibility: can be extended to wide range of differing feature types, with each adding robustness

Algorithm (SIFT)

Keypoint localization:

1. enforce invariance to scale: compute difference of Gaussian for many different scales; non-maximum suppression → local maxima are keypoint candidates
 - idea: find corners as in Harris, but achieve scale invariance
 - method: convolve with Difference of Gaussians (approximates LoG) to identify interesting image pixels; DoG performed at multiple resolutions (image pyramid with octaves) and the local maximum (in space and scale) is selected
2. localize corners: for each minimum, fit quadratic function; compute center with sub-pixel accuracy by setting first derivative to 0
3. eliminate edges: compute ratio of eigenvalues; drop keypoints for which ratio is larger than threshold

Signature computation

4. enforce invariance to orientation: compute orientation by finding the strongest gradient direction in the smoothed image (possibly multiple orientations); rotate patch so that orientation points upwards
 - gradient magnitude and orientation is calculated for each pixel in the keypoint region (region size depends on detected scale); orientation histogram used to choose the orientation (max value)
 - at the end of the process, each keypoint is characterized by coordinate (x,y), scale and orientation
5. compute feature signature (descriptor): compute a *gradient histogram* of the local image region in 4x4 pixel regions; orient so that the largest gradient points up.
Result: feature vector with 128 values (16 fields, 8 gradients)
 - the 16x16 window around each keypoint is rotated wrt the keypoint orientation
 - 16x16 window divided into 16 4x4 windows. For each a histogram of gradient orientation is formed. Histograms concatenated to produce a 16x8=128-values feature vector

Enforce invariance to illumination change and camera saturation

6. normalize the descriptor to unit length to increase invariance to illumination; then threshold all gradients to become invariant to camera saturation

Feature matching

We have extracted features and their descriptors. We now need to match features. Descriptors are key to establish the matches, plus some additional filtering may be used to enforce matching priors. Feature matches are then passed to the next stage of processing pipeline according to what we want to do.

Descriptors and matching

Designed such that Euclidean distances in feature space can be used to rank potential matches. SIFT descriptors are great at this.

Simple matching: set threshold (max. distance) and return all matches from other images within threshold. Problem: false positives and false negatives depending on threshold!

Nearest neighbor matching: some features may have no matches → NN matching may yield many false positives

Nearest neighbor distance ratio: heuristic. Easier to threshold.

Compare distance from the second nearest neighbor, possibly taken from the image that is known not to match the target.

$$\text{NNDR}(D_A, D_B, D_C) = \frac{\|D_A - D_B\|}{\|D_A - D_C\|}$$

D_A : target descriptor; D_B, D_C : closest two neighbors

Efficient matching

Once a matching strategy is chosen, we need to efficiently search for matching candidates.

- Bruteforce: compare all features against all other features
- Indexing: rapidly answer NN or range queries
- Hasing: map descriptors to fixed-size buckets based on some hashing function

Match verification and densification

Once we have some candidate matches, geometric alignment is used to:

- verify which matches are inliers/outliers
- densify candidate matches including all the other matches sufficiently close to the estimated geometric transformation

We usually use RANSAC

8. Flow and Tracking

Goal: track position and movement of an object across a video sequence. We would like to estimate the **2D vector field of velocities** of the image points induced by the relative motion between the viewing camera and the observed objects. This motion is the projection of the 3D velocity field onto the image plane, but the motion field cannot be really observed with a single camera.

Optical Flow: *observed 2D displacements of brightness patterns in the image*

Frame sequences modeled as function $E(x, y, t)$

Brightness constancy equation

Assumption: apparent brightness (or color) will remain constant during the movement.

$$E(x(t), y(t), t) = \text{constant}$$

Taking derivative wrt. t

$$\frac{dE(x(t), y(t), t)}{dt} = 0$$

Using the chain rule

$$\frac{\partial E}{\partial x} \frac{\partial x}{\partial t} + \frac{\partial E}{\partial y} \frac{\partial y}{\partial t} + \frac{\partial E}{\partial t} = 0$$

- $\partial E / \partial x$ and $\partial E / \partial y$ can be computed via convolution (spatial gradient of each image)
- we want to estimate $\vec{v} = \left(\frac{\partial x}{\partial t}, \frac{\partial y}{\partial t} \right) = (u, v)$
- $\partial E / \partial t$ image derivative across frames (pixel-wise difference between consecutive frames)

In vector form $(\nabla E)^T \vec{v} + E_t = 0 \rightarrow E_x u + E_y v + E_t = 0$

We have 3 known coefficients and 2 unknowns (u and v). This is the equation of a line!

Notice that u and v may not be recovered in every possible case, as there might be ambiguity if we look at each pixel.

Aperture problem

Component of the optical flow orthogonal to the spatial image gradient is not constrained by the image brightness constancy equation \rightarrow only the flow in the gradient direction (normal flow) can be determined

That's why we cannot compute the motion field out to the optical flow, because OF is unconditioned in many situations, it depends on what object is being observed.

Let's say we have a big rectangular object completely black and our image frame captures only a central part of it. If the rectangle moves up or down that movement cannot be detected because it's orthogonal to the gradient, and in the image frame would appear like nothing has changed.

Ex:



If we take a point on the rectangle A, the x component of the gradient will be non-zero while

$E_y = 0$. If we look at our brightness constancy equation with respect to that point:

$$E_x u + 0v + E_t = 0$$

In this case v does not matter and so whatever vertical movement of the point is not captured in this way, while we are able to detect horizontal movement (in fact is parallel to the direction of the gradient).

Optical Flow vs. Motion Field

Optical flow: approximation of the motion field under assumptions of:

- Lambertian surfaces (even if a point is moving, its brightness does not change)
- Point-wise light source at infinity
- No photometric distortion

The error is:

- small at points with high spatial gradient
- exactly zero if the brightness gradient is parallel to the object motion

Computing the optical flow

- Differential techniques (e.g. Lukas-Kanade):
 - based on spatial and temporal variations of image brightness at all pixels
 - solve systems of partial differential equations (second and higher order derivatives of image brightness)
 - compute dense flows
- Matching techniques (e.g. KLT)
 - OF is estimated by feature matching only on a sparse subset of image points
 - compute sparse flows
 - idea: find corners in the first image, search for corresponding patches in 2nd image

Lukas-Kanade flow (differential technique)

- based on least-squares fitting of the flow parameters
- not iterative → fast and less biased by possible discontinuities of the motion field
- only 1st order derivatives → less sensitive to noise

$$E_x u + E_y v + E_t = 0$$

One equation per pixel is not enough → additional assumption: optical flow is well approximated by a constant vector within any small patch of the image plane.

Patch Q of size $N \times N$. If we suppose that (u, v) is constant for every pixel in Q , we obtain a system of 25 equations in 2 unknowns.

$$\operatorname{argmin}_{u,v} \sum_{\mathbf{p}_i \in Q} (E_x(\mathbf{p}_i)u + E_y(\mathbf{p}_i)v + E_t(\mathbf{p}_i))^2$$

$$\text{Or, in matrix form: } \operatorname{argmin}_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|^2$$

where

$$\mathbf{X} = \begin{pmatrix} E_x(\mathbf{p}_1) & E_y(\mathbf{p}_1) \\ E_x(\mathbf{p}_2) & E_y(\mathbf{p}_2) \\ \vdots & \vdots \\ E_x(\mathbf{p}_{N^2}) & E_y(\mathbf{p}_{N^2}) \end{pmatrix} \quad \mathbf{y} = - \begin{pmatrix} E_t(\mathbf{p}_1) \\ E_t(\mathbf{p}_2) \\ \vdots \\ E_t(\mathbf{p}_{N^2}) \end{pmatrix}$$

Solving the least-squares problem depends on inverting $A = X^T X$

$$A = X^T X = \begin{pmatrix} \sum E_x E_x & \sum E_x E_y \\ \sum E_y E_x & \sum E_y E_y \end{pmatrix}$$

- A should be invertible ($\det \neq 0$)
- A should be well conditioned: large λ_1 and λ_1/λ_2 not too large \rightarrow both large

A is the structure tensor used in the Harris corner detector, and since we want both λ to be big, we are looking for corners (LK flow works best). Also, aperture problem disappears at corners \rightarrow *corners are a good place to compute the optical flow*

KLT algorithm

Combination of Lukas-Kanade (track patches from frame to frame) and Tomasi-Kanade (select good features).

Algorithm:

1. extract corners
2. for each corner, compute displacement using LK
3. store the displacement of each corner and update corner position
4. add more corners
5. repeat 2-4
6. return long trajectories for each corner point

Correlation-based techniques

Some techniques, instead of finding the displacements with LK, simply compute normalized correlation between patches centered around features of each image. The assumption is that in small periods of time, corners tend to remain corners.

Visual Odometry technique (correlation based)

- extract corners in first and second image
- extract image patches around each corner
- find matching pairs (a,b) where:
 - $a \rightarrow$ corner patch from first image
 - $b \rightarrow$ corner patch from second image
 - b is the best match for a (according to normalized correlation)
 - a is the best match for b
 - good heuristic to get approximate result of the linear assignment problem (given N agents and N tasks, assign exactly one task per agent such that overall cost is minimized)

Data association

How do we determine which new observation should be added to each track?

Idea: predict next position and take the observation closest to that prediction or restrict the search to a gating region

Tracking as inference problem

Interesting properties of a feature are modelled as a discrete time random variable.

True position (X) at each frame is not directly observable, but can be inferred by a sequence of noisy measurements (Y).

We can divide tracking in 3 steps:

1. prediction: $P(X_i | y_0, \dots, y_{i-1})$
2. data association: what are the measurements obtained in the i -th frame that can be used to obtain a better estimate of the current state X_i ?
3. correction (update): $P(X_i | y_0, \dots, y_{i-1}, y_i)$

Simplifications:

- only the last frame matters \rightarrow Markov property. Defines the *system model*
- measurements depend only on the current state. Defines the *measurement model*

Now the problem can be treated as a Bayes inference problem

Linear Kalman filter

Simplest scenario: system and measurement models are linear; noise is zero-mean Gaussian; PDFs are all Gaussian

System model: $\mathbf{x}_k = F_{k-1}\mathbf{x}_{k-1} + \mathbf{v}_{k-1} \quad \mathbf{v}_{k-1} \sim N(0, \mathbf{Q}_{k-1})$

Measurement model: $\mathbf{y}_k = H_k\mathbf{x}_k + \mu_k \quad \mu_k \sim N(0, \mathbf{R}_k)$

F: state transition matrix; can change among the steps

Q: covariance matrix of noise

H: matrix mapping current state \mathbf{x}_k to measurements

R: covariance matrix of measurement noise

Tracking

We check the Mahalanobis distance between a new measurement y and the predicted state mapped in the measurement space. We accept the measurement if it is below a certain threshold that defined an ellipsoidal gating region.

Global Nearest Neighbor

We could have multiple possible features/observations incorporated into track.

GNN: take feature maximizing a score against the track. E.g. $e^{-\text{dist}}$, similarity of appearance

Problem: we can have contention for the same observation (2 tracks "touching"). This is again a form of the Linear Assignment problem. M tracks, N features to assign.

Possible solutions:

- simplex method
- see problem as maximal matching in a weighted bipartite graph
- use approximate solutions

9. Projective Geometry and 2D transformations

Geometric primitives: basic building blocks used to describe 2D and 3D shapes. Projective geometry allows to describe transformations between them in a powerful generic way. Projective geometry will be extremely useful because it will allow us to describe the pinhole camera model with convenient techniques.

2D projective space

Imaging apparatus usually behaves like pinhole camera model \rightarrow many transformations can be described as projective.

$$\mathbb{P}^2 = \mathbb{R}^3 - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} \in \mathbb{P}^2, \quad w \in \mathbb{R} - \{0\}$$

Therefore:

- there are infinitely many ways to describe a point \mathbf{x} (*equivalence classes*)
- points in the Euclidean space are represented via equivalence classes in \mathbb{P}^2

The higher dimensionality allows us to "convert" non-linear operations (products) into linear operations (kind of like the kernel trick in SVMs)

From Euclidean to homogeneous: just add 1 as third component

From homogeneous to Euclidean: divide by w (if nonzero)

Points $[x \ y \ 0]^T$ are called *ideal points* or *points at infinity*. No equivalent inhomogeneous representation.

In \mathbb{P}^2 points and lines are represented in the same way. We can find the intersection \mathbf{x} of two lines \mathbf{u} and \mathbf{u}' using the cross product: $\mathbf{x} = \mathbf{u} \times \mathbf{u}'$. If the lines are parallel, the third coordinate is 0, i.e. they "meet at infinity"!

In the same way, the line \mathbf{l} passing through two points \mathbf{p}_1 and \mathbf{p}_2 is $\mathbf{l} = \mathbf{p}_1 \times \mathbf{p}_2$

We can normalize a line $\mathbf{l} = [n_1 \ n_2 \ d]^T$ so that $\sqrt{n_1^2 + n_2^2} = 1$

With a normalized line, $[n_1 \ n_2]^T$ is the line's normal and d is the distance from the origin. The point-line distance from a line \mathbf{l} and a point \mathbf{x} can be computed as $|\mathbf{l}^T \mathbf{x}|$

Duality principle

Role of points and lines can be interchanged in statements. For instance, the line-line intersection is defined as $p = l \times l'$ and the line passing through two points is defined as $l = p \times p'$. *To any theorem of 2D projective geometry there corresponds a dual theorem which may be derived by interchanging the roles of points and lines in the original theorem.* Usually this involves using the inverse of the matrices in these theorems.

Ideal points and line at infinity

The set of all ideal points lies on a single line, the *line at infinity*, $l_\infty = (0 \ 0 \ 1)^T$

Any line $\mathbf{l} = (a, b, c)^T$ intersects l_∞ in the ideal point $\mathbf{l} \times l_\infty = (b, -a, 0)^T$

Any line $\mathbf{l}' = (a, b, c')^T$ (parallel to \mathbf{l}) intersects l_∞ in the same ideal point $(b, -a, 0)^T$.

The inhomogeneous vector $(b, -a)$ is a vector tangent to l and orthogonal to the line normal (a, b) . The line at infinity can be thought of as the **set of directions of lines in a plane**.

Projective plane

2D projective space \mathbb{P}^2 can be seen as a set of rays in \mathbb{R}^3 . Set of all vectors $k(x_1, x_2, x_3)^T$ forms a ray through the origin. A ray through the origin in \mathbb{R}^3 is a point in \mathbb{P}^2 . A line in \mathbb{P}^2 defines a plane passing through the origin. Two non-identical rays lie on exactly one plane and any two planes intersect in one ray. Lines lying in the plane spanned by x_1 and x_2 represent ideal points, and that plane represents l_∞ .

Points and lines can be obtained by intersecting rays and planes passing through the origin with the plane $x_3=1$.

Conics

$$C = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \quad \text{such that } x \in \mathbb{P}^2 \text{ belongs to the conic iff } x^T C x = 0$$

Since multiplying C by any nonzero scalar does not affect the equation, C is an homogeneous representation of the conic and has 5 degrees of freedom.

Also for the equation above we have a dual version: a dual conic is represented by a matrix $C^* = C^{-1}$ (if C is nonsingular) defined by the set of all the lines tangent to the conic satisfying the equation $l^T C^* l = 0$

2D projectivities

A planar projective transformation is a linear transformation in \mathbb{P}^2 that can be represented by any non-singular 3×3 matrix H . Applied to points, the transformation is written as

$$\begin{bmatrix} X' \\ Y' \\ W' \end{bmatrix} = Hx = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ W \end{bmatrix}$$

For the duality principle, the same transformation can be applied to a line as $l' = H^{-T}l$

2D translation. 2 d.o.f., preserves orientation

$$\begin{bmatrix} X' \\ Y' \\ W \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ W \end{bmatrix}$$

2D rotation. 1 d.o.f., preserves lengths

$$\begin{bmatrix} X' \\ Y' \\ W \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ W \end{bmatrix}$$

2D rigid motion.

3 d.o.f., rotation+translation, preserves lengths

$$\begin{bmatrix} X' \\ Y' \\ W \end{bmatrix} = \mathbf{TR} \begin{bmatrix} X \\ Y \\ W \end{bmatrix}$$

2D similarity. 4 d.o.f., $s \neq 0$

rot + trans + isotropic scale

preserves angles between lines

$$\begin{bmatrix} X' \\ Y' \\ W \end{bmatrix} = \begin{bmatrix} s \cos \theta & -s \sin \theta & t_x \\ s \sin \theta & s \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ W \end{bmatrix}$$

2D affine transformation

non-singular linear transformation

followed by translation

6 d.o.f.

preserves parallelism of lines

$$\begin{bmatrix} X' \\ Y' \\ W \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ W \end{bmatrix}$$

2D projective transformations (homographies)

Projective transformations are also called


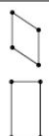


homographies (in their most general version).

Since we work in \mathbb{P}^2 , H is defined up to scale.

9 d.o.f.

Preserve straight lines

$$\begin{bmatrix} X' \\ Y' \\ W' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ W \end{bmatrix}$$

Group	Matrix	Distortion	Invariant properties
Projective 8 dof	$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$		Concurrency, collinearity, order of contact : intersection (1 pt contact); tangency (2 pt contact); inflections (3 pt contact with line); tangent discontinuities and cusps. cross ratio (ratio of ratio of lengths).
Affine 6 dof	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Parallelism, ratio of areas, ratio of lengths on collinear or parallel lines (e.g. midpoints), linear combinations of vectors (e.g. centroids). The line at infinity, l_∞ .
Similarity 4 dof	$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Ratio of lengths, angle. The circular points, I, J (see section 2.7.3).
Euclidean 3 dof	$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Length, area

3

Notice that, in the Euclidean transformation, $r_{11}=r_{22}$, vectors (r_{11}, r_{12}) and (r_{21}, r_{22}) are orthonormal and they compose a rotation matrix.

Projective transformations (homographies)

Homographies will be fundamental in the following chapters:

every time we will use **H** to refer to a matrix, we

will be talking about an homography.

General non-singular linear transformations of homogeneous coordinates.

Can be decomposed into chain of transformations

where:

- H_s is a similarity matrix
- H_A is an affine transformation where K is an upper triangular matrix normalized s.t. $\det(K)=1$
- H_p is a projective transformation

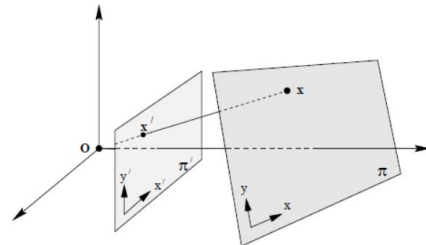
Notice that by "homography" we are referring to a product with a 3x3 matrix. Transformations such as the ones seen before are "special cases" of homographies.

Also notice that, since we are in projective space, we obtain a new vector (or, better, its equivalence class). We can also multiply the H matrix by whatever we want, the vector will just be scaled.

Projective vs. affine

Vector \mathbf{v} is not null in projectivity. It is responsible for non-linear effects of the projectivity and allows such transformation to model *vanishing points*

- Projective: ideal points can be mapped to finite points
 - simply a linear transformation of \mathbb{R}^3
 - if a coordinate system is defined in each plane and points x and x' are represented in homogeneous coordinates (i.e. rays), then the central projection mapping x to x' can be expressed as $x' = Hx$
 - if the coordinate systems are both Euclidean, then mapping defined by central projection is called *perspectivity*
- Affine: ideal points remain ideal



Spatial transformations on images

These projective transformations can be applied to the image domain to transform the geometry of the image plane. Given image $f(\mathbf{x}) = f(x, y)$ and a transformation function $s: \mathbb{R}^2 \rightarrow \mathbb{R}^2$, a spatial transformation changes the image as following $g(\mathbf{x}) = f(s(\mathbf{x}))$

This is different from intensity transformations: $f(\mathbf{x}) = H(f(\mathbf{x}))$

Also, spatial transformations are different from 2D projectivities (homographies) because in this case we are not limited to 3x3 matrices.

Forward warp

When we apply a transformation (e.g. rotation) a pixel $f(\mathbf{x})$ is copied to its corresponding location $\mathbf{x}' = s(\mathbf{x})$ in image $g(\mathbf{x}')$. Problem: \mathbf{x}' usually does not have integer value

Solutions:

- round the value and copy the pixel there \rightarrow cracks and holes
- distribute the value among its n-neighbors (weighted) \rightarrow aliasing and blur

Since both solutions don't provide good enough results, it is common to use inverse warp instead of forward warp.

Inverse warp

Each pixel in the destination image $g(\mathbf{x}')$ is sampled from the original image at $\mathbf{x} = s^{-1}(\mathbf{x}')$

Pro: no holes

Cons:

- transformation function must be invertible (but we already know it is nonsingular)
- point sampling may still occur at non-integer locations \rightarrow requires *interpolation*

Interpolation: estimate the values using the information from nearby samples

Nearest-neighbor interpolation: use the image value at the closest integer location.

Creates artifacts ("blocks") when we zoom

Bilinear interpolation: use the four points around $s^{-1}(\mathbf{x}')$ to get a better (weighted) estimation.

10. 3D Projective Geometry

Points and Duality

A point \mathbf{x} in \mathbb{R}^3 is represented in homogeneous coordinates as a 4D vector. The points at infinity have 0 as the last component and cannot be converted back into inhomogeneous coordinates.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3 \rightarrow \begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix} \in \mathbb{P}^3, w \in \mathbb{R} - \{0\}$$

In \mathbb{P}^3 points and planes are dual, the same way points and lines are dual in \mathbb{P}^2 .

3D planes

A plane in 3D Euclidean space can be represented as the locus of points $\mathbf{p}=(x,y,z)^T \in \mathbb{R}^3$ s.t.

$$\pi_1 x + \pi_2 y + \pi_3 z + \pi_4 = 0$$

In homogeneous coordinates, the same relation can be expressed as $\pi^T \mathbf{x} = 0$, π and $\mathbf{x} \in \mathbb{P}^3$

In a plane $\pi = (\pi_1, \pi_2, \pi_3, \pi_4)^T$, $\mathbf{N}=(\pi_1, \pi_2, \pi_3)$ defines the plane normal. If the vector is normalized s.t. $\|\mathbf{N}\|=1$, then π_4 is the plane distance to the origin.

3 non-collinear points are needed to define a plane. The best way to describe it is by stacking the points in a 3x4 matrix

$$\begin{pmatrix} X_1^T \\ X_2^T \\ X_3^T \end{pmatrix} \pi = 0 \quad \text{s.t.} \quad \begin{matrix} \pi \text{ is obtained (up to scale, since we} \\ \text{are in homogeneous coordinates)} \\ \text{the 1-dimensional right null-} \\ \text{the matrix.} \end{matrix} \quad \begin{pmatrix} \pi_1^T \\ \pi_2^T \\ \pi_3^T \end{pmatrix} X = 0$$

3D planes

Planes and points are dual \rightarrow 3 non-parallel planes define a point. The intersection point of the 3 planes can be obtained in a similar manner by computing the right null-space of the 3x4 matrix composed by stacking the planes

Projective transformations

Linear transformation in \mathbb{P}^3 that can be represented by any non-singular 4x4 matrix

$$\begin{pmatrix} \mathbf{A} & \mathbf{t} \\ V^T & v \end{pmatrix}$$

Where \mathbf{A} is a 3x3 invertible matrix, V^T and \mathbf{t} are 3D vectors and v is a scalar.

The transformation is up to scale \rightarrow 15 d.o.f.





Rigid motion

Rotation around an axis + translation. 6 d.o.f. (3 rotation, 3 translation). Important because it preserves distances, parallelism of planes and lines, and volume.

$$\begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}$$

Chasles' theorem: any particular translation and rotation is equivalent to a rotation around a screw axis together with a translation along the screw axis. The screw axis is parallel to the

rotation axis. It basically says that any rigid motion can be decomposed in a rotation around an arbitrary axis in space followed by a translation along that same axis

Group	Matrix	Distortion	Invariant properties
Projective 15 dof	$\begin{bmatrix} A & t \\ v^T & v \end{bmatrix}$		Intersection and tangency of surfaces in contact. Sign of Gaussian curvature.
Affine 12 dof	$\begin{bmatrix} A & t \\ 0^T & 1 \end{bmatrix}$		Parallelism of planes, volume ratios, centroids. The plane at infinity, π_∞ , (see section 3.5).
Similarity 7 dof	$\begin{bmatrix} sR & t \\ 0^T & 1 \end{bmatrix}$		The absolute conic, Ω_∞ , (see section 3.6).
Euclidean 6 dof	$\begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix}$		Volume.

Plane at infinity

$\pi_\infty = (0 \ 0 \ 0 \ 1)^T$. It contains all the directions (ideal points) $(D_1, D_2, D_3, 0)^T$

Two planes are parallel iff their line of intersection is on π_∞ . A line is parallel to another line, or to a plane, if the point of intersection is on π_∞ .

The plane at infinity is important because it remains fixed under an affine transformation but not under a general projective transformation (same behavior as line at infinity).

The plane at infinity is a fixed plane under the projective transformation H iff H is an affinity.

Note that:

- the plane is not fixed point-wise, it is just mapped to the same plane
- π_∞ may not be the only fixed plane

11. Pinhole Camera

Introduction

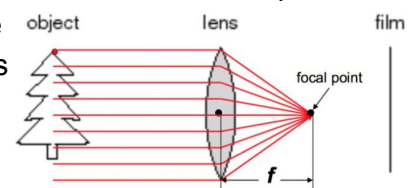
Imaging device receives light from all the points in a scene \rightarrow impossible to reconstruct as-is

Pinhole camera: put a barrier with a small hole (aperture) between the object and the sensor. Blurring is reduced. Idea comes from Leonardo's camera obscura. Note: objects appear smaller and upside down.

Cameras and lenses

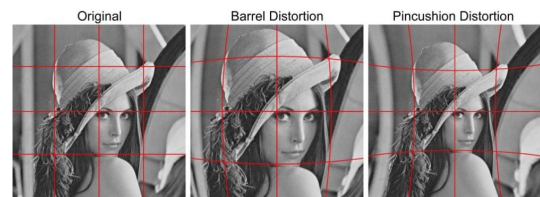
Pinhole size: big=blurring, small=less light and diffraction. Solution: use lenses. All parallel rays converge to one point located at the focal length f . There is a specific distance at which the objects are in focus: objects at other distances generate the so-called circle of confusion.

Rays are deflected from the lens according to Snell's law:



$n_1 \sin \alpha_1 = n_2 \sin \alpha_2$. Lenses have different refractive indices for different wavelengths → *color fringing*.

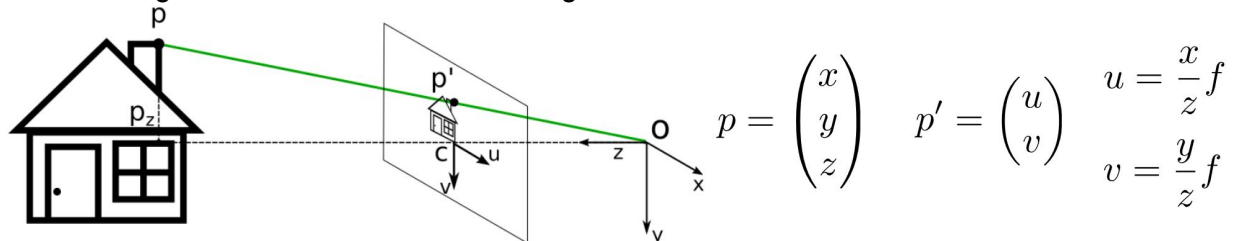
Imperfect lenses also cause radial distortion (barrel, pincushion).



Pinhole camera model and projective geometry

Image plane behind optical center → image upside down

Usually we consider virtual image plane in front of the center of projection. Considering similar triangles, we can obtain the following



In other words, we can exploit similar triangles in order to project a 3D point from the scene to the 2D plane. $\frac{pp_z}{p'c} = \frac{p_z O}{CO} \quad \frac{y}{v} = \frac{z}{f}$

The function is $E: \mathbb{R}^3 \rightarrow \mathbb{R}^2$, $(x, y, z) \rightarrow (xf/z, yf/z)$ and it is not linear

due to the division by z . We can make it linear by using

homogeneous coordinates, expressing it with a linear mapping $E: \mathbb{P}^3 \rightarrow \mathbb{P}^2$

By doing this, the division by z only occurs when we transform P' back into inhomogeneous coordinates.

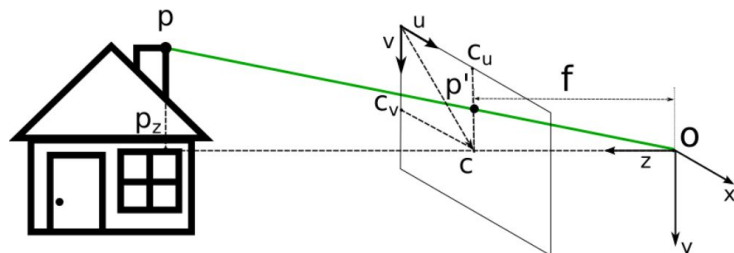
$$P' = \begin{bmatrix} fx \\ fy \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Projection matrix

Principal point

The image reference system is usually placed at the top-left corner of the image. After the projection, the coordinates of p' are expressed w.r.t. the *principal point* $C = (c_u, c_v)^T$.

A final translation with the vector (c_u, c_v) is required.



$$p' = \begin{bmatrix} fx_p + c_x z_p \\ fy_p + c_y z_p \\ z_p \end{bmatrix} = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

Intrinsic parameters

The nonzero 3x3 matrix in the projection matrix is called *matrix of the intrinsic parameters*.

It only depends on the camera's characteristics.

Camera pose

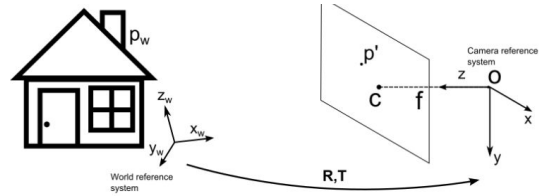
When dealing with multiple cameras it is common to represent points in a common world reference system. A rotation matrix R and a translation vector T express the rigid motion from a world reference system to the camera reference system (camera pose).

World to camera

To be projected, a point $p_w \in \mathbb{P}^3$ in the world reference system must first be transformed into the camera coordinate system. In 4D homogeneous coordinates we got

$$p = \begin{pmatrix} \mathbf{R} & \mathbf{T} \\ 0 & 1 \end{pmatrix} p_w$$

Extrinsic parameters
R and T are related to the camera's position and orientation \rightarrow *extrinsic* parameters.



$O = -RT$ is the position of the camera center with respect to the world reference system.

Complete projection

$$p' = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{R} & \mathbf{T} \end{pmatrix} \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} \rightarrow p' = \mathbf{K} \begin{pmatrix} \mathbf{R} & \mathbf{T} \end{pmatrix} \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix}$$

RT is a (3x4: 3x3|3x1) matrix which, given a point, maps it to the camera's reference system. Applying the projective transformation K represents the point in projective coordinates in the 2D space. In other words, the projection matrix $P = K[R|T]$ maps 4D points to 3D points. The point we obtain is in \mathbb{P}^2 . If we want to obtain the exact coordinates of the point in the image plane, we must convert it into inhomogeneous coordinates, dividing the first two components by the last one.

Finite projective camera

We consider that pixels may not be square ($s \neq 0$) and the produced image is not rectangular.

With K being an upper-triangular non singular matrix, we call $P=KRT$ a finite projective camera.

The f are focal lengths, the c are center coordinates.

A finite projective camera P has a 1-dimensional right null space because its rank is 3, whereas it has 4 columns. The **right null space** is the **homogenous representation of the camera center C**. (We don't have the cross product between 4D vectors, hence we rely on right null space instead.)

Row vectors of a projective cameras may be interpreted geometrically as world planes

- P^{3T} is the vector representing the **principal plane of the camera**, i.e. all the points imaged on l_∞ in the image. $x \in l_\infty \rightarrow x = (x \ y \ 0)^T$. A point $X \in \mathbb{P}^3$ imaged on l_∞ must be such that $PX = (x \ y \ 0)^T$ which implies $P^{3T}X = 0$
- P^{1T} and P^{2T} represent the planes passing through the camera center C and the Y and X axis respectively

$$P = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} (\mathbf{R} \mid \mathbf{T})$$

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} = \begin{bmatrix} P^{1T} \\ P^{2T} \\ P^{3T} \end{bmatrix}$$

Forward projection (3D point to 2D point)

A general projective camera **maps** a point $X \in \mathbb{P}^3$ to the image point $x \in \mathbb{P}^2$ according to

$x = PX$. Points $D = (x \ y \ z \ 0)^T$ on the plane at infinity are mapped to $x = PD = [M \mid p_4]D = M(x \ y$

$z)^T$ and are thus affected only by the first 3x3 submatrix of P : $P=KR \begin{bmatrix} I & t \end{bmatrix}$ where t is the translation vector.

In other words, $M = KR$: it does not depend on the translation, which makes it easy to stitch together panorama scenes in which we only move without rotating the camera!

Backprojection (2D point to set of 3D points)

Given a point x on the image, the set of points $X \in \mathbb{P}^3$ mapping to x is a ray in space passing through C . Let P^+ be the pseudo-inverse of P , defined as $P^+ = P^T(PP^T)^{-1}$. The point x back-projects to the line $P^+x + \lambda C$, $\lambda \in \mathbb{R}$. Indeed, $P(P^+x + \lambda C) = Px + \lambda PC = x$.

By doing so, P will get multiplied by its pseudo-inverse, thus resulting in the identity matrix I . Furthermore, since C (the camera center) is the right-null vector for P , λPC will become 0 . The final result is Ix , i.e. x . In summary, the last formula is basically the projection of the line back into the image, hence $x' = PX$. (considering a general point of the line)

Camera matrix decomposition

For a finite camera we have $P = K[R|t] = [KR|t] = [M|t]$. We can recover both K and R from P using the RQ-decomposition. In other words, given a matrix P , the 4th column (t) is the translation vector, while the first 3 columns can be decomposed in order to obtain the camera's intrinsic parameters K and the rotation matrix R .

Image of points on a plane

We want to image points X_π on a plane π . Since we can choose the world coordinate frame (extrinsic parameters) we choose a coordinate frame such that the plane π corresponds to the XY plane. Therefore all points X_π are of the form $X = (x \ y \ 0 \ 1)^T \in \mathbb{P}^3$

$$x = PX = \begin{pmatrix} p_1 & p_2 & p_3 & p_4 \end{pmatrix} \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} p_1 & p_2 & p_4 \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

$\begin{pmatrix} p_1 & p_2 & p_4 \end{pmatrix}$ is a 3×3 matrix \rightarrow it is an homography in \mathbb{P}^2 ! Therefore we can project points from a 2D projective space (e.g. a desk) into the 2D image plane.

The mapping between points $X_\pi \in \mathbb{P}^2 = (x \ y \ 1)^T$ on π is a homography $x = HX_\pi$ where $H = K(r_1 \ r_2 \ t)$, where the r_i are the first two columns of R and t is the translation vector.

The most general transformation that can occur between a scene plane and an image plane under perspective imaging is a plane projective transformation. If we have objects on a plane we "just" need to find the homography matrix H , it is not a full projection anymore! Also, if we know H , we can recover K , R and T ! We have r_1 and r_2 , we can compute the cross product to obtain r_3 , and we have $t \rightarrow$ we have everything we need.

Homography estimation

Given a set of 3D to 2D point correspondences it is possible to **estimate the plane projective transformation H** . We have $x_1 \dots x_n \in \mathbb{P}^2$, $x'_1 \dots x'_n \in \mathbb{P}^2$. We want to find H such that $x'_i = Hx_i$. Note that x'_i and Hx_i should be equal in \mathbb{P}^2 , that is, they must have the same direction but possibly different magnitude. (same equivalence class)

In vector form we can express the requirement of having same direction but possibly different

$$x'_i \times Hx_i = 0 \text{ magnitude as}$$

We end up with a system of **2 equations and 9**

unknowns for each point. Using 4 point-point

correspondences we obtain the set of equations $Ah=0$

where A is the 8×9 matrix obtained by stacking the two rows given for each point. A has rank 8, thus it has a 1-dimensional null space providing a solution for h . If more than 4 points are given, the problem can be solved

$$\begin{pmatrix} 0^T & -w'_i x_i^T & y'_i x_i^T \\ w'_i x_i^T & 0^T & -x'_i x_i^T \end{pmatrix} \begin{pmatrix} h^1 \\ h^2 \\ h^3 \end{pmatrix} = 0$$

$$\begin{aligned} &\text{minimize } \|Ah\|^2 \\ &\text{s.t. } \|h\| = 1 \end{aligned}$$

as the minimization, whose solution is the **unit singular vector corresponding to the smallest singular value of A**.

DLT algorithm

Minimize algebraic distance $\|Ah\|^2$. This quantity is not geometrically or statistically meaningful. The solution obtained is then used as a starting point for a non-linear minimization of a geometric or statistical cost function.

<u>Objective</u>
Given $n \geq 4$ 2D to 2D point correspondences $\{x_i \leftrightarrow x'_i\}$, determine the 2D homography matrix H such that $x'_i = Hx_i$.
<u>Algorithm</u>
(i) For each correspondence $x_i \leftrightarrow x'_i$ compute the matrix A_i from (4.1). Only the first two rows need be used in general.
(ii) Assemble the $n \times 9$ matrices A_i into a single $2n \times 9$ matrix A .
(iii) Obtain the SVD of A (section A4.4(p585)). The unit singular vector corresponding to the smallest singular value is the solution h . Specifically, if $A = UDV^T$ with D diagonal with positive diagonal entries, arranged in descending order down the diagonal, then h is the last column of V .
(iv) The matrix H is determined from h as in (4.2).

Geometric distance

We want to minimize the transfer error, i.e. the Euclidean distance in the second image between x'_i and Hx_i , i.e. we minimize $\sum_i d(x'_i, Hx_i)$ where d is the non-linear function returning the squared Euclidean distance between a and b converted to inhomogeneous coordinates.

Symmetric transfer error

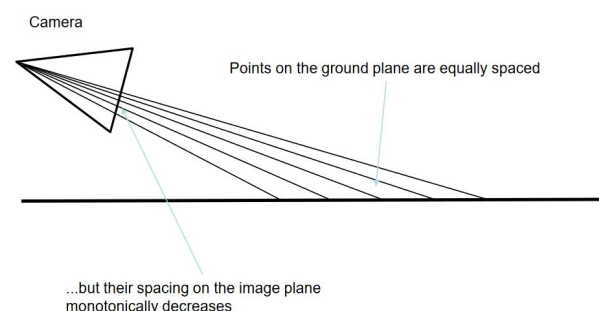
Transfer error assumes that only x'_i has some measurement error and the geometric distance is minimized in just one of the two images. In the more realistic case where image measurement errors occur in both the images, it is preferable that errors be minimized in both images. The symmetric transfer error results in the following:

minimize $\sum_i d(x'_i, Hx_i) + d(x_i, H^{-1}x'_i)$ which requires a non-linear least squares minimization

Vanishing points and lines

In perspective projection the image of an object that stretches off to infinity can have finite extent.

The vanishing point of the line representing the ground plane is obtained by **intersecting the image plane with a ray parallel to the ground plane through the camera center**.



Points at infinity give us much information about the camera's position, as they remove some degrees of freedom from the computation: the homography depends only on the rotation, instead of relying on the translation with respect to the ground plane. (if I look at the horizon from the first and the last floor of a building, it is at the same level at infinite distance)

Vanishing point

Consider a line in space passing through a point A , with direction $D=(d^T, 0)^T$. Line points can be expressed as $X=A+\lambda D$, $\lambda \in 0 \dots \infty$ (λ makes us move along the line).

With a projective camera $P=K(I \mid 0)$ each line point is imaged as: $x=PX=PA+\lambda PD=a+\lambda Kd$

The vanishing point v of the line is defined as $v = \lim_{\lambda \rightarrow \infty} (a + \lambda Kd) = Kd$. This means that v only

depends on the direction -and not on the position- because we are working in the projective space and therefore the magnitude is not relevant, yet for $\lambda \rightarrow \infty$ a becomes irrelevant.

Another way to look at the vanishing point is by considering the plane at infinity π_∞ , i.e. the plane of all the line directions. In other words, if a line has direction d , it intersects π_∞ in the point $X_\infty=(d^T, 0)^T$. If we project X_∞ to our image plane we obtain the vanishing point v that only depends on the direction and on the camera's intrinsic parameters.

$$v = PX_\infty = K \begin{pmatrix} I & 0 \end{pmatrix} \begin{pmatrix} d \\ 0 \end{pmatrix} = Kd$$

Vanishing lines

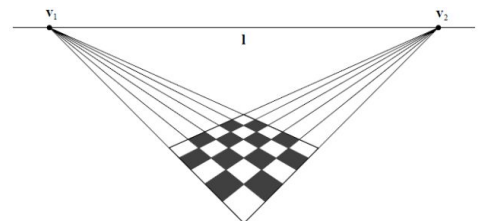
Parallel planes in 3-space intersect π_∞ in a common line. The image of this line is the **vanishing line of the plane**. Geometrically, the vanishing line is obtained by intersecting the image plane with a plane parallel to the observed plane but passing through the camera center.

A plane orientation relative to the camera (i.e. its normal n) can be computed from its vanishing line l s.t. $n=K^T l$. A plane can be rectified by knowing its vanishing line. This is done by constructing a homography matrix so that the plane is rotated parallel to the image plane. $H = KRK^{-1}$ where R is the rotation matrix such that $Rn=(0 \ 0 \ 1)^T$ (last column of R is n , the three lines must be orthonormal). Rectification takes an image and outputs a copy that looks as if we were looking at the scene from the top (e.g. the floor).

Computing vanishing lines

A vanishing line (like the horizon) is not always visible in a scene. A common way to compute it is the following:

1. identify 2 sets of parallel lines in the scene
2. compute the vanishing point for each set
3. the line passing through the two vanishing points is the vanishing line



Lens distortion

As we have seen, whereas the pinhole camera model describes the image projection as a linear operator (when working in projective spaces), lens distortion produces a non-linear displacement of points after their projection. In other words, lines are not projected into lines!

Modeling lens distortion

Lens distortion is usually modeled as a polynomial function of the radial distance from the lens center. This distortion affects the normalized projection coordinates of the points lying in the retinal plane. What is fundamental in this case is obtaining 5 distortion parameters:

$(k_1, k_2, k_3, p_1, p_2)$ which we will obtain by calibrating the camera.

Image undistort

Once the lens distortion parameters are known, an image can be *undistorted* as it would be taken with a perfect pinhole camera. Idea: compute inverse warp from the ideal undistorted image to the original distorted image according to the distortion function.

Algorithm:

1. for each pixel (u,v) in the target (undistorted) image
 - a. move backward in the projection chain to the retinal plane
 $(x' \ y' \ 1)^T = K^{-1} (u \ v \ 1)$
 - b. apply the radial distortion function to obtain $(\tilde{x}', \tilde{y}', 1)^T$
 - c. move forward in the projection chain to get the coordinates in the distorted (source) image: $(u' \ v' \ 1)^T = K(\tilde{x}', \tilde{y}', 1)^T$

12. Camera Calibration

Rationale

An accurate knowledge of the camera's intrinsic parameters is essential for any kind of quantitative geometric measurement in computer vision. Many different approaches exist.

Intrinsic parameters and calibration target

Estimating the camera's intrinsic parameters requires estimating:

- the upper triangular matrix K (2 focal lengths in diagonal, 2 center coordinates, s for non-squared pixels)
- 5 distortion parameters: $(k_1, k_2, k_3, p_1, p_2)$. Usually we have a lot of distortion especially when the focal length is short (e.g. wide angle cameras)

Many parameters can be obtained from the manufacturer of the camera/lenses, but in practice a more accurate estimate of the projection matrix $P = K[R|T]$ can be obtained by observing how the imaging model behaves when projecting a known object (**calibration target**). This object must:

- have distinguishable features
- have known 3D coordinates of each feature in the target reference frame
- lead to well-localized features in the image frame after projection

Usually the calibration process is performed by exposing the target multiple times to collect a set of 3D-2D correspondences.

Computing P

From a single target pose, the recovery of the projection matrix P is conceptually similar to computing a 2D homography. Let $X_1, X_2, \dots, X_n \in P^3$ the points in the reference 3D plane and $x_1, x_2, \dots, x_n \in P^2$ the points in the image 2D plane. We want to find a matrix P s.t.

$$x_i = PX_i \quad \forall i$$

For each point-point correspondence, we obtain a system of 2 equations with 12 unknowns (11 d.o.f.), therefore to find the matrix P we need at least 6 points (5 points plus x or y coordinate of one more point), usually more if we assume that images are noisy. In the latter case, we will adopt the usual approach of minimizing the algebraic (impose normalization

constraint like $\|p\|^2=1$, then use SVD) or geometric (DLT followed by a non-linear optimization) error. We usually rely on the latter, symmetrically to what we do with the homography estimation. The non-linear optimization step is fundamental to achieve a high degree of accuracy.

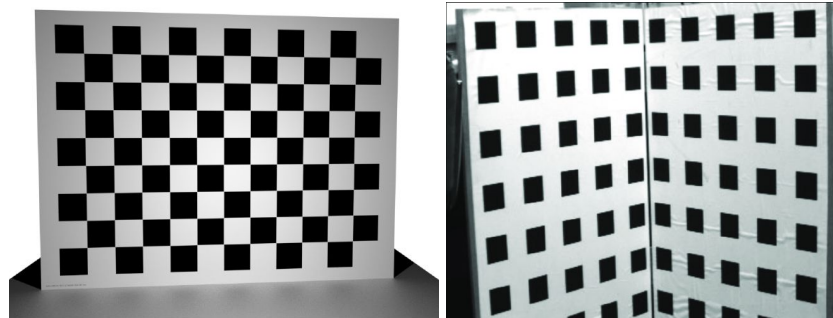
Once P has been obtained, we can decompose it as we have seen in order to find $K[R|T]$.

Degenerate configurations

There are *degenerate configurations* for which there exist infinite solutions for the equations, namely:

- when the camera and all the points lie on a *twisted cubic* (unusual)
- when the points all lie on the union of a plane and a single straight line containing the camera center (*planar target*)

If the known object's points lie on the union of a line and a single plane there is no single solution for the equations. One could use more than one plane, but this makes the "known object" difficult to (re)produce: the first chessboard is definitely easier to make than the second, "folded" one.



Zhang's camera calibration

Zhang's [2000] solution is to use a chessboard (i.e. a planar target) and recover the matrix K of the camera's intrinsic parameters *from the homographies*. We compute the view homographies of the target points and from them we recover K . We then refine this rough estimate with non-linear optimization.

Zhang's algorithm in detail

1. take M views of a picture with N points, changing the position of the picture wrt the camera
2. compute homographies H_i for $i=1, \dots, M$ (one homography per image) from the 2D target projective space to the image plane using DLT (minimize algebraic error) followed by non-linear optimization (minimize the geometric error).

Now we have M matrices of the form

$$H_i = \begin{pmatrix} | & | & | \\ h_{i,0} & h_{i,1} & h_{i,2} \\ | & | & | \end{pmatrix} = \lambda \mathbf{K} \begin{pmatrix} | & | & | \\ r_{i,0} & r_{i,1} & t_i \\ | & | & | \end{pmatrix} \quad \mathbf{K} = \begin{pmatrix} \alpha & \gamma & u_o \\ 0 & \beta & v_o \\ 0 & 0 & 1 \end{pmatrix} \text{ is the upper-triangular}$$

where K is the same for each H_i , because the camera is just being moved but the intrinsic parameters remain the same, while t and r depend on the camera's pose (translation and rotation). Also recall that the third column of R can be obtained as the cross product of the other two.

3. Factor out K from all homographies adding constraints for the matrix R , i.e. that R is orthonormal (columns $r_{i,0}, r_{i,1}, r_{i,2}$ are orthogonal $r_0^T r_1 = r_1^T r_0 = 0$ and of unit length $r_0^T r_0 = r_1^T r_1 = 1$).

Using some algebra:

$$\begin{aligned} h_0 &= \lambda \mathbf{K} r_0 \\ h_1 &= \lambda \mathbf{K} r_1 \\ \mathbf{K}^{-1} h_1 &= \lambda r_1 \\ \mathbf{K}^{-1} h_0 &= \lambda r_0 \end{aligned} \quad \begin{array}{l} \swarrow \text{Since:} \\ (AB)^T = B^T A^T \\ \nwarrow \end{array}$$

$$\begin{aligned} h_0^T \mathbf{K}^{-T} &= \lambda r_0^T \\ h_1^T \mathbf{K}^{-T} &= \lambda r_1^T \end{aligned}$$

Substitute the result into the orthonormal constraints and obtain

$$\begin{aligned} h_0^T \mathbf{K}^{-T} \mathbf{K}^{-1} h_1 &= 0 \\ h_1^T \mathbf{K}^{-T} \mathbf{K}^{-1} h_1 &= h_1^T \mathbf{K}^{-T} \mathbf{K}^{-1} h_1 \end{aligned}$$

4. Knowing K 's shape and that $\mathbf{K}^{-T} \mathbf{K}^{-1} = \mathbf{B}$ is a symmetric matrix, we can rewrite the conditions using the vector $b = (B_0, B_1, B_2, B_3, B_4, B_5)^T$ of elements of B . Using the function below we obtain a system of 2 equations and 6 unknowns for each homography, leading to an overdetermined system of $2 \cdot M$ equations and 6 unknowns of the form $Vb = 0$.

$$h_p^T \mathbf{B} h_q = v_{p,q}(\mathbf{H}) b$$

$$v_{p,q}(\mathbf{H}) = \begin{pmatrix} H_{0,p} \cdot H_{0,q} \\ H_{0,p} \cdot H_{1,q} + H_{1,p} \cdot H_{0,q} \\ H_{1,p} \cdot H_{1,q} \\ H_{2,p} \cdot H_{0,q} + H_{0,p} \cdot H_{2,q} \\ H_{2,p} \cdot H_{1,q} + H_{1,p} \cdot H_{2,q} \\ H_{2,p} \cdot H_{2,q} \end{pmatrix}^T$$

2x6 matrix

$$\begin{pmatrix} v_{0,1}(\mathbf{H}) \\ v_{0,0}(\mathbf{H}) - v_{1,1}(\mathbf{H}) \end{pmatrix} \cdot b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

6 coefficients

6-dimensional row vector obtained from an estimated homography \mathbf{H}

5. We solve the system as a least-squares problem using Singular Value Decomposition to obtain the matrix B that best satisfy the constraints we set on matrix R:

$$\min \|Vb\| \text{ s.t. } \|b\| = 1$$

6. Now that we have B we know that $B = \lambda K^{-T} K^{-1} = (\sqrt{\lambda'} K^{-T})(\sqrt{\lambda'} K^{-1})$ and since K is upper triangular, K^{-T} and K^{-1} are lower and upper triangular. We can thus use the Cholesky Decomposition of B to obtain $\sqrt{\lambda'} K^{-1}$, and normalize by $1/\sqrt{\lambda'} = \sqrt{\lambda'} \cdot K_{3,3}^{-1}$ since we know $K_{3,3}^{-1} = 1$

We now have a really good estimate of the matrix K!

Estimating the extrinsic parameter matrices R and T

Once K is known, the extrinsic parameters can be estimated for each pose by multiplying H by K^{-1} and fixing the scale λ considering that R is an orthonormal matrix.

We can compute R_i (which we will call Q few lines below) and T_i by using

$$r_0 = \lambda K^{-1} h_0, \quad r_1 = \lambda K^{-1} h_1, \quad t = \lambda K^{-1} h_2 = T$$

Then we scale by $\frac{1}{\lambda} = \|K^{-1} h_0\|$ because $\|r_0\| = \|r_1\| = 1$. Finally we obtain $r_2 = r_0 \times r_1$.

The problem, however, is that this method is not guaranteed to find a rotation matrix, as r_0 and r_1 may not be orthogonal. Zhang proposes to approximate R by finding the minimum

Frobenius norm $\min_R \|R - Q\|_F^2 \text{ s.t. } R^T R = I$, which is solvable using Singular Value

Decomposition. This leads to a matrix that is as close as possible to a "real" rotation matrix.

Q is the pseudo-rotation matrix composed by the values obtained above, and will lead to the best possible R using this optimization algorithm.

Distortion Correction

The process we have seen so far does not take into account distortion.

To fix this we need to use all the found parameters for all the images and perform a massive optimization step. In particular, we aim at minimizing the total reprojection error.

$P(a, w_i, X_j)$ is a non-linear function that projects X_j (point in the real world) onto the plane $Z=1$ to get x'_j , applies the distortion function mapping x'_j to its distorted coordinates and computes the affine 2D mapping Kx'_j .

Let $a = (\alpha, \beta, \gamma, u_C, v_C, k_1, k_2, k_3, p_1, p_2)$ and $w_i = (R_i, T_i)$, where all the R_i must be rotation matrices. The reprojection error to be minimized is expressed as the sum of all the squared differences between each point in each image and the same point's projection according to P.

$$E(a, \mathbf{w}) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \|x_{i,j} - P(a, w_i, X_j)\|^2$$

The orthonormal constraint for the matrices R_i is hard to impose while optimizing, and since R has only 3 d.o.f. we can use the "Rodrigues Formula", which is simply a vector $\rho = (\rho_x, \rho_y, \rho_z)$ of magnitude $\|\rho\| = \theta$ (equal to the rotation angle) and direction $\hat{\rho} = \rho / \|\rho\|$ which represents a rotation matrix with rotation angle θ and rotation axis $\hat{\rho}$. This allows us to use a 3x1 vector to represent R instead of a 3x3 matrix

Degenerate configurations

First of all, it is important to notice that the equations derived in step 3 of the algorithm depend on the properties of rotation matrices, thus translating the calibration target without rotating it does not give any useful information to help with the calibration.

Degenerate configurations may still occur, so it is useful to tilt the target after every shot, change the placement of the object in the image, check the reprojection error and remove bad pictures.

13. Epipolar Geometry

Rationale

We want to solve the following problem of 3D reconstruction:

Given a set of an object's 2D points in the image plane and the projection matrix of intrinsic and extrinsic parameters $P = K(R|T)$, project the points from the 2D image onto the 3D space.

Given a single image, for each 2D point there exists an **infinite** set of points in the 3D space that could have been mapped to that 2D point. We simply do not have enough information to "revert" the 3D→2D transformation. For example, with just one image we cannot distinguish a picture of an object from a picture of a picture of that same object.

Solutions:

- *single image*: additional information on the depth (time scene-to-camera); project light (laser scanner); properties of the optics (depth from focus); shading model (shape from shading) [not in the course]
- *multiple images*: Stereo Vision, Structure From Motion, ...
 - idea: compare images from different points of view. We assume the scene is not changing

Epipolar geometry

Epipolar geometry is the *projective geometry between two views of the same scene*; it depends on the intrinsic parameters and the relative position of the two cameras, and it's independent from the scene structure.

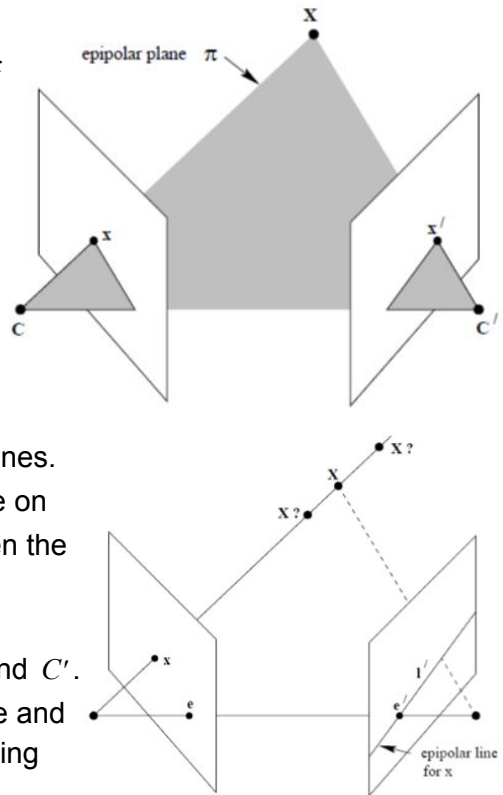
Let a point $X \in \mathbb{P}^3$ be imaged in two views
(corresponding points) $x, x' \in \mathbb{P}^2$; the relation between x
and x' is described by epipolar geometry.

The point X and the two camera centers C and C' are
coplanar in the *epipolar plane* π .

The two rays exiting from the centers C and C' and
passing through the projected points x and x' intersect
at X and lie on π .

The epipolar lines l and l' are the intersection between
the image planes and π ; x and x' lie on the respective lines.
Given a point x , the position of point x' is restricted to be on
the epipolar line for x , l' -- i.e. on the intersection between the
epipolar plane and the "second" image plane.

The *baseline* is the segment joining the two centers C and C' .
The *epipoles* are the points of intersection of the baseline and
the image planes. An epipolar plane is any plane containing
the baseline, and an epipolar line the intersection of an
epipolar plane and an image plane. All epipolar lines intersect the baseline.



Fundamental matrix

The Fundamental matrix is the **algebraic representation of epipolar geometry**.

It represents the fact that for each point x in one view there exists a corresponding line l' on
the other view on which x' must lie.

Geometrical derivation

Let π be **any** plane, and X any 3D point lying on that plane; x and x' are the images of X
in the two images (the plane π won't be used at any point in the derivation).

In this situation, x' must lie on the line l' corresponding to the image of the ray passing
through x and X . Moreover, there exist an homography H_π mapping each x_i to its
correspondent point x'_i through π , i.e.: $x'_i = H_\pi x_i$

A line passing through two points may be defined by the cross product and l' passes
through the epipole e' and x' : $l' = e' \times x'$

The cross product can be represented by a matrix multiplication, transforming $e' \in \mathbb{P}$ into
the rank 2 matrix

$$[e']_\times = \begin{pmatrix} 0 & -e'_z & e'_y \\ e'_z & 0 & -e'_x \\ -e'_y & e'_x & 0 \end{pmatrix}$$

Since $x' = H_\pi x$ and $l' = [e']_\times x'$, we obtain $l' = [e']_\times H_\pi x = Fx$;

We have therefore defined the Fundamental Matrix $F = [e']_\times H_\pi$ which is a rank 2 matrix.

Notice, as previously pointed out, that the plane π is not required in order for F to exist. It is used to define any point mapping through an homography between the first and the second image.

Algebraical derivation

This derivation explicitly relates F and P , giving more insight on the process.

The equation of the ray back-projected from the first camera is $X(\lambda) = P^+x + \lambda C$, where P^+ is the pseudo-inverse of the projection of the first camera (seen in pinhole camera lesson, when talking about backprojection), and C is center of the camera, or the right null vector such that $PC = 0$

In particular $X(\infty) = C$ is the projection of the first camera's center, and $X(0) = P^+x$ is the back-projection of x to X . When projected onto the second image P' they produce the epipolar line $l' = (P'C) \times (P'P^+x)$

The transformation P' is the mapping from the 3D space to the second image, and returns the epipole if multiplied with C , therefore:

$$\begin{aligned} P'X(\infty) \times P'X(0) &= \\ P'C \times P'P^+x &= \quad \text{because } P'C \text{ is the epipole} \\ e' \times (P'P^+x) &= \\ [e']_{\times}(P'P^+)x &= Fx \end{aligned}$$

where $P'P^+ = H_{\pi}$ is the explicit form of the point transfer homography between the two images and $[e']_{\times}(P'P^+) = F$.

Without loss of generality assume $P = K[I|0]$ and $P' = K'(R|T)$ (we can always translate the points so that the first camera is our reference frame). Then P^+ is a 4x3 matrix and C is a 4x1 vector

$$P^+ = \begin{pmatrix} K^{-1} \\ 0^T \end{pmatrix} \quad C = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Hence:

$$\begin{aligned} F &= [e']_{\times}(P'P^+) && \text{since } [e']_{\times} \text{ is the projection of the optical center of the first camera } C \\ & && \text{to the second camera } P' \\ &= [P'C]_{\times}P'P^+ && \text{since } C = (0 \ 0 \ 0 \ 1)^T \text{ we can compute the multiplications} \\ & && P'C = K'T \text{ and } P'P^+ = K'RK^{-1} \\ &= [K'T]_{\times}K'RK^{-1} && \text{because of a property of the cross product (in the appendix of the} \\ & && \text{textbook Hartley, A. Zisserman. Multiple View Geometry in Computer Vision)} \\ &= K'^{-T}[T]_{\times}RK^{-1} \end{aligned}$$

That is, the **Fundamental Matrix is the product between:**

- the inverse transposed **intrinsic matrix** of the second camera
- the cross product matrix of the **translation** of the **second camera wrt. the first**
- the **rotation of the second camera wrt. the first**
- and the inverse of the intrinsic matrix

This means **we just need to know the intrinsic and extrinsic parameters of the two cameras!**

Properties of the Fundamental Matrix

Correspondence condition

For any pair of corresponding points $x \leftrightarrow x'$ the following property holds: $x'^T F x = 0$

Since x and x' correspond, then x' lies on the epipolar line $l' = Fx$.

The dot product between a line and a point on the line is 0, therefore $x'^T l' = 0$.

Substituting l' with Fx we obtain $x'^T l' = x'^T Fx = 0$

This allows us to **obtain F using only point correspondences** (without knowing the projection matrices), and recover the two cameras' respective $P = K(R|T)$.

Other Properties of the Fundamental Matrix

1. The fundamental matrix is oriented: if the fundamental matrix for the pair of cameras (P, P') is F , then the fundamental matrix for the pair of cameras (P', P) is F^T .
2. For any two corresponding points $x \leftrightarrow x'$,
 $l' = Fx$ is the epipolar line on the second image
 $l = F^T x'$ is the epipolar line on the first image
3. For any point x , the epipolar line $l' = Fx$ contains the epipole e' .
Therefore e' is the left null-vector of F (and e is the right null-vector of F):
 $e'^T Fx = (e'^T F)x = 0 \quad \forall x \Rightarrow e'^T F = 0$
Similarly, e is the right null vector of F .
In other words, given F we can obtain the epipoles e, e' (via SVD)
4. F has 7 dof: 8 from the 3x3 matrix, minus 1 from the condition $\det(F) = 0$ because F has rank 2
5. The mapping induced by F is **not invertible**, as F maps points to lines

F produced by a pure translation

If the second camera $P' = K(R|T')$ is not rotated (i.e. $R' = I$) the movement is a pure translation, and the points in \mathbb{P}^3 move in straight parallel lines wrt the translation vector. The *imaged intersection* of those parallel lines is at the vanishing point $v = e$ in the direction of t . v is the epipole of both views and the imaged parallel lines are the epipolar lines!

This greatly simplifies computation: the objects in the scene move with inverse proportion to the distance (when moving in a train near objects move "fast" and far objects appear still). Algebraically it's expressed as $x' = x + Kt/Z$, with Z the depth component of the point.

Algebraically:

$P = K(I|0)$ and $P' = K(I|t)$: the second camera with same K only translates its position

$F = [e']_{\times}(P'P^+) = [e']_{\times}(KK^{-1}) = [e']_{\times}$: the epipole is the vanishing point of the lines.

Let $x = (x, y, 1)^T$ be an image point. The 3D inhomogeneous coordinates of the corresponding point $X = (X, Y, Z)^T$ depend on the unknown depth Z and are obtained by $ZK^{-1}x$.

After translating the point x we obtain:

$$x' = P'X = K(X+t) = x + Kt/Z$$

This means that **the amount of the translation depends on the magnitude of the translation t and on the inverse of the depth Z .**

We can simply notice this effect in real life: objects close to the camera move faster than objects far away from it, and points "infinitely" distant from the camera remain still.

Projective ambiguity

A camera pair (P, P') uniquely determines a fundamental matrix F , but from F we cannot obtain a unique camera pair: given a projective transformation H , the pairs (P, P') and $(PH, P'H)$ produce the same F . The only ambiguity is a right multiplication by an homography.

Essential Matrix

An *essential matrix* is a **fundamental matrix from which we remove the intrinsic parameters**; it's substantially a normalization of the coordinates.

Consider a camera matrix $P = K(R|T)$ with known K . By multiplying both sides by K^{-1} we obtain a projection on the "retinal plane", a plane with *unitary focal length* (normalized coordinates). By using $P = (I|0)$ and $P' = (R|T)$ we obtain the essential matrix $E = [T]_{\times} R$; the relationship between the fundamental and the essential matrices is $E = K'^T F K$.

E is a 3x3 matrix with 5 dof (3 from rotation angles, 3 from translation, -1 from scale ambiguity). E is an essential matrix iff the first 2 singular values are equal and the third is 0. This property is really important to estimate E given a set of point correspondences in the retinal plane. Given E , we can recover up to four (normalized) choices R and T , but only one of the choices triangulates the points in front of the camera (see later).

Normalized 8-point Algorithm - Computing F from points

Let us recall that the fundamental matrix defines the constraint $x'^T F x = 0$ between corresponding points x and x' in two images. We can therefore collect multiple correspondences to estimate F . Similarly to the DLT algorithm, we can estimate F from point correspondences exploiting the aforementioned relation $x'^T F x = 0$ to create a linear equation for each point and solve the system (obtained by stacking the equations) to find the unknown parameters of F .

$$x'x f_{11} + x'y f_{12} + x'f_{13} + y'x f_{21} + y'y f_{22} + y'f_{23} + x f_{31} + y f_{32} + f_{33} = 0.$$

Since F has 8 dof **we need at least 8 point correspondences**; with more than 8 points we can use **least square** approximation; F is a rank 2 matrix, and least squares may not enforce the orthonormality constraint, but we can enforce it later, analogously to Zhang's extrinsic parameter estimation using the Frobenius norm.

The solution is found by applying SVD obtaining $F = USV^T$ where S is the diagonal matrix $S = \text{diag}(r, s, t)$; then we substitute S with $S' = \text{diag}(r, s, 0)$ and obtain $F' = US'V^T$. F' minimizes the quantity below.

$$\text{minimize}_{F'} \|F - F'\| \quad \text{s.t. } \det(F') = 0$$

In practice we need to normalize the input data.

<u>Objective</u>
Given $n \geq 8$ image point correspondences $\{x_i \leftrightarrow x'_i\}$, determine the fundamental matrix F such that $x'_i{}^T F x_i = 0$.
<u>Algorithm</u>
(i) Normalization: Transform the image coordinates according to $\hat{x}_i = T x_i$ and $\hat{x}'_i = T' x'_i$, where T and T' are normalizing transformations consisting of a translation and scaling.
(ii) Find the fundamental matrix \hat{F}' corresponding to the matches $\hat{x}_i \leftrightarrow \hat{x}'_i$ by <ol style="list-style-type: none"> Linear solution: Determine \hat{F}' from the singular vector corresponding to the smallest singular value of \hat{A}, where \hat{A} is composed from the matches $\hat{x}_i \leftrightarrow \hat{x}'_i$ as defined in (11.3). Constraint enforcement: Replace \hat{F}' by \hat{F}' such that $\det \hat{F}' = 0$ using the SVD (see section 11.1.1).
(iii) Denormalization: Set $F = T'^T \hat{F}' T$. Matrix F is the fundamental matrix corresponding to the original data $x_i \leftrightarrow x'_i$.

Triangulation

The problem is the computation of the 3D position of a point X given x and x' .

We assume that errors may be present only in the measured coordinates x and x' , not in P and P' .

The naive triangulation done by back-projecting x and x' doesn't work because in presence of error the 3D lines may not intersect in a 3D point, whereas in 2D the lines will almost surely intersect; estimating a "best" solution for X is hence necessary.

In presence of errors the points x and x' will lie at some distance from the epipolar line (epipolar error).

Mid-Point Triangulation

An easy solution is to find the line perpendicular to the two lines and select the midpoint of the line.

Given points x and x' , and camera centers c and c' , we can write the lines exiting from the centers and passing through the points as $\alpha K^{-1}x + c = l_1$ and $\alpha' K^{-1}x' + c' = l_2$.

Then we need to find α and α' such that $\alpha K^{-1}x - \alpha' K^{-1}x' = c - c'$.

It's a system of 3 equations with 2 unknowns. Then the midpoint is found as $(l_1 + l_2)/2$.

In general this method gives **bad results, as the epipolar error is not equally distributed between the two views**; but it's easy to implement.

Linear Triangulation Method

Linear triangulation method is the most commonly used as it is a good tradeoff between simplicity and accuracy. It involves taking into account the formula of the projection of a point.

Consider the 2D point u and the matrix $P = (p_1 | p_2 | p_3)^T$ where p_i is the i -th row of P

$$x = PX = w(u \ v \ 1)^T;$$

then we can write each component of x as $wu = p_1^T X$, $wv = p_2^T X$ and $w = p_3^T X$;

then we take the equation $w = p_3^T X$ and substitute w in the other two equations obtaining the equations for points u and u'

$$u p_3^T X = p_1^T X, \quad v p_3^T X = p_2^T X \quad \text{for point } x = (u \ v)^T$$

$$u' p_3^T X = p_1'^T X, \quad v' p_3^T X = p_2'^T X \quad \text{for point } x' = (u' \ v')^T$$

Now we write the four equations as a single system $AX = 0$, and we look for non-trivial solutions for X ; there are two methods to find a solution:

1. Working in homogeneous coordinates we find $\min_X \|AX\| \text{ s.t. } \|X\| = 1$

This method assumes that X is not a point at infinity, and this causes numerical instability in case of points at or near infinity.

2. We set $X = (u \ v \ z \ 1)^T$ and reduce $AX = 0$ to a set of 4 non-homogeneous equations in 3 unknowns, solvable via Linear Least Squares.

This method cannot deal with points at infinity, but is *affine invariant*: if cameras are transformed by an affine transformation, we can apply the same transformation to X and obtain the same result. This may not be true in the first method, and is a nice-to-have because the triangulated point is independent on the reference system we choose. This method is more accurate, yet simple enough as it is still a linear function.

Use the first method when dealing with (possible) points at infinity, use the second method otherwise.

Iterative Linear Method

The problem of linear triangulation methods is that they minimize $\|AX\|$, which has no geometrical interpretation. From each equation $u p_3^T X = p_1^T X$ the error is

$\varepsilon = u p_3^T X - p_1^T X$. The geometric error is given instead by $\varepsilon' = u - p_1^T X / p_3^T X$, i.e. the distance between x and its projection \hat{x} .

Notice that with $w = p_3^T X$ we can obtain $\varepsilon' = \varepsilon/w = \varepsilon / p_3^T X$, but w depends on the 3D point X we want to find.

We therefore iteratively solve the problem starting with an initial estimate of X (for example $w = w' = 1$), then update the value until convergence, minimizing both the geometric and the algebraic error.

The iterative linear method is the most accurate as it reduces algebraic error and geometric error. It involves optimizing a nonlinear function in a linear way from a starting point.

Algorithm:

1. Set $w = 1, w' = 1$
2. Solve:
$$1/w \ u \mathbf{p}_3^T X = \mathbf{p}_1^T X \ 1/w$$
$$1/w \ v \mathbf{p}_3^T X = \mathbf{p}_2^T X \ 1/w$$
$$1/w' u' \mathbf{p}'_3^T X = \mathbf{p}'_1^T X \ 1/w$$
$$1/w' v' \mathbf{p}'_3^T X = \mathbf{p}'_2^T X \ 1/w$$
3. Let:
$$w = \mathbf{p}_3^T X \quad w' = \mathbf{p}'_3^T X$$
4. Return to step 2 until convergence (ie. the change in weights is small)

Bonus: information on the project

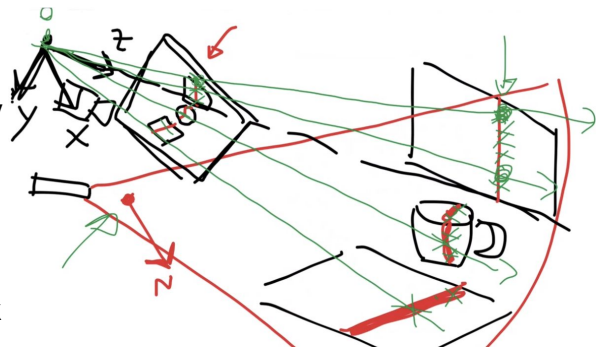
What we are given

- ground truth for calibrating the camera: chessboard
 - notice that images are distorted → calibrating the camera is important to be able to undistort images
- ground truth of objects in the videos: B/W rectangle
- set of images to calibrate the camera
- videos with different objects
 - these objects are not planar → the laser line generates a general curve on them

Intuition

Key idea: we can reconstruct a 3D object since we have two non-parallel planes with known objects (the rectangles) in them

Notice that the laser line is the only thing that moves across frames. This means that we only need to recognize the "wall" and "desk" planes and compute the homographies in the first frame of the video, since it will remain the same. Also, the laser conceptually projects a plane in space: when it intersects the wall/desk planes, it produces lines.



A plane in space can be represented easily with a point P and the normal vector N to that point: the plane is orthogonal to the normal vector N and passes through the point P , hence it is unique.

Pseudocode

Prerequisite

Calibrate the camera "offline" and save the values (so that we only calibrate it once and then always use the same values for all the runs of our code).

We are supposed to find point-point correspondences in the chessboards.

Only for the first frame

Find parameters of desk plane and wall plane. Detect the corners, find the homographies, split them into rotation and translation. Rectangles are perfectly known objects, since we have the ground truth. Notice that in the sampled images they are not rectangles anymore because of camera distortion.

Once we have the homographies (computed for the first frame) we also have the rotation matrices and the translation vectors. The third column in a rotation matrix (computed as the cross-product of the first two columns) is the normal to the plane, and the translation vector

is the point from which the normal exits. If we consider the point $(0\ 0\ 0\ 1)^T$ in both the wall's and the desk's rectangles, it is represented in the camera's reference system by the translation vector t of the homography, because we are multiplying $[R|T]$ by $(0\ 0\ 0\ 1)^T$; analogously, the third column of the rotation matrix is the normal to the plane, because we are multiplying $[R|T]$ by $(0\ 0\ 1\ 0)^T$, hence the result is the third column of the rotation matrix.

For every frame

1. detect red points
2. create the "exiting rays" from those points, find the laser plane
3. intersect those rays with the laser plane

Detecting the red points

In each frame we need to detect the laser line. We likely need some filtering. Notice that, intuitively, *for each line of pixels in each image we have at most one red pixel*. We can use non-maxima suppression, look for the pixel with higher red values, enforce line continuity, ... We should also restrict the search for red pixels in the area between the rectangles: there is no need to analyze the borders of the images. This also saves quite some computation.

Creating the "exiting rays"

First of all, we can describe a ray (line) in space in two ways:

- point P + direction d
- two points P_1 and P_2
 - in our case, one of the two points is the origin (center of the camera), i.e. $[0\ 0\ 0\ (1)]$
 - when x is a red pixel in the image plane, we can compute the point P_2 as $P_2 = K^{-1}x$ x has coordinates such as $[200\ 100\ (1)]$
 - the 3D line in space is the line $L(\alpha) = P_2 + \alpha(P_2 - [0, 0, 0])$, in other words it is the line connecting the camera center to P_2

Now we need to intersect the 3D line in space with the 3D laser plane in space.

Finding the laser plane

1. Identify black-bordered regions (step 1)
2. Detect red points in the regions (step 1)
3. For each red point, create corresponding exiting ray:
 - a. each ray exiting from the camera center, passing through the imaged red pixels in the rectangles in the image plane, intersects the wall and desk planes (represented with point+normal) in the red pixels. This means that we can obtain the 3D positions of the red pixels in the real world (desk plane and wall plane) by intersecting the exiting rays (from the camera center, passing through the imaged red pixels) with the wall and desk planes. We therefore obtain two 3D lines in space, one lying in the wall plane and one in the desk plane. We can then fit a plane between the two lines, because the points are coplanar by definition (they come from a single laser that projects a line). This plane is the laser plane in 3D!
 - b. The intersection between each ray and the laser plane is a 3D point composing the scene that we want to reconstruct. In other words, once we have the laser plane expressed with a $(point, normal)$ tuple, we can reconstruct the points by intersecting rays exiting from the camera with the laser plane
 - c. For fitting a plane between the two lines see [Least Squares Fitting of Data by Linear or Quadratic Structures](#)

