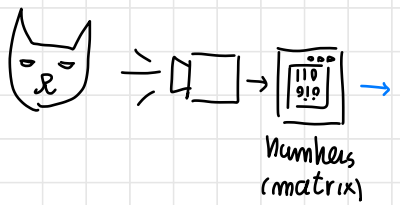
OpenCV

With Python

OPEN CV INTRODUCTION

computer vision → by cameras, systems see the world represented with "numbers"

how can the robot understand what it is looking?



- color (analytics)
- classify { cat? dog?
- detect { how many? where
- (segmentation...)

furthermore, with CV you can detect the players and track motion in a match video + track the ball... etc

and PREDICT its trajectory... based on its velocity and previous position it can predict future position

{ t → t+1 frame }

this is all in the data representing the scene (image)

+ with segmentation you can blur the background in instant portrait mode...

or replace the background completely / change colors... (filtering)

⇒ the tools to use computer vision algorithms

- Open CV
- scikit-image
- pillow

level 1: image foundament

at higher level Pytorch and Tensorflow (deep learning frameworks)
level 2: deep learning

PyTorch and TensorFlow depends on low level computer vision tools

on top of this:
- Hugging face = natural language processing, transformers
- YOLO = obj detection vision framework made from pytorch giving flexibility of detection, pose-est. classification etc.
- Detection 2 = obj detection
- FastAI = transformers models
- MMDetection
- ....

Level 3: higher tools to solve a problem

openCV we can use openCV to solve problems where RULES works better than learning

↓

to problem that can be solved with logic doesn't require complex machine learning solutions

↳ there are problems that can be solved with classical CV algorithms and doesn't require learning based solutions

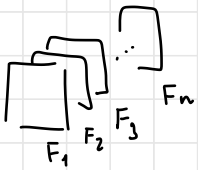
- Read / Write images / video
- edge detection
- tracking of objects
- image manipulation (crop, noising, ...)
- cross-platform (Linux, windows..) Run in C++, than PyThon
- Augmented Reality (detect keypoints)
- object detection
- Color filtering

⇑

With the help of deep learning you can generalize problem statement .... once complexity increases just openCV is not enough...

Video

A sequence of images (frames) ~ flipbook
- when played in rapid succession it create the illusion of motion



1 fps = 1 frame / second

t_0 : F_1 (for first instant we consider first frame)
t_1 : F_2
⋮
t_n : F_n

IF $fps = 1 \rightarrow t_{TOT} = n \times 1 = n$ secs (final output)

IF 6 fps is the "speed"

@ t_0: $F_1 \dots F_6$ all at the same second in sequence (smoother) $t_{TOT} = n/6$ sec

Standard fps:
- 24 fps = standard movies
- 30 fps = common TV shows and online videos
- 60 fps = sports and gaming

120/144 fps for higher requirements
... higher fps can be used to create slow motion video
320, 600, 1200 -> use it to create slow motion by dividing it in more seconds (it requires a powerfull system)

In Video usage: **Codec** is related to the different Video format (MP4, AVI, MKV..)

file format ~ codec

Algorithm used to compress/decompress digital media files (how we want to save the frames in the video... efficient way to store sequence of images..)

each codec use algorithm

different compressions...

FourCC = "4 character code"

terminology about implementation of video.

In openCV to use codec in video handling you need to specify/mention the FourCC e.g.

- MP4V := for general purpose video compression (mp4, avi)
- HEVC/H265 := better compression, high resolution (4K, 8K) (mp4, mkv)
- H.264 := Highly efficient, minimal quality loss (mp4, mkv)

↓

from your video device, if H.264 is older codec used by camera (H265 is the new)... efficient and low loss...
→ but H.265 is better for compression in a better way

Color Spaces (working with colors)

Working with colored images... the color of each pixel is fundamental

3 channels
Red, Green, Blue

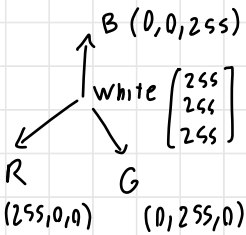
different color spaces exists and it is possible to convert between them

Camera → RGB output → conversion → BGR for Television

need for COLOR SPACE CONVERSION

RGB

represent images with 3 channels Red, Green, Blue



each color combin. of this 3 channels

ADDITIVE MODEL

White = R+G+B
bright pixel

each channel in (0,255) based on its intensity

RGB is how humans perceive colors and how digital devices works

Grayscale

single channel, in the shade of GRAY

{ White := 255
black := 0 }

0 - 255

again as intensity value. Im between its GRAY

⇓

- GRAYSCALE is used because of computational efficiency! Lighter memory and processing
- simplification when the COLOR is not relevant in our problem statement
- Saves space { RGB ~ 3MB / GRAY ~ 1MB

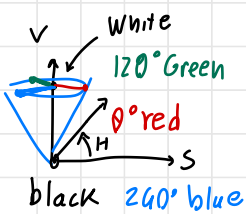
HSV

it is 3 channel

Hue := image color (color type)

Saturation := purity of color

Value := brightness of the color



based on HSV,
H represent the color as the circular value in the CONE
S is the cone direction while Value is vertical

↳

LAB

3 channel representation

L := brightness

A := Green to Red component

B := blue to yellow component

...

spherical represent

↳

BGR

the same as RGB but with opposite color order

default color space uses in openCV

HSV

HUE

H ~ color type 0°-360° represent the color
- 270° = pink
- 300° = red
- 60° = Green
- ...

Type of color based on the circular position in the color wheel ... type of color based on deg

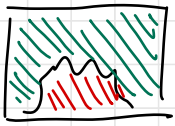
once chosen RED ⇒ if NOT "pure" red, to define the impurity I move by SATURATION

high SAT correspond to pure color, otherwise its a mixing color value
SAT ∈ 0-100%.
Hue ∈ 0-360°
Value ∈ 0-100% → finally value define the brightness of the color

Low SAT is white, and low value tends to black
once selected the Hue

HSV is used for:

- COLOR based segmentation

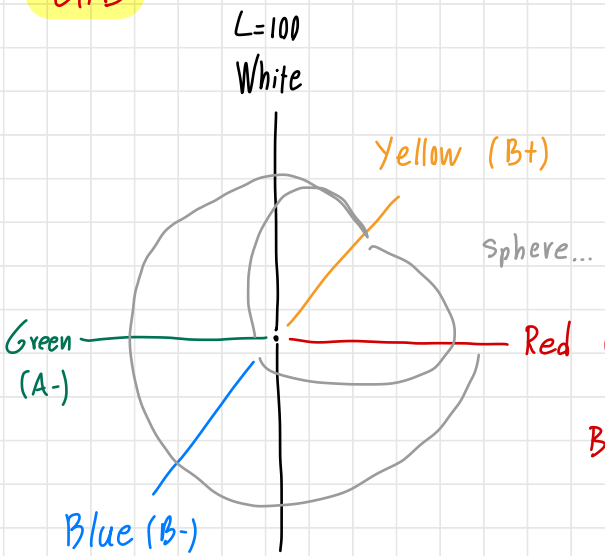


← Green background → to replace background I can use HSV for thresholding
⇓
I can decrease SATURATION to select the specific background color and then remove it with OpenCV and replacing

segment image based on color ←

HSV is intuitive for object tracking and segmentation by separating colors from intensity easily identify color ranges

LAB



the color is created based on

L ~ Light = brightness moving on vertical axis is same as (Value in HSV) respect to White / black

A ~ related to Red percentage between Red - Green

B ~ related to yellow - Blue value in the color

⇓

Im a 3D direction, all responsible for color

(While in HSV NOT all components influence color)

- LAB is alligned to how humans see the world color.
- Used for color corrections and Adjustments
  (the same can be done with HSV, but in HSV it can be more easily defined... In LAB each part is relevant)

BGR

OpenCV standard channel order...

- due to the History reason, OpenCV is OLD, at that time BGR was the standard

BGR ⟷ RGB CONVERSION

RGB
Input image
↓
OpenCV imread(·) → BGR
(it works in BGR space)

→ imshow(·) / imwrite(·) → works with BGR space!

If in OpenCV we convert manually to RGB

imshow(·), imwrite(·) convert again the channels swap channels! Show BGR back

→ (convert back to RGB) Show RGB image this is what human sees...

but in standard software, the RGB image is the expected one!

RGB → software processing → RGB We want output still in RGB

OpenCV make the conversion behind the scene

RGB → imshow(·) / imread(·) → BGR (ERROR!) ☹

expect BGR image and save it in RGB!

RGB input → OpenCV → BGR format used → Don't convert to RGB when doing processing in OpenCV!

COLOR THRESHOLDING

Technique to Filter a specific **Range of colors** in an image by defining upper/lower bounds in a color space.

↓

used to segment, replace, isolate a specific color in an image

(e.g. replace skin color in an image... Color thresholding)

FIRST: it requires the selection of a specific color space to use

RGB — } Brightness effect on color → RGB → Affects all three channels

HSV — } → HSV → Color is separate from brightness

Given input image in color space → understand brightness effect on a specific channel. I want to understand an change brightness

When working in RGB if I want to change ONLY Red brightness and I change its intensity it affects ALL COLORS

RGB cannot be used for particular targeting

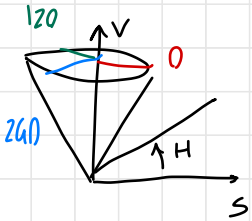
While with HSV I can pick a particular channel selecting the V (brightness), the color H is separated!

↓

meaning HSV is better for color targetting (color separated by brightness)

		SCALED! OpenCV
H ~ color	H ∈ (0, 360°)	(0, 179)
S ~ Vibrancy	S ∈ (0, 100%) →	(0, 255)
V ~ Brightness	V ∈ (0, 100%)	(0, 255)

HSV looks like an inverted cone



0-360° Hue in circular degree
S, V represented in percentage 0-100%.

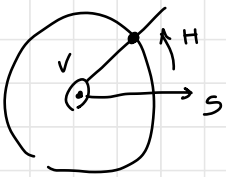
⇓

OpenCV you don't work in DEGREE and PERCENTAGE but with SCALED

⇓

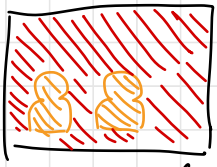
Hue ∈ (0, 179) 30° ≡ 15 in CV
Sat ∈ (0, 255) } better Calculation 100% ≡ 255 in CV
Val ∈ (0, 255) }

Top View



Color thresholding

Original Image

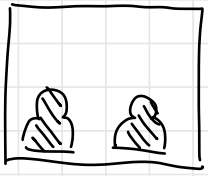


RGB channel we convert in HSV to target specific color in H channel

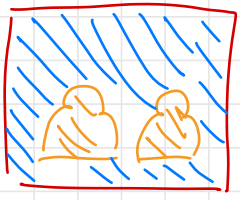
Color thresholding

replace bg with BLUE color

Mask of bg



(*) ⇒ binary mask created by threshold as BW (NOT Gray) is a segmented image YES/NO



Make pixel color blue if MASK > 0 else keep original

⇑

Once we have MASK a conditional change of original image is achieved

to CREATE the image mask

20	55	80
5	100	120
104	119	60

Hue

— to target GREEN (Hue ~ 120°)
S ∈ 0-100 %. := 100
V = 100
we target full GREEN with upper/lower bound

→

0	0	0
0	0	1
0	1	0

≡ binary mask

1 channel mask of BINARY VALUES

to target GREEN:
{ upper bound: 121°
{ lower bound: 119°

[119, 120, 121] target
with S = 255, V = 255

MASKING
when H channel is in this range this is converted as 1, else is 0

---

from BGR image input (openCV)

BGR of desired bg color → HSV (target in HSV)

original image → HSV → threshold ↗ apply mask and change selected masked pixels

conditional image modification ← to change original pixels.
using the BINARY MASK!
↓
I'll get an HSV image to reconvert in BGR for visualization

We work in HSV but visualize in BGR

Resizing, Scaling & Interpolation

Resize and scale images in OpenCV

It controls how pixels are adjusted during resizing...

Resizing and scaling affect image quality requiring the choice of the proper INTERPOLATION method!

- **Resizing** := adjusting image dimension to a specific (W, h)
  from default video quality size
  e.g. 640 x 480 (4:3) standard video resolution → in HD image 1280 x 720 (16:9)
  you need to adjust the INPUT to a new dimension (IMAGE RESIZING)
  in Resizing you change the aspect ratio ←

- **Scaling** := adjusting image dimension maintaining the aspect ratio
  e.g from HD 1280 x 720 (16:9) to Full HD 1920 x 1080 (16:9) ← same!
  ↑
  It can be used for up/down scaling to reduce memory usage/transmission purpose you downscale

+ you use scaling to NORMALIZE input size

IF in a dataset I have a big combination of data resolution, I want to "normalize my input size" → image resizing/scaling OR also for specific usage of images you may require upscaling

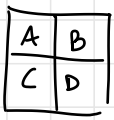
↓

What about INTERPOLATION ?
determines how pixel values are computed when decreasing/increasing image size

↳

Interpolation

Input Image



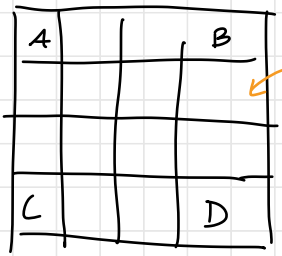
2×2

→ upscaling →

↓

scaling factor of 2

Output Image



4×4

how to fill empty pixels values?

↓

it is decided with the help of INTERPOLATION ALGORITHM

(the same problems holds when you downscale... you need to DROP some pixels and loose info)

e.g. Resizing 1067×1706 → 1067 ×1067

Resizing ... all information in Rectangle aspect ratio comes to new shape in square ratio

Scaling 1067×1706 → 533× 852

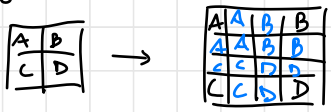
Scale by factor of 2

maintain the aspect Ratio! ⇒ scaling down with half size maintaining the shape of original image

Interpolation multiple methods are provided by Opencv

↓

- CV2. INTER_NEAREST (simplest method) each pixels in resized image get the value of NEAREST pixel in original one



based on Nearest pixel in the empty one created

- CV2. INTER_LINEAR

Uses average of the nearest four pixels to compute new pixel

CV2.INTER_CUBIC
nearest 16 pixels to compute new pixel value ... with smoother result.

CV2.INTER_AREA
calculate the average pixel value in area of original image that maps to a single pixel in resized image

CV2.INTER_LANCZOS4 ecc... uses a mathematical formula (LANCOZ KERNEL) to consider nearest 8x8 pixels neighborhood for each pixel. High-quality results but expensive

When resizing you test the different INTERPOLATION results and compare it to chose the most accurate

the selection depends on problem statement

- NEAREST ~ more pixelated using just nearest
- LINEAR ~ smoother, since it AVERAGE, smoothing the effect by averaging
- CUBIC ~ similar to LINEAR but smoother by using many pixels avg.
- AREA ~ similar to Nearest, since based on area
- LENCOZ4 ~ near CUBIC result, less edges ← to be used when computation is NOT problem

Flipping, Rotating & Cropping Images

required skill for Image processing and augmentation operation

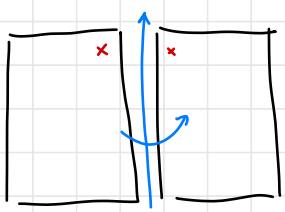
Image augmentation helps to create more generalized deep learning models...
↳ augmented images set helps in dataset with few images

basics of more advanced transformations as affine / perspective transformation

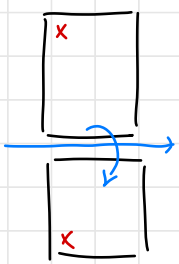
FLIP used to augment datasets (more complex methodologies may be used in more complex augmentation)

↓

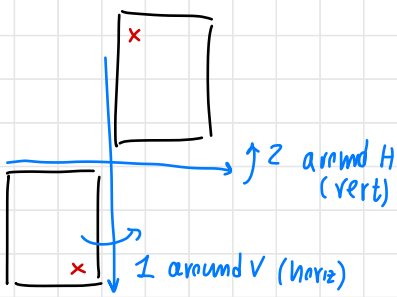
Here we cover the basics of flipping



horizontal flip
(flip around vertical axis)



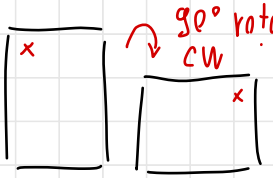
vertical flip
(flip around horizontal axis)



both flip HV

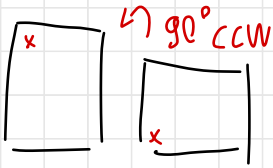
Rotate (basic ROTATION, without affine transformation)

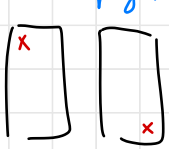
involves rotate by θ degree



→ simple fast transform
doesn't involves interpolation OR transformation matrics

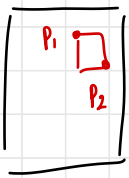
(affine / perspective transformations can perform more complex rotations)





CROPPING involves selecting a specific REGION of the image and removing the rest..

I require two pairs of coordinates P_1, P_2



$P_1 = (x_1, y_1)$
$P_2 = (x_2, y_2)$

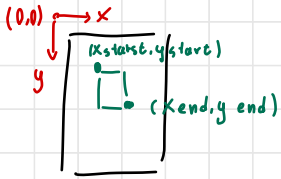
} array slicing over image →

extracted by numpy array slicing

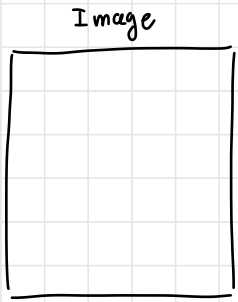
output cropped Image $((x_2 - x_1), (y_2 - y_1))$

1) first you define the region to crop with coordinates
2) then you extract the region by array slicing

(FOCUS ON REGION OF INTEREST) with OpenCV coordinates



COORDINATE SYSTEM IN openCV



In computer vision we perform many tasks on an image such as

1) cropping  select portion of interest

require to specify coordinates

2) draw shapes on image

 draw rectangle $[(x_1, y_1), (x_2, y_2)]$

require coordinates of the shape

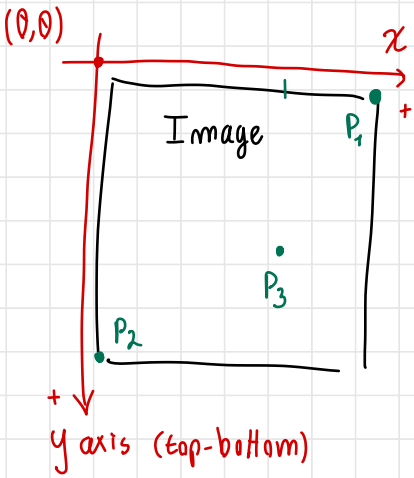
How openCV coordinates work? ←

3) place text in our image

write on a box the class label for example

need coordinates

⇒ CARTESIAN STYLE COORDINATE SYSTEM



origin of any image is in the top-left corner !

e.g $\begin{cases} w = 100 \\ h = 200 \end{cases}$ $\left[\begin{matrix} P_1 = (100, 0) \\ P_2 = (0, 200) \end{matrix}\right]$

$P_3 = (70, 160)$ e.g

DRAWING LINES AND SHAPES WITH OpenCV

↳ such as Rectangles, Circles and other Shapes...

this is fundamental to visualize data and annotate images + create visual graphical overlays ⇐ understand directly with VS code using OpenCV

+ adding text to images

(Not theory, practice with OpenCV)

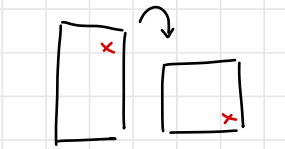
AFFINE & PERSPECTIVE TRANSFORMATION

fundamental image processing, allow to manipulate images in scaling, rotation, shearing, perspective

differently from flipping and minimal Rotation used for image augmentation → more complex augmentation and transformation require complete affine / perspective

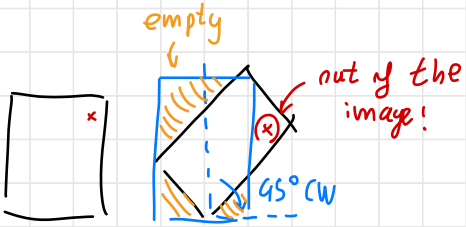
↓

used to correct distorsion, allegning images, prepare for further analysis



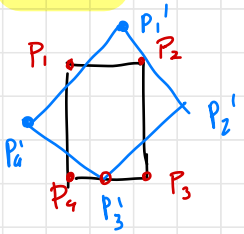
90 CW basic rotation

In a generic rotation with AFFINE TRANSFORMATION



this details needs to be considered and require AFFINE TRANSF. ⇐ When rotating in this ways some points remain out of the canva and others are empty

AFFINE



P_3 (right bottom) → P_3' (center bottom)

mapping $(x,y) \rightarrow (x',y')$

rely on a matrix transformation of the coordinates

the same for PERSPECTIVE

$(x,y) \ldots \; M \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \rightarrow (x',y')$

M: 2×3 matrix

find the new x, y with matrix multiplication

$$M = \begin{bmatrix} a & b & t_x \\ c & d & t_y \end{bmatrix}$$

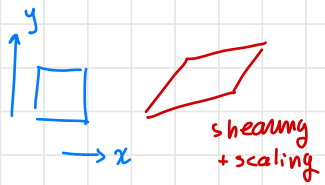
TRANSFORMATION MATRIX describing both affine and perspective

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} ax + by + t_x \\ cx + dy + t_y \end{bmatrix}$$

6 dof transform.

- a, b := control scaling and shearing along x axis
- c, d := control scaling and shearing along y axis



t_x, t_y := control translation components in x, y

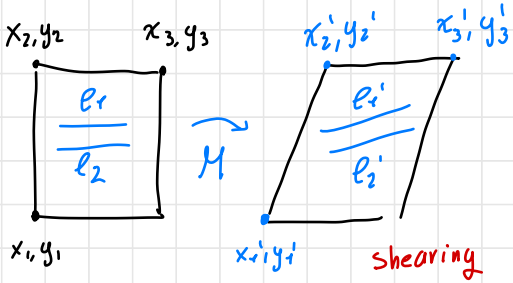
- particular a, b, c, d values control for different type of shearing and scaling based on the relative relation

Affine vs Perspective

Linear mapping method that preserves points, straight lines and planes

- straight lines remain straight
- parallelism preserved
- applies rotation, scaling, translation and shearing

↑ all this transformations are part of affine



$l_1 // l_2$ $\quad l_1' // l_2'$

parallelism preserved and remain straights

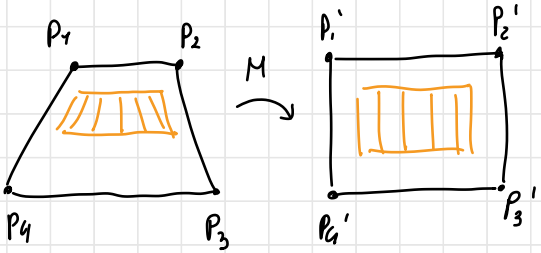
↑ uses the 3x2 matrix to calculate NEW pixel position based on linear relationship between three points

→ to transform based on affine you need three points correspondence

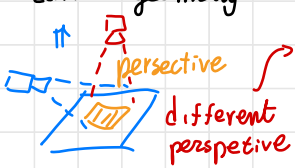
$P_1 \to P_1' \quad P_2 \to P_2' \quad P_3 \to P_3'$

three points fully descibe the affine (3×2 = 6 dof as H)

maps points from one quadrilateral in the INPUT to another quadrilateral in the OUTPUT image



e.g. image of parking lot, taking from different perspective lines doesn't seems parallel... by perspective transf. you recover correct geometry



perspective transform simulate effect of looking @ images from different angle

straight lines are preserved BUT not parallelism

e.g in Road photo

← you lose parallelism!

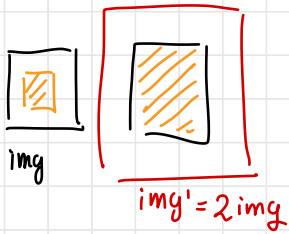
but straight is preserved

What about each component of M?
Relationship in M values...

SCALING := a, d represent scaling factor in x and y, while in pure scaling b = c = 0

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}$$

scaling factor of 2 on the image



for scaling, only a, d act on it!

ROTATION set the a, b, c, d values as

where θ is the rotation value

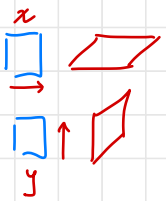
$a = \cos\theta \quad b = -\sin\theta \quad c = \sin\theta \quad d = \cos\theta \Rightarrow$

e.g with $\theta = 90°$: $\begin{bmatrix} 0 & -1 & t_x \\ 1 & 0 & t_y \end{bmatrix} = M$ (t_x: 0, t_y: 0)

map values of (x, y) to a 90° rotated

SHEARING (stretching images)

- shearing along x-axis (horizontal shear) $\Leftrightarrow b \neq 0$
- shearing along y-axis (vertical shear) $\Leftrightarrow c \neq 0$



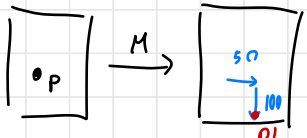
Keeping a, d the same and set either x OR y $a = d$ and change b OR c to shear

($a = d = 1$ keep constant scale)

TRANSCATION

by t_x, t_y regulate the motion in x, y-axis without affecting the shape

$$\begin{bmatrix} 1 & 0 & 50 \\ 0 & 1 & 100 \end{bmatrix}$$



move the image preserving the shapes

$P' = (P_x + 50, P_y + 100)$

PERSPECTIVE require more dof in the transformation matrix M to describe a perspective transform. M is 3×3

$$M = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

- a, b, c ← horizontal scaling, rotation, translation
- d, e, f ← vertical // // //
- g, h, 1 ← perspective component, skew and depth

IMAGE FILTERS IN OpenCV

an image filter is a set of rule applied to an image to extract / emphasize certain features.

FILTER
- SHARPENS
- SMOOTHENS
- ENHANCES
- ....

Many types

e.g. to enphasize edges in a photo we rely on an edge-detection filter while to blur background and smoothen pixels you use Gaussian-blur filter...

→ Filters depends on **KERNELS** (filters ≠ kernels)

↓

this is an NxN matrix that determines how filter operates

e.g

FILTER ⟶ Kernel (3×3)

{ Detecting Horizontal Lines } (HORIZONTAL FILTER)

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

the KERNEL is the matrix that describe our Filter

the Filter depends on a KERNEL, which is the CORE element of it. ← the KERNEL define the rules on how to extract features from images

How is it applied?

e.g Image (4x5) (3x3)

3x3 Filter on a 4x5 image... How?
the Filter is used s.t. it covers all elements of image

50	50	50	100	100
150	150	150	80	80
250	250	250	200	200
100	110	110	110	110

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

FILTER (kernel)

START

1)

50	50	50	100	100
150	150	150	80	80
250	250	250	200	200
100	110	110	110	110

element wise multiplication and summ

$50 \cdot (-1) + 50(-1) + .. + 250(1) = 600 = V_1$

START + 1 move Filter

2)

50	50	50	100	100
150	150	150	80	80
250	250	250	200	200
100	110	110	110	110

$V_2 = 50(-1) + .. + 200(1)$

When we reach the end, we move vertically... the filter stays on the image properly → We get a 3x2 image

With basic filtering
(4x5) x (3x3) → 3x2 out
input filter

{ applying padding and stride to modify filter behavior. }

adding pixels to fill required space to do matrix mult.

step size of filter step in the input image

← padding, stride is handled by OpenCV to get same shape of INPUT and output...

BUT the pixel intensity must be adapted to map to the limit 0÷255 of intensity

each image output pixel $\in [0, 255]$! 600 is NOT FEASIBLE!

When applying filters, to get consistent pixel values (0÷255), we can use two approaches

CLIPPING

IF $p_{new} < 0$: $p_{new} = 0$
ELIF $p_{new} > 255$: $p_{new} = 255$
ELSE $p_{new} = p_{new}$ ☒

(LIMIT CLIPPING)

NORMALIZING

$$p_{new} = \frac{p_{new} - p_{min}}{p_{max} - p_{min}} \times 255$$

(using the pixel intensity range)

(MIN-MAX NORMALIZATION)

↑ all this FILTERING operation is called CONVOLUTION

↳ "convolving" the FILTER on our image based on STRIDING and PADDING

{convolution ~ moving the filter over the image and performing and calculating new values}

- What about the specific KERNEL used?

Why extract horizontal edges? (Intuitively)

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

value 1 → low
value 2 → neutral (0)
value 3 → high

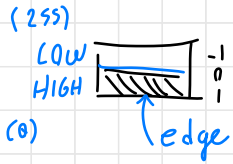
this multiplication depends on the Kernel...

1	50	50	50	100	100
2	150	150	150	80	80
3	250	250	250	200	200
	100	110	110	110	110

$(50+50+50)(-1) +$
$(150+150+150)(0) +$
$(250+250+250)(+1)$

When there is a big value, big presence of horizontal edge SINCE THERE IS A DIFFERENCE IN INTENSITY!

different intensity



edge identified by intensity switch when

get the transition when we pass from low to high

While $\frac{255}{200}$ there is a SHARP HORIZ EDGE...

prominant edge where there is a big pixel gradient!

other kernel to detect other edges exists...

how to create custom filters?

$$\begin{bmatrix} -1 & -1 & -1 \\ x & x & x \\ 1 & 1 & 1 \end{bmatrix}$$

by fine tuning the middle part, we compare intensity of middle pixels such as

IF using a big negative $x = -10, -20$

the difference between above and below pixels is relevant in the original image

to increase the difference by highlighting vertical gradient

↳ give big weights to extract horizontal line information

BLUR FILTERS: AVERAGE, GAUSSIAN, MEDIAN

more advanced filters available in Open CV { Average, Gaussian, Median } blurring

Blurring is a technique which reduces the sharp transitions in an image averaging pixel values in a local neighborhood

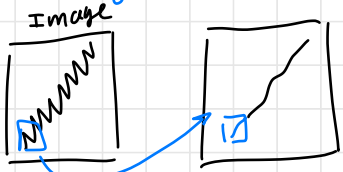
↓ Reducing Noise

"noise" refers to high variance 0-255 in pixels... we make it less noisy by blurring filtering

↓ smoothing images

give smooth images and reduce noise

(Avg Blurring)



averaging

better for smoothening

⇒ multiple methods for blurring →

- **Averaging** (in the output image) each pixel as average of surrounding pixels in Kernel window
- **Gaussian** weighted average, pixels closer to center have higher influence. Smoother + More Natural Blur
- **Median** better for denoise — Replace pixels with median of surrounding pixels. Remove SALT-AND-PEPPER Noise

{ noise of green-red etc... random points in image is properly removed by median blur }

↓ the Kernel size should be odd and require fine tuning to get proper desired blurring effect!

SOBEL, CANNY & LAPLACIAN EDGE DETECTION

important component in a computer vision application, helping to detect object boundaries and structural features in an image

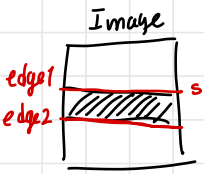
What is an edge ? → edges represents sharp changes in intensity (or color) in an image. Often corresponding to object boundaries, textures and other features

(Reason Why...)

Edge detection
- → Feature extraction
- → shape analysis → detect shapes, → contours, → Boundaries
- → Pre-processing

⇑

→ we detect edges for multiple reasons... such as feature extraction, shape analysis...



e.g

to focus on the person... ⇒
- NO interest on color / gradient
- Interested on image structure (e.g. to train a DL model that doesn't care on color)

CNN, AI model interested only on structural simple abstract information gotten from Edge Detection outputs

- Sobel Filter
- Laplacian Filter
- Canny Edge Detector

e.g

→ structure NOT related to colors but on pixel changes in intensity

edge structure: this is what we care about!

remove unecessary shading focus only on features NOT on details irrelevant

What are the methods to detect those edges?

SOBEL FILTER (first) (second)

detect edges in horizontal and vertical directions. By calculating gradients.

It uses a Filter Kernel that mathematically mirrors derivative operation

horizontal kernel (Gx)

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Gradient 1

vertical kernel (Gy)

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Gradient 2

⇐ this kernels acts as derivative operators

1) CONVOLUTION

CONVOLUTION

↓

I_{Gx}, I_{Gy} processed output images after kernel convolution

then the two Gradients are combined

2) COMBINATION $G = \sqrt{G_x^2 + G_y^2}$: magnitude of gradient of each pixel

$G[0][0] = \sqrt{I_{Gx}[0][0]^2 + I_{Gy}[0][0]^2}$ = value ... do it ∀ pixels

↓

$G[i][j]$ ∀ i, j in image size → you get a (G) image matrix result

MAGNITUDE MATRIX

3) THRESHOLD

then based on this G, we threshold it based on the Average of G. OR another threshold fine tuned

IF value > threshold → $I_{out} = 1$

IF value < threshold → $I_{out} = 0$

mask image based on sobel filter

LAPLACIAN FILTER

where the x-y derivative is separate

detects edges by finding regions of rapid intensity change.
It combines x and y direction derivatives in a single operation (differently from Sobel)

In Sobel we do it separately for I_{Gx}, I_{Gy}

In Laplacian is in a single step

the Laplacian kernel is a discrete approx of continuous math derivative (second order derivative)

there are two kernel options (NOT do both ...)

two kernel, based on how much edge intensity we want!
use kernel 1 OR kernel 2

Kernel 1

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Kernel 2

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

both kernels perform 2nd order derivative.

↓

it detects sudden changes in intensity, highlighting the edge.

Canny Edge the most advanced algorithm, based on Multi-step processing. Combining Sobel and thresholding techniques

- **step 1**: convert image in GRAYSCALE.
  since edges are present in color change informations, NOT on the specific color itself (on structure)
- **step 2**: apply Gaussian Blur technique... adding smoothness to the image, making it less sensitive to small color change variations
- **step 3a** Compute intensity Gradients with Sobel operator, to approx first derivatives in x and y directions
- **step 3b** Calculate magnitude (similarly to Sobel) but using also angle θ information, representing angle of the edge as $\theta = \text{arctangent}\left(\frac{G_y}{G_x}\right)$

Angle of the edge θ (from previous formula)

from angle of edge and magnitude, we perform

- **step 4** Non-maximum suppression

↓

for each pixel, compare gradient magnitude to its two neighbors along gradient direction → if pixel gradient is NOT the largest, suppress (set to 0)

e.g. if you have a pixel intensity to 45°

$\theta = 45°$ edge direction

Gradient direction of 45° → the neighbor pixel is the one around the next

you check Gradient and if NOT maximum it is set to 0

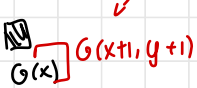
↓

- to suppress unnecessary informations present around main edges...

Reduce Redundant information by NON-MAX suppression

with Gradient direction 45° you compare $G(x,y)$ to $G(x+1, y+1)$ (neighboors) $G(x-1, y-1)$

$G(x-1,y-1)$ ← neighboors of $G(x,y)$ pixel, you keep only strongest and remove all the others in this direction



↓

after suppression

- **step 5)** Double thresholding

↳ High threshold: Pixel above this are strong edges

↳ Low threshold: Pixel below this are entirely suppressed } pixels between are weak edges

e.g $K_H = 100$ $G > 100$ strong edge

$K_L = 50$ $G \in [50, 100]$ weak

$G \leq 50$ suppressed

• step 6 : Edge tracking by Hysteresis

Weak edges kept or discarded based on connectivity to strong edges

If weak edge is connected to a strong one it is kept, otherwise discarded

e.g. When another edge is detected but NOT connected + weak, it is discarded

↑ in CANNY EDGE DETECTOR

multiple techniques are used to get a clean noise-free edge detection

+ Most computationally intensive !

CALCULATING AND PLOTTING HISTOGRAMS

fundamental tool in image processing ... histograms give "sneak peak" on the internal composition of the image (contrast, brightness, dynamic range, equalization potential...)

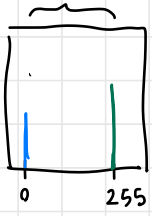
↓

this kind of informations allow to evaluate possible enhancements of the image

HISTOGRAM is a graphical representation of distribution of pixel intensities in an image

Brightness

in middle is "GRAY"



Dark image — Bright image

in a very dark image, eat of pixels will be close to 0. IF very light image you see many in 255 ⇐ brightness information retrieved here!

Dynamic Range

shows how well intensity values are distributed

checking proper distributed, NOT too dark or too bright

Equalization

equalize / enhance the image when histograms show clear information requesting for modification to redistribute

↓

Histograms may be computed in RGB channels domain OR HSV

↓

it can be used for advanced image enhancement techniques as comtrast enhancement and thresholding

↓

such as CLAHE technique

From basics histogram theory... **HISTOGRAM EQUALIZATION**

↓

How to balance images in brightness/contrast with Histogram Equaliz.
→ this helps to enhance images making it more vibrant and well distributed in brightness and contrast

Histogram equalization is a technique to enhance CONTRAST of an image by redistributing its intensity values across full range of values (0÷255) ← try to get a Good contrasted image redistributing pixels intensity... ↓ how does it works?

1) **Calculate Histogram** = show frequency of each intensity value (brightness)

2) **compute cumulative distribution function** (CDF) = accumulates frequency of intensity levels. To understand how intensity values are distributed across the image.
(If brightness is distributed towards left/righ/etc.)

3) **Map Intensity Values** = Using CDF old intensities are mapped to new ones so that distribution is spread on all range

4) **Generate equalized image** = replace old image pixels with new computations

e.g. Step 1 :

from Image = [52, 55, 59, 52, 55, 59, 63, 52]

Count	3	2	2	1
Intensity	52	55	59	63

create a table with intensity frequency (count)

Step 2 compute CDF as cumulative count

Intensity	Count (cumulative)	Normalized CDF (0-255)
52	3	(3/8) * 255 = 95
55	2 + 3 = 5	(5/8) * 255 = 159
59	2 + 3 + 2 = 7	(7/8) * 255 = 223
63	1 + 3 + 2 + 2 = 8	(8/8) * 255 = 255

(highest pixel)

accumulate all values smaller...

Normalization

older values (very dark) →

remapped to New intensity values (brighter values)

Step 3 : equalize image based on previous cdf (↑step 4)

Equalized_Image = [95, 159, 223, 95, 159, 223, 255, 95]

52 55 ... mapping

↑ histogram equalization

↳ In RGB domain Histogram equalization is performed on a different channel space such as LAB or YUV...

When applying Equalization, features hidden in the original image get highlighted

↓

this is why it is applyed on Medical imaging to enhance features

OR in satellite images to improve the contrast in low light photos

OR when improving old documents photo ↓redistribute pixel intensity to enhance features

More advanced Method to equalize images is

CLAHE := Contrast Limited Adaptive Histogram Equalization

↓

More controlled and localized version of histograms equalization working on a specific region of the image rather than globally

→ It improves contrast by applying histogram equalization in small regions or tiles of the image.
Focusing on localized contrast avoiding artifacts like noise amplification (differently from Global histogram equalization)

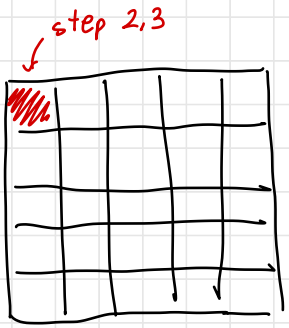
- limits contrast enhancement in each tile to prevent over-brightening

CLAHE process:

1) Divide images into tiles
2) Apply histogram equalization on each tile
3) Apply CONTRAST LIMIT to set maximum limit intensity amplification in each tile for smoother results

{ Repeat step 2, 3 ∀ tile divided in step 1 }

4) Combine tiles with BILINEAR INTERPOLATION to remove seams between tiles



IF there is an high intensity value in a tile. In CLAHE we limit that intensity. Repeat step 2, 3 ∀ tile

← to merge tiles it might possible that there are lines/artifacts in between new equalized tiles. We remove it with bilinear interp.

Example Given a GRAYSCALE image with low contrast

- Intensity ranges from 50-100. Therefore it is too dark lacking definition
- Split image in 8×8 pixel blocks
- Histogram equalization in each block, mapping pixels in 0-255
- CLAHE clip limit: ensures no pixel intensity exceed a specified value (e.g. 181)
  CONTRAST-LIMITING threshold (avoid too bright images)

- Enhanced tiles stitched together with SMOOTH TRANSITION

↓

Use it to:

- Enhance medical images (X-Rays, CT scans) easier to read
- Improves visibility in low-light conditions
  (distributing brighter/darker areas with new intensity values)
- Helps Enhancing specific details in aerial imaging
  ↳ redistributing intensity it enhance images

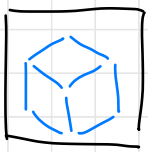
CONTOURS

How to find CONTOURS in images and how to use them for simple object detection tasks.

- Contours are CURVES that join all continuous points along a boundary that have the same color OR intensity

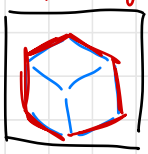
How to find these CONTOURS?

e.g.



Input image

→ find boundary of this object? CONTOUR →



output image

contours extraction

steps of contours extraction

- convert image to Grayscale
- Apply edge detection or thresholding
- Find contours
- Draw contours

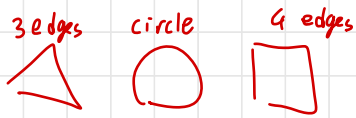
Contours are CURVES or SHAPES which are closed → Boundary of object in an image

NOT a contour! (NOT closed!)

in simple image, based on detected contours is related to number of objects with closed boundary

Usefull for:

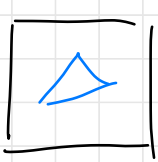
- Object detection (based on contours)
- Shape analysis



based on CONTOUR shape you can find # edges OR shape details to extract type of shape

- Measure object properties such as Area.

once you have object boundary



based on its pixels...

- How much percentage this object occupy in image % (perimeter)
- Area wrt image size

steps of contours extraction

1) convert image to Grayscale : contour detection works on a single channel
2) Apply edge detection or thresholding : we need a MASK that will be used to contour function. Binary mask (BW image 0÷1) thresholding / canny detection to pre-process image as contours mask
3) Find contours : using cv. Find Contours
4) Draw contours

↑ create bounding box or draw the shape as it is on the image.

IMAGE SEGMENTATION WITH OpenCV

process of dividing the image into meaningfull regions.
(e.g. background, foreground, object-specific regions)
ex use cases: obj detection, medical imaging and autonomous driving

examples:

- When using smartphone portrait mode, smooth the background with semantic segmentation

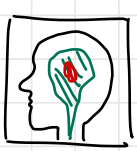
image segmentation mask the person recognize also what is NOT person and blur it

portrait mode uses a binary mask
apply blurring technique on all 0 values mask
where mask is 1 you keep original image

↑
segmentation divide the image to perform selective blurring as object-specific segmentation

- for object detection task such as selective search
  ↙ ↑
  these are used in RCNN object detector. find segments of the object in the scene, then used by RCNN

- Semantic segmentation
  classify each pixel into separate category {0, 1, 2..} Eat used in healthcare



+ many other usage of semantic segmentation such as Autonomous driving
  ↓
  We segment parts of the Road.
  ↙ e.g. vegetation, Road,... DIVIDERS of Road

extracting informations then is used to spatial understanding

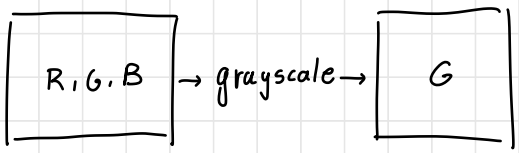
Multiple techniques makes image segmentation possible in OpenCV

1) THRESHOLDING

simplest segmentation approach.

1.a) SIMPLE THRESHOLDING

Apply a thresholding globally, considering all image

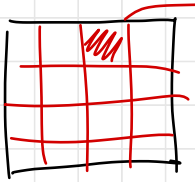


used for masking the image obtaining a binary mask

Intensity above threshold are filled as sharing similar intensity levels.

1.b) ADAPTIVE THRESHOLDING

Instead of global thresholding (on all image), it is applied locally in blocks → local thresholding



calculate threshold of this block based on neighbors pixels

↳ NOT a single t value, each block update dynamically its own value based on local intensity patterns...

Often highlighting regions where high pixel intensity gradient occurs

threshold point computed based on differend method:

- **Mean**: Threshold = Mean of local region - C
  ↑ average pixel intensity in small region under analysis
- **Gaussian**: Threshold = Weighted sum of local region - C
  ↑ pixels closer to center have greater influence. More robust thresholding

↑ C := constant to adjust threshold

▨ → If NOT happy with result... you may adjust the result by using an offset threshold value with C value!
(both for mean and Gaussian)

2) K-MEANS CLUSTERING

machine-learning based approach (used for clustering): that can help in image segment.

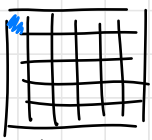
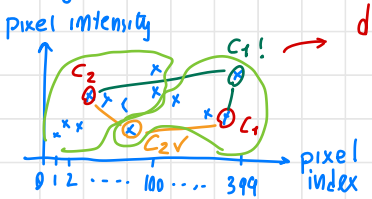


image
20 × 20 = 400 pixels

I(0,0)?
I(0,1)
⋮
→ check pixels intensity in 0 ÷ 400 index and analyze it



→ divide pixel intensity into cathegories (N category)
Initialized randomly
then K-means update Clasters and iteratively update it based on neighbors

← allocate all points based on distance..

grouping by K-means
groups ↘
allow a pixel segmentation!

- you specify the number of clusters (cathegory semantic) K = # class

then the input image will be grouped based on pixel intensity clusters.

⇓ this algorithm works as follow:

1) Flatten image in 2D array (∀ pixel index (row, col))
2) Specify # of clusters (e.g. foregr vs backgr.)
3) Use K-means to group pixels into clusters
4) assign each cluster to specific label (or color)

3) Watershed Algorithm

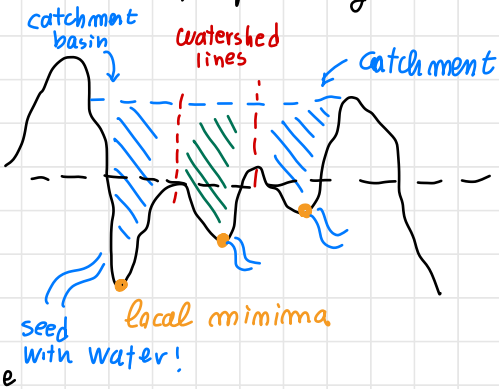
used for image segmentation

treating pixel intensities like topographical surface

↳ shape / orientation of a surface

It separate overlapping objects in an image

finding watershed lines, representing boundaries

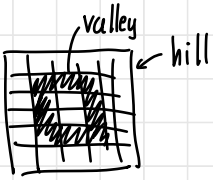


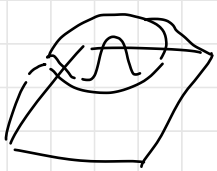
Topological surface (topography study features and forms of land surfaces)

start flooding the topology from local minima

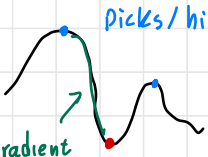
↓

here in CV we say that high intensity in grayscale image it forms an hill while if it is low it forms a valley





← watershed treat grayscale image as topographical map and find regions of high intensity (picks / hills) separated by low intensity (local minima = valley)



between hills we have valley

- All high intensity (bright region) pixels, are mountain picks while dark pixels are valley

↓

Gradient capture slopes / edges showing transitions high-low or opposite

Seed point = different regions of interest

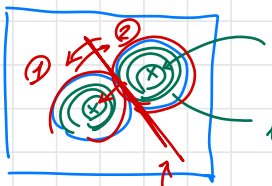
e.g  image with coins, seed points → show objects ↳ represented by local minima

Using local minima watershed alg. finds markers / seed points

→ when water from two regions meets a boundary is formed

↓

example

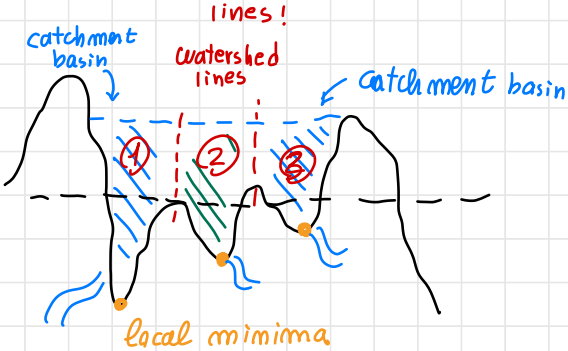


seed point / marker found as local minima (topological surface of grayscale image)
(= local minima)

fill water from minima

When overflow it forms a line

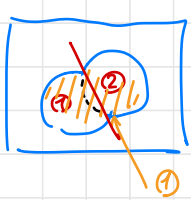
watershed lines!



you fill water from catchment basin...

↖ Watershed works with minima and lines to get semantic

example: With overlapping objects (e.g. coins)



separation cannot be recognized! ...

using thresholding just one object is found...

for computer perspective is JUST one object!

Watershed works in finding the two semantic

finding the watershed line separating the two objects

- watershed "Boundary"
- imagine each seed point as marker → representing a specific object (e.g background OR center of object)
- As processing image, "floods" each region pixel by pixel

↓

When pixels from two regions, the boundary exists and separable is consider.

the "meeting point" forms watershed lines

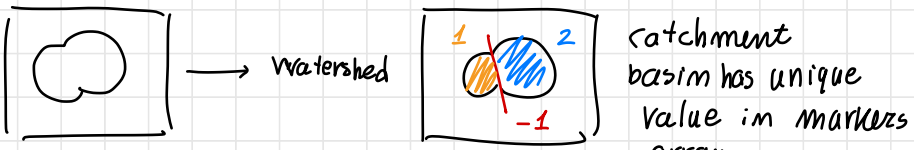
↓

the output of the Algorithm are Markers

- each region identified is assigned a unique label in marker array
- pixels in the boundaries are labeled with specific value (usually -1) in markers output
  ↳ boundaries are NOT separate outputs, but pixels labeled -1

↓

final markes array represent segmented image



catchment basin has unique value in markers array.

↑ array markers

- the purpose of watershed is NOT only to find boundaries, but to find labels to regions for segmentation

↑ different labels than allow to count objects number (NOT possible in thresholding!)

↓

makes output usefull in counting objects and performing instance segmentation + visualize boundaries (watershed lines)

isolate labels → only show specific marker labels isolated

Implementation?

↳

Watershed implementation

1) convert to grayscale
2) binary thresholding to mask foreground/background
3) use morphological operations to remove noise
   (cv. morphology Ex) when noisy image
4) compute distance transform, measuring distance from each pixel to nearest background
5) label foreground/background with markers
6) Watershed function compute watershed lines
   ↓
   mark boundaries where Watershed lines exist (-1 mask)
   ↳ mark boundary and do segmentation from this array

unique segmented classes. ←

DETECT FACIAL FEATURES WITH HAAR CASCADE

machine learning based object-detection algorithms introduced by Paul Viola and Michael James in 2001.
Haar like features + cascade classifiers identify objects in image

rectangular patterns that compare intensity of adjacent regions in an image.
Used to detect contrast in pixel intensity. Helping to identify structures as edges, lines, textures
↓
template / filters that detect specific patterns in an image

apply multiple stages of classifiers sequentially (CASCADE)
IF any of that fails... the next step are skipped and all is discarded!
↓
this make Haar cascade efficient. (NOT useless computations)

These are Boosted ML Algorithms as boosted cascade of weak classifiers.
↳ wathever first classifier cannot predict becomes a task of next one..

(e.g. Discrete Ada Boost (DAB)
Real Ada Boost (RAB)
Logit Boost (LB)
Gentle Ada Boost (GAB))

⇓
Haar Cascade
Use Cases:
in systems with less computation power, where deep learning is NOT possible OR to avoid

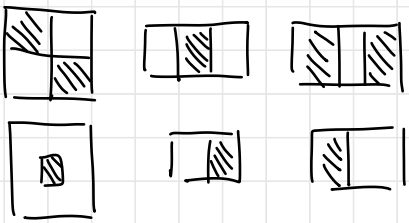
e.g. in older phone, bounding box Face detection is achieved by Haar Cascade.

Whenever we avoid deep learning solutions, Face detection

+ Person / car detections in smart security Without heavy DL models.

+ Person Detection in Robotics for avoid collision OR tracking

Har-like features are rectangular patterns (as anticipated)
... comparing intensity of adjacent regions



(basic patterns) ... others are possible

- detect contrast in pixel intensity, helping identify edges / lines / textures

e.g.
- Edge-Feature: distinguish dark vs bright region like nose edge

resemble an edge ≃

- Line feature: detect horiz / vertical lines like mouth line and nose boundary

- Four-Rectangle Feature: help to detect diagonal edges often used for eyes / windows

Haar-like features are calculated summing up pixel intensity in the rectangular regions + comparing their differences

example

Har Like features (normalized in 0,1)

Image 1: (input)

0.5	0.9	0.1	0.3
0.6	0.7	0.4	0

sum up pixel intensity + compare

Haar := (1+1+1+1) − (0+0+0+0) = 4

Image := (0.5+0.9+0.6+0.7) − (0.1+0.3+0.4)
= 1.9

Haar similarity
= White region - dark region (summed pixels)

such as: { IF pixel ≥ 0.5 bright
ELSE dark

↑ condition of classification of pixels

to compare similarity.

IF instead we consider Image 2 as 1 flipped

we get value = -1.9 (opposite)

comparing Image 1 ~ Haar feature is matching

Closer similarity based on index of similarity

you take blocks of image and compare using Haar-like feature...
calculated by summing pixels and comparing differences

... How Haar-Cascade can help to detect faces?

applying sequences of classifiers

- we have multiple stages... ① → ② - - - - → (i) → .. → (N) → Face detect.

if i fail → no face detected

the stages numbers cascade of validation should be fine tuned

Stage 1: Validation

- eye brow region
- Nose region
- Mouth region

↑ used on face to do a first stage validation

↑ multiple features validation are performed on stage 1 as step by step verification

IF all checks passed, validation 1 passed

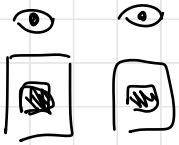
Stage 2 more complex checks on eyebrow-eye... etc

- eyes pattern
- eye left
- eye right

(rotate)

... each stage a more complex check is done

Stage 3 further detailed checks performed on face features
If step2 validation is approved..



check eye

mouth

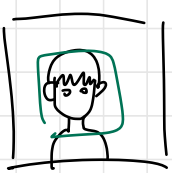
⋮

Stage N Cascade validation of Haar-like features

↓

Face Detected (If all stages have success)

(x, y, w, h) bounding box



detected by Haar-Cascade

↘ IF one of the stages fail the cascade block. fail Detection, No face