intel AI Academy

IMAGE TRANSFORMATIONS

Part II

LEGAL NOTICES AND DISCLAIMERS

This presentation is for informational purposes only. INTEL MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at intel.com.

This sample source code is released under the Intel Sample Source Code License Agreement.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

CANNY EDGE DETECTOR

CANNY EDGE DETECTION

This algorithm performs edge detection by completing the following steps:

1. Remove noise with a 5x5 Gaussian

2. Detect horizontal and vertical edges
   - The x,y gradients are first computed and then combined into four directional derivatives

3. Suppress non-maximums (in direction of gradient)

   Local maxima define edges

CANNY EDGE DETECTION CONTINUED...

4. Apply two thresholds, *high* and *low*

   Points below *low* are dropped

   Points above *high* are kept

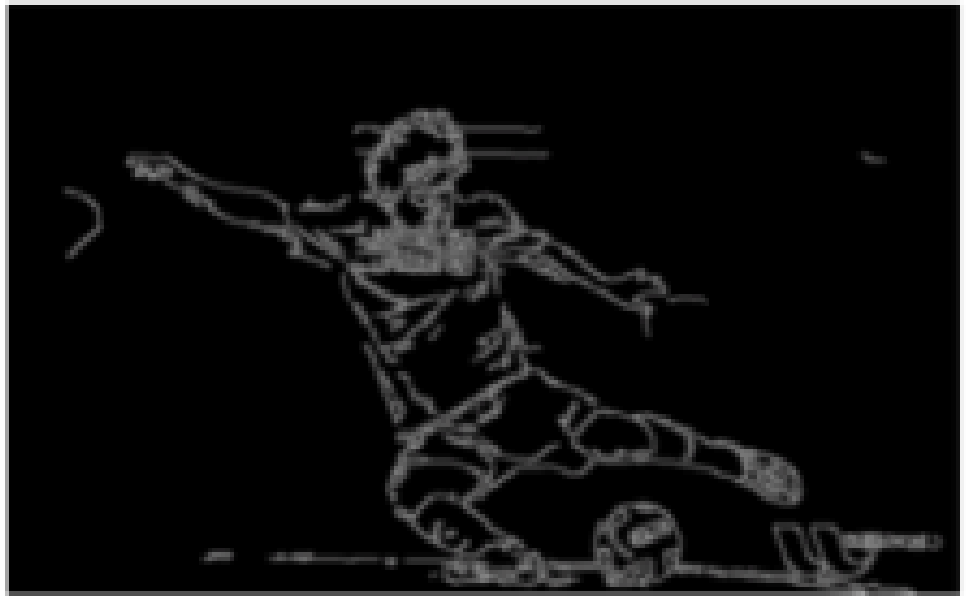
   Points in between are kept if they connect to a kept point

   High: Low thresholds should be between 2:1 and 3:1

Note: Direction of gradient is perpendicular to direction of edge

CANNY EDGE DETECTION

Low Threshold = 100; High Threshold = 200



FOURIER TRANSFORM

PTOLEMY AND COMPLEX REPRESENTATION OF PLANET MOTIONS

Ptolemy and the ancients had a model of the planets:

- Deferents and epicycles

- A series of smaller circles turning on bigger circles

  This modeled the observed motions of the planets fairly well

PTOLEMY AND COMPLEX REPRESENTATION OF PLANET MOTIONS

But the model was not explanatory.

Why? (CLICK ME)

An infinite sum of circles on top of circles can trace ANY fairly well-behaved (continuous and not too wiggly) path ...

This is the fundamental idea of Fourier methods.

https://www.youtube.com/watch?v=QVuU2YCwHjw

PTOLEMY AND COMPLEX REPRESENTATION OF PLANET MOTIONS

The model was descriptive.

- It gave the right positions of planets.

The model was not explanatory.

- The mathematics did not reflect the mechanism of planetary motion (aka gravity).

FOURIER TECHNIQUES

All Fourier techniques take a function as input and output another function.

They all add up to a very special set of terms that form an orthogonal basis for those fairly well-behaved functions.

$$f\big(g(x)\big) \rightarrow h(x)$$

$$f\big(g(x)\big) \rightarrow h(y)$$

$$X_k = \frac{1}{n} \sum_{n=0}^{N-1} X_n$$

X_i = *i*th component of Fourier series, each a component of *orthogonal basis* (see next slide)

Formula for each X_i is in later slides.

NOTE:

In math, we call this a functional / functional form

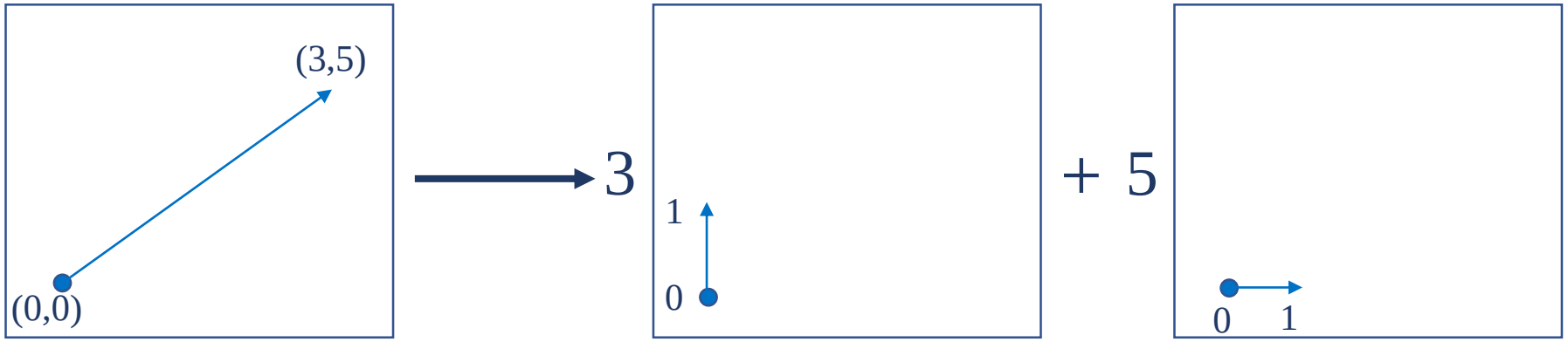
In CS, we call this a second-order function

FOURIER TECHNIQUES

Orthogonal? Basis?

We have experience with bases from working with data

- We can rewrite any (x,y) point in a 2D plane as (x,y)=x(1,0) + y(0,1)



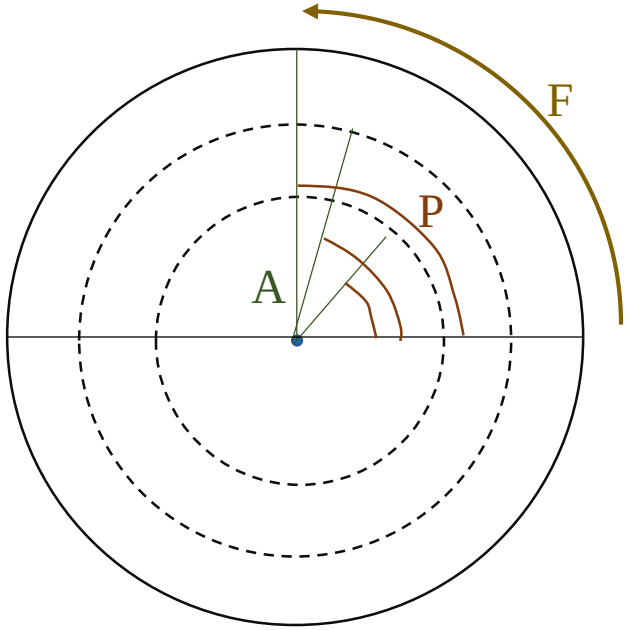
This is an incredibly powerful mathematical idea that can be applied to many sets of mathematical objects including functions!

HOW DOES FOURIER WORK?

The bases of Fourier methods are a bit more complicated than (1,0) and (0,1).

They are functions of a special type: They are circles!

- Information needed to coordinate drawing the same circle at the same time (verbal information)



A = Amplitude (radius, size)
P = Phase angle (starting point)
F = Frequency (speed around the circle)

Image adapted from: https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/

FOURIER METHODS

Fourier methods rewrite a function *f* in terms of an infinite sum of circles.

The rewritten function is defined at a different set of inputs and produces different outputs:

- Inputs: Frequencies (speeds of drawing the circles)
- Outputs: The radii and phases (size of circle and angle of starting point)

HOW DOES FOURIER WORK?

Mathematically, the circles are encoded as either (scary, complex) exponentials, sins, or sins and cosines.

The three are equivalent because of trigonometric identities and Euler's formula.

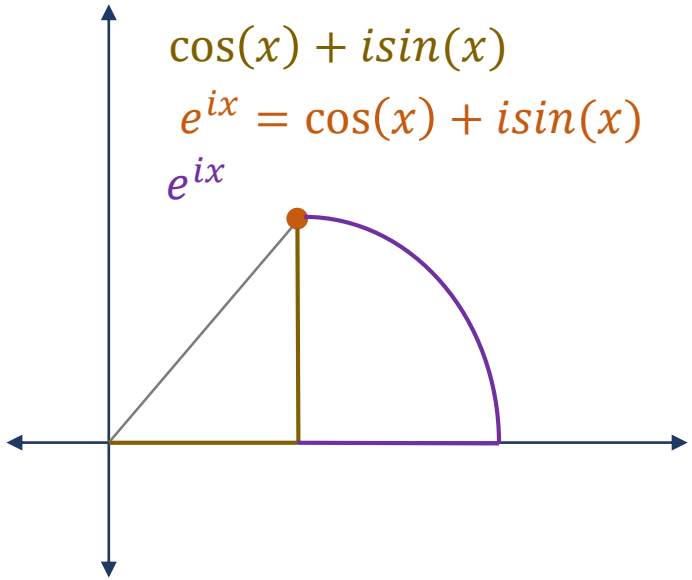


Image adapted from: https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/

FOURIER COMPARISON WITH TAYLOR SERIES

Taylor series rewrites a function as a sum of derivative-polynomial terms (a different basis):

- Function of data to function of data

$$f(x) = \sum_{n=0}^{\infty} \frac{f(n)(a)}{n!}(x-a)n$$

FOURIER COMPARISON WITH TAYLOR SERIES

$$f(x) = \sum_{n=0}^{\infty} \frac{f(n)(a)}{n!}(x-a)n$$

Same but different:

- Yes it is a rewrite.
- In the Taylor series case, both the original and the rewrite are in terms of the same inputs and outputs.

Fourier methods are more extreme: They change the domain and range of the rewritten function!

- *f*(data) → *f*(circles)

FFT AND DFT

We will be using one pair of Fourier methods.

Forward: Data to circle

Backward: Circle to data

FFT AND DFT: FORWARD

Forward: Data to circle

Input $f_{data}([0,N-1]) \rightarrow$

Output $f_{circles}([0,N-1])$

The input function tells us our data values

Input has N data values

Output has N circles

The output function tells us characteristics of the circles in our rewrites

FFT AND DFT: BACKWARD

Backward: Circle to data

Input $f_{circles}([0,N-1]]) \rightarrow$

Output $f_{data}([0,N-1])$

Here we are going from an input function in terms of circles to an output function in terms of our data

Input has N circles

Output has N data values

FFT AND DFT: π

You might be worried that neither of these are necessarily functions of, or around, a circle.

What is a function of a circle?

- A function of a circle is defined for each angle of a walk around a circle [0, 2π] radians

FFT AND DFT: π

We can convert to a function of a circle if

1. Our input is discrete and finite:

   Map our values from [0, N-1] to points around the circle

   Divide [0, 2π] into N parts

   This is what we have with images

   - Images are discrete and finite
   - Edges of the images wrap to make a *circle*

2. Our input is discrete and repeating:

   Do the exact same thing because the circle wraps around itself!

OBLIGATORY MATH #1

$$X_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{-i 2\pi k \frac{n}{N}}$$

Place on Circle
Data
Frequency
Average

OBLIGATORY MATH #2

Notice formulas are almost the same

Place on Circle
Data
Frequency
Average

Only fundamental difference is that the sign of the exponents is opposite.

Different mathematically equivalent equations will move the 1/N to the circle recipe or split it as root N on both recipes.

Circle recipe — Contributions from a given circle

$$X_k = \sum_{n=0}^{N-1} x_n \, e^{-i2\pi k \frac{n}{N}}$$

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \, e^{i2\pi n \frac{k}{N}}$$

Data recipe — Contributions from a given circle

WHY FOURIER?

Filtering

Fast convolutions

Compression

- See JPEG and MP3s!

WHY FOURIER?

Our rewrite in terms of circles can be hacked.

Different circles move at different speeds

- For example, we even have a term for the stationary circle: the *mean*

Speeds (frequencies) are related to image regions of slow and fast variation

- High-speed changes are either edges or noise

Low-speed changes are, taken together, a peculiar averaging of the data.

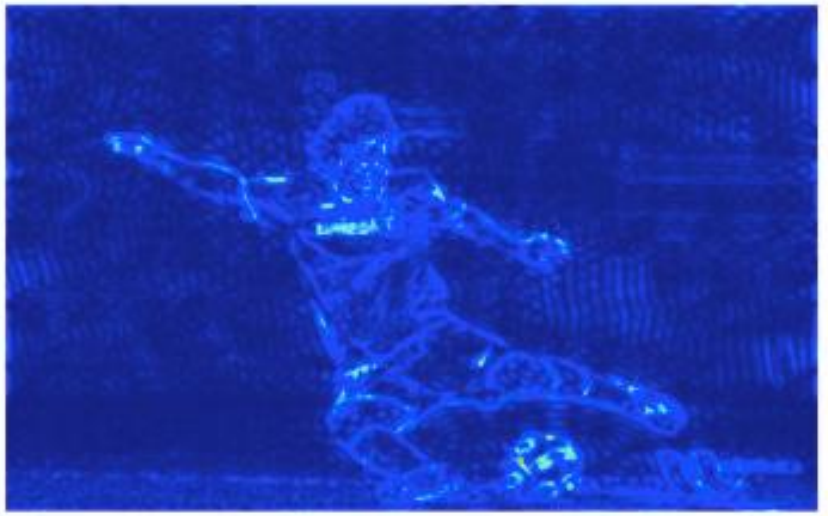
WHY FOURIER?

These give us:

High-pass filters where we keep high frequencies.

Low-pass filters where we keep low frequencies.

FREQUENCY BASED TECHNIQUES: HIGH-PASS FILTERS



High-pass filtering removes low frequencies.
In images, a high frequency is a contour.

WHY FOURIER?

We can take advantage of properties of Fourier transforms.

Convolutions, in circle space, are simply multiplication.

This can allow us to drastically speed up convolutions:

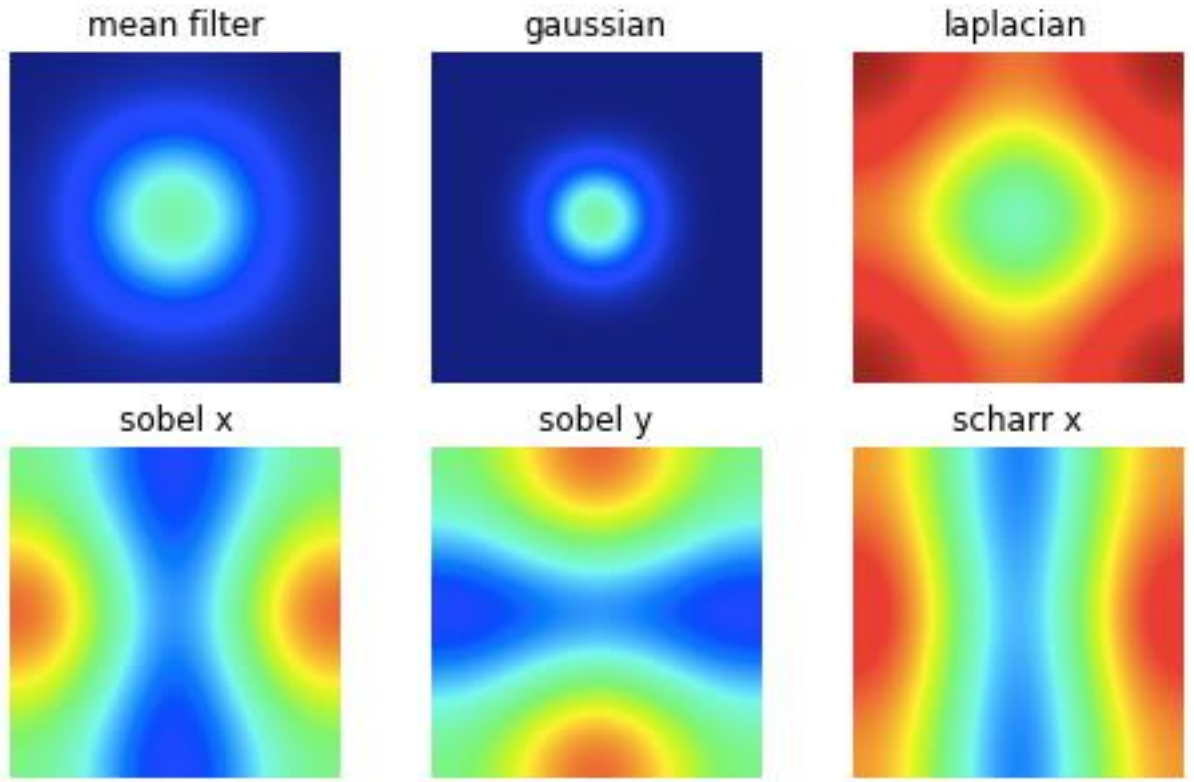
1. Discrete Fourier transform (DFT) the data and the kernel
2. Perform multiplication in circle space
3. Transform back to data space

FILTERS

Simple averaging

Gaussian edge detection

- Laplacian
- Sobel
- Scharr



HOW TO USE FOURIER METHODS

NumPy and OpenCV (and others):

- It uses the fast Fourier transform to compute the discrete Fourier transform.
  Gives data points from an underlying, unknown function

- There are usually some variations for dealing with different types of input.
- There are variations for whether we are going forward (to circle space) or backwards (to data space).

HOW TO USE FOURIER METHODS

There are usually some helper functions to deal with the weirdness of implementation details.

One important detail is that of *size:*

- DFT algorithms are designed to work best with particular sizes
- Different implementations subdivide the problem into different sizes, so they have different ideal input sizes
  - NumPy implementation ideal size is a power of 2
  - OpenCV implementation ideal size is a product of 2s, 3s, 5s
    - For example, 2*2*3*5*5 = 300

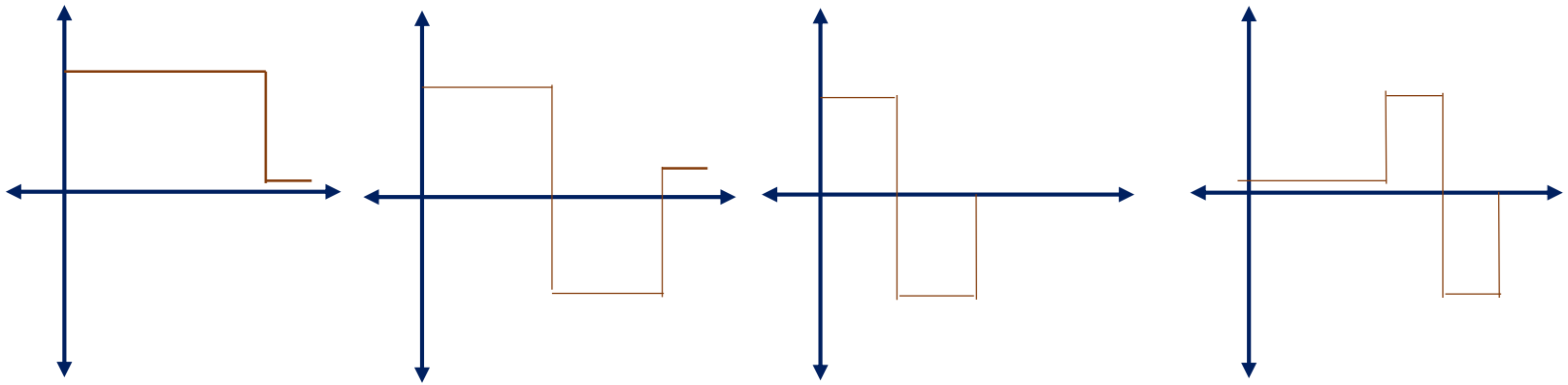
For OpenCV, we can calculate with cv2.getOptimalDFTSize().

WAVELETS

Wavelets are an alternative basis to represent data.

- There are many, many wavelet basis
- Circles can be exponentials or sin/cos

  These are closer to one cycle of *sin* or *cos*



Haar wavelet basis

WAVELETS

A data representation describes the phenomena point-by-point.

Circle space describes the data in terms of cycles.

Wavelets represent a tradeoff between the two.

WAVELETS

We can tell something about both the data and the circle characteristics at the same time.

But …

- We can't know both *exactly.*
- Amazingly, this is the mathematical root of one version of the Heisenberg Uncertainty Principle.

WAVELET APPLICATIONS

Wavelets are:

- Used for denoising
- Used for compression
- May outperform DFT and discrete cosine transform (DCT) in peak detection

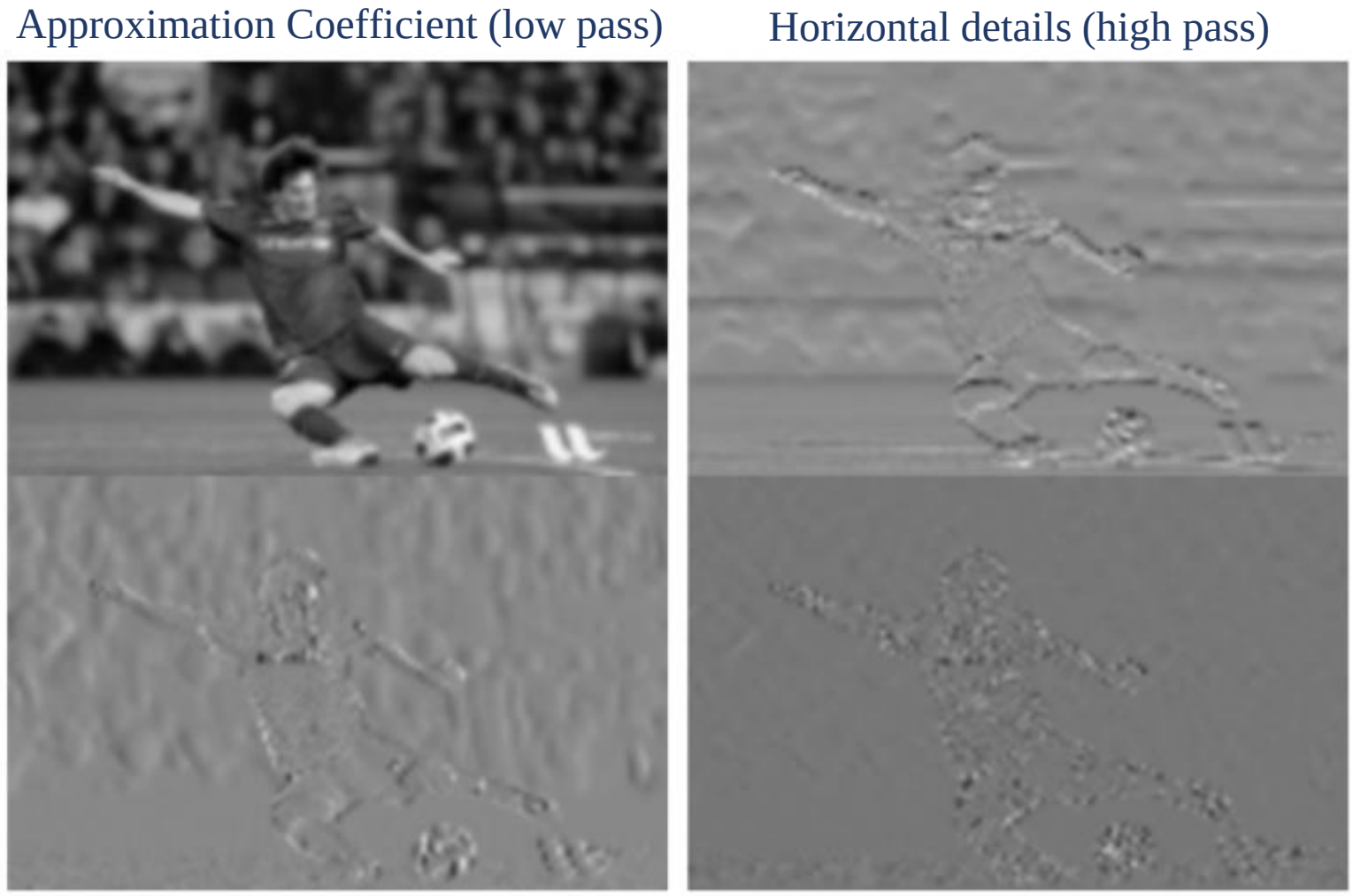
WAVELET APPLICATIONS

Low-pass filter

- Approximation coefficient

High-pass filter

- Detail coefficient
  - Horizontal
  - Vertical
  - Diagonal



DISCRETE WAVELET TRANSFORM

Similar to the FFT and DFT, we use the discrete wavelet transform (DWT).

We use the PyWavelets* (pywt) package to do the calculations.

```
import pywt
cA2, (cH2, cV2, cD2), (cH1, cV1, cD1) =
        pywt.wavedec2(messi, 'haar', level=2)
```

DISCRETE WAVELET TRANSFORM

Level is similar to levels in a Gaussian pyramid:

- One application of DWT results in an approximate image and details.

  We can *add* the two to get back the original image

- Subsequent applications of DWT on the level 1 approximate image give us a level 2 approximate image and level 2 detail images.
- We can continue applying this process recursively.

LINE DETECTION: SUCCESSIVE APPROXIMATION

Transforming from a curve-contour to a *simpler* representation (aka *line simplification*)

- Piecewise-linear polyline
- B-spline curve

If the contour ***is*** a line (with noise) we can simplify it to that line.

If not, the line simplification is used to approximate the original curve.

HOUGH TRANSFORMS

Propose shape from point

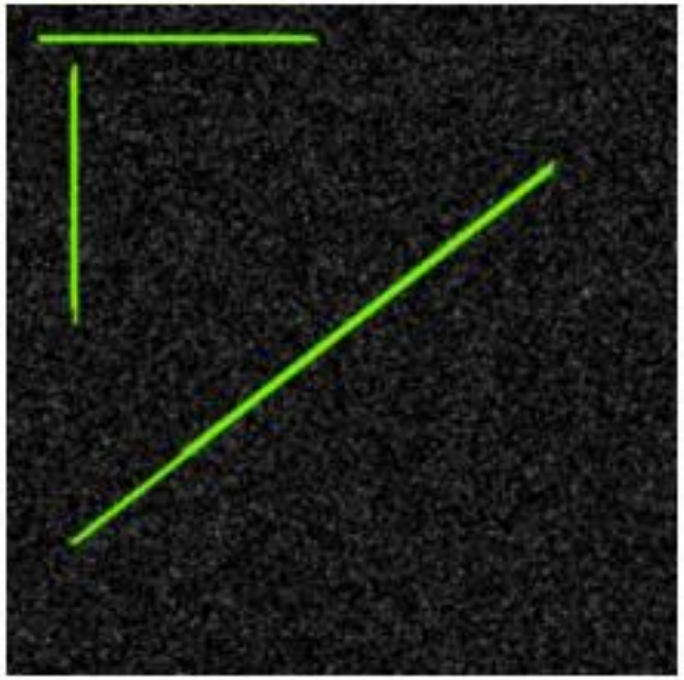
Count points that agree with shape

Keep shapes at points with highest counts

Can do this for any shape we can describe with a small number of parameters (such as point-slope, angle-distance, center-radius, center-scale-angle).

HOUGH TRANSFORMS

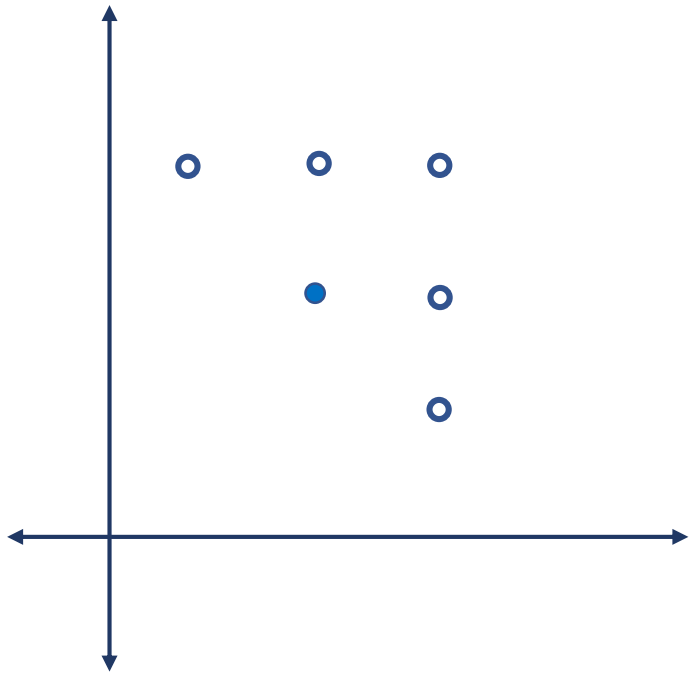
LINES



CIRCLES

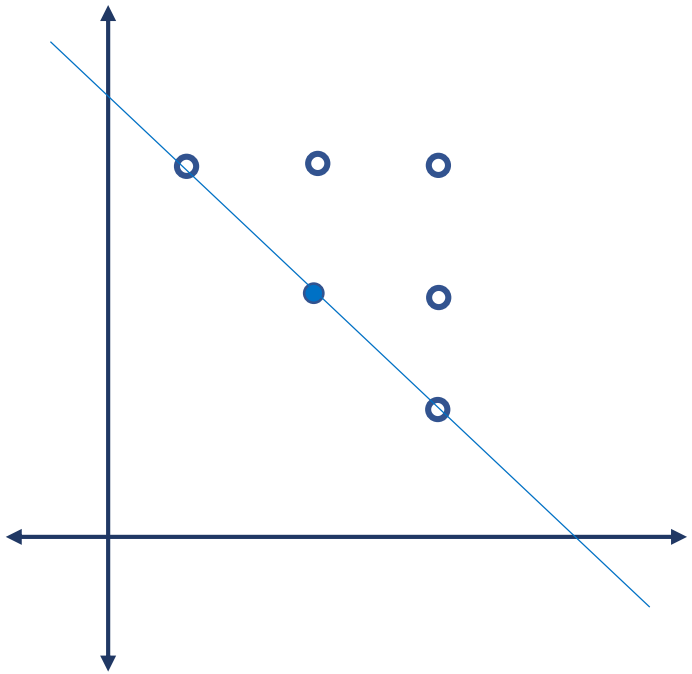
HOUGH TRANSFORMS: AN EXAMPLE

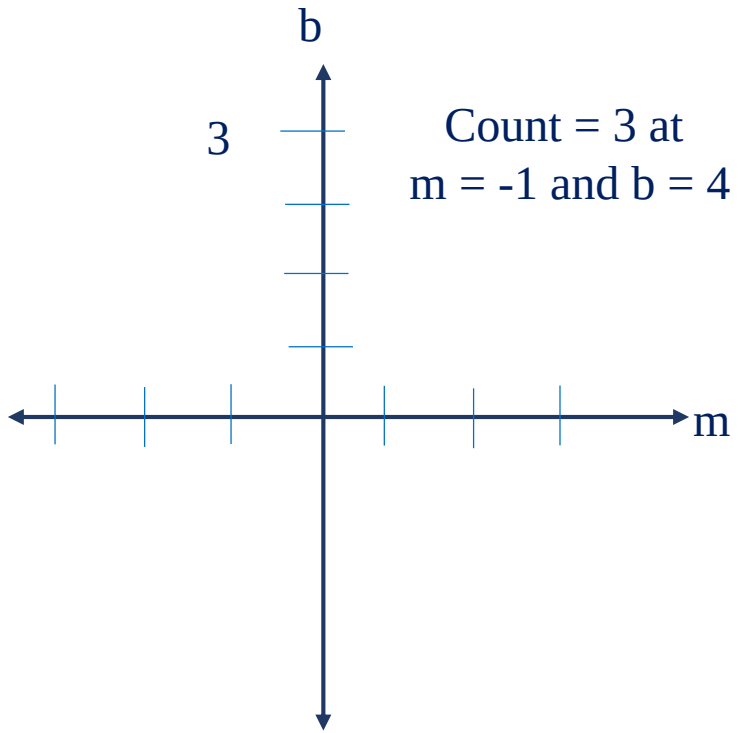
What potential lines do we see here?



HOUGH TRANSFORMS: AN EXAMPLE

Setup → m,b grid





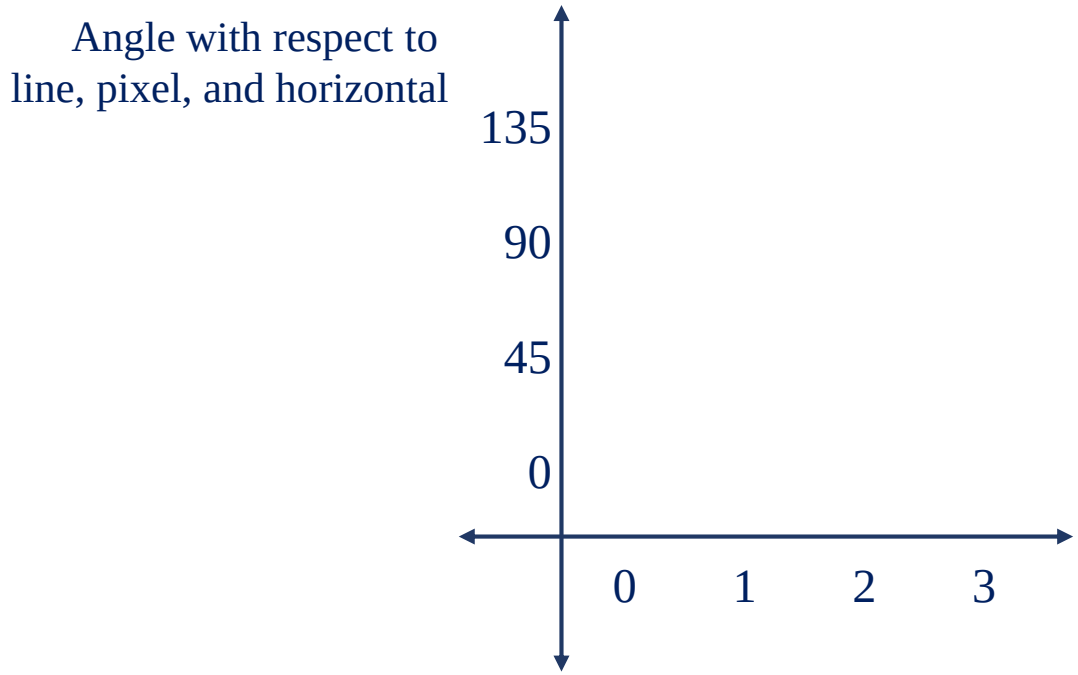
HOUGH TRANSFORMS: AN EXAMPLE

0-(w,h)
bins

0-π) radians
bins

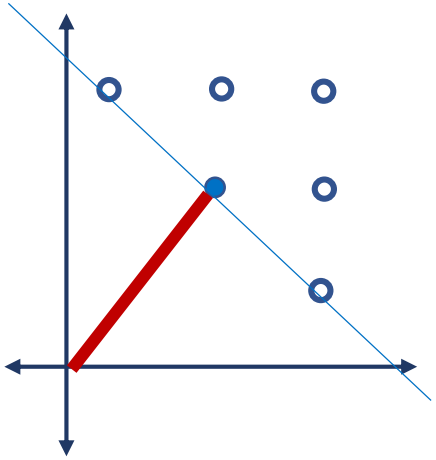
Accumulator over (distances, angles)

Fill d, Θ table (accumulator) with counts of agreeing pixels



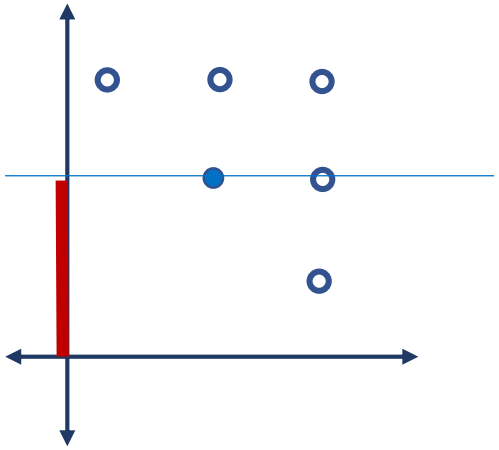
HOUGH TRANSFORMS: AN EXAMPLE

L1



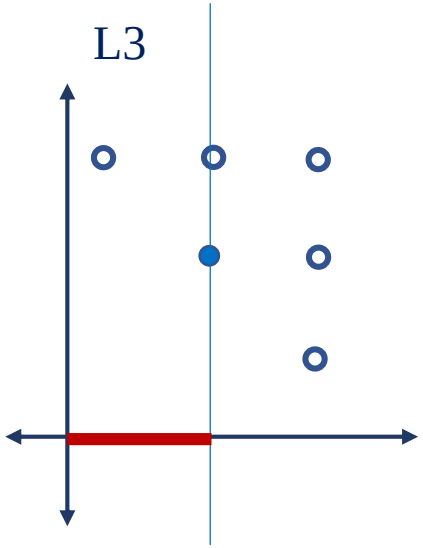
D = 2.83 or ~3
Θ = 45

L2



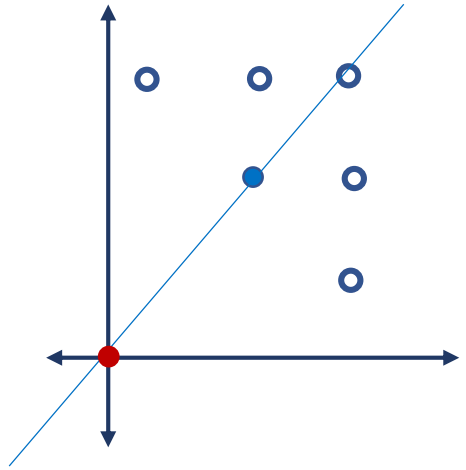
D = 2
Θ = 90

L3



D = 2
Θ = 0

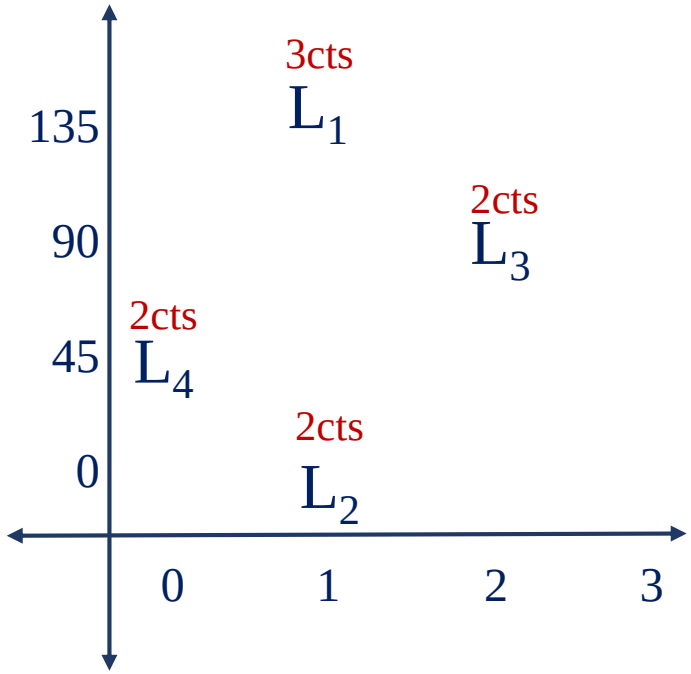
L4



D = 0
Θ = 135

HOUGH TRANSFORMS: AN EXAMPLE

Tally of the points that match the d,Θ model



1 point votes (cts) for 4 lines (L_1-L_4) is 1 vote per angle bin

HOUGH CIRCLE TRANSFORM

The circle is similar to the lines

- But, the vote is performed by taking the circle center and its radius
- Three parameters in this case
- Would be too slow because of combinatorics
- Would produce a very sparse table because of relatively few matching objects

Instead, we use the gradient and combined edges

- Only the connected can vote for the same circle

GENERAL HOUGH TRANSFORM

We can generalize this process to other shapes by using a contour instead of the perimeter of the circle.

Track from a pixel on the contour, rotate the orientation of the contour around a circle

- Count pixels matching at angles.
- Can also have scaling factors.

LINE DETECTION: RANDOM SAMPLE CONSENSUS (RANSAC)

Pair of edge elements are used to form a line hypothesis.

How many others fall on this line?

- Set boundaries for minimum matchers

Can also use RANSAC for image alignment.

Note: see OpenCV fitLine:

- It may use a RANSAC-like method
- Is it RANSAC-like only in use of m-estimator?

EDGE THINNING: DETECTING THICK EDGES

Detect thick edges as one edge (without doubling)

Morphological erosion

Skeletonization

Watershed

Pearling

Convert edges to contours

- findCountours()

LINE DETECTION: HOUGH TRANSFORMS

For occluded lines, we need more advanced techniques like Hough transform.

- NOTE: Hough can include or exclude gradient information.

- With it:

  Do not need to orient the line

  Need fewer accumulators

  Do not need inner loop (work has already been done)