Intermediate-level vision

Adrian F. Clark <alien@essex.ac.uk>
CSEE, University of Essex

What is *intermediate-level* vision?

In the early days of computer vision, it was realized that humans can distinguish objects in scenes from line drawings — so, the argument goes, this should be easier for computers too

The result of this was a focus on edge detection in images

We have already considered Sobel's edge detector, using the masks

1	2	1
0	0	0
−1	−2	−1

1	0	−1
2	0	−2
1	0	−1

Experience with this led to the development of Canny's edge detector in about 1984

John Canny's edge detector

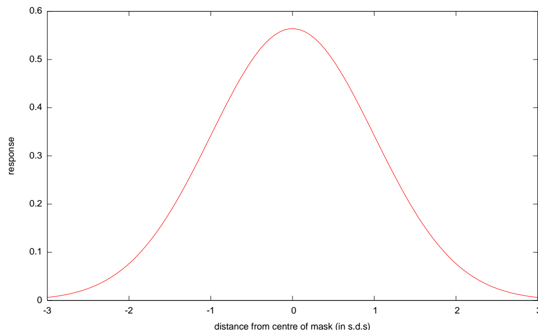
Based around three principles:

1. it should respond only to edges, and all edges should be found
2. edges should be found in the right places in images
3. a single edge should produce a single response

These principles led Canny to come up with a five-step algorithm

Step 1: Gaussian blur

Convolve with a mask having a Gaussian profile to reduce noise and blur the image a little



Gaussian profile

Step 2: Find differences in horizontal and vertical directions

We obtain $H(x, y)$ and $V(x, y)$ respectively in the two directions

```
        x   x+1                                x   x+1
-+---+---+---+---+---+-                -+---+---+---+---+---+-
 |   |   |   |   |   |                  |   |   |   |   |   |
-+---+---+---+---+---+-                -+---+---+---+---+---+-
 |   | - | - |   |   |    y             |   | - | + |   |   |
-+---+---+---+---+---+-                -+---+---+---+---+---+-
 |   | + | + |   |   |    y+1           |   | - | + |   |   |
-+---+---+---+---+---+-                -+---+---+---+---+---+-
 |   |   |   |   |   |                  |   |   |   |   |   |
-+---+---+---+---+---+-                -+---+---+---+---+---+-
```

Step 3: Convert to magnitude and direction

We convert H and V into edge magnitude

$$M(x,y) = \sqrt{H(x,y)^2 + V(x,y)^2}$$

and

$$\theta(x,y) = \tan^{-1} \frac{V(x,y)}{H(x,y}$$

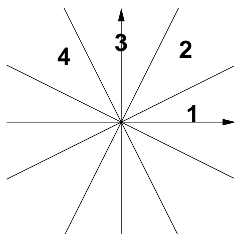
in the usual way

Note that computationally it makes more sense to use `atan2()` than `atan()` as the former takes into account the signs of H and V correctly

Step 4a: Angle quantization

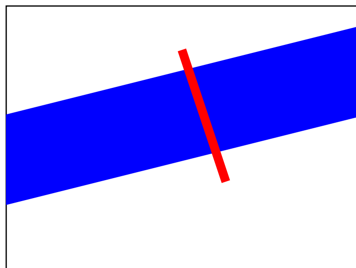
We quantize the angle into four broad directions

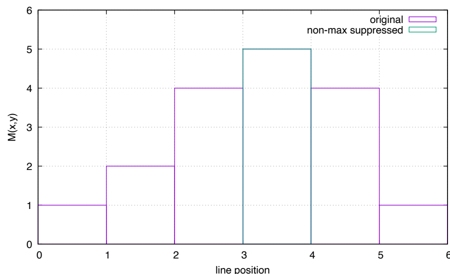
1. $(1, 0)$
2. $(1, 1)$
3. $(0, 1)$
4. $(-1, 1)$



Step 4b: Non-maximum suppression

Having quantized θ, we use it to step across the line and carry out *non-maximum suppression* to thin the line to one pixel width



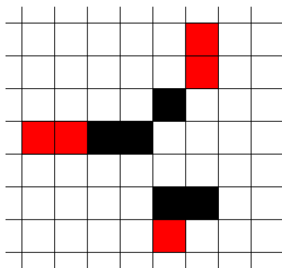


Step 5: Edge linking by hysteresis thresholding

The above stages tend to produce *line segments*, chunks of a line rather than complete ones, so we need to join them together where appropriate

We step over the image and increase the M of pixels with $\tau_L < M < \tau_H$ (in black) into stronger edge pixels only if they lie next to an edge pixel with $M > \tau_H$ (in red)

This is most easily done recursively

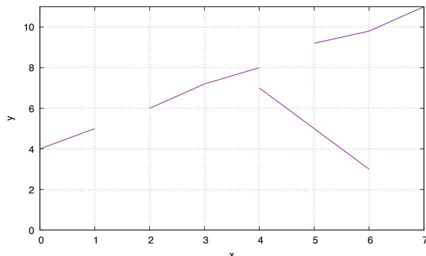


The Hough transform

Finding an edge is only part of the job; we often need to work out the corresponding equation, and the Hough transform can do this

Although we'll look only at the straight line case, the technique can be applied to circles and other shapes

Even after edge linking, edge detectors tend to produce only edge segments so we need to accumulate evidence as to what the line is



Edge segments

The equation of a straight line is

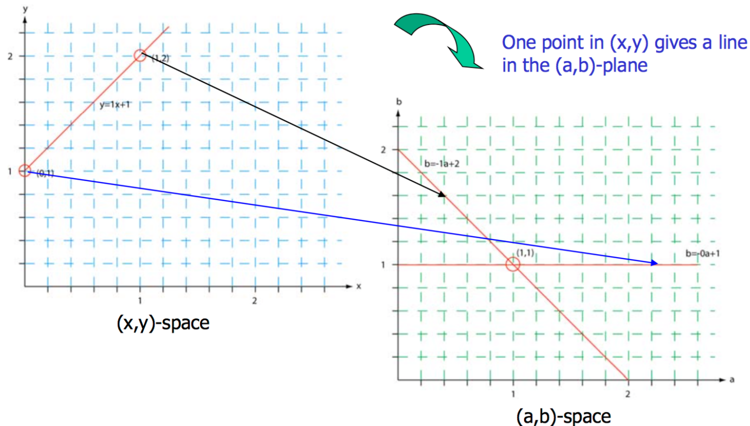
$$y = ax + b$$

In the Hough transform, we re-arrange this into a function of a and b with x and y giving the gradient and intercept respectively

$$b = -xa + y$$

What we do is draw *an entire line* in (a, b) space for each (x, y) point we find in the image

We call the place in which we draw these lines an *accumulator*, adding 1 to each accumulator element where the line appears



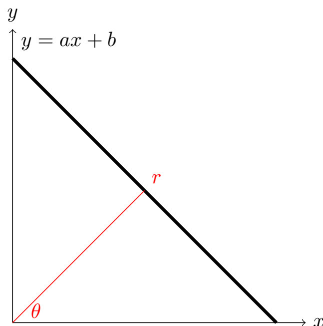
Drawing into the Hough accumulator

Peaks in the accumulator correspond to lines in the original image

Parameteric lines

We cannot use the method described above in practice because the gradient a is infinite for vertical lines

Instead, we use a *parametric* form of the line and use (r, θ) as the axes of the accumulator, meaning that we have to draw a sine wave for each (x, y) point



Applying the Hough transform

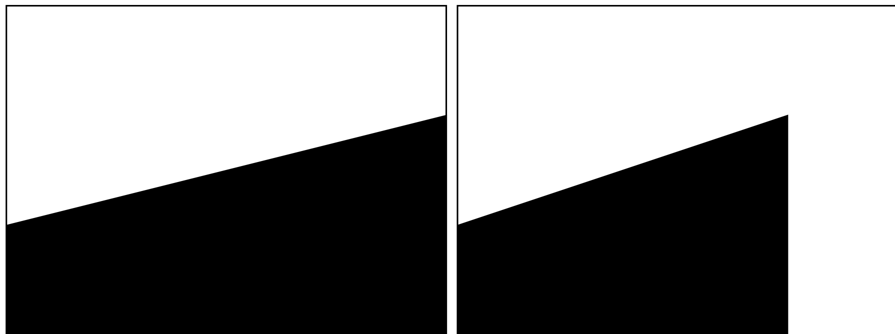
There are innumerable uses of the Hough transform but one of the most obvious is in tracking the white lines at the edges of a road in order to alert the driver if they stray from their lane

Lane-following using the Hough transform

The aperture problem

Despite the focus on edge detection in the past, edges are of limited value in practice because one cannot know *where* on a line a particular pixel is

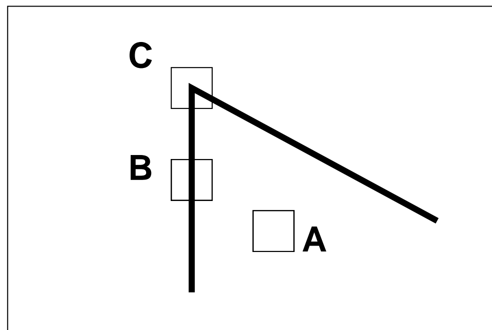
This is not the case for a corner, so corner detection is much more useful



Moravec's corner detector

Consider three different cases:

A: in a uniform region
B: on an edge
C: on a corner



We look at the same four adjacent pixels as in Canny's edge detector, displaced by 1. $(1, 0)$, $(1, 1)$, $(0, 1)$ and $(-1, 1)$

- **At A**, the differences between a pixel and the four adjacent ones is always small
- **At B**, the differences between a pixel and adjacent ones *along* the line is small but *across* the line is large
- **At C**, the differences between a pixel and the four adjacent ones is always large

so we store the *minimum* of the differences in an 'image' and then look for *maxima* in it to find the corners

This works reasonably well

Harris and Stephens

Moravec isn't the state of the art in corner detection; that is arguably the detector due to Harris & Stephens, sometimes called the *Plessey* corner detector; this is in OpenCV

It takes a mathematically more sophisticated approach than Moravec

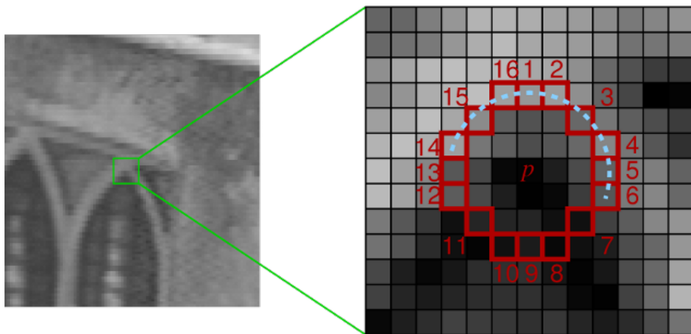
With a little effort, it can be made to run in real time

The FAST corner detector

Another corner detector worth knowing about is FAST, from Cambridge

It is based on the simple observation that over half the pixels surrounding a corner must be light or dark

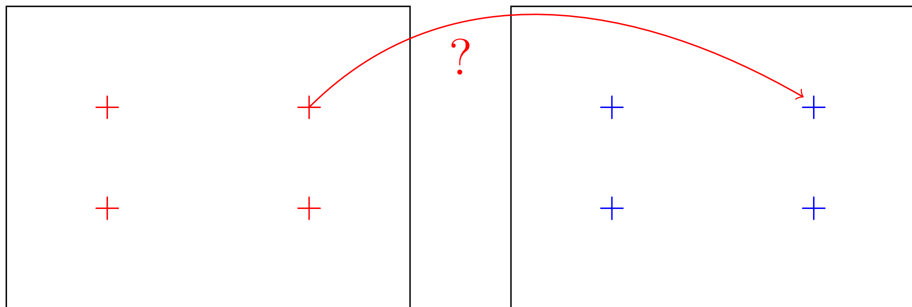
My experience with it is that it generates a lot of false positives



The need to match corners

Finding corners in images is usually a first step in a processing pipeline

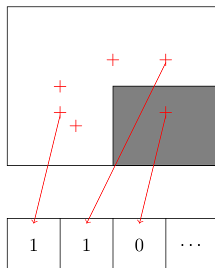
We usually want to find which corner in one image matches which one in another



Describing corners

Although I think my own corner descriptor, `arc`, is better, the way most people describe corners is with BRIEF or its successor ORB

BRIEF samples points around a corner, storing `1` if above the centre value and `0` otherwise



Matching briefs

If we look at the same corner in another image and sample around it in the same relative locations, we should get a similar binary pattern

We can measure how different these patterns are using `XOR` (exclusive `OR`), a single operation on most computers, and then add up the number of bits set to give a score

However, this is badly affected is there is a rotation or change of scale between images

The ORB ("oriented BRIEF") descriptor gets around the rotation problem and is less susceptible to scale changes, and so is widely used

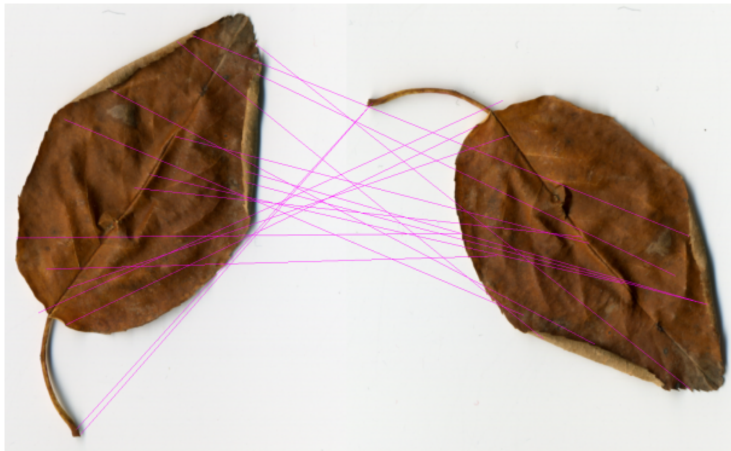
Matching non-corner regions

We often want to match regions other than corners between images, so you might ask whether there is something that can do that

The answer is that there are many, the most important of which is Lowe's *Scale-Invariant Feature Detector* (SIFT)

Although quite slow to compute (and patented), SIFT is reliable and tends to match non-boundary regions — that makes it a good complement to ORB *etc*

SIFT in action



Matches found by SIFT

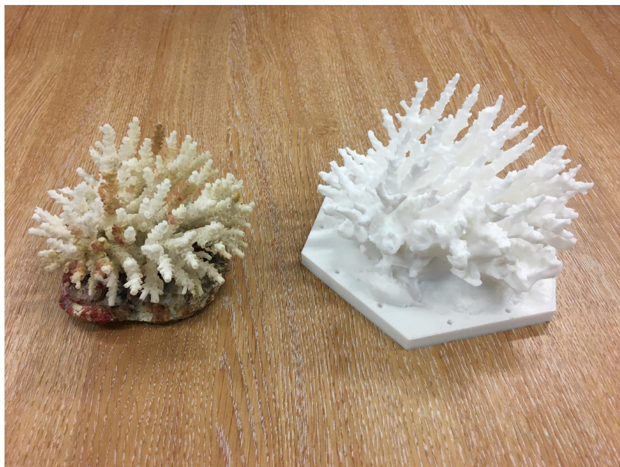
SIFT and *structure from motion*

SIFT is commonly used in *structure from motion*, wherein 3D reconstructions are built from photographs

3D reconstruction of Elmstead church using SIFT

3D reconstruction of a coral reef made at Essex using SIFT

Reconstructions can be pretty accurate



Coral and its 3D print