Combining Hough transform with contour tracing

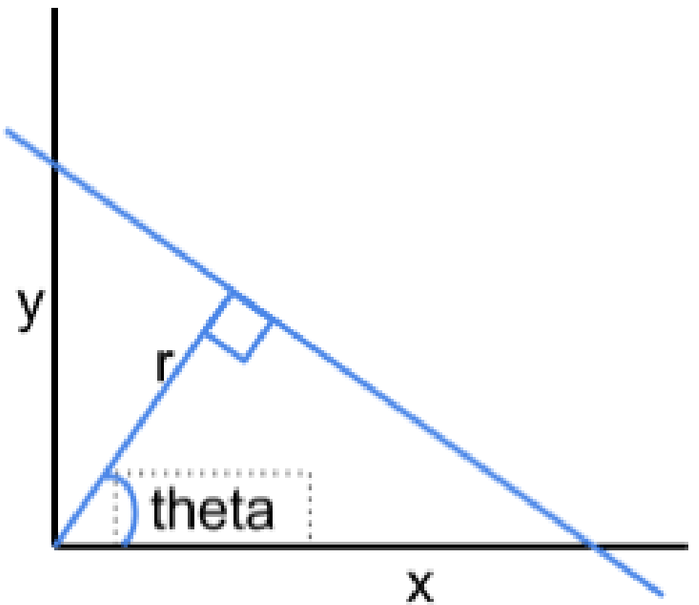
Course: Digital Image Processing
Professor: Nicola Adami
Authors: Marco Cadei 708217, Emanuela Cantoni 708953

Hough Transform

- Hough Space

Line representation: a line is identified by the couple (θ, ρ)

ρ = x*cos(θ) + y*sin(θ)

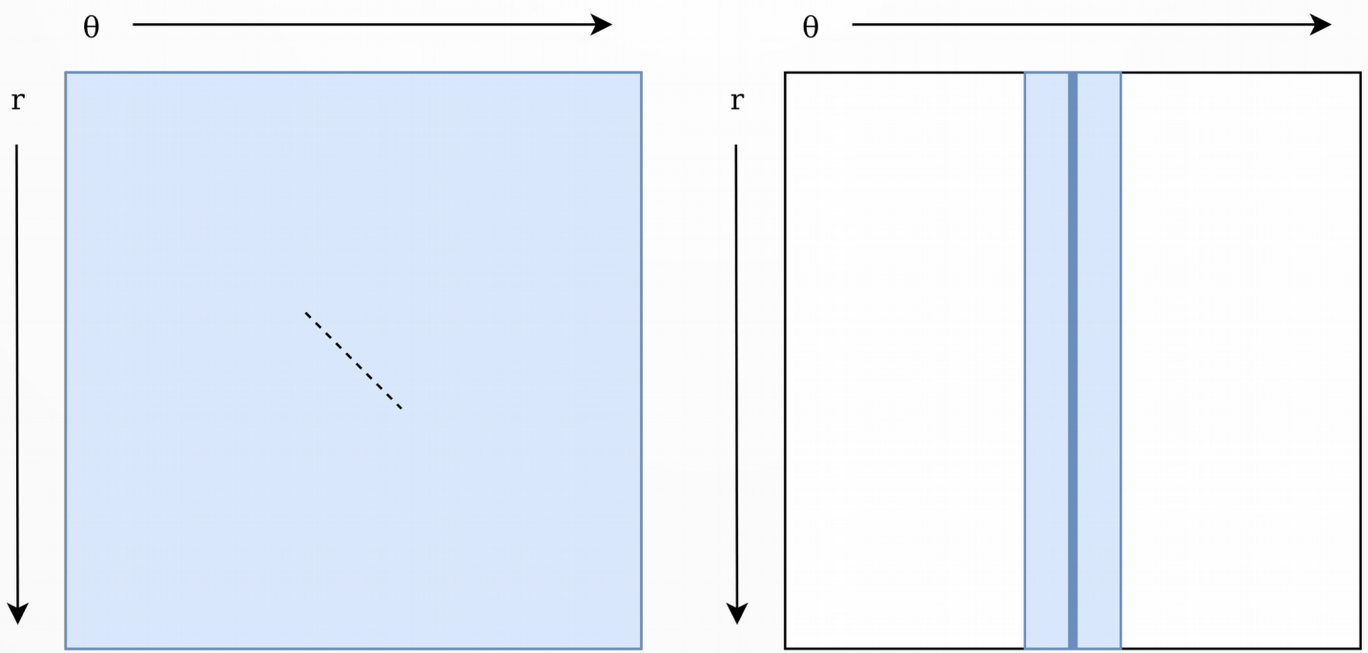


- Objectives

→ Line detection (using a support matrix)
→ For each point P(xp, yp) which belongs to the contour/edge, calculate lines by varying theta (obtaining couples: (rho, theta))
→ Counting the number of occurrences (n) of a particular line (same rho and theta) in the image, If n > threshold : line really exists in the image

Problem

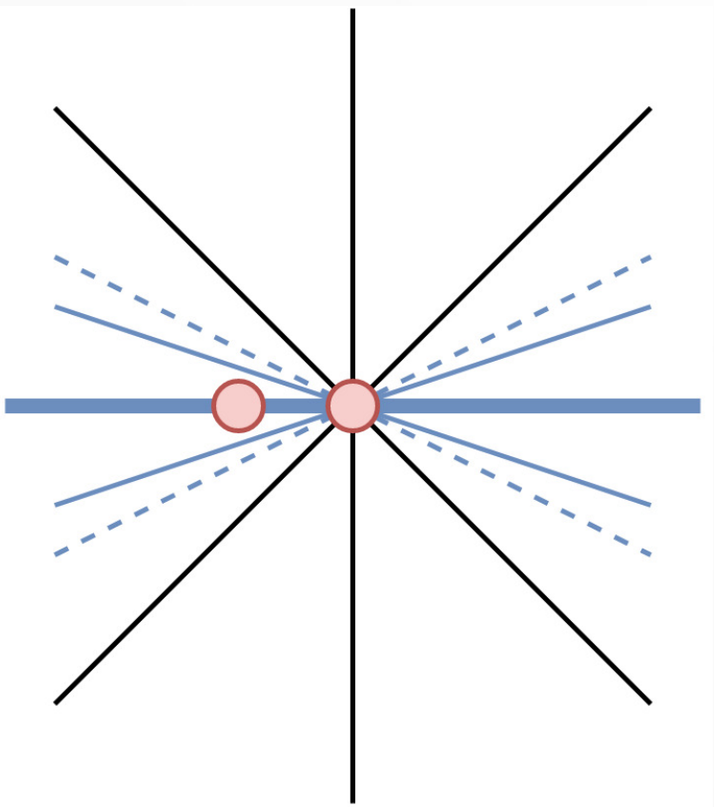
- Basic Implementation: we need to consider all the possible lines - all the possible rho and theta values!



...and if we could try just a subset of all the possible lines?

Solution

- Starting from contour-detection algorithm, we can calculate just few lines



→ Main direction is discovered connecting the new contour pixel with the previous one
→ Lines we care about belong to the range: [main direction – angle; main direction + angle]

Preprocessing

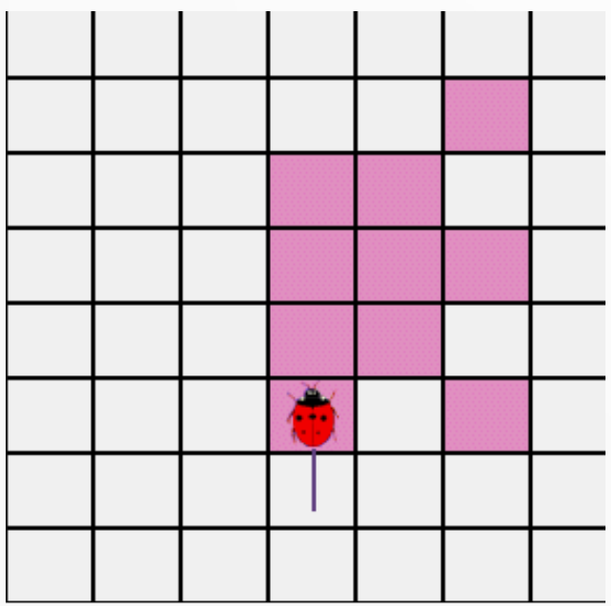
- We start from a whatsoever color image
- Apply Canny edge detector
- This is the starting point for the contour tracing algorithm

The algorithm – contour tracing

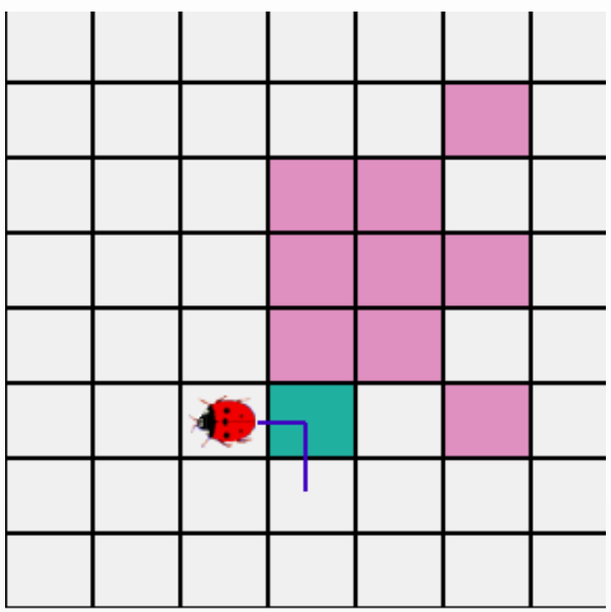
- 2 different algorithms
  - Square Tracing
  - Moore Neighbor
- We need to introduce the concept of direction (north, suoth, east, west)

The algorithm – Square Tracing

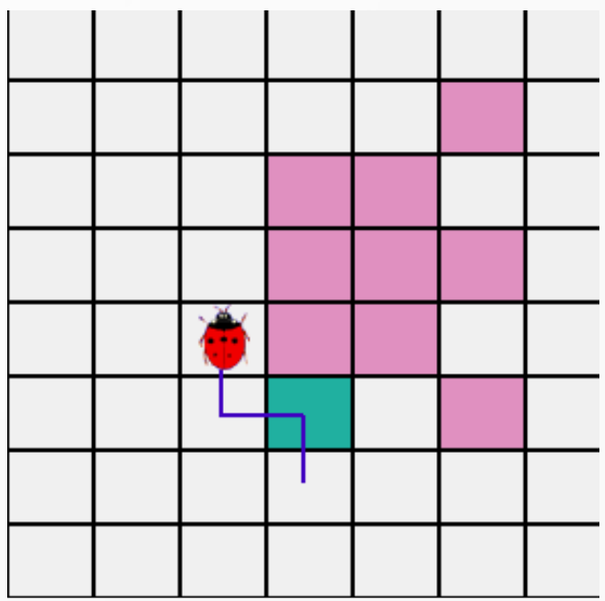
(1) Start from a black pixel (S), turn left and go to the next one
while (current pixel != S):
    (2a) if pixels' black, turn left and go to the next one
    (2b) if pixels' white, turn right and go to the next one



Initially, you are standing on the start pixel
Since the start pixel is a "black" pixel,
you have to turn left



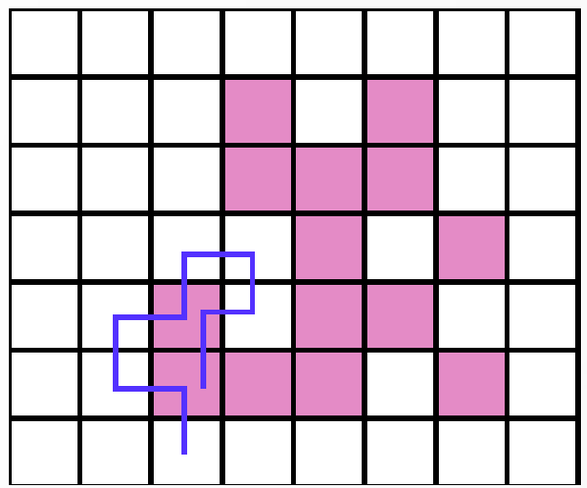
Turn left and mark the start pixel, which
you just walked over, as a boundary
pixel (i.e. green)



Turn right since you were standing on
a white pixel

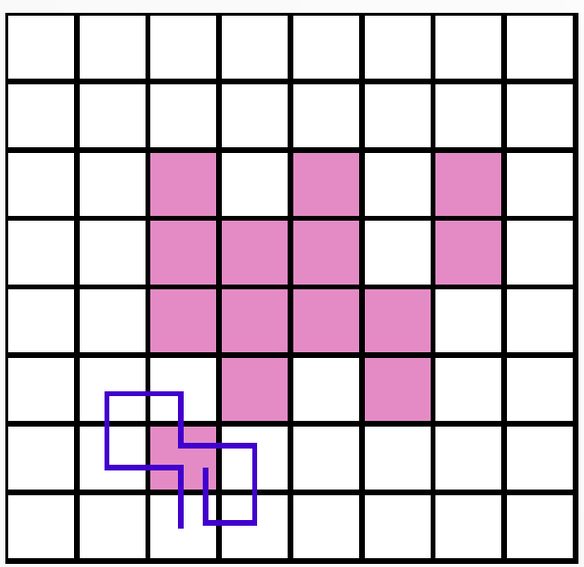
The algorithm – Square Tracing

Stopping criterion could be changed, in order to recognize this kind of pattern (1)



(1)

Jacob's Stopping Criterion: keep going on until you've been walking on the starting pixel, **with the same direction**, twice



(2)

Even introducing this criterion, that's the problem! (2)

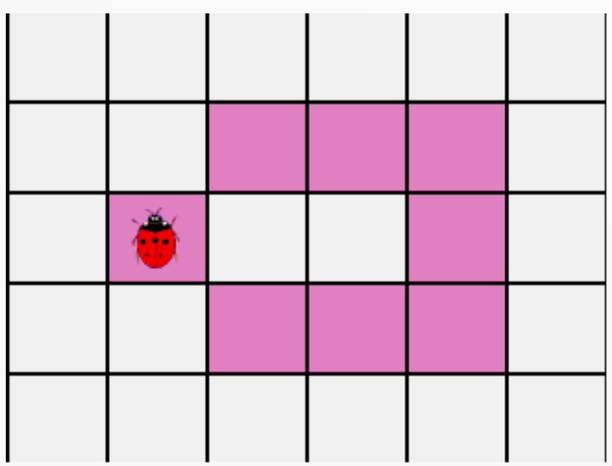
The algorithm – Moore Neighbor

- ...as seen before, Square Tracing algorithm fails to find contours for many patterns
- Moore Neighbor – actually used in our implementation (...and in Matlab as well!)

The algorithm – Moore Neighbor

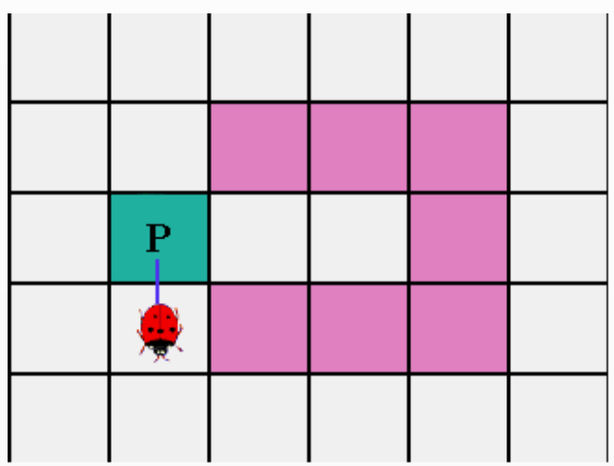
(1) Start from a black pixel (S), Invert direction and go to the next pixel
while (current pixel != S):
    (2) Invert direction and go to the next pixel
    (3) Visit S' Moore neighbors, clockwise, until finding a black pixel, adding it to contour pixels

(1)



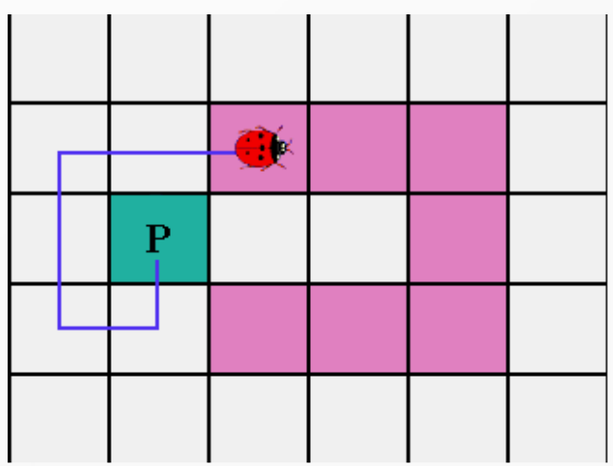
Locate a start pixel

(2)



Backtrack, mark the start pixel as a boundary pixel i.e. green and call it P (current pixel)

(3)

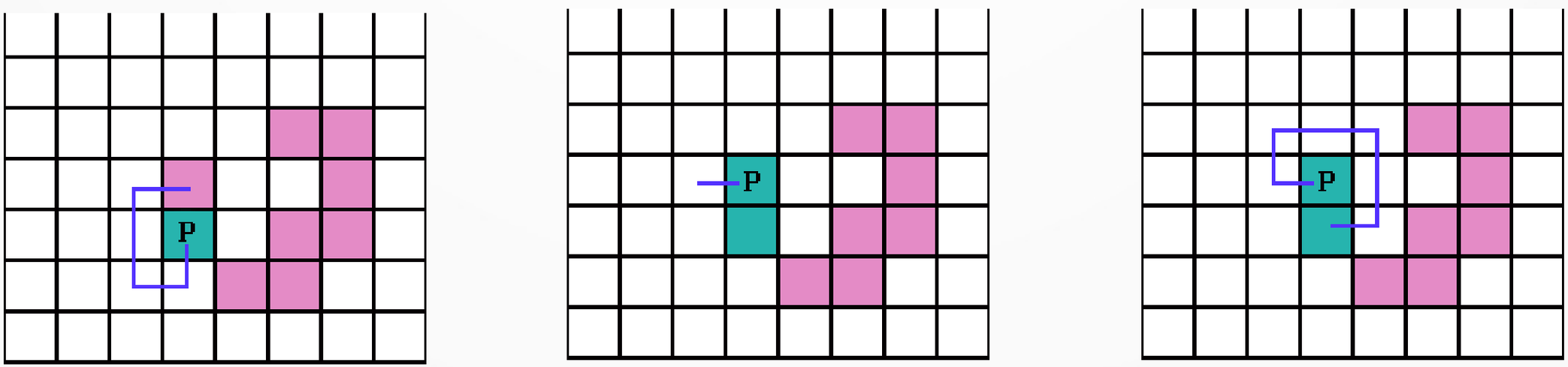


...until you hit a black pixel

The algorithm – Moore Neighbor

- Moore Neighbor – Jacob's Stopping criterion

Introducing this criterion, Moore Neighbor is able to recognize every kind of pattern (made exception for isolated pixel, of course)

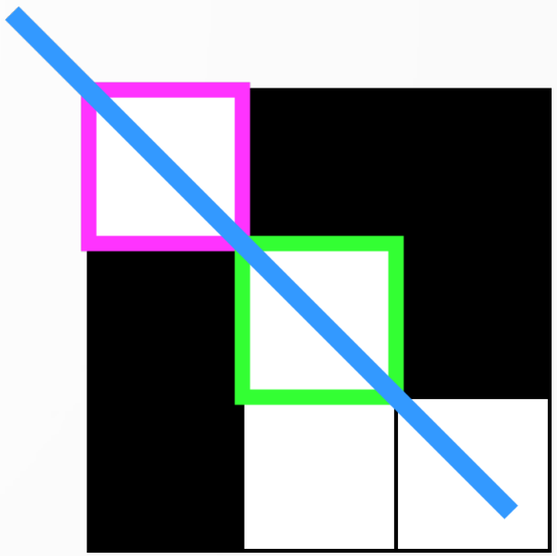


Example without Jacob's stopping criterion

The algorithm – Contour tracing

- How to find more than one contour?
- After finding each contour, we delete contour pixels in the edge map
- Iterate until there’s no edge pixel left

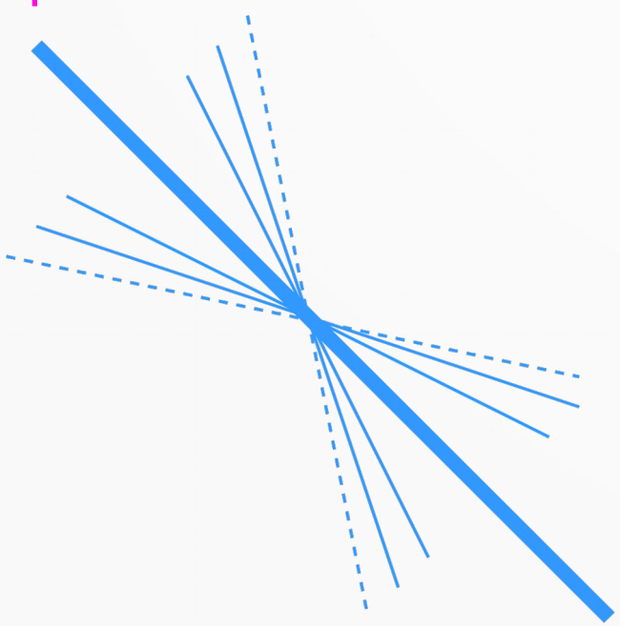
The algorithm – Finding lines



Main direction is calculated as:

$\Theta = \arctan (\Delta y / \Delta x)$

Δy and Δx with respect to current pixel and previous pixel detected

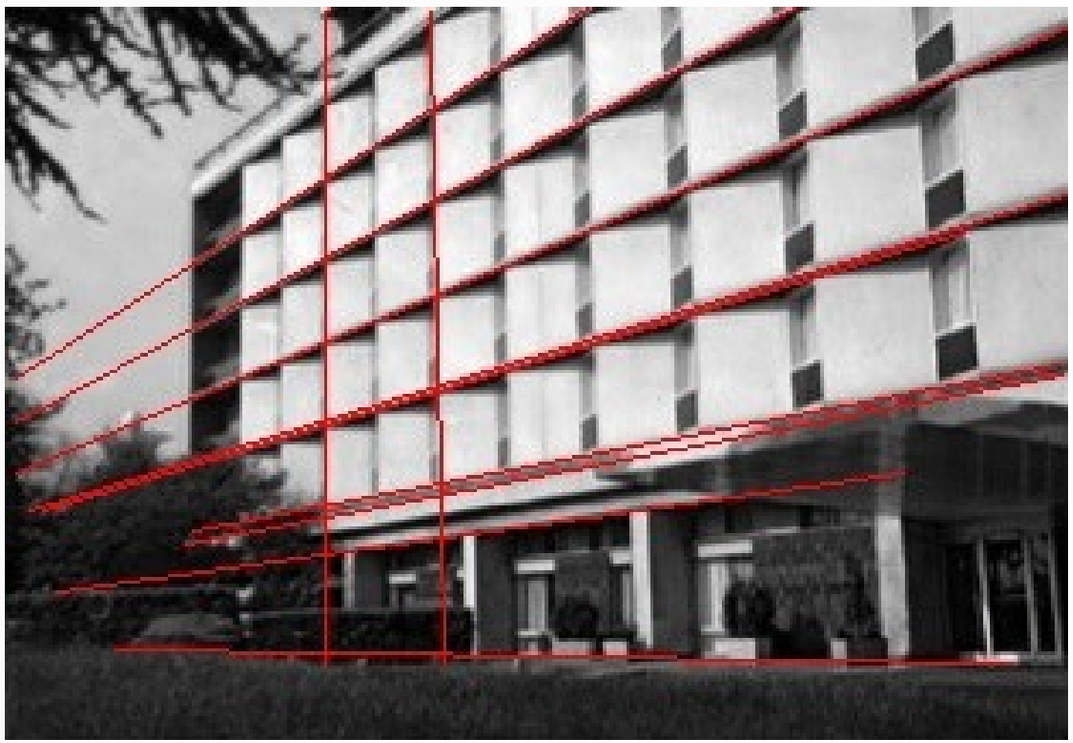


So, it is now possible to try just lines which belong to the blue interval!

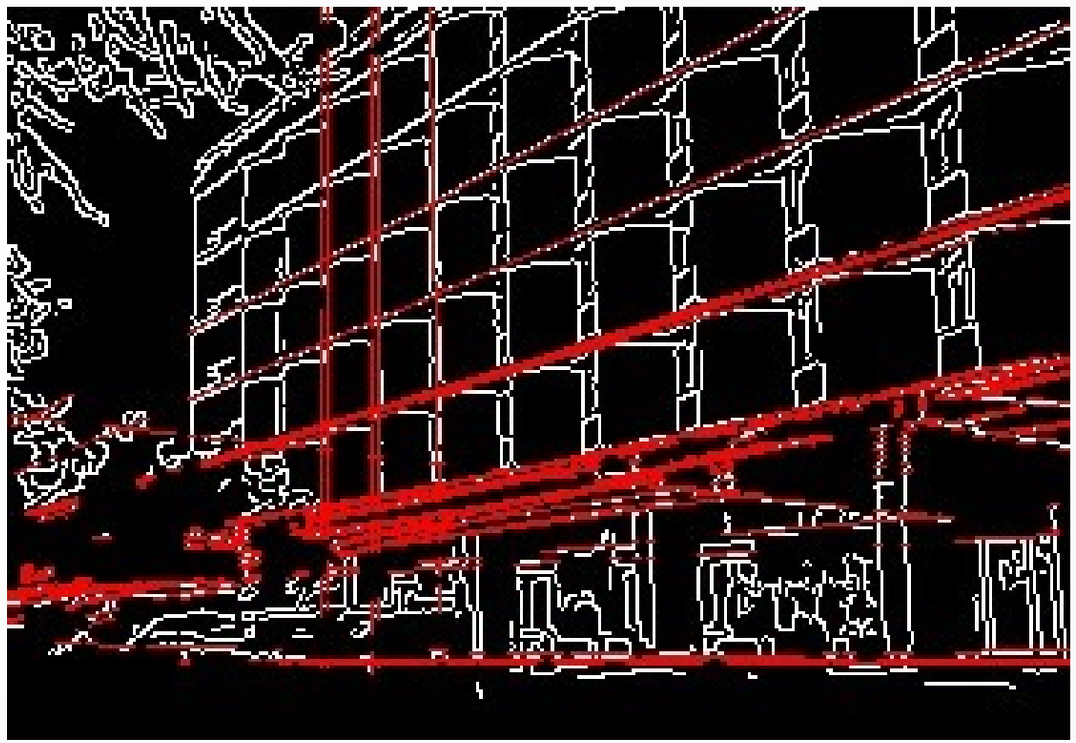
Pseudo-code

```
edge_map = edge_detection(image)

while (not edge_map is empty):
    contour_detection_hough(edge_map)
    delete_last_contour(edge_map)

draw_lines()
```

Performances

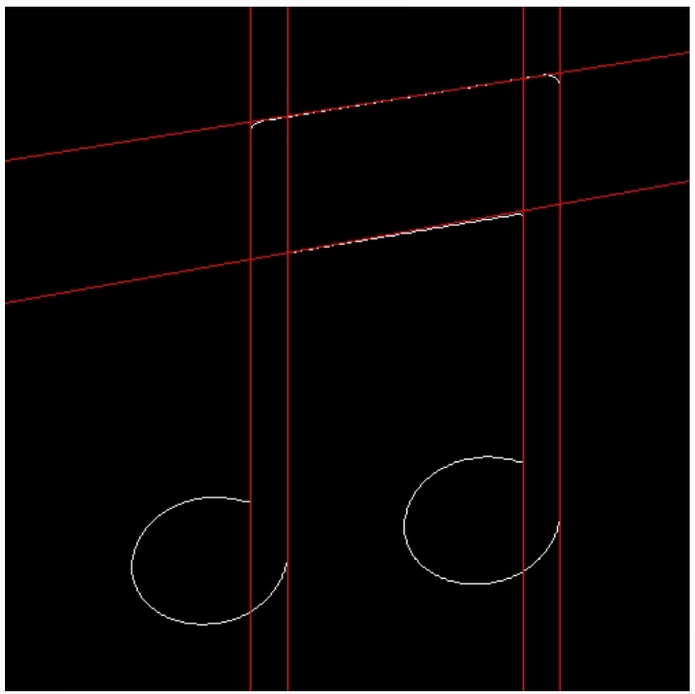


Basic algorithm
27.37 s

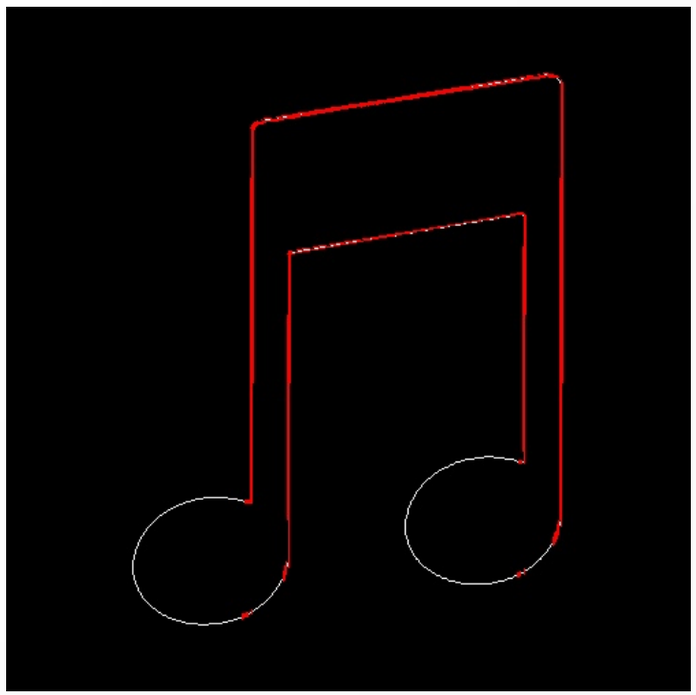


Our algorithm
51.43 s

Performances



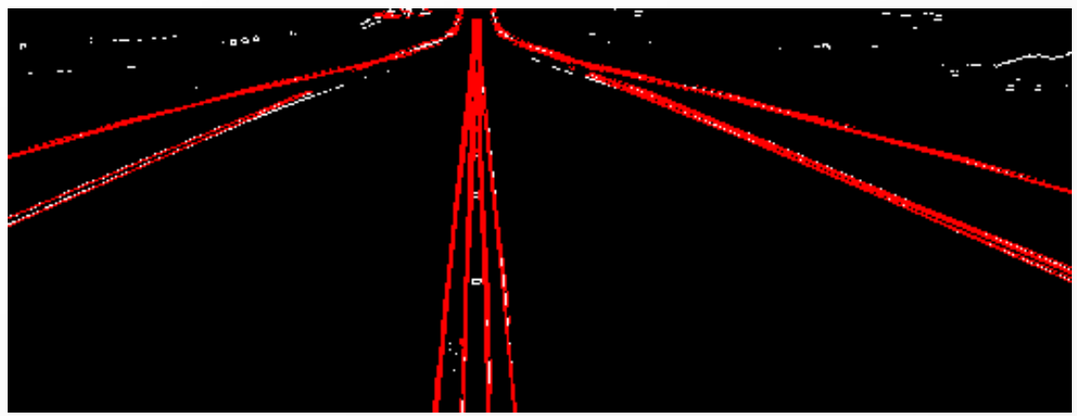
Basic algorithm
8.27 s



Our algorithm
3.09 s

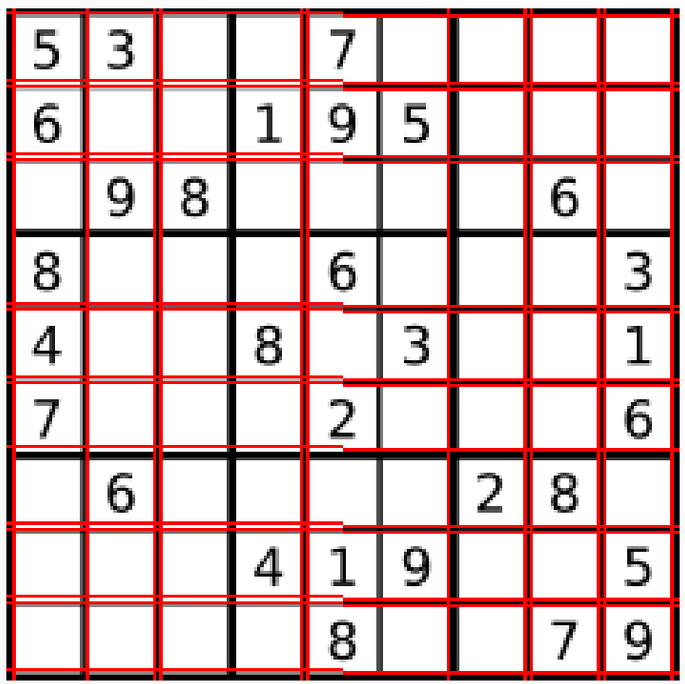
Performances

Basic algorithm
10.77 s

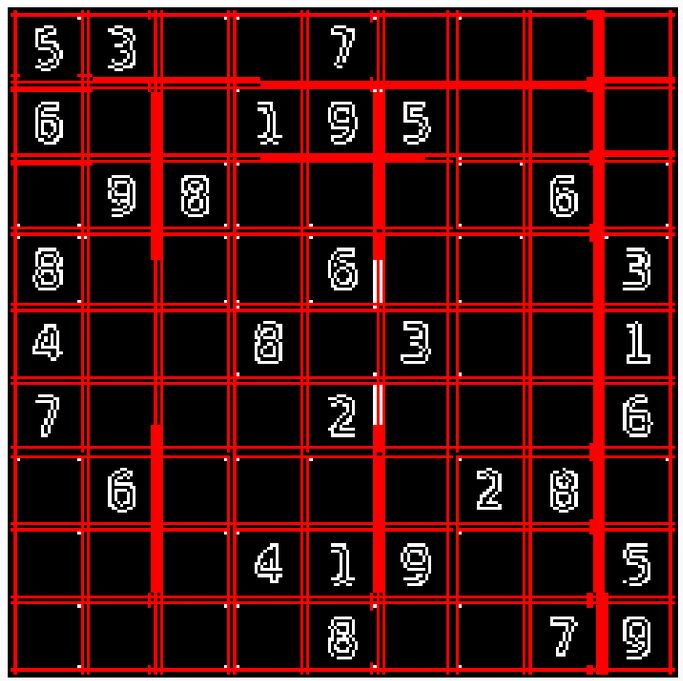


Our algorithm
10.62 s

Performances



Basic algorithm
37.75s



Our algorithm
24.36 s

Issues

- Sensitive to parameters:
  - Canny thresholds
  - Angle of lines
  - Hough threshold
- They have to be chosen with respect to the input image

Improvements

- C/C++ implementation to do a more accurate time comparison
- Automatic and optimal parameters detection

Conclusions

- It has been implemented an algorithm that avoids to generate all lines as done in Hough because that is basically useless
- This approach could lead to a faster Hough and could be farther enhanced with the use of a better contour algorithm
- However, performances should be also compared to more efficient method to do the Hough transform, such as the probabilistic one