Digital Image Processing
Assignment # 3

Submitted by

Ali Akram 260336
Minhaj Khokhar 265439

Submitted to Dr. Shahzor Ahmad

DEPARTMENT OF ELECTRICAL ENGINEERING

College of Electrical and Mechanical Engineering (CEME)

Question

Line Detection via the Hough Transform In this homework, you will implement the Hough Transform to detect straight lines in the provided images, and report results.

- You need not implement the transform for the polar representation of the equation of a straight line and focus only on the slope-intercept form. However, if you feel up to it, you may also go ahead and implement the polar form.
- You need to empirically determine, justify and report the best parameters used such as edge magnitude threshold used to determine edge pixels vs. non-edge pixels, size of the accumulator cells (in terms of pixels) in the Hough space, the min and max range of the Hough parameters a, b, peak detection threshold in the Hough space, etc. You should fix your parameters for all test images given.
- For each image, report the number of lines obtained, the length of smallest and largest line segments (in terms of edge points, i.e., the accumulator cell value), and show the lines obtained (or line segments, if you can do that) superimposed on each (grayscale version of) image. Also show the accumulator space as an image and as a surface plot. Explain your results.
- Compare your results with some built-in / library / freeware function for line detection (whether based on Hough Transform)

Task Distribution

Ali Akram

- Edge detection
- Hough line function
- Analysis
- Report

Minhaj Khokhar

- Peak values function
- Drawing lines on img
- Analysis

Hough transform

Hough transform is a feature extraction method for detecting simple shapes such as circles, lines etc in an image.

Cartesian Coordinates

In image space line is defined by the slope m and the y-intercept b

$$y=mx+b$$

Polar Coordinates

Due to undefined value of slope for vertical lines in cartesian coordinates, we have to move to polar coordinates

$$\rho=x\cos(\theta)+y\sin(\theta)$$

Algorithm

Implementation of our algorithm is as follows:

- Extracting edges of the image using Canny
- initialize parameter space rs, thetas
- Create accumulator array and initialize to zero
- for each edge pixel
```
    for each theta
        calculate r = x cos(theta) + y sin(theta)
        Increment accumulator at r, theta
```
- Find Maximum values in accumulator (lines)
- Extract related r, theta values
- Drawing lines

Edge detection

For edge detection we used canny edge detection however to get better results we used different thresholding for each image to get optimal results

Code

Reading image in opencv and applying canny edge detection

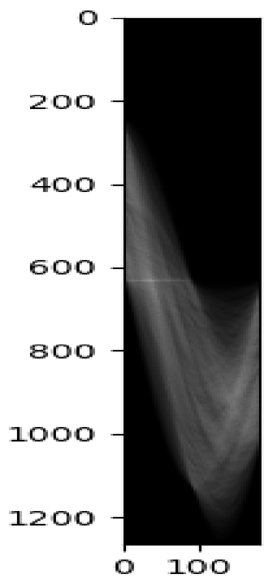
```
images = cv2.imread('Images/airport_inside_0098.jpg')
images = cv2.cvtColor(images, cv2.COLOR_BGR2RGB)
image = cv2.Canny(images, 50, 150, apertureSize=3)
```

we made hough line function which returns accumulator, rho and theta values

```
accumulator, thetas, rhos = houghLine(image)
```

here we showed the hough space of one airport image

```
plt.figure('Original Image')
plt.imshow(images)
plt.set_cmap('gray')
plt.figure('Hough Space')
plt.imshow(accumulator)
plt.set_cmap('gray')
plt.show()
```



Hough line function

It takes img as input and return accumulator of hough, thetas with values between -90 and 90 and values of rho as output

```
def houghLine(image):


    Ny = image.shape[0]
    Nx = image.shape[1]
    Maxdist = int(np.round(np.sqrt(Nx ** 2 + Ny ** 2)))

    thetas = np.deg2rad(np.arange(-90, 90))
    rs = np.linspace(-Maxdist, Maxdist, 2 * Maxdist)

    accumulator = np.zeros((2 * Maxdist, len(thetas)))
    for y in range(Ny):
        for x in range(Nx):
            if image[y, x] > 0:

                for k in range(len(thetas)):

                    r = x * np.cos(thetas[k]) + y * np.sin(thetas[k])
                    accumulator[int(r) + Maxdist, k] += 1

    return accumulator, thetas, rs
```

Peak value function

This function returns range of max values in accumulator cell. We used different thresholding for different images to achieve optimal solution.

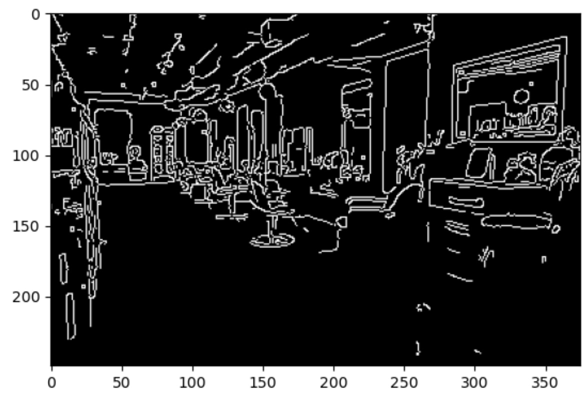
```
a, theta, rho = peak_votes(accumulator, thetas, rhos)
```

Drawing lines

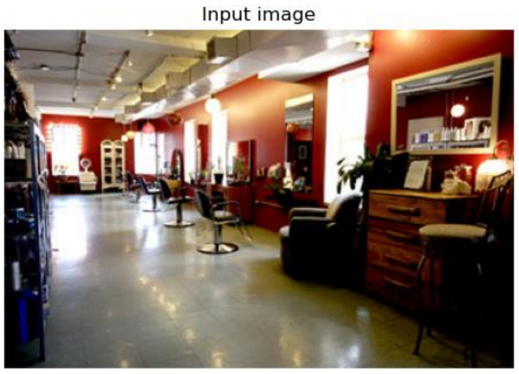
After getting values of rho and thetas we can draw lines based on these values.

```
for rho, theta in zip(rhos, thetas):
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a * rho
    y0 = b * rho
    x1 = int(x0 + 1000 * (-b))
    y1 = int(y0 + 1000 * (a))
    x2 = int(x0 - 1000 * (-b))
    y2 = int(y0 - 1000 * (a))
    cv2.line(images, (x1, y1), (x2, y2), (0, 0, 255), 7)
```

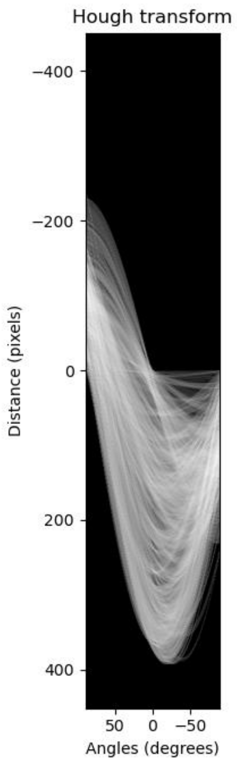
Results



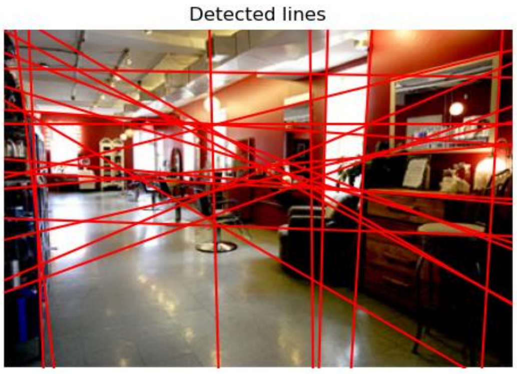
0
50
100
150
200
0
50
100
150
200
250
300
350



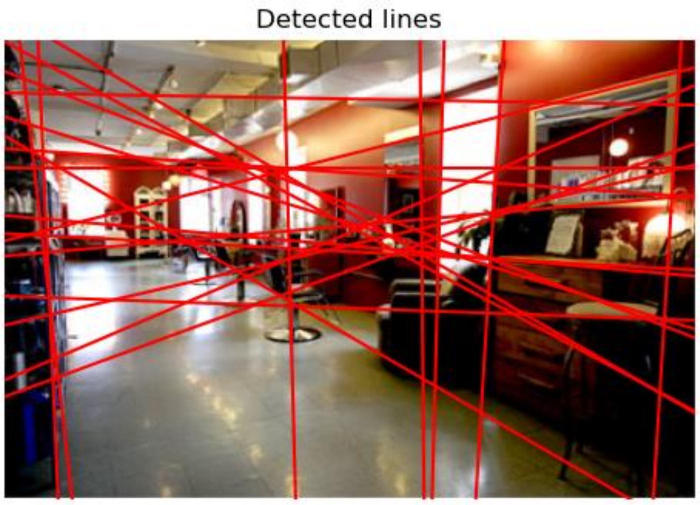
Input image



Hough transform
−400
−200
0
200
400
Distance (pixels)
50
0
−50
Angles (degrees)



Detected lines



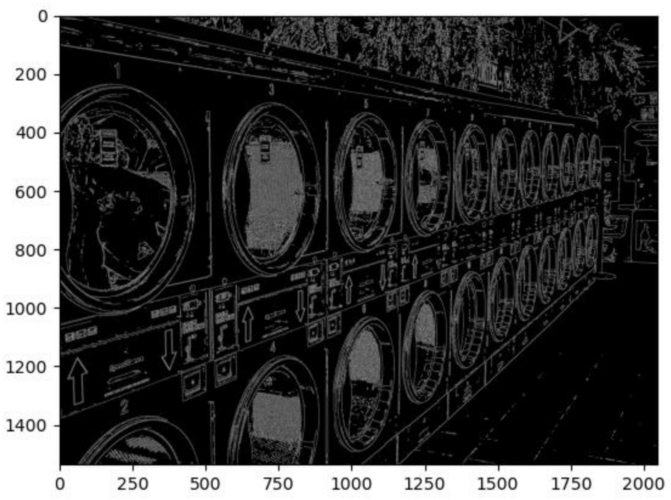
Detected lines

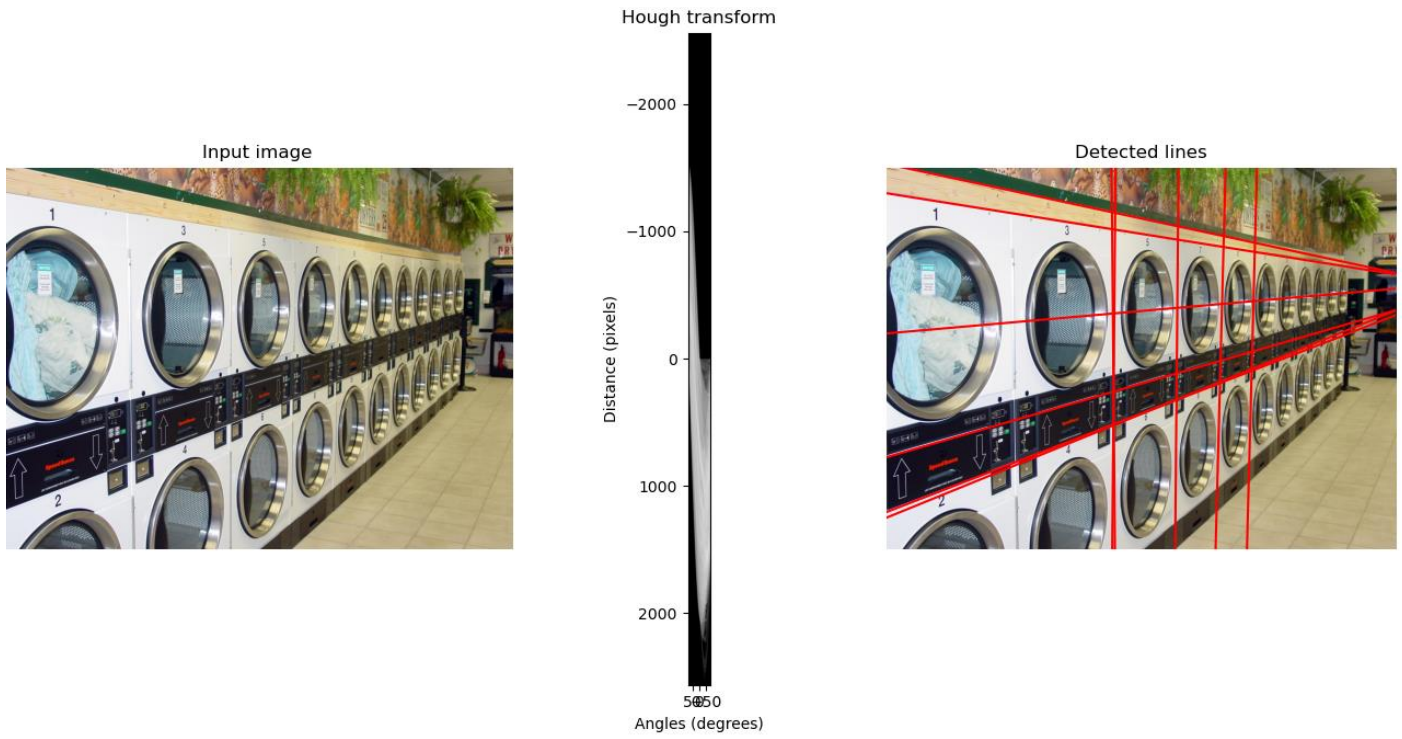
Analysis

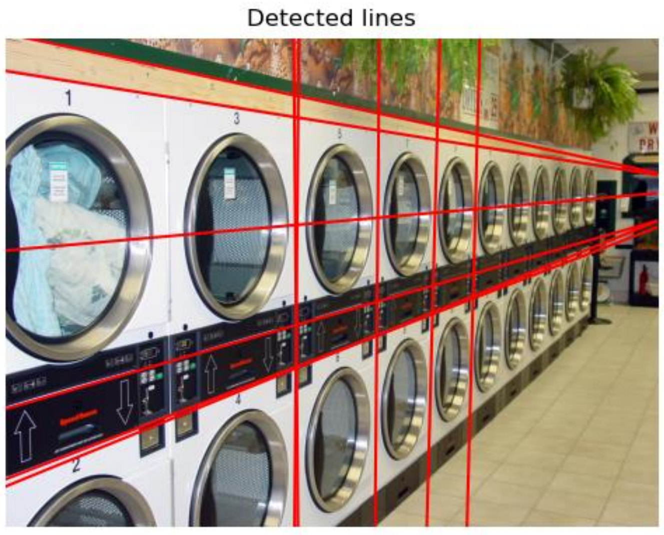
In above results we saw that our algorithm detected 20 lines. The result varies according to different parameters. We also saw that for a small line detection in an image it’s corresponding line is drawn on entire image instead of segment. This problem can be solved by specifying start and end points of each line in our algorithm or by using inbuilt function. The output pictures in results are as follows:

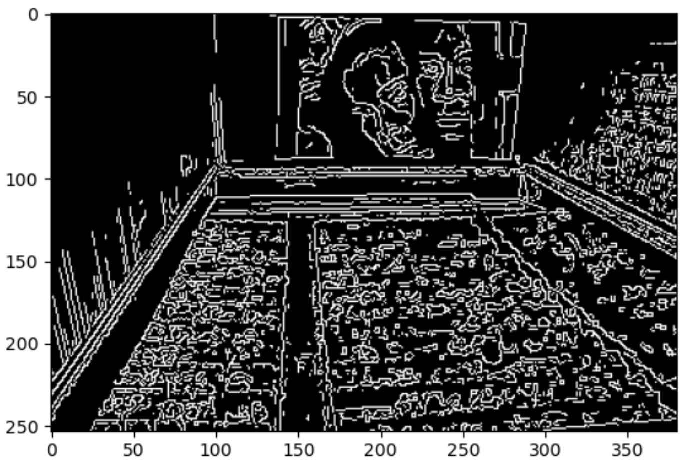
1. The first image is output of edge detection.
2. The second image is showing input image, it’s hough transform and output image.
3. The output image is zoomed and is also our third image in which we have drawn line on input colored image.

Similarly results of other pictures are shown below.

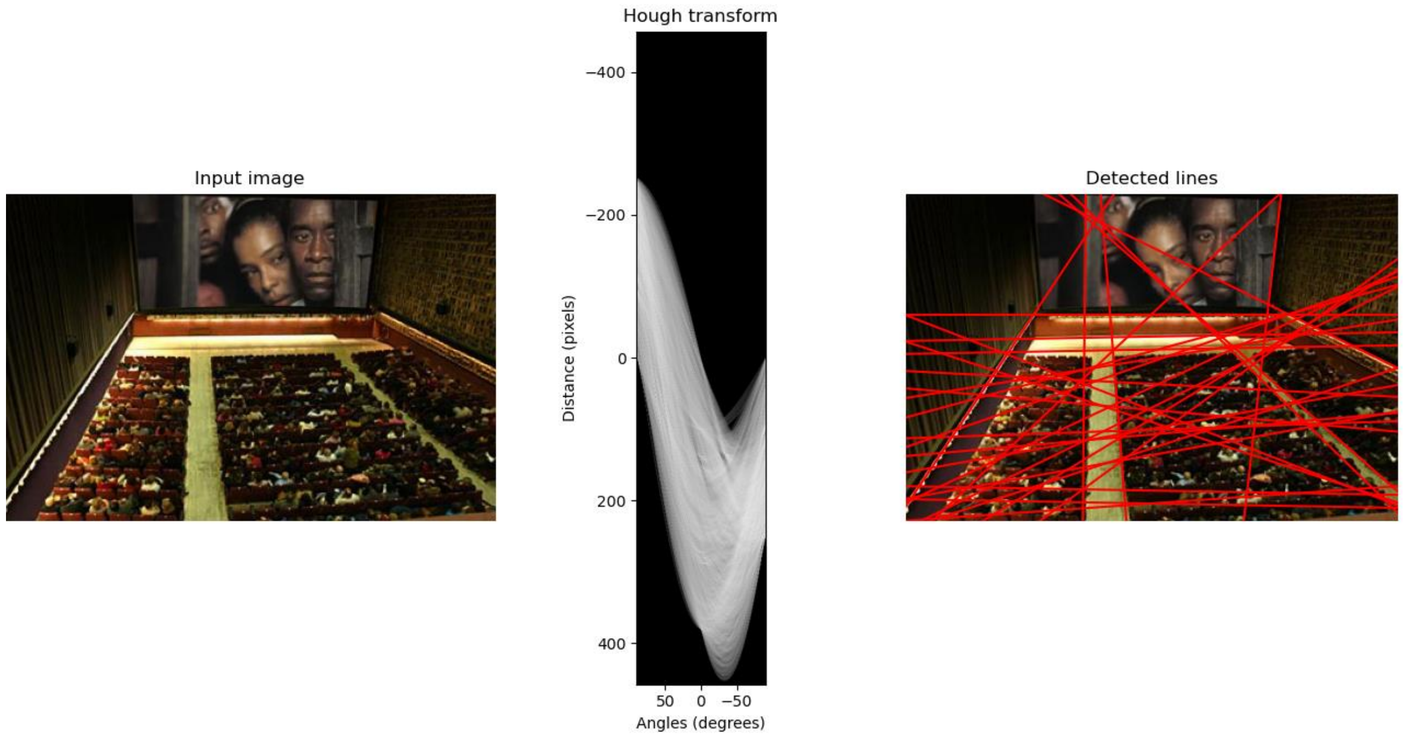

0
200
400
600
800
1000
1200
1400
0
250
500
750
1000
1250
1500
1750
2000


Input image
Hough transform
−2000
−1000
0
1000
2000
Distance (pixels)
50
0
50
Angles (degrees)
Detected lines

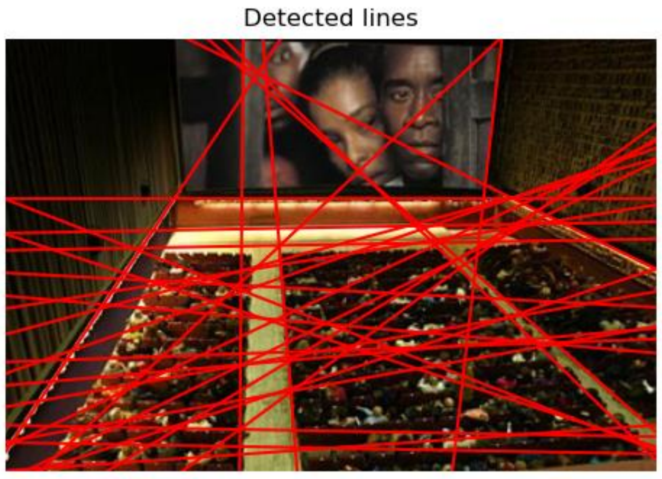

Detected lines



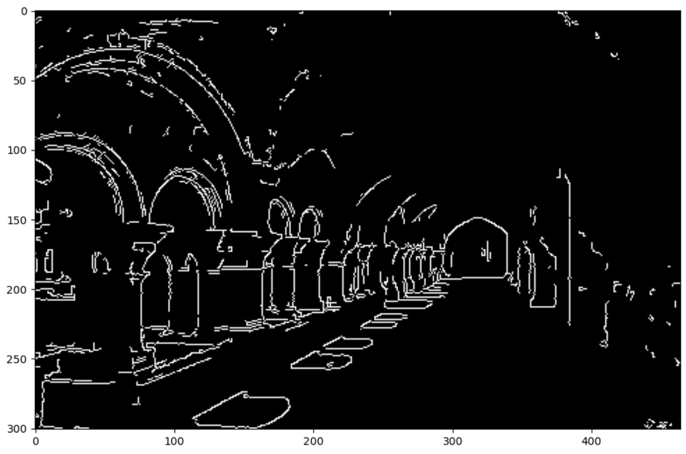
0
50
100
150
200
250
0
50
100
150
200
250
300
350

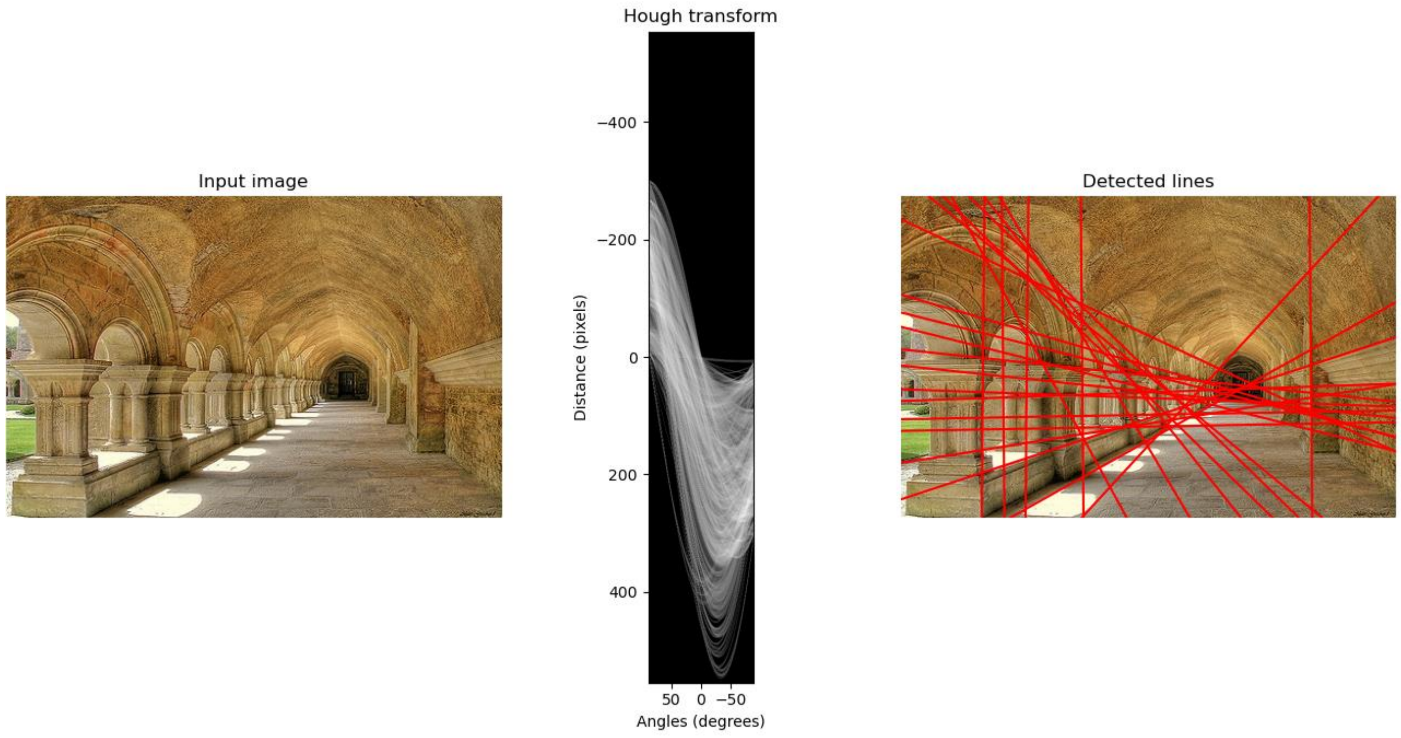


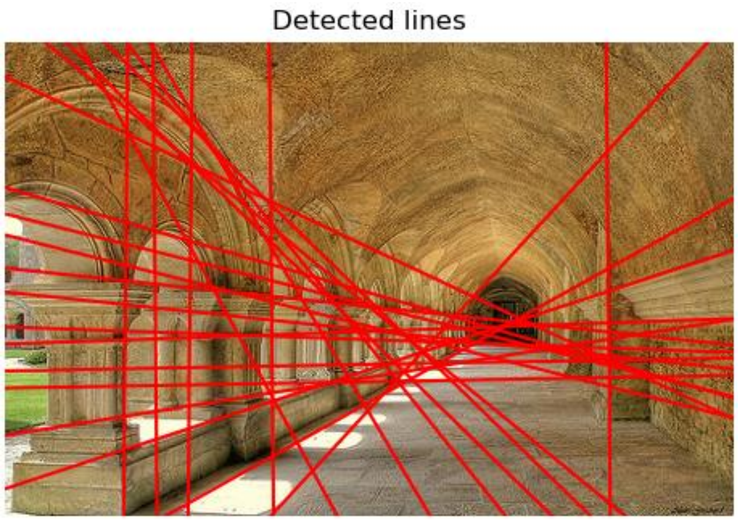
Input image
Hough transform
Detected lines
−400
−200
0
200
400
Distance (pixels)
50
0
−50
Angles (degrees)

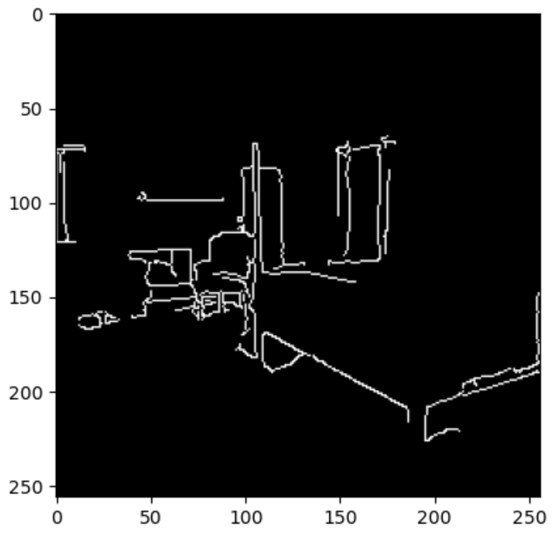


Detected lines

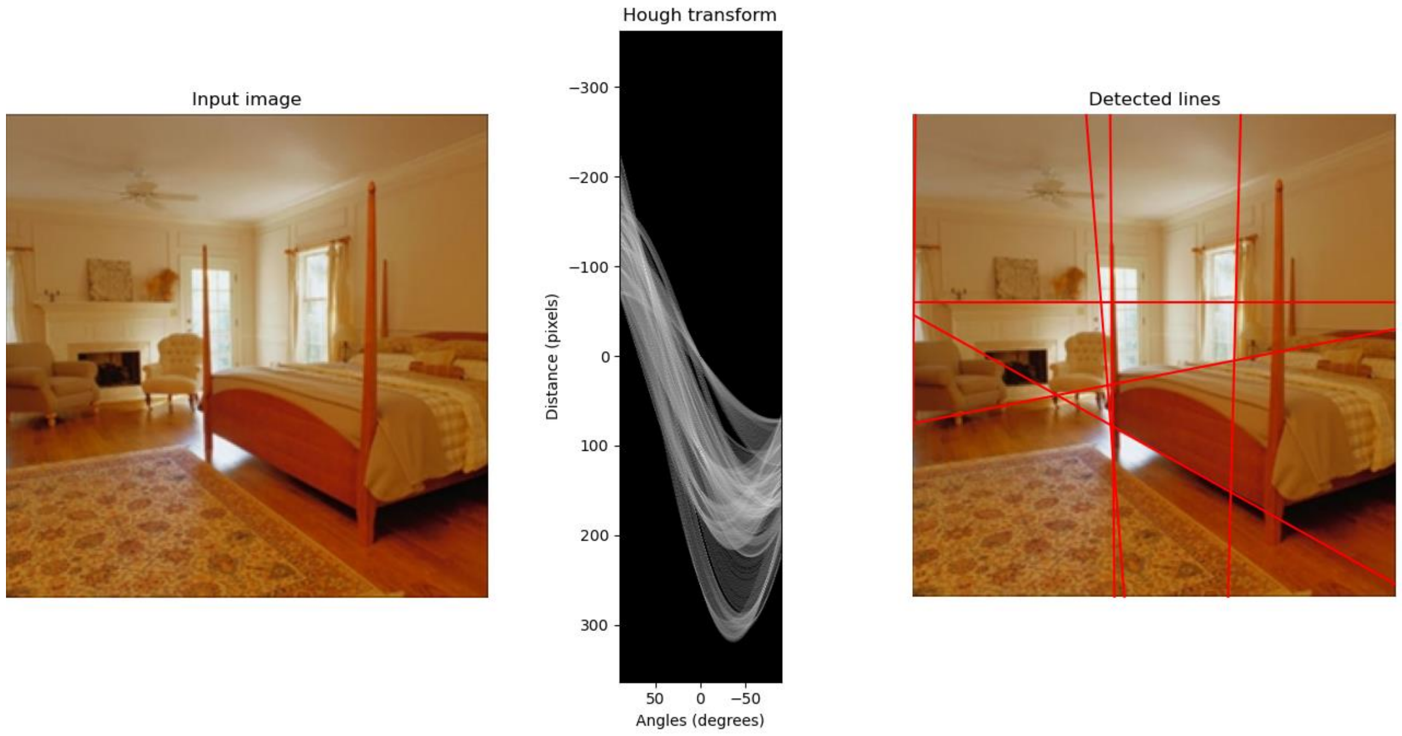

0
50
100
150
200
250
300
0
100
200
300
400


Input image
Hough transform
−400
−200
0
200
400
Distance (pixels)
50
0
−50
Angles (degrees)
Detected lines

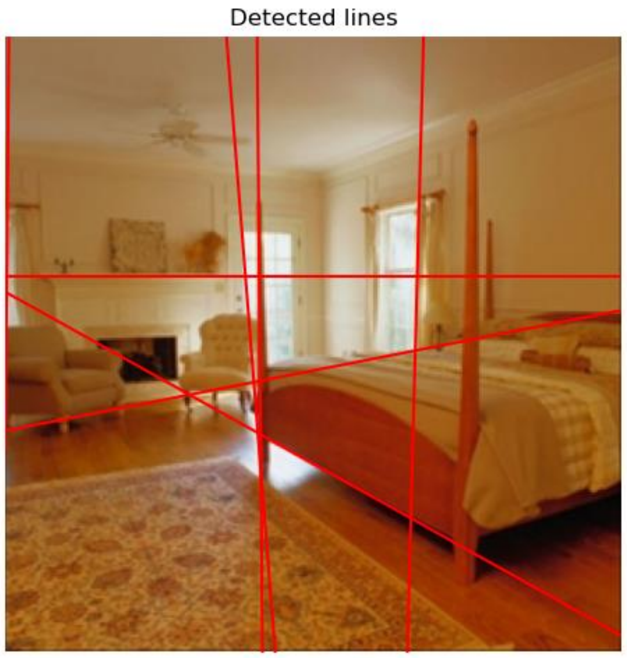

Detected lines



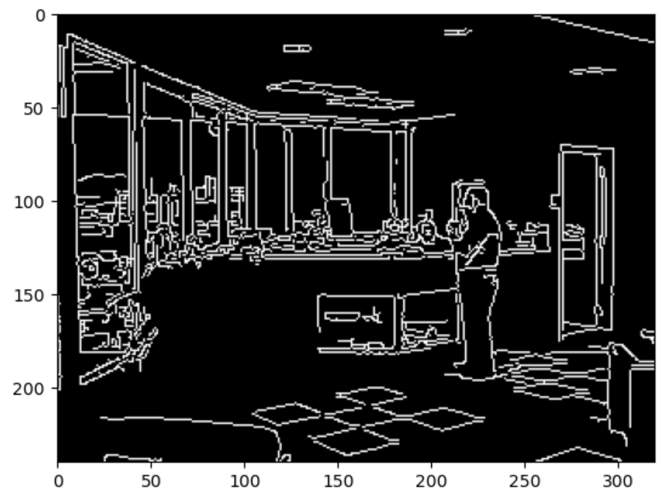
0
50
100
150
200
250

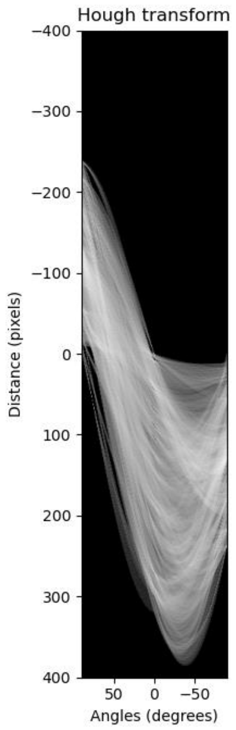


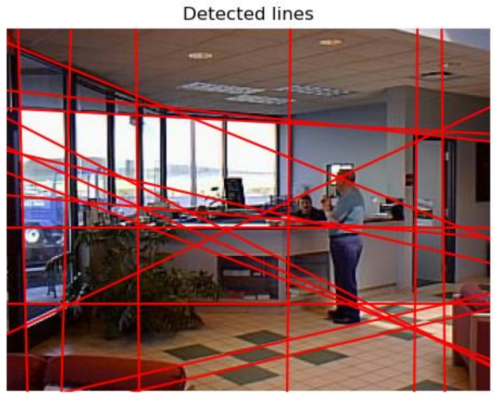
Input image
Hough transform
Detected lines
−300
−200
−100
0
100
200
300
Distance (pixels)
50
0
−50
Angles (degrees)



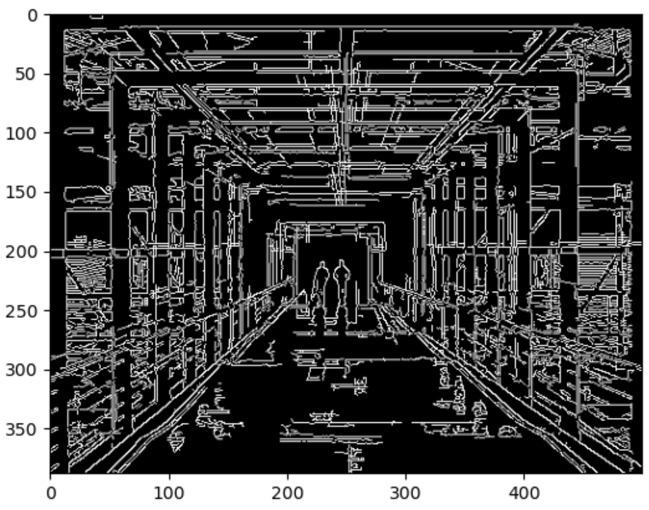
Detected lines



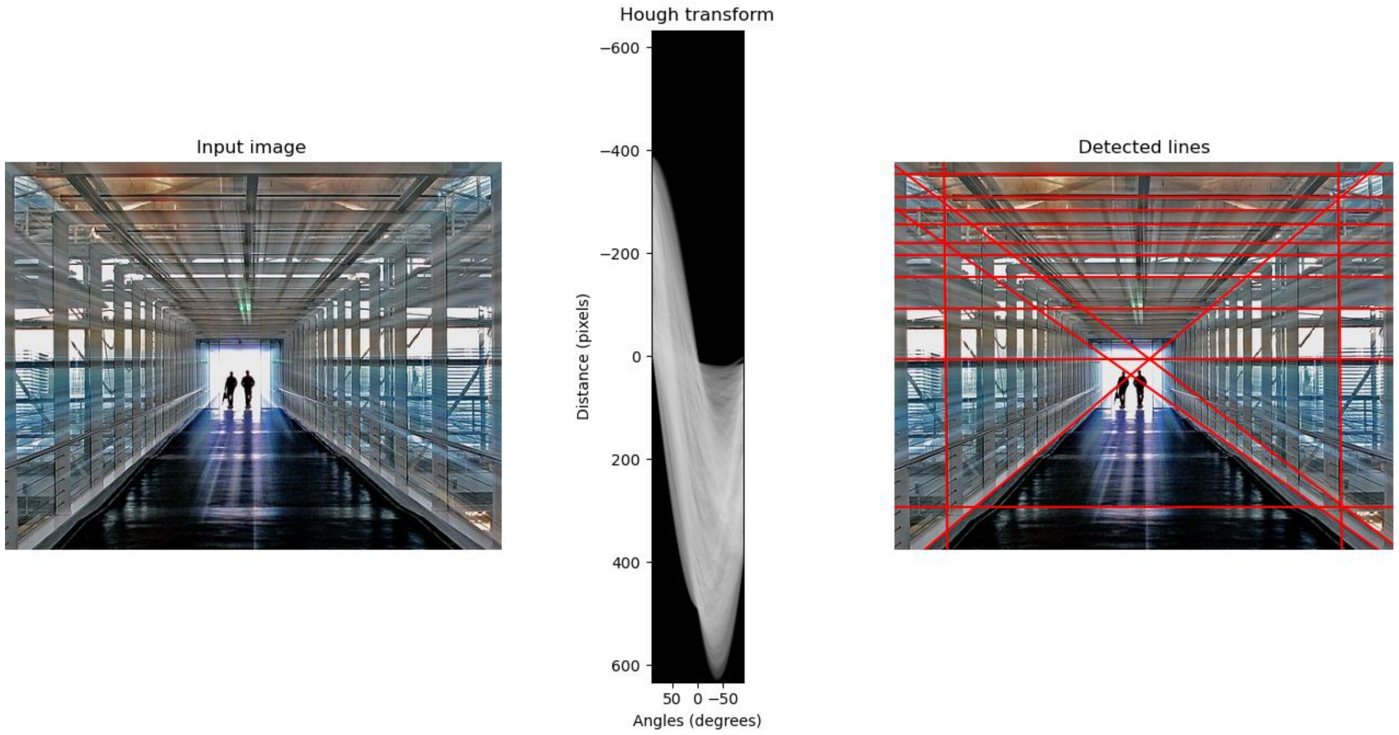




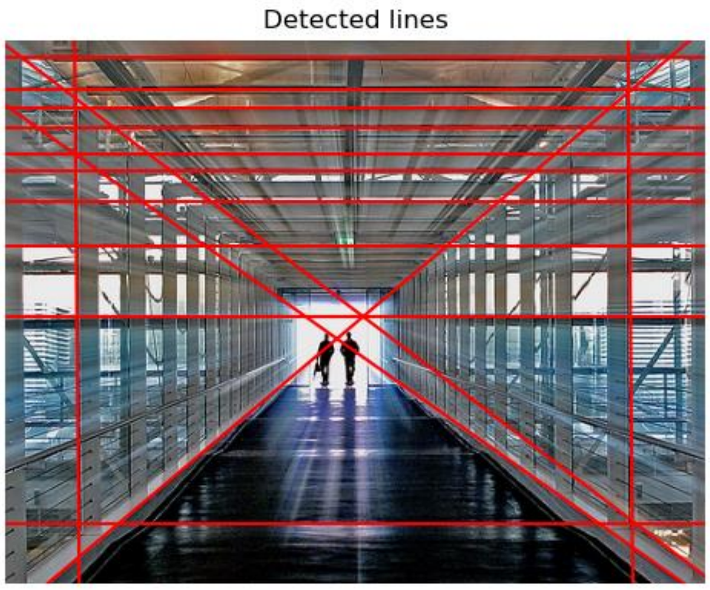
Detected lines



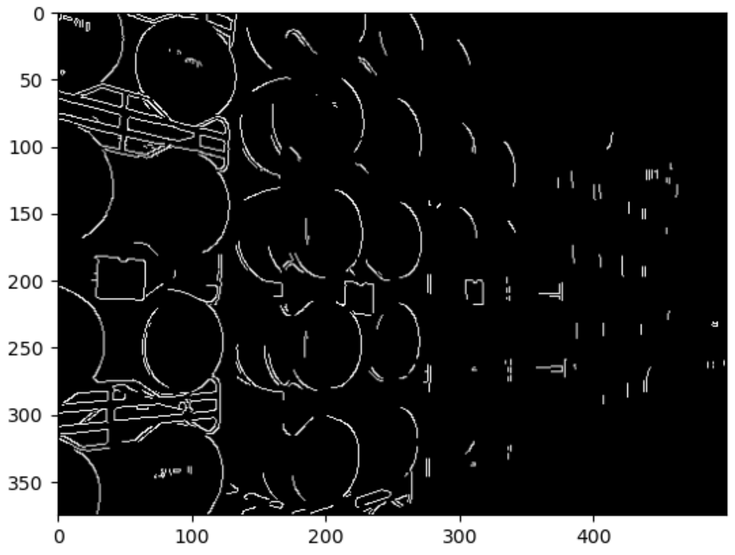
0
50
100
150
200
250
300
350
0
100
200
300
400



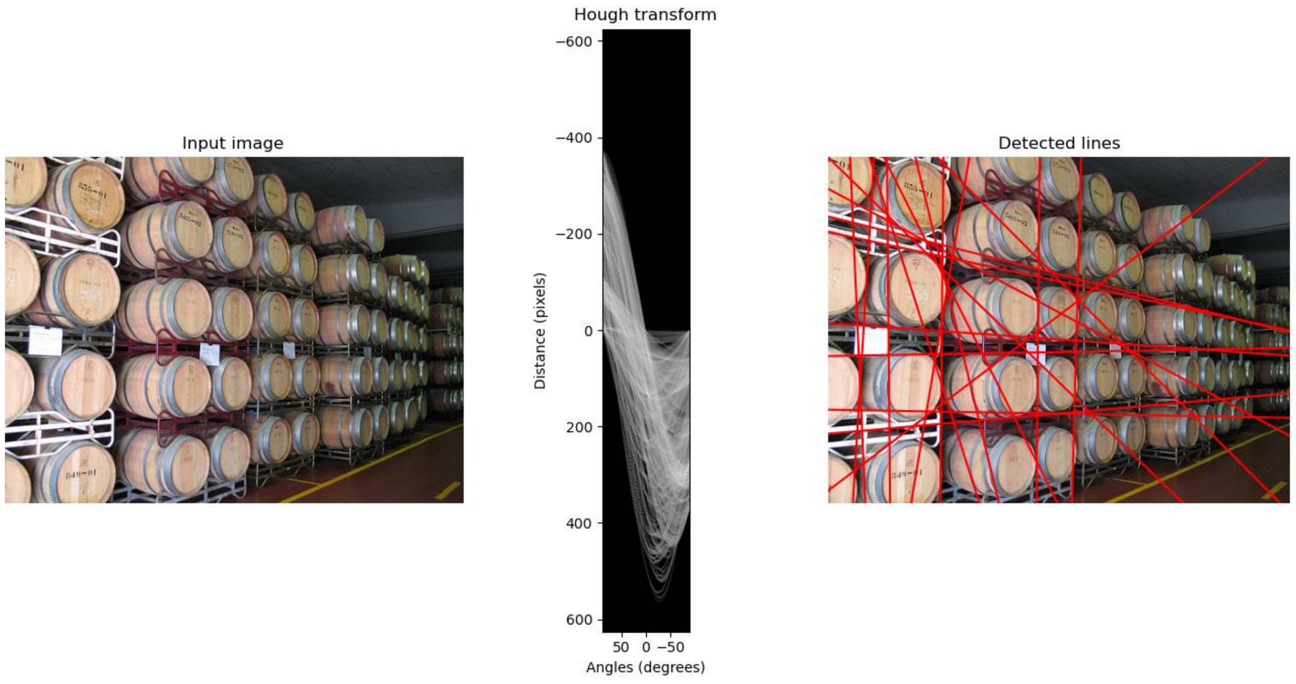
Input image
Hough transform
−600
−400
−200
0
200
400
600
Distance (pixels)
50
0
−50
Angles (degrees)
Detected lines



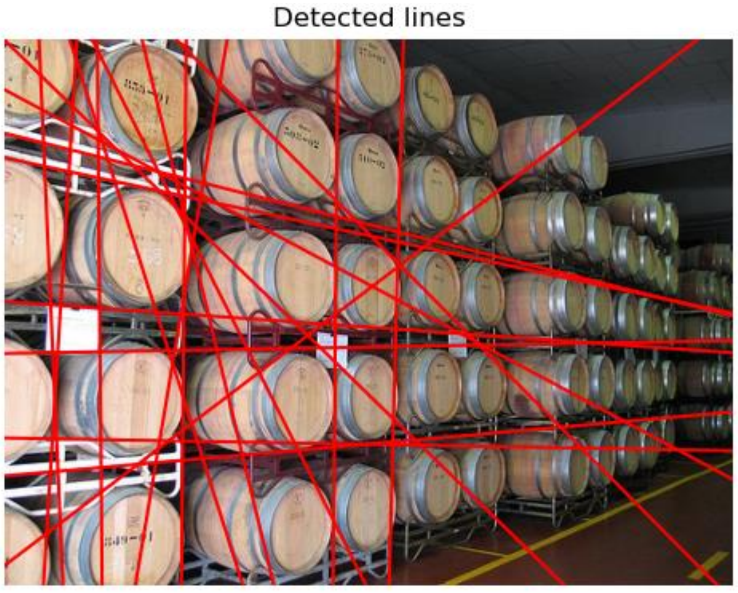
Detected lines



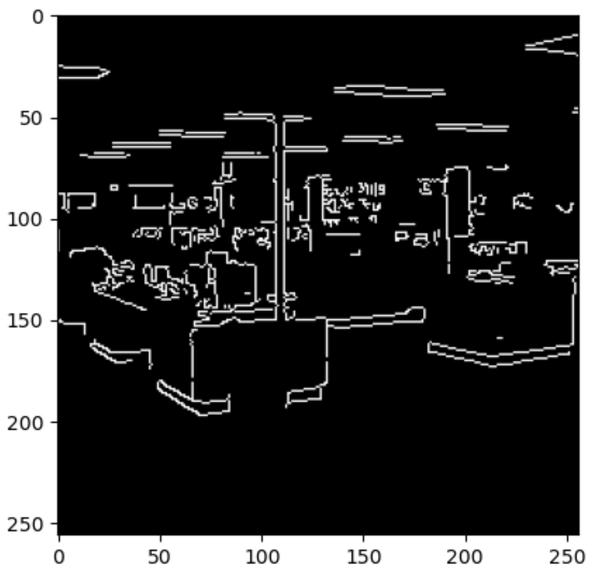
0
50
100
150
200
250
300
350
0
100
200
300
400

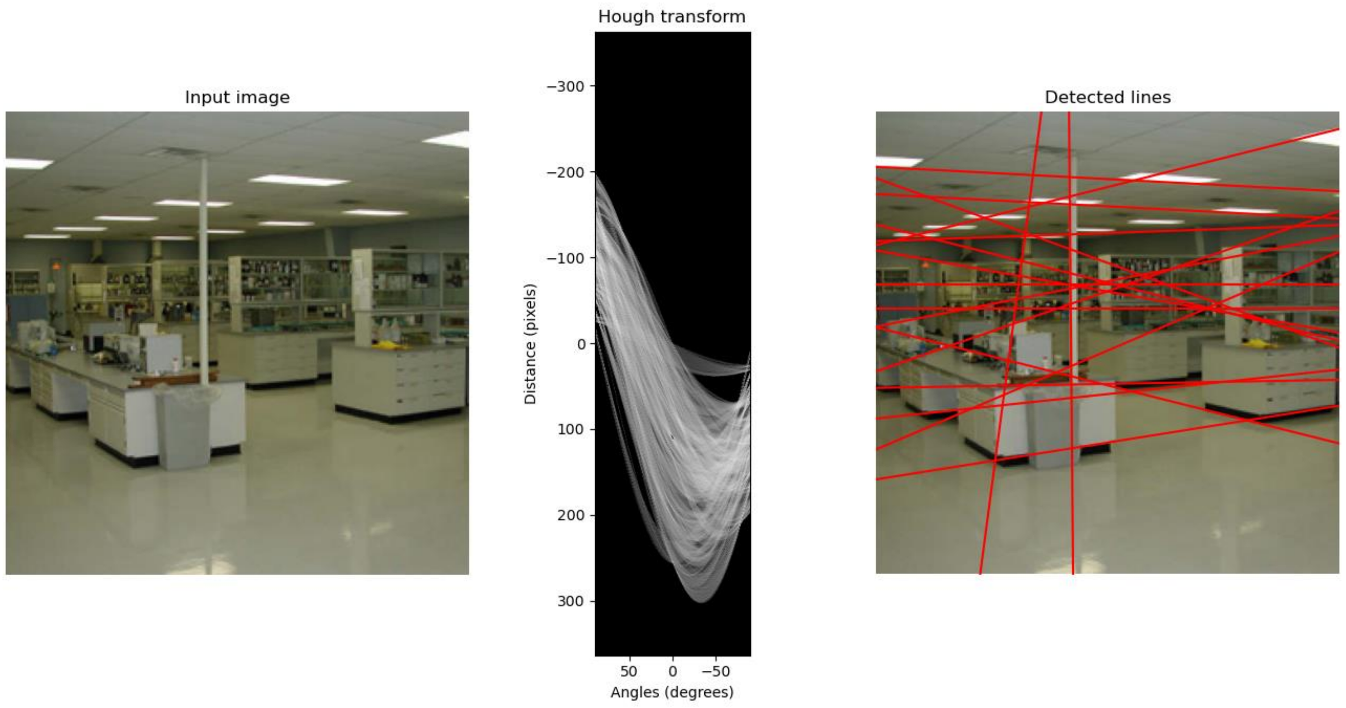


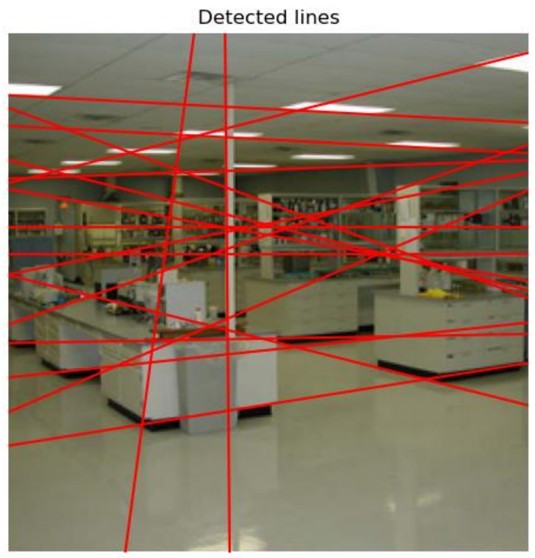
Input image
Hough transform
−600
−400
−200
0
200
400
600
Distance (pixels)
50
0
−50
Angles (degrees)
Detected lines



Detected lines


0
50
100
150
200
250
0
50
100
150
200
250


Input image
Hough transform
Distance (pixels)
Angles (degrees)
−300
−200
−100
0
100
200
300
50
0
−50
Detected lines


Detected lines