Computer Vision
Assignment 6

Alireza Moradi

November 1, 2020

1

This is the hough transformation of 2 intersecting lines. 2 lines because there are 2 points that the waves intersect, And intersecting because they have different θ's so they will intersect somewhere in the space.

2

Because we have to estimate k for circle and we need 3 points to define a circle, There will be a slight change in the formula.

- $w = 0.4$
- $p = 0.99$
- $k = \frac{\log(1-p)}{\log(1-w^3)} \Rightarrow k = \frac{\log(1-0.99)}{\log(1-0.4^3)} \Rightarrow k \simeq 70$

3

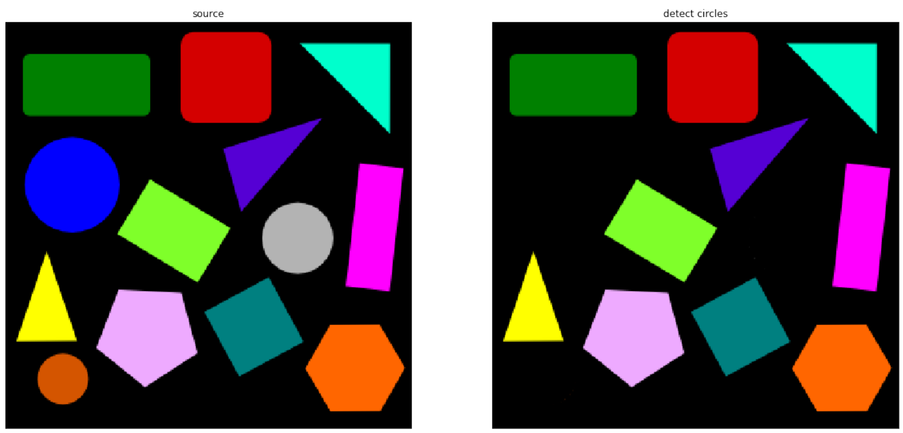
1. LSD's goal is to find the start and end of lines in the image.
2. LSD defines each line by 4 parameters $((x_0, y_0), (x_1, y_1))$ instead of 2 (ρ and θ).
3. LSD's main advantage is that it uses gradient direction.

4 Implementations

4.1 Hough Algorithm

4.1.1 Circles Detection

The key parameter in `cv2.HoughCircles` is `param2` which is `accumulator threshold`. Basically, the higher it is the less circles you get. And these circles have a higher probability of being correct. The best value is different for every image.
Source 1 Source 2

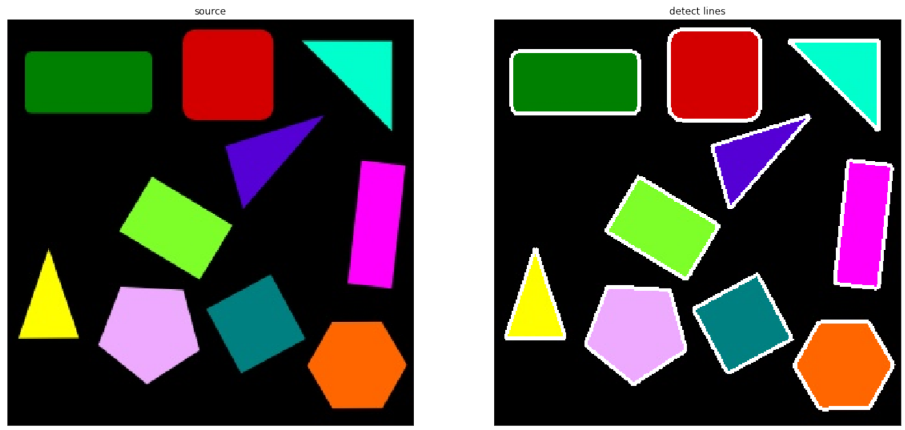



```python
def remove_circles(image):
    '''
    Returns the image which circles have been removed.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        out_img (numpy.ndarray): The result image.
    '''
    out_img = image.copy()

    #Writer your code here
    out_img = cv2.cvtColor(out_img, cv2.COLOR_BGR2GRAY)
    out_img_blurred = cv2.GaussianBlur(out_img,(3,3),1)

    detected_circles = cv2.HoughCircles(out_img_blurred,cv2.HOUGH_GRADIENT ,1, 20, param1 = 100,
                param2 = 30)

    detected_circles = np.around(detected_circles)

    for pt in detected_circles[0, :]:
        a, b, r = pt[0], pt[1], pt[2]

        for i in range(len(image)):
```

```python
            for j in range(len(image[i])):
                if np.sqrt(np.square(i - b) + np.square(j - a)) <= (r + (r/15)):
                    image[i][j] = 0

    return image
```

4.1.2 Line Detection

For this part I applied `Canny` algorithm to detect edges and the applied `Hough` algorithm to the results of Canny to detect lines.
Source




```python
def detect_lines_hough(image):
    '''
    Returns the image which lines have been detected.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        out_img (numpy.ndarray): The result image.
    '''
    out_img = image.copy()

    #Writer your code here

    out_img = cv2.cvtColor(out_img,cv2.COLOR_BGR2GRAY)
    out_img_blurred = cv2.GaussianBlur(out_img,(3,3),1)

    edges = cv2.Canny(out_img_blurred, 30, 110)

    rho = 0.05  # distance resolution in pixels of the Hough grid
    theta = np.pi / 180  # angular resolution in radians of the Hough grid
    threshold = 1  # minimum number of votes (intersections in Hough grid cell)
    min_line_length = 0.5  # minimum number of pixels making up a line
    max_line_gap = 2  # maximum gap in pixels between connectable line segments

    # Run Hough on edge detected image
    # Output "lines" is an array containing endpoints of detected line segments
    lines = cv2.HoughLinesP(edges, rho, theta, threshold, np.array([]),
                        min_line_length, max_line_gap)

```

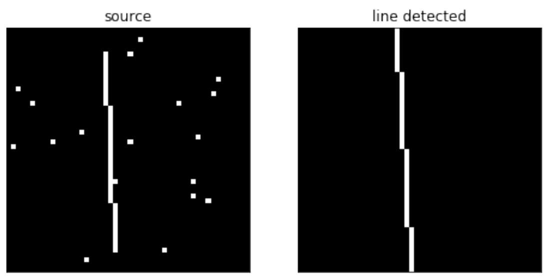
```python
    for line in lines:
        for x1,y1,x2,y2 in line:
            cv2.line(image,(x1,y1),(x2,y2),(255,255,255),2)

    return image
```

4.2 RANSAC

These are the steps I went down:

1. Find all the white points in the image.
2. Randomly select 2 points from the output of step 1.
3. Measure the distance of all the points from the line crossing these 2 points and count and keep those whose distance is below a certain threshold.
4. Replace if this is a better model.
5. Repeat steps 2 to 4 for certain number of iterations.
6. Calculate rho and theta from the best model and all the inliers.

Source

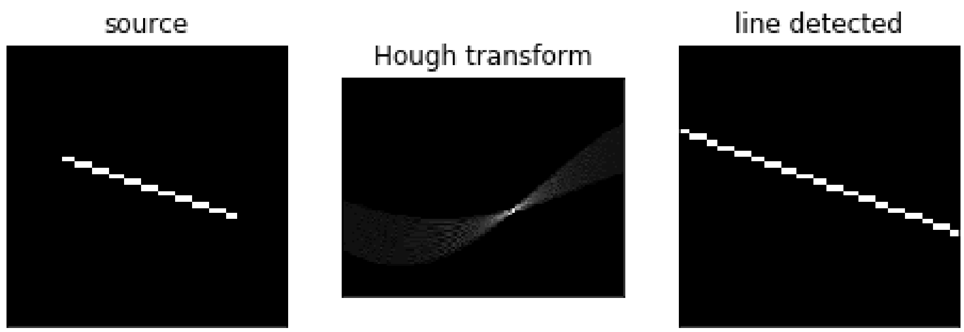



```
def find_points(h,w,img):
    points = []
    for i in range(h):
        for j in range(w):
            if img[i][j] != 0:
                points.append((float(i),float(j)))
    return points

def find_line_distance(p1,p2,p):
    x2,y2 = p2
    x1,y1 = p1
    x0,y0 = p
    return np.abs((y2-y1)*x0 - (x2-x1)*y0 + x2*y1 - y2*x1)/np.sqrt(np.square(y2-y1)+np.square(x2-x1))

def evaluate(points,p1,p2,threshold):
    inliers = 0
    inlier_points = []
    for p in points:
        d = find_line_distance(p1,p2,p)
        if d < threshold:
            inliers += 1
            inlier_points.append(p)

    return inliers, np.array(inlier_points)
```

```python
def ransac(image):
    '''
    Gets input image and return rho and theta of line detected.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        rho (float): The distance from the origin to the line.
        theta (float):  Angle from origin to the line.
    '''

    img = image.copy()
    h, w = img.shape
    rho, theta = 0, 0
    points = find_points(h,w,img)
    best_model = None
    best_inliers = np.array([])
    threshold = 1
    best_evaluation = float('-inf')
    min_inliers = float('inf')

    iter = 24
    for i in range(iter):
        idx0, idx1 = np.random.randint(0,len(points),2)
        while idx0 == idx1:
            idx0, idx1 = np.random.randint(0,len(points),2)
        p1, p2 = points[idx0], points[idx1]

        evaluation, inlier_coords = evaluate(points,p1,p2,threshold)
        if evaluation > best_evaluation:
            best_evaluation = evaluation
            best_inliers = inlier_coords
            best_model = p1, p2

    y_bar, x_bar = np.mean(best_inliers,axis=0)
    y2_bar, x2_bar = np.mean(np.square(best_inliers),axis=0)
    x_bar2, y_bar2 = np.square(x_bar), np.square(y_bar)
    xy_bar = np.sum((best_inliers.T[0] * best_inliers.T[1]),axis=0)/len(best_inliers)
    theta = 0.5 * np.arctan((2*(xy_bar - x_bar*y_bar))/(x2_bar - y2_bar - x_bar2 + y_bar2))
    rho = x_bar * np.cos(theta) + y_bar*np.sin(theta)
    return rho , theta
```

4.3 Hough Algorithm - Manual

I got $0 \leq \rho \leq 180$ and $0 \leq \theta \leq 140$ and step value of 1. Then found the edges using `Canny` algorithm. Iterated through all the edge points and vote for the correspong ρ and θ in the accumulator. Finally I computed the indices of the maximum value of accumulator and returned the corresponding ρ and θ.
Source




```python
def hough_transform_line(image):
    '''
    Returns rho and theat of line detected and hough transform image.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        rho (float): Angle from origin to the line.
        theta (float): The distance from the origin to the line.
        hough_transform (numpy.ndarray): Hough transform image.
    '''
    theta_step = 1

    img = image.copy()
    h,w = img.shape
    accumulator = np.zeros_like(img)
    rho, theta = 0, 0
    rho_size = int(np.sqrt(np.square(h) + np.square(w)))
    thetas = np.deg2rad(np.arange(0,180,step=1))
    rhos = np.linspace(-int(rho_size), int(rho_size), int(rho_size * 2.0))

    accumulator = np.zeros((2*rho_size, len(thetas)))
    print(accumulator.shape)
    edges = cv2.Canny(img, 30, 110)
    edge_positions = np.where(edges != 0)

    cos_theta = np.cos(thetas)
    sin_theta = np.sin(thetas)

```

```python
    for i in range(len(edge_positions[0])):
        y, x = edge_positions[0][i],edge_positions[1][i]
        rho_idx = x * cos_theta + y * sin_theta
        for j in range(len(thetas)):
            accumulator[int(rho_idx[j] + rho_size)][j] += 1

    rho = rhos[np.argmax(accumulator)//180]
    theta = thetas[np.argmax(accumulator)% 180]

    return rho, theta, accumulator
```