# Computer Vision assignment 2

2019-12172 이재원

May 3, 2024

## 1 Math

### 1.1 (a) Derive the polar line representation: $\rho = x\cos\theta + y\sin\theta$

Let x and y be arbitrary points in the line. The normal vector is $(\cos\theta, \sin\theta)$. The inner product of the normal vector and (x, y) is always same as $\rho$. Note that $\rho$ is the distance from the line to the origin.

$$\rho = (x, y) \cdot (\cos\theta, \sin\theta)$$

### 1.2 (a) Derive the polar plane representation in 3D

As same as 1.1, planes can be represented as normal product with $\rho$ too. If we use the spherical coordinate in Figure 1, the normal vector is $(\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$. Let an arbitrary point on the plane be (x, y, z), then

$$\rho = (x, y, z) \cdot (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$$

where $\rho$ is the distance from the line to the origin.

$$\rho = x\sin\theta\cos\phi + y\sin\theta\sin\phi + z\cos\theta$$

## 2 Generalized Hough Transform

### 2.1 Edge Detection

```
def image2edge(image, threshold1=50, threshold2=150):
  #image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  LPF_image = cv2.GaussianBlur(image, (5, 5), 0)
  edge_image = cv2.Canny(LPF_image, threshold1, threshold2)

  return edge_image


template_edge_image = image2edge(template_image)

target_edge_image = image2edge(target_image)
```

The edge detection code uses canny from cv2. Additionally, gaussian filter is used before canny because it acts as a low pass filter. If gaussian filter is not used, the GHT algorithms detect noises too.

### 2.2 Calculating Gradients

```

# Gradient orientation calculation using Sobel operators
def calculate_gradients(image):
    sobelx = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)
    sobely = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)
    plt.imshow(sobelx)
    gradient = np.arctan2(sobely, sobelx)
    return gradient
```

In this code, gradient is calculated by using Sobel filters with kernel size 3.

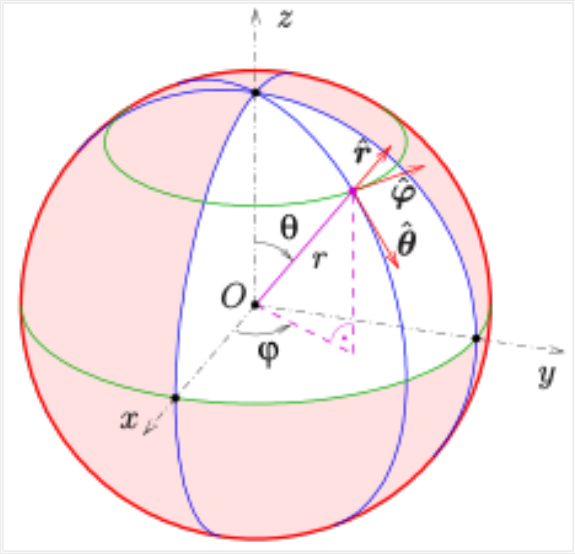



Figure 1: Spherical coordinate. Image source: Wikipedia

## 2.3 Building R table

```
def calculate_r_table(image_edges, gradients, reference_point):
    # R-table dictionary to hold gradients and corresponding (r, alpha)
    r_table = {}
    for (i, j), value in np.ndenumerate(image_edges):
        if value:  # if edge is present
            phi = gradients[i, j]
            r = np.sqrt((reference_point[0] - i) ** 2
                        + (reference_point[1] - j) ** 2)
            alpha = np.arctan2(reference_point[1] - j, reference_point[0] - i)
            if phi not in r_table:
                r_table[phi] = []
            r_table[phi].append((r, alpha))
    return r_table
```

This code is based on the lecture slides in Figure reffig:GHT.

## 2.4 Generalized Hough Transform with scale and rotation invariant

```
def generalized_hough_transform(target_edges,
                                gradients,
                                r_table,
                                scale_r = 1,
                                scale_alpha = 0):

    vote_num = np.zeros(target_edges.shape, dtype=np.uint32)
    for (i, j), value in np.ndenumerate(target_edges):
        if value:  # if edge is present
            phi = gradients[i, j]
            if phi in r_table:
                for r, alpha in r_table[phi]:
                    r = r * scale_r
                    alpha += scale_alpha
                    x = int(i + r * np.cos(alpha))
                    y = int(j + r * np.sin(alpha))
                    if ((0 <= x < vote_num.shape[0])
                        and (0 <= y < vote_num.shape[1])):
                        vote_num[x, y] += 1
    return vote_num
```

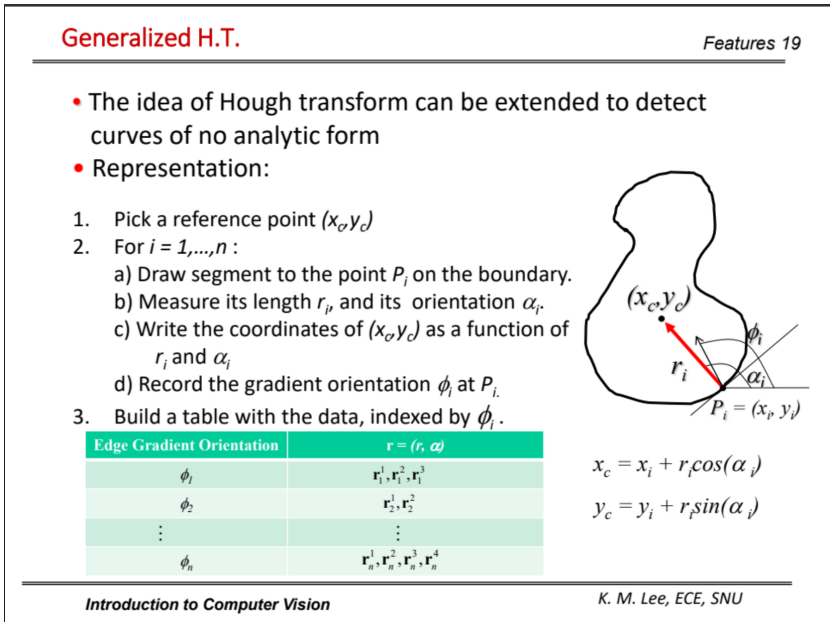



Figure 2: Generalized Hough Transform Slide

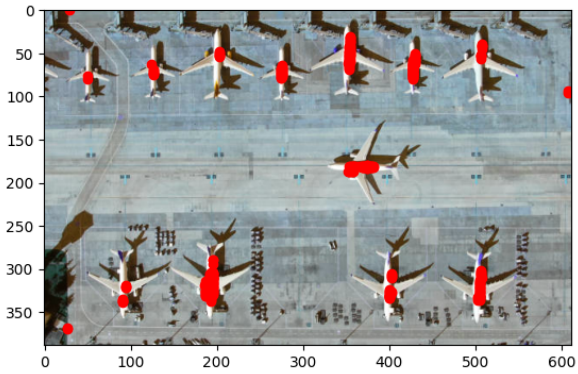
The scale and rotation invariant GHT is implemented by adding scale_r argument and scale_alpha argument. scale_r is multiplied with the r from r_table, and scale_alpha is added from the alpha. Now this function is executed in 2 for-loops changing values of scale_r and scale_alpha.

```
total_votes = []
for scale_r in scale_r_list:
  for scale_alpha in scale_alpha_list:
    vote_num = generalized_hough_transform(target_edge_image,
                                           target_gradients,
                                           r_table,
                                           scale_r=scale_r,
                                           scale_alpha=scale_alpha)
    total_votes.append(vote_num)
```

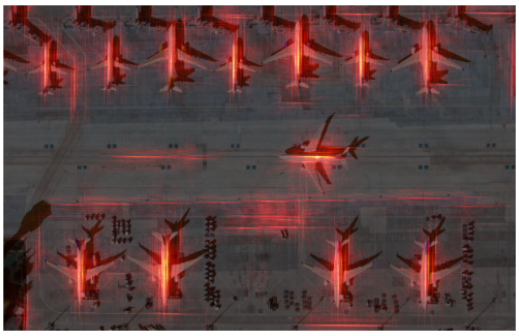
# 3 Results and Discussion

Figure 3a shows all the points where votes are greater or equal to the threshold 46. All the airplanes are detected in the picture. However, there exist points where there are no airplanes near, but was detected as an airplane. For example, the shadow of the airplane was also detected. This can be adjusted by changing the threshold of vote.

Figure 3b is the entire accumulator including the votes smaller than the threshold. You can see that all the airplane edges are well detected. The points inside the airplane is the denser than points at outside of airplane.




(a) Plotting points where votes are bigger or equal than the threshold (46)



(b) Plotting the entire accumulator and target image. The accumulator where the vote is smaller than 5 is erased.

Figure 3: Detected edge results