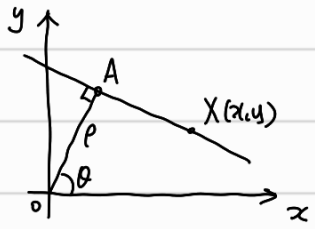
ICV S24 Assignment #2

작성자: 장원재 (2020-10240)

1. Math

(a) Derive the 2D polar line representation : $\rho = x\cos\theta + y\sin\theta$



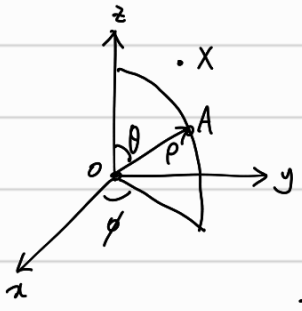
$\overrightarrow{OA} = (\rho\cos\theta, \rho\sin\theta)$, $\overrightarrow{OX} = (x, y)$.

$0 = \overrightarrow{OA} \cdot \overrightarrow{AX} = (\rho\cos\theta, \rho\sin\theta) \cdot (x-\rho\cos\theta, y-\rho\sin\theta)$

$= x\cdot\rho\cos\theta - \rho^2\cos^2\theta + y\cdot\rho\sin\theta - \rho^2\sin^2\theta$

$\rightarrow$ $x\cos\theta + y\sin\theta = \rho$.

(b). Can you derive the polar representation of planes in 3D?



$0 = \overrightarrow{OA} \cdot \overrightarrow{AX} = \overrightarrow{OA} \cdot (\overrightarrow{OX} - \overrightarrow{OA})$

$\overrightarrow{OA} \cdot \overrightarrow{OX} = x\cdot\rho\sin\theta\cos\phi + y\cdot\rho\sin\theta\sin\phi + z\cdot\rho\cos\theta$

$= \|\overrightarrow{OA}\|^2 = \rho^2$

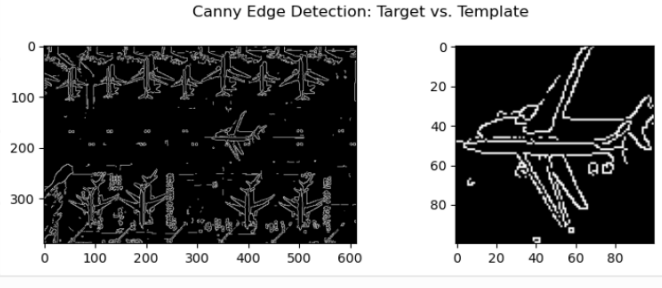
$\rightarrow$ $x\cdot\sin\theta\cos\phi + y\cdot\sin\theta\sin\phi + z\cdot\cos\theta = \rho$

2. Generalized Hough Transform (GHT)

In this problem, I implemented the General Hough Transform (GHT) algorithm shown by [1]. Also, I tested my GHT algorithm on the given target & template images.

- Data Glimpse.

To get edge images of target & template images, I used OpenCV Canny edge detecting algorithm. Edge images of the given images with threshold (150, 200):



- Implementation.

① ② ③

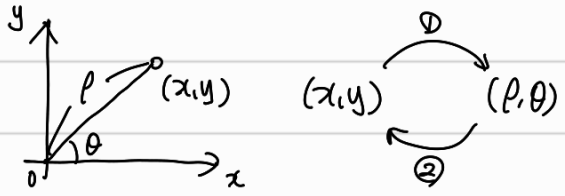
```python
def xy2polar(i, j):
    """
    (y, x) -> (rho, theta)
    theta in radian
    """
    rho = np.sqrt(i*i + j*j)
    theta = np.arctan2(-i, -j) + np.pi
    return rho, theta
```

```python
def polar2xy(rho, theta):
    """
    (rho, theta) -> (y, x)
    theta in radian
    """
    i = (rho * np.sin(theta)).astype(int)
    j = (rho * np.cos(theta)).astype(int)
    return i, j
```

```python
def get_center_angle(edge, bin_num):
    """
    Get the angle of the edge in the center based on Hough Transform.

    - Input
    edge: edge data whose center is filled.
    bin_num: the number of angle(0~pi) cuts.

    - Output
    angle: bin number that edge is contained.
    """
    N, M = edge.shape
    assert edge[N//2, M//2] != 0, "Center is empty!"

    theta = np.linspace(0, np.pi, bin_num+1)[:bin_num]
    cos, sin = np.cos(theta), np.sin(theta)

    y, x = np.where(edge != 0)
    all_curve = x.reshape(-1, 1) * cos + y.reshape(-1, 1) * sin
    center_curve = (M//2) * cos + (N//2) * sin

    focus_score = np.exp(-0.5*(all_curve - center_curve)**2)
    focus_score = np.sum(focus_score, axis=0)
    angle = np.argmax(focus_score)
    return angle
```

① Convert Cartesian coordinates to polar coordinates.

② Convert polar coordinates to Cartesian coordinates.



③ : Get angle of the edge in center based Hough transform.

Pseudo-Code :

for $(x_j, y_j) \in \{(x,y) : edge[x,y] \neq 0\}$ do

  | Draw a line $\rho_j(\theta) = x_j \cos\theta + y_j \sin\theta$ where $0 \le \theta < \pi$.

$(x_c, y_c) \leftarrow$ center coordinate of the edge image

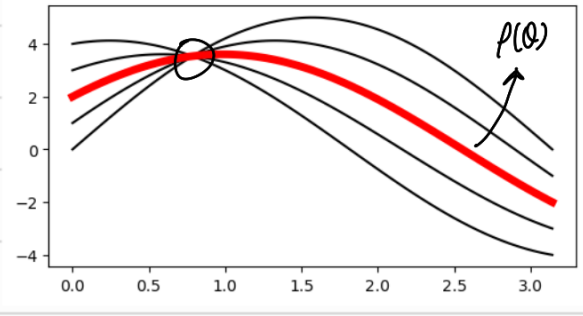
Draw a line $\rho(\theta) = x_c \cos\theta + y_c \sin\theta$ where $0 \le \theta < \pi$.

$\theta_i \leftarrow (i / bin\_num) \cdot \pi \quad (i = 0, 1, \cdots, bin\_num - 1)$

$angle \leftarrow \underset{i}{argmax} \sum_j \exp\left[-\frac{1}{2}\{\rho_j(\theta_i) - \rho(\theta_i)\}^2\right]$

return angle

ex)



Get angle where the other curves are most concentrated on $\rho(\theta)$.

∵ It should focus on the center.

```python
class HitMap:
    """
    Handles the number and the location of hits.
    It is used by HoughTransform class.
    """
    def __init__(self, edge, Rtable, **kwargs):
        self.edge = edge
        self.N, self.M = edge.shape
        self.K = max(Rtable.shape[1], Rtable.shape[2])
        self.hitmap = np.zeros((self.N + 2*self.K, self.M + 2*self.K))
        self.Rtable = Rtable

        self.sampling_size =  kwargs['sampling_size']
        self.bin_num = kwargs['bin_num']
        self.threshold_rate = kwargs['threshold_rate']
        self.hitmap_filter_size = kwargs['hitmap_filter_size']

    def hit(self, i, j):
        """
        Get coordinate of the edge and update hitmap using Rtable.
        """
        h = self.sampling_size // 2
        angle = get_center_angle(self.edge[i-h:i+h, j-h:j+h], self.bin_num)

        K = self.K
        NN, MM = self.Rtable.shape[1]//2, self.Rtable.shape[2]//2
        self.hitmap[i-NN+K:i+NN+K, j-NN+K:j+NN+K] += self.Rtable[angle]

    def done(self):
        """
        No more update. Cut off the useless outer part.
        """
        K = self.K
        self.hitmap = self.hitmap[K:K+self.N, K:K+self.M]
```

edge (i,j) votes using Rtable.

⇒ hitmap += Rtable[θ]

```python
    def show(self, default_img=None):
        """
        Print hitmap on the target image.
        """
        hmap = self.hitmap
        if self.hitmap_filter_size is not None:
            hmap = ndimage.maximum_filter(hmap, size=self.hitmap_filter_size)

        hmax, hmin = np.max(hmap), np.max(hmap) * self.threshold_rate
        # threshold & normalize
        hmap = np.where(hmap > hmin, hmap, hmin)
        hmap = hmap / hmax

        ax = plt.subplot()
        if default_img is None:
            ax.imshow(self.edge, cmap='gray')
        else:
            ax.imshow(default_img, cmap='gray')
        sns.heatmap(hmap, alpha=0.75, xticklabels=False, yticklabels=False, ax=ax)

        plt.title("Hitmap on Original Image")
        plt.tight_layout()
        plt.show()
```

Draws hitmap on the target image.

```python
class HoughTransform:
    """
    Do the General Hough Transform.
    """
    def __init__(self, **kwargs):
        self.kwargs = kwargs
        self.canny_min = kwargs['canny_min']
        self.canny_max = kwargs['canny_max']
        self.sampling_size =  kwargs['sampling_size']
        self.bin_num = kwargs['bin_num']
        self.scale_pool = kwargs['scale_pool']
        assert self.sampling_size % 2 == 1, "sampling_size should be odd!"

    def go(self, target, template):
        """
        Run the whole procedure.
        1. Get Rtable using template_edge.
        2. Vote with HitMap class.
        3. Show the hitmap with target image.
        """
        # Edge & Rtable
        target_edge = cv2.Canny(target, self.canny_min, self.canny_max)
        template_edge = cv2.Canny(template, self.canny_min, self.canny_max)
        Rtable = self.get_Rtable(template_edge)

        # Voting
        N, M = target_edge.shape
        hitmap = HitMap(target_edge, Rtable, **self.kwargs)
        h = self.sampling_size // 2
        for i in range(h, N-h):
            for j in range(h, M-h):
                if target_edge[i, j] == 0: continue
                hitmap.hit(i, j)
        hitmap.done()

        # Get the winner
        hitmap.show(target)
```

Main runner.

: Gee edge images using Canny algorithm & Rtable.

: Every non-empty point in target_edge votes using HitMap class.

: Print hitmap on target image.

```python
    def get_Rtable(self, edge):
        """
        Get Rtable in a form of convolution filter.
        Gaussian filter is used for the smoother Rtable.
        """
        N, M = edge.shape
        ri, rj = N//2, M//2
        h = self.sampling_size // 2

        Rtable = np.zeros((self.bin_num, 3*N, 3*M))
        for i in range(h, N-h):
            for j in range(h, M-h):
                if edge[i, j] == 0: continue
                angle = get_center_angle(edge[i-h:i+h, j-h:j+h], self.bin_num)
                di, dj = i - ri, j - rj
                rho, theta = xy2polar(di, dj)

                rot_angle = (angle + np.arange(self.bin_num)) % self.bin_num
                rot_theta = theta + np.arange(self.bin_num) / self.bin_num * np.pi
                for scale in self.scale_pool:
                    di, dj = polar2xy(scale * rho, rot_theta)
                    ni, nj = ri + di, rj + dj
                    Rtable[rot_angle, N + ni, M + nj] += 1
                    ni, nj = ri - di, rj - dj
                    Rtable[rot_angle, N + ni, M + nj] += 1

        Rtable = self.gaussian_filter(Rtable)
        Rtable = Rtable[:, N:2*N, M:2*M]
        return Rtable

    def gaussian_filter(self, field, sigma=1, fl=3):
        h = fl // 2
        xx = np.arange(-h, h+1)
        x, y = np.meshgrid(xx, xx)
        gaussian = np.exp(-(x*x+y*y)/(2*sigma**2))
        gaussian /= np.sum(gaussian)

        ret = np.zeros_like(field)
        _, N, M = field.shape
        for i in range(N - fl):
            for j in range(M - fl):
                ret[:, i:i+fl, j:j+fl] += field[:, i:i+fl, j:j+fl] * gaussian
        return ret
```

function to generate R table as a conv filter.

Rtable [θ, i, j] : # of votes of edges with angle θ to (i,j)

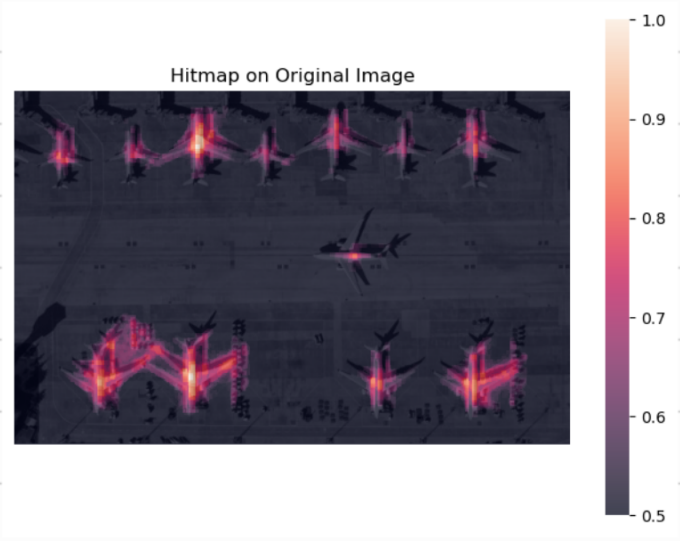
: Get center angle & vector to template center.

: Scale & rotate the vector.

Gaussian filter for smoother Rtable.

- Test 1 : Original Image

We can see that the algorithm successfully detects the objects which is similar to template image. Airplanes with different direction and scale are detected fine, which shows the algorithm is scale & rotation invariant.



Test setting :

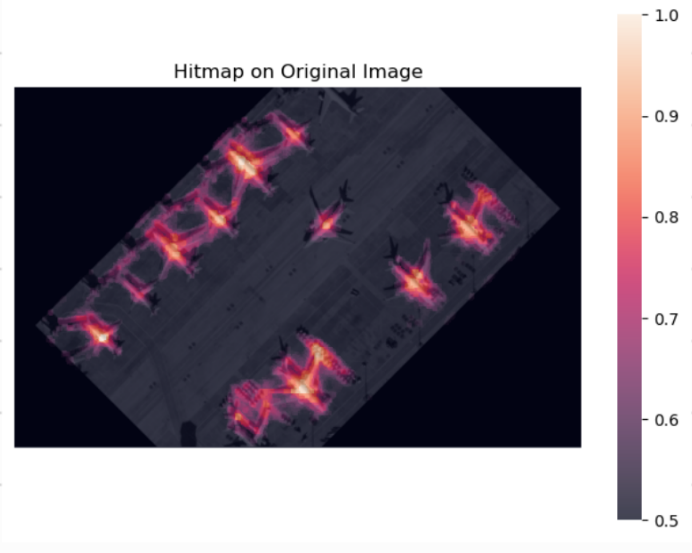
```
'canny_min': 200,
'canny_max': 400,
'sampling_size': 15,
'bin_num': 180,
'threshold_rate': 0.5,
'scale_pool': np.linspace(0.5, 2, 10),

'hitmap_filter_size': 5,
```

- Test 2 : Scaled & Rotated Image.

Because it is scale & rotation invariant, it works fine with an Affine transformed target.

The original target image is rotated by 45° and scaled by 0.8.

We can see that the algorithm detects all proper objects.



(Test setting is same as Tese 1.)

And I could reduce the program runtime by using Rtable as Convolution filter. Single program runtime is around 5 sec.