CS 6643 Project–Segmentation/Finding of objects

Jingyao Li jl9075@nyu.edu
Kuan Chen kc3422@nyu.edu
Guanhua Chen gc2229@nyu.edu
Wei Wang ww1306@nyu.edu

1 Introduction and Motivation

In computer vision, object detection is the process of finding instances of real-word objects such as keys,cars and faces. Object detection algorithms typically use extracted features and learning algorithms to recognize instances of an object category. When it comes to Object detection, the first algorithm we think is Hough Transform.

The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. However, the Hough Transform was initially to detect defined shapes like line or circle[1]. When we want to identify more complex objects, we need use the General Hough Transform.

We can define the template's shape by r-table and find the object in a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the General Hough transform.

Before this project, there also has been some work like writing our own Image Smoothing and Canny edge detection function. Then we implemented the General Hough Transform technique and use it to find the object. Next section explains implementation details corresponding to the methods used, followed by our results.

2 Methods and Approach

2.1 Canny Detection

The first step of segmentation of object is the edge detection of images and templates. All operations of general hough transform are worked on edge image instead of the original image. Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. [2]

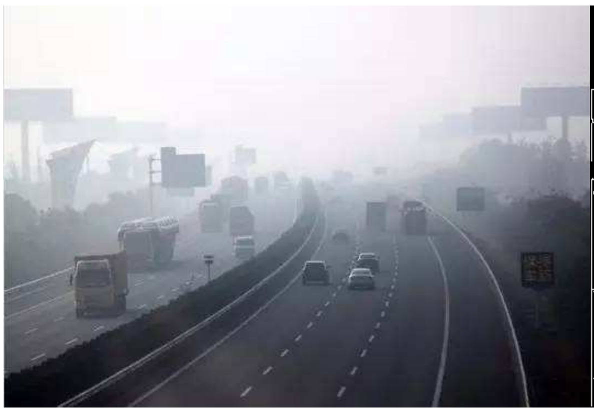

Figure 1: The original image

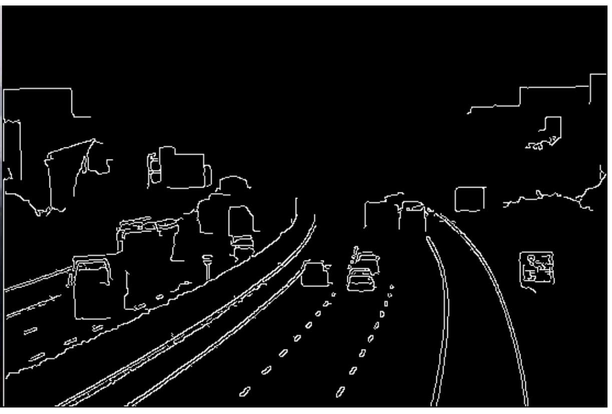

Figure 2: The Canny edge detector

In this project, there are mainly three steps of canny detection to get the edge images.

2.1.1 Magnitude and orientation

Edge detection can be implemented as a horizontal and vertical differentiation filtering. There are several differentiation filters. For example, [-1, 0, 1] is the easiest horizontal differentiation mask. However, this mask is not very accurate to get the derivative of the image. There are some other approximations of derivative filters shown below. Here we use sobel:

$$Sobel = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

To make it more accurate, we choose Sobel Operator as the derivative filters. Application of these two masks for horizontal and vertical filtering results in estimates of image derivatives.

$$dx = \frac{\partial}{x} I(x, y)$$
$$dy = \frac{\partial}{y} I(x, y)$$

The two partial derivatives form a vector (dx, dy). The vector norm $\sqrt{dx^2 + dy^2}$ is the magnitude of the edge image. The more the magnitude is, the more possible the corresponding pixel is to be a edge pixel. The edge orientation of the pixel is calculated as $\alpha = tan^{-1}(\frac{dy}{dx})$.

2.1.2 Non-Maximum Suppression

Non-maximum suppression is the operation to extract a thin line on the ridges of the unsharp edge magnitude images. Given the magnitude and orientation of the edge image, we remove all non-maximum pixels. First, we map the edge orientation, which is finished before, to four groups of direction, east, north-east, north and north-west. Then, for each pixel, we compare its magnitude with its two neighbors of its direction among its 3X3 neighborhood. Finally, we set its magnitude as zero if its magnitude is smaller than both of its two neighbors'. After these operations, we get a sharper edge magnitude images.

2.1.3 Hysteresis Thresholding

We choose hysteresis thresholding to make the edge image clear. Hysteresis thresholding uses two thresholds: low threshold and high threshold t_l, t_h, hence:

- if $\nabla I(x, y) > t_h$, definitely an edge, set magnitude = 1
- if $t_l < \nabla I(x, y) < t_h$, maybe an edge, set magnitude = 0.5
- if $\nabla I(x, y) < t_l$, definitely not an edge, set magnitude = 0

2.2 General Hough Transform

2.2.1 Get Edge Gradient

Before we create the R-table, we should get the gradient of each points as the index of the R-table for each point.

Method to calculate the gradient of the points has shown above using:

$$G = \sqrt{G_x^2 + G_y^2}$$
$$\theta = tan^{-1}(G_y/G_x)$$
$$where \;\; G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A$$
$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

2.2.2 Generate R-Table

Choose a reference point of the template object. (Usually the reference point is the middle point inside the object).For each boundary point x, compute $\varphi(x)$, the gradient direction and r = y − x as shown in the image. Store r as a function of φ. Notice that each index of φ may have many values of r[3]. As we have got the gradient map, we could directly use the gradient value of each point as $\varphi(x)$.

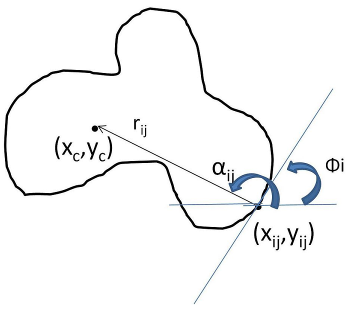

Figure 3: Geometry of shape detection

R-Table:

Table 1: R-table

φ	vector
φ_1	(r11, α11) (r12, α12)... (r1n, α1n)
φ_2	(r21, α11) (r22, α22)... (r2m, α2m)
φ_3	(r31, α31) (r32, α32)... (r3k, α3k)
...	...
φ_{360}	(r360,1, α360,1) (r360,2, α360,2)... (r360,j, α360,j)

Here is the algorithm:

Algorithm 1 Build R-Table

```
1: Gradient_Map = Get_Gradient_Map(Edge_Map), Get the gradient map
2: [height, width] = size(Edge_Map), Get the size of the edge map
3: Reference_point = (round(height / 2), round(width / 2)), Choose a reference point
4: [y, x] = find(Edge Map > 0), find the coordinate for the edge points.
5: R_Table = zeros(360, number of edge points, 2), define the R-Table with initial value: 0 and it has
   two dimensions for direction x and y.
6: Vector_counter = zeros(360, 1), define the counter to count the number of vectors of each index.
7: for each point(x, y) in Edge Map do
8:     $\theta$ = Gradient_Map(x, y)
9:     Vector_counter($\theta$) = Vector_counter($\theta$) + 1
10:    Vector = (x, y) - Reference_point
11:    R_Table($\theta$, Vector_counter($\theta$), 1) = Vector , R_Table in direction y
12:    R_Table($\theta$, Vector_counter($\theta$), 2) = Vector , R_Table in direction x
13: end for
14: return R_Table
```

2.2.3 Apply Accumulator

We apply voting to detect the object in the image. First, convert the sample image into an edge image using canny edge detection algorithm. Then, initialize the Accumulator table: A[Xmin,...,Xmax][Ymin,...,Ymax]. And for each point(x, y) in the sample image, calculate the reference point (Rx, Ry) and increase counters (voting): A([Rx][Ry])++. Possible locations of the object's reference point is given by local maximum of the Accumulator table[3].

Here is the algorithm:

Algorithm 2 Apply Accumulator to vote for reference point

```
1: [height, width] = size(image), Get the height and width of the image to be detected
2: Voting_Result = zeros(height, width)
3: Gradient_Map = Get_Gradient_Map(Image), Get the gradient map
4: for each point(x, y) in Image do
5:     $\theta$ = Gradient_Map(x, y)
6:     for i = 0 in range(Vector_counter($\theta$)) do
7:         New_vector(:, y) = (x, y) - R_Table($\theta$, i ,1), in direction y
8:         New_vector(x,:) = (x, y) - R_Table($\theta$, i ,2), in direction x
9:         if New_vector in correct range then
10:            Voting_Result(New_vector) = Voting_Result(New_vector) + 1
11:        end if
12:    end for
13: end for
14: return Voting_Result
```

The point with the most votes is the reference point of the object.

2.3 Improvement of GHT

Now we have the basic general hough transformation. But it is not enough to find the object in real life, for the object you try to find will not be totally same as the template. Hence we need to improve the general hough transformation with rotation and scaling.

2.3.1 Rotation

For the R-table, the vector from points in the template to the reference point may be rotated by an angle. So the vector n should be multiplied with the affine matrix as follow:

$$x' = xcos\theta - ysin\theta$$
$$y' = xsin\theta + ycos\theta$$

The algorithm will be:

Algorithm 3 R-Table with Rotation

```
1: for angle = 1:360 do
2:     ......same as previous part
3:     $R(\varphi, angle) = (xcos - ysin, xsin + ycos)$
4: end for
```

After rotation of R-table, we need to adjust the rotation of the gradient of each points in the image. After modification with the angle, it can be processed with the according R-Table, here is the algorithm:

Algorithm 4 Apply Accumulator with Rotation

```
1: for angle = 1:360 do
2:     ......same as previous part
3:     for each point do
4:         ......same as previous part
5:         $\varphi' = \varphi - angle$
6:         $Reference = (x, y) + R(\varphi', angle)$
7:     end for
8:     Store the best voting value
9: end for
```

After calculating all 360 degrees, the best voting value of all 360 degrees will be the right rotation angle with high possibility.

2.3.2 Scaling

The process of scaling is similar with the process of rotation. It does not need to change the gradient. So we just modify the R table with different scaling coefficient, the algorithm:

Algorithm 5 Apply Accumulator with Rotation

```
1: for scaling = 0.5:0.1:2 do
2:     ......same as previous part
3:     $R(\phi, scaling) = (x * scaling, y * scaling)$
4: end for
```

3 Data and Results

For test, we use the real-life objects photoed by cellphone, and try to find the object using our algorithm. We think it is a good to test whether our algorithm is useful and can be used in real life.

3.1 Canny Detection

Firstly, we use the Canny Edge detection to process the image. As discussed earlier, a good edge detection could help reduce the cost of the subsequent calculation and get the better result.

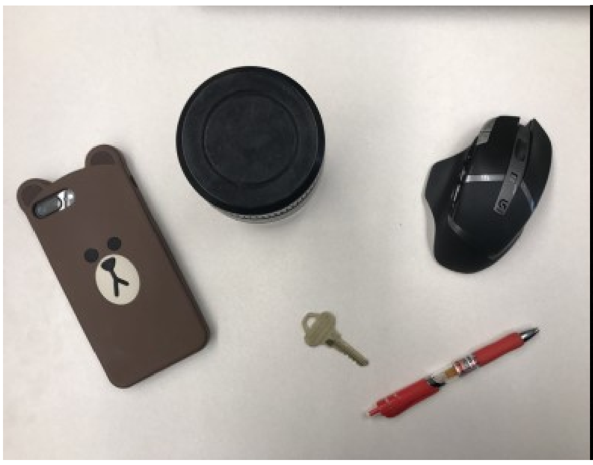

Figure 4: The original image

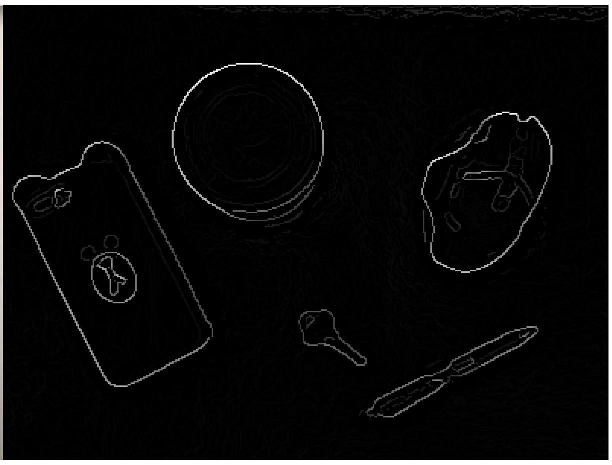

Figure 5: Our Canny edge detector

As it shows, we get a sharp and clear edge image of the original image after canny edge detector.

3.2 General Hough Transformation

Now we get a template, which is a key screenshotted from the original image, and see if we could find the item from the image, without rotation and scaling by general hough transformation.

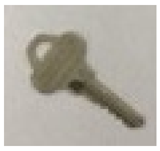

Figure 6: The original image

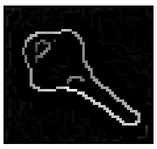

Figure 7: Our Canny edge detector

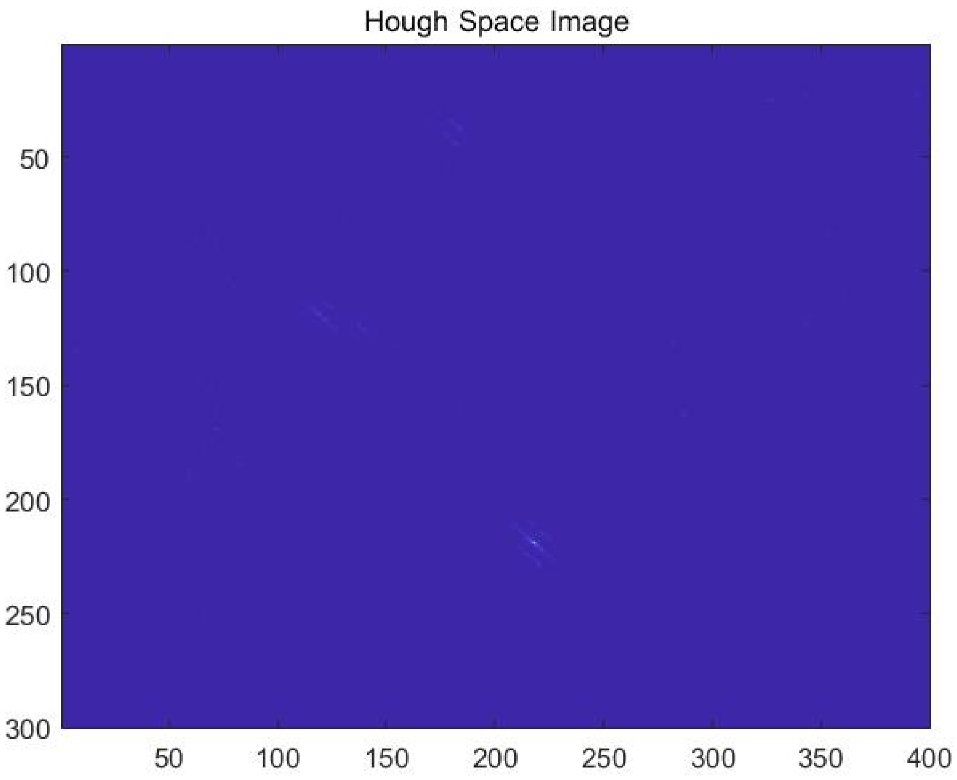

Figure 8: Hough Space Image

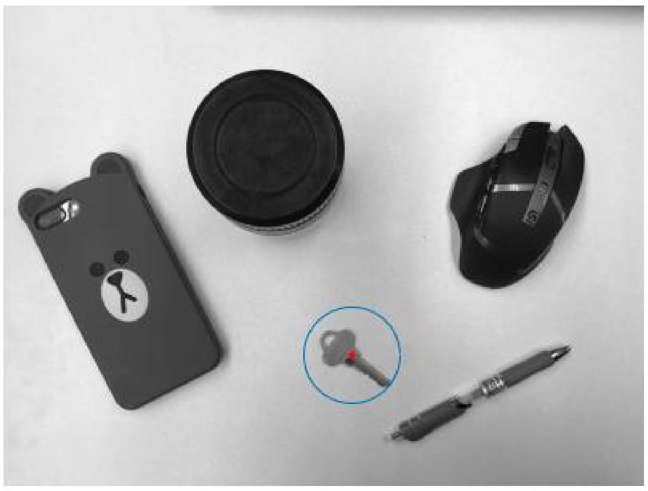

Figure 9: segmentation result

We find the key correctly! From the hough space, we find that there is rarely other bright points except the correct position if our template item is not rotated or scaling. Now let's see the result with rotation and scaling.

3.3 General Hough Transformation with Rotation and Scaling

Now we get another template, which was photographed in another angle.

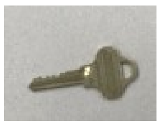
Figure 10: The original image

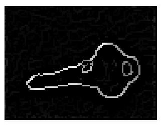
Figure 11: Our Canny edge detector

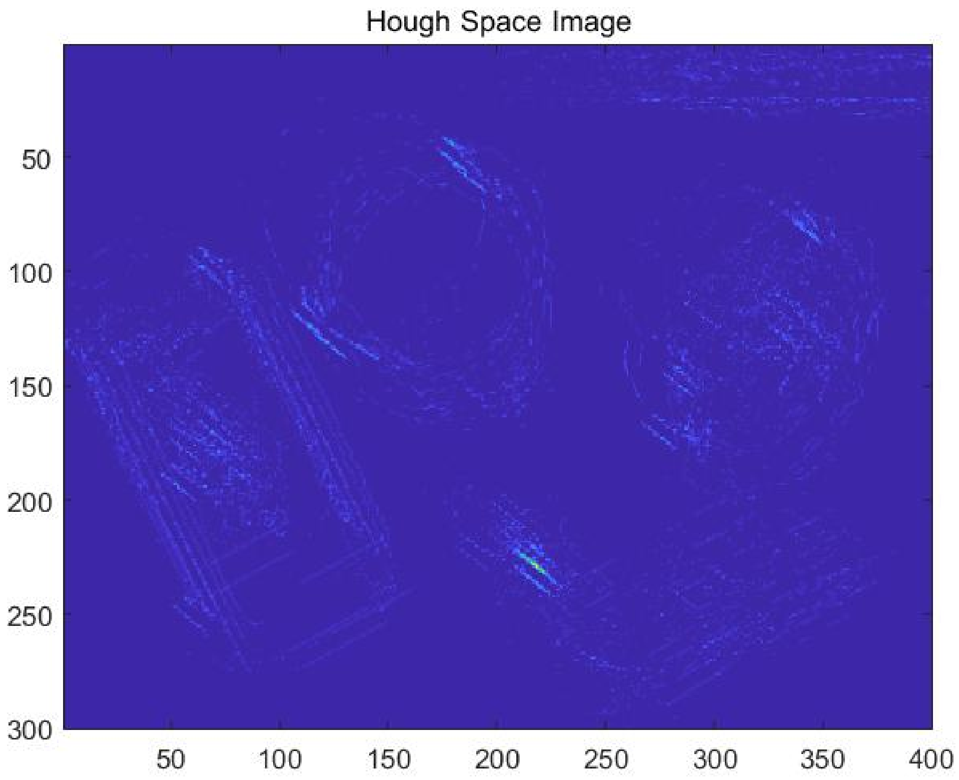

Figure 12: Hough Space Image

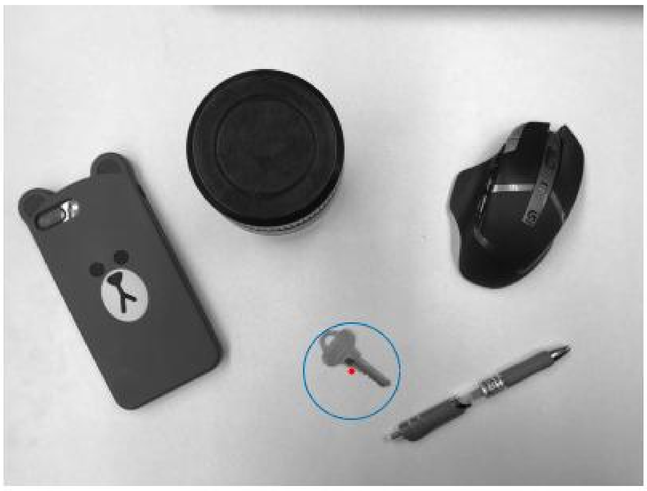

Figure 13: segmentation result

The key is found correctly. Let's see the hough space. Compared the hough space without rotation and scaling, this hough space has more bright points, which means more candidates. But since we consider angles from 0 to 360 degree and scaling from 0.5 to 2, we could still find the rotated and scaling key.

Now let's change another item(a pen), rotating and scaling it like this:

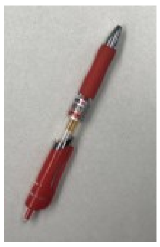

Figure 14: The original image

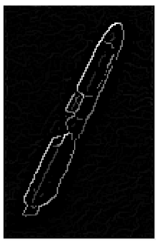

Figure 15: Our Canny edge detector

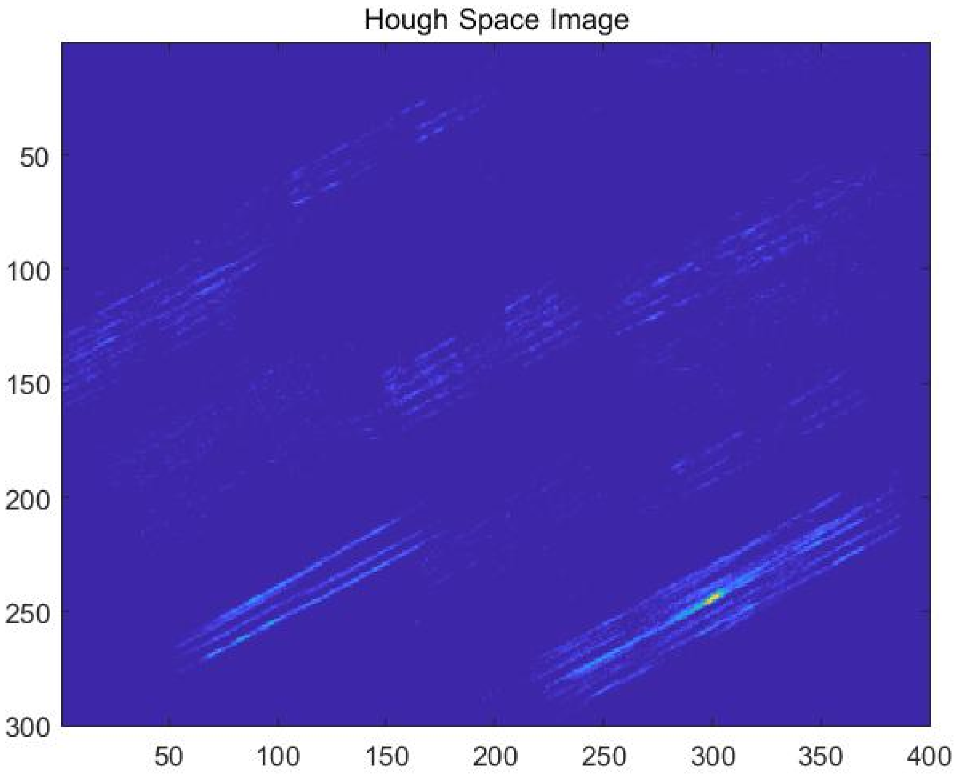

Figure 16: Hough Space Image

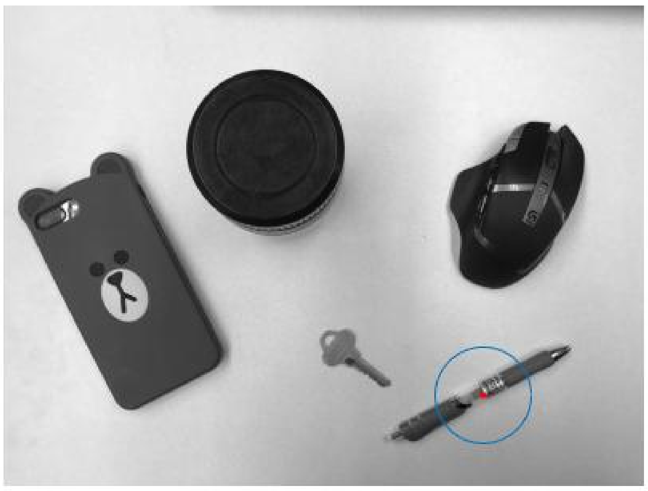

Figure 17: segmentation result

Here is the result, we get it. So the same shape in the image with any angle and scaling, it is easy for our algorithm to get it.

3.3.1 Find multiple objects in same image

If we have a database of templates, we can find all object of database when scan an image. Here we find the pen and key at same time.

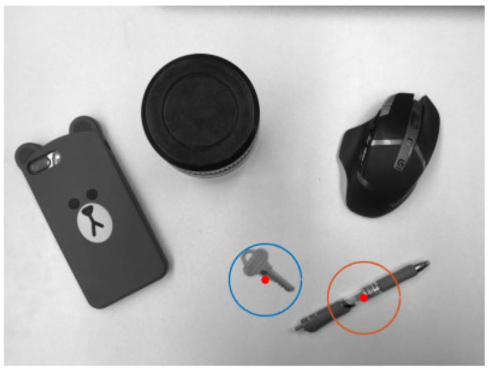

Figure 18: Find two objects at same time

4 Critical Discussion & Conclusion

For this project, we have implemented the General Hough Transform and could find objects accurately including scaling and rotation. Here are some discussion we have according to the GHT about its advantage and disadvantage:

Pros:

- Can detect multiple instances of a model
- Some robustness to noise: noise points unlikely to contribute consistently to any single bin

Cons:

- Complexity of search time increases exponentially with the number of model parameters
- Non-target shapes can produce spurious peaks in parameter space
- It's hard to pick a good grid size

In this project, we implement the complete Canny Edge Detection and General Hough Transformation with rotation and scaling. For the big image with too many pixel the algorithm will take a lot of time or out of memory. We think that is key direction for us to improve the algorithm in the future. The project helps us to enhance our understanding of the General Hough Transformation. It is really a fun for us to get a good result from our implementation of the algorithm. Thanks for the support from Professor and TAs in the whole semester. Happy Summer!

References

[1] Hough transform. In Wikipedia. Retrieved May 5, 2018, from https://en.wikipedia.org/wiki/Hough_transform
[2] Canny edge detector. In Wikipedia. Retrieved May 5, 2018, from https://en.wikipedia.org/wiki/Canny_edge_detector
[3] Generalised Hough transform. In Wikipedia. Retrieved May 5, 2018, from https://en.wikipedia.org/wiki/Generalised_Hough_transform