Assignment 1

Alex Darwiche

Computer Vision 8820

2/11/2025

Please see all the information below. I have a second for each image and the comments associated with that image are below the image where necessary. For the thresholds used in C, those filter sizes appear in the title of the images.

Image B

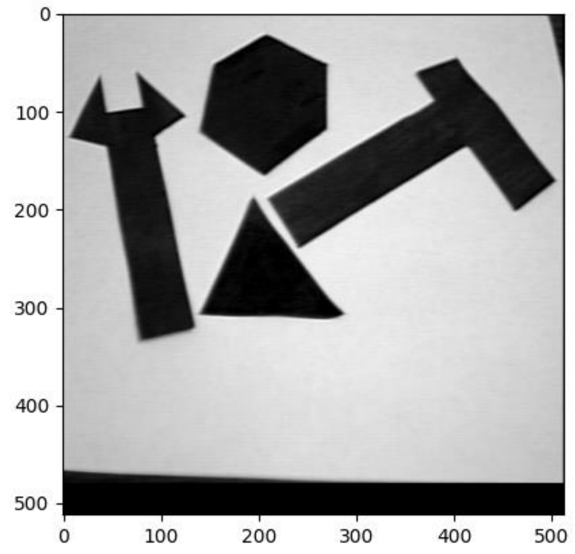
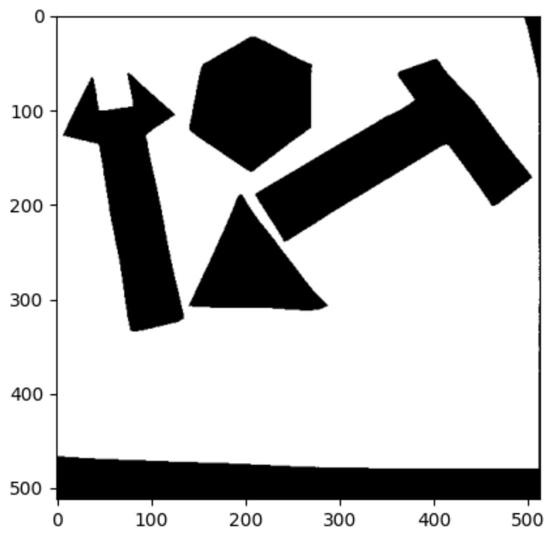
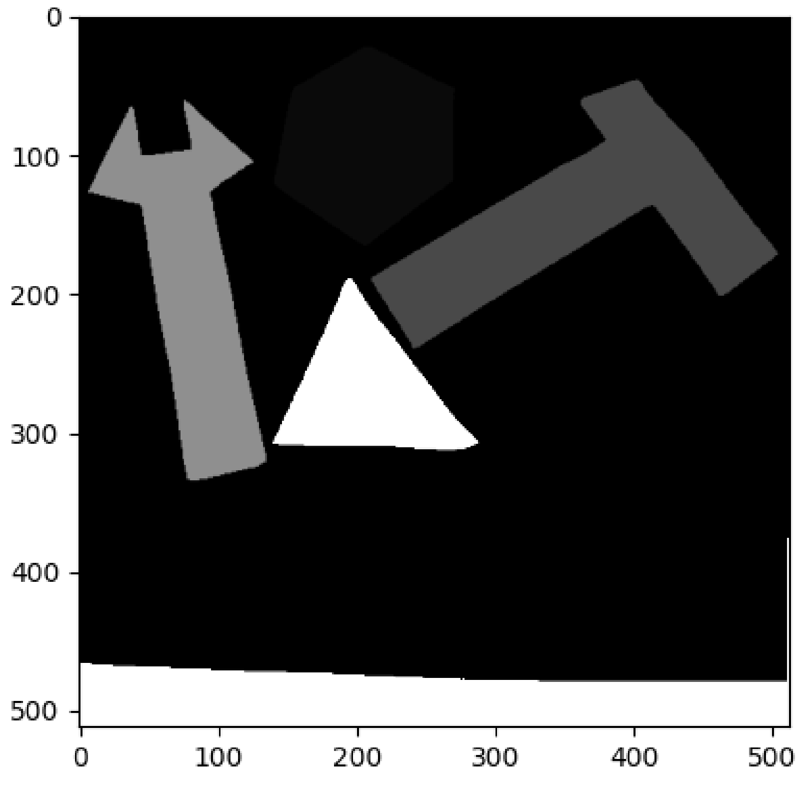


Image B_T



C1 = Threshold is equal to 1000



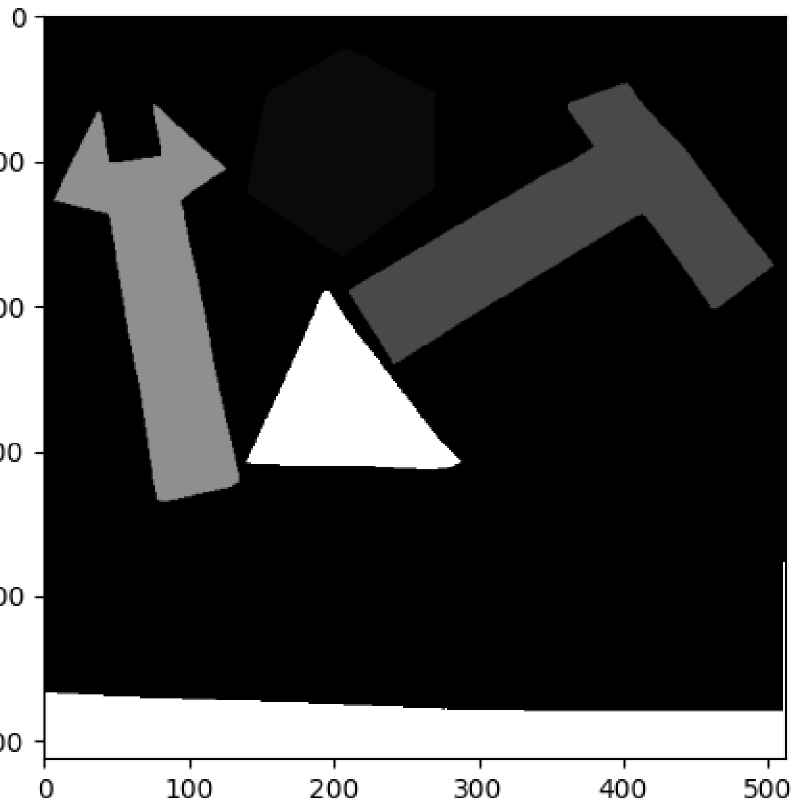
Comments: With this threshold, it appears to remove some of the components that were on the right side of the chart, specifically small components. There are only the 5, nicely visible components in the image.

Output:

```
5 components meet the size criterion.

Component #1:
==============================
Area: 13139
Centroid: [207.32978156632925, 91.73491133267372]
Bounding Box: [141, 270, 23, 165]
Axis of Elongation: 1.8214981616774228°
Eccentricity: 2.56
Perimeter: 448.1736649163083
Compactness: 15.29
Component #2:
==============================
Area: 19742
Centroid: [361.0750683821295, 146.54685442204436]
Bounding Box: [211, 503, 46, 239]
Axis of Elongation: 1.784250015026089°
Eccentricity: 2.84
Perimeter: 876.4966081312851
Compactness: 38.91
Component #3:
==============================
Area: 15309
Centroid: [81.30936050689137, 196.64609053497944]
Bounding Box: [7, 134, 61, 334]
Axis of Elongation: -0.21500142759515406°
Eccentricity: 2.93
Perimeter: 818.9848480983514
Compactness: 43.81
Component #4:
==============================
Area: 8907
Centroid: [207.20758953631974, 269.835410351409]
Bounding Box: [140, 287, 190, 312]
Axis of Elongation: -0.5414654929559858°
Eccentricity: 1.83
Perimeter: 447.20310216783054
Compactness: 22.45
Component #5:
==============================
Area: 18462
Centroid: [241.10589318600367, 492.9758422706099]
Bounding Box: [0, 511, 377, 511]
Axis of Elongation: -0.31615277509164563°
Eccentricity: 2.07
Perimeter: 1308.6274169979697
Compactness: 92.76
```

C2 = Threshold is equal to 5000



Comments: The larger size threshold does not appear to remove any of the components. Given the information below, this makes sense, as all areas are still above 5000.

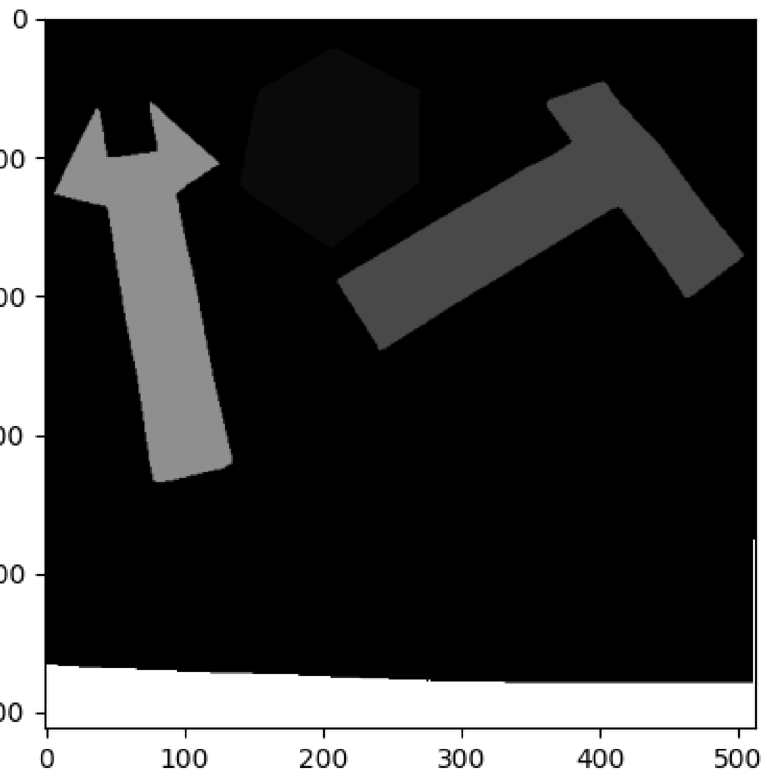
Output:

```
5 components meet the size criterion.

Component #1:
==============================
Area: 13139
```

```
Centroid: [207.32978156632925, 91.73491133267372]
Bounding Box: [141, 270, 23, 165]
Axis of Elongation: 1.8214981616774228°
Eccentricity: 2.56
Perimeter: 448.1736649163083
Compactness: 15.29
Component #2:
==============================
Area: 19742
Centroid: [361.0750683821295, 146.54685442204436]
Bounding Box: [211, 503, 46, 239]
Axis of Elongation: 1.784250015026089°
Eccentricity: 2.84
Perimeter: 876.4966081312851
Compactness: 38.91
Component #3:
==============================
Area: 15309
Centroid: [81.30936050689137, 196.64609053497944]
Bounding Box: [7, 134, 61, 334]
Axis of Elongation: -0.21500142759515406°
Eccentricity: 2.93
Perimeter: 818.9848480983514
Compactness: 43.81
Component #4:
==============================
Area: 8907
Centroid: [207.20758953631974, 269.835410351409]
Bounding Box: [140, 287, 190, 312]
Axis of Elongation: -0.5414654929559858°
Eccentricity: 1.83
Perimeter: 447.20310216783054
Compactness: 22.45
Component #5:
==============================
Area: 18462
Centroid: [241.10589318600367, 492.9758422706099]
Bounding Box: [0, 511, 377, 511]
Axis of Elongation: -0.31615277509164563°
Eccentricity: 2.07
Perimeter: 1308.6274169979697
Compactness: 92.76
```

C3 = Threshold is equal to 10000



Comments: With this threshold, it appears to remove the 1 component that remained and was under 10000 area. This shows how the increasing size filter might make the model run faster, as it doesn't need to look at all components, but it might miss some vital information. The triangle is the component that is no longer present.

Output:

```
4 components meet the size criterion.

Component #1:
==============================
Area: 13139
Centroid: [207.32978156632925, 91.73491133267372]
Bounding Box: [141, 270, 23, 165]
Axis of Elongation: 1.8214981616774228°
Eccentricity: 2.56
Perimeter: 448.1736649163083
Compactness: 15.29
```

```
Component #2:
==============================
Area: 19742
Centroid: [361.0750683821295, 146.54685442204436]
Bounding Box: [211, 503, 46, 239]
Axis of Elongation: 1.784250015026089°
Eccentricity: 2.84
Perimeter: 876.4966081312851
Compactness: 38.91
Component #3:
==============================
Area: 15309
Centroid: [81.30936050689137, 196.64609053497944]
Bounding Box: [7, 134, 61, 334]
Axis of Elongation: -0.21500142759515406°
Eccentricity: 2.93
Perimeter: 818.9848480983514
Compactness: 43.81
Component #4:
==============================
Area: 18462
Centroid: [241.10589318600367, 492.9758422706099]
Bounding Box: [0, 511, 377, 511]
Axis of Elongation: -0.31615277509164563°
Eccentricity: 2.07
Perimeter: 1308.6274169979697
Compactness: 92.76
```

Code:

```
import numpy as np
import matplotlib.pyplot as plt

# File path
file_path = "img/comb.img"

# Image dimensions
width, height = 512, 512
header_size = 512

def show_image(img,cmap_str='gist_gray'):
    norm = plt.Normalize(vmin=0, vmax=100)  # Normalize so that only positive values are highlighted
    plt.imshow(img, cmap=cmap_str,norm=norm)
    plt.show()

# Read the file
with open(file_path, "rb") as f:
    f.seek(header_size)
    image_data = np.frombuffer(f.read(), dtype=np.uint8).copy()

# Reshape into 2D array
image_array = image_data.reshape((height, width))

# Display the base comb image
# show_image(image_array)

# Question 1: Find the Binary Image
def find_binary_img(img):
    B_t = img.copy()
    for ind1, row in enumerate(B_t):
        for ind2, col in enumerate(row):
            B_t[ind1][ind2] = 0 if col < 128 else 1
    return B_t
```

```
b_t = find_binary_img(image_array)

# Display the binary image B_T
# show_image(b_t)

# Question 2: Find Connected Components Iteratively
def iter_connected_comps(img, filter):
    equivalence_table = dict()
    component_data = dict()
    list_equiv = []
    label = 0
    B_image = np.zeros_like(img, dtype=np.int16)

    # First loop through the image and label the components as their encountered, using a new label if
    # the top or left neighbor do not already have a label
    for ind1, row in enumerate(img):
        for ind2, col in enumerate(row):
            if col == 0:
                # if col1
                if ind2 == 0:
                    left_neighbor = 0
                else:
                    left_neighbor = B_image[ind1][ind2-1]

                # if row1
                if ind1 == 0:
                    top_neighbor = 0
                else:
                    top_neighbor = B_image[ind1-1][ind2]

                if top_neighbor > 0 or left_neighbor > 0:
                    min_label = min(x for x in [top_neighbor, left_neighbor] if x > 0)
                    B_image[ind1][ind2] = min_label
                else:
                    B_image[ind1][ind2] = label
                    label += 1

                # Check if the neighbors are already in the equivalence table
                if top_neighbor > 0 and left_neighbor > 0 and left_neighbor != top_neighbor:
                    list_equiv.append([left_neighbor,top_neighbor])

    # delete duplicates in equivalence list
    seen_unique = list()
    seen = set()
    for i in list_equiv:
```

```
    pair = frozenset(i)
    if pair not in seen:
        seen_unique.append(i)
        seen.add(pair)
list_equiv = seen_unique

# Loop through the equivalence list (labels that are equal) and build
# an equivalence table. This should consolidate so that all components
# within a singular object are equal to one another.
for i in list_equiv:

    in_items0 = len([key for key, value in equivalence_table.items() if i[0] in value])>0
    in_items1 = len([key for key, value in equivalence_table.items() if i[1] in value])>0

    # Combine lists
    if i[0] in equivalence_table and i[1] in equivalence_table:
        if i[1] != i[0]:
            equivalence_table[i[0]] = list(set(equivalence_table[i[0]]) | set(equivalence_table[i[1]])).append(i[1])
            del equivalence_table[i[1]]

    elif in_items0 and in_items1:
        key1 = [key for key, value in equivalence_table.items() if i[0] in value][0]
        key2 = [key for key, value in equivalence_table.items() if i[1] in value][0]
        if key1 != key2:
            equivalence_table[key1] = list(set(equivalence_table[key1]) | set(equivalence_table[key2]))
            del equivalence_table[key2]

    # Add if one list exists
    elif i[0] in equivalence_table and i[1] not in equivalence_table[i[0]]:
        if in_items1:
            val = [key for key, value in equivalence_table.items() if i[1] in value][0]
            if val != i[0]:
                equivalence_table[i[0]] = list(set(equivalence_table[i[0]]) | set(equivalence_table[val]))
                del equivalence_table[val]
        else:
            equivalence_table[i[0]].append(i[1])

    # Add if other list exists
    elif i[1] in equivalence_table and i[0] not in equivalence_table[i[1]]:
        if in_items0:
            val = [key for key, value in equivalence_table.items() if i[0] in value][0]
            equivalence_table[i[1]] = list(set(equivalence_table[i[1]]) | set(equivalence_table[val]))
            if val != i[1]:
                del equivalence_table[val]
        else:
            equivalence_table[i[1]].append(i[0])

        elif in_items0:
            key1 = [key for key, value in equivalence_table.items() if i[0] in value][0]
            equivalence_table[key1].append(i[1])

        elif in_items1:
            key1 = [key for key, value in equivalence_table.items() if i[1] in value][0]
            equivalence_table[key1].append(i[0])

        # if none exist
        else:
            equivalence_table[i[0]] = [i[1]]

    for ind1, row in enumerate(B_image):
        for ind2, col in enumerate(row):
            if col > 0:
                if col not in equivalence_table:
                    new_label = [key for key, value in equivalence_table.items() if col in value]
                    if len(new_label) > 0:
                        B_image[ind1][ind2] = new_label[0]

    for ind1, row in enumerate(B_image):
        for ind2, col in enumerate(row):
            if col > 0:
                if col not in component_data:
                    component_data[col] = {'size': 1, 'x': [ind2], 'y': [ind1]}
                else:
                    component_data[col]['size'] += 1
                    component_data[col]['x'].append(ind2)
                    component_data[col]['y'].append(ind1)

    filtered_comps = [x for x in component_data.keys() if component_data[x]['size'] > filter]
    keys = list(component_data.keys())
    for i in keys:
        if i not in filtered_comps:
            for ind1, row in enumerate(img):
                for ind2, col in enumerate(row):
                    if B_image[ind1][ind2] == i:
                        B_image[ind1][ind2] = 0

            del component_data[i]

    def calc_centroid(comp_data):
        for component in (comp_data):
```

```
        centroid_x = sum(component_data[component]['x'])/component_data[component]['size']
        centroid_y = sum(component_data[component]['y'])/component_data[component]['size']
        comp_data[component]['centroid'] = [centroid_x,centroid_y]
    return comp_data

# Find Centroids
component_data = calc_centroid(component_data)
# show_image(B_image)

def find_bounding_boxes(comp_data):
    for component in (comp_data):
        x_min = min(comp_data[component]['x'])
        x_max = max(comp_data[component]['x'])
        y_min = min(comp_data[component]['y'])
        y_max = max(comp_data[component]['y'])

        comp_data[component]['bounding_box'] = ([x_min,x_max,y_min,y_max])
    return component_data

# Find Bounding Boxes
component_data = find_bounding_boxes(component_data)

import math
def calc_elongation(comp_data):
    for component in comp_data:
        a, b, c = 0,0,0
        for ind,_ in enumerate(comp_data[component]['x']):
            a += (comp_data[component]['x'][ind])**2
            b += (comp_data[component]['x'][ind])*(comp_data[component]['y'][ind])
            c += (comp_data[component]['y'][ind])**2
        theta1 = math.atan(b/(a - c))/2
        theta2 = theta1 + math.pi/2
        x2_min = .5*(a+c)+.5*(a-c)*math.cos(2*theta1)+.5*b*math.sin(2*theta1)
        x2_max = .5*(a+c)+.5*(a-c)*math.cos(2*theta2)+.5*b*math.sin(2*theta2)
        if x2_min < x2_max:
            axis_elong = theta1
        else:
            axis_elong = theta2
            tmp = x2_min
            x2_min = x2_max
            x2_max = tmp


        eccentricity = math.sqrt(x2_max)/math.sqrt(x2_min)
        comp_data[component]['axis_of_elongation'] = axis_elong
        comp_data[component]['eccentricity'] = eccentricity
      return comp_data

    # Calc Elongation
    component_data = calc_elongation(component_data)

    def find_perimeter(comp_data,image):
      visited = np.zeros_like(image, dtype=bool)
      for component in comp_data:
        perimeter = 0

        perimeter_pts = []
        perimeter_y = []
        perimeter_flag = True
        ind1 = comp_data[component]['x'][0]
        ind2 = comp_data[component]['y'][0]
        previous_direction = 0
        directions = [[-1,0], [-1,-1], [0,-1],[1, -1],[1, 0],[1, 1],[0, 1], [-1, 1]]

        while perimeter_flag:
          perimeter_pts.append([ind1,ind2])
          if perimeter != 0 and ind1 == comp_data[component]['x'][0] and ind2 ==
comp_data[component]['y'][0]:
            break

          # [1][2][3]
          # [0][x][4]
          # [7][6][5]
          for i in range(8):
            direction = (previous_direction + i + 1) % 8
            diff_x, diff_y = directions[direction]
            if ind1+diff_x >= 0 and ind1+diff_x < 512:
              if ind2+diff_y >= 0 and ind2+diff_y < 512:
                if image[ind2+diff_y,ind1+diff_x] == component:

                  # print(ind1,ind2)
                  # print(ind1+diff_x,ind2+diff_y)
                  # import pdb;pdb.set_trace()
                  perimeter += math.sqrt((diff_x**2) + (diff_y**2))
                  previous_direction = (direction + 4)%8
                  ind2 = ind2+diff_y
                  ind1 = ind1+diff_x
                  break

        comp_data[component]['perimeter'] = perimeter
```

```
            comp_data[component]['perimeter_pts'] = perimeter_pts
        return comp_data

    component_data = find_perimeter(component_data,B_image)

    def draw_perimeters(comp_data):
        for component in comp_data:
            x_coords, y_coords = zip(*comp_data[component]['perimeter_pts'])
            # Create a scatter plot
            plt.scatter(x_coords, y_coords)

            # Add titles and labels
            plt.title("Scatter Plot of Points")
            plt.xlabel("X-axis")
            plt.ylabel("Y-axis")

            # Show the plot
            plt.show()

    def calc_compactness(comp_data):
        for component in comp_data:
            comp_data[component]['compactness'] =
(comp_data[component]['perimeter']**2)/(comp_data[component]['size'])
        return comp_data

    component_data = calc_compactness(component_data)

    return B_image, equivalence_table, component_data

b_image, eql_table, cd = iter_connected_comps(b_t,1000)
b_image2, eql_table2, cd2 = iter_connected_comps(b_t,5000)
b_image3, eql_table3, cd3 = iter_connected_comps(b_t,10000)

# # show_image(b_image)
# # show_image(b_image2)
# show_image(b_image3)

import pdb;pdb.set_trace()
def print_comps(comp_data):
    component_num = 0
    for component in comp_data:
        component_num += 1
        area = comp_data[component]['size']
        centroid = comp_data[component]['centroid']
        bounding_box = comp_data[component]['bounding_box']
        axis_of_elongation = comp_data[component]['axis_of_elongation']
        eccentricity = comp_data[component]['eccentricity']
        perimeter = comp_data[component]['perimeter']
        compactness = comp_data[component]['compactness']

        print(f"Component #{component_num}:")
        print(f"{'='*30}")
        print(f"Area: {area}")
        print(f"Centroid: {centroid}")
        print(f"Bounding Box: {bounding_box}")
        print(f"Axis of Elongation: {axis_of_elongation}°")
        print(f"Eccentricity: {eccentricity:.2f}")
        print(f"Perimeter: {perimeter}")
        print(f"Compactness: {compactness:.2f}")
import pdb;pdb.set_trace()

print_comps(cd)
print_comps(cd2)
print_comps(cd3)
```