Computer Vision Assignment 3

Student Name: Uğur Ali Kaplan
Student ID: 150170042

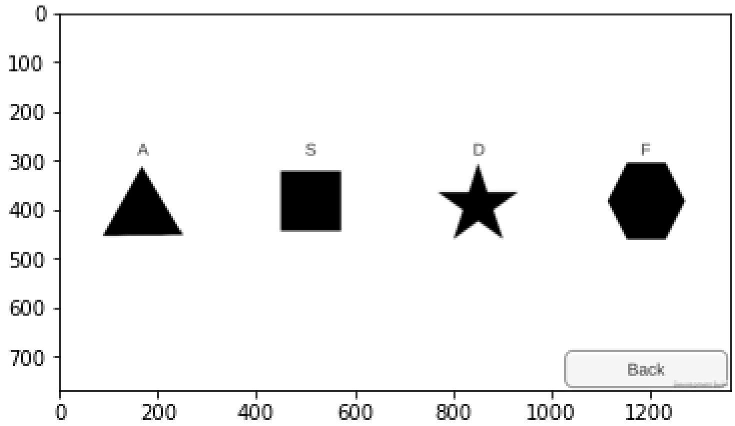
Part 1

Code can be found in the `ss.py` file. I am copying it here for reference:

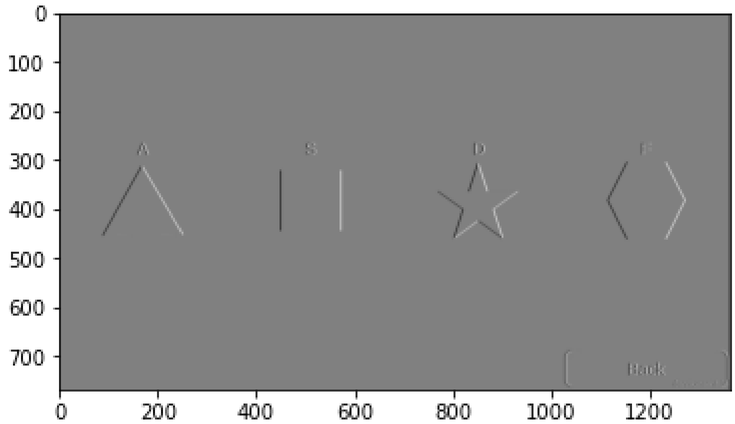
```
import time
import cv2
import pyautogui as pg
import numpy as np
from scipy.signal import convolve2d


time.sleep(5)
ss = pg.screenshot()
ss.save("ss.png")
ss = cv2.imread("ss.png", 0)

# Define vertical and horizontal Sobel filters

S_x = np.array([
    [1, 0, -1],
    [2, 0, -2],
    [1, 0, -1]
])

S_y = np.array([
    [1, 2, 1],
    [0, 0, 0],
    [-1, -2, -1]
])

x = convolve2d(ss, S_x, mode="same")
y = convolve2d(ss, S_y, mode="same")

filtered = np.sqrt(x**2 + y**2)
cv2.imwrite("filtered.png", filtered)
cv2.imwrite("sobel_x.png", np.abs(x))
cv2.imwrite("sobel_y.png", np.abs(y))
print("Done")
```

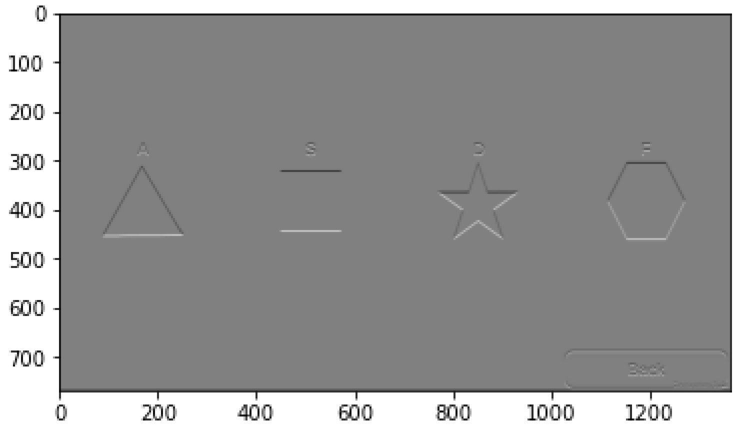
Results are saved as `filtered.png`, `sobel_x.png`, and `sobel_y.png`.

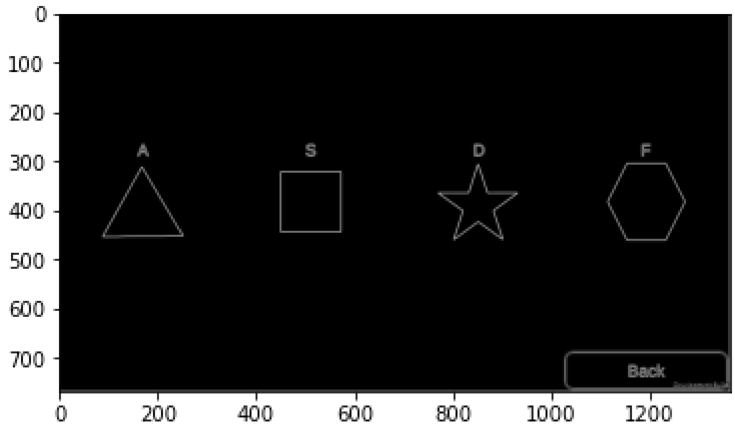
Screenshot:



Sobel Filter Results:





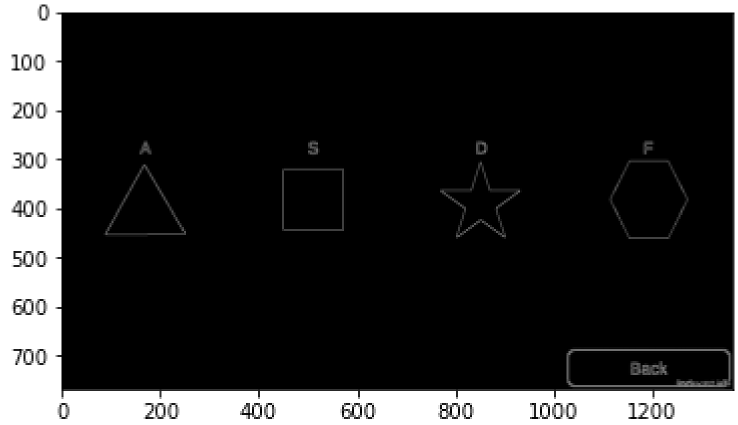


Part 2

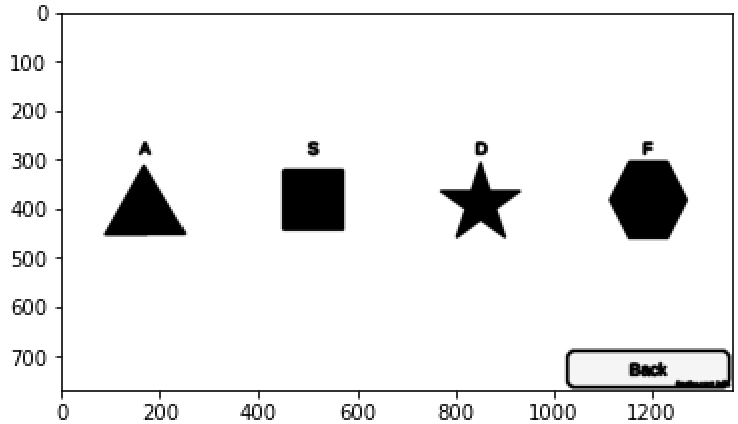
For this part, I have used Canny Edge Detector as instructed.

```
cannyEdges = cv2.Canny(ss, 100, 200) # ss is the screenshot
cv2.imwrite("cannyEdges.png", cannyEdges)
contours, hierarchy = cv2.findContours(cannyEdges, cv2.RETR_LIST,
cv2.CHAIN_APPROX_NONE)
cv2.imwrite("contours.png", cv2.drawContours(ss, contours, -1, (0,255,0), 3))
```

Edges:



Contours:



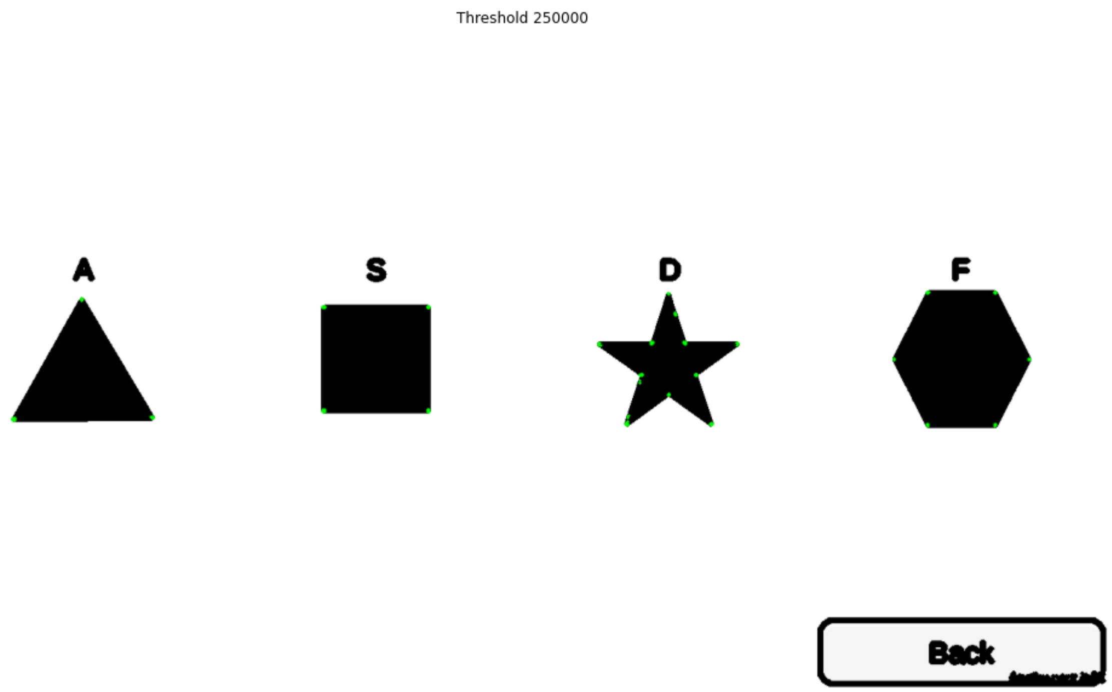
Part 3

In this part, I have implemented a minimum eigenvalue corner detector. Since it is stated we should run the detector on the shapes, I have only filtered a specific part of the screenshot.

```
# Use the image gradients calculated by the Sobel Filter in Part 1.
image_x = x_filtered
image_y = y_filtered

# Loop through the calculated gradients
H, W = image_x.shape
window_H, window_W = 3, 3
corners = list()

for h in range(295, 470 - window_H):
    if h % 10 == 0:
            print(f"H: {h}") # Takes a long time, need to see the progress
    for w in range(0, W - window_W):
        gradients_x = image_x[h:h+window_H, w:w+window_W].astype(np.float32) #
Gradients for the current pixel
        gradients_y = image_y[h:h+window_H, w:w+window_W].astype(np.float32)
```

```
        mean_x, mean_y = np.mean(gradients_x), np.mean(gradients_y)
        gradients_x -= mean_x # Subtract the mean and center the gradients
        gradients_y -= mean_y
        cov = np.zeros((2, 2)) # Initialize covariance matrix
        cov[0, 0] = np.sum(np.multiply(gradients_x, gradients_x))
        cov[0, 1] = np.sum(np.multiply(gradients_x, gradients_y))
        cov[1, 0] = np.sum(np.multiply(gradients_x, gradients_y))
        cov[1, 1] = np.sum(np.multiply(gradients_y, gradients_y))

        eigs = np.linalg.eigvalsh(cov) # Find eigenvalues of the matrix, we can use eigvalsh since cov is symmetric
        if min(eigs) > 100000: # Apply minimum threshold
            corners.append((h, w, min(eigs)))

corners_th250000 = list()
for point in corners:
    if point[2] > 250000:
        corners_th250000.append((point[0], point[1]))

corners_th250000 = np.array(corners_th250000)

# Show the corners on top of the screenshot
fig, ax = plt.subplots(1, figsize = (20, 10))
ax.imshow(ss, cmap="gray")
ax.scatter(corners_th250000[:, 1], corners_th250000[:, 0], s=1, color="lime")
ax.set_title("Threshold 250000")
plt.axis("off")
plt.show()
```

Result:



Part 4

In this part, I have used contours, edge detectors and minimum eigenvalue corner detectors from opencv. My screen resolution is 1366×768.

Code for this part can be found in `playgame.py`. My success rate is around 88% ($\frac{16}{18}$) for both move sequences.

I am using contour count to detect if shapes are overlapping with the input area. If there is a need of pressing the key, I clean the rectangle in the background to prevent the confusion. Then, I use the minimum eigenvalue corner founder from the OpenCV module.

My threshold is only 1, and I have determined ranges for the shapes using this threshold. Increasing the threshold causes information loss, and it is easier to determine if an object is square or triangle if I get the raw output.

```
time.sleep(2)
found = 0
while found < 19:
    time.sleep(0.5)
    ss = np.array(pg.screenshot())
    ss = cv2.cvtColor(ss, cv2.COLOR_BGR2GRAY) # Gray image

    region = ss[580:764, 510:850] # Important part of the screen
    cannyEdges = cv2.Canny(region,100,200) # Detect Edges
    contours, _ = cv2.findContours(cannyEdges, cv2.RETR_LIST,
cv2.CHAIN_APPROX_NONE) # Find Contours

    # If shape is intersecting with the rectangle in the background
    if len(contours) >= 2:
        found += 1 # We are pressing a key
        # Widen the search area
        region = ss[580:764, 510:860]
        region[region == 206] = 255 # Remove the square in the background

        # Detect corners
        dst = cv2.cornerMinEigenVal(region, 2, 1)

        # Normalize
        dst_norm = np.empty(dst.shape, dtype=np.float32)
        cv2.normalize(dst, dst_norm, alpha=0, beta=255,
norm_type=cv2.NORM_MINMAX)
        dst_norm_scaled = cv2.convertScaleAbs(dst_norm)

        e = dst_norm_scaled[dst_norm_scaled > 1].shape[0] # Number of corners

        if e > 1110:
            # hexagon
            #print(f"Hexagon {e}")
            pg.keyDown("f")
            pg.keyUp("f")
        elif e > 400:
            # star
            #print(f"Star {e}")
            pg.keyDown("d")
            pg.keyUp("d")
        elif e > 100:
            # triangle
            #print(f"Triangle {e}")
            pg.keyDown("a")
            pg.keyUp("a")
        else:
```

```
            # square
            #print(f"Square {e}")
            pg.keyDown("s")
            pg.keyUp("s")
```