Name: Lê Hoàng Hiếu

MSSV: 2151013023

1. Capture frames from MP4 or true camera, overlay your text and logo (png/jpg) into frame

```
import cv2
import numpy as np

cap = cv2.VideoCapture('video.mp4')

def read_transparent_png(filename):
    image_4channel = cv2.imread(filename, cv2.IMREAD_UNCHANGED)
    alpha_channel = image_4channel[:,:,3]
    rgb_channels = image_4channel[:,:,:3]

    white_background_image = np.ones_like(rgb_channels, dtype=np.uint8) * 255

    alpha_factor = alpha_channel[:,:,np.newaxis].astype(np.float32) / 255.0
    alpha_factor = np.concatenate((alpha_factor,alpha_factor,alpha_factor), axis=2)

    base = rgb_channels.astype(np.float32) * alpha_factor
    white = white_background_image.astype(np.float32) * (1 - alpha_factor)
    final_image = base + white
    return final_image.astype(np.uint8)

# Load your icon image
icon = read_transparent_png('Logo.png')

icon_width_percentage = 10
icon_height_percentage = 10

icon_position = (10, 10)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    icon_width = 100
    icon_height = 40

```

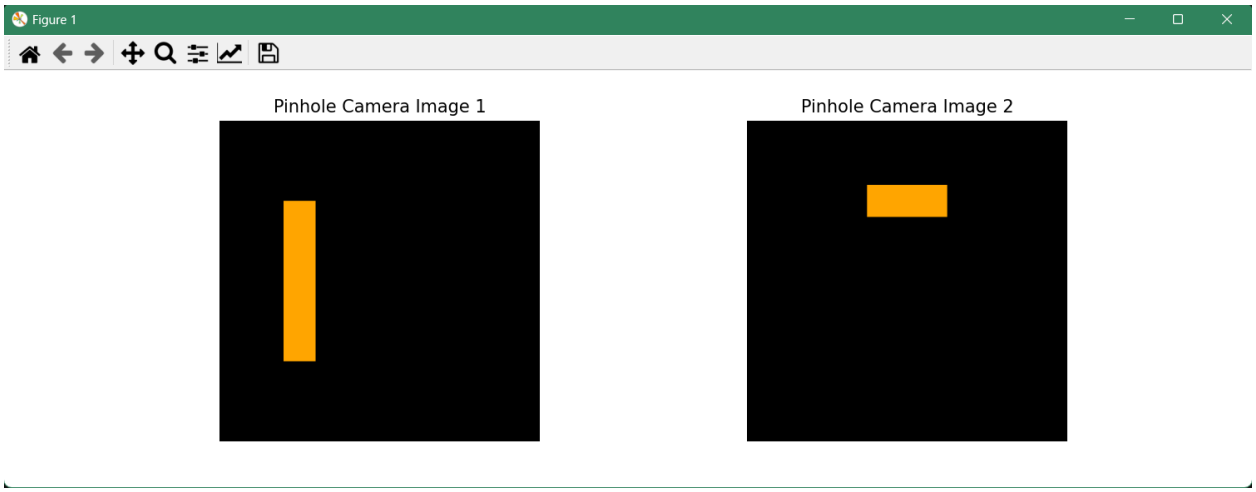
```
    icon_resized = cv2.resize(icon, (icon_width, icon_height))

    # Overlay your icon onto the frame
    frame[icon_position[1]:icon_position[1] + icon_height,
icon_position[0]:icon_position[0] + icon_width] = icon_resized

    text = "Cocogoat"
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(frame, text, (120, 40), font, 0.5, (255, 255, 255), 1,
cv2.LINE_AA)

    # Display the frame
    cv2.imshow('Frame', frame)

    if cv2.waitKey(24) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Output:

2. Simulate image formation by drawing the object through pinhole camera

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

def pinholeCamera(imageSize=(400, 400), pinholeSize=(3, 3), objectPosition=(200,
200), objectSize=(20, 100)):
    # Create a black image
    image = np.zeros((imageSize[0], imageSize[1], 3), dtype=np.uint8)
    # Simulate pinhole by setting a region to white where light passes through
    pinholeStart = (objectPosition[0] - pinholeSize[0] // 2, objectPosition[1] -
pinholeSize[1] // 2)
    pinholeEnd = (pinholeStart[0] + pinholeSize[0], pinholeStart[1] +
pinholeSize[1])
    image[pinholeStart[1]:pinholeEnd[1], pinholeStart[0]:pinholeEnd[0]] = [255,
255, 255]
    # Simulate the object in the scene
    candleStart = (objectPosition[0] - objectSize[0] // 2, objectPosition[1] -
objectSize[1] // 2)
    candleEnd = (candleStart[0] + objectSize[0], candleStart[1] + objectSize[1])
    image[candleStart[1]:candleEnd[1], candleStart[0]:candleEnd[0]] = [0, 165,
255]  # Color for the object
    return image

def plotImages(images, titles):
    fig, axes = plt.subplots(1, len(images), figsize=(12, 4))
    for ax, image, title in zip(axes, images, titles):
        ax.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB
for proper display
        ax.set_title(title)
        ax.axis('off')
    plt.show()

# Simulate pinhole camera image formation with a candle object
pinholeSize = (10, 10)
I1 = pinholeCamera((400, 400), pinholeSize, (100, 200), (40, 200))
I2 = pinholeCamera((400, 400), pinholeSize, (200, 100), (100, 40))

# Display simulated images
plotImages([I1, I2], ['Pinhole Camera Image 1', 'Pinhole Camera Image 2'])
```

Output:

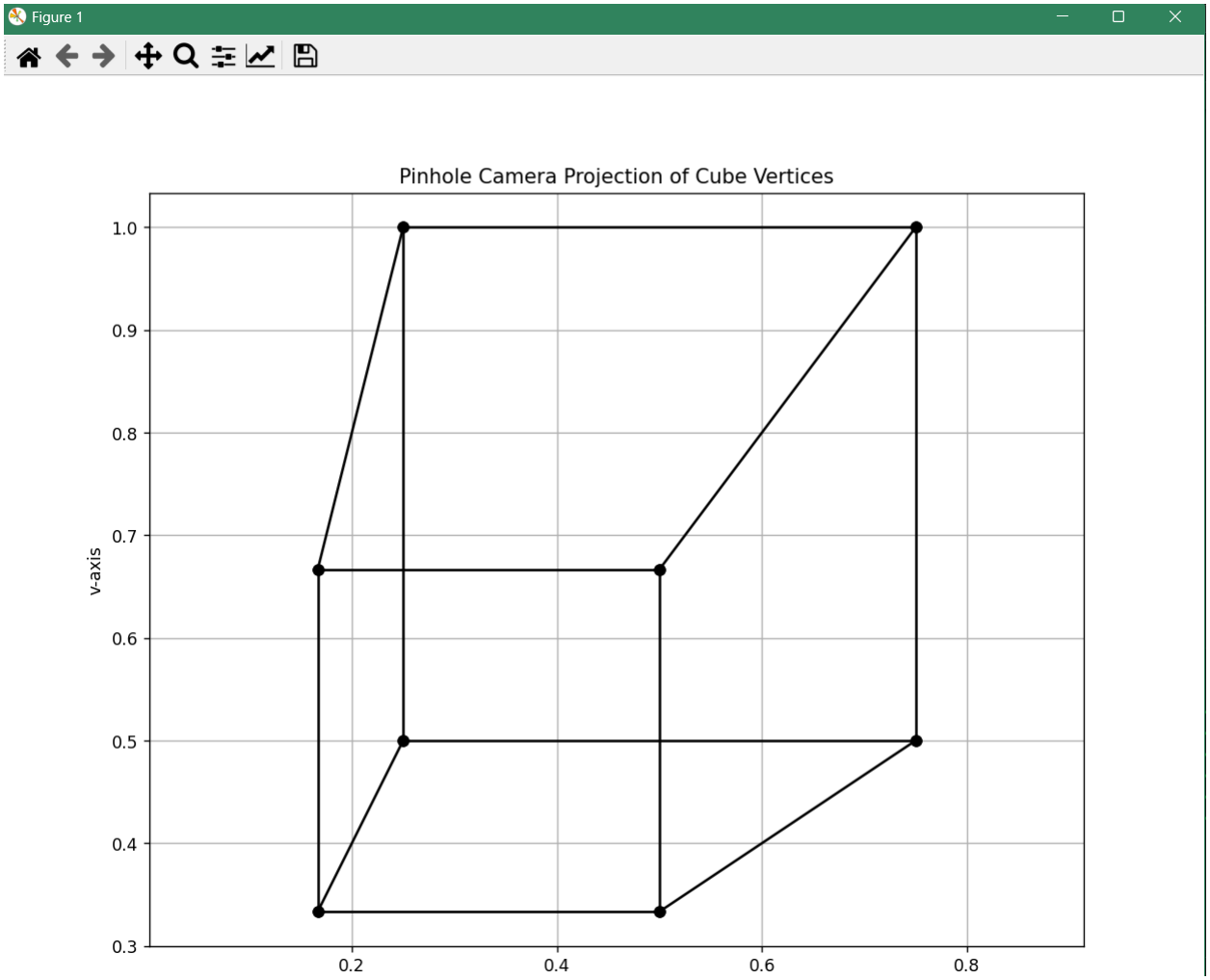


3. Perspective 3D cube to image plane

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

vec = np.array
A = vec([1, 1, 1])
B = vec([-1, 1, 1])
C = vec([1, -1, 1])
D = vec([-1, -1, 1])
E = vec([1, 1, -1])
F = vec([-1, 1, -1])
G = vec([1, -1, -1])
H = vec([-1, -1, -1])
camera = vec([2, 3, 5])
Points = dict(zip("ABCDEFGH", [A, B, C, D, E, F, G, H]))
edges = ["AB", "CD", "EF", "GH", "AC", "BD", "EG", "FH", "AE", "CG", "BF", "DH"]
points = {k: p - camera for k, p in Points.items()}

def pinhole(v):
    x, y, z = v
    return vec([x / z, y / z])

uvs = {k: pinhole(p) for k, p in points.items()}

plt.figure(figsize=(10, 10))
for a, b in edges:
    ua, va = uvs[a]
    ub, vb = uvs[b]
    plt.plot([ua, ub], [va, vb], "ko-")
```

```
plt.title('Pinhole Camera Projection of Cube Vertices')
plt.xlabel('u-axis')
plt.ylabel('v-axis')
plt.axis("equal")
plt.grid()
plt.show()
```

Output:



4. Rotate 3D cube by the axis (X, Y, Z), project rotated object to image plane

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Define points
```

```
A = np.array([1, 1, 1])
B = np.array([-1, 1, 1])
C = np.array([1, -1, 1])
D = np.array([-1, -1, 1])
E = np.array([1, 1, -1])
F = np.array([-1, 1, -1])
G = np.array([1, -1, -1])
H = np.array([-1, -1, -1])

camera = np.array([2, 3, 5])

Points = dict(zip("ABCDEFGH", [A, B, C, D, E, F, G, H]))

edges = ["AB", "CD", "EF", "GH", "AC", "BD", "EG", "FH", "AE", "CG", "BF", "DH"]
points = {k: v - camera for k, v in Points.items()}


def pinhole(v):
    x, y, z = v
    if z == 0:
        return np.array([float('inf'), float('inf')])
    return np.array([x / z, y / z])


def rotate(R, v):
    return np.dot(R, v)


def getRotX(angle):
    Rx = np.zeros((3, 3))
    Rx[0, 0] = 1
    Rx[1, 1] = np.cos(angle)
    Rx[1, 2] = -np.sin(angle)
    Rx[2, 1] = np.sin(angle)
    Rx[2, 2] = np.cos(angle)

    return Rx


def getRotY(angle):
    Ry = np.zeros((3, 3))
    Ry[0, 0] = np.cos(angle)
    Ry[0, 2] = -np.sin(angle)
    Ry[2, 0] = np.sin(angle)
    Ry[2, 2] = np.cos(angle)
    Ry[1, 1] = 1
```

```
    return Ry


def getRotZ(angle):
    Rz = np.zeros((3, 3))
    Rz[0, 0] = np.cos(angle)
    Rz[0, 1] = -np.sin(angle)
    Rz[1, 0] = np.sin(angle)
    Rz[1, 1] = np.cos(angle)
    Rz[2, 2] = 1

    return Rz


angles = [30, 40, 50]

# Plot for Z rotations
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
for i, angle_z in enumerate(angles):
    Rz = getRotZ(np.degrees(angle_z))

    # Apply rotation to points
    ps = {key: rotate(Rz, value) for key, value in points.items()}
    uvs = {key: pinhole(value) for key, value in ps.items()}

    # Plot edges
    for a, b in edges:
        ua, va = uvs[a]
        ub, vb = uvs[b]
        ax[i].plot([ua, ub], [va, vb], "ko-")

    ax[i].set_title(f"Z{angle_z}")
    ax[i].axis("equal")
    ax[i].grid()

plt.show()

# Plot for Y rotations
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
for i, angle_y in enumerate(angles):
    Ry = getRotY(np.degrees(angle_y))

    # Apply rotation to points
    ps = {key: rotate(Ry, value) for key, value in points.items()}
```

```
    uvs = {key: pinhole(value) for key, value in ps.items()}

    # Plot edges
    for a, b in edges:
        ua, va = uvs[a]
        ub, vb = uvs[b]
        ax[i].plot([ua, ub], [va, vb], "ko-")

    ax[i].set_title(f"Y{angle_y}")
    ax[i].axis("equal")
    ax[i].grid()

plt.show()

# Plot for X rotations
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
for i, angle_x in enumerate(angles):
    Rx = getRotX(np.degrees(angle_x))

    # Apply rotation to points
    ps = {key: rotate(Rx, value) for key, value in points.items()}
    uvs = {key: pinhole(value) for key, value in ps.items()}

    # Plot edges
    for a, b in edges:
        ua, va = uvs[a]
        ub, vb = uvs[b]
        ax[i].plot([ua, ub], [va, vb], "ko-")

    ax[i].set_title(f"X{angle_x}")
    ax[i].axis("equal")
    ax[i].grid()

plt.show()
```

Output:

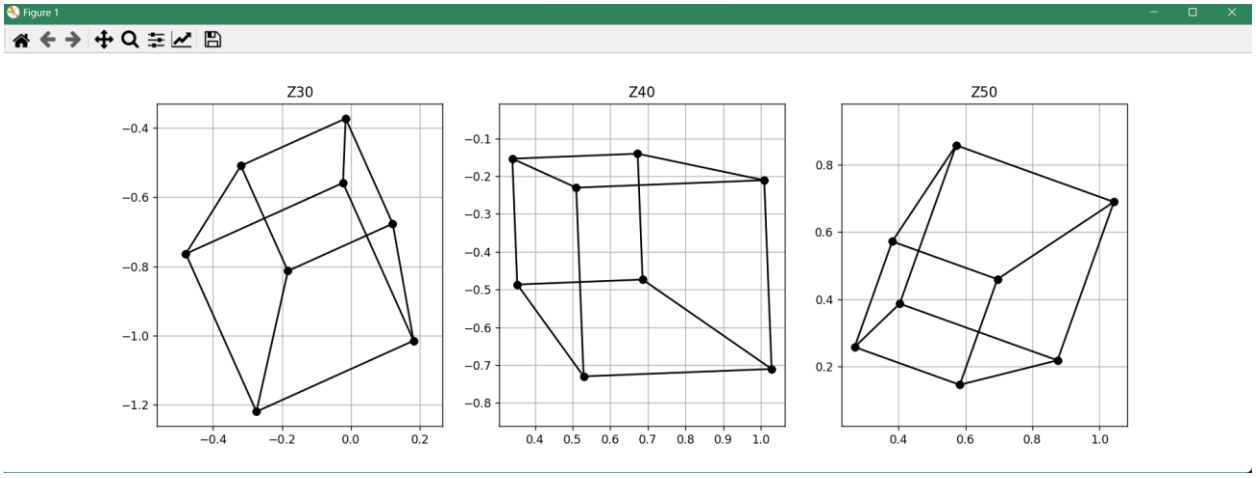


Figure 1
Z30
Z40
Z50

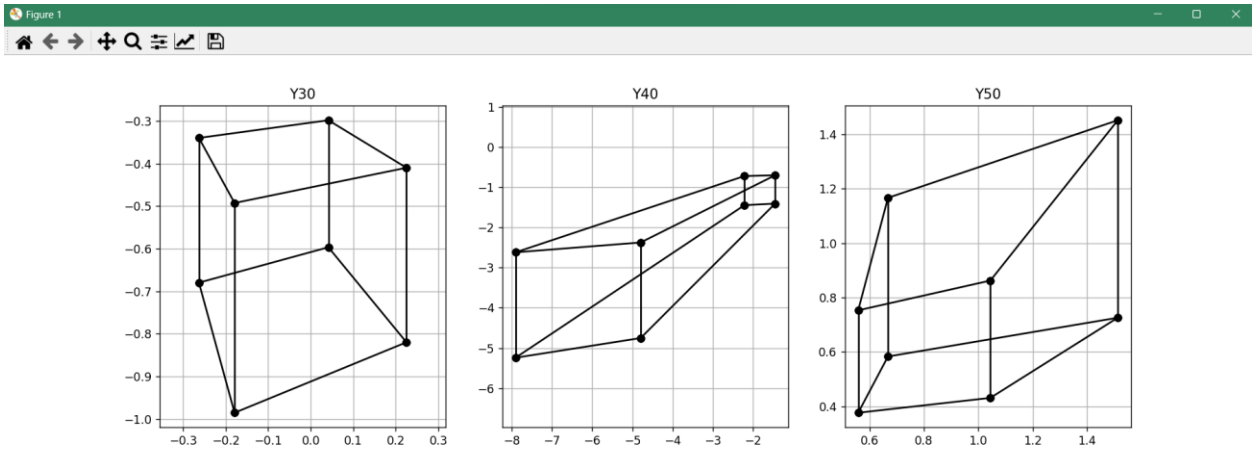


Figure 1
Y30
Y40
Y50

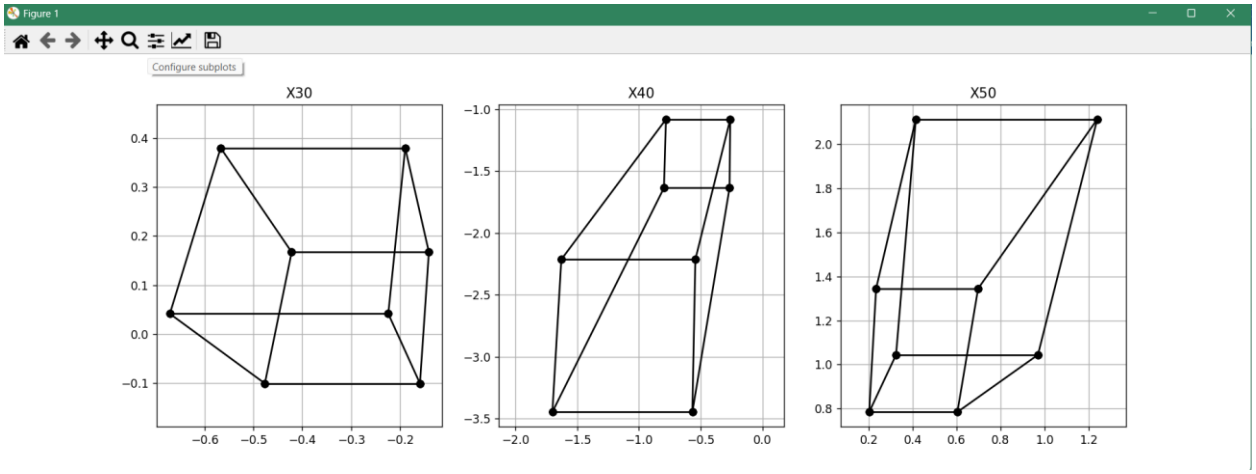


Figure 1
Configure subplots
X30
X40
X50