IMPORT LIBRARIES

In [2]:

```
!pip install sounddevice
```

```
Requirement already satisfied: sounddevice in c:\users\dummy\anaconda3\anaconda\lib\site-
packages (0.4.6)
Requirement already satisfied: CFFI>=1.0 in c:\users\dummy\anaconda3\anaconda\lib\site-pa
ckages (from sounddevice) (1.15.1)
Requirement already satisfied: pycparser in c:\users\dummy\anaconda3\anaconda\lib\site-pa
ckages (from CFFI>=1.0->sounddevice) (2.21)
```

In [3]:

```
!pip install soundfile
```

```
Collecting soundfile
  Using cached soundfile-0.12.1-py2.py3-none-win_amd64.whl (1.0 MB)
Requirement already satisfied: cffi>=1.0 in c:\users\dummy\anaconda3\anaconda\lib\site-pa
ckages (from soundfile) (1.15.1)
Requirement already satisfied: pycparser in c:\users\dummy\anaconda3\anaconda\lib\site-pa
ckages (from cffi>=1.0->soundfile) (2.21)
Installing collected packages: soundfile
Successfully installed soundfile-0.12.1
```

In [4]:

```
!pip install pydub
```

```
Requirement already satisfied: pydub in c:\users\dummy\anaconda3\anaconda\lib\site-packag
es (0.25.1)
```

In [2]:

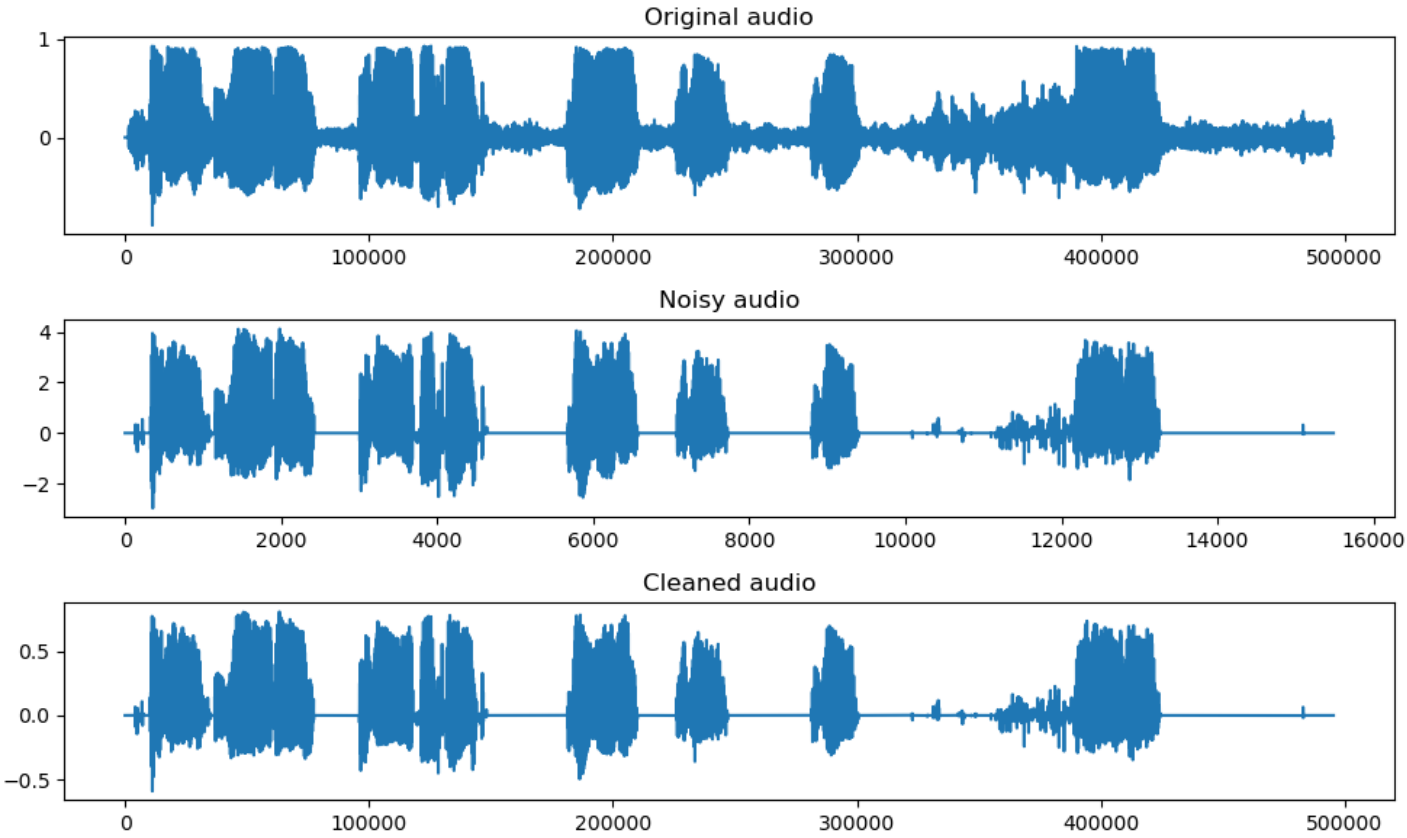
```
import numpy as np
import pywt
import pywt.data
import matplotlib.pyplot as plt
import soundfile as sf
import sounddevice as sd
```

AUDIO TASK

In [ ]:

```
audio, sample_rate = sf.read('C:\\Users\\k200237\\Downloads\\Anas.mp3')

wavelet = 'db4'
level = 5
coeffs = pywt.wavedec(audio, wavelet, level=level)

threshold = 1

threshold_coeffs = [pywt.threshold(c, threshold, mode='soft') for c in coeffs]

denoised_audio = pywt.waverec(threshold_coeffs, wavelet)

sf.write('cleaned_audio1.wav', denoised_audio, sample_rate)

sd.play(denoised_audio, sample_rate)
sd.wait()

plt.figure(figsize=(10, 6))
```

```
plt.subplot(3, 1, 1)
plt.title('Original audio')
plt.plot(audio)

plt.subplot(3, 1, 2)
plt.title('Noisy audio')
plt.plot(threshold_coeffs[0])

plt.subplot(3, 1, 3)
plt.title('Cleaned audio')
plt.plot(denoised_audio)

plt.tight_layout()
plt.show()
```

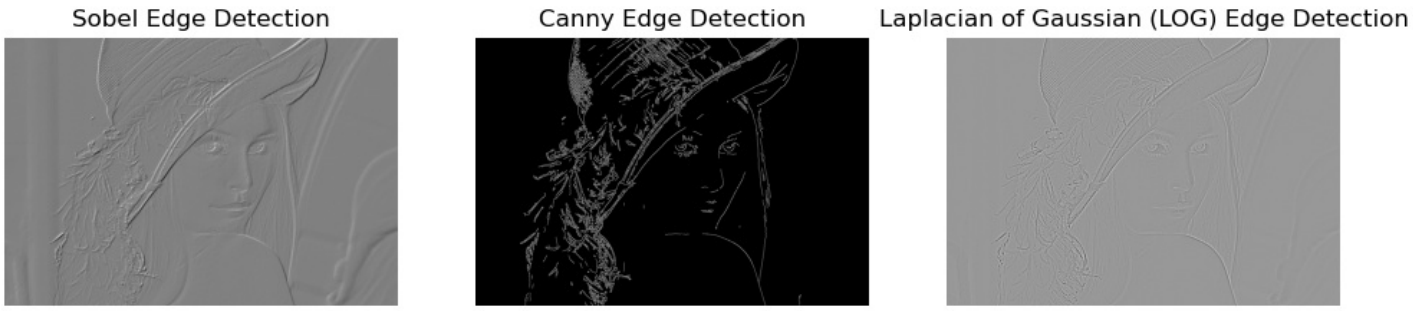


Laplacian of Gaussian (LoG)

In [ ]:

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

image = cv2.imread('C:\\Users\\k200237\\Desktop\\Leena.jpg', cv2.IMREAD_GRAYSCALE)

sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)
sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)
sobel_combined = cv2.addWeighted(sobel_x, 0.03, sobel_y, 0.03, 0)

canny = cv2.Canny(image, 100, 200)

image_log = cv2.GaussianBlur(image, (5, 5), 0)
image_log = cv2.Laplacian(image_log, cv2.CV_64F)

plt.figure(figsize=(12, 6))

plt.subplot(131)
plt.imshow(sobel_combined, cmap='gray')
plt.title('Sobel Edge Detection')
plt.axis('off')
```

```
plt.subplot(132)
plt.imshow(canny, cmap='gray')
plt.title('Canny Edge Detection')
plt.axis('off')

plt.subplot(133)
plt.imshow(image_log, cmap='gray')
plt.title('Laplacian of Gaussian (LOG) Edge Detection')
plt.axis('off')

plt.show()
```



LINES

In [ ]:

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


img = cv2.imread("/content/sample_data/Lines.JPG", cv2.IMREAD_COLOR)  # "lane.png" is the
filename

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

edges = cv2.Canny(gray, 50, 200)

lines = cv2.HoughLinesP(edges, 1, np.pi/180, 68, minLineLength=75, maxLineGap=250)

for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 2)

plt.subplot(131)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(132)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Lines Detected')
plt.axis('off')

plt.subplot(133)
plt.imshow(edges, cmap='gray')
plt.title('Edges Detected')
plt.axis('off')

plt.show()
```

Object Recognition

In [ ]:

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

reference_on_image = cv2.imread("/content/sample_data/ONN.JPG", cv2.IMREAD_COLOR)
reference_off_image = cv2.imread("/content/sample_data/OFF.JPG", cv2.IMREAD_COLOR)

sift = cv2.SIFT_create()

keypoints_on, descriptors_on = sift.detectAndCompute(reference_on_image, None)
keypoints_off, descriptors_off = sift.detectAndCompute(reference_off_image, None)

lab_image = cv2.imread("/content/sample_data/LAB.JPG", cv2.IMREAD_COLOR)

keypoints_lab, descriptors_lab = sift.detectAndCompute(lab_image, None)

FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
search_params = dict(checks=50)
flann = cv2.FlannBasedMatcher(index_params, search_params)

matches_on = flann.knnMatch(descriptors_on, descriptors_lab, k=2)
matches_off = flann.knnMatch(descriptors_off, descriptors_lab, k=2)

matching_threshold = 0.7
min_match_count = 10

matched_computers_on = []
matched_computers_off = []

for i, kp in enumerate(keypoints_lab):
    matches_on_i = matches_on[i] if i < len(matches_on) else []
    matches_off_i = matches_off[i] if i < len(matches_off) else []

    good_matches_on = [match for match in matches_on_i if match.distance < matching_thre
shold]
    good_matches_off = [match for match in matches_off_i if match.distance < matching_th
reshold]

    if len(good_matches_on) > min_match_count:
        matched_computers_on.append(kp)

    if len(good_matches_off) > min_match_count:
        matched_computers_off.append(kp)

def label_and_highlight_computers(image, keypoints, label, color):
    for kp in keypoints:
        x, y = kp.pt
        size = kp.size
        angle = kp.angle
        rect = ((x, y), (size, size), angle)
        box = cv2.boxPoints(rect)
        box = np.int0(box)
        cv2.drawContours(image, [box], 0, color, 2)
        text_x = int(x - size / 2)
        text_y = int(y)
        cv2.putText(image, label, (text_x, text_y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color
, 2)

label_and_highlight_computers(lab_image, matched_computers_on, "ON", (0, 255, 0))
label_and_highlight_computers(lab_image, matched_computers_off, "OFF", (0, 0, 255))

lab_image_rgb = cv2.cvtColor(lab_image, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(8, 8))
```

```
plt.imshow(lab_image_rgb)
plt.axis('off')
plt.title("Recognized Computer States")
plt.show()
```

Your Panoramic Image

In [ ]:

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

image_paths = ["/content/sample_data/1.jpg", "/content/sample_data/2.jpg", "/content/samp
le_data/3.jpg", "/content/sample_data/4.jpg"]

sift = cv2.SIFT_create()

stitched_image = cv2.imread(image_paths[0], cv2.IMREAD_COLOR)

for i in range(1, len(image_paths)):
    image2 = cv2.imread(image_paths[i], cv2.IMREAD_COLOR)

    keypoints1, descriptors1 = sift.detectAndCompute(stitched_image, None)
    keypoints2, descriptors2 = sift.detectAndCompute(image2, None)

    bf = cv2.BFMatcher()

    matches = bf.knnMatch(descriptors1, descriptors2, k=2)

    good_matches = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            good_matches.append(m)

    if len(good_matches) > 5:
        pts1 = np.float32([keypoints1[m.queryIdx].pt for m in good_matches]).reshape(-1,
1, 2)
        pts2 = np.float32([keypoints2[m.trainIdx].pt for m in good_matches]).reshape(-1,
1, 2)
```

```
        M, mask = cv2.findHomography(pts2, pts1, cv2.RANSAC, 5.0)

        image2_warped = cv2.warpPerspective(image2, M, (stitched_image.shape[1] + image2
.shape[1], stitched_image.shape[0]))

        stitched_image = cv2.hconcat([stitched_image, image2_warped])

stitched_image_rgb = cv2.cvtColor(stitched_image, cv2.COLOR_BGR2RGB)
plt.imshow(stitched_image_rgb)
plt.axis('off')
plt.show()
```

Lane detection using the Hough Line Transformation

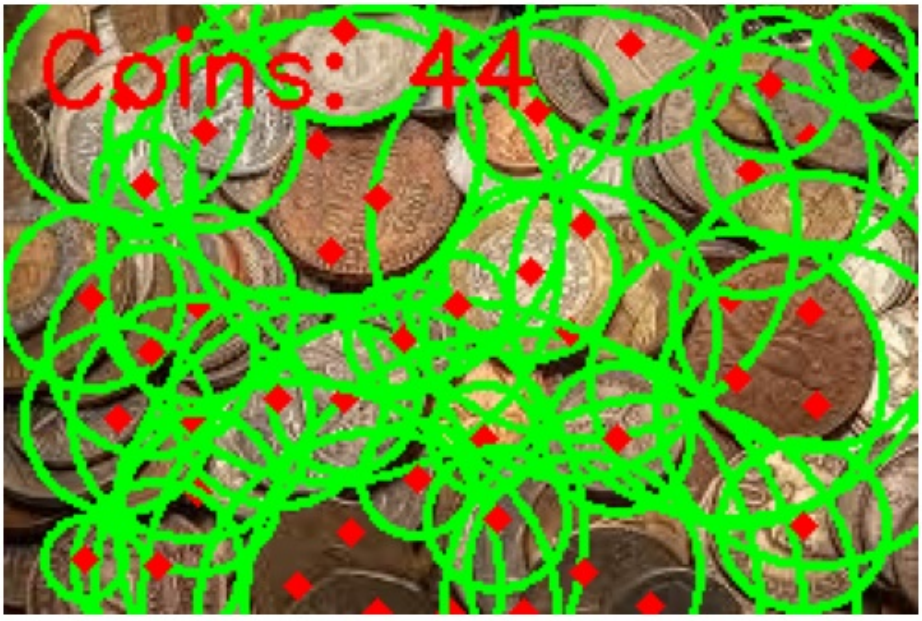
In [ ]:

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

image = cv2.imread("/content/sample_data/Lane_detection_2.jpg")

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

blurred = cv2.GaussianBlur(gray, (5, 5), 0)

edges = cv2.Canny(blurred, 50, 150)

mask = np.zeros_like(edges)
height, width = image.shape[:2]
polygon = np.array([[
    (0, height),
    (width, height),
    (width, height - 200),
    (0, height - 200)
]], dtype=np.int32)
cv2.fillPoly(mask, polygon, 255)
masked_edges = cv2.bitwise_and(edges, mask)

lines = cv2.HoughLinesP(masked_edges, 1, np.pi / 180, threshold=50, minLineLength=100, m
axLineGap=50)

line_image = np.copy(image)
for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(line_image, (x1, y1), (x2, y2), (0, 0, 255), 5)

line_image_rgb = cv2.cvtColor(line_image, cv2.COLOR_BGR2RGB)

plt.imshow(line_image_rgb)
plt.axis('off')
plt.show()
```

Coins Detection and Counting

In [ ]:

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

image = cv2.imread("/content/sample_data/coins.jpg")

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

blurred = cv2.GaussianBlur(gray, (9, 9), 2)

circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, dp=1, minDist=20, param1=50, par
am2=30, minRadius=0, maxRadius=0)

result_image = np.copy(image)

if circles is not None:
    circles = np.uint16(np.around(circles))
    count = len(circles[0, :])
    for i in circles[0, :]:
        cv2.circle(result_image, (i[0], i[1]), i[2], (0, 255, 0), 2)
        cv2.circle(result_image, (i[0], i[1]), 2, (0, 0, 255), 3)


    cv2.putText(result_image, f"Coins: {count}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0, 0, 255), 2)

result_image_rgb = cv2.cvtColor(result_image, cv2.COLOR_BGR2RGB)

plt.imshow(result_image_rgb)
plt.axis('off')
plt.show()
```



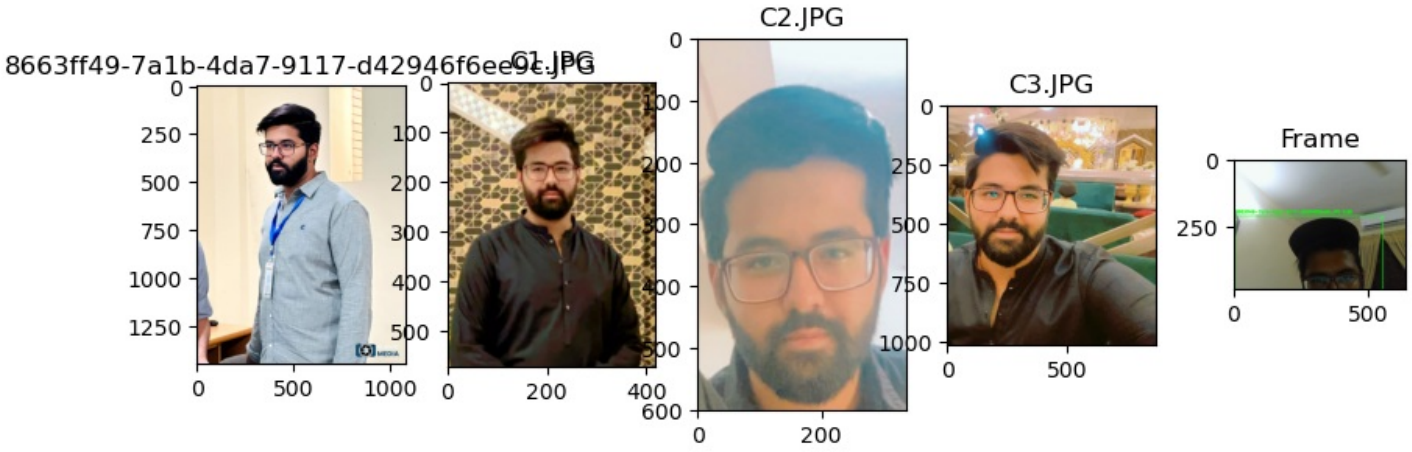
Smart Security System(run this code in jupyter)

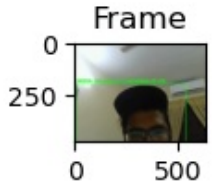
In [ ]:

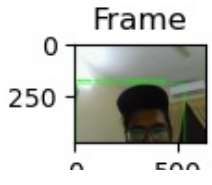
```
import cv2
import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Load YOLO
net = cv2.dnn.readNet("C:\\Users\\Dummy\\Downloads\\yolov3.weights", "C:\\Users\\Dummy\\
Downloads\\yolov3.cfg")

reference_images_dir = "C:\\Users\\Dummy\\Desktop\\MY_PICS"
labels = os.listdir(reference_images_dir)

reference_images = [mpimg.imread(os.path.join(reference_images_dir, label)) for label in
labels]

cap = cv2.VideoCapture(0)

plt.figure(figsize=(10, 5))
for i, ref_img in enumerate(reference_images):
    plt.subplot(1, len(reference_images), i + 1)
    plt.imshow(ref_img)
    plt.title(labels[i])

while True:
    ret, frame = cap.read()

    if not ret:
        break

    height, width, _ = frame.shape

    blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False
)

    net.setInput(blob)
    outs = net.forward(net.getUnconnectedOutLayersNames())

    class_ids = []
    confidences = []
    boxes = []

    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)

                x = int(center_x - w / 2)
                y = int(center_y - h / 2)

                class_ids.append(class_id)
                confidences.append(float(confidence))
                boxes.append([x, y, w, h])

    indices = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

    for i in range(len(boxes)):
        if i in indices:
            try:
                x, y, w, h = boxes[i]
                label = str(labels[class_ids[i]])
                confidence = confidences[i]

                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                cv2.putText(frame, f"{label} {confidence:.2f}", (x, y - 10), cv2.FONT_HE
RSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
```

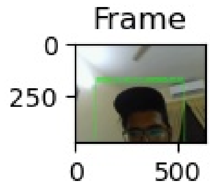
```
            except IndexError:
                pass

    plt.subplot(1, len(reference_images) + 1, len(reference_images) + 1)
    plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    plt.title("Frame")

    plt.show()

    if cv2.waitKey(1) & 0xFF == 27:
        break

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

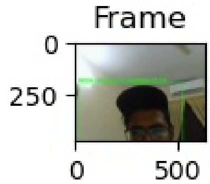
```
C:\Users\Dummy\AppData\Local\Temp\ipykernel_3016\452989538.py:74: MatplotlibDeprecationWa
rning: Auto-removal of overlapping axes is deprecated since 3.6 and will be removed two m
inor releases later; explicitly call ax.remove() as needed.
  plt.subplot(1, len(reference_images) + 1, len(reference_images) + 1)
```

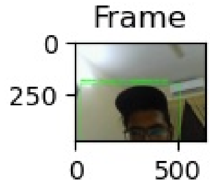


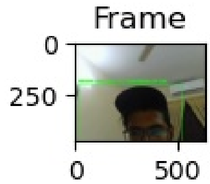


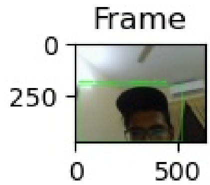


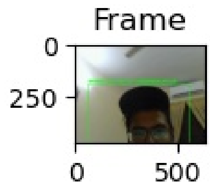


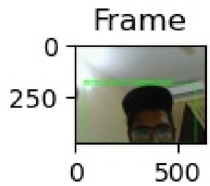












DETECT OBJECTS WITHIN A SPECIFIED RANGE

In [10]:

```
import cv2

def calculate_distance(object_size, focal_length, real_object_size):
    return (focal_length * real_object_size) / object_size

face_cascade = cv2.CascadeClassifier('C:\\Users\\Dummy\\Desktop\\haarcascade_frontalface
_alt.xml')

video_path = "C:\\Users\\Dummy\\Downloads\\invideo-ai-480 The Distance Dilemma_ 5cm vs 10
cm 2023-10-27.mp4"
```

```
cap = cv2.VideoCapture(video_path)

focal_length = 100
real_object_size_5cm = 5

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

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize
=(30, 30))

    for (x, y, w, h) in faces:
        center_x = x + w // 2
        center_y = y + h // 2

        object_size = w
        distance = calculate_distance(object_size, focal_length, real_object_size_5cm)

        detection_range_min = 5
        detection_range_max = 10

        if detection_range_min <= distance <= detection_range_max:
            cv2.putText(frame, f"Face Distance: {distance:.2f} cm", (x, y - 10), cv2.FON
T_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
        else:
            cv2.putText(frame, "Face out of range", (x, y - 10), cv2.FONT_HERSHEY_SIMPLE
X, 0.5, (0, 0, 255), 2)

        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    cv2.imshow("Object Detection", frame)

    if cv2.waitKey(1) & 0xFF == 27:
        break

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