In [16]:
```
#libraries
import numpy as np
import pandas as pd
import cv2
import pywt
import pywt.data
import matplotlib.pyplot as plt
import soundfile as sf
```

In [1]:
```
pip install PyWavelets
```

```
Requirement already satisfied: PyWavelets in c:\users\dell\anaconda3\lib\site-packages (1.3.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\dell\anaconda3\lib\site-packages (from PyWavele
ts) (1.24.3)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
pip install soundfile
```

```
Collecting soundfile
  Downloading soundfile-0.12.1-py2.py3-none-win_amd64.whl (1.0 MB)
Requirement already satisfied: cffi>=1.0 in c:\users\dell\anaconda3\lib\site-packages (from soundfile)
(1.15.0)
Requirement already satisfied: pycparser in c:\users\dell\anaconda3\lib\site-packages (from cffi>=1.0->s
oundfile) (2.21)
Installing collected packages: soundfile
Successfully installed soundfile-0.12.1
Note: you may need to restart the kernel to use updated packages.
```

EXTRA TASKS

TASK 1: Audio recording

In [4]:
```
audio, sample_rate = sf.read("vn.mp3")
```

In [5]:
```
wavelet = "db4"
level = 5
```

In [6]:
```
coeffs = pywt.wavedec(audio, wavelet, level = level)
```

In [15]:
```
threshold = 0.02
```

In [16]:
```
threshold_coeffs = [pywt.threshold(c, threshold, mode = "soft") for c in coeffs]
```

In [17]:
```
denoised_audio = pywt.waverec(threshold_coeffs, wavelet)
```

In [18]:
```
sf.write("cleanvn.wav", denoised_audio, sample_rate)
```

In [3]:
```
pip install sounddevice
```

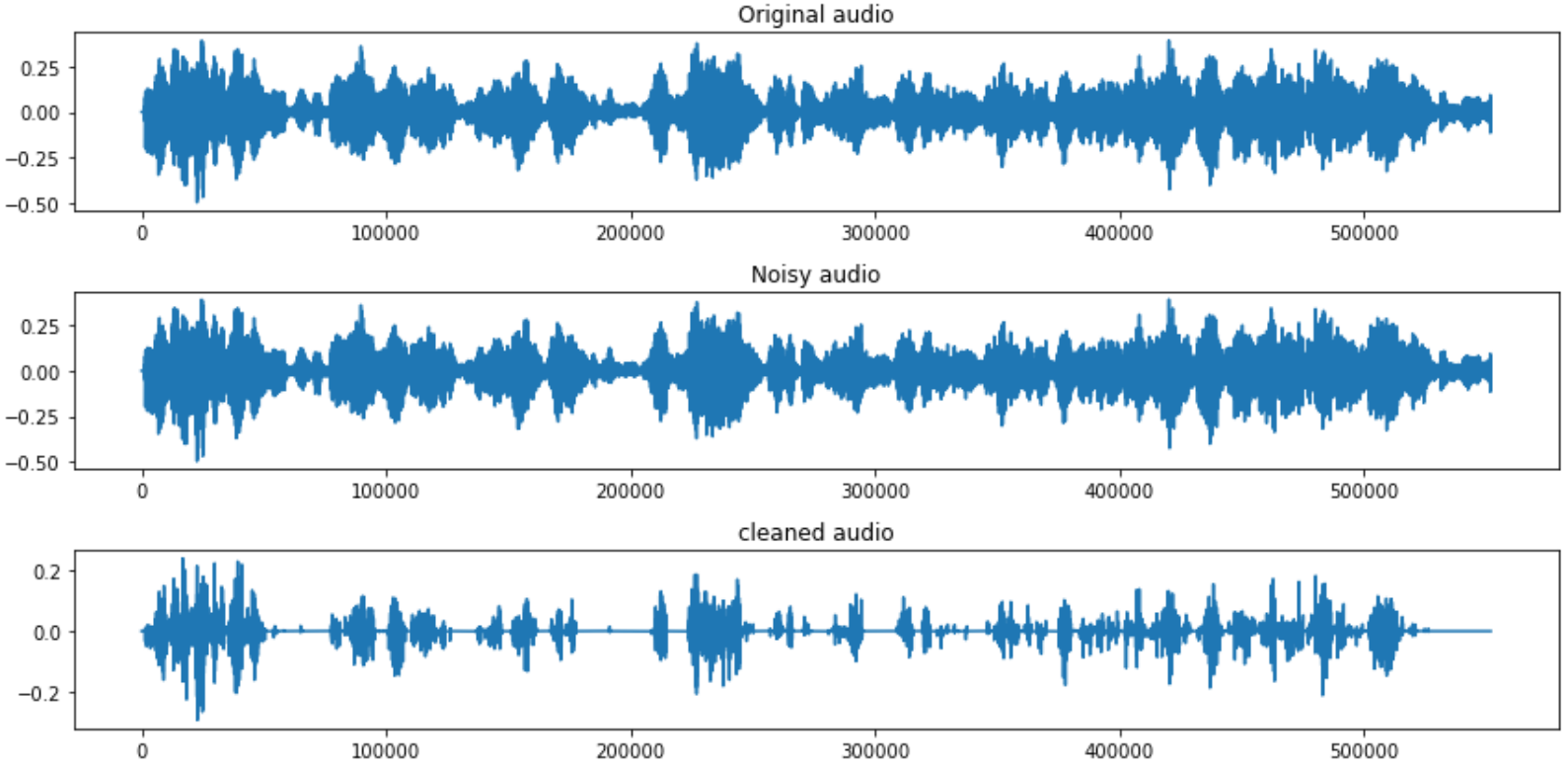
```
Collecting sounddevice
  Downloading sounddevice-0.4.6-py3-none-win_amd64.whl (199 kB)
Requirement already satisfied: CFFI>=1.0 in c:\users\dell\anaconda3\lib\site-packages (from sounddevice)
(1.15.0)
Requirement already satisfied: pycparser in c:\users\dell\anaconda3\lib\site-packages (from CFFI>=1.0->s
ounddevice) (2.21)
Installing collected packages: sounddevice
Successfully installed sounddevice-0.4.6
Note: you may need to restart the kernel to use updated packages.
```

In [20]:
```
import sounddevice as sd
```

In [21]:
```
sd.play(denoised_audio, sample_rate)
```

In [14]:

```
plt.figure(figsize=(12,6))
plt.subplot(3, 1, 1)
plt.title("Original audio")
plt.plot(audio)

plt.subplot(3, 1, 2)
plt.title("Noisy audio")
plt.plot(audio)

plt.subplot(3, 1, 3)
plt.title("cleaned audio")
plt.plot(denoised_audio)

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



TASK 2: Computer Screen (On/off) Detection-Using Reference Images

In [136]:
```
reference_on_image = cv2.imread('on.jpeg', 0)
reference_off_image = cv2.imread('off.jpeg', 0)
```

In [137]:
```
sift = cv2.SIFT_create()
```

In [138]:
```
kp1, des1 = sift.detectAndCompute(reference_on_image, None)
kp2, des2 = sift.detectAndCompute(reference_off_image, None)
```

In [139]:
```
bf = cv2.BFMatcher()
```

In [140]:
```
current_screen = cv2.imread('screenon.jpeg', 0)
```

In [141]:
```
kp3, des3 = sift.detectAndCompute(current_screen, None)
kp4, des4 = sift.detectAndCompute(current_screen2, None)
```

In [142]:
```
# Match descriptors between reference and current images
matches1 = bf.knnMatch(des1, des3, k=2)
matches2 = bf.knnMatch(des2, des3, k=2)
```

In [143]:
```
good_matches1 = []
good_matches2 = []
```

In [144]:
```
for m, n in matches1:
    if m.distance < 0.75 * n.distance:
        good_matches1.append(m)

for m, n in matches2:
    if m.distance < 0.75 * n.distance:
        good_matches2.append(m)
```

In [145]:
```
threshold = 20
```

In [146]:

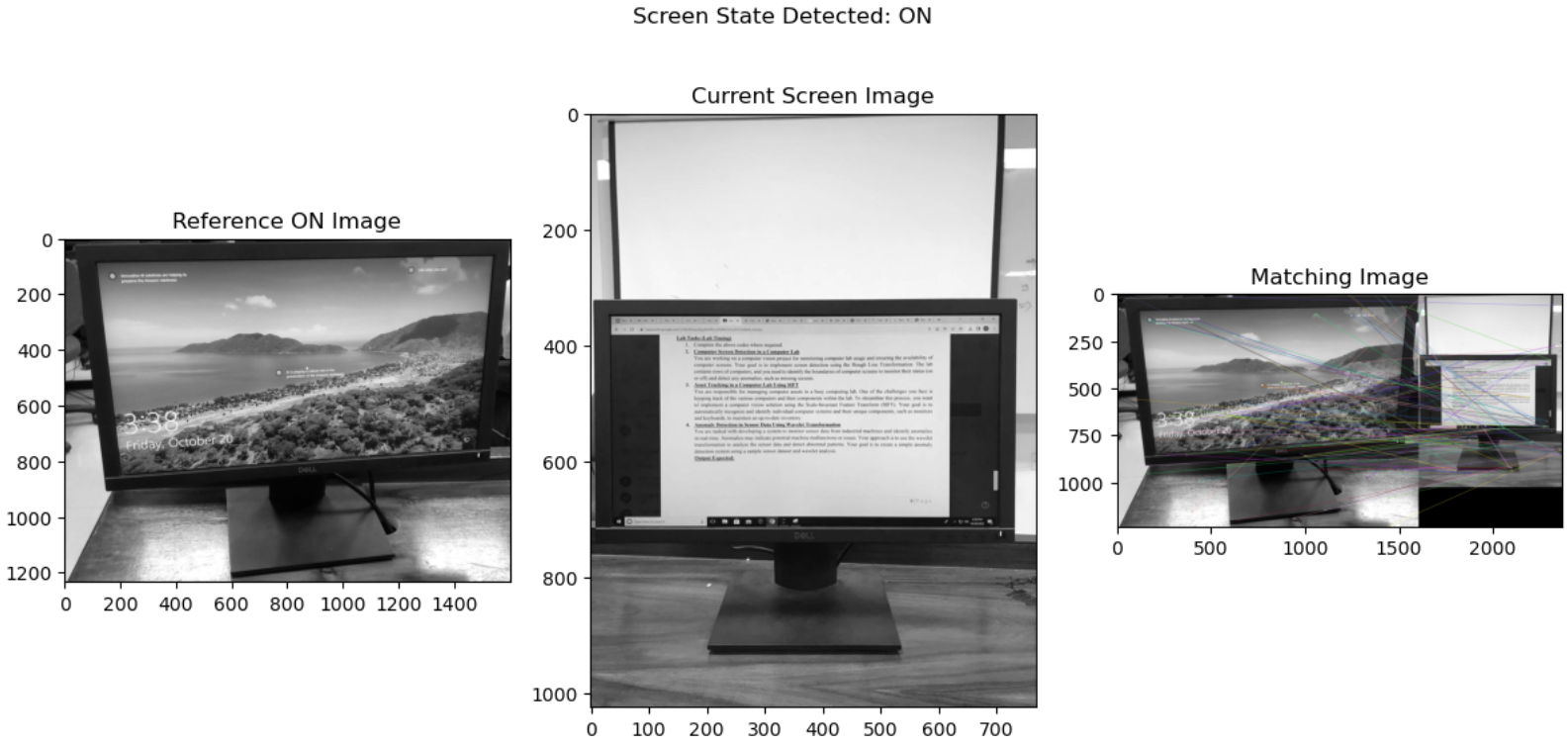
```
if len(good_matches1) > threshold:
    screen_state = "ON"
else:
    if len(good_matches2) > threshold:
        screen_state = "OFF"
    else:
        screen_state = "Cannot be determined"
```

In [147]:

```
matching_image = cv2.drawMatches(reference_on_image, kp1, current_screen, kp3, good_matches1, None, 
```

In [148]:

```
plt.figure(figsize=(12, 6), dpi=100)
plt.subplot(131)
plt.imshow(reference_on_image, cmap='gray')
plt.title('Reference ON Image')
plt.subplot(132)
plt.imshow(current_screen, cmap='gray')
plt.title('Current Screen Image')
plt.subplot(133)
plt.imshow(matching_image)
plt.title('Matching Image')
plt.suptitle(f"Screen State Detected: {screen_state}")
plt.tight_layout()
plt.show()
```



TASK 3: Detecting Your Self in video (Using one of your Reference Image)

In [71]:
```
reference_image = cv2.imread('em.png', cv2.IMREAD_GRAYSCALE)
video_path = 'emaan.mp4'
```

In [72]:
```
sift = cv2.SIFT_create()
```

In [73]:
```
keypoints_ref, descriptors_ref = sift.detectAndCompute(reference_image, None)
```

In [74]:
```
bf = cv2.BFMatcher()
```

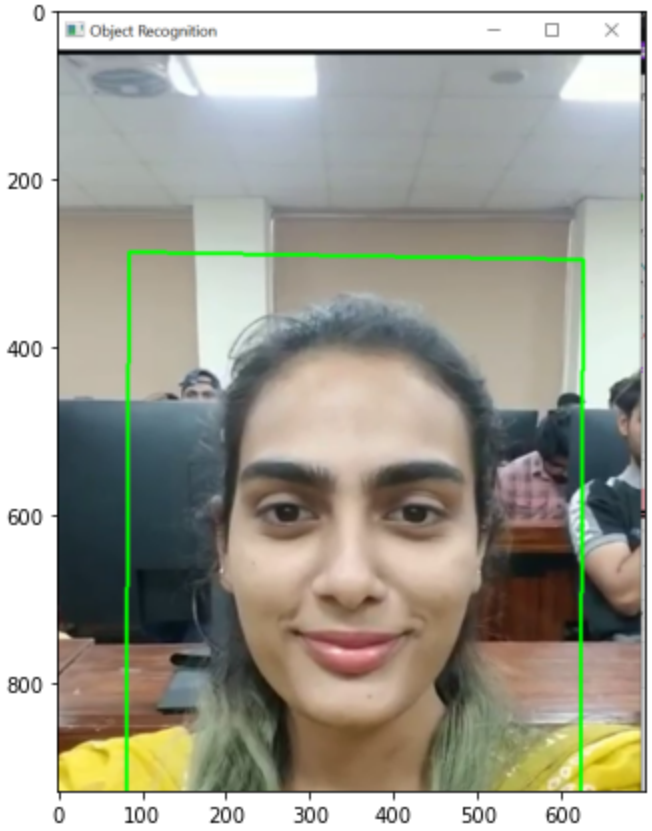
In [75]:

```
cap = cv2.VideoCapture(video_path)
while cap.isOpened():
    ret, frame = cap.read()

    if not ret:
        break

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

    keypoints_frame, descriptors_frame = sift.detectAndCompute(frame_gray, None)

    matches = bf.knnMatch(descriptors_ref, descriptors_frame, k=2)

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

    src_pts = np.float32([keypoints_ref[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
    dst_pts = np.float32([keypoints_frame[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)

    if len(good_matches) > 10:
        # Calculate the homography matrix
        homography_matrix, _ = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

        # Get the dimensions of the reference image
        h, w = reference_image.shape

        # Define the corners of the reference image
        corners = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)

        transformed_corners = cv2.perspectiveTransform(corners, homography_matrix)

        # Draw a bounding box around the recognized object
        frame = cv2.polylines(frame, [np.int32(transformed_corners)], True, (0, 255, 0), 2)

    # Display the frame with the bounding box
    cv2.imshow('Object Recognition', frame)

    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

cap.release()
```

```
44 cv2.destroyAllWindows()
```

In [76]:

```
1 img = cv2.imread('em3.png', cv2.IMREAD_COLOR)
2 plt.figure(figsize=(10,6))
3 plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
4 plt.tight_layout()
5 plt.show()
```



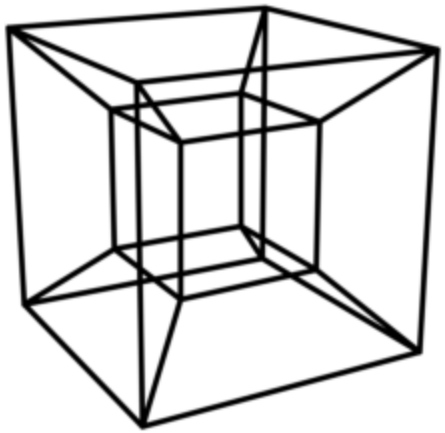
Lab Tasks (Lab Timing)

1. Complete the above codes where required.

In [33]:
```
1 img = cv2.imread('hello.png', cv2.IMREAD_GRAYSCALE)
```

In [36]:
```
1 plt.imshow(img,cmap = 'gray')
2 plt.axis('off')
```

Out[36]: (-0.5, 625.5, 625.5, -0.5)



In [37]:
```
1 #sobel
2 gradient_x = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
3 gradient_y = cv2.Sobel(img, cv2.CV_64F, 0 , 1, ksize=3)
4 gradient_magnitude = np.sqrt(gradient_x**2 + gradient_y**2)
5 gradient_direction = np.arctan2(gradient_y, gradient_x)
```

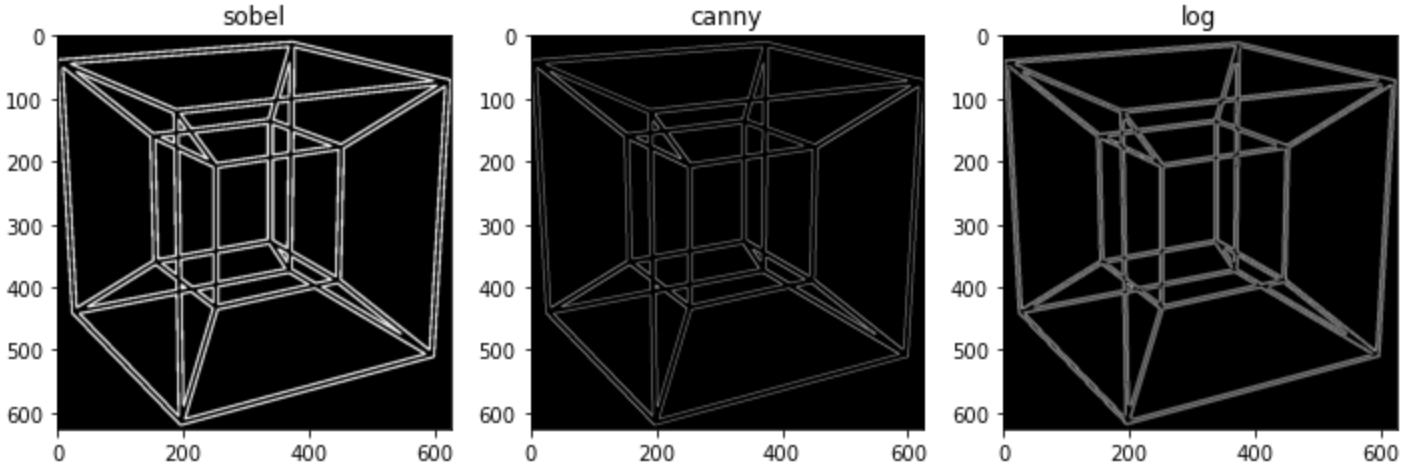
In [38]:
```
1 #canny
2 threshold = 80
3 edges = (gradient_magnitude >threshold).astype(np.uint8) * 255
4
5 canny = cv2.Canny(img, threshold1=100, threshold2=200)
```

In [39]:

```
#log
image_smoothed = cv2.GaussianBlur(img, (5, 5), 0)
laplacian = cv2.Laplacian(image_smoothed, cv2.CV_8U)
log = cv2.Canny(laplacian, threshold1=25, threshold2=50)
```

In [40]:

```
plt.figure(figsize=(10,6))
plt.subplot(1,3,1)
plt.title('sobel')
plt.imshow(edges, cmap='gray')

plt.subplot(1,3,2)
plt.title('canny')
plt.imshow(canny, cmap='gray')

plt.subplot(1,3,3)
plt.title('log')
plt.imshow(log, cmap='gray')

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



2. Computer Screen Detection in a Computer Lab

You are working on a computer vision project for monitoring computer lab usage and ensuring the availability of computer screens. Your goal is to implement screen detection using the Hough Line Transformation. The lab contains rows of computers, and you need to identify the boundaries of computer screens to monitor their status (on or off) and detect any anomalies, such as missing screens.

In [17]:
```
img = cv2.imread('screen.jpg', cv2.IMREAD_GRAYSCALE)
```

In [18]:
```
blurred = cv2.GaussianBlur(img, (9, 9), 0)
```

In [19]:
```
edges = cv2.Canny(blurred, 50, 150)
```

In [20]:
```
bgr = cv2.cvtColor(blurred, cv2.COLOR_GRAY2BGR)
```

In [21]:
```
lines = cv2.HoughLinesP(edges, 1, np.pi/180, threshold=150, minLineLength = 20, maxLineGap = 7)
```

In [22]:
```
if lines is not None:
    for line in lines:
        x1, y1, x2, y2 = line[0]
        angle = np.arctan2(y2 - y1, x2 - x1)
        if -0.4 < angle < 0.4:  # vertical
            # Changed the color to red (rgb format)
            cv2.line(bgr, (x1, y1), (x2, y2), (255, 0, 0), 3)
```

In [23]:

```
plt.figure(figsize=(10,6), dpi = 200)
plt.subplot(2,1,1)
plt.title('original image')
plt.imshow(img, cmap='gray')
plt.axis('off')

plt.subplot(2,1,2)
plt.title('Lines Detected in Red')
plt.imshow(bgr, cmap='gray')
plt.axis('off')

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

original image

Lines Detected in Red

3. Asset Tracking in a Computer Lab Using SIFT

You are responsible for managing computer assets in a busy computing lab. One of the challenges you face is keeping track of the various computers and their components within the lab. To streamline this process, you want to implement a computer vision solution using the Scale-Invariant Feature Transform (SIFT). Your goal is to automatically recognize and identify individual computer systems and their unique components, such as monitors and keyboards, to maintain an up-to-date inventory

In [ ]:
```
#5 images and 1 reference
#do refrence keliay humain do colors use karne honge code main diff keliay
```

In [3]:
```
reference1 = cv2.imread('on.jpeg', cv2.IMREAD_GRAYSCALE)
```

In [4]:
```
sift = cv2.SIFT_create()
```

In [5]:
```
keyref, desref = sift.detectAndCompute(reference1, None)
```

In [7]:
```
recognizedObj = []
```

In [8]:
```
testimgs = ['1.jpeg','2.jpeg','3.jpeg','4.jpeg','5.jpeg']
```

In [19]:

```
for testImgPath in testimgs:
    testimg = cv2.imread(testImgPath, cv2.IMREAD_GRAYSCALE)
    keytest, destest = sift.detectAndCompute(testimg, None)
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(desref, destest, k=2)
    goodMatches = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            goodMatches.append(m)
    if len(goodMatches) > 10:  # Adjust this threshold as needed
        print("Object recognized in test image!")
    else:
        print("Object not recognized.")
```

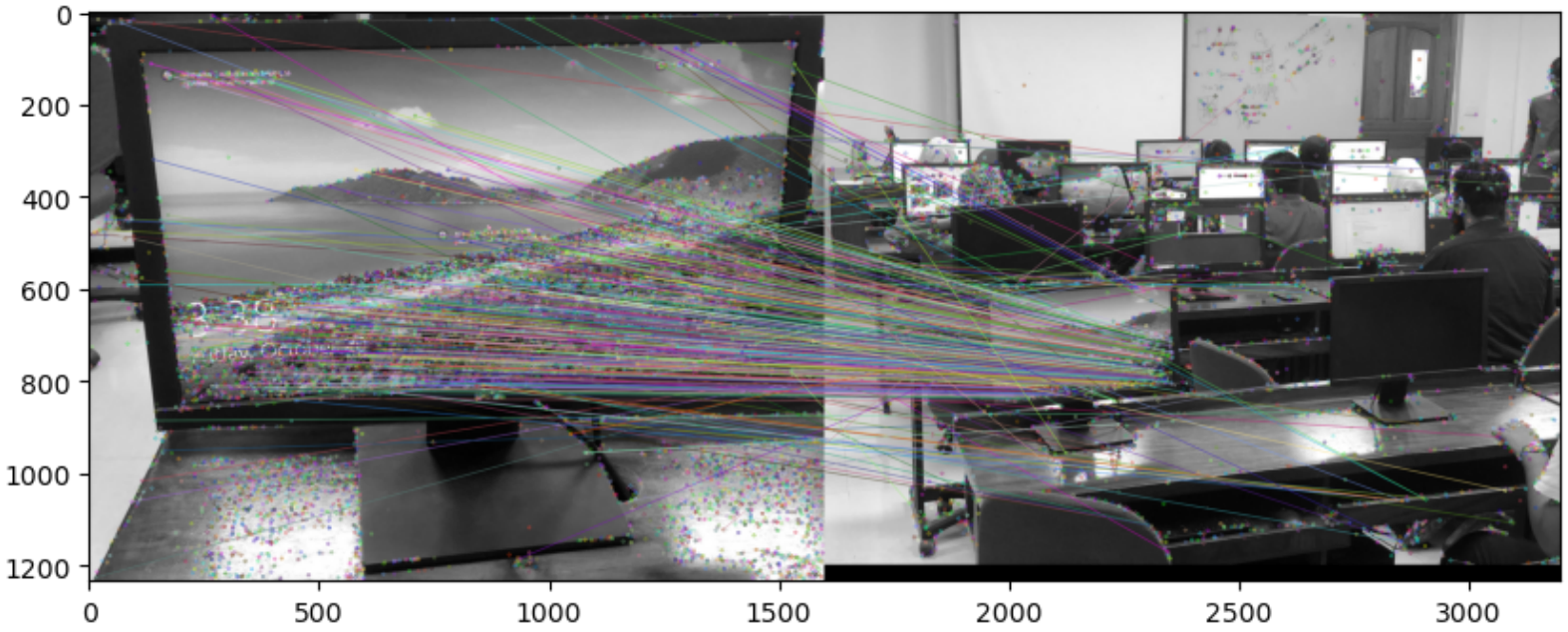
```
Object recognized in test image!
Object recognized in test image!
Object recognized in test image!
Object recognized in test image!
Object recognized in test image!
```

In [16]:

```
result = cv2.drawMatches(reference1, keyref, testimg, keytest, goodMatches, None)
```

In [21]:
```
1 plt.figure(figsize = (10,6), dpi=100)
2 plt.imshow(result)
```

Out[21]: <matplotlib.image.AxesImage at 0x20952c7d5b0>



4. Anomaly Detection in Sensor Data Using Wavelet Transformation

You are tasked with developing a system to monitor sensor data from industrial machines and identify anomalies in real-time. Anomalies may indicate potential machine malfunctions or issues. Your approach is to use the wavelet transformation to analyze the sensor data and detect abnormal patterns. Your goal is to create a simple anomaly detection system using a sample sensor dataset and wavelet analysis. Output Expected:

In [2]:
```
1 sensorData = np.random.rand(600)
```

In [3]:
```
1 wavelet = 'sym4'
```

In [4]:
```
coeffs = pywt.wavedec(sensorData, wavelet) # wavelet transform
```

In [5]:
```
print(coeffs[0])
```

```
[6.40282481 6.63326217 4.96702103 4.40356103 3.72200853 4.21154636
 4.32341814 4.08675034 4.07026694 3.88345162 3.62576064 4.27474182
 4.52079152 4.53076262 4.52613704 4.52170768]
```

In [6]:
```
thresh = 0.43
```

In [7]:
```
denoisedCoeffs = [pywt.threshold(c, thresh) if i > 0 else c for i, c in enumerate(coeffs)]
```

In [8]:
```
# denoisedCoeffs = []
# for i, c in enumerate(coeffs):
#     if i > 0:
#         denoisedCoeffs.append(pywt.threshold(c, thresh))
#     else:
#         denoisedCoeffs.append(c)
```

In [9]:
```
denoisedSignal = pywt.waverec(denoisedCoeffs, wavelet) #inverse wavelet transform
```

In [10]:
```
residuals = sensorData - denoisedSignal
```

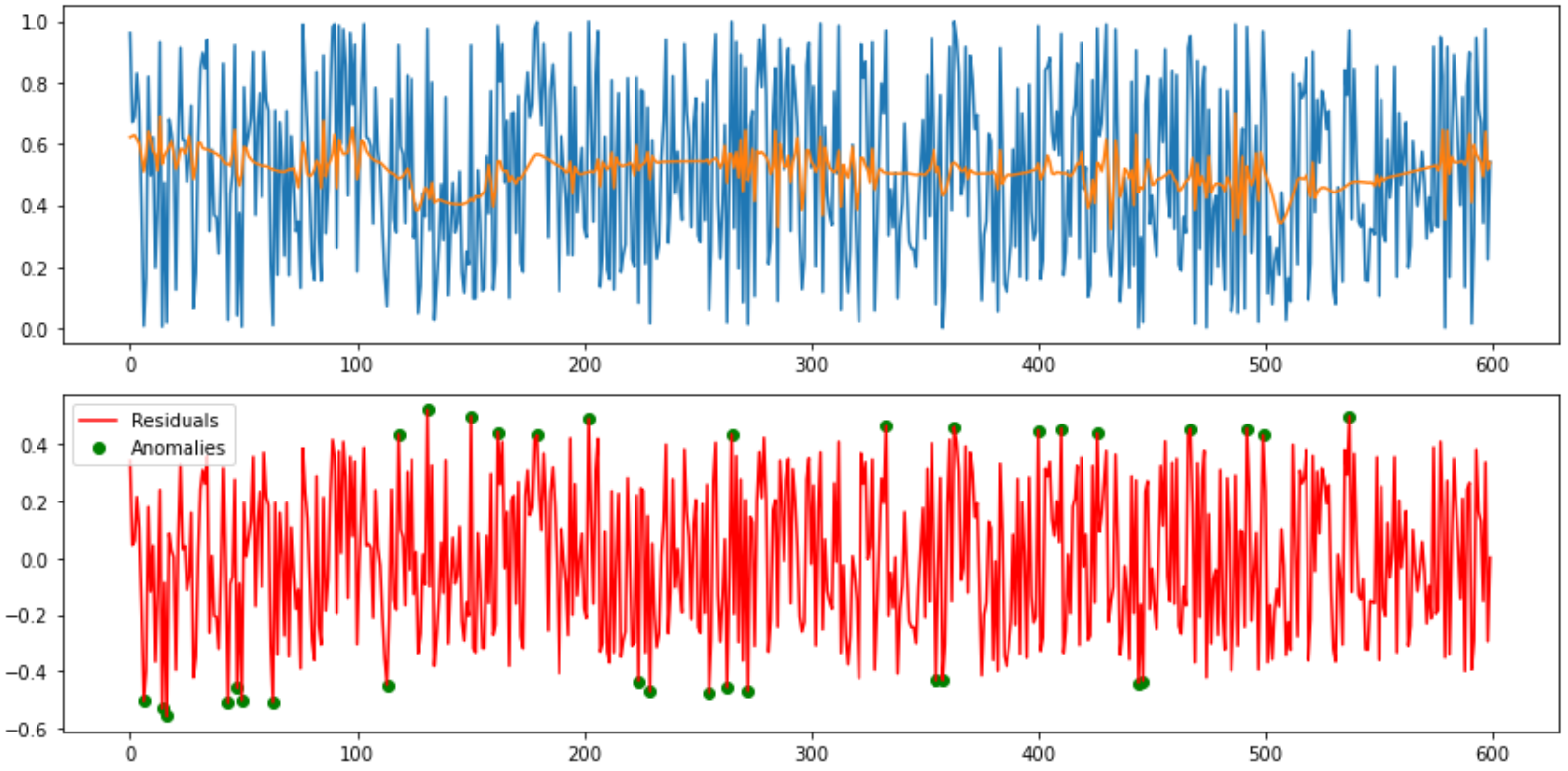
In [12]:
```
anomalies = np.where(np.abs(residuals) > thresh)[0]
```

In [15]:
```
print("Detected Anomalies:", anomalies)
```

```
Detected Anomalies: [  6  14  16  43  47  49  63 113 118 131 150 162 179 202 224 229 255 263
 265 272 333 355 358 363 400 410 426 444 446 467 492 499 537]
```

In [14]:

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

plt.subplot(2, 1, 1)
plt.plot(sensorData, label='Sensor Data')
plt.plot(denoisedSignal, label='Denoised Signal')

plt.subplot(2, 1, 2)
plt.plot(residuals, 'r', label='Residuals')
plt.scatter(anomalies, residuals[anomalies], color='g', marker='o', label='Anomalies')

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



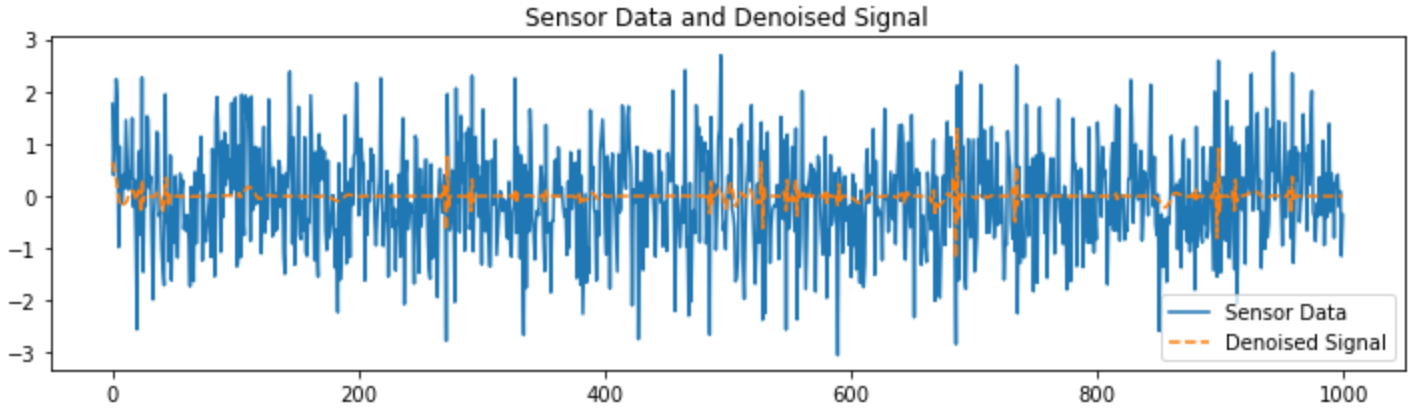
In [17]:

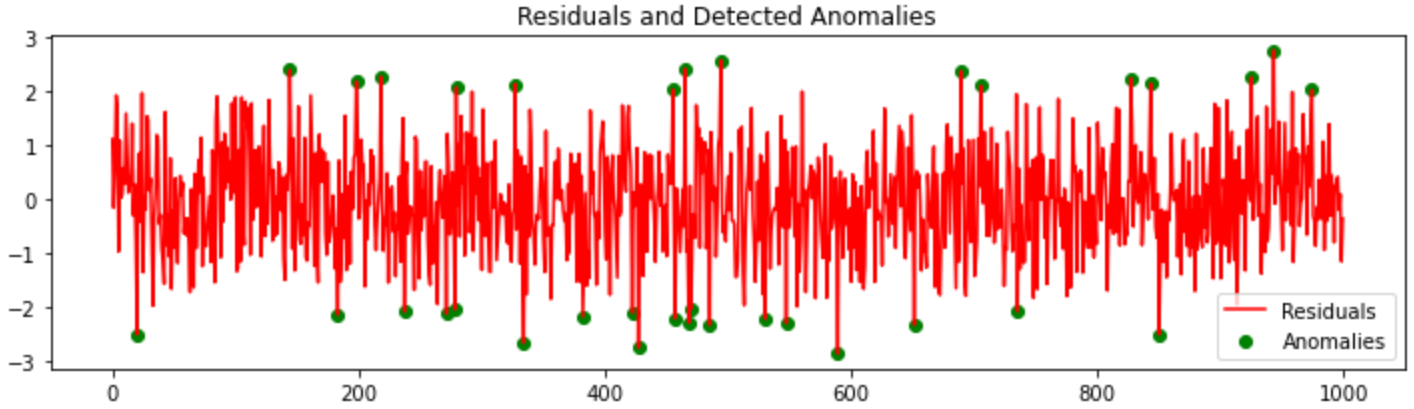
```
#below is code sir provided
```

In [16]:

```
# Generate a sample sensor dataset
np.random.seed(0)
sensor_data = np.random.normal(0, 1, 1000)

# Choose the wavelet family and decomposition level
wavelet = "db4"
level = 3

# Perform the wavelet transformation
coeffs = pywt.wavedec(sensor_data, wavelet, level=level)

# Set a threshold for anomaly detection
threshold = 2.0

# Apply thresholding to the wavelet coefficients
thresholded_coeffs = [pywt.threshold(c, threshold, mode="soft") for c in coeffs]

# Reconstruct the denoised signal
denoised_signal = pywt.waverec(thresholded_coeffs, wavelet)

# Calculate the residuals (differences between original and denoised signal)
residuals = sensor_data - denoised_signal

# Detect anomalies based on the residuals and threshold
anomalies = np.where(np.abs(residuals) > threshold)[0]

# Plot the sensor data, denoised signal, and detected anomalies
plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)
plt.plot(sensor_data, label="Sensor Data")
plt.plot(denoised_signal, label="Denoised Signal", linestyle="--")
plt.legend()
plt.title("Sensor Data and Denoised Signal")

plt.subplot(2, 1, 2)
plt.plot(residuals, label="Residuals", color="red")
plt.scatter(anomalies, residuals[anomalies], color="green", label="Anomalies")
plt.legend()
plt.title("Residuals and Detected Anomalies")

plt.tight_layout()
plt.show()

```

```
# Output detected anomaly indices

print("Detected Anomalies:", anomalies)
```






```
Detected Anomalies: [ 20 144 183 198 218 237 271 278 279 327 334 382 422 427 455 457 465 468
 470 485 494 530 547 589 651 689 705 735 827 843 850 925 943 974]
```

Lab Tasks

1. Object Recognition (Using Video):

You are working on an image processing project, and your task is to implement object recognition using the Scale- Invariant Feature Transform (SIFT) algorithm. You have a reference image of the object you want to recognize and a set of test images. Your goal is to identify and locate the object in each test image, even if it appears at different scales, orientations, or under partial occlusion. You will use SIFT features for object recognition and draw bounding boxes around the recognized objects in the test images.

In [62]:
```
reference_image = cv2.imread('papa.jpeg', cv2.IMREAD_GRAYSCALE)
video_path = 'papa.mp4'
```

In [63]:
```
sift = cv2.SIFT_create()
```

In [64]:
```
keypoints_ref, descriptors_ref = sift.detectAndCompute(reference_image, None)
```

In [65]:
```
bf = cv2.BFMatcher()
```

In [68]:

```
cap = cv2.VideoCapture(video_path)
while cap.isOpened():
    ret, frame = cap.read()

    if not ret:
        break

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

    keypoints_frame, descriptors_frame = sift.detectAndCompute(frame_gray, None)

    matches = bf.knnMatch(descriptors_ref, descriptors_frame, k=2)

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

    src_pts = np.float32([keypoints_ref[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
    dst_pts = np.float32([keypoints_frame[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)

    if len(good_matches) > 10:
        # Calculate the homography matrix
        homography_matrix, _ = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

        # Get the dimensions of the reference image
        h, w = reference_image.shape

        # Define the corners of the reference image
        corners = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)

        transformed_corners = cv2.perspectiveTransform(corners, homography_matrix)

        # Draw a bounding box around the recognized object
        frame = cv2.polylines(frame, [np.int32(transformed_corners)], True, (0, 255, 0), 2)

    # Display the frame with the bounding box
    cv2.imshow('Object Recognition', frame)

    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

cap.release()
```

```
cv2.destroyAllWindows()
```

In [69]:

```
img = cv2.imread('dad.png', cv2.IMREAD_COLOR)
plt.figure(figsize=(10,6))
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.tight_layout()
plt.show()
```

2. Your Panoramic Image

You are working on a project where you need to create a panoramic image by stitching multiple overlapping images together. The task involves taking a set of images captured from a camera while panning, and then using the SIFT algorithm to find key points and match them between adjacent images. Your goal is to automatically align and stitch the images to create a single panoramic image.

In [61]:

```
image1 = cv2.imread('p1.png')
image2 = cv2.imread('p2.png')
image3 = cv2.imread('p3.png')

# Convert images to grayscale
gray1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)
gray3 = cv2.cvtColor(image3, cv2.COLOR_BGR2GRAY)

# Initialize the SIFT detector
sift = cv2.SIFT_create()

# Find the key points and descriptors in all three images
keypoints1, descriptors1 = sift.detectAndCompute(gray1, None)
keypoints2, descriptors2 = sift.detectAndCompute(gray2, None)
keypoints3, descriptors3 = sift.detectAndCompute(gray3, None)

# Create a BFMatcher (Brute-Force Matcher) and perform matching for all pairs
bf = cv2.BFMatcher()

matches12 = bf.knnMatch(descriptors1, descriptors2, k=2)
matches23 = bf.knnMatch(descriptors2, descriptors3, k=2)

# Apply Lowe's ratio test to filter good matches
good_matches12 = []
for m, n in matches12:
    if m.distance < 0.75 * n.distance:
        good_matches12.append(m)

good_matches23 = []
for m, n in matches23:
    if m.distance < 0.75 * n.distance:
        good_matches23.append(m)

# Check if we have enough good matches to compute the homography
if len(good_matches12) < 4 or len(good_matches23) < 4:
    print("Not enough good matches to compute homography.")
else:
    # Create lists of corresponding points
    src_pts12 = np.float32([keypoints1[m.queryIdx].pt for m in good_matches12]).reshape(-1, 1, 2)
    dst_pts12 = np.float32([keypoints2[m.trainIdx].pt for m in good_matches12]).reshape(-1, 1, 2)

    src_pts23 = np.float32([keypoints2[m.queryIdx].pt for m in good_matches23]).reshape(-1, 1, 2)
```

```
    dst_pts23 = np.float32([keypoints3[m.trainIdx].pt for m in good_matches23]).reshape(-1, 1, 2)

    # Calculate the homography matrices using RANSAC
    homography_matrix12, mask12 = cv2.findHomography(src_pts12, dst_pts12, cv2.RANSAC, 5.0)
    homography_matrix23, mask23 = cv2.findHomography(src_pts23, dst_pts23, cv2.RANSAC, 5.0)

    # Warp the images
    height, width = gray2.shape
    warped_image1 = cv2.warpPerspective(image1, homography_matrix12, (width, height))
    warped_image3 = cv2.warpPerspective(image3, homography_matrix23, (width, height))

    # Blend the warped images and the second image
    result = cv2.addWeighted(warped_image1, 1, image2, 1, 0)
    result = cv2.addWeighted(result, 0.7, warped_image3, 0.5, 0)

    # Display the result
    plt.figure(figsize=(18,14))
    plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
    plt.show()
```


0
100
200
300
400
500
600
0
100
200
300
400
500
600
700

3. Lane detection using the Hough Line Transformation

You are working on an autonomous vehicle project, and one of the critical tasks is to detect lane markings on the road to ensure safe and accurate navigation. Your goal is to implement lane detection using the Hough Line Transformation. You have a set of images captured from the vehicle's camera. Your task is to identify and draw the detected lane lines on these images to assist in lane-keeping and control algorithms.

In [148]:
```
image = cv2.imread('lane.png')
```

In [149]:
```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

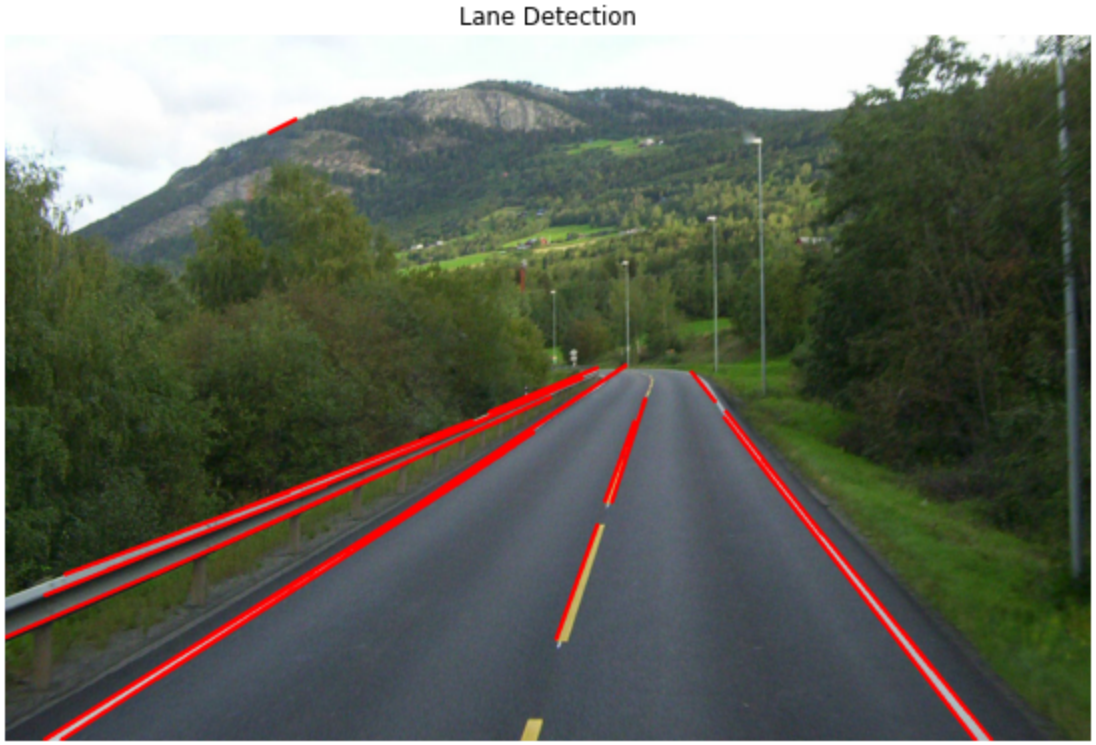
In [150]:
```
gray_blurred = cv2.GaussianBlur(gray, (11, 11), 0)
```

In [151]:
```
edges = cv2.Canny(gray_blurred, 50, 150, apertureSize=3)
```

In [152]:
```
lines = cv2.HoughLinesP(edges, 1, np.pi / 180, threshold=100, minLineLength=10, maxLineGap=18)
```

In [153]:
```
for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 3)
```

In [154]:
```
plt.figure(figsize = (10,8), dpi = 70)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Lane Detection")
plt.axis('off')
plt.show()
```



4. Coins Detection and Counting

You are working on a computer vision project that involves the identification and counting of coins in images. Your task is to implement coin detection using the Hough Circle Transformation. You have a set of images containing coins of various sizes and positions. Your goal is to automatically identify and count the coins in these images.

In [155]:
```
image = cv2.imread('coins.png')
```

In [156]:
```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

In [157]:
```
gray_blurred = cv2.GaussianBlur(gray, (9, 9), 2)
```

In [158]:
```
circles = cv2.HoughCircles(
    gray_blurred,
    cv2.HOUGH_GRADIENT,
    dp=1,
    minDist=100,
    param1=70,
    param2=40,
    minRadius=5,
    maxRadius=100
)
```

In [160]:

```
if circles is not None:
    circles = np.uint16(np.around(circles))
    detected_coins = circles[0, :]

    for circle in detected_coins:
        center = (circle[0], circle[1])
        radius = circle[2]
        cv2.circle(image, center, radius, (0, 0, 255), 3)
        coin_count = len(detected_coins)

    plt.figure(figsize = (10,8), dpi = 70)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(f"Detected Coins: {coin_count}")
    plt.axis('off')
    plt.show()
else:
    print("No coins detected.")
```

5. Smart Security System

You are working on a security system for an organization, and one of the key tasks is to identify unauthorized objects that are placed near sensitive areas. To achieve this, you need to implement boundary detection using computer vision. Your goal is to detect the boundaries of objects that are within a predefined security zone in a real-time video stream. When an unauthorized object enters this zone, an alarm should be triggered.

In [ ]:
```
1 #this detects unauthorized object from the video.
```

In [13]:

```
reference_image = cv2.imread('unauthorizedObj.jpeg', cv2.IMREAD_GRAYSCALE)
video_path = 'cctv.mp4'
```

In [14]:

```
cap = cv2.VideoCapture(video_path)

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

    if not ret:
        break

    frame = cv2.resize(frame, (reference_image.shape[1], reference_image.shape[0]))

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

    difference = cv2.absdiff(reference_image, frame_gray)

    _, thresholded = cv2.threshold(difference, 30, 255, cv2.THRESH_BINARY)

    # Find contours in the thresholded image
    contours, _ = cv2.findContours(thresholded, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Iterate over the detected contours
    for contour in contours:
        # Calculate the area of the contour
        area = cv2.contourArea(contour)

        # Define a threshold for minimum contour area to filter small objects
        min_contour_area = 1000

        if area > min_contour_area:
            # Draw a bounding rectangle around the detected object
            x, y, w, h = cv2.boundingRect(contour)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)  # Red color

    # Display the frame with detected boundaries
    cv2.imshow('Security System', frame)

    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

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

```
41
```

In [ ]:

```
1
```