**Chapter 2: Advanced Image Transformations**

* **Project Title**: Geometric Image Manipulation with Interactive Controls
* **Objective**: Apply scaling, rotation, and affine transformations with real-time user adjustments using trackbars.
* **Detailed Code Explanation**:
  + **Initialization**: Load an image using cv2.imread in color format and retrieve its dimensions for transformation calculations.
  + **Trackbar Setup**: Create a window with cv2.namedWindow and add trackbars using cv2.createTrackbar for scale (0-200%), rotation angle (0-360°), and shear factor (0-1) to allow dynamic user input.
  + **Scaling**: Define a callback function to update the scale factor from the trackbar, resizing the image with cv2.resize using bilinear interpolation for smooth scaling.
  + **Rotation**: Compute a rotation matrix with cv2.getRotationMatrix2D based on the trackbar angle, applying it with cv2.warpAffine to rotate around the image center, adjusting output dimensions to avoid cropping.
  + **Affine Transformation**: Introduce a shear effect by defining source and destination points dynamically based on the shear factor, calculating the affine matrix with cv2.getAffineTransform, and applying it with cv2.warpAffine.
  + **Real-Time Display**: Continuously update the transformed image in a loop using cv2.imshow, refreshing based on trackbar inputs until the user presses 'q' to exit.
  + **Error Handling**: Check for successful image loading and handle invalid trackbar values to prevent crashes.

import cv2

import numpy as np

def nothing(x):

pass

image = cv2.imread('image.jpg')

if image is None:

raise ValueError("Image not found!")

rows, cols = image.shape[:2]

cv2.namedWindow('Transformations')

cv2.createTrackbar('Scale (%)', 'Transformations', 100, 200, nothing)

cv2.createTrackbar('Rotation (°)', 'Transformations', 0, 360, nothing)

cv2.createTrackbar('Shear', 'Transformations', 0, 100, nothing)

while True:

scale = cv2.getTrackbarPos('Scale (%)', 'Transformations') / 100.0

angle = cv2.getTrackbarPos('Rotation (°)', 'Transformations')

shear = cv2.getTrackbarPos('Shear', 'Transformations') / 100.0

# Scaling

scaled = cv2.resize(image, None, fx=scale, fy=scale, interpolation=cv2.INTER\_LINEAR)

# Rotation

M\_rot = cv2.getRotationMatrix2D((cols/2, rows/2), angle, 1)

rotated = cv2.warpAffine(image, M\_rot, (cols, rows))

# Affine with shear

pts1 = np.float32([[50,50], [200,50], [50,200]])

pts2 = np.float32([[50+shear\*50,50], [200+shear\*50,50], [50,200]])

M\_affine = cv2.getAffineTransform(pts1, pts2)

affine = cv2.warpAffine(image, M\_affine, (cols, rows))

cv2.imshow('Transformations', affine)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cv2.destroyAllWindows()

**Chapter 3: Thresholding & Segmentation**

* **Project Title**: Adaptive Object Segmentation with Noise Reduction
* **Objective**: Segment objects using multiple thresholding techniques, incorporating noise reduction for robustness.
* **Detailed Code Explanation**:
  + **Image Loading**: Read the image in grayscale using cv2.imread with cv2.IMREAD\_GRAYSCALE to simplify processing.
  + **Noise Reduction**: Apply a Gaussian blur with cv2.GaussianBlur (kernel size 5x5) to reduce noise, improving thresholding accuracy.
  + **Binary Thresholding**: Use cv2.threshold with a user-defined threshold (e.g., 127) via a trackbar, allowing real-time adjustment to separate objects from the background.
  + **Adaptive Thresholding**: Implement cv2.adaptiveThreshold with the Gaussian method, dynamically adjusting the threshold based on local pixel neighborhoods (block size 11, constant 2).
  + **Morphological Operations**: Apply cv2.morphologyEx with a closing operation (5x5 kernel) to fill small gaps in the segmented regions, enhancing object integrity.
  + **Visualization**: Create a window with trackbars using cv2.namedWindow and cv2.createTrackbar, displaying the original, binary, and adaptive thresholded images side-by-side with cv2.hconcat.
  + **Loop Control**: Update the display in a loop, exiting with 'q', ensuring smooth interaction and real-time feedback.

import cv2

import numpy as np

def nothing(x):

pass

image = cv2.imread('image.jpg', cv2.IMREAD\_GRAYSCALE)

if image is None:

raise ValueError("Image not found!")

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

cv2.namedWindow('Thresholding')

cv2.createTrackbar('Binary Thresh', 'Thresholding', 127, 255, nothing)

while True:

thresh\_val = cv2.getTrackbarPos('Binary Thresh', 'Thresholding')

\_, binary = cv2.threshold(blurred, thresh\_val, 255, cv2.THRESH\_BINARY)

adaptive = cv2.adaptiveThreshold(blurred, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY, 11, 2)

kernel = np.ones((5,5), np.uint8)

binary = cv2.morphologyEx(binary, cv2.MORPH\_CLOSE, kernel)

adaptive = cv2.morphologyEx(adaptive, cv2.MORPH\_CLOSE, kernel)

combined = cv2.hconcat([image, binary, adaptive])

cv2.imshow('Thresholding', combined)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cv2.destroyAllWindows()

**Chapter 4: Edge Detection**

* **Project Title**: Interactive Edge Detection with Parameter Tuning
* **Objective**: Detect edges using the Canny algorithm with adjustable thresholds and noise reduction.
* **Detailed Code Explanation**:
  + **Image Preprocessing**: Load the image in grayscale and apply cv2.GaussianBlur (5x5 kernel) to reduce noise, ensuring cleaner edge detection.
  + **Trackbar Integration**: Set up a window with cv2.namedWindow and add trackbars for low threshold (0-255) and high threshold (0-255) using cv2.createTrackbar.
  + **Canny Edge Detection**: Retrieve trackbar values and apply cv2.Canny to detect edges, where the low threshold filters weak edges and the high threshold confirms strong edges.
  + **Edge Refinement**: Dilate the edge map with cv2.dilate (3x3 kernel) to thicken edges for better visibility, controlled by an optional iteration parameter.
  + **Display Logic**: Concatenate the original image and edge map horizontally with cv2.hconcat for side-by-side comparison, updating in real-time within a loop.
  + **User Interaction**: Allow the user to adjust thresholds dynamically, exiting with 'q' via cv2.waitKey.
  + **Validation**: Ensure trackbar values are valid (low < high) to prevent algorithm errors.

import cv2

import numpy as np

def nothing(x):

pass

image = cv2.imread('image.jpg', cv2.IMREAD\_GRAYSCALE)

if image is None:

raise ValueError("Image not found!")

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

cv2.namedWindow('Canny Edges')

cv2.createTrackbar('Low Thresh', 'Canny Edges', 100, 255, nothing)

cv2.createTrackbar('High Thresh', 'Canny Edges', 200, 255, nothing)

while True:

low = cv2.getTrackbarPos('Low Thresh', 'Canny Edges')

high = cv2.getTrackbarPos('High Thresh', 'Canny Edges')

if low >= high:

high = low + 1

edges = cv2.Canny(blurred, low, high)

edges = cv2.dilate(edges, np.ones((3,3), np.uint8), iterations=1)

combined = cv2.hconcat([image, edges])

cv2.imshow('Canny Edges', combined)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cv2.destroyAllWindows()

**Chapter 5: Contour Detection**

* **Project Title**: Contour-Based Object Analysis with Properties
* **Objective**: Detect contours, calculate properties (area, perimeter), and visualize them with annotations.
* **Detailed Code Explanation**:
  + **Image Preparation**: Load a color image, convert it to grayscale with cv2.cvtColor, and apply binary thresholding with cv2.threshold to isolate objects.
  + **Contour Detection**: Use cv2.findContours with cv2.RETR\_EXTERNAL to detect outer contours and cv2.CHAIN\_APPROX\_SIMPLE to reduce memory usage by simplifying contour points.
  + **Property Calculation**: For each contour, compute the area with cv2.contourArea and perimeter with cv2.arcLength, filtering out small contours (area < 100) to reduce noise.
  + **Annotation**: Draw contours on the image with cv2.drawContours (green, thickness 2), and use cv2.putText to label each contour with its area and perimeter at the centroid (calculated via moments).
  + **Moments Calculation**: Use cv2.moments to find the centroid (cx, cy) of each contour for accurate text placement.
  + **Display**: Show the annotated image with cv2.imshow, looping until 'q' is pressed to exit.
  + **Error Handling**: Validate image loading and contour detection results to ensure robustness.

import cv2

import numpy as np

image = cv2.imread('image.jpg')

if image is None:

raise ValueError("Image not found!")

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

\_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY)

contours, \_ = cv2.findContours(binary, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

for contour in contours:

area = cv2.contourArea(contour)

if area < 100:

continue

perimeter = cv2.arcLength(contour, True)

M = cv2.moments(contour)

cx = int(M['m10'] / M['m00']) if M['m00'] != 0 else 0

cy = int(M['m01'] / M['m00']) if M['m00'] != 0 else 0

cv2.drawContours(image, [contour], -1, (0, 255, 0), 2)

cv2.putText(image, f"A:{int(area)} P:{int(perimeter)}", (cx, cy), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 0, 0), 1)

cv2.imshow('Contours with Properties', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Chapter 6: Shape Detection**

* **Project Title**: Advanced Shape Recognition with Polygon Approximation
* **Objective**: Detect and classify shapes (triangle, rectangle, circle) with detailed polygon approximation and labeling.
* **Detailed Code Explanation**:
  + **Preprocessing**: Load the image, convert to grayscale, and apply binary thresholding to create a binary mask of shapes.
  + **Contour Extraction**: Detect contours with cv2.findContours using cv2.RETR\_EXTERNAL to focus on outer boundaries.
  + **Polygon Approximation**: For each contour, use cv2.approxPolyDP with a dynamic epsilon (1% of perimeter) to approximate it to a polygon, reducing noise and improving shape detection.
  + **Shape Classification**: Analyze the number of vertices: 3 for triangle, 4 for rectangle; for >8, fit an ellipse with cv2.fitEllipse and check aspect ratio to confirm circles.
  + **Centroid Calculation**: Compute the centroid using moments (cv2.moments) for accurate label placement.
  + **Annotation**: Label each shape with cv2.putText and draw the approximated contour with cv2.drawContours (red, thickness 2).
  + **Display Loop**: Show the annotated image, exiting with 'q', ensuring all shapes are correctly identified and visualized.

import cv2

import numpy as np

image = cv2.imread('shapes.jpg')

if image is None:

raise ValueError("Image not found!")

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

\_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY)

contours, \_ = cv2.findContours(binary, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

for contour in contours:

epsilon = 0.01 \* cv2.arcLength(contour, True)

approx = cv2.approxPolyDP(contour, epsilon, True)

M = cv2.moments(contour)

cx = int(M['m10'] / M['m00']) if M['m00'] != 0 else 0

cy = int(M['m01'] / M['m00']) if M['m00'] != 0 else 0

if len(approx) == 3:

shape = "Triangle"

elif len(approx) == 4:

shape = "Rectangle"

elif len(approx) > 8:

(x, y), (MA, ma), angle = cv2.fitEllipse(contour)

aspect\_ratio = MA / ma

shape = "Circle" if 0.9 <= aspect\_ratio <= 1.1 else "Polygon"

else:

shape = "Polygon"

cv2.drawContours(image, [approx], -1, (0, 0, 255), 2)

cv2.putText(image, shape, (cx, cy), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 1)

cv2.imshow('Shape Detection', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Chapter 7: Histogram Analysis**

* **Project Title**: Interactive Histogram Equalization and Analysis
* **Objective**: Analyze and enhance image contrast using histograms with real-time equalization control.
* **Detailed Code Explanation**:
  + **Image Loading**: Read a color image with cv2.imread and split it into BGR channels using cv2.split.
  + **Histogram Calculation**: Compute histograms for each channel with cv2.calcHist, specifying 256 bins and range [0, 256].
  + **Trackbar Setup**: Create a window with a trackbar (cv2.createTrackbar) to toggle histogram equalization on/off (0-1).
  + **Equalization**: When enabled, apply cv2.equalizeHist to each channel individually, then merge them back with cv2.merge to enhance contrast.
  + **Visualization**: Plot histograms using Matplotlib within the loop, creating subplots for each channel (B, G, R) and updating based on equalization state.
  + **Image Display**: Show the original or equalized image with cv2.imshow, refreshing in real-time as the trackbar changes.
  + **Loop Management**: Exit with 'q', ensuring proper cleanup with cv2.destroyAllWindows.

import cv2

import numpy as np

import matplotlib.pyplot as plt

def nothing(x):

pass

image = cv2.imread('image.jpg')

if image is None:

raise ValueError("Image not found!")

b, g, r = cv2.split(image)

cv2.namedWindow('Image')

cv2.createTrackbar('Equalize', 'Image', 0, 1, nothing)

plt.ion()

while True:

equalize = cv2.getTrackbarPos('Equalize', 'Image')

if equalize == 1:

b\_eq = cv2.equalizeHist(b)

g\_eq = cv2.equalizeHist(g)

r\_eq = cv2.equalizeHist(r)

display\_img = cv2.merge((b\_eq, g\_eq, r\_eq))

hist\_b = cv2.calcHist([b\_eq], [0], None, [256], [0, 256])

hist\_g = cv2.calcHist([g\_eq], [0], None, [256], [0, 256])

hist\_r = cv2.calcHist([r\_eq], [0], None, [256], [0, 256])

else:

display\_img = image

hist\_b = cv2.calcHist([b], [0], None, [256], [0, 256])

hist\_g = cv2.calcHist([g], [0], None, [256], [0, 256])

hist\_r = cv2.calcHist([r], [0], None, [256], [0, 256])

cv2.imshow('Image', display\_img)

plt.clf()

plt.subplot(131), plt.plot(hist\_b, 'b'), plt.title('Blue')

plt.subplot(132), plt.plot(hist\_g, 'g'), plt.title('Green')

plt.subplot(133), plt.plot(hist\_r, 'r'), plt.title('Red')

plt.pause(0.01)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

plt.close()

cv2.destroyAllWindows()

**Chapter 8: Motion Detection**

* **Project Title: Real-Time Motion Detection with Sensitivity Control**
* **Objective: Detect motion in a video stream with adjustable sensitivity and bounding box visualization.**
* **Detailed Code Explanation:**
  + **Background Subtraction: Initialize cv2.createBackgroundSubtractorMOG2 with a history of 500 frames and variance threshold adjustable via a trackbar.**
  + **Video Capture: Open a video file or webcam with cv2.VideoCapture, checking for successful initialization.**
  + **Trackbar Setup: Add a trackbar for variance threshold (0-100) to control motion sensitivity, affecting the foreground mask generation.**
  + **Frame Processing: For each frame, apply the subtractor with subtractor.apply to get the foreground mask, then reduce noise with cv2.morphologyEx (opening, 3x3 kernel).**
  + **Contour Detection: Find contours in the mask with cv2.findContours, filtering by area (>500) to ignore small noise.**
  + **Bounding Boxes: Draw green rectangles around detected motion regions using cv2.boundingRect and cv2.rectangle, with thickness 2.**
  + **Real-Time Display: Show the frame with annotations, exiting with 'q', releasing resources properly.**

import cv2

import numpy as np

def nothing(x):

pass

subtractor = cv2.createBackgroundSubtractorMOG2(history=500)

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

if not cap.isOpened():

raise ValueError("Video not found!")

cv2.namedWindow('Motion Detection')

cv2.createTrackbar('Variance', 'Motion Detection', 25, 100, nothing)

while True:

ret, frame = cap.read()

if not ret:

break

var\_thresh = cv2.getTrackbarPos('Variance', 'Motion Detection')

subtractor.setVarThreshold(var\_thresh)

mask = subtractor.apply(frame)

mask = cv2.morphologyEx(mask, cv2.MORPH\_OPEN, np.ones((3,3), np.uint8))

contours, \_ = cv2.findContours(mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

for contour in contours:

if cv2.contourArea(contour) > 500:

x, y, w, h = cv2.boundingRect(contour)

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

cv2.imshow('Motion Detection', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

**Chapter 9: Face Detection**

* **Project Title**: Multi-Face Detection with Scale and Neighbor Tuning
* **Objective**: Detect multiple faces in an image with adjustable detection parameters for precision.
* **Detailed Code Explanation**:
  + **Classifier Loading**: Load the Haar cascade classifier with cv2.CascadeClassifier using the frontal face XML file, verifying successful loading.
  + **Image Processing**: Load the image, convert to grayscale with cv2.cvtColor for compatibility with the classifier.
  + **Trackbar Controls**: Set up trackbars for scale factor (1.05-1.5) and min neighbors (1-10) to tune detection sensitivity and reduce false positives.
  + **Face Detection**: Use detectMultiScale with trackbar values to detect faces, specifying a minimum size (e.g., 30x30) to filter small detections.
  + **Visualization**: Draw blue rectangles around detected faces with cv2.rectangle, adding a count label with cv2.putText above each face.
  + **Dynamic Update**: Update the display in a loop based on trackbar adjustments, exiting with 'q'.
  + **Robustness**: Handle cases where no faces are detected gracefully, avoiding empty loops.

**Objective:**

Detect multiple faces in an image using Haar Cascade classifiers with interactive trackbars to adjust detection parameters like scale factor and minNeighbors for enhanced precision and fewer false positives.

import cv2

def nothing(x):

pass

# Load the Haar Cascade classifier

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

if face\_cascade.empty():

raise IOError("Failed to load face cascade classifier!")

# Load and preprocess the image

image = cv2.imread('group\_photo.jpg')

if image is None:

raise ValueError("Image not found!")

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Create a named window and trackbars

cv2.namedWindow('Face Detection')

cv2.createTrackbar('Scale x100', 'Face Detection', 105, 150, nothing) # Scale from 1.05 to 1.5

cv2.createTrackbar('Min Neighbors', 'Face Detection', 5, 10, nothing)

while True:

# Get trackbar values

scale\_val = cv2.getTrackbarPos('Scale x100', 'Face Detection') / 100.0

min\_neighbors = cv2.getTrackbarPos('Min Neighbors', 'Face Detection')

# Detect faces

faces = face\_cascade.detectMultiScale(

gray,

scaleFactor=scale\_val,

minNeighbors=min\_neighbors,

minSize=(30, 30)

)

# Copy image for drawing

output = image.copy()

face\_count = 0

for (x, y, w, h) in faces:

face\_count += 1

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

cv2.putText(output, f"Face #{face\_count}", (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 0, 0), 2)

if face\_count == 0:

cv2.putText(output, "No faces detected", (20, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, (0, 0, 255), 2)

# Show result

cv2.imshow('Face Detection', output)

# Exit on 'q'

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cv2.destroyAllWindows()

**Chapter 10: Object Tracking**

* **Project Title**: Color-Based Object Tracking with ROI Selection
* **Objective**: Track an object in a video based on a user-selected color region of interest (ROI).
* **Detailed Code Explanation**:
  + **Video Initialization**: Open a video with cv2.VideoCapture, reading the first frame to select the ROI.
  + **ROI Selection**: Use cv2.selectROI to let the user manually select an object, extracting its HSV histogram with cv2.calcHist.
  + **Histogram Backprojection**: For each frame, convert to HSV with cv2.cvtColor, calculate backprojection with cv2.calcBackProject using the ROI histogram, and threshold it to create a mask.
  + **Noise Reduction**: Apply cv2.morphologyEx (opening, 5x5 kernel) to clean the mask, reducing false positives.
  + **Contour Tracking**: Find the largest contour in the mask with cv2.findContours, drawing a bounding box with cv2.rectangle if the area exceeds a threshold (e.g., 500).
  + **Real-Time Loop**: Display the frame with the tracked object, exiting with 'q', releasing resources cleanly.
  + **Error Checking**: Validate video loading and ROI selection to ensure smooth execution.

import cv2

import numpy as np

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

if not cap.isOpened():

raise ValueError("Video not found!")

ret, frame = cap.read()

if not ret:

raise ValueError("Cannot read video!")

roi\_box = cv2.selectROI("Select ROI", frame, fromCenter=False)

roi = frame[int(roi\_box[1]):int(roi\_box[1]+roi\_box[3]), int(roi\_box[0]):int(roi\_box[0]+roi\_box[2])]

hsv\_roi = cv2.cvtColor(roi, cv2.COLOR\_BGR2HSV)

roi\_hist = cv2.calcHist([hsv\_roi], [0, 1], None, [180, 256], [0, 180, 0, 256])

cv2.normalize(roi\_hist, roi\_hist, 0, 255, cv2.NORM\_MINMAX)

while True:

ret, frame = cap.read()

if not ret:

break

hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

dst = cv2.calcBackProject([hsv], [0, 1], roi\_hist, [0, 180, 0, 256], 1)

\_, mask = cv2.threshold(dst, 50, 255, cv2.THRESH\_BINARY)

mask = cv2.morphologyEx(mask, cv2.MORPH\_OPEN, np.ones((5,5), np.uint8))

contours, \_ = cv2.findContours(mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

if contours:

c = max(contours, key=cv2.contourArea)

if cv2.contourArea(c) > 500:

x, y, w, h = cv2.boundingRect(c)

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

cv2.imshow('Object Tracking', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

**Chapter 11: Applying Filters for Artistic Effects**

* **Project Title**: Custom Artistic Filters with Parameter Tuning
* **Objective**: Create cartoon and sketch effects with adjustable filter parameters for customization.
* **Detailed Code Explanation**:
  + **Image Loading**: Load a color image with cv2.imread, ensuring it’s valid for processing.
  + **Trackbar Setup**: Create trackbars for bilateral filter diameter (5-15), sigma color (10-150), and edge threshold (1-10) to control smoothing and edge detection.
  + **Smoothing**: Apply cv2.bilateralFilter with trackbar-adjusted parameters to smooth the image while preserving edges, critical for cartoon effects.
  + **Edge Detection**: Convert to grayscale, apply cv2.adaptiveThreshold with a dynamic constant from the trackbar to detect edges, and invert for the mask.
  + **Cartoon Effect**: Combine the smoothed image and edge mask with cv2.bitwise\_and, creating a cartoon-like output.
  + **Sketch Effect**: Use cv2.pencilSketch with default parameters as an alternative, selectable via a toggle trackbar (0-1).
  + **Real-Time Display**: Show the processed image in a loop, updating based on trackbar inputs, exiting with 'q'.

import cv2

import numpy as np

def nothing(x):

pass

image = cv2.imread('image.jpg')

if image is None:

raise ValueError("Image not found!")

cv2.namedWindow('Artistic Filters')

cv2.createTrackbar('Diameter', 'Artistic Filters', 9, 15, nothing)

cv2.createTrackbar('Sigma Color', 'Artistic Filters', 75, 150, nothing)

cv2.createTrackbar('Edge Thresh', 'Artistic Filters', 2, 10, nothing)

cv2.createTrackbar('Mode (0=Cartoon, 1=Sketch)', 'Artistic Filters', 0, 1, nothing)

while True:

d = cv2.getTrackbarPos('Diameter', 'Artistic Filters') | 1 # Ensure odd

sigma = cv2.getTrackbarPos('Sigma Color', 'Artistic Filters')

edge\_thresh = cv2.getTrackbarPos('Edge Thresh', 'Artistic Filters')

mode = cv2.getTrackbarPos('Mode (0=Cartoon, 1=Sketch)', 'Artistic Filters')

if mode == 0:

blurred = cv2.bilateralFilter(image, d, sigma, sigma)

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

edges = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE\_THRESH\_MEAN\_C, cv2.THRESH\_BINARY, 9, edge\_thresh)

output = cv2.bitwise\_and(blurred, blurred, mask=edges)

else:

output, \_ = cv2.pencilSketch(image, sigma\_s=60, sigma\_r=0.07, shade\_factor=0.05)

cv2.imshow('Artistic Filters', output)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cv2.destroyAllWindows()

**Image Processing with OpenCV: 10 Chapters with Mini-Projects**

**Slide 1: Advanced Image Transformations**

* **Learning Outcome**: Learn to apply scaling, rotation, and affine transformations using OpenCV for precise geometric image manipulation.
* **Programming Task**: Write a Python script to resize, rotate, and perform affine transformations on images and display them for comparison.
* **Why It Matters**: Essential for data augmentation, image alignment, and pre-processing in computer vision.
* **Real-World Example**: Correcting misaligned medical scans or adjusting satellite imagery orientation.
* **Skill Gained**: Mastery of OpenCV transformation functions and visualization techniques.
* **Mini-Project Overview**: Apply scaling, rotation, and affine transformations to an image and display results side by side.
* **Code Snippet**:

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread('image.jpg')

resized = cv2.resize(image, (200, 200))

rows, cols = image.shape[:2]

M = cv2.getRotationMatrix2D((cols/2, rows/2), 45, 1)

rotated = cv2.warpAffine(image, M, (cols, rows))

pts1 = np.float32([[50,50],[200,50],[50,200]])

pts2 = np.float32([[10,100],[200,50],[100,250]])

M = cv2.getAffineTransform(pts1, pts2)

affine = cv2.warpAffine(image, M, (cols, rows))

plt.figure(figsize=(10, 5))

plt.subplot(141), plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)), plt.title('Original')

plt.subplot(142), plt.imshow(cv2.cvtColor(resized, cv2.COLOR\_BGR2RGB)), plt.title('Resized')

plt.subplot(143), plt.imshow(cv2.cvtColor(rotated, cv2.COLOR\_BGR2RGB)), plt.title('Rotated')

plt.subplot(144), plt.imshow(cv2.cvtColor(affine, cv2.COLOR\_BGR2RGB)), plt.title('Affine')

plt.show()

* **Code Explanation**: Loads an image, applies resizing, rotation (45 degrees), and an affine transformation (warping based on point mapping), then displays all versions using Matplotlib.

**Slide 2: Thresholding & Segmentation**

* **Learning Outcome**: Segment objects from backgrounds using binary and adaptive thresholding.
* **Programming Task**: Apply binary and adaptive thresholding to extract objects using OpenCV.
* **Why It Matters**: Fundamental for isolating regions in medical, industrial, and security imaging.
* **Real-World Example**: Separating text from scanned documents or detecting tumors in scans.
* **Skill Gained**: Image preprocessing and region isolation.
* **Mini-Project Overview**: Compare binary and adaptive thresholding on an image.
* **Code Snippet**:

import cv2

import matplotlib.pyplot as plt

image = cv2.imread('image.jpg', cv2.IMREAD\_GRAYSCALE)

\_, binary = cv2.threshold(image, 127, 255, cv2.THRESH\_BINARY)

adaptive = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY, 11, 2)

plt.figure(figsize=(10, 5))

plt.subplot(131), plt.imshow(image, cmap='gray'), plt.title('Original')

plt.subplot(132), plt.imshow(binary, cmap='gray'), plt.title('Binary')

plt.subplot(133), plt.imshow(adaptive, cmap='gray'), plt.title('Adaptive')

plt.show()

* **Code Explanation**: Loads a grayscale image, applies binary thresholding (fixed value) and adaptive thresholding (local regions), then visualizes results for comparison.

**Slide 3: Edge Detection**

* **Learning Outcome**: Detect object boundaries using Sobel and Canny algorithms.
* **Programming Task**: Apply Canny edge detection and visualize edge maps.
* **Why It Matters**: Crucial for object recognition, computer vision, and robotics.
* **Real-World Example**: Lane detection in self-driving cars or barcode recognition.
* **Skill Gained**: Gradient-based image processing.
* **Mini-Project Overview**: Detect edges in an image using Canny.
* **Code Snippet**:

import cv2

import matplotlib.pyplot as plt

image = cv2.imread('image.jpg', cv2.IMREAD\_GRAYSCALE)

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

plt.figure(figsize=(10, 5))

plt.subplot(121), plt.imshow(image, cmap='gray'), plt.title('Original')

plt.subplot(122), plt.imshow(edges, cmap='gray'), plt.title('Canny Edges')

plt.show()

* **Code Explanation**: Loads a grayscale image, applies Canny edge detection with specified thresholds, and displays the original and edge map.

**Slide 4: Contour Detection**

* **Learning Outcome**: Detect and analyze shapes in images.
* **Programming Task**: Find and draw contours around objects.
* **Why It Matters**: Enables shape recognition, object detection, and tracking.
* **Real-World Example**: Detecting cells in microscopic images or coins in financial apps.
* **Skill Gained**: Morphological feature extraction.
* **Mini-Project Overview**: Detect and draw contours on an image.
* **Code Snippet**:

import cv2

import matplotlib.pyplot as plt

image = cv2.imread('image.jpg')

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

\_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY)

contours, \_ = cv2.findContours(binary, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

cv2.drawContours(image, contours, -1, (0, 255, 0), 3)

plt.figure(figsize=(5, 5))

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.title('Contours')

plt.show()

* **Code Explanation**: Converts an image to binary, finds external contours, draws them in green, and displays the result.

**Slide 5: Shape Detection**

* **Learning Outcome**: Recognize geometric shapes using contours.
* **Programming Task**: Detect and label shapes like circles, rectangles, and triangles.
* **Why It Matters**: Useful for classifying objects by geometry in computer vision.
* **Real-World Example**: Sorting objects in manufacturing pipelines.
* **Skill Gained**: Feature classification and pattern matching.
* **Mini-Project Overview**: Detect and label shapes in an image.
* **Code Snippet**:

import cv2

import numpy as np

image = cv2.imread('shapes.jpg')

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

\_, binary = cv2.threshold(gray, 127, 255, cv2.THRESH\_BINARY)

contours, \_ = cv2.findContours(binary, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

for contour in contours:

approx = cv2.approxPolyDP(contour, 0.01 \* cv2.arcLength(contour, True), True)

if len(approx) == 3:

shape = "Triangle"

elif len(approx) == 4:

shape = "Rectangle"

else:

shape = "Circle"

cv2.putText(image, shape, (approx[0][0][0], approx[0][0][1]), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 0), 2)

cv2.imshow('Shapes', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

* **Code Explanation**: Identifies shapes by approximating contours, labels them based on vertex count, and displays the annotated image.

**Slide 6: Histogram Analysis**

* **Learning Outcome**: Understand image intensity distributions via histograms.
* **Programming Task**: Plot and analyze histograms using Matplotlib and OpenCV.
* **Why It Matters**: Aids in adjusting contrast, brightness, and identifying colors.
* **Real-World Example**: Enhancing low-light footage or balancing photo colors.
* **Skill Gained**: Visual and statistical image analysis.
* **Mini-Project Overview**: Compute and plot an image histogram.
* **Code Snippet**:

import cv2

import matplotlib.pyplot as plt

image = cv2.imread('image.jpg')

hist = cv2.calcHist([image], [0], None, [256], [0, 256])

plt.plot(hist)

plt.title('Histogram')

plt.show()

* **Code Explanation**: Loads an image, calculates its histogram for the blue channel, and plots it to analyze intensity distribution.

**Slide 7: Motion Detection**

* **Learning Outcome**: Detect moving objects with background subtraction.
* **Programming Task**: Track motion in a video stream.
* **Why It Matters**: Key for surveillance, automation, and interaction systems.
* **Real-World Example**: Detecting intruders in CCTV or tracking wildlife.
* **Skill Gained**: Real-time video processing and foreground extraction.
* **Mini-Project Overview**: Detect motion in a video using background subtraction.
* **Code Snippet**:

import cv2

subtractor = cv2.createBackgroundSubtractorMOG2()

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

while True:

ret, frame = cap.read()

if not ret:

break

mask = subtractor.apply(frame)

cv2.imshow('Motion', mask)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

* **Code Explanation**: Uses MOG2 to subtract the background, applies it to video frames, and displays the foreground mask in real-time.

**Slide 8: Face Detection**

* **Learning Outcome**: Use Haar cascades for face detection.
* **Programming Task**: Detect faces and draw bounding boxes in an image.
* **Why It Matters**: Core to biometric systems and social media features.
* **Real-World Example**: Phone unlocking or photo tagging.
* **Skill Gained**: Using pre-trained models for detection.
* **Mini-Project Overview**: Detect faces in an image with Haar cascades.
* **Code Snippet**:

import cv2

face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

image = cv2.imread('image.jpg')

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

faces = face\_cascade.detectMultiScale(gray, 1.1, 4)

for (x, y, w, h) in faces:

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

cv2.imshow('Faces', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

* **Code Explanation**: Loads a Haar cascade, detects faces in a grayscale image, and draws blue rectangles around them.

**Slide 9: Object Tracking**

* **Learning Outcome**: Track objects using color detection.
* **Programming Task**: Track a colored object in a video.
* **Why It Matters**: Vital for navigation, analytics, and augmented reality.
* **Real-World Example**: Tracking players in sports or objects with robots.
* **Skill Gained**: Real-time tracking with color spaces.
* **Mini-Project Overview**: Track a colored object in a video.
* **Code Snippet**:

import cv2

import numpy as np

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

while True:

ret, frame = cap.read()

if not ret:

break

hsv = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

lower = np.array([30, 150, 50])

upper = np.array([255, 255, 180])

mask = cv2.inRange(hsv, lower, upper)

contours, \_ = cv2.findContours(mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

if contours:

c = max(contours, key=cv2.contourArea)

x, y, w, h = cv2.boundingRect(c)

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

cv2.imshow('Tracking', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

* **Code Explanation**: Converts video frames to HSV, masks a color range, finds the largest contour, and draws a green bounding box around it.

**Slide 10: Applying Filters for Artistic Effects**

* **Learning Outcome**: Create artistic filters like cartoon effects.
* **Programming Task**: Convert an image to a cartoon style.
* **Why It Matters**: Enhances user experience in photo editing apps.
* **Real-World Example**: Instagram or Snapchat filters.
* **Skill Gained**: Creative image manipulation.
* **Mini-Project Overview**: Apply a cartoon effect to an image.
* **Code Snippet**:

import cv2

image = cv2.imread('image.jpg')

blurred = cv2.bilateralFilter(image, 9, 75, 75)

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

edges = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE\_THRESH\_MEAN\_C, cv2.THRESH\_BINARY, 9, 2)

cartoon = cv2.bitwise\_and(blurred, blurred, mask=edges)

cv2.imshow('Cartoon', cartoon)

cv2.waitKey(0)

cv2.destroyAllWindows()

* **Code Explanation**: Smooths the image with bilateral filtering, detects edges with adaptive thresholding, and combines them for a cartoon effect.