SRM Institute of Science and Technology

Department of Computer Applications

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Circular – 2020-21

MCA GAI 3rd semester

IoT Devices with Computer Vision Technologies (PGI20D16J)

Lab Manual

Lab 1: Explain working of Raspberry Pi

- **Title:** Working of Raspberry Pi
- **Aim:** To understand the basic architecture, components, and working principles of Raspberry Pi.
- **Procedure:**
    1. **Introduction to Raspberry Pi:** Discuss the Raspberry Pi as a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and developing countries.
    2. **Hardware Overview:**
        - **Processor (SoC):** Explain the System on a Chip (SoC), typically Broadcom BCMxxxx series, which integrates the CPU, GPU, and other peripherals.
        - **RAM:** Describe the onboard RAM.
        - **GPIO Pins:** Detail the General Purpose Input/Output (GPIO) pins, their role in connecting external hardware components (sensors, LEDs, motors), and their importance for IoT projects.
        - **Connectivity:** Discuss various connectivity options like USB ports, Ethernet port, Wi-Fi, Bluetooth (depending on model), HDMI, and camera/display interfaces.
        - **Storage:** Explain the use of microSD card for operating system and data storage.
    3. **Operating System:** Describe Raspberry Pi OS (formerly Raspbian) as the recommended operating system, a Debian-based Linux distribution optimized for the Raspberry Pi hardware. Mention other compatible OS options.
    4. **Boot Process:** Explain how the Raspberry Pi boots from the microSD card, loading the firmware and then the operating system.
    5. **Basic Usage:**
        - **Headless Setup:** Explain how to set up Raspberry Pi without a monitor, keyboard, or mouse (e.g., via SSH).
        - **Desktop Environment:** Describe the graphical desktop environment for direct interaction.
        - **Command Line Interface (CLI):** Emphasize the importance of the terminal for system administration and running scripts.
    6. **Applications in IoT and Computer Vision:** Discuss how Raspberry Pi's small form factor, low power consumption, and GPIO capabilities make it ideal for IoT projects, while its processing power and support for cameras make it suitable for computer vision applications.
- **Source Code:** N/A (Conceptual explanation)
- **Input:** N/A

- **Expected Output:** N/A

Lab 2: Controlling LED with Raspberry Pi

- **Title:** Controlling LED with Raspberry Pi
- **Aim:** To learn how to control a Light Emitting Diode (LED) by programming the General Purpose Input/Output (GPIO) pins of a Raspberry Pi.
- **Procedure:**
    1. **Hardware Setup:**
        - Connect the long leg (anode) of an LED to a 330-ohm resistor.
        - Connect the other end of the resistor to a GPIO pin on the Raspberry Pi (e.g., GPIO 17).
        - Connect the short leg (cathode) of the LED to a GND (Ground) pin on the Raspberry Pi.
        - Ensure the Raspberry Pi is powered off during wiring.
    2. **Software Setup:**
        - Boot your Raspberry Pi and open a terminal.
        - Ensure the `RPi.GPIO` library is installed (`sudo apt-get install python3-rpi.gpio`).
    3. **Write Python Code:** Create a Python script that imports the `RPi.GPIO` library, sets the GPIO mode, configures the chosen pin as an output, and then toggles the LED on and off with a delay.
    4. **Run the Code:** Execute the Python script from the terminal.
- **Source Code:**

```
# Source Code: controlling_led.py

import RPi.GPIO as GPIO
import time

# Set the GPIO mode (BOARD or BCM)
# GPIO.setmode(GPIO.BOARD) refers to the pin number on the board
# GPIO.setmode(GPIO.BCM) refers to the Broadcom SOC channel
GPIO.setmode(GPIO.BCM)

# Define the GPIO pin connected to the LED
LED_PIN = 17 # Example using GPIO 17 (physical pin 11)

# Set up the GPIO pin as an output
GPIO.setup(LED_PIN, GPIO.OUT)

print(f"Controlling LED on GPIO pin {LED_PIN}")
print("Press Ctrl+C to exit")

try:
    while True:
        # Turn the LED on
        GPIO.output(LED_PIN, GPIO.HIGH)
        print("LED ON")
        time.sleep(1) # Wait for 1 second

        # Turn the LED off
        GPIO.output(LED_PIN, GPIO.LOW)
        print("LED OFF")
        time.sleep(1) # Wait for 1 second
```

```
except KeyboardInterrupt:
    # Clean up GPIO settings when the script is interrupted
    print("\nExiting program and cleaning up GPIO...")
    GPIO.cleanup()
```

- **Input:** None (the script runs automatically, but user can interrupt with `Ctrl+C`).
- **Expected Output:** The LED connected to the specified GPIO pin will blink on and off every second. The terminal will display "LED ON" and "LED OFF" messages accordingly.

Lab 3: Interfacing Light Sensor with Raspberry Pi

- **Title:** Interfacing Light Sensor with Raspberry Pi
- **Aim:** To read analog light intensity values from a Light Dependent Resistor (LDR) using a Raspberry Pi. Since Raspberry Pi doesn't have a built-in Analog-to-Digital Converter (ADC), we will use an external ADC module (e.g., MCP3008) or a simple RC circuit for demonstration.
- **Procedure:**
    1. **Hardware Setup (using RC circuit for simplicity):**
        - Connect one leg of the LDR to a 3.3V pin on the Raspberry Pi.
        - Connect the other leg of the LDR to one end of a 10k ohm resistor.
        - Connect the other end of the 10k ohm resistor to a GND pin on the Raspberry Pi.
        - Connect a GPIO pin (e.g., GPIO 4) to the junction between the LDR and the 10k ohm resistor. This forms a voltage divider.
        - (Optional: For more accurate readings, use an MCP3008 ADC. Connect its SPI pins to Raspberry Pi's SPI pins and the LDR to one of its analog input channels.)
    2. **Software Setup:**
        - Boot your Raspberry Pi.
        - Ensure the `RPi.GPIO` library is installed.
        - (If using MCP3008, install `spidev` and `Adafruit_CircuitPython_MCP3xxx` libraries).
    3. **Write Python Code (using RC circuit discharge time):** Create a Python script that measures the time it takes for a capacitor (formed by the LDR and resistor) to charge/discharge, which varies with light intensity.
    4. **Run the Code:** Execute the Python script from the terminal.
- **Source Code (using RC circuit discharge time for basic demonstration):**

```
# Source Code: light_sensor.py

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

# Define the GPIO pin connected to the LDR/resistor junction
LDR_PIN = 4 # Example using GPIO 4 (physical pin 7)

def rc_time(pin):
    count = 0

    # Output on the pin for a short period to discharge capacitor
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.LOW)
    time.sleep(0.1)

    # Change the pin back to input and measure the charge time
    GPIO.setup(pin, GPIO.IN)

    # Count until the pin goes HIGH (capacitor charges)
    while GPIO.input(pin) == GPIO.LOW:
        count += 1
    return count
```

```
print(f"Reading light intensity from GPIO pin {LDR_PIN}")
print("Move your hand over the LDR to see changes. Press Ctrl+C to exit.")

try:
    while True:
        light_value = rc_time(LDR_PIN)
        print(f"Light Intensity: {light_value}")
        time.sleep(0.5) # Read every 0.5 seconds

except KeyboardInterrupt:
    print("\nExiting program and cleaning up GPIO...")
    GPIO.cleanup()
```

- **Input:** Changes in ambient light intensity falling on the LDR.
- **Expected Output:** The terminal will continuously display numerical values representing the light intensity. Higher values indicate more light (faster charge/discharge time), and lower values indicate less light (slower charge/discharge time).

Lab 4: Describe gateway as a service deployment in IoT toolkit

- **Title:** Gateway as a Service Deployment in IoT Toolkit
- **Aim:** To understand the concept, architecture, and deployment models of "Gateway as a Service" within the context of an IoT toolkit or platform.
- **Procedure:**
    1. **Introduction to IoT Gateways:**
        - Define an IoT gateway as a physical device or virtual entity that acts as a bridge between IoT devices/sensors at the edge and cloud-based IoT platforms.
        - Explain its primary functions: protocol translation, data aggregation, data filtering, security, and local processing (edge computing).
    2. **What is "Gateway as a Service" (GaaS)?**
        - Explain GaaS as a model where the functionality of an IoT gateway is provided as a managed service by an IoT platform provider (e.g., AWS IoT Greengrass, Azure IoT Edge, Google Cloud IoT Edge).
        - Highlight that instead of deploying and managing physical gateways, users can leverage pre-configured software components or virtual instances that run on various edge devices, managed centrally by the cloud provider.
    3. **Key Features and Benefits of GaaS:**
        - **Simplified Deployment & Management:** Reduces the operational overhead of managing physical gateways.
        - **Scalability:** Easily scale gateway capabilities up or down based on demand.
        - **Centralized Control:** Manage and monitor edge deployments from a single cloud console.
        - **Security:** Inherits security features from the cloud provider, including secure device provisioning, authentication, and data encryption.
        - **Edge Intelligence:** Enables running machine learning models, analytics, and business logic directly at the edge, reducing latency and bandwidth usage.
        - **Protocol Agnostic:** Often supports various device protocols, simplifying integration.
        - **Over-the-Air (OTA) Updates:** Facilitates remote updates and maintenance of gateway software.
    4. **Deployment Models in IoT Toolkits:**
        - **Software Modules/Runtimes:** The gateway functionality is often provided as a lightweight runtime or set of modules that can be deployed on various edge hardware (e.g., Raspberry Pi, industrial PCs, custom hardware).
        - **Containerization:** GaaS components are frequently deployed as Docker containers or similar containerized applications, allowing for portability and isolated execution.
        - **Cloud-Managed Edge:** The cloud platform manages the lifecycle of the gateway software, including deployment, updates, and monitoring, while the customer provides the underlying hardware.
    5. **Use Cases:**
        - Industrial IoT (IIoT) for connecting legacy equipment.
        - Smart cities for aggregating sensor data.
        - Retail for local processing of video analytics.
        - Healthcare for secure data transfer from medical devices.

6. **Comparison with Traditional Gateways:** Briefly discuss the advantages of GaaS over traditional, self-managed gateway solutions (e.g., reduced complexity, faster time to market).

- **Source Code:** N/A (Conceptual explanation)
- **Input:** N/A
- **Expected Output:** N/A

Lab 5: Weather Monitoring System

- **Title:** Weather Monitoring System
- **Aim:** To build a basic weather monitoring system using a Raspberry Pi and a DHT11/DHT22 temperature and humidity sensor, displaying the readings.
- **Procedure:**
    1. **Hardware Setup:**
        - Connect the VCC pin of the DHT11/DHT22 sensor to the 3.3V pin on the Raspberry Pi.
        - Connect the GND pin of the sensor to a GND pin on the Raspberry Pi.
        - Connect the data pin of the sensor to a GPIO pin on the Raspberry Pi (e.g., GPIO 4).
        - (Optional but recommended: Add a 10k ohm pull-up resistor between the data pin and 3.3V).
        - Ensure the Raspberry Pi is powered off during wiring.
    2. **Software Setup:**
        - Boot your Raspberry Pi.
        - Install the necessary libraries for DHT sensors. A common one is `Adafruit_DHT`.
            - `sudo apt-get update`
            - `sudo apt-get install build-essential python-dev`
            - `git clone https://github.com/adafruit/Adafruit_Python_DHT.git`
            - `cd Adafruit_Python_DHT`
            - `sudo python3 setup.py install`
    3. **Write Python Code:** Create a Python script that imports the `Adafruit_DHT` library, specifies the sensor type and GPIO pin, and then continuously reads and prints temperature and humidity values.
    4. **Run the Code:** Execute the Python script from the terminal.
- **Source Code:**

```
# Source Code: weather_monitor.py

import Adafruit_DHT
import time

# Sensor type: DHT11 or DHT22
# Change this depending on your sensor
SENSOR_TYPE = Adafruit_DHT.DHT11 # or Adafruit_DHT.DHT22

# GPIO pin connected to the sensor's data pin
GPIO_PIN = 4 # Example using GPIO 4 (physical pin 7)

print(f"Weather Monitoring System (Sensor: DHT11/DHT22 on GPIO {GPIO_PIN})")
print("Press Ctrl+C to exit")

try:
    while True:
        # Read temperature and humidity from the sensor
        humidity, temperature = Adafruit_DHT.read_retry(SENSOR_TYPE, GPIO_PIN)

        # Check if readings are valid
        if humidity is not None and temperature is not None:
```

```
            print(f"Temp={temperature:.1f}˚C  Humidity={humidity:.1f}%")
        else:
            print("Failed to retrieve data from sensor. Retrying...")

        time.sleep(2) # Read every 2 seconds

except KeyboardInterrupt:
    print("\nExiting weather monitoring system.")
```

- **Input:** Environmental temperature and humidity.
- **Expected Output:** The terminal will continuously display the current temperature in Celsius and relative humidity percentage, updated every few seconds.

Lab 6: IoT based Soil Moisture Monitoring Device

- **Title:** IoT based Soil Moisture Monitoring Device
- **Aim:** To develop an IoT device using Raspberry Pi and a soil moisture sensor to monitor the moisture levels in soil and display the readings.
- **Procedure:**
    1. **Hardware Setup:**
        - Connect the VCC pin of the soil moisture sensor module (often has a comparator chip like LM393) to the 3.3V or 5V pin on the Raspberry Pi (check sensor module voltage requirements).
        - Connect the GND pin of the sensor module to a GND pin on the Raspberry Pi.
        - Connect the analog output (A0) pin of the soil moisture sensor module to an analog input pin of an ADC (e.g., MCP3008) connected to the Raspberry Pi's SPI pins. (Raspberry Pi doesn't have native analog input, so an ADC is crucial).
        - Alternatively, if the sensor module has a digital output (D0), connect it to a GPIO pin for simple wet/dry detection. We will assume an ADC for more granular readings.
        - Ensure the Raspberry Pi is powered off during wiring.
    2. **Software Setup:**
        - Boot your Raspberry Pi.
        - Enable SPI interface on Raspberry Pi (via `sudo raspi-config`).
        - Install `spidev` and `Adafruit_CircuitPython_MCP3xxx` libraries for MCP3008.
            - `pip3 install spidev`
            - `pip3 install adafruit-circuitpython-mcp3xxx`
    3. **Write Python Code:** Create a Python script that initializes the MCP3008, reads the analog value from the connected channel, and converts it into a meaningful moisture percentage or level.
    4. **Run the Code:** Execute the Python script from the terminal.
- **Source Code (using MCP3008 ADC):**

```
# Source Code: soil_moisture_monitor.py

import busio
import digitalio
import board
import adafruit_mcp3xxx.mcp3008 as MCP
from adafruit_mcp3xxx.analog_in import AnalogIn
import time

# Create the SPI bus
spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)

# Create the chip select (CS) for the MCP3008
cs = digitalio.DigitalInOut(board.D5) # Example using GPIO 5 (physical pin 29)

# Create an MCP3008 object
mcp = MCP.MCP3008(spi, cs)

# Create an analog input channel on MCP3008 (e.g., channel 0)
# Connect soil moisture sensor A0 to MCP3008 CH0
```

```
chan = AnalogIn(mcp, MCP.P0)

print("Soil Moisture Monitoring Device")
print("Place sensor in soil. Press Ctrl+C to exit.")

try:
    while True:
        # Read the raw analog value (0-65535 for 16-bit, or 0-1023 for 10-bit)
        raw_value = chan.value
        voltage = chan.voltage

        # Convert raw value to a moisture percentage (adjust mapping based on calibration)
        # Example mapping: 65535 (dry) to 0% moisture, 0 (wet) to 100% moisture
        # This mapping needs calibration for your specific sensor and soil
        moisture_percentage = ((65535 - raw_value) / 65535) * 100
        if moisture_percentage < 0: moisture_percentage = 0
        if moisture_percentage > 100: moisture_percentage = 100

        print(f"Raw ADC Value: {raw_value}, Voltage: {voltage:.2f}V, Moisture: {moisture_percentage:.2f}%")

        time.sleep(1) # Read every 1 second

except KeyboardInterrupt:
    print("\nExiting soil moisture monitoring system.")
```

- **Input:** Changes in soil moisture level detected by the sensor.
- **Expected Output:** The terminal will continuously display the raw analog value, corresponding voltage, and an estimated moisture percentage. When the sensor is in dry soil, the moisture percentage will be low; when in wet soil, it will be high.

Lab 7: Install OpenCV Displaying images OpenCV

- **Title:** Installing OpenCV and Displaying Images
- **Aim:** To install the OpenCV library on a system and write a basic Python program to load an image from a file and display it in a window.
- **Procedure:**
    1. **Install OpenCV:**
        - Open a terminal or command prompt.
        - Install OpenCV using pip: `pip install opencv-python`
        - (For Raspberry Pi, you might need `pip3 install opencv-python` or a more specific build for ARM architecture if general install fails).
    2. **Prepare an Image:** Place an image file (e.g., `test_image.jpg`) in the same directory as your Python script, or provide its full path.
    3. **Write Python Code:** Create a Python script that imports `cv2`, uses `cv2.imread()` to load an image, `cv2.imshow()` to display it, and `cv2.waitKey()` and `cv2.destroyAllWindows()` for proper window management.
    4. **Run the Code:** Execute the Python script.
- **Source Code:**

```
# Source Code: display_image.py

import cv2
import os
import numpy as np

# Define the image file name
image_name = "test_image.jpg" # Make sure this image exists in the same directory

# Create a dummy image if it doesn't exist for demonstration purposes
if not os.path.exists(image_name):
    print(f"'{image_name}' not found. Creating a placeholder image...")
    # Create a blank white image (300x200 pixels, 3 channels for color)
    dummy_image = np.zeros((200, 300, 3), dtype=np.uint8) + 255 # White image
    cv2.putText(dummy_image, "Placeholder Image", (50, 100), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2)
    cv2.imwrite(image_name, dummy_image)
    print(f"Placeholder image '{image_name}' created.")

# Read the image
img = cv2.imread(image_name)

# Check if image was loaded successfully
if img is None:
    print(f"Error: Could not load image '{image_name}'. Please check the file path.")
else:
    # Display the image
    cv2.imshow("Loaded Image", img)

    # Wait for a key press (0 means wait indefinitely)
    cv2.waitKey(0)

    # Destroy all OpenCV windows
```

- `cv2.destroyAllWindows()`

- **Input:** An image file named `test_image.jpg` in the same directory as the script, or the script will create a placeholder.
- **Expected Output:** A new window titled "Loaded Image" will open, displaying the `test_image.jpg` (or the placeholder image). The window will close when any key is pressed.

Lab 8: Reading & Writing images OpenCV

- **Title:** Reading & Writing Images with OpenCV
- **Aim:** To learn how to read an image from a file, perform a simple modification (e.g., convert to grayscale), and then save the modified image to a new file using OpenCV.
- **Procedure:**
    1. **Prepare an Image:** Ensure you have an image file (e.g., `input_image.jpg`) in the same directory as your Python script.
    2. **Write Python Code:** Create a Python script that reads the image, converts it to grayscale using `cv2.cvtColor()`, and then saves the grayscale image using `cv2.imwrite()`. It will also display both original and grayscale images.
    3. **Run the Code:** Execute the Python script.
- **Source Code:**

```
# Source Code: read_write_image.py

import cv2
import os
import numpy as np

# Define input and output image file names
input_image_name = "input_image.jpg"
output_image_name = "grayscale_image.jpg"

# Create a dummy image if it doesn't exist for demonstration purposes
if not os.path.exists(input_image_name):
    print(f"'{input_image_name}' not found. Creating a placeholder image...")
    # Create a blank colored image (300x200 pixels)
    dummy_image = np.zeros((200, 300, 3), dtype=np.uint8)
    dummy_image[:, :, 0] = 255 # Blue channel
    dummy_image[:, :, 1] = 100 # Green channel
    dummy_image[:, :, 2] = 50  # Red channel
    cv2.putText(dummy_image, "Colorful Placeholder", (30, 100), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
    cv2.imwrite(input_image_name, dummy_image)
    print(f"Placeholder image '{input_image_name}' created.")

# Read the input image
img_color = cv2.imread(input_image_name)

# Check if image was loaded successfully
if img_color is None:
    print(f"Error: Could not load image '{input_image_name}'. Please check the file path.")
else:
    print(f"Successfully loaded '{input_image_name}'.")

    # Convert the image to grayscale
    img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
    print("Image converted to grayscale.")

    # Save the grayscale image
    cv2.imwrite(output_image_name, img_gray)
    print(f"Grayscale image saved as '{output_image_name}'.")
```

```
    # Display both original and grayscale images
    cv2.imshow("Original Image", img_color)
    cv2.imshow("Grayscale Image", img_gray)

    # Wait for a key press
    cv2.waitKey(0)

    # Destroy all OpenCV windows
    cv2.destroyAllWindows()
```

- **Input:** An image file named `input_image.jpg`. If not present, a placeholder image will be created.
- **Expected Output:** Two windows will appear: "Original Image" showing the loaded image and "Grayscale Image" showing its grayscale version. A new file named `grayscale_image.jpg` will be created in the same directory. Both windows will close on key press.

Lab 9: Draw a Rectangle Draw a Circle

- **Title:** Drawing Geometric Shapes with OpenCV
- **Aim:** To learn how to draw basic geometric shapes like rectangles and circles on an image using OpenCV's drawing functions.
- **Procedure:**
    1. **Write Python Code:** Create a Python script that generates a blank image and then uses `cv2.rectangle()` to draw a rectangle and `cv2.circle()` to draw a circle on it.
    2. **Run the Code:** Execute the Python script.
- **Source Code:**

```
# Source Code: draw_shapes.py

import cv2
import numpy as np

# Create a blank black image (height, width, channels)
# 400x600 pixels, 3 channels for BGR color
image = np.zeros((400, 600, 3), dtype=np.uint8)

# Define colors in BGR format
blue = (255, 0, 0)
green = (0, 255, 0)
red = (0, 0, 255)
white = (255, 255, 255)

# Draw a rectangle
# cv2.rectangle(image, start_point, end_point, color, thickness)
# start_point: top-left corner (x, y)
# end_point: bottom-right corner (x, y)
# thickness: -1 for filled rectangle
cv2.rectangle(image, (50, 50), (200, 150), blue, 2) # Blue rectangle, 2 pixels thick
cv2.rectangle(image, (250, 50), (400, 150), green, -1) # Green filled rectangle

# Draw a circle
# cv2.circle(image, center_coordinates, radius, color, thickness)
# center_coordinates: (x, y)
# thickness: -1 for filled circle
cv2.circle(image, (125, 275), 70, red, 3) # Red circle, 3 pixels thick
cv2.circle(image, (350, 275), 70, white, -1) # White filled circle

# Display the image
cv2.imshow("Shapes on Image", image)

# Wait for a key press
cv2.waitKey(0)

# Destroy all OpenCV windows
cv2.destroyAllWindows()
```

- **Input:** None (the script generates the image).

- **Expected Output:** A window titled "Shapes on Image" will display a black canvas with a blue-bordered rectangle, a green-filled rectangle, a red-bordered circle, and a white-filled circle. The window will close on key press.

Lab 10: Text in Images

- **Title:** Adding Text to Images with OpenCV
- **Aim:** To learn how to overlay text onto an image using OpenCV, controlling properties like font, size, color, and thickness.
- **Procedure:**
    1. **Write Python Code:** Create a Python script that generates a blank image and then uses `cv2.putText()` to add various text strings with different styles.
    2. **Run the Code:** Execute the Python script.
- **Source Code:**
- `# Source Code: text_in_images.py`
-
- `import cv2`
- `import numpy as np`
-
- `# Create a blank black image`
- `image = np.zeros((400, 600, 3), dtype=np.uint8) + 50 # Dark grey background`
-
- `# Define text properties`
- `font = cv2.FONT_HERSHEY_SIMPLEX`
- `font_scale = 1`
- `color = (0, 255, 255) # Yellow in BGR`
- `thickness = 2`
- `line_type = cv2.LINE_AA # Anti-aliased line`
-
- `# Add text to the image`
- `# cv2.putText(image, text, org, fontFace, fontScale, color, thickness, lineType)`
- `# org: bottom-left corner of the text string in the image.`
-
- `cv2.putText(image, "Hello, OpenCV!", (50, 50), font, font_scale, color, thickness, line_type)`
-
- `# Another text with different properties`
- `cv2.putText(image, "Computer Vision", (50, 150), cv2.FONT_HERSHEY_COMPLEX, 1.2, (255, 0, 0), 3, cv2.LINE_AA)`
-
- `# Text with a different color and smaller size`
- `cv2.putText(image, "IoT Devices", (50, 250), cv2.FONT_HERSHEY_PLAIN, 0.8, (0, 255, 0), 1, cv2.LINE_AA)`
-
- `# Text with background (by drawing a filled rectangle first)`
- `text_to_add = "PGI20D16J"`
- `(text_width, text_height), baseline = cv2.getTextSize(text_to_add, font, font_scale, thickness)`
- `cv2.rectangle(image, (50, 300), (50 + text_width, 300 + text_height + baseline), (100, 100, 100), -1) # Grey background`
- `cv2.putText(image, text_to_add, (50, 300 + text_height), font, font_scale, white, thickness, line_type)`
-
-
- `# Display the image`
- `cv2.imshow("Image with Text", image)`
-

```
# Wait for a key press
cv2.waitKey(0)

# Destroy all OpenCV windows
cv2.destroyAllWindows()
```

- **Input:** None (the script generates the image).
- **Expected Output:** A window titled "Image with Text" will display a dark grey canvas with several lines of text, each with different fonts, sizes, and colors. One text string will have a grey background. The window will close on key press.

Lab 11: Color Space OpenCV Thresholding OpenCV

- **Title:** Color Space Conversion and Thresholding with OpenCV
- **Aim:** To understand and apply color space conversions (e.g., BGR to HSV, BGR to Grayscale) and various image thresholding techniques (e.g., binary, Otsu's) using OpenCV.
- **Procedure:**
    1. **Prepare an Image:** Use an image with varying colors and intensities (e.g., `color_image.jpg`).
    2. **Write Python Code:** Create a Python script that loads the image, converts it to grayscale and HSV color spaces, and then applies different thresholding methods to the grayscale image.
    3. **Run the Code:** Execute the Python script.
- **Source Code:**

```
# Source Code: color_thresholding.py

import cv2
import numpy as np
import os

# Define the image file name
input_image_name = "color_image.jpg"

# Create a dummy image if it doesn't exist for demonstration purposes
if not os.path.exists(input_image_name):
    print(f"'{input_image_name}' not found. Creating a placeholder image...")
    dummy_image = np.zeros((200, 300, 3), dtype=np.uint8)
    # Draw some colored shapes
    cv2.circle(dummy_image, (100, 100), 50, (0, 0, 255), -1) # Red circle
    cv2.rectangle(dummy_image, (180, 50), (280, 150), (255, 0, 0), -1) # Blue rectangle
    cv2.putText(dummy_image, "Color Test", (70, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
    cv2.imwrite(input_image_name, dummy_image)
    print(f"Placeholder image '{input_image_name}' created.")

# Read the input image
img = cv2.imread(input_image_name)

if img is None:
    print(f"Error: Could not load image '{input_image_name}'. Please ensure it exists.")
else:
    # 1. Color Space Conversion

    # Convert BGR to Grayscale
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    print("Converted to Grayscale.")

    # Convert BGR to HSV (Hue, Saturation, Value)
    hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    print("Converted to HSV.")

    # Display color space conversions
    cv2.imshow("Original Image (BGR)", img)
```

```
    cv2.imshow("Grayscale Image", gray_img)
    cv2.imshow("HSV Image", hsv_img)

    # 2. Thresholding on Grayscale Image

    # Simple Binary Thresholding
    # Pixels above threshold become maxValue, others become 0
    ret, thresh_binary = cv2.threshold(gray_img, 127, 255,
cv2.THRESH_BINARY)
    cv2.imshow("Binary Threshold (127)", thresh_binary)
    print("Applied Binary Threshold.")

    # Inverse Binary Thresholding
    ret, thresh_binary_inv = cv2.threshold(gray_img, 127, 255,
cv2.THRESH_BINARY_INV)
    cv2.imshow("Inverse Binary Threshold (127)", thresh_binary_inv)
    print("Applied Inverse Binary Threshold.")

    # Truncate Thresholding
    # Pixels above threshold become threshold value, others remain same
    ret, thresh_trunc = cv2.threshold(gray_img, 127, 255,
cv2.THRESH_TRUNC)
    cv2.imshow("Truncate Threshold (127)", thresh_trunc)
    print("Applied Truncate Threshold.")

    # To Zero Thresholding
    # Pixels below threshold become 0, others remain same
    ret, thresh_to_zero = cv2.threshold(gray_img, 127, 255,
cv2.THRESH_TOZERO)
    cv2.imshow("To Zero Threshold (127)", thresh_to_zero)
    print("Applied To Zero Threshold.")

    # To Zero Inverse Thresholding
    # Pixels above threshold become 0, others remain same
    ret, thresh_to_zero_inv = cv2.threshold(gray_img, 127, 255,
cv2.THRESH_TOZERO_INV)
    cv2.imshow("To Zero Inverse Threshold (127)", thresh_to_zero_inv)
    print("Applied To Zero Inverse Threshold.")

    # Otsu's Binarization (automatically finds optimal threshold value)
    ret, otsu_threshold = cv2.threshold(gray_img, 0, 255,
cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    print(f"Otsu's Threshold value: {ret}")
    cv2.imshow("Otsu's Threshold", otsu_threshold)
    print("Applied Otsu's Threshold.")

    # Wait for a key press
    cv2.waitKey(0)

    # Destroy all OpenCV windows
    cv2.destroyAllWindows()
```

- **Input:** An image file named `color_image.jpg`. If not present, a placeholder image will be created.
- **Expected Output:** Multiple windows will open, displaying the original image, its grayscale version, its HSV version, and several versions of the grayscale image after applying

different thresholding techniques (Binary, Inverse Binary, Truncate, To Zero, To Zero Inverse, and Otsu's). All windows will close on key press.

Lab 12: Finding Contours

- **Title:** Finding Contours in Images with OpenCV
- **Aim:** To detect and draw contours (boundaries of objects) in an image using OpenCV, which is fundamental for object detection and shape analysis.
- **Procedure:**
    1. **Prepare an Image:** Use a binary image or an image that can be easily converted to binary (e.g., `shapes.png`) where objects are clearly distinguishable from the background.
    2. **Write Python Code:** Create a Python script that loads the image, converts it to grayscale, applies thresholding to get a binary image, then uses `cv2.findContours()` to detect contours and `cv2.drawContours()` to draw them on the original image.
    3. **Run the Code:** Execute the Python script.
- **Source Code:**

```
# Source Code: find_contours.py

import cv2
import numpy as np
import os

# Define the image file name
input_image_name = "shapes.png"

# Create a dummy image if it doesn't exist for demonstration purposes
if not os.path.exists(input_image_name):
    print(f"'{input_image_name}' not found. Creating a placeholder image...")
    dummy_image = np.zeros((300, 300, 3), dtype=np.uint8) + 255 # White background
    cv2.circle(dummy_image, (100, 100), 40, (0, 0, 0), -1) # Black circle
    cv2.rectangle(dummy_image, (180, 50), (250, 120), (0, 0, 0), -1) # Black rectangle
    cv2.putText(dummy_image, "Triangle", (50, 250), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2)
    # Draw a black triangle
    pts = np.array([[150, 200], [100, 280], [200, 280]], np.int32)
    pts = pts.reshape((-1, 1, 2))
    cv2.fillPoly(dummy_image, [pts], (0, 0, 0))

    cv2.imwrite(input_image_name, dummy_image)
    print(f"Placeholder image '{input_image_name}' created.")

# Read the input image
img = cv2.imread(input_image_name)

if img is None:
    print(f"Error: Could not load image '{input_image_name}'. Please ensure it exists.")
else:
    # Convert the image to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Apply binary thresholding to get a binary image
    # Objects should be white (255) and background black (0)
```

```
    ret, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV)

    # Find contours
    # cv2.findContours(image, mode, method)
    # mode: RETR_EXTERNAL (retrieves only extreme outer contours)
    # method: CHAIN_APPROX_SIMPLE (compresses horizontal, vertical, and diagonal segments)
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Create a copy of the original image to draw contours on
    img_contours = img.copy()

    # Draw all found contours on the image
    # cv2.drawContours(image, contours, contourIdx, color, thickness)
    # contourIdx: -1 to draw all contours
    cv2.drawContours(img_contours, contours, -1, (0, 255, 0), 2) # Green color, 2 pixels thick

    print(f"Found {len(contours)} contours.")

    # Display the original, thresholded, and contoured images
    cv2.imshow("Original Image", img)
    cv2.imshow("Thresholded Image", thresh)
    cv2.imshow("Image with Contours", img_contours)

    # Wait for a key press
    cv2.waitKey(0)

    # Destroy all OpenCV windows
    cv2.destroyAllWindows()
```

- **Input:** An image file named `shapes.png`. If not present, a placeholder image with a circle, rectangle, and triangle will be created.
- **Expected Output:** Three windows will appear: "Original Image", "Thresholded Image" (binary version), and "Image with Contours" showing the original image with green outlines around the detected shapes. The number of found contours will be printed to the console. All windows will close on key press.

Lab 13: Image Edge Detection OpenCV

- **Title:** Image Edge Detection with OpenCV (Canny Edge Detector)
- **Aim:** To apply edge detection algorithms, specifically the Canny edge detector, to an image using OpenCV to identify significant changes in image intensity.
- **Procedure:**
    1. **Prepare an Image:** Use an image with clear features (e.g., `building.jpg`).
    2. **Write Python Code:** Create a Python script that loads the image, converts it to grayscale, applies a Gaussian blur to reduce noise, and then uses `cv2.Canny()` to perform edge detection.
    3. **Run the Code:** Execute the Python script.
- **Source Code:**

```
# Source Code: edge_detection.py

import cv2
import numpy as np
import os

# Define the image file name
input_image_name = "building.jpg"

# Create a dummy image if it doesn't exist for demonstration purposes
if not os.path.exists(input_image_name):
    print(f"'{input_image_name}' not found. Creating a placeholder image...")
    dummy_image = np.zeros((300, 400, 3), dtype=np.uint8) + 150 # Grey background
    cv2.rectangle(dummy_image, (50, 50), (350, 250), (50, 50, 50), -1) # Dark grey rectangle (building)
    cv2.rectangle(dummy_image, (120, 100), (180, 150), (200, 200, 200), -1) # Window 1
    cv2.rectangle(dummy_image, (220, 100), (280, 150), (200, 200, 200), -1) # Window 2
    cv2.line(dummy_image, (50, 50), (200, 0), (50, 50, 50), 5) # Roof line 1
    cv2.line(dummy_image, (350, 50), (200, 0), (50, 50, 50), 5) # Roof line 2
    cv2.putText(dummy_image, "Simple Building", (100, 290), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2)
    cv2.imwrite(input_image_name, dummy_image)
    print(f"Placeholder image '{input_image_name}' created.")

# Read the input image
img = cv2.imread(input_image_name)

if img is None:
    print(f"Error: Could not load image '{input_image_name}'. Please ensure it exists.")
else:
    # Convert to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Apply Gaussian blur to reduce noise (important for Canny)
    # cv2.GaussianBlur(src, ksize, sigmaX)
    # ksize: Gaussian kernel size (width, height) - must be odd
```

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

    # Apply Canny edge detector
    # cv2.Canny(image, threshold1, threshold2)
    # threshold1: lower threshold for hysteresis procedure
    # threshold2: upper threshold for hysteresis procedure
    edges = cv2.Canny(blurred, 50, 150)

    # Display the original, grayscale, and edge-detected images
    cv2.imshow("Original Image", img)
    cv2.imshow("Grayscale Image", gray)
    cv2.imshow("Canny Edges", edges)

    # Wait for a key press
    cv2.waitKey(0)

    # Destroy all OpenCV windows
    cv2.destroyAllWindows()
```

- **Input:** An image file named `building.jpg`. If not present, a placeholder image of a simple building will be created.
- **Expected Output:** Three windows will appear: "Original Image", "Grayscale Image", and "Canny Edges" showing the detected edges of objects in the image as white lines on a black background. All windows will close on key press.

Lab 14: Image Scaling & Rotation using OpenCV

- **Title:** Image Scaling and Rotation with OpenCV
- **Aim:** To perform common image transformations such as scaling (resizing) and rotation using OpenCV functions.
- **Procedure:**
    1. **Prepare an Image:** Use any image (e.g., `flower.jpg`).
    2. **Write Python Code:** Create a Python script that loads the image, scales it up and down using `cv2.resize()`, and rotates it using `cv2.getRotationMatrix2D()` and `cv2.warpAffine()`.
    3. **Run the Code:** Execute the Python script.
- **Source Code:**

```
# Source Code: scale_rotate_image.py

import cv2
import numpy as np
import os

# Define the image file name
input_image_name = "flower.jpg"

# Create a dummy image if it doesn't exist for demonstration purposes
if not os.path.exists(input_image_name):
    print(f"'{input_image_name}' not found. Creating a placeholder image...")
    dummy_image = np.zeros((200, 200, 3), dtype=np.uint8) + 200 # Light grey
    cv2.ellipse(dummy_image, (100, 100), (80, 40), 0, 0, 360, (0, 150, 255), -1) # Orange ellipse (petal)
    cv2.circle(dummy_image, (100, 100), 20, (0, 255, 255), -1) # Yellow circle (center)
    cv2.putText(dummy_image, "Flower", (70, 190), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2)
    cv2.imwrite(input_image_name, dummy_image)
    print(f"Placeholder image '{input_image_name}' created.")

# Read the input image
img = cv2.imread(input_image_name)

if img is None:
    print(f"Error: Could not load image '{input_image_name}'. Please ensure it exists.")
else:
    # Get original dimensions
    height, width = img.shape[:2]
    print(f"Original image dimensions: {width}x{height}")

    # 1. Image Scaling (Resizing)

    # Scale down to half size
    # fx, fy: scaling factors along horizontal and vertical axes
    img_half = cv2.resize(img, None, fx=0.5, fy=0.5, interpolation=cv2.INTER_AREA)
    print(f"Scaled down image dimensions: {img_half.shape[1]}x{img_half.shape[0]}")
```

```
    # Scale up to double size
    img_double = cv2.resize(img, None, fx=2, fy=2, interpolation=cv2.INTER_CUBIC)
    print(f"Scaled up image dimensions: {img_double.shape[1]}x{img_double.shape[0]}")

    # Scale to a specific dimension (e.g., 300x300)
    img_fixed_size = cv2.resize(img, (300, 300), interpolation=cv2.INTER_LINEAR)
    print(f"Fixed size image dimensions: {img_fixed_size.shape[1]}x{img_fixed_size.shape[0]}")

    # 2. Image Rotation

    # Define rotation parameters
    center = (width // 2, height // 2) # Center of the image
    angle = 45 # Rotation angle in degrees
    scale = 1.0 # Scaling factor (1.0 means no scaling during rotation)

    # Get the 2D rotation matrix
    # cv2.getRotationMatrix2D(center, angle, scale)
    M = cv2.getRotationMatrix2D(center, angle, scale)

    # Perform the affine transformation (rotation)
    # cv2.warpAffine(src, M, dsize)
    # dsize: size of the output image (width, height)
    img_rotated = cv2.warpAffine(img, M, (width, height))
    print(f"Image rotated by {angle} degrees.")

    # Display results
    cv2.imshow("Original Image", img)
    cv2.imshow("Scaled Down (0.5x)", img_half)
    cv2.imshow("Scaled Up (2x)", img_double)
    cv2.imshow("Fixed Size (300x300)", img_fixed_size)
    cv2.imshow("Rotated by 45 degrees", img_rotated)

    # Wait for a key press
    cv2.waitKey(0)

    # Destroy all OpenCV windows
    cv2.destroyAllWindows()
```

- **Input:** An image file named `flower.jpg`. If not present, a placeholder image of a simple flower will be created.
- **Expected Output:** Several windows will open, displaying the original image, a scaled-down version, a scaled-up version, a version resized to a fixed dimension, and a version rotated by 45 degrees. Dimensions of transformed images will be printed to the console. All windows will close on key press.

Lab 15: Image Translation OpenCV Image Filtering OpenCV

- **Title:** Image Translation and Filtering with OpenCV
- **Aim:** To perform image translation (shifting) and apply various image filtering techniques (e.g., blurring, sharpening) using OpenCV.
- **Procedure:**
    1. **Prepare an Image:** Use any image (e.g., `city.jpg`).
    2. **Write Python Code:** Create a Python script that loads the image, translates it horizontally and vertically, and then applies different filters like average blurring, Gaussian blurring, median blurring, and sharpening (using a custom kernel).
    3. **Run the Code:** Execute the Python script.
- **Source Code:**

```
# Source Code: translate_filter_image.py

import cv2
import numpy as np
import os

# Define the image file name
input_image_name = "city.jpg"

# Create a dummy image if it doesn't exist for demonstration purposes
if not os.path.exists(input_image_name):
    print(f"'{input_image_name}' not found. Creating a placeholder image...")
    dummy_image = np.zeros((300, 400, 3), dtype=np.uint8) + 100 # Dark grey background
    cv2.rectangle(dummy_image, (50, 100), (150, 250), (50, 50, 50), -1) # Building 1
    cv2.rectangle(dummy_image, (180, 50), (280, 250), (70, 70, 70), -1) # Building 2
    cv2.rectangle(dummy_image, (300, 120), (380, 250), (60, 60, 60), -1) # Building 3
    cv2.putText(dummy_image, "Cityscape", (150, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
    cv2.imwrite(input_image_name, dummy_image)
    print(f"Placeholder image '{input_image_name}' created.")

# Read the input image
img = cv2.imread(input_image_name)

if img is None:
    print(f"Error: Could not load image '{input_image_name}'. Please ensure it exists.")
else:
    # Get original dimensions
    height, width = img.shape[:2]

    # 1. Image Translation (Shifting)

    # Define translation matrix: M = [[1, 0, tx], [0, 1, ty]]
    # tx: shift in x-direction (positive for right, negative for left)
    # ty: shift in y-direction (positive for down, negative for up)
    tx, ty = 50, 30 # Shift 50 pixels right, 30 pixels down
    M_translate = np.float32([[1, 0, tx], [0, 1, ty]])
```

```
    img_translated = cv2.warpAffine(img, M_translate, (width, height))
    print(f"Image translated by ({tx}, {ty}) pixels.")

    # 2. Image Filtering

    # Average Blurring (Box Filter)
    # cv2.blur(src, ksize)
    # ksize: blurring kernel size (width, height)
    img_blur_avg = cv2.blur(img, (5, 5))
    print("Applied Average Blurring.")

    # Gaussian Blurring
    # cv2.GaussianBlur(src, ksize, sigmaX)
    # sigmaX: Gaussian kernel standard deviation in X direction (0 means calculated from ksize)
    img_blur_gaussian = cv2.GaussianBlur(img, (5, 5), 0)
    print("Applied Gaussian Blurring.")

    # Median Blurring (good for salt-and-pepper noise)
    # cv2.medianBlur(src, ksize)
    # ksize: aperture linear size (must be odd and > 1)
    img_blur_median = cv2.medianBlur(img, 5)
    print("Applied Median Blurring.")

    # Sharpening (using a custom 2D convolution kernel)
    # Define a sharpening kernel
    sharpen_kernel = np.array([[-1, -1, -1],
                               [-1,  9, -1],
                               [-1, -1, -1]])
    img_sharpened = cv2.filter2D(img, -1, sharpen_kernel) # -1 means output image has same depth as input
    print("Applied Sharpening Filter.")

    # Display results
    cv2.imshow("Original Image", img)
    cv2.imshow("Translated Image", img_translated)
    cv2.imshow("Average Blurred", img_blur_avg)
    cv2.imshow("Gaussian Blurred", img_blur_gaussian)
    cv2.imshow("Median Blurred", img_blur_median)
    cv2.imshow("Sharpened Image", img_sharpened)

    # Wait for a key press
    cv2.waitKey(0)

    # Destroy all OpenCV windows
    cv2.destroyAllWindows()
```

- **Input:** An image file named `city.jpg`. If not present, a placeholder image of a simple cityscape will be created.
- **Expected Output:** Multiple windows will open, displaying the original image, a translated version (shifted), and various filtered versions (average blurred, Gaussian blurred, median blurred, and sharpened). All windows will close on key press.