Computer Vision User Manual

Amirreza Akbari | 810899045

Reza Baghestani | 810899046

Course: Foundations of Computer Vision

Semester: 4022

Instructor: Dr. Seyfipoor

Assignment: HW #02

Table of Contents

1. Introduction

1.1. Overview and Purpose of the Manual

Computer vision is a rapidly advancing field that has found applications in various industries such as healthcare, automotive, retail, and security. It leverages machine learning and artificial intelligence to enable computers to interpret and understand the visual world. The purpose of this manual is to provide a comprehensive guide for individuals who are interested in learning and working with computer vision tools and libraries. This manual covers the installation and setup of essential software, detailed instructions on using important Python libraries, and step-by-step guides to implementing computer vision projects.

This manual is designed for both beginners and intermediate users who have a basic understanding of programming and are keen to delve into the world of computer vision. By the end of this manual, readers will be equipped with the knowledge and skills to start their own computer vision projects and further explore the capabilities of this exciting field. Whether you are a student, researcher, or professional, this guide aims to facilitate your journey into the realm of computer vision.

Readers can expect to learn how to download and install necessary software such as Visual C++, Qt, and Python, and set up a development environment using PyCharm IDE. Essential Python libraries for computer vision, such as OpenCV, Caer, NumPy, Matplotlib, Scikit-Image, and Pillow (PIL), will be introduced with instructions on how to use their basic and most important functionalities. Each library will be presented with installation steps, key features, and common use cases.

Furthermore, the manual will provide code examples for basic image and video processing tasks, followed by more complex projects like video analysis and object detection. These projects will be explained step-by-step, from the initial setup to the final implementation, ensuring that readers can follow along and apply the concepts learned. Resources and references to useful links, documentation, and recommended readings will also be included to support and extend the learning experience.

In addition to technical instructions, this manual will also highlight best practices and common pitfalls in computer vision projects. By sharing practical insights and tips, we aim to help readers avoid common mistakes and optimize their workflows. The ultimate goal of this manual is to empower readers with the tools and knowledge needed to successfully undertake computer vision projects and contribute to advancements in this dynamic field.

1.2. Introduction to Computer Vision

Computer vision is a field of artificial intelligence that enables machines to interpret and understand the visual world. Using digital images from cameras and videos, along with advanced algorithms and deep learning models, machines can accurately identify and classify objects, and then react to what they "see." Computer vision is essential for a range of applications, from facial recognition and medical image analysis to autonomous driving and retail analytics.

At its core, computer vision aims to mimic the human visual system. However, unlike human vision, which relies on biological processes, computer vision relies on algorithms and computational methods. These methods transform visual data into information that can be analyzed and acted upon. The rapid advancements in computational power and machine learning techniques have significantly accelerated the development and adoption of computer vision technologies.

Important Fields and Subjects in Computer Vision

Computer vision's diverse applications highlight its transformative impact across industries. As technology advances, the potential for new and innovative uses of computer vision continues to grow, making it an exciting and dynamic field to explore. Here are some examples:

- **Image Processing:**
    - **Definition:** Image processing involves the manipulation and analysis of digital images to enhance their quality or extract useful information. This field includes techniques such as noise reduction, contrast enhancement, and image transformation.
    - **Techniques:** Common techniques include filtering, morphological operations, and histogram equalization.
    - **Applications:** Image processing is foundational for preparing images for further analysis in applications like medical imaging, satellite imagery, and industrial inspection.

- **Feature Extraction:**
    - **Definition:** Feature extraction focuses on identifying key points and patterns in images, such as edges, corners, and textures. These features are crucial for object recognition and image matching.
    - **Techniques:** Techniques include edge detection (e.g., Canny edge detector), corner detection (e.g., Harris corner detector), and texture analysis.
    - **Applications:** Feature extraction is used in applications like facial recognition, image stitching, and object tracking.
- **Image Segmentation:**
    - **Definition:** Image segmentation is the process of partitioning an image into different regions or objects. It simplifies image analysis by separating the foreground from the background and isolating objects of interest.
    - **Techniques:** Techniques include thresholding, clustering (e.g., k-means), and deep learning-based segmentation (e.g., U-Net).
    - **Applications:** Image segmentation is crucial in medical imaging (e.g., tumor detection), autonomous driving (e.g., road segmentation), and scene understanding.
- **Object Detection and Recognition:**
    - **Definition:** Object detection involves identifying and locating objects within an image or video. Object recognition goes a step further by classifying the detected objects.
    - **Techniques:** Techniques range from traditional methods like Haar cascades to modern deep learning approaches like YOLO (You Only Look Once) and SSD (Single Shot Multibox Detector).
    - **Applications:** Object detection and recognition are used in applications like surveillance, retail analytics, and autonomous vehicles.
- **Motion Analysis:**
    - **Definition:** Motion analysis studies the movement of objects in a sequence of images or video. This includes techniques like optical flow and object tracking.
    - **Techniques:** Techniques include background subtraction, Kalman filtering, and deep learning-based tracking (e.g., DeepSORT).
    - **Applications:** Motion analysis is essential for video surveillance, sports analytics, and autonomous navigation.

Applications of Computer Vision

- **Healthcare:**
    - **Medical Imaging:** Computer vision assists in diagnosing diseases through analysis of medical images such as X-rays, MRIs, and CT scans. Techniques like image segmentation and pattern recognition help identify abnormalities and monitor treatment progress.
    - **Surgery Assistance:** Computer vision is used in robotic-assisted surgeries to enhance precision and minimize invasiveness.
- **Automotive:**
    - **Autonomous Vehicles:** Computer vision enables autonomous vehicles to perceive their surroundings, recognize traffic signs, detect obstacles, and navigate safely. Techniques like object detection, lane detection, and motion analysis are crucial for autonomous driving.
    - **Driver Assistance:** Advanced driver-assistance systems (ADAS) use computer vision to provide features like lane departure warning, adaptive cruise control, and automatic emergency braking.
- **Retail:**
    - **Customer Analytics:** Computer vision analyzes customer behavior in stores, tracking movement patterns and interactions with products. This data helps optimize store layouts and marketing strategies.
    - **Automated Checkout:** Vision-based systems like Amazon Go use computer vision to automatically track items picked by customers and streamline the checkout process.
- **Security:**
    - **Surveillance:** Computer vision enhances security systems through intelligent video surveillance. Techniques like motion detection, facial recognition, and anomaly detection help monitor and secure premises.
    - **Access Control:** Vision-based access control systems use facial recognition to grant or deny access to restricted areas.
- **Entertainment:**
    - **Augmented Reality (AR) and Virtual Reality (VR):** Computer vision enables immersive experiences by tracking the user's movements and integrating digital content with the real world. Applications include gaming, training simulations, and interactive advertising.

- **Content Creation:** Vision technologies assist in creating realistic visual effects and animations in movies and video games.

- **Agriculture:**
  - **Crop Monitoring:** Computer vision is used to monitor crop health, detect pests, and estimate yields. Drones equipped with vision systems capture aerial images for analysis.
  - **Automated Harvesting:** Vision-guided robots assist in tasks like fruit picking and sorting, increasing efficiency and reducing labor costs.

2. Getting Started

2.1. Download and Installation of Softwares

To effectively work on computer vision projects, it is crucial to have the appropriate software tools installed. This section covers the process of downloading and installing essential software tools required for computer vision projects. This includes Visual C++ for compiling and running C++ programs, Qt for developing cross-platform applications, and Python, the primary language used throughout this manual. Detailed steps and video links are provided to ensure a smooth installation experience, ensuring that all the necessary tools are ready to start computer vision projects.

Visual C++

Visual Studio is a comprehensive integrated development environment (IDE) developed by Microsoft. It supports multiple programming languages, including C++, C#, and Python, making it versatile for various development tasks. Visual Studio provides powerful debugging capabilities, a rich set of built-in tools, and seamless integration with Azure cloud services. It is widely used for developing desktop applications, web applications, and mobile apps across different platforms.

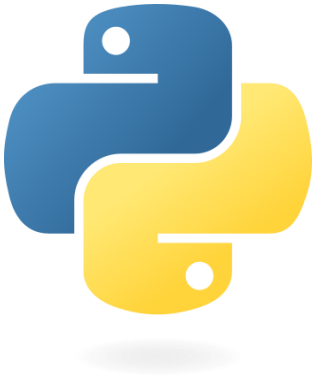
1. **Download Visual Studio:**
    - Visit the Visual Studio download page
    - Choose the appropriate version (Community, Professional, or Enterprise).
2. **Install Visual Studio:**
    - Run the installer and select the "Desktop development with C++" workload.
    - Follow the on-screen instructions to complete the installation.
    - Watch the installation video
    - Watch the installation video

Qt

Qt is a powerful cross-platform application framework primarily used for developing graphical user interfaces (GUIs). It provides a comprehensive set of libraries and tools that enable developers to create applications that run on multiple operating systems with minimal code changes. Qt supports C++ and Python programming languages and is renowned for its performance, scalability, and extensive documentation. It is ideal for developing applications ranging from embedded systems to desktop software.

1. **Download Qt:**
    - Visit the Qt download page
    - Choose the open-source or commercial version.
2. **Install Qt:**
    - Run the installer and select the required components, including the Qt framework and Qt Creator IDE.
    - Follow the on-screen instructions to complete the installation.
    - Watch the installation video
    - Watch the installation video

Python



Python is a high-level, interpreted programming language known for its simplicity and readability. It emphasizes code readability and allows programmers to express concepts in fewer lines of code compared to other languages. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming. It has a vast standard library and a thriving ecosystem of third-party libraries, making it suitable for various applications such as web development, data analysis, artificial intelligence, and of course, computer vision.

1. **Download Python:**
    - Visit the Python download page
    - Download the latest version compatible with your OS.
2. **Install Python:**
    - Run the installer and ensure to check the box "Add Python to PATH."
    - Follow the on-screen instructions to complete the installation.
    - Watch the installation video
    - Watch the installation video

2.2. Installation of IDE

Optimizing the development environment is pivotal for enhancing productivity and facilitating seamless coding practices. The focus now shifts to setting up PyCharm, an IDE tailored for Python development. PyCharm stands out for its robust features, user-friendly interface, and strong support for Python libraries central to computer vision. This section provides comprehensive instructions on downloading, installing, and configuring PyCharm for Python, setting the stage for an efficient development journey.

PyCharm

PyCharm is a popular integrated development environment (IDE) specifically designed for Python development. Developed by JetBrains, PyCharm offers smart code completion, code navigation, and refactoring tools that enhance productivity. It supports web development frameworks like Django and Flask, scientific computing libraries like NumPy and Matplotlib, and of course, frameworks and libraries used in computer vision projects such as OpenCV. PyCharm is available in both professional and community editions, catering to developers of all levels.

1. **Download PyCharm:**
    - Visit the PyCharm download page
    - Choose the Community (free) or Professional (paid) version.
2. **Install PyCharm:**
    - Run the installer and follow the on-screen instructions.
    - Configure PyCharm to use the installed Python interpreter.
    - Watch the installation video

2.3 Setting Up Development Environment

To effectively develop computer vision projects, it is crucial to set up a comprehensive development environment. This involves creating a new project in PyCharm, installing essential libraries, and verifying the setup with a simple script. Here's a detailed guide:

1. Create a New Project in PyCharm

Setting up a new project ensures a clean environment to work on your computer vision tasks.

1. **Open PyCharm:**
    - Launch PyCharm from your start menu or applications folder.
2. **Create a New Project:**
    - Click on "Create New Project" from the welcome screen.
3. **Set Project Location:**
    - Choose a directory where you want to save your project.
4. **Configure Python Interpreter:**
    - In the "New Project" dialog, specify the location of the Python interpreter installed previously.
    - Click on "Existing Interpreter" and select the interpreter from the list.
5. **Create the Project:**
    - Click on "Create" to set up the project.

2. Install Required Libraries

Once the project is created, the next step is to install the necessary libraries that will be used throughout the computer vision projects.

1. **Open the Terminal in PyCharm:**
    - Inside PyCharm, open the terminal by selecting "View" > "Tool Windows" > "Terminal" or by using the shortcut Alt + F12.
2. **Install Libraries Using pip:**

- In the terminal, enter the following command to install the essential libraries:

```
pip install opencv-python numpy matplotlib scikit-image pillow caer
```

- This command installs OpenCV for computer vision tasks, NumPy for numerical operations, Matplotlib for plotting, Scikit-Image for image processing, Pillow for image manipulation, and Caer for additional vision functionalities.

3. Verify the Installation

To ensure everything is set up correctly, run a simple script to import and test the libraries.

1. **Create a Test Script:**
   - In PyCharm, create a new Python file (e.g., test_setup.py) in your project directory.
2. **Write the Script:**
   - Add the following code to the script to import the libraries and perform basic operations:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
from PIL import Image
import caer

# Print library versions to verify installations
print("OpenCV version:", cv2.__version__)
print("NumPy version:", np.__version__)
print("Matplotlib version:", plt.__version__)
print("Scikit-Image version:", data.__version__)
print("Pillow version:", Image.__version__)
print("Caer version:", caer.__version__)

# Test OpenCV by reading and displaying an image
img = cv2.imread('path_to_an_image.jpg')
if img is not None:
    cv2.imshow('Test Image', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
else:
    print("OpenCV Test: Image not found or unable to read.")

# Test NumPy by creating and printing an array
```

```
array = np.array([1, 2, 3])
print("NumPy Test: Array -", array)

# Test Matplotlib by plotting a graph
plt.plot([1, 2, 3], [4, 5, 6])
plt.title("Matplotlib Test")
plt.show()

# Test Scikit-Image by displaying a sample image
sample_img = data.camera()
plt.imshow(sample_img, cmap='gray')
plt.title("Scikit-Image Test")
plt.show()

# Test Pillow by opening and displaying an image
try:
    pil_img = Image.open('path_to_an_image.jpg')
    pil_img.show()
except IOError:
    print("Pillow Test: Image not found or unable to open.")

# Test Caer by loading a sample image
caer_img = caer.data.sunset()
caer.imshow(caer_img, 'Caer Test Image')
```

3. **Run the Script:**
    - Run the script by right-clicking on the file in the project directory and selecting “Run” or by pressing Shift + F10.
    - Verify that there are no errors and that each library performs its intended operation.

By following these steps, the development environment will be set up and ready for the implementation of various computer vision projects. This setup ensures that all necessary tools and libraries are available, providing a solid foundation for advanced computer vision tasks.

3. Essential Python Libraries for Computer Vision

Understanding the essential Python libraries for computer vision is crucial for developers aiming to excel in image processing and analysis. These libraries serve as foundational tools, providing diverse functionalities essential for tasks such as image manipulation and feature extraction. By familiarizing themselves with these libraries, developers gain the capability to implement sophisticated computer vision algorithms and applications effectively. This section of the manual guides readers through the installation process, introduces basic commands, and provides practical usage examples, empowering them to harness these powerful tools for their own projects.

OpenCV

Description: OpenCV (Open Source Computer Vision Library) is developed by a community-driven team of researchers and engineers at Intel. It is a comprehensive open-source library for real-time computer vision tasks. OpenCV supports a wide range of image and video processing functionalities, including image filtering, feature detection, object tracking, and machine learning algorithms for computer vision tasks. OpenCV is highly optimized and widely used in both academic research and industrial applications due to its speed and efficiency in handling large-scale image processing tasks.

- OpenCV is written in C++ and has Python bindings, making it efficient and versatile.
- It has a large active community, providing continuous updates and support for the latest computer vision techniques.
- OpenCV's capabilities extend beyond traditional computer vision to include deep learning modules for advanced tasks like image segmentation and object recognition.

Installation:

```
pip install opencv-python
```

Basic Commands:

```
cv2.imread() # Read an image from a file.
cv2.imshow() # Display an image in a window.
cv2.imwrite() # Save an image to a file.
```

More Info:

- OpenCV Official Website
- OpenCV Documentation
- OpenCV Github Repository
- Youtube Video: OpenCV Course - Full Tutorial with Python

NumPy

Description: NumPy (Numerical Python) is developed by a team of contributors and is a fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. NumPy's efficient data structures and operations make it indispensable for tasks such as data analysis, numerical simulations, and image processing in Python.

- NumPy arrays are homogeneous and memory-efficient, allowing for fast array operations and manipulation.
- It includes a broad range of mathematical functions for array manipulation, linear algebra, Fourier transforms, and random number generation.
- NumPy arrays are the foundation for many other Python libraries and frameworks used in scientific computing and data analysis.

Installation:

```
pip install numpy
```

Basic Commands:

```
np.array() # Create an array.
np.zeros() # Create an array filled with zeros.
np.ones() # Create an array filled with ones.
```

More Info:

- NumPy Official Website
- NumPy Documentation
- Numpy Github Repository
- Youtube Video: Python Numpy Tutorial for Beginners

Pillow (PIL)

Description: Pillow (Python Imaging Library) is developed by a team of contributors and is a fork of the original Python Imaging Library (PIL) that adds support for more image file formats and features. It provides basic image processing capabilities such as opening, manipulating, and saving different image file formats. Pillow is easy to use and is often used for simple image editing tasks and format conversions in Python applications.

- Pillow supports a wide range of image file formats, including JPEG, PNG, TIFF, and BMP.
- It provides basic image manipulation operations such as resizing, cropping, rotating, and applying filters to images.
- Pillow is a lightweight and straightforward library, making it suitable for tasks that require simple image processing functionalities without the complexity of larger libraries like OpenCV.

Installation:

```
pip install pillow
```

Basic Commands:

```
Image.open() # Open an image.
Image.save() # Save an image.
Image.filter() # Apply a filter to an image.
```

More Info:

- Pillow Official Website
- Pillow Documentation
- Pillow Github Repository
- Youtube Playlist: Python Pillow Library (PIL) - Image processing

Matplotlib

Description: Matplotlib is developed by a team of contributors and is a plotting library for creating static, animated, and interactive visualizations in Python. It provides a MATLAB-like interface for generating plots, charts, histograms, and other graphical representations of data. Matplotlib is highly customizable and supports a variety of output formats, making it suitable for both interactive use and creating publication-quality figures.

- Matplotlib integrates seamlessly with NumPy arrays, making it easy to visualize data stored in NumPy arrays.
- It offers multiple plotting styles and customization options, allowing users to create complex visualizations with ease.
- Matplotlib's interactive mode enables real-time manipulation of plots and charts, enhancing the user experience in data exploration and analysis.

Installation:

```
pip install matplotlib
```

Basic Commands:

```
plt.imshow() # Display data as an image.
plt.plot() # Plot data points.
plt.show() # Display a figure.
```

More Info:

- Matplotlib Official Website
- Matplotlib Documentation
- Matplotlib Github Repository
- Youtube Video: Matplotlib Crash Course

Scikit-Image

Description: Scikit-Image is developed by an open-source community and is a collection of algorithms for image processing in Python. It provides a comprehensive set of tools for tasks such as image segmentation, feature extraction, and image restoration. Scikit-Image is built on top of NumPy, SciPy, and Matplotlib, leveraging their functionalities to provide efficient and scalable image processing capabilities.

- Scikit-Image implements a wide range of algorithms for image enhancement, geometric transformations, and object detection.
- It supports integration with other Python libraries like scikit-learn for machine learning tasks involving image data.
- Scikit-Image emphasizes ease of use and interoperability with other scientific computing libraries, making it suitable for both research and practical applications in image processing.

Installation:

```
pip install scikit-image
```

Basic Commands:

```
skimage.io.imread() # Read an image.
skimage.io.imshow() # Display an image.
skimage.filters.gaussian() # Apply Gaussian filter.
```

More Info:

- Scikit-Image Official Website
- Scikit-Image Documentation
- Scikit-Image Github Repository
- Youtube Video: Introduction to image processing using scikit-image in Python

Caer

Description: Caer is developed by Jason Dsouza and is designed as a lightweight, high-level computer vision library built on top of OpenCV and other Python libraries. It simplifies the process of working with complex image datasets and provides a user-friendly interface for rapid prototyping and experimentation in computer vision applications. Caer is designed to be intuitive and easy to use, making it ideal for beginners and professionals alike who need efficient tools for image processing and analysis.

- Caer emphasizes ease of use and performance, leveraging OpenCV's capabilities while providing additional abstractions for common computer vision tasks.
- It supports GPU acceleration, enabling faster processing of large datasets and complex operations.
- Caer integrates seamlessly with other Python libraries like NumPy and Matplotlib for enhanced data manipulation and visualization capabilities.

Installation:

```
pip install caer
```

Basic Commands:

```
caer.imread() # Read an image from a file.
caer.imsave() # Save an image to a file.
caer.cvt_color() # Convert image colors.
```

More Info:

- Caer Official Website
- Caer Documentation
- Caer Github Repository

4. Basic Commands and Functionalities

Image Processing

- Reading and Displaying an Image

```
import cv2
img = cv2.imread('image.jpg')
cv2.imshow('Image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- Converting to Binary

```
binary = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)[1]
```

- Converting to Grayscale

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

- Applying Gaussian Blur

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

- Edge Detection using Sobel

```
sobel_x = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)
sobel_edges = cv2.magnitude(sobel_x, sobel_y)
```

- Edge Detection using Canny

```
edges = cv2.Canny(img, 100, 200)
```

- Sharpening

```
sharpened_image = cv2.detailEnhance(initial_image)
```

Video Processing

- Reading and Displaying a Video

```
import cv2
cap = cv2.VideoCapture('video.mp4')
while(cap.isOpened()):
    ret, frame = cap.read()
    if ret:
        cv2.imshow('Video', frame)
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    else:
        break
cap.release()
cv2.destroyAllWindows()
```

- Converting Video Frames to Grayscale

```
import cv2
cap = cv2.VideoCapture('video.mp4')
while(cap.isOpened()):
    ret, frame = cap.read()
    if ret:
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        cv2.imshow('Grayscale Video', gray)
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    else:
        break
cap.release()
cv2.destroyAllWindows()
```

- Detecting Edges in Video Frames

```
import cv2
cap = cv2.VideoCapture('video.mp4')
while(cap.isOpened()):
    ret, frame = cap.read()
    if ret:
        edges = cv2.Canny(frame, 100, 200)
        cv2.imshow('Edges in Video', edges)
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    else:
        break
cap.release()
```

```
cv2.destroyAllWindows()
```

- Blurring Video Frames

```
import cv2
cap = cv2.VideoCapture('video.mp4')
while cap.isOpened():
    ret, frame = cap.read()
    if ret:
        blurred = cv2.GaussianBlur(frame, (15, 15), 0)
        cv2.imshow('Blurred Video', blurred)
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    else:
        break
cap.release()
cv2.destroyAllWindows()
```

- Drawing Contours on Video Frames

```
import cv2
cap = cv2.VideoCapture('video.mp4')
while cap.isOpened():
    ret, frame = cap.read()
    if ret:
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        _, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)
        contours, _ = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
        cv2.drawContours(frame, contours, -1, (0, 255, 0), 3)
        cv2.imshow('Contours in Video', frame)
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    else:
        break
cap.release()
cv2.destroyAllWindows()
```

- Detecting Faces in Video Frames

```
import cv2
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
cap = cv2.VideoCapture('video.mp4')
while cap.isOpened():
    ret, frame = cap.read()
    if ret:
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)
        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
        cv2.imshow('Face Detection', frame)
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    else:
        break
cap.release()
cv2.destroyAllWindows()
```

- Tracing Objects in Video Frames

```
import cv2
cap = cv2.VideoCapture('video.mp4')
tracker = cv2.legacy.TrackerMOSSE_create()
ret, frame = cap.read()
bbox = cv2.selectROI('Tracking', frame, False)
tracker.init(frame, bbox)
while cap.isOpened():
    ret, frame = cap.read()
    if ret:
        success, bbox = tracker.update(frame)
        if success:
            p1 = (int(bbox[0]), int(bbox[1]))
            p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
            cv2.rectangle(frame, p1, p2, (255, 0, 0), 2, 1)
        else:
            cv2.putText(frame, 'Tracking failure detected', (100, 80),
cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 0, 255), 2)
        cv2.imshow('Object Tracking', frame)
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    else:
        break
cap.release()
cv2.destroyAllWindows()
```

5. Step-by-Step Computer Vision Projects

Project 1: Advanced Image Processing

This project delves into advanced image processing tasks. It includes reading an image from a file, resizing the image, rotating the image, detecting and drawing contours, and applying image segmentation using K-means clustering. These advanced tasks provide a deeper understanding of image processing techniques used in real-world applications.

```
import cv2
import numpy as np

# Read an image from a file
img = cv2.imread('image.jpg')

# Resize the image
resized_img = cv2.resize(img, (800, 600))
cv2.imshow('Resized Image', resized_img)

# Rotate the image by 45 degrees
(h, w) = resized_img.shape[:2]
center = (w // 2, h // 2)
matrix = cv2.getRotationMatrix2D(center, 45, 1.0)
rotated_img = cv2.warpAffine(resized_img, matrix, (w, h))
cv2.imshow('Rotated Image', rotated_img)

# Convert the image to grayscale
gray = cv2.cvtColor(rotated_img, cv2.COLOR_BGR2GRAY)

# Detect edges using Canny edge detection
edges = cv2.Canny(gray, 50, 150)

# Find and draw contours
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(rotated_img, contours, -1, (0, 255, 0), 2)
cv2.imshow('Contours', rotated_img)

# Apply K-means clustering for image segmentation
Z = img.reshape((-1, 3))
Z = np.float32(Z)

# Define criteria, number of clusters(K) and apply K-means
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
```

```
K = 4
_, labels, centers = cv2.kmeans(Z, K, None, criteria, 10,
cv2.KMEANS_RANDOM_CENTERS)

# Convert back into uint8 and make the original image
centers = np.uint8(centers)
segmented_img = centers[labels.flatten()]
segmented_img = segmented_img.reshape((img.shape))
cv2.imshow('Segmented Image', segmented_img)

# Save the processed images
cv2.imwrite('resized_image.jpg', resized_img)
cv2.imwrite('rotated_image.jpg', rotated_img)
cv2.imwrite('contours_image.jpg', rotated_img)
cv2.imwrite('segmented_image.jpg', segmented_img)

# Wait for a key press and close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Project 2: Advanced Video Analysis

This project focuses on advanced video analysis tasks, such as reading a video, stabilizing the video frames, applying motion detection, and tracking moving objects using background subtraction and contour detection. These tasks are crucial for surveillance and monitoring applications.

```
import cv2
import numpy as np

# Capture video from a file
cap = cv2.VideoCapture('video.mp4')

# Initialize the background subtractor
fgbg = cv2.createBackgroundSubtractorMOG2()

while cap.isOpened():
    ret, frame = cap.read()
    if ret:
        # Apply background subtraction
        fgmask = fgbg.apply(frame)

        # Find contours of moving objects
        contours, _ = cv2.findContours(fgmask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

```
        # Draw bounding boxes around moving objects
        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)

        # Display the frame with motion detection
        cv2.imshow('Motion Detection', frame)

        # Break the loop if 'q' is pressed
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    else:
        break

# Release the video capture object and close all windows
cap.release()
cv2.destroyAllWindows()
```

Project 3: Advanced Object Detection and Tracking

This project extends object detection to include object tracking. It involves loading a pre-trained deep learning model, detecting objects in an image, and tracking the detected objects across multiple frames in a video. The project uses the OpenCV DNN module for detection and the CSRT tracker for tracking.

```
import cv2
import numpy as np

# Load the pre-trained model and class labels
net = cv2.dnn.readNetFromCaffe('deploy.prototxt',
'res10_300x300_ssd_iter_140000.caffemodel')

# Initialize the video capture object
cap = cv2.VideoCapture('video.mp4')

# Initialize the tracker
tracker = cv2.TrackerCSRT_create()
initBB = None

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

```
    (h, w) = frame.shape[:2]

    # If we are currently tracking an object, update the tracker
    if initBB is not None:
        (success, box) = tracker.update(frame)
        if success:
            (x, y, w, h) = [int(v) for v in box]
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
            cv2.putText(frame, "Tracking", (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 255, 0), 2)
        else:
            initBB = None

    # Otherwise, detect objects in the frame
    else:
        blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0, (300,
300), (104.0, 177.0, 123.0))
        net.setInput(blob)
        detections = net.forward()

        # Loop over the detections
        for i in range(detections.shape[2]):
            confidence = detections[0, 0, i, 2]
            if confidence > 0.5:
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")

                # Initialize the tracker with the first detected object
                initBB = (startX, startY, endX - startX, endY - startY)
                tracker.init(frame, initBB)
                break

    # Display the output frame
    cv2.imshow('Object Detection and Tracking', frame)

    # Break the loop if 'q' is pressed
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

# Release the video capture object and close all windows
cap.release()
cv2.destroyAllWindows()
```

6. Resources and References

6.1. Useful Links

- Deep Learning for Computer Vision with Python and TensorFlow – Complete Course
- Advanced Computer Vision with Python - Full Course
- Computer Vision and Perception for Self-Driving Cars (Deep Learning Course)
- PyTorch and Monai for AI Healthcare Imaging - Python Machine Learning Course
- Stanford Computer Vision
- Computer Vision — Andreas Geiger
- Introduction to Computer Vision
- Computer Vision and Image Processing - Fundamentals and Applications

6.2. Links Included in the Manual

Downloads

- Visual Studio
- Qt
- Python
- Pycharm

Installations

- Visual Studio #1
- Visual Studio #2
- Qt #1
- Qt #2
- Python #1
- Python #2

- Pycharm

Official Websites

- OpenCV
- NumPy
- Pillow (PIL)
- Matplotlib
- Scikit-Image
- Caer

Documentations

- OpenCV
- NumPy
- Pillow (PIL)
- Matplotlib
- Scikit-Image
- Caer

Github Repositories

- OpenCV
- NumPy
- Pillow (PIL)
- Matplotlib
- Scikit-Image
- Caer

Tutorials

- OpenCV Course - Full Tutorial with Python
- Python Numpy Tutorial for Beginners

- Python Pillow Library (PIL) - Image processing
- Matplotlib Crash Course
- Introduction to image processing using scikit-image in Python

6.3. Recommended Readings

- **Computer Vision: Algorithms and Applications** by Richard Szeliski This book provides a comprehensive overview of fundamental algorithms and techniques used in computer vision, suitable for both beginners and experienced practitioners.
- **Deep Learning for Computer Vision** by Rajalingappaa Shanmugamani An introduction to applying deep learning techniques for computer vision tasks, with practical examples and case studies.
- **Hands-On Computer Vision with OpenCV** by Ata Akhtar and Tayyab Siddiqui A practical guide to mastering OpenCV with hands-on projects and real-world applications.
- **Multiple View Geometry in Computer Vision** by Richard Hartley and Andrew Zisserman A detailed exploration of geometric principles and their application in computer vision, including 3D reconstruction and stereo vision.
- **Digital Image Processing** by Rafael C. Gonzalez and Richard E. Woods A classic textbook covering a wide range of image processing techniques, from basic concepts to advanced algorithms.
- **Pattern Recognition and Machine Learning** by Christopher M. Bishop A comprehensive resource on pattern recognition, machine learning algorithms, and their applications in computer vision.
- **Learning OpenCV 4: Computer Vision with Python** by Adrian Kaehler and Gary Bradski An updated guide to OpenCV, focusing on practical implementations and Python programming for computer vision tasks.
- **Programming Computer Vision with Python** by Jan Erik Solem A hands-on introduction to computer vision, with practical Python examples and projects.
- **Deep Learning** by Ian Goodfellow, Yoshua Bengio, and Aaron Courville An in-depth resource on deep learning techniques, foundational concepts, and advanced topics relevant to computer vision.
- **Computer Vision: A Modern Approach** by David A. Forsyth and Jean Ponce A comprehensive textbook covering modern computer vision methods, including feature extraction, object recognition, and motion analysis.