MARCH 20, 2023

Cleveland State University

EEC 623 – SOFTWARE QUALITY ASSURANCE

PYTHON-OPENCV

Madhu Prakash Behara - 2845381
Sai Rohit Avula - 2837016
Ajay motharapu - 2861616

Introduction:

The software testing project aims to test an open-source Python software named "Python-OpenCV."

Image processing is a technique to perform a set of operations on an image in order to extract or keep any useful information in it. It is a type of signal processing where the input is any image and the output is an image with desired characteristics or features.

OpenCV (Open Source Computer Vision Library) is a powerful open-source software widely used in image processing, deep learning, and computer vision applications. Using the library, we can process images and videos to extract key features which help in detecting objects or regions of interest (ROI) in an image. It provides support for multiple programming languages such as C++, Java, and Python, and seamlessly integrates with libraries, such as NumPy, to process images as numerical arrays.

GITHUB REPO: **https://github.com/opencv/opencv**.

1.2. Major functions of the software:

The Python-OpenCV software library provides a wide range of computer vision functions, including image and video processing, object detection, face recognition, optical character recognition, and machine learning algorithms.

TASKS

The following tasks are to be performed during the software testing project:

Installation of the software & Set up a testing environment:

Pre-Requisites and Other Softwares
Python
matplotlib
NumPy

Installing OpenCV
For Linux users:
pip3 install opencv-python

For Windows users:
pip install opencv-python

Minimum Hardware requirement:

opencv-3.4.0 needs to compile at least 1GB RAM with 2GB swap memory compulsory cause less than this configuration opencv will not able to compile on that system to generate object code.

OpenCV Basic Operation on Images

- Access pixel values and modify them.
- Access Image Properties.
- Setting Region of Image.
- Splitting and merging images.
- Change the image color.

Write test driver/stubs & Design test cases:

Test drivers/stubs will be written to simulate the behavior of the software's components. These drivers will be used to test the software's functionality and check for bugs.

Test cases will be designed to check the software's functionality thoroughly. The test cases will include unit testing, integration testing, and system testing.

Perform tests:

Tests will be executed to check the software's functionality and to identify any issues or bugs. The results of the tests will be analyzed to determine if the software is functioning correctly.

Schedule

The following is a list of project tasks with their starting and ending dates:

3.1. Installation of the software:
Start Date: 03/20/23
End Date: 03/27/23

3.2. Set up a testing environment:
Start Date: 03/20/23
End Date: 03/27/23

3.3. Write test driver/stubs:
Start Date: 03/27/23
End Date: 04/10/23

3.4. Design test cases:
Start Date: 04/10/23
End Date: 04/20/23

3.5. Perform tests:
Start Date: 04/20/23
End Date: 04/27/23

Overall, this project aims to test the **Python-OpenCV** software library thoroughly and identify any issues or bugs in the software.

CSU | Cleveland State University

EEC 623 SOFTWARE QUALITY ASSURANCE
TEST PLAN
PYTHON - OPENCV

Madhu Prakash Behara - 2845381
Sai Rohit Avula - 2837016
Ajay motharapu - 2861616

1 Software to be tested

OpenCV (Open Source Computer Vision Library)
cutoff date of September 2021, **the latest stable version of OpenCV is 4.5.3**. You can check the official OpenCV website for the latest version and download instructions.

To install the latest version of OpenCV in Python, you can use the pip package manager. Here's the command to install OpenCV:

```
pip install opencv-python
```

This will install the latest stable version of OpenCV along with its dependencies.

To determine the version of OpenCV installed in your Python environment, you can use the following code:

```
import cv2
print(cv2.__version__)
```

This will print the version number of OpenCV installed in your Python environment.
.

1 Software to be tested

To determine the platform on which OpenCV is running, you can use the following code:

```
import platform
print(platform.platform())
```

This will print the platform information of your system, including the operating system and architecture.

In general, OpenCV can be installed on various platforms, including Windows, Linux, and macOS. The installation method may differ depending on the platform you are using

2 Functions to be Tested

OpenCV is a fairly functional library that may be applied to a variety of computer vision and image processing tasks. Some of the frequently used OpenCV functions that can be tested are listed below:

Reading and displaying images: OpenCV has functions like imread() and imshow for reading and displaying images ().

Scaling and resizing of images is possible with OpenCV because to its resize and scale features ().

OpenCV has a number of image filtering functions, including **blur(), GaussianBlur(), and medianBlur ()**.

Image thresholding is possible with OpenCV thanks to its threshold() and adaptiveThreshold functions ().

Edge detection in images: OpenCV includes tools for doing this, such Canny ().

Finding contours in images is made possible by OpenCV using functions like findContours ().

2 Functions to be Tested

Image feature extraction: OpenCV includes tools like ORB() and SIFT for identifying and extracting characteristics from images ().

Images can be transformed using the functions provided by OpenCV, including rotation, translation, and perspective transformation.

Processing of movies: OpenCV has tools for processing videos, including reading, displaying, and removing frames from videos.

Learning algorithms: OpenCV offers algorithms for learning tasks including object identification, face recognition, and image categorization.

3 Testing Strategy

Unit testing is an important practice in software development to ensure that individual units or components of code are working as expected. In the context of Python OpenCV, unit testing involves testing individual functions, classes, or modules that use OpenCV to ensure that they are functioning correctly and producing the expected output.

Pytest is a popular testing framework for Python that makes it easy to write and run unit tests. It provides a simple syntax for defining test functions and test cases, and can automatically discover and run tests in your codebase

To write unit tests for Python OpenCV using pytest, you can define test functions that call the OpenCV functions or modules you want to test and then use assert statements to check whether the output is as expected. Here are some guidelines for writing unit tests using pytest:

- Define a test function for each unit of code you want to test.
- Use fixtures to set up any necessary data or objects for your tests.
- Use assert statements to check whether the output of the function or module being tested is as expected.
- Use parametrized tests to test a function or module with different input values or parameters.
- Use mocks or stubs to isolate your code from external dependencies or resources, and to test specific scenarios or edge cases.

3 Testing Strategy

Here is an example of a simple unit test using pytest to test the functionality of the **cv2.imread** function:

```
import cv2
import numpy as np

def test_image_reading():
    # Test if image is correctly read
    img = cv2.imread('test_image.jpg')
    assert isinstance(img, np.ndarray)
    assert img.shape == (480, 640, 3)
```

In this example, we define a test function **test_image_reading** that calls the **cv2.imread** function to read an image file and then uses assert statements to check whether the output is as expected. We use the **isinstance** function to check whether the output is a numpy ndarray, and we check the shape of the output to ensure that it matches the expected shape.
To run this test using pytest, you can simply run the pytest command from the command line in the directory containing your test file. pytest will automatically discover and run all test functions in your codebase and report any failures or errors.

3 Testing Strategy

System testing is a subset of software testing that examines the system as a whole to determine whether or not it complies with the requirements and performs as intended. System testing for Python OpenCV entails checking the functionality of the complete program or system that utilizes OpenCV.

Examples of system testing scenarios for a Python OpenCV application are shown below:

pipeline for testing image processing: For an image processing task, test the complete pipeline, including the reading of the image, image preprocessing, feature extraction, classification, and output. To make sure the pipeline functions well and generates accurate results, test it using a range of photos.

Test the whole video processing pipeline, including the reading of the video, frame extraction, preprocessing, feature extraction, classification, and output. To make sure the pipeline functions properly and generates accurate results, test it using a range of films.

Test your application's object identification and tracking capabilities by giving it a variety of movies featuring various items in various settings. Check to see if the application can accurately detect and track the objects, even when they overlap.

4 Testing Work Product

The numerous deliverables or artifacts created throughout the software development lifecycle, like requirements, designs, codes, and documentation, are tested when working with Python OpenCV.
Examples of work items that can be tested in Python OpenCV are as follows:

Verify the testability and verifiability of the requirements using the relevant test cases.

Design: Test your Python OpenCV application's design to make sure it adheres to the specifications and has a solid structural foundation.
Make sure the design can be readily maintained and expanded in the future and that it complies with accepted coding standards.

Code: Test the code for your Python OpenCV application to make sure it is accurate, effective, and complies with the design guidelines.
Verify the operation of individual functions and modules and make sure they function properly when combined by using unit testing and integration testing methodologies.

Test the Python OpenCV application's documentation to make sure it is clear, accurate, and comprehensive.
Make sure the documentation is user- and developer-friendly and covers every aspect of the application, including installation, configuration, and usage.

4 Testing Work Product

User interface: Test your Python OpenCV application's user interface to make sure it is simple to use and satisfies users' demands. Verify that the user interface complies with the requirements and design guidelines and that it gives the user the right kind of feedback.

Performance: Test your Python OpenCV application's performance to make sure it satisfies the necessary performance standards and can manage the anticipated workload. Verify the application's effectiveness and responsiveness as well as its ability to handle huge datasets or videos without crashing or performing slowly.

Overall, testing work products in Python OpenCV involves verifying that each deliverable meets the requirements and quality standards and works correctly with other components of the system. By testing each work product throughout the software development lifecycle, you can ensure that your Python OpenCV application is robust, reliable, and meets the needs

5 Test Record Keeping

To keep records of your OpenCV project, you can use the built-in logging module in Python.
Here's how you can set up logging for your OpenCV project:

Import the logging module at the top of your script:

```
import logging
```

Set up the logging configuration with the desired format, level, and file location:

```
logging.basicConfig(filename='example.log', level=logging.DEBUG,
format='%(asctime)s %(levelname)s:%(message)s')
```

In this example, we set the logging level to DEBUG, which means all messages with a severity level of DEBUG or higher will be recorded in the log file. The format string specifies how the messages will be formatted in the log file.

Use the logging module to record messages throughout your script:

```
logging.debug('This is a debug message')
logging.info('This is an info message')
logging.warning('This is a warning message')
logging.error('This is an error message')
logging.critical('This is a critical message')
```

When you run your script, the log messages will be recorded in the file specified in the logging configuration.

5 Test Record Keeping

Here's an example of how you might use logging in an OpenCV project to record information about image processing:

```
import cv2
import logging

logging.basicConfig(filename='example.log',      level=logging.DEBUG,
format='%(asctime)s %(levelname)s:%(message)s')
img = cv2.imread('example.jpg')
logging.debug('Image read successfully')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
logging.debug('Image converted to grayscale')
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
logging.debug('Image blurred with Gaussian filter')
edges = cv2.Canny(blurred, 50, 150)
logging.debug('Edges detected using Canny edge detection')
cv2.imshow('Edges', edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In this example, we log messages at each step of the image processing pipeline to keep track of what's happening in the script. If anything goes wrong, we can refer to the log file to see where the problem occurred.

6 Testing tools and Environment

There are several testing tools and environments that can be used for testing Python OpenCV applications. We are going to Use

Pytest: Pytest is a popular testing framework for Python that makes it easy to write and run unit tests. It provides a simple syntax for defining test functions and test cases, and can automatically discover and run tests in your codebase. Pytest is a good choice for testing Python OpenCV applications as it is easy to use and provides many useful features such as fixtures, parametrized tests, and mocking

The testing environment and requirements for Python OpenCV depend on the specific needs and requirements of your application. Here are some general guidelines to consider:

Operating System: Python OpenCV can be run on a variety of operating systems including Windows, Linux, and macOS. Make sure to select an operating system that is supported by both OpenCV and the testing tools you plan to use.

Python Version: Python OpenCV is compatible with Python 3.x versions. Make sure to use a compatible version of Python when testing your application.

6 Testing tools and Environment

Hardware Requirements: Python OpenCV can run on a wide range of hardware, from low-end devices such as Raspberry Pi to high-end servers. The hardware requirements for your application depend on the complexity and workload of your application. For example, if you are working with large video datasets, you may need a more powerful machine with more RAM and a fast processor.

IDEs: Integrated development environments (IDEs) such as PyCharm, Visual Studio Code, and Eclipse can be used to write and test Python OpenCV applications. Choose an IDE that is compatible with your operating system and testing framework.

Dependencies: Make sure to install all the necessary dependencies required to run OpenCV and the testing framework. This includes NumPy, Matplotlib, and any other libraries that your application depends on.

7 Testing Schedule

Week 1:

- Review project requirements and design
- Install and configure testing environment
- Identify test cases and scenarios

Week 2-3:

- Perform unit testing using pytest and unittest frameworks
- Collaborate with developers to fix issues found during testing
- Write test reports and update test cases

Week 4-5:

- Perform system testing on Python OpenCV application
- Perform integration testing with other systems, if necessary
- Collaborate with developers to fix issues found during testing
- Write test reports and update test cases

7 Testing Schedule

Roles:

- **Madhu Prakash Behara - 2845381- Test Lead:**
  - Responsible for managing the overall testing process
  - Ensures that testing is conducted according to the project schedule
  - Assigns testing tasks to Rohit and Ajay
  - Coordinates with developers to fix issues found during testing
  - Reviews and approves test reports

- **Sai Rohit Avula - 2837016 - Unit Tester:**
  - Responsible for performing unit testing using pytest and unittest frameworks
  - Identifies and reports issues found during unit testing
  - Collaborates with developers to fix issues found during testing
  - Writes test reports and updates test cases

- **Ajay motharapu - 2861616 - System Tester:**
  - Responsible for performing system testing on Python OpenCV application
  - Performs integration testing with other systems, if necessary
  - Identifies and reports issues found during system testing
  - Collaborates with developers to fix issues found during testing
  - Writes test reports and updates test cases

APRIL 10, 2023

Cleveland State University

EEC 623 – SOFTWARE QUALITY ASSURANCE

TEST CASE SPECIFICATION

Madhu Prakash Behara - 2845381
Sai Rohit Avula - 2837016
Ajay motharapu - 2861616

Test Case Identifier: TC 001

Test Items/feature: Face detection

Preconditions: OpenCV library installed and test image available

Test Steps:

- Load test image using cv2.imread()
- Convert the image to grayscale using cv2.cvtColor()
- Apply face detection using cv2.CascadeClassifier()
- Verify that at least one face is detected using the assert statement
- Output Specifications: At least one face is detected in the image

Inter-case Dependencies: N/A

Test Result:

Test Case Identifier: TC 002

Test Items/feature: Multiple face detection

Preconditions: OpenCV library installed and test image available with multiple faces

Test Steps:

- Load test image using cv2.imread()
- Convert the image to grayscale using cv2.cvtColor()
- Apply face detection using cv2.CascadeClassifier()
- Verify that multiple faces are detected using the assert statement
- Output Specifications: Multiple faces are detected in the image

Inter-case Dependencies: TC 001

Test Result:

Test Case Identifier: TC 003

Test Items/feature: Face detection on the video stream

Preconditions: OpenCV library installed and video stream available

Test Steps:

- Capture video stream using cv2.VideoCapture()
- Convert each frame to grayscale using cv2.cvtColor()
- Apply face detection using cv2.CascadeClassifier()
- Verify that at least one face is detected in each frame using the assert statement
- Output Specifications: At least one face is detected in each frame of the video stream

Inter-case Dependencies: TC 001

Test Result:

Test Case Identifier: TC 004

Test Items/feature: False positive detection

Preconditions: OpenCV library installed and test image available with no faces

Test Steps:

- Load test image using cv2.imread()
- Convert the image to grayscale using cv2.cvtColor()
- Apply face detection using cv2.CascadeClassifier()
- Verify that no faces are detected using assert statement
- Output Specifications: No faces are detected in the image

Inter-case Dependencies: TC 001

Test Result:

Test Case Identifier: TC 005

Test Items/feature: The face detection accuracy

Preconditions: OpenCV library installed and test image available with a clear face

Test Steps:

- Load test image using cv2.imread()
- Convert the image to grayscale using cv2.cvtColor()
- Apply face detection using cv2.CascadeClassifier()
- Verify that the detected face matches the actual face using an assert statement and visual inspection
- Output Specifications: The detected face matches the actual face in the image

Inter-case Dependencies: TC 001

Test Result:

Test Case Identifier: TC 006

Test Items/feature: Image filtering

Preconditions: OpenCV library installed and test image available

Test Steps:

- Load test image using cv2.imread()
- Apply filter using cv2.filter2D() function
- Verify that the filtered image is not identical to the original image using the assert statement Output Specifications: The filtered image should show a noticeable difference from the original image

Inter-case Dependencies: N/A

Test Result:

Test Case Identifier: TC 007
Test Items/feature: Object tracking
Preconditions: OpenCV library installed and test video available with moving object
Test Steps:

- Load test video using cv2.VideoCapture()
- Define ROI using cv2.selectROI() function
- Apply object tracking using cv2.Tracker() class
- Verify that the object is correctly tracked using an assert statement and visual inspection Output Specifications: The object should be tracked accurately throughout the video

Inter-case Dependencies: N/A
Test Result:

Test Case Identifier: TC 008
Test Items/feature: Template matching
Preconditions: OpenCV library installed and test image available with template and target image Test Steps:

- Load test images using cv2.imread()
- Convert images to grayscale using cv2.cvtColor()
- Define template using cv2.selectROI() function
- Apply template matching using cv2.matchTemplate()
- Verify that the location of the target image in the template is correctly identified using an assert statement and visual inspection Output Specifications: The location of the target image in the template should be accurately identified

Inter-case Dependencies: N/A
Test Result: