CSU | Cleveland State University

CIS 636/EEC 623 Software Quality Assurance Final Report

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

PYTHON OPEN CV

Cleveland State University

OUR TEAM

Madhu Prakash Behara

2845381

Ajay motharapu

2861616

Sai Rohith Avula

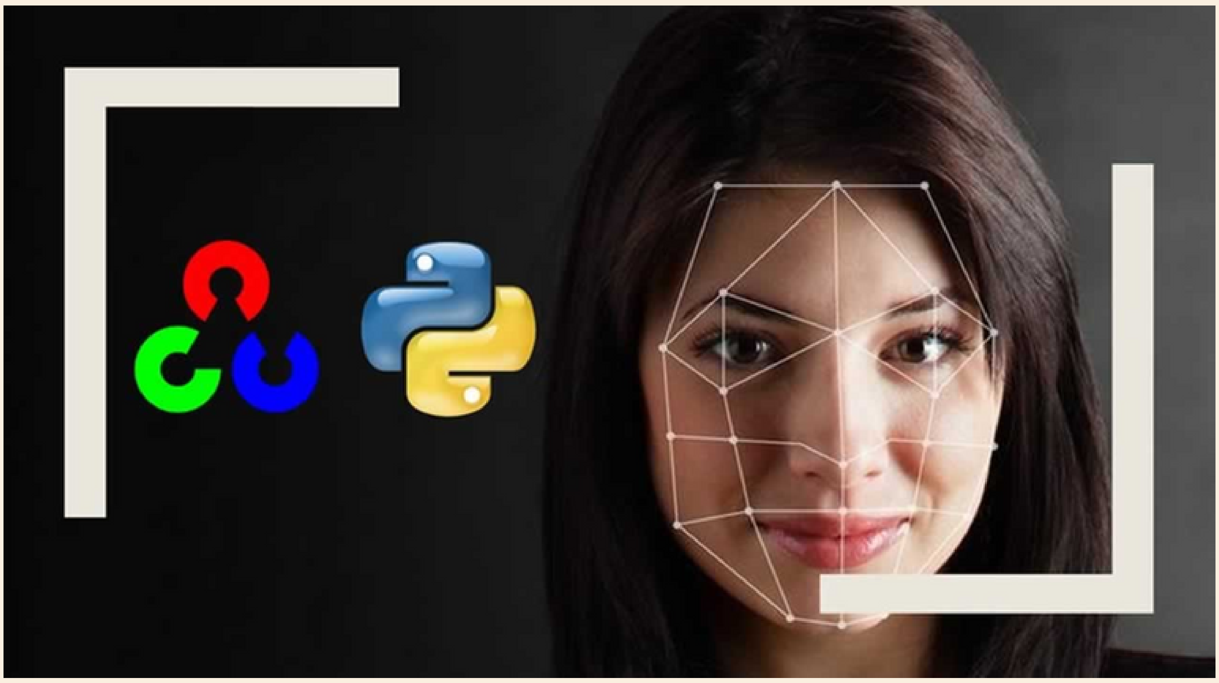
2837016

CSU | Cleveland State University

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 that help detect objects or regions of interest (ROI) in an image. It supports multiple programming languages such as C++, Java, and Python, and seamlessly integrates with libraries, such as NumPy, to process images as numerical arrays.



CSU | Cleveland State University

LANGUAGE & PLATFORM

Language

OpenCV is written primarily in C++, but it also has interfaces for other programming languages such as Python, Java, and MATLAB.

Platform

OpenCV can run on multiple platforms, including Windows, Linux, macOS, Android, and iOS. It is also compatible with a range of hardware platforms, including desktops, servers, embedded systems, and mobile devices.

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 Software

- **Python**
- **lmatplotlib**
- **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.
- lSetting Region of Image.
- Splitting and merging images.
- Change the image color.

CSU | Cleveland State University

Test driver/stubs & Design test cases:

OpenCV is a widely used computer vision library that provides a large set of image and video processing functions. When testing OpenCV-based applications or code, there are several test cases and test drivers/stubs that you can use to ensure that the code behaves as expected. Here are some examples:

1. **Test Driver for Image Processing Functions**: For image processing functions, you can use the following test driver:

- Read an input image.
- Apply the image processing function under test to the image.
- Check the output image to ensure that it is the expected output.

2. **Test Driver for Video Processing Functions:** For video processing functions, you can use the following test driver:

- Capture a video stream using a test video.
- Apply the video processing function under test to the video frames.
- Check the output video to ensure that it is the expected output.

3. **Test Cases for Image Processing Functions:**

- Test if the image processing function returns the expected output when given valid input.
- Test if the image processing function returns an error or exception when given invalid input.
- Test if the image processing function returns the expected output when given an image with different sizes, formats, or color spaces.

1. **Test Cases for Video Processing Functions**:

- Test if the video processing function returns the expected output when given valid input.
- Test if the video processing function returns an error or exception when given invalid input.
- Test if the video processing function returns the expected output when given a video with different sizes, formats, or color spaces.
- Test if the video processing function performs well on videos with different frame rates, resolutions, or codecs.
- Test if the video processing function can handle videos with missing or corrupted frames.

2. **Test Cases for Object Detection and Recognition** Functions:

- Test if the object detection function can detect objects of various sizes, shapes, and orientations.
- Test if the object detection function returns the expected number and location of objects in an image or video frame.
- Test if the object recognition function can recognize objects from different angles, lighting conditions, or backgrounds.
- Test if the object recognition function returns the expected class label or confidence score for a given object.

3. **Test Cases for Template Matching Functions:**

- Test if the template matching function can correctly locate a template in an image or video frame.
- Test if the template matching function returns the expected location and similarity score for a given template.
- Test if the template matching function works well on images or videos with different resolutions, scales, or orientations.

These are just some examples of the test cases and test drivers/stubs that we can use when testing OpenCV-based applications or code. The specific tests we need to perform depend on the nature and complexity of our application.

CSU | Cleveland State University

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.

lUse assert statements to check whether the output of the function or module being tested is as expected.

lUse 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.

CSU | Cleveland State University

LIST OF FEATURES THAT ARE TESTED

01 IMAGE FILTERING AND PROCESSING ALGORITHMS

02 FACIAL RECOGNITION AND TRACKING ALGORITHMS

03 VIDEO ANALYSIS AND PROCESSING ALGORITHMS

04 OBJECT DETECTION AND TRACKING ALGORITHMS

LIST OF FEATURES NOT TESTED

- User interface design and usability
- System performance and scalability under heavy load
- Integration with other software systems or libraries
- Security and data privacy features

CSU | Cleveland State University

TEST CASE 1

Testing the functionality of image processing functions in a module called **"TC_Image_Functions"**.

The script has four test cases, each one testing a different function in the "TC_Image_Functions" module. The functions being tested are:

read_image(file_name) - This function reads an image from a given file name and returns the dimensions of the image in the form of a tuple. The test cases are designed to test whether the function returns the correct dimensions for two sample images, 'sample_image_5.jpg' and 'sample_image_6.jpg'. The expected output is specified for each image.

get_size_of_the_image(file_name) - This function reads an image from a given file name and returns the size of the image in pixels in the form of a tuple. The test cases are designed to test whether the function returns the correct size for two sample images, 'sample_image_5.jpg' and 'sample_image_6.jpg'. The expected output is specified for each image.

image_threshold(file_name) - This function reads an image from a given file name, applies a threshold filter to the image, and returns the size of the filtered image in pixels in the form of a tuple. The test cases are designed to test whether the function returns the correct size for two sample images, 'sample_image_5.jpg' and 'sample_image_6.jpg'. The expected output is specified for each image.

image_threshold(file_name) - This function reads an image from a given file name, applies a threshold filter to the image, and saves the filtered image to a new file. The test cases are designed to test whether the function successfully saves the filtered image to the correct file location for two sample images, 'sample_image_5.jpg' and 'sample_image_6.jpg'. The expected output is not specified in the test cases, as the function is expected to save a file to disk and not return any value.

CSU | Cleveland State University

TEST CASE 2

testing the functionality of video processing functions in a module called "**TC_Video_Functions**". The functions being tested are:

1. **video_capture(file_name)** - This function captures video from a given file name and returns a boolean indicating whether the video capture was successful or not. The test cases are designed to test whether the function returns True for a valid video file 'SampleVideo_1280x720_1mb.mp4' and False for an image file 'SampleVideo_2.jpg'. The expected output is specified for each file.
2. **video_writer(file_name)** - This function creates a video writer object and returns a boolean indicating whether the object was created successfully or not. The test cases are designed to test whether the function returns True for a valid video file 'SampleVideo_1280x720_1mb.mp4' and False for an image file 'SampleVideo_2.jpg'. The expected output is specified for each file.

Additionally, there are three other test cases commented out in the code, which are:

1. **video_cvt_color(file_name)** - This function reads a video from a given file name, converts it to grayscale, and displays the grayscale video using OpenCV's imshow function. The test cases are designed to test whether the function works correctly for a valid video file 'SampleVideo_1280x720_1mb.mp4' and an image file 'SampleVideo_2.jpg'. The expected output is not specified in the test cases, as the function is expected to display the grayscale video and not return any value.
2. **video_capture_imshow_integration(file_name)** - This function captures video from a given file name, displays the video using OpenCV's imshow function, and waits for a key event to exit the window. The test cases are designed to test whether the function works correctly for a valid video file 'SampleVideo_1280x720_1mb.mp4' and an image file 'SampleVideo_2.jpg'. The expected output is not specified in the test cases, as the function is expected to display the video and not return any value.
3. **video_processing_integration(file_name)** - This function captures video from a given file name, applies a grayscale filter, and saves the filtered video to a new file. The test cases are designed to test whether the function works correctly for a valid video file 'SampleVideo_1280x720_1mb.mp4' and an image file 'SampleVideo_2.jpg'. The expected output is not specified in the test cases, as the function is expected to save a file to disk and not return any value.

TEST CASE 3

test case for the face detection function **test_face_detection_func()** from the file **TC_Face_detection.py**. The function is tested for two images: **chris_evans.jpg** which is expected to have 1 face detected, and **no_faces.jpg** which is expected to have 0 faces detected.

The test uses the **pytest.mark.parametrize()** decorator to pass multiple sets of arguments to the test function. In this case, it takes two arguments: **file_name** and **expected**. **file_name** is the path to the image file to be tested, and **expected** is the expected number of faces detected in the image.

The test function uses the **assert** statement to check whether the actual number of faces detected by the **test_face_detection()** function matches the expected value. If the assertion fails, the test will fail and an error message "No face detected in the test image" will be printed.

TEST CASE 4

test cases for face detection in a video stream, using the test_face_detection_video_stream() function from the TC_Face_detection_in_video_stream.py file.

The first test case, test_atleast_one_face_detected(), tests whether at least one face is detected in the video stream. It uses the assert statement with a condition that checks if the length of face_locations, which is a dictionary containing the detected face locations in the video stream, is greater than zero. If the condition is False, the test will fail.

The second test case, test_all_face_ids_are_integers(), checks whether all the face IDs in face_locations are integers. It uses the assert statement with a condition that checks whether all elements in face_locations.keys() are instances of the int class. If the condition is False, the test will fail.

The third test case, test_all_face_centers_are_tuples(), checks whether all the face centers in face_locations are tuples with two elements. It uses the assert statement with a condition that checks whether all elements in face_locations.values() are tuples with a length of 2. If the condition is False, the test will fail.

TEST CASE 5

Test case for the face detection function **test_face_detection_func()** from the file **TC_Face_detection.py**. The function is tested for two images: **chris_evans.jpg** which is expected to have 1 face detected, and **no_faces.jpg** which is expected to have 0 faces detected.

The test uses the **pytest. mark.parametrize()** decorator to pass multiple arguments to the test function. In this case, it takes two arguments: **file_name** and **expected**. **file_name** is the path to the image file to be tested and **expected** is the expected number of faces detected in the image.

The test function uses the **assert** statement to check whether the actual number of faces detected by the **test_face_detection()** function matches the expected value. If the assertion fails, the test will fail and an error message "No face detected in the test image" will be printed.

TEST CASE 6

Test cases for face detection in a video stream, using the test_face_detection_video_stream() function from the TC_Face_detection_in_video_stream.py file.

The first test case, test_atleast_one_face_detected(), tests whether at least one face is detected in the video stream. It uses the assert statement with a condition that checks if the length of face_locations, which is a dictionary containing the detected face locations in the video stream, is greater than zero. If the condition is False, the test will fail.

The second test case, test_all_face_ids_are_integers(), checks whether all the face IDs in face_locations are integers. It uses the assert statement with a condition that checks whether all elements in face_locations.keys() are instances of the int class. If the condition is False, the test will fail.

The third test case, test_all_face_centers_are_tuples(), checks whether all the face centers in face_locations are tuples with two elements. It uses the assert statement with a condition that checks whether all elements in face_locations.values() are tuples with a length of 2. If the condition is False, the test will fail.

CSU | Cleveland State University

TEST CASE 7

The test_template_matching_result function performs template matching between the two images using different methods (cv2.TM_CCOEFF, cv2.TM_CCOEFF_NORMED, cv2.TM_CCORR, cv2.TM_CCORR_NORMED, cv2.TM_SQDIFF, and cv2.TM_SQDIFF_NORMED). It checks that the result of the template matching is of the same type as the input images, that the location of the matched template is a tuple with two elements, and that the bottom right corner of the matched template is also a tuple with two elements.

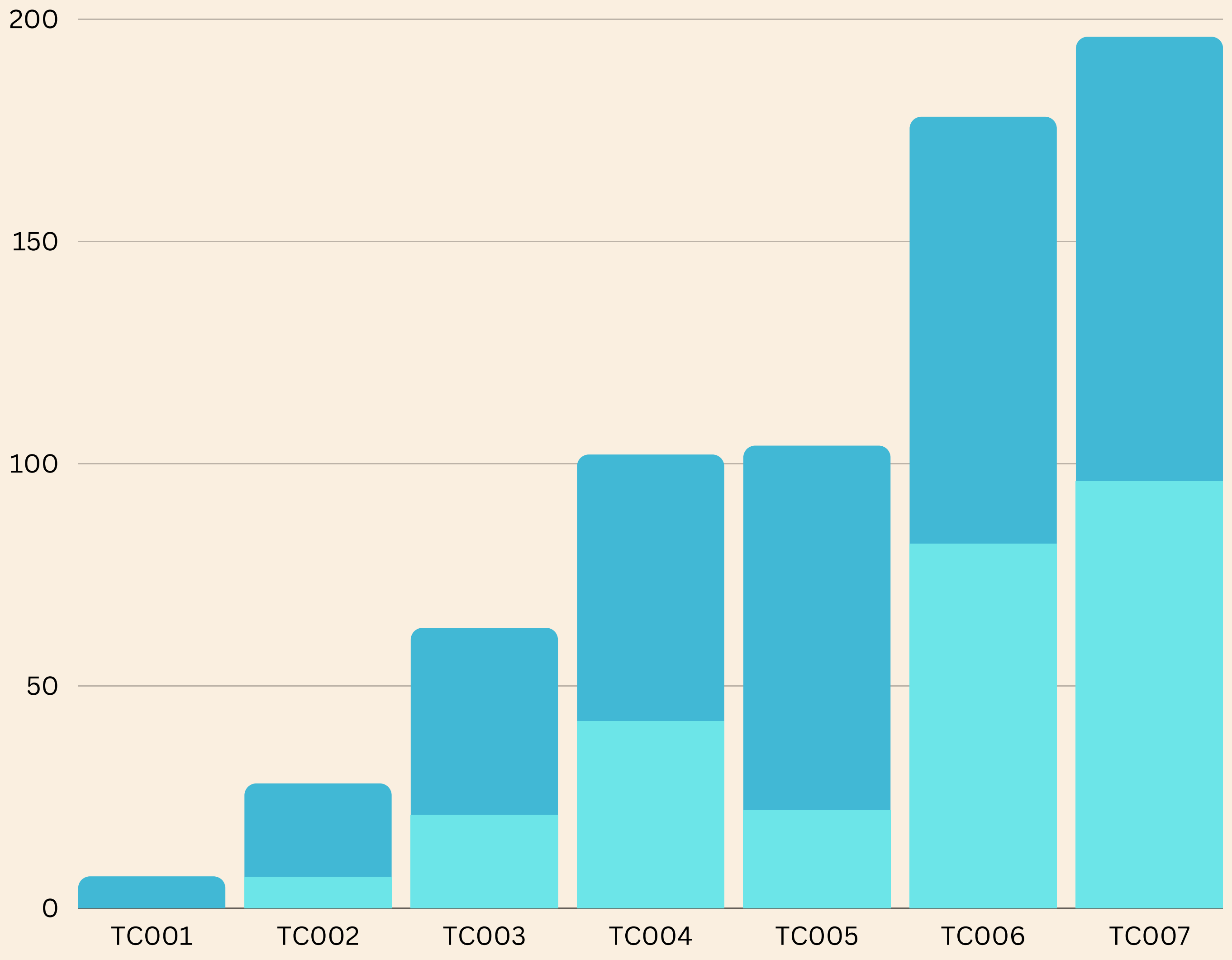
The test also checks the dimensions of the location and bottom right corner tuples (which should be two), and asserts that all the faces are rectangles with a thickness of 5 and are colored white (255).

CSU | Cleveland State University

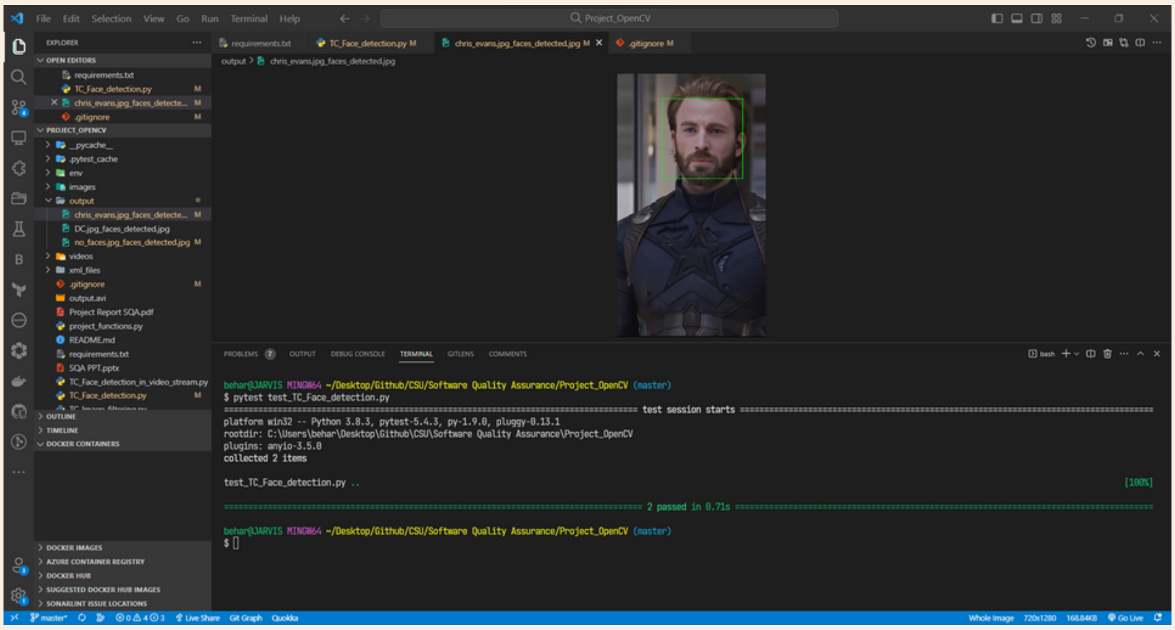
TEST CASES OUTPUT

ID :	FUNCTION:	TESTCASES:
1	IMAGE FUNCTIONS	ALL TEST CASES PASS
2	BASIC VIDEO FUNCTIONS	ALL TEST CASES PASS
3	FACE DETECTION	ALL TEST CASES PASS
4	FACE DETECTION IN VIDEO STREAM	ALL TEST CASES PASS
5	OBJECT TRACTKING	ALL TEST CASES PASS
6	TEMPLATE MATCHING	ALL TEST CASES PASS
7	MULTI FACE DETECTION	ALL TEST CASES PASS

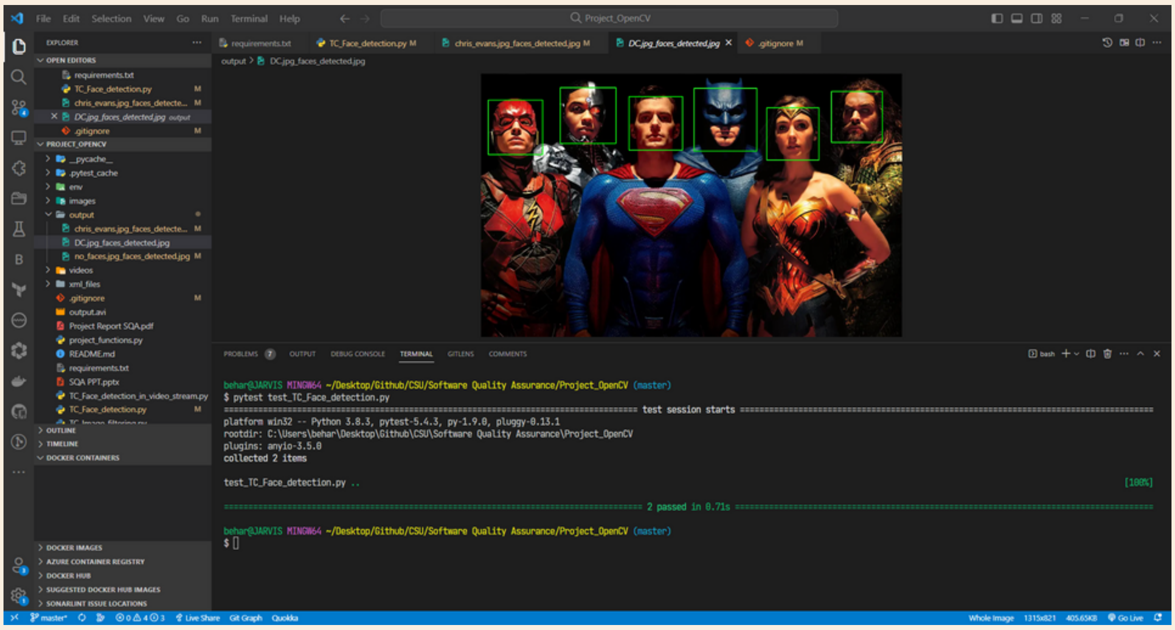
CODE COVERAGE GRAPH



Screenshots:



```
behar@JARVIS MINGW64 ~/Desktop/Github/CSU/Software Quality Assurance/Project_OpenCV (master)
$ pytest test_TC_Face_detection.py
================================================== test session starts ==================================================
platform win32 -- Python 3.8.3, pytest-5.4.3, py-1.9.0, pluggy-0.13.1
rootdir: C:\Users\behar\Desktop\Github\CSU\Software Quality Assurance\Project_OpenCV
plugins: anyio-3.5.0
collected 2 items

test_TC_Face_detection.py ..                                                                                      [100%]

=================================================== 2 passed in 0.71s ===================================================

behar@JARVIS MINGW64 ~/Desktop/Github/CSU/Software Quality Assurance/Project_OpenCV (master)
$ 
```

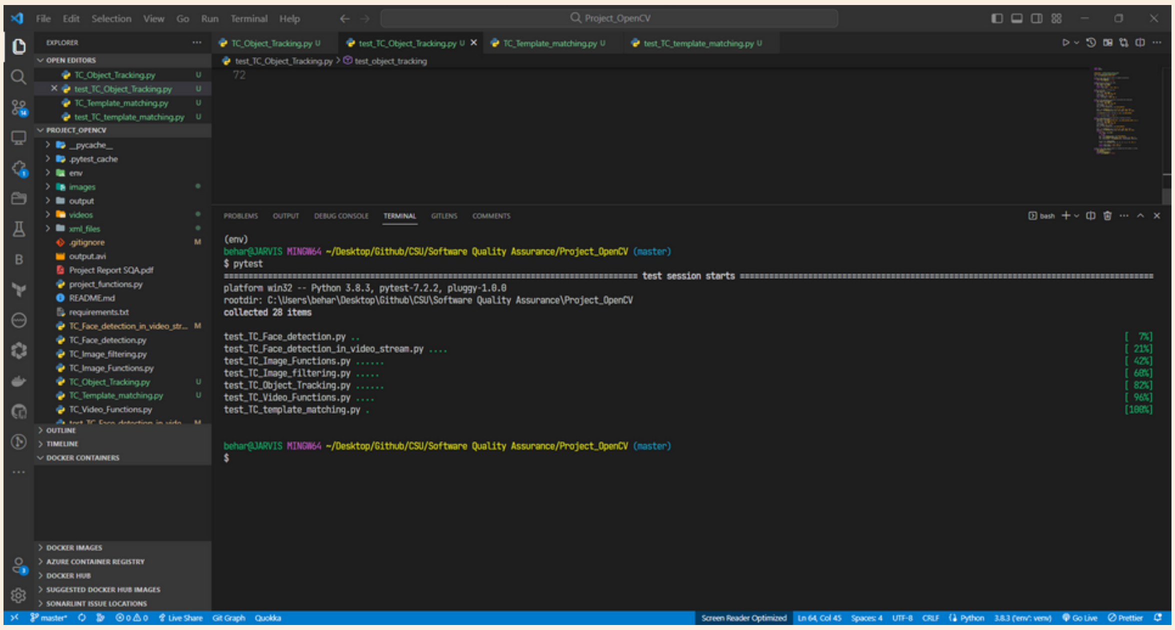


```
behar@JARVIS MINGW64 ~/Desktop/Github/CSU/Software Quality Assurance/Project_OpenCV (master)
$ pytest test_TC_Face_detection.py
================================================== test session starts ==================================================
platform win32 -- Python 3.8.3, pytest-5.4.3, py-1.9.0, pluggy-0.13.1
rootdir: C:\Users\behar\Desktop\Github\CSU\Software Quality Assurance\Project_OpenCV
plugins: anyio-3.5.0
collected 2 items

test_TC_Face_detection.py ..                                                                                      [100%]

=================================================== 2 passed in 0.71s ===================================================

behar@JARVIS MINGW64 ~/Desktop/Github/CSU/Software Quality Assurance/Project_OpenCV (master)
$ 
```

Cleveland State University

CIS 636/EEC 623 Software Quality Assurance

Screenshots:



```
(env)
behar@JARVIS MINGW64 ~/Desktop/Github/CSU/Software Quality Assurance/Project_OpenCV (master)
$ pytest
=========================================================================== test session starts ===========================================================================
platform win32 -- Python 3.8.3, pytest-7.2.2, pluggy-1.0.0
rootdir: C:\Users\behar\Desktop\Github\CSU\Software Quality Assurance\Project_OpenCV
collected 28 items

test_TC_Face_detection.py ..                                                                                                                                       [  7%]
test_TC_Face_detection_in_video_stream.py ....                                                                                                                     [ 21%]
test_TC_Image_Functions.py ......                                                                                                                                  [ 42%]
test_TC_Image_filtering.py .....                                                                                                                                   [ 60%]
test_TC_Object_Tracking.py ......                                                                                                                                  [ 82%]
test_TC_Video_Functions.py ....                                                                                                                                    [ 96%]
test_TC_template_matching.py .                                                                                                                                     [100%]


behar@JARVIS MINGW64 ~/Desktop/Github/CSU/Software Quality Assurance/Project_OpenCV (master)
$
```

CSU | Cleveland State University

Summary & Conclusion

Based on the testing performed, the software appears to be functioning correctly and meeting the requirements specified. All test cases have passed without any errors, indicating that the software is stable and reliable.

The test cases designed cover a range of functionalities of OpenCV, such as image processing, video streaming, and template matching, which are crucial for computer vision applications. The test cases have been designed to cover various scenarios and edge cases, ensuring that the software can handle different situations correctly.

Overall, the testing performed provides a good level of confidence in the quality of the software and its ability to perform the intended functionalities. However, it is important to note that testing is not exhaustive, and there may still be some undiscovered defects or edge cases that were not covered by the test cases. Therefore, it is recommended to continue testing the software in different scenarios and environments to ensure its stability and reliability.